POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

A monitoring system for embedded devices widely distributed

Supervisor:

Prof. Cataldo Basile

Candidate:

Rosario Iudica

Academic Year 2021/2022
Torino

Abstract

Today, the technological world is increasingly affected by cyber-attacks and cybercrime, and, at the same time, it is proliferating. Consequently, some ways of protection become essential. As a result, one of the most critical countermeasures is the detection of these cyber-attacks. Each attack leaves traces in the target system in different forms. In this respect, the purpose of this thesis is, firstly, a deeper analysis of various monitoring and logging techniques, the source information they process, and the solutions which better can detect most attacks, second allowing a better comprehension of what is going on through the creation of security alerts. Moreover, detection needs to be optimized to avoid false positives, i.e., alerts for harmless and not anomalous events. Hence, this thesis also proposes an alert correlation, an additional technique that permits the improvement of accuracy, correctness, and efficiency of the security logging process. For this purpose, various monitoring and logging tools have been compared based on tailored discriminants to find the solution that best fits the proposed case study, a platform with various embedded devices spread worldwide that need to be monitored from the security point of view. Once the chosen solution has been described in a detailed way, it is implemented in the proposed platform, considering a list of suitable events to monitor the case study. For the completeness of the detection, various adjustments have been created without significant degradation of performance following the chosen solution's semantics. Several tests have been carried out to validate the tool's effectiveness: simulated cyber-attacks, tests for information gathering capabilities, and performance impact tests. The performed tests highlighted the excellent capabilities of the chosen product, demonstrating how a monitoring and logging tool is one of the most valuable lines of defense against cyber-attacks. However, to improve the defense capabilities,

introducing another product that permits correlating the outputs of the logging tool is desirable, without underestimating the use of constant vigilance about new vulnerabilities and attack techniques employed by attackers.

Ringraziamenti

Vorrei cogliere l'occasione per ringraziare il *Politecnico di Torino*, che in questi anni mi ha permesso di accrescere il mio bagaglio culturale migliorando i miei punti di forza e limando le mie debolezze. Voglio ringraziarlo, inoltre, perché mi ha permesso, nonostante la situazione pandemica che ci ha colpiti, una didattica sempre attenta alle mie esigenze.

Vorrei, inoltre, ringraziare di vero cuore il relatore della tesi, il *professore Cataldo Basile* che ha sempre dimostrato una grande disponibilità e mi ha permesso lo svolgimento di questo lavoro di tesi in azienda, dandomi la possibilità di fare esperienza nel mondo del lavoro.

Un altro sentito ringraziamento va all'azienda che ha permesso la stesura di questo lavoro, *Drivesec*. All'interno di Drivesec ho potuto sperimentare la grande gentilezza e disponibilità di tutti coloro che vi lavorano. In particolare, volevo esprimere la mia immensa gratitudine al CEO, *Giuseppe Faranda-Cordella* e al CTO, *Rossella Lertora* per le possibilità di crescita che mi hanno dato e per la fiducia che hanno riposto in me. Inoltre, volevo esprimere la mia riconoscenza verso i dipendenti per il loro supporto e per avermi dato la possibilità di imparare tanto.

Ovviamente, tutto ciò non sarebbe stato possibile se non avessi avuto al mio fianco delle figure indispensabili che mi aiutassero nei grandi momenti di difficoltà che ho incontrato durante questo percorso. In particolare, ringrazio *i miei genitori* per avermi sempre sostenuto, *mia sorella* per essere stata il faro a cui appoggiarsi ogni volta ne avessi bisogno e a *mio cognato* per

avermi guidato nei vari adempimenti burocratici.

Un altro ringraziamento va ai miei *amici* che sono stati fonte di allegria e leggerezza durante questi anni. Parlo ovviamente degli amici di sempre, coloro che non mi hanno mai lasciato solo e che hanno fatto sempre il tifo per me, anche in modo non convenzionale.

Un ringraziamento alle mie *nonne*, ai miei *zii* e a tutti i miei *cugini* poiché ho ricevuto da parte loro preziosi consigli al momento opportuno.

Ai nuovi pochi *colleghi* che ho avuto la possibilità di conoscere volevo esprimere la mia gratitudine per aver condiviso un pezzo di percorso assieme a me. Inoltre, un ringraziamento ai vecchi colleghi che ho incontrato a Catania ai quali devo molto per il metodo di studio acquisito e di cui ho sentito ogni giorno la mancanza.

Un ringraziamento a tutte le altre persone che mi sono state accanto e che hanno creduto in me.

Per ultimo, ma proprio per questo importantissimo, un ringraziamento speciale alla mia dolce metà, *Adriana*. La ringrazio perché c'è stata in qualsiasi momento io ne avessi bisogno ed è stata anche lei un faro a cui aggrapparsi. Grazie al suo prezioso aiuto questo lavoro di tesi è stato meno gravoso e la sua presenza ha riempito di affetto ogni istante.

Table of Contents

List of Figures

Acronyms

AP

Access Point

API

Application Programming Interface

ARP

Address Resolution Protocol

BSD

Berkeley Software Distribution

Can

Controller Area Network

Can-FD

Controller Area Network Flexible Data-Rate

CIS

Center Internet Security

CIS-CAT

CIS Configuration Assessment Tool

CVE

Common Vulnerabilities and Exposures

DAC

Discretionary Access Control

DNS

Domain Name System

DoS

Denial of Service

EMM

Enterprise Mobility Management

FIM

File Integrity Monitoring

GDB

GNU Debugger

GECOS

General Electric Comprehensive Operating Supervisor

GNU

GNU's Not Unix

HIDS

Host-based Intrusion Detection System

HTTPS

Hypertext Transfer Protocol Secure

ID

Identifier

IDS

Intrusion Detection System

IoT

Internet of Things

IP

Internet Protocol

JSON

JavaScript Object Notation

LTE

Long Term Evolution

MD-5

Message Digest 5

MIB

Management Information Base

NIST

National Institute of Standards and Technology

OpenSCAP

Open-Security Content Automation Protocol

OS

Operating System

OWASP

Open Web Application Security Project

PAM

Pluggable Authentication Modules

PCA

Principal Component Analysis

PCI-DSS

Payment Card Industry Data Security Standard

PID

Process Identifier

RBAC

Role-based Access Control

REST

Representational State Transfer

SCA

Security Configuration Assessment

SCAP

Security Content Automation Protocol

SHA-1

Secure Hash Algorithm-1

SHA-256

Secure Hash Algorithm-256

SIEM

Security Information and Event Management System

SIM

Security Incident Management

SNMP

Simple Network Management Protocol

SOC

Security Operations Center

SQLite

Structured Query Language Lite

SSH

Secure Shell

SSL

Secure Sockets Layer

SSO

Single Sign On

TCB

Trusted Computing Base

TCP

Transmission Control Protocol

TLS

Transport Layer Security

UDP

Unified Datagram Protocol

UFW

Uncomplicated Firewall

UNIX

Uniplexed Information and Computing System

URL

Uniform Resource Locator

USB

Universal Serial Bus

WiFi

Wireless Fidelity

XDR

Extended Detection and Response

XML

Extensive Markup Language

YAML

YAML Ain't Markup Language

YARA

Yet Another Recursive/Ridiculous Acronym

Introduction

The world is conscientious about safety and security about themselves but most of the time people never consider their data. Data can be manipulated, stolen, sold, observed and those are the reasons why cyber-security has born. In according to NIST's definition, Cyber-security is "the process of protecting information by preventing, detecting, and responding to attacks"[1] where the attacks are committed against personal and commercial data. The only way people could protect data is improving the overall security of a system and one of the aspects that improve this process is the detection of security events and their analysis. This paper's main goal is to explore the techniques for detecting attacks, what can be estimated as an attack, and what is the best solution to fit those problems. In the first chapter, firstly, the source information for finding attacks have been described with a very fine granularity trying to describe how they would be collected, and what are more important than the other; secondly, the real subject of this thesis has been presented: the Intrusion Detection Systems. They have been dissected with very meticulously defining the techniques and the sub-techniques used for describing how the source information are analyzed by all them; in addition, they have been classified according to what are the source information dividing, among all, the host-based and the network-based ones. Immediately after, an improvement for the existent systems has been described, focusing the attention on what are the most important information the cyber-security teams are interested. Thus, the alert correlation has been presented trying to distinguish, as done for the Intrusion Detection Systems, the techniques and sub-techniques. Finally, the last part of the first chapter presents the case study that has been taken in consideration, trying to describe how it is composed and how it works.
In the second chapter, all the most known security and monitoring tools

have been presented, and they have been examined by different discriminants and the best solution that fits with the case study has been chosen for the implementation and, sequentially, it has been described in a deeper way. That is, firstly, the architecture has been presented trying to distinguish the different elements that compose the platform; thereafter, the entire platform has been described entirely while in the third sub-section, the capabilities of the chosen monitoring tool have been introduced. At the end of the chapter, the knowledge used by the monitoring tool has been described not in a deep way.

The last chapter describes how the Intrusion Detection System chosen has been implemented in the case study, under the constraints of a test environment, and various events have been monitored, validating the capabilities of the chosen tool.

Chapter 1

Security Logging and Monitoring

Nowadays, all systems permit explicit most part of useful information about security events through system logs. “A log is a file generated by a software system that contains a set of events, i.e., changes that happened in such system”[2]. In according to AppDynamics “the purpose of logging is to create an ongoing record of application events”[1]. Following this definition, it is clear how much important the logging process is; however it does not concern only the effortless process of collecting events but also their analysis. “The logged event data needs to be available to review and there are processes in place for appropriate monitoring, alerting and reporting”[3]. From the latter NIST definition, an unfamiliar word comes out: *monitoring*. “Monitoring is the live review of application and security logs using various forms of automation”[4]. “Using a combination of logging tools and real-time monitoring systems helps improve observability”[1]: indeed, “one of the top ten security risks identified by NIST is ”Insufficient logging and monitoring”. The security risks allow criminals to further attack systems”[2]. So basically, for NIST a good logging process and further consumption, correlation, analysis, and management are needed. Moreover, for better management of security risk, the attention is related to *security logging*.

1.1 Security Logging

“Log data needs to tell a succinct but complete story”[1]. Basically, logs must the following features:

- Complete
- Descriptive
- Contextual

OWASP gives an explanation about application logging, which could be used as a source of security logging. Indeed, logs coming from applications can be used for: the identification of security incidents, monitoring of policy violations, the establishment of policy violations, assistance to non-repudiation controls, provisioning of information about problems and unusual conditions, contributions of additional-specific data for incident investigation and helps defending against vulnerability identification and exploitation through attack detection. Moreover, following the OWASP definition, application logs are very useful for recording other types of elements such as security events, business processes, audit trails, performance events, data from requests for information, and so on. In addition, logs "can provide information useful during a digital investigation to identify who, what, when, and how a security breach occurred”[2]. Moreover, “it is important not to log too much, or too little”[3]: “logging too little poses the risk of missing relevant information for posterior analysis”[2]; otherwise “logging too much has the drawback of adding more code to write and maintain, producing non-informative logs having limited usefulness, and increasing usage of resources”[2]. The interest should not be based only on the logs but also on the collection, storing, verification, and monitoring process. Basically, the logging process is not so easy to implement, even if it seems to be. According to OWASP, the first phase is about the identification of event data sources which can be varied and heterogeneous. Here are some examples: client software, embedded instrumentation code, network firewalls, network and host intrusion detection systems, closely related applications, database application, reputation monitoring services, operating system and so on.
Another important step is the location where the event data must be stored.

Commonly, the preferred locations are file systems and databases. If the former is used, it is better to use a separate partition than those used by the operating system, other application files and user-generated content. In the latter case, it is preferable a separate database account used only for log data. Moreover, a standard format over secure protocols is preferable since it facilitates the integration with centralized logging services.
The following step is crucial because "the level and content of security monitoring, alerting and reporting needs to be set during the requirements and design stage of projects and should be proportionate to the information security risks"[3]. The most important data to log are input and output validation failures, authentication/authorization success and failures, session management failures, application errors and system events, logging initialization and so on.
In addition, to fulfill the collection of data, other events can be considered interesting, as follow: sequencing failure, excessive use, modifications to configuration, criminal activities, suspicious behaviors and so on. More specifically, "the [...] logs must record "when, where, who and what" for each event"[3]:

- When:
  - Log date and time
  - Event date and time
- Where:
  - Service
  - Geolocation
  - Code location
- Who:
  - Source address
  - User
- What:
  - Type of events
  - Security event

– Security relevant event flag
– Description

Additionally, it could be useful to record also secondary time source event, object affected by suspicious behaviors, internal classification and external one, the action performed and some extended details. Until now, only what is good to log has been introduced, but effectively a rule must be followed: "never log data unless it is legally sanctioned"[3]. It means that the following elements should not be collected directly in the logs: application source code, session identifier, access tokens, sensitive data, password, database connection string, cryptographic values, payment cardholder data, data of high-security classification, commercial data and, generally, information which is not legal to gather. Sometimes, it could happen that examples of the latter elements could be collected for investigation matters but they must be treated in a particular way before being recorded. For OWASP, the next phase of the collection process is based on the verification that everything is working well, such as the logging process is working correctly also during fuzz, penetration and performance tests and it does not cause any type of side-effect. In addition, the just cited process should work even in case of the external network is down and also a test about what happens in case of logging failures must be done. Moreover, the verification continues testing the events are classified in the proper way and in agreement with the chosen standard, verifying access control on the event log data and checking if the logging process can be used to cause a denial of service attack.
"The logging mechanism and collected event data must be protected from misuse"[3]: protection is needed in two different situations when data is in transit and where it is stored. In the former case it could be useful to use a secure transmission protocol in case of untrusted networks and to verify the event data origin. While, in the latter case, when data are at rest it seems to be better to build in tamper detection in order to understand if a record has been modified or deleted, storing and/or copying log data to read-only destinations, recording all the accesses to the logs and restricting the privileges to read log data. Event data are very useful for defense purposes, so it could happen that they become targets of attacks. To prevent security issues confidentiality, integrity, availability, and accountability must be achieved. "A confidentiality attack enables an unauthorized party to access sensitive information stored in the logs"[3]. In addition, it is preferable to check who can modify log data, for example, an attacker whit read

access can exfiltrate secrets. Differently, when availability is lost, downtime is expected and, for example, an attacker can flood log files to exhaust disk space available both for logging and other purposes and he can also destroy all the present logs. When an attack occurs, it must be possible to identify who causes the loss of accountability even if the attacker can be able to prevent log writes or log damages to cover what he does or he can use a wrong identity to be logged such that he is considered out of responsibility. Following the OWASP definition, here is a list to detail the important events to monitor: authentication login success and failure, authentication password changes, authorization failures, and changes, input validation failures, malicious activities, sensitive management, operating system actions, user management and so on.

1.2 Intrusion Detection System

Logging must be used "to identify activity that indicates that a user is behaving maliciously"[4], so the logging process is totally necessary for the intrusion detection. Moreover, "NIST describes the intrusion as an attempt [...] to bypass the security mechanisms of a computer or network"[5] and the "intrusion detection is the process of monitoring the events occurring in a computer system or network, and analyzing them for signs of intrusions"[5]. According to Debar et al. (1999), an "intrusion detection system dynamically monitors the actions taken in a given environment, and decides whether these actions are symptomatic of an attack or constitute a legitimate use of the environment"[6]. More in detail, the "intrusion detection is the act of detecting actions that attempt to compromise the confidentiality, integrity or availability of a system/network"[7]. In a similar way to the previous definitions, Garcia-Teodoro et al. (2009) define Intrusion Detection Systems as security tools that are useful to strengthen the security of information and communication systems. According to Patcha and Park (2007), the "generic architectural model of an intrusion detection system contains the following modules"[7]:

- Audit data collection: "data is collected and analyzed by the intrusion detection algorithm in order to find traces of suspicious activity"[7]

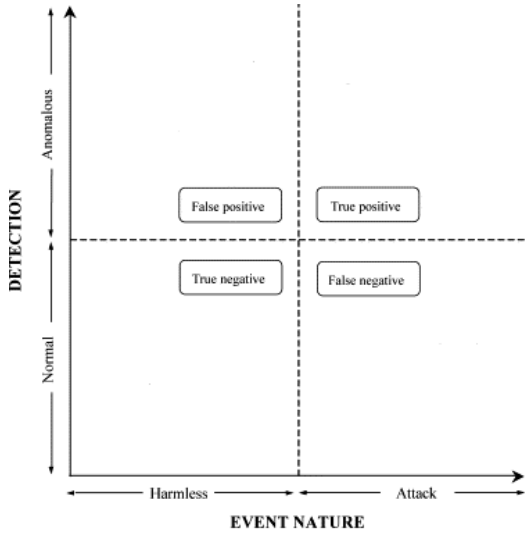

Figure 1.1: Possible cases during a detection

- Audit data storage: the audit data usually are stored for a long time due to later reference[7]
- Analysis and detection: "the processing block is the heart of an intrusion detection system"[7]. Here the algorithm detects if some suspicious activities happen.
- Configuration data: they are useful for configuring the intrusion detection system, describing how to gather data and respond, etc.
- Reference data: these are "information about known intrusion signature [...] or profiles of normal behavior"[7]
- Active/processing data: intermediate results
- Alarm: the output of the intrusion detection system, and it could also be an automated response to an intrusion

Differently, according to Garcia-Teodoro et al. (2009), a general IDS architecture is composed of four functional types of the box: event box, database box, analysis box and response box. Event boxes are the sensors that monitor the target system, acquiring information about events to be analyzed. Instead, a database box is used to store information from the previous Event boxes

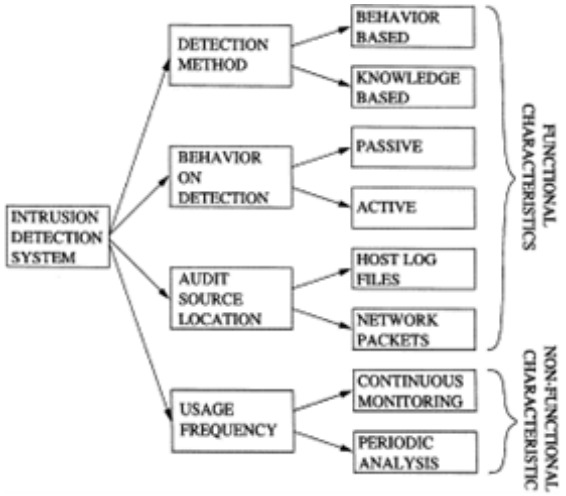

Figure 1.2: Characteristics of intrusion-detection systems

for subsequent processing. The analysis boxes are processing modules used to analyze events and detect potential hostile behavior, so that some kind of alarm will be generated if necessary. Finally, the response boxes aim to execute a response if an intrusion occurs. “A common drawback of IDS technologies is that they cannot supply absolutely accurate detection. False positives (FP) and false negatives (FN) are two indicators to assess the degree of accuracy”[5]. From Figure 1.1, Juan et al. (2004) describe the result of detection in four different categories: false positive, true positive, true negative and false negative. The false positive result is the true limiting factor in the detection because the detection is incorrect in the case of a harmless event. Moreover, a "side effect of false positives, is that an attack or malicious activity on the network/system could go undetected because of all the previous false positives”[7]. Differently, the true positive results from a correct operation made by the detector. Similarly, the true negative is a good result because, as the true positive, it indicates that the detector is working correctly. Finally, another negative result is the false negative because it is the real challenge of every detector: when the false negative rate is very low, it means that the detector identifies every attack.
According to Debar et al. (1999), to evaluate the efficiency of an intrusion detection system three different measures could be used: accuracy, where inaccuracy “occurs when an intrusion-detection system flags as anomalous or intrusive a legitimate action in the environment”[6], a performance which describes “the rate at which audit events are processed”[6] and in case of low

performance, the intrusion detection cannot be labeled as real-time and the completeness which decreases when the intrusion detection system is not able to detect attacks. Going deeper into the evaluation of intrusion detection systems, two additional properties must be presented: fault tolerance because the intrusion detection systems must be resistant to attacks, particularly denial of service, otherwise, intrusions cannot be detected and timeliness, because the system "has to perform and propagate its analysis as quickly as possible to enable the security officer to react before much damage, has been done"[6]. Following Figure 1.2, the Intrusion Detection Systems can be classified in four different ways: detection method, behavior on detection, audit source location and usage frequency.
About the first discriminant, three different categories are present: Signature-based Detection (also named Knowledge-based Detection), Anomaly-based Detection (also known as Behavior-based Detection), and Stateful Protocol Analysis.

