TÉCNICO LISBOA

Segurança e Arquiteturas Avançadas de Redes

Module 1 Report

METI 2022/2023

Report written by:

Group 7

Afonso Dias - ist193571

João Pereira - ist193587

Francisco Correia - ist176470

TÉCNICO LISBOA

Table of contents

MAC table overflow

Introduction

MAC stands for Media Access Control and is an address and a unique identifier assigned to network interfaces with a 48-bit format expressed in hexadecimal and is typical written in the format of six groups of two hexadecimal digits, for example: “00:1A:2B:3C:4D:5E”.

It is also known as a physical address (or Ethernet address) and is assigned by the manufacturer of the NIC (Network Interface Card) or the networking device itself.

A MAC table, also known as a MAC address table or forwarding table, is a data structure used in network switches to map MAC addresses to the corresponding switch ports. IT is a crucial component of Ethernet switching and plays a key role in forwarding network traffic efficiently within a local area network (LAN).

The MAC table maintains a list of MAC addresses along with the associated switch ports. Each entry in the table typically includes the MAC address and the port number. The MAC table is dynamically updated as the switch continues to receive Ethernet frames.

The MAC table is crucial for the operation of Layer 2 switching and enables the switch to make forwarding decisions based on MAC addresses. It allows the switch to create a logical map of the network by associating MAC addresses with specific switch ports. This helps optimize network performance, reduces unnecessary network traffic, and improves the overall efficiency of the LAN.

The next figure represents the network topology used for this simulation along with each device IP and Mac address.

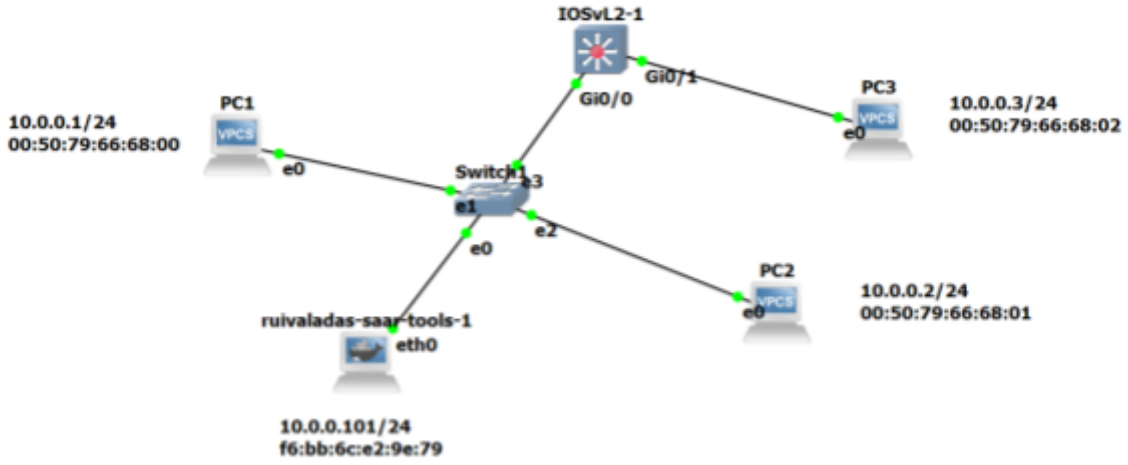


Figure 1: Network Topology

Figure 2 illustrates a normal functioning of a MAC address table. The command used shows, on IOSvL2-1, the mac addresses of each machine and the interface used to reach

that machine. For example for PC3, the interface would be G0/0 and the MAC address 00:50:79:66:68:02.

```
Switch#show mac address-table
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    72e9.1e4e.dc62    DYNAMIC     Gi0/3
   1    c201.aeb3.0000    DYNAMIC     Gi0/0
   1    c202.aed3.0000    DYNAMIC     Gi0/1
   1    c203.aef3.0000    DYNAMIC     Gi0/2
Total Mac Addresses for this criterion: 4
```

Figure 2: MAC address table

Attack overview

A MAC table overflow, also known as MAC table exhaustion or MAC address flooding attack, occurs when the MAC address table of a network switch becomes full or exceeds its capacity. This can lead to various network disruptions and potentially allow for malicious activities.

During a MAC table overflow attack, an attacker floods the switch with a large number of forged Ethernet frames, each containing a different source MAC address. The goal is to overwhelm the switch's MAC table and consume all available entries by sending a flood of frames with unique MAC addresses and filling up the MAC table more quickly than the switch can handle itself.

As the MAC table becomes full, the switch starts evicting MAC address entries to make room for new entries. The eviction process typically follows a least recently used (LRU) algorithm, removing the least recently seen MAC addresses from the table, however, if the attacker continues flooding the switch with new MAC addresses, legitimate MAC address entries will also be evicted from the table.

This attack can lead to several consequences, they can be:

- **Network Disruptions**, for example network congestion and performance degradation since the MAC table is overwhelmed;
- **Denial-of-Service (DoS) Attacks**: Frames can fail to be properly forward and this can disrupt the network connectivity for legitimate users or devices;
- **MAC address Spoofing**: By flooding the network with forged MAC addresses, the attacker can confuse network devices and potentially gain unauthorized access or intercept network traffic leading to other attacks;

The script, programmed in python (version 3), used to perform the attack goes by the name “camov.py”. This code receives the number of packets the attacker wants to send and generates them with a random source and destination MAC and IP addresses using the “scapy” library. The purpose of this script is to simulate a MAC overflow attack.

We can see in Capture 1, the IP packet flow when the attacker runs the script using new IP’s and MAC addresses and flooding the switches with them. In Figure 3, we observe with more detail the packet content used to perform the MAC table overflow.

ip

No.	Time	Source	Destination	Protocol	Length	Info
11	14.232447	197.67.166.10	42.58.216.224	IPv4	34	
12	14.240472	194.229.132.115	37.135.109.131	IPv4	34	
13	14.240488	92.211.231.105	112.173.31.148	IPv4	34	
14	14.240497	81.30.5.199	131.211.253.58	IPv4	34	
15	14.240503	225.41.116.138	120.95.193.117	IPv4	34	
16	14.240510	253.168.236.112	142.118.39.187	IPv4	34	
17	14.240517	98.51.148.0	41.53.237.133	IPv4	34	
18	14.240524	72.219.96.52	6.189.248.60	IPv4	34	
19	14.240530	151.51.244.252	68.43.112.212	IPv4	34	
20	14.240537	116.3.106.12	110.241.96.181	IPv4	34	

Capture 1: Packet flow

```
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 20
    Identification: 0x0001 (1)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: IPv6 Hop-by-Hop Option (0)
    Header Checksum: 0xa66b [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 92.211.231.105
    Destination Address: 112.173.31.148
```

Figure 3: Packet details

Countermeasures

In order to mitigate MAC table overflow attacks, network administrators can implement the Port Security measure by configuring the switch to limit the number of MAC addresses allowed on each switch port. This helps prevent excessive MAC addresses from flooding the MAC table. The next figure contains the commands used for implementing Port Security on the IOSvL2 switch . We will explain briefly how Port Security works and why every line of code we see below is important to prevent Mac Table Overflow attacks.

```
1  int g0/0
2  switchport mode access
3  switchport port-security
4  switchport port-security maximum 1
5  switchport port-security violation protect
6  switchport port-security aging time 1
7
```

Figure 4: Port Security configuration

After selecting the interface GigabitEthernet 0/0 in the first command, we configure that interface as an access port, meaning it connects to an end device rather than another switch. **Switchport port-security command** enables port security on the selected interface and **switchport port-security maximum 1** sets the maximum MAC address allowed to be learned and associated with this port to 1. **Switchport port-security violation protect** is a command that prevents any further traffic passing through the port in case of violation. To finish **Switchport port-security aging time 1** declares that after one minute of inactivity of a MAC address, the address is removed from the running config.

We can see in Figure 5 the Port security enabled for interface g0/0 on IOSvL2 switch after the configuration.

```
Switch#show port-security interface g0/0
Port Security              : Enabled
Port Status                : Secure-up
Violation Mode             : Protect
Aging Time                 : 1 mins
Aging Type                 : Absolute
SecureStatic Address Aging : Disabled
Maximum MAC Addresses      : 1
Total MAC Addresses        : 0
Configured MAC Addresses   : 0
Sticky MAC Addresses       : 0
Last Source Address:Vlan   : 0000.0000.0000:0
Security Violation Count   : 0

Switch#
```

Figure 5: Show Port security command

```
PC1> show

NAME   IP/MASK              GATEWAY          MAC                LPORT  RHOST:PORT
PC1    10.0.0.1/24          10.0.0.2         00:50:79:66:68:00  20010  127.0.0.1:20011
       fe80::250:79ff:fe66:6800/64

PC1>
```

Figure 6: IP and Mac address for PC1

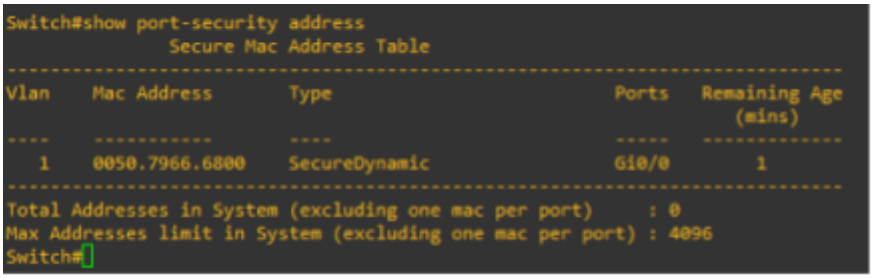


Figure 7: MAC address table showing the MAC address for PC1 after ping

As we can see in Figure 7, after the ping, the PC1 IP address joins the switch MAC address table. And since the PortSecurity configuration is set to allow the maximum of one MAC address we cannot ping to PC3 when the switch's MAC address table is already full, so the violation triggers and the traffic is blocked as we can observe in figure 8 and 9.

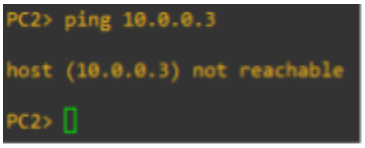


Figure 8: PC2 tries to ping PC3

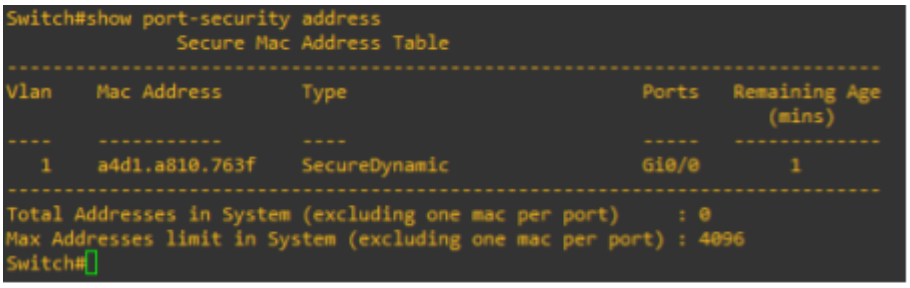


Figure 9: MAC table with one of the addresses from the attacker's script

Switch Compatibility

After investigating which switches support Port Security we found the following switches:

- **Cisco Catalyst 2960 series;**

And after investigating which switches don't support Port Security we found the following switch, for example:

- An unmanaged switch. It can be a basic network home based switch like **NETGEAR GS105**.

TÉCNICO LISBOA

Attack Feasibility

The feasibility of this attack depends on the characteristics of the target switch and its ability to handle the flood of forged mac addresses. In our opinion the feasibility will depend on several factors:

- **Switches technologies**: Recent switches have a MAC table size bigger than the previous ones making them more difficult to overflow. The attacker's success will depend on the robustness of the switch's hardware and software to handle the flood of frames and that can mitigate the impact on the network performance;
- **Countermeasures implementation**: As we mentioned before network administrators can implement measures to prevent MAC table overflow's attacks like Port Security. They can also implement and deploy intrusion prevention systems (IPS) or switches with built-in protection against MAC flooding attacks. These countermeasures also depend on the switches used and chosen, not all switches have these features;
- **Switch fail behavior**: If MAC address table becomes full it can enter into a fail-open mode and behaves as if it were a hub, broadcasting all incoming frames to all connected ports which can allow an attacker to intercept traffic and possibly help him to make other attacks and discover more about the network itself;
- **Attacker skills and resources:**: Being the main goal of the MAC table overflow attack to flood the switch with a high volume of forged frames with different source MAC addresses, the attack performance will depend on how the attack was programmed and on the attacker resources;

In conclusion, the feasibility of a MAC table overflow was probably diminished due to the improvements of network switch technologies, their security measures and switches hardware resources. Additionally, network administrators need to be aware of this kind of attacks and take proper security measures such as port security and MAC limiting which will reduce the probability of success of these attacks and mitigate their impact.

DHCP spoofing

Introduction

DHCP stands for Dynamic Host Configuration Protocol and is used for automatically assigning Internet Protocol (IP) addresses and other communication parameters to devices connected to a certain network.

A DHCP Discover message is broadcasted when a client is trying to connect to a network and a DHCP offer message is the response from the server carrying the client's MAC address, the offered IP address, the subnet mask, the lease duration and the server IP address, to successfully assign an IP address to the client. The next figure demonstrates the message exchange flow between a client and two DHCP servers.

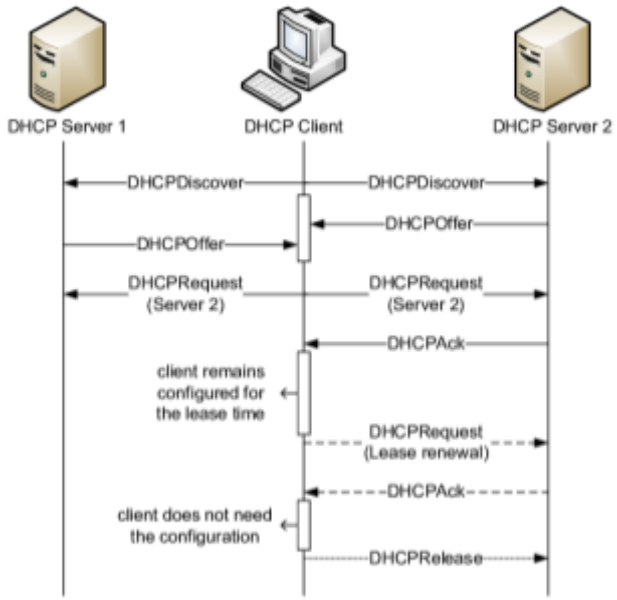


Figure 10: DHCP messages flow

We can see below in Figure 11 the network topology used to simulate the DHCP spoofing attack.

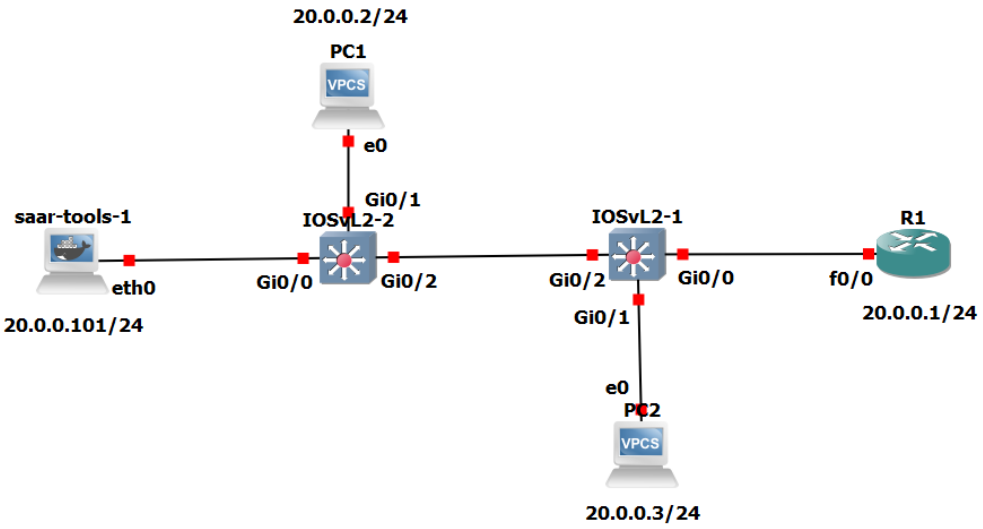


Figure 11: Network Topology

The list of commands used to configure the DHCP server in R1 are the following :

```
conf t
service dhcp
ip dhcp pool 0
network 20.0.0.0 /24
default-router 20.0.0.1
exit
Int f0/0
ip addr 20.0.0.1 255.255.255.0
no shutdown
end
write
```

Figure 12: DHCP configuration

We configured on interface F0/0 of R1 the IP address 20.0.0.1, being the default router with the DHCP service enabled for the network 20.0.0.0/24.

TÉCNICO LISBOA

Attack overview

The attack, called DHCP Spoofing, consists in impersonating the DHCP server by broadcasting a DHCP Offer, depending on the following race condition: The DHCP Offer of the rogue DHCP server must arrive first at the victim than the one of the legitimate DHCP server. In the network topology of this simulation we have two PC's with different paths to the rogue Server to demonstrate how this race condition works.

The attacker sets up a rogue DHCP server on the network, or it can also compromise a legitimate DHCP server, and is configured to respond to DHCP discover messages from clients with a DHCP offer message. This offer includes IP address, subnet mask, default gateway, DNS server and other DHCP options. When the client receives the DHCP offer messages, makes an DHCP Request in order to request the offered IP configuration. The rogue DHCP server intercepts the DHCP request message and responds with a DHCP acknowledgement message, confirming the lease of the offered IP configuration. The client then assumes that the rogue DHCP server is the legitimate server and configures its network interface accordingly.

This attack can lead to several consequences, they can be:

- **Unauthorized Control**: The attacker gains unauthorized control over the client's network connection;
- **Traffic Manipulation**: With the unauthorized control, the attacker can intercept, monitor or modify the client's network traffic which enables various activities, for example, man-in-the-middle attacks, data interception, session hijacking and eavesdropping;
- **Network Disruptions**: by providing incorrect details, the attacker can cause network conflicts or even exhaust network resources as a component of a Denial-of-Service (DoS) attacks;

In order to run the script containing the attack, we ran the script, named "*dhcpspoof.py*" in python programming language (version 3) with the following arguments, as we can observe in the next Figure:

- **Interface**: The name of the network interface through which the DHCP spoofing attack will be conducted. In this case, it is eth0;
- **Pool_start**: The starting IP address of the IP pool that will be used for assigning IP addresses to the spoofed DHCP clients. This should be specified in the format of a valid IPv4 address. In this case it is 20.0.0.2;
- **Subnet_mask**: The subnet mask to be used for the network. It defines the network range and helps determine the network and broadcast addresses. In this case it is 255.255.255.0;
- **Gateway**: The IP address of the default gateway that will be provided to the spoofed DHCP clients. In this case it is 20.0.0.101;
- **Dns_server**: The IP address of the DNS (Domain Name System) server that will be provided to the spoofed DHCP clients. In this case it is 20.0.0.101;

```
root@ruivaladas-saar-tools-1:/home# ip address add 20.0.0.101/24 dev eth0
root@ruivaladas-saar-tools-1:/home# python3 dhcpspoof.py eth0 20.0.0.2 20.0.0.254 255.255.255.0 20.0.0.101 20.0.0.101
Leased address 20.0.0.2
```

Figure 13: DHCP spoofing ran in the attacker's console

We can see in the capture 2 the DHCP Discover message from the PC1 broadcasted across the network and after received from the Rogue DHCP server and the DHCP offer sent to the PC1 successfully impersonating a legitimate DHCP server.

dhcp

No.	Time	Source	Destination	Protocol	Length	Info
8	12.135902	0.0.0.0	255.255.255.255	DHCP	406	DHCP Discover - Transaction ID 0x3a83e229
9	12.167310	20.0.0.101	20.0.0.2	DHCP	338	DHCP Offer - Transaction ID 0x3a83e229
10	12.203290	20.0.0.1	20.0.0.2	DHCP	342	DHCP Offer - Transaction ID 0x3a83e229
12	13.140894	0.0.0.0	255.255.255.255	DHCP	406	DHCP Request - Transaction ID 0x3a83e229
13	13.161402	20.0.0.101	20.0.0.2	DHCP	338	DHCP ACK - Transaction ID 0x3a83e229

Capture 2: DHCP spoofing messages flow between PC1 and the rogue DHCP server

Countermeasure

In order to countermeasure this attack, switches often use DHCP Snooping, a security feature that helps prevent unauthorized or malicious DHCP activity on a network.

