Infocus
< http://www.securityfocus.com/infocus/1790 >

Metasploit Framework, Part Two

by Pukhraj Singh *and* K.K. Mookhey
last updated September 8, 2004

Metasploit Framework, Deuxième Partie

Traduction française par Jérôme ATHIAS
Dernière mise à jour : 11/09/2004

Note de l'éditeur: Ce document a été complètement réécrit, incluant des corrections importantes par rapport à l'article précédent, et traite des changements avec la version 2.2 du MSF.

1. Introduction

Dans la première partie de cette série d'articles, nous avons vu comment l'écriture d'exploits est encore difficile et prend beaucoup de temps. Nous avons vu les difficultés courantes rencontrées pendant le développement d'exploits et comment le Metasploit Framework peut résoudre bon nombre de ces problèmes. Cet article va commencé par une brève introduction à l'interface console et expliqué comment choisir et utiliser un module d'exploit. Puis nous examinerons le système d'environnement, comment il fonctionne, et quelles fonctions peuvent être activées via celui-ci.

2. Les Bases

Le MSF installé a trois environnements de travail, la *msfconsole*, l'interface *msfcli* et l'interface *msfweb*. Dans tous les cas, la principale zone de travail (et la préférable) pour MSF est la *msfconsole*. C'est une interface en lignes de commandes efficace qui possède son propre jeu de commandes et environnement système. Bien que le Framework fut conçu pour tourner sur un système de genre Unix, comme Linux ou BSD, il tourne également sur Windows grâce à l'environnement Cygwin. L'installateur Windows, du site metasploit.com, inclus une version allégée et pré configurée de Cygwin.

Pendant l'initialisation de la *msfconsole*, des vérifications standards sont effectuées. Si tout fonctionne correctement, nous verrons l'écran de la Figure 1.

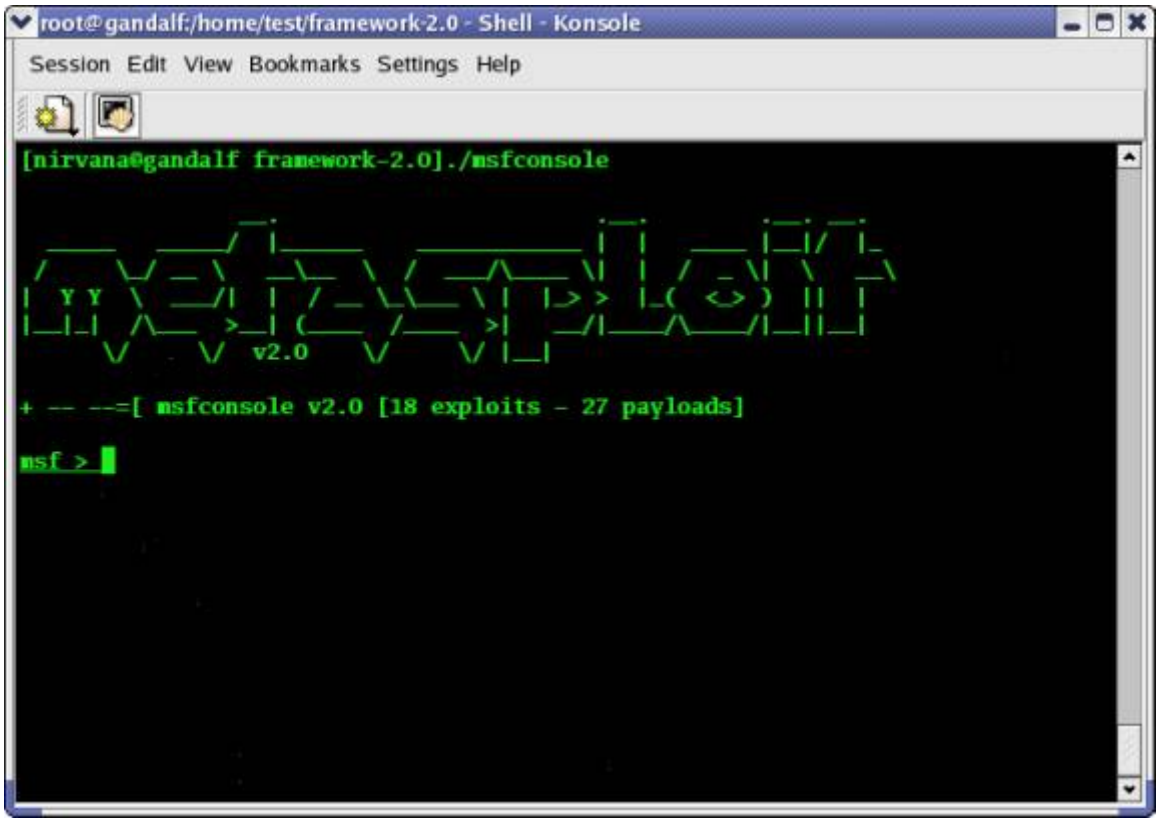

Figure 1

Maintenant, le prompt de commandes (msf>) de la *msfconsole* est actif. La console est très flexible, et si l'utilisateur entre n'importe quelle commande non reconnue, elle cherchera dans le PATH de l'environnement pour un exécutable correspondant. Si un fichier correspondant est trouvé, il est exécuté comme avec un prompt de commandes standard.

Instinctivement, en tapant la commande **help**, l'on obtient une liste des commandes disponibles comme présenté en Figure 2.

