Infocus
< http://www.securityfocus.com/infocus/1789 >

Metasploit Framework (Part One of Three)
The Prometheus Of Exploitation

by Pukhraj Singh *and* K.K. Mookhey
last updated July 12, 2004

Metasploit Framework (Partie 1 sur 3)

Traduction personnelle française :
Jérôme ATHIAS
Dernière mise à jour : 16/08/2004

sploit
(n.) Exploit. A defect in the game code (see bug) or design that can be used to gain unfair advantages. *(Source: Dictionary of MMORPG Terms)*

Au moment présent, la communauté de développement d'exploits (pirates (hackers) et professionnels de la sécurité informatique) est plus sensible que jamais. Le délai entre la publication d'un bulletin d'alerte et le développement d'un exploit a diminué considérablement. Le développement d'exploits, qui fut considéré comme un art, est parvenu à la portée d'une large population. L'administrateur de sécurité réseau se doit d'être plus vigilant que jamais face à un ennemi qui a toujours une longueur d'avance avec le tout dernier exploit entre ses mains.

Les utilitaires de développement d'exploits et les frameworks automatisés pour tester et simuler les exploits constituent la nécessité actuelle. Le Metasploit Framework (MSF) répond à cette tâche. Ca dernière version possède une agilité et des muscles qui n'ont rien à envier aux très coûteux logiciels commerciaux équivalents et la faculté de coder un exploit en un minimum de temps, du fait d'une interface de développement parfaitement adaptée. Avec un environnement de développement d'exploits complet, des exploits fonctionnels, des tailles efficaces et des entêtes (handlers) ajoutés, cela en fait L'outil que doivent utiliser les testeurs d'infiltration.

Cet article fournit un aperçu des bases des frameworks de développement d'exploits, avec un intérêt particulier sur le Metasploit Framework et comment il peut être utiliser pour gagner du temps et économiser des ressources. Nous décrirons son utilisation avec des illustrations graphiques, détaillerons les différentes commandes disponibles, décrirons les fonctionnalités proposées, donnerons des exemples pratiques, et plus important encore, nous utiliserons ces possibilités pour développer de nouveaux exploits et tester de nouvelles techniques.

1. Prologue

Je souhaiterais débuter mon article en faisant référence à des événements récents. Le bulletin Microsoft (MS04-011) décrit et résout un nombre important de vulnérabilités de

sécurité dans différents systèmes d'opérations Windows [ref 1]. Deux d'entre elles qui m'ont intéressées sont le SSL PCT et les overflows de l'Authorité de Sécurité Locale (Local Security Authority) qui peuvent conduire à la compromission à distance. Comme dans la plupart des cas, presque immédiatement, des exploits fonctionnels ont été mis publiquement à disposition, laissant les administrateurs et les professionnels de la sécurité non-avertis et non-préparés.

En vous mettant à la place d'un administrateur système d'une société d'informatique banale, les exploits s'ajoutent toujours de plus en plus au fardeau de la gestion de la sécurité. Déjà dit et fait, c'est une course sauvage où les attaquants malicieux sont au devant du jeu, mais part une approche méthodique, le professionnel en sécurité peut inverser la donne.

Les patchs de sécurité, IDS, pare-feu (firewalls), etc ne constituent pas les seuls critères de protection. Cédant face à la pression de la situation, beaucoup de coins et recoins du réseau peuvent être rendus non-protégés et non-verrouillés, ce qui est généralement l'origine d'une compromission. C'est ce qui arrive lorsque l'on entend parler d'un grand réseau compromis par des pirates utilisant des vulnérabilités connues. Et c'est exactement la raison pour laquelle le ver informatique Sasser a contaminé 250 000 ordinateurs, même deux semaines après que Microsoft ait mis à disposition le patch de sécurité indiqué comme hautement important.

