Mestrado em Engenharia Informática
Dissertação
Relatório Final

# Implementation of a distributed intrusion detection and reaction system

João Pedro dos Santos Soares
joaops@student.dei.uc.pt

Orientador:
Prof. Dr. Jorge Granjal

Co-Orientador:
Eng. Ricardo Ruivo

Data: 1 de Setembro de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Mestrado em Engenharia Informática
Dissertação
Relatório Final

# Implementation of a distributed intrusion detection and reaction system

João Pedro dos Santos Soares
joaops@student.dei.uc.pt

Orientador:
Prof. Dr. Jorge Granjal

Co-Orientador:
Eng. Ricardo Ruivo

Júri Arguente:
Prof. Dr. Mário Rela

Júri Vogal:
Prof. Dr. Paulo Simões

Data: 1 de Setembro de 2016

FCTUC DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

# Abstract

Security was not always an important aspect in terms of networking and hosts. Nowadays, it is absolutely mandatory. Security measures must make an effort to evolve at the same rate, or even at a higher rate, than threats, which is proving to be the most difficult of tasks. In this report we will detail the process of the implementation of a real distributed intrusion detection and reaction system, that will be responsible for securing a core set of networks already in production, comprising of thousands of servers, users and their respective confidential information.

*Keywords*: intrusion detection, intrusion prevention, host, network, security, threat, vulnerability, hacking

## Acknowledgments

As we step into the most important project of my student life, there's a couple of people to whom I am extremely grateful, without them, this project and my life in general wouldn't be the same.

I want to start by thanking my supervisor: Professor Jorge Granjal, who I quickly grew to admire for his dedication to this project. I can't stress enough the importance of the presence of a supervisor in the development of a thesis project, from the beginning until the very end, and this was certainly the case. Even on holidays, Professor Jorge Granjal attended to every meeting and made sure I received the appropriate feedback and counseling every time.

Along with Professor Jorge Granjal, comes Engineer Ricardo Ruivo, my co-supervisor and work colleague, who more than that, became my friend and mentor at work. When I started working at SIC (Serviço de Informática e Comunicações), there was no better way I could be welcomed if not with Ricardo's good mood and professionalism. He taught me a great deal of what I know now, and is one of the main reasons I wish to continue to work at SIC in the following year. His abilities and skills at work still amaze me to this day.

I must also thank my family, that even if not directly involved with this project, supported me for as long as I can remember, not ever doubting my abilities and always encouraging me to go further and to work harder. All this while providing me with all the resources and freedom that I wanted and/or needed. I'm eternally grateful and lucky to have been born surrounded by my wonderful parents and sister.

I also want to thank all my close friends that contributed to my resilience throughout the year, by providing me with incredibly fun, and also sometimes crazy moments. In this group of friends I want to give a special thanks to Maxi (João Lourenço), who played a great part in helping me build my current personality, by constantly sharing his optimistic view with me, which was one of the major reasons that kept me going, and made me pursue the master's degree.

Last but not least, I want to thank the newest addition to my life: Rita, my girlfriend, who I've known for almost a year, but only a few months ago have I started to look at her the way I already should have in the beginning. Her presence alone inspired me and encouraged me to work, and helped me surpass my sporadic laziness. It impresses me how far we've come and what we've achieved in only a couple of months, we are definitely not normal, and hopefully we stay that way for more years to come.

# List of Figures

1 Gantt Chart for the first semester. . . . . . . . . . . . . . . . . . . . . . . . . 18
2 Gantt Chart for the second semester. . . . . . . . . . . . . . . . . . . . . . . . 19
3 New Gantt Chart for the second semester. . . . . . . . . . . . . . . . . . . . . 20
4 Switched network versus a hubbed network. . . . . . . . . . . . . . . . . . . . 25
5 The association (and dissociation) process of a client to a wireless access point. 28
6 Using an intermediary to communicate between wireless clients. . . . . . . . 29
7 TCP's three-way handshake . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
8 A normal TCP data exchange . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
9 TCP sequence number being brute-forces . . . . . . . . . . . . . . . . . . . . . 38
10 Wireshark logs of a DNS query . . . . . . . . . . . . . . . . . . . . . . . . . . 40
11 Amplification attack in practice . . . . . . . . . . . . . . . . . . . . . . . . . . 41
12 A minimalist diagram of the department's network infrastructure. . . . . . . 63
13 Top vulnerabilities when aggregating both inbound and outbound traffic. . . 65
14 Top vulnerabilities exploited in attacks to the department's network. . . . . 66
15 Top targeted machines in attacks to the department's network. . . . . . . . . 66
16 Top origin countries in attacks to the department's network. . . . . . . . . . 67
17 Top vulnerabilities exploited in attacks originated at the department's network (Outlier). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
18 Top machines launching attacks originated at the department's network (Outlier). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
19 Top vulnerabilities exploited in attacks originated at the department's network. 69
20 Top machines launching attacks originated at the department's network. . . 70
21 Top targeted countries in attacks originated at the department's network. . . 71
22 Logical architecture diagram. . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
23 Physical architecture diagram. . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
24 Interaction between the components of our architecture. . . . . . . . . . . . 85
25 Network diagram of the current architecture implementation . . . . . . . . . 95
26 Puppet manifest hierarchy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
27 Log flow diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
28 Top vulnerabilities detected by the NIDS . . . . . . . . . . . . . . . . . . . . . 104
29 Reader and crawler interaction . . . . . . . . . . . . . . . . . . . . . . . . . . . 105
30 Testing setup diagram. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110

# List of Tables

1 Comparison of traffic capture techniques. . . . . . . . . . . . . . . . . . . . . . . 49
2 Comparison of detection strategies. . . . . . . . . . . . . . . . . . . . . . . . . . . 56
3 Comparison of intrusion detection/prevention systems. . . . . . . . . . . . . . 61
4 Server specifications. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
5 Availability scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
6 Modifiability scenario 1 - Updates . . . . . . . . . . . . . . . . . . . . . . . . . . 76
7 Modifiability scenario 2 - Network topology changes . . . . . . . . . . . . . . 76
8 Performance scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
9 Testability scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
10 Usability scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
11 Scalability scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78
12 Architecture validation tests. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
13 Reaction times of the three IDS applications. . . . . . . . . . . . . . . . . . . . 134
14 Phase 1 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135
15 Phase 2 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136
16 Phase 3 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 137
17 Statistics producing test. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 138
18 Phase 1 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
19 Phase 2 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
20 Phase 3 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 140
21 HIDS Availability test results. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 140
22 Phase 1 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142
23 Phase 2 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142
24 Phase 3 results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 143
25 Detailed HIDS scalability testing. . . . . . . . . . . . . . . . . . . . . . . . . . . 144
26 Misuse-based detection validation. . . . . . . . . . . . . . . . . . . . . . . . . . 145
27 Detailed SQL injection test results. . . . . . . . . . . . . . . . . . . . . . . . . . 146
28 Detailed cross-site scripting test results. . . . . . . . . . . . . . . . . . . . . . . 146
29 Detailed NTP amplification attack test results. . . . . . . . . . . . . . . . . . . 146
30 Anomaly-based detection validation. . . . . . . . . . . . . . . . . . . . . . . . . 147
31 Detailed FTP bruteforcing test results. . . . . . . . . . . . . . . . . . . . . . . . 147
32 Detailed TCP Syn Flood test results - Phase 1. . . . . . . . . . . . . . . . . . 148
33 Detailed TCP Syn Flood test results - Phase 2. . . . . . . . . . . . . . . . . . 148
34 Detailed HTTP directory scan test results. . . . . . . . . . . . . . . . . . . . . 148
35 Detailed Port scanning test results. . . . . . . . . . . . . . . . . . . . . . . . . . 148
36 Detailed system availability threatening test. . . . . . . . . . . . . . . . . . . . 149
37 Detailed OSSEC integrity checking testing. . . . . . . . . . . . . . . . . . . . . 149
38 Detailed OSSEC rootkit detection testing. . . . . . . . . . . . . . . . . . . . . . 149

# Index

**1 Introduction** **17**

1.1 Contextualization . . . 17
1.2 Objectives . . . 17
1.3 Life cycle and Planning . . . 18
1.4 Report structure . . . 20

**2 State of the Art** **23**

2.1 Threats and vulnerabilities . . . 23
2.1.1 Physical Layer . . . 23
2.1.2 Link Access Layer . . . 24
2.1.3 Network Layer . . . 31
2.1.4 Transport Layer . . . 34
2.1.5 Application Layer . . . 39
2.2 Detecting intrusions: Technologies and strategies . . . 44
2.2.1 Techniques for achieving network visibility . . . 44
2.2.2 Detection strategies . . . 49
2.2.3 Comparison of detection strategies . . . 56
2.2.4 Types of intrusion detection systems . . . 56
2.3 Preventing intrusions: Technologies and strategies . . . 57
2.3.1 Network intrusion prevention . . . 57
2.3.2 Host intrusion prevention . . . 57
2.4 Exploring intrusion detection/prevention systems . . . 58
2.4.1 Free & open-source products . . . 58
2.4.2 Commercial products . . . 60
2.4.3 Comparison of intrusion detection/prevention systems . . . 61

**3 Preliminary Work** **63**

3.1 Traffic capture and analysis strategy . . . 64
3.1.1 Capturing the network traffic . . . 64
3.1.2 Analyzing network traffic . . . 65
3.2 Preliminary analysis results and conclusion . . . 65
3.2.1 Attacks against the DEI's network . . . 66
3.2.2 Attacks originated at the DEI's network . . . 67

**4 System Architecture** **73**

4.1 Functional Requirements . . . 73
4.1.1 Detecting . . . 73
4.1.2 Reacting . . . 74
4.1.3 Alerting and Logging . . . 74
4.1.4 Displaying . . . 74
4.1.5 Configuring . . . 75
4.1.6 Securing . . . 75

4.2 Quality Attributes . . . . . 75
4.2.1 Availability . . . . . 75
4.2.2 Modifiability . . . . . 76
4.2.3 Performance . . . . . 77
4.2.4 Testability . . . . . 77
4.2.5 Usability . . . . . 77
4.2.6 Scalability . . . . . 78
4.3 Proposed architecture . . . . . 79
4.3.1 Logical architecture . . . . . 79
4.3.2 Physical architecture . . . . . 83
4.3.3 Selection of the preferred technologies . . . . . 85
4.4 Architecture validation and evaluation strategies . . . . . 87
4.4.1 Penetration testing tools . . . . . 87
4.4.2 Traffic Generators . . . . . 89
4.4.3 Packet Sniffing . . . . . 89
4.4.4 Availability testing . . . . . 89
4.4.5 Modifiability testing . . . . . 89

**5 Implementation** **91**
5.1 Accessing the data . . . . . 91
5.1.1 Network data . . . . . 91
5.1.2 Host data . . . . . 92
5.2 High availability . . . . . 92
5.2.1 NIDS high availability . . . . . 92
5.2.2 HIDS high availability . . . . . 94
5.3 Current architecture implementation . . . . . 94
5.4 Configuration Management . . . . . 96
5.5 Logging . . . . . 98
5.5.1 Standard logging format . . . . . 98
5.5.2 The logging flow . . . . . 101
5.6 The Reactor - Reaction/alerts engine . . . . . 105
5.7 Intrusion displaying . . . . . 107
5.7.1 Log.io . . . . . 107
5.7.2 Splunk . . . . . 108

**6 Architecture validation** **109**
6.1 Testing setup . . . . . 110
6.2 Test list & results . . . . . 111
6.3 Observed limitations . . . . . 116
6.3.1 #T04 . . . . . 116
6.3.2 #T05, #T07 & #T15 . . . . . 116

**7 Conclusion and future work** **119**

**A Detailed test results** **129**
A.1 #T01: Misuse-based system test . . . 129
A.1.1 Layer 2: Link Access . . . 129
A.1.2 Layer 3: Network . . . 129
A.1.3 Layer 4: Transport . . . 130
A.1.4 Layer 5: Application . . . 131
A.2 #T02: Host-based system test . . . 131
A.2.1 General intrusions . . . 131
A.2.2 Integrity Checks . . . 132
A.2.3 Rootkit Checks . . . 132
A.3 #T03: Anomaly-based system test . . . 133
A.4 #T06: Logging mechanism validation . . . 134
A.5 #T05 & #T07: Reaction/Alerting test . . . 134
A.5.1 Phase 1 - 100Mbps: Results . . . 135
A.5.2 Phase 2 - 1000Mbps: Results . . . 136
A.5.3 Phase 3 - MAX: Results . . . 137
A.6 #T08: Intrusion displaying application test . . . 137
A.7 #T09: Statistics server/data analyzer test . . . 138
A.8 #T10: Automatic reporting generation test . . . 138
A.9 #T11: Configuration manager test . . . 138
A.10 #T13: Availability & Modifiability test . . . 138
A.10.1 NIDS . . . 138
A.10.2 HIDS . . . 140
A.11 #T14: Modifiability test . . . 141
A.12 #T15 & #T17: Performance and scalability test . . . 142
A.12.1 NIDS . . . 142
A.12.2 HIDS . . . 143
A.13 #T16: Testability tests . . . 144
A.13.1 Misuse-based (Snort) . . . 144
A.13.2 Anomaly-based (Bro) . . . 146
A.13.3 Host-based (OSSEC) . . . 148

**B Developed scripts description** **151**

# 1 Introduction

Nowadays, attacking networks and hosts worldwide has become a standard. Everyday millions of networks and devices are maliciously proof tested and scanned for a single breach, a single line of code missing in some application or one simple "true" that should be "false" in a service configuration.

Firewalls and access control blacklists have long stopped being enough, networks need a reliable mechanism that stands in the first line of defense, analyzing every bit that traverses the network, detecting suspicious activity and reacting to it.

The Department of Informatics Engineering and Pólo II's networks are no exception. Due to our dimension and relevance, we are constantly the targets of many organized attacks just craving for a minimal flaw that could scale up to a catastrophe. We feel the need of an intrusion detection and reaction system, and from this point on, the situation could only become more dangerous, as attacks are becoming automated, more synchronized and new vulnerabilities are discovered everyday.

It is most definitely a complex task to be implemented in a large network already in production, but at the same time, it is what makes it motivating and rewarding.

## 1.1 Contextualization

This project is to be included in the department's systems administration scope, and maintaining it will make part of the everyday tasks' of the network managers and system administrators.

Not only will it embrace the Department of Informatics Engineering network as its home network, but also include the entire Pólo II communications infrastructure in its area of effect. Nonetheless, the system's architecture will not be confined to this scope only, seeing that in the future, it may scale to even bigger proportions, like the whole network of University of Coimbra.

## 1.2 Objectives

Our goal is to start by investigating the state of the art related to threats and vulnerabilities, so we can learn the theory, targets and objectives behind each attack. We will then search for detection and prevention strategies, methods and technologies with the objective of combining them, and building a **large scale distributed intrusion detection and reaction system**.

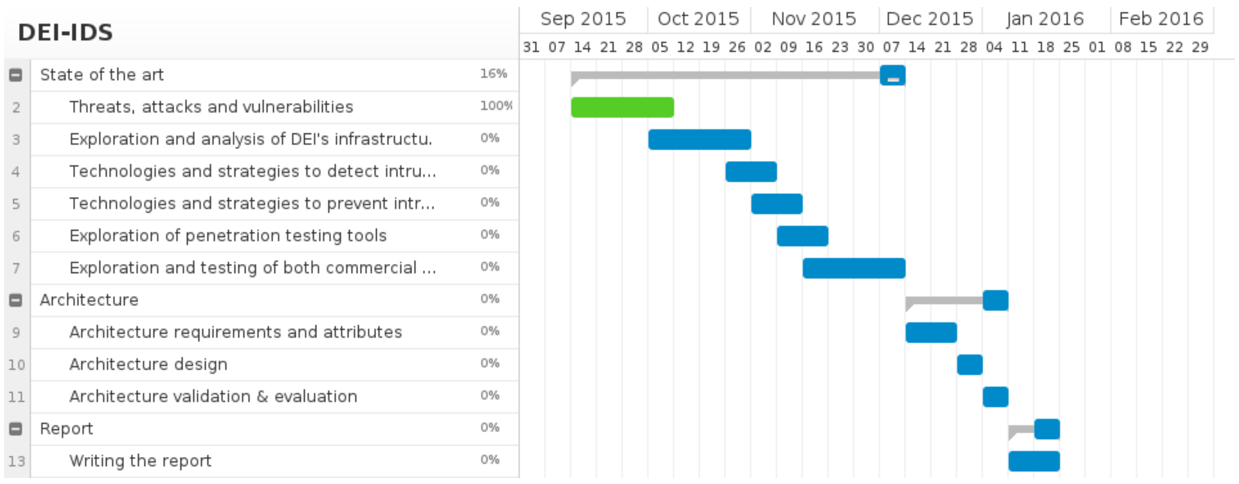
We aim for more than a simple detection and reaction system. Reaching information, filtering data and delivering it at the right time is also equally important, for this reason, we intend to build, implement, validate and evaluate an architecture from scratch, that will also support **centralized logging integration and displaying**, as well as a f**ully customizable alerts system**.

## 1.3 Life cycle and Planning

Defining a life cycle for such a project is not the easiest of tasks. We will start by delineating requirements and proposing an architecture, taking into account what we know about the project and its scope, and only then, after the previous steps were completed, the development and implementation phase will start. This would lead us to a **waterfall model**, however realistically, even though we have a clear idea of what will happen along both semesters, a system as complex as this one will inevitably suffer unexpected minor changes and tweaks in the future, as it is to be inserted in a somewhat volatile environment that is networking, also constantly evolving and being upgraded.

Nonetheless, **we opted to follow the traditional waterfall model**, as we feel it is the best suited model for this kind of project, and try as best as we can to follow each step of it, without going back to make changes, unless it is strictly unavoidable.

The following image (Figure 1) shows our planning for the first semester through a Gantt Chart:



| | | |
|---|---|---|
| | State of the art | 16% |
| 2 | Threats, attacks and vulnerabilities | 100% |
| 3 | Exploration and analysis of DEI's infrastructu. | 0% |
| 4 | Technologies and strategies to detect intru... | 0% |
| 5 | Technologies and strategies to prevent intr... | 0% |
| 6 | Exploration of penetration testing tools | 0% |
| 7 | Exploration and testing of both commercial ... | 0% |
| | Architecture | 0% |
| 9 | Architecture requirements and attributes | 0% |
| 10 | Architecture design | 0% |
| 11 | Architecture validation & evaluation | 0% |
| | Report | 0% |
| 13 | Writing the report | 0% |

Figure 1: Gantt Chart for the first semester.

The following Gantt chart (Figure 2) represents the planning for the second semester.

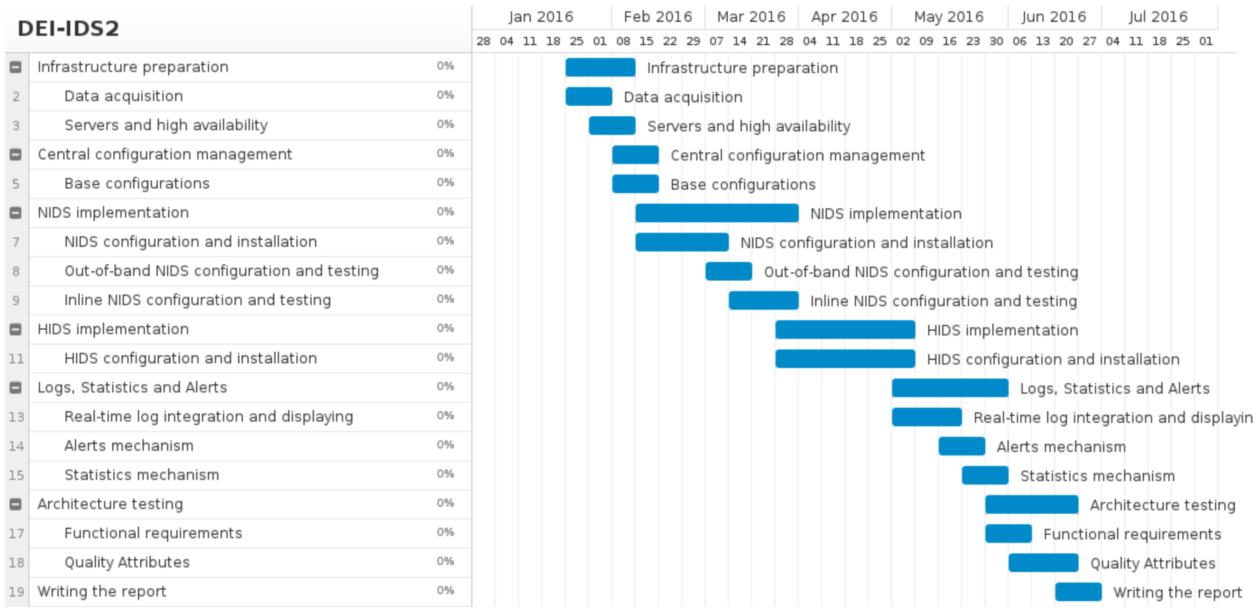


Figure 2: Gantt Chart for the second semester.

We will start by preparing the whole infrastructure, which involves configuring the data acquisition devices, like network taps and switches, as well as getting every server ready with high availability in consideration.

Then, every machine will be prepared to be remotely configured by our centralized configuration management server. The implementation of NIDS starts in the 16th of February and lasts 6 weeks, which includes configuration and installation of every component, integration with the configuration manager, out-of-band and inline IDS/IPS configuration and testing.

The period starting 29th of March until the 2nd of May will be dedicated solely to the implementation of the HIDS manager and agents, also their integration with the configuration manager.

Starting at the 3rd of May, we will spend the next 4 weeks configuring the centralized logging mechanism and its integration with the remaining components. In this period of time, we will also set up the real-time intrusion displaying application, the alerts mechanism and the statistics producer/displaying server.

Finally the last 5 weeks of the semester will be dedicated to testing, validating and evaluating the entire architecture already in production, as well as finishing the writing of the final report.

### Planning deviations

As with most engineering projects that rely on external factors, **we also encountered a few drawbacks** on the second semester, that made us deviate from the original planning.

Two of our core nodes (NIDS2 and HIDS2) didn't arrive in time and its delivery was pushed to the end of July. Without these nodes, **we wouldn't be able to validate the architecture**, since most of the quality attributes and functional requirements were dependent on high availability and load balancing.

After knowing this would happen, we decided to spend more time improving our logging, statistics and reaction/alerting mechanisms, and also build a torrent parser and filename resolver (Reader and Crawler, explained in Section 5.5.1). Likewise, we were also able to plan a **more efficient testing methodology and setup**.

Along with the mentioned drawbacks, we made the decision of **not including an inline NIDS in our current architecture solution**, as explained in 6.3.1.

The following Gantt chart (Figure 3) represents the planning for the second semester, with all the drawbacks and new due dates included.

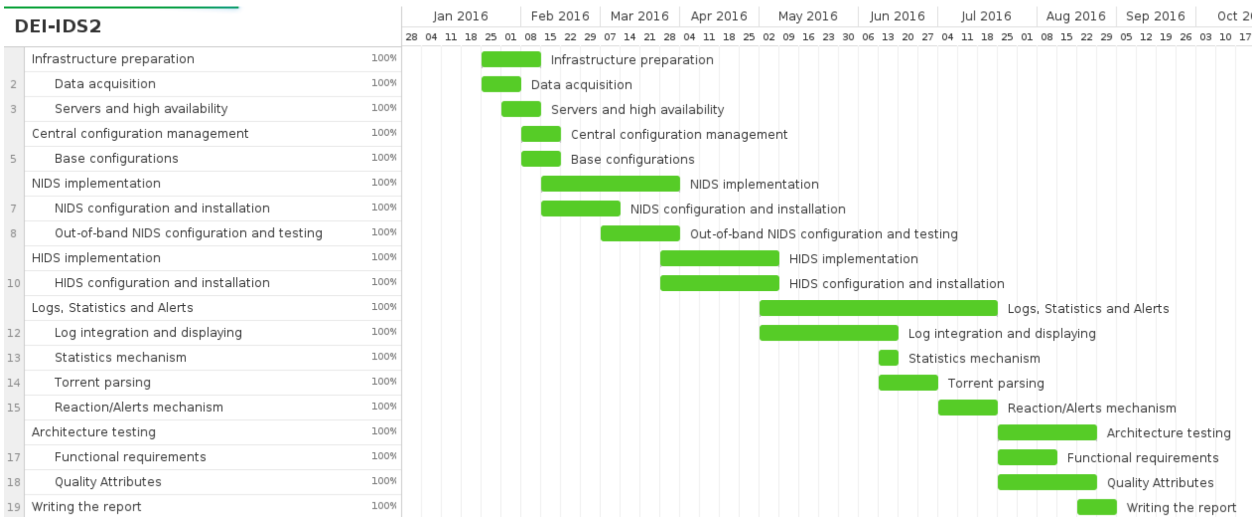


Figure 3: New Gantt Chart for the second semester.

## 1.4 Report structure

This report structure is as follows: The first chapter (Chapter 2) is dedicated to the state of the art. We will start by exploring the state of the art of threats of vulnerabilities, in which we will follow the five layers of the TCP/IP model and detail their respective function and flaws. Then, we will investigate the theory and the strategies behind intrusion detection systems, which includes data acquisition techniques and detection strategies. The next step is repeating the previous step but for prevention strategies, both host-based and network-based. Finally, to conclude the state of the art section, we will search the market for both commercial and open-source intrusion detection and prevention solutions, and place them side by side, highlighting each one's advantages and disadvantages.

The following chapter will be about preliminary work (Chapter 3), in which we will explore and conduct a series of intrusion detection tests in the department's network, and analyze its results by using a data analysis application.

We will then reach one of the core chapters of the report: The system's architecture (Chapter 4), which will include the definition of both functional requirements and quality attributes, followed by the final architecture proposal and validation/evaluation strategies.

The following chapter marks the beginning of the second part of this project: Implementation (Chapter 5), which will address the practical component of our work, by detailing everything we implemented, followed by the architecture validation and testing (Chapter 6).

Lastly, we will present and propose our plans for the future and conclusions/closing thoughts (Chapter 7).

# 2 State of the Art

## 2.1 Threats and vulnerabilities

In order to start developing a complete intrusion detection system for already stable and developed networks like DEI's and Pólo II's communications infrastructures, we must first conduct a study on the state of the art for current technologies and alternatives regarding intrusion detection and prevention.

Nevertheless, even before starting to dig into the intrusion detection/prevention world, there's a certain layer that must explored beforehand: **Threats and vulnerabilities**. Without getting a good overview on the issues that major networks are going through (or went through), there's no way we can build an effective defense mechanism.

Having said this, in the following subsections we will follow and analyze the **layers of the TCP/IP model** and list the most common and the most dangerous threats in each one, as well as briefly explain the theory behind each attack and why are they successful. We feel the TCP/IP stack is the most appropriate way to categorize vulnerabilities and their areas of effect, as opposed to the OSI model, which although being more complex, would provide us with unnecessary granularity for this matter.

### 2.1.1 Physical Layer

The physical layer supports the **last step before the actual transmission of data**, since it is responsible for the physical communication between stations [2]. This layer has a really tight relationship with hardware (i.e. network interface controllers) and involves highly complex techniques in order to transmit bits of information through physical mediums like copper, optic fiber, or air.

Considering this is a report about intrusion detection/prevention systems (IDS/IPS), **we will not cover security and vulnerabilities at this layer**, as they mostly consist of physical acts that cannot be prevented by neither an IDS or an IPS. Examples of physical layer exploits can range from physical theft/destruction of devices or the unplugging of network/power cables, to more advanced techniques like intentionally changing a network structure for eavesdropping purposes. Security against these kinds of threats can only be achieved with physical means like gates, fences, video surveillance, biometric validation devices, among many others.

We would like to note however, that although being a highly unexplored territory, with a lack of successful products in the market, **intrusion detection at the physical layer is evolving**, specially when it comes to wireless networks (for example, our Wireless LAN Controller at the department partially implements physical layer security by jamming ad hoc networks in order to preserve the electromagnetic spectrum). There are many proposals in the literature for IDS/IPS at this layer, for example [76] and [78], but even though we are aware of the relevance of this layer, we still chose not to cover it in depth.

### 2.1.2 Link Access Layer

The Link Access Layer (or Data-Link Layer in the OSI Model) is the layer responsible for dealing with **data transmission between two directly connected stations**. It defines protocols for communication between the two nodes, as well as establishing and terminating connections. It is the realm of Media Access Control (MAC) Addresses [12] and Wireless Local Area Networks (WLANs)/Virtual Local Area Networks (VLANs) protocols [2].

It is a commonly neglected layer when it comes to intrusion detection, due to its low level interaction with hardware and heavy dependency on rigid protocol standards [11], when compared to the upper three layers. An highly trusted layer of this importance, without proper authentication mechanisms to limit access, **cannot go unrevised**, as it is crucial to the interoperability between devices and is usually widely targeted for exploitation of vulnerabilities.

We will now go through the most common issues with the Link Access Layer, in addition to the major gaps in the protocols supported by it. As this is the layer where basically all communication starts, we will split the vulnerabilities in three parts: **Wired vulnerabilities**, which will include vulnerabilities related exclusively to wired networks, **Wireless vulnerabilities** for wireless networks and **General vulnerabilities** for vulnerabilities that concern both types of networks.

#### 2.1.2.1 Wired Vulnerabilities

Before heading into the real threats of wired networks, we must make an emergency stop on **Hubbed networks**.

Hubbed networks are innately a vulnerability. A hub is a network device that by default, **broadcasts all network packets to every station on the network**. This basically means that every node connected to the hubbed network, will **inevitably share their traffic** with every other node.

This is a type of network that relies on the device's network card to decline the packets that are not meant for it. The implications are **disastrous**, as any malicious user can simply change his network interface controller (NIC) to *Promiscuous Mode*, which will cause the controller to **accept all traffic**, instead of only accepting the frames it is supposed to receive, thus easily sniffing the entire network.

In conclusion, the following vulnerabilities will be focused mainly on switched networks, which in contrast, only send traffic to devices that intend to receive it. This is primarily related to hubbed networks no longer being as relevant and popular as they were before, as a result of natural technology evolution. Figure 4 shows a comparison of a hubbed network and a switched network.

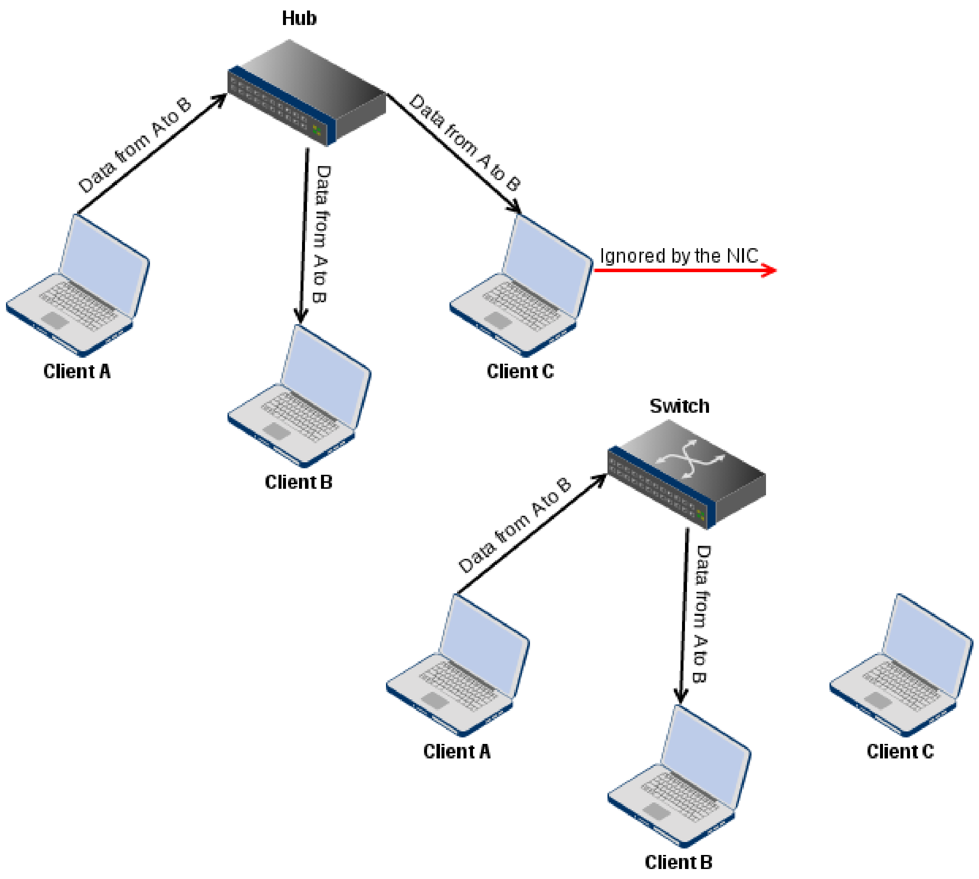


Figure 4: Switched network versus a hubbed network.

## Content Addressable Memory (CAM) Table Exhaustion

As explained in the previous point, the main difference between switched and hubbed networks is that in switched networks, hosts only receive traffic that belongs to them. To make this possible, switches must keep a track of all the network hosts' physical address, mapped to the respective switch port in which to send traffic. This mapping is done in the **Content Addressable Memory (CAM) table**.

The main problem here is that in order to update the CAM table, a host can simply broadcast his MAC address **without any form of validation or authentication**.

This opens up for a wide variety of possible exploits, one of them being the **CAM Table Exhaustion**, which consists in **repeatedly broadcasting spoofed MAC addresses** in order to "flood" the switch's CAM table with fake addresses, thus filling up the allocated memory for the purpose [1]. At this point, the switch will become incapable of maintaining the address-port mapping and, for the sake of keeping the traffic flowing in the network, it will start acting as a hub, which as we've seen before, is definitely not a good sign, as the entire network traffic has just been compromised.

We should also note that this attack only works on certain switches (usually older/inexpensive), whose software and/or firmware doesn't properly preserve the security and consistency of their CAM table.

**Forms of attack:** Using a simple tool like **macof** [4], that essentially generates random MAC addresses and advertises them to the desired switch [3].

**Why does this succeed?** No form of validation and/or authentication when clients broadcast their physical addresses. Also lacking time-based defense mechanisms against flooding techniques like this one.

#### 2.1.2.2 Wireless Vulnerabilities

<u>Deauthentication Attack</u>

When it comes to understanding how **Deauthentication Attacks** work, we must first understand how a wireless client connects and disconnects from a wireless network.

In order to connect to a wireless network, a wireless client will start by sending an **Authentication Frame**. This is a 802.11 standard frame and it is essentially a block of data containing information about the station's NIC. The access point will then reply with another Authentication Frame which will contain the response for the client's request: **Acceptance**, **Rejection** or **Challenge**. Acceptance means the client can now associate with the access point, while Rejection means the opposite. Challenge just introduces an additional authentication phase where the client must send an encrypted text back to the access point (used in WEP/WPA wireless networks). [12]

For the sake of simplification, let's assume the client was accepted by the access point. The next step for the wireless client consists in sending an **Association Request Frame**, which the access point will then reply with an **Association Response Frame**. At this point, the client will finally be associated with the access point and free to interact with the network. Figure 5 shows the complete association process of a client to a wireless access point.

What if a client wishes to **Deauthenticate** from the network? Well, there is another 802.11 frame for that purpose, the **Deauthentication Frame** [12]. This frame can be sent by either a client or an AP wanting to terminate its secure communication. The other party will then interpret this frame and safely drop the communication.

The issue here is that **this frame goes unencrypted**, as with all the standard 802.11 frames. There's also no form of "signing" or "authentication" when sending these frames. Any malicious user can simply **forge a deauthentication frame** for any client he wants, by just knowing the victim's MAC address, which will result in the victim being abruptly disconnected from the network. In addition, the attacker can simply **flood the network with deauthentication frames**, thus permanently preventing clients from joining it [1].

On a side note, this attack is also widely used for WPA/WPA2 password cracking, in which an attacker would force a client to deauthenticate and authenticate again, just to capture its **initial handshake** packets. The attacker would then proceed to brute-force the handshake using dictionary attacks, in the hopes of cracking the wireless pre-shared key.

**Forms of attack:** Tools like **airplay-ng** [7] or the **Metasploit Framework** [5] provide an easy enough interface to flood networks with Deauthentication Attacks.

**Why does this succeed?** 802.11 frames must go unencrypted as it is a rigid standard on the Link Access Layer. There's no form of authentication and no threshold over time for preventing the flood of deauthentication frames. There's however an approved amendment for this matter, named 802.11w, although it is still in the process of being developed in major networks, as well as being supported in most network card drivers.

## <u>Fake Access Point Attack</u>

Anyone with the right NIC can create an access point. However, wireless clients are the ones to choose whether or not they want to associate with it. In a **fake access point** attack, the victim is practically **forced** to associate with a **malicious access point**. This is generally more threatening in conjunction with **social engineering techniques**, for example if an attacker deliberately creates a free hotspot with the intent to start a man-in-the-middle attack against its victims.

With the purpose of exploiting this vulnerability, we must go through two more 802.11 standard frame types: **Probe Request Frame** and **Probe Response Frame**. A station sends a Probe Request Frame when it is looking for information about nearby stations. Commonly, wireless clients send probe requests looking for networks they have associated with in the past. An access point can reply to a Probe Request Frame by sending a Probe Response Frame. [12]

We should note that, as with all the 802.11 frames, these ones also go unencrypted, **without any kind of security**. These are frames sent by the network interface controllers when a station is looking to associate with another station, usually an access point.

