Metasploit Framework Version 2.2
User Crash Course

Introduction

Ce document est une esquisse de manuel utilisateur pour la version 2.2 du Metasploit Framework, son but est de fournir une vue d'ensemble basique de ce qu'est le Framework, comment il fonctionne, et ce que vous pouvez faire avec. Comme pour la plupart des projets open-sources, la documentation correcte passe après le développement en cours. Si vous souhaitez contribuer au projet et avez de véritables talents d'écriture, veuillez contacter les développeurs à l'adresse email indiquée à la fin de ce document.

Le Framework Metasploit est un environnement complet pour écrire, tester et utiliser des exploits. Cet environnement fournit une plateforme solide pour les tests d'intrusion, le développement de shellcodes, et la recherche de vulnérabilités. La majorité du Framework est composée de code orienté objet Perl, avec des composants optionnels écrits en C, assembleur, et Python.

Il y a actuellement deux développeurs centraux, deux contributeurs significatifs, et un vaste groupe de personnes qui ont fournis des idées ou codes qui ont contribués au projet. Veuillez vous référer aux Credits du module exploit pour une liste complète de tous les gens impliqués dans le projet.

Installation

Installation sur Unix

Installer le Framework est aussi simple que d'extraire le tarball, aller dans le répertoire créé, et lancer votre interface utilisateur péférée. Nous recommandons fortement que vous compiliez et installez le module Perl Term::ReadLine::Gnu trouvable dans le sous-répertoire 'extras'. Ce paquetage active le support de la complétion par tabulation dans l'interface *msfconsole*, *msfconsole* est l'IU préférable pour l'utilisation de tous les jours. Si le support SSL est désiré, vous devriez installer le module Perl Net::SSLeay également, il peut aussi être trouvé dans le sous-répertoire 'extras'.

Pour réaliser une installaton étendue au système, nous vous recommandons de copier l'ensemble du répertoire Framework dans un endroit globalement accessible (/usr/local/msf) et d'ensuite créer des liens symboliques depuis les applications msf* vers un répertoire dans le path système (/usr/local/bin). Les modules spécifiques aux utilisateurs peuvent être placés dans le répertoire $HOME/.msf/<TYPE>, où TYPE représente exploits, payloads, nops, ou encoders.

Installation sur Windows

Après des mois à travailler sur les bugs ActiveState, nous avons finalement décider de l'abandonner et de supporter uniquement Cygwin Perl. L'installateur Win32 du Metasploit Framework comprend une copie allégée de l'environnement Cygwin, c'est la manière conseillée d'utiliser le Framework sur une plateforme Windows. Si vous voulez installer le Framework dans un environnement Cygwin existant; veuillez vous référer au fichier 'docs/QUICKSTART.cygwin' dans le répertoire d'installation; il existe un grand nombre de problèmes pour installer les modules Term::ReadLine::Gnu et Net::SSLeay qui demande une grande dextérité pour les résoudre.

Mise en garde sur les Plateformes

Comme nous avons essayé de supporter le plus de plateformes possibles, il y a quelques problèmes de compatibilité qui sont apparus. Si vous planifiez d'accèder à l'interface msfweb depuis un système MacOS X, soyez averti qu'Internet Explorer rencontrera des problèmes au dernier stade du processus d'exploit. Ce problème résulte du fait que IE n'est pas capable d'afficher des données incrémentales depuis des serveurs HTTP 1.0. Le support du raw socket est actuellement non fonctionnel dans Cygwin, AIX, HP-UX, et probablemen Solaris. Cela affectera votre possibilité de spoofer les attaques basées sur UDP utilisant la variable d'environnement UdpSourceIp. Les utilisateurs de Windows pourraient rencontrer des problèmes en utilisant l'installateur Win32 sur un système qui possède déjà une version de Cyqwin plus ancienne installée.