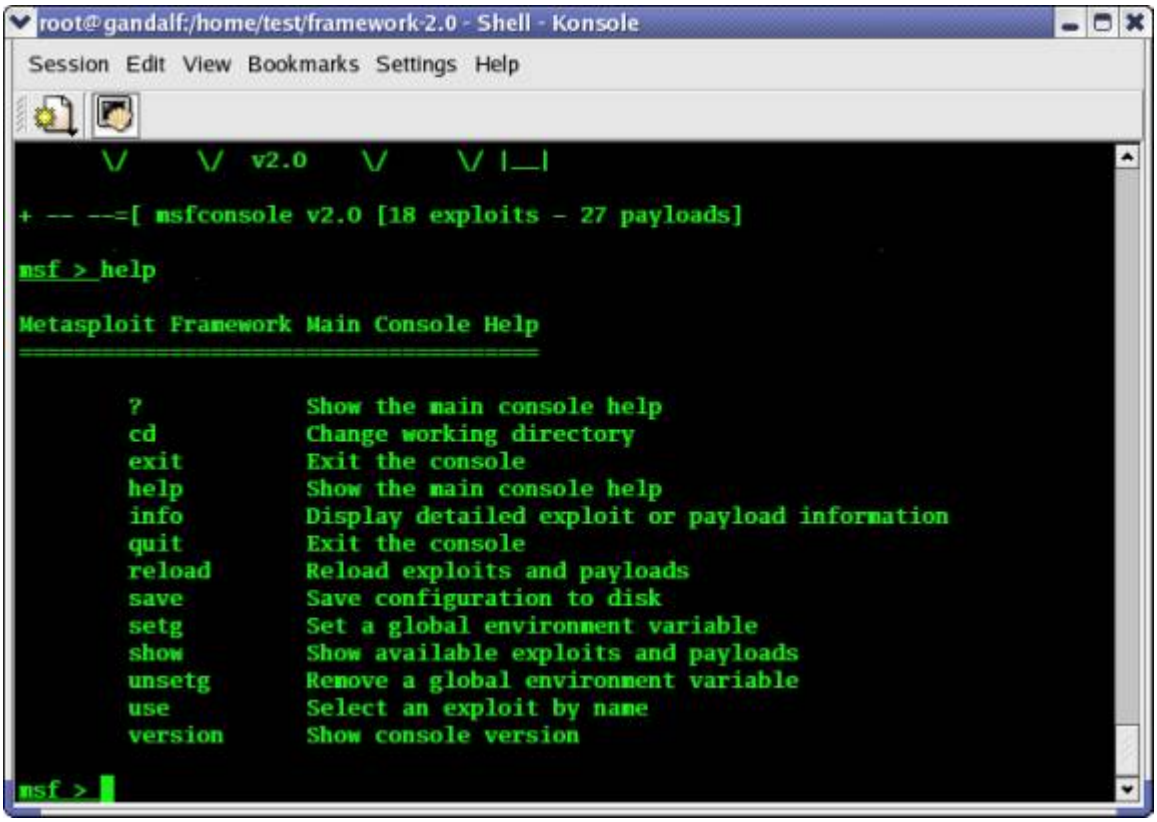

Figure 2

La commande **show exploits** liste les exploits disponibles. Il y a des exploits à distance pour différentes plateformes et applications comme Windows, Linux, IIS, Apache, etc, qui aident à tester la flexibilité et comprendre le fonctionnement du MSF. Cela est montré en Figure 3, ci-dessous.