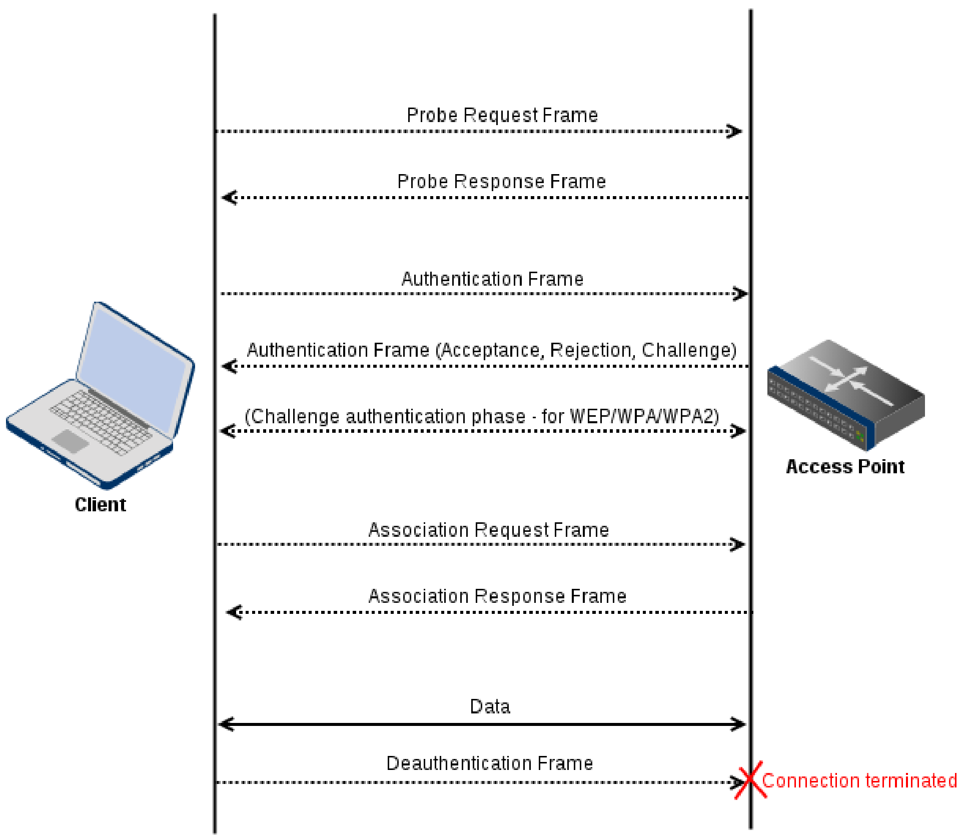

Probe Request Frame
Probe Response Frame
Authentication Frame
Authentication Frame (Acceptance, Rejection, Challenge)
(Challenge authentication phase - for WEP/WPA/WPA2)
Client
Access Point
Association Request Frame
Association Response Frame
Data
Deauthentication Frame
Connection terminated

Figure 5: The association (and dissociation) process of a client to a wireless access point.

In a fake access point attack, a malicious user will **replicate the characteristics of a trusted access point**, spoofing things like its MAC address and using the same Service Set Identifier (SSID) to broadcast the wireless network, possibly with more power than the original AP. Then, the malicious access point can simply start **responding to all probe requests**, thus making the victims believe they are associating with their trusted station, when in reality they are being the targets of a man-in-the-middle attack [1].

An attacker can also combine the previous vulnerability (Deauthentication attack) with a fake access point attack. By flooding the victim's trusted network with deauthentication frames, the malicious user can easily prevent the victim from associating with its trusted station, eventually **forcing it** to associate with its **second best alternative** - the fake access point.

**Forms of attack:** Tools like **hostpad** [6] or **airbase-ng** [7] can be used to create access points. **macchanger** [8] is a tool that makes it easy enough to spoof mac addresses.

**Why does this succeed?** Clients automatically connect to access points that match the characteristics they look for with probe requests. In part, this vulnerability is also possible due to the previous one.

## Hidden Node Attack

A Hidden Node Attack is once again another vulnerability in which wireless frames from the 802.11 standard are exploited. This time, we will talk about **Request To Send (RTS) Frames** and **Clear to Send (CTS) Frames**. [12]

In wireless communications, nodes can sometimes **share an intermediary** to communicate. This happens in situations where a node can't directly communicate with another one (any node can be considered the hidden node, it depends on the perspective) but there is a station that will serve as an intermediary, that can effectively communicate with both.

For the sake of minimizing collisions, there must exist some sort of synchronization mechanism between stations. Considering wireless **half-duplex nature**, two nodes cannot send and receive traffic at the same time, so in order for a node to receive data, it must first stop transmitting. Having said this, when a node wishes to transmit data, it will broadcast a **Request to Send Frame**, stating that it is ready to start sending information.

This is when the intermediary node starts acting. After receiving the RTS frame from the communicating station, it will broadcast a Clear to Send Frame to all neighbour stations, making them **halt data transmission** for a certain period of time, until all traffic from the communicating node is received, therefore successfully avoiding collisions. Figure 6 shows the process of using an intermediary to communicate between two clients that can't reach each other in a wireless network.

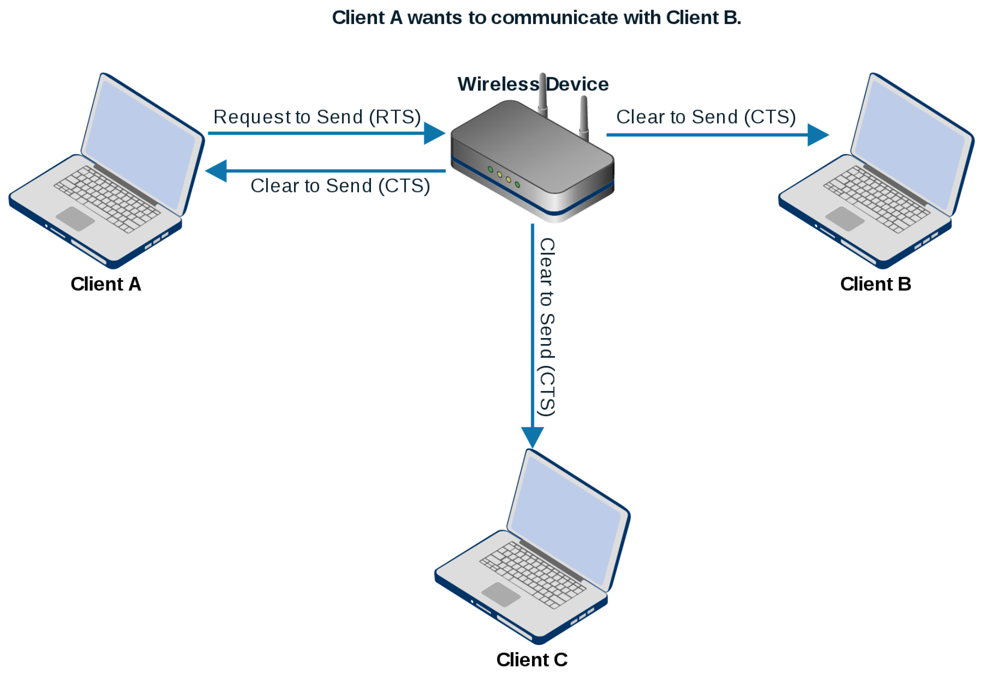


Figure 6: Using an intermediary to communicate between wireless clients.

A hidden node attack consists in **flooding the network with these frames**. If a malicious user decides to repeatedly broadcast RTS Frames, all the neighbour stations will assume they are about to receive data, thus respectively broadcasting CTS Frames, which will disrupt all the communication in the network, by **making every neighbour node stop transmitting data**. [1]

On the other hand, if an attacker starts sending multiple CTS Frames, its neighbour stations will interpret it as being a normal node, trying to receive data and avoiding collisions, therefore they will also stop transmitting, which will have a very similar consequence to the situation explained above.

**Forms of attack:** The **Metasploit Framework** has all the tools required to flood networks with CTS/RTS frames.

**Why does this succeed?** No default threshold for CTS/RTS frames over time, therefore allowing these frames to flood the network.

#### 2.1.2.3 General Vulnerabilities

**Address Resolution Protocol (ARP) [13] Spoofing Attack**

The Address Resolution Protocol is a **critical protocol** when it comes to communication inside a multiple access network. It is responsible for translating the network layer addresses (IP) into link access layer addresses (MAC). It essentially provides core functions when **transitioning between layer 3 and layer 2**.

Similarly to the CAM table addressed previously in the wired vulnerabilities, there is also an ARP table in every device on the network. This table keeps a mapping of all Internet Protocol addresses to the respective physical address.

The ARP table is **regularly cached** and can be updated in **two distinct situations**:

1. The client wishes to communicate with a device but lacks the mapping of its addresses, so it asks other devices on the network for it (**ARP Request**, followed by an **ARP Reply**).

2. A device decides to update (or refresh) the mapping of its addresses on every other device's ARP table. To do so, it can simply send gratuitous ARP messages, without the need of being preceded by an ARP Request. [1]

This can lead to a world of possible exploits, as ARP messages **do not require any form of authentication or validation**. A malicious user can forge ARP Messages and update the ARP Tables of all the devices on the network with fake mappings, which will then result in traffic being **routed through unwanted points**. This provides an easy mean for man-in-the-middle attacks and can **easily disrupt an entire network** if all the mappings on every ARP Table lead to dead ends. Another consequence of these attacks is something called an **ARP Storm**, which consists in both the attacker and the victim broadcasting ARP Messages, essentially fighting for a place in all the ARP tables.

**Forms of attack:** The **arpspoof** tool from the dsniff package [4] can be used to forge ARP messages.

**Why does this succeed?** No validation of ARP Messages. Clients inevitably update their ARP tables after receiving ARP messages, regardless of the legitimacy of the packet.

### 2.1.3 Network Layer

The network layer has a **strong relation with the well known Internet Protocol (IP)**. It is in this layer that the routing of the packets is determined, taking into account the destination and all the intermediate link nodes and networks. As opposed to the previous layer, this one deals with IP addresses (mapped using the Address Resolution Protocol explained before) to identify the traffic's source and destination, instead of physical addresses (MAC). [2]

Layer 3 is a less complex layer, as it only provides basic functions like:

- Packet routing, for outgoing packets.
- Packet capturing, for incoming packets.
- Limited error detection and diagnosis, for when traffic doesn't reach its destination.

We should also note that this layer **does not provide reliability of service**, seeing that this concerns the layers above, which will implement higher level transmission protocols such as TCP (Transmission Control Protocol).

With that in mind, attacks at this level can have a fairly significant impact **if security protocols are neglected**, since both ends have **no way of validating each other**, neither can they assess the legitimacy of all the networks their traffic went through before, reaching its destination. In order to avoid reliability issues when identifying and validating IP addresses, encryption and authentication technologies **can be implemented at this level**, as well as techniques to preserve data integrity. An example of one of those technologies is the **Internet Protocol Security (IPSec) [14]** - a standard architecture for security at the network layer, involving multiple protocols for authentication, data encryption and key management. [11].

#### 2.1.3.1 IP address spoofing

We can't mention network layer vulnerabilities without starting with IP spoofing. This technique has been around for years and it is still widely used, even in the layers above.

IP spoofing has a very simple concept. According to the standard [15], IP packets must include a header containing both the source and the destination IP address of the packet. IP spoofing is just the **creation of an IP packet with a forged header**, containing **different IP addresses** (destination addresses can also be spoofed, although it is rarely done).

By doing this, an attacker can, not only **hide his own IP address**, but also redirect the respective response traffic to a different source, thus making an attack **much harder to trace**.

Some uses of IP spoofing include **amplification attacks**, which essentially rely on a server response to be of much greater size than the request, in order to redirect the response traffic to the target/victim (examples of amplification attacks will be given ahead, in Section 2.1.5). IP spoofing is also used to make packets seem like they came from a private network (without going through a Network Address Translation (NAT) [16] router), or even the target's internal network, although this variation is already blocked by most firewalls on the market, as it uses a well known range of IP addresses [17]. This defense technique is called **ingress filtering**.

Upper layers may also offer their own means of protection against IP spoofing techniques, but after all, this is still a **very serious problem** in the intrusion detection scope.

**Forms of attack:** IP spoofing can be easily done with a tool like **Netfilter's iptables** [10], which is just a firewall management application. We can go further by utilizing more specific tools for this purpose, like **Nmap** [9].

**Why does this succeed?** Some networks still do not apply ingress filtering at their gateway and at this layer there are no transport protocols, so IP addresses are not validated (for example with a handshaking phase).

#### 2.1.3.2 Volumetric DoS attacks

Before heading into more specific forms of attacks, there is an important concept that must be addressed: **Denial of Service (DDoS) attacks**.

In short, DoS consists in an **attempt to make a service, a machine or an entire network unavailable**. A well known variation of Denial of Service attacks is Distributed Denial of Service (DDoS), which aims for the same purpose, although with multiple attack sources.

There are three types of DoS attacks that will be mentioned in this report: **Network Layer DoS**, **Transport Layer DoS** and **Application Layer DoS**. DoS attacks at the network layer are the most basic form of DoS, however it is the type of attack that **requires the most resources**. We call them **Volumetric Denial of Service attacks**. [18]

A volumetric DoS attack **targets the bandwidth of a network** by trying to exceed its capacity. The requirement for this attack to work is very straightforward: The attacker must have more bandwidth than the victim. This is usually achieved by having multiple machines flooding the target with irrelevant traffic, which in this case becomes a volumetric DDoS attack [19]. This is often exploited by using botnets (also known as zombie armies), which are a set of unaware infected machines (usually distributed all around the world), controlled by a central command-and-control server (C&C), that belongs to hackers or botnet buyers.

#### 2.1.3.3 Internet Control Message Protocol (ICMP) [21] based attacks

ICMP is one of the few protocols used in the network layer. It is mostly used by network devices for sending control and error messages, but can also be used by end-to-end machine applications for network reconnaissance (e.g. tools like ping, nmap, traceroute). The RFC 792 gives us some examples of ICMP usage:

*"ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route."* [21]

Many techniques try to take advantage of this simplistic protocol, however most of them are already extinct, since they are filtered by the majority of firewalls and operating systems.

**ICMP Flooding**

A basic form of a volumetric DoS attack in which the attacker uses ICMP traffic to flood the victim's network. It is commonly caused by simple applications like the "ping" utility, although it is modified to bypass the waiting for the victim's reply, in order to maximize outgoing traffic. As with all the volumetric DoS attacks, for this to work, the attacker's outgoing bandwidth must be greater than the victim's incoming bandwidth. [20]

**Forms of attack:** Any basic ICMP traffic generator, like the "ping" utility.

**Why does this succeed?** The attacker generates more traffic than the target can handle.

**"Ping of death"**

An almost extinct technique, that exploits the incapability of a machine (or operating system) to process packets that do not respect the IP standard [15]. Attackers used to create ICMP packets that exceeded the 65536 bytes limit, therefore crashing machines that did not handle it correctly [18].

**Forms of attack:** Any basic ICMP traffic generator, like the "ping" utility.

**Why does this succeed?** Operating systems were not prepared to handle packets out of the IP standard.

**Smurf attacks**

This is a more complex type of attack than the previous ones, although it is also practically extinct.

A smurf attack consists in exploiting a misconfiguration on a network that allowed ICMP packets (namely "pings") sent to the broadcast address, to be **delivered to every host on the network**. This would enable a single user to generate traffic to all the users inside a network, which would in some cases exceed the network's capacity. [18]

There is a variation of this attack also involving IP spoofing. The attacker would spoof the source IP of the packets, thus making all the hosts on a network reply to a different source address, which commonly **resulted in a volumetric DDoS attack**.

This technique is much less popular nowadays, as most operating systems do not reply to ICMP packets sent to the broadcast address.

**Forms of attack:** Any basic ICMP traffic generator, like the "ping" utility.

**Why does this succeed?** Operating systems were not configured to ignore ICMP packets sent to the broadcast address.

### 2.1.4 Transport Layer

The transport layer is responsible for **packaging data streams** and getting them **ready for transport**. There are multiple techniques that can be used for this purpose, some of them focusing on **reliability** and **flow control**, and others solely aiming **speed, data delivery** and **overhead reduction**. [2]

These techniques, often referred to as "transport protocols" can be considered a **double-edged sword**. While they provide effective means for delivering data, they are also **the source of many exploits and key vulnerabilities at this layer**.

*"Many transport protocols seem to have been implemented under the belief that they would be dealing with well-behaved communication from both the upper and the lower levels - a false assumption in the hostile world of the global public Internet."* [2]

#### 2.1.4.1 Quick overview of transport layer protocols

In order to better understand the risks of overlooking security at this layer, we cannot avoid giving a quick overview of the two main transport protocols: the **User Datagram Protocol (UDP) [22]** and the **Transmission Control Protocol (TCP) [23]**.

**User Datagram Protocol (UDP)**

- Connectionless, which means there is no handshaking phase in the communication.
- No packet identification mechanism.
- No packet retransmission, no delivering reliability. If a packet fails to deliver, it will be lost.
- Contains a checksum to preserve data integrity.
- Very efficient in processing traffic due to its simplicity.

**Transmission Control Protocol (TCP)**

- Handshaking phase at the beginning of the communication to initialize the connection.
- Each packet is numbered with an incremental sequence number that uniquely identifies the packet.
- Flow control and data retransmission techniques in case a packet fails to deliver.
- A protocol focused on reliability, at the cost of major overheads and consequently making it much slower than the alternative.

#### 2.1.4.2 TCP-based attacks

**TCP Syn Flood**

We've talked about volumetric DoS attacks (also known as network layer DoS) but now it's time to address a **more intelligent type of Denial of Service**. The main difference between DoS at the transport layer and DoS at the network layer is that, instead of relying only on bandwidth as an attack vector, transport layer DoS tries to achieve the same purpose of all DoS attacks (to take down a network or a service), with less resources, by **exploiting vulnerabilities in the transport layer protocols**.

**TCP Syn Flood** is a form of transport layer Denial of Service. In order to understand how this attack works, we must first go through the basics of a TCP connection. As mentioned before, TCP **is not** a stateless/connectionless protocol like UDP, instead it requires an initial **handshake** to establish a connection. That initial handshake is the target of a Syn Flood attack.

The handshake of the TCP is usually called **three-way handshake** and it consists in a series of messages exchanged between the client and the server [23]. Figure 7 shows the handshaking process of TCP.

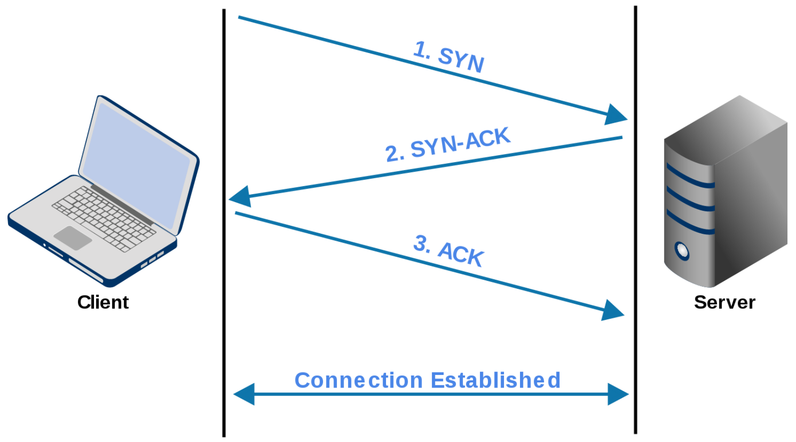


Figure 7: TCP's three-way handshake

1. The client wants to initiate a connection so it sends a **SYN packet** to the server.
2. The server acknowledges the client's message, therefore sending a **SYN-ACK packet** back to the client.
3. At last, the client will reply with a final **ACK packet**, hence establishing a successful connection with the server.

The issue here is that the server must maintain the data about the connection status, in order to correctly map the client's final ACK to the first SYN. The TCP Syn Flood attack exploits this condition by sending **only the first SYN packet multiple times** (commonly referred to as a "half-open connections"). The server will eventually **run out of memory** while trying to store large amounts of half-open connections, thus leading to a system crash.[18]

Usually, a malicious user will also spoof his IP address. This way, the server will reply with a SYN-ACK packet to a **different machine**, which in turn, will most likely **discard it**. [19]

**Forms of attack:** The **Metasploit Framework** offers easy ways of conducting Syn Flood attacks.

**Why does this succeed?** The initial handshake of a TCP connection has no way of validating IP addresses. Also, the server might not implement **Syn Cookies** (the main form of defense against this type of attack).

## TCP Sequence Number Prediction

TCP Sequence Number Prediction is a type of brute-force attack that allows an attacker to spoof an IP address using the TCP. It is a very time consuming attack and certainly not bandwidth friendly, it is also not practicable in most real scenarios due to the evolution of firewalls and routers.

Even so, we believe it is an important proof of concept, demonstrating that TCP is not a flawless solution and that IP-based authentication is never enough to authenticate a client or secure a host/network. Every TCP packet contains a header comprising of two 32-bit integers that will validate the identity of both the client and the server: the sequence number and the acknowledgment number. The following diagram (Figure 8) shows a perfectly normal TCP connection, including the sequence numbers and the acknowledgment numbers.

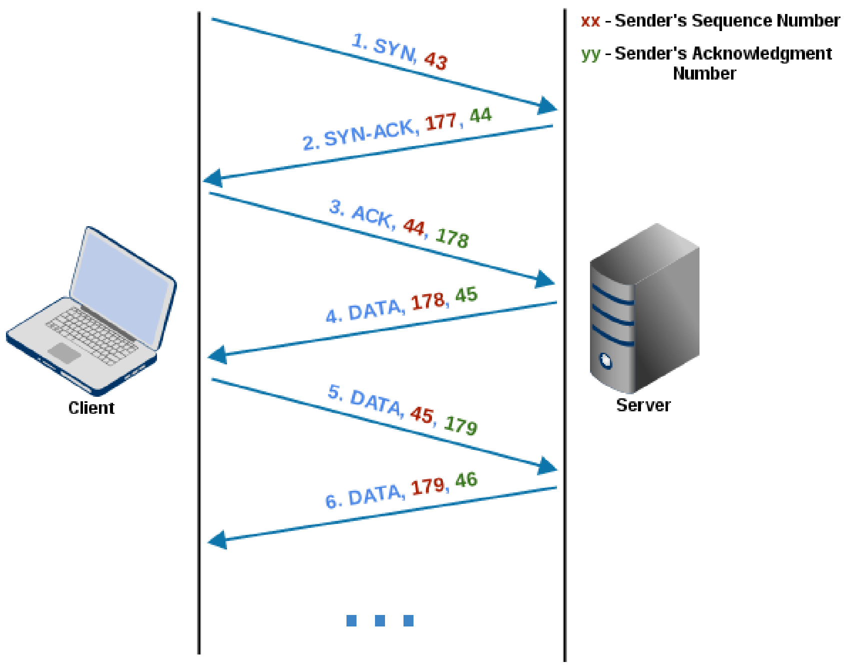


Figure 8: A normal TCP data exchange

By analyzing Figure 8, we realize that the client and server's identity is validated through the sequence and acknowledgment numbers. The sender's sequence number is always incremented by the receiver and sent in its next packet, as the acknowledgment number. As for the sender's acknowledgment number, it becomes the receiver's sequence number for the next packet. What if a third-party successfully brute-forces the initial sequence number? Let's take a look at the following diagram (Figure 9)

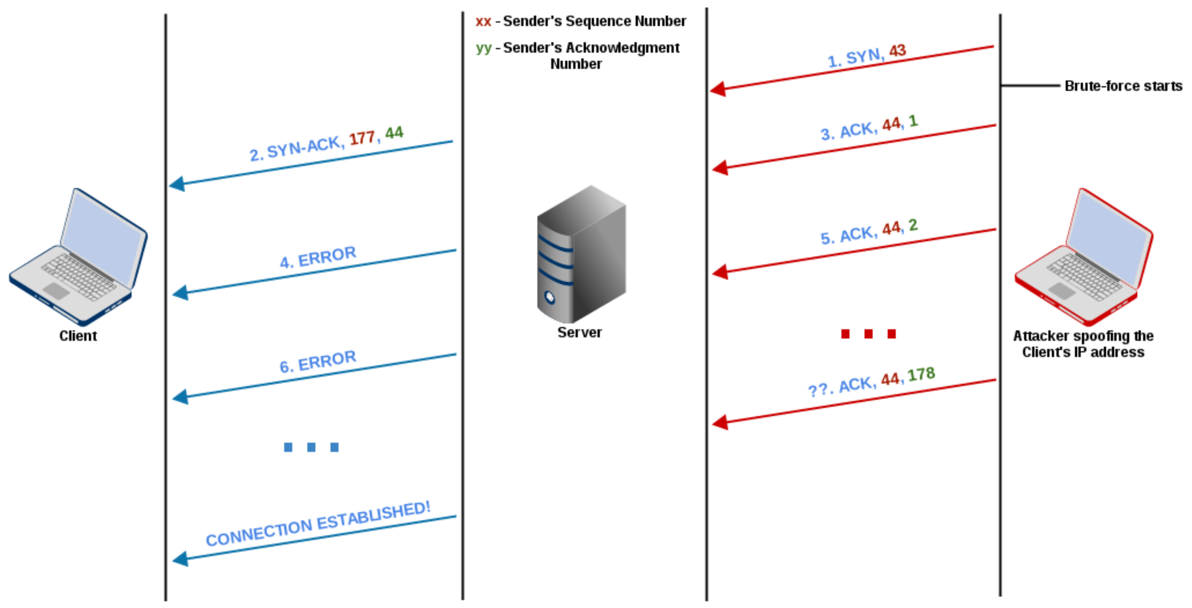


Figure 9: TCP sequence number being brute-forces

After several attempts, the attacker has successfully spoofed the client's IP address! It is most likely not an easy accomplishment since the brute-force has to be successful before the server terminates the connection attempt (the time before terminating the connection varies among different operating systems).

**Forms of attack:** Simple brute-force scripts with libraries able to manipulate network packets.

**Why does this succeed?** It is a natural vulnerability of the TCP, but definitely not an easy one to exploit. The usage of Transport Layer Security (TLS)/Secure Sockets Layer (SSL) encryption makes this attack almost impossible to replicate.

#### 2.1.4.3 UDP Vulnerabilities

As opposed to TCP, UDP is a stateless/connectionless protocol, which means that, although significantly faster, it lacks non-repudiation and authentication mechanisms. Most exploits involving UDP, revolve around IP spoofing.

Through IP spoofing, an attacker can execute **UDP replaying attacks**, that consist in capturing UDP packets (for example through a man-in-the-middle attack) and replaying them several times, which can trigger unpredictable reactions at the connection's endpoint.

Out of the many exploits that are made possible by UDP, **amplification attacks** are among the most popular ones and will be covered in Section 2.1.5

**Forms of attack:** The **Metasploit Framework** offers easy ways of exploiting UDP and its vulnerabilities. tcpreplay [72] provides means of replaying and editing previously captured UDP and TCP packets.

**Why does this succeed?** Due to the nature of UDP. Being a low overhead protocol without any handshaking mechanism or authentication, specifically made to deliver data as fast as possible, it's up to the applications built upon it to secure themselves from said attacks.

### 2.1.5 Application Layer

We have reached the final layer of the TCP/IP chain: the Application Layer (according to the OSI Model, this would correspond to the session, presentation and application layers all together). It is considered by many as an **abstraction layer** since it is not directly related to the network itself, but to the **applications utilizing it**, thus making this the highest level layer of the entire chain. [2]

Vulnerabilities at this layer are vast and highly dependent on the applications running inside the network and on user-oriented protocols comprised by them (examples of user-oriented protocols will be given ahead). Most threats at this layer come from **poor application designing issues**, allowing its resources to be used freely by unintended parties.

*"For the purposes of information security, the Application Layer can be considered the realm where user interaction is obtained and high-level functions operate above the network layer. These high level functions access the network from either a client or server perspective, with peer-based systems filling both functions simultaneously."* [2]

#### 2.1.5.1 Application Layer DoS attacks

The main difference between Application Layer DoS and the other types of DoS is that, at this layer a DoS will **aim to deny the usage of a service/application**, and not the host itself or the whole network. However, security breaches in applications may lead to crashes and/or resource depletion (e.g. Memory, CPU, Bandwidth), which can then have a much wider range of effect. [24]

The techniques used to perform application layer DoS attacks fully depend on the target service, but they mostly aim at **service and/or database misconfiguration** and **bugs within the program's code**, like buffer overflows, malformed data and the non-existing sanitization of user inputs.

One of the most popular types of application layer DoS attacks is the Slowloris attack, developed by Robert "RSnake" Hansen which aims to bring down an Hypertext Transfer Protocol (HTTP) [25] server by using a single device and minimal resources on the side of the attacker. In theory it is very similar to how the SYN Flood works in the transport layer, although with HTTP requests instead of SYN Requests. The purpose is to fill the connection pool of a web server with partial/uncompleted HTTP requests, therefore denying additional connection attempts.

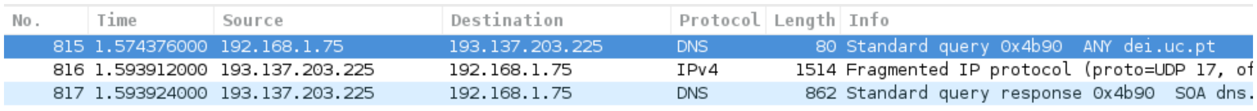
**Forms of attack: nmap** comes with a couple of scripts to perform application layer DoS attacks, including the Slowloris attack.

**Why does this succeed?** Insecure applications, security design flaws or lack of maintenance (and patching) by the sysadmins.

#### 2.1.5.2 Network Time Protocol (NTP) [26] & Domain Name System (DNS) [27] amplification attacks

Amplification attacks are one of many ways to exploit IP spoofing vulnerabilities on protocols such as UDP. In this attack, an attacker will spoof his IP address and send a small request to a service (usually UDP-based), that will **trigger a much larger response**. That response however, will be sent to the spoofed IP address. If conducted correctly, an attacker with few resources (e.g. low uplink bandwidth) can take advantage of a server's bandwidth and successfully **amplify his attack**, therefore performing a **DoS attack on his target**.

But what's the relation between amplification attacks and NTP or DNS? Both NTP and DNS can easily be queried for large responses using UDP! The following image (Figure 10) shows the basis of an amplification attack:



```
No.   Time         Source           Destination      Protocol Length Info
  815 1.574376000  192.168.1.75     193.137.203.225  DNS          80 Standard query 0x4b90  ANY dei.uc.pt
  816 1.593912000  193.137.203.225  192.168.1.75     IPv4       1514 Fragmented IP protocol (proto=UDP 17, of
  817 1.593924000  193.137.203.225  192.168.1.75     DNS         862 Standard query response 0x4b90  SOA dns.
```

Figure 10: Wireshark logs of a DNS query

In this example, we used the dig tool [75] to query the department's DNS server for information about the dei.uc.pt domain. We can see that the request has a length of 80 bytes, however the response sent by the server adds up to a total of 2376 bytes, which is approximately **30 times more than the original request**! With this in mind, an attacker with, for example 20Mbps of upload bandwidth, can amplify his attack and perform a 600Mbps DoS, initiated by the DNS server. NTP has a very similar functionality (e.g. the monlist query, among many others), although less popular nowadays, as NTP queries are usually restricted to the local host or network in recent NTP server implementations.

The following diagram (Figure 11) shows a visual representation of an amplification attack, with a spoofed IP address:

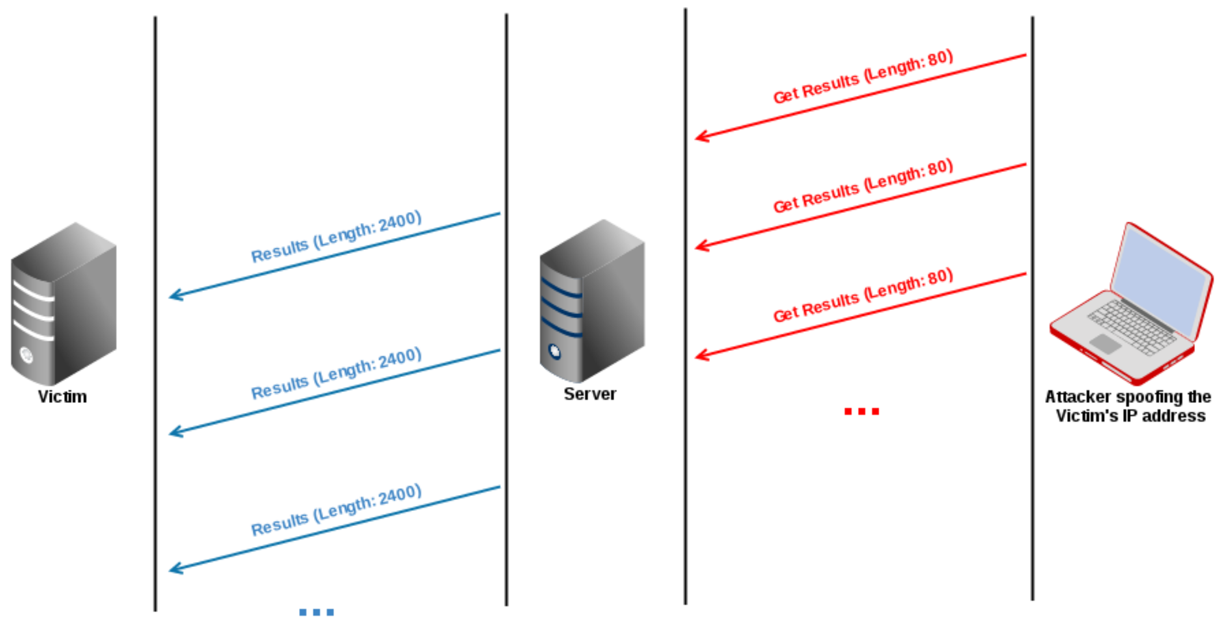


Figure 11: Amplification attack in practice

**Forms of attack:** The **Metasploit Framework** offers many ways of conducting amplification attacks, although it can be easily scripted using the appropriate libraries.

**Why does this succeed?** Due to the nature of UDP, but mainly because there are many open servers (both NTP and DNS) that do not perform rate limiting on queries from external sources.

#### 2.1.5.3 Dynamic Host Configuration Protocol (DHCP) [28] Starvation attack

DHCP is the protocol responsible for the initial configuration of new hosts in a network. It provides new clients with an IP address out of a predefined pool, a gateway, DNS servers, among many other possible configurations.

The DHCP server is responsible for this management and keeps a **track of the assigned addresses** mapped to the respective MAC addresses for a certain period of time, named the **DHCP lease**. Once the lease period is reached, the addresses associated with it are released.

The only way the DHCP server has of distinguishing a new client from an old one, is exactly through the client's MAC address. DHCP starvation consists of forcing a DHCP server to **use up the entire pool of addresses**, by continuously making DHCP requests while constantly generating new client MAC addresses (spoofing).

As soon as the DHCP server runs out of addresses, it **becomes unable to answer any more DHCP requests**, which will allow an attacker to set up a **rogue DHCP server** to answer requests from new clients. The consequences include the **possibility of man-in-the-middle attacks** (by providing the attacker's desired gateway on the DHCP offer),

**DNS hijacking** (by providing the attacker's desired rogue DNS servers on the DHCP offer), which can lead to many **phishing and scam attempts** among many other threats.

**Forms of attack:** **macchanger** provides an easy way of quickly spoofing and generating new MAC addresses, however the **Metasploit Framework** has built-in scripts that automatically conduct a DHCP starvation attack.

**Why does this succeed?** Lack of rate limiting of DHCP requests on the DHCP server. Also due to DHCP packets being sent via broadcast, so that any server (which includes rogue servers) can reply to DHCP requests without any form of authentication associated.

#### 2.1.5.4 Cross-site Scripting (XSS) attack

Cross-site Scripting is a vulnerability generally **found in web applications**. Essentially, an attacker takes advantage of a trusted web application to **send malicious code to a victim's browser**. Since most websites make use of JavaScript to deal with client-side content, browsers will execute the malicious code as part of the web application source. Nowadays, cross-site scripting has become one of the most commonly reported web application vulnerabilities together with SQL injection [73].

**Reflected or Non-Persistent** cross-site scripting is the most common type of XSS attack and can occur when a website does not properly sanitize a request. Usually, the attacker will use e-mail or any other form of digital messaging to reach its victim. The main idea of reflected XSS is to **blend in a script in a supposedly harmless URL**, leading to a trusted, but vulnerable website. Consider the following example of a website containing a search engine:

```
www.verysafewebsite.com/search?name=Pringles<script>...</script>
```

In this example, as soon as a victim clicks the URL, it will be redirected to *verysafewebsite.com* and immediately trigger an HTTP GET request - a search, as seen embedded in the URL. If the website somehow displays the victim's input (the request's *name* parameter) on the victim's browser (very common in search engines to indicate what was searched for), without properly sanitizing it, the script included in the request will be interpreted as normal JavaScript code and run without notice, since the browser will assume it is coming from a trusted source. This script will have access to the same resources as the website (due to a concept called the **same-origin policy**), which includes the victim's **cookies** (containing the victim's **session** and other sensitive information) and permission to freely edit the web page content and possibly make additional HTTP requests.

A much more problematic and devastating type of XSS attack is called **Persistent or Stored XSS**, in which an attacker will exploit a vulnerable part of a website that stores inputs indefinitely without checking them beforehand. The website will then store the attacker's input, containing the malicious script, and deliver it to every single victim that accesses that resource. The following example shows an HTTP POST, used to publish a message to a social network:

```
www.falsebook.com/post?content=Pringles<script>...</script>
```

Imagining said social network doesn't sanitize user inputs before displaying them, or even before storing them in a database, this would mean that the message content would be stored and displayed word for word to every user that requests that resource. This would cause the script embedded in the URL to be executed in several different machines, possibly causing a session hijacking in a large scale!

**Forms of attack:** Using automated scripts or even manually trying various forms of XSS, which can usually be found on *"cheat sheets"*, such as OWASP's XSS Prevention Cheat Sheet [74].

**Why does this succeed?** Due to web applications not properly sanitizing their user's inputs, through encoding and validation (e.g. transforming '<' into '&lt'). Additional ways of avoiding XSS include the usage of Content Security Policy (CSP), which can be included in the HTTP header response of a web page and will contain a list of directives that specify the sources from where scripts can be loaded.

#### 2.1.5.5 Structured Query Language (SQL) [29] Injection attack

SQL Injection is a code injection vulnerability targeted at applications that make use of a SQL-based database management server (DBMS). This attack, similarly to cross-site scripting, also relies on the incorrect handling of user inputs on the applications side. According to OWASP's Top 10 vulnerabilities, SQL injection together with other injection flaws occupied the first position in 2013 [73].

The risks of a successful SQL injection attack can go from minor to absolutely devastating, to the point of accessing, modifying and/or erasing all the data persisted on the database.