It operates by inspecting and validating DHCP messages exchanged between DHCP clients and servers.

This security measure defines trusted and untrusted ports. Trusted ports are those connected to known and trusted DHCP servers, while untrusted ports connect to end-user devices or unknown/external networks. Also, DHCP snooping inspects DHCP packets traversing untrusted ports. It examines attributes such as source ip address, destination IP address, MAC address and other DHCP specific fields.

DHCP Snooping has a binding table, the switch maintains a DHCP snooping binding database. This table maps the MAC address of a device to the IP address it has been assigned by a DHCP server. The binding table is built based on legitimate DHCP server responses received on trusted ports. This table is also used for comparing with DHCP packet information in order to filter unauthorized DHCP traffic.

DHCP Snooping has an DHCP Option 82, also known as the "DHCP Relay Agent Information Option". This option adds information about the switch and port from which the DHCP request originated, helping to further validate the DHCP message and prevent spoofing.

In order to implement DHCP spoofing we ran the following commands:

```
Switch#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#int g0/0
Switch(config-if)#ip dhcp snooping trust
Switch(config-if)#end
Switch#write
Building configuration...
```

Figure 14: DHCP Snooping configuration

We configured and claimed the interfaces connected between routers and switches as trusted (never connections to end-users although) with the command: “ip dhcp snooping trust” in order to prevent this attack.

Since PC1 is reached first from the rogue DHCP server, once the attacker wins the race condition as a result of the shorter path he has to the PC1 comparing to the legitimate DHCP server, it is still a victim of DHCP spoofing and we only could prevent this attack on PC2 as we can observe in the next packet capture where the DHCP Offer message is only received from the legitimate DHCP server :

dhcp

No.	Time	Source	Destination	Protocol	Length	Info
6	4.658651	0.0.0.0	255.255.255.255	DHCP	406	DHCP Discover - Transaction ID 0x4d8f6775
7	4.678942	20.0.0.1	20.0.0.3	DHCP	342	DHCP Offer - Transaction ID 0x4d8f6775
8	4.689832	20.0.0.101	20.0.0.3	DHCP	338	DHCP Offer - Transaction ID 0x4d8f6775
9	5.664473	0.0.0.0	255.255.255.255	DHCP	406	DHCP Request - Transaction ID 0x4d8f6775
10	5.687887	20.0.0.1	20.0.0.3	DHCP	342	DHCP ACK - Transaction ID 0x4d8f6775

Capture 3: DHCP Snooping messages flow after the implementation on PC2

In conclusion, this countermeasure is very useful to protect against attacks since it can block rogue DHCP servers that attempt to distribute unauthorized IP addresses to clients and it also can protect against DHCP starvation attacks, where an attacker exhausts the available IP address pool by requesting multiple addresses.

Switch Compatibility

After investigating which switches support DHCP Snooping we found the following switches:

- **Cisco Nexus 2000,3000,5000,7000 and 9000 series, Cisco Catalyst 6500 series and Cisco Catalyst 3650**.

And after investigating which switches don’t support DHCP Snooping we found the following switch, for example:

- **D-Link DGS-105**. This switch is a basic unmanaged switch typically used for small home or office networks. It lacks advanced security features like DHCP snooping, making it a simpler and more cost-effective option for basic networking needs but without the added layer of DHCP protection.

Attack feasibility

In our opinion, the feasibility of a DHCP Spoofing attack depends on several factors like:

- **Attackers skills and resources**: because this particular attack requires the attacker to have access to the target network. The attacker needs to be able to send DHCP packets to the network and intercept the responses. Also, the attacker needs a good understanding of the network's DHCP infrastructure and the IP address allocation process.

- **Network Security and Architecture**: If the network is actively monitored and anomalies are detected, it can help identify and prevent DHCP spoofing attacks. Regular network monitoring and analysis can detect unusual DHCP traffic patterns or multiple DHCP servers on the network, triggering alerts for potential attacks. In a poorly designed or insecure network environment where there are limited or no security measures implemented, DHCP spoofing attacks can be more feasible. For example, if there are no DHCP snooping or IP source verification mechanisms in place, it becomes easier for an attacker to successfully execute the attack.

- **Race condition:** Coupled with the network Architecture factor for the attack to be successful, the rogue DHCP server needs to reach the victim before the legitimate DHCP server.

- **Switches Technologies**: DHCP snooping, a security feature available on many network switches as we saw earlier, helps mitigate DHCP spoofing attacks by verifying the legitimacy of DHCP servers. If the network infrastructure includes switches with DHCP snooping enabled, it significantly reduces the feasibility of the attack.

To sum up, DHCP spoofing attacks are technically feasible but they are conditioned by the factors presented above. If proper security measures are implemented such as Port Security the likelihood and impact of such attacks can be significantly reduced.

ARP poisoning (MitM)

Introduction

ARP stands for Address Resolution Protocol and is used for mapping an Internet Protocol (IP) to a corresponding MAC (Media Access Control) address for communication purposes. When a device wants to communicate, for example making a ping, to another device on the same network, it needs to know the MAC address of the destination device. The ARP protocol resolves this mapping.

The next figure has a simplified overview of how ARP works. Basically, when a device wants to know the other device's MAC address broadcasts an ARP request packet containing the

message asking “Who has this IP address?” and the device with the corresponding IP address responds with an ARP reply, providing its MAC address. Finally the requesting device updates its ARP table with the IP-to-MAC mapping.

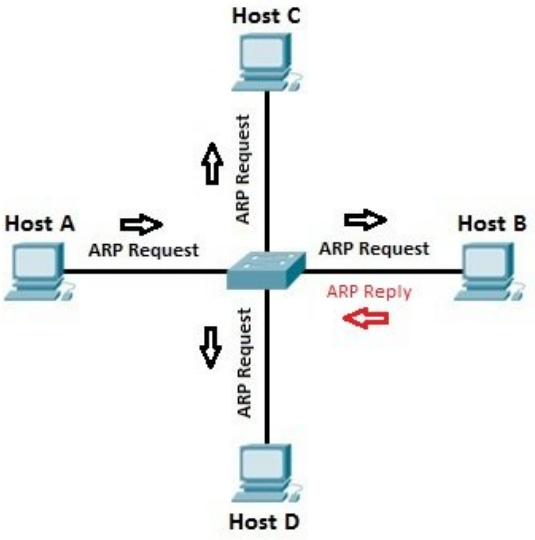


Figure 15: ARP messages flow

In order to simulate an ARP poisoning attack we used the following network topology:

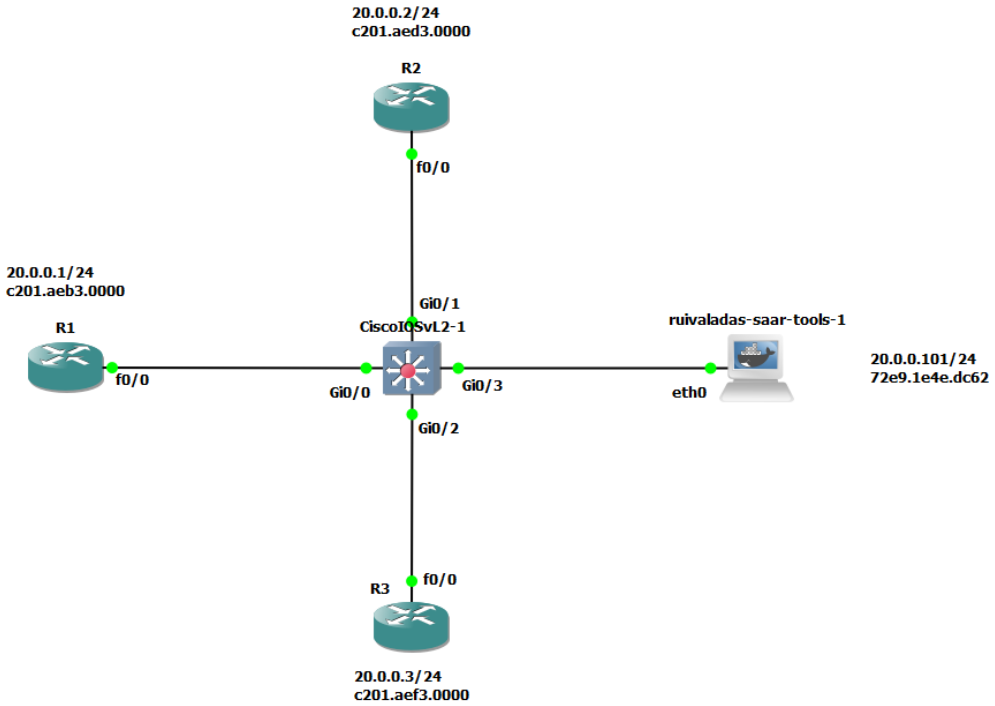


Figure 16: Network Topology

TÉCNICO LISBOA

Attack overview

An ARP poisoning attack consists in intercepting ARP requests and replies on a local network in order to manipulate ARP tables of a targeted device. To initiate, the attacker first scans the network and gathers information about the network topology (IP and MAC addresses of devices) to choose a target device. Secondly, the attacker sends forged ARP reply packets to devices inside the same network impersonating another device, for example the router/gateway, and after the targeted device receives the ARP reply, updates its ARP table, associating the attacker's MAC address with the IP address the attacker used to impersonate. This can be useful since once the targeted device wants to send network traffic intended for the router/gateway, for example, it sends it to the attacker's MAC address instead, so the attacker can intercept and analyze the traffic and possibly modify and inspect it. There are several consequences with this attack such as:

- **Denial-of-Service (DoS)**: The attacker can cause network disruptions and congestion or even isolate devices from network access by sending ARP replies and creating conflicts in ARP tables;
- **Session Hijacking:** The attacker can hijack for example web browsing sessions by associating their MAC address with the targeted device IP address with an open session;
- **Man-in-the-Middle (MitM):** The attacker can eavesdrop on sensitive information which can evolve to modify packet content and create further attacks;

The script used to perform ARP poisoning is called "arpmitm.py" and is written in the programming language python version 3. This script receives three main arguments:

- **Attacker Interface:** This argument specifies the attacker network interface on which the attack will be performed;
- **IP1 and IP2:** This argument specifies the IP address of the first victim (Victim 1) and second victim device (Victim 2), being the IP address of the devices that will be impersonated or targeted by the ARP poisoning attack.

This script starts by enabling IP forwarding and disabling ICMP redirects to ensure that the attacker's machine can forward packets between the victims.

The script sends to both victims forged ARP reply packets pretending in case of Victim 1 to be Victim 2 and in case of Victim 2 to be Victim 1, poisoning both ARP caches and causing their communication to be redirected through the attacker's machine. The ARP reply packets contain manipulated information, associating the attacker's MAC address with the IP address of the targeted victim. This allows the attacker to intercept, modify or monitor the network traffic exchanged between Victim 1 and Victim 2.

For example, we can run the attack by executing in the attacker's device console the following command: python3 arpmitm.py eth0 20.0.0.3 20.0.0.1.

The next capture illustrates R2 perspective after the ARP poisoning was performed. c2:03:ae:f3:00:00 is the version 6 of the Internet Protocol of the attacker's PC (ruivaladas-saar-tools-1).

```
    25 19.847469     e6:92:75:fa:93:19    c2:02:ae:d3:00:00    ARP          60 20.0.0.3 is at c2:03:ae:f3:00:00
    26 19.855505     e6:92:75:fa:93:19    c2:02:ae:d3:00:00    ARP          60 20.0.0.3 is at c2:03:ae:f3:00:00
    27 19.893026     e6:92:75:fa:93:19    c2:02:ae:d3:00:00    ARP          60 20.0.0.3 is at c2:03:ae:f3:00:00
    28 19.898734     e6:92:75:fa:93:19    c2:02:ae:d3:00:00    ARP          60 20.0.0.3 is at c2:03:ae:f3:00:00
    29 19.932782     e6:92:75:fa:93:19    c2:02:ae:d3:00:00    ARP          60 20.0.0.3 is at c2:03:ae:f3:00:00
    30 19.936480     e6:92:75:fa:93:19    c2:02:ae:d3:00:00    ARP          60 20.0.0.3 is at c2:03:ae:f3:00:00
    31 19.956953     e6:92:75:fa:93:19    c2:02:ae:d3:00:00    ARP          60 20.0.0.3 is at c2:03:ae:f3:00:00
```

Capture 4: ARP messages flow in R2

Countermeasure

In order to protect against these attacks, a countermeasure called Dynamic ARP Inspection (DAI) was configured on the switch.

DAI requires VLAN (Virtual Local Area Network) configuration because it operates at the Layer 2 level of the network and VLANs are used to segregate and isolate network traffic. Each VLAN has a separate broadcast domain which limits the ARP packets to the same VLAN and this isolation prevents ARP poisoning attacks from affecting devices in other VLANs. DAI can be applied and configured to operate independently on different VLANs within a network. VLAN has also a tagging system to mark each packet to a VLAN identifier which ensures that the switch can associate ARP packets with their respective VLANS and apply the appropriate DAI policies and validation checks. VLAN also has Access Control Lists (ACLs) and security policies to define specific rules and restrictions for each VLAN, including DAI related policies. ACLs also can be used to control which devices are allowed to send ARP packets and specify exceptions and actions based on the VLAN context.

We can observe below the VLAN implementation in switch CiscoIOSvL2-1 command line:

```
Switch#show vlan

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi0/0, Gi0/1, Gi0/2, Gi0/3
                                                Gi1/0, Gi1/1, Gi1/2, Gi1/3
                                                Gi2/0, Gi2/1, Gi2/2, Gi2/3
                                                Gi3/0, Gi3/1, Gi3/2, Gi3/3
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup

VLAN Type  SAID       MTU   Parent RingNo BridgeNo Stp  BrdgMode Trans1 Trans2
---- ----- ---------- ----- ------ ------ -------- ---- -------- ------ ------
1    enet  100001     1500  -      -      -        -    -        0      0
1002 fddi  101002     1500  -      -      -        -    -        0      0
1003 tr    101003     1500  -      -      -        -    -        0      0
1004 fdnet 101004     1500  -      -      -        ieee -        0      0
1005 trnet 101005     1500  -      -      -        ibm  -        0      0

Remote SPAN VLANs
------------------------------------------------------------------------------


Primary Secondary Type              Ports
------- --------- ----------------- ------------------------------------------
```

Figure 17: Switch VLAN information

Despite the fact the VLAN configuration should not be the same for the interface connected to the attacker, in this case Gi0/3, DAI was sufficient to prevent the ARP poisoning attack. Dynamic ARP Inspection (DAI) integrates DHCP Snooping, already explained in the previous section along with its configuration, its main purpose is to monitor and validate DHCP traffic to ensure that only authorized DHCP servers provide IP addresses to the clients. The verified IP-to-MAC bindings obtained from DHCP snooping can be used as trusted information by DAI. So in order to implement this countermeasure, a DHCP server

was configured in a c3725 router, in our network topology was R1. The commands used to enable the DHCP service were the following:

```
conf t
service dhcp
ip dhcp pool 0
network 20.0.0.0 /24
default-router 20.0.0.1
exit
int f0/0
ip addr 20.0.0.1 255.255.255.0
no shutdown
end
write
```

Figure 18: DHCP implementation on R1

R2 and R3 were configured as an end-user machines and to receive their IPs by DHCP with the commands below:

```
conf t
no ip routing
int f0/0
ip add dhcp
no shutdown
end
write
```

Figure 19: R2 and R3 configuration

In order to implement on vlan 1 the Dynamic ARP inspection we ran the following command in the CiscoIOSvL2-1 switch: "**ip arp inspection vlan 1**".

The categorization DHCP Snooping makes of trust and untrusted ports is helpful in case of being an untrusted port to validate the ARP packets. When an ARP packet arrives on an untrusted port, DAI examines the packet to verify its integrity, checking:

- source IP and making a MAC validation by comparing both to the switch's ARP cache content;
- If the sender's IP address is not being used by other device on the network, making an IP address Lease Validation;
- If the sender's IP-to-MAC binding is consistent with previous ARP records stored in the switch's Dynamic ARP Inspection table. This table is called ARP Inspection Database and stores IP-to-MAC bindings collected from trusted sources;

After DAI implementation we could observe for each interface their trust state:

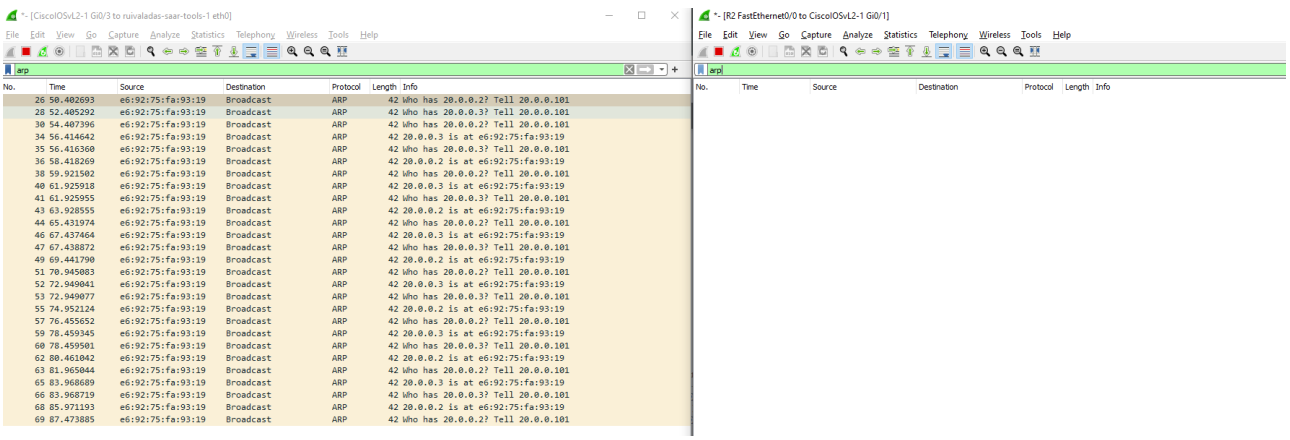
```
Switch#show ip arp inspection interfaces

Interface        Trust State     Rate (pps)     Burst Interval
---------------  -----------     ----------     --------------
Gi0/0            Untrusted               15                  1
Gi0/1            Untrusted               15                  1
Gi0/2            Untrusted               15                  1
Gi0/3            Untrusted               15                  1
Gi1/0            Untrusted               15                  1
Gi1/1            Untrusted               15                  1
Gi1/2            Untrusted               15                  1
Gi1/3            Untrusted               15                  1
Gi2/0            Untrusted               15                  1
Gi2/1            Untrusted               15                  1
Gi2/2            Untrusted               15                  1
Gi2/3            Untrusted               15                  1
Gi3/0            Untrusted               15                  1
Gi3/1            Untrusted               15                  1
Gi3/2            Untrusted               15                  1
Gi3/3            Untrusted               15                  1
```

Figure 20: ARP inspection on CiscoIOSvL2

When a packet is considered suspicious it is normally dropped, but the switch can also take other actions such as logging the event, sending an alert to the network administrator or applying a security policy configured for DAI.

As an example of what we just explained, the next figure illustrates two captures, the left one is the connection between the attacker (“ruivaladas-saar-tools-1”) to the switch, as we can confirm in our network topology and the right one the connection between R2 and the switch. After the countermeasure was implemented R2 didn't receive any ARP packets as the right capture demonstrates.