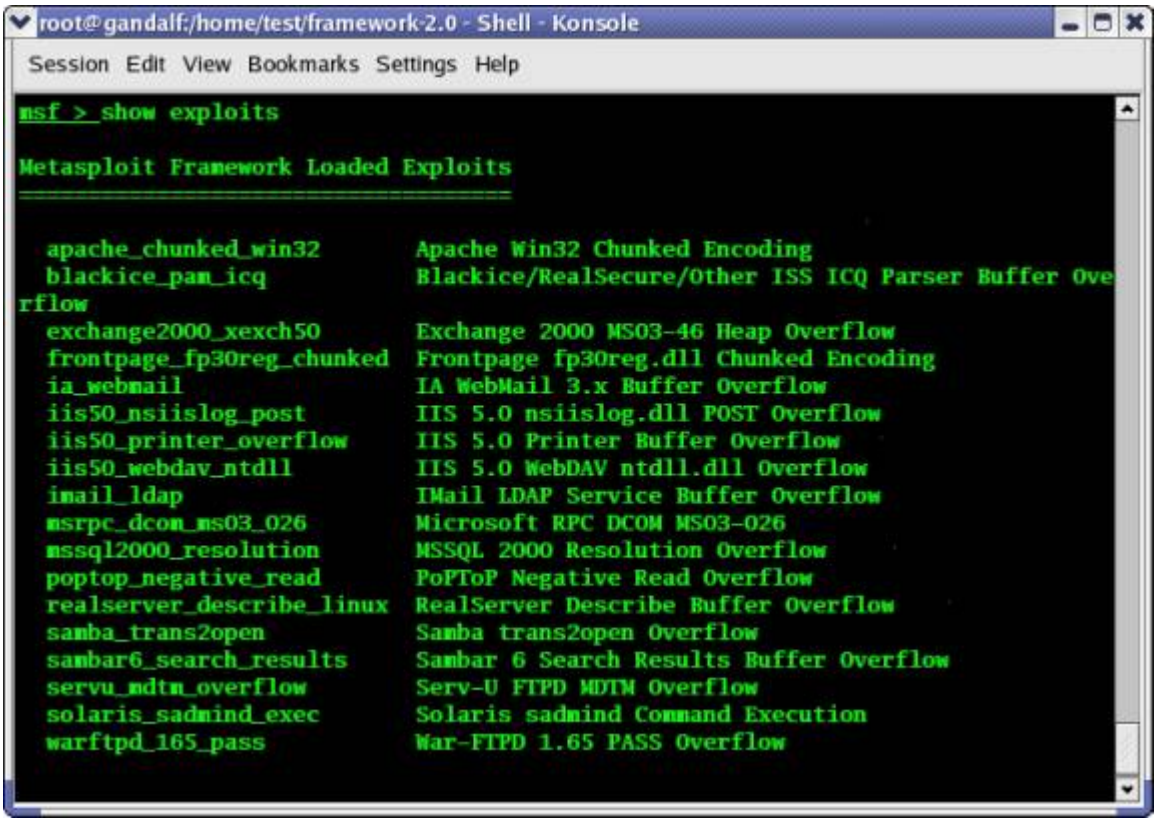

Figure 3

Vous vous avez pu le constater, l'installation par défaut du Metasploit Framework 2.0 contient 18 exploits et 27 payloads, ce qui constitue une collection relativement impressionnante.

Pour lister les payloads présents, exécutez la commande **show payloads**. Les payloads sont soignés, efficaces et très bien écrits. Ces payloads accomplissent un très grand nombre de tâches, comme lancer un shell inversé sur un port en écoute, ajouter de nouveaux comptes utilisateurs, ou uploader et exécuter le programme de votre choix. MSF supporte même la création de payloads dynamiques, en utilisant la librairie InlineEgg comme montré en Figure 4.

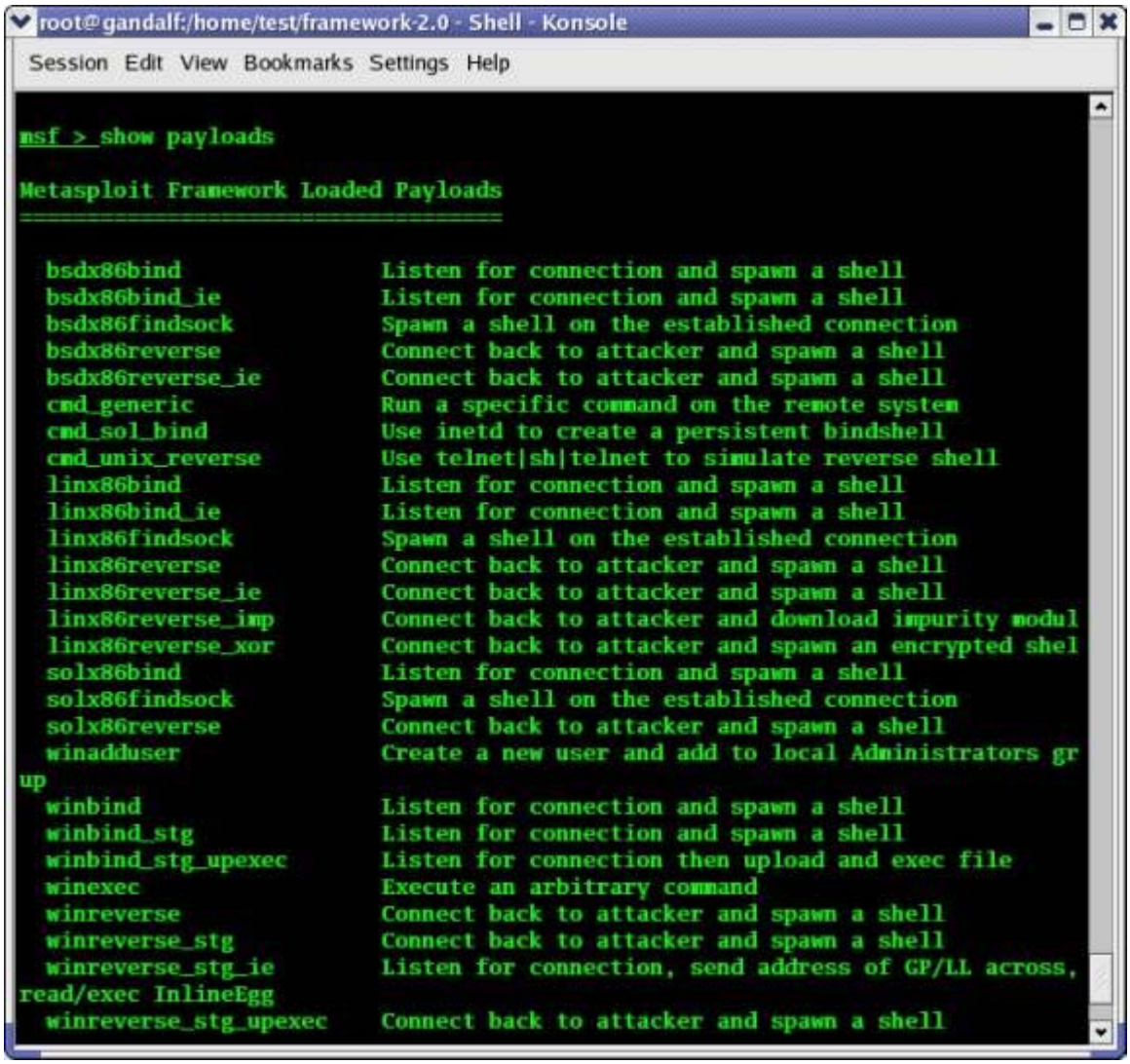

Figure 4

Les informations spécifiques à un exploit peuvent être consultées avec la commande **info exploit nom_exploit** qui fournit des informations comme les cibles disponibles, les choses nécessaires à l'exploit, les détails de la vulnérabilité elle-même, et même des références où vous pouvez trouver plus d'informations ! Cela est montré en Figure 5.