The following example consists of a common SQL query, that will search a specific user by his name in a certain database table and return it.

```
SELECT * FROM User WHERE name = 'Pringles'
```

It is a perfectly normal query and very useful when we want, for example, to verify if a user with such name actually exists in the database. The problem is not the query itself, but the applications using it. Suppose a website uses that query to validate a user's registration, and guarantee that no other user with the same name exists on the system. The user's name will be inserted in that query and it will most likely execute normally, however, let's take a look at the following query that is generated if a user inputs the name *'OR'1'='1*

```
SELECT * FROM User WHERE name = ''OR'1'='1'
```

After the quotes have been closed, the rest of the user's name (OR'1'='1) will be interpreted not as text, but as SQL! This query will result in a database search for a user named " (blank), which probably doesn't exist, but will also apply the OR operator, which in this case is always true. The results are simple: since the condition is invariably true, this query will return all the information about every user on the database. The severity of the consequences may escalate if we get creative on the choice of the username, for example the following query would wipe the entire user table:

```
SELECT * FROM User WHERE name = '';DROP TABLE 'User
```

**Forms of attack:** Tools like the **Metasploit Framework** and **sqlmap** [86] provide exhaustive SQL injection attempts and possible database takeover attempts for web applications.

**Why does this succeed?** Incorrect handling of user input parameters, no character escaping, incorrect management of database permissions, and in the case of programming languages like Java, the non-existing usage of prepared statements.

## 2.2 Detecting intrusions: Technologies and strategies

In this section we will go through some of the most well known and efficient techniques for successful network and application intrusion detection. First however, we should explore methods that will allow us - network engineers, to get in direct contact with said intrusions.

### 2.2.1 Techniques for achieving network visibility

A network, just like practically everything, is in continuous evolution. Network engineers are constantly adjusting and rearranging their network, in order to adapt to the newest hardware, software and security patterns. As to be expected, the threats landscape is no exception and is also in motion, sometimes moving even faster than the previous.

Rule of thumb: **Vulnerabilities exist and will always exist**. A network or application is **always** vulnerable. Security threats appear every day and attackers are using increasingly stealthier techniques and pursuing harder to monitor interfaces to fully capitalize on weak points. A good example of this matter is presented on Dave Shackleford's "Optimized Network Monitoring for Real-World":

*"Many of the newest and most malicious bot variants affecting organizations today are emulating the most common traffic types to hide the command and control (C&C) channels used to communicate with the bots. To avoid detection, most of the attackers' outbound traffic uses standard ports like 80 and 443, which typically cannot be inspected deeply due to the high volume of web traffic constantly traversing the network and the time latency such inspection would cause to daily operations."*[32]

Situations like this demand total insight of all the traffic traversing the network so that security experts can quickly identify and resolve the issues. This is called **Network Visibility** and is one of the most important concepts when building a robust intrusion detection system.

*"Poor network visibility obscures insight into your applications, infrastructure, and security."*[30]

The first step to achieve total network visibility is to **get access to the data**. We will now talk about (and compare) various strategies used to access the data flowing inside a network and their respective pros and cons.

#### 2.2.1.1 Port Mirroring

The first technique we will address is **port mirroring**, also referred to as Switch Port Analyzer (SPAN) by Cisco [33].

Port mirroring consists in the ability of a switch to **mirror the traffic of one or more of its ports** (or even from a whole VLAN) to a monitoring port (also called the SPAN port). In an ideal scenario, the monitoring port will be connecting the switch to a security monitoring device like an intrusion detection system or a forensic data recorder.

**Advantages**

- Inexpensive
- Included freely in **almost** every managed switch on the market
- Remotely configurable
- Great when monitoring non-critical networks (preferably lightly used as well)

**Disadvantages**

- **Packet dropping**: By applying port mirroring, we are concentrating inbound and/or outbound traffic of one or multiple ports **into a single port**, which will clearly lead to a bottleneck in situations where the traffic flowing in those ports exceeds the speed limit of the monitoring port. For example, if a network is under heavy usage, let's say 750Mb/s inbound and outbound traffic and all the switch's ports cap at 1Gb/s, when trying to monitor all the traffic in this situation, we will exceed the capacity of a switch port by 500Mb/s (750Mb/s inbound + 750Mb/s outbound = 1500Mb/s total >1Gb/s cap), which will eventually force the switch to drop packets. We should also note that there is usually no warnings/notifications when a switch starts dropping SPAN packets or delivering inaccurate timestamps. This is an intolerable behaviour seeing that, in order to achieve total network visibility, it is **absolutely critical** to see **every single packet** traversing the network.
- **Switch overloading**: Besides its functions as a network switch, port mirroring will cause the switch to also act as a packet duplicator, **thus increasing its CPU load**. Most switch manufacturers set port mirroring/SPAN as the **lowest priority task**, therefore in situations where the switch becomes overloaded, it will stop copying packets in order to restore its normal functioning state. This behaviour was confirmed by some

switch manufacturers, one of them being Cisco:
*"Cisco warns that the switch treats SPAN data with a lower priority than regular port-to-port data. In other words, if any resource under load must choose between passing normal traffic and SPAN data, the SPAN loses and the mirrored frames are arbitrarily discarded. This rule applies to preserving network traffic in any situation. (...) If there is not enough capacity for the remote SPAN traffic, the switch drops it."*[33]

- **Not a passive technology**: Port mirroring/SPAN **does not preserve** the physical and data-link layer's information, meaning that the respective layer errors will not be mirrored, thus making the troubleshooting of hardware problems close to impossible. [31] Some switches may also **disregard** the mirroring of **undersized/oversized packets**, **VLAN tags**, as well as packets with cyclic redundancy check (CRC). Lastly, port mirroring will also change the interpacket gap of ethernet frames and might not preserve the original order of the traffic.[32]

- **Vulnerable to changes**: Having to reconfigure a switch to set up port mirroring might not be the best idea, specially in a production environment, as this may accidentally expose or bring down the network.

- **Not a standard**: Port mirroring/SPAN is not a standard among all the switches. Some older equipments may not even support it at all. Furthermore, this functionality is highly related to the equipment itself and its configurations will vary from manufacturer to manufacturer. In fact, some switches may even randomly reassign monitoring ports to other purposes, without any form of notification. [30] Having all these **hardware/manufacturer dependencies** for such critical functionalities in a constantly evolving network is most likely not what we want.

- Most switches cannot efficiently support more than one or two monitor ports, which in many cases may not be enough to gather data from all the subnets in a large network and send it to different security monitoring devices.

**Remote Switch Port Analyzer (RSPAN)**

RSPAN is an extension of SPAN that can be used to collect traffic from switch ports **all over the network** and route that information through a single VLAN (dedicated to this purpose) to a centralized security monitoring device.

#### 2.2.1.2 Network taps

A network tap is a hardware device that can be placed between two communicating network devices in order to physically mirror the flowing traffic between them and send it to a security monitoring device (through a monitoring port).

**Advantages**

- Aims to provide **100% network visibility** by physically mirroring all the link traffic.
*"Optical taps for fiber links use optical splitters to divert part of the light from the link*

*to a monitor port…Taps for copper links perform a similar function electronically."*[30]

- Creates exact copies of the traffic regardless of how busy the link is, including **physical layer and data-link layer errors**, as well as keeping the original ethernet frames interpacket gaps.
- It is a **passive technology**, therefore it doesn't affect the traffic in any way.
- Even though it is a hardware device, it cannot be considered a critical failure point. Network taps are not an addressable device (no physical or logical address) thus rendering it invulnerable to the upper layer security threats.
- Usually network taps will keep the traffic flowing **even if they lose power**.

**Disadvantages**

- Very expensive alternative, specially when used in large networks where various teams are capturing traffic in specific locations and sending it to their own monitoring devices.[32]

### Aggregation taps

Aggregation taps are a specific type of network taps that aggregate both the inbound and outbound traffic traversing between two networks and send it to a single monitoring port in a security monitoring device. There is a clear disadvantage regarding this technique (quite similar to a SPAN disadvantage): **Packets will begin dropping** as soon as the inbound plus outbound traffic leaving the tap, exceed the monitoring system's (single) NIC speed limit (bottleneck).

In order to mitigate this issue, aggregation taps have a **built-in buffer** that caches the analyzing output when the network is under heavy usage, thus making it less likely to drop packets. The buffer will then start sending the cached packets in more opportune times (when the network utilization drops to a point where both cached traffic plus ongoing traffic can both be sent simultaneously). However, if the bursts of activity also exceed the buffer's capacity, then **the problem remains unsolved**.

In conclusion, aggregation taps are a nice and cheap alternative when the monitoring device is equipped with a standard single NIC (e.g. laptops). Nonetheless, to guarantee that 100% of the traffic successfully reaches the intrusion detection system, we might need a more complex type of network taps: **Full-duplex taps**.

### Full-duplex taps

Full-duplex taps intend to cover the faults of aggregation taps, as it is only method we've seen until now that **guarantees 100% network visibility**. A full-duplex network tap will send inbound and outbound traffic through different ports thus solving the bottleneck issue of aggregation taps. However, in order for this technique to be effective, the security monitoring device must be equipped with a dual-receive NIC, which is most definitely not the cheapest alternative.

#### 2.2.1.3 Software-level traffic mirroring

Although this method is not usually associated with port mirroring techniques, we still decided to mention it in this report, seeing that, in spite of all the disadvantages that it brings, it can still be quite useful in certain situations.

Software-level traffic mirroring consists in utilizing the **operating system functions** to duplicate the traffic on one interface to another. It can be accomplished in many different ways, depending on the host's operating system. On Linux for example, **netfilter's iptables** includes a module developed exactly for this purpose - **TEE**, which will automatically duplicate and route the mirrored traffic to a different gateway (the security monitoring device).

**Advantages**

- Inexpensive, included in almost every operating system.
- Appropriate for preliminary non-critical analysis on small to medium networks.
- If the device does not support port mirroring (SPAN), this might be the only alternative before acquiring network taps!

**Disadvantages**

- Same problem as switch port mirroring and aggregation taps: **the interface bottleneck**, when aggregating inbound and outbound traffic.
- **CPU Overload**: The host device will have to deal with routing the traffic, as well as duplicating it and sending it through a different interface. This will highly increase the CPU usage and temperature and might lead to crashing the operating system.
- Tools at the operating system level like iptables will only handle traffic correspondent to higher layers of the TCP/IP chain. In this case, physical layer and data-link layer traffic would not be mirrored.
- It is highly dependant on the operating system's kernel and on the tools used for the purpose. A simple kernel update may ruin the whole scenario.

#### 2.2.1.4 Comparison of traffic capture techniques

We've addressed the most popular and effective techniques for achieving network visibility, it is time to summarize what we learned by placing all of them side by side and compare the core attributes of each one:

| Attribute | Port-mirroring (SPAN/RSPAN) | Aggregation taps | Full-duplex taps | Software-level traffic mirroring |
|---|---|---|---|---|
| **Passivity** | No | Yes | Yes | No |
| **Full physical and data-link layer mirroring (with errors)** | No | Yes | Yes | No |
| **Packet dropping <50% network utilization** | No | No | No | No |
| **Packet dropping >50% network utilization** | Yes | Yes | No | Yes |
| **CPU-friendly** | No | Yes | Yes | No |
| **Plug & Play** | No | Yes | Yes | Yes |
| **Scalability** | Hard | Easy | Easy | Hard |
| **Price (regarding scalability)** | Inexpensive | Expensive | Expensive | Inexpensive |
| **Infrastructure-dependent** | Yes | No | No | Yes |

Table 1: Comparison of traffic capture techniques.

The previous table shows us that port mirroring is a perfectly allowable technique for small enterprises, or small subnetworks, as it is inexpensive (included in almost every switch in the market) and only shows its serious flaws when dealing with huge amounts of traffic. The same goes for software-level traffic mirroring, however it's probably wiser to use it only in testing environments. For large networks, whose bandwidth is prone to surpass 50% utilization, a full-duplex network tap is undoubtedly the best option, although also being far more expensive than the others.

### 2.2.2 Detection strategies

After picking the optimal technique to achieve maximum network visibility, we must start thinking about the strategy to adopt in order to efficiently find the actual intrusions scrambled **among all the legitimate traffic**.

We have to keep in mind that most applications and systems are developed without considering network security. They may be secure in their own ways, in a controlled environment, however when deployed into a real world network, things might not stay the same as the concept **range of security goes from local to global**. These systems and applications are usually the main target of attackers, the weakest links in a network.

The first question one must ask is: What is an intrusion? An intrusion is defined as a group of actions that attempt to **compromise a resource** in terms of **availability**, **integrity** or **confidentiality** [34]. Intrusion detection consists in identifying said actions and works under the assumption that attacks (and attackers) produce noticeably different behaviours than those of legitimate users.

The traffic patterns generated by intruders (usually patterns that deviate from the **expected network behaviour**) can be considered **anomalies** [35]. For example, if a machine hosting a Simple Mail Transfer Protocol (SMTP) [36] server is constantly under heavy load, it probably means the machine is infected and/or sending out/relaying unauthorized messages to various recipients (often called Spam). This generates abnormal network usage patterns that should trigger the (anomaly-based) intrusion detection system. But what can be considered "abnormal"? Even better, what is "normal"? In order to address anomalies, we must define **normality**.

An **anomaly-based intrusion detection system** usually consists of two modules: **Modeling** and **Detection**. The modeling module will define the concept of normality by analyzing the network and training the system to achieve a **normality model**. The detection module will use the normality model to detect network anomalies, by calculating the deviation of the arriving packets to the normality model. If this deviation reaches a certain threshold, then we are facing a network anomaly.

Besides anomaly-based, there's one more type of intrusion detection systems that we will address: **Misuse-based**. While anomaly-based attempts to find "anomalous" patterns in traffic, by comparing it to a normality model, misuse-based intrusion detection systems only search and identify **already known intrusions**, by comparing the traffic to an "abnormality model" (anything it doesn't know, is considered normal). This basically means that a misuse-based intrusion detection system **cannot** detect new intrusions, **it doesn't evolve on its own** and needs to be constantly updated, which may take considerable amounts of time and work.

We will now go through several anomaly-based and misuse-based strategies to detect network anomalies, as well as their respective advantages and disadvantages. We've decided to follow the classification scheme defined in [35], even though the authors make it clear that defining such scheme is not a straightforward task *"primarily because there is substantial overlap among the methods used"*.

#### 2.2.2.1 Statistical

Statistical intrusion detection consists in a strategy that revolves around a statistical model. There must exist a statistical model that corresponds to the **normal network operations**. A certain pattern is considered an anomaly when, after going through a **statistical inference test**, has a low probability to be generated by the stochastic model assumed, therefore we can conclude that it is irrelevant to the normal network functioning.

**Advantages**

- Easy to understand, it's basically a method based on deviations. If something deviates too much (i.e. reaches a certain threshold that can be defined by the administrator) from normality, it is considered an anomaly.
- Can usually achieve a solid normality model because it monitors the network for long periods of time.
- Does not require any previous training since it learns the network behaviour through observing it.
- Provides accurate notifications in case of malicious intrusion attempts. The accuracy of these notifications is often related to the threshold defined.

**Disadvantages**

- Does not do well in networks where anomalies or irrelevant traffic **highly exceed the normal traffic**, because it will generate an **incorrect** normality model (i.e. it will assume that the normality is abnormality).
- The intrusion detection highly depends on a solid normality model, which may take too long to be generated in complex networks, thus inducing an undesired delay before starting to detect intrusions.
- **Vulnerable to being trained by an attacker**. If an attacker knows that such system will be implemented, he can easily mold the network in such a way that his traffic will be considered normal by the normality model generated.
- Not all behaviours can be generated through stochastic models.
- Very hard to define all the metrics or variables in order to find an acceptable **balance** between false positives and false negatives.

#### 2.2.2.2 Classification-based

This strategy consists in observing a pattern and attempting to **classify** it following a **predefined training set**. The arriving object will be judged according to some of its attributes and will then be inserted into the **appropriate labeled class** (i.e. a group of similar objects). The number of classes may vary from method to method but in the end they are all divided by a **border** (most likely not linear), which intends to separate anomalies from normal traffic.

**Advantages**

- High detection rates for known attacks.
- Easily adaptable, we just need to tweak the training set in order to update its classification methods.

**Disadvantages**

- Depends on a predefined training set, which itself depends on assumptions made by the people that developed it.
- **Cannot detect unknown intrusions** until the training set is updated and the system is retrained.
- Heavy consume of resources.

#### 2.2.2.3 Clustering & outlier-based

Clustering & outlier-based strategies are quite similar to classification-based. Instead of relying on a predefined training set to classify the objects, this strategy intends to **group instances into clusters**, where every object inside a cluster is somewhat **related** to every other on the same cluster. In the end, bigger clusters have **higher chances of happening**.

This strategy also accounts for outliers, which are slightly similar to some cluster, but are highly unlikely of happening. Then again, this strategy assumes that anomalies are uncommon events in a network, not the other way around.

**Advantages**

- Can easily and efficiently detect rare/isolated attacks (outliers).
- High intrusion detection rate in networks with low granularity.
- Stable performance when compared to the previous strategies.
- Reduced computational complexity in networks with low granularity.

**Disadvantages**

- Once again, this strategy crumbles if the anomalies **exceed the normal network traffic**.
- It works under the assumption that bigger clusters correspond to normal activities, and smaller clusters (and outliers) correspond to anomalies, which sometimes might not be accurate.
- Heavily dependent on the metrics and parameters defined. *"Use of an inappropriate proximity measure affects the detection rate negatively"* [35]
- **Highly complex and resource intensive** when combining a high number of clusters with a high number of outliers (very granular network).

#### 2.2.2.4 Soft computing

Soft computing strategies cover many different approaches. We will briefly go through each one, once again following the division proposed by [35].

- **Genetic Algorithm:** As the name implies, this strategy consists in an attempt to transform the problem into a genetic problem, in a way that in can be interpreted by a framework that uses *"chromosome like data structures"*. It is a search heuristic based on evolutionary ideas.

- **Artificial Neural Network (ANN):** ANN is a strategy based on the human brain and the way it functions (obviously under the assumption that it functions differently from a digital machine). Our brain uses neurons in order to perform massive "computations" in relatively short amounts of time. This technique tries to mimic that behaviour, by training the neural network and adapting the interconnection strength of neurons to achieve a desired goal.

- **Fuzzy Set:** Fuzzy systems make use of fuzzy rules (often very vague rules) and fuzzy inputs to determine the probability of occurrence of an event, either specific or general, based on the rules defined.

- **Rough Set:** This strategy is based purely on approximations, this is, upper and lower approximations to regular sets. First described by Zdzistaw Pawlak in 1981: *"The key to the presented approach is provided by the exact mathematical formulation of the concept of approximative (rough) equality of sets in a given approximation space; an approximation space is understood as a pair (U, R), where U is a certain set called universe, and $R \subset U \times U$ is an indiscernibility relation."* [37] This is an often simple technique that requires small sized training datasets.

- **Ant Colony:** This is a very well known probabilistic technique for solving problems that can be reduced to finding optimal paths through graphs. It is based on the behaviour of a colony of ants finding their way to sources of food.

- **Artificial Immune System:** This is yet another strategy based on human behaviour. This one intends to mimic human's immune system by making use of learning and memory to detect anomalies. For example, if a human catches a certain disease (anomaly), the impact of the next diseases of the same type will be much lower.

**Advantages**

- Feedback from the environment is not needed when categorizing features.

- ANNs can train and test instances incrementally using various algorithms.

- Rough sets are simple yet very useful in detecting inconsistencies in the dataset and generating non-redundant rule sets.

- Can be used effectively for data clustering, feature extraction and similarity detection.

**Disadvantages**

- During neural network training, **overfitting may occur**. This means that the neural network will have a small error on a specific training set, however, when presented with new data it won't be able to generalize, thus presenting a larger error.
- Generally most of these techniques **do not scale well**.
- When it comes to fuzzy approaches, the definition and updating of rules at runtime is a difficult task.
- **Unreliable techniques** when there isn't enough normal network traffic to train them.

##### 2.2.2.5 Knowledge-based

This strategy consists in matching events (i.e. packets) to **predefined rules** or patterns of attack. These rules are built by trying to generalize **instances of known attacks** and are then matched against the flowing traffic, by the intrusion detection system (which at this point has absolutely no information besides the rules themselves).

The more rules it matches, the more confidence will the intrusion detection system have when matching subsequent instances. We should also note that a single instance **may fire various rules**!

**Advantages**

- High detection rate when the ruleset provided is considerable and covers most attacks, as well as normal instances.
- Very flexible, new rules can easily be added/removed.
- Robust and very scalable.

**Disadvantages**

- **Cannot detect unknown attacks**.
- Fully dependent on a predefined ruleset to actually detect anomalies.
- Hard to produce reliable rules and update them.
- Can generate a large amount of false alarms because the system had no previous training about the normal traffic in the network.
- Choosing the right rules/ruleset can be a very laborious task.

#### 2.2.2.6 Combination learner

Combination learner methods can be split in two distinct categories:

**Ensemble-based methods** These methods consist in combining several individual classifiers in order to obtain a single overall classifier that outperforms every other. It is quite similar to the concept of N-Version programming, there are multiple classifiers that judge the same instance and each one provides a result, which is then used to reach a final decision, based on the votes of every classifier. There is also the concept of multiple classifier systems (MCS) that basically introduces various stages of classifiers (at each stage a classifier may decide to send the instance to the next stage, instead of deciding the result in the current stage) that gradually increment in complexity and difficulty and are trained on more difficult cases.

**Advantages**

- Multiple classifier systems are a good alternative when the individual classifiers are weak.
- Scalable for large datasets.
- Easily configurable (due to comprehensive parameters)

**Disadvantages**

- Hard to obtain real-time performance.
- Selecting the appropriate classifier is not an easy task.
- Can become slow for large datasets.

**Hybrid methods** As explained previously, there are two types of intrusion detection systems: Anomaly-based and misuse-based. We also went through the strengths and weaknesses of each one and we can conclude that both types are **practically complementary**. Hybrid methods are an **attempt to combine both**!

**Advantages**

- Can detect **both known and unknown attacks** by exploiting misuse-based and anomaly-based detection.
- Depending on the approach adopted, it can inherit the respective advantages.

**Disadvantages**

- High computational cost
- The misuse-based part still remains a laborious task.
- This still quite an **unexplored area**, thus there is much room for improvement and no “standard” has been set.

### 2.2.3 Comparison of detection strategies

As we reach the end of this section, we realize that every strategy has its own strengths and weaknesses and there is no single strategy that outclasses every other. The decision on which strategy to adopt must take into account the problems we're addressing and the whole network environment and its conditions. The following table gives us a summary of the common features we can find on these strategies.

| Attribute | Statistical | Classification | Clustering & outlier | Soft computing | Knowledge | Ensemble | Hybrid |
|---|---|---|---|---|---|---|---|
| **Detects unknown threats (anomaly-based)** | Yes | No | Yes | Yes | No | No | Yes |
| **Detects known threats (misuse-based)** | Yes* | Yes* | Yes* | Yes* | Yes | Yes* | Yes |
| **Requires previous training** | No | No | No | Yes | No | No | Maybe |
| **Scalability** | Hard | Medium | Hard | Hard | Easy | Medium | Hard |
| **Complexity** ** | High | Medium | High | High | Medium | Medium | High |
| **Reliability** *** | Low | Medium | High | Medium | Low | Medium | Low |

* It *can* detect known attacks, but not as reliably as misuse-based strategies.
** Complexity is related to the implementation and maintenance of such strategy.
*** Reliability is related to the detection of false positives and false negatives.

Table 2: Comparison of detection strategies.

### 2.2.4 Types of intrusion detection systems

Intrusion detection can be performed at two levels: at the network level, with a **Network Intrusion Detection System (NIDS)** or at the host level, with a **Host-based Intrusion Detection System (HIDS)**.

While a NIDS reads and analyzes all the incoming and outgoing (local) traffic (every single packet or flow) **trying to find anomalous patterns**, a HIDS monitors one or more specific hosts, aiming to **detect suspicious internal activity at the operating system level**, in situations where such activity is not justifiable, for example an e-mail client that deliberately starts reading/altering/exposing protected system resources should be considered abnormal and immediately detected by an HIDS. It can achieve such conclusions, though techniques like log monitoring and file integrity checking, while also analyzing the incoming and outgoing traffic of each of the host's interfaces.

## 2.3 Preventing intrusions: Technologies and strategies

Intrusion prevention is as important as intrusion detection. It consists in the reaction to a trigger issued by the intrusion detection system. Without intrusion prevention, intrusions would be detected, but not stopped, attackers would be identified, but not blocked.

We will now talk about the most common strategies to prevent intrusions and in the end, we will notice that every strategy converges to the same purpose: blocking the attack by any means possible.

### 2.3.1 Network intrusion prevention

Before starting to talk about network intrusion prevention, there's one important concept that we must introduce: **inline** and **out-of-band** IDS/IPS. The main difference between the two is that an inline IDS/IPS will be **placed directly between the source and the destination of the traffic**, this means that, with an inline system, every communication must first go through the IDS/IPS, before reaching its destination. This will of course have impact on the average network latency, since every packet must be processed and analyzed before being routed. It also introduces a single point of failure in the communication: **if the IDS/IPS is down, communication is impossible!**

As for an **out-of-band IDS/IPS**, it will use an inline hardware device (e.g. network tap, switch) to **capture**, **duplicate** the traffic and **route** the mirrored traffic to the ID-S/IPS. This method **should not affect the network latency**, however, by delaying the intrusion/prevention, we are allowing potentially malicious packets to reach their destination without being analyzed and filtered first. When opting for the optimal IDS/IPS placement method, there is a clear trade-off between security and availability/reliability.

At the network level, intrusion prevention is relatively simple to interpret. When an intrusion is detected by the IDS, the IPS will have multiple options available: Drop the packet (only possible if placed inline, between the source and destination), interact with network firewalls to block the attacker, reset the TCP session, etc.

### 2.3.2 Host intrusion prevention

At the host level, intrusion prevention is completely different. Intrusions are detected not only through the host network interface (thus detecting packets that reached or leave the endpoint), but also through communication between applications, between processes and interactions with the operating system's kernel.

Prevention can be achieved in multiple ways: killing processes, terminating a user's session, banning a user (even by using authentication mechanisms like LDAP/Kerberos), altering file/folder permissions or through active response, which consists of one or many predefined scripts that will run once certain intrusion exceeds a threshold of seriousness (this allows the administrator to configure his own prevention techniques for each type of intrusion).

## 2.4 Exploring intrusion detection/prevention systems

In the previous sections, we explored the theory behind intrusion detection and prevention systems by analyzing every strategy and techniques they can implement. In this section, we will partially **explore the actual market** and link what we've learned before, to the current most highly regarded products out there.

One of the most important criteria for selecting an IDS/IPS is the update frequency, as this partially dictates the reliability of such product. That said, every product mentioned below, either free or commercial, was **updated at least once** in the current year.

### 2.4.1 Free & open-source products

#### Bro [48] (NIDS)

Bro is an open-source NIDS and network analysis framework that passively monitors all the network traffic, looking for suspicious activity. It implements both **anomaly-based** and **misuse-based** intrusion detection strategies (mostly **statistical** and **knowledge-based** respectively), thus making it an **hybrid intrusion detection system** [50].

It can only run in UNIX-based systems and claims to perform real-time analysis, support integration with external applications, support comprehensive IPv6, as well as perform sanity checks, port-independent analysis of many application layer protocols (including analysis of file content with MD5/SHA1 computation for fingerprinting), tunnel detection and analysis (which involves tunnel decapsulation), support *"IDS-style pattern matching"* and output well-structured logging for offline analysis and forensics [49].

#### Snort [46] (NIDS)

Snort is an open-source **misuse-based** (implementing a **knowledge-based strategy**) NIDS and consists of 5 components [51]: the Packet Decoder, which takes the packets and prepares them for the preprocessors, the Preprocessor(s), which modifies packets before they reach the detection engine (for example, it may consist of Snort plugins to normalize protocol headers), the Detection Engine, which is the core of Snort and matches the packets to the available rules, the Logging/Alerting System to generate alerts and log messages and the Output Module to generate the final output of the program, including packet dumps.

It supports most operating systems (UNIX-based, Windows and MacOS) and claims to perform protocol analysis, pattern matching, detect probes, buffer overflows, stealth port scans, OS fingerprinting attempts, etc. Snort comes with no rules (i.e patterns) by default, however there are multiple rule sources available that can complement each other, such as community rules (a set of rules maintained by the Snort community) and the Talos rules (either non-free or registered users only with a 1 month update delay).

### Suricata [52] (NIDS)

Suricata is an open-source intrusion detection and prevention system that implements mostly **misuse-based** strategies (**knowledge-based**) but also to some extent **anomaly-based strategies**, through **statistical** models, which makes it a **hybrid** intrusion detection and prevention system.

It is viewed by many as a direct competitor to Snort due to its multi-threading capabilities, GPU acceleration and additional anomaly-based detection techniques and protocol identification. Although Suricata uses a different architecture from Snort, it claims to be compatible with all the Snort rules format. It also claims to be highly scalable and to automatically produce statistics on network usage (e.g. packet loss ratio) [53] and can be configured to log more than packets and alerts (for example logging SSL certificates and HTTP requests).

### OSSEC [54] (HIDS)

OSSEC is an open-source HIDS, implementing both **anomaly-based** and **misuse-based** intrusion detection strategies. It is a highly scalable multi-plataform system, supported on most operating systems (Linux, OpenBSD, FreeBSD, MacOS, Solaris and Windows).

It claims to have a *"powerful correlation and analysis engine integrating log analysis"* and to perform file integrity checking, rootkit detection, centralized policy enforcement, Windows registry monitoring, among many others. All this integrated with a real-time alerting and active response engine, as well as a centralized server for logging and agent management [55] [56].

### Samhain [57] (HIDS)

Samhain is an open-source HIDS and probably the only real competitor of OSSEC in the open-source field. It implements both **anomaly-based** and **misuse-based** intrusion detection strategies and runs a client-server architecture, where Samhain clients are the HIDS themselves and the Samhain server is responsible for collecting, analyzing and storing the data into a relational database [58].

It is a mulit-platform application for POSIX systems and claims to perform file integrity checking, rootkit detection, port monitoring, detection of rogue SUID executables, hidden processes, as well as log file monitoring/analysis.

### 2.4.2 Commercial products

The following list of products is based on the company relevancy and product market share [59], the order is irrelevant.

#### Cisco FirePOWER [60] (NIDS)

Cisco FirePOWER is an IDS/IPS series that consists of a fully developed hardware structure that integrates a firewall plus a *"Next-Generation Intrusion Prevention System (NGIPS)"*.

Unfortunately, as with almost every commercial product out there, this one also doesn't provide much documentation about the internal functioning of the system neither does it provide any detail on its internal architecture, however, by analyzing its features we can infer that it implements both **anomaly-based** and **misuse-based** intrusion detection strategies.

It is a highly expensive product (the upper bound can reach as high as 830000€as of December 2015) [62] and claims to provide integrated real-time contextual awareness, full-stack visibility, and intelligent security automation [61].

#### IBM Threat Protection System (TPS) [63] (NIDS)

IBM Security's Threat Protection System is an IDS/IPS that integrates various IBM's services and includes more than 400 security tools from over 100 vendors. Its main focus is stopping advanced threats.

Once again, the documentation for this product is very limited, however it claims to prevent attacks in real time, detect advanced threats using data analytics and correlating massive sets of data in real time. It also claims to detect stealthy threats not only **using signatures**, but also based on their behaviour by using **anomaly-based detection strategies** [64].

#### McAfee Advanced Threat Defense (ATD) [65] (NIDS & HIDS)

McAfee Advanced Threat Defense is a system, or should we say a complete architecture, that focuses on *"advanced detection for stealthy, zero-day malware"*.

By using various **anomaly-based and misuse-based strategies**, as well as real-time emulation defenses (sandboxing), it claims to perform zero-day threat protection (i.e. unknown vulnerabilities) as well as known threats protection, detection of sandbox evasion techniques and detection of code obfuscation [67].

Besides being used in McAfee's IDS/IPS products, the Advanced Threat System is also implemented in McAfee's HIDS [66], a scalable solution with a centralized control platform, which provides endpoint protection and is supported in most operating systems.

**Juniper Sky Advanced Threat Prevention (SATP) [68] (NIDS & HIDS)**

Juniper Sky Advanced Threat Prevention is a cloud-based service, integrated with Juniper's SRX series firewall products [69] whose main focus is to deliver a dynamic cloud-based advanced malware protection.

This is not an actual product, but a complete service/architecture that can easily be integrated into an IDS/IPS and/or an HIDS architecture. It implements many different **anomaly-based and misuse-based strategies** and claims to prevent sophisticated zero-day attacks and many other unknown threats. It also uses a sandbox system to *"trick malware into activating and self-identifying"*, as well as many **machine-learning algorithms** in order to adapt to the *"ever-changing landscape"*.

### 2.4.3 Comparison of intrusion detection/prevention systems

Table 3 exposes the common features of the addressed products. Although being closed-source and expensive, the commercial products show clear advantages over the open-source products, since they all include both misuse-based and anomaly-based techniques, thus detecting both known and unknown threats.

| Product | NIDS | HIDS | Anomaly-based | Misuse-based | Multi-platform |
|---|---|---|---|---|---|
| **Bro** | Yes | No | Yes | Yes | No |
| **Snort** | Yes | No | No | Yes | Yes |
| **Suricata** | Yes | No | Yes* | Yes | Yes |
| **OSSEC** | No | Yes | Yes | Yes | Yes |
| **Samhain** | No | Yes | Yes | Yes | Yes |
| **Cisco FirePOWER** | Yes | No | Yes | Yes | N/A |
| **IBM TPS** | Yes | No | Yes | Yes | N/A |
| **McAfee ATD** | Yes | Yes | Yes | Yes | Yes |
| **Juniper SATP** | Yes | Yes | Yes | Yes | Yes |

* Only partially (to some extent) [77].

Table 3: Comparison of intrusion detection/prevention systems.

## 3 Preliminary Work

In this section we will explore the communication's infrastructure of the Department of Informatics Engineering. We decided that, before starting right away picking a network intrusion detection system, we needed to get a good overview on the network itself, the infrastructure that carries it, the clients using it and most importantly the daily issues and challenges it faces.

First and foremost, we've designed a minimalist diagram of the network infrastructure in order to easily identify and understand the core points that need to be addressed when deploying an intrusion detection system. The diagram can be viewed in Figure 12 and will be explained thoroughly ahead.

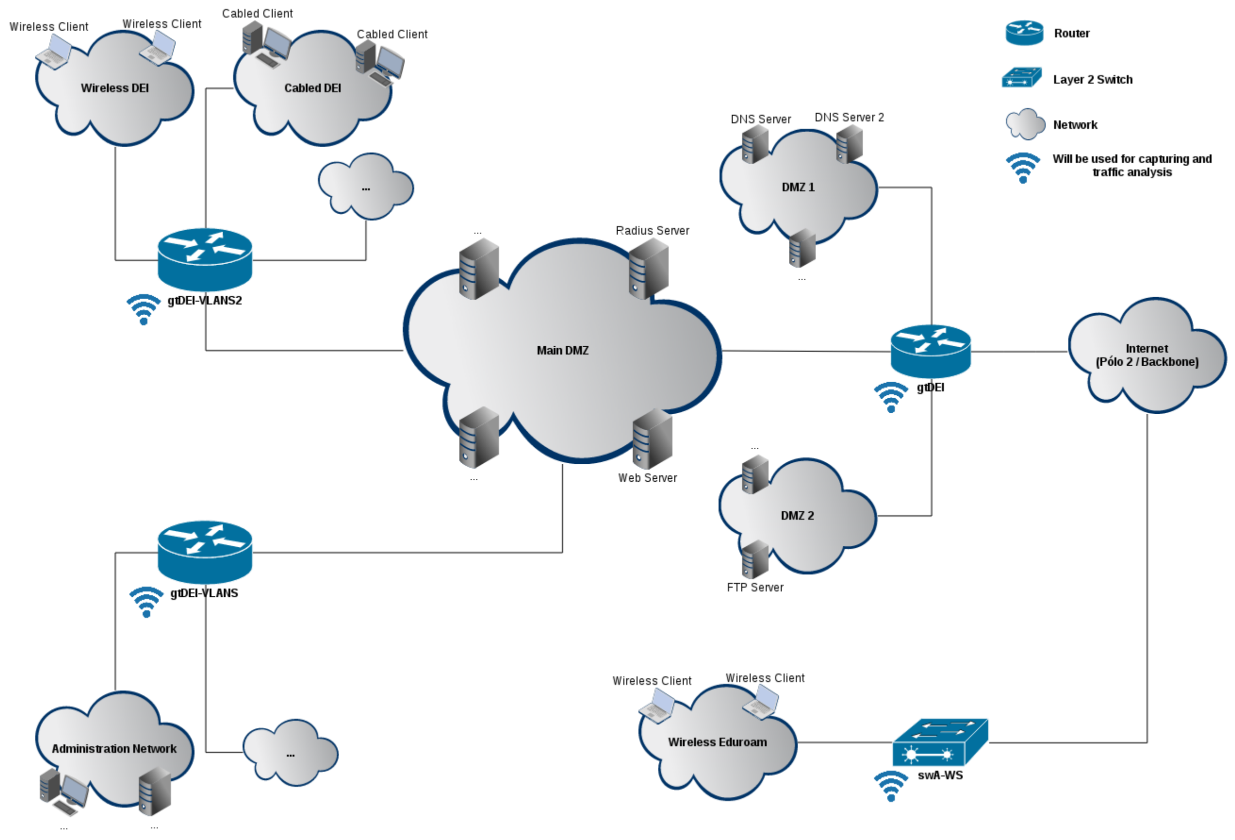


Figure 12: A minimalist diagram of the department's network infrastructure.

As one can see in the diagram, there are 4 different core points (routers and/or switches in this case) in the infrastructure that, when combined, make up for all the traffic that traverses the department's network: **gtDEI**, **gtDEI-Vlans**, **gtDEI-Vlans2** & **swA-WS**.

- **gtDEI:** Can be used to capture all the traffic traversing the department's cabled network, DEI's wireless network and all DMZs. Note that traffic from internal networks will go through NAT routers, thus hiding information of internal addresses when it reaches the gateway.