Capture 5: CiscoIOSvL2-1 and R2 perspective

We can observe also the switch command line denying the attack after we added the countermeasures in the next figure:

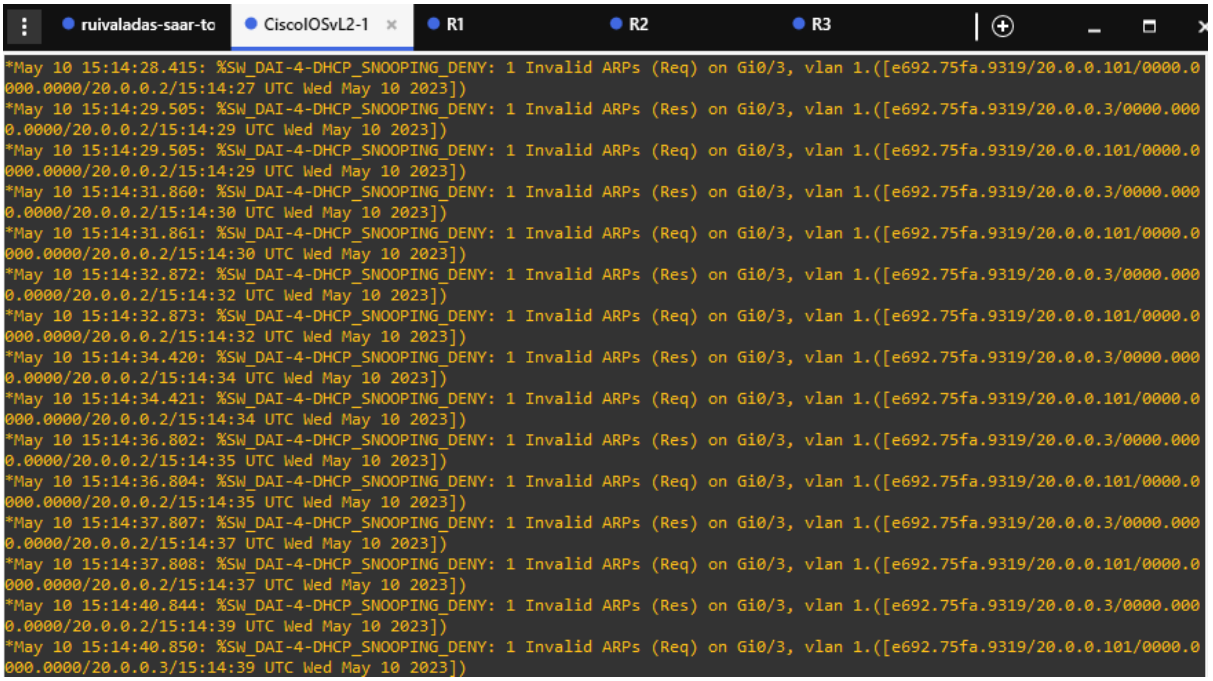


Figure 21: CiscoIOSvL2-1 command Line

Switch Compatibility

After investigating which switches support Dynamic ARP Inspection we found the following switches:

- An Cisco Catalyst 2960 series switch can be a switch used for Dynamic ARP Inspection since its purpose is to integrate security features and to be used in enterprise networks context.

And after investigating which switches don't support DHCP Snooping we found the following switch, for example:

- A normal unmanaged switch like NETGEAR GS108 can be an example of a switch that does not integrate advanced security functionalities such as DAI.

Attack feasibility

The success of this Attack will depend on several factors such as:

-Network Topology: If the network is a LAN (Local Area Network) with the same broadcast domain, it is easier to perform the attack by broadcasting ARP packets, intercepting and manipulating them;

- Network Security and architecture: Since the attacker can perform the attack with a specific goal such as session hijacking or man-in-the-middle attacks the vulnerabilities already present in the network are important to his success.

- Switches technology: The security features the switch has, like DAI or similar mechanisms, are crucial for detecting and preventing this attacks;

- Countermeasures: Implementation of measures by a network administrator are crucial. Several measures can be applied such as Dynamic ARP Inspection, port security, MAC address filtering, ARPwatch, Intrusion Detection Systems or security protocols like Secure ARP (SARP) in order to mitigate the risk or even prevent these attacks.

- Attacker skills and resources: The effectiveness of the attack will also depend on the attacker's knowledge and resources.

To conclude, it is crucial to implement appropriate security measures, apply the best practices and monitor network activity in order to reduce the feasibility and effectiveness of ARP Poisoning attacks.

DNS spoofing

Introduction

DNS stands for Domain Name System and it is a naming system with the main goal to translate human-readable domain names into IP (Internet protocol) addresses.

For example when a device connected to the internet wants to type a domain name into a web browser, such as facebook.com, the device sends a request to a DNS server to resolve the domain name to its corresponding IP address. The DNS server of the web server is 8.8.8.8

The network topology used to simulate a DNS spoofing attack is represented in the following figure:

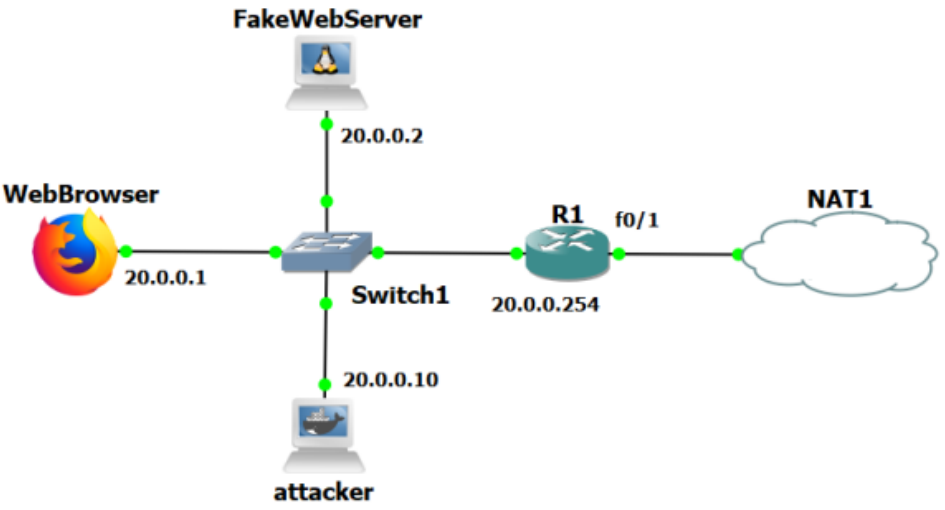


Figure 22: Network Topology

NAT stands for Network Address Translation, a technique that allows local network devices (internal network) to share a single public IP address for communication with an external network such as the Internet.

This is a segment of the router's startup config file that shows how the NAT setup was done:

```
interface FastEthernet0/0
ip address 20.0.0.254 255.255.255.0
ip nat inside
ip virtual-reassembly
duplex auto
speed auto
!
interface FastEthernet0/1
ip address dhcp
ip nat outside
ip virtual-reassembly
duplex auto
speed auto
!
ip forward-protocol nd
!
!
no ip http server
no ip http secure-server
ip nat pool ovrld 192.168.122.1 192.168.122.63 prefix-length 24
ip nat inside source list 7 pool ovrld overload
!
access-list 7 permit 20.0.0.0 0.0.0.255
no cdp log mismatch duplex
```

Figure 23: NAT setup

The lines 1 to 6 from the code provided above represent the configuration of the router's interface connected to the switch (FastEthernet0/0) with IP address 20.0.0.254 and a subnet mask of 255.255.255.0. This interface is designated as nat inside meaning that interface belongs to the internal network.

In the Router perspective the connection between R1 and NAT1(lines 8 to 13) is made through the interface FastEthernet0/1 and is configured to obtain its IP address dynamically through DHCP and as an outside interface for NAT, meaning that interface connects to the external network.

Before explaining the last lines of the NAT configuration it is important to elucidate what a NAT pool is. A NAT pool is a range of public IP addresses that can be used for translating private IP addresses to public IP addresses in a Network Address Translation configuration.

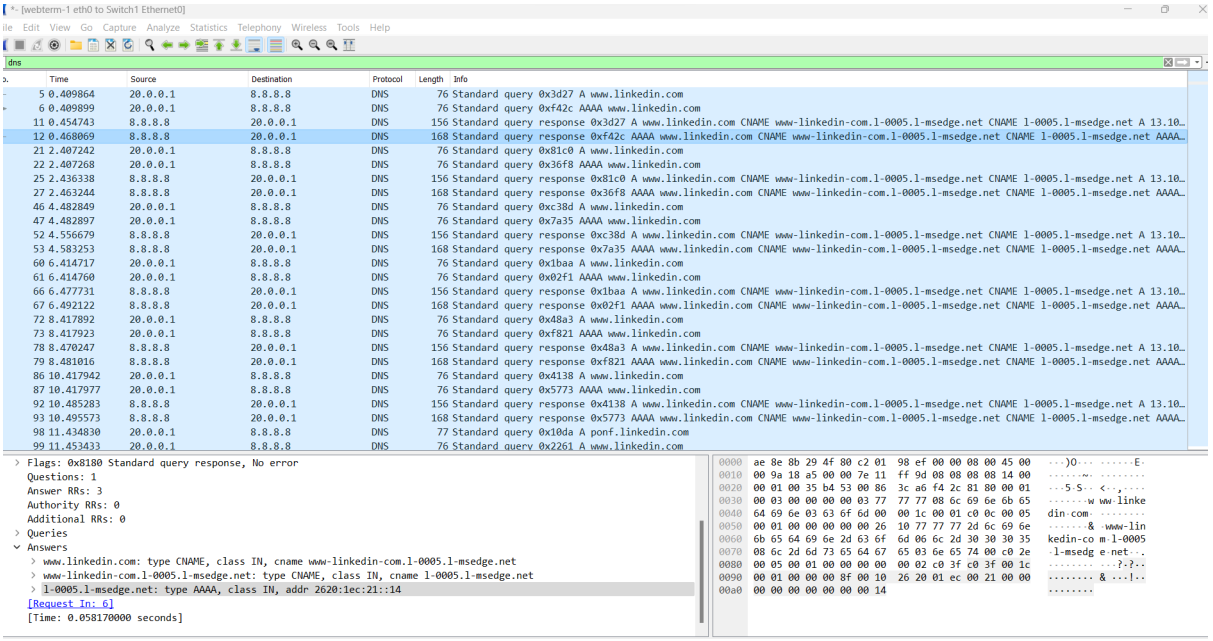
It allows multiple internal devices with private IP addresses to share a limited number of public IP addresses. So from the line 18 to 21, we create a map pool named "ovrl" with a range of IP addresses from 192.168.122.1 to 192.168.122.63 using a subnet mask of /24 and allowed multiple internal IP addresses to share a single external IP address(line 21). We also disabled any web server services (http and https).

To conclude we create in line 23 an access control list (ACL) entry that permits traffic from any IP address within the network range 20.0.0.0/24 with the number 7.

After analyzing the next two figures, we can conclude that the web server could connect to www.linkedin.com and the setup was realized with success.

Hostname	Family	TRR	Addresses	Expires (Seconds)
www.linkedin.com	ipv4	false	13.107.42.14 2620:1ec:21::14	119

Figure 24: Firefox DNS cache when accessing Linkedin



Capture 6: DNS packet exchange when browser access Linkedin

Attack overview

DNS spoofing consists in manipulating the Domain Name System (DNS) to redirect network traffic to malicious or unauthorized destinations by modifying responses to DNS queries leading users to be directed to fake or malicious servers.
DNS spoofing can be combined with MitM attacks such as ARP poisoning. We will proceed to explain how these two techniques work in tandem. The attacker initiates making an ARP poisoning attack, sending ARP requests to all IP addresses in the network and waits for ARP responses. Then proceeds to poison the ARP caches by sending forged ARP packets to associate the attacker's MAC address with the IP address of the target host so all network traffic intended for the target hosts will be redirected through the attacker's machine.
Then the attacker sniffs DNS packets on the network interface and creates a fake DNS response using the requested domain name and the spoofed IP address. It creates a DNS Resource Record object with the spoofed IP address. Finally the code sends the forged DNS response back to the original requester, replacing the legitimate DNS response.
In order to perform this attack we need three main arguments:

- **Interface:** network interface on which the attack will be performed;
- **hosts_file:** file containing the mappings between domain names and corresponding spoofed IP addresses;
- **network:** network range that will be targeted for the ARP poisoning attack;

The next figure represents the command used to run the attack:

```
root@ruivaladas-saar-tools-1:/home# python3 dnsspoof.py eth0 atk.txt 20.0.0.0/24

[*] Enabling IP forwarding and disabling ICMP redirects...

Scanning hosts for MAC addresses...
20.0.0.1
20.0.0.2
20.0.0.254
Poisoning hosts...
Finished poisoning ARP caches
```

Figure 25: Script detects interfaces and poisons their ARP tables

The following figures illustrates the success of the attack:

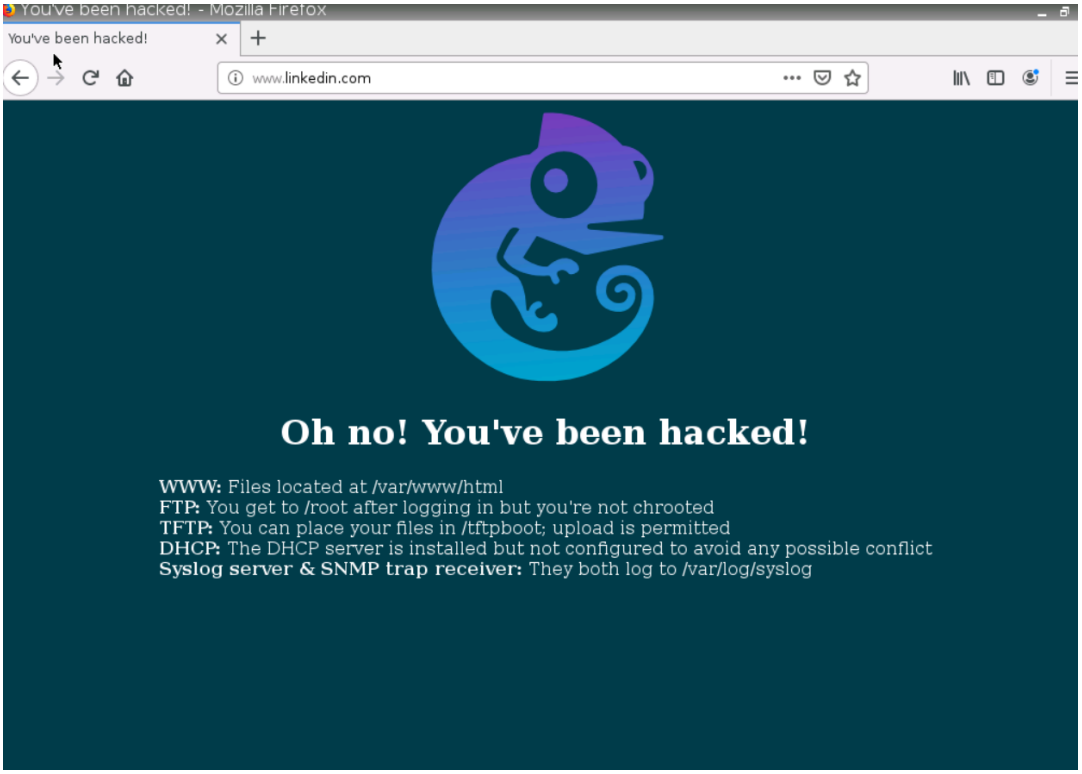
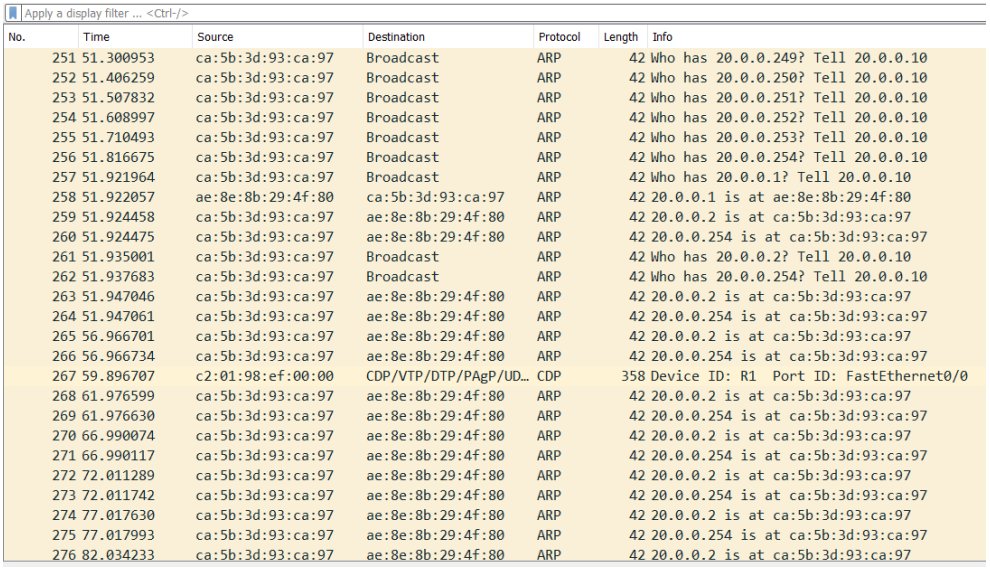
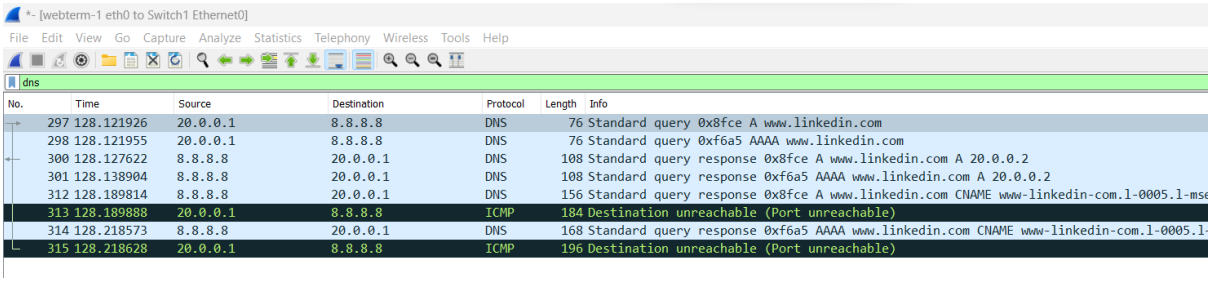


Figure 26: Webterm gets redirected to the page hosted by the fake web server when trying to connect to Linkedin



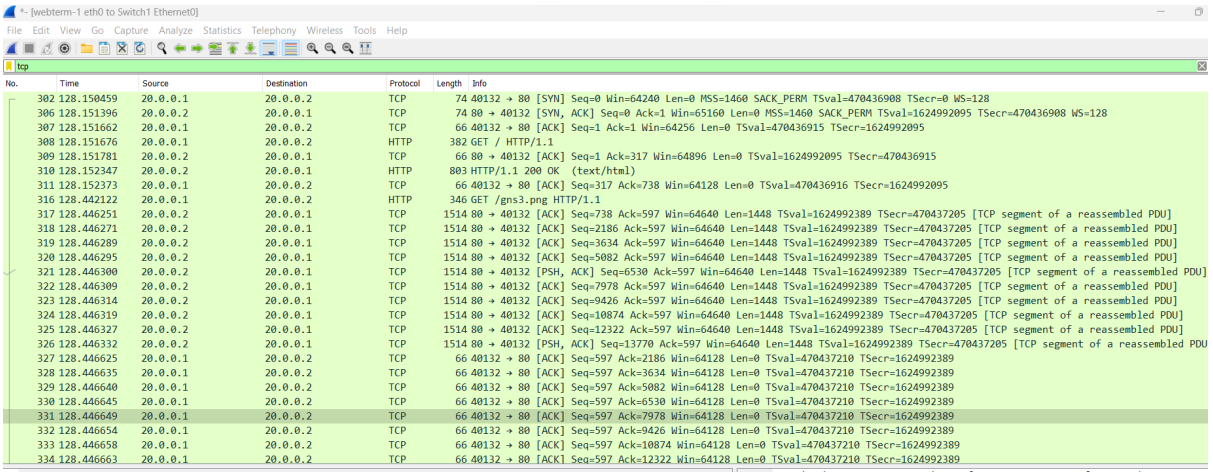
No.	Time	Source	Destination	Protocol	Length	Info
251	51.300953	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.249? Tell 20.0.0.10
252	51.406259	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.250? Tell 20.0.0.10
253	51.507832	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.251? Tell 20.0.0.10
254	51.608997	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.252? Tell 20.0.0.10
255	51.710493	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.253? Tell 20.0.0.10
256	51.816675	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.254? Tell 20.0.0.10
257	51.921964	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.1? Tell 20.0.0.10
258	51.922057	ae:8e:8b:29:4f:80	ca:5b:3d:93:ca:97	ARP	42	20.0.0.1 is at ae:8e:8b:29:4f:80
259	51.924458	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.2 is at ca:5b:3d:93:ca:97
260	51.924475	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.254 is at ca:5b:3d:93:ca:97
261	51.935001	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.2? Tell 20.0.0.10
262	51.937683	ca:5b:3d:93:ca:97	Broadcast	ARP	42	Who has 20.0.0.254? Tell 20.0.0.10
263	51.947046	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.2 is at ca:5b:3d:93:ca:97
264	51.947061	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.254 is at ca:5b:3d:93:ca:97
265	56.966701	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.2 is at ca:5b:3d:93:ca:97
266	56.966734	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.254 is at ca:5b:3d:93:ca:97
267	59.896707	c2:01:98:ef:00:00	CDP/VTP/DTP/PAgP/UD…	CDP	358	Device ID: R1 Port ID: FastEthernet0/0
268	61.976599	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.2 is at ca:5b:3d:93:ca:97
269	61.976630	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.254 is at ca:5b:3d:93:ca:97
270	66.990074	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.2 is at ca:5b:3d:93:ca:97
271	66.990117	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.254 is at ca:5b:3d:93:ca:97
272	72.011289	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.2 is at ca:5b:3d:93:ca:97
273	72.011742	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.254 is at ca:5b:3d:93:ca:97
274	77.017630	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.2 is at ca:5b:3d:93:ca:97
275	77.017993	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.254 is at ca:5b:3d:93:ca:97
276	82.034233	ca:5b:3d:93:ca:97	ae:8e:8b:29:4f:80	ARP	42	20.0.0.2 is at ca:5b:3d:93:ca:97