1.2.1 Knowledge-based detection

For Garcia-Teodoro et al. (2009), the intrusion detection system contains information about specific attacks and system vulnerabilities and looks for some attempts to exploit them. The technique "relies on a predefined set of attack signatures" [7]. Moreover, the intrusion detection system tries to match incoming packets and/or command sequences like known attacks to find specific patterns. According to Patcha and Park (2007), decisions are based on the knowledge acquired from the model of intrusive process and after observing what the traces left in the system are. In other words, "signature-based schemes provide very good detection results for specified, well-known attacks"[8]. In agreement with Debar et al. (1999), if on the one hand, the accuracy of this technique can be considered good, on the other hand, the completeness depends on how frequently the knowledge of attacks is updated. This approach has many benefits: it has a low possibility of encountering false alarms, the contextual analysis made by the intrusion detection system is detailed and it detects "previously unseen intrusion events"[8]. About the drawbacks, the difficulty of gathering information about known attacks increases and "maintaining the knowledge base of the intrusion-detection system requires careful analysis of each vulnerability and

it is, therefore, a time-consuming task"[6]. Furthermore, this approach is more related to a specific environment; in addition, "the signature detection system must have a signature defined for all of the possible attacks that an attacker may launch"[7]. Another drawback is the inability to detect new intrusions, even if these are variants of already known attacks and a little understanding of states and protocols.

Expert Systems (Rule based)

These systems contain a set of rules that describes attacks. "Rule-based languages are a natural tool for modeling the knowledge that experts have collected about attacks"[6]. This means that a "systematic browsing of the audit trail in search of evidence of attempts to exploit known vulnerabilities"[6] is required. Moreover, the rules should also describe the goal of the attacker and the action required to reach these goals in a deeper way. According to Garcia-Teodoro et al. (2009), "expert systems are intended to classify the audit data according to a set of rules involving three steps"[8]: starting from the training data, different attributes and classes are identified; then, a set of classification rules, parameters or procedures are deduced and finally, the audit data are classified properly. Nevertheless, the usage of rule-based languages encounters some limitations: knowledge engineering and processing speed. The former comes from the difficulty of extracting knowledge about attacks and, in addition, translating the knowledge into production rules using audits as input is even more difficult. Instead, the latter is related to performance issues: "use of an expert system shall require that all audits be imported into the shell as facts, and only then can reasoning take place"[6].

Signature Analysis

This approach is very similar to the expert system one, but "the knowledge acquired is exploited in a different way"[6]: "the semantic description of the attacks is transformed into information that can be found in the audit trail in a straightforward way"[6]. As well as the expert systems, basically the drawbacks are the same; however, in this approach, all the possible facets of

the attacks must be represented as signatures.

State-transition

This technique can be considered conceptually similar to model-based reasoning: "it describes the attacks with a set of goals and transitions, but represents them as state-transition diagrams"[6]. With this approach, the model behavior is captured in three different elements: states, transitions and actions. When a deviation from the expected behavior occurs, such as the transition to an unexpected state, an alarm is triggered. According to Garcia-Teodoro et al. (2009), this approach is very useful for modelling network protocols.

1.2.2 Anomaly-based detection

This detection assumes that "an intrusion can be detected by observing a deviation from normal or expected behavior of the system or the users"[6]. In agreement with Patcha and Park (2007), for the distinction between normal and expected behavior a model must be created: it is a baseline profile of the normal system, network, or program activity. "Thereafter, any activity that deviates from the baseline is treated as a possible intrusion"[7]. Otherwise, there is another possibility: "to model the abnormal behavior of the system and raise an alarm when the difference between the observed behavior and the expected one falls below a given limit"[8]. Furthermore, Liao et al. (2013) describe the anomaly as a deviation from a known behavior. In other words, anything which does not fit with a previously learned behavior can be considered an intrusion. Differently, according to Juan et al. (2004) the application of anomaly-based intrusion detection consists of the following hypothesis: "to assume that anomalous events are suspicious from a security point of view"[9]. If, from a side, the completeness increases, the accuracy becomes lower. Moreover, "the analysis of user activity is a natural approach to detect intrusions"[9]. To Juan et al. (2004), the former hypothesis, also called the Suspicious Hypothesis, cannot be always satisfied due to the differences between normal/anomalous and harmless/attack events. In more detail, there is an important statement: the intrusive activity is a subset

of anomalous activity and, of course, "intrusive activity does not always coincide with anomalous activity"[7]. For Patcha and Park (2007), four different possibilities exist: intrusive but not anomalous which is the case of false negatives because the intrusion detection system reports the absence of intrusions; not intrusive but anomalous which are the false positives reported as intrusive even if they are not; not intrusive and not anomalous which is the case of the true negatives and intrusive and anomalous which are the true positives. In order to minimize the false negatives, "thresholds that define an anomaly are set low"[7]. Some advantages of behavior-based intrusion detection are less dependency on operating-system-specific mechanisms; the capability of detecting abuse of privilege types of attack and to detect insider attacks, for instance, if a user using a stole account can perform actions that are outside of the normal behavior, the system generates an alarm; the difficulty for an attacker to understand what activities trigger an alarm and what do not because the system is based on profiles which are customizable and the capability for the system to detect previously unknown attacks because the profile of intrusive activity is not based on signatures which represent intrusive activity. Instead, the drawbacks user profiles are not accurate due to the high false alarm rate because of normal activities which are not compliant with the behavior profile but at the same time, they cannot be considered as intrusions. Moreover, behaviors can change over time, with the need for periodic online retraining of the profiles during which the system is not available. Furthermore, the intrusion can happen during the training phase and, as result, the modelled profile contains also intrusive behaviors increasing the false negative rate finally the maintenance of the behavior profiles can be time-consuming. The anomaly detection techniques can be classified in three categories:

- Statistical based: in this case the behavior of the system is represented from a random viewpoint

- Knowledge-based: this category tries to capture the behavior from available system data even if it can be assumed to be a knowledge intrusion detection system

- Machine learning based: this one is based on the establishment of explicit and implicit model that allows categorizing the patterns analyzed

Statistical based

In statistical methods, "the user or system behavior is measured by a number of variables sampled over time" and the "time sampling period ranges from very short to long"[6]. Moreover, Patcha and Park (2007) stated that the behavior profiles are generated starting from the activity of subjects measuring the activity intensity, the audit record distribution, the distribution of activities over categories (also called categorical measures), and such ordinal measures. In the statistical approach, two different profiles are created: the current and the stored ones. When the system or network events are processed, the system updates the current profile and then a comparison with the stored one is done calculating the anomaly score. The comparison is done using "a function of abnormality of all measures within the profile"[7]. "The score normally indicates the degree of irregularity for a specific event, such that the intrusion detection system will flag the occurrence of an anomaly when the score surpasses a certain threshold"[8]. Inside the statistical approaches, different models can be presented: univariate model, multivariate model and time series model. In the former, parameters are modelled as independent Gaussian random variables accepting a range of values for every variable. In the multivariate model, the correlation between two or more metrics is introduced: this model brings out a better level of discrimination. While, the time series model uses an interval timer, together with an event counter or resource measure, and notices the order and the inter-arrival times of the events together with their values. "Thus, an observed traffic instance will be labeled as abnormal if its probability of occurrence is too low at a given time"[8]. From a side, the statistical approaches bring out some benefits: no need of prior knowledge about the normal activity of the target system because they learn the expected behavior from the observations and the provisioning of "accurate notification of malicious activities occurring over long periods of time"[8]. For Garcia-Teodoro (2009) an attacker could be able to train the system which does not point out any abnormal activity during an attack, the settings about parameters/metrics are difficult tasks, especially for the balance between false positives and false negatives and not all the variables can be modelled as a statistical distribution, therefore some of these schemes assume a quasi-stationary process, which cannot be considered as a realistic process.

Machine learning based

"Machine learning can be defined as the ability of a program and/or a system to learn and improve their performance on a certain task or group of tasks over time"[7]. The machine learning-based techniques "are based on establishing an explicit or implicit model that enables the patterns analyzed to be categorized"[8]. The behavioral model has to be trained with labelled data and this is the most negative drawback because this procedure demands a lot of resources. For Garcia-Teodoro et al. (2009) the machine learning principles are very similar to the static ones, but the former creates a model which improves thanks to the previous results. Instead, for Patcha and Park (2007), the execution strategy of these systems may change on the basis of newly acquired information.

System call-based sequence analysis Even if the sliding window method is a sequential learning methodology, it "converts the sequential learning problem into the classical learning problem"[7]. A window classifier h_w maps an input window of width w into an individual output value y. The classifier predicts the value $y_{i,t}$ using the following window:

$$< x_{i,t-d}, x_{i,t-d+1}, \ldots, x_{i,t+d-1}, x_{i,t+d} >$$

where $d = \frac{w+1}{d}$.
The window classifier is able to convert each sequential training example (x_i, y_i) into new windows and then the machine learning algorithm is applied. In other words, a new sequence x is classified by converting it to windows; next the window classifier predicts each y_t and then all the y_ts are concatenated to the predicted sequence y. The sliding window method does not take into consideration the correlations between nearby y_t values because the only correlations captured are the ones predicted from nearby x_t values. By way of explanation, if some correlations among the y_t values independent from the x_t values exist, they are not captured. This technique is used in the host-based intrusion detection system where the source of information is the system calls. Because the system calls are irregular by nature, the false positive rate may increase, making difficult the differentiation between normal and anomalous system calls. Furthermore, Patcha and Park (2007) underline how the computational overhead coming from monitoring every system call is very high.

Bayesian Networks The Bayesian network is a model which encodes probabilistic relationships among variables of interest. "When used in conjunction with statistical techniques, Bayesian networks have several advantages for data analysis"[7]: even if data is missing, the networks are able to handle the situation thanks to the interdependencies between variables, casual relationship can be represented, and it is useful to predict the consequences of an action and when there is the need to combine prior knowledge with data, the problems can be modelled thanks to the presence of both casual and probabilistic relationships. Obviously, there are also disadvantages such as the fact that the accuracy of this method depends on the choice of the model: "selecting an accurate model will lead to an inaccurate detection system"[7]. Moreover, "the classification capability of naïve Bayesian network is identical to a threshold-based system that computes the sum of the outputs obtained from the child ones"[7]. The definition of child nodes, are the information nodes pointed by the hypothesis node, also called the root node. In the naïve case, the network is restricted, and it has only two layers and there is complete independence between the information nodes.

Markov Models Within this category, two different approaches can be distinguished: Markov chains and hidden Markov models. In the former case, Ye et al. (2004) builds up an anomaly detection technique based on Markov chains. In their paper, a new window of size N is created to observe the system call event sequences from the recent past starting from the current time t. "For the audit events, $E_{t-N+1}, \ldots, E_t$, in the window at time t, the type of each audit event was examined, and the sequence of states $X_{t-(N-1)}, \ldots, X_t$ appearing in the window was obtained"[10] where X_i is the state that the audit event E_i takes. "The larger the probability, the more likely the sequence of states results from normal activities"[7]. If an attack occurs, the sequence of states is presumed to receive "a low probability of support from the Markov chain model of the normal profile"[7]. In short, the Markov chain can be considered a set of states which are interconnected through transition probabilities. The former determines the topology and the capabilities of the model. "During the first training phase, the probabilities associated with the transitions are estimated from the normal behavior of the target system"[8]. Therefore, the associated probability (anomaly score) obtained from the observed sequence is compared with a fixed threshold in

order to detect the anomalies. About the hidden Markov model, Garcia-Teodoro et al. (2009) write that the system is seen as a Markov process that states and transitions which are hidden and only the productions are observable. In more detail, here the challenge is to determine the states and the transitions from the parameters which are observable: "the variable of the system that is influenced by the state of the system"[7]. For each state, there are two different variables: the probability of producing one of the observable system outputs and the probability which indicates what could be the next state. Each state has a different output probability distribution and the system changes over time; thanks to them, "the model is capable of representing non-stationary sequence"[7]. The aim is to model normal system behavior and for this reason the parameters of the hidden Markov model have to be found: "sequences of normal events collected from normal system operation are used as training data"[7]. Therefore, the detection activity can start thanks to probability measures which are used as thresholds and are compared with test data. For Patcha and Park (2007), three different problems are addressed in this specific model:

- Evaluation problem: given a sequence of observations as input to determine what is the probability that the observed sequence has been generated by the model
- Learning problem: the building up of the model starting from audit data which correctly describes the observed behavior
- Decoding problem: it describes how to determine what is the most likely set of hidden states coming from the observations;

As seen in the previous machine learning techniques, also Markov Models have the same drawback: the usage of resources. For this reason, this "technique for the anomaly detection is not scalable for real-time operation"[7].

Neural networks In according to Garcia-Teodoro (2009), neural networks have been created with the idea of simulating the operation of the human brain and they have been adopted in the field of anomaly intrusion detection. "Neural networks are algorithms that learn about the relationship between input-output vectors and "generalize" them to obtain new input-output in

a reasonable way"[6]. Even if this approach could be used in a knowledge-based intrusion detection tool, "the neural network learns to predict the behavior of the various users and daemons in the system"[7]. The neural network has been used for the creation of user profiles, for the prediction of the next command from a sequence of previous ones and for identifying the intrusive behavior of traffic patterns. According to Debar (1999), this detection approach and statistics have some points in common; therefore, the former is preferred because it expresses in a simpler way the nonlinear relationships between variables and the automation in the learning/retraining of the neural network. The main advantage is the "tolerance to imprecise data and uncertain information"[7] and the capability to deduct solutions even without knowing the regularities in the data. However, this detection approach has several drawbacks. Firstly, the network may fail in identifying a satisfactory solution since a lack of sufficient data or because no learnable function is available. In addition, by the side of Patcha and Park (2007), the training of the system can be slow and expensive; the speed, it depends on the fact that the training data has to be collected and analyzed and also the neural network has to deal with the weights of the individual neurons so that the correct solution can be provided. Finally, the neural networks suffer the lack of a "descriptive model that explains why a particular decision has been taken"[8] and so "the neural network cannot propose a reasoning or an explanation of the attack"[6].

Fuzzy logic The fuzzy logic derives from "fuzzy set theory under which reasoning is approximate rather than precisely deduced from classical predicate logic"[8]. This detection approach has been used for two reasons: some quantitative parameters, used in the context of intrusion detection, can be seen as fuzzy variables and the "concept of fuzziness helps to smooth out the abrupt separation of normal behavior from abnormal behavior"[7]. Indeed, a given data point that is inside or outside a normal interval is defined as anomalous or normal independently from the distance from or within the interval. If on the one hand, fuzzy logic is very effective in port scans and probes, the high resource consumption could be a big problem.

Genetic algorithms They are a "particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and recombination"[8]. In more detail, it is a search technique useful to find approximate solutions to optimization and search problems; moreover, it is based on probabilistic rules instead of deterministic ones. Thus, these algorithms can be used for deriving "classification rules and/or selecting appropriate features or optimal parameters for the detection process"[8]. The main advantages are the capability to converge to a solution from multiple sources and the "flexibility and robustness as global search method"[7]. Here are some approaches using the genetic algorithm as a network-based intrusion detection system: the derivation of classification rules and using them to select appropriate features or to determine the parameters of related functions using different techniques to acquire the rules. As a drawback, high resource consumption is still present also in this approach.

Clustering and outlier detection (data grouping) It is a technique for finding some patterns in unlabeled data with many dimensions. In the clustering algorithm, for Patcha and Park (2007), the outlier is an object which is not placeable in a cluster of a data set and this element become very interesting from an intrusion detection point of view: the outlier can be seen as an attack/intrusion. Moreover, "data points are modeled using stochastic distribution and points are determined to be outliers based on their relationship with this model"[7]. Based on Patcha and Park's (2007) article, to cluster-based anomaly detection, two different approaches exist: both normal and intrusion unlabeled data are used to train the anomaly detection model and only normal data are used to train the model. The former comes from the idea that the attacks are a small percentage of the entire data; therefore, the anomalies and the attacks do not hold in large clusters, but they are outliers; the latter can be used to create a profile of normal activity. In a simpler way, the clustering techniques group the observed data into clusters searching for similarity or for distance measurement. Inside each cluster, a representative point is chosen and then, new data points are classified according to the distance (or the similarity) with this representative point. If some points do not belong to any cluster, they are outliers. As written before, an important discriminant is distance which could be euclidean or Mahalanobis. The first approach does not consider features that could have

very different variability or which could be measured in different scales. Features with large scales of measurement or high variability dominate the other ones. For this reason, the Mahalanobis distance has been proposed and it uses "group means and variances for each variable, and the correlation and covariances between measures"[7]. Moreover, in this scheme a threshold is computed in according to the most distant points from the mean of the normal data and, at the same time, it is a user-defined percentage of the total number of points. If a data point has a distance greater than this threshold, it is an outlier.
As an advantage, the clustering techniques "determine the occurrence of intrusion events only from the raw audit data, and so the effort required to tune the IDS is reduced"[8]; Patcha and Park (2007) write that it depends on whether the clustering is able to learn from audit date without the need of explicit description of the attacks.