- **gtDEI-Vlans & gtDEI-Vlans2:** Can be used to achieve visibility on the traffic between internal networks and between different subnets before NAT is applied.
- **swA-WS:** Could be used to capture all eduroam's wireless network traffic, however communication between this switch and the Wireless LAN Controller (not present in the diagram) is encapsulated and provides no useful data for now.

## 3.1 Traffic capture and analysis strategy

In this section we will explain the main steps we took to conduct a preliminary analysis of the state of the network. Every decision made at this point was merely a temporary solution in order to achieve results as fast as possible, and is not in any way related to the final architecture of the system.

The following table (Table 4) shows the specifications of the machine we used in this phase to capture and analyze the traffic. It's worth mentioning that this machine's CPU was above 80% usage at all times, which clearly indicates the need of scalability for a system of this type:

| System Specifications | |
|---|---|
| **Operating System** | CentOS Release 6.7 (Final) |
| **CPU** | Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, 8Mb Cache |
| **Disk** | Samsung SSD 850 EVO 2.5" SATA III 500GB |
| **RAM** | Kingston 2x8Gb RAM DIMM @ 1600 MHz |
| **Network Card** | 2xIntel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection |

Table 4: Server specifications.

### 3.1.1 Capturing the network traffic

Techniques for achieving network visibility are explained in section 2.2.1, nonetheless and regardless of its efficiency, in order to capture the department's network traffic at this stage, we used the following techniques:

- **Port mirroring (2.2.1.1):** Port mirroring was used to capture the traffic in **swA-WS**, as it was quickly configurable and arguably the easiest way to achieve results.
- **Software-level traffic mirroring (2.2.1.3):** Software-level traffic mirroring was used on **gtDEI**, due to it being a Linux router and not supporting the lower level port mirroring (2.2.1.1).

### 3.1.2 Analyzing network traffic

Now that all the desired traffic is reaching our intrusion detection machine, we need an intrusion detection software and some way to produce statistics from its output.

Intrusion detection strategies are addressed on section 2.2.2 and intrusion detection systems on section 2.4, nevertheless, for now we decided to use Snort [46] as our intrusion detection software, as it is a well known open-source alternative and highly regarded in the intrusion detection field. Snort provides us with thousands of known attack patterns (via community rules and commercial rules), some of them already deprecated and others with high chances of triggering false-positives, however for testing purposes, we decided to use them all (See Appendix B).

As for producing statistics through the analysis of Snort's alert logs, we chose Splunk [47] as our data analyzer, as it is a powerful product that includes a free release, suitable for preliminary tests.

## 3.2 Preliminary analysis results and conclusion

After running Snort with Splunk for over a month in the department's network (while in full production mode), we've achieved very interesting results, which will be presented next. We decided to split the results in two parts: Attacks **against** the department's network and attacks **originated at** the department's network.

The following chart (Figure 13) shows an overview of the most exploited vulnerabilities detected by Snort, when aggregating both inbound and outbound traffic of the network. Please note that, in order to achieve the most accurate results possible, many outliers have been removed, such as clear false positives and irrelevant traffic.

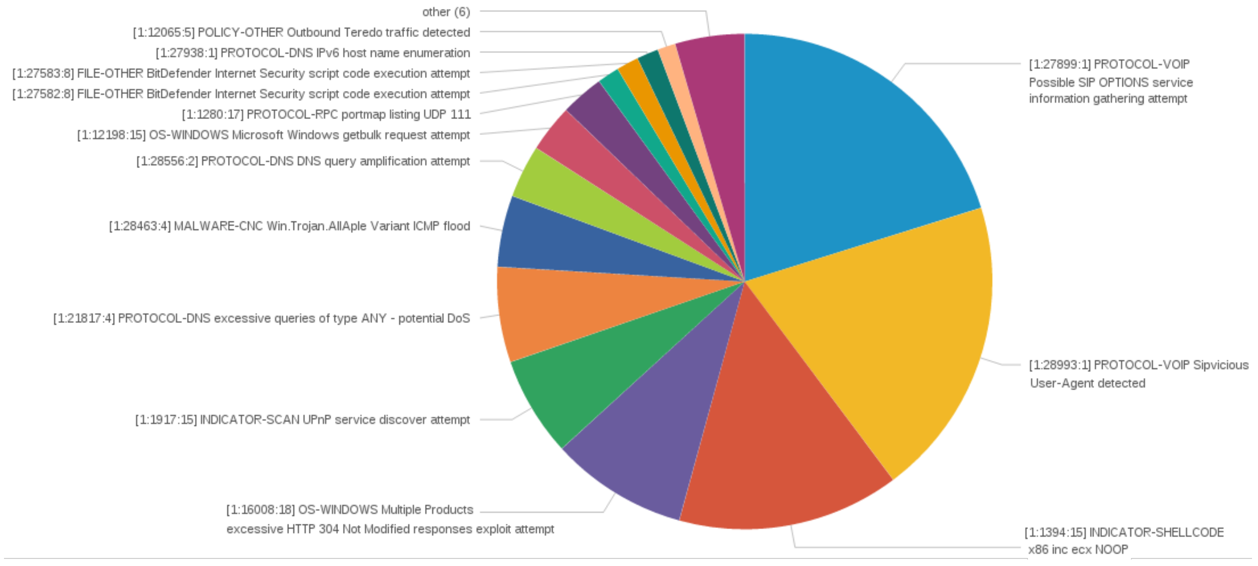


Figure 13: Top vulnerabilities when aggregating both inbound and outbound traffic.

### 3.2.1 Attacks against the DEI's network

In Figure 13, we see the overall most exploited vulnerabilities, the next chart (Figure 14) shows only vulnerabilities exploited in direct attacks to the department's network.

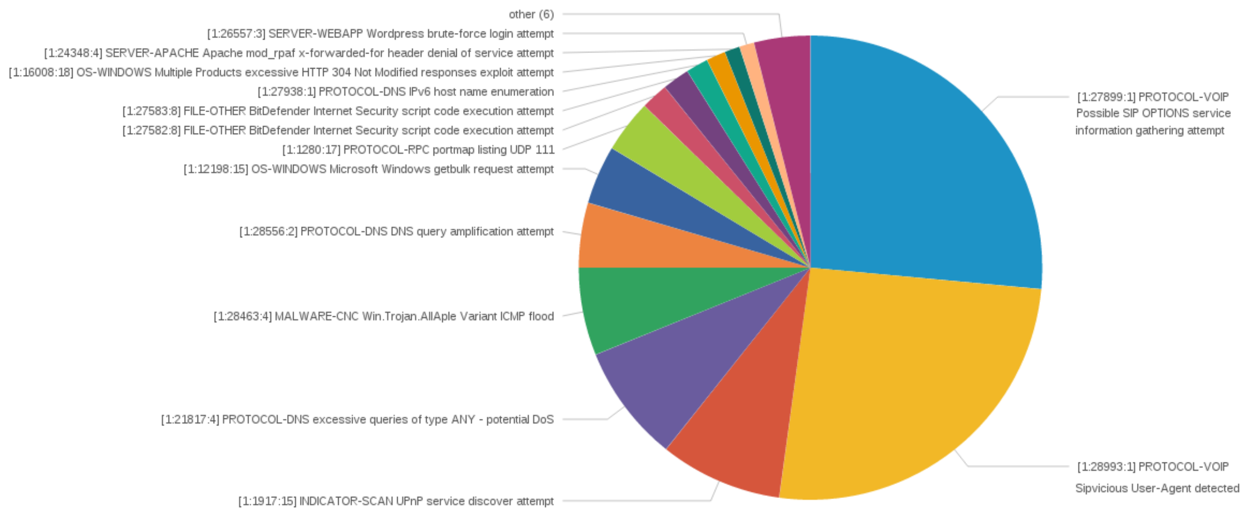


Figure 14: Top vulnerabilities exploited in attacks to the department's network.

By analyzing both previous charts, we can conclude that the majority of attacks are generated outside the network. The next chart (Figure 15) shows the most targeted machines (IP address and reverse DNS) for attacks to the department's network.

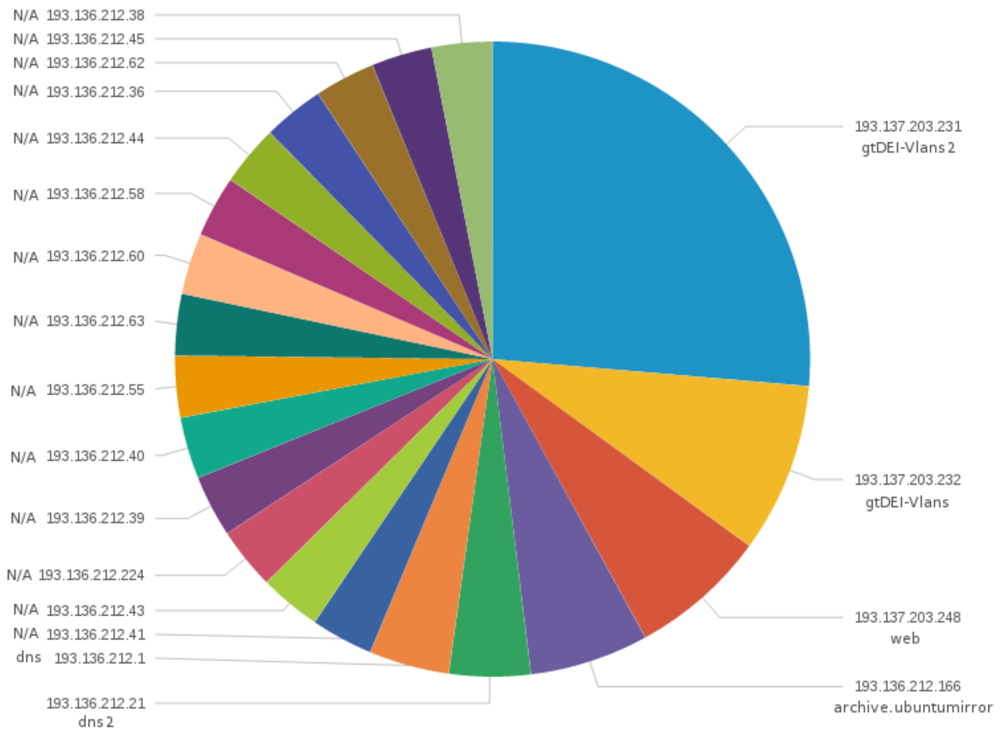


Figure 15: Top targeted machines in attacks to the department's network.

After analyzing this chart, we can conclude that almost 50% of the attacks don't have a specific target, thus the attacker probes the whole network looking for machines to infect.

The next chart (Figure 16) shows the country of origin of most of the attacks targeted at the department's network.

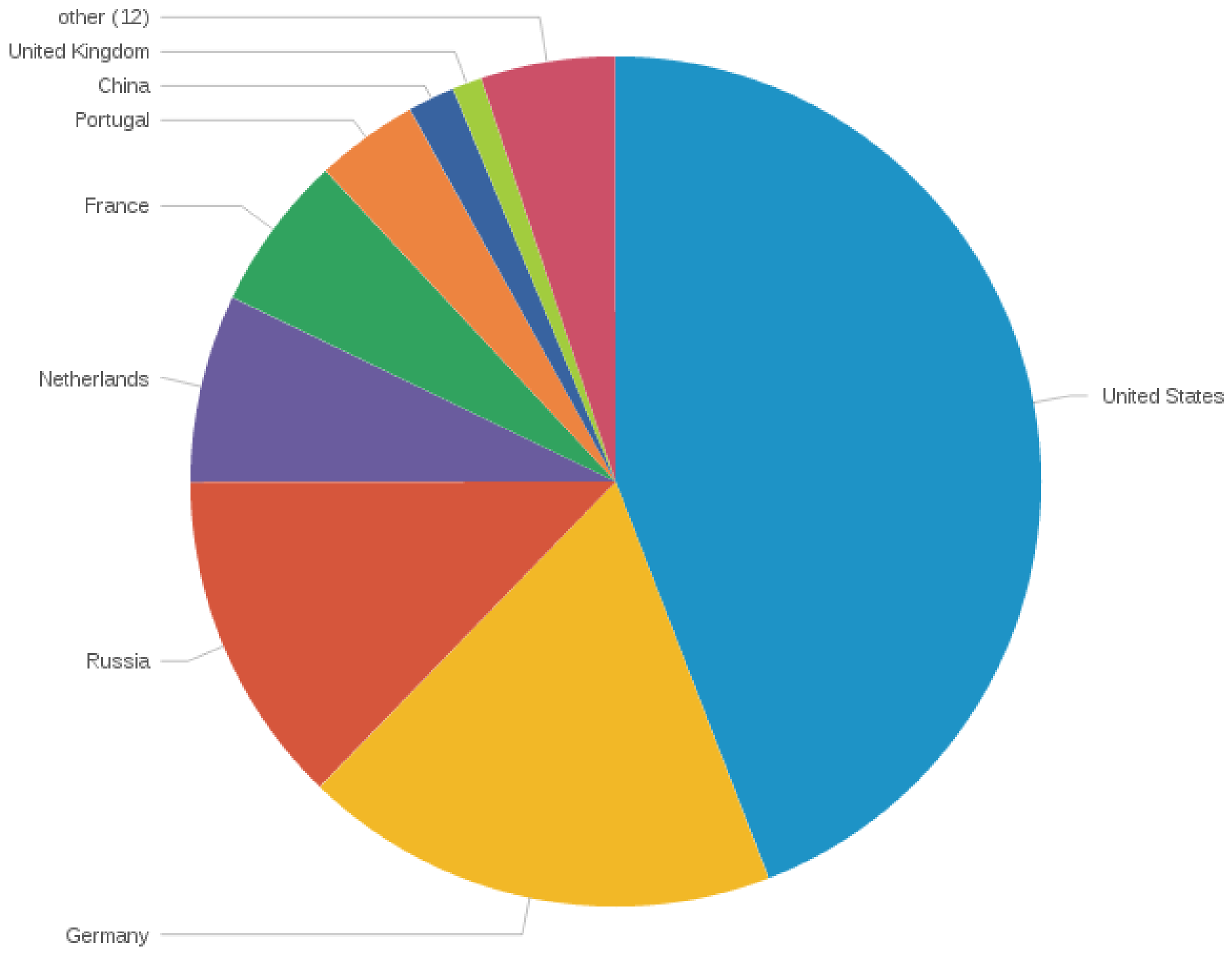


Figure 16: Top origin countries in attacks to the department's network.

United States is clearly the source of most of these attacks. After some investigation, we found out that there is actually a botnet situated in the United States, composed of approximately 60 hosts, launching attacks at our network.

### 3.2.2 Attacks originated at the DEI's network

The department's network is not only the victim of attacks, but can also be the attacker itself. This usually means that there can be either users launching attacks on behalf of our network, or infected/compromised hosts inside the network (which may even belong to a botnet). The following chart (Figure 17) shows the vulnerabilities exploited in attacks originated at the department's network.

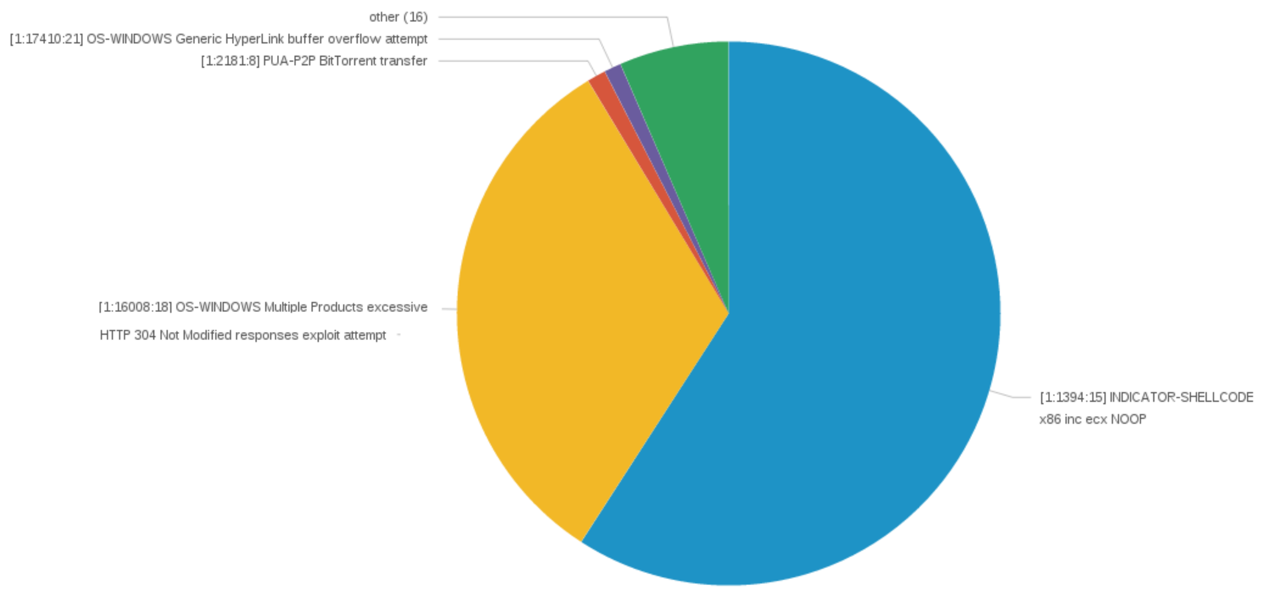


Figure 17: Top vulnerabilities exploited in attacks originated at the department's network (Outlier).

After obtaining such a strange chart that highlights only two vulnerabilities, we immediately realized something must be wrong, as in a network as diversified as the department's network, it is unlikely for the malicious traffic to be majorly composed of only two types. This brings us to the following chart (Figure 18):

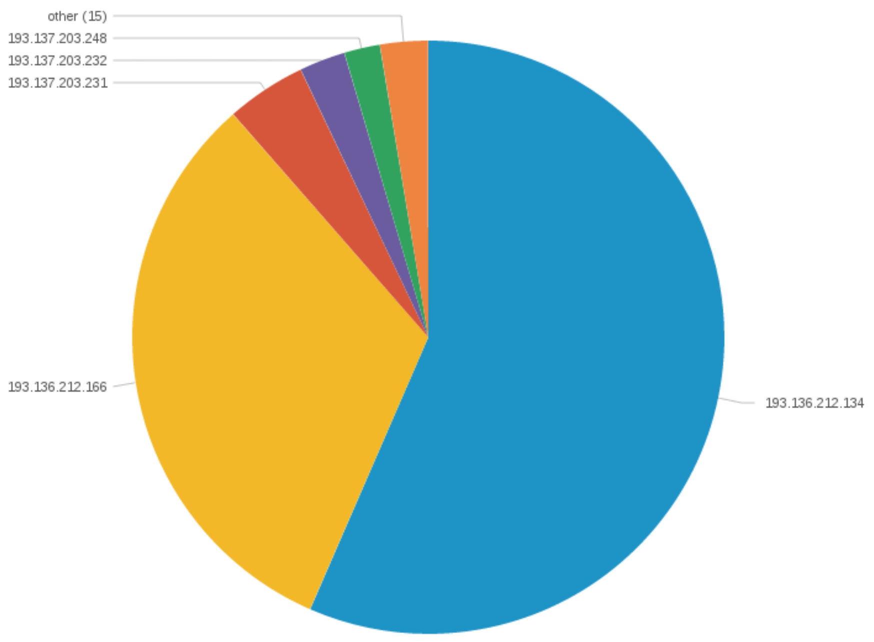


Figure 18: Top machines launching attacks originated at the department's network (Outlier).

Looking at both previous charts, there were two possible options: Either we have compromised machines launching the majority of our attacks, or they are just false positives detected by Snort.

After a clear analysis of both machines responsible for the majority of the attacks and their status and context, we concluded that both cases were mere false positives detected by Snort, thus proving that the system is not perfect, nor 100% reliable, and false positives may appear. We need to deal with them by analyzing the context of the issue and continuously tweak the IDS/IPS in order to increase its reliability.

After removing both false positives, we obtained the following vulnerabilities chart (Figure 19:

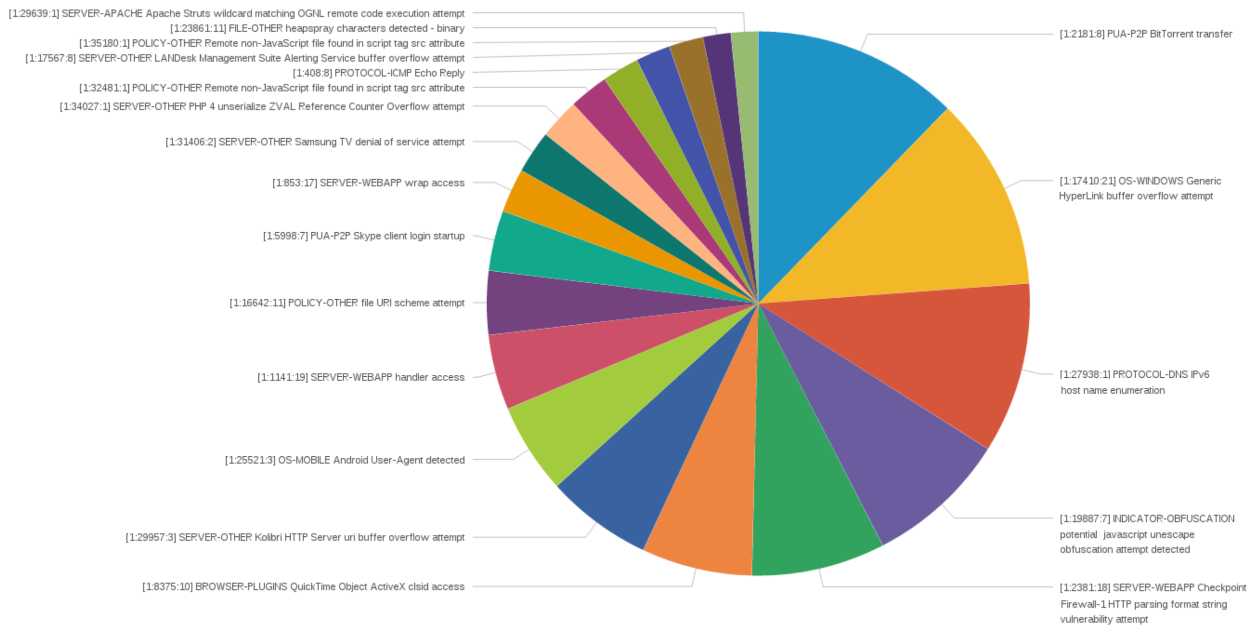


Figure 19: Top vulnerabilities exploited in attacks originated at the department's network.

We now obtained a much nicer chart, with various vulnerabilities being exploited, most of which may also be false positives, and others merely showing attempts at violation of our network policies (e.g. BitTorrent traffic). The machines issuing these attacks can be seen next, on Figure 20.

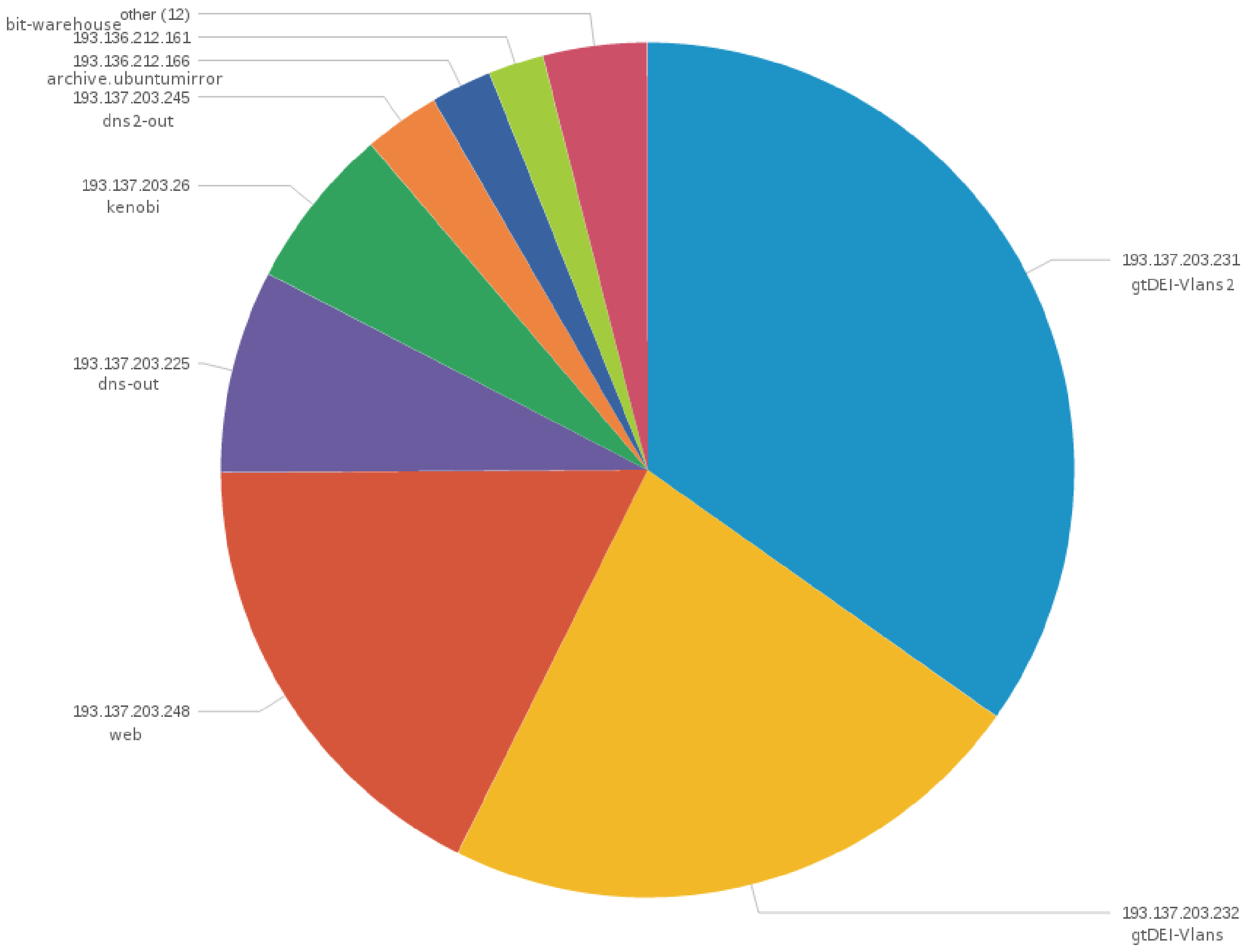


Figure 20: Top machines launching attacks originated at the department's network.

Out of curiosity, the following chart (Figure 21) represents the countries targeted by our department's machines in said "attacks":

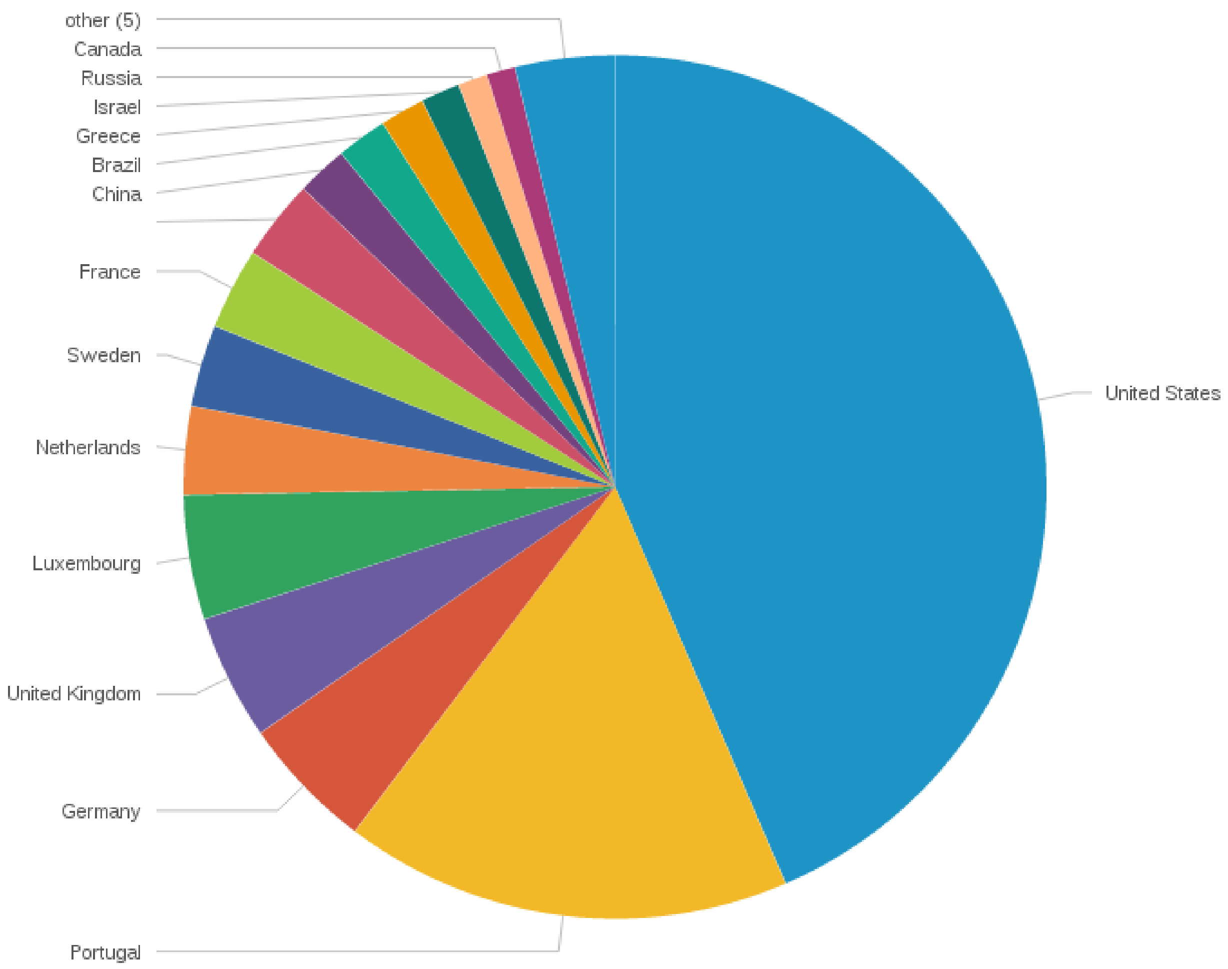


Figure 21: Top targeted countries in attacks originated at the department's network.

After achieving such results, we inevitably conclude that our network, just like any other, is not invulnerable to attacks and is under constant scanning for unhandled vulnerabilities and unpatched hosts. An IDS/IPS is needed to fight these issues and will increase not only our network's security and reliability, but also the trust of our users.

It should also be taken into account that all the vulnerabilities listed were detected by a misuse-based network intrusion detection system, which we already know doesn't cover every existing type of threat. HIDS and anomaly-based NIDS are also relevant and should be considered, since in the end, every system complements each other and covers each other's flaws in some way.

# 4 System Architecture

We have now reached the core section of this report. All our previous investigation has led to the definition of a system architecture most suited to the needs of our scenario - the entire Pólo II network and the communication's network and hosts of the Department of Informatics Engineering, and in the future the entire network of the University of Coimbra.

The first step consists in defining and presenting the functional requirements, a certain set of functions that the system must support, as well as various tasks it must execute within said functions [70]. Then, we will cover non-functional requirements, also known as quality attributes, a set of potentially relevant attributes that the system must comprise in specific situations [71]. Finally, we will introduce our architecture proposal, along with its evaluation and validation strategies.

## 4.1 Functional Requirements

The following are the functions the IDS/IPS must support when in production. We will split the functional requirements in various categories, thus making it easier to define the scope of each one.

### 4.1.1 Detecting

- **#FR01 - Detect known attacks using misuse-based detection:** Easily the most basic function of an IDS, which is to detect known vulnerabilities at every relevant layer (possibly excluding layer 1 - the physical layer, for obvious reasons).

- **#FR02 - Detect attacks at the host level:** The IDS must also act as an host-based IDS, by effectively analyzing and identifying attacks at the operating system level (for multiple operating systems), which includes vulnerabilities that intend to explore flaws in local applications. Detection strategies must include at the very minimum: log analysis, file integrity checking and rootkit detection.

- **#FR03 - Detect attacks based on the connection context:** By analyzing both the headers and the body of the packets, the IDS must be capable of detecting attacks based on the context of the connection, which includes protocols, ports and source/destination addresses. This doesn't necessarily mean that there must exist a corresponding documented vulnerability.

- **#FR04 - Detect major changes/spikes in traffic:** This requirement is primarily related to DoS/DDoS attacks. The system must be capable of detecting major changes/spikes in traffic by analyzing its throughput, and triggering a reaction as soon as specific types of traffic surpass certain thresholds (that must also be customizable by sysadmins), therefore effectively detecting unpredictable traffic spikes, that are likely to be an indication of a denial of service attack.

- **#FR05 - Detect attacks using anomaly-based detection:** The system must apply one or more anomaly-based intrusion detection techniques (addressed in Section 2.2.2), in order to identify not only known vulnerabilities but also possibly unknown attacks.

- **#FR06 - Detect attacks real-time (inline):** Attacks must be detected real-time, before reaching the target machines, therefore the IDS/IPS must support being placed inline with the traffic. This means that not a single malicious packet can reach a host behind the IDS/IPS without being analyzed and filtered (if needed) first.

### 4.1.2 Reacting

- **#FR07 - Promptly react to ongoing attacks:** Responses comprised by the system to mitigate ongoing attacks must include, in the case of a NIDS: disrupting/terminating connections and interacting with multiple firewall systems (such as iptables, Cisco's, etc.) in order to block the attack's origin, therefore effectively denying it access to a network or a host, for a certain period of time or even permanently. When it comes to HIDS, reactions to intrusions must include: process killing, user session killing, user banning and support the automatic triggering of external scripts.

### 4.1.3 Alerting and Logging

- **#FR08 - Provide detailed and centralized logging of attacks:** The IDS must be capable of composing detailed alert log files of the detected intrusions, providing sufficient information to make the context of the suspicious event clear. In case of a NIDS, this log must also include all the core information of the connection, such as the protocol being used, the source and destination addresses and ports, arrival time of the intrusive packet, among others. As for HIDS' logs, it is absolutely necessary that it contains the application that triggered the detection, and possibly the process ID and the user running it. Besides logging every intrusion detected, the IDS must also be able to centralize the logs composed by every node in the scenario. There must also exist a logging clean up mechanism with an adjustable period (e.g. 4 months), after which every information older than that period must be permanently erased.

- **#FR09 - Quickly notify network managers:** When an anomaly is detected and surpasses a predefined threshold of severity, notifications must be immediately sent to key people, like the system administrators/network managers or any other configurable third-party. There must exist a mapping between severity and key people, that will dictate who is notified depending on the severity of the detected intrusion. These notifications can be sent via various means (such as SMS, e-mail, instant messaging) and must contain a brief summary of the exploited vulnerability, its origin and its target.

### 4.1.4 Displaying

- **#FR10 - Provide a centralized mean of displaying the detected intrusions:** The system must provide ways for the system administrators to analyze traffic anomalies in near real-time, with sufficient detail and human-readable information, through a centralized platform, accessible remotely.

- **#FR11 - Provide a centralized mean of displaying history and statistics:** Besides near real-time information displaying, the IDS/IPS must also be able to keep a track of past vulnerabilities by composing a history and generating statistics based on that. These statistics must include the top vulnerabilities exploited, popular target machines, popular target services/ports and popular countries of origin. This will allow network managers to analyze traffic tendencies and produce management reports in the future. There must exist a history clean up mechanism with an adjustable period (e.g. 4 months), after which every information older than that period must be permanently erased.

- **#FR12 - Generate and send automatic reports:** Based on all the collected information and statistics, the IDS/IPS must also be able to automatically generate periodic reports containing the top vulnerabilities exploited in said period, as well as the top target machines and services. These reports can be sent to various destinations, easily adjustable by the system administrators.

### 4.1.5 Configuring

- **#FR13 - Provide a centralized mean of managing and deploying configurations:** The architecture must include a centralized configuration management and deployment tool, in order to automate and speed up the process of updating the configurations of every system's component in a large scale environment.

### 4.1.6 Securing

- **#FR14 - Secure the communication between the system's components:** The communication between system components has to be authenticated, using access control mechanisms based on public/private keys. Every communication that goes through public networks or subnetworks must also be secured, via encryption mechanisms like TLS/SSL.

## 4.2 Quality Attributes

Quality attributes will be presented using separate scenarios, each one presented in six portions: Source of stimulus, Stimulus, Artifact, Environment, Response and Measure.

