Study, implementation and configuration of an open source IDS solution in INESC TEC

a project authored by

Nuno Gabriel Moreira dos Santos

Master´s Degree in Information, Communication and Multimedia Technologies

Telecommunications branch

oriented by

Prof. Cláudia Freitas and Prof. Gil Coutinho

September 18 of 2021

Abstract

The increasing diversity of attacks and threats circulating on the global network has been making security issues increasingly crucial. Organizations are spending more and more resources on developing and implementing cybersecurity techniques and methods to be able to keep up with the growing complexity of attacks and, above all, mitigate the negative impacts on infrastructure. It is reasonable to assume that if certain anomalies can be detected on time, the risk of being affected is potentially lower. Considering the context of Institute for Systems and Computer Engineering, Technology and Science (INESC TEC) [1] an institution of high technological value that is largely dependent on the digital environment, it is a considerable challenge to ensure that all information and all communications are properly protected. The main objective is to add another layer of security, such as enabling the centralized availability of information regarding the detection of malicious activities that may harm the institution's network. This document proposes, as a first phase, a brief study on some security subjects like cybersecurity, information security and network security. Not less important, an analysis of several existing open source IDS solutions is carried out based on the current market. Among the existing solutions, the focus of the work is directed towards the implementation and configuration of an open source solution.

Keywords: IDS. Cybersecurity. Open source. INESC TEC.

[1] Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESC TEC)

Resumo

A crescente diversidade dos ataques e das ameaças que circulam na rede global, contribuiu para que as questões de segurança sejam cada vez mais cruciais. As organizações despendem cada vez mais recursos no desenvolvimento e implementação de técnicas e métodos de segurança cibernética que sejam capazes de acompanhar a crescente complexidade dos ataques e, que principalmente, sejam capazes de mitigar os impactos negativos na infraestrutura. Podemos afirmar que se for possível detetar certas anomalias atempadamente, o risco de ser afetado é potencialmente menor. Considerando o INESC TEC uma instituição de alto valor tecnológico dependente, em grande parte, do ambiente digital, é inerente o desafio em assegurar que toda a informação e toda a comunicação estejam devidamente protegidas. O objetivo passa por adicionar mais uma camada segurança, isto é, possibilitar a disponibilização centralizada de informação relativa à deteção de atividades maliciosas que possam ter um impacto negativo na rede da instituição. Este documento propõe na primeira fase um breve estudo sobre alguns assuntos relacionados com a segurança cibernética, segurança da informação e segurança de rede. Não menos importante, é realizada uma análise de diversas soluções Intrusion Detection System (IDS) open source existentes o mercado atual. De entre as soluções existentes, o foco do trabalho é direcionado à implementação e configuração de uma solução Network Intrusion Detection System (NIDS) assente em código aberto.

Palavras-chave: IDS. Cibersegurança. Open source. INESC TEC.

Acknowledgements

First of all, I want to thank my family for giving me every opportunity to get me here and for always accompanying me with all their love, understanding and affection.

I want to give a special thanks to my girlfriend for giving me not only all the love in the world but also the strength and motivation to complete this journey.

My appreciation also extends to my colleagues Fernando Sousa, António Carlos Sá and André Freitas from INESC TEC for sharing with me the wisdom of years of experience working in this institution. Thank you for welcoming me so well and for teaching me so much during my day-to-day work.

To all my friends for all the good moments spent during this two years. They were a great help in the most stressful times.

To my advisors Cláudia Freitas and Gil Coutinho for their guidance and for sharing all their wisdom and advices.

Contents

Acronyms **9**

1 Introduction **12**

1.1 Context . . . 13

1.2 Objectives . . . 14

1.3 Host institution . . . 16

2 State of art **18**

2.1 Cybersecurity . . . 18

2.2 Information Security . . . 19

2.3 Network Security . . . 20

2.4 Open Source Security . . . 21

2.5 Attacks . . . 24

2.5.1 Malware . . . 25

2.5.2 Statistics . . . 26

2.5.3 Passwords Guessing . . . 27

2.5.4 Ransomware . . . 27

2.5.5 DoS . . . 28

2.6 IDS Systems . . . 29

2.6.1 IDS Architecture . . . 31

2.6.2 Alerts . . . 32

2.6.3 Signatures . . . 33

2.6.4 IDS Types . . . 34

2.7 Open Source IDS tools . . . 40

2.7.1 OSSEC . . . 40

2.7.2 Hogzilla IDS . . . 41

2.7.3 Tripwire . . . 42

2.7.4 Wazuh . . . 42

2.7.5 Snort . . . 43
2.7.6 Zeek . . . 46
2.7.7 Suricata . . . 47
2.7.8 IDS comparative table . . . 50
2.8 SIEM - Security Information and Event Management . . . 50
2.8.1 OSSIM . . . 51
2.8.2 ELK . . . 52
2.8.3 SIEMonster Community Edition . . . 54
2.8.4 Wazuh and OSSEC . . . 55
2.8.5 Splunk Free . . . 55
2.8.6 Graylog . . . 56
2.8.7 SIEM comparative table . . . 57
2.9 NSM - Network Security Monitor . . . 57
2.9.1 RockNSM . . . 57
2.9.2 Security Onion . . . 58
2.9.3 SELKS . . . 60
2.9.4 NSM comparative table . . . 61
2.10 Related work . . . 61

3 Scenario Overview **68**
3.1 Implementation Design . . . 68
3.2 Available resources . . . 69

4 Implementation **70**
4.1 IDS implementation . . . 70
4.1.1 First Phase - SELKS implementation . . . 70
4.1.2 Second Phase - Suricata implementation . . . 71
4.1.3 Third phase - Elastic Stack implementation . . . 74

5 Tests and Results **79**
5.1 Tuning Suricata . . . 79
5.1.1 Running mode . . . 79

5.1.2 AF_PACKET . . . 80
5.1.3 Rules . . . 81
5.1.4 Max-pending-packets . . . 82
5.1.5 Network interfaces . . . 82
5.1.6 Detection engine . . . 83
5.1.7 Host OS . . . 85
5.1.8 Memcap values . . . 86
5.2 Tuning Elastic Stack . . . 93
5.3 Impact Tests . . . 93
5.3.1 Suricata resource usage . . . 93
5.3.2 Elastic Stack . . . 94
5.3.3 Traffic Rates . . . 94
5.4 Optimization results . . . 95
5.4.1 Workers vs Autofp . . . 95
5.4.2 Alert otimization . . . 96

6 Conclusions **100**
6.1 Difficulties Faced . . . 101
6.2 Future work . . . 101

References **102**

A Suricata default rule file configuration **113**

B Suricata rule configuration **114**

C Suricata fast.log output **116**

D Suricata eve-alerts.json output **117**

E Elasticsearch curl output **118**

F Filebeat Suricata module configuration **119**

G Filebeat default index name change **120**

H Suricata `suricata.log` file output **121**

I Suricata `enable.conf` configuration file **123**

J `ethtool` configuration **125**

K Kibana TLS configuration **127**

L Filebeat TLS configuration **128**

M Suricata `threshold.conf` configuration file **129**

N `suricata.log` - Processed rules **131**

O ElastAlert2 critical activity rule **133**

List of Figures

1.1 Organizational structure of INESC TEC. Taken from: [3] 17

2.1 IDS architecture schema. Taken from: [25] 31
2.2 Active IDS 34
2.3 Passive IDS 35
2.4 Network-based IDS 35
2.5 Host-based IDS 36
2.6 Signature-based architecture 37
2.7 Anomaly-based architecture. Taken from: [34] 37
2.8 Distinction between signature-based and anomaly-based IDS. Taken from: [37] 39
2.9 Schematic architecture of OSSEC. Taken from: [39] 40
2.10 Schematic architecture of Hogzilla: Taken from: [41] 41
2.11 Schematic architecture of Wazuh. Taken from: [45] 42
2.12 Schematic architecture of Snort. Taken from: [37] 44
2.13 Snort signature example. Taken from: [54] 46
2.14 Snort HTTP signature. Taken from: [57] 48
2.15 Suricata HTTP signature Nr.1. Taken from: [57] 49
2.16 Suricata HTTP signature Nr.2. Taken from: [57] 49
2.17 Suricata schematic architecture: Taken from: [37] 49
2.18 Schematic architecture of OSSIM. Taken from: [60] 51
2.19 ELK stack functionalities. Taken from: [61] 53
2.20 Beats schematic architecture. Taken from: [62] 53
2.21 RockNSM schematic architecture. Taken from: [71] 58
2.22 Security Onion Traffic flow. Taken from: [72] 59
2.23 Security Onion architecture. Taken from: [74] 60

3.1 IDS implementation design 68

4.1 Elastic Stack implementation 74
4.2 Kibana dashboards structure 78

5.1 Traffic spike interface `enp2s0f1` 87
5.2 Joomla exploit 97
5.3 MySQL scan 97
5.4 MySQL brute force login attempt 97
5.5 MySQL brute force login failure 98
5.6 Crypto mining activity 98

List of Tables

2.1 Advantages of open source software 23
2.2 Disadvantages of open source software 23
2.3 Total network/malware attacks in the 2019 and 2020 27
2.4 Types of IDS alerts 32
2.5 IDS Signatures 33
2.6 Anomaly-based detection techniques 38
2.7 Snort achitecture modules 45
2.8 Preprocessors included in Snort 45
2.9 Suricata signature actions 48
2.10 IDS comparison 50
2.11 OSSIM solution tools 52
2.12 SIEMonster functionalities 55
2.13 SIEM comparison 57
2.14 NSM comparison 61

4.1 Suricata configured log outputs 73

5.1 `AF_PACKET` configuration 79
5.2 `AF_PACKET` configuration 80
5.3 Rule management configuration files 81
5.4 Running modes resource usage 94
5.5 Traffic rates 94
5.6 `Autofp` vs `Workers` stats 95
5.7 `Workers` mode stats 96

Acronyms

ARP Address Resolution Protocol

DDoS Distributed Denial of Service

DNS Domain Name System

DoS Denial of Service

ET Emerging Threats

FCT Fundação para a Ciência e a Tecnologia

FCUP Faculdade de Ciências da Universidade do Porto

FEUP Faculdade de Engenharia da Universidade do Porto

FTP File Transfer Protocol

GDPR General Data Protection Regulation

HIDS Host Intrusion Detection System

HTTP Hypertext Transfer Protocol

IANA Internet Assigned Numbers Authority

ICMP Internet Control Message Protocol

IDS Intrusion Detection System

IDWG Intrusion Detection Working Group

IETF Internet Engineering Task Force

IGMP Internet Group Message Protocol

ILM Index Lifecycle Management

iLO HP Integrated Lights-Out

INEGI Instituto de Ciência e Inovação em Engenharia Mecânica e Engenharia Industrial

INESC TEC Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência

IP Internet Protocol

IPS Intrusion Prevention System

ISEP Instituto Superior de Engenharia do Porto

ITL Information Technology Laboratory

ITU International Telecommunication Union

JVM Java Virtual Machine

MPM Multi-Pattern-Matcher

NIC Network Interface Controller

NIDS Network Intrusion Detection System

NIST National Institute of Standards and Technology

NSM Network Security Monitoring

OISF Open Information Security Foundation

OSSEC Open Source Security Event Correlator

OSSIM Open Source Security Information Management

PCAP Packet capture

RCE Remote Code Execution

RSS Receive Side Scaling

SAS Serviço de Administração e Sistemas

SEM Security Event Management

SIEM Security Information and Event Management

SIG Serviço de Informática de Gestão

SIM Security Information Management

SMB Server Message Block

SMTP Simple Mail Transfer Protocol

SOC Security Onion Console

SRC Serviço de Redes e Comunicações

SSH Secure Shell

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

UMinho Universidade do Minho

UP Universidade do Porto

UTAD Universidade de Trás-os-Montes e Alto Douro

VPN Virtual Private Network

Chapter 1

Introduction

Currently, we are increasingly dependent on information technologies and systems for economic and social development. But, despite all the advantages of a more technologically evolved world, one must be aware that technology has contributed to the exponential growth of malicious activities, exploitation of vulnerabilities and threats to cybersecurity.

As a curiosity, it is estimated that close to two thirds of the world population will have access to the internet by 2023. Also by 2023, the number of devices connected to the IP network will be three times greater than the world population. This means that it is estimated that there will be about 29.3 billion devices connected to the network [1].

This project proposes the study and implementation of a security solution in INESC TEC, with the main objective of adding another level of protection to the already existing security methods.

1.1 Context

As an institution centered on technological development and innovation that processes and stores large amounts of information, cybersecurity must be considered a priority. With such a high volume of traffic, it is difficult to understand what is circulating through the network. For that reason, it would be necessary to centralize and organize the availability of the most relevant information about the most critical security alerts with the implementation of an IDS.

There are already security mechanisms in place, either network based (blade IPS on the firewall) or host based (Nessus to detect and report specific machine vulnerabilities). With this tool we intend to ascertain the order of magnitude and danger of incoming attacks, but more importantly, to know which attacks pass to the internal network.

In July 2020, there was an attack on an important research center which, like INESC TEC, is an associated laboratory with the Universidade do Porto (UP). In a report published in the newspaper Expresso [2], the Institute of Science and Innovation in Mechanical Engineering and Industrial Engineering (INEGI) [1] was allegedly the target of a malicious ransomware [2]. The attackers took advantage of a vulnerability in the network to launch an information encryption attack, then asking for a financial rescue so that the information could be recovered. This attack has raised a great deal of concern amongst those in charge of the associated institutions. Since that moment, the need has arisen to rethink certain issues related to security.

This project proposes the implementation of an open source solution that can contribute to improving the security of INESC TEC's technological assets and services, without the high costs of a commercial solution and with the flexibility and optimization that an open source solution can offer. We chose network based, the scope of application being all of servers, as well as all workstations that use its infrastructure, either locally or remotely. This tool is expected to give network administrators more power to analyze and detect intrusions that could have a negative impact on the institution.

[1] Instituto de Ciência e Inovação em Engenharia Mecânica e Engenharia Industrial (INEGI)

[2] Software attack aimed at extort the attacked person or organization. The attack blocks access to the system or encrypts important files so that a financial ransom is required in order to recover all access

1.2 Objectives

The main objective of this project is the study and implementation of the IDS solution that best suits the needs of the organization, so that those responsible for security have an auxiliary tool capable of observing and detecting intrusions in the computer network. In order to facilitate the development of the project, a set of specific objectives was defined.

In an initial phase, the study of the state of the art in IDS solutions is carried out, particularly those based on open source platforms. Then, it is necessary to map the institution's network infrastructure and analyze the requirements needed to implement the solution. Finally, the installation and configuration of the IDS tool and the data visualization tool that make up the idealized solution are carried out. Since it will be implemented in a production environment and in a network with a large data flow, the benefits and disadvantages of being a open source solution should be studied.

After implementation, the focus is on analyzing the results in order to optimize the platform as much as possible. It is intended to provide administrators with a soution for data analysis. Security must be everyone's responsibility and, therefore, it is also critical to develop proposals for methodologies and good practices in order to ensure a correct approach to cybersecurity. The main activities that define the realization of this project will be presented below:

- Scenario Overview
  - Network architecture
  - Necessary resources
  - Implementation design

- Implementation
  - Installation and configuration of the IDS probes
  - Installation and configuration of the data visualization platform
- Preliminary tests
  - Impact tests
  - Functional testing
  - Results analysis
  - Validation of the chosen solution
  - Analysis of the benefits of implementing the solution

In the final phase, the necessary optimizations and final tests are carried out to ensure the best possible level of productivity.

- Optimization
  - Collection of the most relevant data on the preliminary tests
  - Optimization of configuration as needed
- Finals Test
  - Impact tests
  - Functional testing
  - Results analysis

1.3 Host institution

INESC TEC is a non-profit institution, associated with the Foundation for Science and Technology [3] and whose majority member is UP. It focuses its activity on scientific research and technological development. Through technology transfer, knowledge transfer and advanced consultancy, the institution's mission is to promote the technological development of society. The institution's main values are, according to the organization's website [3], "Excellence in research, people-centered, freedom of investigation, cooperation and social responsibility".

Altogether, the institution is divided into 13 research centers that are grouped into four clusters (Informatics, Industrial and Systems Engineering, Intelligent Systems Networks and Energy) in five different locations, being the Instituto Superior de Engenharia do Porto (ISEP), the Faculdade de Engenharia da Universidade do Porto (FEUP), the Faculdade de Ciências da Universidade do Porto (FCUP), the Universidade do Minho (UMinho) and the Universidade de Trás-os-Montes e Alto Douro (UTAD). In 2012, the organizational and scientific management model was exported internationally, with the creation of INESC P&D Brasil in order to promote science and knowledge in the country. Through the transmission of scientific knowledge and the development of new technologies, the institution seeks to promote innovation and technological growth in society [3].

INESC TEC currently has more than 700 researchers, in addition to the technical support teams. IT support services are responsible for all the management and maintenance of the IT infrastructure and provide necessary technical support to all employees, and are divided into three teams:

- Systems Administration Service (SAS) [4] - responsible for the administration of servers and computer systems.
- Network and Communications Service (SRC) [5] - responsible for the administration and maintenance of voice communications resources and network services.

[3] Fundação para a Ciência e a Tecnologia (FCT)
[4] Serviço de Administração e Sistemas (SAS)
[5] Serviço de Redes e Comunicações (SRC)

- IT management service (SIG) [6] - responsible for the administration and maintenance of the institution's management systems.

The figure 1.1 demonstrates the schematic of the organizational structure of INESC TEC.

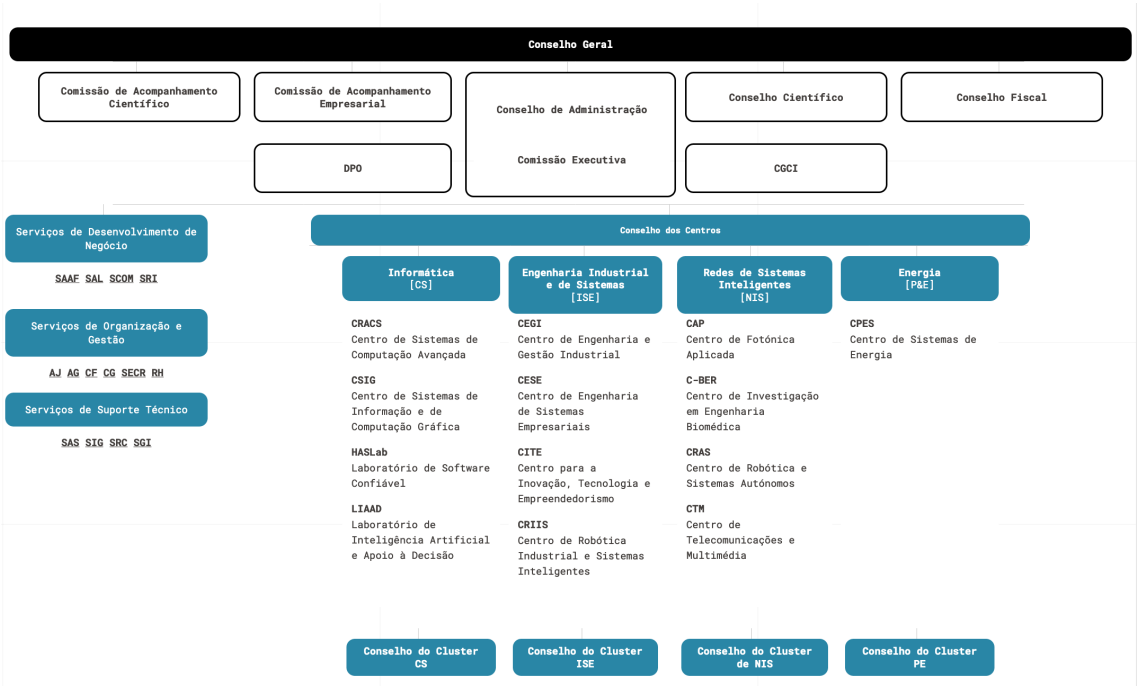

Figure 1.1: Organizational structure of INESC TEC. Taken from: [3]

[6] Serviço de Informática de Gestão (SIG)

Chapter 2

State of art

This chapter aims to briefly expand on the subject of cybersecurity and to address the main characteristics and components of the type of system to be implemented. In addition, reseach done by various authors is analysed, in order to understand the state of art in relation to developments in intrusion detection systems.

2.1 Cybersecurity

Cyberspace is a complex environment. It is a space of collective responsibility, personal and digital interaction, capable of offering a world of opportunities. However, in the same way that it offers such positive aspects, it can also enable the creation of risk and threat environments capable of significant impacts on society.

While information security focuses only on information regardless of its format, the International Telecommunication Union (ITU) website [4] defines cybersecurity as the "*Collection of tools, policies, security concepts security, security protections, guidelines, risk management approaches, actions, training, best practices, assurance and technologies that can be used to protect the cyber environment and the user's organization and assets. connected computing devices, personnel, infrastructure, applications, services, telecommunications systems and all information transmitted and/or stored in the cyber environment.*"

The concept of cybersecurity adds a new dimension to the resources and assets it seeks to protect. It is plausible to claim that it is intrinsically linked to the fundamental principles of the concept of information security but, despite the apparent ambiguity between the two, there are certain types of aspects that differ [5].

Thus, cyberspace security goes beyond the concept of information security insofar as "*[...] adds an important ethical dimension, because problems like cyber bullying extend to beyond the law and present an ethical problem that must be dealt with by society.*" [5]

We can conclude that *"Cyberspace is more than the internet, including not only hardware, software and information systems but also people and social interaction within networks."* [6]

2.2 Information Security

The authors of the article [5] state that "*All security concerns the protection of assets against the various threats posed by certain inherent vulnerabilities*". Information security should be limited to a set of fundamentals that form the basis for understanding how an organization's security control and management should be developed. These are the fundamental questions that must be studied in order to develop a productive and efficient security methodology [7]:

- Which assets should be protected?
- How are these assets threatened?
- Which solutions should be taken, taking into account existing threats?

These assets consist of a set of physical, material or technological resources that have significant value for an organization's activity. In a first phase, a survey of the assets that are most critical for the organization should be carried out. Next, understand what are the organization vulnerabilities and how they can be compromised. Finally, after a careful risk assessment, it is important to pool the necessary technical and operational efforts, seeking to mitigate the risks associated with the performance of the organization's functions.