1.2.3 Additional Approaches

Here is the description of two different techniques which are not directly related to the intrusion detection system, but they are auxiliary schemes.

Principal Component Analysis

PCA is a dimensionality reduction technique created due to the extension and complexity of datasets. In mathematical terms, PCA makes a translation: n correlated random variables are represented as $d \ll n$uncorrelated variables which are "linear combination of the original variables"[7]. In general, the transformation is divided into two steps:

- Linear combination of the original variables with the largest variance. This step is the "projection on the direction in which the variance of the projection is maximized"[7]

- Linear combination of the original variables with the second largest variance and orthogonal to the first step

Association Rule Discovery

This technique is used to obtain correlations between features extracted from the training dataset. In more detail, given a database D of transactions where each transaction $T \in D$ denotes a set of items in the database, an association rule is the following implication:

$$X \Rightarrow Y,\ where\ X \subset D,\ Y \subset D\ and\ X \cap Y = \varnothing.$$

In this technique, the rule confidence and the rule support are two important concepts to deal with. The probability of rule confidence is the conditional probability $P(Y \subseteq T \mid X \subseteq T)$ and a rule has support s in the database D if $s\%$ of transactions in D contain $X \cap Y$. Instead, the rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if $c\%$ of transactions in X also contain Y.
Furthermore, the association rules can be used "to construct a summary of anomalous connections detected by the intrusion detection system"[7]; for example, as written by Garcia-Teodoro et al. (2009), these rules could be used to find internal relations between data which belongs to a specific connection.

1.2.4 Stateful Protocol Analysis

In according to Liao et al. (2013), another category can be added to the previous ones: the Stateful Protocol Analysis also called Specification-based Detection, an approach which indicates that the intrusion detection system could "know and trace the protocol states"[5]. Even if this approach seems very similar to the anomaly-based one, the Stateful Protocol Analysis depends on "vendor-developed generic profiles to specific protocols"[5]. The advantages of this approach are the knowledge of the protocol states and the capability to distinguish unexpected sequences of command while the drawbacks are the resource consumption due to the protocol state tracing and examination, the inability to inspect attacks that look like normal protocol behaviors, and it could be incompatible with specific operating systems.

1.2.5 Hybrid System

A third solution can be taken into consideration: the hybrid intrusion detection system. According to Garcia-Teodoro et al. (2009), it is a combination of a signature-based detection module with a complementary anomaly-based scheme. In this system, anomaly detection is useful to determine new or unknown attacks while the signature one is good in detecting known attacks. Moreover, if a patient attacker is able to retrain the anomaly detection module so that it accepts the attack as normal behavior, the signature detection is able to detect this kind of attack. Basically, the combination of the two detection techniques "seeks to improve the overall intrusion detection performance of signature-based systems"[8] and, at the same time the usual high false positive rate of the anomalous-based systems decreases. Approaches of the hybrid system have been developed and could be categorized as detection in parallel and detection in sequence. Although the combination of two different detection technologies seems to improve the capability of the intrusion detection system, the result is not always better. Indeed, as written by Patcha and Park (2007), the different technologies examine the system and the network in different ways. Hence, the challenge to creating a hybrid intrusion detection system is about "getting these different technologies to interoperate effectively and efficiently"[7].

1.2.6 Passive versus active intrusion detection

According to Liao et al. (2013), a passive intrusion detection system means that when an attack is detected, only an alarm is generated without any countermeasure applied. Differently, an active intrusion detection system generates scripts "both to suppress the vulnerability [...] and to restore the system in the previous state"[6].

1.2.7 Host-based intrusion detection

A host-based intrusion detection system analyzes all the events which are related to the operating system information such as process identifiers, system calls searching for privilege escalation attempts, unauthorized logins, access

to sensitive files, and malware. It comes from the assumption that the "host-based intrusions leave trails in the audit data"[7]. Moreover, only a host-based IDS is able to analyze end-to-end encrypted communication activities while as limitations it is challenging the detection accuracy due to the lack of context knowledge, there are delays in the generation of alert and in centralized reporting, and if some security controls already exist some conflict could happen.
Furthermore, if an anomaly-based approach is used, the analysis of the user behavior is not accurate because a user could install new programs, changes his working hours, learns new commands, etc. For these reasons, as written by Juan et al. (2004), user profile which is built upon user behavior cannot be accurate and they cause a large number of false alarms.

Host-based intrusion detection sources

Only host audit sources can describe the activities of a user in a given machine but, at the same time, they are susceptible to alterations in case a successful attack happens. Hence, host-based intrusion detection systems have to "process the audit trail and generate alarms"[6] just before the audit trails are subverted or the intrusion detection system crashes. Now, the information sources of the host-based intrusion detection system are described.

System sources This information concerns information of the processes which are currently active on the computer, and they are very precise because the kernel memory is directly examined. As defined by Debar et al. (1999), the only drawback is about the lack of a structured way to collect and store the audit information.

Accounting Accounting provides "information on the consumption of shared resources by the users of the system"[6] where the resources are, for example, processor time, memory, disk or network usage, and applications launched. This information source has many drawbacks:

- The position of the accounting files is the same as the disk partition

of the */tmp* directory; hence if the partition is fulfilled, the accounting stops

- Accounting cannot be activated for selected users
- Timestamp is not so precise: the precision to the second and this does not permit sorting and sequence of the actions because in the accounting file they are logged in the order in which they terminate
- Commands are not precisely identified: only the first eight characters of the name of the command are stored in the accounting record. It means that path information and command line arguments are lost
- Only the applications which terminate are recorded in the accounting file; hence, the daemon activity is not recorded
- The accounting action happens only when the application terminates; thus, the accounting file is updated only when the intrusion has happened

Due to these drawbacks, accounting is not used for knowledge-based intrusion detection and rarely for behavior-based ones.

Syslog Syslog is an audit service provided to applications by the operating system that "receives a text string from the application, prefixes it with a timestamp and the name of the system on which the application runs, and then archives it"[6] in a local or remote way. Even if it is a slight audit source with little audit date per machine, a large network can generate a great number of messages.

C2 security audit Differently from other information sources, security audit records concern "security-significant events on the system"[6]. The idea is to record "the crossing of instructions executed by the processor in the user space and instructions executed in the Trusted Computing Base (TCB) space"[6] which is the kernel, in the UNIX environment. In more detail, TCB is considered trusted and the actions which can impact the system are the services from the TCB; hence, the actions executed in the user space cannot harm the security of the system. Therefore, the record consists of the execution of the system call of all the applications launched by the user, and

context switches, memory allocation, internal semaphores and consecutive file reads are not present. The details which are stored are the user and group identification, the parameters of the system call execution, the return code from the execution, and the error code. About the advantages: the audit events are classified in order to make simpler the configuration of the audit system; the information which has been gathered are parametrized according to user, class, audit event, or failure or success of the system call; in case of error status, usually a run out of the disk, of the audit system, the machine shutdowns and a strong identification of the user are performed. On the other hand, according to Debar et al. (1999), this source information suffers certain drawbacks: in the case of detailed monitoring, heavy use of system resources occurred and in particular, processor performance is reduced by 20% while the requirements for local disk space storage and archiving become higher; if the audit file system is fulfilled, a denial-of-service attack is possible; the set up of the audit service is difficult due to the number of parameters and also because the standard configuration minimizes the performance recording only classes of rare events; moreover, the parametrization concerns too often subjects and actions rather than the objects on which the action is performed; finally, due to the big amount and complexity of the information obtained, exploiting them is difficult: hence, the heterogeneity of the audit system interfaces and the audit record formats of the various operating systems makes the exploitation harder.

1.2.8 Network-based intrusion detection

With the widespread use of the Internet, the host-based intrusion detection systems were not good in the detection of attacks against the network such as DNS spoofing, TCP hijacking, port scanning, ping of death and so on because the host audit trails are not interested in the network area. Therefore, "specific tools have been developed that sniff network packets in real-time, searching for these network attacks"[6]. This technology is "capable to analyze the broadcast scopes of AP protocols"[5]. Regarding the drawbacks, the article of Liao et al. (2013) describes the following ones: the inability to monitor wireless protocols, the high values of false positive and false negative rates, and, in case of high load, the inability to analyze fully the traffic.

Network-based intrusion detection sources

The network-based intrusion detection systems capture network traffic at specific network places using sensors and then, this traffic is analyzed to recognize suspicious incidents. Below is the description of the information sources.

SNMP information The Simple Network Management Protocol (SNMP) Management Information Base (MIB) is "a repository of information used for network management purposes"[6]. This information is routing tables, addresses, names, and some performance/accounting data such as counters which measure traffic at various network interfaces and at different layers of the network, as stated in the article of Debar et al. (1999). However, these MIB counters at higher levels do not contain much more information because not all the correlations between these counters are computed due to their similar behavior of themselves.

Network packets To gather information about the events that occur on the network architecture, an efficient way could be the network and the most efficient way to capture the packets is before entering the server instead of after. Most of the denial-of-service attacks are originated from the network and for this reason, a host-based intrusion detection system cannot detect them. Moreover, the network sniffers could work in two different ways:

- Low-level performing pattern matching, signature analysis and other kinds of analysis of the raw content of TCP or IP packets. Although the analysis is very quick, some session information is lost
- High level working at application gateway: each packet is analyzed with respect to the application, or the protocol followed. Furthermore, this analysis depends also on the particular machine protected because the implementation of protocols is not equal in all the network stacks

However, certain problems may be encountered: difficulty to detect network-specific attacks if not analyzing network traffic instead of audit information on the host; the host performance is impacted by this type of auditing

because the information is collected on a separate machine without knowing the rest of the network; each audit trail format is different from the other ones even if the standardization towards TCP/IP simplifies this problem and when tools analyze the payload of the attack using the signature analysis, it means that for efficient analysis, the tools should know all the types of machines or application for which the packet is intended and it is not feasible. Moreover, in according to Debar et al. (1999), there are also drawbacks: no link between the information contained in a packet and its user source; in the case of switched networks, the placement of the sniffer is not an easy task. The choices are switches or gateways between the protected system and the outside world. The former solution is the best because permits better audit information, but it is costly. Despite these difficulties, the switched networks are less vulnerable to sniffer attacks. Furthermore, in the case of encryption, the payload of the packets is unable to be analyzed and also the obfuscation of the packet contents decreases the comprehensiveness of the signature. Finally, because all these tools are based on commercial operating systems and their network stacks are vulnerable to attacks, they are susceptible to denial-of-service attacks.

1.2.9 Additional Technologies

In addition to the host-based intrusion detection technology and the network-based one, according to Liao et al. (2013), other three types of approaches are possible: Wireless-based IDS, Network Behavior Analysis, and Mixed IDS. Wireless-based IDS is very similar to network-based technology, but it is relative to wireless network traffic. It has the advantage of being more accurate since its narrow focus, but it has also several drawbacks, as stated by Debar et al. (1999): it cannot avoid evasion techniques; the used sensors are susceptible to physical jamming attacks and, if the wireless protocols are insecure, this technology does not mean that they become more secure.
Instead, the Network Behavior Analysis system tries to recognize attacks with unexpected traffic flows but also it is very similar to the network-based approach. If from a side, it has superior detection powers in the reconnaissance scanning, in reconstructing the malware infection and DoS attacks, its detection activity is very slow and not in real-time. The latter has the goal to detect attacks in a complete and accurate way.

1.2.10 Continuous monitoring versus periodic analysis

This distinction is about how the tools perform their analysis. A static intrusion-detection tool "periodically takes a snapshot of the environment and analyzes"[6] it, looking for some vulnerable software, configuration errors or other anomalies. The analysis consists of verifying the version of the applications installed to be sure the latest security patches have been applied, checking if weak passwords are present, verifying the contents of special files or verifying the configuration of open network services. Of course, the snapshot is a description of the system which is valid only in the instant it is taken. This approach has several problems: the security patches could be not present in the legacy systems, and these security assessments are a slow process, in particular when the target is a networked environment where each system has to be analyzed. Moreover, for Debar et al. (1999) between two different snapshots, a vulnerability can be easily exploited; for this reason, the frequency of the snapshots is important.
Differently, a dynamic intrusion-detection tool "performs a continuous, real-time analysis by acquiring information about the action taken on the environment immediately after they happen"[6]. The monitoring phase takes place in real-time or in batches, reviewing the information sources accumulated over a given period of time. Even if, the security of the system improves due to real-time analysis and constant assessment of the security, the dynamic intrusion detection is a costly process both for the transport of the audits and also during their processing.

1.3 Alert correlation

The Intrusion Detection Systems have been deployed for monitoring and defending networks and hosts from malicious attacks. In addition, they can be combined with other preventive security mechanisms. However, these systems suffer from several limitations such as a large volume of alerts outputted every day and a large percentage of the alerts are false positives; real-time applications are inefficient; furthermore, as stated by Elshoush and Osman (2011), certain attacks are not detected by the intrusion detection system, while for Ramaki et al. (2015), inability to detect incomplete events that will

turn in future attacks. Due to these drawbacks, the accuracy of intrusion detection systems must be improved and thanks to alert management and the alert correlation it can be achieved. The management of the alerts has four main objectives: the reduction of false alarms, the comprehensiveness of the cause of the false alerts, the creation of a higher-level view or scenario of the attacks and the creation of responses to attacks by understanding the relationships among the different alarms, as written by Elshoush and Osman (2011). In other words, the alert correlation system is "a system which receives incidents from various heterogeneous systems, reduces the required information for assessment, removes false alerts, and detects high-level attack patterns"[11]. The alarm correlation approaches can be split into three categories: implicit correlation, explicit and semi-explicit ones. The former "uses data-mining paradigms in order to fuse, aggregate and cluster large alert dataset"[12]: this approach, generally, fails to enhance the semantics of the alert. The explicit correlation relies on a language used to specify logical and temporal constraints between alert patterns searching for complex attack scenarios: when a complete or partial attack scenario is found, a higher-level alert is generated. Finally, the latter correlation consists of an association between preconditions and postconditions. In other words, "the correlation process receives individual alerts and tries to build alert threads by matching the preconditions of some attacks with the postconditions of some prior ones"[12]. Differently, in according to Mirheidari et al. (2013), alert correlation algorithms can be divided into three different categories:

- Similarity-based
- Knowledge-based
- Statistical based

1.3.1 Similarity-based Algorithms

The aim is to compute the difference between two alerts or an alert with a cluster of alerts: the goal is "to cluster similar alerts in time"[11]. The advantage is that a precise definition of attack types is not needed; however, only when similarity factors for alert features are defined the correlation is achievable.

Simple rules

The main aspect of this approach is to define simple rules "to express relations among alert features which can be combined together"[11]. The knowledge required is rules structures and information about the similarity between alerts. As a drawback, these rules have some limits in the definition of attack types, and they are able to detect only sequences based on attack class. Moreover, the required memory is "linearly proportional to alert input rate multiplied by the time window length"[11]. However, this is a good algorithm since it has good parallelism capabilities.

Hierarchical Rules

In these algorithms, the rules are expressed in different abstract layers of hierarchy. The alert values are compared with a "linear calculation degree proportional to generalization hierarchy tree depths"[11]. The required knowledge is about the definition of generalization trees while deep network structure and elements knowledge is not required. As before, the required memory is linear but, in this algorithm, it is equivalent to generalization tree sizes.

Machine Learning

In this category, the comparison factors are generated automatically. Machine learning algorithms can be divided into supervised and unsupervised ones. The former uses a set of clustered alerts as prerequisites and this set is used to set the parameters of the decision-making model. While the unsupervised algorithm does not have any type of prerequisites because the measure of similarity is done directly by the algorithm. Three different branches can be described:

- The algorithms cluster alerts based on "decision tree learning by previous data features"[11]. They can be used both for detecting similar alerts and also for attack sequences. Moreover, this branch needs a huge and comprehensive set of training examples for the creation of a decision tree.

Each new alert is compared to all the meta-alerts which exist in the same time window and the comparison is performed with one decision tree. These algorithms are not very flexible and compatible with new conditions because they need to pre-train the decision tree with new data.

- In this branch, the algorithm performs "alert clustering based on alert Reconstruction Error by a neural network"[11]. This category does not need particular previous knowledge except a set of alerts, and, for this reason, the precision is not so high. Regarding processing power and comparison modularity, the algorithm is fast and simple because for each alert the reconstruction error is calculated, and it is independent of the existing meta-alerts. Moreover, the retraining is not so important because it is done only for the alert precision

- The last branch is focused on learning and applying "true and false alert patterns based on labeled data by the system supervisor"[11]. The decision-making is based on searching the similarity among alerts that are inside the same time range. In order to partition the algorithm, the alert observed must be limited to specific clusters. The memory usage is very high due to the partitioning and also to the access of the information related to all the clusters; therefore, even if flexibility is very good, parallelism cannot be achieved

1.3.2 Knowledge-based Algorithms

They are based on a knowledge base of attack definitions, and they can be divided into Prerequisites/Consequences algorithms and Scenario ones.

Prerequisites/Consequences

These algorithms are based on the definition of pre-requisites and possible results; therefore, "each incident is chained to other incidents by a network of conjunction and dis-junction combination and generates the possible network of attacks"[11]. The previous knowledge is based on the description of all existing prerequisites/consequences of alerts and a database with the

network configurations and structure. The outputs of the algorithms can be considered without bias because they are based on real alert meanings. This category has high processing power, low unity and parallelism ability and it requires huge previous data. These drawbacks depend on the fact that when a new alert arrives, any kind of relationship with the other alerts within the same time window must be taken into consideration and, at the same time, the precision improves due to the continuous maintenance and updating of a lot of incidents for a different resource.

Scenario

This category is based on the idea that "many intrusions include various steps which must run one by one to success the attack"[11]. So, low-level alerts are compared with pre-defined intrusion steps and are correlated to find a sequence of alerts related to each attack. The algorithms have to maintain all the scenarios and improve this list when new scenario attacks are discovered. In particular, when a new low-level alert arrives, it is compared with the current scenario and if a certain threshold is exceeded, the alert is attached to the scenario; otherwise, if the alert is below the threshold but it is compatible with another scenario, a new scenario is generated. The scenarios can be described in various languages and, in detail, the specifying attack steps, prerequisites, and goals are described. The required processing resources concern the definition of rules and for this reason parallelization and unification are difficult; however, the flexibility and extendibility improve because "the system behavior must change in real-time according to any change or extension in rules"[11]. The challenge is about defining the attack scenarios even if there exist various automatic attack scenario learning methods.