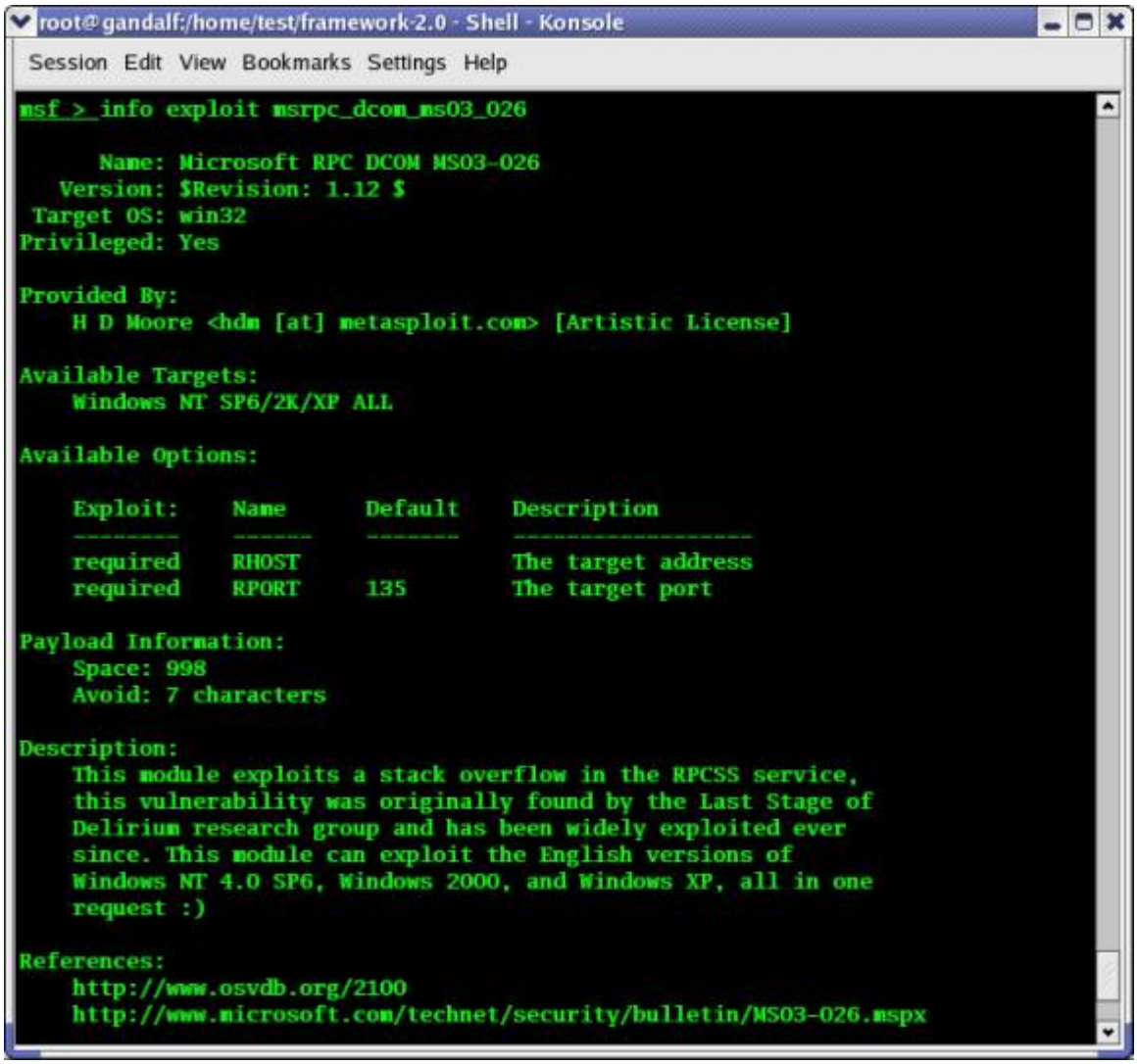

Figure 5

De la même manière, des informations sur un payload spécifique peuvent être obtenues avec la commande info **payload nom_payload**. A partir de la version 2.2 du MSF, vous pouvez utiliser **info nom_module**, sans avoir à spécifier le type, comme montré en Figure 6.