Information, in the broadest sense of the word, encompasses processes, people and technologies. In order to protect this information, it is essential to use methods that ensure compliance with the five fundamental pillars of information security [7]:

- Confidentiality - ensures that information should not be made available to unknown persons or entities.
- Integrity - ensures that any changes made to the information or system must be duly authorized in order to maintain its consistency.
- Availability - ensures that information and systems remain functional and accessible to all authorized persons.
- Authenticity - ensures that the user and the information are legitimate and reliable.
- Legitimacy - ensures that all actions originating in an organization are exclusively associated as having originated in that same organization.

2.3 Network Security

The increasing complexity of possible threats on the network means that the detection functions consume more and more time and resources. For an organization to be truly protected, the installation of firewalls should be one of the minimal requirements. However, prevention is necessary but might not be enough [8].

We should also consider the ability to detect potential threats in a timely manner. The combination of prevention and detection is the key to good security. For that, a set of principles that constitute one of the bases of development for a network security architecture must be followed, being: Principle of Integrity, Principle of Consistency, Principle of Periodicity, Principle of Practicality, Principle of Hierarchy and the Principle of Multiple Protection [9].

Initially, the Integrity Principle ensures that the information is accurate and reliable. On the other hand, the Principle of Consistency ensures that the security architecture remains consistent in relation to security requirements and that the security measures adopted follow certain previously established standards. The Principle of Periodicity defends that the security system should never remain stagnant. Due to the complexity of the attacks, the system must remain properly updated in order to keep up with the constant evolution in this field. The Practicality Principle, on the other hand, argues that users technical knowledge is never the same. The system must therefore be as

practical and intuitive as possible. The Hierarchy Principle ensures that network security must meet the needs of different levels of operation, that is, for example, user permissions or information confidentiality. Finally, the Multiple Protection Principle ensures multiple protection paths capable of providing redundancy in the event of failure.

A network security system must consist of a coordinated combination of various technologies, such as firewalls, intrusion detection systems, data encryption, Virtual Private Network (VPN) and authentication controls. However, security is not just about technology. The human factor is also essential in the development and management of this type of systems [9].

2.4 Open Source Security

Security is an increasingly sensitive issue. With an increasingly digital world, users know that implementing systems that guarantee maximum security for their systems and their information is almost imperative. However, implementing and maintaining security solutions can be too expensive.

The commercial software is not typically accompanied by the source code and only those who are involved in its development have the possibility to know this code. This implies that users are unable to manipulate the software structure according to their needs [10].

The open source software widens the range of possibilities, as it allows the development and manipulation of the most diverse variants of software for the specific needs of each user. In addition, for some users, the open source concept means being able to contribute to code review and optimization processes, since most open source projects have a central repository that records the entire history of modifications and new versions.

Open source is understood to mean all software in which the source code is distributed, tipically free of charge and which includes licenses that allow users to modify and redistribute the software. The software models can be compared using their degree of openness and licenses. Regarding openness, software can be a Black Box (no code is distributed), a Closed Box (only the object code is distributed) or an Open Box (when source and object codes are distributed). As for licenses, software can be [11]:

- Comercial - software completely restricted and paid for by the user.
- Shareware - software with features available for a limited time. The source code and object code are neither distributed nor changed.
- Freeware - software free of cost and without restrictions on distribution and use. However, the source code and object code are not distributed.
- Shared source - paid software in which only the source code is distributed, but only for viewing or changing for non-commercial purposes.
- Open source - normally free of cost and without any restrictions regarding distribution, use, source codes and object codes.

Open source software allows virtually any user to analyze their code for improvements or vulnerabilities. This ensures greater transparency and confidence as the software is constantly reviewed and improved. However, the fact that it is open source does not guarantee that it is more secure. Even if there is a possibility that the code can be reviewed and analyzed by anyone, the software needs a continuous review process and the user is primarily responsible for the security of its implementation [11].

Security expert Marcus Ranum says that being open source does not imply that the software is better or more reliable. Still in the same article, the specialist Spafford says that what determines the trust of open source software is the quality and care in its development.

Proponents of open source software say the flaws will be discovered more quickly if there are a large number of users reviewing and analyzing the code. Marcus Ranum disagrees stating that *"In my experience, very few people actually read the code. They just look at the readme files. When I wrote the first public firewall toolkit, probably 2,000 sites used it, but only about 10 people gave me feedback or patches. I'm not impressed with the open source concept"*.

Proponents of commercial software argue that a programmer specializing in the software in question will do a much better job than a simple person who analyzes open source code. One of the main criticisms of proponents of proprietary software has to do with the widespread availability of application code, so attackers can much more easily look for exploitable vulnerabilities. Defenders of open source refute this argument, stating that closed software is also attacked and that full code availability allows vulnerabilities to be more easily discovered [10]. The table 2.1 demonstrates some advantages of this type of software.

Advantages	Why?
Cost	normally with no costs or low prices that attract users
Flexibility	users can customize according to their needs
Quality	software developed with a focus on what really matters to the user

Table 2.1: Advantages of open source software

In spite of everything, the implementation and development of software open source also imposes some disadvantages and concerns as shown in the table 2.2.

Disadvantages	Why?
Backdoors	distributed code and vulnerable to attacks
Support	do not always have viable technical support
Use and Maintenance	major development focus on functionality

Table 2.2: Disadvantages of open source software

2.5 Attacks

An attack can be defined as an action resulting from the realization of a threat. This deliberate action aims to exploit the existing vulnerabilities with the objective of compromising the security of the asset to which it is directed, which may cause irreversible consequences. Attacks can be classified as active or passive [7].

A passive attack occurs when there is an attempt to compromise the information of a given system, without affecting the normal functioning of the system. The information is only compromised but is not modified.

On the other hand, the main objective of an active attack is to impact the normal functioning of a given system in order to compromise and tamper with information. These attacks have the ability to modify the original flow of information or even create a false flow in order to persuade users that the data is reliable. Active attacks can be divided into four main categories: attacks by deception, retransmission, modification and Denial of Service (DoS). Deception attacks seek to assume false identities and ensure privileged access to information that would not normally be available to them. In retransmission attacks, information is passively captured so that it is maliciously repeated or delayed for the benefit of the attacker. Modifying attacks seek to modify the header in order to divert a certain message to another destination or even simply modify the content of the original message. The DoS usually depends on an overload methodology until blocking, in order to disable the normal functioning of a system or a network.

Typically, cyber attacks consist of four steps: targeting, checking for vulnerabilities, attack methods and analyzing results. The first step is to choose the target of the attack. For this, the attacker seeks to obtain as much information as possible about the target. Then, the vulnerability of the target system is checked for possible vulnerabilities. In the third stage, depending on the information obtained previously, the attacker chooses the tools and methods he will use to attack his target. Finally, the attacker does an analysis of the results. If the tools they chose allowed access to the information they wanted, the attacker ends the attack. Otherwise, the attacker must return to the second stage in order to search for new vulnerabilities [12].

In addition to considering the vulnerabilities that make attacks possible, it is also essential to understand the reasons that lead an attacker to execute his plan [13]. This methodology will be essential for any organization to understand the existing threats and develop an objective risk analysis. From a general perspective, attackers are concerned with stealing information, destroying information or simply incapacitating certain assets through DoS attacks. Pointing out some statistics for the year 2020, according to the article [14] of Visão magazine, national companies were attacked an average of 410 times a week, with 95% of these attacks arriving by email. Vulnerability exploitation attacks affect about 70% of organizations.

It is practically impossible to follow all types of attacks existing these days. Morais [15] states in his study that attacks are distinguished through the methods with which they are carried out and the impacts they intend to cause. In order for an agile and competent security policy to be implemented, knowledge of the different types of threats is effectively necessary.

2.5.1 Malware

Any malicious software that may affect a network or system is called malware. However, this is an especially broad category in view of the exponential growth in the quantity and quality of malicious types of software that circulate on networks. Some of the most common malware types are viruses, worms, trojan horse and exploits. This type of attack can also be defined as an automatic attack, due to the fact that they can be executed by software and without human intervention.

A virus consists of software or scripts that are housed in seemingly harmless programs in order to modify or destroy information and reproduce. In order to be able to infect the system, they need the contaminated program to be executed, that is, they depend on human intervention. After being activated, they have the ability to propagate among several system files always depending on the security of the attacked system. A worm is very similar to a virus, however, it doesn't need any program for it to be hosted on the system. It is a type of automated malware, that is, it can spread without the need to infect other files on the system and does not need any human intervention. It can be propagated via email or over network connections, allowing the attacker to infect other

systems completely autonomously. A Trojan horse is a malicious program inserted into an apparently legitimate and trustworthy program. After their insertion, they have the ability to obtain a specific level of control over the system. An exploit is a program that looks for vulnerabilities in order to obtain full control of the system [15]. Referred to in one of Avast blogs [16], an exploit is nothing more than a way to successfully promote or exploit a vulnerability in the system. However, an exploit alone is not necessarily malicious. Although it is typically used maliciously, an attacker can simply use the exploit tool in order to gain unauthorized access to the system and install malicious software.

2.5.2 Statistics

Some statistics show that the Windows operating system remains the most affected system with 84% of all reported cases. However, there was a big increase in the percentage of malware dedicated to macOS operating systems. Despite their reliable security reputation, the growing popularity of Apple devices and the ability of attackers to develop more complex attacks has made these devices increasingly more targeted [17].

In Portugal, throughout 2020, there were a total of 251,810 malware attacks. The top malware attack presented on the list was `Application.Hacktool.JQ` with a total of 51,796. Hacktools are used to patch software so they can run without a valid license. The second was `JS:Trojan.Cryxos.2735` with a total of 27,629 attacks. This attack presents a message that the computer has been blocked and data are being stolen. Then, user´s are redirected to call a phone number for assistance. `Win32/Heri`, a malware capable of replicating and infecting other programs comes in third with a total of 18,581 attacks [18].

In regard to network attacks, there were a total of 71,712 attacks. "Unknown" attacks were the most common with a total of 18,626. The second most common attack was `WEB-CLIENT Cisco WebEx Chrome Extension Remote Code Execution -1 (CVE-2017-3823)`[1] with a total of 13,768 attacks and finally the third being `WEB Cross-site Scripting-36`[2] with 12,635 attacks.

[1] Vulnerability that affects the browser extensions for Cisco WebEx Meetings Server and Cisco WebEx Centers when running in Windows.

[2] Malicious scripts are injected into otherwise benign and trusted websites.

Comparing with data from 2019, it is possible to observe that malware attacks have decreased substancially. However, exactly the opposite happened in relation to network attacks that had a significant increase as it is shown in table 2.3.

Year	Total malware attacks	Total network attacks
2019	882,952	17,980
2020	251,810	71,712

Table 2.3: Total network/malware attacks in the 2019 and 2020

2.5.3 Passwords Guessing

One of the most basic aspects of a security policy is based on educating users on the correct use of passwords. This is because it is one of the solutions to prevent password guessing attacks. In the guessing method, malicious programs try to guess the system password and are usually divided into three types: Brute force, Dictionary and Sniffing.

Brute Force attacks occur when the attacker tries to generate all possible combinations until he hits the correct one. Dictionary attacks occurs when the attacker uses the most familiar and common words to reach his goal. On the other hand, password sniffing attacks use programs that allow the monitoring of IP traffic. Typically, attackers gain unauthorized access to the system through the same local network and, with the sniffing program, they are able to process the flow of network packets, in order to be able to decipher some password. Although these attacks exist, they are already beginning to be much less effective in the face of developments in the security of network protocols such as, for example, the encryption of information carried in the packet.

2.5.4 Ransomware

Extortion remains one of the main motivations behind an attack. These attacks are known as ramsomware. These are attacks that consist of encrypting the data so that a monetary ransom is then requested so that the information is made available again.

Without this financial benefit, the organization may never recover the compromised information again. As mentioned in the blog of NordVPN [17], one of the largest providers of VPN services in the world, most of these attacks are associated with the download of files/emails containing malicious links. Also, the operating system most affected by ransomware is Windows, representing 87% of all ransomware attacks in 2020. Much is due to the fact that it is the most widely used operating system, having a 35% of the world market share [19]. That said, the awareness of users to this type of problems can be fundamental to reduce the success of the attacks but also organizations must have the ability to mitigate and control any attacks that may arise, thus being able to resume their normal activity in the shortest time possible.

2.5.5 DoS

Several reasons lead the attacker to carry out attacks of this type, from pure personal satisfaction to political or financial issues. According to [1], these attacks can grow from 7.9 million in 2018 to 15.4 million by 2023. In February of 2020, Imperva Research Labs [20] published a statistical report that analyzed 46000 DoS attacks from May to December 2019. The top three industries targeted by DoS are the gaming (36%), gambling (31,25%) and computers and internet (26,51%). As for the most affected countries, it is noteworthy the fact that the eastern region of Asia comprises about 77% of all the attacks.

Attacks referred to as Distributed Denial of Service (DDoS) differ from a common DoS attack in that multiple systems are used to attack a single target system. The web page of the security company Kaspersky [21], states that "*DDoS attacks are used to disable the target organization's online presence or major business processes - and this can have a lasting impact on the victim*". While DDoS attacks are carried out from a multitude of attacking hosts against one target host, a common DoS attack is carried out from one to one and can be divided into multiple types of attacks, which are presented below.

In a SYN flooding attack, large amounts of SYN packets are sent (packet that signal connection requests) until the system isn't able to respond properly. In a Transmission Control Protocol (TCP) session, the source computer sends a SYN packet to which the destination computer responds with a SYN-ACK, acknowledging that it has received the request and that it has been accepted. For each packet received by the destination

computer it sends a SYN-ACK but, however, it does not get a response from the source computer as that one is only trying to send as many SYN packets as possible. Since it does not get any kind of response and continues to send ACK requests, the destination computer is overloaded and leaves the TCP ports in a half-open state. The TCP protocol defines a limited number of half-open sessions and when that number is exceeded, the system proceeds to reject any connection requests until those sessions are resolved [13].

Flood Attacks are the most common DoS attacks and consists of sending more traffic than the target system can handle. In a Smurf Attack, the attacker sends a huge amount of Internet Control Message Protocol (ICMP) requests to the broadcast address. The source IP address of these requests is falsified, so that the computers assume that the request originated in the victim's system. Thus, all of these computers will respond to the ICMP request by flooding the victim's system with responses that will most likely prevent the machine from reacting.

A Teardrop Attack consists of sending fragmented and corrupt IP packets, which can cause the system to simply stop responding to the user inputs. Knowing that, for example, Smurf attack depends on the use of spoofed IP addresses, this method is also, in itself, a type of attack. They are known as Spoofing Attacks. A router is known to switch IP packets based on the destination address. However, the packet also consists of a source address that is used by the recipient so that it can respond to requests. These attacks aim to modify the packets by falsifying the source address so that the attacker can intercept the information. Although the attacker is able to intercept the communication packets between two hosts, the destination system recognizes the packet as coming from a legitimate source. This is a method also known as Man-in-the-Middle Attack.

2.6 IDS Systems

An intrusion is defined as any type of access or unauthorized activity designed to compromise information security. By identifying suspicious events occurring on a system or network, IDS adds another level of protection to any security infrastructure [22].

Through real-time traffic analysis the system seeks to identify traffic patterns, anomalies and possible threats. In addition, an IDS system instead of blocking traffic only

triggers alerts so that security administrators are aware of intrusions that could have a negative impact on the infrastructure [9]. With this, organizations are provided with a very useful tool in the identification of vulnerabilities and threats as well as in the control of security policy violations.

However, the implementation of this type of systems requires significant theoretical and technical knowledge [23]. The creation of a working group made it possible to coordinate a generalized structure in the area of intrusion detection systems. The group known as Intrusion Detection Working Group (IDWG), constituted in Internet Engineering Task Force (IETF), defined an IDS architecture based on four functional modules [24]:

- Event Block - Collects information
- Database Block - Elements that store the collected information
- Analysis Block - Information processing and analysis module
- Response Block - Execution of the intrusion response

According to the architecture blocks of an IDS solution described above, an IDS solution is composed of three main components: agents, management server and configuration interface. The agent (or sensor) is responsible for monitoring and collecting information about events. The centralized management server consists of a mechanism capable of storing the data collected by the sensors, so that they can then be analyzed before the set of standards that define the alerts. Finally, it is necessary to have some kind of interface between the user and the IDS system so that it is possible to configure the system, monitor the agents and display the collected information. Some important considerations when implementing a IDS solution are: [23]:

- Redundancy in the event of hardware or power failure
- Strategic planning of the system placing
- Necessary resources
- Technical documentation and technical knowledge

- Future needs
- Scalability
- Regular updates
- Quality and accuracy of the collected data

2.6.1 IDS Architecture

A IDS solution is fundamentally designed according to three steps: data collection and processing, analysis based on detection models and outputs. Initially, all information about the events must be collected and processed. These records are the log files that contain all the useful information.

The detection engine analyzes the information collected based on models such as signatures or machine learning algorithms, so that the decision process can then be started regarding the actions that the IDS system must perform (alert, discard, register or even ignore) [25]. The figure 2.1, represents the layout of a IDS solution.

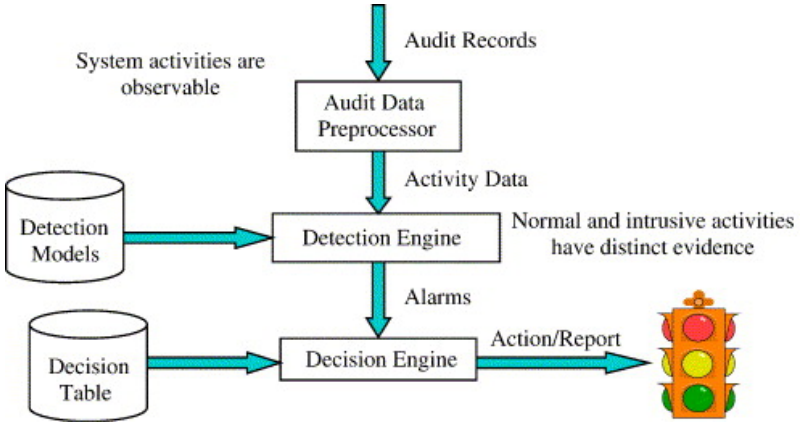

Figure 2.1: IDS architecture schema. Taken from: [25]

2.6.2 Alerts

The Redscan [26] company page, provider of intrusion detection and response services, states that *"The vast quantity of alerts generated by intrusion detection can be a significant burden for internal teams. Many system alerts are false positives but rarely do organisations have the time and resources to screen every alert, meaning that suspicious activity can often slip under the radar."*.

Therefore, it is critical that in addition to the IDS implementation, administrators work extensively on optimizing the system according to their needs in order to reduce the amount of false alerts and increase efficiency in detecting genuine threats. Many organizations choose to implement secondary solutions such as Security Information and Event Management (SIEM), which will be covered in section 2.9, in order to facilitate the analysis of information [27].

There are four types of alerts that can be triggered by the IDS systems, represented in the table 2.4 [28].

Alert	Designation
True negative	normal traffic that is not flagged
False negative	malicious traffic that is not flagged
True positive	malicious traffic that is flagged
False positive	normal traffic that is flagged

Table 2.4: Types of IDS alerts

2.6.3 Signatures

Signatures or, as some times referred, rules, are a set of expressions processed by IDS for detection functions. They are expressions that aim to identify compatibility in the information collected by the IDS, seeking to identify vulnerabilities and apply some types of actions. Signatures can be created by the administrators themselves according to their needs.

However, administrators can use pre-defined sets of signatures distributed by different suppliers which can be freely accessible or paid for through some type of subscription. Considering the wide variety of existing signatures and the frequency with which they are changed and updated, it is recommended to use signature management applications such as PulledPork or Oinkmaster [29].

A signature consists of three fundamental blocks: action, header and options. The first block represents the action that the system must take in case of detecting any correspondence between the analyzed traffic and the signature. The header determines the protocol and IP addresses that constitute the signature action variables and the options that specify details and signature-specific attributes [30]. Table 2.5 shows some of the signatures available on the market.

Name	Licence	Provider
Threat hunting rules	Free	Etnera a.s.
Secureworks suricata-enhanced ruleset	Commercial	Secureworks
Positive Technologies Attack Detection Team ruleset	Free	Positive Technologies
Emerging Threat Pro ruleset	Commercial	Proofpoint
Emerging Threats Open ruleset	Free	Proofpoint
Snort3-community-rules	Free	Talos (VRT)

Table 2.5: IDS Signatures

2.6.4 IDS Types

This section aims to discuss the most important features that allow the classification of IDS systems. The IDS systems can be classified according to the detection methods or the type of events they analyze. Although they are typically implemented for detection only, this systems can also be implemented in a more preventive way. That is, administrators can choose to configure the system so as to block and drop traffic right after suspicious activity is detected.

Active

An active IDS is also known as IDPS (Intrusion Detection and Prevention System). This is because, in addition to the detection functions, this type of IDS also has the ability to discard traffic that matches the configured rules. For this to be possible, traffic must pass directly through these systems. The figure 2.2 represents the implementation of an active IDS.

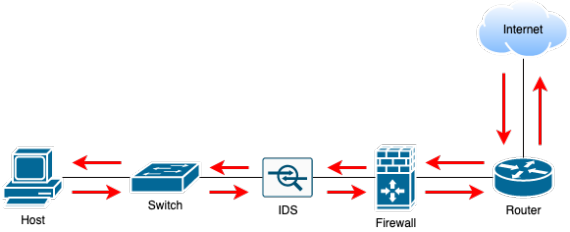

Figure 2.2: Active IDS

Passive

A passive IDS only monitors, detects and alerts the administrator of possible security threats [31]. These systems do not require traffic to pass directly through them and typically receive a copy of the traffic via port-mirroring.

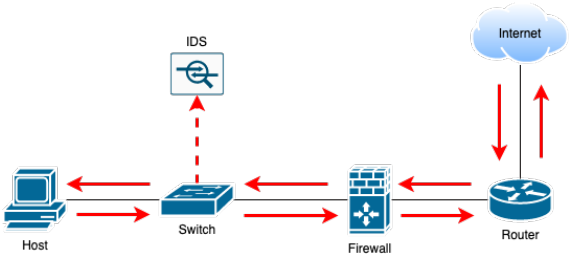

Figure 2.3: Passive IDS

Network-based

If it is network-based, the IDS system is responsible for monitoring all traffic on a given network segment. Through the analysis of the application, transport and network layers of the TCP/IP model it seeks to identify network packets that indicate malicious activities. This type of IDS conducts most of its analysis at the application layer. However, it also analyzes the other two layers in order to facilitate the process. For example, if the TCP port number associated with the event is known, it is possible to infer which application will be used [23].