### 4.2.1 Availability

Availability is directly related to the system uptime and the frequency of failures and faults. For something as important as an IDS/IPS, availability is undeniably necessary. If a single system node is not up, even if for a short amount of time, it could mean a potential threat that was just waiting for the right moment, to hit a vulnerable core network point.

| **Source of stimulus** | IPS/IDS internal code or external sources such as physical human interaction |
|---|---|
| **Stimulus** | A system node crashes/becomes unavailable (e.g. Segmentation Fault), is attacked or is physically compromised |
| **Artifact** | The machine processor/memory in the case of a crash or a whole system node in the case of physical damaging |
| **Environment** | Normal operation or situations of system overloading |
| **Response** | The system continues operating as normal. |
| **Measure** | The system continues in a normal state of operation, and no interruptions are identified. |

Table 5: Availability scenario

### 4.2.2 Modifiability

Modifiability is related to the system's capability of handling changes. In this case we consider changes things such as kernel updates, internal IDS/IPS software updates and network topology changes. The system must be prepared to deal with these modifications, and every modification must be backwards compatible. This means that the system should not be negatively affected by any change,

| **Source of stimulus** | System administrators and developers |
|---|---|
| **Stimulus** | A critical kernel update is issued, network card's firmware updates, internal IDS/IPS software/architecture updates |
| **Artifact** | The HIDS/NIDS nodes. |
| **Environment** | At runtime |
| **Response** | The system continues operating as normal. |
| **Measure** | Cost in terms of system downtime (if applicable) |

Table 6: Modifiability scenario 1 - Updates

| **Source of stimulus** | System administrators |
|---|---|
| **Stimulus** | A new subnetwork or host is added and intends to be monitored |
| **Artifact** | Network |
| **Environment** | At runtime |
| **Response** | The system continues operating as normal, without causing any instability. |
| **Measure** | The system continues in a normal state of operation, and no interruptions are identified |

Table 7: Modifiability scenario 2 - Network topology changes

### 4.2.3 Performance

*"Performance is about timing"*[71]. In this case, performance is related to the load the system can handle. If we're talking about an inline IDS/IPS, we must take into account the additional network delay, as every packet is processed prior to reaching the target machine and, under high load, performance becomes a much heavier factor and cannot be compromised, since this would lead to a decrease in overall network quality and availability of core services to the end-users.

| | |
|---|---|
| **Source of stimulus** | One or multiple machines all over the world |
| **Stimulus** | Packets arrive at sporadic times |
| **Artifact** | IDS/IPS |
| **Environment** | Normal operation or under high load (e.g. Volumetric DDoS) |
| **Response** | Every packet is analyzed correctly. |
| **Measure** | In a normal production state, the system must be able to support up to **10Gbps of traffic**. |

Table 8: Performance scenario

### 4.2.4 Testability

Testability is related to the system's ability to cope with intrusion detection tests, issued by the testers and/or system administrators. If talking about an IDS/IPS, the majority of these tests will consist in penetration tests, vulnerability exploiting and actual attack scenario simulations. The system must be prepared to run and still detect intrusions while being tested, without taking extreme measures against the testers, and must also effectively record the results of each test.

| | |
|---|---|
| **Source of stimulus** | Authorized tester |
| **Stimulus** | The deployment/implementation of critical fractions of the IDS/IPS or periodic penetration tests (e.g. every 2 months). |
| **Artifact** | IDS/IPS |
| **Environment** | In production (at runtime). |
| **Response** | The system successfully accepts and runs the tests without inducing any downtime. |
| **Measure** | Percentage of detected vulnerabilities (issued by the tester). |

Table 9: Testability scenario

### 4.2.5 Usability

Usability is related to how easy it is for the end-user to achieve a desired result, through the use of a certain system. However, in this case, we can only consider usability in components where network managers and system administrators directly interact with the system, for example the real-time logging displaying or the statistics application.

| **Source of stimulus** | Network Managers and System Administrators |
|---|---|
| **Stimulus** | The use of the real-time intrusion displaying application, or the statistics application. |
| **Artifact** | IDS/IPS |
| **Environment** | At development and production (runtime). |
| **Response** | The system presents the desired results and their context, in an organized and human-readable way, while also making it easy to navigate, filter and interpret the data. |
| **Measure** | Number of operations to achieve the desired result. |

Table 10: Usability scenario

#### 4.2.6 Scalability

Scalability also plays a very important role regarding IDS/IPS, and is highly related to performance. In order to maximize performance, we may need to split our system in various different nodes, aiming to distribute the load among them. Not only that, a highly scalable system can easily embrace the addition of other monitoring points, such as a whole new network. Lastly, by allowing the system to scale, we're also improving availability, by removing the single point of failure factor that is a single node.

| **Source of stimulus** | One or multiple machines all over the world |
|---|---|
| **Stimulus** | Packets arrive at sporadic times |
| **Artifact** | IDS/IPS |
| **Environment** | Normal operation or under high load (e.g. Volumetric DDoS) |
| **Response** | Every packet is analyzed correctly. |
| **Measure** | Throughput (packets analyzed per second, traffic in Gbps) per node and CPU load of each node. |

Table 11: Scalability scenario

## 4.3 Proposed architecture

We will now present our proposal for the system architecture, taking into account every functional requirement and quality attribute. It should be noted that this architecture was tested throughout the first and second semesters, and in the end we realized it is a valid architecture that attends to all our needs, although it may still evolve accordingly.

The first step will be to expose a higher level view of the architecture, through a logical diagram containing the core components and modules of our system, the connections between them and limited detail about certain quality attributes, while disregarding technical, physical and low level details.

Next we will transition from the logical architecture into a physical view, much closer to the real world, this time detailing every physical connection between machines, when applied to the current department's network, as presented before on chapter 3.

At last, a choice has to be made as to what products to adopt, that better incorporate in this architecture. We will explain the reasons behind our choices and all the criteria applied in the decision process. As we've stated before, nothing is definite and if at later time, we realize that said products are not well suited for the current architecture, changes will inevitably happen.

### 4.3.1 Logical architecture

The logical architecture diagram is shown in Figure 22 and right below it, the description and purpose of each component. There are three scalable modules, which means that, if necessary, such module will support being scaled and automatically adapt, without needing reconfiguration or affecting the whole system.

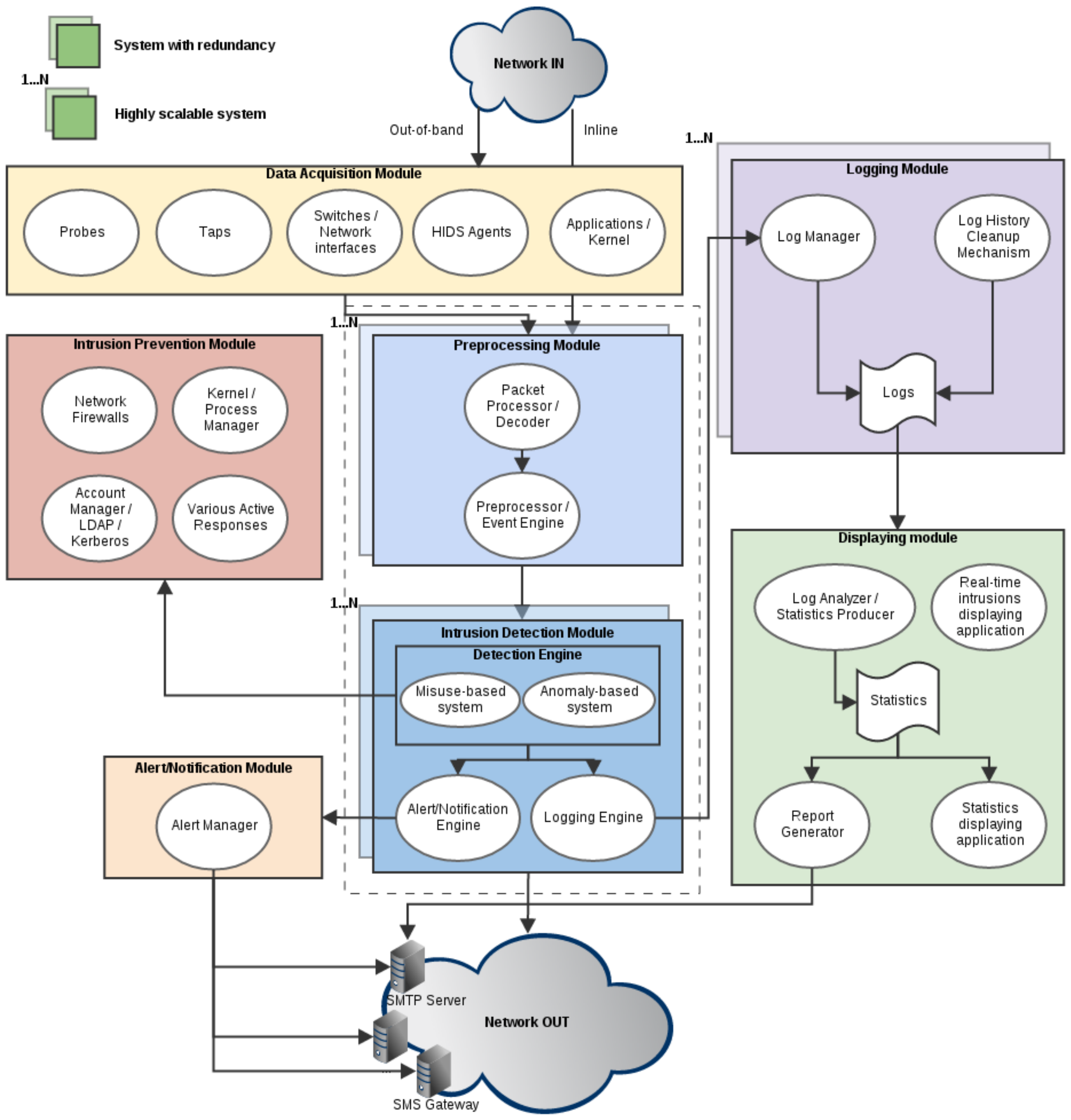


Figure 22: Logical architecture diagram.

**Data Acquisition Module** This module is responsible for acquiring every single bit of data that reaches the detection system. It is composed of mostly physical components, but also virtual.

- **Probes:** A network device or program that is collecting network traffic and mirroring it to a monitoring device (as seen in Section 2.2.1).
- **Taps:** A fully passive network probe.
- **Switches/Network Interfaces:** Switches and network interfaces are also con-

sidered data collectors, since they are responsible for delivering traffic to hosts and networks.

- **HIDS Agents:** In common HIDS architectures, HIDS agents send data about their status to the HIDS central server.
- **Applications/Kernel:** Data exchanged between applications and the kernel is crucial for HIDS Agents to detect security breaches, so it is also considered a data acquisition point.

**Preprocessing Module** The preprocessing module receives, interprets and prepares the data for the intrusion detection module.

- **Packet Processor/Decoder:** Usually a library or an application that captures the data from the various components of the data acquisition module.
- **Preprocessor/Event Engine:** An application whose goal is to apply every necessary change to normalize a block of data (e.g. a network packet) before it reaches the detection engine, usually user-defined and responsible for things like packet decryption and corrections to the packet header.

**Intrusion Detection Module** The core module of the whole system - responsible for detecting, reporting and logging intrusions.

- **Misuse-based system:** A component of the intrusion detection module that uses misuse-based strategies to detect intrusions.
- **Anomaly-based system:** A component of the intrusion detection module that uses anomaly-based strategies to detect intrusions.
- **Alert/Notification Engine:** This engine is responsible for triggering alerts (i.e. notifications) after a serious threat has been encountered. It will then pass on this information to the Alert/Notification module.
- **Logging Engine:** The logging engine is responsible for creating detailed logs for every intrusion detected and sending it to the Logging module.

**Intrusion Prevention Module** The intrusion prevention module, also known as the reaction module, is composed of 4 components whose main goal is to react to the detected intrusion, by blocking it and/or blocking the intruder if necessary.

- **Network Firewalls:** An inline mechanism composed of various rules that dictate what types of traffic can and cannot traverse the network. In this case it can be used to block either the intrusion or the intruder, for example by blacklisting his address.
- **Kernel/Process Manager:** The kernel or the process manager of the system will receive commands from an HIDS Agent, that will require it to deal with a suspicious process or application, by suspending it or terminating it.
- **Account Manager/LDAP/Kerberos:** The account manager that will be used to block an intruder at the authentication level. The blocking can be triggered by either the HIDS or the NIDS.

- **Various Active Responses:** This component represents actions (e.g. scripts) defined by the network and system's administrator, to be triggered upon the detection of specified types of intrusions.

**Alert/Notification Module** This module is responsible for receiving alerts from the intrusion detection module and actually delivering them to the network's administrators.

- **Alert Mananger:** This component will distribute messages to the key people, through numerous communication means. For example, in this, case it will communicate with an SMTP Server and with an SMS Gateway, to send e-mails and SMS respectively, to the system's administrators.

**Logging Module** The logging module receives the logs in their raw form from the intrusion detection module, prepares them to be written to a file and to be sent to the displaying module.

- **Log Manager:** Responsible for receiving the logs and rearranging them the best way possible, to be more easily readable by system's administrators, and interpreted by the displaying module. It then generates log files accordingly.
- **Logs:** The artifact generated by the Log Manager - the log files.
- **Log History Cleanup Mechanism:** A mechanism used to delete older log files, as specified in #FR08.

**Displaying Module** This module is responsible for every information displayed and sent to the end-user, in this case the system administrators.

- **Log Analyzer/Statistics Producer:** This component will receive the logs from the logging module, and produce relevant statistics to the end-user.
- **Statistics:** The artifact generated by the previous component - a statistics file.
- **Real-time intrusions displaying application:** This component is independent from every other inside this module. Its only function is to receive the logs from the logging module and display them, in near real-time, to the system administrators.
- **Report Generator:** Based on the statistics produced by the Log Analyzer/Statistics Producer, this component will generate reports containing key information about the state of the network, and the network's hosts, over a specific period of time. It will then send the reports to numerous key people, defined by the network administrators.
- **Statistics displaying application:** This application is responsible for displaying the generated statistics to the end-user. This means, not only displaying the information contained in the statistics artifact, but also manipulating that information in order to generate human readable data structures, like charts, graphs and maps. This component is usually combined with the Log Analyzer/Statistics Producer, in the physical architecture.

### 4.3.2 Physical architecture

The physical architecture diagram is shown in figure 23 and will be thoroughly explained ahead.

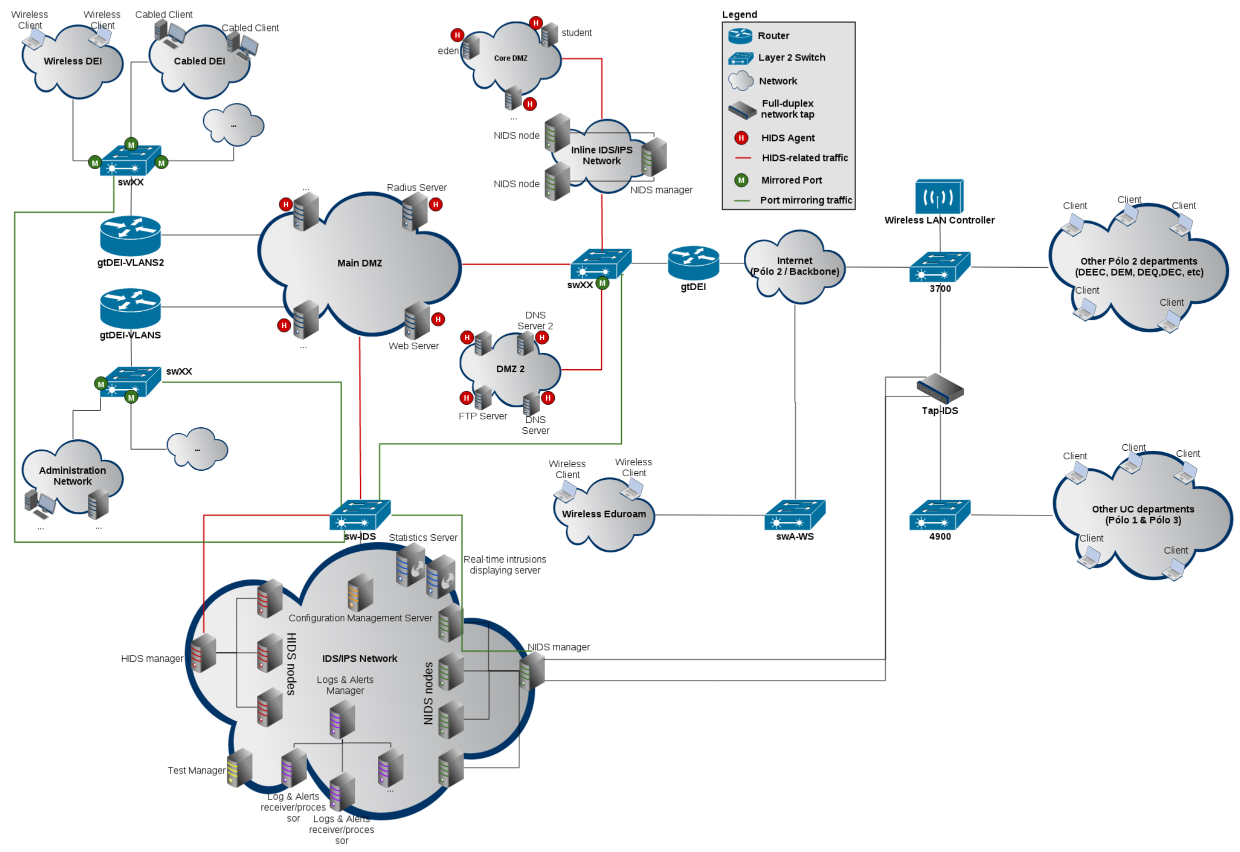


Figure 23: Physical architecture diagram.

We intend to build a **whole new subnetwork** solely dedicated to the IDS/IPS, this way we **enhance security** by avoiding external interaction with the system, while also logically separating the system and its components from the rest of the network, making it **virtually independent**.

Inbound and outbound traffic from every Pólo II department will be captured **using a full-duplex network tap**, placed between the two core switches that carry every single packet flowing to and from these networks. A network tap was the obvious choice here since we cannot risk to lose packets at such a core point of the network, but most importantly, we must not allow the switches to be overloaded and putting at risk the whole Pólo II internet connection.

Traffic between internal subnetworks inside our department will be captured **using port mirroring** at their respective entrance switches. The mirrored traffic will then be forwarded through a dedicated port, until it reaches the IDS network. We opted to use port mirroring in this case, because using taps to mirror every single subnetwork quickly becomes infeasible in terms of price, and we feel it is an unnecessary measure, since the traffic between subnetworks is only a minor portion of the whole network traffic, therefore the chance of missing packets is negligible.

Additionally, the inbound and outbound traffic to our group of core servers (i.e. Core DMZ) will be secured by an **IDS/IPS network placed inline**, which means that for this specific network, we won't be using any strategy to capture and mirror traffic, instead we will force every packet to be firstly routed through our IDS/IPS network, and only then to its destination. Our core servers, the information they sustain and the traffic travelling to and from them contains **sensitive and potentially classified** data about our network and our users, for this reason we must secure it the best way possible, by detecting and blocking intrusions real-time.

When it comes to the host-related intrusions, as seen in the diagram, every server (both core and non-core) will **include a HIDS Agent**, that will exchange **encrypted data** with the HIDS Manager via the **IDS VLAN**, created for this purpose.

The management and configuration of every server/service will be centralized and managed by a single entity - the Configuration Management Server. This entity will be responsible for **pushing and updating the configurations** of every component of our system, hence making the process of scaling this architecture less complex.

Lastly, we have the Test Manager that will **hold and execute all the test cases and tools** necessary to validate the architecture. This machine will be whitelisted by every IDS/IPS and HIDS manager and nodes, and will be used to a great extent throughout the whole development and later production phase.

The physical architecture diagram however, doesn't highlight how the architecture components interact with each other, what kind of integrations are we including and what is exchanged between communications. For this matter, we've built an additional diagram that focuses solely on the most relevant elements of the architecture and their interactions.

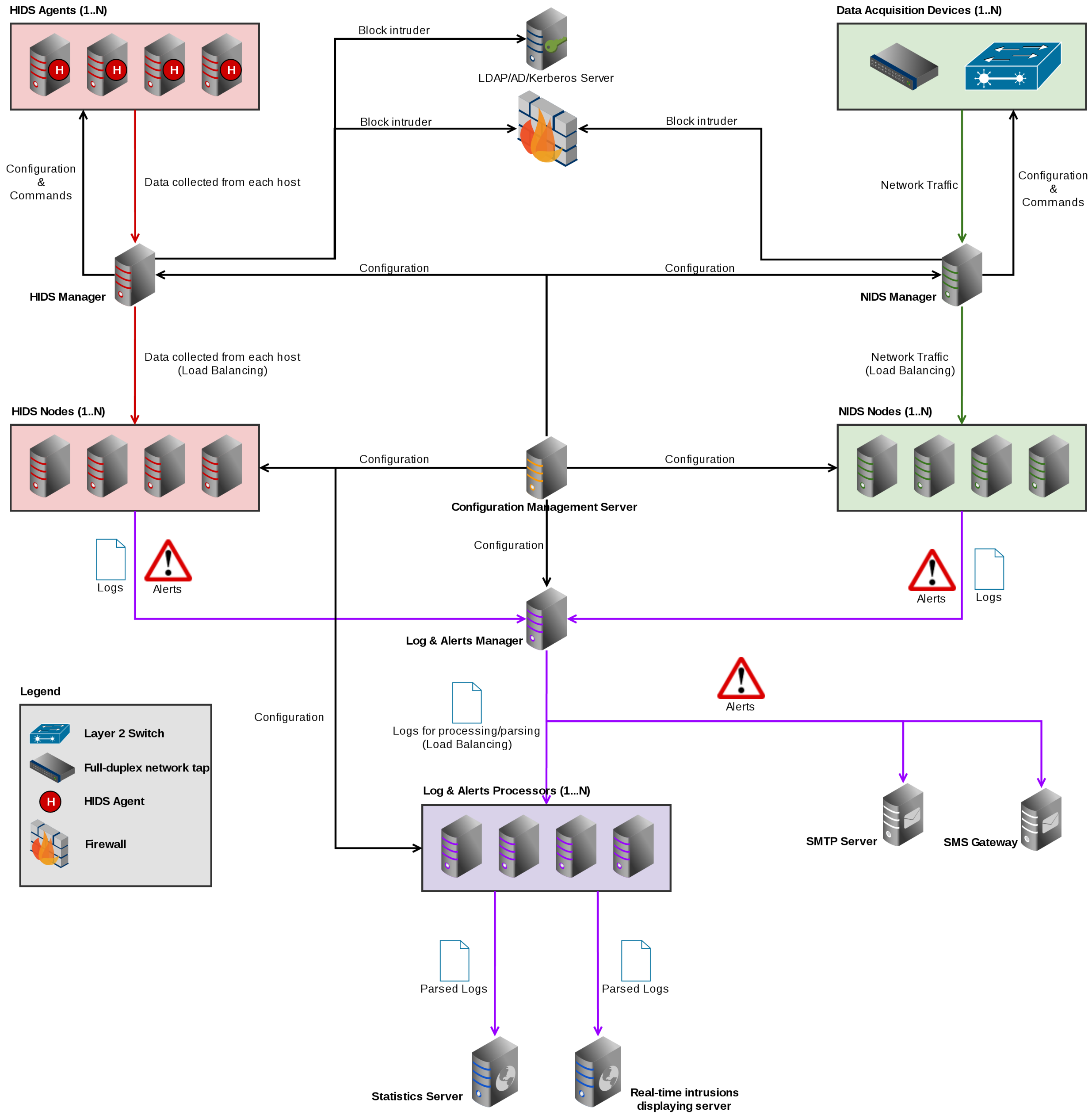


Figure 24: Interaction between the components of our architecture.

The interactions between components, including network firewalls and the logs & alerts servers, statistics server and the intrusions displaying server are exposed in the above diagram (Figure 24).

### 4.3.3 Selection of the preferred technologies

The last step before finishing an architecture proposal is to pick the tools that will make it a reality. Please notice that every product chosen in this subsection is related to every component of the architecture, and not only the HIDS/NIDS, which means that many of these products were not addressed before in this report, however due to past experiences, we believe they are the best possible choice for our system.

It is also relevant to mention that our architecture is completely independent from any product. It is up to the product to adapt to our architecture, and certainly not the other way around. If we find ourselves in the future, somehow restricted by a product's functions, a product change will be considered.

#### 4.3.3.1 Selection of the intrusion detection/prevention technologies

The heart of our architecture lies at the intrusion detection systems, hence this is the most relevant product selection and definitely the hardest.

Commercial products are undeniably powerful, with high quality standards and bring many advantages to their clients, including **24/7 technical support and maintenance**. However, the **ability to audit** an intrusion detection system can be crucial in the selection of the preferred product. Even though we appreciate the benefits of owning a commercial product, we still believe that the advantages of the open-source products justify their choice. Considering this, we conclude that our best options reside in the open-source field, as it will provide us not only with means to audit every single line of code of our IDS/IPS, but also give us **freedom to implement** our own features, and tweak the system to answer our necessities. It is also worth noting that, open-source products have the innate benefit of having a **strong community supporting them**, which will provide us with **numerous means of technical support** and **documentation**.

With this in mind, our choice for an **anomaly-based NIDS** becomes obvious: **The Bro Network Security Monitor.** It is the only NIDS with real support of anomaly-based detection among the few alternatives we have at our disposal.

When it comes to the **misuse-based NIDS**, our choice was not as simple. Snort and Suricata are very similar and some may even say Suricata is superior in terms of performance due to its multi-threading support, which would definitely be a plus when scaling our architecture. However, Snort is a much better documented and established product, with thousands of support forums and many open-source frontend tools. Our final choice was **Snort**, partially due to our past experiences with it, but most importantly due to Snort 3.0 [79], the next big version of Snort, with many new added features, including better documentation and multi-threading support!

Finally, our preferred **HIDS** is **OSSEC**. Although Samhain and OSSEC seem close in terms of performance and detection, we feel OSSEC's version control system, the interaction with its clients, its bug fixing frequency and the large amount of contributors is much more appealing to us and was easily the deciding factor of our selection.

#### 4.3.3.2 Selection of the remaining technologies

**Statistics:** The statistics engine and server will be supported by **Splunk**. We were really impressed by the power of this tool when doing the preliminary analysis, and certainly intend to use it once again. Splunk will also take care of generating and sending automatic reports (specified on #FR12 - Section 4.1).

**Real-time intrusion displaying:** We chose **log.io** [80] for our real-time intrusion (log) displaying application. Log.io provides everything we need for the purpose, including a real-time filter to easily highlight only the desired information.

**Configuration Management:** As for the configuration management tool there were two possible options: puppet [84] or chef [85], both have open-source releases and both receive frequent updates. Nonetheless, we opted for **puppet** because it is a tool we feel more comfortable with and worked with in the past.

**Logs & Alerts:** The log merging and parsing from every component, as well as the whole management of alerts (which includes the mapping of threat severity to alerting key people) will be supported by **our own scripts**, to be programmed in **Python**.

## 4.4 Architecture validation and evaluation strategies

Architecture proposals are complemented with architecture validation strategies. These strategies consist in various methods that intend to, not only prove that our architecture is adequate to the problem, but also verify if most requirements, both functional and non-functional are being complied.

### 4.4.1 Penetration testing tools

Penetration testing tools will be our basis for **validating functional requirements** related to the detection and reaction to intrusions (**#FR01** to **#FR07**), and also the quality attribute related to **testing**. Here we will explore some of the most renowned penetration testing frameworks in the intrusion detection field, in an attempt to find various methods to validate our proposal (SecTools [45] provides a much more complete list of all kinds of intrusion detection tools, what is shown here is just a mere fraction of it). These tools will not be reviewed in full detail, but rather slightly in order to avoid a deviation from the main scope of this report.

#### Kali Linux/Backtrack [38]

We cannot start a penetration testing section without mentioning Kali Linux, formerly known as Backtrack. This is not just a tool, but a whole **debian-based linux distribution** with **over 600 penetration testing tools included**. It is free and absolutely indispensable when heading into the penetration testing world. Most of the following tools are already included in Kali.

### Metasploit [5]

Metasploit is a **penetrating testing platform** that consists of multiple projects, both free (Metasploit Framework, included in Kali Linux) and commercial (Pro and Community). It is widely used and probably the most popular penetration testing tool out there. covering **more than 900 vulnerability tests** of all kinds, through the powerful MSFconsole, which is basically the central command station for Metasploit.

### Nessus [39]

Nessus is a vulnerability scanner that includes both free and commercial versions. It is **highly scalable** and is mostly **intended and optimized for cloud environments**. It consists of multiple agents spread across a network in order to extend the scan coverage and is capable of performing *"configuration and compliance checks, malware detection, web application scanning and more"*. It also incorporates a flexible report/summary and notification system.

### Core Impact [40]

Core Impact is a commercial multi-vector penetration testing tool that is focused in finding vulnerabilities across an entire infrastructure. It covers network, web, endpoint, mobile and wireless vulnerabilities and also validates vulnerability patching on the part of system administrators. It is proud to claim having **"25% more unique Common Vulnerability Exploits (CVE) than the competition"**.

### Canvas [41]

Canvas is a commercial vulnerability exploitation framework. It is a **less expensive alternative** to Metasploit or Core Impact, covering **over 800 exploits in both network and applications**. Canvas prioritizes *"high-value vulnerabilities such as remote, pre-authentication, and new vulnerabilities in mainstream software"*. It is flexible and highly customizable and **updates regularly with new exploits** as soon as they are stable.

### Acunetix [42]

Acutenix is both a web vulnerability scanner as well as a network security scanner. It is a highly expensive product, however it covers **more than 500 different web vulnerabilities** and **more than 35000 known network vulnerabilities**, thus making it a highly regarded product in the penetration testing field.

### Netsparker [43]

Netsparker is a **web application security scanner** claimed to be ***"the only false-positive free solution"*** due to its not only detection, but also exploitation capabilities. It is an expensive product and does not offer a free version.

**w3af [44]**

w3af (Web Application Attack and Audit Framework) is an **open-source web application vulnerability testing tool**. It is still rising in popularity due to its open-source nature and ease of use. It mostly finds and exploits vulnerabilities in web pages directly **related to user inputs**, for example SQL Injection and XSS.

**Nmap [9]**

Besides being the most well known port scanner and network probe in the area, Nmap is also widely used to detect and exploit known **vulnerabilities across all the TCP/IP layers**, through its hundreds of NSE scripts. It is portable, lightweight, open-source and very well documented.

### 4.4.2 Traffic Generators

Traffic generating tools are the best way to validate the robustness of an IDS/IPS. With open-source tools like Ostinato [81], we will be able to simulate a high load network usage, which in turn will allow us to validate the system's **performance** and **scalability**, while also validating functional requirements like **#FR03** and **#FR04**.

### 4.4.3 Packet Sniffing

Packet sniffing tools like Wireshark [82] and tcpdump [83] will provide us means to validate the **security** of the communications between the architecture components, specified on **#FR14**. By simulating man-in-the-middle attacks, we will verify if every communication is being **properly encrypted** while traversing the network.

### 4.4.4 Availability testing

In order to test **availability**, we intend to intentionally induce instability and eventually cause some of our components to become unavailable (for example, through disconnecting an ethernet cable). This way we can successfully simulate a failure scenario and evaluate how will the system behave in this situation.

### 4.4.5 Modifiability testing

When it comes to testing **modifiability**, we are planning on creating various subnetworks with diverse virtual hosts and configure our system to start monitoring it. We will then measure the system's response by identifying any interruptions that might have happened during the modification period. Our goal is to have an easily expandable system, in which such operations involve minimal tweaking on our part.

# 5 Implementation

This section marks the beginning of the second part of this report. Here, we will mostly address the **practical component of our work**, by specifying everything we implemented, while also stating the main differences between our planned architecture and the finished architecture.

Every new architecture component and/or technology that gets introduced, will be presented in this section, some with more detail than others. Apart from implementation, every problem and obstacle we encountered is also worth mentioning, as it may well bring unexpected limitations to our system, thus forcing us to deviate from the ideal solution.

## 5.1 Accessing the data

The first step in the implementation of an intrusion detection system is fetching and reaching the data for analysis. In our case, we handle two different kinds of data: the **network data**, which represents the network traffic from the entire Pólo II, and the **host data**, which is collected from core servers inside the DEI's internal network.

### 5.1.1 Network data

In section 2.2.1, we reviewed techniques for achieving network visibility and concluded that our best option would be acquiring a network tap, due to its **passiveness and reliability**. In the beginning we realized a full-duplex network tap would be our best option, as it would mirror the incoming and outgoing traffic into two different monitoring ports, however, the network tap market is very reduced and clearly still under development. For this reason, full-duplex network taps are very rare and expensive, and most of the times, the companies behind them do not inspire confidence or reliability.

That being said, we opted for an alternate path - **a dual output network tap**. The particularity of this device is that, besides doing its passive traffic mirroring job, it outputs the mirrored traffic into two ports. This means that we can have two monitoring devices directly connected to the tap (instead of having only one), and both will receive full copies of the network traffic mirrored by the tap. Later we will see how this particular feature was extremely helpful.

The **main disadvantage** of this alternative is that, our Small Form-factor Pluggable (SFP) transceivers, connected to our monitoring servers have a 10Gbps capacity, while our network tap can handle traffic up to 100Gbps. In the future, when the combined incoming plus outgoing traffic from Pólo II surpasses 10Gbps (e.g. 5Gbps incoming, 7.5Gbps outgoing), we will be forced to upgrade all our SFPs, and possibly every network interface controller.

### 5.1.2 Host data

Host data is achieved through OSSEC, more specifically through an OSSEC agent. Ideally, this agent would be installed in every server inside the DEI's internal network, but in order to avoid damaging core services in production, we chose to reduce its range to only 11 servers in the beginning - **the most likely targets for intrusions**. This way we can audit OSSEC's performance, tweak its analysis parameters, while also measuring its impact in the core services availability. We will later expand its range to all the other servers, after reassuring that it doesn't impact the host's performance to a level where it can compromise our services.

The central configuration manager will deploy all the necessary configurations in order to install the OSSEC agent in every host. The hosts will then pull their respective configurations and continuously send the captured data to the OSSEC manager. This end-to-end communication between the agents and the manager is established through a dedicated VLAN, **solely created for IDS-related traffic**.

Finally, the data reaches the OSSEC manager, which analyses the data, produces the respective log files and if necessary, triggers any active response defense mechanisms.

## 5.2 High availability

**Availability is a crucial aspect of our architecture**, it is something we wish to maximize, therefore we must develop the appropriate mechanisms that guarantee the effectiveness of this quality attribute. Achieving high availability and scalability in a system with such a high variety of technologies involved is definitely not an easy task, and sometimes limitations unexpectedly show up and commitments have to be made in order to avoid deviations from the requirements.

The following subsections will approach the high availability techniques applied, according to the quality attributes defined in section 4.2.1 (Availability) and 4.2.6 (Scalability), firstly in the perspective of the NIDS and then in the perspective of the HIDS.

### 5.2.1 NIDS high availability

When it comes to the NIDS, as mentioned before, our system implements two variants of intrusion detection: anomaly-based detection (Bro) and misuse-based detection (Snort). These two technologies, although serving the same purpose, **must be handled differently**.

Snort is a misuse-based NIDS, it does perform packet correlation, but only to a certain extent (session tracking, with the use of preprocessors) [102], this essentially means that Snort mostly analyses each packet independently from one another, not taking into account past events (or events over time), or the whole flow which said packet belongs to. In conclusion, load balancing for Snort can easily be achieved by splitting the traffic, without worrying too much about the flows we may be breaking in the process.

On the other hand, Bro, as an anomaly-based NIDS, performs flow analysis and packet/event correlation [103]. We cannot split the traffic the same way we do with Snort, breaking a traffic flow will cause Bro to lose all its references and thus, rendering it useless as an anomaly-based NIDS.

#### 5.2.1.1 Ideal solution (Bro)

According to the the requirement #FR01 defined in 4.1, **all the traffic must be analyzed**, including layer 2 traffic. For this reason, load balancing techniques at the network layer and above are not worth considering.

As mentioned in the previous section, Bro can't be handled the same way as Snort, therefore the optimal solution for Snort is clearly not the optimal solution for Bro. Load balancing at the flow level can be achieved by hardware devices specially built for the purpose. The Bro development team recommends [90] cPacket [91] or OpenFlow [92] switches, however, although being the optimal solution, it's a solution that we cannot sustain.

In the next section we will talk about our proposed solutions in order to avoid running out of budget, while also trying to respect all the functional and non-functional requirements.

#### 5.2.1.2 Current solution

When it comes to Snort, one of the alternatives for load balancing at layer 2 is using modern switches with something called "Link aggregation" [94], or "EtherChannel" [93] in Cisco terminology. This will cause the switch to logically combine two or more of its ports and balance the traffic between them, in order to increase the throughput beyond what a single port can support. Besides, if for some reason a port and/or interface happens to fail, or the host connected to that port becomes unavailable, the switch will automatically adjust the traffic proportion among the remaining ports.

At first sight, this seems like the optimal solution for load balancing at layer 2, however, we must take into account that we're dealing with a system that will support traffic whose bandwidth can go up to 10Gbps. Switches capable of performing link aggregation with 10Gbps ports can be a massive hit in our budget. Nevertheless, there's **one more reason why this isn't the ideal solution**: In the case a **kernel-related issue** occurs in one of the nodes receiving traffic from the switch (disabling the node's ability to process it), the **switch won't realize the node is down**, as the network interface and the connection link will still be established, and the machine will still be turned on.

As an alternative to a switch's link aggregation, we decided to use **Linux's built-in modules for interface bonding and network bridging**. Network bridging is a very well-known concept that has been around for approximately 15 years, and it can essentially transform a linux machine into a virtual network switch.

Interface bonding on the other hand, is a concept very similar to link aggregation, in a way that it also combines two or more interfaces and splits the traffic among them. With that said, these two modules when combined, can achieve an identical behavior of a modern switch's link aggregation.

The main advantage of this solution when compared to the previous alternative is that, **besides detecting when the link/port/node goes down** (just like a switch), we can **also detect when a kernel-related issue occurs** in one of the nodes, this includes: kernel panics and/or network adapter's firmware/driver crashes, **since we can program it!** In favor of covering this possible, although unlikely outcome, we built a script (See Appendix B) that continuously checks the availability of all the nodes, **using layer 3 pings** and re-organizes the bridge and the interface bondings as needed! This only works because, when a kernel-related issue occurs, **the machine will be unable to reply to pings**, even with the connection link still up! Nonetheless, this solution is not perfect, as, in this case, we cannot guarantee a 0% packet loss, since it takes time for the kernel to send/receive pings and adjust the bridge and the bonding (the delay and packet loss are measured in Section A.10.1).

When it comes to Bro and load balancing at the flow level, unfortunately we couldn't find any viable solutions that perform a reliable flow load balancing at a sustainable price, therefore Bro will receive its feed **directly from the second output of the network tap** and analyze the traffic without any form of load balancing.

### 5.2.2 HIDS high availability

The HIDS (OSSEC), as opposed to the NIDS, fetches its data directly from a host and relays it to the HIDS manager through the network layer, thus excluding the need for a layer 2 load balancing mechanism.

There are many ways to achieve a high availability environment under these conditions, one of them is by making use of existing technologies for clustering in linux systems, such as **Pacemaker** [87] and **Corosync**. These technologies, when combined, define an open-source, scalable and highly available cluster manager, suitable for both small and large deployments, also supporting multiple redundancy configurations (quorate, resource-driven clusters, active/passive & active/active configurations).

By creating a shared resource between the HIDS nodes, Pacemaker can guarantee its availability by moving it among nodes, whenever the resource's original host becomes unavailable. Not only can Pacemaker **handle machine failures**, through failover mechanisms, but it can also **apply load balancing techniques** in order to split the requests to a specific resource among other cluster nodes (active/active cluster configuration).