Systèmes d'Exploitation Supportés

Le Framework devrait tourner sur pratiquement tous les systèmes d'exploitation basés sur Unix qui incluent une version complète et moderne de l'interpréteur Perl (5.6+). Chaque version stable du Framework est testée avec quatre plateformes primaires:

- Linux (x86, ppc) (2.4, 2.6)
- Windows NT (4.0, 2000, XP, 2003)
- BSD (Open 3.x, Free 4.6+)
- MacOS X (10.3.3)

Les plateformes suivantes sont connues pour être problématiques:

- Windows 9x (95, 98, ME)
- HP-UX 11i (requires Perl upgrade)

Nous avons reçus un grand nombre de messages confirmant que le Framework fonctionne correctement sur Solaris, AIX, et même sur Zaurus de Sharp (basé sur Linux). Ces systèmes nécessitent souvent une version mise à jour de Perl en conjonction avec les utilitaires GNU pour fonctionner correctement.

Mettre à jour le Framework

A partir de la version 2.2, le Framework inclut l'utilitaire de mise à jour en ligne *msfupdate*. Ce script peut être utilisé pour télécharger et installer la dernière version du Framework depuis le site internet metasploit.com. Il effectue des mises à jour par fichier en comparant les checksum des fichiers locaux avec ceux disponibles sur le site internet. Ce processus s'effectue à travers une connection SSL fiable, en supposant que le module Net::SSLeay a été installé. Ce n'est pas totalement sûr et dépend encore de la sécurité du serveur du site metasploit.com. Pour en apprendre d'avantage sur l'utilitaire msfupdate, exécutez le simplement avec l'argument –h.

Si vous ne souhaitez pas utiliser le système de mise à jour en ligne, vous pouvez toujours télécharger les modules mis à jour sur le site metasploit.com. Une interface sera disponible dans un futur proche pour télécharger la dernière version stable immédiatement.

Allons-Y

La Console d'Interface

Après que vous ayez installé le Framework, vous devriez vérifier que tout fonctionne correctement. La manière laplus rapide de faire cela est d'exécuter la console utilisateur *msfconsole*. Cette interface devrait afficher un logo Metasploi en ASCII, afficher la version actuelle, le nombre de payloads, le nombre d'exploits, et finir par un prompt 'msf'. Depuis ce prompt, tapper **help** pour obtenir une liste des commandes valides. Vous êtes actuellement dans le mode 'principal', cela vous permet de liter les exploits, lister les payloads, et configurer les options globales. Pour lister tous les exploits diponibles, tapper **show exploits**. Pour obtenir plus d'informations sur un exploit donné, tapper **info nom_du_module**.

Efficacité de la Console

La console a été réalisée en pensant à l'efficacité et peut être utiliser comme un shell standard dans beaucoup de situations. Si vous entrer une commande inconnue, la console scannera le path système pour déterminer si vous avez tapper une commande externe. Si elle trouve une correspondance, cette commande sera exécutée avec les arguments spécifiés. Cela vous permet d'utiliser votre ensemble d'outils standards sans avoir à quitter la console. La complétion par TAB agit sur le nomde fichier par défaut quand la commande entrée n'est pas une commande interne à la console. Cela vous permet de naviguer dans les fichiers du système normalement, comme en utilisant un shell classique.

Choisir un Exploit

Depuis le prompt msf, vous pouvez sélectionner un exploit avec la commande **use**. Cette commande prend le nom du module d'exploit comme premier argument, rentre en mode exploit, et charge l'environnement temporaire pour cet exploit. Vous pouvez passer d'un exploit actif à un autre avec la commande **use** et revenir au shell principal avec la commande **back**.

Bases pour un Exploit

Après avoir choisi un exploit, votre commande de sélection disponible change. Entrer la commande **help** pour avoir une idée de ce qui est disponible. La commande **show** renvoie alors un ensemble d'arguments complètement différents, cela vous permet de voir les options standards, les options avancées, les cibles de l'exploit, et les payloads compatibles. La commande **check** invoque le mode de vérification de vulnérabilité de l'exploit sélectionné. La commande exploit lance actuellement l'exploit sélectionné.