These systems are typically strategically placed between border routers or on VPN servers, thus making it possible to analyze all traffic circulating on the network. Figure 2.4 represents the implementation of a network-based IDS.

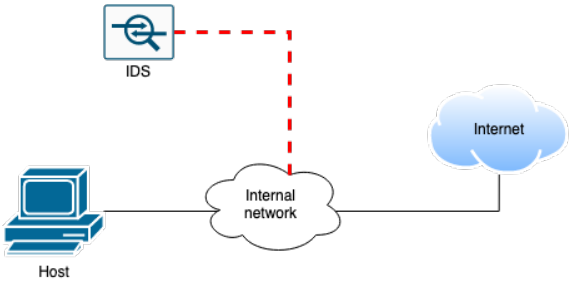

Figure 2.4: Network-based IDS

Host-based

A host-based IDS seeks to identify suspicious activity that occurs in a single host. The system, represented in the figure 2.5, analyzes various components of a system such as the operating system, processes, applications or hardware. The collection of information is guaranteed through an agent installed individually on each machine. However, this type of IDS has three major disadvantages compared to network-based IDS. Firstly, being a tool limited to a single host it does not have a general view of the infrastructure. Then, the operation of the agent requires the consumption of resources, which can affect the performance of the machine. Finally, the IDS agent will have to communicate periodically with the external IDS server, which in turn will analyze the alerts obtained. Because this communication is not in real time there may be delays in the administrators response to possible attacks. Furthermore, if the attack consists of a denial of service, the agent may not be able to effectively send the alert data to the central server [32].

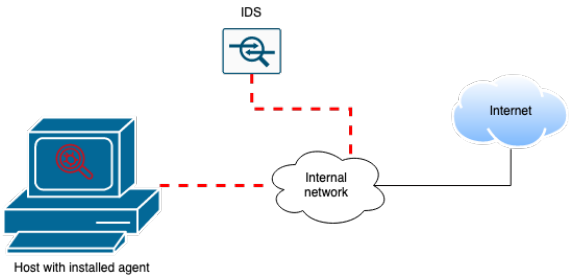

Figure 2.5: Host-based IDS

Signature-based

Regardless of whether it is host-based or network based, an IDS system analyzes traffic and identifies threats using two methods: signature-based and anomaly-based [33].

Detection of suspicious activity can be based on signatures of known attacks or signatures defined by the administrator himself. These signatures constitute databases that define traffic patterns related to previously investigated and identified attacks. Events are analyzed and relevant information is collected. This information is subsequently processed in search of some kind of relationship with the configured signatures. An alert

will be generated if the system identifies that the traffic coincides with an attack pattern defined by the signatures. These systems are classified as signature-based IDS and their architecture is represented in the figure 2.6.

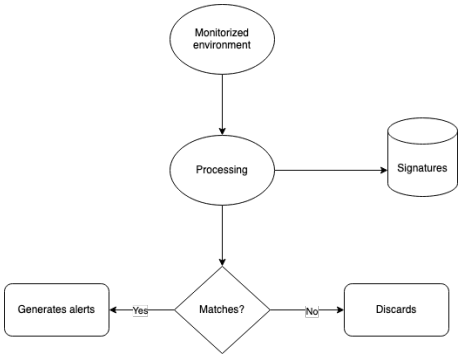

Figure 2.6: Signature-based architecture

Anomaly-based

An anomaly-based IDS seeks to identify malicious activity by analyzing anomalous behavior. These IDS build an initial profile by monitoring various characteristics of a network or system, over a period of time, typically referred to as "Training Period". The architecture of these types of systems is represented in the figure 2.7.

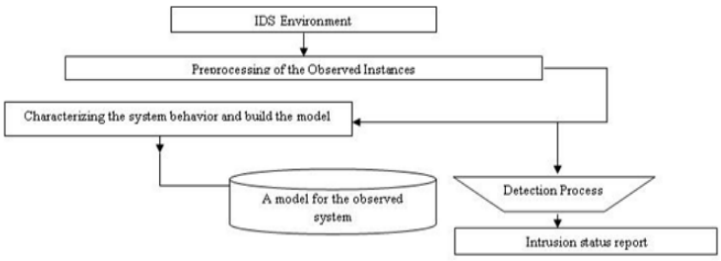

Figure 2.7: Anomaly-based architecture. Taken from: [34]

After creating this profile, the IDS uses several methods to be able to compare the current values with the values collected in the initial profile. The techniques of data mining have become increasingly influential in the development of intrusion detection so-

lutions. The most recent studies in anomaly-based IDS focus on data collection in the training phase using advanced machine-learning [35] techniques. The table 2.6 presents some detection techniques used by an anomaly-based IDS.

Technique	Definition
Statistical-based	based on statistical properties
Knowledge-based	extract data about attacks and vulnerabilities
Machine learning-based	optimization automation
User intention identification	categorize the usual activity of the system or user

Table 2.6: Anomaly-based detection techniques

Signature-based vs Anomaly-based

A signature-base IDS guarantees low frequency of false alarms and efficient detection for known attack patterns. The low level of false positives is due to the fact that the events analyzed are correlated according to previously established attack signatures. However, false positives will always be present. According to Bolzoni and Etalle [36], the optimization of a signature-based IDS should be done based on the deactivation of signatures for vulnerabilities that are not present in the environment that the system is monitoring. The optimization of the system must be constant and requires a complete analysis of the environment. In addition, signatures are regularly updated and new vulnerabilities are discovered every day. In contrast, this means that this type of system has the disadvantage of only recognizing attacks that match the standards defined in those signatures, being unable to detect new variants of attacks that are not addressed by that signatures.

An anomaly-based IDS depends on comparing the activity of a current network or system, with a baseline that represents the activity considered as typical or regular. It is capable of detecting zero-day attacks, that is, attacks that exploit vulnerabilities that are still unknown. However, this leads to a greater number of false positives being triggered. Despite suspicions, certain anomalous actions that cause deviations from the initial profile built in the training phase, do not unequivocally guarantee that an attack is in progress. Figure 2.8 outlines the distinction between signature-based IDS and anomaly-based IDS.

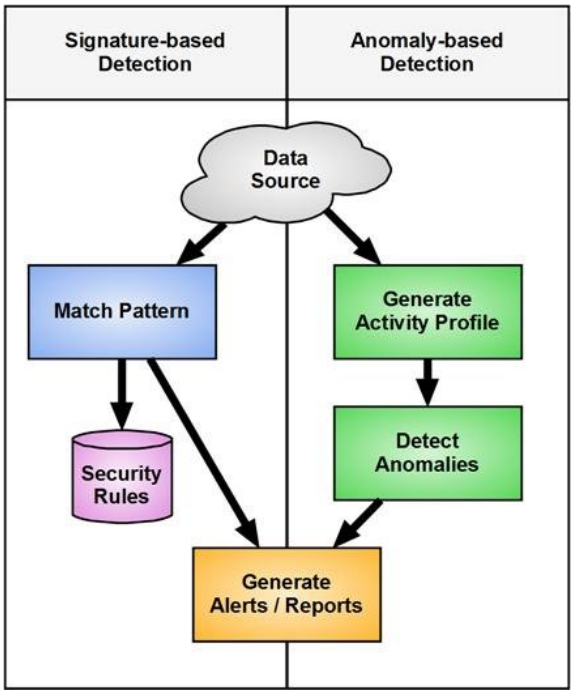

Figure 2.8: Distinction between signature-based and anomaly-based IDS. Taken from: [37]

The detection capacity of anomaly-based IDS is intrinsically associated with the phase of understanding and learning the behavior of a network or a system. An anomaly-based IDS should consist of a detection core that has the ability to process and analyze a wide variety of technologies and protocols. However, this analytical capacity proves to be too expensive in terms of computational resources due to the need to resort to machine learning algorithms and processing large amounts of data in the learning phase [24].

As Jyothsna and Munivara Prasad [35] refer, *"Defining signature sets is one of the main disadvantages of anomaly-based detection. The efficiency of the system depends on the implementation and effective testing of signature sets across all protocols. In addition, a variety of protocols used by different vendors affect the rule that defines the process"*.

2.7 Open Source IDS tools

This section intends to present some characteristics of several IDS solutions existing on the market. The selection was based on the most common tools that are developed in open source, as it is one of the main objectives of the project.

2.7.1 OSSEC

Open Source Security Event Correlator (OSSEC) is a multiplatform tool developed for intrusion detection on hosts (host-based). Performs log analysis, Windows logs monitoring, rootkit detection, real-time alerts and file integrity checking. It is able to respond actively and in real time in preventing threats. The solution consists of small programs installed on the systems to be monitored (agents) and a management server responsible for aggregating and analyzing the collected data. However, it also allows data collection on devices that do not support the installation of an agent [38]. Its architecture is represented in the figure 2.9.

Figure 2.9: Schematic architecture of OSSEC. Taken from: [39]

2.7.2 Hogzilla IDS

It is an anomaly-based IDS tool developed with the objective of monitoring network traffic, supported by various tools such as: Snort, SFlows, GrayLog, Apache Spark, HBase and libnDPI. Some of the malicious traffic that the tool can detect consists of malware, DoS attacks, port-scan, spammers, among others [40]. The figure 2.10 represents the solution architecture.

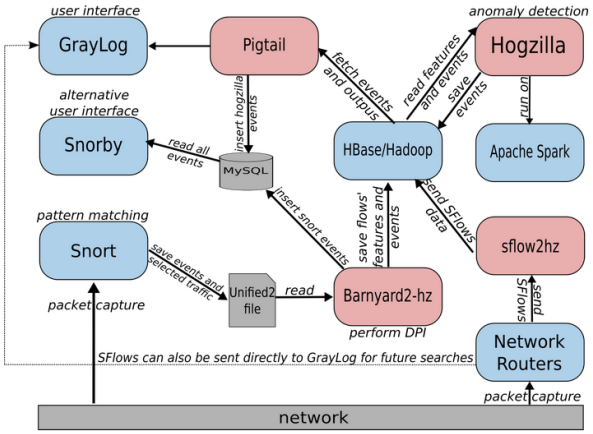

Figure 2.10: Schematic architecture of Hogzilla: Taken from: [41]

SFlows was recently implemented in Hogzilla and is one of the main components of this solution. According to [42], *"By providing unprecedented visibility into network usage and active routes in today's complex networks, SFlows provides the data needed to effectively control and manage network usage , ensuring that network services provide a competitive advantage"*. SFlows is a high-speed network monitoring technology implemented in Hogzilla in order to simplify obtaining data over the network. In the first phase, the identification of traffic flows was only supported by Snort. This was also useful for the training phase of Hogzilla, since it is an anomaly-based tool. However, in large networks, performance, computational cost and the complexity of analyzing a large volume of traffic is significant. SFlows seeks to overcome these difficulties. able to perform a deep inspection of the packets on the network. Nevertheless, it is an auxiliary tool that allows the detection of anomalies that cannot be detected based on signatures [43].

2.7.3 Tripwire

Tripwire is a host-based IDS tool developed exclusively for Linux systems, in order to detect changes in files or directories. Consists essentially of a configuration file, a policy and a database. The policy lists all files and directories that should be stored in the database. The database consists of the list of files and directories that are being stored. Tripwire periodically compares the list of current files and directories with those stored in the database, so you can alert if changes are identified. However, it is not able to generate real-time alerts [44].

2.7.4 Wazuh

Similar to OSSEC, Wazuh is a multiplatform host-based IDS tool composed of a collector agent and centralized management server, as represented by the figure 2.11. The various agents try to collect all the data so that they are processed and presented to the user through the centralized node.

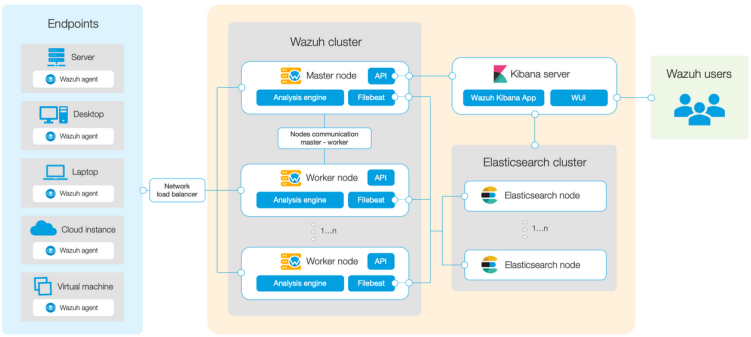

Figure 2.11: Schematic architecture of Wazuh. Taken from: [45]

It consists of a SIEM with the ability to obtain, aggregate, index and analyze data from different systems. It also has the ability to respond actively in order to prevent threats. It combines anomaly-based and signature-based techniques in the search for intrusions and vulnerabilities. It has full integration with the Elastic stack so that the generated alerts are indexed, stored and presented in an intuitive interface for visualization and data management [46].

2.7.5 Snort

Snort is a network-based IDS tool owned by Cisco since 2013. In addition to being a IDS, it can also be configured and implemented as an Intrusion Prevention System (IPS). As for detection methods, Snort combines signature-based and anomaly-based methods. However, it has the disadvantage of not making the most of the machine's processing resources because it is a single-threaded tool. The Snort3 version, still in development, seeks to overcome this limitation by reformulating the tool architecture to support multithreading. In order to overcome the disadvantage of the limited processing power of Snort, one typically resorts to the creation of multiple instances of the tool for each CPU core of the machine. However, this is not a viable option in terms of scalability [37].

Snort can be configured to analyze packets associated with services that do not exist on the network. The network administrator can define specific signatures that warn of traffic that seeks to exploit network services or ports that are not being used on your network [47]. Snort has always been considered agnostic to the type of application protocol. In order to adapt and not lose ground to its most direct competitor (Suricata), the creators of Snort launched OpenAppID in one of the most recent versions of Snort. This plugin allows software to identify the protocols associated with the packet [48].

There are three types of implementation: active mode, passive mode and active test mode. In active mode (inline), Snort acts as a IPS blocking traffic classified as malicious. In passive mode, the traffic flow has no impact and Snort only analyzes traffic typically through a port-mirror, in search of possible suspicious activity. Finally, the active-test mode simulates the performance in active mode, but with the particularity of not interfering with the traffic flow of the network [49].

Snort architecture is represented in the figure 2.12. In a first phase, the capture module performs the capture of packets from the network interface. By default, this module is based on the Libpcap library on Linux systems and the Winpcap library on Windows systems. After capture, the decoding module analyzes the contents of the collected packets. The preprocessors receive the packets and perform some filtering, analysis and manipulation functions before they are sent to the detection engine. This process is

particularly advantageous as it avoids an additional load on the hardware of the machine. The detection engine applies the signatures in order to understand whether an alert should be generated. Finally, the output module guarantees the presentation of the information to the administrator [50].

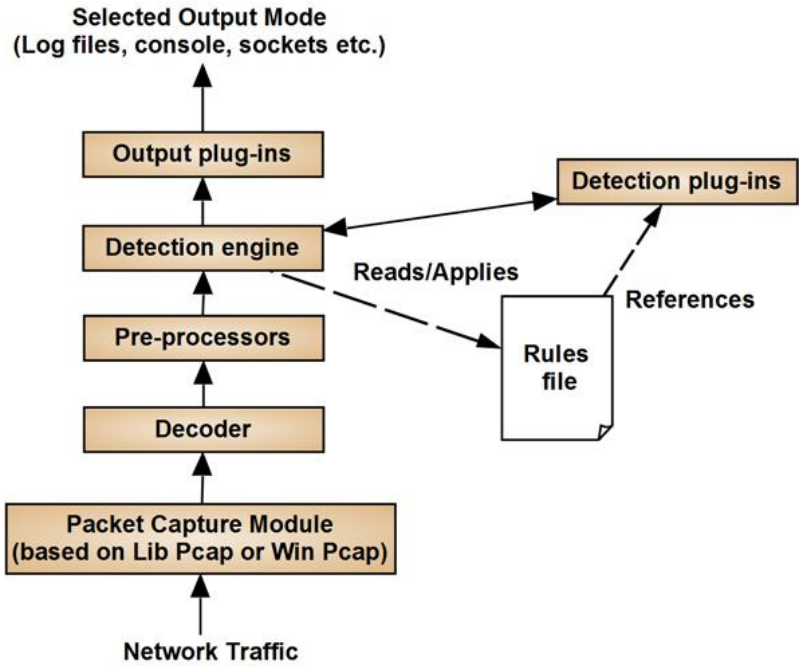

Figure 2.12: Schematic architecture of Snort. Taken from: [37]

The base format for Snort log files is Unified2. This means that they will be written to the disk in binary format and that it will be more difficult for the administrator to analyze. However, there is the option to use the barnyard2 tool in order to transform the logs into the syslog format. It is also possible to convert the format of logs into pcap files, which can then be analyzed with programs such as Wireshark.

The Snort detection engine analyzes packets based on extensive attack signature lists. Despite this, Snort has the particularity of not relying exclusively on signatures for an alert to be triggered. Through several preprocessors, Snort guarantees the creation of alerts based on anomalous behavior. However, this can generate a significant number of false alerts that contribute to less accuracy in detecting malicious events. In certain cases, a possible solution will be to disable or adjust the configuration of this module as needed

[37]. The 2.7 table represents the modules of the Snort architecture.

Module	Function
Packet capture	packet capture for analysis
Decoder	extract useful information from the network packet
Preprocessor	packet filtering, identification and manipulation
Detection engine	comparison of captured packets with defined signatures
Signatures/Rules	list of signatures related to attacks
Output	information display

Table 2.7: Snort achitecture modules

It is also relevant to realize that the preprocessors must be properly configured in order to prevent the transit of useless information to the detection engine [37]. Preprocessors are responsible for activities such as stream reassembly, defragmenting IP, detecting anomalies such as port scans or inspection at the application layer level of the TCP/IP model. In short, the preprocessors perform a prior investigation of the packets before they reach the detection engine, where the traffic will be analyzed according to the defined signatures [51]. The 2.8 table shows some of the preprocessors that are installed in Snort by default [52].

Name	Function
http_decode	normalization of web traffic
PortScan	detection of portscans
Degfrag	defragmentation of IP addresses
Stream	reassembly of the TCP stream

Table 2.8: Preprocessors included in Snort

As previously mentioned, signatures are composed of three blocks (share, header and options). By default, and if Snort is implemented passively, the available actions are alert, log or pass. However, if Snort is actively implemented and acting as a IPS there are three more options, which are drop (blocks and registers the packet), reject (blocks, registers the packet and responds with ICMP unreachable) and sdrop (block the packet but does not register it) [53].

The signature header specifies protocols, IP addresses and network ports that the signature intends to analyze. Snort is limited to the TCP, User Datagram Protocol (UDP), ICMP and IP protocols. The figure 2.13 represents an example of a Snort signature [54].

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|"; msg: "mountd access";)
```

Figure 2.13: Snort signature example. Taken from: [54]

2.7.6 Zeek

Formerly known as Bro, Zeek is a network-based IDS and anomaly-based tool. However, Zeek does not have a prevention feature (IPS) and, unlike Snort and Suricata, it can only be passively implemented. The tool allows administrators to write their own code in a specific script language, in order to specify analysis tasks according to their needs [55].

The official documentation for Zeek [56], makes a point of highlighting that *"Zeek goes well beyond the capabilities of other network monitoring tools, which usually remain limited to a small set of tasks analysis tools embedded in code. We emphasize in particular that Zeek is not a classic signature-based intrusion detection system. While supporting this functionality as well, the Zeek scripting language facilitates a much broader spectrum of approaches to detect malicious activity"*.

The tool performs an extensive analysis and categorization of traffic and alerts the administrator about possible threats. All information is stored in structured log files so that they can be processed by external software. In addition to a comprehensive analysis of all the connections observed on the monitored network interface, these logs allow for a deeper analysis such as, for example, all Hypertext Transfer Protocol (HTTP) sessions with the requested URIs, Domain Name System (DNS) requests with responses, Secure Sockets Layer (SSL) certificates, content of Simple Mail Transfer Protocol (SMTP) sessions, among many others. In addition to recording information in log files, Zeek has features such as extracting files from HTTP sessions, reporting on vulnerable versions of software on the network, identifying web applications, Secure Shell (SSH) brute-forcing detection and SSL certificate validation [56].

Zeek is specifically developed for monitoring high performance networks. This high performance is achieved through scalable load balancing techniques. Typically, clusters are created and a load balancer distributes traffic over a number of machines that run individual instances of Zeek. Each one of these instances analyze their individual share of traffic. The centralized management server acts as the coordinator of all these processes and provides a configuration and management interface for all the logs [56].

2.7.7 Suricata

It is a tool supported by Open Information Security Foundation (OISF) and developed based on the Snort architecture. It is a powerful multiplatform network-based and signature-based IDS that can also be implemented and configured as an IPS.

Unlike Snort, Suricata is a multi-threaded tool. This contributes to being able to take better advantage of the machine's processing resources and analyze a larger volume of traffic in a more efficient way. The packets are placed in stream queues that will later be captured and processed by a given thread. The administrator can configure and define the number of detection threads, depending on the number of cores on the machine. As a tool that shares the same structural basis, in addition to paid and community signatures that can be downloaded from various suppliers, Suricata can also be implemented based on Snort signatures [37].

Suricata signatures are quite identical to those used in Snort, however, Suricata signatures are not as limited as those of Snort in terms of protocols. Suricata can use signatures that analyze TCP, UDP, ICMP and IP protocols, as well as application layer protocols such as HTTP, File Transfer Protocol (FTP), Transport Layer Security (TLS), Server Message Block (SMB), DNS, SSH and SMTP [30]. Table 2.9 shows the actions that can be performed by Suricata signatures.

Action	Function
alert	generates alerts and registers the packet
pass	ignores the packet
drop	rejects the packet and generates an alert
reject	sends RST/ICMP unreach messages to the source
rejectsrc	same as a reject
rejectdst	sends RST/ICMP unreach messages to the recipient
rejectboth	sends RST/ICMP messages to both source and destination

Table 2.9: Suricata signature actions

Suricata has the ability to detect protocols regardless of the network port that is being used. For example, in Snort the preprocessor that processes HTTP traffic will have to be previously configured on specific ports. On the other hand, Suricata is agnostic about the network ports used by the traffic. If the user wants to detect traffic from the application layer without specifying any network port, it is sufficient to write a signature that specifies the protocol in the header along with the term "any" as it can be seen in figures 2.14, 2.15 and 2.16.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS ...
```

Figure 2.14: Snort HTTP signature. Taken from: [57]

```
alert http $HOME_NET -> $EXTERNAL_NET any ...
```