1.3.3 Statistical-based Algorithms

This category is based on the idea that "relevant attacks have similar statistical attributes and a proper categorization can be found by detecting these similarities"[11]. These algorithms store the causal relationships between

incidents and analyze their frequencies using previous data statistical analysis and then the attack steps are created. After this learning phase, this knowledge is used for the correlation between the attack stages.

Statistical Traffic Estimation

The basic idea of this category is to find repetition and non-similarity patterns in the alerts. Context knowledge is not required because the raw material is the statistics of each alert. The processing load depends only on the statistical model which is used even if some model parameters can be changed at runtime according to new data. One of the applications of these algorithms is detecting alerts that normally occur together, and an important further task is determining the alert priorities. The analysis of alerts is based on time windows: in each time window, "statistical information about all alerts is calculated and the resulting statistics are compared to the previous ones"[11]. In addition, input data can be pre-partitioned and this means that each alert belonging to a unit is processed independently: parallelization improves.

Causal Relation Estimation

The aim of these algorithms is "finding alert sequence or association dominant patterns and using these patterns for detecting false cases, or proper combination"[11]. In other words, the purpose of this category is the creation of a model for determining correlation relationships between alerts. The algorithm can be divided into two steps: training and functional. The former is performed offline with archive data; therefore, the training phase does not affect the practice use. The flexibility of this algorithm is not so high due to the need for the training phase but if the user interferes during the learning of the relationships, the algorithm behavior can change faster.

Reliability Degree Combination

The goal of this category concerns the introduction of an algorithm that combines reliability and completely similar alerts where the reliability is

equivalent to alert repetitions. In other words, the importance/priority of an alert depends on approval by other resources. The idea can be simplified by removing all the processing details and accepting "the amount of an alert repetition as a factor independent from alert importance and resource history"[11]. The Causal Relation Estimation algorithms, these ones work in two phases: training and function. As before, the training phase does not interfere with function one and the algorithm speed during the execution of the order $O(1)$, too. Input data are simply partitioned and therefore parallelism is not needed. The flexibility is slow due to the need of retraining with a lot of data the model every time new conditions are needed; otherwise, "the reliability to different resource opinions can change by the direct interference of the system supervisor and changing the algorithm behavior in real time"[11].

1.4 Case study: Weseth

"Weseth is a [...] platform, that enables the remote verification of the security posture and the effectiveness of the security countermeasures"[13]. Moreover, according to the Drivesec website, remote testing reduces the time of execution of vulnerability assessments and penetration tests and, at the same time, allows the execution of tests in a realistic environment avoiding boring setup time. The architecture, of Weseth could be divided into four different pieces: Weseth Box, Weseth Server, Weseth Web App, and Weseth Client. The Weseth Box is "an embedded device which exports a large number of network interfaces"[13] and it is connected to remote components such as test benches, manufacturing lines, vehicles, and so on. In short, it is directly connected to the IoT component which has to be tested and different protocols as well as Can, Can-FD, WiFi, Bluetooth, Ethernet and USB are available. Furthermore, Weseth boxes are widely distributed around the world and they could easily reach test benches even in case of difficult conditions. The Weseth server implements "the core platform logistic, including:

- Authentication
- Communication multiplexing

- Security
- Testing session logging"[13]

In addition, the connection with the Weseth Box is based on the LTE protocol.
The Weseth Web App is the place where customers who need Vulnerability Assessment and Penetration testing tasks, can find and engage Cyber researchers.
The Weseth client can be downloaded from the Weseth Web App and it is used by the researcher to perform its tasks in a completely remote way. Through it, the researcher can communicate with the test benches: the client is in communication with the server which has a preferential channel with the Weseth Box. Each command which is inserted in the Client is directly sent to the test bench. Among the various use cases, there are: remote access, remote functional testing, remote component reflash, remote analyses, remote diagnostic and repair, remote continuous assessment, and so on.

Chapter 2

Security and Monitoring tools

This chapter aims to describe the state of the art of various intrusion detection systems. In particular, for the architecture introduced in the previous chapter, a host-based intrusion detection system has been chosen because the main objective is to detect and monitor the action performed in the host container. The first part is about the presentation of the discriminant used to decide what is the best tailored HIDS solution for the architecture, then the selected product, Wazuh, is analyzed deeply.

2.1 Discriminants

In this paragraph the discriminants which have been taken into consideration for the decision of the best solution are pointed out:

- Multiplatform: even if the architecture is based on a Linux operating system, this feature has been taken into consideration for future change.
- File integrity: this is one of the most important aspects to consider for the decision of the HIDS. It is a module that detects when any file, directory, or registry is modified, deleted, or added to the file system.

- Log Analysis: As the previous one, this module is useful to analyze the logs which are gathered from the host system. The Log Analysis module is in charge to collect, analyze and correlate the logs to alert if something suspicious is happening.
- Rootkit detection: a rootkit is "a set of procedures or codes which are used to hide the code modules, files, registry entries, etc."[14]. In other words, a rootkit is a set of elements that provides continuous privileged access to a computer and at the same time, its presence is hidden. The chosen intrusion detection system has to detect when a system is modified in a way common to rootkits.
- Active response: even if it is out of scope for the specific use case, this feature could be very useful in a future deployment because it permits taking immediate action when specified alerts are triggered.
- Real-time alerting: it is a feature that permits to receive alerts as soon as possible when critical incidents happen. Of course, the priority of critical incidents is configured by the customer, but real-time alerting is useful to split the regular noise from interesting events.
- Documentation completeness: it describes the quality of available documentation, before its deployment.
- Setup difficulty: this discriminant determines how easy is the setup and/or installation of the various nodes for correct functioning.
- Readable output: it describes if the alert coming from the analysis of the source information is easily readable or not.
- CVEs: this element describes if the solution has some specific CVEs and, in particular, here the interest is on the latest version of the product.
- Agent-based: this discriminant describes the possible architecture that could be implemented in the HIDS. The agent-based monitoring means that the daemons collect logs, and the file system information from the boxes and then the analysis is performed in a centralized management server. Moreover, centralized management permits the of definition policies across multiple operating systems.
- Open source: it describes if the product is a commercial or an open source one.

Discriminants	Logging and Monitoring tools									
	AIDE	EVENTLOG ANALYZER	IBM MaaS360	LACEWORK T. D.	OSSEC	SAGAN	SAMHAIN	SOLARWINDS SEM	TRIPWIRE L. C.	WAZUH
Multiplat.	*No*	*Yes*	*Yes*	*Yes*	*Yes*	*No*	*Yes*	*Yes*	*No*	*Yes*
File Integr.	*Yes*	*Yes*	–	*Yes*	*Yes*	*No*	*Yes*	*Yes*	*Yes*	*Yes*
Log Anal.	*No*	*Yes*	–	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
Rootkit det.	*No*	–	*Yes*	*No*	*Yes*	*No*	*Yes*	–	*No*	*Yes*
Active resp.	*No*	*No*	*Yes*	*Yes*	*Yes*	*No*	*No*	*Yes*	*Yes*	*Yes*
RT alerts	–	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
Doc complet.	*No*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*No*	*Yes*
Setup Diff.	*Yes*	*Yes*	*No*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
Read. out.	*No*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	–	*Yes*
CVEs	*No*	*No*	*No*	*No*	*Yes*	*No*	*No*	*No*	*No*	*No*
Agent	*No*	*Yes*	*Yes*	*Yes*	*Yes*	*No*	*Yes*	*Yes*	*Yes*	*Yes*
OS	*Yes*	*No*	*No*	*No*	*Yes*	*Yes*	*Yes*	*No*	*No*	*Yes*

Table 2.1: Peculiarities of the most known monitoring and logging tools

Here a general description of each HIDS taken from Table 2.1:

- AIDE: it is a "file and directory integrity checker"[15], and it is present in this list just for the file integrity module. It creates a database of files on the system and then the database is used to ensure the integrity and detect system intrusions. However, AIDE is not able to provide

"sureness about change"[15] in certain types of files.

- EventLog Analyzer: it is a "web-based, real-time, log monitoring and compliance management solution for Security Information and Event Management System (SIEM)"[16]. It is very useful to improve network security and perform audits for various regulations.

- IBM MaaS360: it is a "comprehensive mobile device management solution for monitoring and managing [...] mobile devices from a web-based portal"[17]. Moreover, portal administration functions, device management, software distributions, policy self-service, and device compliance functions are also supported by the enterprise mobility management (EMM) platform.

- Lacework Threat Detection: it is a general intrusion detection system that delivers "cloud infrastructure compliance and security for develops, workloads, and cloud containers"[18].

- Ossec: it is "a platform to monitor and control "[19] systems. It is composed of all the aspects of a host-based intrusion detection such as log monitoring and the Security Incident Management(SIM)/Security Information and Event Management(SIEM)[1]; all them create an open source solution.

- Sagan: it is a log analysis engine and it has been designed "with a Security Operations Center (SOC) in mind"[20]. This means that Sagan analyzes logs across many different platforms in many different locations in real-time and with "heavy lifting analysis before putting the event in front of a human"[20]. Moreover, it has also a few peculiarities of correlation engines even if it is very simple.

- Samhain: it is "a file and host integrity and intrusion alert system suitable for single hosts as well as for large, UNIX-based networks"[21]. This solution can be used with the client/server paradigm where hosts are monitored and, then, the central log server collects the events from them.

[1]SIEM technology supports threat detection, compliance and security incident management through the collection and analysis of security events.

- Solarwinds Security Event Manager: it is a SIEM "virtual appliance that adds value to existing security products"[22] increasing efficiencies in the administration, management, and monitoring security policies in networks. Solarwinds SEM allows access to log data both for forensic analysis and for threat analysis looking for suspicious activities and/or anomalies.
- Tripwire Log Center: it is one of the proposed solutions by Tripwire and it is in charge to collect, analyze, and correlate log data from devices, servers, and applications. "Its correlation engine automatically identifies and responds to events of interest using a logical flow of one or more conditions"[23]. Moreover, it allows customers to discover assets never identified through analysis of their log data.
- Wazuh: it is "a security platform that provides unified XDR[2] and SIEM protection for endpoints and cloud workloads"[25]. It is composed by a single universal agent and three components for the central monitoring manager: server, indexer, and dashboard.

From Table 2.1, the solutions with the most researched requirements are Ossec and Wazuh. However, Wazuh has been preferred to Ossec for various reasons. First of all, Wazuh is an extension of Ossec and it maintains many features and configurations from it. Moreover, Wazuh is more up-to-date than Ossec and the new versions of the latter have only bug fixed. Moreover, Wazuh can be developed in different clusters and it can be integrated with cloud providers such as AWS, Microsoft Azure, and so on, too. For these and other reasons, Wazuh is the solution that has been chosen for the Weseth platform.

2.2 Wazuh

At first, an overview of the Wazuh components and architecture is described, then the user manual, the capabilities, and the functionality are pointed out.

[2]"XDR is a security threat detection and incident response tool that natively integrates multiple security products into a cohesive security operations system that unifies all licensed components"[24]

2.2.1 Wazuh elements

Wazuh is composed of four different elements: Wazuh Indexer, Wazuh Server, Wazuh Dashboard, and Wazuh Agent.

Wazuh Indexer

The Indexer is a "highly scalable, full-text search and analytics engine"[25]. Its goal is to index and store the alert generated by the Server and, at the same time, it allows real-time data search with analytics capabilities. This component stores data as JSON documents and each document has the task of correlating a set of keys, field names, or properties with their corresponding values. In short, "an index is a collection of documents that are related to each other"[25]. Moreover, to protect the system against hardware failures, the documents are distributed across containers known as shards and can be distributed across multiple nodes. Wazuh defines four different indices to store different event types:

- Wazuh-alerts: Alert generated by the Wazuh server and created "each time an event trips a rule with a high enough priority"[25]
- Wazuh-archives: All the events received by the Wazuh server
- Wazuh-monitoring: Data about the Wazuh agent status over time
- Wazuh-statistics: Data about Wazuh server performance

The indexer is fast, scalable, and resilient and owns features such as data rollups, alerting, anomaly detection and index lifecycle management.

Wazuh Server

The Wazuh Server "analyzes the data received from the agents, triggering alerts when threats or anomalies are detected"[25]. The server architecture is composed by different elements: agent enrollment service, agent connection service, analysis engine, Wazuh RESTful API, Wazuh cluster daemon and

Filebeat. The agent enrollment service is the one for enrolling new agents by providing and distributing unique authentication keys to each agent. This service supports authentication via TLS/SSL certificates or fixed passwords. The agent connection service receives data from agents and it uses the previously exchanged keys to validate the agent's identity and to encrypt the communication between the agent and the server. Moreover, this service is useful for managing the agent configuration remotely. The analysis engine is the core of the Wazuh server because it is in charge of performing data analysis. The analysis is performed thanks to two different items: decoders and rules. The decoder is useful for identifying the type of information being processed and extracting relevant data elements from the log messages. Then, through the use of rules, the engine tries to identify specific patterns in the decoded data that could trigger alerts and, in case of dangerous ones, an automated countermeasure could start. The Wazuh RESTful API is a service that allows endpoints to interact with the Wazuh infrastructure allowing configuration settings of agents and servers, monitoring the status of the infrastructure, managing and editing decoders and rules, and querying the state of agents. The Wazuh cluster daemon is used when Wazuh servers are scaled horizontally and deployed as a cluster. Filebeat is "a lightweight shipper for forwarding and centralizing log data"[26] and it is used to forward events and alerts to the Wazuh indexer. Moreover, it provides load balancing when the indexer is clustered.

Wazuh Dashboard

The dashboard is "a flexible and intuitive web user interface for mining, analyzing, and visualizing security events and alerts data"[25]. Moreover, thanks to the dashboard, the entire Wazuh platform can be managed and monitored. The access to the dashboard could be restricted with role-based access control (RBAC) and single sign-on (SSO) mechanisms. Detailing them, Wazuh dashboard features are data visualization and analysis, agents monitoring and configuration, platform management, and developer tools. The former service represents the web interface for the visualization of different types of data collected and the corresponding alerts generated by the Wazuh server. The agents monitoring and configuration is the service through which the status of the agents can be monitored. In general, the platform can be managed directly from the dashboard, configuring the Wazuh

server, creating custom rules and decoders, and monitoring the status of the different Wazuh components. In the end, the developer tools are useful to process log messages to check how they are decoded and, in the case of custom rules, if they are matched or not.

Wazuh Agent

The agent is in charge to protect the system and it provides various services such as threat prevention, detection and response capabilities. Moreover, it is used to collect source information such as security-related events, operational data and information about its status and configuration from the system and forwards them to the Wazuh server. The communication between the agent and the server "takes place through a secure channel (TCP or UDP)"[25], which provides data encryption and compression in real-time. At the same time, the channel provides flow control mechanisms to avoid flooding and queueing events when they increase a lot and protection for the network bandwidth. Moreover, the Wazuh agent has a modular architecture, that is, each module performs only one action and it can be configured independently from the other ones. Following, a description of each module is provided.
The log collector is in charge to read log files and Windows events, "collecting operating system and application messages"[25]. Moreover, JSON events can be enriched with additional data. According to the Wazuh documentation, the command execution allows the agent to run commands periodically, collect outputs, and report them to the Wazuh server for further analysis. One of the most important modules is File Integrity Monitoring (FIM) which is in charge to monitor the file system and reporting if new files are added or if already present ones are deleted or modified. Moreover, the FIM reports file attributes, permission, and ownership changes. When one of these events happens "who, what, and when details"[25] are captured in real-time. Another module is the one in charge to perform the security configuration assessment (SCA) which provides "continuous configuration assessment, utilizing out-of-the-box checks based on the Center Internet Security (CIS) benchmarks"[25]. Furthermore, the agent performs also a system inventory action that consists of periodic scans, and collection of inventory data such as the version of the operating system, network interfaces and so on. The malware detection element uses a non-signature-based approach in order to detect anomalies and eventual rootkits. It is also able to search hidden processes, files and hidden

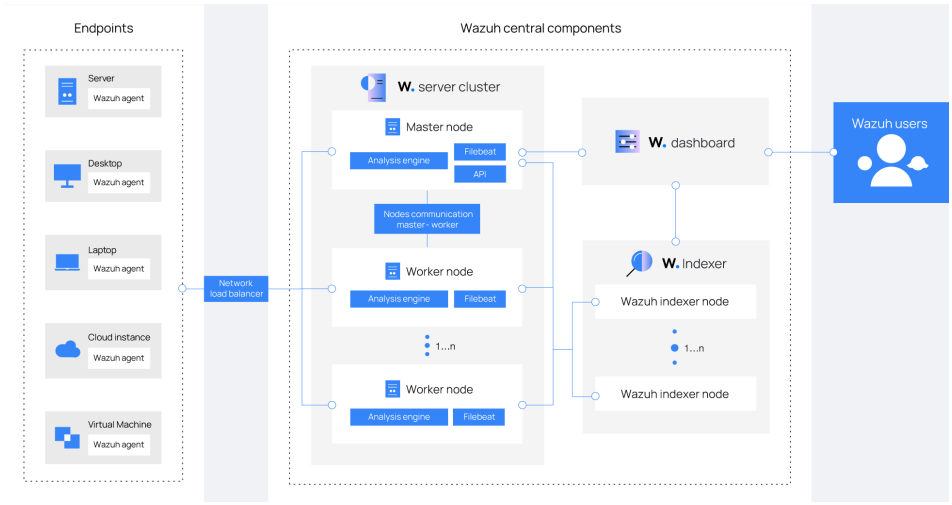

Figure 2.1: Description of Wazuh Architecture

ports during the system calls monitoring. Another block is the active response one and it "runs automatic actions when threats are detected, triggering responses"[25] which could be blocking a network connection, stopping a running process or deleting a malicious file. Container Security monitoring is the element that allows monitoring changes in containers. The last module is the one related to monitoring cloud security thanks to the detection of changes in the cloud infrastructure and the collection of cloud services log data. All these modules are customizable for the user's needs through the introduction of new security policies, responses and so on.

2.2.2 Architecture

In short, the Wazuh architecture is based on "agents [...] that forward security data to a central server"[25]. Right after, the Wazuh server decodes and analyzes the received security data and passes the results to the Indexer which is in charge of index and storing. Moreover, the results are forwarded by Filebeat which uses TLS encryption. More precisely, Filebeat "reads the Wazuh server output data and sends it to the Wazuh Indexer"[25]. From Figure 2.1, it seems that the Server and the Indexer are deployed in different nodes. It happens when many endpoints should be monitored together with a large volume of data anticipated or in case of high availability is needed. As

written before, when the server receives security data, it has to decode and rule-check the received events through the analysis engine. When data are indexed by the indexer, "the Wazuh dashboard is used to mine and visualize the information"[25].