Environnements

Le système d'environnement est un composant principal du Framework; les interfaces l'utilise pour configurer diverses options, les payloads l'utilise pour les opcodes, les exploits l'utilise pour définir les paramètres, et il est utilisé en interne pour passer les options entre les modules. Le système d'environnement est logiquement divisé en un environnement Global et Temporaire.

Chaque exploit maitient son propre environnement Temporaire, qui surpasse l'environnement Global. Quand vous choisissez un exploit avec la commande **use**, l'environnement Temporaire pour cet exploit est chargé et le précédent est sauvegardé en dehors. Si vous repasser à l'exploit précédent, l'environnement Temporaire de cet exploit est rechargé.

Environnement Global

L'environnement Global est accédé par la console via les commandes **setg** et **unsetg**. L'exemple suivant montre l'état de l'environnement Global après une installation récente. Lancer **setg** sans arguments affiche l'environnement global courant, lancer **unsetg** sans arguments va effacer tout l'environnement global. Les paramètres deconfiguration sont chargés automatiquement quand l'interface se lance.

```
+ -- --=[ msfconsole v2.2 [34 exploits - 33 payloads]

msf > setg
AlternateExit: 2
DebugLevel: 0
Logging: 0
msf >
```

Environnement Temporaire

L'environnement Temporaire est accédé via les commandes **set** et **unset**. Cet environnement ne s'applique seulement qu'au module d'exploit actuellement chargé; passer à un autre exploit via la commande **use** va entrainer l'environnement Temporaire du module en cours à être désactivé et remplacer par l'environnement du nouveau module.

Si aucun exploit n'est actuellement actif, les commandes **set** et **setg** ne seront pas disponibles. Repasser au module d'exploit originel entrainera la restauration

de l'environnement originel. Les environnements temporaires inactifs sont simplement enregistrés en mémoire et activés lorsque leur module associé est choisi. Les exemples suivants montrent comment la commande **use** sélectionne un exploit actif et comment la commande **back** ramène au mode principal.

```
msf > use apache_chunked_win32
msf apache_chunked_win32 > set
msf apache_chunked_win32 > set FOO BAR
FOO -> BAR
msf apache_chunked_win32 > set
FOO: BAR
msf apache_chunked_win32 > back
msf > use poptop_negative_read
msf poptop_negative_read > set
msf poptop_negative_read > back
msf > use apache_chunked_win32
msf apache_chunked_win32 > set
FOO: BAR
msf apache_chunked_win32 >
```

Environnement Sauvegardé

La commande **save** peut être utilisée pour enregistrer l'environnement Global et tous les environnements Temporaires sur le disque. L'environnement sauvegardé est écrit dans ~/.msf/config et sera chargé lorsque n'importe quelle interface utilisateur est exécutée.

Utiliser les Environnements Efficacement

Cet environnement système fractionné vous permet de gagner du temps pendant le développement d'un exploit et le test de pénétration. Les options communes des exploits peuvent être définies une fois dans l'environnement Global et utilisées automatiquement pour n'importe quel exploit que vous utiliserez par la suite.

L'exemple suivant montre comment les environnements globaux LPORT, LHOST, et PAYLOAD peuvent être utilisés pour gagner du temps lors de l'exploitation d'un ensemble de cibles basées sur Windows. Si cet environnement était défini et qu'un exploit Linux était utilisé, l'environnement Temporaire (via **set** et **unset**) pourrait être utilisé pour surpasser ces valeurs par défaut.

```
msf > setg LPORT 1234
LPORT -> 1234
msf > setg LHOST 192.168.0.10
LHOST -> 192.168.0.10
msf > setg PAYLOAD win32_reverse
PAYLOAD -> win32_reverse
msf > use apache_chunked_win32
msf apache_chunked_win32(win32_reverse) > show options
Exploit and Payload Options
===========================

 Exploit:    Name      Default    Description
 --------    ------    -------    ------------------
 optional    SSL                  Use SSL
 required    RHOST                The target address
 required    RPORT     80         The target port

 Payload:    Name        Default    Description
 --------    --------    -------    ----------------------------------------
 optional    EXITFUNC    seh                Exit technique: "process", "thread", "seh"
 required    LPORT       1234               Local port to receive connection
 required    LHOST       192.168.0.10       Local address to receive connection
```