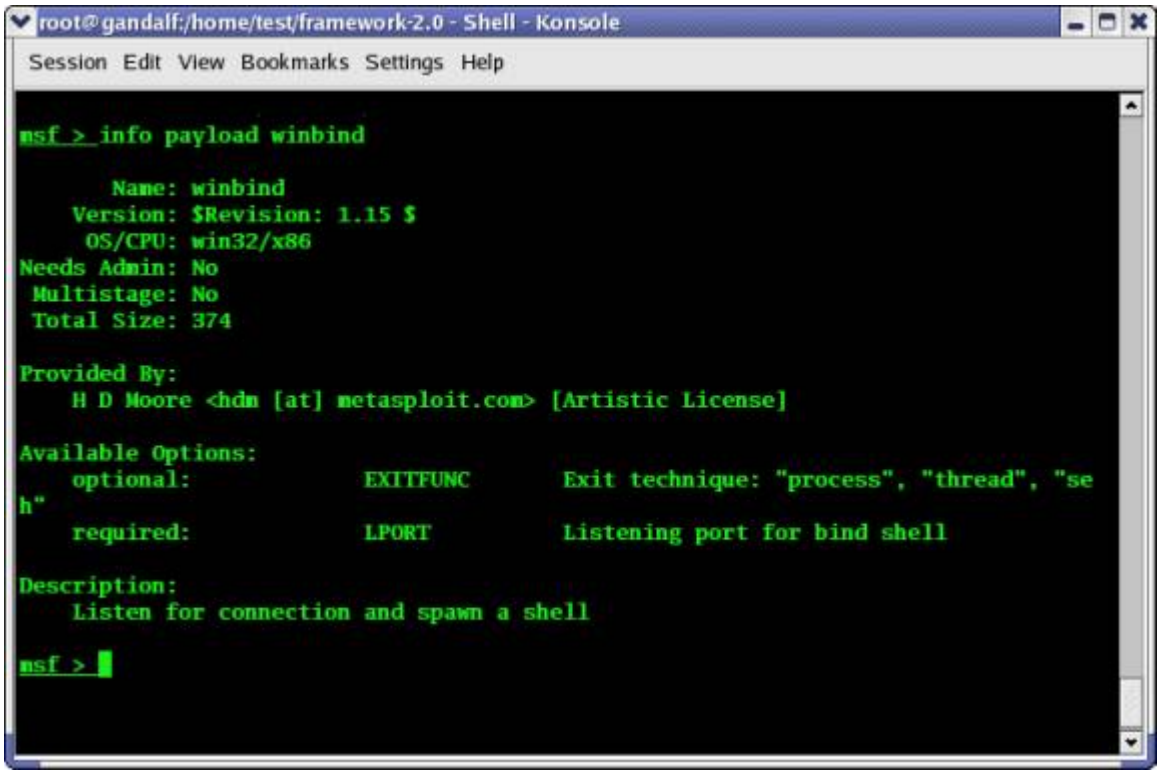

Figure 6

3. Utiliser Un Exploit

Maintenant nous allons décrire la procédure pour choisir un exploit spécifique et le lancer. La commande **use nom_exploit** active l'environnement de l'exploit pour l'exploit **nom_exploit**.

Si vous choisissez l'exploit Microsoft RPC DCOM MSO3-026 en utilisant le nom *msrpc_dcom_ms03_026*, vous noterez peut être que le prompt change de *msf>* en *msf msrpc_dcom_ms03_026 >*. Cela indique que vous êtes en train de travailler dans l'environnement temporaire de l'exploit. La commande **show** peut être utilisée pour voir les informations sur l'exploit en cours. La commande **show options** affiche les différents paramètres qui sont requis pour utiliser l'exploit, comme montré en Figure 7.

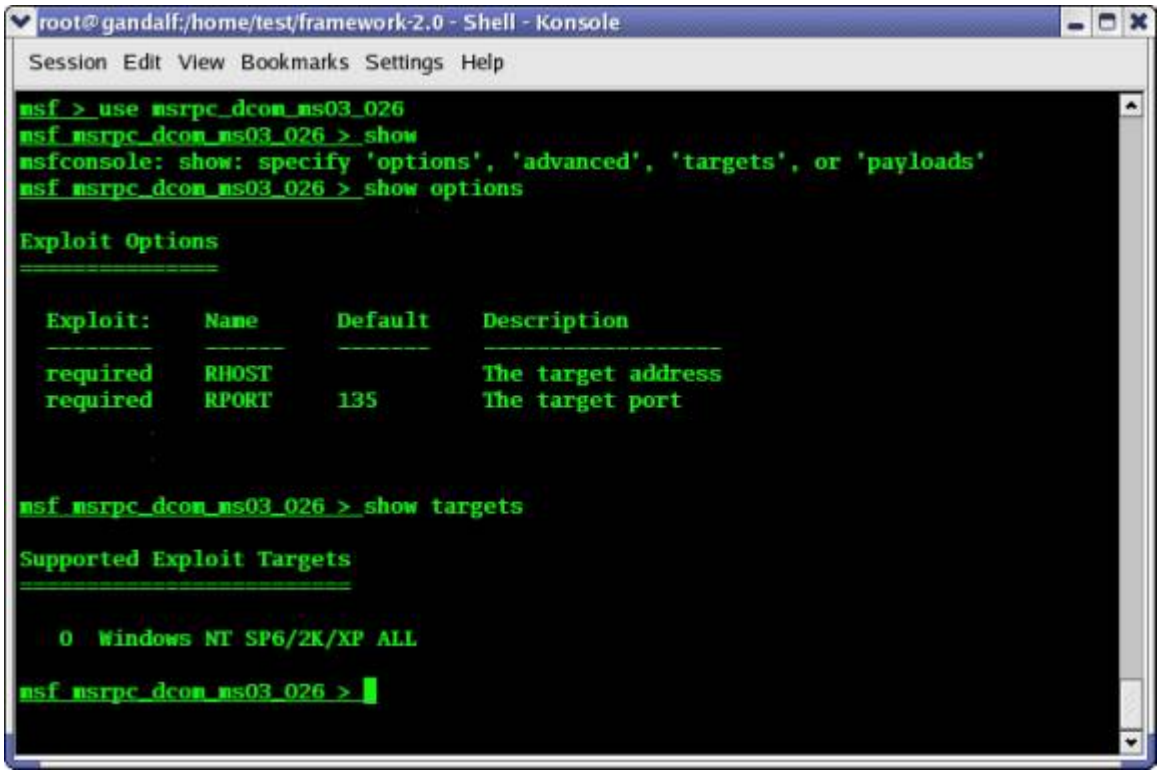

Figure 7

Il est clair que cet exploit requière deux paramètres, RHOST (l'adresse de la cible) et RPORT (et le port de la cible, 135 par défaut dans ce cas). La commande **show targets** va lister les cibles disponibles pour le module d'exploit sélectionné. Comme vous pouvez le voir, ce module a seulement une seule cible, qui fonctionne sur NT 4.0 SP6, plus toutes les versions de Windows 2000, et toutes les versions de Windows XP.

La commande **show payloads** liste tous les payloads qui sont compatibles avec l'exploit sélectionné. MSF fait du bon travail en vous empêchant d'utiliser un mauvais payload pour un exploit donné.