De mon point de vue, la solution est l'habituelle approche, "pensez comme un pirate". Le testeur d'infiltration devrait se lancer dans une fête de pirate en testant son propre réseau, dans ce que j'appelle *Threat Evasion Penetration Testing (Test de menaces et d'intrusion)*. C'est là que les frameworks d'exploits entrent en jeu, en automatisant l'art de l'exploitation.

2. Travail sur le terrain

Les exploits sont encore rattachés à la crainte. Les testeurs occupés considèrent sans intérêt l'idée du développement d'exploit comme passe-temps d'un pirate. C'est vrai d'une certaine manière. Le développement d'exploits en lui-même est un art. Il requiert des connaissances primordiales, de la patience, beaucoup de temps et par dessus tout l'esprit immortel d'apprendre par essai-et-erreur.

2.1 Organisation de la mémoire

Les techniques d'exploitation basiques peuvent être méthodiquement catégorisées, comme tout autre problème technique. Avant d'aller plus loin, dans tous les cas, le lecteur devra connaître le processus basique de l'organisation de la mémoire [ref 2]. Un processus tournant en mémoire a les sous-structures suivantes :

> ***Code*** est le segment en lecture seule qui contient le code exécutable compilé du programme.
>
> ***Data*** et ***BSS*** sont les segments en écriture contenant les segments de données et les variables statiques, globales, initialisés et non-initialisés.
>
> ***Stack*** est une structure de données basée sur l'ordre "Premier-Rentré-Premier-Sorti". Les éléments sont posés (*push*) et retirés (*pop*) depuis le sommet de la pile. Un pointeur de pile (Stack Pointeur : SP) est un registre qui pointe sur le sommet de la pile (dans la majorité des cas). Lorsque une donnée est posée sur la pile, le SP

pointe dessus (le sommet de la pile). La pile grossit vers des adresses mémoires négatives. Cela est utilisé pour stocker le contexte d'un processus. Un processus pose toutes ses données locales et dynamiques sur la pile. Le pointeur d'instruction (Instruction Pointer : IP) est un registre utilisé pour pointer sur l'adresse de la prochaine instruction à exécuter. Le processeur examine l'IP à chaque fois pour trouver la prochaine instruction à exécuter. Lorsqu'une redirection abrupte se met en place (généralement due à un *jmp* ou un *call*), l'adresse de la prochaine instruction, après le retour de la redirection, peut être perdue. Afin de prévenir de ce problème, le programme stocke l'adresse de la prochaine instruction qui doit être exécutée (après un retour de jmp ou call) sur la pile, et elle est appelée l'adresse de retour (return address) (implémentée à travers l'instruction assembleur *RET*). C'est ainsi qu'un programme normal contenant beaucoup de fonctions d'appels (calls) et d'instructions *goto* conserve le chemin correct d'exécution.

Le ***Heap*** est basiquement le reste de l'espace mémoire alloué au processus. Il stocke les données qui ont une durée de vie comprise entre les variables globales et les variables locales. L'allocateur et le désallocateur travaillent respectivement pour assigner de l'espace aux données dynamiques et vider la mémoire heap [ref 3].

Voilà pour un bref survol des bases de l'organisation d'un processus. Maintenant, je décris quelques techniques utilisées de manière récurrente pour troubler l'harmonie de l'organisation d'un processus.

2.2 Buffer overflows (Dépassements de capacité de tampon)

Ce terme donne des frissons à toute personne qui a eu affaire à eux, que ce soit un programmeur, un testeur d'application ou un administrateur de sécurité. Certains disent qu'il s'agit du plus gros risque de sécurité de la décennie [ref 4]. La technique d'exploitation est franche et meurtrière. La pile du programme stocke les données dans l'ordre, ainsi les paramètres passés à la fonction sont enregistrés en premier, puis l'adresse de retour, puis le SP précédent et plus tard, les variables locales. Si des variables (comme des tableaux) sont passées sans vérifications des limites, elles peuvent être amenées au débordement en poussant une grande quantité d'informations, ce qui corrompt la pile, conduisant à la sur écriture de l'adresse de retour et par conséquent à un problème de segmentation (segmentation fault). Si le tour est astucieusement joué, nous pouvons modifiés les buffers pour pointer sur n'importe quel emplacement, entraînant l'exécution de code arbitraire [ref 5].

