\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer 2 Attacks

and how to mitigate them

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Author*:

Gabriel Rosas

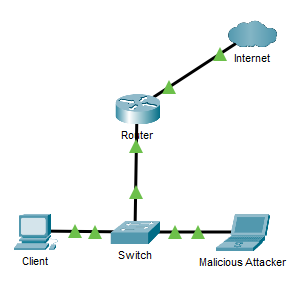


Purpose

With reliance on the internet being at an all-time high, proper security is critical to user safety. In this paper, I cover a couple popular attacks easily performed in Kali Linux, an open-source Linux distribution designed for digital forensics and network penetration testing. Afterwards, I’ll demonstrate some techniques used to mitigate them.

Background Information

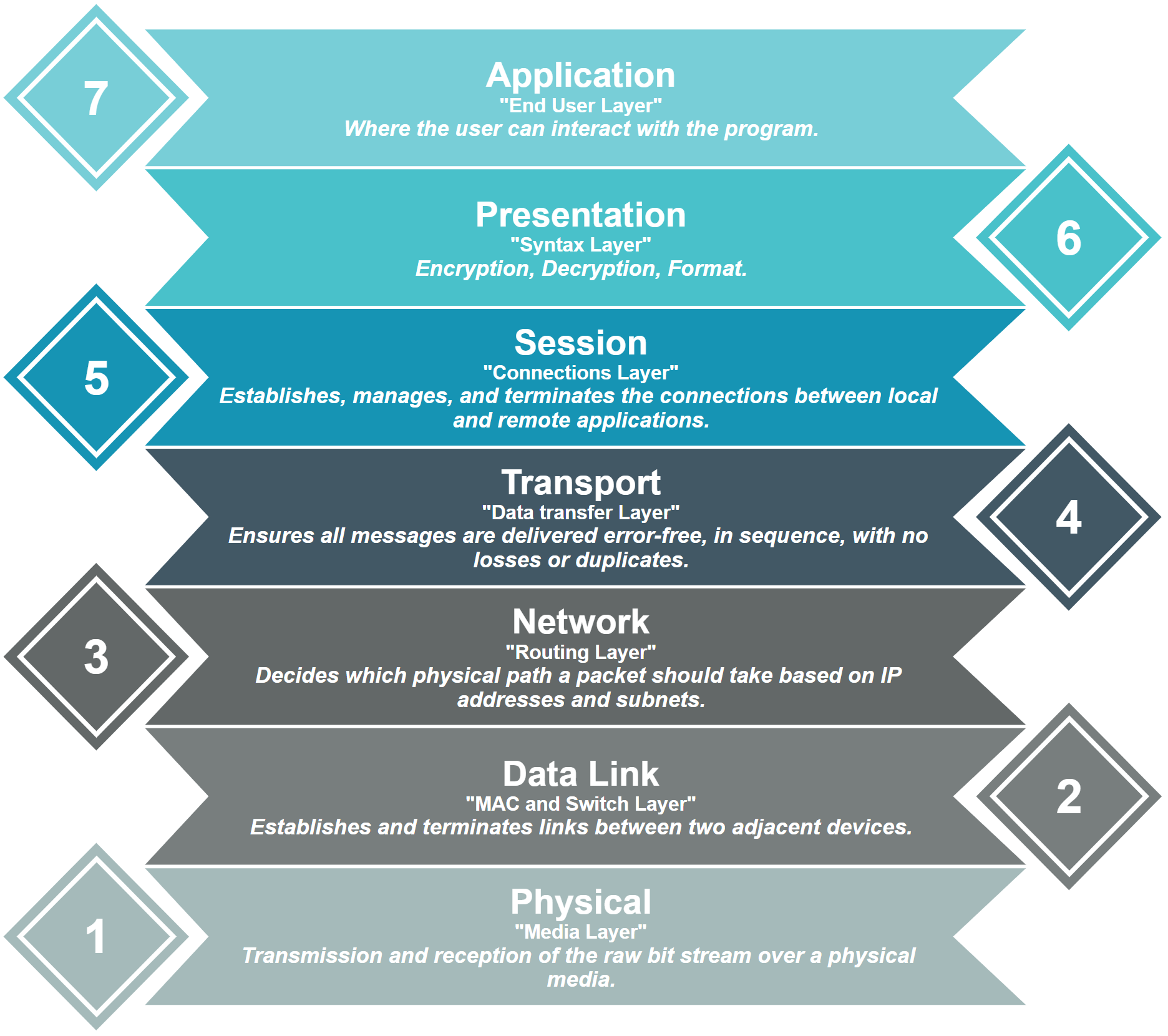
If you’re responsible for the security of a company, even down to a small private network, you want to be aware of the vulnerabilities that attackers may exploit. All these attacks assume a switched ethernet network running **internet protocol** (IP). In other words, an attacker has physically infiltrated your building and has access to a specific network device: the **switch**.



A simple way to think of a switch is like a power strip: a single outlet in the wall can be split into multiple outlets if needs be. While a power strip extends the number of outlets, a switch extends the number of ethernet ports. A typical router only has two or three ethernet ports, so at times when more are needed, a network administrator would introduce a switch.

What is Layer 2?