Figure 2.15: Suricata HTTP signature Nr.1. Taken from: [57]

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (app-layer-protocol:http; ...
```

Figure 2.16: Suricata HTTP signature Nr.2. Taken from: [57]

Like Snort, Suricata relies on external libraries to capture packets. In this case, the AF_PACKET and PF_RING libraries are used on Linux systems [47].

In addition, Suricata supports file extraction and automatic detection of protocols in order to apply the appropriate signatures to the type of traffic. It can also probe the reputation for IP addresses which makes it possible to identify malicious traffic from addresses reported as dangerous [37].

The tasks of registering and analyzing information about protocols such as HTTP, DNS and TLS/SSL make Suricata a very efficient tool in a network security monitoring environment. The output information is done in JSON format, which facilitates integration with data processing aggregation tools [58]. It's architecture is represented in the figure 2.17.

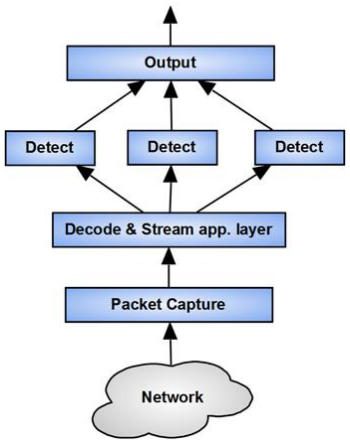

Figure 2.17: Suricata schematic architecture: Taken from: [37]

2.7.8 IDS comparative table

This section presents a comparative table of the various characteristics of the IDS discussed above.

IDS	Events	Detection method	OS	Agent OS
OSSEC	Host-based	Anomaly-based/Signature-based	Linux	Multiplataform
Hogzilla IDS	Network-based	Anomaly-based	Linux	n/a
Wazuh	Host-based	Anomaly-based/Signature-based	Linux	Multiplataform
Tripwire	Host-based	Signature-based	Linux	Multiplataform
Zeek	Network-based	Anomaly-based	Linux	n/a
Snort	Network-based	Anomaly-based/Signature-based	Multiplataform	n/a
Suricata	Network-based	Signature-based	Multiplataform	n/a

Table 2.10: IDS comparison

2.8 SIEM - Security Information and Event Management

Implementation of an IDS system should be the main focus of an intrusion detection architecture. However, it is important to integrate tools that simplify the risk management process [12].

These tools result from the combination of the concepts of Security Event Management (SEM) and Security Information Management (SIM). While a Security Information Management (SIM) solution seeks to correlate events and monitor logs in real time, a Security Event Management (SEM) solution stores logs over a period of time for further analysis [59].

A Security Information and Event Management (SIEM) system aims to complement the implementation of IDS tools. While keeping the focus on open source software, a preliminary study on the most common SIEMon the market was carried out.

2.8.1 OSSIM

Open Source Security Information Management (OSSIM) is an open source security event management software. It is referred to as one of the most diverse and complete solutions in terms of analytical capabilities. Its architecture is outlined in the figure 2.18

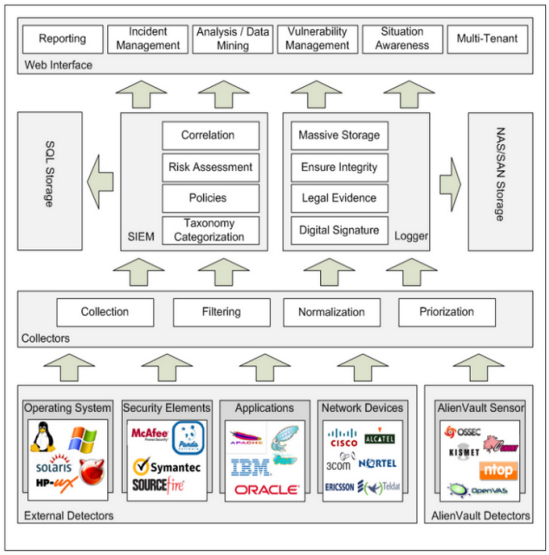

Figure 2.18: Schematic architecture of OSSIM. Taken from: [60]

The event correlation functionality proves to be an advantage as it allows for a much more complete and detailed risk analysis. This tool is composed of multiple security monitoring tools in order to collect and analyze all the information gathered in a single centralized environment [12].

All the information is collected and filtered by the collecting agents so that it is then sent to the SIEM module. This is the module that performs the most important functions of the system, such as the correlation of events and detection of possible intrusions. Through the web interface, analysts can view all information about the events analyzed and access the system settings. The solution consists of the tools shown in the table 2.11 [60].

Tool	Function
Arpwatch	monitor changes in IP and MAC addresses
P0f	operating system detection
Pads	detect anomalies in services
Nessus	vulnerability check
Snort/Suricata	network-based intrusion detection
Spade	Snort anomaly-based preprocessor
Tcptrack	TCP sessions information
Ntop	network monitoring
Nmap	network port recognition
Nagios	service monitoring
OSSEC	host-based intrusion detection

Table 2.11: OSSIM solution tools

2.8.2 ELK

It is considered one of the most popular solutions in terms of centralization, analysis, processing and storage of logs related to events. The stack consists of three tools: Elasticsearch for storage and indexing, Logstash as a log aggregator/processor and Kibana as a display interface [61]. However, some say that it cannot be fully considered as a SIEM due to the lack of alerting capabilities and event correlation capabilities. The absence of these capabilities can be filled by various open source and commercial add-ons [61]. Some of the main features of the ELK stack are represented in the figure 2.19.

	Yes/No	The but...
Log collection	Yes	One of the ELK Stack's core capabilities. Large and diversified data pipelines involve more than the ELK Stack however.
Log processing	Yes	Complex processing requires close monitoring and architectural considerations in designing the pipeline.
Storage	Yes	Requires organizational engineering commitment and expertise for guaranteeing HA and scalability. For historical data, long-term retention is required.
Querying	Yes	Depends on accurate parsing. Requires a certain amount of expertise.
Correlation	No	ELK does not provide correlation rules. Correlation will depend heavily on analytical work.
Dashboards	Yes	Extremely powerful, requires expertise.
Alerts	No	Is not provided out-of-the-box. Can use hosted ELK, commercial or open source add-ons.
Incident management	No	Totally out of scope for the ELK Stack. Requires hooking in additional tools.

Figure 2.19: ELK stack functionalities. Taken from: [61]

Recently, a new tool has been integrated into the ELK stack - it's called Beats. These are log shippers installed on the monitored machines. Figure 2.20 represents the data flow in the ELK stack. The data collected by Beats can be sent directly to Elasticsearch or to Logstash.

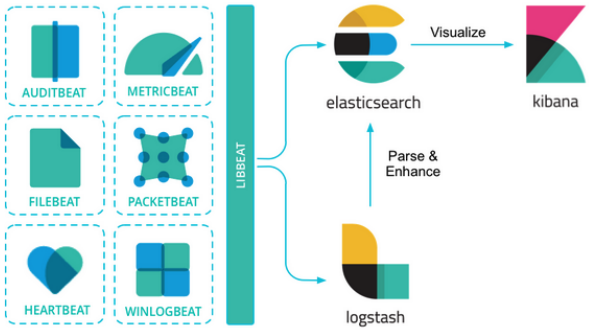

Figure 2.20: Beats schematic architecture. Taken from: [62]

To send the data directly to Elasticsearch, the user must use specific modules that were recently developed in order to process information from specific sources (Suricata, Snort, Zeek, etc.) without the need to use Logstash processing [62]. There are currently eight types of Beats that analyze and collect different types of data:

- Auditbeat - collects data from Linux systems audit framework
- Filebeat - collects general log files
- Functionbeat - collects cloud data
- Heartbeat - collects service availability data
- Journalbeat - collects syslog registry
- Metricbeat - collects system metrics
- Packetbeat - collects network traffic
- Winlogbeat - collects Windows events

2.8.3 SIEMonster Community Edition

It is an IDS solution composed of several open source security tools. In addition to the community and free version, it has paid versions with all available features. However, due to the fact that it is a relatively recent tool, there is still no extensive information on its operation.

SIEMonster [63] official website states that the community edition comes fully loaded requiring a minimum of 32GB of RAM and 8 vCPUs. It only allows monitoring of up to 100 machines, so it will not be a viable option for implementation in large production environments.

Author Bernardo [64] decided to remove this tool from his study due to significant needs in terms of computational resources. Some of the SIEMonster features are represented in the table 2.12.

Tool	Function
Open Distro Elasticsearch	indexing and data visualization tool
The Hive	incident manager
MITRE ATT&CK	informational database on attacks
MISP Framework	threat information and sharing platform
PatrOwl	security operations manager
ApacheKafka	distribution platform for distributed data
Ni-FI	data flow visualization
ElastAlert	alarming

Table 2.12: SIEMonster functionalities

2.8.4 Wazuh and OSSEC

The Wazuh and OSSEC Host Intrusion Detection System (HIDS) can be considered as SIEM solutions. However, in the case of OSSEC, this statement is open for discussion. Although it has a significant capacity for collecting and analyzing information, it lacks the necessary log management capabilities that must be part of a SIEM [65]. Both tools were analyzed in the sections 2.7.4 and 2.7.1, respectively.

2.8.5 Splunk Free

Splunk Free is a real-time information monitoring and analysis platform. It is, in certain aspects, similar to the ELK stack. However, the free version is limited to 500MB of information per day, which turns out to be very little for the current needs of most organizations [66].

2.8.6 Graylog

Graylog is considered one of the best centralization and management solutions for logs. In addition, the tool has quite interesting features to be considered a viable option in the process of choosing a SIEM. The open source version of Graylog also allows the correlation of SIEM events which turns out to be one of the most attractive features in this type of tools [67].

The Graylog component works in conjunction with Elasticsearch instances to manage logs. Elasticsearch only serves as a tool for indexing and storing logs so that they can then be analyzed and managed. Graylog adds an abstraction layer on top of Elasticsearch that allows the user more simplified access to data [68].

Graylog's official page [69] states that the introduction of the new event correlation functionality has opened up new possibilities for implementing this tool as a SIEM. Graylog has the capacity to ingest, process and store large amounts of information, coming from multiple sources depending on the specific needs of each organization. The web interface has options for search and organization of information that are quite complete and facilitates the tasks of analysis of the administrators.

According to the official documentation of Graylog [70], before implementing the solution one must define which strategy to implement, that is, *"Even in a small organization, modern environments produce a lot of data from log [...]. Today, 5GB a day is not unusual for a small environment. A more complex environment can produce a thousand times more than that. Assuming an average event size of 500k, 5GB per day is equivalent to 125 log events every second, which is equivalent to about 10.8 million events per day. With so much information to be generated, an effective management strategy is needed"*.

That said, two strategies are indicated below: minimalist and maximalist. In the case of the minimalist, only the logs identified as essential are collected. This proves to be an advantage in reducing the processing load and the volume of events analyzed, but more importantly, the logs that are not needed do not interfere with the analyst's work, thus maintaining a greater focus on the events that really matter. The maximalistic approach defines that all logs from multiple sources are collected and saved, which implies a higher

usage of resources and processing, not being the most practical option in terms of costs.

2.8.7 SIEM comparative table

This section provide some comparative information about the SIEM tools discussed above. Splunk Free is considered an open source tool, although is restricted to 500MB of data per day which is a very low compared to the organizations necessities nowadays.

SIEM	License	Event Correlation	Log Management	Vulnerability Checks	Scalability
OSSIM	Open source	Yes	No	Yes	No
ELK	Open source	No	Yes	No	Yes
SIEMonster	Open source	Yes	Yes	Yes	No
Wazuh	Open source	No	Yes	Yes	Yes
OSSEC	Open source	Yes	Yes	Yes	Yes
Splunk Free	500MB per day	No	Yes	No	Yes
Graylog	Open source	Yes	Yes	No	Yes

Table 2.13: SIEM comparison

2.9 NSM - Network Security Monitor

These are tools that are dedicated exclusively to the detection and analysis of data related to the network. They are usually distributed as complete Linux security solutions. The main open source Network Security Monitoring (NSM) that are worth mentioning are Security Onion, RockNSM and SELKS.

2.9.1 RockNSM

This platform consists of several tools which together form a robust and complete network security monitoring solution [71]:

- AF_PACKET - Packet capture
- Zeek - Traffic analysis
- Suricata - Network-based intrusion detection
- FSF - File verification
- Apache Kafka - Message management
- Logstash - Log processing
- Elasticsearch - Indexing and data search
- Kibana - Graphical data visualization
- Docket - PCAP file rotation
- Stenographer - Packet capture and storage

To better understand its architecture, figure 2.21 demonstrates the schematic representation of its functional flow.

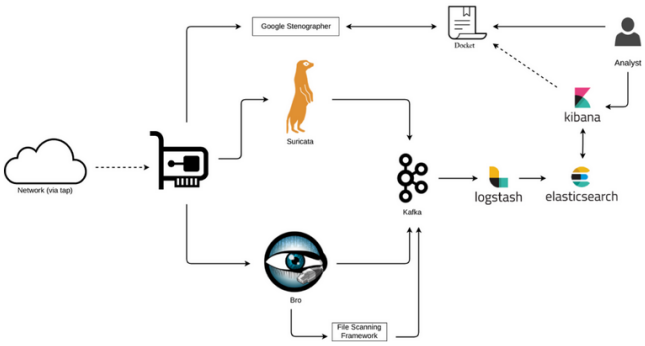

Figure 2.21: RockNSM schematic architecture. Taken from: [71]

2.9.2 Security Onion

According to the official documentation [72], it is a Linux distribution composed of several tools for network monitoring, intrusion detection and log analysis. In addition to providing software that can be downloaded and installed free of charge, Security

Onion Solutions also has complete proprietary hardware solutions in order to enhance the performance of your product.

The platform is divided into three main functionalities: packet capture, intrusion detection and data analysis. Despite being typically considered as a NSM, Security Onion provides analysis and event detection capabilities on hosts. The documentation further states that *"As an analyst, being able to correlate events based on hosts with network-based events can be the difference in identifying a successful attack. A new addition to Security Onion 2 is osquery, which is another free and open source endpoint agent. In addition, Security Onion can collect data via Syslog or another agent transport such as Beats"*.

That said, in addition to the powerful network monitoring capabilities, Security Onion allows the possibility to analyze systems. For this function is used the Wazuh HIDS, already described in the 2.7.4 section. Figure 2.22 represents the network traffic flow in Security Onion. Traffic is captured via the network interface, processed by the IDS tools and transported by agents to the ELK stack in order to be indexed and displayed to the user.

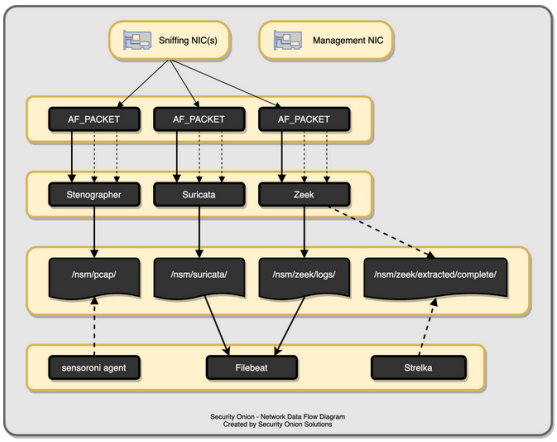

Figure 2.22: Security Onion Traffic flow. Taken from: [72]

For data management and analysis, the platform includes a Security Onion Console (SOC), which is a web interface that allows the administrator to manage the analyzed data. Figure 2.23 represents the architecture of the solution and we can see that the monitoring tools send their logs to the ELK stack, which in turn will process and prepare

the information so that it is made available to the user. Some external platforms can be very useful for sending alerts generated by the system. In the public repository of Security Onion [73], a detailed explanation of all the possible implementation models for the platform is available.

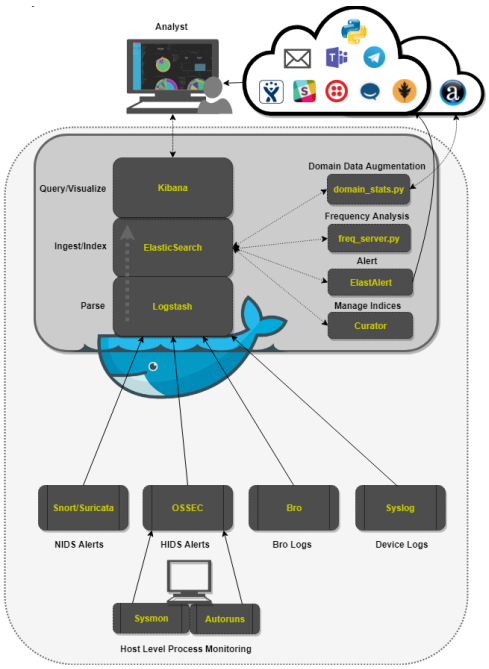

Figure 2.23: Security Onion architecture. Taken from: [74]

2.9.3 SELKS

It consists of an open source Debian distribution, developed by the company Stamus Networks. It comprises Suricata for intrusion detection, ELK stack for filtering, storage and visualization and also the community edition of Scirius. Scirius is a signature management tool in order to facilitate the optimization process of Suricata [75].

Recently, the tools EveBox and Arkime have been added. EveBox is a web based tool for managing alerts and events generated by Suricata. Arkime consists of a packet capture and analysis system.

The platform has an active community and very straightforward documentation on the GitHub platform. Some of the contributors in the development of this project are members of the Suricata development team [76].

2.9.4 NSM comparative table

This section provides a comparison table between the NSM discussed above. Note that only Security Onion is able to analyze both network and host events (through Suricata NIDS and Wazuh HIDS). While SELKS and RockNSM are just open source distributions that can be installed on any compatible device, Security Onion Solutions also provides a completely optimized solution based on proprietary hardware.

NSM	OS	IDS Tool	Proprietary Hardware	Data storage	Full Packet Capture	Minimum Specs
RockNSM	Centos7	Suricata	No	ELK	Yes	4+ CPU cores, 8GB RAM, 256GB storage
Security Onion	Linux (Centos7 and Ubuntu 18.04)	Suricata/Wazuh	Yes	ELK	Yes	12GB RAM, 4 CPU cores, 200GB storage
SELKS	Debian	Suricata	No	ELK	Yes	2 CPU cores, 8GB RAM

Table 2.14: NSM comparison

2.10 Related work

This section presents some studies and research carried out over the years in the intrusion detection area.

Authors Hu, Yu and Asghar [47] analyze the challenges of implementing an open source IDS solution on high-speed networks. They seek to study the implementation of the Snort and Suricata tools on a 100GB/s network and analyze their performance without resorting to new methods of packet capturing or updating the available hardware. The authors found that a multithreaded architecture can significantly improve the performance of IDS, in terms of dropped packets. Both IDS have demonstrated that they are more efficient at traffic below 60GB/s. Above that, some packet loss starts to be detected.

The authors Luthuli, Oki, Tarwireyi and Adigun [55] assess the effects of hardware during a DoS attack, using Bro (currently Zeek). For their analysis, they resorted to the TCP flood attack [3] and three metrics of performance evaluation, being the availability of resources, the quantity of discarded packets and the network transfer rate. The authors conclude that the greater the processing capacity of the machine, the more packets will be processed. Zeek categorizes TCP flood traffic as a threat and therefore records everything in its logs. The TCP packets are sent at a very high rate, which implies an above average processing effort for the IDS system. The authors concluded that Zeek is overloaded in terms of processing, due to the fact that it is single threaded. They also concluded that the reason for the packets to be dropped is directly associated with the use of the machine's resources.

Morais [15] studies and implements an IDS/IPS solution at the Faculty of Sciences of the University of Lisbon. In this case, the author chooses to implement a hybrid solution (host-based and network-based), using OSSEC and Snort respectively. Some tests are carried out in a closed environment in order to understand the efficiency of the solution in detecting malicious events, as well as the impact on the existing infrastructure. The author concludes that the implementation of firewalls and IDS solutions is not enough for an infrastructure to be safe and protected. In addition, it is also very important to adopt efficient security policies, competent management teams and an education of users in order to avoid situations that could jeopardize the security of organizations.

The authors Alhomoud, Munir, Disso, Awan and Al-Dhelaan [77] carry out the study of the performance of Snort and Suricata in different operating systems and in a high-speed network. They chose to analyze all scenarios with identical signature configurations, different protocols and different packet sizes. They concluded that there was a significant number of packet loss with virtualization, due to the virtual allocation of RAM and disk space. The number of packets received on the network interface is greater than what the virtual machine can handle due to the limitations of data transfer on disk, which will directly affect the performance of the IDS systems. In high-speed networks, the authors concluded that Suricata is more efficient in Linux environments, while Snort is more efficient in FreeBSD environments.

[3] It is a DoS attack type that seeks to overwhelm the target server with incomplete TCP sessions.

Albin [78] seeks to make a performance evaluation between Suricata and Snort. The author states that both solutions triggered false positives and false negatives. However, it is said that this fact is due to the inefficient set of signatures that was used. The determination of the system that has the best detection method was inconclusive. The author states that Suricata uses more computational resources than Snort, largely due to its multi-threaded architecture. In addition, he identified Suricata's advantage in terms of scalability. The author states that the need to implement several instances of Snort, in order to compensate for their deficiency in terms of processing, can prove to be complex in terms of operation and maintenance. The author also reinforces the idea that the optimization of the tool according to the needs of the organization is very important. If signatures are properly configured and optimized, the ratio of false negatives to false positives can be considerably lower. As Alhomoud, Munir, Disso, Awan and Al-Dhelaan indicate in [77], the author states that the implementation of IDS in a virtualized environment proved to be a complicated experience.

Tavares [12] proposes the implementation of a security event management solution (Open Source Security Information Management (OSSIM)). The solution allows for staggered installation on several machines, which is always an advantage in terms of flexibility of computational resources. The author makes a point of emphasizing that the correlation of events is one of the strengths of OSSIM, since it is very intuitive and offers a wide variety of possibilities. There are certain limitations such as scarce documentation, significant usage of hardware resources, inconsistency in updates and some Snort inefficiency in detecting events within the internal network. However, the user interface is intuitive and diverse in terms of visualizations and data presented proving to be a very robust and complete solution despite the limitations found.

Calado [29] defines as its main objective the implementation of a low-cost open source IDS solution. A comparison was made between Snort and Suricata, and it was immediately apparent that the performance of Snort was significantly inferior to that of Suricata. The author states that all configurations and optimizations performed in Snort were unable to reduce packet loss. The solution was implemented in IDS mode, together with a Linux-based firewall. Although the number of packets lost by Snort was minimal, the author says that the smallest loss is unacceptable to happen in such

a complex infrastructure. In addition, Suricata proved to be easier to configure and its extensive documentation proved to be a fundamental aid in the implementation process. In terms of consumption of computational resources, both solutions proved to be low consumers of RAM memory. In less demanding environments, the consumption of 1GB of RAM would be sufficient. In more demanding environments, RAM consumption has never been greater than 10GB. The author states that it is extremely important to properly configure and maintain the solution, in order to avoid wasting resources. The signatures management must be very careful, with the aim of reducing the number of false alarms as much as possible. As for packet capture, the PF_Ring library has considerably improved the performance of the solution.

The authors White, Fitzsimmons and Matthews [79] faced several obstacles in the scalability of Snort and Suricata in their base configurations. Their results revealed Suricata's superior performance over Snort, both in a single-threaded and multi-threaded configuration. The authors expected an advantage of Snort in a single-threaded configuration, which did not turn out to be true. In addition, Suricata was able to use, on average, less computational resources than Snort.

The authors Chintan and Kirtee [80] compare the performance of Snort and Suricata. They conclude that, analyzing the same amount of traffic, Suricata proved to use more computing resources because of being built in a multi-threaded architecture. However, Suricata has the advantage of supporting an increase in traffic without resorting to the configuration of multiple instances, while Snort is limited to less than 300Mbps bandwidth in a single instance.

Rodfoss [81] performs a comparison between three IDS tools, being Zeek, Snort and Suricata. In a first analysis, the author reveals that Suricata was the simplest tool to install and configure. The Suricata installation included all the necessary packages for its operation. In turn, Zeek and Snort caused some problems in terms of missing or incompatible packages. According to the author, the tool that caused the most complications was Zeek. As for the testing scenario, the author chose to analyze traffic captured by tcpdump. After analyzing the approximately 40GB file, Snort turned out to be the most sensitive tool generating 400,000 alerts. Zeek has also created a considerable number of alerts (100,000). However, many alerts are not critical and it is possible to set an alert

limit generated in Snort. The alerts file presents easy-to-read information, which is also a positive point. On the other hand, Zeek does not aggregate all the alert information in one file. The tool creates specific log files for each protocol, which facilitates the discovery of different types of attacks and intrusions. However, the author states that the amount of information in Zeek logs is scarce compared to that presented in Snort. Suricata had a significantly lower number of alerts, largely due to the limited set of signatures that were used. The alerts are saved in a specific file called fast.log.

Mauno Pihelgas [82] performs a comparison between the IDS tools Zeek, Snort and Suricata. In that study, several experiments are carried out on a 1Gbit/s network. However, the author chose not to analyze and test the accuracy of the detection rules, the main metric for evaluating performance being packet loss. In a first phase of experimentation, the author establishes that the systems must be implemented in their base configurations. In a second experiment, the configuration has undergone some modifications. Finally, different packet capture modules were tested. In their base configurations, the systems were only able to handle 100Mbit/s of traffic. Above that, packet loss begins to be observed. After optimizations, Snort was able to handle approximately 450Mbit/s of traffic, however, it was not possible to configure Snort with the PF_RING capture module due to errors not specified by the author. Suricata proved to be quite efficient in all experiments, largely due to its multi-threaded architecture. However, the author noted that Suricata consumed much more memory than the other two tools. Using the AF_PACKET and PF_RING modules, Suricata had no loss of packets analyzing 1Gbit/s traffic. Due to compatibility errors, the author reveals that Zeek was not part of several experiments. Despite this, in 1Gbit/s traffic and using the PF_RING module, packet loss was null.

Authors Hu, Asghar and Brownlee [83] carried out different experiments and different configurations on Snort, Zeek and Suricata on a high-speed network. The solutions were tested in a 10Gbit/s environment, with different traffic flows and alternating between base configurations and optimizations in order to understand the impact in terms of resources and packet losses. The authors concluded that, in terms of packet loss, Snort and Suricata proved to be very effective after optimizing the configuration and capture methods, with only 0.3% of packets lost. Suricata proved to be more efficient with an average use of 23% processing power in more demanding environments due to Suricata's ability

in distributing processing load among the available CPUs. As for memory, the author reveals that the different network environments tested do not have a significant impact on memory consumption. In order to optimize the performance of Zeek, the author suggests techniques for distributing traffic across multiple instances. As for Snort, some methods and algorithms that enable the optimization of the tool were analyzed.

Authors Saboor, Akhlaq and Aslam [84] studied the performance of Snort configured on different types of hardware and during an DoS attack. The authors report that by optimizing single-core systems, Snort decreased the ratio of lost packets by almost 50%. On multi-core systems, no improvements were revealed. The increase in RAM showed an improvement in the performance of Snort. The authors conclude with their study that the optimization of hardware improves the performance of Snort. However, this only happens on single-core systems. In multi-core systems, the optimization of hardware does not enhance the packet management and detection capability of Snort. The author reveals that the reason for this is due to the single-core limitation of Snort architecture. The author also concludes that, in both configurations of hardware (single-core and multi-core), it was not possible to detect any evolution in the detection of DoS (TCP flooding) attacks. Finally, it is said that Snort is not able to detect these attacks when the limit Rate_filteration [4] defined is not exceeded.

Authors Wagh, Pachghare and Kolh [86] carried out a study on IDS solutions based on Machine Learning techniques. In a first analysis, it is stated that Machine Learning is a very complex technique and that it still proves to be a little immature and not very optimized for use in IDS solutions. In addition, the authors reveal some challenges and obstacles inherent in the development of IDS solutions using these techniques, such as the time needed to process large amounts of data in the learning phase, the inability to reduce the numbers of false positives, the need of datasets standard and complex and very resource-consuming implementation.

The authors Vazão, Santos and Piedade Rabadão [59] analyzed and carried out a comparative study of four of the main open source SIEM solutions available on the

[4]Snort functionality that allows defense against attacks DoS. Filters are used to modify the action of a signature when the flow of events indicates a possible attack. It provides the possibility for users to configure a new action to be performed for a certain period of time, when a given event flow rate is exceeded [85].

market, being the OSSIM, ELK stack, Splunk Free and Graylog. The analysis was carried out based on test scenarios that made it possible to identify the functionalities and the configuration complexity of each of the solutions. The authors concluded that based on the results obtained in the tests, OSSIM and Splunk Free would not be scalable and presented some deficiencies in providing the necessary documentation for their configuration. In addition, they identified that OSSIM is the most limited solution in terms of searching for information, that is, the user is unable to analyze all the content of the collected logs. The configuration of email alerting proved to be more complex in OSSIM, largely due to the lack of specific documentation. The ELK stack is the only solution that only allows sending email only by configuring external tools. One of the main points of this project was the development of a solution in accordance with the new General Data Protection Regulation (GDPR). At this point, the authors concluded that the ELK stack is the only one that does not need authentication to access the web interface, but it already has the functionality of data pseudo-anonymization. The remaining solutions involve authentication when accessing the web interface. Graylog even has the functionality of assigning permission levels to different users.

Chapter 3

Scenario Overview

Before proceeding with the implementation, it is essential to understand the environment in which the solution will be implemented, as well as all the resources needed for its development. This chapter aims to present such criteria.

3.1 Implementation Design

The whole ideia of this project is to provide a tool that combined with the existing security tools, help to protect the INESC TEC network. That said, the solution was designed so that network administrators have a way to assess the danger and types of attacks that reach our network through the internet, but above all, being able to detect and analyze malicious activities that may be occurring within the internal network. Figure 3.1 represents the implementation design.

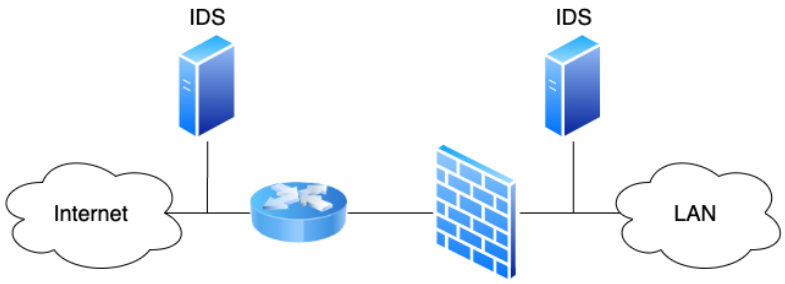

Figure 3.1: IDS implementation design

3.2 Available resources

Considering that the choice of the IDS system falls on Suricata and given the fact that this is a tool developed on a multi-threaded architecture, it is essential to have a machine capable of guaranteeing high levels of processing in order to extract maximum utility of the tool. Along the research, it was concluded that it would not make sense to implement Suricata in a virtualized environment due to the fact that several authors refer that the number of discarded packets is significantly higher, largely due to the virtual allocation of resources. In addition, the IDS solution will not be complete without the implementation of the ELK stack, which will centralize information from Suricata.

For Suricata, the available server is a HP Proliant DL380. This machine has two 2 Intel(R) Xeon(R) CPU E5520 @ 2.27GHz each with 4 cores and 8 threads, 23 GB of RAM and eight 146.8 GB disks configured in RAID 50 making up a total of 780 GB disk space. Despite the server age, the machine's specifications make it suitable for this project while also allowing for the reuse of equipment. It has four Broadcom Inc. NetXtreme II BCM5709 Gigabit Ethernet network cards in which three of them will be used to receive copies of the real traffic through port mirroring. In addition, it has an HP Integrated Lights-Out (iLO) interface that consists of a dedicated out-of-band management [1] interface.

The Elastic Stack was implemented on a Virtual Machine running Ubuntu 18.04 LTS. This machine must have enough resources to process the large amount of information coming from Suricata. A virtual machine was created with 6 vCPUs and 20 GB of RAM with the possibility of increasing it if necessary. Regarding disk space, the value is dependent of the amount of logs and retention periods. A base requirement is set to at least 500GB which is reasonable value to start with.

[1] Management traffic is independent from the main traffic

Chapter 4

Implementation

This chapter aims to cover the implementation and configuration of the IDS solution.

4.1 IDS implementation

IDS implementation was divided in three main phases, each of them described below.

4.1.1 First Phase - SELKS implementation

This is a complete open source and out of the box solution that combines network monitoring tools that work together to provide a complete network monitoring environment, mentioned in the previous section 2.9. It is based on Debian and developed by StamusNetworks [75]. The ISO image is available through their GitHub repository [76], which has a well organized amount of documentation. The latest release dates back to 2017, proving that is not actively developed and making it a less than ideal solution for production environments. At this moment, only the `enp2s0f1` interface was configured as it was intended to take a minimal configuration approach with this solution. Being an out of the box solution, it could provide some insights without major complications in terms of configuration. SELKS documentation is structured as a step by step guide, being quite simple to understand. Furthermore, the developers provide some tips and considerations about the tuning needed to optimize the solution. The user can get the solution up and running with the execution of a few scripts, with Scirius to provide a web based rule management platform and Kibana with predefined indexes and dashboards to visualize Suricata data.

4.1.2 Second Phase - Suricata implementation

Second phase consisted on building the final solution. Since the project concept is based on open source software, Ubuntu Server 18.04 was the chosen operating system. As it is intended to implement Suricata in passive mode, port mirroring was used in order to distribute a copy of the actual traffic all the way to Suricata. Interface `enp2s0f1` remained unchanged while interfaces `enp3s0f0` and `enp3s0f1` were configured to receive internal traffic. One of the main beneficts of Suricata is its extensive, intuitive and complete documentation. Following the official documentation [87], installation is quite simple through the dedicated PPA repository which in turn installs the most recent stable version, Suricata 6.0.2 released in March 2, 2021.

Address groups

Suricata needs some information about the network environment. It highly recommended to specify which internal networks exist in the environment through `$HOME_NET` variable. By default, `$HOME_NET` is set to three different network ranges being 192.168.0.0/16, 10.0.0.0/8 and 172.16.0.0/12[1]. It is recommended to specify in detail each set of networks that make up the local network in order to improve the accuracy and performance of Suricata. `$EXTERNAL_NET` variable must be set to `!$HOME_NET`. This means that every network that is not set in `$HOME_NET` will be treated as an external network. Suricata rules make use of these variables to alert on internal or external threats.

Packet capture

Suricata relies on the capture method to distribute the packets over the existing threads. It can be configured with an external high speed packet capture library called `PF_RING`. This method was discarded due to incompatibility with the NICs used on this server. That said, `AF_PACKET` was used. `AF_PACKET` is the default packet capture method in Linux operating systems. Firstly, we need to specify the monitoring interfaces and the number of processing threads reserved for each of them. Also, `AF_PACKET` defaults the `cluster_type` value to `cluster_flow` which means that the capture method is instructed

[1]Private address blocks reserved by Internet Assigned Numbers Authority (IANA) [88]

to hash by 5 tuple [2]. Suricata documentation recommends the use of `cluster_flow` on the majority of setups as `cluster_cpu` and `cluster_qm` are recommended on more powerful systems with dedicated capture cards.

Rule management

Users can download and install the rules manually but the developers recommend using a management tool to do so. For versions 4.1 and later, `suricata-update` tool is bundled in the Suricata package and it will be installed by default. This is the official tool for downloading and managing Suricata rules. After installing Suricata, users can simply execute `suricata-update` to download the Emerging Threats (ET) Open ruleset, which is the open source set of rules provided by ET.

It is recommended to run `suricata-update` regularly because rulesets are constantly updated. By default, `suricata-update` will only download the ET Open ruleset although, the user can enable other sources by using `suricata-update list-sources` and `suricata-update enable-source [ruleset name]`. In this setup, only the ET Open ruleset was used.

An important consideration is that, by default, `suricata-update` will merge all rules into a single file (suricata.rules), located in `var/lib/suricata/rules` directory. By default, Suricata uses the this file to indicate which rules are being used for intrusion detection purposes. But, combining this large number of rules in a single file revealed not be the best approach as it makes rule organization process a rather complex task. For that reason, there is the option to run `suricata-update` along with the `--no-merge` flag. This option allows the download of the original and unmerged ET Open ruleset files into `var/lib/suricata/rules`. Now, as rules are unmerged into individual files it will be necessary to define which rules to use in the Suricata configuration file (see listing B.1). This revealed to be a more organized rule management method when faced with the large number of rules that compose the ruleset.

[2]Refers to five different values that compose a TCP/IP connection, being source/destination IP, source/destination ports and protocol

Log outputs

Suricata outputs logs into two main log outputs, being `fast.log` and `eve.json`. Alerts and protocol specific records are logged in Extensible Event Format (known as EVE) into `eve.json` file. Suricata allows that multiple `eve.json` files in order to divide the different type of event into separate files. As for `fast.log`, it exclusively outputs Suricata alert events and cannot be divided into separate files. Also, every log file is rotated every day at midnight. This rotation creates a new log file, saving the old one with a date extension and logs older than seven days are deleted in order to save disk space. Table 4.1 represents Suricata configured outputs. Two `eve.json` outputs were created in order to separate alert events from other type of events that are less relevant. `fast.log` was kept activated but the information it provides is quite scarce compared to the one provided by the `eve-alerts.json` file (see listings C.1 and D.1) respectively.

File	Purpose
fast.log	outputs exclusively the alerts
eve-alerts.json	only outputs alerts in `json` format
eve-events.json	only outputs protocol specific events in `json` format

Table 4.1: Suricata configured log outputs

Running modes

Suricata can have multiple running modes. By default, the runmode option is disabled in Suricata configuration file. If enabled, Suricata defaults to `autofp`. This mode ensures that capture threads and packet processing threads are separated. This is, traffic is captured by one or more packet capture threads which then passes the packets to multiple packet processing threads.

4.1.3 Third phase - Elastic Stack implementation

The Elastic Stack consists of a set of tools that handle, process and store data from multiple sources. This is one of the best data storage and handling solutions and offers great flexibility and scalability. Furthermore, it has a very active community with forums and extensive supporting documentation. After some research it was found that the implementation of Logstash would not be necessary. Recently, Filebeat has a specific module that processes and parses logs that are in the Suricata Eve `JSON` format, without the need for Logstash pipelines. Note that this module only works with Suricata v4.0.4 or above. Figure 4.1 presents a schematic of the stack structure that was implemented in this project.

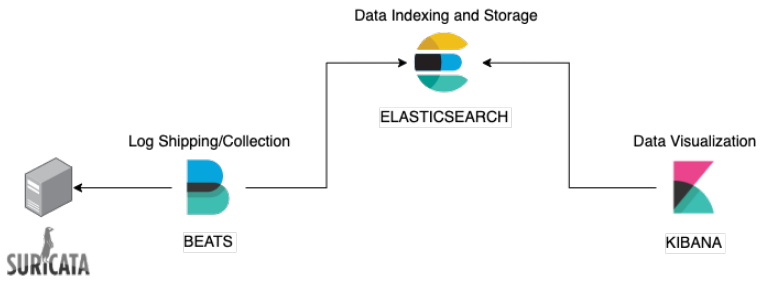

Figure 4.1: Elastic Stack implementation

In this phase, only the implementation of a single node Elastic Stack was considered. It is important to bear in mind that the stack must be uniform regarding the versions installed, which means that all tools across the stack must be running the same versions which is v7.13.1 in this setup. Field `network.host` must be uncommented and changed to the server IP address or hostname. Also, being a single-node implementation, it is necessary to specify `discovery.type:single-node`. This setting provides the capability for a node to elect itself master preventing it from joining a cluster with any other node. Listing E.1 indicates that the Elasticsearch node is up and running.

Like Elasticsearch, Kibana is also only accessible on localhost and through port 5601. Therefore, it is necessary to uncomment `server.host` on `kibana.yml` configuration file and change it to the server IP address or hostname. Furthermore, Kibana needs to be aware of existing Elasticsearch instances. That is why `elasticsearch.host` must be pointed to the Elasticsearch instance address.

By default, Elasticsearch and Kibana are not password protected. To improve security, Elastic Stack can be secured in three different layers:

- Minimal Security - prevents unauthorized access to local cluster by setting passwords to built-in users.
- Basic Security - adds TLS for protecting internal communication between nodes and prevents unauthorized nodes to join the cluster. External HTTP traffic between Elasticsearch and Kibana is not encrypted
- Basic Security with HTTPS - configures TLS for external HTTP traffic between Elasticsearch and Kibana

Being a single-node setup, only the minimal security layer can be implemented. First, `xpack.security.enabled:true` must be added into `elasticsearch.yml` configuration file in order to enable Elasticsearch´s security features. Then, to communicate with the cluster, the built-in users passwords must be generated. This built-in users have a fixed set of privileges and cannot be authenticated until their passwords have been created. The users are:

- `elastic` - superuser
- `kibana_system` - for Kibana to connect and communicate with Elasticsearch
- `logstash_system` - for storing logstash data in Elasticsearch
- `beats_system` - for storing beats data in Elasticsearch
- `apm_system` - for storing APM Server data in Elasticsearch
- `remote_monitoring_user` - for storing and collecting Metricbeat data in Elasticsearch

Elasticsearch documentation [89] indicates that "Unless you enable anonymous access, all requests that don't include a user name and password are rejected". Running `elasticsearch-setup-passwords interactive` sets up the built-in users passwords. For now, this user will be used by Kibana and Filebeat to store their data in Elasticsearch. The `kibana_system` user credentials must be added into the *kibana.yml* configuration file (see listing 4.1). To prevent writing the password in plain text, a keystore must be used. This user does not have permissions to log into the Kibana web page. Users must use the `elastic` built-in user to log into Kibana for the first time after enabling the security features. It is not recommended to use `elastic` superuser unless full access to the cluster is required. As such, it is recommended to create additional users with specific privileges.