Capture 7: ARP packet flow when the script is ran



No.	Time	Source	Destination	Protocol	Length	Info
297	128.121926	20.0.0.1	8.8.8.8	DNS	76	Standard query 0x8fce A www.linkedin.com
298	128.121955	20.0.0.1	8.8.8.8	DNS	76	Standard query 0xf6a5 AAAA www.linkedin.com
300	128.127622	8.8.8.8	20.0.0.1	DNS	108	Standard query response 0x8fce A www.linkedin.com A 20.0.0.2
301	128.138904	8.8.8.8	20.0.0.1	DNS	108	Standard query response 0xf6a5 AAAA www.linkedin.com A 20.0.0.2
312	128.189814	8.8.8.8	20.0.0.1	DNS	156	Standard query response 0x8fce A www.linkedin.com CNAME www-linkedin-com.l-0005.l-mse
313	128.189888	20.0.0.1	8.8.8.8	ICMP	184	Destination unreachable (Port unreachable)
314	128.218573	8.8.8.8	20.0.0.1	DNS	168	Standard query response 0xf6a5 AAAA www.linkedin.com CNAME www-linkedin-com.l-0005.l-
315	128.218628	20.0.0.1	8.8.8.8	ICMP	196	Destination unreachable (Port unreachable)

Figure 27: DNS packet flow when Webterm accesses the fake Linkedin page



No.	Time	Source	Destination	Protocol	Length	Info
302	128.150459	20.0.0.1	20.0.0.2	TCP	74	40132 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=470436908 TSecr=0 WS=128
306	128.151396	20.0.0.2	20.0.0.1	TCP	74	80 → 40132 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1624992095 TSecr=470436908 WS=128
307	128.151662	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=470436915 TSecr=1624992095
308	128.151676	20.0.0.1	20.0.0.2	HTTP	382	GET / HTTP/1.1
309	128.151781	20.0.0.2	20.0.0.1	TCP	66	80 → 40132 [ACK] Seq=1 Ack=317 Win=64896 Len=0 TSval=1624992095 TSecr=470436915
310	128.152347	20.0.0.2	20.0.0.1	HTTP	803	HTTP/1.1 200 OK (text/html)
311	128.152373	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=317 Ack=738 Win=64128 Len=0 TSval=470436916 TSecr=1624992095
316	128.442122	20.0.0.1	20.0.0.2	HTTP	346	GET /gns3.png HTTP/1.1
317	128.446251	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [ACK] Seq=738 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
318	128.446271	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [ACK] Seq=2186 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
319	128.446289	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [ACK] Seq=3634 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
320	128.446295	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [ACK] Seq=5082 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
321	128.446300	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [PSH, ACK] Seq=6530 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
322	128.446309	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [ACK] Seq=7978 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
323	128.446314	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [ACK] Seq=9426 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
324	128.446319	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [ACK] Seq=10874 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
325	128.446327	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [ACK] Seq=12322 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
326	128.446332	20.0.0.2	20.0.0.1	TCP	1514	80 → 40132 [PSH, ACK] Seq=13770 Ack=597 Win=64640 Len=1448 TSval=1624992389 TSecr=470437205 [TCP segment of a reassembled PDU]
327	128.446625	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=597 Ack=2186 Win=64128 Len=0 TSval=470437210 TSecr=1624992389
328	128.446635	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=597 Ack=3634 Win=64128 Len=0 TSval=470437210 TSecr=1624992389
329	128.446640	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=597 Ack=5082 Win=64128 Len=0 TSval=470437210 TSecr=1624992389
330	128.446645	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=597 Ack=6530 Win=64128 Len=0 TSval=470437210 TSecr=1624992389
331	128.446649	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=597 Ack=7978 Win=64128 Len=0 TSval=470437210 TSecr=1624992389
332	128.446654	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=597 Ack=9426 Win=64128 Len=0 TSval=470437210 TSecr=1624992389
333	128.446658	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=597 Ack=10874 Win=64128 Len=0 TSval=470437210 TSecr=1624992389
334	128.446663	20.0.0.1	20.0.0.2	TCP	66	40132 → 80 [ACK] Seq=597 Ack=12322 Win=64128 Len=0 TSval=470437210 TSecr=1624992389

Capture 8: TCP flow when Webterm accesses the fake Linkedin page

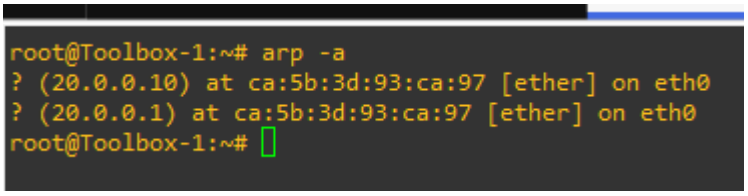


Figure 28: Toolbox ARP table post-attack

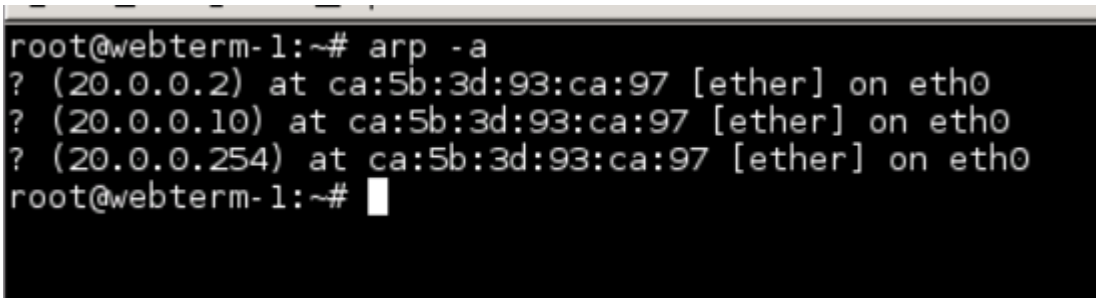


Figure 29: Webterm ARP table post-attack

Countermeasure

For this attack several countermeasures can be implemented such as:

- **Intrusion and Prevention Systems:** Monitoring network traffic for suspicious patterns or known attack signatures;
- **DNS network communication:** Using DNS over HTTPs (DoH) or DNS over TLS(DoT) helps protect DNS traffic from interception and tampering;
- **DNS configuration:** DNSSEC is a set of extensions to DNS that adds security features by using digital signatures to verify the authenticity and integrity of DNS responses.DNS response Rate Limiting is a technique that limits the rate at which a DNS resolver processes identical responses coming from a specific IP address. They can be both used to mitigate the risk of this attack.
- **Port Randomization:** Randomizing the source port of DNS queries and responses difficult the spoofing of DNS responses.
- **DNS caching:** Caching can help prevent repeated querying of the malicious DNS responses;
- **Network configuration:** Designing the network properly, segmenting the network and implementing proper network access controls make it difficult for attackers to gain unauthorized access and perform this attack.

Attack feasibility

The feasibility of this attack will depend on several factors such as:

- **Security measures:** security measures such as DNSSEC, port randomization can mitigate the risk of the attack;

- **Network configuration:** If the attacker can compromise the authentication or authorization mechanisms within the DNS infrastructure, they can manipulate DNS configuration;
- **Network Monitoring:** If the network is properly monitored and detection systems are taken in place is hard to the attacker go unnoticed and perform successfully the attack;

Overall if appropriate security measures are implemented, despite the fact DNS spoofing attacks can be feasible, the risk of the attack is reduced significantly.

RIP poisoning

Introduction

RIP stands for Routing Information Protocol and is designed to facilitate the exchange of routing information across the network, between routers. It is a protocol suitable for small networks because for large networks it has limitations such as slower convergence, scalability issues and reliance on hop count as the sole metric. Each router running RIP maintains a routing table that contains information about the network topology by listing the available networks and the associated metrics (hop count) to reach those networks.
The next figure illustrates the network topology used in this exercise:

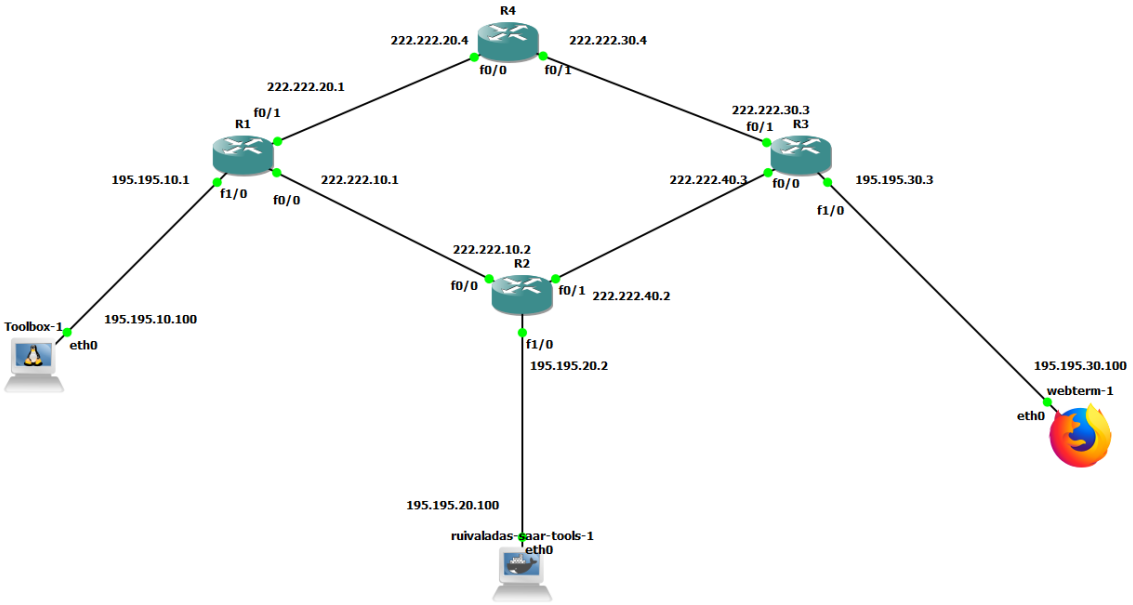


Figure 30: Network Topology

In order to set up the network we enabled for each router the RIP protocol to advertise the respective network prefixes into the RIP domain: **network 222.222.10.0**, **network 222.222.20.0** and **network 195.195.10.0**. For each interface we also configured the respective network address as we can observe in the network topology figure. For the

attacker configuration (ruivaladas-saar-tools-1) we configured the ip address, the default gateway as the R2 router and started the Nginx web server. The list of commands is represented in the next figure:

```
!R1
conf t
int f0/0
ip add 222.222.10.1 255.255.255.0
no shut
int f0/1
ip add 222.222.20.1 255.255.255.0
no shut
int f1/0
ip add 195.195.10.1 255.255.255.0
no shut
router rip
version 2
network 222.222.10.0
network 222.222.20.0
network 195.195.10.0
no auto-summary
end
!attacker
ifconfig eth0 195.195.20.100 netmask 255.255.255.0
ifconfig eth0:0 195.195.10.100 netmask 255.255.255.0
route add default gw 195.195.20.2
service nginx start
python3 rippoi.py 195.195.10.100 255.255.255.0 eth0 1

!toolbox
ifconfig eth0:0 195.195.10.100 netmask 255.255.255.0
route add default gw 195.195.10.1


!webserver
ifconfig eth0:0 195.195.30.100 netmask 255.255.255.0
route add default gw 195.195.30.3
```

Figure 31: Snippet of the Network configuration

Attack overview

RIP poisoning is an attack with the main goal to manipulate the routing information in a network running RIP. It is a form of network attack that aims to disrupt or manipulate the routing tables of routers participating in the RIP protocol. The attack advertises false routes by injecting false routing information into the network, typically with a lower hop count than the legitimate routes in order to mislead routers into believing that the attacker's routes are the shortest paths to the desired networks.
It's important to mention that RIP poisoning attacks are most effective in networks that solely rely on RIP for routing and do not have proper security measures in place.
In order to perform the attack we ran on the attacker configuration a script called rippoi.py designed in the programming language python version 3. The script takes four command-line arguments:

- **IP address:** IP for which routing information is being sent (target ip);
- **Subnet mask:** The subnet mask associated with the network address. It defines the range of IP addresses that belong to the network;
- **Interface**: Interface through which the packets will be sent;

- **Interval:** The interval between sending each RIP packet, specified as a floating-point number. It determines the rate at which the packets are sent;

The next capture and figure illustrates the RIP poisoning attack messages flow on R2 perspective after the attack was performed.

4	22.073863	195.195.20.100	224.0.0.9	RIPv2	66 Response
5	23.076023	195.195.20.100	224.0.0.9	RIPv2	66 Response
6	24.078106	195.195.20.100	224.0.0.9	RIPv2	66 Response
7	24.892234	195.195.20.2	224.0.0.9	RIPv2	166 Response
8	25.080065	195.195.20.100	224.0.0.9	RIPv2	66 Response
9	26.082044	195.195.20.100	224.0.0.9	RIPv2	66 Response
10	27.083718	195.195.20.100	224.0.0.9	RIPv2	66 Response
11	28.086384	195.195.20.100	224.0.0.9	RIPv2	66 Response
12	29.088014	195.195.20.100	224.0.0.9	RIPv2	66 Response
13	30.006842	c2:02:06:69:00:10	c2:02:06:69:00:10	LOOP	60 Reply
14	30.090954	195.195.20.100	224.0.0.9	RIPv2	66 Response
15	31.092706	195.195.20.100	224.0.0.9	RIPv2	66 Response
16	32.093789	195.195.20.100	224.0.0.9	RIPv2	66 Response
17	33.095811	195.195.20.100	224.0.0.9	RIPv2	66 Response
18	34.099081	195.195.20.100	224.0.0.9	RIPv2	66 Response
19	35.101267	195.195.20.100	224.0.0.9	RIPv2	66 Response
20	36.102879	195.195.20.100	224.0.0.9	RIPv2	66 Response

Capture 9: Rip poisoning attack capture

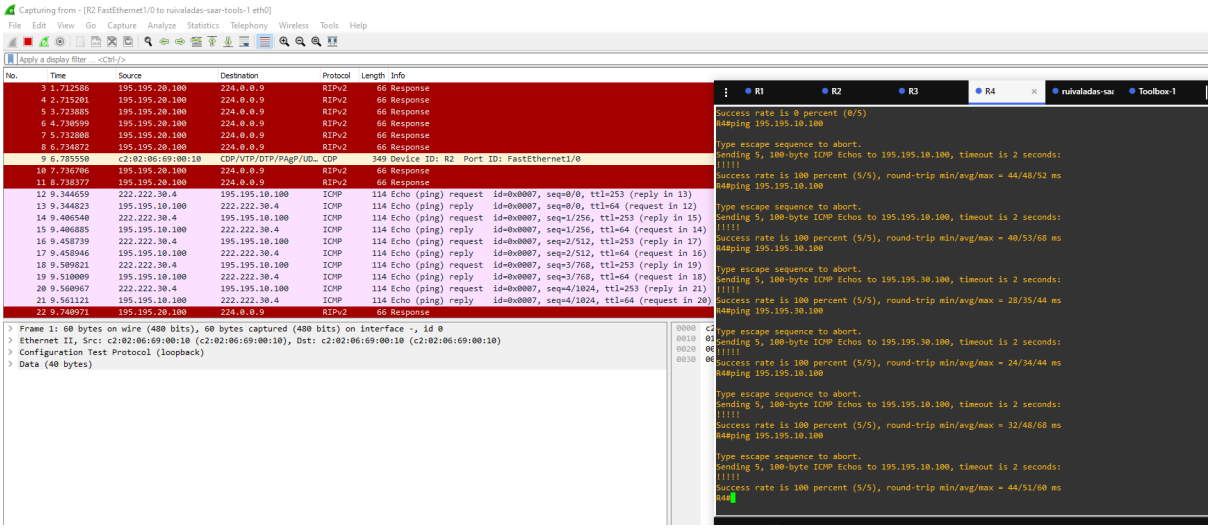


Figure 32: Rip poisoning attack demonstration

Countermeasure

When the auto-summary command is enabled in the RIP configuration, it automatically summarizes subnets of network boundaries, advertising summarized routes instead of individual subnets. In the context of RIP poisoning attacks, enabling this feature can help mitigate the impact of the attack by limiting the granularity of the routes that are advertised. Since RIP poisoning attacks involve injecting malicious routes with more specific network prefixes, enabling auto-summary will aggregate these routes into a single summary route. This reduces the effectiveness of the attack as the injected routes will not be propagated individually but instead summarizes into a large network range.

To prevent RIP poisoning also other countermeasures can be implemented such as:

- **Authentication:** Enable authentication mechanisms for RIP;
- **Network Monitoring and Security:** Implement proper security measures and tools to detect unusual or unexpected changes in routing tables can help to mitigate the risk of the attack;

- **Network segmentation:** Divide the network into separate segments such as VLANs or subnets can isolate RIP traffic to specific segments;
- **Filtering**: Implement inbound and outbound route filtering to control the routes;

Attack feasibility

The effectiveness and the feasibility of the the RIP poisoning attack and its impact on the network depends on several factors such as:

- **Relative Location:** If the legitimate web server and web browser are both located in the same network segment and the attacker's fake web server is in a different network segment, the effectiveness of the attack may be limited for example;
- **Auto-Summary feature:** Enabling the auto-summary in RIP can impact the effectiveness of the attack. With auto-summary enabled, the injected malicious routes with specific network prefixes may get aggregated into a single summary route and reduces the impact of the attack by obscuring the individual injected routes and making it harder for the attacker to redirect traffic to the fake web server;
- **Prefix Length used by the attacker:** For example, if the attacker injects routes with longer prefix lengths, the routers may prefer those routes over the legitimate routes, redirecting traffic to the fake web server;
- **Network design:** The feasibility of the attack can vary depending on the network topology. In a flat network with fewer routes, the attacker has a higher chance to be succeed;
- **Network security:** If countermeasures are applied as we mentioned in the previous section, the attack can be less feasible;
- **Attacker resources and skills:** The attacker needs to have directed access ,like an unauthorized physical or logical access to the network, or control over a network segment to inject the malicious RIP updates;

On a final note, the feasibility of RIP poisoning attack depends on factors such as those mentioned above. The existence of more secure routing protocols and network security improvements, these attacks are becoming less common.

TÉCNICO LISBOA

Protecting a campus network using a ZBPF

Network structure

The objective of this exercise is to configure a Zone-Based Policy Firewall (or ZBPF for short) in order to secure communications in a campus network. This network consists of 4 zones: 2 private zones (PR1 and PR2) that host a router and a PC each, a DMZ (demilitarized zone) that contains various services (such as DNS, Web and Email servers) and a representation of the outside of the network (referred to as OUT). We decided to use addresses from the range 10.0.0.0/24 for the DMZ, 10.1.1.0/24 for PR1, 10.2.2.0/24 for PR2 and 10.3.3.0/24 for OUT (see Table 1).