An important aspect to take into consideration is how alerts and log data are managed by Wazuh. If from a side, rotation and backups of archive files are in charge of the user, Wazuh offers the possibility to rely completely on the storage actions performed by the Wazuh indexer and this alternative is preferred in case of multi-node architecture.

2.2.3 User manual

In this section, the most important aspects of the entire Wazuh platform are described.

Wazuh server Administration

As written before, the Wazuh server receives data from agents and triggers alerts when an event matches a rule but, at the same time, the "manager also works as an agent on the local machine"[25] and it means that all the modules present in the agent are also present in the server. When the server receives an event, it has to assign a "severity level depending on which rules it matches from the ruleset"[25] and the default behavior is to log alerts with a severity level higher than three. Of course, this severity threshold could be changed according to necessities.
Furthermore, the Wazuh server allows to be integrated with external APIs such as Slack which is "a simple way to post messages from 3rd-party apps"[25], PagerDuty which is an incident response platform, VirusTotal which is able to inspect the malicious file and also other external software. The server could be configured to send alert to Syslog, it can output the alerts into a database, generates daily reports about the alerts triggered each day, and it could send email alerts to one or more email addresses; the email alerts could be sent in generic and granular ways such as based on alert level, alert level and agent, rules identification or based on rule group.

Wazuh agent enrollment

It is the process "of registering Wazuh agents as authorized members of the Wazuh solution"[25]. This process allows the Wazuh manager to register agents and generate unique keys for them which are also used to encrypt the communication between agent and manager and to validate the identity of the agents. The enrollment may happen via manager API or via agent configuration. In the former way, a unique key is generated from the server and it has to be manually imported to the agent. Differently, when the manager API is used, the agent is enrolled after the IP address of the Wazuh Manager has been inserted. This processing method is very risky and for this reason, additional security options could help to secure it and these are password authentication, manager identity verification, and agent identity check. In the former way, the password is used to ensure the agents enrolled with the Wazuh manager are authenticated. The other two approaches need a certificate authority that signs certificates both for the Wazuh manager and agents and, if an already configured one is not present, the manager could act in its place. For the manager identity validation, the root certification authority certificate has to be copied to the endpoint while for agent one, the certificate and the corresponding key should be created from the manager and subsequently copied to the endpoints; the agent verification has two options: the one with host validation specifying the agent host name or IP address or the one without host validation where the same certificate could be used on multiple agents because no hostname or IP address is specified. Of course, the latter is insecure.

Agent management

After the agent installation, it must be registered with the server in order to establish communication and this is the enrollment process. In addition, an agent remains registered until it is removed. An agent could have four different states as shown in the image below.
In Figure 2.2 the four states are shown: never connected, pending, active, and disconnected. The former is when the "agent has been registered but has not yet connected to the manager"[25]. The pending state happens when the authentication process has not already finished due to a lack of information

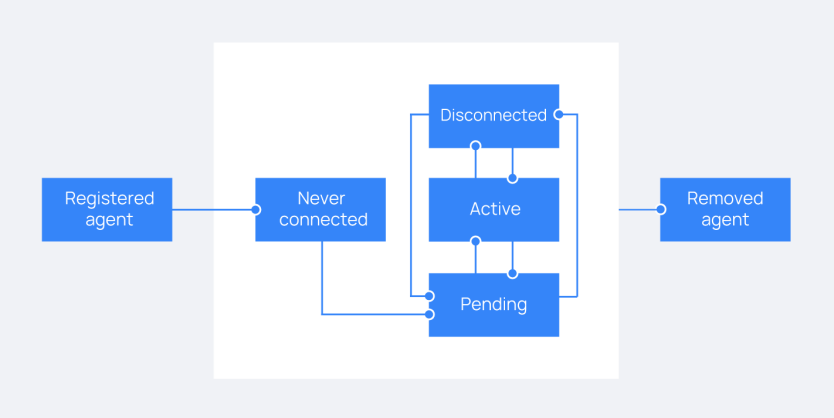

Figure 2.2: Wazuh Agent life cycle

from the agent itself. This state persists until the agent sends the other requested information to the manager. The active state means that the agent has connected to the manager while the disconnected one means that the manager has not received any *keep alive* message from the agent.

Deploying a Wazuh cluster

By Wazuh's definition, the cluster "is made up of manager type nodes"[25] where one of them is the master node while the others become worker nodes. In particular, in a cluster, the managers work together "to enhance the availability and scalability of the service"[25]. Just the latter represent the reasons for using a cluster: the horizontal scalability multiplies the capacity of processing the events while in case a server fails, high availability is achieved thanks to the various nodes. The master node "centralizes and coordinates worker nodes"[25] and, at the same time, allows the maintenance of critical and required data consistent across all nodes. Moreover, it is in charge of agent registration and detection, and synchronization of rules and decoders. Differently, the worker nodes are able to synchronize integrity files from the master node, send agent status updates to the master, and redirect agent enrollment requests to the master.

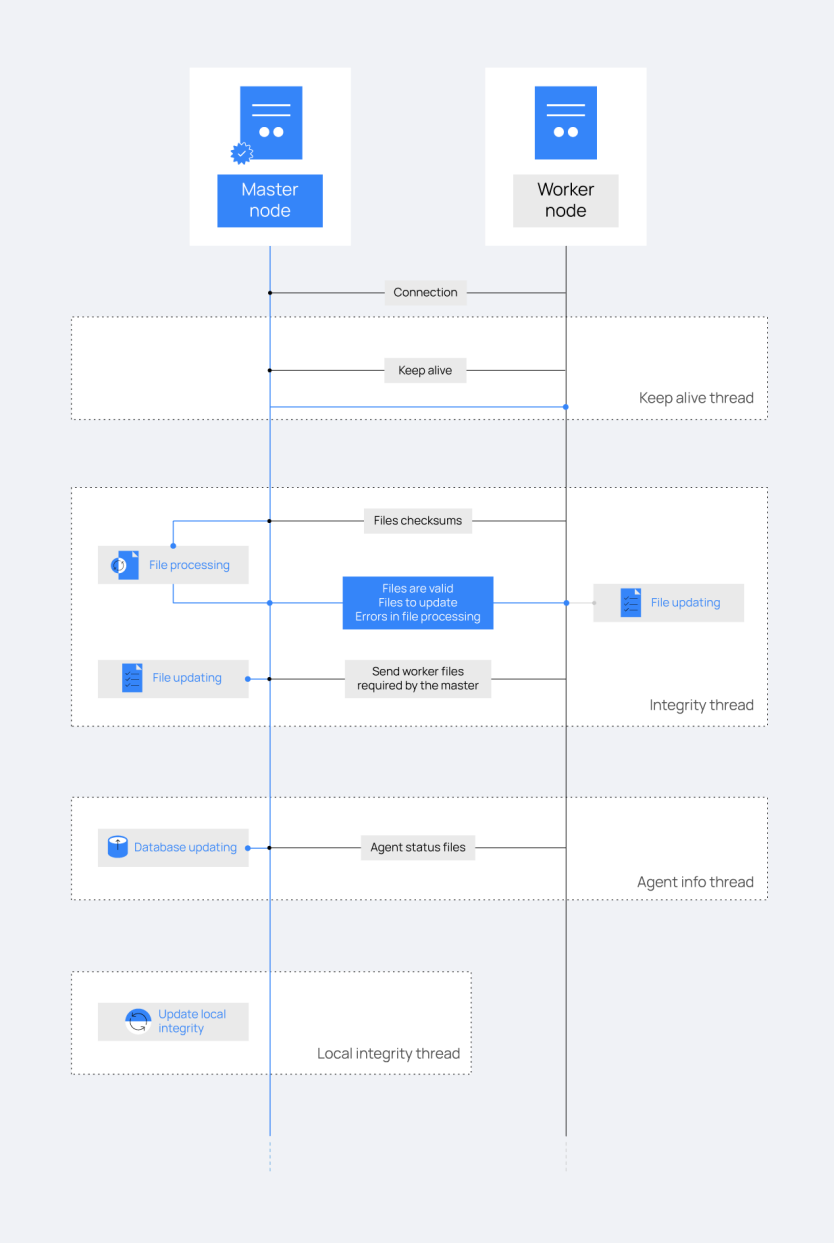

Figure 2.3: Wazuh cluster

How the cluster works "The cluster is managed by a daemon [...] that communicates with all the nodes following a master-worker architecture"[25].

Figure 2.3 shows the various communications between the master and the

worker nodes, and the different independent threads running in the worker: keep-alive thread, agent info thread, integrity thread, and file integrity thread. The former "sends a keep-alive to the master every so often"[25] and it is necessary to not close the connection between the master and the worker. The agent info thread is in charge to send OS information and the statuses of the agents which report to the specific worker node. Moreover, the master should verify if an agent exists or not before saving useless and unnecessary information. Differently, the integrity thread should synchronize the files sent by the master to the workers. The files exchanged concern the Wazuh agent keys file, custom rules and decoders, and agent group files and assignments. The file integrity thread is in charge to check the integrity of each file calculating its checksum and modification time. This thread is run by the master node in order to avoid the calculation of the integrity of each worker connection.

2.2.4 Capabilities

This section contains a detailed explanation of how the following capabilities work: log data collection, file integrity monitoring, auditing who-data, anomaly and malware detection, security configuration assessment, monitoring security policies, monitoring system calls, command monitoring, active response, agentless monitoring, anti-flooding mechanism, agent labels, system inventory, and vulnerability detection.

Log data collection

It is the "real-time process of making sense of the records generated by servers or devices"[25], that is, it receives logs from text files, Windows event logs, or directly via remote Syslog. The goal of log data collection is the "identification of application or system errors, misconfigurations, intrusion attempts, policy violation, or security issues"[25].
From Figure 2.4, in particular, in the analysis part, the analysis process is described in detail and it can be divided into three sub-processes: pre-decoding, decoding, and rule matching. In the former step, "static information from well-known fields"[25] are extracted from log headers. In the decoding

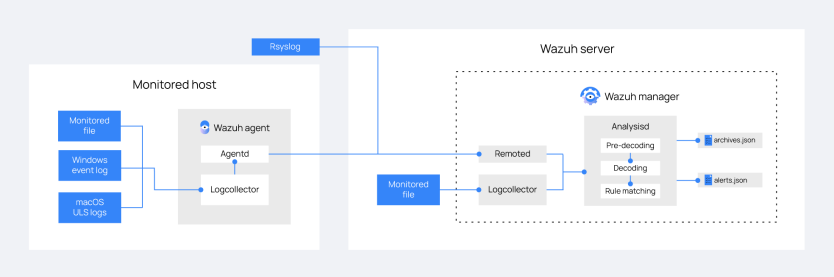

Figure 2.4: Wazuh Log analysis flow

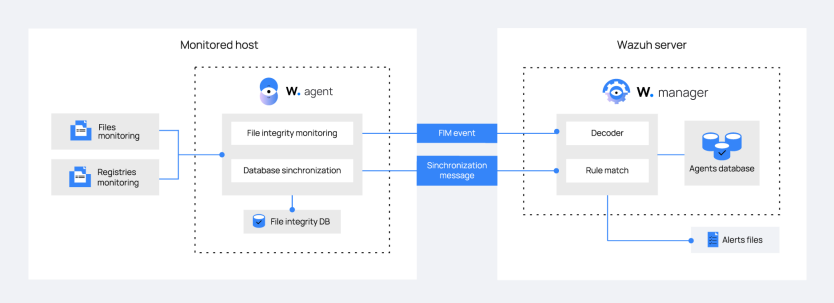

Figure 2.5: Wazuh File integrity monitoring

phase, the message is dissected in order to identify what type of log it is, and then extraction of specific fields is performed. The last sub-process consists of the comparison between the extracted log information and the rules looking for matches. Furthermore, if one or more rules are matched, the manager creates corresponding alerts.

File integrity monitoring

The FIM system "watches selected files and triggers alerts when these files are modified"[25]. As shown in Figure 2.5, the FIM module is located in the Wazuh agent and it runs a periodic scan of the system storing checksums and attributes in a local database. Thereafter, the comparison between new files' checksums and old ones is performed and if modifications are detected, they are reported to the Wazuh manager. Moreover, not only do agents have information about their files but also the Wazuh manager has its file inventory always updated. The FIM synchronization "is based on periodic calculations of integrity between the Wazuh agent and the Wazuh manager

database"[25] and only modified files are updated in the manager decreasing the data transfer. Every time an agent is restarted, FIM database is clear and, of course, if modifies happen when the Wazuh agent is not running, they are not reported to the manager as well as the updates which occur after the last scan and before the restart. Some of the information gathered from the FIM module are path, size, hard links, FIM event mode, file permissions, user ID of the owner of the file, group ID of the group that shares the ownership, user name and group name of the owner, MD-5, SHA-1 and SHA-256 hashes of the file, timestamp of the file changes, the inode of the file, the changes in the files, the processor that triggers the event, and so on.

Auditing who-data

This capability permits us to know "the user who made the changes on the monitored files and the program name or process used to carry them out"[25]. This information, as the FIM one, are processed by the *syscheck* module and reported to the manager. The alert contains different information such as the ID, the name, the Audit user ID and name during the login, the group ID and the group name of the user who ran the process that modified the monitored file, and the ID, and the name and the parent process ID of the process that modifies the file.

Anomaly and malware detection

It refers to the "act of finding patterns in the system that do not match the expected behavior"[25]. When malware is installed, it modifies the system to hide from the user and, even if it uses different techniques to achieve this, Wazuh owns a broad spectrum of approaches to finding anomalous patterns about possible intruders.
From Figure 2.6 it is clear how the *rootcheck* module is the main component for this task, however, also the *syscheck* module has an important role. The latter permits the detection of directories that have been changed and, in particular, it can check if the malware has replaced file, directories, and commands. Although that, *rootcheck* is the most important module because it permits to check running processes, hidden ports, unusual files and

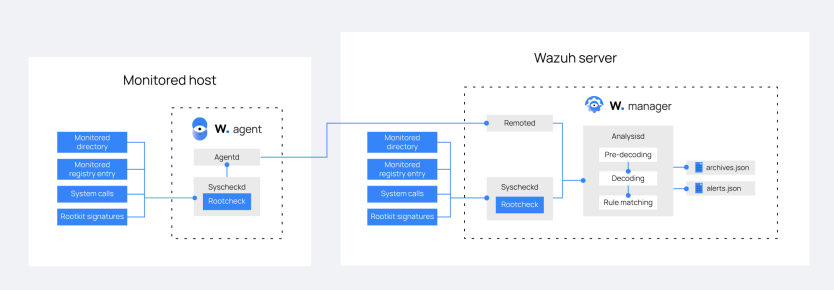

Figure 2.6: Wazuh intrusion and anomaly detection

permissions, hidden files using system calls, to scan */dev* directory, network interfaces. and to perform rootkit checks. About the running processes, it could happen that a malicious one prevents itself from being seen in the system's list of processes and *rootcheck* inspect all the PIDs (Process Identifiers) "looking for discrepancies with different system calls (getsid, getpid)"[25]. In addition, the module has to check every port in the system because malware can use hidden ports to talk with the attacker. *Rootcheck* tries to bind each port with the *bind()* function; if it is not possible and, at the same time, the port is not in the *netstat* output, malware may be present. The *syscheckd* module is useful to look for files owned by the root but with write permissions for other users or hidden directories and files. At the same time, Wazuh searches differences between the *stat size* and the file size performing *fopen* + *read* calls. Moreover, hidden files could be found thanks to the comparison between the number of nodes in each directory and the output of *opendir* + *readdir*. If the results of the comparisons do not match, malware could be present. The */dev* directory is the container for some hidden files even if it should contain only device-specific files. The scanning of network interfaces could be useful to check if one of them is in promiscuous mode which is an indicator that malware could be present. Finally, the rootkit checks are performed by comparing the file system with a database of rootkit signatures. One problem with this feature is the lack of updated rootkit signatures.

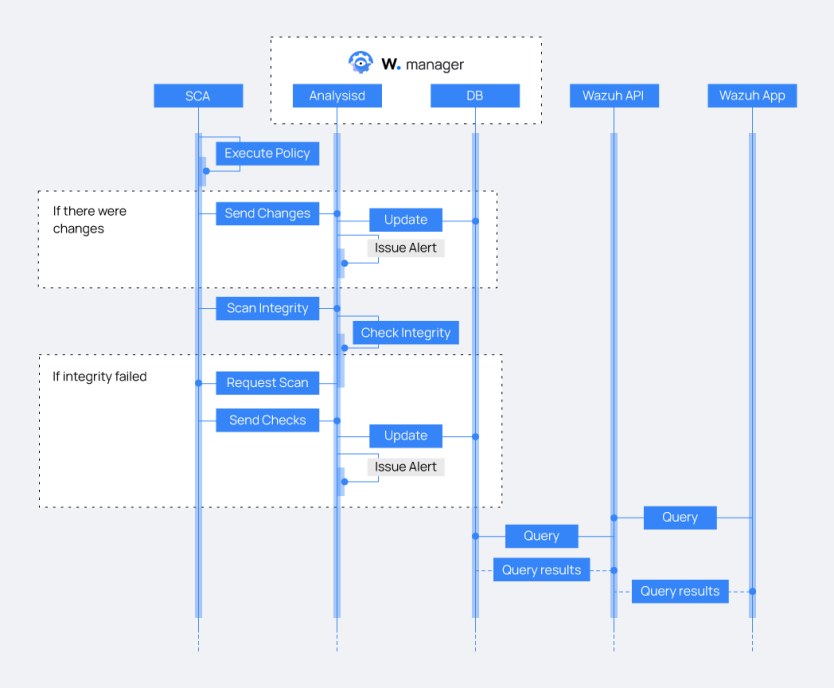

Figure 2.7: Wazuh SCA integrity and alerting flow

Security Configuration Assessment

The Security Configuration Assessment (SCA) has been created by the Wazuh team to overcome limitations such as the license needed to use CIS-CAT tool and the dependence of *rootcheck* module from the *syscheck* daemon that have policies feed often outdated. The principle goal of the SCA module is to provide the best user experience performing scans about hardening and configuration policies. Moreover, the module includes the possibility to store the state of the last SCA scan which is stored in the Wazuh manager, to create new alerts if SCA states change, and to use YAML format for policy files.