Variables d’ Environnement

L’environnement peut être utilisé pour configurer plein d’aspets du Framework, ordonnés des paramètres de l’interface utilisateur aux options spécifiques de timeout dans l’API de socket réseau. Cette section décrit les variables d’environnement les plus couramment utilisées.

DebugLevel

Cette variable est utilisée pour contrôler la verbosité des messages de debugging rendus par les composants du Framework. Définir cette valeur à 0 empêchera les messages d’être affichés (valeur par défaut). Les valeurs supportées par la variable DebugLevel sont comprises entre 0 et 5.

Logging

Cette variable est utilisée pour activé ou désactivé la journalisation de la session. Les journaux de session sont enregistrés dans ~/.msf/logs par défaut, le répertoire peut être changé en utilisant la variable d’environnement LogDir. Vous pouvez utiliser l’utilitaire **msflogdump** pour voir les logs de session générés. Ces logs contiennent l’environnement complet pour l’exploit ainsi que les timestamps par paquet.

LogDir

Cette option spécifie le répertoire dans lequel seront enregistrés les logs. C’est par défaut ~/.msf/logs. Il y a deux types de fichiers de logs, le log principal et les logs de session. Le log principal va enregistré chaque action significative effectuée par l’interface console. Un nouveau log de session sera créé pour chaque tentative d’exploitation réussie.

Encoder

Cette variable peut être définie par une liste d’Encodeurs préférés séparés par une virgule. Le Framework essaiera cette liste d’Encodeurs en premier (dans l’ordre), puis échouera à chaque Encodeur restant. Les Encodeurs peuvent être listés avec la commande **show encoders**.

```
msf> set Encoder ShikataGaNai
```

EncoderDontFallThrough

Cette option demande au Framework de ne pas échouer aux Encodeurs restant si toute la liste des encodeurs préférés échoue. Ceci est utile pour conserver votre caractère furtif sur un réseau, et ne pas échouer accidentellement sur un Encodeur non désiré du fait que votre Encodeur préféré à échouer.

Nop

Cela a le même comportement que l'Encodeur décrit ci-dessus, sauf qu'il est utilisé pour spécifier une liste de modules générateurs de Nop préférés. Les générateurs de Nop peuvent être listés avec la commande **show nops**.

```
msf> set Nop Pex
```

NopDontFallThrough

Cette option a le même comportement que le EncoderDontFallThrough, sauf qu'elle s'applique à la liste des Nop préférés.

RandomNops

Cette option permet l'utilisation de nop aléatoires au lieu de l'opcode nop standard. RandomNops devrait être stable avec tous les modules d'exploit inclus dans le Framework.

ConnectTimeout

Cette option vous permet de définir le timeout de connection pour les sockets TCP. Cette valeur vaut 10 par défaut et pourrait nécessiter d'être augmentée pour exploiter les systèmes à travers des liens lents.

RecvTimeout

Cette option définit le nombre maximum de secondes autorisé pour les lectures de socket qui spécifient la valeur spécifique de longueur -1. Elle pourrait devoir être augmentée si vous exploitez des systèmes avec une connection lente et rencontrez des problèmes.

RecvTimeoutLoop

Cette option définit le nombre maximum de secondes pour attendre des données sur un socket avant de le renvoyer. A chaque fois que des données sont reçus pendant cette période, la boucle recommence. Elle pourrait devoir être augmentée si vous exploitez des systèmes avec une connection lente et rencontrez des problèmes.

Proxies

Cette variable d'environnement force tous les sockets TCP à passés pae cette chaîne de proxys. Le format de la chaîne est type:adresse:port pour chaque proxy, séparés par des virgules. La version 2.2 inclue le support des proxys de types socks4 et http.