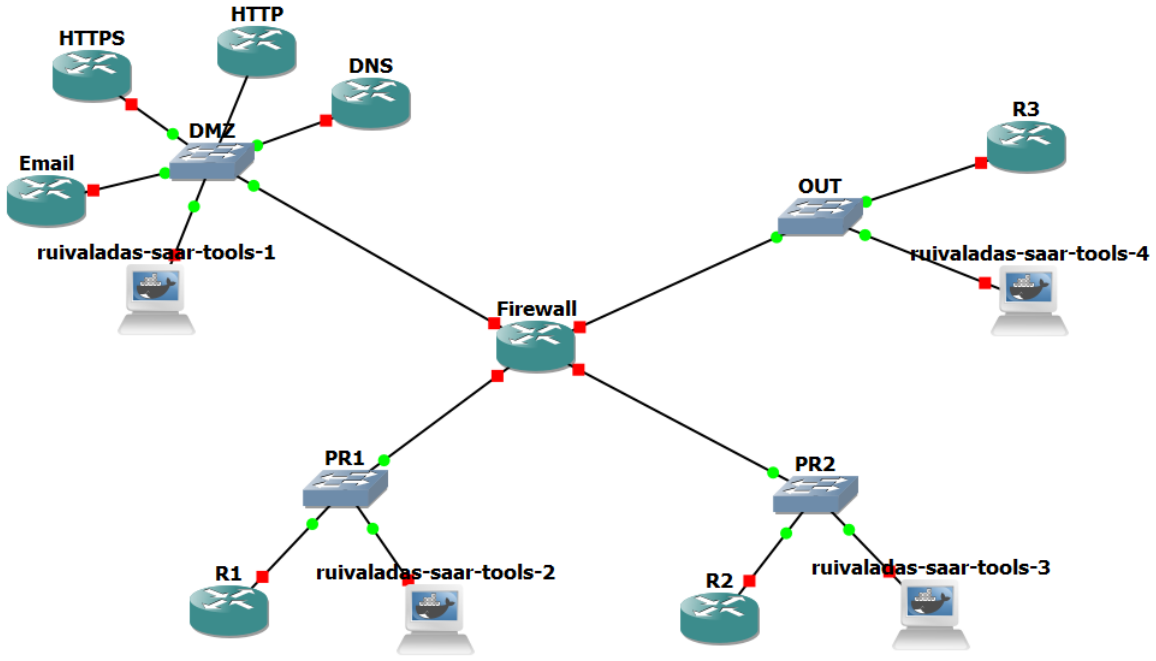


Figure 1: Network topology

Device Name	Interface	Zone	IP Address
Firewall	f0/0	DMZ	10.0.0.1/24
DNS	f0/0	DMZ	10.0.0.2/24
HTTP	f0/0	DMZ	10.0.0.3/24
HTTPS	f0/0	DMZ	10.0.0.4/24
Email	f0/0	DMZ	10.0.0.5/24
saar-tools-1	eth0	DMZ	10.0.0.6/24
Firewall	f1/0	PR1	10.1.1.1/24
R1	f0/0	PR1	10.1.1.2/24
saar-tools-2	eth0	PR1	10.1.1.3/24
Firewall	f1/1	PR2	10.2.2.1/24
R2	f0/0	PR2	10.2.2.2/24
saar-tools-3	eth0	PR2	10.2.2.3/24
Firewall	f0/1	OUT	10.3.3.1/24
R3	f0/0	OUT	10.3.3.2/24
saar-tools-4	eth0	OUT	10.3.3.3/24

Table 1: Network IP configuration and zone attribution for every interface

Firewall configuration

We devised a set of policies for the ZBPF that whitelist the necessary protocols for each pair of zones, source and destination, and prevent the other protocols' packets from traversing in that direction. Below you will find the complete set of commands for the firewall policy configuration:

---

conf t

zone security outside

exit

zone security dmz

exit

zone security pr1

```
exit
zone security pr2
exit
int f0/0
zone-member security dmz
exit
int f0/1
zone-member security outside
exit
int f1/0
zone-member security pr1
exit
int f1/1
zone-member security pr2
exit
class-map type inspect match-any dmz-out
match protocol icmp
match protocol dns
match protocol smtp
exit
class-map type inspect match-any pr-dmz
match protocol http
match protocol https
match protocol icmp
match protocol dns
match protocol pop3
match protocol imap
match protocol smtp
exit
class-map type inspect match-any out-dmz
match protocol http
match protocol https
match protocol icmp
match protocol dns
match protocol smtp
exit
class-map type inspect match-any pr-out
match protocol http
match protocol https
match protocol icmp
```

```
match protocol dns
exit
class-map type inspect match-any all-self
match protocol ssh
exit
policy-map type inspect DMZ-OUT
class dmz-out
inspect
exit
exit
policy-map type inspect PR-DMZ
class pr-dmz
inspect
exit
exit
policy-map type inspect OUT-DMZ
class out-dmz
inspect
exit
exit
policy-map type inspect PR-OUT
class pr-out
inspect
exit
exit
policy-map type inspect ALL-SELF
class all-self
pass
exit
exit
zone-pair security dmz-to-out source dmz destination outside
service-policy type inspect DMZ-OUT
exit
zone-pair security pr1-to-dmz source pr1 destination dmz
service-policy type inspect PR-DMZ
exit
zone-pair security pr2-to-dmz source pr2 destination dmz
service-policy type inspect PR-DMZ
exit
zone-pair security out-to-dmz source outside destination dmz
```

```
service-policy type inspect OUT-DMZ
exit
zone-pair security pr1-to-out source pr1 destination outside
service-policy type inspect PR-OUT
exit
zone-pair security pr2-to-out source pr2 destination outside
service-policy type inspect PR-OUT
exit
zone-pair security dmz-to-self source dmz destination self
service-policy type inspect ALL-SELF
exit
zone-pair security out-to-self source outside destination self
service-policy type inspect ALL-SELF
exit
zone-pair security pr1-to-self source pr1 destination self
service-policy type inspect ALL-SELF
exit
zone-pair security pr2-to-self source pr2 destination self
service-policy type inspect ALL-SELF
end
write
```

Policy testing

After the firewall's configuration, we performed extensive testing in order to look for any irregular behaviour or missing rules. We decided to start by testing the ICMP protocol, which is allowed from any zone to both DMZ and OUT. As such, PR1 and PR2 will never reply to any ping requests. To perform this test, we simply used the **ping** command in one device from each zone to another device from each of the other zones. The following figures (Figures 2 to 5) show that the ICMP packets are flowing as intended by our design.

No.	Time	Source	Destination	Protocol	Length	Info
5	16.033992	10.1.1.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response found!)
6	17.046552	10.1.1.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=1/256, ttl=255 (reply in 7)
7	17.088067	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) reply id=0x0000, seq=1/256, ttl=254 (request in 6)
8	17.098504	10.1.1.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=2/512, ttl=255 (reply in 9)
9	17.151200	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) reply id=0x0000, seq=2/512, ttl=254 (request in 8)
10	17.173576	10.1.1.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=3/768, ttl=255 (reply in 11)
11	17.289829	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) reply id=0x0000, seq=3/768, ttl=254 (request in 10)
12	17.339907	10.1.1.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=4/1024, ttl=255 (reply in 13)
13	17.470078	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) reply id=0x0000, seq=4/1024, ttl=254 (request in 12)
17	27.892494	10.1.1.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=0/0, ttl=255 (no response found!)
18	28.894202	10.1.1.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=1/256, ttl=255 (no response found!)
20	30.884990	10.1.1.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=2/512, ttl=255 (no response found!)
21	32.990103	10.1.1.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=3/768, ttl=255 (no response found!)
23	34.878013	10.1.1.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=4/1024, ttl=255 (no response found!)
28	43.330945	10.1.1.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=0/0, ttl=255 (no response found!)
29	44.096298	10.1.1.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=1/256, ttl=255 (reply in 30)
30	44.195779	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) reply id=0x0002, seq=1/256, ttl=254 (request in 29)
31	44.206493	10.1.1.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=2/512, ttl=255 (reply in 32)
32	44.350583	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) reply id=0x0002, seq=2/512, ttl=254 (request in 31)
33	44.365262	10.1.1.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=3/768, ttl=255 (reply in 34)
34	44.529413	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) reply id=0x0002, seq=3/768, ttl=254 (request in 33)
35	44.589493	10.1.1.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=4/1024, ttl=255 (reply in 36)
36	44.675658	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) reply id=0x0002, seq=4/1024, ttl=254 (request in 35)

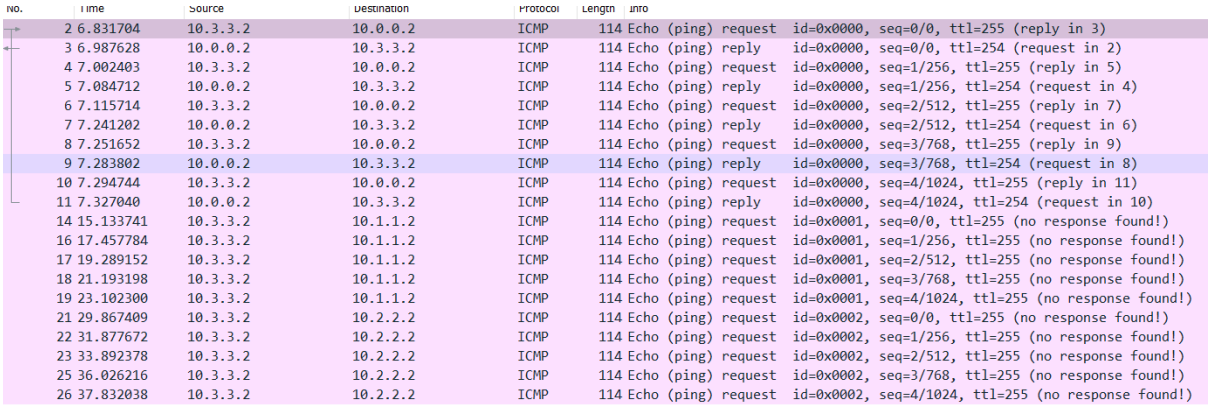
Figure 2: ICMP packet flow from PR1 (network 10.1.1.0/24) to the other zones

No.	Time	Source	Destination	Protocol	Length	Info
7	12.827388	10.2.2.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response found!)
9	13.831199	10.2.2.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=1/256, ttl=255 (reply in 10)
10	13.879626	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) reply id=0x0000, seq=1/256, ttl=254 (request in 9)
11	13.908359	10.2.2.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=2/512, ttl=255 (reply in 12)
12	14.044475	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) reply id=0x0000, seq=2/512, ttl=254 (request in 11)
13	14.055805	10.2.2.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=3/768, ttl=255 (reply in 14)
14	14.233634	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) reply id=0x0000, seq=3/768, ttl=254 (request in 13)
15	14.263755	10.2.2.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=4/1024, ttl=255 (reply in 16)
16	14.435391	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) reply id=0x0000, seq=4/1024, ttl=254 (request in 15)
20	21.118002	10.2.2.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=0/0, ttl=255 (no response found!)
21	22.130859	10.2.2.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=1/256, ttl=255 (no response found!)
22	24.086028	10.2.2.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=2/512, ttl=255 (no response found!)
23	26.041422	10.2.2.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=3/768, ttl=255 (no response found!)
24	28.105792	10.2.2.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=4/1024, ttl=255 (no response found!)
28	35.422439	10.2.2.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=0/0, ttl=255 (no response found!)
29	36.534114	10.2.2.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=1/256, ttl=255 (reply in 30)
30	36.619016	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) reply id=0x0002, seq=1/256, ttl=254 (request in 29)
31	36.675297	10.2.2.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=2/512, ttl=255 (reply in 32)
32	36.751283	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) reply id=0x0002, seq=2/512, ttl=254 (request in 31)
33	36.856501	10.2.2.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=3/768, ttl=255 (reply in 34)
34	36.991106	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) reply id=0x0002, seq=3/768, ttl=254 (request in 33)
35	37.041100	10.2.2.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=4/1024, ttl=255 (reply in 36)
36	37.107882	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) reply id=0x0002, seq=4/1024, ttl=254 (request in 35)

Figure 3: ICMP packet flow from PR2 (network 10.2.2.0/24) to the other zones

No.	Time	Source	Destination	Protocol	Length	Info
2	8.708169	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (no response found!)
5	10.065416	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0000, seq=1/256, ttl=255 (no response found!)
6	12.173460	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0000, seq=2/512, ttl=255 (no response found!)
7	14.665123	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0000, seq=3/768, ttl=255 (no response found!)
8	16.121123	10.0.0.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0000, seq=4/1024, ttl=255 (no response found!)
10	22.729616	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=0/0, ttl=255 (no response found!)
11	24.622177	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=1/256, ttl=255 (no response found!)
12	26.940523	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=2/512, ttl=255 (no response found!)
13	29.715247	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=3/768, ttl=255 (no response found!)
15	30.912249	10.0.0.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0001, seq=4/1024, ttl=255 (no response found!)
23	39.435659	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=0/0, ttl=255 (no response found!)
25	40.419948	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=1/256, ttl=255 (reply in 26)
26	40.452549	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0002, seq=1/256, ttl=254 (request in 25)
27	40.463172	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=2/512, ttl=255 (reply in 28)
28	40.532880	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0002, seq=2/512, ttl=254 (request in 27)
29	40.543719	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=3/768, ttl=255 (reply in 30)
30	40.702990	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0002, seq=3/768, ttl=254 (request in 29)
31	40.723531	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) request id=0x0002, seq=4/1024, ttl=255 (reply in 32)
32	40.796248	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) reply id=0x0002, seq=4/1024, ttl=254 (request in 31)

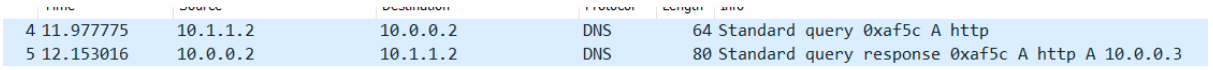
Figure 4: ICMP packet flow from DMZ (network 10.0.0.0/24) to the other zones



No.	Time	Source	Destination	Protocol	Length	Info
2	6.831704	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=0/0, ttl=255 (reply in 3)
3	6.987628	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) reply id=0x0000, seq=0/0, ttl=254 (request in 2)
4	7.002403	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=1/256, ttl=255 (reply in 5)
5	7.084712	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) reply id=0x0000, seq=1/256, ttl=254 (request in 4)
6	7.115714	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=2/512, ttl=255 (reply in 7)
7	7.241202	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) reply id=0x0000, seq=2/512, ttl=254 (request in 6)
8	7.251652	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=3/768, ttl=255 (reply in 9)
9	7.283802	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) reply id=0x0000, seq=3/768, ttl=254 (request in 8)
10	7.294744	10.3.3.2	10.0.0.2	ICMP	114	Echo (ping) request id=0x0000, seq=4/1024, ttl=255 (reply in 11)
11	7.327040	10.0.0.2	10.3.3.2	ICMP	114	Echo (ping) reply id=0x0000, seq=4/1024, ttl=254 (request in 10)
14	15.133741	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=0/0, ttl=255 (no response found!)
16	17.457784	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=1/256, ttl=255 (no response found!)
17	19.289152	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=2/512, ttl=255 (no response found!)
18	21.193198	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=3/768, ttl=255 (no response found!)
19	23.102300	10.3.3.2	10.1.1.2	ICMP	114	Echo (ping) request id=0x0001, seq=4/1024, ttl=255 (no response found!)
21	29.867409	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0002, seq=0/0, ttl=255 (no response found!)
22	31.877672	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0002, seq=1/256, ttl=255 (no response found!)
23	33.892378	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0002, seq=2/512, ttl=255 (no response found!)
25	36.026216	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0002, seq=3/768, ttl=255 (no response found!)
26	37.832038	10.3.3.2	10.2.2.2	ICMP	114	Echo (ping) request id=0x0002, seq=4/1024, ttl=255 (no response found!)

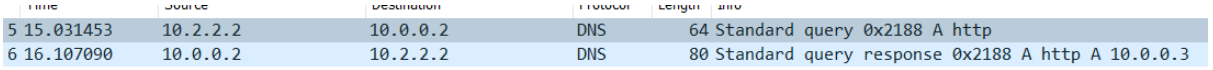
Figure 5: ICMP packet flow from OUT (network 10.3.3.0/24) to the other zones

We then started testing the DNS protocol, which is allowed everywhere except from one private zone to another. For this, we first enabled two DNS servers (one in the DMZ's DNS and the other in OUT's R3) and associated all of the DMZ server's names to their respective IP addresses. We also notified the devices we used to send the requests of the DNS servers' existence. After this initial configuration, we proceeded to use the **ping** command again, but this time we substituted the IP address of the device we wanted to ping for the device's name. The results are shown below (Figures 6 to 11):



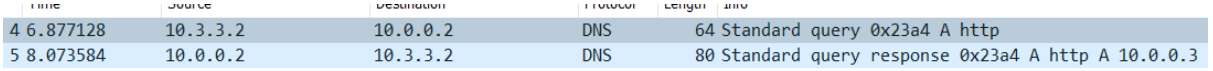
4	11.977775	10.1.1.2	10.0.0.2	DNS	64	Standard query 0xaf5c A http
5	12.153016	10.0.0.2	10.1.1.2	DNS	80	Standard query response 0xaf5c A http A 10.0.0.3

Figure 6: DNS lookup from PR1 (network 10.1.1.0/24) to DMZ (network 10.0.0.0/24)



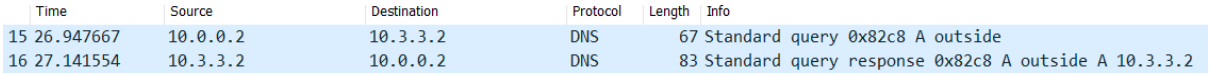
5	15.031453	10.2.2.2	10.0.0.2	DNS	64	Standard query 0x2188 A http
6	16.107090	10.0.0.2	10.2.2.2	DNS	80	Standard query response 0x2188 A http A 10.0.0.3

Figure 7: DNS lookup from PR2 (network 10.2.2.0/24) to DMZ (network 10.0.0.0/24)



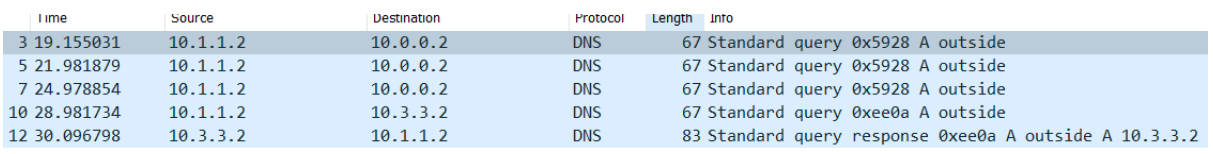
4	6.877128	10.3.3.2	10.0.0.2	DNS	64	Standard query 0x23a4 A http
5	8.073584	10.0.0.2	10.3.3.2	DNS	80	Standard query response 0x23a4 A http A 10.0.0.3

Figure 8: DNS lookup from OUT (network 10.3.3.0/24) to DMZ (network 10.0.0.0/24)



Time	Source	Destination	Protocol	Length	Info
15 26.947667	10.0.0.2	10.3.3.2	DNS	67	Standard query 0x82c8 A outside
16 27.141554	10.3.3.2	10.0.0.2	DNS	83	Standard query response 0x82c8 A outside A 10.3.3.2

Figure 9: DNS lookup from DMZ (network 10.0.0.0/24) to OUT (network 10.3.3.0/24)



	Time	Source	Destination	Protocol	Length	Info
3	19.155031	10.1.1.2	10.0.0.2	DNS	67	Standard query 0x5928 A outside
5	21.981879	10.1.1.2	10.0.0.2	DNS	67	Standard query 0x5928 A outside
7	24.978854	10.1.1.2	10.0.0.2	DNS	67	Standard query 0x5928 A outside
10	28.981734	10.1.1.2	10.3.3.2	DNS	67	Standard query 0xee0a A outside
12	30.096798	10.3.3.2	10.1.1.2	DNS	83	Standard query response 0xee0a A outside A 10.3.3.2