Layer 2 refers to the second layer (Data Link Layer) of the Open Systems Interconnection (OSI) model. There are seven Layers in total:

****

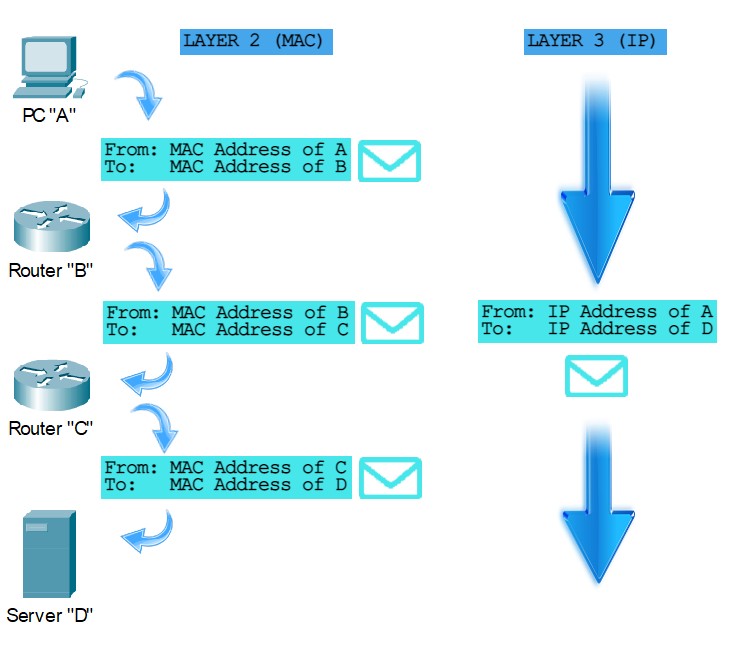
Although the modern internet doesn’t strictly follow the OSI model, it is very useful for troubleshooting and breaking down networking problems into smaller, bite-sized chunks. If a problem can be narrowed down to one specific Layer, a lot of extra work can be avoided.

IP addresses vs. MAC addresses

Initially it may seem that IP and MAC addresses are redundant since they both uniquely identify a network device. However, they serve different purposes. MAC operates at Layer 2 of the OSI model – hence why many of these attacks include the stealing and forging of MAC addresses – while IP operates at Layer 3.

During transmission, MAC addresses are used to direct packets from **device to device** while IP addresses from **source to destination**, across multiple devices. The source and destination MAC addresses will change based on where the packet is in transit, but the source and destination IPs won’t.

For example, the average train running through the UK will stop at different stations, whether it’s a station the passenger wants to get off at or not. The overhead display will show the previous and next stations, prompting the passenger of where they are in transit. While the passenger only has use for the specific stations they enter and exit, the train will still stop at each station regardless. Layer 2 is a bit like station to station and Layer 3 is more entry station to exit station.



How does a switch function?

Back in the old days, during the times of hubs and repeaters, packets were broadcasted through every port on the device. If ten computers were connected to a hub, and two were trying to communicate, every device would receive the packets then manually drop them unless they were the destined host. You can imagine how hackers took advantage of being dealt free data not destined for them. Eliminating security flaws and saving bandwidth in mind, network engineers created a better device: the switch.

Each port on a switch stores the MAC address of the connected device. Instead of relying on broadcasts, switches intelligently direct traffic to the correct destination based on MAC address. The **Content-Addressable Memory (CAM) table** maps each *port* to a *MAC address* so the switch can easily direct data using the destination MAC address on a packet through the correct port.

If a host on the ***FastEthernet 0/1*** port requests a webpage, data will route to and from the webserver and eventually back to the switch (though it will probably take mere milliseconds). After reaching the switch, the packets forward through the port associated with their destination MAC address. Switches can be joined together to support even more ports. A switch-to-switch connection is known as a **trunk port**.

Spoofing and Sniffing

There are three ultimate goals of an attacker: capturing a host’s data, altering a host’s data, or denying a host access to a certain service (DOS). Many attacks incorporate **spoofed** MAC addresses to **sniff** a legitimate host’s packets. Spoofing is where an attacker identifies themselves as different device by falsifying their data. Sniffing refers to capturing packets intended for a different device. Before we get into the attacks, I recommend familiarizing with the general commands.

Command Reference

|  |  |
| --- | --- |
| **Command** | A statement necessary for a configuration to work, denoted in bold |
| **[*Argument*]** | An argument necessary for a command to function, denoted in bold italics. |
| *Optional-Statement*  *<Optional Argument>* | An optional argument or statement, not necessary for a command to function, denoted in italics |

// IPv4 DHCP Configuration

Router(config)# **ip dhcp exclude-address [*Initial IP address*]** <*End IP address*>

* Set an IP or range of IPs to exclude from the pool

If the network administrator so chooses to exclude a range of IP addresses, the range would be from the *Initial IP* to the *End IP*, inclusive. The *End IP* argument is not necessary when excluding only one IP. Excludes are typically reserved for pre-configured static IP addresses, for example, interfaces on the router.

Router(config)# **ip dhcp pool [*pool name*]**

* Creates a pool for distributing routing information

Dynamic Host Configuration Protocol (DHCP) automates the assignment of IP addresses to devices on the local network. An IP pool is used to define the sequential range of IP addresses that the server will divvy out to clients.

Router(dhcp-config)# **network [*network address*] [*subnet mask*]**

* Configures the pool to distribute the set subnet

Router(dhcp-config)# **default-router [*IP address*]**

* Sets the client’s default gateway to the specified IP address

Router(dhcp-config)# **dns-server [*IP address*]**

* Sets the client’s DNS server to the specified IP address

Router# **show ip dhcp binding**

* Displays the current mappings of the IP pool to the clients

// VLAN Configuration

Switch(config)# **vlan [*id*]**

* Create a vlan with specified *id*

A Virtual Lan (VLAN) is used to partition groups of devices on a switched network.

Switch(config-if)# **switchport access vlan [*id*]**

* Configure an interface to be part of a specified vlan

If a user desires to configure a vlan across multiple interfaces, use the command **interface range [*interface*] [*start-end id*]**.