Nous devons configurer toutes les options listées comme 'nécessaires' avant de pouvoir utiliser l'exploit. Pour cet exploit nous avons seulement une option de cible unique, donc nous paramétrons la variable *TARGET* à 0, avec la commande **set TARGET 0**. Beaucoup d'exploits choisiront une cible correcte par défaut. Nous paramétrons maintenant l'adresse IP du serveur cible avec la commande **set RHOST 192.168.0.27**.

Ensuite, nous devons spécifier le payload requis (shellcode) pour l'exploit. On configure le *PAYLOAD* à *winbind*, avec la commande **set PAYLOAD winbind**. Les noms des payloads peuvent changer entre les versions de MSF, vérifier toujours avec la commande **show payloads** après une mise à jour. Ce payload particulier va faire écouter le serveur sur un port et déclencher un shell de commandes quand une connexion est réalisée. Cela montre la flexibilité du système de payload du MSF. Chaque exploit inclus dans le MSF permet de choisir et utiliser des payloads arbitraires, même ceux customisés que vous développez vous-même. Notez que le prompt change de *msf msrpc_dcom_ms03_026* > en *msf msrpc_dcom_ms03_026(winbind)* > après avoir sélectionné un payload. Maintenant nous utilisons la commande **show options** pour vérifier quelles options ont été configurées et lesquelles nécessitent d'être configurées. Comme nous pouvons le voir, nous devons encore

définir une valeur pour la variable *LPORT* cf Figure 8 avec **set LPORT 1536**.

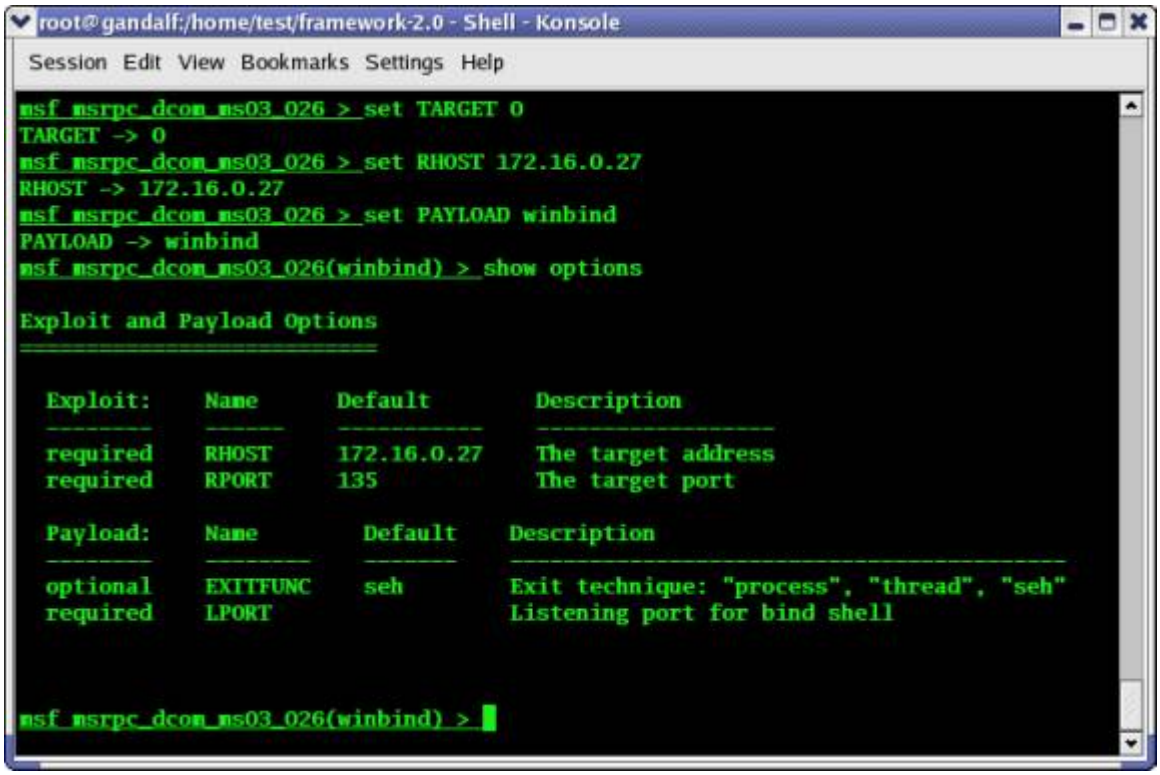

Figure 8

La variable *EXITFUNC* est disponible pour presque tous les payloads Windows. Cette variable contrôle comment le payload va se nettoyer après qu'il est accompli sa tâche. Quelques vulnérabilités peuvent être « exploitées répétitivement, simplement en utilisant une valeur différente pour *EXITFUNC*. Heureusement, vous aurez rarement à vous soucier de cela comme beaucoup d'exploits choisissent automatiquement la meilleure valeur pour vous. A moins que vous sachiez ce que vous faites, cette valeur n'aura pas à être définie. Choisir une mauvaise valeur peut endommager le havoc sur le système exploité.

Beaucoup d'exploits et de payloads ont un autre ensemble d'options, appelées options avancées (advanced options). Elles peuvent être affichées avec la commande **show advanced**. Les options avancées peuvent réaliser des tâches comme modifier une requête d'exploit pour parer à un signature IDS, changer les paramètres de force brute, ou spécifier des adresses de retour exactes à utiliser.

A ce niveau, tout est prêt et toutes les variables ont été définies. Nous faisons une vérification finale sur l'exploit avec la commande **show options** et vérifions que nous sommes prêt à y aller.

Tout semble parfait. C'est le moment de la démonstration !