```
# If your Elasticsearch is protected with basic
   authentication, these settings provide
# the username and password that the Kibana server uses to
   perform maintenance on the Kibana
# index at startup. Your Kibana users still need to
   authenticate with Elasticsearch, which
# is proxied through the Kibana server.
elasticsearch.username: "kibana_system"
elasticsearch.password: "pass"
```

Listing 4.1: `kibana_system` user configuration

Filebeat configuration

Suricata logs need to be fetched, treated and sent to the stack for indexing and visualization. Filebeat consists of a log shipper, installed as an agent on the monitoring server, that can collect and parse multiple log data through various modules that simplify the parsing and visualization of common log formats. Filebeat provides a Suricata module that can be enabled in order to parse Suricata `JSON` log files. Module configuration is located in `/etc/filebeat/modules.d/suricata.yml` and the configuration is shown in listing F.1.

Before starting Filebeat, `filebeat setup` command was used to load the recommended index template as well as some predefined dashboards into Kibana. Note that this step does not load the ingest pipelines[3] used to parse log lines as their are automatically loaded, by default, the first time we run the module and connect Filebeat to Elasticsearch.

Also, Filebeat indices are named filebeat-7.13.3-yyyy.MM.dd, where yyyy.MM.dd is the date when the events were indexed. This name was changed to Suricata in order to keep things more organized and easier to identify. To accomplish that, it was necessary to explicity define the new index name on Filebeat configuration file. The new index is created with the name suricata-ids-yyyy.MM.dd (see listing G.1. Note that after Filebeat version 7.0, Index Lifecycle Management (ILM) is enabled by default. ILM is a feature that manages and automates Filebeat indices rollover as they age or reach a certain size. To change the default index name, ILM must be disabled or we must explicity define the ILM rollover alias and ILM pattern.

Kibana custom dashboards

Despite predefined dashboards provided by the Suricata Filebeat module, these are not so useful for this setup. Therefore, it was necessary to create more personalized dashboards. Also, this was made in order to satisfy one of the main objectives of this project, providing a customized data visualization interface.

[3]It consists of a set of processes that perform specific data transformations before they are indexed into a data stream or an index

Suricata monitors both internal and external traffic. Typically, external traffic generates a higher amount of false alerts and traffic noise than internal traffic but, on the other hand, internal traffic is more relevant in terms of analysis. That said, it makes perfect sense that these visualizations and dashboards are distinguished. The dashboards structure is shown in 4.2.

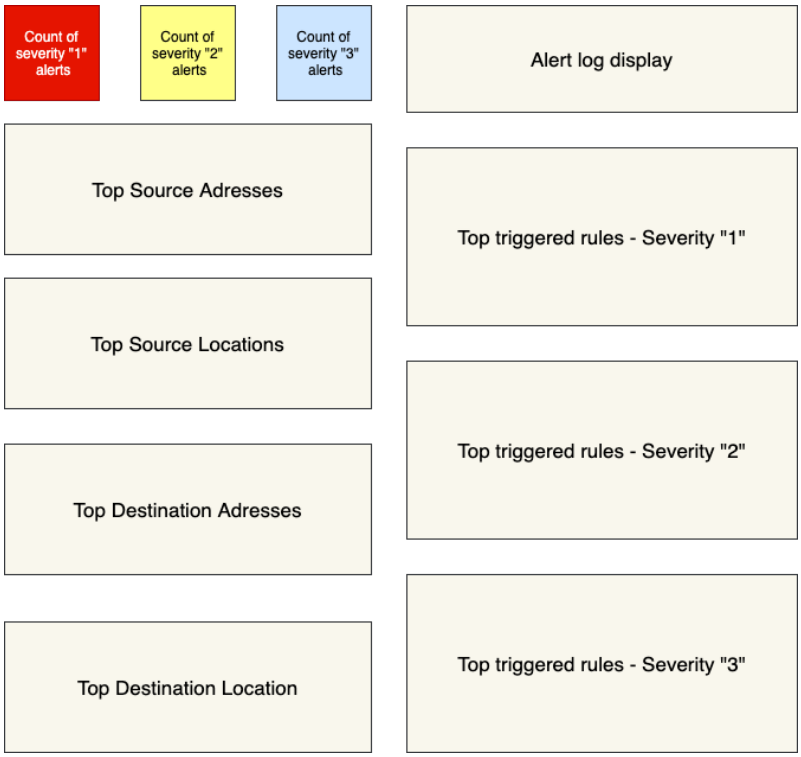

Figure 4.2: Kibana dashboards structure

Chapter 5

Tests and Results

5.1 Tuning Suricata

Although these are the required steps to run Suricata, performance otimization is essencial to get the best out of the packet detection. During a one hour run with Suricata default configuration, packet drop percentage approached 4% as seen in table 5.1.

Total Packets	Total Drops	Average CPU Usage	Average Memory Usage
197251789	7579873 (3.84%)	20%	9%

Table 5.1: `AF_PACKET` configuration

For that reason, Suricata documentation provides some specific tuning considerations [90]. Most of these optimizations aim to improve hardware performance but also to reduce the packet drop percentages. Ideally, the number of dropped packets must be below 1% and we must ensure that there is a compromise between resource use and performance optimization.

5.1.1 Running mode

Suricata runmode was changed to `workers`. Generally, this mode unsures a better efficiency as it is optimized for higher performance. This mode ensures that the packets are balanced over the existing processing threads that contain the full packet pipeline. Workers mode has the advantage of keeping the full packet processing in a single thread. Tests revealed that `workers` mode performs much better than `autofp`, as will be analyzed further on table 5.6.

5.1.2 AF_PACKET

Some AF_PACKET optimizations were made in order to increase Suricata´s performance and reduce the packet drop count. This turned out not to be a straightfoward task as every setup operate differently. Table 5.2 presents the AF_PACKET configuration.

Field	Value	Purpose
interface	enp2s0f1/enp3s0f0/enp3s0f1	Listening interfaces
threads	auto	number of receive threads per interface
cluster-type	cluster_flow	Packets of a given flow are sent to the same socket
defrag	yes	Defragmentation done by Suricata
tpacket-v3	yes	Recommended for IDS deployments
use-mmap	yes	Needed to use ring-size feature
mmap-locked	yes	Avoids memory from being swapped
ring-size	100 000	Packet buffer size per thread
block-size	32768	Packet buffer size per thread

Table 5.2: AF_PACKET configuration

Suricata server has 16 available threads. With threads:auto setup, Suricata is capable of evenly distributing traffic through the available hardware threads. That said, this enables 48 processing threads as seen in listing H.1.

A ring buffer[1] is composed of a number of blocks. These blocks are connected regions of physical memory and Eeach block is divided into a number of frames. Ring-size and Block-size values make up the AF_PACKET ring buffer configuration. Ring-size corresponds to the number of packets allowed in the buffer. It means that there will be a limit of 100000 packets in each buffer of the available threads and increasing this value will have an impact on memory usage. After some tests, this revealed to be a reasonable value to start with.

[1]Data structure that stores incoming packets until the device driver can process them.

`Block-size` corresponds to the size in bytes of each block and it should be a power of 2 and a multiple of 4096. By default, this value is set to 32768 and remains unchanged.

5.1.3 Rules

`Suricata-update` generates four configuration files represented in table 5.3. This files are used to customize Suricata rules.

File	Purpose
`enable.conf`	enable rules
`disable.conf`	disable rules
`modify.conf`	change specific parameters of a rule
`drop.conf`	convert rules to drop

Table 5.3: Rule management configuration files

The rule matching process of these files is done by signature ID, regular expression or group of rules. It is important to manage the rules used according to the needs of each setup. Managing such a large amount of rules provided by the ET Open ruleset requires intensive analysis work. This analysis would be too time-consuming and would not be feasible for a short period of time.

Within each ruleset, there are rules that are commented and therefore not enabled. The time taken to complete this project will not allow for intensive rule-by-rule analysis of each ruleset. Listing I.1 represents `enable.conf` configuration file which includes all the rulesets configured in Suricata configuration file. Note that if we decide to enable an entire ruleset all the rules inside that ruleset are automatically enabled. Same happens when we insert a group of rules into `disable.conf`, instead all the rules that compose that ruleset are disabled. These two files are automatically loaded and processed when running `suricata-update`.

Currently these files are processed in order, therefore `disable.conf` is processed before the `enable.conf`. Thus, when the enable file is processed, it can revert what was previously configured in the disable file. This is a known limitation that the developers are aware off [91].

5.1.4 Max-pending-packets

This option controls the number of simultaneous packets that Suricata can handle. Documentation suggests that this value should be between 10000 and 65000. A higher value results in higher CPU and memory usage. However, if this value is too high it can lead to malfunction. In this setup, this value was set at 65000 in order to maximize the amount of packets analyzed by Suricata.

5.1.5 Network interfaces

It is known that packet handling is optimized through the network cards offloading techniques. Suricata documentation recommends disabling all of these techniques, as they lead to break TCP state tracking [92]. Despite that, checksum offloading can be left enabled when using `AF_PACKET`. Listing J.1 shows the `ethtool` offloading configurations that were replicated through all interfaces. Also, some network cards use a technique called Receive Side Scaling (RSS). This is meant to improve performance as traffic is distributed through multiple queues. This technique uses an asymmetric hash to distribute traffic, which leads to both sides of a flow ending up on a different queue causing unpredictable packet processing. None of the supported packet capture methods resolve this problem, but one workaround is to reduce the number of RSS queues to a single queue and preventing traffic from travel between different queues. Despite that, these optimizations were not enough to solve this problem. Running in `workers` mode, a large amount of packets is seen on the wrong thread (see listing 5.1).