KALI Linux Commands

**$ sudo su**

* Enter root mode

**$ ip a**

* Display device interface settings, synonymous to *ipconfig* on Windows

**$ sudo ip addr add [*IP address*]/[*subnet prefix*] dev [*interface*]**

* Add an IP address to a (dev)ice interface

**//** Graphical Applications

**$ sudo ettercap -G**

* Opens the Ettercap Graphical interface

Ettercap is a program primarily focused on sniffing packets and man-in-the-middle (MITM) attacks. It is possible to sniff in four modes: IP based, MAC based, ARP based (full-duplex) and PublicARP based (half-duplex) across different interfaces.

**$ sudo yersinia -G**

* Opens the Yersinia Graphical interface

Yersina is a framework for performing Layer 2 attacks. It is designed to exploit weaknesses in certain networking protocols, such as STP, CDP, DTP, DHCP, HSRP, 802.1Q, 802.1X, ISL and VTP.

**$ wireshark**

* Opens the Wireshark Graphical interface

Wireshark is a packet analyzer, designed to pick up packets “on the wire” (a live interface).

// Creating VLAN interfaces on Kali (VLAN Hopping)

**$ ip link add link [*interface*] name [*name*] type vlan id [*id*]**

* Create a VLAN on a specified *interface*, typically *eth0*, with a specified *name* and *id*.

**$ ip addr add [*IP address*]/[*subnet prefix*] dev [*link name*]**

* Add the IP address of the specified link

**$ ip link set dev [*link name*] up**

* Set the link to up

// NMAP

**$ nmap** <*scan type(s)>* <*options*> **[*target*]**

* Probe at a network or target to find information about hosts, open ports, and software

Common NMAP Settings

* **Host Discovery**
  + **-*sL***: List targets only
  + *-****sn***: Scan a network but omit port scans
* **Service and Version Detection**
  + **-*sV***: Attempts to determine the version of the service running on port
  + -***A***: Enables OS detection, version detection, script scanning, and traceroute
  + -***O***: Remote OS detection using TCP/IP stack fingerprinting
* **Port Specification**
  + -***p* [*port*]**: Scan a specific port (for range use ports *a-b*)

Attacks

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*ARP Spoofing*

*DHCP Starvation and MITM*

*VLAN Hopping*

*CDP Flooding*

*Introductory NMAP*

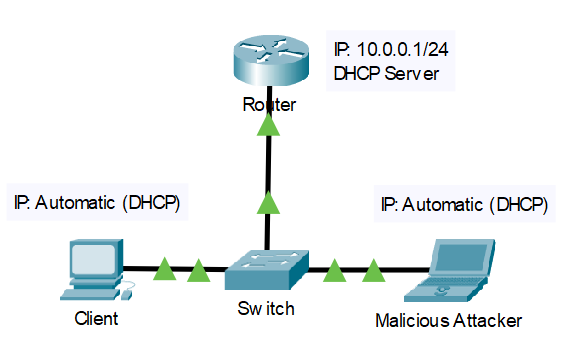
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**Address Resolution Protocol (ARP) Spoofing**

ARP spoofing is a type of attack where false ARP packets are sent over a Local Area Network (LAN), linking an attacker’s MAC address with an IP of a device on the network. Once the attacker's MAC address is connected to an authentic IP, the attacker will receive data intended for that IP. ARP attacks are used to intercept, modify, or stop data in-transit. ARP spoofing attacks can only occur on LANs that use ARP.

Topology



*This attack assumes a pre-configured DHCP server. For information on creating a basic DHCP server on a CISCO router, please refer to the commands above.*

**ARP Spoofing with Ettercap Graphical**

Open Ettercap and proceed with default settings, making sure the primary interface is ethernet (typically **eth0**).



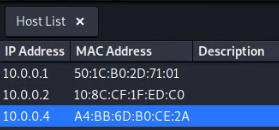
Open the host list (top left for v0.8.3.1).



Navigate to the settings tab, go to “Hosts” and “Scan for hosts”

|  |  |  |
| --- | --- | --- |
|  |  |  |

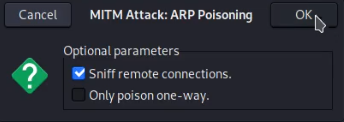
Add the router and a host to Target 1 and Target 2.



Navigate to the MITM tab and select the ARP poisoning attack.

|  |  |
| --- | --- |
|  |  |

Begin with default settings.



At this point, Ettercap is broadcasting false ARP messages in hope to link your MAC address with the IP of the target. You should be able to open a packet sniffer, such as *Wireshark*, and sniff up packets destined for the legitimate host.

Mitigation

To prevent attacks like ARP spoofing, network engineers must ensure that only valid ARP requests and responses are relayed. **Dynamic ARP Inspection** (DAI) intercepts every ARP message and verifies their MAC address to IP binding before the local ARP cache is updated. Only clients with the correct bindings are forwarded through, otherwise the packets are dropped. There are many unique configurations for DAI, for example, rate limiting, set MAC authentication, or access lists. However, the method I will cover is simply trusting specific ports to skip authentication.

Mitigation Commands

Switch(config)# **ip arp inspection vlan [*id*]**

* Enables DAI on the specified vlan

Switch(config-if)# **ip arp inspection trust**

* Skips DAI on the current interface

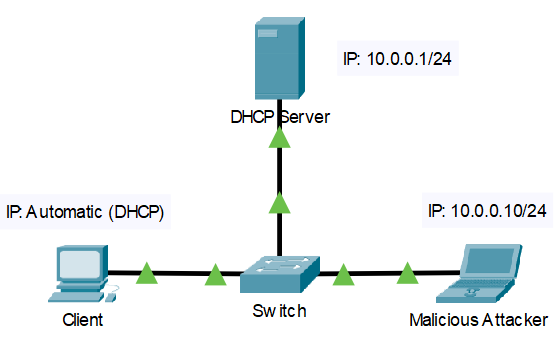
By trusting an interface, you are specifying that you do not want Dynamic ARP Inspection to run. Links to known devices are typically the interfaces you’d want to trust.