2.3 Heap overflows (Dépassements de capacité du Heap)

La mémoire allouée dans un heap est organisée en une liste doublement liée. En provoquant un dépassement de capacité, nous pouvons modifier les pointeurs de la liste liée pour pointer sur la mémoire. Les heap overflows sont difficiles à exploiter et sont plus courants dans Windows du fait qu'ils contiennent plus de données préominentes qui peuvent être exploitées. Dans le cas d'un système d'allocation de mémoire malloc, les informations concernant la mémoire disponible et allouée sont stockées dans le heap. Un dépassement peut être déclenché en exploitant cette information de gestion, nous permettant ensuite d'écrire dans des emplacements mémoires aléatoires, ce qui peut entrainer l'exécution de code [ref 6].

Comment est donc déclenché ce dépassement? Il existe beaucoup d'armes dans l'armurerie

comme les chaînes (strings) et les fonctions de manipulation de chaînes, les pointeurs nuls (null pointers), dépassement d'entiers (integer overflows), problèmes de signatures (signed issues) et certaines conditions qui peuvent être utiles pour générer des conditions d'exceptions dans un processus [ref 7].

J'insiste sur le fait que cet article ne se veut pas un guide exhaustif des différentes techniques d'exploitation. Nous fournissons juste un rapide aperçu de ce qui est important, afin de bien comprendre les points à venir dans les articles suivants. Ce ne sont que des indicateurs pour une référence future.

3. La naissance d'un Exploit

Vient maintenant la partie la plus excitante, le codage d'un exploit. Le corps ou structure d'un exploit peut être divisé en différents composants, comme décrit en **Figure 1** [ref 8]. Nous présentons une petite collection d'exploits qui nous aidera à analyser la figure suivante.