ForceSSL

Cette variable d'environnement force tous les sockets TCP à utiliser le protocole SSL. C'est utile seulement quand un module d'exploit ne fournit pas l'option utilisateur "SSL".

UdpSourceIp

Cette variable d'environnement peut être utilisée pour contrôler l'adresse IP source depuis laquelle sont envoyés tous les datagrammes UDP. Cette option n'est effective seulement avec les exploits basés sur UDP (MSSQL, ISS, etc). Cette option est dépendante du fait d'être capable d'ouvrir un raw socket, ce qui n'est normalement disponible qu'aux utilisateurs root ou administrateurs. Dans la version 2.2, cette fonctionnalité ne fonctionne pas avec l'environnement Cygwin.

NinjaHost

Cette variable d'environnement peut être utilisée pour rediriger toutes les connections payloads sur un serveur socketNinja. Cette valeur devrait être l'adresse IP du système faisant tourner la console socketNinja (perl sockectNinja.pl –d).

NinjaPort

Cette variable d'environnement peut être utilisée avec la variable NinjaHost pour rediriger les connections payloads sur un système faisant tourner la console socketNinja. Cette valeur devrait être le numéro de port de la console socketNinja.

NinjaDontKill

Cette option peut être utilisée pour exploiter différents systèmes en même temps et est particulièrement utile en lançant un exploit basé sur UDP sur une adresse de broadcast réseau.

AlternateExit

Cette option est une rustine au bug trouvé dans certaines versions de l'interpréteur Perl. Si l'interface *msfconsole* plante avec un segmentation fault en sortie, essayez en mettant la valeur de cette variable à "2".

Utilisation du Framework

Choisir un Module Exploit

Depuis l'interface *msfconsole*, vous pouvez voir les modules d'exploit disponibles avec la commande **show exploits**. Sélectionner un exploit avec la commande **use**, en spécifiant le nom court dumodule comme argument. La commande **show info** peut être utilisée pour voir les informations sur un module d'exploit spécifique.

Configurer l'Exploit Actif

Lorsque vous avez sélectionné un exploit, la prochaine étape est de déterminer les options requises. Cela peut être fait avec la commande **show options**. La plupart des exploits utilisent **RHOST** pour spécifier l'adresse cible et **RPORT** pour définir le port cible. Utilisez la commande **set** pour configurer les valeurs appropriées pour toutes les options requises. Si vous avez des questions sur ce que fait une certaine option, référez-vous au code source du module. Des options avancées sont disponibles avec quelques modules d'exploit, elles peuvent être affichées avec la commande **show advanced**.

Vérifier les options de l'Exploit

La commande **check** peut être utilisée pour déterminer si le système cible est vulnérable au module d'exploit actif. Cette une manière rapide pour vérifier que toutes les options sont correctement règlées et que la cible est actuellement vulnérable à l'exploitation. Tous les modules d'exploits n'intègre pas la fonctionnlité de vérification. Dans beaucoup de cas il est presque impossible de déterminer si un service est vulnérable sans l'exploiter directement. Une commande **check** ne devrait jamais entrainerle service à planter ou devenir non disponible. Beaucoup de modules affichent simplement des informations de versions et considèrent que les avez analysées avant d'entamer le processus.

Choisir le Payload

Le payload est le code qui sera exécuté sur le système cible après une tentative d'exploitation réussie. Utilisez la commande **show payloads** pour lister tous les payloads comptibles avec l'exploit en cours. Si vous êtes derrière un pare-feu, vous pourriez vouloir utiliser un payload bind shell, si votre cible est derrière un pare-feu et vousnon, vous devriez utiliser un payload de connection inverse. Vous pouvez utiliser la commande **info payload_name** pour voir les informations détaillées d'un payload déterminé.

Lorsque vous avez choisi un payload, utilisez la commande **set** pour spécifier le nom du modle payload comme valeur de la variable d'environnement **PAYLOAD**. Quand le payload a été défini, utilisez la commande **show options** pour afficher toutes lesoptions du payload. La plupart des payloads nécessite au moins une option.