**Dynamic Host Configuration Protocol (DHCP) Starvation & MITM**

In a DHCP Starvation attack, an attacker broadcasts many DHCP requests with spoofed MAC addresses in order to drain the pool of a DHCP server. If the DHCP server responds, its available addresses will deplete within a short amount of time. Once there are no more available addresses, any legitimate client trying to connect to the network won’t be allocated an IP address, denying them service to the internet. After “starving” the pool, attackers can try to assert themselves as the new DHCP server.

An attacker launches a rogue DHCP server on their machine. Without access to information from the genuine DHCP server, new clients receive false DHCP settings from the attacker on their requests. These settings might be designed to forward all the client’s data to the attacker, then out to the router like nothing was wrong in the first place.

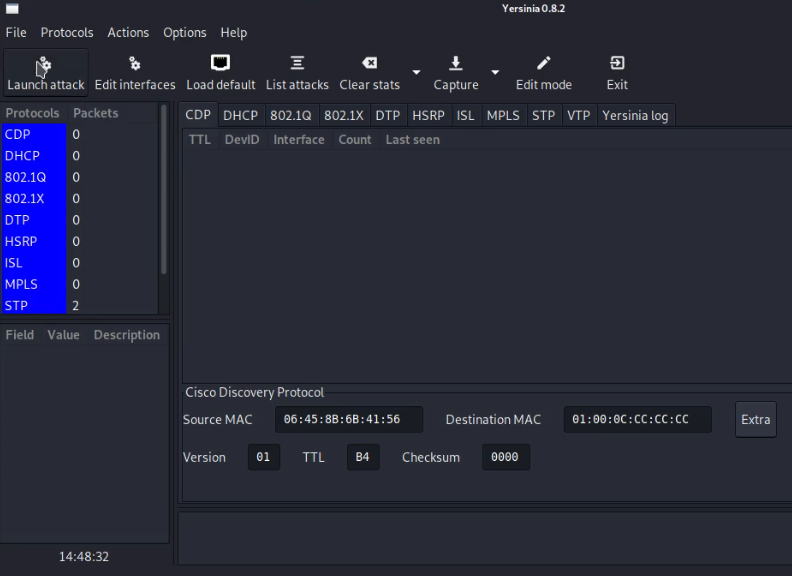
Topology



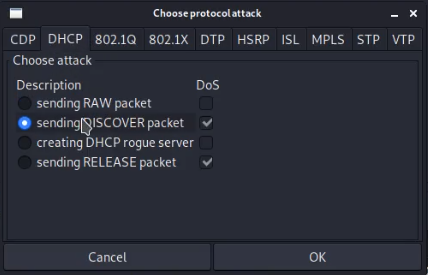
*This attack assumes a pre-configured DHCP server. For information on creating a basic DHCP server on a CISCO router, please refer to the commands above.*

**DHCP Starvation with Yersinia**

Upon starting Yersinia, find and hit “Launch attack”.



A popup should appear. Enter the DHCP tab and start sending DISCOVER packets.

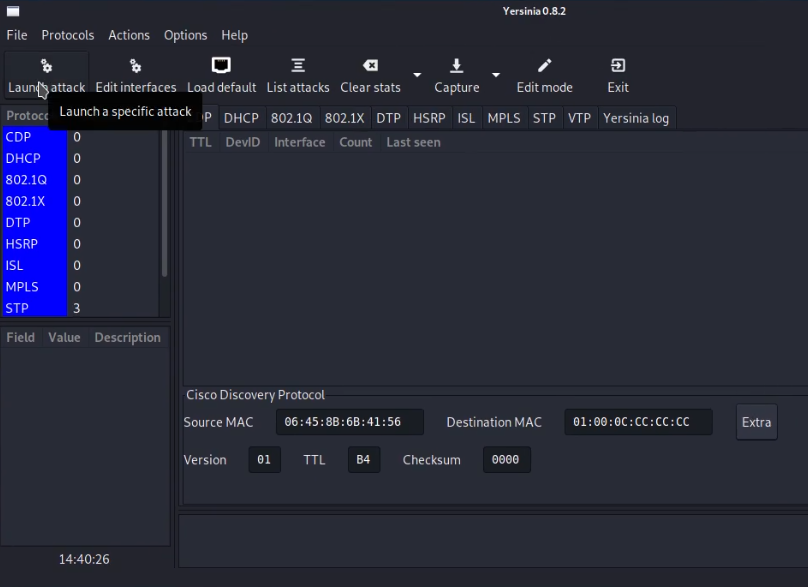


Yersinia should begin to flood the network with DHCP discover packets, devouring the DHCP server’s pool. Any client that requests DHCP information will not be able to obtain it from the legitimate DHCP server.

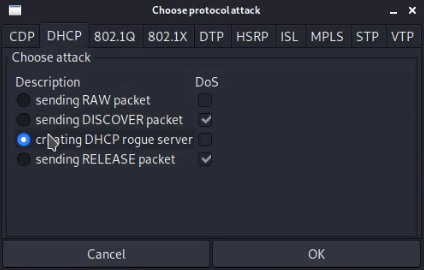
**Setting up a Rogue DHCP server in Yersinia**

Once the legitimate server’s DHCP pool has been exhausted, an attacker can take the opportunity to set up a fake one. The attacker could set up a fake DHCP server without starving the pool, but that would put them in direct competition with the real server. To guarantee their position as the sole DHCP server, a competent attacker would exhaust the pool before imitating a server.