The SCA is a way "to determine opportunities where hosts could have their attack surface reduced"[25]; indeed, SCA performs scans looking for exposures and misconfigurations in monitored hosts. "Those scans assess the configuration of the hosts using policy files that contain rules to be tested against the actual configuration of the host"[25]. Examples of SCA assessments are: when it is necessary to change passwords, remove unnecessary

software, and so on. The policies are written in the YAML format because it is easily readable such that they can be quickly understood by humans; in addition, new custom policies could be created in an easy way. In every SCA check two types of definitions are gathered: metadata and logical descriptions with particular attention to *condition* and *rules* fields. In the former information, another field is present: *compliance* is used "to specify if the check is relevant to any compliance specifications"[25]. Furthermore, the Wazuh agents send only events necessary to keep updated on the global status. In the SCA module, each agent has its local database and when a change in the SCA checks is found, only the differences are sent to the manager. In case there is no change, a summary event is sent in order to avoid wasting network traffic while keeping the manager up to date. Each scan could have three different results, that are *passed*, *failed*, and *not applicable*. Describing Figure 2.7, two different elements are depicted: the integrity mechanism and the alerting flow. Regarding integrity, two different mechanisms are present: one for policy files and the other one for scan results. These mechanisms have been created to "ensure integrity between agent-side and manager-side states"[25]. The integrity of policy files mechanism is in charge to maintain policy files and scan results aligned. When a policy file change is detected, SCA invalidates the previous results about that policy in the database, and a new request for that specific policy is performed. Therefore, whenever the hash of a policy file changes, the following steps are performed: a log message appears in the manager, it flushes the stored data for the specific policy, the agent sends the scan results for the policy, and, in the end, the manager updates its database. Differently, regarding the integrity of the scan results, if the version between the agent side and the manager side differs, the manager requests the agent for its latest scan data and refreshes the database.

Monitoring security policies

Policy monitoring is "the process of verifying that all systems conform to a set of predefined rules regarding configuration settings and approved application usage"[25]. In order to accomplish this task, three different items are used by Wazuh: *Rootcheck*, *OpenSCAP*, and *CIS-CAT*. About *rootcheck* it has been replaced by the SCA module since the newest version of Wazuh. Indeed, the *rootcheck* engine performs checks if a process is running, if a file is present, and

if the content of a file contains a pattern. *OpenSCAP* is an integration with Wazuh that provides the possibility to perform configuration and vulnerability scans of an agent and it allows to verify of security compliance and to perform vulnerability and specialized assessments. The former is done thanks to the OpenSCAP policies which defines "the requirements that all systems [...] must meet in order to be in line with applicable *security policies* and/or *security benchmarks*"[25]. In according to the Wazuh documentation, the Security Content Automation Protocol is a way to express and manipulate security data in standardized ways. The security compliance evaluation process is done through four different components: SCAP scanner, security policies, profiles, and evaluation. The scanner is used to read SCAP policy and to perform checks on whether or not the system is compliant with it. The security policies, recognizable even as SCAP content, determine what should be the settings of a system and they are composed of machine-readable descriptions of the rules the system should follow. Moreover, each security policy is composed of one or more profiles and each profile is made by sets of rules and values. In other words, the profile is a "subset of rules within the policy"[25]. Finally, the evaluation is the "process performed by the OpenSCAP scanner on agent according to a specific security policy and profile"[25].

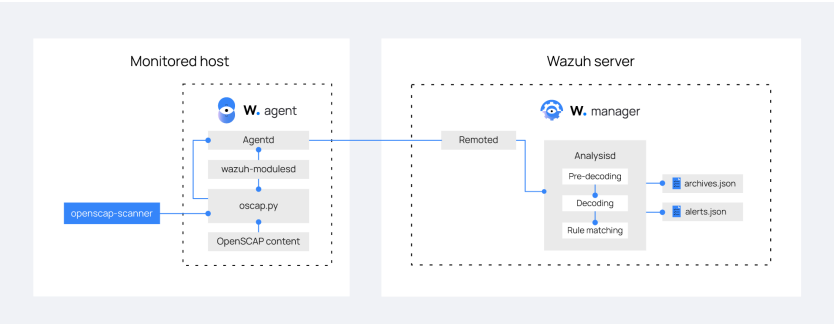

Figure 2.8: Wazuh OpenSCAP flow

The OpenSCAP flow is described by Figure 2.8 where the agent runs the *openscap-scanner* periodically according to the configuration and each result the scan provides is sent to the manager which could generate an alert in case the results' states fail.
The last component used by Wazuh for monitoring security policies is the *CIS-CAT* integration. Center for Internet Security is "an entity dedicated to safeguarding private and public organizations against cyber threats"[25] and

it is in charge to provide CIS benchmarks guidelines. Moreover, CIS-CAT Pro is a tool that scans the targeted systems generating a report about the differences between the system settings and the CIS benchmarks. Wazuh integrates CIS-CAT, which is a proprietary product, into the Wazuh agents and the result of each scan are reported as alerts.

Monitoring system calls

In Linux operating system, there is a way to prove "detailed real-time logging about the events that are happening on"[25] the system and this is *Audit*. It uses a set of rules to define what information to take from the log files. Three different types of rules exist control rules which allow modification of the configuration and the behavior of the Audit system; file system rules which permit audit access to particular files or directories; and system call rules in charge to log system calls that specified programs make. Wazuh permits the integration of the output of Audit rules thanks to decoders and rules which are suitable.

Command monitoring

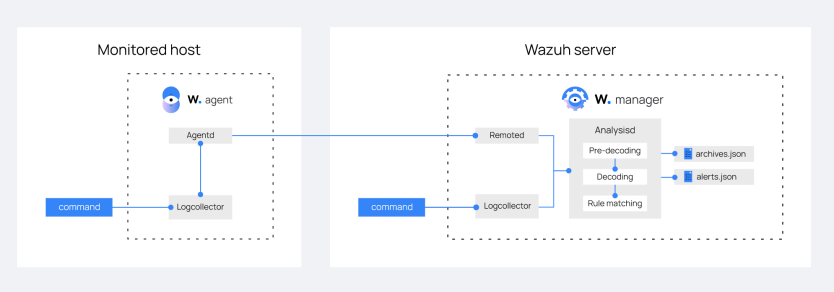

Figure 2.9: Wazuh command monitoring flow

Among others, Wazuh can monitor items that are not present in the log files. In order to accomplish this, Wazuh incorporates "the ability to monitor the output of specific commands and treat the output as though it were log file content"[25]. As shown in Figure 2.9, both the Wazuh manager and Wazuh agent should be configured to monitor remote commands output. After configurations are done, new custom rules could be created in order to

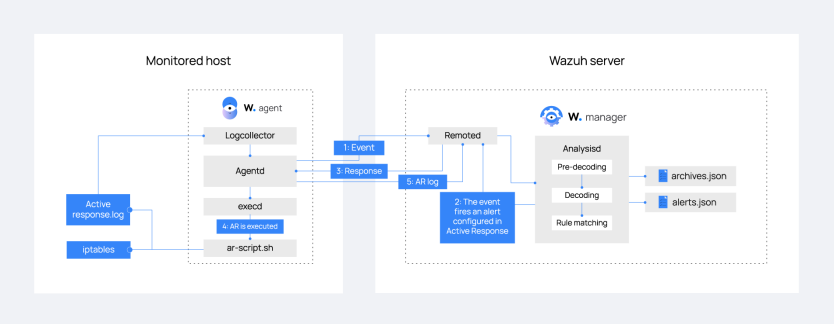

Figure 2.10: Wazuh active response workflow

process the output and, eventually, trigger an alert. Some possible usages could be monitoring disk space utilization, load average, changes in network listeners, running processes, and so on.

Active response

The active response is an action that permits the application of countermeasures against active threats. It executes a script "in response to the triggering of specific alerts based on alert level or rule group"[25]. Indeed, in case of poor implementation of rules and responses, the system's vulnerability increases.
Looking at Figure 2.10, when an event received by the Wazuh manager fires an alert configured in Active Response, the response is sent to the *Agentd* module of the agent, which executes the received response. Thereafter, the Active Response log is sent to the manager. First of all, active responses could be either stateful or stateless. The former undo the action after a specific period of time while the stateless responses are configured "as one-time actions without an event to revert the original effect"[25]. Moreover, the stateful responses perform the "basic actions of a stateless AR to undo later the process based on the command configuration"[25]. Every time an active response may be executed, the associated commands could be executed locally, that is, on the agent that generates the alert, on the server, on a defined agent specifying the IDs of the agents where running the script, or on every agent in the environment. To create a new active response, at first, a command must be defined and it will initiate certain scripts in response to a trigger. Secondly, the active response configuration should say when

and where the command should be executed. Wazuh already owns default active response scripts such as disabling an account, adding an IP address to the iptables deny list, adding an IP address to the firewalld drop list, adding an IP address to the /etc/hosts.deny file, different firewall-drop responses, restarting Wazuh, adding an IP address to null route, and so on.
In addition, Wazuh can integrate YARA, which is a "versatile Open Source pattern-matching tool aimed to detect malware samples based on rule description"[25]. A use case could be to execute automatically YARA scans through the active response module when a Wazuh File Integrity Monitoring alert is triggered. In this way, the resource consumption decreases because the scans are interested only in the modified files in the environment. Another use case could be the integration with VirusTotal, which monitor a directory in real-time, and if the modified file is malicious, it is deleted thanks to a triggered active response.

Agentless monitoring

In the case of routers, firewalls, switches, and Linux/BSD systems, the monitoring cannot happen through an agent, but agentless monitoring could be the correct solution. In this case, alerts are triggered "when checksums on the output changes"[25] and they show both the checksum and the diff output of the change. To create an agentless solution, SSH is needed, particularly its authentication. In general, the device should be added to the manager, then it monitors all the connected devices and BSD integrity check, Linux integrity check, generic diff check, and pix configurations check are performed.

Anti-flooding mechanism

This mechanism has been introduced to prevent a negative impact on the manager's network in case of a large burst of events sent from an agent. The solution is a leaky bucket queue that collects all the generated events and the sending rate is lower than the specified events per the second threshold. Going in deeper, agents collect information from a huge number of sources and they could send all the gathered information to the manager separated into individual events. In the absence of congestion control, all those single

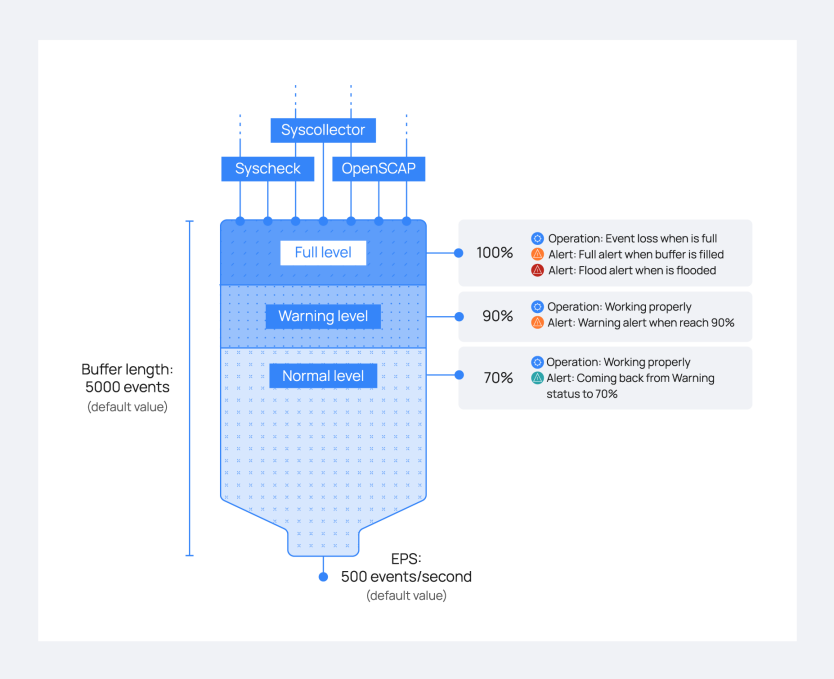

Figure 2.11: Wazuh anti-flooding bucket

events can saturate the manager's network, causing a loss of availability. Different misconfigurations can cause this scenario, such as real-time FIM of a directory with files that changes often, applications that retry on errors without a rate limiting, and so on. To handle this situation, two different mechanisms have been designed: agent-to-manager anti-flooding and internal agent anti-flooding controls. The first mechanism precludes the saturation of the network or of the manager by an agent through an agent-side leaky bucket queue; differently, the latter uses internal limits in different modules of the agent trying to slow down the rate at which events are sent.

The leaky bucket is "a congestion control located in agents and focused on agent-to-manager communication"[25]. The events generated by the agent are collected in a buffer of a specified size and, then, they are sent at a certain rate below a chosen threshold. From Figure 2.11, four different flooding situations are shown: warning alert, full alert, flood alert, and normal alert. The former triggers an alert on the manager when the occupied capacity reaches a threshold of 90 percent by default. The full alert is more serious than the warning one because "a full bucket will drop incoming events"[25].

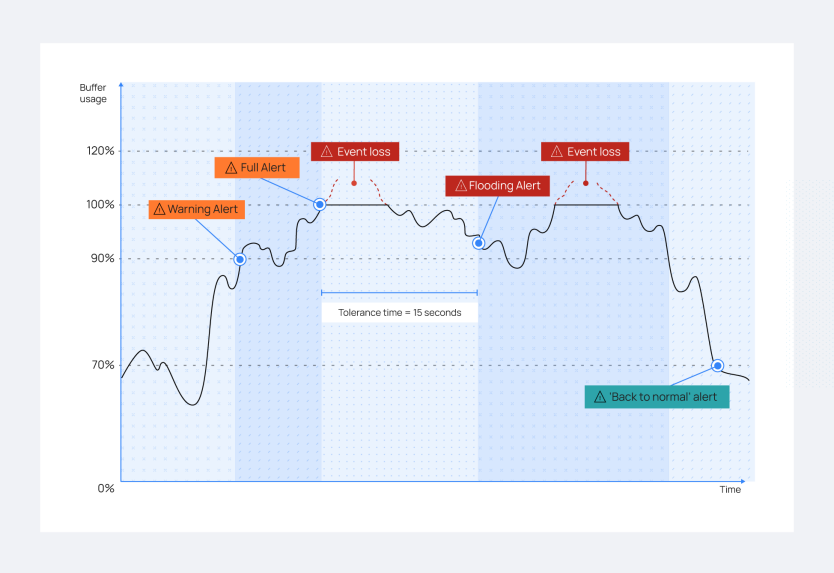

Figure 2.12: Wazuh buffer usage with flooding

The flood alert is generated if, after a certain amount of time since the full alert has been received, the buffer level has not fallen below the warning level. The last case is generated when the buffer level is under a certain value, 70 percent by default, after a warning alert or higher has been triggered. Each agent could configure the buffer differently: disabling it, configuring the queue size, or the events per second. The queue size is "the maximum number of events that can be held in the leaky bucket at one time"[25] and it should be set as the expected rate at which the agent may generate events. The events per second are the rate at which the events are pulled out from the agent's buffer at most. Moreover, warning, normal thresholds and a tolerance time for triggering a flooding alert could be configured.
As shown in Figure 2.12, the green area represents a normal status, that is, the buffer is working normally and in this situation, no alert should be generated. When a number of events occur, as happens in the orange area, the buffer usage could reach the warning level triggering a warning alert. If events arrive with a speed higher than its processing, 100% of the buffer usage could be reached, triggering the full alert. In this situation, all newly arriving events are dropped because no space in the queue is available. At this point, a timer is started and two possible things could happen: the buffer usage decrease to 70% before the timer expiration and no alert flooding is triggered, as shown in Figure 2.13 or the buffer usage stays high even after

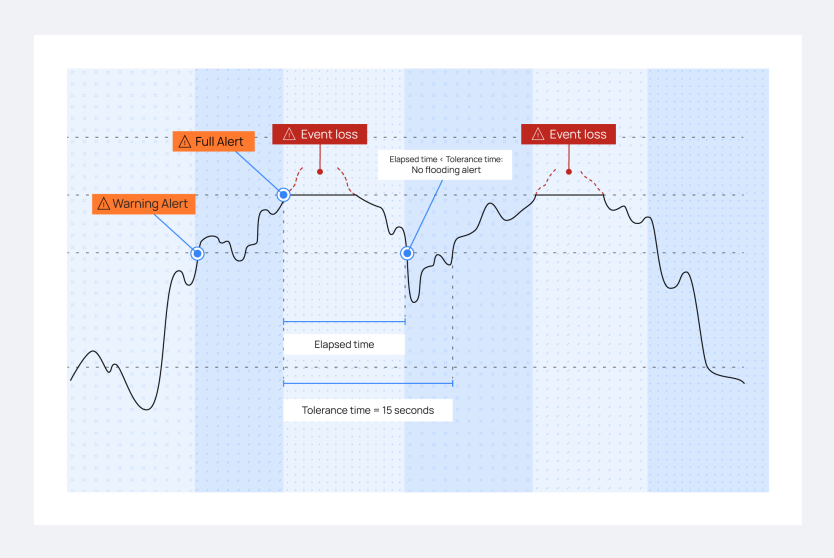

Figure 2.13: Wazuh buffer usage without flooding

the timer expiration and the flooding status (red area) is reached. When the former happens, only a module shutdown or generation of excessive events decreases can bring the situation back to normal, and when it happens a normal alert is generated.
In order to avoid those situations, the Wazuh agent modules that could cause saturation have been limited. The *Logcollector* module cannot "read more than a configurable maximum number of lines from the same file per read-cycle"[25] while CIS-CAT and *Syscollector* modules send their report to the manager at a regular speed instead of sent as soon as a scan would complete.

Agent labels

This feature has been added to include specific information about the agent in the generated alerts. In large environments, it could be useful for grouping agents with the same time zone, for example. To configure labels, a simple XML structure is included in the alert, nesting the labels through separating "key" terms in JSON formatted alerts.

System inventory

The agents could collect also interesting system information and store them into a SQLite database for each agent on the manager side and the *Syscollector* module is in charge of this task. When the agent starts, the former module runs periodically scans of defined targets and the collected data are sent to the manager that updates appropriate tables of the database. The collected information could be about hardware, operating system, packages, network interfaces, ports, processes, windows updates, and so on.