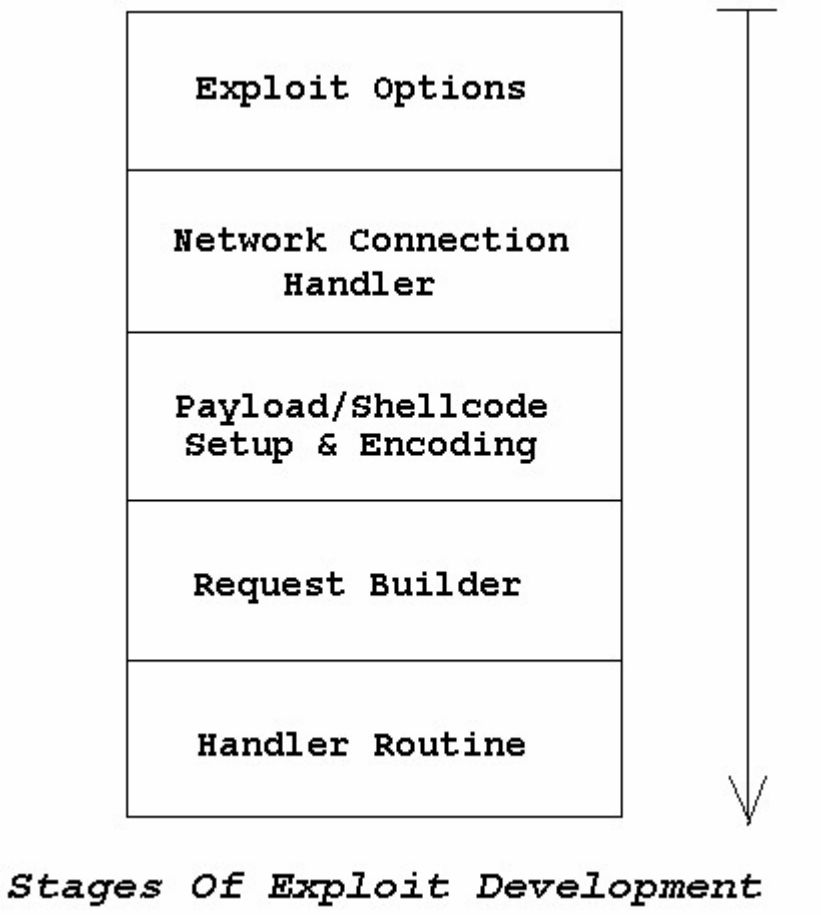

Figure 1

3.1 Shellcode

Il constitue le code utile qui sera exécuté après l'exploitation. Dans la plupart des cas, nous redirigeons le chemin d'exécution, ainsi le code injecté sera exécuté [ref 9]. A partir de là, l'adresse de retour est détournée pour pointer sur ce shellcode. Il est constitué d'instructions assembleurs encodées dans une chaîne au format binaire qui réalisent des opérations comme créer un shell. Un bon shellcode se doit d'être un compromis entre à la

fois, la taille et la complexité. Il y a énormément de vérifications à effectuer pendant la création d'un payload comme conserver un test des caractères réservés. De nos jours, les payloads ont été adaptés pour être extrememement petits et nécessitent moins de place. Ils peuvent exécuter beaucoup d'opérations complexes de l'ouverture d'un socket en écoute au lancement d'une compilation sur l'ordinateur distant.

3.2 Vecteur d'Injection (Injection vector)

Le pointeur ou offset où le shellcode est placé dans un processus et sur lequel l'adresse de retour est modifiée pour pointer dessus.

3.3 Le constructeur de requête (Request builder)

C'est le code qui déclenche l'exploit. Si il est lié à des fonctions de chaînes, alors les langages de script sont généralement préférés.

3.4 Handler Routine

Cette partie occupe générallement la majorité du code. C'est un handler pour le shellcode opérant des opérations comme se connecter à un shell distant (bindshell), ou connecter la console à un socket.

3.5 Handler des options utilisateur (User options handler)

C'est basiquement une interface utilisateur lui proposant différentes opérations de contrôle comme la sélection d'une cible, d'un offset, la verbosité, le débuggage et d'autres options. Cela constitue la majorité du code de l'exploit et le rend plus encombrant.

3.6 Handler de connection réseau (Network connection Handler)

Cela comprend les différentes routines qui conduisent les connections réseau comme la résolution de nom, la gestion de socket, le flux d'erreurs, etc.

Comme nous pouvons le voir, il existe beaucoup de code non nécessaire et répétitif qui rendent l'exploit volumineux et soumis à des risques d'erreurs.

4. Quelques problèmes courants

Dans la course au développement, beaucoup de problèmes sont rencontrés qui gênent le processus de développement de l'exploit. La mauvaise course des gens qui tentent de sortir l'exploit en premier conduit à beaucoup de code mauvais et compliqué.

Plusieurs exploits requièrent une bonne compréhension de concepts profonds et de la recherche, comme les exploits basés sur les protocoles réseau (RPC, SMB et SSL) et les APIs obscures. Comme peut d'informations sont révélées dans les bulletins d'alertes, il y a toujours un besoin d'expérience.

Trouver les valeurs cibles est également un gros casse tête qui implique un grand nombre

de tests et d'erreurs.

Enfin, la plupart des payloads sont codés en dur et la plus petite modification rend l'exploit inutilisable.

Un grand nombre de pare feu et de systèmes IPS détectent et bloquent les shellcodes.

Le temps est le principal souci, et certains exploits consomment beaucoup de temps et de concentration, ces deux aspects sont les atouts précieux d'un chercheur en sécurité.

Ceci étant dit, coder un exploit est un enfer dans un travail confus!