Enter Yersinia and hit “Launch attack”

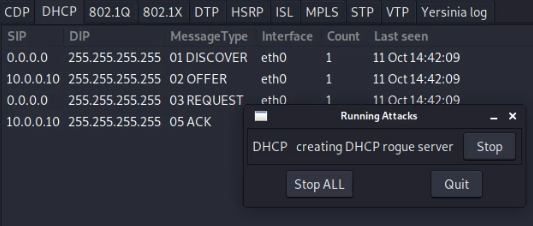


Choose the DHCP tab and create a rogue server



|  |  |
| --- | --- |
| After creating a DHCP server, you should be prompted to configure some parameters. To the right are some examples I used.   * Server ID – the IP of your PC * Start IP – the beginning address of the IP pool * End IP – the end address of the IP pool * Subnet Mask – the mask of the pool * Router – set the default-router for the client * DNS Server – set the DNS server of the client * Domain – set the domain of the client |  |

The rogue DHCP server should now create. Logs can be seen within the DHCP tab in Yersinia, notifying the user of user requests.



Mitigation

Enabling some form of DHCP snooping is a typical the way to counter a DHCP attack. A network engineer should configure DHCP snooping on a switch and trust the port they know to contain the legitimate DHCP server. By enabling DHCP snooping without trusting any ports, every DHCP response is denied, including responses from a genuine DHCP server. They should also restrict the number of DHCP requests able to be sent through a port so attackers can’t flood the network with fake requests.

Mitigation Commands

// Preventing a DHCP starvation attack

Switch(config-if)# **ip dhcp snooping limit rate [*packets per second*]**

* Limits the maximum number of DHCP packets allowed through an interface in a given second

// Preventing a rogue DHCP server

Switch(config)# **ip dhcp snooping**

* Enables DHCP snooping on the switch globally

Switch(config)# **ip dhcp snooping vlan [*id*]**

* Enables DHCP snooping on the specified vlan

By design, every interface initially resides within *VLAN 1*, the default vlan. If a switch has no user-configured VLANs, enable snooping on *VLAN 1*.

Switch(config)# **no ip dhcp snooping information option**

* Disable DHCP Option 82

Option 82 can cause problems for certain switches so it’s best to disable it.

Switch(config-if)# **ip dhcp snooping trust**

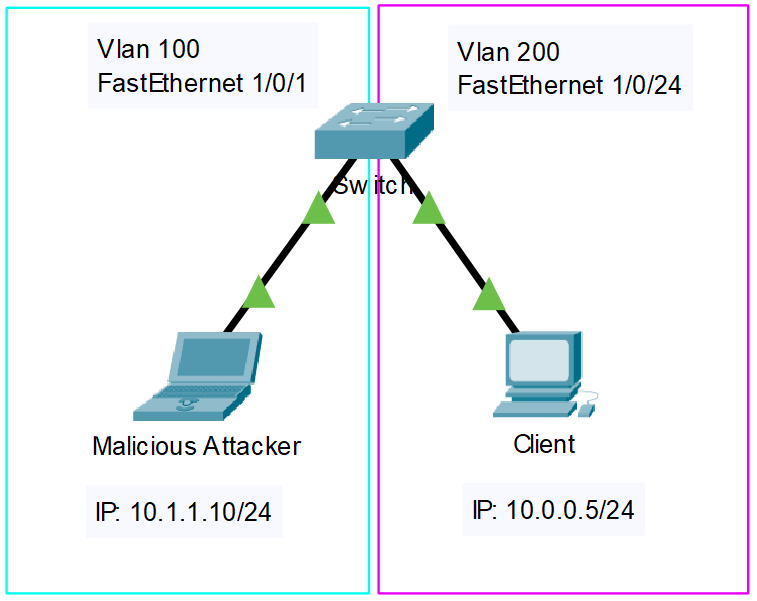
* Trust the interface of the legitimate DHCP server

**Virtual Lan (VLAN) Hopping**

Before we get into the attack, we need to understand a particular feature on switches – VLANs. Imagine a square room built entirely out of doors. Anyone can pass through a door and leave through another since there is no interference, no rules dictating otherwise. Now, consider a solid wall placed to divide that room into two sections. While the outer foundation is still all doors, the inside is completely split with no ability to communicate with the other section. In essence, this is what you can do to a switch. You can section ports on a switch into a certain VLAN and those ports will only be able to communicate with their respective VLAN. So even if two devices were plugged in directly next to each other, they could only relay traffic if they were on the same VLAN.

The exception to this rule – and what hackers like to exploit – is trunk ports. Switch-to-switch links. Unlike the typical access ports, trunk ports support multiple VLANs so network administrators can extend their VLAN configurations across multiple switches. However, a hacker might pretend to be a trunk link, enabling them access to every VLAN if the connected switch accepts the request. This attack manipulates **Dynamic Trunking Protocol** (DTP) to bypass or “hop” into a different VLAN than what a device is originally part of.

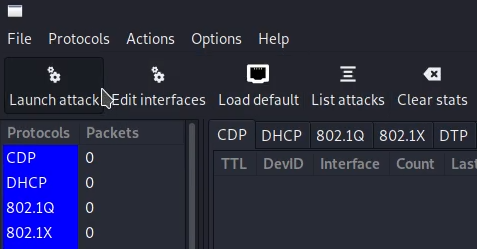
Topology



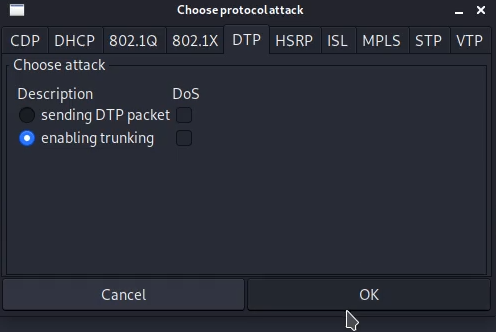
*This attack assumes a network with configured VLANs such as this topology. For more information on configuring VLANs, please refer to the command section above.*

**Enabling a DTP Trunking Link**

After starting Yersinia, find and hit “Launch attack”.