La commande **exploit** lance l'attaque, faisant ce qu'elle a à faire pour exécuter le payload sur le système distant.

La commande **check** peut être utilisée pour savoir si le système cible est vulnérable ou pas à

l'attaque. La fonctionnalité de vérification n'est pas disponible avec tous les exploits, mais peut être utile pour déterminer si un système est patché avant de tenter de l'exploiter.

4. Ajouter de Nouveaux Exploits / Modules

Ajouter de nouveaux exploits au MSF est simple comme bonjour. L'exploit compatible avec MSF pour le Buffer Overflow IIS 5.x SSL PCT a été divulgué publiquement le 24/04/2004 (http://www.k-otik.com/exploits/04242004.iis5x_ssl_pct.pm.php). Pour les besoins de cet article, nous allons ajouter cet exploit à notre base MSF.

Après avoir téléchargé l'exploit, l'utilisateur doit noter le nommage du module Perl pour l'exploit. Le nom du fichier doit être le même que le nom du paquetage, en d'autres termes, **Msf::Exploit::iis5x_ssl_pct** doit être enregistré en **iis5x_ssl_pct.pm**. Maintenant, copier le moule dans le sous répertoires des exploits (dans le cas où vous utilisez Windows, c'est /home/framework-2.0/exploits). Dès que le fichier est copié dedans, il est prêt à être utilisé, et vous n'avez même pas à relancer la console. Utilisez la commande **show exploits** pour vérifier que le module a été correctement chargé.

```
msf > show exploits

Metasploit Framework Loaded Exploits
====================================

apache_chunked_win32        Apache Win32 Chunked Encoding
exchange2000_xexch50        Exchange 2000 MS03-46 Heap Overflow
ia_webmail                  IA WebMail 3.x Buffer Overflow
iis50_nsiislog_post         IIS 5.0 nsiislog.dll POST Overflow
iis50_printer_overflow      IIS 5.0 Printer Buffer Overflow
iis50_webdav_ntdll          IIS 5.0 WebDAV ntdll.dll Overflow
iis5x_ssl_pct               IIS 5.x SSL PCT Overflow
imail_ldap                  IMail LDAP Service Buffer Overflow
msrpc_dcom_ms03_026         Microsoft RPC DCOM MSO3-026
mssql2000_resolution        MSSQL 2000 Resolution Overflow
poptop_negative_read        PoPToP Negative Read Overflow
...
```

L'exploit a été correctement ajouté à la liste. L'exploit est lancé de la même manière que n'importe quel autre exploit dans MSF. La version 2.2 du MSF permet aux utilisateurs de garder leur propre répertoire privé d'exploits, payloads, encodeurs, et nops. L'installation d'un nouvel exploit peut être soit pour un système ou par utilisateur.

5. Environnements de la Console

Dans les paragraphes précédents, nous avons fait quelques références aux variables et environnements, sans expliquer ce qu'ils sont. Un environnement est simplement un emplacement nommé pour les variables. Lorsque vous configurez une variable dans MSF, cela créé une nouvelle entrée dans l'environnement courant. Les environnements ont utilisés pour spécifier les paramètres des exploits et configurer un environnement temporaire qui surpasse l'environnement global.

5.1 Environnement Global

L'environnement global est accédé à travers les commandes **setg** et **unsetg**. Appeler la commande **setg** affiche l'environnement global courant et appeler la commande **unsetg** réinitialise tous ls paramètres de l'environnement global.

Comme montré ci-dessous en Figure9, nous mettons les valeurs à *LHOST*, *LPORT* et *PAYLOAD* dans l'environnement global comme valeurs permanentes et enregistrons les changements avec la commande **save**.