## 5.3 Current architecture implementation

This is a **fundamental section** to read before we start going further into the implementation details of this project. In this section we will present the current network diagram of the system, i.e. our current implementation of the architecture proposed on Section 4.3, with all its commitments.

The diagram below (Figure 25) reveals all of the machines/devices directly related to our system, and the interaction between them. Only by interpreting and understanding this diagram, will the reader be fully ready to proceed reading this report, as it will minimize any possible false assumptions that may occur in the sections ahead.

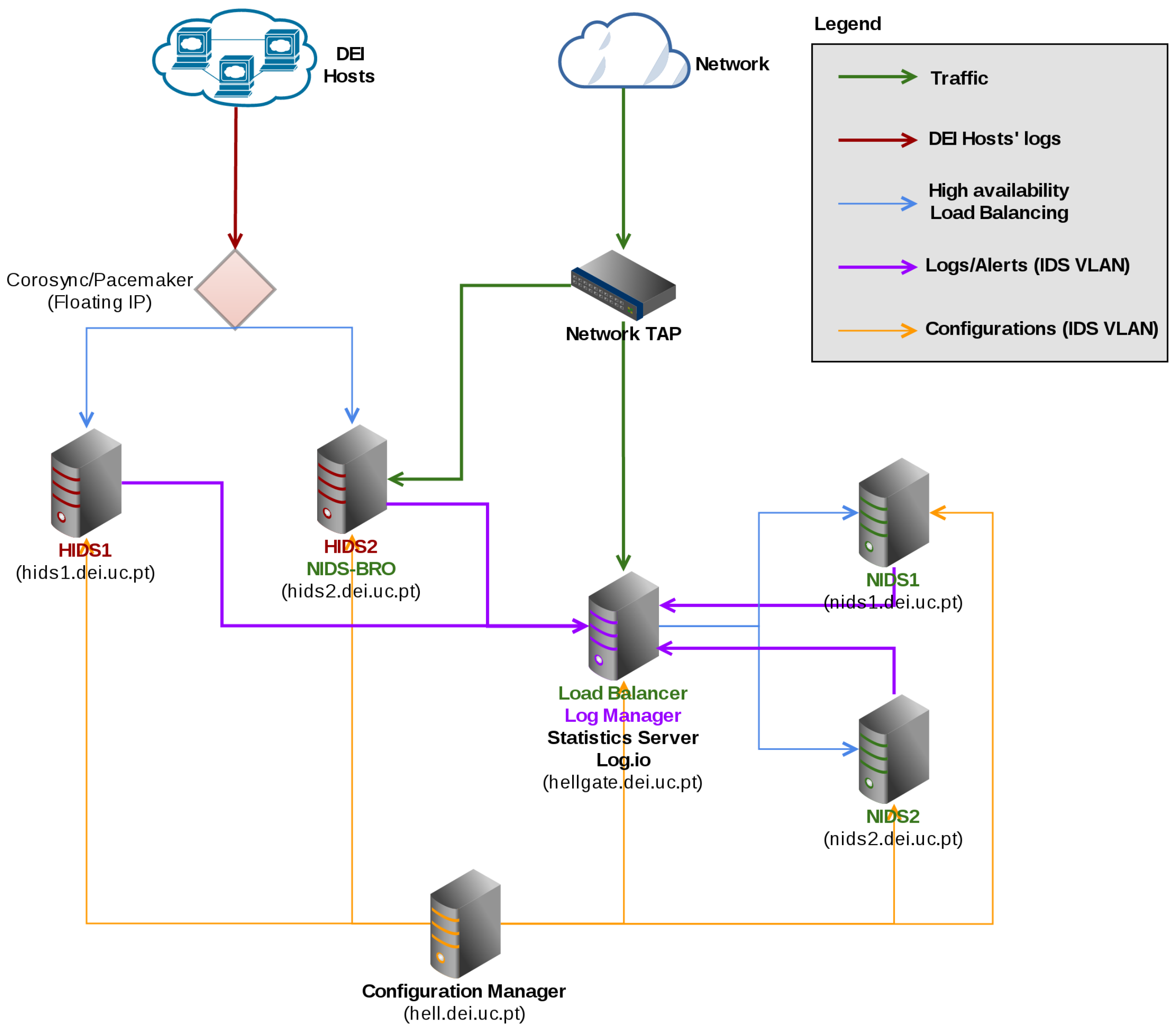


Figure 25: Network diagram of the current architecture implementation

By inspecting the above diagram, we understand that many of the original architecture elements, present in the first physical architecture diagram (Figure 23), that were supposed to become individual servers, were "compressed" into machines that aren't solely dedicated to one task. This was a result of **physical (hardware), budget limitations**

**and specially the efficient use of the available resources**, which also caused our current solution to have **two single points of failures (SPOF)**: The TAP (unavoidable) and "hellgate.dei.uc.pt", which serves multiple core purposes.

## 5.4 Configuration Management

Puppet was our choice for configuration management and deployment, and although much effort was put into writing classes/manifests (Puppet files that contain instructions) and understanding Puppet, there's not so much to be said, as it just...works flawlessly!

We made sure to write every manifest in a way that, we don't ever need to adjust configurations on any machine, except for the Puppet Master. As of now, our Puppet manifests **can successfully automatically deploy our full IDS solution in just a couple of minutes**! We can even issue a mass software update (e.g. Snort, Snort's rules, OSSEC, Bro) by simply uncommenting a few lines.

Let's take a look at a manifest example. The following listing is our manifest for deploying a NIDS node (Snort only):

```
node 'nids1' {
        contain base
        $IFNAME="em2"
        $IPADDR="10.66.0.2"
        $NETMASK="255.255.0.0"
        contain network
        contain snort

        file_line { 'set-interface':
                path => '/root/snort-start.sh',
                line => 'IFACE=p2p1',
                match => '^IFACE=.*',
                require => Class['snort'],
        }

        include snort-service
}
```

Listing 1: Puppet manifest for a NIDS node deployment (Snort only)

The above manifest is a minimalist view of the deployment of one NIDS node. It starts by performing all the **base configurations** (by loading the "base" class), for example install git, tcpdump, configure NTP, configure the system's locales, among others. Then, we configure our **IDS VLAN parameters**, the interface, IP address and netmask, which will then be loaded into the "network" class. Finally, it loads the "snort" class, which contains all the steps to a **successful snort installation and/or update**, this includes source code download, compilation, rules download, rsyslog configurations, etc. The last couple of lines define the interface on which Snort will listen, and the "snort-service" class makes sure that the service is constantly running.

After a more in depth analysis, it becomes clear that this is indeed a **complex part of the system**, that can't simply be put into words. Since manifests are hardly understandable by people who aren't fully aware of the Puppet language, we decided to only include a minimalist scheme of the manifests' hierarchy. This scheme can be found in Figure 26 (keep in mind that not all manifests were included in this diagram):

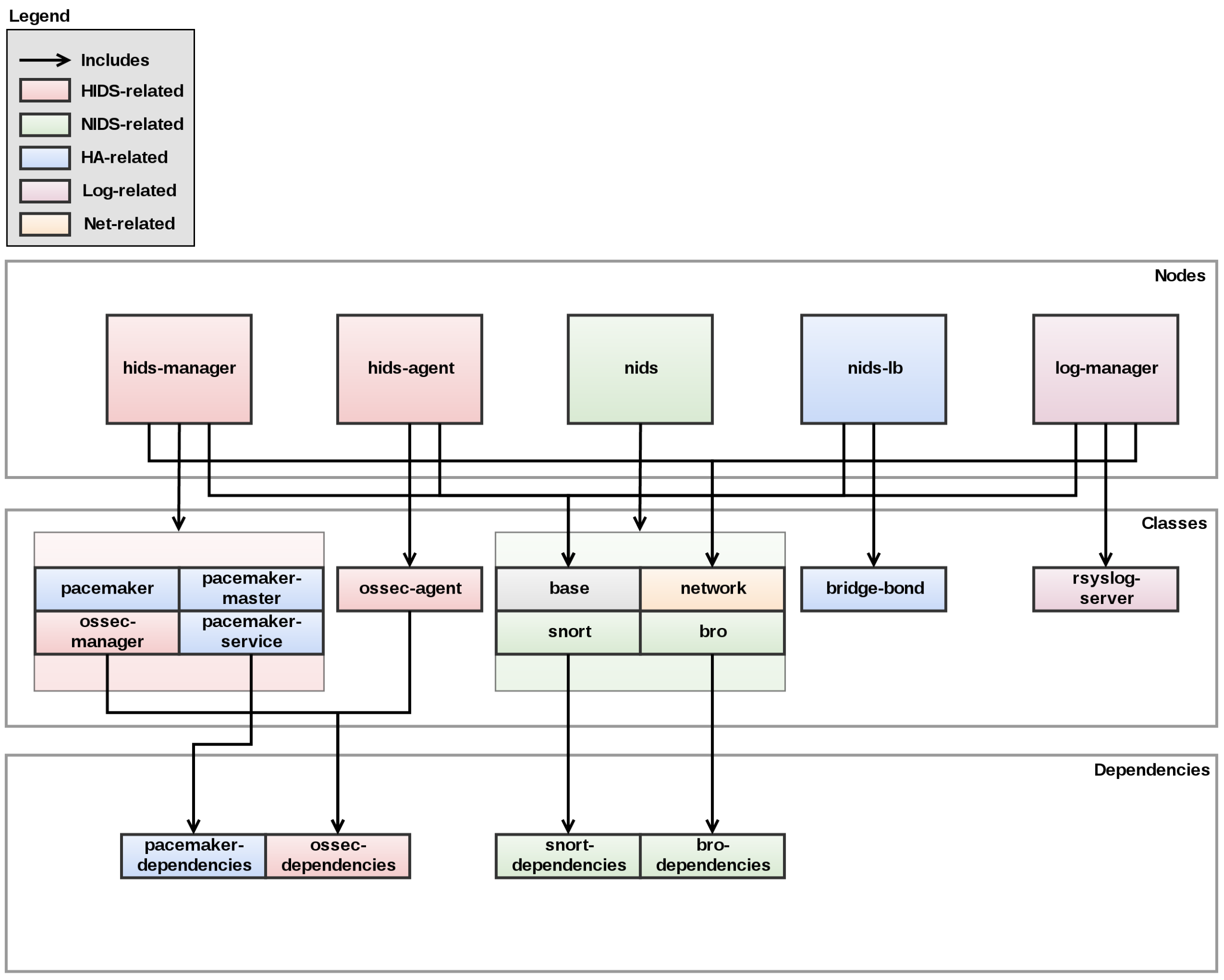


Figure 26: Puppet manifest hierarchy

At first sight, the above diagram seems complicated, but let's give it a more thoughtful analysis: The top-level corresponds to the nodes declaration, every manifest in the top-level contains declarations like the one in listing 1. **Every node also includes the "base" class**, whose purpose was explained above. H**ids-manager, nids and log-manager all include the network class**, which deploys configurations related to the IDS VLAN (for logging purposes).

Apart from these classes, starting on the left, hids-manager nodes contains: three pacemaker classes, responsible for deploying pacemaker-related configurations, and a ossec-manager class, which deploys the ossec manager's related configurations. As for the hids-agent, it contains only the ossec-agent class. Nids nodes can contain snort and/or bro, and the nids load balancer only needs the bridge and bonding configurations (for high availability and load balancing). Lastly, the log manager node contains the rsyslog-server class, which in this case, configures rsyslog to receive logs from all the other nodes, deploys and installs all the necessary scripts (See section 5.5.1 and Appendix B), installs and configures Splunk and log.io, etc.

The bottom-level is rather intuitive as it contains the manifests that are responsible for **installing all the dependencies** of their respective middle-level classes.

## 5.5 Logging

One of the core goals of this work is to develop a way to **centralize** and **standardize** all the logs generated by each IDS machine, be it a NIDS or a HIDS. The main purpose of this goal is not only to provide an easy way for system administrators to **quickly identify potential threats** inside their network and/or hosts (sparing them the burden of checking every machine and analyzing different logging formats), but also to compress the logs before being fed to our statistics server, by discarding certain information that is rather useless for statistical purposes.

Logs can be quite confusing to analyze, specially when there are multiple logging formats involved and we have to get used to all of them. All of this combined with the struggle to find the information we are looking for among different lines, timestamps, identifiers and delimiters, leaves us with a not so pleasant experience, that can easily be improved if we start looking at **system administrators as humans** and not as machines.

Lastly, one of the main goals of creating a standard logging format is the ability to combine the logs from different sources, before feeding them to a **statistical/data analyzer**, **without losing any relevant data**! This way we can **efficiently correlate them** and produce more **comprehensive and interesting statistics**, which otherwise, by feeding unfiltered logs, with multiple formats to data analyzers, wouldn't be feasible.

### 5.5.1 Standard logging format

Before heading straight to the presentation of our unified logging format, we will first introduce the raw logging formats of both HIDS and one of our NIDS (Snort) in order to identify the main issues with each one. The following listings show a random sample of a log/alert generated after an intrusion was detected by Snort (Listing 2) and OSSEC (Listing 3) respectively.

```
Jun 13 18:36:52 nids1 snort: [**] [1:2181:8] "PUA-P2P
   BitTorrent transfer" [**]
Jun 13 18:36:52 nids1 snort: [Classification: Potential
   Corporate Privacy Violation] [Priority: 1]
Jun 13 18:36:52 nids1 snort: 06/13/16-18:36:51.680065 00:1F:6C:
   BC:73:41 -> 54:A2:74:EF:19:79 type:0x8100 len:0x7E
Jun 13 18:36:52 nids1 snort: 193.137.201.212:60283 ->
   54.164.76.184:52663 TCP TTL:126 TOS:0x0 ID:26345 IpLen:20
   DgmLen:108 DF
Jun 13 18:36:52 nids1 snort: ***AP*** Seq: 0x1279604  Ack: 0
   xB6B05352  Win: 0x4029  TcpLen: 20
```

Listing 2: An alert sample generated by Snort

After a brief analysis of the previous log, we can conclude that Snort clearly presents the information in a way, such that, the intrusion **details overshadow the crucial data** for a quick anomaly identification. The intrusion in question generated five lines of logs, however **only a few contain actual human-readable information** (e.g. intrusion classification, source and destination IP addresses/ports) that is useful for statistical purposes, while the remaining data is only valuable for situations that call for a deep packet inspection (e.g. packet ID, packet length, packet time to live (TTL), source and destination mac addresses).

It should also be noted that, besides the trouble of promptly diagnosing the intrusion, this specific alert has single-handedly **generated 514 bytes worth of logs**, of which **approximately 120 bytes are duplicated information** that appears in every line, such as the log date, server and syslog tag (granted that these specific parameters were generated by rsyslog and not Snort itself).

```
Jun 13 08:52:03 hids1 ossec: ** Alert 1465804320.14911626: mail
    - syslog,linuxkernel,service_availability,
Jun 13 08:52:03 hids1 ossec: 2016 Jun 13 08:52:00 (web.dei.uc.
   pt) 10.66.0.3->/var/log/messages
Jun 13 08:52:03 hids1 ossec: Rule: 5108 (level 12) -> 'System
   running out of memory. Availability of the system is in risk
   .'
Jun 13 08:52:03 hids1 ossec: Jun 13 08:52:00 web kernel: Out of
    memory: Kill process 12813 (httpd) score 98 or sacrifice
   child
```

Listing 3: An alert sample generated by OSSEC

Despite facing the same duplicated data issue brought in by rsyslog, there's only so much that can be said about the log generated by OSSEC. With the exception of the first line, the remaining ones comprise valuable and human-readable data that rapidly describe the anomaly in question. Nevertheless, we still believe four lines for a single anomaly is not justifiable, specially since we desire for more selective and objective information.

Having said this, now is the time for us to present our solution to the problems mentioned above. Please note that **our logging solution should not be considered a replacement** for the individual application logs, but merely an underdevelopment frontend for system administrators and a way to feed data analyzers only with the information we deem necessary. The individual logs **should not be deleted in any case**, as they provide us with core data for a deep anomaly analysis, if required.

The following listing (Listing 4) presents our standard logging format, whose parameters will be described afterwards. This format was built based on recommendations from different professionals, which can be found in [106], [107] & [108].

```
Application|Server|Date|LogDate|Severity|Classification|
   Description|SrcIP|SrcName|SrcPort|DstIP|DstName|DstPort|
   Protocol
```

Listing 4: Our standard logging format parameters

- **Application**: The application that generated the alert (e.g. OSSEC, Snort, Bro)
- **Server**: The server running the application (e.g. nids1, nids2, hids1, hids2)
- **Date**: The date in which the intrusion occurred
- **LogDate**: The date in which the alert was logged
- **Severity**: The severity of the intrusion, according to each application's criteria (1 (severe) - 4 (light) for Snort, 1 (light) - 15 (severe) for OSSEC, Bro currently does not include a severity meter).
- **Classification**: A brief description of the intrusion
- **Description**: A more detailed description of the intrusion
- **SrcIP**: The source IP that generated the intrusion
- **SrcName**: The source hostname that generated the intrusion
- **SrcPort**: The source port or application that generated the intrusion
- **DstIP**: The target IP address of the intrusion
- **DstName**: The target hostname of the intrusion
- **DstPort**: The target port or application of the intrusion (e.g. 22 or sshd)
- **Protocol**: The protocol (transport layer or network layer) used by the intrusion (e.g. TCP, UDP, ICMP)

We will now take a look at the same samples shown before, when converted to our logging format.

```
snort|nids1|2016-06-13 18:36:51|2016-06-13 18:36:52|1|Potential
     Corporate Privacy Violation|PUA-P2P BitTorrent transfer
    |193.137.201.212|res-server3.res-poloii.uc.pt
    |60283|54.164.76.184|ec2-54-164-76-184.compute-1.amazonaws.
    com|52663|TCP
```

Listing 5: The snort alert sample after conversion

As we can see, the Snort alert was stripped of all the duplicated and unnecessary data, while also being **compressed by approximately 54%**(from 514 bytes to 236). Originally, Snort's logs didn't include neither source or destination hostnames, however, when converted to our logging format, **both addresses are resolved to their respective hostnames**, through a DNS resolver (all the conversion process will be explained in the next section), thus making it easier for a human to identify the involved entities.

```
ossec|hids1|2016-06-13 08:52:00|2016-06-13 08:52:03|12|System
    running out of memory. Availability of the system is in risk
    .|Out of memory: Kill process 12813 (httpd) score 98 or
    sacrifice child|None|None|None|193.137.203.248|web.dei.uc.pt
    |kernel|None
```

Listing 6: The OSSEC alert sample after conversion

The OSSEC alert on the other hand, wasn't compressed as much as Snort's, although still being summarized in one line only. Every field that is **not applicable in the intrusion context** (or simply doesn't exist) has its value set to "None", in this case specifically, source IP addresses/hostnames and protocols are not applicable to a kernel critical warning.

### 5.5.2 The logging flow

In this section, we will explain the overall flow of the logs, from the moment an intrusion is detected by either the HIDS or the NIDS, until their final stop - the statistics server and the real-time intrusion displaying server. The following subsections will then detail the applications used in every step, as well as describe the whole processing of information before it is sent out to the next phase.

The following diagram (Figure 27) shows the logging flow of our system, identifying every step the logs must perform before reaching their final destination.

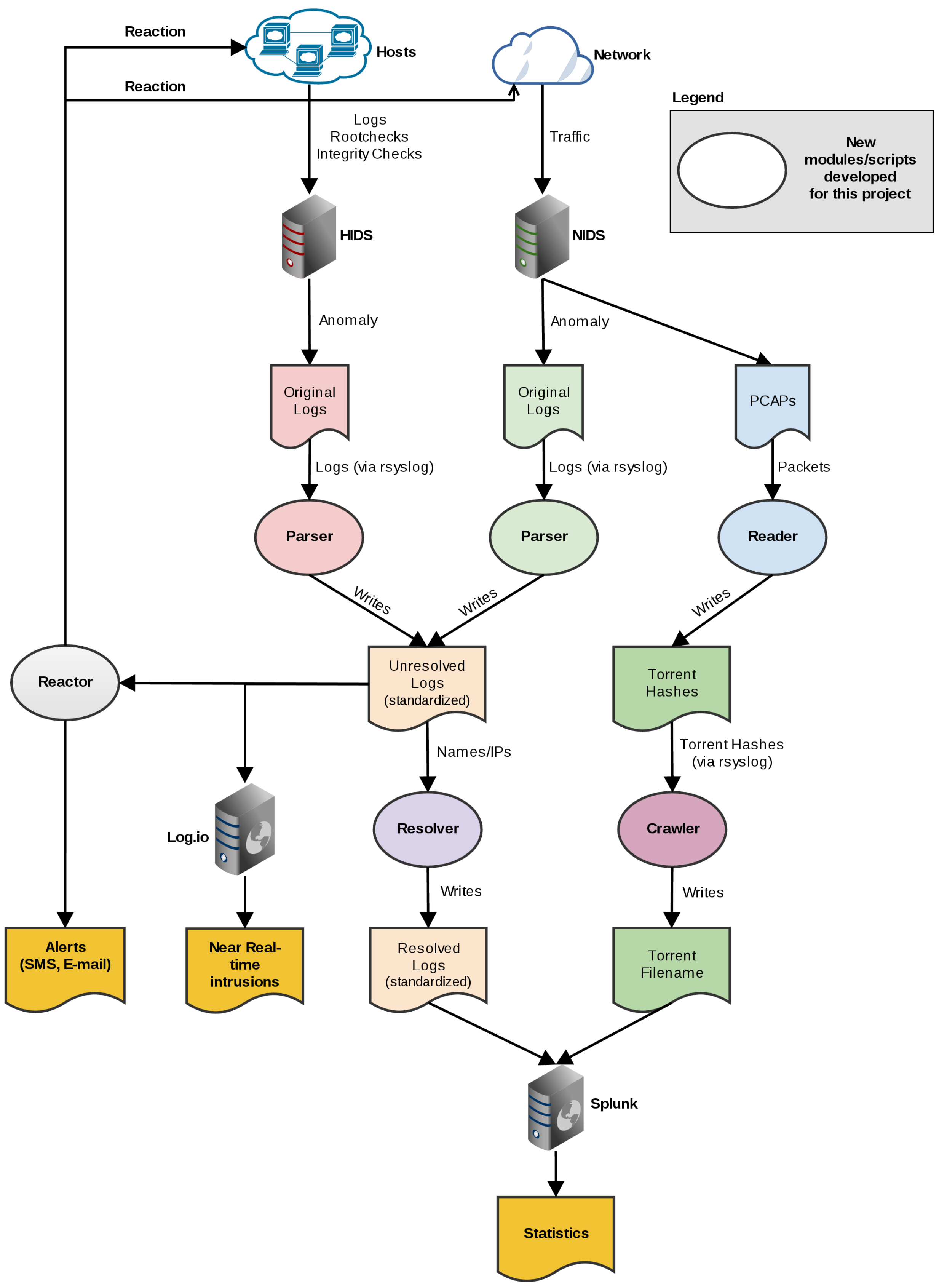


Figure 27: Log flow diagram

**Parser & Resolver**

For the sake of converting the logs in their original format (i.e. Snort's, OSSEC's, etc) into a standardized logging format, we need **parsers** that can learn and interpret each tool's logging language. With this in mind, using Python as our programming language of choice, we've managed to create **a parser for each tool** (Snort, OSSEC & Bro), that **receives logs in near real-time**, **parses them**, and **outputs** the result - **unified logs** - into an output file. Each parser was built with **flexibility** and **adaptability** in mind, and mostly consist of **predefined regular expressions**, meaning that in the future, if developers decide to update their application's logging format, we only need to update the regular expressions at our end and the parser will continue to work flawlessly.

In the light of what was stated previously, we intend to have every IP address resolved to the respective hostname and vice versa. For this reason, we built **the resolver**, a simple Python script that receives the unified logs, resolves every IP address and/or hostname (if applicable) using the DNS server assigned by DHCP, and outputs the result into the final file, which is then ready to be fed to Splunk - our data analyzer.

It's also worth mentioning that, many addresses and hostnames are unresolvable, others may take way too long to be resolved and we must keep in mind that we're handling high volumes of logs, therefore a couple of innocent five second waits may **easily accumulate and delay all our logs by hours, even days**. In order to minimize delays, we've included a **DNS caching mechanism** in our resolver, meaning that previously resolved addresses will be saved and won't be resolved again. In addition to that, we've also defined a **1 second timeout for each DNS request**. All of this combined was **still not enough** to prevent logs from being somewhat delayed, hence we've realized it's more appropriate to feed our real-time log displaying server with **only the unresolved logs** (near real-time results), whereas our statistics server is fed with the resolver's output (slightly delayed results), as seen in Figure 27.

For more technical details about this topic, please consult Appendix B that contains the developed scripts description.

**Reader & Crawler**

After taking a quick look at Figure 27, the word "torrent" comes up for the first time in this report. **BitTorrent traffic is a serious issue** in networks shared by hundreds of people, specially in academic environments, plus, it magnifies when it comes to wireless networks. This type of traffic generates massive amounts of data and **is capable of completely jamming** an entire communications infrastructure effortlessly, therefore decaying the overall quality of experience of our users and the quality of service of our network.

Besides the mentioned technical aspects of this matter, BitTorrent traffic is often understandably associated with the illegal downloading of copyrighted material, which can lead to reports and Digital Millennium Copyright Act (DMCA) takedowns carried out by content producers and Internet Service Providers (ISPs).

Although initially this problem was not planned to be dealt with, it cannot be overlooked anymore, as it represents a large chunk of the Pólo II's traffic and approximately **75% of the anomalies detected by our NIDS**, as can be seen in the following chart (Figure 28).

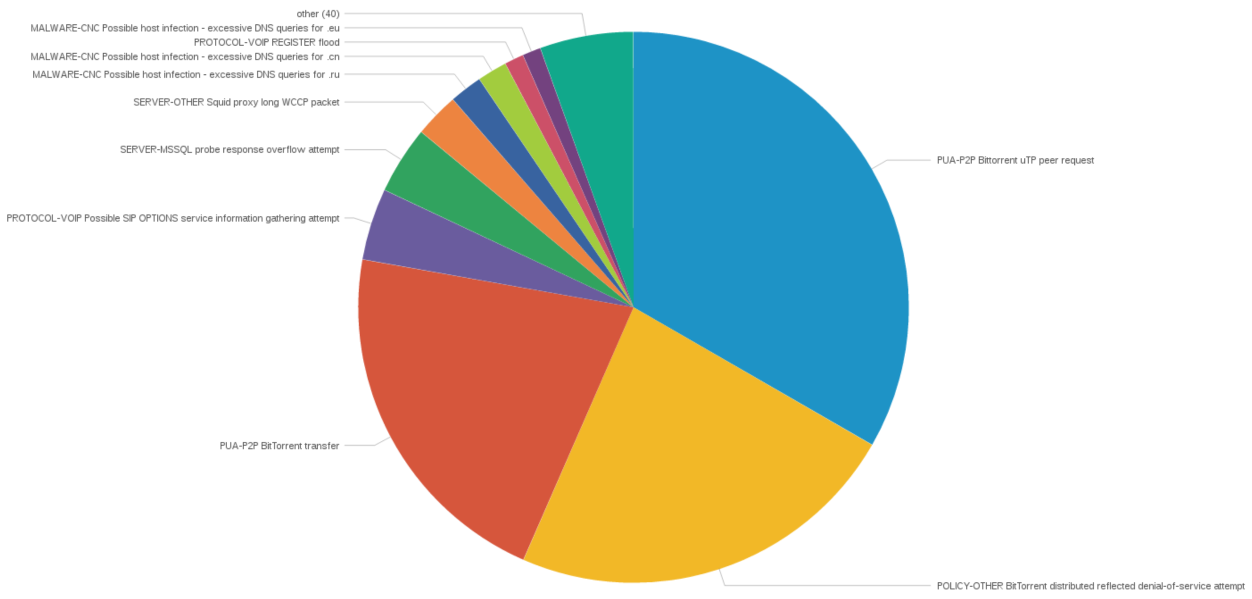


Figure 28: Top vulnerabilities detected by the NIDS

As a result of this decision, we've configured our NIDS to log the headers of every BitTorrent-related packet into a PCAP file. This file will then be interpreted by a **PCAP reader**, in this case a bash script that calls Wireshark, more specifically, its minimalist counterpart - tshark [95]. The PCAP reader will automatically extract every valuable field from the packet, this includes the **timestamp**, the **source IP address** and the **torrent hash** (a unique string that identifies a torrent file), and finally writes it all into an output file.

This data will then be sent via rsyslog to our **crawler**, whose job is to **resolve the torrent hash into its respective filename**. This will allow us to keep a record of downloaded files in the network, mapped to the respective source IP address, which will then be used to validate (or contradict) copyright claims if necessary. The crawler successfully accomplishes this, not by downloading the torrent metadata (which would generate even more unwanted BitTorrent traffic), but **by crawling search engines like Google's**, which usually contain results from popular torrent trackers, therefore containing the desired torrent filename. This behaviour is summarized in Figure 29:

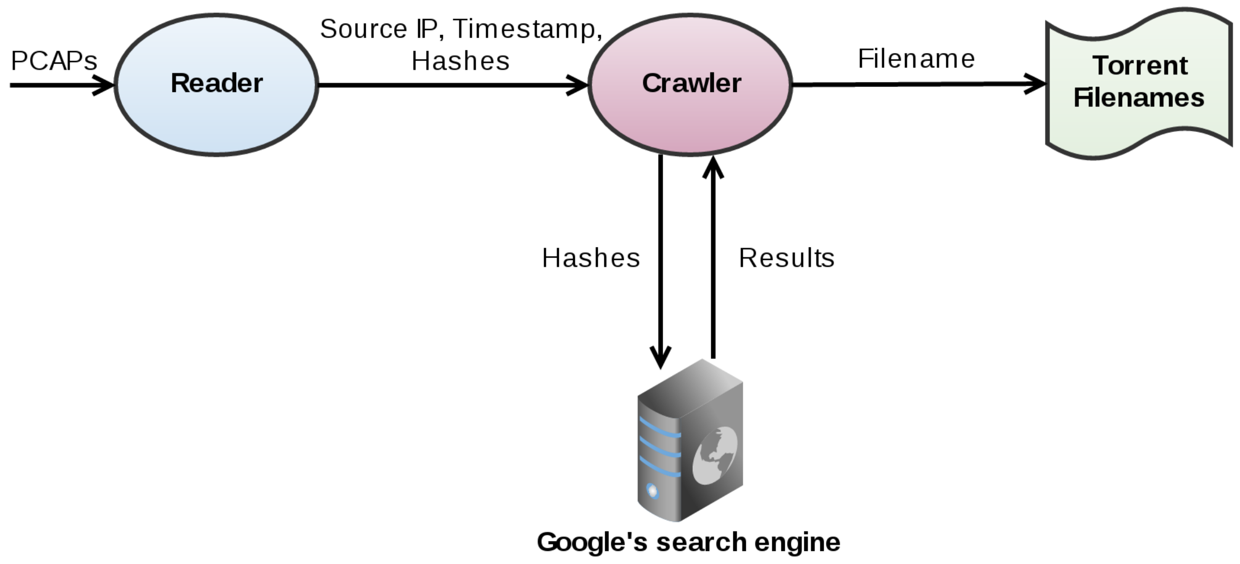


Figure 29: Reader and crawler interaction

It is a system under constant tweaking, that still has its flaws, for example, it may mistakenly choose the wrong result, when the search engine returns a bundle of results (at the moment the choice is based on **string similarity algorithms**). Another issue that is out of our control is related to the s**earch engine's timeout mechanisms**, i.e. if our crawler's throughput bypasses a certain threshold of searches, it may be temporarily blacklisted by a secure search engine like Google's. Although not being fully resolved, we've managed to mitigate this problem by implementing caching mechanisms on our end, and forcing 1 second delays (also known as sleeps) between searches.

For more technical details about this topic, please consult Appendix B that contains the developed scripts description.

## 5.6 The Reactor - Reaction/alerts engine

The Reactor is an application whose purpose is to centralize all the reactions of the IDS. It **receives the unresolved logs from the parsers** (as seen in Figure 27), and by matching each log with a set of predefined rules, it **carries out a certain action**. At the current moment, the Reactor supports various actions, such as IP address blocking, process/user session killing, the sending of SMS & e-mail and the triggering of external scripts. It is an extensible application, receiving constant tweaks and flexible enough to adapt to future changes.

There's no better way to detail our Reactor if not with an example of some rules, interpreted by the application:

```
[APPLICATION]=="snort" and [SEVERITY]==1 and [PROTOCOL]=="TCP"
   and [DSTIP]=="193.137.203.79" | block([SRCIP],"core-routers
   ",120)
[APPLICATION]=="snort" and [SEVERITY]==1 and [PROTOCOL]=="TCP"
   and [DSTIP]=="193.137.203.79" | mail("sysadmins")
```

Listing 7: rules/rules.conf - Rules to be interpreted by the Reactor

The first thing we must look at is the division performed by the pipe ("|"). To the left of the pipe, we have the **condition**, to the right we have the **action**. **The action will be carried out only if the condition is proved to be true.**

In this case, our condition is pretty much self explanatory, if the application that triggered the alert is "snort", the severity of the intrusion is "1", the protocol is "TCP" and the target is "193.137.203.79", then the Reactor will **block the IP that generated the intrusion**, in a **set of machines named "core-routers"**, for **120 minutes**. By analyzing the second rule, we can also perceive that the Reactor will also **send an e-mail** to a **contact group named "sysadmins"**. As one might have already noticed, **both the condition and the action are pure Python code!**

The tags that we see (e.g. [APPLICATION] or [DSTIP]) are simple global variables that match the parameter names chosen for our standard logging format (Listing 4).

Nonetheless, there is still one question left to be answered: How are "core-routers" or "sysadmins" defined? They are both defined in files with the JSON format, as we can see in the listings below:

```
{
        "name": "sysadmins",
        "mails": ["sysadmin1@dei.uc.pt","sysadmin2@dei.uc.pt","
           sysadmin3@dei.uc.pt"]
}
```

Listing 8: contacts/sysadmins.conf - A contact group definition

```
{
        "name": "core-routers",
        "ips": ["10.5.0.254","10.3.0.254"]
}
```

Listing 9: routers/core-routers.conf - A router group definition

That said, we can conclude that this Reactor solution **frees us from each application's reaction system**, and provides us with a much needed flexibility, since we are the ones programming the reactions and their respective triggering conditions. If at some point in the future, we decide to change our IDS applications, the Reactor will not need to be altered, unless the standard logging format changes as well!

For more technical details about this topic, please consult Appendix B that contains the developed scripts description.

## 5.7 Intrusion displaying

In this section we talk about **intrusion displaying**, basically the last step of an intrusion log in the log flow (Figure 27), which is to be displayed to system administrators, whether it is in the form of a **near real-time displaying** application or in the form of **history, statistics and/or automated reports**.

The following applications required minimal configurations, as they are essentially "plug-and-play", however we still chose to reference them in this report, since they make for a core part of our architecture.

### 5.7.1 Log.io

Log.io is our near real-time intrusion displaying application. As explained before, it consists of a web application that displays our unresolved logs in near real-time. It is composed of harvesters that *"watch log files for changes, send new log messages to the server, which broadcasts to web clients."* [80].

It is a really simple application to deploy, and very powerful when in production, as it allows for **regular expression matching in real-time**, hence enabling system administrators to filter intrusions, and only display the ones they wish to see. The following listing (Listing 10) is a sample of our log.io's harvester configuration:

```
exports.config = {
  nodeName: "hellgate.dei.uc.pt",
  logStreams: {
    "ids": [
      "/root/ids-logs/ids.log"
    ],
    "ids-resolved": [
      "/root/ids-logs/splunk/ids-full.log"
    ],
    "resolver-logs": [
      "/root/ids-logs/resolver.log"
    ],
    "exceptions": [
      "/root/ids-logs/exceptions.log"
    ],
    "torrents": [
      "/root/ids-logs/torrents.log"
    ],
    "torrent-parser-logs": [
      "/root/ids-logs/torrent-parser.log"
    ]
  },
  server: {
    host: '0.0.0.0',
    port: 28777
  }
}
```

Listing 10: log.io harvester configuration

As we can see, the harvester configuration only needs the files it is supposed to track, and then it is ready to be deployed!

### 5.7.2 Splunk

Splunk was our choice for a data analyzer and statistics producer and it was mentioned before in this report, in Section 3.1.2.

It required only basic setup, seeing that it provided a Graphical User Interface (GUI) in which we perform every configuration, without ever needing to issue any command or even accessing a terminal.

The only configurations required for Splunk were: providing the files from which it would build its internal database (in this case, the resolved intrusion logs), and "train it" to **learn the format of said files**. Splunk comes with a set of common log files format (including Snort's), however since we are dealing with our standard logging format, we had to configure it to accurately extract all the desirable parameters from our log lines - **this was done through the use of delimiters** (in our case, the pipe "|").

Considering Splunk is mostly an application that provides visual data and its effectiveness was partially shown in the preliminary results (Section 3.2), we will not cover it in depth in this report, but rather perform a demonstration of its potential in a future presentation.

# 6 Architecture validation

Validation plays a big part in the development and implementation of an architecture, and depending on the methodology used (in this case the waterfall model), it can be seen as the final step towards the completion of a project. With that said, this section will mark the conclusion of our project, as we reach **the final stages of the waterfall model**.

The following table (Table 12) briefly describes all the tests performed in order to validate every functional and non-functional requirement of the proposed architecture. A more detailed description of the tests (if applicable) can be found in the appendices of this report (this includes testing measurements, accurate results, etc.).

In the case a test fails and, consequently, the requirement is not successfully validated, it will be addressed in the next section, explaining why it failed and what are the next steps so as to correct it.