Vulnerability detection

Wazuh allows the detection of "vulnerabilities in the applications installed on agents using the Vulnerability Detector module"[25]. To accomplish this task, the agent collects a list of installed applications and then it is sent periodically to the manager. At the same time, the manager builds up a "global vulnerability database from publicly available CVE repositories"[25] and this information are matched with the agents' list of applications. A new alert is generated when a CVE affects a package installed on the monitored hosts. Consequently, the package is labeled as *vulnerable* and the alerts are stored in a per-agent vulnerabilities inventory. The module runs scans on startup or every period of time. When new packages are installed, a new scan is performed to check if they contain vulnerabilities. In case new CVEs information updates the database of vulnerabilities or an interval of time between two full scans expires, all the packages and the operating system are re-scanned. Therefore, three different types of scans are possible: baseline, full scan and partial scan. The former is triggered the first time the Vulnerability Detector is enabled and it performs a "full scan for every single package installed as well as the operating system"[25]. Moreover, the CVE inventory is updated with the found vulnerabilities and eventual alerts are generated. The full scan type is very similar to the baseline one and it happens when new packages are installed or when the timer between two full scans expires. The partial scan happens only when new packages are scanned. In addition, two considerations can be done: the timer between two full scans is set to protect the manager's performance. Furthermore, every vulnerability could be labeled as *valid*, which indicates that the vulnerability has not been

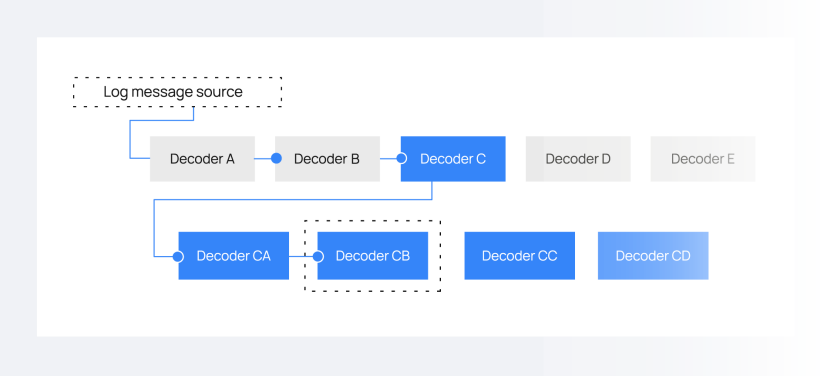

Figure 2.14: Wazuh Log Analysis without Sibling decoders

packaged yet, as *pending* in case the vulnerability should be confirmed during a full scan, and *obsolete* which indicates the vulnerability is no more present in the system. Finally, two different alert types are generated: detection alerts in case new vulnerabilities are added to the inventory and removal alerts when vulnerabilities have been removed from the inventory.

2.2.5 Ruleset

It is a set of rules used to detect attacks, intrusions, software misuse, configuration problems, application errors, malware, rootkits, system anomalies, or security policy violations in the monitored system. At version 4.3 of Wazuh, more than 3000 rules are supported and they accomplish technologies such as Syslog, Docker, Suricata, Telnet, SSH, and so on. Moreover, in the Wazuh repository new rules, decoders and rootkits can be found and, at the same time, each rule is mapped to PCI-DSS compliance controls in order to identify better when an alert is related to a specific compliance requirement. In order to analyze the information coming from the received events, Wazuh uses decoders in order to "identify event types and then extract the most relevant fields"[25].
From OSSEC, Wazuh takes thirteen predefined fields such as user, source and destination IP address, source and destination port, protocol, the action performed, ID, URL, data, extra data, status, and system name. From this set, only eight fields can be extracted simultaneously. However, it is often necessary to extract more than eight fields and, at the same time, the data

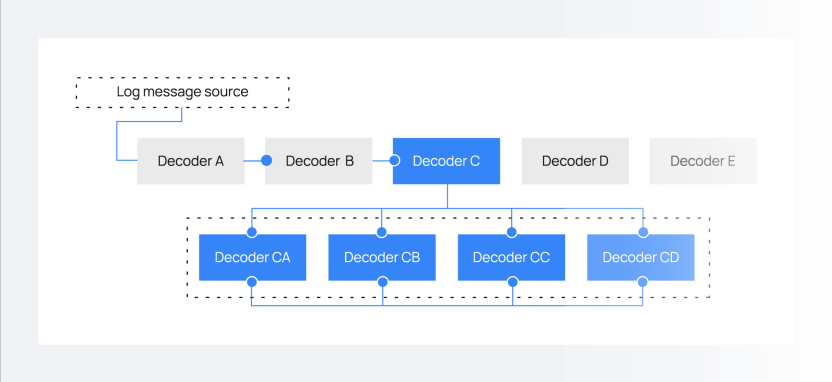

Figure 2.15: Wazuh Log Analysis with Sibling decoders

interested is not part of the predefined set of fields. For these reasons, Wazuh extends OSSEC original set to decode an unlimited number of fields where the field names are related to what is extracted. This is done through the *<order>* tag into a JSON field. Another important feature of Wazuh is the possibility of Sibling Decoders which are a decoder-building strategy helping the creation of custom rules. Indeed, different logs have different needs and different nested information, in particular in the case of dynamically structured logs. At first, it could be useful to understand better how the *analysisd* module works: Wazuh collects log events from a vast amount of sources; then, for every message, "the ruleset is analyzed using a very simple and resource-efficient logic that allows the Wazuh manager to handle large amounts of log data requiring few resources"[25]. Every decoder without a parent checks the log and when a condition is met, all the decoder children check again the log, as shown in Figure 2.14. Furthermore, when a particular parent decoder is found, the other parents' ones are no more checked and, of course, it means that if decoders are built too generic in the matching conditions, they could result in false positives and it could happen that the right decoders are not triggered.
In the case of dynamically structured logs, the provided information could be omitted or changed in the order, thus it becomes impossible to create all necessary decoders to match "each one of the possible combinations in which security-relevant data may be received"[25]. For these reasons, the necessity of sibling decoders which take advantage of the parent-children matching logic, the user can create a set of decoders where each one's parent of the others, as shown in Figure 2.15. The consequence is that when one decoder

is matched, all the other "sibling" ones are checked whilst they extract pieces of information at a time. Another consequence is that if the log information varies or is omitted, the *analysisd* module is the same ability to extract as much information as possible. In addition, another advantage is present: the sibling decoders are more readable with respect to the long regular expression strings used in "normal" decoders.

Chapter 3

Implementation of Wazuh in the Weseth platform

In the previous chapter, among the various proposed solutions, one has been chosen for implementation in the Weseth platform: Wazuh. According to the platform necessities, different events should be taken into consideration. In the first section, the test environment to perform tests in order to validate the Wazuh effectiveness is presented; thereafter, in the second section the events to be monitored are presented; finally, in the rest of the chapter, all the tests performed are described.

3.1 Test Environment

First of all, for the validation of the performed tests, a test environment has been built up. Two Weseth boxes have been used for its creation of it. In one of them, an Ubuntu Server distribution has been used upon which the Wazuh indexer, Wazuh server, Wazuh dashboard, and Filebeat have been installed. The other one is an original Weseth Box, where the Wazuh agent has been installed. Moreover, they communicate in a local network configuration without being exposed to the public Internet. The Wazuh documentation recommends installing an indexer, server, and dashboard

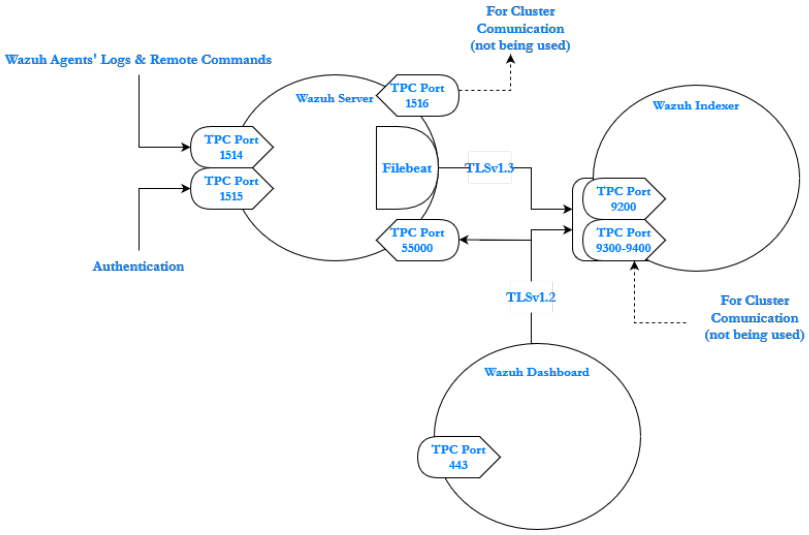

Figure 3.1: Diagram for Wazuh default installation

in three different places but for the test use case and in order to resemble the entire structure they have been placed in only one box. The original configuration should follow Figure 3.1.
As shown in Figure 3.1, three components should communicate between them through the TLS protocol with pre-determined ports. In the test environment taken into consideration, all components are installed in a single box, communications between them happen in localhost (IP 127.0.0.1) and the only opened ports to external are the port 1514, where Wazuh agents send their log data, and port 1515, for the enrollment of new ones for the Wazuh server. In the same way, for the Wazuh Dashboard only port 443 is opened even if it is the standard port for HTTPS traffic. Furthermore, due to the lack of clusters, ports 1516 and 9300-9400 are closed, as seen in the Listing 3.1.

Listing 3.1: Configuration of Wazuh cluster for test environment

```
<cluster>
  <name>wazuh</name>
  <node_name>node01</node_name>
  <node_type>master</node_type>
  <key></key>
  <port>1516</port>
  <bind_addr>0.0.0.0</bind_addr>
  <nodes>
```

```
    <node>NODE_IP</node>
  </nodes>
  <hidden>no</hidden>
  <disabled>yes</disabled>
</cluster>
```

3.2 Events to monitor

When the researcher has to talk with the box, it uses the Weseth Client which sends commands to the Weseth Box. For this reason, the principal target of monitoring is the Weseth Box, over which the Wazuh agent may be installed. The events to monitor should be referred to the box operating system and applications. About the former, it is a custom distribution based on Debian, which is a UNIX-based OS. The company, Drivesec, is interested in monitoring events, but at the same time, two challenges are open. The first one concerns the limited resources that can be used by the Wazuh agent daemon but the most important challenge concerns the network communication because LTE allows limited usage of bandwidth between the Box and the Wazuh Server. According to that, the events sent by the Box to the Server should be limited to avoid wasting useless bandwidth for not-so-interesting log data and information. Therefore, a solution has been provided: as seen in the previous chapter, Wazuh allows monitoring system calls through the Linux service *Auditd*. The latter is the "userspace component to the Linux Auditing System"[27] and it is "responsible for writing audit records to the disk"[27]. It allows three different types of rules that are control rules which allow configuring the Audit system's behavior, file system rules used as file watches, and system call rules. In practice, both Wazuh and auditd rules allow monitoring the file system, the first with the FIM module while the second with specific rules. The difference is that Wazuh allows monitoring, in real-time, only the entire directory and this means that a huge number of logs are created and this is not compliant with the box constraints. Differently, auditd rules allow a more granular definition of what the system is interested in, allowing it to monitor, in real-time, both single files and entire directories. From those premises, the file system monitoring is in charge of auditd both for the detection of anomalies in single files and entire directories. Therefore, a trade-off between security interesting

events and resource constraints should be found, and Drivesec has decided to monitor the following activities:

- Integrity of file and directories
- Hardware issues
- Process crashes
- Kernel module loading/unloading
- Kernel parameters tuning
- Privilege escalation attempts
- Peripherals attachment
- Statistical resource usage
- Process states such as debugger attaching
- Power state changes
- Discretionary Access Control
- User and Group Management
- Firewall violations
- Dangerous and suspicious activities

Integrity of file and directories

For the integrity of a file, it refers to additions, removals, and changes of content or file's attributes, while to the integrity of the directory. It refers to the addiction, modification, deletion, and modification of files in or in the directory itself. About files and directories, as written above, auditd rules have been used, and the elements that are interesting from a security point of view are files controlling user accounts and groups, firewall rules configuration files, service unit files, network configurations, DNS configurations, containerization software configurations, cron and scheduled

jobs, shell/profile configurations, and digital keys and certificates. The interest is in writes and changes of attributes to these files and directories and for each one, two rules should be generated. This comes from the necessity to specify the key (-k) of each rule; indeed, Wazuh allows to insert keys in the auditd rule such that it will be recognized by Wazuh itself and will be consequently analyzed. However, from tests performed, even if two different keys can be used to label a rule, it is better to specify two different ones: one for the write changes and the other one for the changes of attributes activities.

Listing 3.2: Auditd rules for write and attributes changes of */etc/passwd*

```
1 ## Rule for monitoring file writes
2 -w /etc/passwd -p w -k audit-wazuh-w
3
4 ## Rule for monitoring file attributes changes
5 -w /etc/passwd -p a -k audit-wazuh-a
```

From the Listing 3.2, the specified permission *-p w* for the writes changes is labelled with the key *audit-wazuh-w* while the permission for the attributes changes *-p a* is labelled with the key *audit-wazuh-a*. Hence, generally, for each file and directory defined above, two different auditd rules have been added such that they create logs in the monitored hosts, in particular in the */var/log/audit/audit.log* file. In order to allow the forwarding of these logs, the Listing 3.3 is added to the configuration file, *ossec.conf*, as shown below.

Listing 3.3: Configuration for the forwarding of auditd logs

```
1 <localfile>
2     <log_format>audit</log_format>
3     <location>/var/log/audit/audit.log</location>
4 </localfile>
```

Test performed As seen in the Listing 3.4, two simple actions have been performed: insertion of a new line and, immediately after, its cancellation. In the line 1 of the Listing 3.4, the variable StringArray contains all the file the system is interested in.

Listing 3.4: Tests performed for validate file integrity changes

```
for val in "\${StringArray[@]}"; do
    echo "Append my text to a file, please" >> \$val
    sed -i '' -e '\$ d' \$val
done
```

At the same time, also directories should be monitored, particularly directories within which all the files are important from the monitoring point of view.

Listing 3.5: Tests performed for validate directory integrity changes

```
string1="file"
for val in "${StringArray[@]}"; do
    str=$val$string1
    touch $str
    echo "New line inside the file" >> $str
    rm $str
done
```

As shown in the Listing 3.5, the actions performed for each directory, put as an item in the *list* StringArray, are insertion of a new file, appending a string, and, immediately, the file itself is removed.

Test result Remembering that the directories and files integrities are monitored thanks to auditd rules, each log is treated by Wazuh following the decoder in Listing 3.6.

Listing 3.6: Decoder used for file integrity

```
<decoder name="auditd-syscall">
  <parent>auditd</parent>
  <regex offset="after_regex">type=PATH msg=audit\(\S+\): item=\S+ name="
    (\.+)" inode=(\S+) dev=\S+ mode=(\S+) ouid=\S+ ogid=\S+ |type=PATH msg
    =audit\(\S+\): item=\S+ name=\((null)\) inode=(\S+) dev=\S+ mode=(\S+)
     ouid=\S+ ogid=\S+ </regex>
  <order>audit.file.name, audit.file.inode, audit.file.mode</order>
</decoder>
```

Moreover, when this decoder is triggered and a line is appended to the file, the rule in Listing 3.7 is triggered.

Figure 3.2: Wazuh alert for file write access

Figure 3.3: Wazuh alert for file sed command

Listing 3.7: Rule used for write in a file

```
1 <rule id="80781" level="3">
2   <if_sid>80780</if_sid>
3   <field name="audit.file.name">\.+</field>
4   <description>Audit: Watch - Write access: $(audit.file.name).</
      description>
5   <group>audit_watch_write,gdpr_IV_30.1.g,</group>
6 </rule>
```

The triggered rule creates the alert with a level of 3 seen at Figure 3.2 when the command in line 2 of Listing 3.4 is run.
Similarly, when the line is deleted from the file, as shown in the line 3 of the Listing 3.4, the rule in Listing 3.7 is triggered again creating the alert at Figure 3.3, with an alert level of 3.
Instead, for the command in line 6 of Listing 3.5, the rule in Listing 3.8 is triggered and the alert in Figure 3.4 is created, with again an alert level of 3.

Listing 3.8: Rule used for write in a directory

```
1 <rule id="80791" level="3">
2   <if_group>audit_watch_write</if_group>
3   <match>type=DELETE</match>
4   <description>Audit: Deleted: $(audit.file.name).</description>
5   <mitre>
6     <id>T1070.004</id>
7   </mitre>
8   <group>audit_watch_delete,audit_watch_write,gdpr_II_5.1.f,gdpr_IV_30.1.
      g,</group>
9 </rule>
```

At the same time, if tests at Listing 3.4 and Listing 3.5 are performed with

Figure 3.4: Wazuh alert for file deleting in a monitored directory

directories and files that are not interesting from a security point of view, no alerts are generated by Wazuh.

Hardware issues

Hardware issue refers to problems with the hardware devices such as the Weseth box, the LTE modem, and so on. From this aspect, various problems have been encountered. Different hardware manufacturers define different structures and types of logs, and embracing them is impossible. The adopted solution is trying to embrace the most using a simple detector and rule. When a log such as *[8.510810] ata1: SError: { UnrecovData 10B8B BadCRC }* is analyzed, Wazuh does not use any type of decoder, however the rule at Listing 3.9 is triggered; it tries to match some bad words defined at lines 1 of the Listing 3.9.

Listing 3.9: Rule used for generic errors

```
1 <var name="BAD_WORDS">core_dumped|failure|error|attack| bad |illegal |
    denied|refused|unauthorized|fatal|failed|Segmentation Fault|Corrupted
    </var>
2
3 <rule id="1002" level="2">
4     <match>$BAD_WORDS</match>
5     <description>Unknown problem somewhere in the system.</description>
6     <group>gpg13_4.3,</group>
7 </rule>
```

However, there is another problem: by default, the alerts shown in the Wazuh dashboard have as 3 the minimum alert level while the rule at Listing 3.9 produces an alert level of 2. Therefore, ultimately, the alerts as shown in Figure 3.5 are generated only if the configuration of the Wazuh manager is changed as shown in the Listing 3.10, or changing the corresponding Wazuh rule with an alert level of 3.

Figure 3.5: Wazuh alert for generic hardware issue

Listing 3.10: Manager configuration about alert level

```
<alerts>
    <log_alert_level>2</log_alert_level>
    <email_alert_level>12</email_alert_level>
</alerts>
```

This event has been the most difficult to emulate due to the challenge in creating hardware problems. The only way to accomplish it has been to copy a log with hardware issue in the log file.

Process crashes

These events are related to processes that crash abnormally. Wazuh has, by default, some rules to accomplish the monitoring of these events.

Test performed As seen in the Listing 3.11 two single actions have been performed: starting the *top* command in the foreground and, then, as shown at line 2 of the same Listing, killing it with a *SIGSEV* signal that is a segmentation fault one, and, in Linux, it corresponds to 11.