Figure 10: DNS lookup from PR1 (network 10.1.1.0/24) to OUT (network 10.3.3.0/24)

3	9.840780	10.2.2.2	10.0.0.2	DNS	67 Standard query 0xc8eb A outside
5	12.705214	10.2.2.2	10.0.0.2	DNS	67 Standard query 0xc8eb A outside
6	15.783847	10.2.2.2	10.0.0.2	DNS	67 Standard query 0xc8eb A outside
9	19.726283	10.2.2.2	10.3.3.2	DNS	67 Standard query 0x3129 A outside
11	20.776552	10.3.3.2	10.2.2.2	DNS	83 Standard query response 0x3129 A outside A 10.3.3.2

Figure 11: DNS lookup from PR2 (network 10.2.2.0/24) to OUT (network 10.3.3.0/24)

In order to test HTTP and HTTPS we enabled the respective services in the DMZ's HTTP and HTTPS devices. We also enabled both of these services on OUT's R3 in order to test requests to the outside of the network. Previously, we had configured the firewall to always allow both of these protocols except when the packets are travelling between private zones or if they are travelling from DMZ to OUT. To test if this was truly the case, we used the **telnet** command directed towards port 80 for HTTP and port 443 for HTTPS. The results are in the screenshots below (Figures 12 to 17):

	Time	Source	Destination	Protocol	Length	Info
3	8.169982	10.1.1.3	10.0.0.3	TCP	74	40242 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1443708122 TSecr=0 WS=128
4	8.274054	10.0.0.3	10.1.1.3	TCP	58	80 → 40242 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
5	8.274417	10.1.1.3	10.0.0.3	TCP	54	40242 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
7	22.645481	10.1.1.3	10.0.0.3	TCP	71	40242 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17 [TCP segment of a reassembled PDU]
8	22.859040	10.0.0.3	10.1.1.3	TCP	54	80 → 40242 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
9	22.859653	10.1.1.3	10.0.0.3	TCP	54	40242 → 80 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
10	22.941290	10.0.0.3	10.1.1.3	TCP	54	80 → 40242 [ACK] Seq=2 Ack=19 Win=4111 Len=0
11	31.359550	10.1.1.3	10.0.0.4	TCP	74	46804 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2690647760 TSecr=0 WS=128
12	31.424551	10.0.0.4	10.1.1.3	TCP	58	443 → 46804 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
13	31.424986	10.1.1.3	10.0.0.4	TCP	54	46804 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
14	41.830930	10.1.1.3	10.0.0.4	TCP	71	46804 → 443 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17
15	41.926597	10.0.0.4	10.1.1.3	TCP	54	443 → 46804 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
16	41.927371	10.1.1.3	10.0.0.4	TCP	54	46804 → 443 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
17	42.043789	10.0.0.4	10.1.1.3	TCP	54	443 → 46804 [ACK] Seq=2 Ack=19 Win=4111 Len=0

Figure 12: HTTP/HTTPS packet flow from PR1 (network 10.1.1.0/24) to DMZ (network 10.0.0.0/24)

1	0.000000	10.2.2.3	10.0.0.3	TCP	74	40516 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=186891130 TSecr=0 WS=128
2	0.074215	10.0.0.3	10.2.2.3	TCP	58	80 → 40516 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
3	0.074617	10.2.2.3	10.0.0.3	TCP	54	40516 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
7	8.468610	10.2.2.3	10.0.0.3	TCP	71	40516 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17 [TCP segment of a reassembled PDU]
8	8.552973	10.0.0.3	10.2.2.3	TCP	54	80 → 40516 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
9	8.553778	10.2.2.3	10.0.0.3	TCP	54	40516 → 80 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
10	8.647291	10.0.0.3	10.2.2.3	TCP	54	80 → 40516 [ACK] Seq=2 Ack=19 Win=4111 Len=0
11	12.223377	10.2.2.3	10.0.0.4	TCP	74	34268 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1716603548 TSecr=0 WS=128
12	12.459653	10.0.0.4	10.2.2.3	TCP	58	443 → 34268 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
13	12.460038	10.2.2.3	10.0.0.4	TCP	54	34268 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
14	18.026016	10.2.2.3	10.0.0.4	TCP	71	34268 → 443 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17
15	18.067776	10.0.0.4	10.2.2.3	TCP	54	443 → 34268 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
16	18.068362	10.2.2.3	10.0.0.4	TCP	54	34268 → 443 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
17	18.133765	10.0.0.4	10.2.2.3	TCP	54	443 → 34268 [ACK] Seq=2 Ack=19 Win=4111 Len=0

Figure 13: HTTP/HTTPS packet flow from PR2 (network 10.2.2.0/24) to DMZ (network 10.0.0.0/24)

1	0.000000	10.3.3.3	10.0.0.3	TCP	74	53898 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=208658807 TSecr=0 WS=128
2	0.203289	10.0.0.3	10.3.3.3	TCP	58	80 → 53898 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
3	0.203739	10.3.3.3	10.0.0.3	TCP	54	53898 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	7.293600	10.3.3.3	10.0.0.3	TCP	71	53898 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17 [TCP segment of a reassembled PDU]
5	7.347613	10.0.0.3	10.3.3.3	TCP	54	80 → 53898 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
6	7.348829	10.3.3.3	10.0.0.3	TCP	54	53898 → 80 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
7	7.487823	10.0.0.3	10.3.3.3	TCP	54	80 → 53898 [ACK] Seq=2 Ack=19 Win=4111 Len=0
9	11.152006	10.3.3.3	10.0.0.4	TCP	74	56190 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2093925420 TSecr=0 WS=128
10	11.201979	10.0.0.4	10.3.3.3	TCP	58	443 → 56190 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
11	11.203240	10.3.3.3	10.0.0.4	TCP	54	56190 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
13	18.484112	10.3.3.3	10.0.0.4	TCP	71	56190 → 443 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17
14	18.643537	10.0.0.4	10.3.3.3	TCP	54	443 → 56190 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
15	18.644055	10.3.3.3	10.0.0.4	TCP	54	56190 → 443 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
16	18.685827	10.0.0.4	10.3.3.3	TCP	54	443 → 56190 [ACK] Seq=2 Ack=19 Win=4111 Len=0

Figure 14: HTTP/HTTPS packet flow from OUT (network 10.3.3.0/24) to DMZ (network 10.0.0.0/24)

5	15.607967	10.0.0.6	10.3.3.2	TCP	74 43628 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=703081133 TSecr=0 WS=128
6	16.632530	10.0.0.6	10.3.3.2	TCP	74 [TCP Retransmission] [TCP Port numbers reused] 43628 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=14
7	18.652779	10.0.0.6	10.3.3.2	TCP	74 [TCP Retransmission] [TCP Port numbers reused] 43628 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=14
11	22.717733	10.0.0.6	10.3.3.2	TCP	74 [TCP Retransmission] [TCP Port numbers reused] 43628 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=14
12	27.726138	10.0.0.6	10.3.3.2	TCP	74 57674 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=703093251 TSecr=0 WS=128
13	28.728038	10.0.0.6	10.3.3.2	TCP	74 [TCP Retransmission] [TCP Port numbers reused] 57674 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1
15	30.753204	10.0.0.6	10.3.3.2	TCP	74 [TCP Retransmission] [TCP Port numbers reused] 57674 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1
16	35.005892	10.0.0.6	10.3.3.2	TCP	74 [TCP Retransmission] [TCP Port numbers reused] 57674 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1

Figure 15: HTTP/HTTPS packet flow from DMZ (network 10.0.0.0/24) to OUT (network 10.3.3.0/24)

2	12.880914	10.1.1.3	10.3.3.2	TCP	74 48642 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1288395054 TSecr=0 WS=128
3	13.013831	10.3.3.2	10.1.1.3	TCP	58 80 → 48642 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
4	13.014375	10.1.1.3	10.3.3.2	TCP	54 48642 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
5	20.560926	10.1.1.3	10.3.3.2	TCP	71 48642 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17 [TCP segment of a reassembled PDU]
6	20.749767	10.3.3.2	10.1.1.3	TCP	54 80 → 48642 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
7	20.750518	10.1.1.3	10.3.3.2	TCP	54 48642 → 80 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
8	20.898306	10.3.3.2	10.1.1.3	TCP	54 80 → 48642 [ACK] Seq=2 Ack=19 Win=4111 Len=0
9	23.860306	10.1.1.3	10.3.3.2	TCP	74 41828 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1288406033 TSecr=0 WS=128
10	23.891115	10.3.3.2	10.1.1.3	TCP	58 443 → 41828 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
11	23.891411	10.1.1.3	10.3.3.2	TCP	54 41828 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
13	28.861319	10.1.1.3	10.3.3.2	TCP	71 41828 → 443 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17
14	28.936461	10.3.3.2	10.1.1.3	TCP	54 443 → 41828 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
15	28.937291	10.1.1.3	10.3.3.2	TCP	54 41828 → 443 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
16	28.989637	10.3.3.2	10.1.1.3	TCP	54 443 → 41828 [ACK] Seq=2 Ack=19 Win=4111 Len=0

Figure 16: HTTP/HTTPS packet flow from PR1 (network 10.1.1.0/24) to OUT (network 10.3.3.0/24)

2	15.274924	10.2.2.3	10.3.3.2	TCP	74 53892 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1071672837 TSecr=0 WS=128
3	15.416369	10.3.3.2	10.2.2.3	TCP	58 80 → 53892 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
4	15.416688	10.2.2.3	10.3.3.2	TCP	54 53892 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
5	21.461983	10.2.2.3	10.3.3.2	TCP	71 53892 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=17 [TCP segment of a reassembled PDU]
6	21.513598	10.3.3.2	10.2.2.3	TCP	54 80 → 53892 [FIN, PSH, ACK] Seq=1 Ack=18 Win=4111 Len=0
7	21.514283	10.2.2.3	10.3.3.2	TCP	54 53892 → 80 [FIN, ACK] Seq=18 Ack=2 Win=64239 Len=0
8	21.601777	10.3.3.2	10.2.2.3	TCP	54 80 → 53892 [ACK] Seq=2 Ack=19 Win=4111 Len=0
9	25.661252	10.2.2.3	10.3.3.2	TCP	74 54872 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1071683223 TSecr=0 WS=128
10	25.703864	10.3.3.2	10.2.2.3	TCP	58 443 → 54872 [SYN, ACK] Seq=0 Ack=1 Win=4128 Len=0 MSS=536
11	25.704338	10.2.2.3	10.3.3.2	TCP	54 54872 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
12	34.211593	10.2.2.3	10.3.3.2	TCP	72 54872 → 443 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=18
13	34.333971	10.3.3.2	10.2.2.3	TCP	54 443 → 54872 [FIN, PSH, ACK] Seq=1 Ack=19 Win=4110 Len=0
14	34.334612	10.2.2.3	10.3.3.2	TCP	54 54872 → 443 [FIN, ACK] Seq=19 Ack=2 Win=64239 Len=0
15	34.411857	10.3.3.2	10.2.2.3	TCP	54 443 → 54872 [ACK] Seq=2 Ack=20 Win=4110 Len=0

Figure 17: HTTP/HTTPS packet flow from PR2 (network 10.2.2.0/24) to OUT (network 10.3.3.0/24)

Finally, we tested for SSH connectivity from all areas to the firewall. For this we used the saar-tools appliances as SSH clients and configured user authentication in the firewall itself. We then used **telnet** on port 22 to connect to the firewall. Some examples of SSH packet exchanges can be found below (Figures 18 and 19):

8	9.157616	10.0.0.1	10.0.0.6	SSH	74 Server: Protocol (SSH-1.99-Cisco-1.25)
12	18.565248	10.0.0.6	10.0.0.1	SSH	73 Client: Encrypted packet (len=19)

Figure 18: SSH from DMZ (network 10.0.0.0/24) to the firewall

5	2.291688	10.1.1.1	10.1.1.3	SSH	74 Server: Protocol (SSH-1.99-Cisco-1.25)
10	22.526704	10.1.1.3	10.1.1.1	SSH	73 Client: Encrypted packet (len=19)

Figure 19: SSH from PR1 (network 10.1.1.0/24) to the firewall

After the testing was finished we concluded that our network was configured according to the parameters specified in the laboratory guide, as only the required protocols can transmit packets in each of the source-destination zone pairs.

Defense against DoS attacks

Network structure

In this exercise we used a previous network with the objective of defending against two types of attacks: ICMP flood and TCP SYN flood.

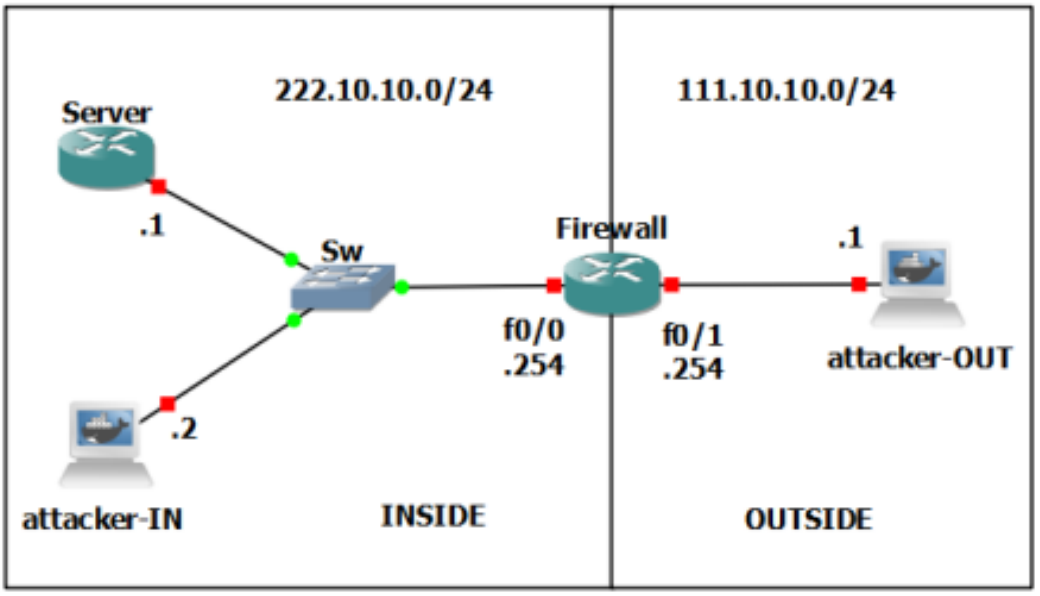


Figure 20: Network topology

Attack overview

Both attacks consist of the same thing: sending a big amount of requests to the server, which it will acknowledge and proceed to wait for the attacker's response. This response will never occur, making the server unable to do its service for valid requests (Denial of Service).

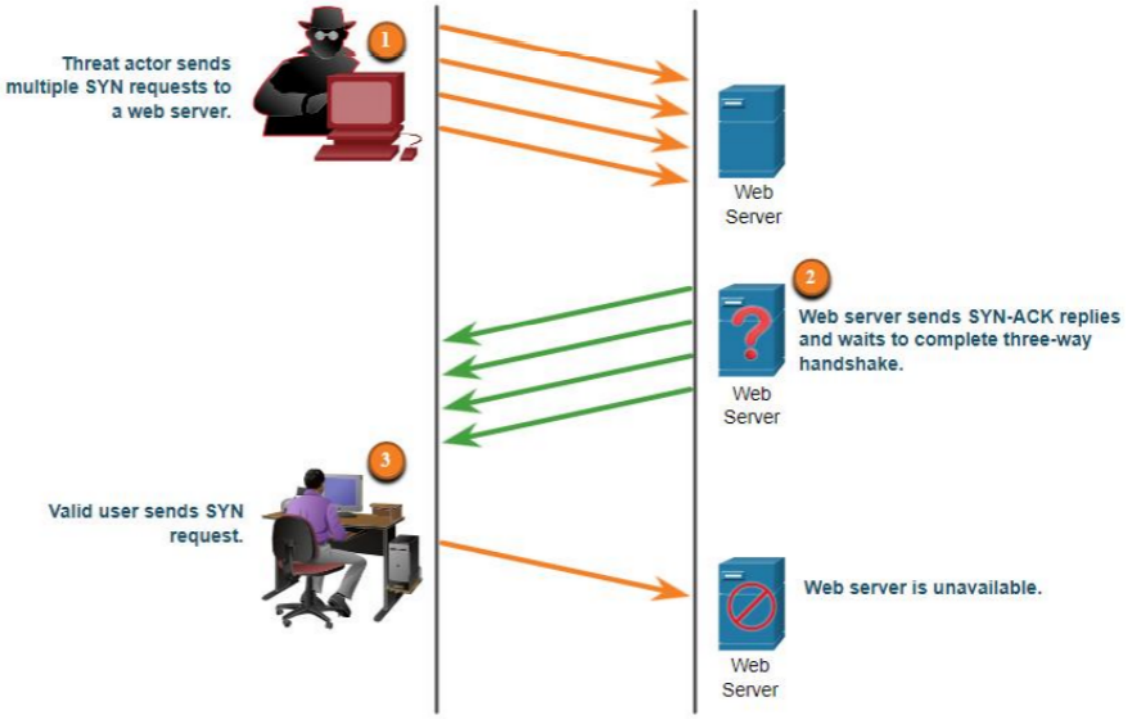


Figure 21: example of a TCP SYN flood attack

First we did an **hping** command that sends a big amount of TCP SYN to the server, but with the ZBPF firewall configuration that we have they will not be able to pass through, like we can see in Figures 22 and 23. Our server (R2) didn't receive any TCP packets (Figure 22), although they are sent from the attacker and reach the firewall (Figure 23).

```
R2#show tcp statistics
Rcvd: 0 Total, 0 no port
      0 checksum error, 0 bad offset, 0 too short
      0 packets (0 bytes) in sequence
      0 dup packets (0 bytes)
      0 partially dup packets (0 bytes)
      0 out-of-order packets (0 bytes)
      0 packets (0 bytes) with data after window
      0 packets after close
      0 window probe packets, 0 window update packets
      0 dup ack packets, 0 ack packets with unsend data
      0 ack packets (0 bytes)
Sent: 0 Total, 0 urgent packets
      0 control packets (including 0 retransmitted)
      0 data packets (0 bytes)
      0 data packets (0 bytes) retransmitted
      0 data packets (0 bytes) fastretransmitted
      0 ack only packets (0 delayed)
      0 window probe packets, 0 window update packets
0 Connections initiated, 0 connections accepted, 0 connections established
0 Connections closed (including 0 dropped, 0 embryonic dropped)
0 Total rxmt timeout, 0 connections dropped in rxmt timeout
0 Keepalive timeout, 0 keepalive probe, 0 Connections dropped in keepalive
R2#
```