Les options avancées sont fournies par une poignée d'options de payload, utilisez la commande **show advanced** pour les voir.

Choisir la Cible

Beaucoup d'exploits vont nécessiter que la variable d'environnement **TARGET** soit définie sur le numéro d'index de la cible désirée. La commande **show targets** va lister toutes les cibles fournies par le module d'exploit. Beaucoup d'exploits utiliseront par défaut une sorte de force brute sur la cible; cela ne sera pas désiré dans toutes les situations.

Lancer l'Exploit

La commande **exploit** lancera l'attaque. Si tout ce passe bien, votre payload sera exécuté et vous fournira probablement un shell de commande interactif sur le système exploité.

L'Interface de Lignes de Commande

Si la console ne répond pas à vos besoins, vous devriez essayer l'interface msfcli.

Cette interface prendune chaîne de correspondance comme premier argument, suivie par les options au format VAR=VAL, et enfin un code d'action pour spécifier ce qui doit être fait. La chaîne de correspondance est utilisée pour déterminer quel exploit vous voulez lancer; dans le cas où plusieurs modules correspondent, une liste des modulespossibles sera fournie.

Le code d'action est une lettre; **S** pour résumé(summary), **O** pour options, **A** pour options avancées, **P** pour payloads, **T** pour cibles (targets), **C** pour tenter une vérification de vulnérabilité (check), et **E** pour exploit. L'environnement sauvegardé sera chargé et utilisé au démarrage, cela vous permet de configurer différentes options par défaut dans l'environnement Global de la *msfconsole*, les enregistrées, et de les utiliser dans l'interface *msfcli*.

L'interface de ligne de commandes est bien faite pour l'automatisation d'exploitation et le test de batchs, combinée avec un payload customisé et un scanner intelligent, elle peut être impitoyable :)

L’interface Web

L’interface *msfweb* est un serveur web fonctonnel qui vous permet de lancer des attaques depuis votre naviguateur internet. Cette interface est encore très primitive, maispeut être utile pour les utilisateurs travaillant en équipe (pen-testing, etc). La connection à l’hôte exploité est proxifiée sur un port aléatoire en écoute sur le serveur web et un lien par protocole telnet est fourni à l’utilisateur sur cet écouteur créé dynamiquement.

L’interface msfweb ne fournit pratiquement aucune sécurité quelle quelle soit, n’importe qui sur le réseau peut se connecter au serveur web ou au port dynamqie de proxy sélectionné. La configuration par défaut est d’écouter sur l’adresse loopback seulement, cela peutêtre modifié en passant l’option –a avec une valeur consistant en adresse:port. Tout comme l’interface de ligne de commandes, l’environnement sauvegardé est chargé au démarrage et peut affecté les paramètres des modules. Nous ne recommandons pas que l’interface msfweb soit utilisée dans des environnements de production.

Fonctionnalités Avancées

Cette section couvre quelques unes des fonctionnalités avancées qui peuvent être trouvées dans cetet version. Ces fonctions peuvent être utilisées avec n'importe quel exploit compatible et souligne la puissance de développement de code d'attaque en utilisant un framework d'exploit.

Les Payloads Python InlineEgg

La librairie InlineEgg est une classe Python pour générer dynamiquement de petits programmes en langage assembleur. L'utilisation la plus courante de cette librairie est de créer rapidement des payloads avancés d'exploit. Cette librairie fut développée par Gera pour être utilisé avec le produit Core ST's Impact. Core a mis à disposition cette librairie au public sous une licence d'exploitation non commerciale.

Le Metasploit Framework supporte les payloads InlineEgg à travers l'interface du module ExternalPayload; cela permet un support transparent si le langage de script Python est installé. Pour activer les payloads InlineEgg, la variable d'environnement **EnablePython** doit être définie avecune valeur différente de 0. Ce changement fut réalisé dans la version 2.2 pour accélérer le processus de redémarrage de module.