```
capture.kernel_packets      | Total      | 51073718
tcp.pkt_on_wrong_thread     | Total      | 8594085
```

Listing 5.1: Packet seen on wrong thread

This is a known issue with fragmented packets. If there is a mix of fragmented and unfragmented TCP/IP packets in the same flow they will get different hashes and most likely end up in different queues. This can be resolved by forcing sd hash[2] on the Network Interface Controller (NIC) via ethtool. However, Broadcom NICs installed on this server do not support this change. `Autofp` mode can mitigate this problem because it only uses a single stream tracker to process all TCP packets, regardless of whether they are fragmented or not. On the other hand packet drop counts using `autofp` are considerably higher.

5.1.6 Detection engine

Suricata uses a prefilter approach to evaluate which rules need to be analysed. Only one condition from each rule is added to the prefilter engine. By default, Multi-Pattern-Matcher (MPM) is the prefilter algorithm used by Suricata detection engine (see listing 5.2). MPM is an algorithm that helps Suricata exclude some rules from being examined. Each rule can have one or more patterns that are used by MPM algorithm. Only rules that have matching patterns are individually analysed. This algorithm performs in three simple steps [93]:

1. receives packet.
2. MPM searches for patterns.
3. Only matched rules are analysed.

[2]Hash on source and source and destination IP addresses

```
  prefilter:
    # default prefiltering setting. "mpm" only creates MPM/
       fast_pattern
    # engines. "auto" also sets up prefilter engines for
       other keywords.
    # Use --list-keywords=all to see which keywords support
       prefiltering.
    default: mpm
```

Listing 5.2: Prefilter Engine configuration

The `mpm-algo` variable indicates which algorithm is used for MPM. On commodity hardware, Suricata documentation suggests configuring `mpm-algo: ac-ks`, which is `Aho-Corasick Ken Steele variant`. Also, when using `mpm-algo: ac-ks`, it is recommended to set `detect.sgh-mpm-context` to `full`. Suricata documentation refers that setting it to `full` means that every rule group has its own `mpm-context` (see listing 5.3).

```
detect:
  profile: high
  custom-values:
    toclient-groups: 100
    toserver-groups: 100
  sgh-mpm-context: full
```

Listing 5.3: Detect settings

Suricata detection engine separates rules by groups but not every type of rule gets its own group. Instead, rules with similiar properties are placed in the same group. This method can prevent packets from being inspected against rules that they don´t match. The variable `detect.profile` controls the amount of rule groups that are created. Normally, setting this value to `high` (see listing 5.3) provides better rule matching efficiency.

5.1.7 Host OS

Fragmented packets are processed differently between operating systems. Each operating system has its policies regarding defragmentation and stream reassembly. It is essencial for Suricata to know in operating system is implemented in order to how to process packets. That said, `host-os-policy` needs to be set (see listing 5.4).

```
host-os-policy:
  # Make the default policy windows.
  windows: []
  bsd: []
  bsd-right: []
  old-linux: []
  linux: [x.x.x.x] # Suricata IP address
  old-solaris: []
  solaris: []
  hpux10: []
  hpux11: []
  irix: []
  macos: []
  vista: []
  windows2k3: []
```

Listing 5.4: Host-os-policy setting

5.1.8 Memcap values

At last, it was necessary to optimize the `memcap` values for defrag-engine, stream-engine and flow-engine. This value represents the maximum amount of memory these engines can use.

Flow-engine keeps track of the traffic flows. Packets with the same protocols, source and destination IPs, source and destination ports belong to the same flow and are connected internally. Flow allocation uses memory so it is necessary to determine which value should be used in order to otimize Suricata performance.

`Prealloc` variable has a specific importance within the flow-engine. It determines the number of flows that Suricata keeps prepared in memory. Suricata keeps these flows in memory in order to protect itself against flood attacks. This is because for packages that do not yet belong to a certain flow, Suricata creates a new flow. Knowing this, an attacker can generate a lot of packets with different tuples in order to flood the flow-engine, which needs to create a lot of new flows. In order to mitigate these possible attacks, Suricata maintains a predetermined number of flows in memory. This value is set to 10000 by default and has not been changed. The `emergency-recovery` value determines the percentage of prellocated flows in which the flow-engine returns to normal mode. By default, the value is set to 30% and has not been changed. Without optimizations, Suricata reserves 128MB to the flow-engine as seen in listing 5.5

```
flow:
  memcap: 128mb
  hash-size: 65536
  prealloc: 10000
  emergency-recovery: 30
  #managers: 1 # default to one flow manager
  #recyclers: 1 # default to one flow recycler thread
```

Listing 5.5: Flow-engine default configuration

Tests revealed that this value was not enough as the flow-engine ran out of memory. When `memcap` value is reached, the flow-engine enters emergency mode. This mode ensures that the engine will use shorter timeouts in order to create more space for new flows. As a result, between 9pm and 11:59pm the flow-engine entered emergency mode 28 times using the default `memcap` settings.

This can be somehow be explained by some traffic spikes that sometimes may occur in the network. During these tests, it was determined that the engine started entering emergency mode around the same time the interface `enp2s0f1` had some traffic spikes that lead to a large amount of packets being sent to Suricata, as it can be seen in figure 5.1.

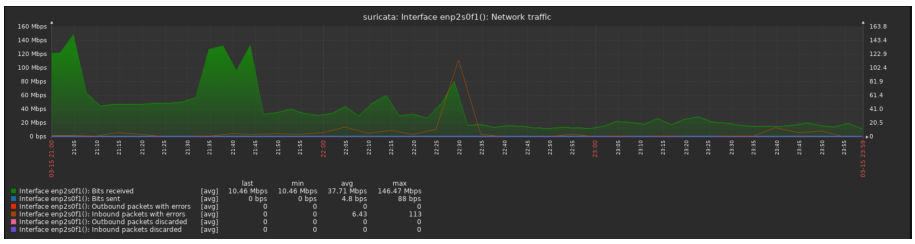

Figure 5.1: Traffic spike interface `enp2s0f1`

As we can see, a traffic spike occured shortly after after 9pm. Listing 5.6 shows that at 9:01pm the engine entered for the first time into emergency mode.

```
Date: 3/15/2021 -- 21:01:08
-------------------------------------------
Counter                          | TM Name       | Value
-------------------------------------------
flow.emerg_mode_entered          | Total         | 1
flow.emerg_mode_over             | Total         | 0
```

Listing 5.6: Flow-engine first emergency mode entry

By the time the first spike ended, which was around 9:11pm, the flow-engine had entered emergency mode five times (see listing 5.7).

```
Date: 3/15/2021 -- 21:11:00
-------------------------------------------
Counter                          | TM Name        | Value
-------------------------------------------
flow.emerg_mode_entered          | Total          | 5
flow.emerg_mode_over             | Total          | 5
```

Listing 5.7: Flow-engine emergency mode after first traffic spike

This value remained constant until the next traffic spike, which started around 9:30pm. At around 9:32pm, the flow-engine had entered emergency mode seven times, as we can see in listing 5.8.

```
Date: 3/15/2021 -- 21:32:05
-------------------------------------------
Counter                          | TM Name        | Value
-------------------------------------------
flow.emerg_mode_entered          | Total          | 7
flow.emerg_mode_over             | Total          | 7
```

Listing 5.8: Flow-engine emergency mode before second traffic spike

After the second and third traffic spike ended, flow-engine had entered emergency mode 15 times (see listing 5.9).

```
Date: 3/15/2021 -- 22:35:01
-------------------------------------------
Counter                              | TM Name       | Value
-------------------------------------------
flow.emerg_mode_entered              | Total         | 15
flow.emerg_mode_over                 | Total         | 15
```

Listing 5.9: Flow-engine emergency mode after second and third traffic spikes

Until 11:59pm, traffic remained somewhat stable although the flow-engine entered emergency mode another 13 times, making a total of 28 emergency mode entries during the analysed time span (see listing 5.10.

```
Date: 3/15/2021 -- 23:59:02
-------------------------------------------
Counter                              | TM Name       | Value
-------------------------------------------
flow.emerg_mode_entered              | Total         | 28
flow.emerg_mode_over                 | Total         | 28
```

Listing 5.10: Flow-engine emergency mode total entries

To avoid the flow-engine entering emergency mode, it was necessary to set the flow `memcap` to a higher value. This value was set to 1GB (see listing 5.11). This value revealed to be enough to avoid the engine entering emergency mode as seen in listing 5.12.

```
flow:
  memcap: 1gb
  hash-size: 65536
  prealloc: 10000
  emergency-recovery: 30
  #managers: 1 # default to one flow manager
  #recyclers: 1 # default to one flow recycler thread
```

Listing 5.11: Flow-engine optmized settings

```
Date: 6/22/2021 -- 21:45:40 (uptime: 0d, 01h 02m 40s)
flow.emerg_mode_entered              | Total          | 0
flow.emerg_mode_over                 | Total          | 0
```

Listing 5.12: Flow-engine emergency mode with optimized flow-engine settings

The stream-engine is divided into two main functions: the stream tracking and stream reassembly. Stream-tracking-engine is responsible for monitoring the TCP connections. As for the stream-reassembly-engine it is responsible to reconstruct the flows to their original state. These values are, by default, set to 64MB for the `stream-tracking-memcap` and 256MB for `stream-reassembly-memcap` (see listing 5.13).

```
stream:
  memcap: 64mb
  checksum-validation: yes      # reject incorrect csums
  inline: auto                  # auto will use inline mode
     in IPS mode, yes or no set it statically
  reassembly:
    memcap: 256mb
```

Listing 5.13: Stream-engine default configuration

By default, the stream-engine is not processing packets with wrong TCP checksums. This can be changed by disabling `stream-checksum_validation`. After running a one hour test without `memcap` optimization, stream-engine ran out of memory and started to drop some packets (see listing 5.14). This value tends to grow depending on the amount of traffic that reaches Suricata.

```
Date: 6/22/2021 -- 19:11:51 (uptime: 0d, 01h 19m 16s)
-------------------------------------------------
Counter                                 | TM Name         | Value
-------------------------------------------------
tcp.segment_memcap_drop                 | Total           | 23129
```

Listing 5.14: Stream-engine drops

Optimizing the stream-engine `memcap` values, changing both to 1GB, revealed to be an improvement to the engine performance (see listing 5.15). After running another test for a complete hour, we can observe that there were no drops.

```
Date: 6/22/2021 -- 20:36:44 (uptime: 0d, 01h 14m 57s)
-------------------------------------------------
Counter                                 | TM Name        | Value
-------------------------------------------------
tcp.segment_memcap_drop                 | Total          | 0
```

Listing 5.15: Stream-engine with no drops

Lastly, the defrag-engine is responsible for the reconstruction of fragmented packets. Networks can have many fragmented packets that need to be reconstructed before Suricata is able to inspect them. This engine keeps fragmented packets in memory until the other fragments appear in order to reconstruct the whole packet. This value was also set to 1GB in accordance with previously configured values, although the concrete impact of this optimization was not detected (see listing 5.16).