## 6.1 Testing setup

This section will briefly explain the setup behind the tests performed for the architecture validation. The diagram below (Figure 30) shows all the devices involved in the tests and the interactions between them:

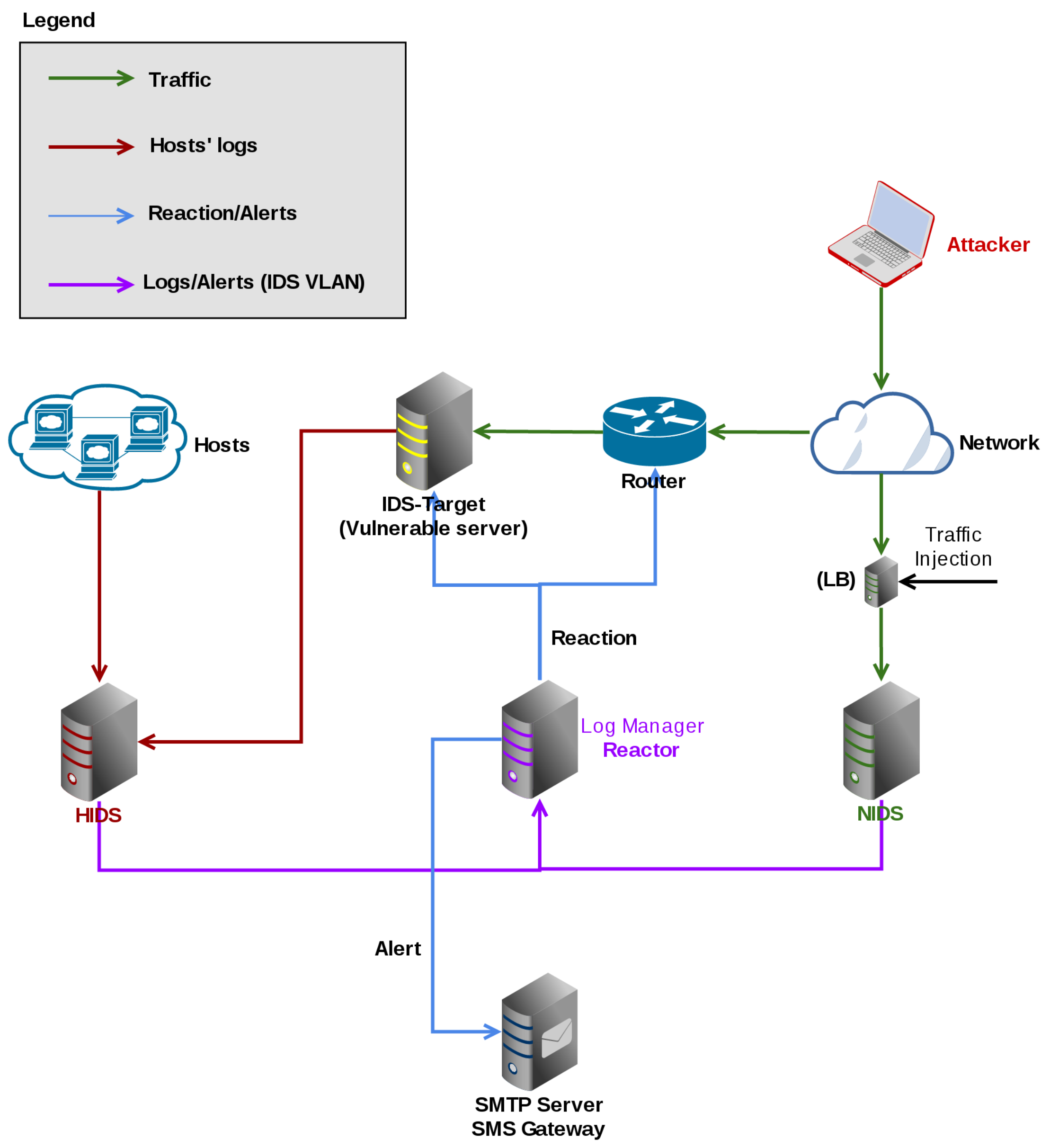


Figure 30: Testing setup diagram.

In summary, we will be using an **intentionally vulnerable test machine** (running web applications like the Damn Vulnerable Web Application (DVWA) [96] among other vulnerable services), which will be our **victim** in this testing scenario. This test machine will be monitored by our HIDS, and all the traffic reaching it is monitored by our NIDS. There is also an **attacker** that is **external to the network**, **from which every network-**

**based attack will be launched**. Lastly, we have a Router that will serve as the Reactor's proof of concept (i.e. where the Reactor will block the intruder), and an SMTP Server so as to test the alerting capabilities of the system.

## 6.2 Test list & results

| Test ID | Test Requirement | | |
|---|---|---|---|
| Description | Conditions & Input | Expected Outcome | Result |
| **#T01** | #FR01 | | |
| This test will verify that our misuse-based NIDS (Snort) is correctly detecting intrusions according to its rules. By validating the logs generated by the NIDS, we can verify if Snort is actually detecting intrusions from the data-link layer up to the application layer. | Snort's logs in production | We successfully find logs from intrusions at every layer | ✓ See A.1 |
| **#T02** | #FR02 | | |
| This test will verify that our HIDS (OSSEC) is correctly detecting anomalies at the operating system level. By validating the logs generated by the HIDS, we can verify if OSSEC is actually detecting intrusions, performing integrity checks and rootkit checks. | OSSEC's logs in production | We successfully find logs from intrusions, integrity checks and rootkit checks. | ✓ See A.2 |
| **#T03** | #FR03, #FR04 & #FR05 | | |
| This test will verify that our anomaly-based NIDS (Bro) is correctly detecting intrusions and unusual traffic. By validating the logs generated by BRO we can verify not only if it is actually detecting intrusions using anomaly-based intrusion detection techniques, but also its ability to detect abormal network traffic | Bro's logs in production | Bro correctly identifies intrusions and abnormal traffic and successfully logs the intrusion | ✓ See A.3 |
| **#T04** | #FR06 | | |
| This test will verify that our NIDS is detecting intrusions in real-time. By launching simple attacks at arbitrary hosts, we can effectively understand if our IDS is blocking the intrusions before it reaches the host itself. | For example, a SQL Injection attack directed at a test machine inside our network | The intrusion is successfully detected and stopped, before reaching its destination | Not tested **See 6.3.1** |

| Test ID | Test Requirement | | |
|---|---|---|---|
| Description | Conditions & Input | Expected Outcome | Result |
| **#T05** | #FR07 | | |
| This test will verify that our intrusion reaction system is promptly reacting to intrusions. By building simple conditions for certain intrusions (for testing purposes), we are able to verify that, in both NIDS and HIDS, our reaction system will take the appropriate measures to normalize the situation (be it via IP blocking in the case of the NIDS, proccess killing in the case of the HIDS, etc). With this test we also intend to measure the period of time between the detection of an intrusion and its respective reaction. | Normal operation of both HIDS and NIDS. Simple rules are fed to the reaction system, indicating what intrusions it must react to. | The reaction system successfully takes the appropriate measures to react to the intrusion. | ⊛ See A.5 and **6.3.2** |
| **#T06** | #FR08 | | |
| This test will verify that our logging mechanism has every parameter required by #FR08. By analyzing the logs produced by our parsers, we can verify if it contains all the parameters indicated in the functional requirement. | Log file of both HIDS and NIDS under normal operation. | Every parameter is directly or indirectly contained in the logs. | ✓ See A.4 |
| **#T07** | #FR09 | | |
| This test will verify that our alerting system is promptly alerting network managers/system administrators in the event of an intrusion. By building simple rules that map the intrusion severity to groups of contacts, we can force our alerting system to notify certain groups of people via sms and e-mail. With this test we also intend to measure the period of time between the detection of an intrusion and its respective notification. | The alerting system will be fed with rules that map an intrusion condition (e.g. severity >10) to an action (e.g. e-mail sysadmins, where sysadmins is a group of contacts). | Every person is successfully notified using the methods described in the rule. | ⊛ See A.5 and **6.3.2** |

| Test ID | Test Requirement | | |
|---|---|---|---|
| Description | Conditions & Input | Expected Outcome | Result |
| **#T08** | #FR10 | | |
| This test will validate the functionalities of our real-time intrusion displaying system (log.io). By accessing our application, we must be able to watch the intrusions in near real-time. | Normal operation of both HIDS and NIDS. Logs are constantly fed to log.io. | The application successfully shows the intrusion logs in near real-time | ✓ See A.6 |
| **#T09** | #FR11 | | |
| This test will validate the ability of producing statistics from the logs in our standard format, by our statistics server/data analyzer. By accessing our application, we must be able to watch the intrusion history and easily produce statistics. | Normal operation of both HIDS and NIDS. Logs are constantly fed to Splunk. | The application successfully shows the intrusion history and produces statistics without trouble. | ✓ See A.7 |
| **#T10** | #FR12 | | |
| This test will validate the ability of producing statistical periodic reports from the logging history, by our statistics server/data analyzer. By creating report rules, our statistics server must automatically send periodic reports to the desired key people, containing all the information required by #FR12. | Normal operation of both HIDS and NIDS. A report rule is fed to Splunk. | The application successfully generates periodic reports and sends it via e-mail to the desired key people | ✓ See A.8 |
| **#T11** | #FR13 | | |
| This test will validate the effectiveness of our centralized configuration manager (Puppet). By adding a new node to our IDS cluster (e.g. a new NIDS node), Puppet should deploy all its configurations automatically, without the need to perform individual settings in each host. | Normal operation of the system. A new node is added to the IDS cluster. | Puppet successfully deploys all the configurations and the node automatically enters in production state | ✓ See A.9 |

| Test ID | Test Requirement | | |
|---|---|---|---|
| Description | Conditions & Input | Expected Outcome | Result |
| **#T12** | #FR14 | | |
| This test will validate the security in the communication between our IDS nodes. Since most communications are performed using a private VLAN dedicated solely for this purpose, no IDS-related traffic goes through public networks. Communication between nodes is also based on public/private keys when applicable, therefore this functional requirement is already successfully verified and validated according to our current architecture implementation. | Normal operation of the system. | The communication between IDS nodes does not go through any public networks. | ✓ |
| **#T13** | Availability & Modifiability | | |
| This test will validate the availability of our IDS. In this test we will force a downtime in one of the nodes, in both HIDS and NIDS, and the system must continue to work identically as before. With this, we can also validate the first scenario of modifiability, since every downtime that it may bring is covered by availability. | A node goes down while the system is in normal operation. | In the case of the NIDS, our solution is not flawless, however it is not expected to lose any packets, **unless** we're dealing with a kernel-related issue (as explain in 5.2.1.2). In the case of the HIDS, the number of logs sent by the agents must equal the number of logs processed by the HIDS, therefore not dropping any logs in the process of rebalancing the cluster. | ✓ See A.10 |

| Test ID | Test Requirement | | |
|---|---|---|---|
| Description | Conditions & Input | Expected Outcome | Result |
| **#T14** | Modifiability | | |
| This test will validate the second scenario of modifiability. By adding a new host or network to be monitored, the system must continue to work flawlessly. | A node or network is added to the system (both HIDS and NIDS) while in normal operation. | In the case of the NIDS, the new node/network's traffic must automatically become monitored by our system. In the case of the HIDS, the new host must connect to the HIDS manager, thus being successfully added to the HIDS cluster with no service interruptions. | ✓ See A.11 |
| **#T15** | Performance | | |
| This test will validate the performance of our IDS, more specifically our NIDS. As stated in the quality attribute definition, our system must be able to handle 10Gbps of traffic. By using a traffic injection application, we will simulate this kind of situation and measure the packet loss and the CPU Load of our nodes. | High load situation, i.e. approximately 10Gbps of traffic | The IDS can analyze every packet with minimal to no packet losses. | ⊛ See A.12 and **6.3.2** |
| **#T16** | Testability | | |
| This test will validate the ability of our IDS to be tested. By using various penetration testing tools (Section 4.4.1), we will develop three sets of tests, each containing numerous intrusions for their respective category - misuse-based, anomaly-based and host-based. We will then verify the results of each test in each category. | High intrusion load situation. The network and hosts are under attack as a result of being injected with the sets of tests described before. | The IDS correctly detects every intrusion tested. | ✓ See A.13 |

| Test ID | Test Requirement | | |
|---|---|---|---|
| Description | Conditions & Input | Expected Outcome | Result |
| **#T17** | Scalability | | |
| This test will validate the ability of our system to scale and adapt to its own growth. As stated in the quality attribute definition, scalability is highly related to performance. For this reason, this test is very similar to the performance test, however this time, we will measure the CPU load and packets (NIDS) / logs (HIDS) processed per second, per node. With these numbers, we will be capable of estimating the full theoretical potential of our current architecture, as well as the theoretical limits of future architectures, when more nodes/networks/hosts are added to the system. | High load situation, i.e. approximately 10Gbps of traffic | The values measured should be roughly the same in every node, meaning that the system's load is equally distributed among its nodes. | ✓ See A.12 |

Table 12: Architecture validation tests.

## 6.3 Observed limitations

### 6.3.1 #T04

This test wasn't performed, although being a requirement that is indeed supported by our architecture, **we lacked the means to support it**, and it would be **extremely risky** to perform inline detection with the current hardware, since it would **introduce a serious delay** inside our network communications.

### 6.3.2 #T05, #T07 & #T15

These tests didn't entirely fail, thus not being marked with an ✗. The problem observed is shared amongst the three tests: **Snort under large amounts of traffic throughput** ($\sim$8Gbps).

We would like to start by explaining the slight packet loss in every phase at #T15. This is a **hardware issue** that is verified in all our Dell's nodes. We've taken every step in order to try and fix this problem (changing kernel parameters, like the TCP Offload, switching network adapters, switching SFPs, switching fiber cables) but we had no success, hence we believe the problem resides in the Dell server's motherboard.

Apart from this small issue that couldn't be avoided, by analyzing the results of these tests, we can undoubtedly spot the problem: **the CPU Load of every node is minimal**, even under such traffic throughput, which means that **Snort is apparently not making good use of the totality of the system's resources**. After some e-mail exchanges with

Snort's developers, we've been told that **Snort 3 does not yet perform internal load balancing** among the system's CPUs (although being advertised as such, as of August 2016 [79]), meaning that, at the current time, it can only use 1 CPU. This was quite a letdown for us, since each of our NIDS node consists of 12 CPU cores, where only 1 is being used.

Having said that, we cannot possibly fail these tests, since our architecture does indeed support theoretical throughput values up to 10Gbps, however, our current solution doesn't, which is why we also couldn't pass them.

On the other hand, the HIDS has successfully passed all the three tests!

# 7 Conclusion and future work

After this long year of work, I can assuredly say that we've achieved our goal! We've successfully implemented and validated the architecture proposed on section 4.3, which we are very proud of. In the end, we've accomplished nearly everything we desired - we've implemented a modern solution, for problems that were unaddressed up until now - an effective distributed intrusion detection system, covering both the DEI's hosts and the **entire Pólo II network**, something which in the very beginning was planned for a much smaller scale! Nevertheless, although we are pleased with the current result, there are still plenty of plans for the future.

For now, the major priority is to **improve the current implementation of the architecture**, so it becomes as close as possible to the proposed architecture. We will **address the current limitations** of our solution, which includes the multithreading issue on Snort, that prevents our solution from supporting 10Gbps of traffic (explained in 6.3.2). Once that is fully taken care of, to a degree such that delays are amply minimized, we will strive to **place our system inline with the network** (something that was included in the initial plan, but had to be discarded due to the risks involved), hence allowing us to block an intrusion before it even reaches its target!

One of the main goals in the short run, is also to integrate this solution with my colleague's thesis project, which implemented a new virtualization alternative in our department. This new approach gives considerably more power and independence to the end-users, by allowing them to freely create and manage their own networks and virtual hosts. There is no better time for an intrusion detection system to prove its worth, if not now.

In the long run, we aspire to **increase the coverage of our system**, possibly to the entire network of the University of Coimbra, and we can only imagine where it could go next!

As for me, personally: what an amazing year it has been! There's no way I can express my feelings at the moment in only a couple of paragraphs. In the span of a year, I've managed to erase all the doubts I had in terms of life goals, discovered my passion for networking and security and learned more than I can think of. I'm really proud of what I've accomplished, and I feel honored to have worked with such amazing supervisors. From here on out, things can only get better, the ambition will grow, the stakes will rise accordingly, but this project and these times are something I will never forget.

# References

[1] TJ OConnor, *Detecting and Responding to Data Link Layer Attacks*, SANS Institute InfoSec Reading Room, October 2010.

[2] Damon Reed, *Applying the OSI Seven Layer Network Model To Information Security*, SANS Institute InfoSec Reading Room, November 2003.

[3] Brad Bowers, *Dsniff and Switched Network Sniffing*, SANS Penetration Testing, 2000-2002.

[4] Dug Song, *Dsniff - A collection of tools for network auditing and penetration testing*, `http://www.monkey.org/~dugsong/dsniff/`, Latest release in 2000

[5] Rapid7 Team, *Metasploit - World's most used penetration testing software*, `https://www.metasploit.com/` Latest release in 2015

[6] Jouni Malinen, *hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*, `https://w1.fi/hostapd/` Latest release in 2015

[7] Thomas d'Otreppe, *Aircrack-ng - A set of tools for auditing wireless networks*, `http://www.aircrack-ng.org/` Latest release in 2015

[8] Alvaro Lopez Ortega, *GNU MAC Changer - A utility that makes the manipulation of MAC addresses of network interfaces easier.*, `https://github.com/alobbs/macchanger` Latest release in 2014

[9] Nmap Project, *Nmap: the Network Mapper - Free Security Scanner*, `https://nmap.org/` Latest release in 2015

[10] Netfilter Project, *Netfilter: Firewalling, NAT, and packet mangling for linux*, `http://www.netfilter.org/projects/iptables/index.html` Latest release in 2015

[11] Interlink Networks, *Link Layer and Network Layer Security for Wireless Networks*, Interlink Networks, LLC, 2006.

[12] IEEE Computer Society, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standards Association, 2012

[13] David C. Plummer, *RFC 826: An Ethernet Address Resolution Protocol*, Internet Engineering Task Force (IETF), November 1982.

[14] S. Frankel & S. Krishnan, *RFC 6071: P Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap*, Internet Engineering Task Force (IETF), February 2011.

[15] Information Sciences Institute, University of Southern California, *RFC 791: INTERNET PROTOCOL*, Internet Engineering Task Force (IETF), September 1981.

[16] P. Srisuresh & M. Holdrege, *RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations*, Internet Engineering Task Force (IETF), August 1999.

[17] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot & E. Lear, *RFC 1918: Address Allocation for Private Internets*, Internet Engineering Task Force (IETF), February 1996.

[18] Subramani Rao, *Denial of Service attacks and mitigation techniques: Real time implementation with detailed analysis*, SANS Institute InfoSec Reading Room, 2011.

[19] Imperva, *The Top 10 DDoS Attack Trends*, Imperva, 2015.

[20] *Unknown Author*, *ICMP Attacks Illustrated*, SANS Institute InfoSec Reading Room, 2001.

[21] J. Postel, *RFC 792: INTERNET CONTROL MESSAGE PROTOCOL*, Internet Engineering Task Force (IETF), September 1981.

[22] J. Postel, *RFC 768: User Datagram Protocol*, Internet Engineering Task Force (IETF), August 1980.

[23] Information Sciences Institute, University of Southern California, *RFC 793: TRANSMISSION CONTROL PROTOCOL*, Internet Engineering Task Force (IETF), September 1981.

[24] Shay Chen, *Application Denial of Service*, OWASP, May 2007.

[25] Fielding, et al., *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*, Internet Engineering Task Force (IETF), June 1999.

[26] Mills, et al., *RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification*, Internet Engineering Task Force (IETF), June 2010.

[27] P. Mockapetris, *RFC 1035: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*, Internet Engineering Task Force (IETF), November 1987.

[28] R. Droms, *RFC 2131: Dynamic Host Configuration Protocol*, Internet Engineering Task Force (IETF), March 1997.

[29] ISO/IEC, *ISO/IEC 9075-1:2011 Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*, International Standards Organization (ISO), December 2011.

[30] Ixia Technologies, *Taps vs. SPAN - The Forest AND the Trees: Full Visibility into Today's Networks*, Ixia, 915-3534-01 Rev. A, September 2015.

[31] Network Instruments, *ANALYZING FULL-DUPLEX NETWORKS*, Network Instruments, 2014.

[32] Dave Shackleford, *Optimized Network Monitoring for Real-World Threats*, SANS Institute InfoSec Reading Room, July 2011.

[33] Cisco Systems, *USING THE CISCO SPAN PORT FOR SAN ANALYSIS* , Cisco Systems, 2005.

[34] Richard Heady, George Luger, Arthur Maccabe & Mark Sevilla, *The Architecture of a Network Level Intrusion Detection System*, Department of Computer Science - University of New Mexico, August, 1990

[35] Monowar H. Bhuyan, D. K. Bhattacharyya & J. K. Kalita, *Network Anomaly Detection: Methods, Systems and Tools*, IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, First Quarter 2014

[36] J. Klensin, *RFC 2821: Simple Mail Transfer Protocol*, Internet Engineering Task Force (IETF), April 2001.

[37] Zdzistaw Pawlak, *Rough Sets*, International Journal of Computer and Information Sciences, Vol. 11, No. 5, 1982.

[38] Offensive Security, *Kali Linux - "The quieter you become, the more you are able to hear"*, `https://www.kali.org/`, Latest release in August, 2015

[39] Tenable, *Nessus - The Most Widely Deployed Vulnerability Assessment & Management Solution* `http://www.tenable.com/products/nessus-vulnerability-scanner` Latest release in 2015

[40] Core Security, *Core Impact Pro - Comprehensive multi-vector penetration testing*, `http://www.coresecurity.com/core-impact-pro`, Latest release in 2015

[41] Immunity Inc, *Canvas*, `http://www.immunityinc.com/products/canvas/` Latest release in November 2015

[42] Acunetix, *Acunetix*, `https://www.acunetix.com/` Latest release in November 2015

[43] Netsparker, *Netsparker - The only False-positive-free web application security scanner*, `https://www.netsparker.com/web-vulnerability-scanner/`, Latest release in 2015

[44] w3af, *w3af - A Web Application Attack and Audit Framework*, `http://w3af.org/` Latest release in 2015

[45] SecTools.Org, *SecTools.Org: Top 125 Network Security Tools* `http://sectools.org/`

[46] Snort (Cisco), *Snort: A free lightweight network intrusion detection system for UNIX and Windows*, `https://www.snort.org/`, Latest release in 2015

[47] Splunk, *Splunk: The leading platform for Operational Intelligence*, `http://www.splunk.com/`, Latest release in 2015

[48] The Bro Project, *The Bro Network Security Monitor*, `https://www.bro.org/` Latest release in September, 2015

[49] The Bro Project, *Bro introduction*, `https://www.bro.org/sphinx/intro/index.html`, January 2016

[50] ScriptRock, *Top Free Network-Based Intrusion Detection Systems (IDS) for the Enterprise*, ScriptRock, April 2015

[51] Pritika Mehra, *A brief study and comparison of Snort and Bro Open Source Network Intrusion Detection Systems*, International Journal of Advanced Research in Computer and Communication Engineering Vol. 1, Issue 6, August 2012

[52] The Suricata Team, *Suricata: Open Source IDS / IPS / NSM engine*, `http://suricata-ids.org/`, Latest release in December 2015

[53] Open Information Security Foundation, *Suricata Documentation*, `https://redmine.openinfosecfoundation.org/projects/suricata/wiki/index`

[54] OSSEC Team, *OSSEC: Open Source HIDS SECurity*, `https://ossec.github.io/`, Latest release in November 2015

[55] OSSEC Team, *OSSEC's documentation*, `https://ossec.github.io/docs/`

[56] Joe Schreiber, *Open Source Intrusion Detection Tools: A Quick Overview*, AlienVault January 13, 2014

[57] Samhain Labs, *The SAMHAIN file integrity / host-based intrusion detection system*, `http://la-samhna.de/samhain/`, Latest release in December 2015

[58] Samhain Labs, *Samhain Documentation*, `http://la-samhna.de/samhain/s_documentation.html`,

[59] Algis Kibirkstis, *Intrusion Detection FAQ: What Are The Top Selling IDS/IPS and What Differentiates Them from Each Other?*, SANS, November, 2009

[60] Cisco, *Cisco FirePOWER 8000 Series Appliances*, `http://www.cisco.com/c/en/us/products/security/firepower-8000-series-appliances/index.html`,

[61] Cisco, *Cisco FirePOWER 8000 Series Appliances Data Sheet*, `http://www.cisco.com/c/en/us/products/collateral/security/firepower-8000-series-appliances/datasheet-c78-732955.html`

[62] CISCO GPL 2015, *The Best Cisco Global Price List Checking Tool*, `http://ciscoprice.com/gpl/FP8390`

[63] IBM, *Protect your business with IBM Security*, `http://www-03.ibm.com/security/solutions/advanced-threats.html`

[64] IBM, *IBM Threat Protection System*, `http://www-03.ibm.com/security/threat-protection/?lnk=sec_home`

[65] McAfee, *McAfee Advanced Threat Defense*, `http://www.mcafee.com/us/products/advanced-threat-defense.aspx`

[66] McAfee, *McAfee Host Intrusion Prevention for Server*, `http://www.mcafee.com/us/products/host-ips-for-server.aspx`

[67] McAfee, *McAfee Advanced Threat Defense: Detect advanced targeted attacks*, `http://www.mcafee.com/us/resources/data-sheets/ds-advanced-threat-defense.pdf`

[68] Juniper Networks, *Sky Advanced Threat Prevention: Advanced malware protection from the cloud*, `https://www.juniper.net/us/en/products-services/security/sky-advanced-threat-prevention/?dd=true`

[69] Juniper Networks, *SRX650: Ideal for securing regional distributed enterprise locations*, `https://www.juniper.net/us/en/products-services/security/srx-series/srx650/`

[70] Therese R. Metcalf & Leonard J. LaPadula, *Intrusion Detection System Requirements: A Capabilities Description in Terms of the Network Monitoring and Assessment Module of CSAP21*, MITRE: Center for Integrated Intelligence Systems, September, 2000

[71] Len Bass, Paul Clements & Rick Kazman, *Software Architecture in Practice*, Addison-Wesley Professional, 2003

[72] AppNeta, *Tcpreplay - Pcap editing and replaying utilities*, `http://tcpreplay.appneta.com/`, Latest release in January, 2016

[73] OWASP, *Top 10 2013*, `https://www.owasp.org/index.php/Top_10_2013-Top_10`

[74] OWASP, *XSS (Cross Site Scripting) Prevention Cheat Sheet*, `https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet`

[75] Internet Systems Consortium, *BIND - The most widely used Name Server Software*, `https://www.isc.org/downloads/bind/`

[76] Alexandros Fragkiadakis, Sofia Nikitaki & Panagiotis Tsakalides, *Physical-layer Intrusion Detection for Wireless Networks using Compressed Sensing*, IEEE Xplore, October, 2012

[77] Aanval, *SIEM-Based Intrusion Detection: Advantages of Using Open-Source Snort and Suricata IDS/IPS in a SIEM*, `http://wiki.aanval.com/`, July 15, 2013

[78] A. A. Tomko, C. J. Rieser & L. H. Buell, *PHYSICAL-LAYER INTRUSION DETECTION IN WIRELESS NETWORKS*, IEEE Xplore, October, 2006

[79] Snort (Cisco), *Snort 3: A free lightweight network intrusion detection system for UNIX and Windows*, `https://www.snort.org/snort3.0`

[80] NarrativeScience, *log.io: Real-time log monitoring in your browser*, `http://logio.org/` Latest release in August, 2014

[81] Srivats P., *Ostinato: Packet Traffic Generator and Analyzer*, `http://ostinato.org/`, Latest release in December, 2015

[82] , Wireshark Foundation. *Wireshark · Go Deep*, https://www.wireshark.org/, Latest release in December 29, 2015

[83] Tcpdump & Libpcap, *tcpdump: A powerful command-line packet analyzer*, http://www.tcpdump.org/, Latest release in April 22, 2015

[84] PuppetLabs, *Puppet: Automate. Move Faster. Increase Reliability*, https://puppetlabs.com/puppet/puppet-open-source, Latest release in January. 2016

[85] Chef Sofware, Inc *Chef: A configuration management tool designed to bring automation to your entire infrastructure*, https://www.chef.io/, Latest release in January, 2016

[86] Bernardo Damele A. G. & Miroslav Stampar, *sqlmap: Automatic SQL injection and database takeover tool*, http://sqlmap.org/, Latest release in January, 2016

[87] ClusterLabs, *Pacemaker: A scalable high availability cluster resource manager*, http://clusterlabs.org/ Latest release in April, 2016

[88] Nmap Project, *Ncat Reference Guide*, Nmap.org, https://nmap.org/book/ncat-man.html

[89] Rafeeq Ur Rehman, *Intrusion Detection with SNORT: Advanced IDS Techniques Using SNORT, Apache, MySQL, PHP, and ACID*, Prentice Hall, May 8, 2003 http://www.informit.com/articles/article.aspx?p=101171

[90] The Bro Project, *Bro Cluster Architecture*, https://www.bro.org/sphinx/cluster/index.html, August 25, 2016

[91] cPacket Networks, *cPacket — The Next Generation of Network Performance Monitoring*, https://www.cpacket.com/

[92] Open Networking Foundation, *OpenFlow Switch Specification*, Open Networking Foundation, December 19, 2014

[93] Cisco, *Chapter: Configuring EtherChannels*, https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3750x_3560x/software/release/12-2_55_se/configuration/guide/3750xscg/swethchl.html, November 10, 2014

[94] IEEE, *802.1AX-2014 - IEEE Standard for Local and metropolitan area networks – Link Aggregation*, IEEE Standards, December 24, 2014

[95] , Wireshark Foundation. *tshark - Dump and analyze network traffic*, https://www.wireshark.org/docs/man-pages/tshark.html, Latest release in December 29, 2015

[96] , The DVWA Team. *Damn Vulnerable Web Application: DVWA*, http://www.dvwa.co.uk/, Latest release in October 5, 2015

[97] , Ferruh Mavituna, *SQL Injection Cheat Sheet*, Netsparker, https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/, March 29, 2016

[98] , Robert ”RSnake” Hansen, *XSS Filter Evasion Cheat Sheet*, OWASP, https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet, August 7, 2016

[99] , The Bro Project, *Bro Script Index*, https://www.bro.org/sphinx/scripts/policy/frameworks/dpd/detect-protocols.bro.html August 25, 2016

[100] , Michal Purzynski, *excessive_http_errors_topk.bro*, https://github.com/michalpurzynski/bro-gramming/blob/master/excessive_http_errors_topk.bro February 18, 2016

[101] , Sebastian (SuperUser), *A linux program that uses memory*, http://superuser.com/a/278965 May 4, 2011

[102] , Cisco, *README.stream5: The Stream preprocessor*, https://www.snort.org/faq/readme-stream5

[103] , The Bro Project, *Why Choose Bro?*, https://www.bro.org/why_choose_bro.pdf

[104] , spoonfork (OSSEC), *Analysis of a rootkit: Tuxkit*, OSSEC Documentation, https://ossec-docs.readthedocs.io/en/latest/rootcheck/analysis-tuxkit.html

[105] , David Schweikert, *fping: A program to send ICMP echo probes to network hosts*, https://fping.org/, Latest release in October 21, 2015

[106] , Colin Watson, Eoin Keary & Alexis Fitzgerald, *Logging Cheat Sheet*, OWASP, https://www.owasp.org/index.php/Logging_Cheat_Sheet, January 20, 2016

[107] , SANS Institute, *Information Logging Standard*, SANS Consensus Policy Resource Community, June, 2014

[108] , Karen Kent & Murugiah Souppaya, *Guide to Computer Security Log Management*, National Institute of Standards and Technology, September, 2006

[109] , Cisco, *Snort Users Manual*, The Snort Project, March 18, 2016

[110] jah, *File ntp-monlist*, Nmap.org, https://nmap.org/nsedoc/scripts/ntp-monlist.html

[111] Aleksandar Nikolic, *File ftp-brute*, Nmap.org, https://nmap.org/nsedoc/scripts/ftp-brute.html

[112] Ron Bowes, Andrew Orr & Rob Nicholls, *File http-enum*, Nmap.org, https://nmap.org/nsedoc/scripts/http-enum.html

[113] Arvid Norberg, *uTorrent Transport Protocol*, BitTorrent.org, http://www.bittorrent.org/beps/bep_0029.html, October 20, 2012

# Appendices

# A Detailed test results

## A.1 #T01: Misuse-based system test

This test consists in searching the logs generated by Snort (while it was in production for approximately 2 months) and finding samples of alerts that were generated by intrusions affecting each layer individually. Together with the log sample, we will include the rule that triggered the respective alert.

### A.1.1 Layer 2: Link Access

While Snort is considered a layer 3 and above IDS [89], it allows multiple preprocessing rules that can essentially **detect vulnerabilities even at layer 2** [109] (although most of them are yet to be ported to Snort 3).

```
[**] [112:3:1] "ARPSPOOF_ETHERFRAME_ARP_MISMATCH_DST" [**]
08/18/16-09:26:11.946194
```

Listing 11: Layer 2 vulnerability detected by Snort

The rule that triggers this alert is in the following listing, however the rule by itself is innefective without the matching preprocessor module.

```
alert ( msg:"ARPSPOOF_ETHERFRAME_ARP_MISMATCH_DST"; sid:3; gid
   :112; rev:1; metadata:rule-type preproc; classtype:bad-
   unknown; )
```

Listing 12: Layer 2 vulnerability Snort rule

### A.1.2 Layer 3: Network

The following alert is triggered by an ICMP packet, which operates at layer 3.

```
[**] [1:28463:4] "MALWARE-CNC Win.Trojan.AllAple Variant ICMP
   flood" [**]
[Classification: A Network Trojan was detected] [Priority: 1]
06/08/16-20:17:45.903710 00:1F:6C:BC:73:41 -> 54:A2:74:EF:19:79
    type:0x8100 len:0x4F
207.102.138.40 -> 193.137.201.141 ICMP TTL:65 TOS:0x80 ID:26524
    IpLen:20 DgmLen:61 DF
Type:8  Code:0  ID:62464   Seq:38187  ECHO
```

Listing 13: Layer 3 vulnerability detected by Snort

```
alert icmp $HOME_NET any -> $EXTERNAL_NET any ( msg:"MALWARE-
   CNC Win.Trojan.AllAple Variant ICMP flood"; dsize:33; itype
   :8; content:"Babcdefghijklmnopqrstuvwabcdefghi";
   detection_filter:track by_src, count 20, seconds 10;
   metadata:impact_flag red,policy balanced-ips alert,policy
   security-ips alert; reference:url,(omitted); classtype:
   trojan-activity; sid:28463; rev:4; )
```

Listing 14: Layer 3 vulnerability Snort rule

### A.1.3 Layer 4: Transport

The following alert corresponds to a Micro Transport Protocol (uTP) peer request, which is, as implied by its name, operates at layer 4 [113].

```
[**] [1:16282:4] "PUA-P2P Bittorrent uTP peer request" [**]
[Classification: Potential Corporate Privacy Violation] [
   Priority: 1]
06/07/16-18:02:40.018321 00:0F:23:F6:65:80 -> 54:A2:74:EF:19:79
    type:0x8100 len:0xA1
194.210.171.237:8999 -> 37.187.111.125:6881 UDP TTL:127 TOS:0x0
    ID:21206 IpLen:20 DgmLen:143
Len: 115
```

Listing 15: Layer 4 vulnerability detected by Snort

```
alert udp $HOME_NET any <> $EXTERNAL_NET any ( msg:"PUA-P2P
   Bittorrent uTP peer request"; content:"info_hash"; content:"
   get_peers"; reference:url,www.bittorrent.org/beps/bep_0000.
   html; classtype:policy-violation; sid:16282; rev:4; )
```

Listing 16: Layer 4 vulnerability Snort rule

### A.1.4 Layer 5: Application

The following listing contains an NTP amplification attack alert, which, just like every other application vulnerability, operates at layer 5.

```
[**] [1:29393:3] "SERVER-OTHER ntp monlist denial of service
   attempt" [**]
[Classification: Attempted Denial of Service] [Priority: 2]
06/08/16-10:02:06.136632 00:1F:6C:BC:73:41 -> 30:E4:DB:97:41:FF
    type:0x8100 len:0x40
185.94.111.1:55677 -> 193.137.201.143:123 UDP TTL:3 TOS:0x80 ID
   :54321 IpLen:20 DgmLen:36
Len: 8
```

Listing 17: Layer 5 vulnerability detected by Snort

```
alert udp $EXTERNAL_NET any -> $HOME_NET 123 ( msg:"SERVER-
   OTHER ntp monlist denial of service attempt"; flow:to_server
   ; content:"|17 00 03 2A|",depth 4; detection_filter:track
   by_dst, count 1000, seconds 5; metadata:service ntp;
   reference:cve,2013-5211; classtype:attempted-dos; sid:29393;
    rev:3; )
```

Listing 18: Layer 5 vulnerability Snort rule

## A.2 #T02: Host-based system test

This test consists in searching the logs generated by OSSEC and finding samples of alerts that correspond to general intrusions, integrity checks and rootkit checks.

### A.2.1 General intrusions

In this sample, we can see that OSSEC can also detect bruteforcing attempts, in this case SSH bruteforcing to one of our servers.

```
** Alert 1466456445.86015967: mail  - syslog,attacks,
2016 Jun 20 22:00:45 (eden.dei.uc.pt) 10.66.0.13->/var/log/
   secure
Rule: 40112 (level 12) -> 'Multiple authentication failures
   followed by a success.'
Src IP: 94.61.126.233
User: (omitted)
```

Listing 19: SSH bruteforcing detected by OSSEC

Besides intrusions, OSSEC will also trigger an alert when the system's availability is at risk, as shown in the following listing.

```
** Alert 1466169370.30860640: mail  - syslog,linuxkernel,
   service_availability,
2016 Jun 17 14:16:10 (web.dei.uc.pt) 10.66.0.3->/var/log/
   messages
Rule: 5108 (level 12) -> 'System running out of memory.
   Availability of the system is in risk.'
Jun 17 14:16:06 web kernel: Out of memory: Kill process 28667 (
   httpd) score 85 or sacrifice child
```

Listing 20: System running out of RAM detected by OSSEC

### A.2.2 Integrity Checks

In the following sample, we can see OSSEC detecting changes in a critical file in the system, by performing an integrity check.

```
** Alert 1465540907.12843309: mail  - ossec,syscheck,
2016 Jun 10 07:41:47 (eden.dei.uc.pt) 10.66.0.13->syscheck
Rule: 551 (level 7) -> 'Integrity checksum changed again (2nd
   time).'
Integrity checksum changed for: '/etc/passwd'
Size changed from '75690' to '75760'
Old md5sum was: '178e615b0405e034857af2900c33f827'
New md5sum is : '0d120d4e05e505dbf5eef38d0a7f9251'
```

Listing 21: Integiry checks by OSSEC

### A.2.3 Rootkit Checks

The following listing shows OSSEC performing a rootkit check and detecting a possible kernel level rootkit in one of our SMTP servers.

```
** Alert 1467303008.58246406: mail  - ossec,rootcheck,
2016 Jun 30 17:10:08 (smtp.dei.uc.pt) 10.66.0.5->rootcheck
Rule: 510 (level 7) -> 'Host-based anomaly detection event (
   rootcheck).'
Process '13655' hidden from /proc. Possible kernel level
   rootkit.
```

Listing 22: Rootkit checks by OSSEC

## A.3 #T03: Anomaly-based system test

This test consists in searching the logs generated by Bro and finding samples of alerts that correspond to intrusions (detected using anomaly-based detection strategies) and unusual traffic.