Under the DTP Protocol tab, enable trunking.



This is all we need from Yersinia – negotiating our link with the switch to be a trunk port – the rest is done in the Kali terminal. Open a new terminal and enter *root mode* by typing **sudo su**.

**Creating a VLAN interface on Kali**

*Root mode* grants us access to the highest privilege commands without us having to type **sudo** before every command we enter. Our goal now is to create a VLAN on Kali to match the one on the switch we wish to “hop”. The one in this example is *VLAN 200*, which we do like this:

**$ ip link add link *eth0* name *eth0.200* type vlan id *200***



We can verify this link has been added using the **ip link** command.

With the VLAN now added in Kali, we need to create an IP for this link. The IP should match the subnet of the victim’s VLAN, which happens to be **10.0.0.0/24** and*VLAN 200* in this example.

**$ ip addr add *10.0.0.10/24* brd *10.0.0.255* dev *eth0.200***



Finally, set the device link to up.

**$ ip link set dev *eth0.200* up**



If everything plays out correctly, we should be able to ping and reach devices in an external VLAN, in this case *VLAN 200*.

Mitigation

As I mentioned earlier, *switch-to-switch* connections have *trunk* links while *switch-to-device* connections have *access* links. A default, unconfigured port will negotiate the connection based on the neighboring device. In an ideal world where all devices are honest, this method works perfectly fine. But as we’ve just learnt, attackers can manipulate the negotiation process to become trunk ports.

Instead of going through the entire negotiation process, we can designate ports as either trunk or access. This is known as port-security, which encompasses many more features than just port types. A network engineer should only configure trunk ports when a switch-to-switch link is guaranteed in their topology, the rest being access ports.

Mitigation Commands

// Defining a Trunk port

Switch(config-if)# **switchport trunk encapsulation dot1q**

* Specify an encapsulation method

To define a trunk port, encapsulation must not be set as “auto”. Dot1q is generally used.

Switch(config-if)# **switchport mode trunk**

* Set the mode to trunk.

// Defining an Access port

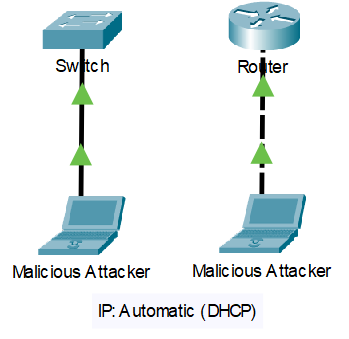
Switch(config-if)# **switchport mode access**

* Set the mode to access.

**Cisco Discovery Protocol (CDP) Flooding**

**Cisco Discovery Protocol** (CDP) is a CISCO proprietary protocol used to request information about directly connected devices, such as model, IOS, and IP. In order to exchange information, devices first need to establish a neighborship. Establishing a neighborship expends resources on the device. Not a lot of resources, but if many CDP requests are sent to a device in a short period of time, the processor runs out and stops managing common tasks. CDP Flooding is your run-of-the-mill **denial of service** (DOS) attack where an attacker overloads a device’s processor by repeatedly sending CDP packets.

Topology



*All you need for this attack is a connection to a CISCO device. In this example, I only attack a switch, but the same process can be done for a router, or practically any other CISCO device.*

For reference, let’s take a gander at how processes on a Switch look *before* a CDP Flood. Here are some of the commands I used:

Device# **show cdp traffic**

* Get the total packets output and input on the device

Device# **show processes cpu sorted | i CPU utilization | CDP Protocol**

* Check the amount of CPU usage expended by CDP

Device# **show processes cpu history**

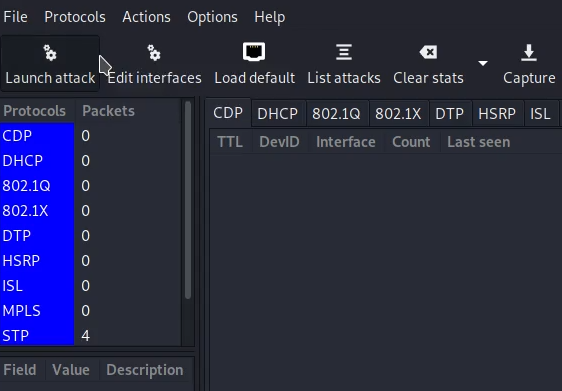
* Graphically see the CPU percentage expended over time

|  |
| --- |
| Outputs |
| Switch#show cdp traffic  CDP counters :  Total packets output: 589, Input: 0  Hdr syntax: 0, Chksum error: 0, Encaps failed: 0  No memory: 0, Invalid packet: 0,  CDP version 1 advertisements output: 0, Input: 0  CDP version 2 advertisements output: 589, Input: 0  Switch#show processes cpu sorted | i CPU utilization | CDP Protocol  CPU utilization for five seconds: 4%/0%; one minute: 5%; five minutes: 5%  156 9 3142 2 0.00% 0.00% 0.00% 0 CDP Protocol  Switch#show processes cpu history  554444444444444444444455555444444444444444444444444444444666  100  90  80  70  60  50  40  30  20  10 \*\* \*\*\*\*\* \*  0....5....1....1....2....2....3....3....4....4....5....5....6  0 5 0 5 0 5 0 5 0 5 0  CPU% per second (last 60 seconds)  \* = maximum CPU% # = average CPU% |

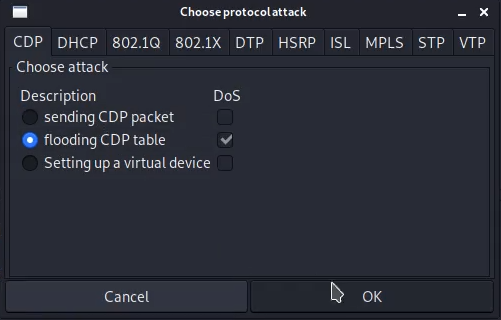
There are no notable processes hindering the CPU of the switch.