```
# Defrag settings:
defrag:
  memcap: 1gb
  hash-size: 65536
  trackers: 65535 # number of defragmented flows to follow
  max-frags: 65535 # number of fragments to keep (higher than
      trackers)
  prealloc: yes
  timeout: 60
```

Listing 5.16: Defrag-engine optimized

5.2 Tuning Elastic Stack

By default, Elasticsearch sets the Java Virtual Machine (JVM) heap size based on the available memory. Also, Elasticsearch memory lock needs to be enabled in order to prevent Elasticsearch from swapping its memory.

To improve security, it was decided to encrypt traffic between Kibana and the web browser. It was necessary to generate a certificate and a private key for the server and Kibana was configured to enable TLS for inbound connections (see listing K.1).

Filebeat needs to be able to connect to both Kibana and Elasticsearch in order to load predefined assets for parsing and visualizing data. So, if Kibana is configured with TLS, it is also necessary to configure TLS in Filebeat for having communication between Filebeat and Kibana. Certificates were generated and added to the `filebeat.yml` configuration file in the section corresponding to the Kibana output (see listing L.1).

5.3 Impact Tests

This section aims to assess the hardware impact of the implemented tools. These tests were carried out after the implementations and optimizations explained in the previous sections.

5.3.1 Suricata resource usage

`Workers` mode is optimized for higher performance and the flow balancing happens outside Suricata. Also, there is no need for packet capture threads as the NIC makes sure packets are balanced over multiple processing threads that contain the full packet pipeline. In `autofp` mode there are multiple capture threads that capture the packet and do the decoding. Then, the packets are passed to the packet processing threads and the flow balancing happens inside Suricata. Also, documentation only suggests the use of this runmode in cases of Packet capture (PCAP) files processing or certain IPS setups. Table 5.5 represents the average resource usage in both modes.

Mode	Average CPU usage	Average memory usage
Workers	21%	78%
Autofp	85%	91%

Table 5.4: Running modes resource usage

5.3.2 Elastic Stack

The Elastic stack has remained stable since its implementation without the need for major performance optimizations. CPU usage and memory usage were averaging about 17% and 82% respectively.

ILM and retention policies were set to roll over when the index is 30 days old or any primary shard reaches 50 gigabytes.

5.3.3 Traffic Rates

Network traffic rates can have a major impact on Suricata's performance. Suricata should be optimized taking into account the amount of network traffic in each setup. It is important to take into consideration that traffic spikes and elephant flows[3] may have a negative impact on Suricata's performance, causing an unwanted increase in dropped packets. These values represent the traffic observed on each of the interfaces, during a one week span. It is also possible to see some traffic spikes during office hours.

Interface	Average traffic rate
enp2s0f1	30.0 Mbps
enp3s0f0	52.6 Mbps
enp3s0f1	292.0 Mbps

Table 5.5: Traffic rates

[3]Network session that consumes a disproportionate amount of bandwidth

5.4 Optimization results

This section presents some results obtained from all the optimizations that were done in order to improve Suricata's performance.

5.4.1 `Workers` vs `Autofp`

Since these experiments were being conducted in live traffic and the fact that there were no consistent configurations for each setup, it was difficult to extract detailed information about all the conducted tests. The testing methodology was based on a trial-and-error methodology until an ideal configuration was reached. After a one hour long running test, it is possible to conclude that `workers` mode has much better performance than `autofp` mode. As seen in the table 5.6, CPU usage and dropped packets percentage in `autofp` mode are substantially higher than those observed in `workers` mode.

Runmode	Total packets	Total drops (%)	Average CPU Usage	Average Memory Usage
Workers	184621837	86536 (0.04%)	20%	78%
Autofp	186722423	76196032 (40%)	85%	91%

Table 5.6: `Autofp` vs `Workers` stats

After the results obtained with `autofp` mode, together with the fact that is usually not recommended for this type of implementation, it was decided not to give further consideration to this mode and to focus on workers mode. Although it was not possible to reduce the dropped packets percentage to 0%, the percentage is constantly below 1% which is already an acceptable value according to the documentation, as seen in table 5.7.

Time	Total captured packets	Total dropped packets (%)
12 hours	2494617501	12784041 (0.51%)
48 hours	8791767730	31913107 (0.36%)

Table 5.7: `Workers` mode stats

5.4.2 Alert otimization

The biggest challenge in implementing this type of solutions is the detection of false positives. In a network environment as complex as INESC TEC´s it is predictable that there will be too much noise and it is essential that the tool is as tuned as possible. In addition, each network environment is different and the tool must always be optimized for the existing traffic. Despite this, it was possible to obtain some interesting results regarding false positives, that were suppressed in Suricata's threshold configuration file:

- `GPL SQL probe response overflow attempt` - Alert triggered during Zoom conference calls. The vast majority of alerts have a public source address belonging to Zoom aswell as source port 881/udp which is used in Zoom meetings.
- `ET MALWARE Possible Downadup/CONFICKER-C P2P encrypted traffic UDP Ping Packet` - This alert was triggered by IPsec NAT traversal traffic.
- `ET EXPLOIT Possible CVE-2020-11899 Multicast out-of-bound read` - Triggered from Link-Local Multicast and DNS multicast IPv6 traffic.
- `ET WEB_SERVER Possible CVE-2014-6271 Attempt` - Triggered from a NESSUS system imlemented on INESC TEC network.

It was also possible to obtain results on more worrisome alerts. The figure 5.2 represents an alert triggered in the presence of an exploit that seeks to exploit a critical Remote Code Execution (RCE) vulnerability in a Joomla website.

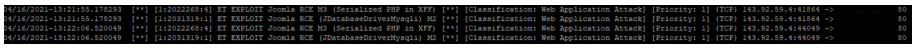

Figure 5.2: Joomla exploit

In the figure 5.3 we can see several network scans on port 3306 which is used by MySQL.

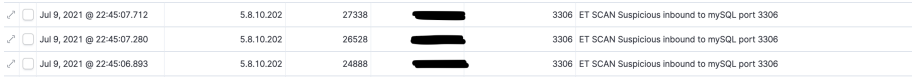

Figure 5.3: MySQL scan

The figure 5.4 represents an attempted brute force login to MySQL performed after the previous scans. This machine, despite being behind the firewall, has port 3306 open to the internet. However, the login attempts were unsuccessful, as it can be seen in figure 5.5.

Figure 5.4: MySQL brute force login attempt

[redacted]	3306	5.8.10.202	33068	ET SCAN Multiple MySQL Login Failures Possible Brute Force Attempt
[redacted]	3306	5.8.10.202	29610	ET SCAN Multiple MySQL Login Failures Possible Brute Force Attempt

Figure 5.5: MySQL brute force login failure

Finally, figure 5.6 represents mining activity from a machine inside INESC TEC local network. This alert was obtained during the first implementation described in the 4.1.1 section.

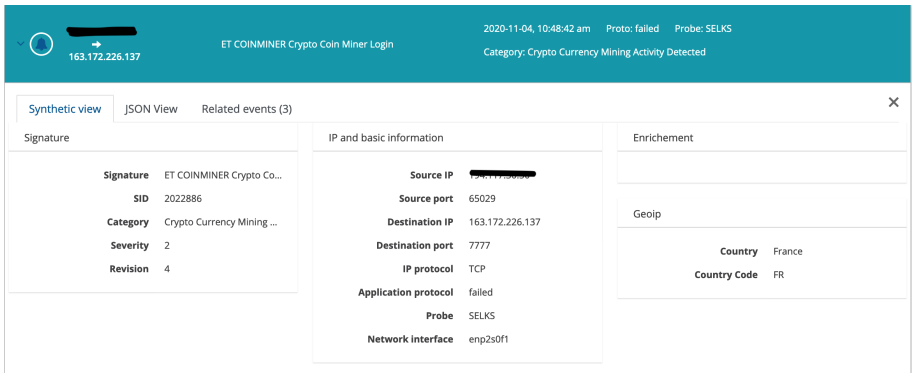

Figure 5.6: Crypto mining activity

It is important to notice that `suricata-update` reads rule management files in a specific order. Currently, `disable.conf` is read before the `enable.conf` which can revert any change that was made in the `disable.conf` file. For now, if we need to disable a specific rule from a rule group that is enabled, we need to add that specific rule in the `Threshold.conf` file. In this file it is possible to supress or threshold specific rules (see listing M.1. This listing represents `Threshold.conf` file of this setup, which is supressing a set of individual rules that were triggering a high number of false positives. Rules can be suppressed completely or can be specifically suppressed for specific source or destination addresses. Listing N.1 shows that Suricata is using a total of 21586 rules.

Although customized data visualization and dashboards have been created in Kibana it was also thought that it would be important to implement a method to filter the most critical alerts and send them to an external platform such as email. For this purpose, ElastAlert2 was implemented.

ElastAlert2 in an automated rule-based alerting tool that periodically queries Elasticsearch and then matches patterns from pre-configured rule-types. Then, if there are any matches, ElastAlert will send an alert message based on the multiple alert types. The available rule-types are defined as follow:

- Frequency - match when X events occur in Y time
- Spike - match when an event rate changes
- Flatline - match when there are less X events in Y time
- Blacklist or Whitelist - when a field matches a blacklist or whitelist
- Any - every event of a given filter
- Change - when a field has different values within a time frame

For this tool to be really useful in such a complex network, administrators should be spared from being bombarded with false warnings. Directory `etc/elastalert2/rules` should contain all the rules used by ElastAlert2. Each rule defines parameters on what triggers a match and a list of alerts to trigger for each match. Listing O.1 presents a critical activity rule created for this setup. Option `rule-type:any` defines that any event matching the `rule.category`[4] filter values, which are defined in `etc/suricata/classification.config`. Then, an alert will be triggered which is then sent to the configured email address.

[4]Field from Suricata `eve` log file that categorizes Suricata alerts

Chapter 6

Conclusions

This project contributed to the improvement of security in the network infrastructure of INESC TEC. It proved very useful to have the ability to analyze, detect and potentially prevent malicious activities capable of compromising the security of the institution's infrastructure. With this work it has been shown that it is possible to implement a fully customizable open source solution in a complex environment. Research done throughout this project demonstrated that Suricata would perform better when compared to similar tools. Nevertheless, it is essential that the solution is optimized and configured according to the environment and hosted hardware as a non-customized setup can lead to an increase of wasted resources and most importantly to packet loss. Rule management requires special attention as new attacks and new threats arise day after day. It is critical that the administrators keep the solution constantly updated in order to keep up with this evolution. It was possible to successfully complete all tasks planned for the system deployment, such as:

- Study of the state of the art in IDS solutions
- INESC TEC infraestructure scenario overview
- IDS implementation
- Data analysis platform implementation
- Preliminary tests
- Otimization
- Final Tests

6.1 Difficulties Faced

Due to the fact that the IDS is being implemented in real life traffic and in a complex network environment, it proved to be quite challenging to obtain tangible results regarding Suricata's alert accuracy. It was not possible to conduct isolated tests that would allow to trigger specific rules and generate specific alerts. Furthermore, the determination of false positives is a very time consuming process that requires a gradual analysis in order to keep improving Suricata's alert accuracy. Also, optimizations are not always a completely straightforward task because they depend on multiple factors like hardware and the network environment.

6.2 Future work

Although the implemented solution fulfills the necessary requirements imposed at the beginning of this project, this type of solutions requires continuous development and optimization work. Some ideas for future work could be:

- Rule otimization and consequent reduction of false positives
- Improvement of rule management process
- Possible update to existing hardware in order to improve otimization capabilities
- Improvement and optimization of the alert triggering and notification process

References

[1] *Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper*. Cisco. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (visited on 12/07/2020).

[2] *Hackers bloquearam rede do Instituto de Engenharia Mecânica da Universidade do Porto*. Jornal Expresso. URL: https://expresso.pt/sociedade/2020-08-03-Hackers-bloquearam-rede-do-Instituto-de-Engenharia-Mecanica-da-Universidade-do-Porto (visited on 12/10/2020).

[3] *INESC TEC*. INESC TEC. URL: https://www.inesctec.pt/pt/instituicao (visited on 11/12/2020).

[4] *Cybersecurity*. URL: https://www.itu.int/en/ITU-T/studygroups/com17/Pages/cybersecurity.aspx (visited on 11/14/2020).

[5] Rossouw von Solms and Johan van Niekerk. "From information security to cyber security". In: *Computers & Security*. Cybercrime in the Digital Economy 38 (Oct. 1, 2013), pp. 97–102. ISSN: 0167-4048. DOI: 10.1016/j.cose.2013.04.004. URL: http://www.sciencedirect.com/science/article/pii/S0167404813000801 (visited on 11/12/2020).

[6] Katharina Ziolkowski, Liina Areng, and NATO Cooperative Cyber Defence Centre of Excellence. *Peacetime regime for state activities in cyberspace: international law, international relations and diplomacy*. OCLC: 1033720962. 2013. (Visited on 11/15/2020).

[7] Dário Carreira. "Cibersegurança e Privacidade - Introdução, Definição e Conceitos". ISMAI, Sept. 26, 2019. (Visited on 11/15/2020).

[8] Eric Cole. *Network Security Bible*. Google-Books-ID: Iq8lPbhGRuYC. John Wiley & Sons, Mar. 31, 2011. 938 pp. ISBN: 978-0-470-57000-5. (Visited on 11/15/2020).

[9] Y. Sun. "The Study on Network Information Security". In: *2016 International Conference on Network and Information Systems for Computers (ICNISC)*. 2016 International Conference on Network and Information Systems for Computers (ICNISC). Apr. 2016, pp. 85–88. DOI: 10.1109/ICNISC.2016.028. (Visited on 11/15/2020).

[10] G. Lawton. "Open source security: opportunity or oxymoron?" In: *Computer* 35.3 (Mar. 2002). Conference Name: Computer, pp. 18–21. ISSN: 1558-0814. DOI: 10.1109/2.989921. (Visited on 11/11/2020).

[11] Marit Hansen, Kristian Köhntopp, and Andreas Pfitzmann. "The Open Source approach — opportunities and limitations with respect to security and privacy". In: *Computers & Security* 21.5 (Oct. 2002), pp. 461–471. ISSN: 01674048. DOI: 10.1016/S0167-4048(02)00516-3. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167404802005163 (visited on 01/07/2021).

[12] Luís Tavares. "Análise de eventos de segurança: baseado no OSSIM". In: (2015), p. 108. URL: https://mei.di.uminho.pt/sites/default/files/dissertacoes//eeum_di_dissertacao_pg20692.pdf (visited on 01/07/2021).

[13] B. Harris and R. Hunt. "TCP/IP security threats and attack methods". In: *Computer Communications* 22.10 (June 25, 1999), pp. 885–897. ISSN: 0140-3664. DOI: 10.1016/S0140-3664(99)00064-X. URL: http://www.sciencedirect.com/science/article/pii/S014036649900064X (visited on 11/24/2020).

[14] *Visão — Cibersegurança: há mais de 400 ataques por semana a empresas portuguesas*. Visão. Sept. 22, 2020. URL: https://visao.sapo.pt/exameinformatica/noticias-ei/software/2020-09-22-ha-mais-de-400-ataques-por-semana-a-empresas-portuguesas/ (visited on 12/05/2020).

[15] Guilherme Filipe Zorego Rodrigues Morais. "Análise e implementação de sistemas IDS e IPS". In: (2011). Accepted: 2013-09-17T15:55:32Z. URL: https://repositorio.ul.pt/handle/10451/9177 (visited on 11/11/2020).

[16] *Exploits: What You Need to Know*. Exploits: What You Need to Know. URL: https://www.avast.com/c-exploits (visited on 01/21/2021).

[17] *Must-know cybersecurity statistics & facts — NordVPN*. Oct. 9, 2020. URL: https://nordvpn.com/blog/cybersecurity-statistics/ (visited on 12/05/2020).

[18] *Threat Landscape — Malware Statistics & Trends by Year, Country & More*. URL: https://www.secplicity.org/threat-landscape/ (visited on 12/07/2020).

[19] *Operating System Market Share Worldwide*. StatCounter Global Stats. URL: https://gs.statcounter.com/os-market-share (visited on 12/05/2020).

[20] *2019 Global DDoS Threat Landscape Report — Imperva*. Blog. Feb. 5, 2020. URL: https://www.imperva.com/blog/2019-global-ddos-threat-landscape-report/ (visited on 12/12/2020).

[21] *Kaspersky DDoS Protection — DDoS Solutions — Kaspersky*. URL: https://www.kaspersky.com/small-to-medium-business-security/ddos-protection (visited on 12/07/2020).

[22] Navya Iyengar. "Evaluation of Network Based IDS and Deployment of multi-sensor IDS". In: (July 22, 2020), p. 4. URL: https://arxiv.org/abs/2007.11654 (visited on 12/13/2020).

[23] K A Scarfone and P M Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. NIST SP 800-94. Edition: 0. Gaithersburg, MD: National Institute of Standards and Technology, 2007, NIST SP 800–94. DOI: 10.6028/NIST.SP.800-94. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf (visited on 12/13/2020).

[24] P. García-Teodoro et al. "Anomaly-based network intrusion detection: Techniques, systems and challenges". In: *Computers & Security* 28.1 (Feb. 1, 2009), pp. 18–28. ISSN: 0167-4048. DOI: 10.1016/j.cose.2008.08.003. URL: http://www.sciencedirect.com/science/article/pii/S0167404808000692 (visited on 11/11/2020).

[25] Tarek S. Sobh. "Wired and wireless intrusion detection system: Classifications, good characteristics and state-of-the-art". In: *Computer Standards & Interfaces* 28.6 (Sept. 1, 2006), pp. 670–694. ISSN: 0920-5489. DOI: 10.1016/j.csi.2005.07.002. URL: http://www.sciencedirect.com/science/article/pii/S092054890500098X (visited on 12/18/2020).

[26] *The Key Challenges of IDS and how to overcome them.* Redscan. Oct. 16, 2017. URL: https://www.redscan.com/news/the-key-challenges-of-intrusion-detection-and-how-to-overcome-them/ (visited on 01/09/2021).

[27] *7 Best Intrusion Detection Software - IDS Systems - DNSstuff.* Software Reviews, Opinions, and Tips - DNSstuff. Feb. 18, 2020. URL: https://www.dnsstuff.com/network-intrusion-detection-software (visited on 01/09/2021).

[28] Deris Stiawan, Abdul Hanan Abdullah, and Mohd. Yazid Idris. "Characterizing Network Intrusion Prevention System". In: *International Journal of Computer Applications* 14.1 (Jan. 12, 2011), pp. 11–18. ISSN: 09758887. DOI: 10.5120/1811-2439. URL: http://www.ijcaonline.org/volume14/number1/pxc3872439.pdf (visited on 01/09/2021).

[29] João Paulo da Costa Calado. "Open source IDS/IPS in a production environment: comparing, assessing and implementing". In: (2018). Accepted: 2018-11-22T15:16:46Z. URL: https://repositorio.ul.pt/handle/10451/35418 (visited on 01/07/2021).

[30] *6.1. Rules Format — Suricata 6.0.0 documentation.* URL: https://suricata.readthedocs.io/en/suricata-6.0.0/rules/intro.html (visited on 01/09/2021).

[31] S. S. Tirumala, H. Sathu, and A. Sarrafzadeh. "Free and open source intrusion detection systems: A study". In: *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*. 2015 International Conference on Machine Learning and Cybernetics (ICMLC). Vol. 1. July 2015, pp. 205–210. DOI: 10.1109/ICMLC.2015.7340923.

[32] *Host-Based Intrusion Detection Systems - an overview — ScienceDirect Topics.* URL: https://www.sciencedirect.com/topics/computer-science/host-based-intrusion-detection-systems (visited on 12/14/2020).

[33] A. Warzyński and G. Kołaczek. "Intrusion detection systems vulnerability on adversarial examples". In: *2018 Innovations in Intelligent Systems and Applications (INISTA)*. 2018, pp. 1–4. DOI: 10.1109/INISTA.2018.8466271. (Visited on 12/18/2020).

[34] Aiman Moyaid Said, Dhanapal Durai Dominic, and Ibrahima Faye. "Real-time network anomaly detection architecture based on frequent pattern mining technique".

In: *2013 International Conference on Research and Innovation in Information Systems (ICRIIS)*. 2013 International Conference on Research and Innovation in Information Systems (ICRIIS). Kuala Lumpur, Malaysia: IEEE, Nov. 2013, pp. 392–397. ISBN: 978-1-4799-2487-5 978-1-4799-2486-8. DOI: 10.1109/ICRIIS.2013.6716742. URL: http://ieeexplore.ieee.org/document/6716742/ (visited on 01/24/2021).

[35] Veeramreddy Jyothsna and Koneti Munivara Prasad. "Anomaly-Based Intrusion Detection System". In: *Computer and Network Security*. Ed. by Jaydip Sen. IntechOpen, June 10, 2020. DOI: 10.5772/intechopen.82287. URL: https://www.intechopen.com/books/computer-and-network-security/anomaly-based-intrusion-detection-system (visited on 12/21/2020).

[36] Damiano Bolzoni and Sandro Etalle. "APHRODITE: an Anomaly-based Architecture for False Positive Reduction". In: (May 2006), p. 21. URL: https://www.researchgate.net/publication/1959242_APHRODITE_an_Anomaly-based_Architecture_for_False_Positive_Reduction (visited on 01/24/2021).

[37] Roman Fekolkin. "Intrusion Detection and Prevention Systems: Overview of Snort and Suricata". In: (2014), p. 5. (Visited on 12/30/2020).

[38] *OSSEC Documentation — OSSEC*. URL: https://www.ossec.net/docs/ (visited on 12/19/2020).

[39] *OSSEC Architecture — OSSEC*. URL: https://www.ossec.net/docs/docs/manual/ossec-architecture.html (visited on 01/01/2021).

[40] *FAQ · Hogzilla IDS*. Hogzilla IDS. URL: https://ids-hogzilla.org/faq/ (visited on 12/26/2020).

[41] *Installing Hogzilla · Hogzilla IDS*. Hogzilla IDS. URL: https://ids-hogzilla.org/install/ (visited on 01/01/2021).

[42] *About sFlow Overview @ sFlow.org*. URL: https://sflow.org/about/index.php (visited on 01/01/2021).

[43] *sFlow Support · Hogzilla IDS*. Hogzilla IDS. URL: https://ids-hogzilla.org/sflow/ (visited on 01/01/2021).

[44] 18 Jan 2018 Michael Kwaku Aboagye Feed 281up 4 comments. *Securing the Linux filesystem with Tripwire*. Opensource.com. URL: https://opensource.com/article/18/1/securing-linux-filesystem-tripwire (visited on 12/27/2020).

[45] Wazuh. *Architecture - Getting started · Wazuh 4.0 documentation*. URL: https://documentation.wazuh.com/4.0/getting-started/architecture.html (visited on 01/01/2021).

[46] Wazuh. *A comprehensive open source security platform · Wazuh · The Open Source Security Platform*. Wazuh. URL: https://wazuh.com/product/ (visited on 12/26/2020).

[47] Qinwen Hu, Se-Young Yu, and Muhammad Rizwan Asghar. "Analysing performance issues of open-source intrusion detection systems in high-speed networks". In: *Journal of Information Security and Applications* 51 (Apr. 1, 2020), p. 102426. ISSN: 2214-2126. DOI: 10.1016/j.jisa.2019.102426. URL: http://www.sciencedirect.com/science/article/pii/S2214212619306003 (visited on 11/11/2020).

[48] *Open Source IDS: Snort or Suricata? [Updated 2019]*. Infosec Resources. URL: https://resources.infosecinstitute.com/topic/open-source-ids-snort-suricata/ (visited on 01/09/2021).

[49] Surya Bhagavan Ambati and Deepti Vidyarthi. "A BRIEF STUDY AND COMPARISON OF, OPEN SOURCE INTRUSION DETECTION SYSTEM TOOLS". In: 1.10 (Dec. 2013), p. 7. URL: http://www.iraj.in/journal/journal_file/journal_pdf/3-27-139087836726-32.pdf (visited on 12/28/2020).

[50] F. Hock and P. Kortiš. "Commercial and open-source based Intrusion Detection System and Intrusion Prevention System (IDS/IPS) design for an IP networks". In: *2015 13th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. 2015 13th International Conference on Emerging eLearning Technologies and Applications (ICETA). Nov. 2015, pp. 1–4. DOI: 10.1109/ICETA.2015.7558466. (Visited on 11/11/2020).

[51] *Preprocessors - an overview — ScienceDirect Topics*. URL: https://www.sciencedirect.com/topics/computer-science/preprocessors (visited on 01/05/2021).

[52] *Pós Graduação em Redes Informáticas e Segurança de Redes*. URL: https://paginas.fe.up.pt/~mgi98020/pgr/snort.htm (visited on 01/05/2021).

[53] *3.2 Rules Headers*. URL: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node29.html (visited on 01/09/2021).

[54] *Writing Snort Rules*. URL: https://paginas.fe.up.pt/~mgi98020/pgr/writing_snort_rules.htm#rule_header (visited on 01/09/2021).

[55] Welcome Luthuli et al. "Evaluating the Effects of Hardware Configurations on Bro under DDoS Attacks". In: *2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)*. 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC). Plaine Magnien: IEEE, Dec. 2018, pp. 1–6. ISBN: 978-1-5386-6477-3. DOI: 10.1109/ICONIC.2018.8601247. URL: https://ieeexplore.ieee.org/document/8601247/ (visited on 12/29/2020).

[56] *Introduction — Zeek User Manual v3.2.3*. URL: https://docs.zeek.org/en/current/intro/index.html (visited on 12/29/2020).

[57] *6.35. Differences From Snort — Suricata 6.0.0 documentation*. URL: https://suricata.readthedocs.io/en/suricata-6.0.0/rules/differences-from-snort.html#automatic-protocol-detection (visited on 01/10/2021).

[58] *Features*. Suricata. Sept. 14, 2012. URL: https://suricata-ids.org/features/ (visited on 12/29/2020).

[59] Ana Vazao et al. "SIEM Open Source Solutions: A Comparative Study". In: *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*. 2019 14th Iberian Conference on Information Systems and Technologies (CISTI). Coimbra, Portugal: IEEE, June 2019, pp. 1–5. ISBN: 978-989-98434-9-3. DOI: 10.23919/CISTI.2019.8760980. URL: https://ieeexplore.ieee.org/document/8760980/ (visited on 01/09/2021).

[60] Lucas Souza. *[SIEM] Introdução ao OSSIM*. Medium. Mar. 4, 2020. URL: https://lucxs.medium.com/siem-introdu%C3%A7%C3%A3o-ao-ossim-98a45fffeb19 (visited on 01/08/2021).

[61] *Using the ELK Stack for SIEM*. Logz.io. June 20, 2018. URL: https://logz.io/blog/elk-siem/ (visited on 01/08/2021).

[62] *What are Beats? — Beats Platform Reference [7.10] — Elastic*. URL: https://www.elastic.co/guide/en/beats/libbeat/current/beats-reference.html (visited on 01/08/2021).

[63] *Download Community Edition Today*. URL: https://go.siemonster.com/Community-Edition (visited on 01/09/2021).

[64] Louis Bernardo. "Targeted Attack Detection by Means of Free and Open Source Solutions". MSc Thesis. Jan. 7, 2019. (Visited on 01/10/2021).

[65] *11 of the Leading Open Source SIEM Tools*. Logz.io. May 7, 2018. URL: https://logz.io/blog/open-source-siem-tools/ (visited on 01/09/2021).

[66] *About Splunk Free - Splunk Documentation*. URL: https://docs.splunk.com/Documentation/Splunk/8.1.1/Admin/MoreaboutSplunkFree (visited on 01/21/2021).

[67] *Visualize and Correlate IDS Alerts with Open Source Tools — Graylog*. URL: https://www.graylog.org/post/visualize-and-correlate-ids-alerts-with-open-source-tools (visited on 01/09/2021).

[68] *The thinking behind the Graylog architecture and why it matters to you — Graylog 4.0.0 documentation*. URL: https://docs.graylog.org/en/4.0/pages/ideas_explained.html (visited on 01/23/2021).

[69] *What's Inside — Graylog Features*. URL: https://www.graylog.org/features (visited on 01/23/2021).

[70] *Planning Your Log Collection — Graylog 4.0.0 documentation*. URL: https://docs.graylog.org/en/4.0/pages/getting_started/planning.html (visited on 01/23/2021).

[71] *ROCK NSM*. URL: https://rocknsm.io/ (visited on 01/10/2021).

[72] *About — Security Onion 2.3 documentation*. URL: https://docs.securityonion.net/en/2.3/about.html#security-onion (visited on 01/10/2021).

[73] *Security-Onion-Solutions/security-onion*. GitHub. URL: https://github.com/Security-Onion-Solutions/security-onion (visited on 01/10/2021).

[74] *Architecture — Security Onion 16.04.7.2 documentation*. URL: https://docs.securityonion.net/en/16.04/architecture.html (visited on 01/22/2021).

[75] Stamus Networks LLC. *Stamus Networks — SELKS*. URL: https://www.stamus-networks.com/scirius-open-source (visited on 01/10/2021).

[76] *StamusNetworks/SELKS*. original-date: 2014-05-15T13:13:50Z. Jan. 9, 2021. URL: https://github.com/StamusNetworks/SELKS (visited on 01/10/2021).

[77] Adeeb Alhomoud et al. "Performance Evaluation Study of Intrusion Detection Systems". In: *Procedia Computer Science* 5 (2011), pp. 173–180. ISSN: 18770509. DOI: 10.1016/j.procs.2011.07.024. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050911003498 (visited on 01/11/2021).

[78] Eugene Albin. "A comparative analysis of the Snort and Suricata intrusion-detection systems". In: (Sept. 2012), p. 69. URL: https://calhoun.nps.edu/bitstream/handle/10945/5480/11Sep_Albin.pdf?sequence=1&isAllowed=y (visited on 01/10/2021).

[79] Joshua S. White, Thomas Fitzsimmons, and Jeanna N. Matthews. "Quantitative analysis of intrusion detection systems: Snort and Suricata". In: SPIE Defense, Security, and Sensing. Ed. by Igor V. Ternovskiy and Peter Chin. Baltimore, Maryland, USA, May 28, 2013, p. 875704. DOI: 10.1117/12.2015616. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2015616 (visited on 01/12/2021).

[80] "Comparison of Different Intrusion Detection and Prevention Systems". In: *International Journal of Emerging Technology and Advanced Engineering* (Dec. 2012). ISSN: 22502459. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.440.5084&rep=rep1&type=pdf (visited on 01/12/2021).

[81] Jonas Taftø Rødfoss. "Comparison of open source network intrusion detection systems". In: (2011). Accepted: 2013-03-12T08:06:00Z. URL: https://www.duo.uio.no/handle/10852/8951 (visited on 01/12/2021).

[82] Mauno Pihelgas. "A COMPARATIVE ANALYSIS OF OPEN- SOURCE INTRUSION DETECTION SYSTEMS". In: 2012. URL: http://mauno.pihelgas.eu/

files/Mauno_Pihelgas-A_Comparative_Analysis_of_OpenSource_Intrusion_Detection_Systems.pdf (visited on 01/12/2021).

[83] Qinwen Hu, Muhammad Rizwan Asghar, and Nevil Brownlee. "Evaluating network intrusion detection systems for high-speed networks". In: *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*. 2017 27th International Telecommunication Networks and Applications Conference (ITNAC). Melbourne, VIC: IEEE, Nov. 2017, pp. 1–6. ISBN: 978-1-5090-6796-1. DOI: 10.1109/ATNAC.2017.8215374. URL: http://ieeexplore.ieee.org/document/8215374/ (visited on 01/12/2021).

[84] Amtul Saboor, Monis Akhlaq, and Baber Aslam. "Experimental evaluation of Snort against DDoS attacks under different hardware configurations". In: *2013 2nd National Conference on Information Assurance (NCIA)*. 2013 2nd National Conference on Information Assurance (NCIA). Rawalpindi, Pakistan: IEEE, Dec. 2013, pp. 31–37. ISBN: 978-1-4799-1288-9 978-1-4799-1287-2. DOI: 10.1109/NCIA.2013.6725321. URL: http://ieeexplore.ieee.org/document/6725321/ (visited on 01/12/2021).

[85] *2.4 Event Processing*. URL: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node19.html (visited on 01/24/2021).

[86] Sharmila KishorWagh, Vinod K. Pachghare, and Satish R. Kolhe. "Survey on Intrusion Detection System using Machine Learning Techniques". In: *International Journal of Computer Applications* 78.16 (Sept. 18, 2013), pp. 30–37. ISSN: 09758887. DOI: 10.5120/13608-1412. URL: http://research.ijcaonline.org/volume78/number16/pxc3891412.pdf (visited on 01/21/2021).

[87] *Suricata User Guide — Suricata 6.0.2 documentation*. URL: https://suricata.readthedocs.io/en/suricata-6.0.2/ (visited on 05/24/2021).

[88] *rfc1918*. URL: https://datatracker.ietf.org/doc/html/rfc1918 (visited on 09/08/2021).

[89] *Set up minimal security for Elasticsearch — Elasticsearch Guide [7.13] — Elastic*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/7.13/security-minimal-setup.html (visited on 06/18/2021).

[90] *9.3. Tuning Considerations — Suricata 6.0.2 documentation.* URL: https://suricata.readthedocs.io/en/suricata-6.0.2/performance/tuning-considerations.html# (visited on 06/11/2021).

[91] *Suricata - disable.conf seems not working - Rules.* en. URL: https://forum.suricata.io/t/suricata-disable-conf-seems-not-working/1504/2 (visited on 09/09/2021).

[92] *9.2. Packet Capture — Suricata 6.0.2 documentation.* URL: https://suricata.readthedocs.io/en/suricata-6.0.2/performance/packet-capture.html (visited on 06/10/2021).

[93] *10.1. Suricata.yaml — Suricata 6.0.2-dev documentation.* URL: https://suricata.readthedocs.io/en/latest/configuration/suricata-yaml.html#pattern-matcher-settings (visited on 06/11/2021).

Appendix A

Suricata default rule file configuration