Cette version inclue des exemples InlineEgg pour Linux, BSD, et Windows. Les exemples Linux sont linx86_reverse_ie, linux86_bind_ie, et linux86_reverse_xor. Ces payloads peuvent être sélectionnés et utilisés de la même manière que tout autre payload. Les contenus des payloads sont générés dynamiquement pas les scripts Python dans le sous répertoire payloads/external. Les payloads BSD fonctionne pratiquement comme leurs équivalents Linux.

L'exemple InlineEgg Windows est nommé win32_reverse_stg_ie et fonctionne d'une façon légèrement différente. Ce payload a une option nommée **IEGG**, cette option spécifie le chemin du script Python InlineEgg qui contient votre payload final. C'est un payload en étapes; la première étape est une connection inverse standard, la seconde étape envoie l'adresse de GetProcAddress et LoadLibraryA à travers la connexion, et la troisième étape est générée localement et envoyée par le réseau. Un exemple de script InlineEgg est inclus dans l sous répertoire payloads/external, appellé 'win32_stg_winexec.py'. Pour plus d'informations sur le InlineEgg, veuillez voir le site internet deGera, localisé ici:

http://community.corest.com/~gera/ProgrammingPearls/InlineEgg.html

Injection ELF Impure (Impurity ELF Injection)

L'impureté (Impurity) fut un concept développé par Alexander Cuttergo qui dévrivit une méthode pour charger et exécuter un nouvel exécutable ELF en mémoire. Cette technique permet à des payloads arbitraires complexes d'être écrits en C standard, la seule nécessité est un chargeur de payload spécial. Le Framework inclu un chargeur Linux pour les exécutables Impurs, le payload s'appelle **linx86_reverse_impurity** et requiert que l'option **PEXEC** soit paramètrée avec le répertoire de l'exécutable. Les exécutables Impurs doivent être compilées d'une façon spéciale, veuillez voir la documentation dans le sous répertoire src/shellcode/linux/impurity pour plus d'informations sur ce processus. L'application "shelldemo" incluse dans le sous répertoire data vous permet de lister, accéder, lire, écrire, et ouvrir des handles de fichiers dans le processus exploité. Le message originel de la mailing list est archivé en ligne ici:

http://archives.neohapsis.com/archives/vuln-dev/2003-q4/0006.html

Chainage de Proxies

Le Framework inclu un support transparent des proxies TCP, cette version possèdes des routines handler pour les serveurs HTTP CONNECT et SOCKSv4. Pour utiliser un proxy avec un exploit donné, la variable d'environnement **Proxies** doivent être paramètrées. La valeur de cette variable est une liste de serveurs proxy séparés par une virgule, où chaque serveur est au format hôte:port. Les valeurs de type sont 'http' pour HTTP CONNECT et 'socks4' pour SOCKS v4. La chaîne de proxies peut être den'importe quelle taille; des tests ont démontrés que le système était stable avec plus de 500 proxies SOCKS et HTTP configurés au hasard dans une liste. La chaîne de proxies masque uniquement la requête de l'exploit, la connexion automatique au payloas n'est pas relayée à travers la chaîne de proxies pour l'instant.

Les Payloads Win32 UploadExec

Alors que les sytèmes UNIX possèdent en principe tous les outils don't avez besoin après l'exploitation, les sytèmes Windows manquent cruellement d'un kit d'outils en lignes de commande. Les payloads UploadExec inclus dans cette version vous permettent de simultanément exploiter un système Windows, uploader votre outil favori, et de l'exécuter, tout ca par le biais de la connexion socket du payload. Combinée avec un rootkit autoextractible ou un interpréteur de langage de script (perl.exe!), cela peut s'avérer une fonction très puissante.