Figure 22: TCP statistics from OUT to server when using hping

81	13.602770	111.10.10.1	222.10.10.1	TCP	54 2336 → 80 [SYN] Seq=0 Win=512 Len=0
82	13.619035	111.10.10.1	222.10.10.1	TCP	54 2337 → 80 [SYN] Seq=0 Win=512 Len=0
83	13.633099	111.10.10.1	222.10.10.1	TCP	54 2338 → 80 [SYN] Seq=0 Win=512 Len=0
84	13.649596	111.10.10.1	222.10.10.1	TCP	54 2339 → 80 [SYN] Seq=0 Win=512 Len=0
85	13.663524	111.10.10.1	222.10.10.1	TCP	54 2340 → 80 [SYN] Seq=0 Win=512 Len=0
86	13.678853	111.10.10.1	222.10.10.1	TCP	54 2341 → 80 [SYN] Seq=0 Win=512 Len=0
87	13.693319	111.10.10.1	222.10.10.1	TCP	54 2342 → 80 [SYN] Seq=0 Win=512 Len=0
88	13.708602	111.10.10.1	222.10.10.1	TCP	54 2343 → 80 [SYN] Seq=0 Win=512 Len=0
89	13.723502	111.10.10.1	222.10.10.1	TCP	54 2344 → 80 [SYN] Seq=0 Win=512 Len=0
90	13.740766	111.10.10.1	222.10.10.1	TCP	54 2345 → 80 [SYN] Seq=0 Win=512 Len=0
91	13.752943	111.10.10.1	222.10.10.1	TCP	54 2346 → 80 [SYN] Seq=0 Win=512 Len=0
92	13.767810	111.10.10.1	222.10.10.1	TCP	54 2347 → 80 [SYN] Seq=0 Win=512 Len=0
93	13.782639	111.10.10.1	222.10.10.1	TCP	54 2348 → 80 [SYN] Seq=0 Win=512 Len=0
94	13.792914	111.10.10.1	222.10.10.1	TCP	54 2349 → 80 [SYN] Seq=0 Win=512 Len=0
95	13.812653	111.10.10.1	222.10.10.1	TCP	54 2350 → 80 [SYN] Seq=0 Win=512 Len=0
96	13.827724	111.10.10.1	222.10.10.1	TCP	54 2351 → 80 [SYN] Seq=0 Win=512 Len=0
97	13.842355	111.10.10.1	222.10.10.1	TCP	54 2352 → 80 [SYN] Seq=0 Win=512 Len=0
98	13.853936	111.10.10.1	222.10.10.1	TCP	54 2353 → 80 [SYN] Seq=0 Win=512 Len=0
99	13.872173	111.10.10.1	222.10.10.1	TCP	54 2354 → 80 [SYN] Seq=0 Win=512 Len=0
100	13.887062	111.10.10.1	222.10.10.1	TCP	54 2355 → 80 [SYN] Seq=0 Win=512 Len=0
101	13.901934	111.10.10.1	222.10.10.1	TCP	54 2356 → 80 [SYN] Seq=0 Win=512 Len=0
102	13.918196	111.10.10.1	222.10.10.1	TCP	54 2357 → 80 [SYN] Seq=0 Win=512 Len=0

Figure 23: hping from OUT to server

Afterwards, we did a similar attack but now we added the **–random-source** flag to the command. What this does is that in every TCP SYN that is sent the source IP is randomized. With this, the ZBPF firewall is not able to track the source of the attack and just lets most of the packets pass through, as seen in Figures 24 and 25.

```
R2#show tcp statistics
Rcvd: 628 Total, 314 no port
      0 checksum error, 0 bad offset, 0 too short
      0 packets (0 bytes) in sequence
      0 dup packets (0 bytes)
      0 partially dup packets (0 bytes)
      0 out-of-order packets (0 bytes)
      0 packets (0 bytes) with data after window
      0 packets after close
      0 window probe packets, 0 window update packets
      0 dup ack packets, 0 ack packets with unsend data
      0 ack packets (0 bytes)
Sent: 314 Total, 0 urgent packets
      314 control packets (including 0 retransmitted)
      0 data packets (0 bytes)
      0 data packets (0 bytes) retransmitted
      0 data packets (0 bytes) fastretransmitted
      0 ack only packets (0 delayed)
      0 window probe packets, 0 window update packets
0 Connections initiated, 0 connections accepted, 0 connections established
628 Connections closed (including 0 dropped, 314 embryonic dropped)
0 Total rxmt timeout, 0 connections dropped in rxmt timeout
0 Keepalive timeout, 0 keepalive probe, 0 Connections dropped in keepalive
R2#
```

Figure 24: TCP statistics from OUT to server when using hping with random source

4	15.202329	96.176.53.194	222.10.10.1	TCP	54 1576 → 80 [SYN] Seq=0 Win=512 Len=0
5	15.217315	166.251.168.242	222.10.10.1	TCP	54 1577 → 80 [SYN] Seq=0 Win=512 Len=0
6	15.246069	73.128.52.190	222.10.10.1	TCP	54 1578 → 80 [SYN] Seq=0 Win=512 Len=0
7	15.246097	119.40.129.52	222.10.10.1	TCP	54 1579 → 80 [SYN] Seq=0 Win=512 Len=0
8	15.275028	37.213.38.114	222.10.10.1	TCP	54 1580 → 80 [SYN] Seq=0 Win=512 Len=0
9	15.275062	104.251.104.40	222.10.10.1	TCP	54 1581 → 80 [SYN] Seq=0 Win=512 Len=0
10	15.304828	244.113.172.107	222.10.10.1	TCP	54 1582 → 80 [SYN] Seq=0 Win=512 Len=0
11	15.304858	21.131.181.235	222.10.10.1	TCP	54 1583 → 80 [SYN] Seq=0 Win=512 Len=0
12	15.315300	7.194.15.73	222.10.10.1	TCP	54 1584 → 80 [SYN] Seq=0 Win=512 Len=0
13	15.315340	93.32.131.251	222.10.10.1	TCP	54 1585 → 80 [SYN] Seq=0 Win=512 Len=0
14	15.335649	31.156.40.248	222.10.10.1	TCP	54 1586 → 80 [SYN] Seq=0 Win=512 Len=0
15	15.335676	7.176.201.196	222.10.10.1	TCP	54 1587 → 80 [SYN] Seq=0 Win=512 Len=0
16	15.376474	31.116.228.233	222.10.10.1	TCP	54 1588 → 80 [SYN] Seq=0 Win=512 Len=0
17	15.376507	213.114.151.125	222.10.10.1	TCP	54 1589 → 80 [SYN] Seq=0 Win=512 Len=0
18	15.391361	144.138.8.31	222.10.10.1	TCP	54 1590 → 80 [SYN] Seq=0 Win=512 Len=0
19	15.405706	58.171.193.228	222.10.10.1	TCP	54 1591 → 80 [SYN] Seq=0 Win=512 Len=0
20	15.420322	120.255.164.181	222.10.10.1	TCP	54 1592 → 80 [SYN] Seq=0 Win=512 Len=0
21	15.434544	24.125.197.120	222.10.10.1	TCP	54 1593 → 80 [SYN] Seq=0 Win=512 Len=0
22	15.434568	95.49.7.242	222.10.10.1	TCP	54 1594 → 80 [SYN] Seq=0 Win=512 Len=0
23	15.466354	194.156.144.125	222.10.10.1	TCP	54 1595 → 80 [SYN] Seq=0 Win=512 Len=0
24	15.466381	76.213.49.210	222.10.10.1	TCP	54 1596 → 80 [SYN] Seq=0 Win=512 Len=0
25	15.476494	31.128.196.49	222.10.10.1	TCP	54 1597 → 80 [SYN] Seq=0 Win=512 Len=0
26	15.486637	250.8.147.60	222.10.10.1	TCP	54 1598 → 80 [SYN] Seq=0 Win=512 Len=0
27	15.496774	205.93.190.21	222.10.10.1	TCP	54 1599 → 80 [SYN] Seq=0 Win=512 Len=0

Figure 25: hping with random source from OUT to server

We added a first countermeasure to try to stop the DoS attacks by creating a rate limiting policy. With this the firewall should be dropping more packets than before (Figures 27 and 29). These tests were performed for both the ICMP and HTTP protocols.

```
130 46.786631   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=31232/122, ttl=63 (no response found!)
131 46.786637   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=31488/123, ttl=63 (no response found!)
132 46.786641   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=31744/124, ttl=63 (no response found!)
133 46.786646   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=32000/125, ttl=63 (no response found!)
134 46.786650   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=32256/126, ttl=63 (no response found!)
135 46.786654   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=49523/29633, ttl=63 (no response found!)
136 46.816839   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=49779/29634, ttl=63 (no response found!)
137 46.816866   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=50035/29635, ttl=63 (no response found!)
138 46.816872   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=50291/29636, ttl=63 (no response found!)
139 46.816876   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=50547/29637, ttl=63 (no response found!)
140 46.816880   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=50803/29638, ttl=63 (no response found!)
141 46.816884   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=51059/29639, ttl=63 (no response found!)
142 46.816889   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=51315/29640, ttl=63 (no response found!)
143 46.816893   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=51571/29641, ttl=63 (no response found!)
144 46.816897   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=51827/29642, ttl=63 (no response found!)
145 46.816902   111.10.10.1     222.10.10.1     ICMP    42 Echo (ping) request  id=0x5f00, seq=52083/29643, ttl=63 (no response found!)
```

Figure 26: hping ICMP flood

```
policy exists on zp OUT-TO-IN
 Zone-pair: OUT-TO-IN

  Service-policy inspect : DOS_POLICYMAP

    Class-map: WWW_CLASSMAP (match-all)
      Match: access-group name TO_IN_HOST
      Match: protocol http

   Inspect
        Session creations since subsystem startup or last reset 0
        Current session counts (estab/half-open/terminating) [0:0:0]
        Maxever session counts (estab/half-open/terminating) [0:0:0]
        Last session created 00:10:14
        Last statistic reset 00:03:11
        Last session creation rate 0
        Maxever session creation rate 0
        Last half-open session total 0
        TCP reassembly statistics
        received 0 packets out-of-order; dropped 0
        peak memory usage 0 KB; current usage: 0 KB
        peak queue length 0

       Police
        rate 128000 bps,8000 limit
        conformed 0 packets, 0 bytes; actions: transmit
        exceeded 0 packets, 0 bytes; actions: drop
        conformed 0 bps, exceed 0 bps

    Class-map: ICMP_CLASSMAP (match-all)
      Match: access-group name TO_IN_HOST
      Match: protocol icmp

   Inspect
        Packet inspection statistics [process switch:fast switch]
        icmp packets: [0:1155]

        Session creations since subsystem startup or last reset 1
        Current session counts (estab/half-open/terminating) [0:0:0]
        Maxever session counts (estab/half-open/terminating) [0:1:0]
        Last session created 00:00:57
        Last statistic reset 00:03:11
        Last session creation rate 0
        Maxever session creation rate 1
        Last half-open session total 0
        TCP reassembly statistics
        received 0 packets out-of-order; dropped 0
        peak memory usage 0 KB; current usage: 0 KB
        peak queue length 0

       Police
        rate 128000 bps,8000 limit
        conformed 1155 packets, 48510 bytes; actions: transmit
        exceeded 5426 packets, 227892 bytes; actions: drop
        conformed 0 bps, exceed 0 bps

    Class-map: class-default (match-any)
      Match: any
      Drop
        0 packets, 0 bytes
```

Figure 27: hping ICMP flood firewall log

	Time	Source	Destination	Protocol	Length	Info
644	14.809865	27.176.7.194	222.10.10.1	TCP	60	19220 → 80 [SYN] Seq=0 Win=512 Len=0
645	14.809870	15.239.174.4	222.10.10.1	TCP	60	19221 → 80 [SYN] Seq=0 Win=512 Len=0
646	14.809874	78.231.143.38	222.10.10.1	TCP	60	19222 → 80 [SYN] Seq=0 Win=512 Len=0
647	14.809879	135.251.10.18	222.10.10.1	TCP	60	19223 → 80 [SYN] Seq=0 Win=512 Len=0
648	14.809884	236.201.145.10	222.10.10.1	TCP	60	19224 → 80 [SYN] Seq=0 Win=512 Len=0
649	14.809888	237.142.193.123	222.10.10.1	TCP	60	19225 → 80 [SYN] Seq=0 Win=512 Len=0
650	14.809892	47.24.217.28	222.10.10.1	TCP	60	19226 → 80 [SYN] Seq=0 Win=512 Len=0
651	14.809897	142.80.20.109	222.10.10.1	TCP	60	19227 → 80 [SYN] Seq=0 Win=512 Len=0
652	14.809902	120.130.219.137	222.10.10.1	TCP	60	19228 → 80 [SYN] Seq=0 Win=512 Len=0
653	14.809906	150.85.45.233	222.10.10.1	TCP	60	19229 → 80 [SYN] Seq=0 Win=512 Len=0
654	14.809910	10.217.132.155	222.10.10.1	TCP	60	19230 → 80 [SYN] Seq=0 Win=512 Len=0
655	14.825188	229.70.251.179	222.10.10.1	TCP	60	19231 → 80 [SYN] Seq=0 Win=512 Len=0
656	14.825217	214.146.107.185	222.10.10.1	TCP	60	19232 → 80 [SYN] Seq=0 Win=512 Len=0
657	14.825222	193.8.118.203	222.10.10.1	TCP	60	19233 → 80 [SYN] Seq=0 Win=512 Len=0
658	14.825227	70.132.163.132	222.10.10.1	TCP	60	19234 → 80 [SYN] Seq=0 Win=512 Len=0
659	14.825231	231.7.232.236	222.10.10.1	TCP	60	19235 → 80 [SYN] Seq=0 Win=512 Len=0
660	14.825236	190.160.174.147	222.10.10.1	TCP	60	19236 → 80 [SYN] Seq=0 Win=512 Len=0
661	14.825240	241.171.72.57	222.10.10.1	TCP	60	19237 → 80 [SYN] Seq=0 Win=512 Len=0
662	14.825245	225.212.160.52	222.10.10.1	TCP	60	19238 → 80 [SYN] Seq=0 Win=512 Len=0
663	14.825249	214.67.208.78	222.10.10.1	TCP	60	19239 → 80 [SYN] Seq=0 Win=512 Len=0
664	14.825254	134.57.68.190	222.10.10.1	TCP	60	19240 → 80 [SYN] Seq=0 Win=512 Len=0
665	14.825258	137.10.68.86	222.10.10.1	TCP	60	19241 → 80 [SYN] Seq=0 Win=512 Len=0
666	14.825263	65.40.171.135	222.10.10.1	TCP	60	19242 → 80 [SYN] Seq=0 Win=512 Len=0
667	14.825267	124.156.191.84	222.10.10.1	TCP	60	19243 → 80 [SYN] Seq=0 Win=512 Len=0
668	14.825271	155.225.241.22	222.10.10.1	TCP	60	19244 → 80 [SYN] Seq=0 Win=512 Len=0

Figure 28: hping tcp syn flood

```
Zone-pair: OUT-TO-IN

 Service-policy inspect : DOS_POLICYMAP

   Class-map: WWW_CLASSMAP (match-all)
     Match: access-group name TO_IN_HOST
     Match: protocol http

  Inspect
       Packet inspection statistics [process switch:fast switch]
       tcp packets: [74:603]

       Session creations since subsystem startup or last reset 877
       Current session counts (estab/half-open/terminating) [0:0:0]
       Maxever session counts (estab/half-open/terminating) [0:877:0]
       Last session created 00:00:54
       Last statistic reset 00:01:40
       Last session creation rate 0
       Maxever session creation rate 877
       Last half-open session total 0
       TCP reassembly statistics
       received 0 packets out-of-order; dropped 0
       peak memory usage 0 KB; current usage: 0 KB
       peak queue length 0

      Police
       rate 128000 bps,8000 limit
       conformed 677 packets, 36558 bytes; actions: transmit
       exceeded 200 packets, 10800 bytes; actions: drop
       conformed 0 bps, exceed 0 bps

   Class-map: ICMP_CLASSMAP (match-all)
     Match: access-group name TO_IN_HOST
     Match: protocol icmp

  Inspect
       Session creations since subsystem startup or last reset 0
       Current session counts (estab/half-open/terminating) [0:0:0]
       Maxever session counts (estab/half-open/terminating) [0:0:0]
       Last session created 00:04:01
       Last statistic reset 00:01:40
       Last session creation rate 0
       Maxever session creation rate 0
       Last half-open session total 0
       TCP reassembly statistics
       received 0 packets out-of-order; dropped 0
       peak memory usage 0 KB; current usage: 0 KB
       peak queue length 0

      Police
       rate 128000 bps,8000 limit
       conformed 0 packets, 0 bytes; actions: transmit
       exceeded 0 packets, 0 bytes; actions: drop
       conformed 0 bps, exceed 0 bps

   Class-map: class-default (match-any)
     Match: any
     Drop
       0 packets, 0 bytes
R1#
```

Figure 29: hping tcp syn flood firewall log

We can add another countermeasure in order to help prevent TCP SYN flood attacks, which is to make the firewall remove half-open connections when it detects lack of activity for an extended period of time. This is achieved by sending a Reset packet (Figure 30) whenever the connection exceeds the timeout threshold that we defined. In the Wireshark print found below (Figure 31) we can see this in action, with the red lines representing a Reset packet in response to the SYN packet sent beforehand.

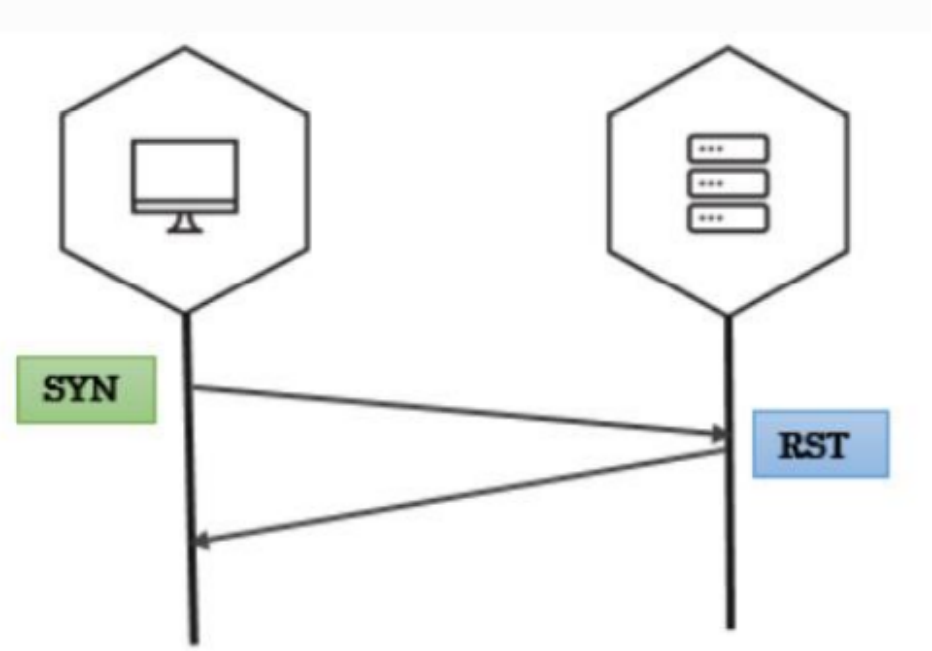


Figure 30: Example of the countermeasure

129	17.782810	220.252.42.131	222.10.10.1	TCP	54 1623 → 80 [SYN] Seq=0 Win=512 Len=0
559	19.425995	220.50.206.18	222.10.10.1	TCP	54 18386 → 80 [SYN] Seq=0 Win=512 Len=0
1107	22.975211	220.72.214.112	222.10.10.1	TCP	54 15955 → 80 [SYN] Seq=0 Win=512 Len=0
2107	24.945794	220.72.214.112	222.10.10.1	TCP	60 15955 → 80 [RST] Seq=1 Win=0 Len=0
168	17.918164	220.75.218.98	222.10.10.1	TCP	54 38499 → 80 [SYN] Seq=0 Win=512 Len=0
411	18.807380	221.113.104.24	222.10.10.1	TCP	54 2344 → 80 [SYN] Seq=0 Win=512 Len=0
1618	24.524387	221.113.104.24	222.10.10.1	TCP	60 2344 → 80 [RST] Seq=1 Win=0 Len=0
401	18.766678	221.132.97.216	222.10.10.1	TCP	54 2334 → 80 [SYN] Seq=0 Win=512 Len=0
1598	24.513589	221.132.97.216	222.10.10.1	TCP	60 2334 → 80 [RST] Seq=1 Win=0 Len=0
810	20.817023	221.144.14.37	222.10.10.1	TCP	54 5599 → 80 [SYN] Seq=0 Win=512 Len=0
362	18.566750	221.148.161.178	222.10.10.1	TCP	54 46209 → 80 [SYN] Seq=0 Win=512 Len=0
1522	24.455593	221.148.161.178	222.10.10.1	TCP	60 46209 → 80 [RST] Seq=1 Win=0 Len=0
966	22.318591	221.162.102.219	222.10.10.1	TCP	54 51225 → 80 [SYN] Seq=0 Win=512 Len=0
849	21.104971	221.165.150.181	222.10.10.1	TCP	54 21514 → 80 [SYN] Seq=0 Win=512 Len=0
1000	22.447878	221.216.94.110	222.10.10.1	TCP	54 51259 → 80 [SYN] Seq=0 Win=512 Len=0
185	17.997304	221.6.237.14	222.10.10.1	TCP	54 38516 → 80 [SYN] Seq=0 Win=512 Len=0
1210	24.151513	221.67.120.107	222.10.10.1	TCP	60 25875 → 80 [SYN] Seq=0 Win=512 Len=0
787	20.726133	221.83.246.165	222.10.10.1	TCP	54 5576 → 80 [SYN] Seq=0 Win=512 Len=0
1277	24.223949	222.10.10.254	222.10.10.1	ICMP	70 Destination unreachable (Host unreachable)
1279	24.223996	222.10.10.254	222.10.10.1	ICMP	70 Destination unreachable (Host unreachable)
1281	24.224005	222.10.10.254	222.10.10.1	ICMP	70 Destination unreachable (Host unreachable)
1283	24.224014	222.10.10.254	222.10.10.1	ICMP	70 Destination unreachable (Host unreachable)
1285	24.224023	222.10.10.254	222.10.10.1	ICMP	70 Destination unreachable (Host unreachable)
1287	24.224032	222.10.10.254	222.10.10.1	ICMP	70 Destination unreachable (Host unreachable)
1289	24.224041	222.10.10.254	222.10.10.1	ICMP	70 Destination unreachable (Host unreachable)

Figure 31: ZBPF closing half-open sessions

AAA with TACACS+

AAA explained

The term AAA stands for Authentication, Authorization and Accounting. Authentication is the act of proving one's identity, therefore it's similar to when you fill out a login form with a username and a password that proves that you're that very user (see Figure 1).