```
##
## Configure Suricata to load Suricata-Update managed rules.
##

default-rule-path: /var/lib/suricata/rules

rule-files:
  - suricata.rules
```

Listing A.1: Default rule file configuration

Appendix B

Suricata rule configuration

```
##
## Configure Suricata to load Suricata-Update managed rules.
##

default-rule-path: /var/lib/suricata/rules
rule-files:

##
## Configure Suricata to load Suricata-Update managed rules.
##

default-rule-path: /var/lib/suricata/rules
rule-files:

 - emerging-sql.rules
 - emerging-exploit.rules
 - emerging-coinminer.rules
 - emerging-dos.rules
 - emerging-exploit_kit.rules
 - emerging-phishing.rules
 - emerging-malware.rules
 - emerging-user_agents.rules
 - emerging-scan.rules
 - emerging-attack_response.rules
```

```
- emerging-worm.rules
- tor.rules
- botcc.rules
- botcc.portgrouped.rules
- emerging-hunting.rules
- emerging-web_server.rules
- emerging-web_client.rules
- emerging-voip.rules
- emerging-p2p.rules
```

Listing B.1: Rule files configuration

Appendix C

Suricata `fast.log` output

```
07/15/2021-00:42:42.061938  [**] [1:2009582:3] ET SCAN NMAP -
   sS window 1024 [**] [Classification: Attempted Information
    Leak] [Priority: 2] {TCP} 162.142.125.69:56180 -> x.x.x.x
   :3365
```

Listing C.1: `fast.log` output

Appendix D

Suricata `eve-alerts.json` output


```
{"timestamp":"2021-06-16T16:42:59.681482+0100","flow_id
   ":2023683833161226,"in_iface":"enp2s0f1","event_type":"
   alert","vlan":[101],"src_ip":"167.248.133.28","src_port
   ":25548,"dest_ip":"x.x.x.x","dest_port":5432,"proto":"TCP
   ","metadata":{"flowbits":["ET.Evil","ET.DshieldIP"]},"
   alert":{"action":"allowed","gid":1,"signature_id
   ":2010939,"rev":3,"signature":"ET SCAN Suspicious inbound
   to PostgreSQL port 5432","category":"Potentially Bad
   Traffic","severity":2,"metadata":{"created_at":["2010
   _07_30"],"former_category":["HUNTING"],"updated_at":["2018
   _03_27"]}},"flow":{"pkts_toserver":1,"pkts_toclient":0,"
   bytes_toserver":60,"bytes_toclient":0,"start":"2021-06-16
   T16:42:59.681482+0100"}}
}
```


Listing D.1: `eve-alerts.json` output

Appendix E

Elasticsearch `curl` output

```
root@suricata-elk01:~# curl -X GET "x.x.x.x:9200"
{
  "name" : "suricata-elk01",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "HOIBmrB_RnyC0pV-p2DJDg",
  "version" : {
    "number" : "7.13.1",
    "build_type" : "deb",
    "build_hash" : "9a7758028e4ea59bcab41c12004603c5a7dd84a9
       ",
    "build_date" : "2021-05-28T17:40:59.346932922Z",
    "build_snapshot" : false,
    "lucene_version" : "8.8.2",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Listing E.1: Elasticsearch `curl` output

Appendix F

Filebeat Suricata module configuration

```
# Module: suricata
# Docs: https://www.elastic.co/guide/en/beats/filebeat/7.x/
   filebeat-module-suricata.html

- module: suricata
  # All logs
  eve:
    enabled: true

    # Set custom paths for the log files. If left empty,
    # Filebeat will choose the paths depending on your OS.
    var.paths:
      - "/var/log/suricata/eve-events.json"
      - "/var/log/suricata/eve-alerts.json"
```

Listing F.1: Suricata Filebeat module configuration

Appendix G

Filebeat default index name change

```
# ============ Elasticsearch template setting ===============

setup.ilm.enabled: auto
setup.ilm.rollover_alias: "suricata-ids" #Write your alias
setup.ilm.pattern: "{now/d}-000001"
setup.template.name: "suricata-ids"
setup.template.pattern: "suricata-ids-*"
setup.dashboards.index: "suricata-ids-*"
setup.template.settings:
  index.number_of_shards: 4
  #index.codec: best_compression
  #_source.enabled: false

# ------------------- Elasticsearch Output ------------------

output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["suricata-elk01.inesctec.pt:9200"]
  index: "suricata-ids-%{+yyyy.MM.dd}"
```

Listing G.1: Filebeat changed index name

Appendix H

Suricata `suricata.log` file output

```
9/7/2021 -- 12:06:40 - <Config> - Enabling locked memory for
   mmap on iface enp2s0f1
9/7/2021 -- 12:06:40 - <Config> - Enabling tpacket v3 capture
    on iface enp2s0f1
9/7/2021 -- 12:06:40 - <Config> - Using flow cluster mode for
    AF_PACKET (iface enp2s0f1)
9/7/2021 -- 12:06:40 - <Config> - Using defrag kernel
   functionality for AF_PACKET (iface enp2s0f1)
9/7/2021 -- 12:06:40 - <Perf> - 16 cores, so using 16 threads
9/7/2021 -- 12:06:40 - <Perf> - Using 16 AF_PACKET threads
   for interface enp2s0f1
9/7/2021 -- 12:06:40 - <Config> - enp2s0f1: enabling zero
   copy mode by using data release call
9/7/2021 -- 12:06:40 - <Info> - Going to use 16 thread(s)
9/7/2021 -- 12:06:40 - <Config> - Enabling locked memory for
   mmap on iface enp3s0f0
9/7/2021 -- 12:06:40 - <Config> - Enabling tpacket v3 capture
    on iface enp3s0f0
9/7/2021 -- 12:06:40 - <Config> - Using flow cluster mode for
    AF_PACKET (iface enp3s0f0)
9/7/2021 -- 12:06:40 - <Config> - Using defrag kernel
   functionality for AF_PACKET (iface enp3s0f0)
9/7/2021 -- 12:06:40 - <Perf> - 16 cores, so using 16 threads
```

```
9/7/2021 -- 12:06:40 - <Perf> - Using 16 AF_PACKET threads
   for interface enp3s0f0
9/7/2021 -- 12:06:40 - <Config> - enp3s0f0: enabling zero
   copy mode by using data release call
9/7/2021 -- 12:06:40 - <Info> - Going to use 16 thread(s)
9/7/2021 -- 12:06:40 - <Config> - Enabling locked memory for
   mmap on iface enp3s0f1
9/7/2021 -- 12:06:40 - <Config> - Enabling tpacket v3 capture
    on iface enp3s0f1
9/7/2021 -- 12:06:40 - <Config> - Using flow cluster mode for
    AF_PACKET (iface enp3s0f1)
9/7/2021 -- 12:06:40 - <Config> - Using defrag kernel
   functionality for AF_PACKET (iface enp3s0f1)
9/7/2021 -- 12:06:41 - <Perf> - 16 cores, so using 16 threads
9/7/2021 -- 12:06:41 - <Perf> - Using 16 AF_PACKET threads
   for interface enp3s0f1
9/7/2021 -- 12:06:41 - <Config> - enp3s0f1: enabling zero
   copy mode by using data release call
9/7/2021 -- 12:06:41 - <Info> - Going to use 16 thread(s)
9/7/2021 -- 12:06:41 - <Config> - using 1 flow manager
   threads
9/7/2021 -- 12:06:41 - <Config> - using 1 flow recycler
   threads
9/7/2021 -- 12:06:41 - <Info> - Running in live mode,
   activating unix socket
9/7/2021 -- 12:06:41 - <Info> - Using unix socket file '/var/
   run/suricata/suricata-command.socket'
9/7/2021 -- 12:06:41 - <Notice> - all 48 packet processing
   threads, 4 management threads initialized, engine started.
```

Listing H.1: Configured threads - `suricata.log` file

Appendix I

Suricata `enable.conf` configuration file

```
# suricata-update - enable.conf

# Example of enabling a rule by signature ID (gid is optional
   ).
# 1:2019401
# 2019401

# Example of enabling a rule by regular expression.
# - All regular expression matches are case insensitive.
# re:heartbleed
# re:MS(0[7-9]|10)-\d+

# Examples of enabling a group of rules.
# group:emerging-icmp.rules
# group:emerging-dos
# group:emerging*

group:emerging-sql.rules
group:emerging-exploit.rules
group:emerging-coinminer.rules
group:emerging-dos.rules
group:emerging-exploit_kit.rules
```

```
group:emerging-phishing.rules
group:emerging-malware.rules
group:emerging-user_agents.rules
group:emerging-scan.rules
group:attack_response.rules
group:emerging-worm.rules
group:tor.rules
group:botcc.rules
group:botcc.portgrouped.rules
group:emerging-hunting.rules
group:emerging-web_server.rules
group:emerging-web_client.rules
group:emerging-voip.rules
group:emerging-p2p.rules
group:sslblacklist.rules
```

Listing I.1: `enable.conf`

Appendix J

`ethtool` configuration

```
Features for enp2s0f1:
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: on
        tx-checksum-ip-generic: off [fixed]
        tx-checksum-ipv6: on
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: off
        tx-scatter-gather: off
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: off
        tx-tcp-segmentation: off
        tx-tcp-ecn-segmentation: off
        tx-tcp-mangleid-segmentation: off
        tx-tcp6-segmentation: off
udp-fragmentation-offload: off
generic-segmentation-offload: off
generic-receive-offload: off
large-receive-offload: off [fixed]
rx-vlan-offload: off
tx-vlan-offload: off
ntuple-filters: off [fixed]
receive-hashing: on
```

```
highdma: on [fixed]
rx-vlan-filter: off [fixed]
vlan-challenged: off [fixed]
tx-lockless: off [fixed]
netns-local: off [fixed]
tx-gso-robust: off [fixed]
tx-fcoe-segmentation: off [fixed]
tx-gre-segmentation: off [fixed]
tx-gre-csum-segmentation: off [fixed]
tx-ipxip4-segmentation: off [fixed]
tx-ipxip6-segmentation: off [fixed]
tx-udp_tnl-segmentation: off [fixed]
tx-udp_tnl-csum-segmentation: off [fixed]
tx-gso-partial: off [fixed]
tx-sctp-segmentation: off [fixed]
tx-esp-segmentation: off [fixed]
fcoe-mtu: off [fixed]
tx-nocache-copy: off
loopback: off [fixed]
rx-fcs: off [fixed]
rx-all: off [fixed]
tx-vlan-stag-hw-insert: off [fixed]
rx-vlan-stag-hw-parse: off [fixed]
rx-vlan-stag-filter: off [fixed]
l2-fwd-offload: off [fixed]
hw-tc-offload: off [fixed]
esp-hw-offload: off [fixed]
esp-tx-csum-hw-offload: off [fixed]
rx-udp_tunnel-port-offload: off [fixed]
```

Listing J.1: `ethtool` disabled offloading

Appendix K

Kibana TLS configuration

```
# Enables SSL and paths to the PEM-format SSL certificate and
    SSL key files, respectively.
# These settings enable SSL for outgoing requests from the
   Kibana server to the browser.
server.ssl.enabled: true
server.ssl.certificate: /etc/certs/suricata-
   elk01_inesctec_pt_cert.cer
server.ssl.key: /etc/certs/private.key
```

Listing K.1: Kibana TLS configuration

Appendix L

Filebeat TLS configuration

```
# ============================ Kibana
   ============================

# Starting with Beats version 6.0.0, the dashboards are
   loaded via the Kibana API.
# This requires a Kibana endpoint configuration.
setup.kibana:

  # Kibana Host
  # Scheme and port can be left out and will be set to the
     default (http and 5601)
  # In case you specify and additional path, the scheme is
     required: http://localhost:5601/path
  # IPv6 addresses should always be defined as: https
     ://[2001:db8::1]:5601
  host: "https://suricata-elk01.inesctec.pt:5601"
  ssl.enabled: true
  ssl.certificate_authorities: ["/etc/pki/root/
     suricata_inesctec_pt_interm.cer"]
  ssl.certificate: "/etc/pki/server/suricata_inesctec_pt.pem"
  ssl.key: "/etc/pki/server/suricata_inesctec_pt.key"
```

Listing L.1: Filebeat TLS configuration

Appendix M

Suricata `threshold.conf` configuration file

```
suppress gen_id 1, sig_id 2030387

#EXPLOIT

suppress gen_id 1, sig_id 2001022

#MALWARE

suppress gen_id 1, sig_id 2008438
suppress gen_id 1, sig_id 2012229
suppress gen_id 1, sig_id 2009205
suppress gen_id 1, sig_id 2009206
suppress gen_id 1, sig_id 2009207
suppress gen_id 1, sig_id 2009208



#SQL

suppress gen_id 1, sig_id 2102329, track by_src, ip [<ZOOM IP
    addresses>]
```

```
#ET WEB_SERVER


suppress gen_id 1, sig_id 2019244, track by_src, ip
   194.117.26.222/24
```

Listing M.1: `threshold.conf` file

Appendix N

suricata.log - Processed rules

```
5/8/2021 -- 16:07:00 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-sql.rules
5/8/2021 -- 16:07:00 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-exploit.rules
5/8/2021 -- 16:07:00 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-coinminer.rules
5/8/2021 -- 16:07:00 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-dos.rules
5/8/2021 -- 16:07:00 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-exploit_kit.rules
5/8/2021 -- 16:07:01 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-phishing.rules
5/8/2021 -- 16:07:02 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-malware.rules
5/8/2021 -- 16:07:03 - <Info> - Rule with ID 2026440 is
   bidirectional, but source and destination are the same,
   treating the rule as unidirectional
5/8/2021 -- 16:07:06 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-user_agents.rules
5/8/2021 -- 16:07:06 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-scan.rules
5/8/2021 -- 16:07:06 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-attack_response.rules
```

```
5/8/2021 -- 16:07:06 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-worm.rules
5/8/2021 -- 16:07:06 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-hunting.rules
5/8/2021 -- 16:07:06 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-web_server.rules
5/8/2021 -- 16:07:06 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-web_client.rules
5/8/2021 -- 16:07:07 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-voip.rules
5/8/2021 -- 16:07:07 - <Config> - Loading rule file: /var/lib
   /suricata/rules/emerging-p2p.rules
5/8/2021 -- 16:07:07 - <Config> - Loading rule file: /var/lib
   /suricata/rules/sslblacklist.rules
5/8/2021 -- 16:07:07 - <Info> - 17 rule files processed.
   21586 rules successfully loaded, 0 rules failed
5/8/2021 -- 16:07:07 - <Info> - Threshold config parsed: 14
   rule(s) found
```

Listing N.1: Suricata processed rules

Appendix O

ElastAlert2 critical activity rule

```
name: Critical Activity Detected

alert_subject: "{0} - {1} to {2}"
alert_subject_args:
- suricata.eve.alert.signature
- source.address
- destination.address

# (Required)
# Type of alert.
# the frequency rule type alerts when num_events events occur
    with timeframe time
# The any rule will match everything. Every hit that the
   query returns will generate an alert.
type: any

# (Required)
# Index to search, wildcard supported
index: suricata-ids-*

realert:
  minutes: 0
```

```
filter:
- terms:
    rule.category: ["Executable code was detected", "
       Detection of a Denial of Service Attack", "Targeted
       Malicious Activity was Detected", "Exploit Kit
       Activity was Detected", "Malware Command and Control
       Activity Detected", "Crypto Currency Mining Activity
       Detected"]
- terms:
    suricata.eve.in_iface: ["enp3s0f0", "enp3s0f1"]


alert:
- "email"
email:
- "ngms.17@gmail.com"
smtp_host: "smtp.gmail.com"
smtp_port: 465
smtp_ssl: true
from_addr: "ngms.17@gmail.com"
smtp_auth_file: "smtp_auth_file.yaml"
```

Listing O.1: ElastAlert2 critical activity rule