Payloads Win32 de DLL Injection

La version 2.2 du Framework inclu un payload en étapes qui est capable d'injecter une DLL modifiée en mémoire en combinaison avec n'importe quel exploit Win32. Ce payload permet qu'aucun fichier ne soit écrit sur le disque, la DLL est chargée directement en mémoire et est lancée comme un nouveau thread dans le processus exploité. Ce payload a été éveloppé par Jarkko Turkulainen et Matt Miller et est une des meilleures techniques post-exploitation développée à ce jour. Pour créer une DLL qui pourra être utilisée avec ce payload, utiliser l'environnement de développement de votre choix et générez une DLL Win32 standard. Cette DLL doit exporter une fonction appellée Init qui prend un seul argument, une valeur integer qui contient le descripteur du socket de la connexion du payload. La fonction Init devient le point d'entrée pour le nouveau thread dans le processus exploité. Quand le processus est complété, il doit retourner et permettre à la fin du chargeur (loader stub) de quitter le processus en accord avec la variable d'environnement **EXITFUNC**.

DLL Injection d'un Serveur VNC

L'un des premiers payloads de DLL injection développé fut un serveur VNC customisé. Ce serveur a été écrit par Matt Miller et est basé sur le code source de RealVNC. Des modifications additionnelles furent réalisées pour permettre au serveur de fonctionner avec des services réseaux exploités non interactifs. Ce payload vous permet d'accéder immédiatement au bureau d'un système exploité en utilisant pratiquement n'importe quel exploit Win32. La DLL est chargée dans le processus distant en utilisant n'importe quel systèmes de chargement par étapes, lancé en tant que nouveau thread dans le processus exploité, et les écouteurs de requêtes de client VNC sur le même socket utilisés pour lancer la DLL. Le Framework écoute simplement sur une socket locale pourun client VNC et proxifie les données à travers la conexion du payload vers le serveur.

Le serveur VNC va tenter d'obtenir l'accès complet au bureau interactif courant. Si la première tentative échoue, il appellera RevertToSelf() puis renouvellera à nouveau la tentative. Si il échoue encore une fois à obtenir l'accès complet à ce bureau, il tombera dans un mode lecture seule. En mode lecture seule, l'utilisateur du Framework peut voir le contenu du bureau, mais pas interagir avec celui-ci. Si l'accès complet est obtenu, le serveur VNC lancera un shell de commandes sur le bureau avec les privilèges du service exploité. Ceci est utile dans les situations où un utilisateur sans privilèges est sur le bureau interactif, mais que le service exploité tourne avec les privilèges Système.

Si il n'y a pas d'utilisateur interactif connecté au système ou que l'écran a été verouillé, le shell de commandes peut être utilisé pour lancer l'explorer.exe de toute façon. Cela peut perturber réellement des utilisateurs quand un écran de connexion a également un menu démarrer. Si le bureau interactif est changé, du fait que quelqu'un se connecte au système ou que l'écran est verouillé, le serveur VNC déconnectera le client. Les prochaines versions tenteront de suivre un changement de bureau.

Pour utiliser les payloads d'injection VNC, indiquez le chemin complet du serveur VNC comme valeur de l'option de la **DLL**. Le serveur VNC peut être trouvé dans le sous répertoire data d'installation du Framework et est nommé 'vncdll.dll'. Le code source de la DLL peut être trouvé dans le sous répertoire src/shellcode/win32/dllinject/vncinject d'installation du Framework.

Informations Supplémentaires

Site Internet

Le site internet metasploit.com est le meilleur endroit pour vérifier les mises à jour de modules et nouvelles versions. Ce site internet héberge également la base de données Opcode et une collection de shellcodes Windows décente.

Mailing List

You can subscribe to the Metasploit Framework mailing list by sending a blank email to framework-subscribe [at] metasploit.com. This is the preferred way to submit bugs, suggest new features, and discuss the Framework with other users.

Développeurs

Si vous souhaitez être impliqué dans le développement de la prochaine version du Framework, veuillez contacter les développeurs. Ils peuvent être joints à:

msfdev [at] metasploit.com.

Traduction personnelle française : Jérôme ATHIAS – jerome(@]ATHIAS.fr