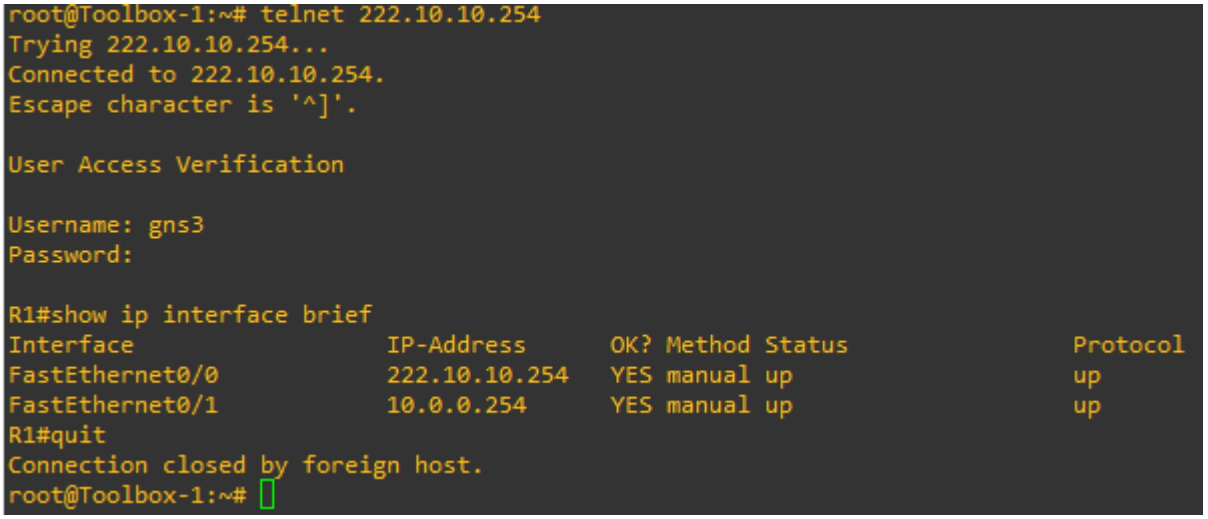


Figure 1: AAA authentication example

Authorization is the act of giving someone permission to perform a specific action, which is like what you would encounter on the TACACS+ configuration file, **/etc/tacacs+/tac_plus.conf**. In this file there's a section where you create groups of users and assign each group with specific permissions (see Figure 2), these permissions dictate what each user is allowed to do.

```
user = gns3 {
    name = "Admin User"
    member = admin
    login = des AxKP5aUynXxrg
                service = junos-exec {
                        local-user-name = remote-admin
        }
}

user = readonly {
    name = "R/O User"
    member = read-only
    login =  des AxKP5aUynXxrg
        service = junos-exec {
            local-user-name = remote-read-only
               }
}

group = admin {
        default service = permit
        service = exec {
                priv-lvl = 15
                }
        }

group = read-only {
        service = exec {
                priv-lvl = 15
                }
        cmd = show {
                permit .*
                }
        cmd = write {
                permit term
                }
        cmd = dir {
                permit .*
                }
        cmd = admin {
                permit .*
                }
        cmd = terminal {
                permit .*
                }
        cmd = more {
                permit .*
                }
        cmd = exit {
                permit .*
                }
        cmd = logout {
                permit .*
                }
}

root@AAA-1:~#
```

Figure 2: AAA authorization

Accounting is the act of logging for the sake of, for example, auditing or tracking someone's activity. In the context of TACACS+, the **/var/log/tac_plus.acct** file (see Figure 3) is a prime example of Accounting, as it logs user activity on the server.

```
root@AAA-1:~# cat /var/log/tac_plus.acct
May 19 15:40:09 10.0.0.254      gns3    tty2    222.10.10.1     start   task_id=6       timezone=UTC    service=shell
May 19 15:40:42 10.0.0.254      gns3    tty2    222.10.10.1     stop    task_id=6       timezone=UTC    service=shell   disc-cause=1    disc-cause-ext=9  p
re-session-time=6       elapsed_time=41 stop_time=1684510842
root@AAA-1:~#
```

Figure 3: AAA accounting

The TACACS+ protocol

When using a TACACS+ based solution, there will be several packets transmitted between server and client (see Figure 4). These packets are encrypted with a key, which is stored in the config file.

	Time	Source	Destination	Protocol	Length	Info
5	9.713080	10.0.0.254	10.0.0.1	TACACS+	89	Q: Authentication
7	9.713931	10.0.0.1	10.0.0.254	TACACS+	109	R: Authentication
11	12.541292	10.0.0.254	10.0.0.1	TACACS+	75	Q: Authentication
12	12.541420	10.0.0.1	10.0.0.254	TACACS+	82	R: Authentication
14	14.457691	10.0.0.254	10.0.0.1	TACACS+	75	Q: Authentication
15	14.458592	10.0.0.1	10.0.0.254	TACACS+	72	R: Authentication
23	14.487351	10.0.0.254	10.0.0.1	TACACS+	112	Q: Authorization
25	14.487991	10.0.0.1	10.0.0.254	TACACS+	84	R: Authorization
33	14.513076	10.0.0.254	10.0.0.1	TACACS+	131	Q: Accounting
35	14.513868	10.0.0.1	10.0.0.254	TACACS+	71	R: Accounting
45	23.575185	10.0.0.254	10.0.0.1	TACACS+	218	Q: Accounting
47	23.576113	10.0.0.1	10.0.0.254	TACACS+	71	R: Accounting

Figure 4: TACACS+ messages

After decrypting said packets we are now able to see some details that were previously hidden, such as the user's authentication information in the Authentication packets (see Figure 5), the user's address and privilege level in the Authorization packets (see Figure 6) and login and logout timestamps in the Accounting packets (see Figure 7).

```
▾ Decrypted Request
    Action: Inbound Login (1)
    Privilege Level: 1
    Authentication type: ASCII (1)
    Service: Login (1)
    User len: 0
    Port len: 4
    Port: tty2
    Remaddr len: 11
    Remote Address: 222.10.10.1
    ASCII Data Length: 0
```

```
Decrypted Reply
    Status: Send Username (0x04)
    Flags: 0x00
    Server message length: 37
    Server message: \nUser Access Verification\n\nUsername:
    Data length: 0
```

```
Decrypted Request
    Flags: 0x00
    User length: 4
    User: gns3
    Data length: 0
```

```
Decrypted Reply
    Status: Send Password (0x05)
    Flags: 0x01(NoEcho)
    Server message length: 10
    Server message: Password:
    Data length: 0
```

```
Decrypted Request
    Flags: 0x00
    User length: 4
    User: gns3
    Data length: 0
```

```
Decrypted Reply
    Status: Authentication Passed (0x01)
    Flags: 0x00
    Server message length: 0
    Data length: 0
```

Figure 5: AAA authentication decrypted packets

```
Decrypted Request
    Auth Method: TACACSPLUS (0x06)
    Privilege Level: 1
    Authentication type: ASCII (1)
    Service: Login (1)
    User len: 4
    User: gns3
    Port len: 4
    Port: tty2
    Remaddr len: 11
    Remote Address: 222.10.10.1
    Arg count: 2
    Arg[0] length: 13
    Arg[0] value: service=shell
    Arg[1] length: 4
    Arg[1] value: cmd*
```

```
Decrypted Reply
    Auth Status: PASS_ADD (0x01)
    Server Msg length: 0
    Data length: 0
    Arg count: 1
    Arg[0] length: 11
    Arg[0] value: priv-lvl=15
```

Figure 6: AAA authorization decrypted packets

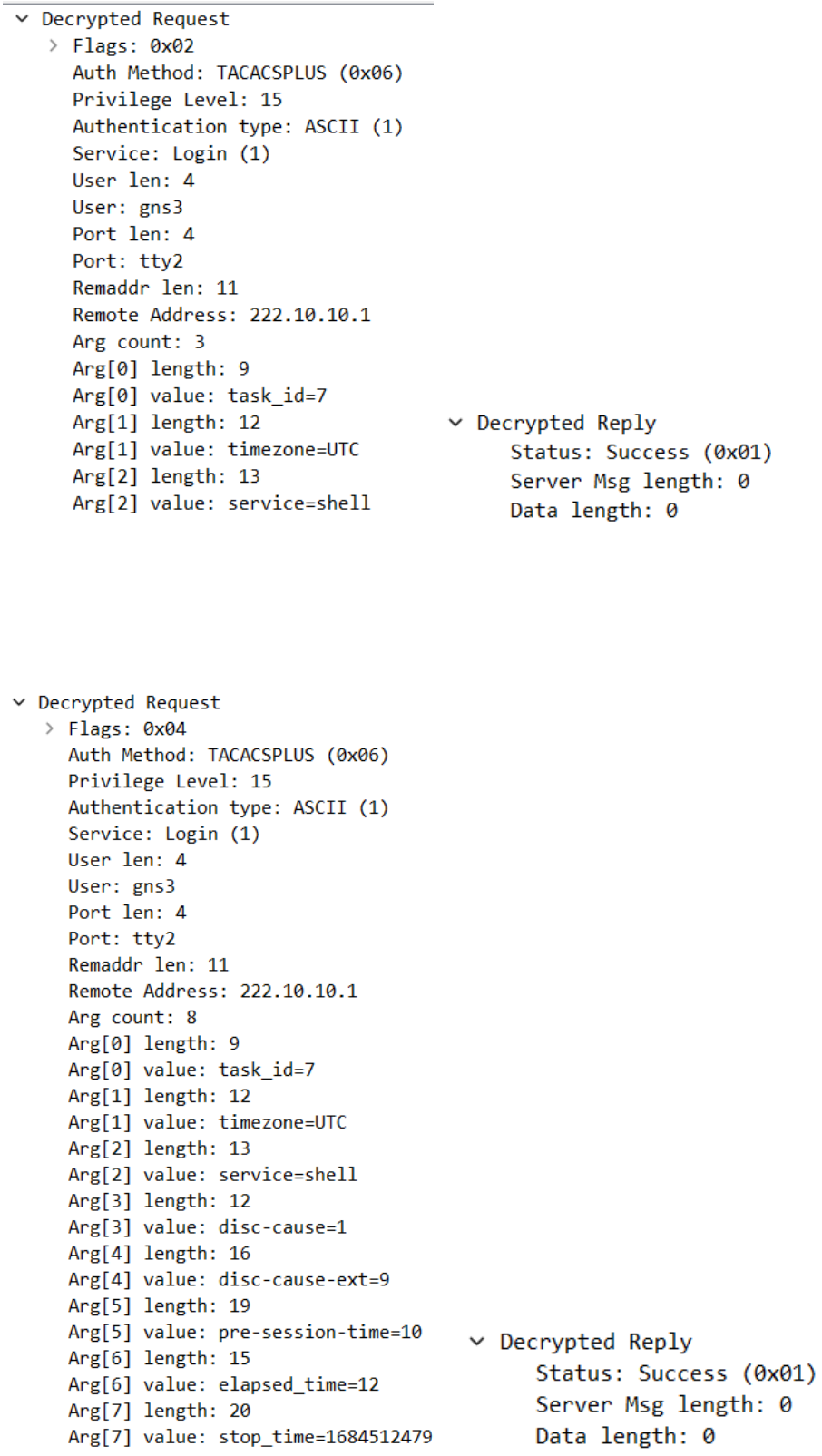


Figure 7: AAA accounting decrypted packets

There's 3 types of packets, one for each A in AAA. The purpose of these packets is very similar to what was explained in the previous section. The Authentication packets handle the user login, the Authorization packets give the user a privilege level based on the user's group and the Accounting packets track the user's activity in the server.

Password generation

The passwords for all users are stored in the TACACS+ config file, however these are not simply written there in plain text. All passwords are encrypted using a DES (Data Encryption Standard) hash, which uses a salt in order to encode the plain text. This is done by using the **tac_pwd** command and the result is then appended to the config file with the following syntax: des <hashed password>.

Local-based vs Server-based

In the case of server-based AAA, the router works as a proxy of sorts. In other words, it communicates with the AAA server as a client in the client's stead, which means the actual client has a higher degree of anonymity compared to what they would have in a local based AAA solution.

Furthermore, a server-based solution is better the larger your network is. When there are multiple network access servers, it is easier to store user profiles in a central point than to have a copy of the user profiles list for every access controller. This makes it so that when there is a need to update a user profile, you will only need to perform the operation once instead of multiple times.

802.1X Authentication

Introduction

The EAP-PEAP protocol consists of creating a secure channel which is encrypted before the password authentication occurs. In the case of 802.1x authentication, the method uses a public server key to certificate a secure tunnel communication between server and client. Said communication uses protocols such as SSL/TLS for the tunnel.

Authentication

First we need to configure the authenticator and add an IP address to the VLAN. Afterwards, we run a config in the supplicant that, in the end, shows that the authentication was successful. In Wireshark we can see the authentication taking place, the last message represents that the connection was indeed successful.

15	18.072452	10.1.1.100	10.1.1.50	RADIUS	281 Access-Request id=1
16	18.072703	10.1.1.50	10.1.1.100	RADIUS	122 Access-Challenge id=1
17	18.110018	10.1.1.100	10.1.1.50	RADIUS	297 Access-Request id=2
18	18.155239	10.1.1.50	10.1.1.100	RADIUS	118 Access-Challenge id=2
19	18.168563	10.1.1.100	10.1.1.50	RADIUS	491 Access-Request id=3
20	18.172630	10.1.1.50	10.1.1.100	RADIUS	1110 Access-Challenge id=3
21	18.183318	10.1.1.100	10.1.1.50	RADIUS	297 Access-Request id=4
22	18.183510	10.1.1.50	10.1.1.100	RADIUS	282 Access-Challenge id=4
23	18.226282	10.1.1.100	10.1.1.50	RADIUS	427 Access-Request id=5
24	18.226900	10.1.1.50	10.1.1.100	RADIUS	157 Access-Challenge id=5
25	18.246446	10.1.1.100	10.1.1.50	RADIUS	297 Access-Request id=6
26	18.246621	10.1.1.50	10.1.1.100	RADIUS	140 Access-Challenge id=6
27	18.283484	10.1.1.100	10.1.1.50	RADIUS	330 Access-Request id=7
28	18.283849	10.1.1.50	10.1.1.100	RADIUS	174 Access-Challenge id=7
29	18.298971	10.1.1.100	10.1.1.50	RADIUS	384 Access-Request id=8
30	18.299344	10.1.1.50	10.1.1.100	RADIUS	182 Access-Challenge id=8
31	18.314897	10.1.1.100	10.1.1.50	RADIUS	328 Access-Request id=9
32	18.315171	10.1.1.50	10.1.1.100	RADIUS	146 Access-Challenge id=9
33	18.355053	10.1.1.100	10.1.1.50	RADIUS	337 Access-Request id=10
34	18.355454	10.1.1.50	10.1.1.100	RADIUS	207 Access-Accept id=10

Figure 8: EAP-PEAP protocol

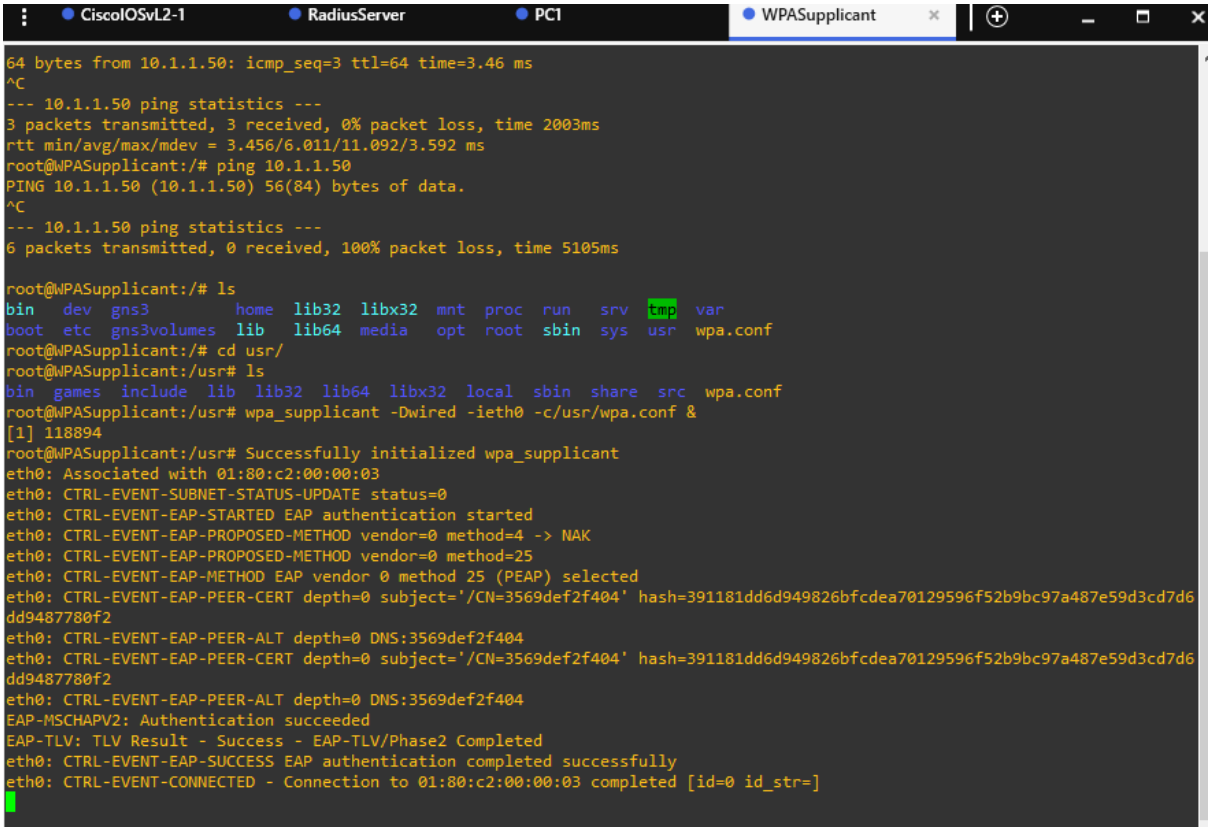


Figure 9: config running in the supplicant

Usage

EAP-PEAP is designed to provide strong security for wireless network authentication. With this in mind, we can affirm that it is very effective against Eavesdropping attacks since it secures confidentiality, and also against Replay Attacks since it uses challenge-response exchanges and cryptographic techniques to ensure that each session is unique.
However, this protocol is not as effective against some types of attacks like man-in-the-middle, because the attacker could impersonate the server, intercept the communication and, with this, gain access to information.

Bibliography

Cisco Systems, Inc - Cisco [Consult. 19 May 2023]. Disponível na internet: https://www.cisco.com/c/pt_pt/index.html

Galaxy Technologies LLC - Getting Started with GNS3[Consult. 20 May 2023]. Disponível na internet: https://docs.gns3.com/docs/

Cisco Networking Academy - NetAcademy [Consult. 19 May 2023]. Disponível na internet: https://www.netacad.com/courses/cybersecurity/network-security