The following notice was triggered after an HTTP server replied with an **unusual amount of HTTP errors** to the same requestor.

```
1471513597.962577       -       -       -       -       -
        -       -       -       -       MozillaHTTPErrors::
   Excessive_HTTP_Errors_Victim Excessive HTTP errors for
   requests to 192.168.1.2       0 in 5.0 mins, eps: 0
   192.168.1.2     -       -       -       bro     Notice::
   ACTION_LOG      60.000000       F       -
```

Listing 23: Excessive HTTP errors detected by BRO

The following notice was triggered due to a host being targeted by a port scan.

```
1471513574.920176       -       -       -       -       -
        -       -       -       -       Scan::Port_Scan
   192.168.1.7 scanned at least 15 unique ports of host
   192.168.1.2 in 0m0s        remote  192.168.1.7192.168.1.2  -
        -       bro     Notice::ACTION_LOG      3600.000000
       F       -       -       -       -       -
```

Listing 24: Port scanning detected by BRO

Bruteforcing is also included in the category of unusual traffic, since an FTP server under a bruteforcing environment will send an **unusual amount of error replies** in a short period of time (i.e. "Invalid username" and/or "Invalid password" replies).

```
1471512262.996273       -       -       -       -       -
        -       -       -       -       FTP::Bruteforcing
        192.168.1.7 had 20 failed logins on 1 FTP server in 0
   m6s        -       192.168.1.7     -       --      bro
   Notice::ACTION_LOG      3600.000000     F       -       -
        -       -       -
```

Listing 25: Bruteforcing detected by BRO

Lastly, BRO was also able to detect anomalies in many protocols, of which SSL/TLS, as shown in the following listing.

```
1471517048.430518       COVXlI1TlPQOU7MS6h      193.136.230.208
    55741   40.113.91.177   443     tcp     SSL     Invalid
   version in TLS connection. Version: 59320
1471517048.600086       C6rBS32zdmWs24MHll      193.136.230.208
    55744   40.113.91.177   443     tcp     SSL     Invalid
   headers in SSL connection. Head1: 20, head2: 211, head3: 115
```

Listing 26: SSL/TLS anomalies detected by BRO

## A.4 #T06: Logging mechanism validation

This is a simple observation test that consists in verifying that every parameter required in #FR08 is present in our standard logging format. This is also verified in Section 5.5.1.

## A.5 #T05 & #T07: Reaction/Alerting test

In order to test the reaction/alerting times of our system, we fed our Reactor with only four simple rules:

```
[APPLICATION]=="snort" and [DSTIP]=="193.137.203.79" | block([
   SRCIP],"test",120)
[APPLICATION]=="ossec" and [DSTIP]=="193.137.203.79" | block([
   SRCIP],"test",120)
[APPLICATION]=="bro" and [DSTIP]=="193.137.203.79" | block([
   SRCIP],"test",120)
[DSTIP]=="193.137.203.79" | mail("sysadmins")
```

Listing 27: Reactor testing rules

In summary, this would trigger a reaction from our Reactor only when the destination IP address of the intrusion equals "193.137.203.79", which corresponds to our testing machine. The Reactor will then block the intruder's source IP on every device inside the group "test", and will also send an e-mail alert to every contact inside the group "sysadmins".

The table below (Table 13) shows the **reaction times of our three IDS applications**. The following values were calculated using a sample of **100000 logs**, gathered from a bulk generated over a period of approximately 2 months (while the system was in production).

| Application | Average reaction time (Delay) | Standard deviation |
|---|---|---|
| **Snort** | 4.95s | 2.92s |
| **OSSEC** | 5.18s | 3.08s |
| **Bro** | 6.91s | 1.00s |

Table 13: Reaction times of the three IDS applications.

After analyzing the above data, we can verify that Snort was indeed our **fastest application** in terms of reaction/alerting time, however, we must take into account that these tests were performed during a period where network occupation at Pólo II was hitting its **lowest throughput values in the entire year** (July-August), therefore we **cannot consider it a representative result** of Snort's overall performance. Having said this, we decided to go further and perform a more in depth reaction test, only targeting Snort.

This test was divided in **three phases: 100Mbps, 1000Mbps and MAX**, each phase corresponding to a different traffic throughput. In each phase, the respective traffic throughput was injected into our load balancer over a period of **5 minutes**. During these 5 minutes, **10 intrusions were injected** together with the remaining traffic at **random times** (in order to minimize the impact of rsyslog's pooling interval that is explained below).

We should note that rsyslog is continuously pooling intrusion logs and sending them to our Reactor **every second**, which means that the following reaction times also include the "1 second pooling interval" from rsyslog.

### A.5.1 Phase 1 - 100Mbps: Results

| **Average throughput:** 119Mbps | | | | |
|---|---|---|---|---|
| Intrusion # | Arrival date * | Logging Date * | Reaction Date * | Delay ** |
| **1** | :24 | :33 | :33 | 9 |
| **2** | :44 | :45 | :45 | 1 |
| **3** | :56 | :03 | :03 | 7 |
| **4** | :18 | :23 | :23 | 5 |
| **5** | :08 | :15 | :15 | 7 |
| **6** | :26 | :35 | :35 | 9 |
| **7** | :03 | :13 | :13 | 10 |
| **8** | :00 | :03 | :04 | 4 |
| **9** | :31 | :34 | :34 | 3 |
| **10** | :04 | :05 | :05 | 1 |
| **CPU Load at NIDS1:** 0.38 (3.17%) | | | | |
| **CPU Load at NIDS2:** 0.42 (3.5%) | | | | |
| **Average delay:** 5.6s | | | | |
| **Standard deviation:** 3.31s | | | | |

* *Not the full date, only the seconds part*
** *The delay between the arrival of the packet and the respective reaction, in seconds*

Table 14: Phase 1 results

### A.5.2 Phase 2 - 1000Mbps: Results

| **Average throughput:** 990Mbps | | | | |
|---|---|---|---|---|
| Intrusion # | Arrival date * | Logging Date * | Reaction Date * | Delay ** |
| **1** | :23 | :24 | :24 | 1 |
| **2** | :56 | :06 | :06 | 10 |
| **3** | :06 | :15 | :15 | 9 |
| **4** | :30 | :34 | :34 | 4 |
| **5** | :52 | :55 | :56 | 4 |
| **6** | :26 | :34 | :34 | 8 |
| **7** | :43 | :46 | :46 | 3 |
| **8** | :03 | :06 | :06 | 3 |
| **9** | :36 | :44 | :44 | 8 |
| **10** | :19 | :26 | :26 | 7 |
| **CPU Load at NIDS1:** 0.64 (5.33%) | | | | |
| **CPU Load at NIDS2:** 0.65 (5.42%) | | | | |
| **Average delay:** 5.7s | | | | |
| **Standard deviation:** 3.06s | | | | |

* *Not the full date, only the seconds part*
** *The delay between the arrival of the packet and the respective reaction, in seconds*

Table 15: Phase 2 results

### A.5.3 Phase 3 - MAX: Results

| **Average throughput:** 7.87Gbps *** | | | | |
|---|---|---|---|---|
| Intrusion # | Arrival date * | Logging Date * | Reaction Date * | Delay ** |
| **1** | :15 | :24 | :24 | 9 |
| **2** | :00 | :05 | :05 | 5 |
| **3** | :44 | :46 | :46 | 2 |
| **4** | :58 | :04 | :05 | 7 |
| **5** | :17 | :24 | :25 | 8 |
| **6** | N/A (queued) | N/A (queued) | N/A (queued) | N/A (queued) |
| **7** | N/A (queued) | N/A (queued) | N/A (queued) | N/A (queued) |
| **8** | N/A (queued) | N/A (queued) | N/A (queued) | N/A (queued) |
| **9** | N/A (queued) | N/A (queued) | N/A (queued) | N/A (queued) |
| **10** | N/A (queued) | N/A (queued) | N/A (queued) | N/A (queued) |
| **CPU Load at NIDS1:** 0.87 (7.25%) | | | | |
| **CPU Load at NIDS2:** 0.89 (7.42%) | | | | |
| **Average delay: N/A** | | | | |
| **Standard deviation:** N/A | | | | |

* *Not the full date, only the seconds part*
** *The delay between the arrival of the packet and the respective reaction, in seconds*
*** *Represents the maximum throughput of traffic injected with the current hardware. 10Gbps would be possible with more processing power.*

Table 16: Phase 3 results

Following the analysis of all the three tables, we understand that Snort keeps a rather constant average delay between Phase 1 and Phase 2, however in Phase 3, we can see that the **delay can't be determined** due to missing values (since intrusive packets are being queued). In every phase, the CPU Load is minimal, reaching a **maximum of 7.42%**. We're in the presence of a **limitation in our current implementation**, which is explained in Section 6.3.2.

## A.6 #T08: Intrusion displaying application test

This is a simple observation test that consists in verifying that our real-time intrusion displaying application (log.io) is correctly presenting all the logs/intrusions in near real-time. Since the application is in same server that receives all the logs, there is minimal delay between their arrival and presentation, therefore we don't feel the need to perform measurements in this test.

## A.7 #T09: Statistics server/data analyzer test

This is a simple observation test that consists in verifying that our statistics application (Splunk) is correctly showing the intrusion history and producing the statistics required by #FR11. This is easily verifiable by just accessing the application, and the results are as follows:

| Test | Result |
|---|---|
| Generate top vulnerabilities statistic | ✓ |
| Generate top target machines statistic | ✓ |
| Generate top target services/ports statistic | ✓ |
| Generate top countries of origin statistic | ✓ |

Table 17: Statistics producing test.

## A.8 #T10: Automatic reporting generation test

This is a simple observation test that consists in configuring Splunk to generate periodic reports containing the information required by #FR12. This test is easily verifiable since we rightly receive said reports and can successfully analyze all the required parameters.

## A.9 #T11: Configuration manager test

This is a simple observation test that consists in verifying that Puppet is properly deploying all the configurations to a new node that is added to the pool. This test has passed since the moment we started developing the IDS, since it is our core service for deploying all the configurations.

## A.10 #T13: Availability & Modifiability test

### A.10.1 NIDS

As explained in section 5.2.1.2, forcing an unpredictable shutdown in one of the NIDS nodes, in a way that the connection link goes down, **will not affect the availability of our system**, since the traffic will be correctly distributed among the remaining nodes, almost instantly (thus we don't feel the need to measure this).

Nevertheless there is still one scenario we must consider: The scenario where a kernel-related issue occurs, in this case we simulated a network driver crash in one of our nodes (by unloading the respective kernel module). Keep in mind that **this will keep the connection link established** and the machine will still be **operational in a layer 2 perspective**.

This is a **very unlikely** scenario to occur, but we'd still rather be prepared for it.

We once again decided to split the test in **three phases**, each corresponding to a traffic throughput (100Mbps, 1000Mbps and MAX) and **measure the packet loss and the delay before the load is rebalanced among the remaining nodes**, in each phase, as soon as one node becomes unable to reply to pings.

#### A.10.1.1 Phase 1 - 100Mbps: Results

| **Average throughput:** 106Mbps | |
|---|---|
| Metric | Value |
| **Packets Sent** | 250116 (229.95 MB) |
| **Packets Received** | 244713 (224.50 MB) |
| **Packets Lost** | 5403 **(5.45 MB)** |
| **Delay** | **0.41s** |
| **Service Downtime** | 0s (0%) |

Table 18: Phase 1 results

#### A.10.1.2 Phase 2 - 1000Mbps: Results

| **Average throughput:** 980Mbps | |
|---|---|
| Metric | Value |
| **Packets Sent** | 1391764 (1.88 GB∼1.98 GB *) |
| **Packets Received** | 1395815 (1.88 GB∼1.98 GB *) |
| **Packets Lost** | 4051 **(0∼96.64 MB *)** |
| **Delay** | **0s∼0.79s *** |
| **Service Downtime** | 0s (0%) |

* *Not enough granularity to find the real value.*

Table 19: Phase 2 results

#### A.10.1.3 Phase 3 - MAX: Results

| **Average throughput:** 8.20Gbps ** | |
|---|---|
| Metric | Value |
| **Packets Sent** | 9279782 (13.90 GB~14.00 GB *) |
| **Packets Received** | 9178824 (13.69 GB~13.79 GB *) |
| **Packets Lost** | 100958 **(118.11 MB~311.39 MB *)** |
| **Delay** | **0.11s~0.30s** * |
| **Service Downtime** | 0s (0%) |

* *Not enough granularity to find the real value.*
** *Represents the maximum throughput of traffic injected with the current hardware. 10Gbps would be possible with more processing power.*

Table 20: Phase 3 results

After analyzing all the previous tables, we realize that the delay before the load balancer rebalances the load in this scenario is minimal, with an **average minimum of approximately 0.17s** and an **average maximum of 0.5s**, utterly independent from the traffic throughput. This translates into the slightest packet loss, even at speeds reaching 8.20Gbps, which makes us very proud of this solution!

### A.10.2 HIDS

For the HIDS availability test, we will once again force an unpredictable shutdown in one of the HIDS nodes and verify if the service as a whole continues serving its purpose. Since we couldn't find a favorable way to correlate the logs sent by every agent to the logs processed by each node, we determined that a good way to measure the lost logs in the transition process, would be to **force the insertion of logs** (e.g. by intentionally making errors) in one of our test machines and validate if each log triggered its corresponding OSSEC alert.

While generating a **constant stream of logs (roughly 2 per second)**, we took down the HIDS node responsible for the test machine, and the results are presented in the table below:

| Metric | Value |
|---|---|
| **Logs Generated** | 23 |
| **Logs Processed (Alerts triggered)** | 23 |
| **Logs Dropped** | 0 |
| **Service Downtime** | 0s (0%) |

Table 21: HIDS Availability test results.

By analyzing the previous table, we can see that **0 logs were dropped** while transferring the resources from one node to the other! We find it amazing how fast IP availability works, and how seamlessly the cluster rebalancing is done. As a further proof, we included the times of each log in the following listing. We can observe that, the logs are being processed by HIDS1, and at some point (the failure point), they start being processed by HIDS2.

```
[root@hids1 ~]# tail -f /var/ossec/logs/alerts/alerts.log |
   grep -i "(ids-target) 10.66.0.17"
23:03:43 (ids-target) 10.66.0.17->/var/log/secure
23:03:43 (ids-target) 10.66.0.17->/var/log/secure
23:03:45 (ids-target) 10.66.0.17->/var/log/secure
23:03:45 (ids-target) 10.66.0.17->/var/log/secure
23:03:47 (ids-target) 10.66.0.17->/var/log/secure
23:03:47 (ids-target) 10.66.0.17->/var/log/secure
23:03:49 (ids-target) 10.66.0.17->/var/log/secure
23:03:49 (ids-target) 10.66.0.17->/var/log/secure
23:03:49 (ids-target) 10.66.0.17->/var/log/secure
23:03:51 (ids-target) 10.66.0.17->/var/log/secure
23:03:51 (ids-target) 10.66.0.17->/var/log/secure
23:03:53 (ids-target) 10.66.0.17->/var/log/secure
23:03:53 (ids-target) 10.66.0.17->/var/log/secure
Connection to hids1 closed by remote host.
Connection to hids1 closed.

>>>>> HIDS1 FAILURE <<<<<

[root@hids2 ~]# tail -f /var/ossec/logs/alerts/alerts.log |
   grep -i "(ids-target) 10.66.0.17"
23:03:55 (ids-target) 10.66.0.17->/var/log/secure
23:03:55 (ids-target) 10.66.0.17->/var/log/secure
23:03:57 (ids-target) 10.66.0.17->/var/log/secure
23:03:57 (ids-target) 10.66.0.17->/var/log/secure
23:03:57 (ids-target) 10.66.0.17->/var/log/secure
23:03:59 (ids-target) 10.66.0.17->/var/log/secure
23:03:59 (ids-target) 10.66.0.17->/var/log/secure
23:04:01 (ids-target) 10.66.0.17->/var/log/secure
23:04:01 (ids-target) 10.66.0.17->/var/log/secure
23:04:03 (ids-target) 10.66.0.17->/var/log/secure
```

Listing 28: Logs processed by OSSEC in the availability test

## A.11 #T14: Modifiability test

This is a simple observation test that has once again passed since the moment we started developing the system, after all, this represents a big part of what we have done until now - adding hosts and networks to be monitored.

## A.12 #T15 & #T17: Performance and scalability test

### A.12.1 NIDS

As we stated before, performance and scalability tests were performed simultaneously due to their similarity, the only difference being the metrics collected for each test.

The NIDS' performance and scalability test was divided in **three phases: 100Mbps, 1000Mbps and MAX**, each phase corresponding to a different traffic throughput. In each phase, the respective traffic throughput was injected into our load balancer over a period of **5 minutes**. Please note that the even though we tried to keep a stable throughput over the established period, that was not always possible, since every test was performed **while the system was in production**, meaning that "real traffic" was also taken into account during this test.

#### A.12.1.1 Phase 1 - 100Mbps: Results

| **Average throughput:** 112Mbps | | |
|---|---|---|
| Metric | **NIDS1** | **NIDS2** |
| **Packets Sent** | 2489108 | 2488597 |
| **Packets Received** | 2415532 | 2415477 |
| **Packets Analyzed** | 2415532 (8051.77/second) | 2415477 (8051.59/second) |
| **Packets Queued** | 0 | 0 |
| **Packets Dropped** | 73576 (2.956%) | 73120 (2.938%) |
| **CPU Load** | 0.43 (3.58%) | 0.39 (3.25%) |

Table 22: Phase 1 results

#### A.12.1.2 Phase 2 - 1000Mbps: Results

| **Average throughput:** 966Mbps | | |
|---|---|---|
| Metric | **NIDS1** | **NIDS2** |
| **Packets Sent** | 13379540 | 13379749 |
| **Packets Received** | 13330682 | 13330571 |
| **Packets Analyzed** | 13330371 (44434.57/second) | 13329290 (44430.97/second) |
| **Packets Queued** | 311 (0.0023%) | 1281 (0.0096%) |
| **Packets Dropped** | 48858 (0.367%) | 49178 (0.377%) |
| **CPU Load** | 0.55 (4.58%) | 0.60 (5%) |

Table 23: Phase 2 results

#### A.12.1.3 Phase 3 - MAX: Results

| **Average throughput:** 7.95Gbps * | | |
|---|---|---|
| Metric | **NIDS1** | **NIDS2** |
| **Packets Sent** | 106665757 | 106664785 |
| **Packets Received** | 106618316 | 106618588 |
| **Packets Analyzed** | 19713757 (65712.52/second) | 18178468 (60594.89/second) |
| **Packets Queued** | 86904559 (81.51%) | 88440120 (82.95%) |
| **Packets Dropped** | 47441 (0.044%) | 46197 (0.043%) |
| **CPU Load** | 0.78 (6.5%) | 0.79 (6.58%) |

* *Represents the maximum throughput of traffic injected with the current hardware. 10Gbps would be possible with more processing power.*

Table 24: Phase 3 results

After analyzing all the three tables, we understand that the NIDS has a slight packet loss regardless of the traffic throughput. We can also observe that, when speeds reach values such as 7.95Gbps, **packets start being placed into a queue**, reaching values as high as **80% of the packets sent**! In every phase, the CPU Load is minimal, reaching a **maximum of 6.58%**. We're in the presence of a **limitation in our current implementation**, which is explained in Section 6.3.2.

### A.12.2 HIDS

The HIDS scalability test involves verifying that both nodes are analyzing similar amounts of logs, under a similar CPU load.

As explained before, on Section 5.2.2, the load balancing solution when it comes to the HIDS is a combination of Pacemaker and Corosync, and it is **configured to perform the load distribution via IP address hash**. In summary, each HIDS node is responsible for a specific set of agents, i.e. if there is a total of 30 agents/servers, and only two HIDS nodes, then each node will be responsible for 15 agents. This essentially means that, **the load balancing is not performed on the amount logs sent by each agent, but on the amount of agents**. Having said this, it is expected that the number of logs processed by each node will have a reasonable difference, since a popular agent may generate more logs than two or three "less popular" ones combined.

This test involved **11 agents (and over 320 log sources)** and had a total duration of **five minutes**, where we gathered the CPU load and the number of logs processed by each node. These values vary depending on the time of the day, since it relies on the utilization of each server.

| Metric | HIDS1 | HIDS2 |
|---|---|---|
| CPU Load | 0.03 (0.25%) | 0.05 (0.42%) |
| Logs Processed | 1775 (5.92 logs/second) | 1538 (5.13 logs/second) |

Table 25: Detailed HIDS scalability testing.

Through the analysis of the previous table, we can conclude that our system's load is **properly distributed among our nodes**. We can also observe that both our nodes are **under minimal CPU Load**, meaning that we are assumably wasting hardware resources on a system that doesn't really need them, at least for now. In the future, after adding all the remaining core servers, this will most likely not be the case.

## A.13 #T16: Testability tests

As stated in the architecture validation tests (Table 12), testability validation involves creating a test batch for each detection strategy that our system implements. We will now go through each detection strategy and detail their respective testing procedure.

### A.13.1 Misuse-based (Snort)

As we know by now, Snort implements a misuse-based detection strategy, meaning that it mostly analyzes each packet individually from the rest, and matches it against several predefined rules. In order to validate our misuse-base detection efficiency, we chose a couple of vulnerabilities and penetration testing tools (Section 4.4.1) that will help us achieve that purpose.

| Vulnerability | Available tools | Description | Result |
|---|---|---|---|
| SQL Injection | ncat [88], Nmap, sqlmap, Metasploit Framework | Easily the most common application layer vulnerability (according to OWASP's top 10 [73]) and one we surely don't want to miss. We start by launching simple SQL Injection variations and then use one of the available tools in order to begin launching more developed and sneakier attacks. | ✓ (See A.13.1.1) |
| Cross-site Scripting | ncat [88], Nmap, w3af, Nessus | Similar to the previous test, we start by launching simple XSS variations and then switch to one of the available tools in order to begin launching more developed and sneakier attacks. | ✓ (See A.13.1.2) |
| Amplification Attack (NTP) | Nmap, Metasploit Framework | With the help of the available tools, we perform NTP amplification attacks against a vulnerable NTP server. | ✓ (See A.13.1.3) |

Table 26: Misuse-based detection validation.

#### A.13.1.1 SQL Injection: Detailed results

We started by testing common SQL injection queries that can be found in [97]. Out of all the available penetration testing tools, **we chose sqlmap** for a more in depth SQL Injection test, as it is a renowned tool in the SQL injection field. The first sqlmap run consists in a **minimal run**, that stops after the first vulnerable point is found. The second sqlmap run consists of a **full run**, which only stops when every possible injection point is thoroughly tested.

| Test | Result |
|---|---|
| `-- (SM) DROP sampletable;--` | ✓ |
| `ID: 10;DROP members --` | ✓ |
| `' or 1=1--` | ✓ |
| `') or ('1'='1--` | ✓ |
| `' UNION SELECT 1, 'user', 'something', 1--` | ✓ |
| sqlmap (minimal run) | ✓ 12 alerts out of 58 HTTP requests * |
| sqlmap (full run) | ✓ 3470 alerts out of 8023 HTTP requests * |

* *Not every HTTP request is necessarily intrusive*

Table 27: Detailed SQL injection test results.

#### A.13.1.2 Cross-site Scripting: Detailed results

For this test, we chose the most in-depth XSS strings that can be found in [98] and skipped the use of an automated XSS tool, since we believe OWASP's cheat sheet is a near-perfect reference in the XSS field.

| Test | Result |
|---|---|
| `<SCRIPT SRC=http://xss.rocks/xss.js></SCRIPT>` | ✓ |
| `<IMG SRC="javascript:alert('XSS');">` | ✓ |
| `<IMG SRC=# onmouseover="alert('xxs')">` | ✓ |
| `<SCRIPT/XSS SRC="http://xss.rocks/xss.js"></SCRIPT>` | ✓ |
| `<<SCRIPT>alert("XSS");//<</SCRIPT>` | ✓ |
| `<SCRIPT =">" SRC="httx://xss.rocks/xss.js"></SCRIPT>` | ✓ |

Table 28: Detailed cross-site scripting test results.

#### A.13.1.3 NTP Amplification Attack: Detailed results

As for the NTP amplification attack, we used two of the available tools that perform variations of this attack: **nmap** and the **Metasploit Framewor**k.

| Test | Result |
|---|---|
| Nmap (ntp-monlist.nse) [110] | ✓ |
| Metasploit (auxiliary/scanner/ntp/ntp_monlist) | ✓ |

Table 29: Detailed NTP amplification attack test results.

### A.13.2 Anomaly-based (Bro)

Validating the efficiency of anomaly-based detection is not as linear as validating misuse-based detection. Bro was our choice for the anomaly-based IDS, however, using it throughout these months, has proved that it is **much more than a simple IDS**. It is a complete **network analyzer and traffic classifier** that besides supporting the traditional "IDS-style pattern matching" [49], it can also detect time-based attacks (e.g. bruteforces, floods) and unusual traffic (e.g. DNS packets on port 80). In summary, with Bro, we can essentially treat the traffic as input variables and **manipulate it programmatically**, through scripts written in the Bro language.

The following table describes vulnerabilities and network policy breaches that were tested using Bro:

| Vulnerability | Available Tools | Description | Result |
|---|---|---|---|
| Bruteforce Attack (FTP) | Nmap | Consists in launching an FTP bruteforce attack using one of the available tools. | ✓ (See A.13.2.1) |
| Unusual Traffic | A browser, apache, nc, dig | This test involves two phases: one of them consists in forcing the HTTP traffic through unconventional ports and verify that Bro is able to correctly identify it as unusual traffic. | ✓ (See A.13.2.2) |
| HTTP directory scan | Nmap | Consists in scanning an application server for interesting files and directories | ✓ (See A.13.2.3) |
| Port scan | Nmap, Metasploit Framework | Consists in scanning a host, looking for oppened ports and services | ✓ (See A.13.2.4) |

Table 30: Anomaly-based detection validation.

#### A.13.2.1 Bruteforce Attack (FTP): Detailed results

| Test | Detection Script | Result |
|---|---|---|
| Nmap (ftp-brute.nse) [111] | /policy/protocols/ftp/detect-bruteforce.bro | ✓ |

Table 31: Detailed FTP bruteforcing test results.

#### A.13.2.2 Unusual Traffic: Detailed results

For this test, we started by using the well-known dig tool on the client-side to perform DNS requests on unusual ports. On the server-side, we used nc to start listening for TCP connections on port 80. This triggered many "weird" (weird.log) alerts on Bro, some with the message "bad_HTTP_request" and others with "unknown_protocol_2".

| Test | Detection Scripts | Result |
|---|---|---|
| dig & nc | Many | ✓ |

Table 32: Detailed TCP Syn Flood test results - Phase 1.

The second phase makes for the **main goal of this test**. It consisted in starting an HTTP server on our test server, listening on port 145 and performing normal HTTP requests on the client. By using the DPD framework [99], Bro managed to detect not only the **protocol being used** on the connection ( *"ProtocolDetector::Protocol_Found HTTP on port 145/tcp"*), but also correctly **identified the HTTP server** (*"ProtocolDetector::Server_Found HTTP server on port 145/tcp"*).

| Test | Detection Script | Result |
|---|---|---|
| Browser & apache | /policy/frameworks/dpd/detect-protocols.bro | ✓ |

Table 33: Detailed TCP Syn Flood test results - Phase 2.

#### A.13.2.3 HTTP directory scan: Detailed results

| Test | Detection Script | Result |
|---|---|---|
| Nmap (http-enum.nse) [112] | /policy/protocols/http/detect-bruteforce.bro (See Appendix B) | ✓ |

Table 34: Detailed HTTP directory scan test results.

#### A.13.2.4 Port scan: Detailed results

| Test | Detection Script | Result |
|---|---|---|
| Nmap | /policy/misc/scan.bro | ✓ |

Table 35: Detailed Port scanning test results.

### A.13.3 Host-based (OSSEC)

Host-based detection is our last chance at detecting a vulnerability, considering the moment it is detected by OSSEC, it will already have infected its target. For the sake of avoiding duplicated information, these tests will not cover network-based vulnerabilities (as they were *hopefully* detected by our two NIDS3), but solely rootkits/worms and issues that may threaten the server's availability.

#### A.13.3.1 RAM Stress Test

For this test, we used a simple script [101] written in the C language, that continuously performs numerous memory allocations, until the kernel abruptly kills the process. The results are as follows:

| Test | OSSEC Response * | Result |
|---|---|---|
| RAM Stress | Rule: 5108 (level 12) ->"System running out of memory. Availability of the system is in risk." Out of memory: Kill process 18045 (a.out) score 852 or sacrifice child | ✓ |

Table 36: Detailed system availability threatening test.

#### A.13.3.2 Malicious Binary Test (Integrity Check)

For this test, we decide to **replace a well known system binary - "cd" - with a malicious one**. Once again, we targeted our test server and the results are in the following table.

| Binary | OSSEC Response * | Result |
|---|---|---|
| cd | Rule: 550 (level 7) ->"Integrity checksum changed." Integrity checksum changed for: "/bin/cd" Size changed from "26" to "130288" | ✓ |

Table 37: Detailed OSSEC integrity checking testing.

#### A.13.3.3 Rootkit Test

In favor of testing the rootkit detection capabilities of OSSEC, we simulated the existence of **four different rootkits** in one of our test servers, the results are as follows:

| Rootkit | OSSEC Response * | Result |
|---|---|---|
| TRK | Rule: 510 (level 7) ->"Host-based anomaly detection event (rootcheck)." **Rootkit "TRK" detected** by the presence of file "/usr/bin/soucemask". | ✓ |
| Optickit | Rule: 510 (level 7) ->"Host-based anomaly detection event (rootcheck)." **Rootkit "Optickit" detected** by the presence of file "/usr/bin/xchk". | ✓ |
| Tuxkit [104] | Rule: 510 (level 7) ->"Host-based anomaly detection event (rootcheck)." **Rootkit "Tuxkit" detected** by the presence of file "/dev/tux". File "/dev/tux/.proc" present on /dev. Possible hidden file. | ✓ |
| AjaKit | Rule: 510 (level 7) ->"Host-based anomaly detection event (rootcheck)." **Rootkit "AjaKit" detected** by the presence of file "/lib/.ligh.gh". | ✓ |

* *OSSEC generated multiple alerts, this is only a sample.*

Table 38: Detailed OSSEC rootkit detection testing.

# B Developed scripts description

## snort-install.sh

This bash script is deployed by Puppet on **every NIDS (Snort) node** and is responsible for downloading, compiling and installing the latest version of Snort 3 (using Git).

## torrent-pcap-parser.sh (Reader)

This bash script is deployed by Puppet on **every NIDS node** and is added as a cron job to execute every 2 minutes. When executed, the reader will analyze the latest intrusion PCAP produced by the NIDS (starting on the packet where it left in the previous execution) and use tshark to extract all the core information from BitTorrent packets - frame number, frame timestamp, source IP address and torrent info hash.

Lastly, this information is appended to a file that is sent, via rsyslog, to the log manager.

## bro-install.sh

This bash script is deployed by Puppet on **every NIDS (Bro) node** and is responsible for downloading, compiling and installing the latest version of Bro (using Git).

## ossec-agent-install.sh

This bash script is deployed by Puppet on **every HIDS (OSSEC) node** and is responsible for downloading, compiling and installing the latest version of OSSEC (using Git). Since OSSEC's installation requires user interaction (for answering simple yes and no questions), this script is fed with a text file containing the predefined answers. These answers will define the type of installation (agent or manager), and the features that we want to include and enable in the application.

## ossec-agent-update.sh

This bash script is deployed by Puppet on **every HIDS (OSSEC) node** and is very similar to the script described in B, the difference being the answers' text file that is fed to the script.

## ossec-manager-install.sh

This bash script is deployed by Puppet on **every HIDS (OSSEC) node** and is very similar to the script described in B, the difference being the answers' text file that is fed to the script.

## ossec-manager-update.sh

This bash script is deployed by Puppet on **every HIDS (OSSEC) node** and is very similar to the script described in B, the difference being the answers' text file that is fed to the script.

## create-cluster.sh

This bash script is deployed by Puppet on **only one HIDS (OSSEC) node** and consists in the first time setup of the high availability cluster. The script uses pacemaker's command-line shell (pcs) to create the cluster, define a cluster password, enable and/or disable resources/features and define the load balancing method.

## create-bridge.sh

This bash script is deployed by Puppet on **the NIDS (Snort) load balancer** and supports the creation of the bridge and bonding interfaces, responsible for connecting the network TAP to every NIDS node, as well as the round-robin load balancing mechanism.

## bridge-ha.sh

This bash script is deployed by Puppet on **the NIDS (Snort) load balancer** and is responsible for rebalancing the bridge and bonding **only when a kernel-related issue occurs** in one of the NIDS nodes. This script uses layer 3 pings (using the "fping" tool [105] to realize if every node is reachable, and in case a node is unreachable, it is removed from the bonding interface, therefore rebalancing the load for the remaining nodes.

## install-splunk.sh

This bash script is deployed by Puppet on **the statistics/data analyzer server** and is responsible for downloading and installing the latest version of Splunk, from its official website.

## ossec-parser.py (Parser)

This python script is deployed by Puppet on **the log manager** and is responsible for converting the OSSEC logs, from their original format, into our standard logging format. The script starts with various regular expressions definitions (this way they can be altered effortlessly), that essentially define how the parsing is done, then a class is loaded, containing all the parameters of the standard logging format. Finally, the script starts continuously reading the logs from the original log file, and by matching original log lines with the defined regular expressions, it can successfully generate a new log file, using our logging format.

Every python script developed also includes a centralized exception logging mechanism, that allows us to understand when and why an exception occurs (since we most likely wouldn't notice it otherwise).

### snort-parser.py (Parser)

This python script is deployed by Puppet on **the log manager** and is very similar to the script described in B, the difference being the regular expressions defined at the start, in addition to a few simple conditions that allow us to extract more information from the original logs.

### bro-parser.py (Parser)

This python script is deployed by Puppet on **the log manager** and is very similar to the script described in B, the difference being the regular expressions defined at the start, in addition to a few simple conditions that allow us to extract more information from the original logs.

### resolver.py

This python script is deployed by Puppet on **the log manager** and is, in short, a DNS resolver, developed from the scratch to attend all our needs. The resolver starts by continuously reading logs from the output file of the parsers, and extracts every unresolved hostname or IP address. A new thread is then created for each unresolved address/hostname, therefore providing asynchronism and preventing undesired delays. Each request must be resolved in **under 1 second**, or it will be skipped. After a request has been completed (even if unsuccessful/skipped), it is saved into memory (cached), so it is resolved instantly the next time it appears. The resolved values are then placed in the respective log line, which is outputed to a new file, that is fed to our statistics/data analyzer. Everyday at 5 a.m (when the network usage is usually at its lowest) the cache is cleaned, hence avoiding memory leaks or even swapping.

### reactor.py (Reactor)

This python script is deployed by Puppet on **the log manager** and is the core of our reaction and alerting mechanism. Firstly, we define global variables containing the positions of our log parameters in a log line (e.g. the parameter "application" is at the position 0 in our format). We also define SMTP servers for e-mail alerting and the IP blocking command (in this case, iptables). Secondly we load and validate the rules, the contacts and the routers from their respective configuration files. Lastly, the script starts continuously reading logs from the output of our parsers, and iterates every rule for each log line received. The rules are divided by a pipe, as explained in 5.6, and each side of the pipe is interpreted as Python code (using the built-in "eval" function).

Note: After applying an IP address block in any device, the reactor will persist the action (in the case of iptables, by using the "iptables-save" command) and perform a cleanup once the action period expires.

## torrent-hash-parser.py (Crawler)

This python script is deployed by Puppet on **the log manager** and is responsible for converting torrent info hashes into the respective torrent filename. We start by defining the search engine URL (in this case Google's), the user agent and the regular expression to match when the results are retrieved (in this case, it will match only the titles of the web pages returned by Google's search results). It then starts continuously reading the output of B (torrent-pcap-parser) and extracts the info hash from each line. The info hash is then used as the search input in our defined search engine. Eventually, multiple results (web page titles, usually containing the torrent filename, when searching for hashes) will be retrieved, the choice of the right title is done based on string similarity (strings will be given a score, based on their similarity), i.e. if 20 results are retrieved and 5 of them are very similar, then the probability of one of them being the filename we want is much higher than one of the remaining.

Every resolution (even if unsuccessful) is then cached to prevent further searches for the same hash. Similar lines received sequentially (same source IP, frame time and torrent hash) are also skipped in order to minimize delays. Every search is succeeded by a 1 second sleep, since most search engines include an IP blocking mechanism when the search throughput per IP address surpasses a certain threshold.

Besides exception logging, the Crawler also includes a verbose logging mechanism, so as to understand when a torrent hash is incorrectly resolved, this log includes cache hits and misses and all the string similarity scores.

## uncomment.py

This python script is **not deployed by Puppet** and it is used to uncomment Snort's rules after downloading their latest version (since most of them are commented by default). It includes a list of blacklisted rules that should stay commented that often generate numerous false positives and/or overwhelming amounts of intrusion logs.

## detect-bruteforce.bro (HTTP)

This bro script is deployed by Puppet on **every NIDS (Bro) node** and is the script responsible for identifying HTTP bruteforce attacks, by triggering an alert when excessive HTTP errors occur. The original script is distributed on GitHub by Michal Purzynski [100] and this is a mere adaptation of said script for our environment.