**CDP Flooding with Yersinia**

Begin by opening Yersinia, then click “Launch attack”.



When prompted by the dialogue box, enter the CDP tab and flood the CDP table.



Congratulations, this is all you need to do for a CDP Flooding attack.

Hopping back on the switch, I ran the commands from earlier. CDP annihilated the performance of the switch.

|  |
| --- |
| Outputs |
| Switch#show cdp traffic  CDP counters :  Total packets output: 45, Input: 32509  Hdr syntax: 0, Chksum error: 0, Encaps failed: 0  No memory: 0, Invalid packet: 0,  CDP version 1 advertisements output: 1, Input: 32509  CDP version 2 advertisements output: 44, Input: 0  Switch#show processes cpu sorted | i CPU utilization | CDP Protocol  CPU utilization for five seconds: 99%/24%; one minute: 82%; five minutes: 32%  156 51464 3412 15083 34.56% 29.34% 10.42% 0 CDP Protocol  Switch#show process cpu history  999999999999999999999999999999999999999999999999999999999999  889999999999888889999988888999999999988888999998888899999999  100 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  90 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  80 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  70 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  60 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  50 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  40 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  30 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  20 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  0....5....1....1....2....2....3....3....4....4....5....5....6  0 5 0 5 0 5 0 5 0 5 0  CPU% per second (last 60 seconds)  \* = maximum CPU% # = average CPU% |

Mitigation

CDP Flooding is one of the simplest attacks I’ve performed so far. With devastating effects, it should be clear CDP must not be overlooked. Luckily, the solution is rather straightforward – disable it. Unless you are an engineer in a constantly changing network, there is no desperate need for CDP.

Mitigation Commands

Device(config)# **no cdp run**

* Completely disable CDP on a device

Every CDP packet will be dropped, and the device will not run Cisco Discovery Protocol.

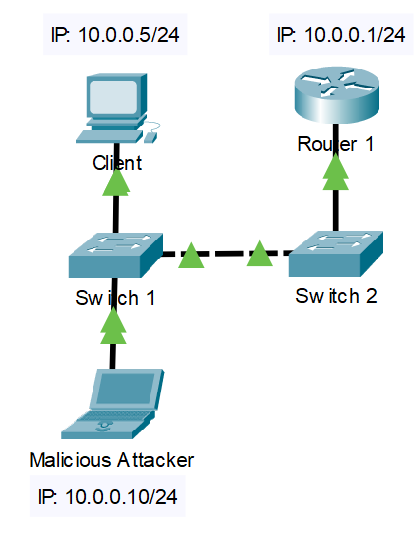
Device(config-if)# **no cdp enable**

* Rather than shutting down CDP, disable it on an interface

**Mapping Networks with NMAP**

Secure networks will likely have all the mitigations we’ve covered in place. Which is a great first step, but vulnerabilities still exist. Attackers can use NMAP (a network mapper) to find open, exploitable ports and software versions of devices on a network. In this section, I will try to analyze the data gained through running NMAP commands to figure out potential vulnerabilities in my network.

Topology



Let’s start by running the command **nmap *-O -sV 10.0.0.0/24*** (see command section for reference).

|  |
| --- |
| $ nmap -O -sV 10.0.0.0/24  Starting Nmap 7.91 ( https://nmap.org ) at 2021-11-01 14:59 PDT  mass\_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid servers with --dns-servers  Nmap scan report for 10.0.0.1  Host is up (0.0011s latency).  Not shown: 999 closed ports  PORT STATE SERVICE VERSION  23/tcp open telnet Cisco router telnetd (password required but not set)  MAC Address: 50:1C:B0:2D:71:01 (Cisco Systems)  OS details: Cisco 836, 890, 1751, 1841, 2800, or 2900 router (IOS 12.4 - 15.1), Cisco Aironet 1141N (IOS 12.4) or 3602I (IOS 15.3) WAP, Cisco Aironet 2600-series WAP (IOS 15.2(2))  Network Distance: 1 hop  Service Info: Device: router  Nmap scan report for 10.0.0.5  Host is up (0.00090s latency).  Not shown: 995 closed ports  PORT STATE SERVICE VERSION  135/tcp open msrpc Microsoft Windows RPC  139/tcp open netbios-ssn Microsoft Windows netbios-ssn  445/tcp open microsoft-ds?  2179/tcp open vmrdp?  5357/tcp open http Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)  MAC Address: A4:BB:6D:B0:CE:2A (Dell)  Device type: general purpose  Running: Microsoft Windows 10  OS CPE: cpe:/o:microsoft:windows\_10  OS details: Microsoft Windows 10 1709 - 1909  Network Distance: 1 hop  Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows  Nmap scan report for 10.0.0.10  Host is up (0.000082s latency).  All 1000 scanned ports on 10.0.0.10 are closed  Too many fingerprints match this host to give specific OS details  Network Distance: 0 hops  OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .  Nmap done: 256 IP addresses (3 hosts up) scanned in 29.04 seconds |

Practically out of thin air, I’m able to view information on three devices NMAP found:

* A Router
  + IP: *10.0.0.1/24*, MAC: *50:1C:B0:2D:71:01*
  + IOS version likely in the range 12.4 to 15.1
  + 1 Open port
    - 23 – Telnet
  + 1 hop away
* A Host
  + IP: *10.0.0.5/24*, MAC: *A4:BB:6D:B0:CE:2A*
  + Dell running Windows 10
  + 5 Open ports
    - 135 – msrpc
    - 139 – netbios-ssn
    - 445
    - 2179
    - 5357 – http
  + 1 hop away
* The Kali Laptop running the attack

IP and MAC addresses are useful for countless reasons. Open ports give an attacker entryway into a device. By knowing the hop count, we roughly deduce what the topology looks like. A keen eye might spot port 445 (SMB). Port 445 was significantly venerable to an attack called *EternalBlue* back in SMB version 1.0 but is disabled on Windows 10 and above.