5. Allons-y!

Entrons dans le Metasploit Framework (MSF)! Conformément au guide *MSF User Crash Course* [ref 10];

> "*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.*"

De mon point de vue, le Framework Metasploit est une solution unique à tous les problèmes discutés précedemment. Le framework a évolué en une extension dans sa version 2.0. Il est plus stable, possède des fonctionnalités très attractives et une interface utilisateur très instinctive pour le développement d'exploits.

Les fonctionnalités majeures qui avantagent encore plus le MSF à travers les différentes options sont :

- Il est principalement écrit en Perl (avec quelques parties en assembleur, Python et en C), ce qui signifie un code propre et efficace et un développement rapide de plug-ins.
- Un support préintégré pour des extensions d'outils, librairies et des fonctions de debuggage, encoding, logging, timeouts et nops aléatoires et SSL.
- Une API d'exploit et un environnement compréhensibles, intuitifs, modulaires et extensibles.
- Des payloads mutli-plateformes et multi-fonctions hautement optimisés qui sont exécutables dynamiquement.
- Un support de handlers et callbacks avancé, qui réduit considérablement le code de l'exploit.
- Le support d'options réseaux et de protocoles variés qui peuvent être utilisés pour développer du code dépendant du protocole.
- Des exploits supplémentaires inclus, qui aident à tester des techniques d'exploitation et des exemples d'exploits développés.
- C'est un produit Open Source possédant une communauté de développeurs assurant le support.
- Support de fonctionnalités avancées et d'outils tiers comme InlineEgg, Impurity, UploadExec et du chaînage de proxies.

Il est clair que MSF est définitivement un outil que le testeur d'infiltration se doit de connaître. Il donne à l'art de l'exploitation un nouveau paradigme.

6. Installation

Actuellement, le Metasploit Framework fonctionne efficacement sur Linux et Windows. Il existe quelques problèmes de compatibilité, mais elles peuvent être ignorées. La dernière version est téléchargeable pour Windows et Linux à l'adresse http://www.metasploit.com/projects/Framework/downloads.html.

L'installation est triviale et intuitive, et les paquetages d'installations sont autoextractibles. L'installation simple est montrée en **Figure 2**. Dans le cas de Linux, décompresser l'archive (qui est au format framework-2.x.x.tar.gz), là où le répertoire du framework contient les binaires qui sont pour différents utilitaires. Lors du chargement sous Linux, il est indiqué que les modules Term::ReadLine::Gnu (pour le support de la complétion par tab) et Net::SSLeay (pour le support SSL) sont installés (ils sont trouvés dans le répertoires extras).