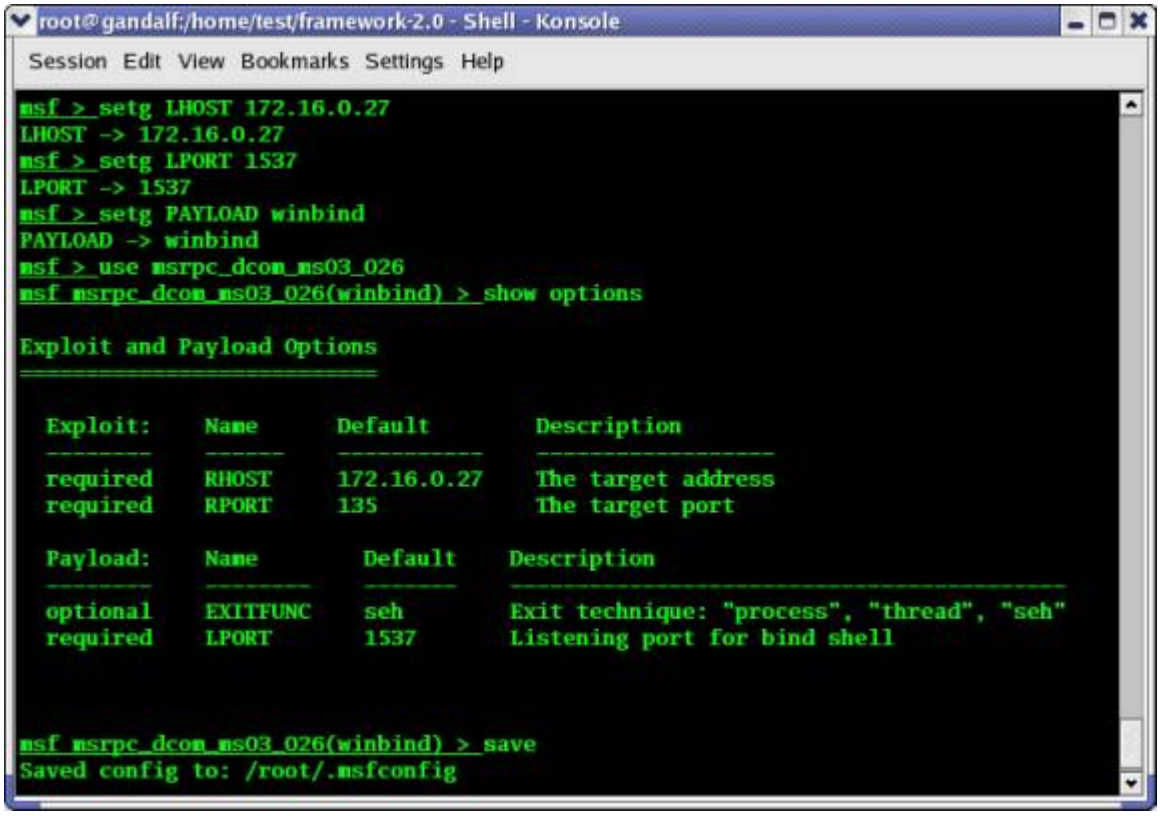

Figure 9

La commande **save** écrit tous les environnements courants dans un fichier sur le disque. Les versions 2.0 et 2.1 placent cette donnée dans le fichier $HOME/.msfconfig, et la version 2.2 place les environnements sauvegardés dans $HOME/.msf/config. Les environnements sauvegardés sont chargés la prochaine fois que n'importe quelle interface utilisateur MSF est lancée. C'est une pratique courante de régler les environnements globaux comme *LHOST* et *LPORT* et de les enregistrer sur le disque, supprimant la nécessité de les configurer pour chaque exploit.

5.2 Environnement Temporaire

Les environnements temporaires sont des sous environnements qui surpassent les paramètres globaux. L'Environnement Temporaire est spécifique à l'exploit sélectionné en cours. Chaque environnement d'exploit est isolé du reste, permettant à l'utilisateur de facilement naviguer entre les exploits pré configurés avec la commande **use**.

5.3 Paramètres d'Environnement Avancés

Le MSF fournit quelques paramètres avancés qui sont configurés à travers les variables d'environnements. Ces paramètres inclus le système de journalisation, les options de socket, et les paramètres de débuggage.

5.3.1 Options de Journalisation

Les fonctionnalités suivantes peuvent être activées en configurant la variable *Logging* (globale comme son nom temporaire) à une valeur différente de zéro. Le répertoire des logs est paramétré en changeant la variable *LogDir* (globale comme son nom temporaire) qui vaut par défaut *$HOME/.msflogs*. L'utilitaire **msflogdump** peut être utilisé pour voir les logs de sessions. A partir de la version 2.2, les logs sont enregistrés dans *$HOME/.msf/logs*.

5.3.2 Options de Socket

Les différents paramètres de timeout et de proxy peuvent être changés en configurant les variables d'environnement suivantes.

Msf::Socket::Proxies (nom global) ou *Proxies* (nom temporaire): Cette variable peut être utilisée pour définir les paramètres de proxy (SOCKS4 et HTTP) pour les connexions réseaux. Il supporte les chaînage de proxies qui peuvent être spécifiées au format type :hôte :port et séparé par des virgules pour chaque serveur proxy.

Msf::Socket::RecvTimeout (nom global) ou *RecvTimeout* (nom temporaire): Cela spécifie le nombre maximum de secondes autorisées pour lire depuis un socket.

Msf::Socket::ConnectTimeout (nom global) ou *ConnectTimeout* (nom temporaire): Cela sert à spécifier la période de dépassement du délai de connexion pour un socket (10 secondes par défaut).

Msf::Socket::RecvTimeoutLoop (nom global) ou *RecvTimeoutLoop* (nom temporaire): Défini le temps maximum (en secondes) pour attendre une connexion avant de fermer le socket. Cette boucle est réactivée après chaque réception de données.

5.3.3 Options de Debuggage

La variable d'environnement *DebugLevel* spécifie le niveau de debuggage et les options de verbosité pour le Framework et les modules. La verbosité varie suivant la valeur de la variable, entre 0 et 5.

5.3.4 Options des Payloads

Par défaut, le processus d'encodage va tourner dans tous les modules jusqu'à ce qu'il en trouve un qui permette l'ensemble de caractères particuliers pour l'exploit en cours. L'ordre des modules d'encodage peut être réglé dans l'ordre séparé par des virgules dans la variable d'environnement *Encoding*. De la même manière, la variable *Nop* est utilisée pour spécifier l'ordre des routines de génération nop. Cela peut être utile quand vous devez palier à certaines signatures IDS.

La variable *RandomNops* indique au module de génération nop d'utiliser des séquences randomisées d'instructions de type nop au lieu du opcode nop standard. Cela peut également être utilisé pour prévenir des signatures IDS. La version 2.2 inclus le support de génération nop rusée, où chaque exploit peut spécifiés les registres qui ne devront pas être modifiés par les opcodes de type nop.

6. Conclusion

Après avoir lu la seconde partie de cet article, vous devriez avoir bien compris ce qu'est le Metasploit Framework et comment vous pouvez commencer à l'utiliser. Nous avons décrit l'interface msfconsole, les processus généraux pour choisir et utiliser un exploit, et comment le système d'environnements fonctionne.

Cet article laisse la place à la troisième et dernière partie, qui sera publiée cette semaine, qui expliquera les autres interfaces utilisateur, les utilitaires d'aide inclus, et quelques explications pour développés vos propres modules d'exploits. Nous discuterons de son futur en anticipant les nouvelles fonctionnalités qui seront ajoutées au framework.

References

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.