Mitigation

There is no straightforward counter to NMAP. Unlike the other attacks so far, NMAP is basically countered by NMAP. It should be used beforehand to find vulnerabilities (mainly open ports) that would be better closed. Close any unused ports.

Process

Installing Kali

As my only objective was to perform the attacks above, I decided to search for an attack tool. There are many applications online that can run various attacks, but the one with the most potential appeared to be a tool called Kali Linux. It came with a lot of base programs and those that weren’t pre-installed could be via the command-line interface. At least that’s what I thought.

I installed Kali Linux as a windows application, allowing me to run it natively from the Windows 10 operating system. *Ettercap* was a common name that popped up during my research, so I wanted to launch it on Kali. However, the Kali distribution for windows didn’t automatically come with Ettercap, so I had to install it manually. That’s where my first problem arose. I ran the install command for Ettercap, some information was released into the console, and just like magic Ettercap was now a program I could activate. Sadly, like magic, Ettercap didn’t work. Or at least when it opened, it would immediately crash. I narrowed the problem down to a couple of missing dependencies. I installed those dependencies only to have new dependencies reveal their absences. This became an endless cycle with no clear end in sight.

When the pool of dependencies became too deep for me to see the bottom, I started looking into other options to run Kali. At this point in time, I didn’t realize Kali was an actual operating system, like Windows or Mac. *Kali Linux* implied to me “a tool on Linux”. Of my knowledge at the time, Ubuntu was the only Linux distribution. Therefore, if I got Ubuntu, Kali would come installed, right?

That left me with two options if I wanted to run Ubuntu: use a Windows Virtual Machine (VM) or image it on a spare laptop. Both were equally viable since I was in a position where nothing would be lost if something went wrong. I chose to try a VM, Windows Subsystem for Linux (WSL), which I got working after following a tutorial online. The VM was extremely slow and glitchy and I didn’t have any strong feelings towards fixing the issues, so I settled on imaging the laptop. In hindsight, I should have done more research on Kali instead of blindly firing and hoping fixes would miraculously land.

The software I chose to flash Ubuntu on my USB drive was BalenaEtcher. Etcher’s sleek, modern design made the interface clear and the instructions straightforward. Select image, select drive. With Ubuntu flashed on my USB, I opened the spare laptop and booted from the USB. A couple of questions and drive partitions later, I could launch Ubuntu. Which is great because Ubuntu was totally what I needed. Except it wasn’t.

In the middle of booting Ubuntu, one of my peers entered the room and asked what I planned on doing with it. I replied, explaining how I was about to install Kali, when he told me Kali Linux was its own operating system. Luckily, I had just gotten the experience of flashing USBs and imaging drives, so I repeated the steps once more for Kali. With Kali now imaged, I opened the console and entered *sudo ettercap -G*, the command for launching Ettercap Graphical, as a test to see if programs worked. It ran pre-installed without any dependency issues.

Kali Features

A handful of problems I’ve run across in Kali have been solved, simply by adding *sudo* to the start of the command. *Sudo* specifies the highest-level root privileges a user can obtain and is often required for higher-level commands to run properly. For example, Ettercap will initially open without *sudo* privileges but then crashes when running major actions.

There are many working tutorials online for most of the attacks, however sometimes the commands were deprecated. Specifically, during my VLAN hopping attack, the deprecated command *vconfig* was used. *vconfig* used to be the way to create VLANs on Kali, but when run, would only produce an output telling the user to use *iproute2* instead. So, I searched up the *iproute2* documentation and found a command that claimed to set up a VLAN. I replaced deprecated sections of tutorials with synonymous *iproute2* commands, piecing together a working alternative.

Ettercap

Within the Ettercap interface, there is a list of hosts. But sometimes the hosts don’t show up automatically. I’m particularly proud of figuring out this problem without searching online and instead by exploring the Ettercap’s layout and options; one can scan for hosts manually in the hosts settings. I was also confused about the *targets* system Ettercap has. In an Ettercap attack, two IPs need to be specified. I believe this is because one is set as the router while the other is set as the target host. Generally, throughout references I’ve seen, the best practice is to set the router as Target 1 and the victim as Target 2.

Yersinia

After installing Yersinia and running a DHCP attack, I learnt the starvation part worked while the rogue server failed to create. At one point when I launched Yersinia, I noticed a warning in the console stating I didn’t have *libvlc-bin* installed, a dependency for Yersinia. Considering this might be a reason as to why the rogue server failed, I set about installing the package. A website claimed the command *sudo apt-get install -y libvlc-bin* would work, so I ran it then re-installed Yersinia. I seemed to be able to create rogue servers afterwards, though I’m not sure whether to credit the above or just restarting the computer.

Conclusion

In a couple of weeks, someone with no knowledge on Kali Linux or penetration testing was able to perform attacks on a live network. Anyone can learn these tools, so it really is important to set up proper defenses on a network containing sensitive data. In learning the attacks, I feel I understand the protocols exploited to a higher depth.