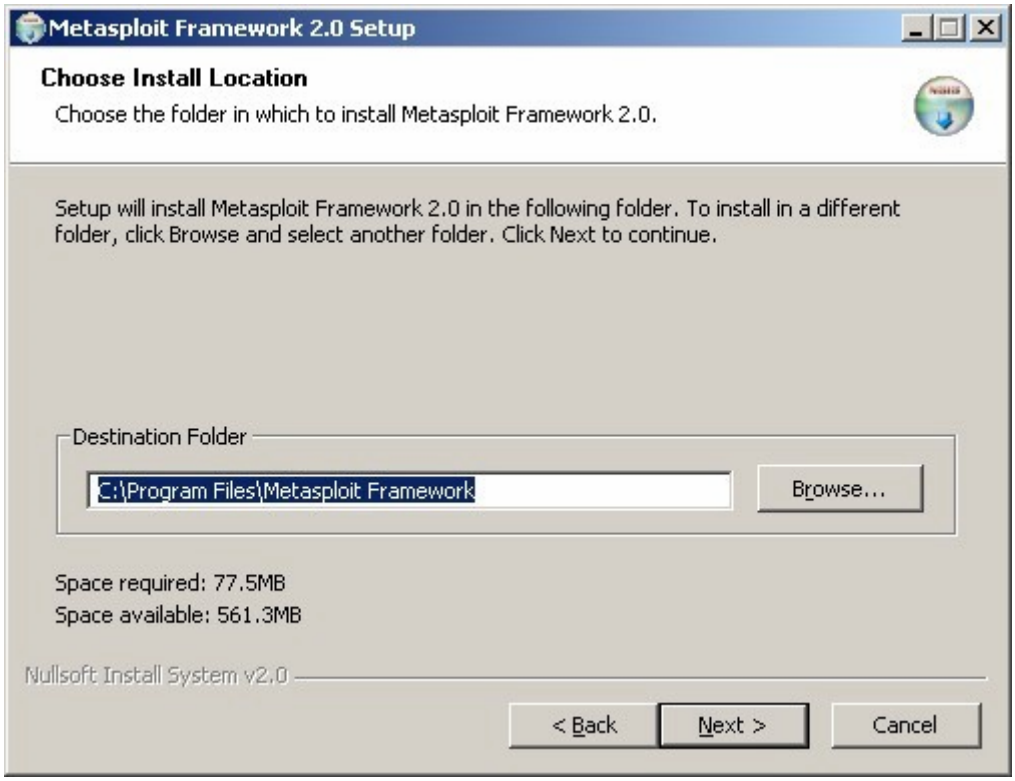

Figure 2

L'environnement Windows est basé sur un environnement Cygwin allégé, ce qui est une sage décision car cela fournit une console très portable à l'utilisateur. Néanmoins, il existe des problèmes avec le support d' Active State Perl, car il ne supporte que le Cygwin Perl. L'installation est regroupée dans un exécécutable, qui installe le Metasploit Framework dans le répertoire spécifié (voir Figure 2) et ajoute les raccourcis.

7. Conclusion de la Première Partie

Dans la première partie de cet article, nous avons fait un tour à travers les différentes technqiues d'exploitation qui sont couramment utilisées. Nous avons fouillé dans les bases du développement d'un exploit et vu comment le code d'un exploit peut être découpé en sous-structures logiques. Nous avons abordé les problèmes majeurs rencontrés dans le développement d'exploits et vu comment le Metasploit Framework se présente comme une

solution à ces problèmes. Nous avons également parlé de quelques unes de ses fonctionnalités et de son installation.

Plus tard, dans la seconde partie, nous exposerons un aperçu élaboré de son utilisation et de ses options variées, du lancement et de l'ajout de nouveaux exploits, des règlages de l'environnement et des autres fonctionnalités avancées du Metasploit Framework.

Références

1. Microsoft Security Bulletin http://www.microsoft.com/technet/security/bulletin/ms04-011.mspx

2. Stack, Pointers and Memory, Lally Singh http://www.biglal.net/Memory.html

3. A Memory Allocator, Doug Lea http://gee.cs.oswego.edu/dl/html/malloc.html

4. Buffer overflows likely to be around for another decade, Edward Hurley http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci860185,00.html

5. Buffer Overflows Demystified, Murat http://www.enderunix.org/docs/eng/bof-eng.txt

6. Badc0ded - How to exploit program vulnerabilities, http://community.core-sdi.com/~juliano/bufo.html

7. Once upon a free(), Phrack 57 Article 9 by Anonymous http://www.phrack.org/show.php?p=57

8. Presentation on Advanced Exploit Development at HITB, HD Moore (PDF) http://conference.hackinthebox.org/materials/hd_moore/HDMOORE-SLIDES.pdf

9. Designing Shellcode Demystified, Murat http://www.enderunix.org/docs/en/sc-en.txt

10. Crash Course User Guide for Metasploit Framework, (PDF) http://metasploit.com/projects/Framework/docs/CrashCourse-2.0.pdf

About the authors

Pukhraj Singh is a security researcher at Network Intelligence (I) Pvt. Ltd. His areas of interest include working with exploits, monitoring honeypots, intrusion analysis and penetration testing.

K. K. Mookhey is the CTO and Founder of Network Intelligence.

View more articles by K.K. Mookhey on SecurityFocus.

Comments or reprint requests can be sent to the editor.