Listing 3.11: Tests performed for validate process crashes events

```
./top&
kill -11 $(pidof top)
```

Test result The decoders used are different and they could be found in the *0040-auditd_decoders.xml* file while the rule triggered for the specific event is the one shown in Listing 3.12 while the alert generated is the one at Figure 3.6 with an alert level of 10.

Figure 3.6: Wazuh alert for process crash

Listing 3.12: Rule used for process crashes detection

```
<rule id="80711" level="10">
  <if_sid>80700</if_sid>
  <field name="audit.type">ANOM_ABEND</field>
  <description>Auditd: Process ended abnormally.</description>
  <group>audit_anom,gdpr_IV_30.1.g,gdpr_IV_35.7.d,gpg13_4.14,hipaa_164
    .312.b,nist_800_53_AU.6,nist_800_53_SI.4,pci_dss_10.6.1,pci_dss_11.4,
    tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

Kernel module loading/unloading

These events refer to the loading or unloading of kernel modules in a Linux environment where a module is a piece of code. In Linux, two different commands can be used to achieve the loading and they are *modprobe* and *insmod* where the former needs the path where the module is placed, while the latter watches itself directly in the default module directory. Moreover, the former can be configured with the *modprobe.conf* file, and with the *modprobe.d* directory: for these reasons, they are interesting from a security point of view and they have monitored thanks to the auditd rules similar to the ones in Listing 3.2. On the contrary, in order to unload the loaded module, the *rmmod* command is used, and monitored. For the commands, the auditd rules shown in the Listing 3.13 have been used.

Listing 3.13: Auditd rules for kernel modules loading/unloading

```
## Kernel module loading and unloading
-a always,exit -F perm=x -F auid!=-1 -F path=/usr/sbin/insmod -k audit-
    wazuh-c
-a always,exit -F perm=x -F auid!=-1 -F path=/usr/sbin/modprobe -k audit-
    wazuh-c
-a always,exit -F perm=x -F auid!=-1 -F path=/usr/sbin/rmmod -k audit-
    wazuh-c
-a always,exit -F arch=b64 -S finit_module -S init_module -S
    delete_module -F auid!=-1 -k audit-wazuh-c
```

As shown in the Listing before, the interest is on the command, and, for this reason, the key *-k audit-wazuh-c* has been added; indeed, it is analyzed by the decoders and rules for logs coming from auditd rules related to commands. Moreover, here the permission is related to the execution of commands and each audit rule is marked with the permission *perm=x*.

Test performed As seen in the Listing 3.14, at first an example module is loaded with *insmod*, then it is removed with *rmmod*; lately, the same module is loaded with the command *modprobe*, and, finally, it is removed again.

Listing 3.14: Tests performed for validate kernel module loading/unloading

```
insmod /lib/modules/5.10.0-13-amd64/kernel/fs/ufs/ufs.ko
rmmod /lib/modules/5.10.0-13-amd64/kernel/fs/ufs/ufs.ko
modprobe ufs
rmmod /lib/modules/5.10.0-13-amd64/kernel/fs/ufs/ufs.ko
```

Test result As seen in the case of process crashes, even here different decoders from the *0040-auditd_decoders.xml* file have been used to find as much information as possible from the logs. Differently, the triggered rule is the one in Listing 3.15 that has an alert level of 3.

Listing 3.15: Rule used for kernel module loading and unloading

```
<rule id="80792" level="3">
  <if_sid>80700</if_sid>
  <list field="audit.key" lookup="match_key_value" check_value="command">
    etc/lists/audit-keys</list>
  <description>Audit: Command: $(audit.exe).</description>
  <group>audit_command,gdpr_IV_30.1.g,</group>
</rule>
```

The rule embraces both loading and unloading actions and the generated alerts are always the same independently from the action performed. An example alert is shown in Figure 3.7.

Figure 3.7: Wazuh alert for kernel module loading/unloading

Kernel parameters tuning

Kernel parameters are tunable values that can be adjusted while the system is running and their tuning could change the system's behavior. For that reason, the tuning should be monitored. The tuning happens with the command *sysctl* and also here the auditd rule shown in the Listing 3.16 is able to detect both the reading of already existent parameters and writing of new ones.

Listing 3.16: Auditd rules for kernel parameters tuning loading/unloading

```
1 -a always,exit -F perm=x -F auid!=-1 -F path=/usr/sbin/sysctl -k audit-
    wazuh-c
```

Even in the latter, the permission concerns the execution and the key used is *audit-wazuh-c*. Moreover, also the file */etc/sysctl.conf* and the directory */etc/sysctl.d* contains configuration and kernel parameters: they are also monitored with the file and directory integrity checks.

Test performed For the kernel parameters tuning, the test performed is the one seen in the Listing 3.17 where the command *sysctl* is run twice. In line 1, the command shows all the kernel parameters, while in the second one, the parameter *net.ipv4.ip_forward* is set to 1.

Listing 3.17: Tests performed for validate kernel parameters tuning

```
1 sysctl -a
2 sysctl -w net.ipv4.ip_forward=1
```

Test result Every time one log coming as a result of auditd rules is created, it triggers the decoders at file *0040-auditd_decoders.xml* and the rule at Listing 3.15.

Privilege escalation attempts

Privilege escalation means exploiting bugs, and misconfigurations of applications and operating systems to gain more privileges that allow to access resources not accessible before. Two different types of privilege escalation exist vertical and horizontal. In the vertical case, a user with low privileges tries to gain higher ones; differently, in the horizontal privilege escalation, a normal user tries to gain the privileges of another normal user. Wazuh is able to detect privilege escalation attempts analyzing logs in */var/log/auth.log*, as shown in the Listing 3.18.

Listing 3.18: Configuration for the forwarding of auth.log

```
<localfile>
  <log_format>syslog</log_format>
  <location>/var/log/auth.log</location>
</localfile>
```

To be more specific and to know who is the user after the *sudo* command, it is necessary to configure PAM. So, each login with PAM or SSH is logged and Wazuh analyzes their logs, therefore no new rules or auditd rules have been added.

Test performed In Listing 3.19 the tests performed for validate the detection of privilege escalation attempts.

Listing 3.19: Tests performed for validate privilege escalation attempts

```
## Run sudo as normal user with the correct password
echo <correct_password> | sudo -S sleep 1 && sudo su - root
## Run sudo as normal user with the wrong password
echo <wrong_password> | sudo -S sleep 1 && sudo su - root
## Run sudo as normal user with the wrong password for 5 times
for i in {1..5}
do
    echo <wrong_password> | sudo -S sleep 1 && sudo su - root
done

## Run ssh login with an existent user and correct password
sshpass -p <correct_password> ssh <username>@<ip_address>
```

```
## Run ssh login with an existent user and wrong password
sshpass -p <wrong_password> ssh <username>@<ip_address>
## Run ssh login with an existent user and wrong password for 15 times
for i in {1..15}
do
    sshpass -p <wrong_password> ssh <username>@<ip_address>
done
## Run ssh login with a non-existent user
sshpass -p <password> ssh <username>@<ip_address>
## Run ssh login to a different port
sshpass -p <password> ssh <username>@<ip_address> -p 55
## Run ssh login to a different port for 10 times
for i in {1..10}
do
    sshpass -p <password> ssh <username>@<ip_address> -p 55
done
## Run ssh login and stopping it immediately after
sshpass -p <password> ssh <username>@<ip_address>&
kill -11 $(pidof sshpass)
```

The first part concerns the correct and wrong login with the PAM module; the second one concerns the correct and wrong attempt to login with SSH. About the first part, the tests are written in that way because the standard input has to be filled with the echoed password and the sleep of 1 second has been used for that reason.

Test result For the line 2 of Listing 3.19 the decoder in Listing 3.20 is triggered together with the rules at Listing 3.21 and at Listing 3.22.

Listing 3.20: Decoder used for first time sudo executed

```
<decoder name="sudo-fields">
  <parent>sudo</parent>
  <prematch>\s</prematch>
  <regex>^\s*(\S+)\s*:</regex>
  <order>srcuser</order>
  <fts>name,srcuser,location</fts>
  <ftscomment>First time user executed the sudo command</ftscomment>
</decoder>
```

Listing 3.21: Rule used for first time sudo executed

Figure 3.8: Wazuh alert for the first time sudo is executed

Figure 3.9: Wazuh alert for sudo success

```
<rule id="5403" level="4">
  <if_sid>5400</if_sid>
  <if_fts />
  <description>First time user executed sudo.</description>
  <mitre>
  <id>T1548.003</id>
  </mitre>
</rule>
```

Listing 3.22: Rule used for sudo success

```
<rule id="5402" level="3">
  <if_sid>5400</if_sid>
  <regex> ; USER=root ; COMMAND=| ; USER=root ; TSID=\S+ ; COMMAND=</
    regex>
  <description>Successful sudo to ROOT executed.</description>
  <mitre>
    <id>T1548.003</id>
  </mitre>
  <group>pci_dss_10.2.5,pci_dss_10.2.2,gpg13_7.6,gpg13_7.8,gpg13_7.13,
    gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AC.7,
    nist_800_53_AC.6,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

Moreover, for the same test performed, the alert with a level of 4 is triggered, as shown in Figure 3.8, and the alert with level 3 is seen in Figure 3.9.
For the line 4 at Listing 3.19, various decoders that are present in the file *0205-pam_decoders.xml* have been used to gather all the possible information. Furthermore, the triggered rule at Listing 3.23 generates the alert with alert level of 5 shown in Figure 3.10.

Figure 3.10: Wazuh alert for sudo failure

Figure 3.11: Wazuh alert for two sudo failures

Listing 3.23: Rule used for sudo failure

```
<rule id="5503" level="5">
  <if_sid>5500</if_sid>
  <match>authentication failure; logname=</match>
  <description>PAM: User login failed.</description>
  <mitre>
    <id>T1110.001</id>
  </mitre>
  <group>authentication_failed,pci_dss_10.2.4,pci_dss_10.2.5,gpg13_7.8,
    gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.14,
    nist_800_53_AC.7,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
 </rule>
```

From line 6 to line 9 of Listing 3.19, a user login fails for 5 times due to wrong passwords. The first time, the alert at Figure 3.10 is generated. The second time, the login action triggers only a rule without decoders that is the one in Listing 3.24 generating the alert at Figure 3.11 with alert level of 10.

Listing 3.24: Rule used for twice sudo failures

```
<rule id="2502" level="10">
  <match>more authentication failures;|REPEATED login failures</match>
  <description>syslog: User missed the password more than one time</
    description>
  <mitre>
    <id>T1110</id>
  </mitre>
  <group>authentication_failed,pci_dss_10.2.4,pci_dss_10.2.5,gpg13_7.8,
    gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.14,
    nist_800_53_AC.7,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

Figure 3.12: Wazuh alert for three sudo failures

The third time the login is failed, the triggered rule is at Listing 3.25 that generates the alert with level 10 of Figure 3.12.

Listing 3.25: Rule used for three sudo failures

```
<rule id="5404" level="10">
  <if_sid>5401</if_sid>
  <match>3 incorrect password attempts</match>
  <description>Three failed attempts to run sudo</description>
  <mitre>
    <id>T1548.003</id>
  </mitre>
  <group>pci_dss_10.2.4,pci_dss_10.2.5,gpg13_7.8,gdpr_IV_35.7.d,
   gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AC.7,
   tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

Every time the login fails more than one time, the triggered rule is the one at Listing 3.23 and the Figure 3.10 shows the alert generated.
The second part of the tests performed concerns SSH login. In line 12 of Listing 3.19, a SSH login is run with an existent username and a correct password. The incoming log triggers the decoder at Listing 3.26 and the rule at Listing 3.27.

Listing 3.26: Decoder used for SSH login success

```
<decoder name="sshd-success">
  <parent>sshd</parent>
  <prematch>^Accepted</prematch>
  <regex offset="after_prematch">^ \S+ for (\S+) from (\S+) port (\S+)</
   regex>
  <order>user, srcip, srcport</order>
  <fts>name, user, location</fts>
</decoder>
```

Figure 3.13: Wazuh alert for SSH login success

Listing 3.27: Rule used for SSH login success

```
<rule id="5715" level="3">
  <if_sid>5700</if_sid>
  <match>^Accepted|authenticated.$</match>
  <description>sshd: authentication success.</description>
  <mitre>
    <id>T1078</id>
    <id>T1021</id>
  </mitre>
  <group>authentication_success,gdpr_IV_32.2,gpg13_7.1,gpg13_7.2,
   hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AC.7,pci_dss_10.2.5,
   tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

When, respectively, the decoder and the rule are triggered, the alert with level 3 at Figure 3.13 is generated.
The test performed at line 14 of Listing 3.19 refers to SSH login failure due to a wrong password. The decoder is the one at Listing 3.28, while the rule is the one at Listing 3.29.

Listing 3.28: Decoder used for SSH login failure

```
<decoder name="ssh-failed">
  <parent>sshd</parent>
  <prematch>^Failed \S+ </prematch>
  <regex offset="after_prematch">^for (\S+) from (\S+) port (\d+)</regex>
  <order>user, srcip, srcport</order>
</decoder>
```

Listing 3.29: Rule used for SSH login failure

```
<rule id="5760" level="5">
  <if_sid>5700,5716</if_sid>
  <match>Failed password|Failed keyboard|authentication error</match>
  <description>sshd: authentication failed.</description>
  <mitre>
    <id>T1110.001</id>
```

Figure 3.14: Wazuh alert for SSH login failure

```
    <id>T1021.004</id>
  </mitre>
  <group>authentication_failed,gdpr_IV_35.7.d,gdpr_IV_32.2,gpg13_7.1,
    hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AC.7,pci_dss_10.2.4,
    pci_dss_10.2.5,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

The consequent alert, with level 5, is the one in Figure 3.14.
From line 16 up to line 19 of Listing 3.19, for fifteen times, a SSH login is performed with a wrong password and the result are two different alerts generated from the triggering of the decoder at Listing 3.30 and the rules at Listing 3.31.

Listing 3.30: Decoder used for SSH login failure excesses

```
<decoder name="sshd-exceed">
  <parent>sshd</parent>
  <prematch> exceeded for </prematch>
  <regex offset="after_prematch">^(\S+) from (\S+) port (\d+) </regex>
  <order>user, srcip, srcport</order>
</decoder>
```

Listing 3.31: Rule used for SSH login failures excesses

```
<rule id="5758" level="8">
  <if_sid>5700,5710</if_sid>
  <match>^error: maximum authentication attempts exceeded </match>
  <description>Maximum authentication attempts exceeded.</description>
  <mitre>
    <id>T1110</id>
  </mitre>
  <group>authentication_failed,gpg13_7.1,</group>
</rule>
```

The consequent alert is the one at Figure 3.15.
Moreover, because all these consequent tries are part of an attack pattern,

Figure 3.15: Wazuh alert for SSH login failures excesses

Figure 3.16: Wazuh alert for multiple SSH login failures

the rule at Listing 3.32 is triggered causing the generation of an alert with level 10 of Figure 3.16.

Listing 3.32: Rule used for multiple SSH login failures

```
<rule id="40111" level="10" frequency="12" timeframe="160">
  <if_matched_group>authentication_failed</if_matched_group>
  <same_source_ip />
  <description>Multiple authentication failures.</description>
  <mitre>
    <id>T1110</id>
  </mitre>
  <group>authentication_failures,pci_dss_10.2.4,pci_dss_10.2.5,gpg13_7.1,
   gpg13_7.8,gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU
   .14,nist_800_53_AC.7,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

Step by step, when failures increase, new attack patterns are found such as brute force attack that is monitored thanks to the rule at Listing 3.33 that fires the alert at Figure 3.17 when the failures are more than the value *frequency* of the rule: in this case 8.

Listing 3.33: Rule used for SSH brute force attack

```
<rule id="5763" level="10" frequency="8" timeframe="120" ignore="60">
  <if_matched_sid>5760</if_matched_sid>
  <same_source_ip/>
  <description>sshd: brute force trying to get access to the system.
   Authentication failed.</description>
  <mitre>
    <id>T1110</id>
  </mitre>
```

Figure 3.17: Wazuh alert for SSH brute force attack

```
8   <group>authentication_failures,gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164
      .312.b,nist_800_53_SI.4,nist_800_53_AU.14,nist_800_53_AC.7,pci_dss_11
      .4,pci_dss_10.2.4,pci_dss_10.2.5,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7
      .3,</group>
9 </rule>
```

In line 21 of Listing 3.19, a SSH login is performed with a non existent user causing the triggering of the decoder at Listing 3.34 and the rule at Listing 3.35.

Listing 3.34: Decoder used for SSH login with invalid user

```
1 <decoder name="ssh-invalid-user">
2   <parent>sshd</parent>
3   <prematch>^Invalid user|^Illegal user</prematch>
4   <regex offset="after_prematch">(\S+) from (\S+)</regex>
5   <order>srcuser,srcip</order>
6 </decoder>
```

Listing 3.35: Rule used for SSH login with invalid user

```
1  <rule id="5710" level="5">
2    <if_sid>5700</if_sid>
3    <match>illegal user|invalid user</match>
4    <description>sshd: Attempt to login using a non-existent user</
       description>
5    <mitre>
6      <id>T1110.001</id>
7      <id>T1021.004</id>
8      <id>T1078</id>
9    </mitre>
10   <group>authentication_failed,gdpr_IV_35.7.d,gdpr_IV_32.2,gpg13_7.1,
       hipaa_164.312.b,invalid_login,nist_800_53_AU.14,nist_800_53_AC.7,
       nist_800_53_AU.6,pci_dss_10.2.4,pci_dss_10.2.5,pci_dss_10.6.1,tsc_CC6
       .1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
11 </rule>
```

Figure 3.18: Wazuh alert for SSH login with invalid user

Figure 3.19: Wazuh alert for SSH timeout

The rule above generates the alert with level 5 of the Figure 3.18.
In line 23 of Listing 3.19, a SSH login is performed with a correct user but with a port where the SSH service is not hosted. The result is a connection timed out that is detected from the simple decoder at Listing 3.36 and from the rule at Listing 3.37.

Listing 3.36: Decoder used for SSH

```
<decoder name="sshd">
  <program_name>^sshd</program_name>
</decoder>
```

Listing 3.37: Rule used for SSH timeout

```
<rule id="5704" level="4">
  <if_sid>5700</if_sid>
  <match>fatal: Timeout before authentication for</match>
  <description>sshd: Timeout while logging in.</description>
</rule>
```

The resulting alert is the one in Figure 3.19.
From line 25 up to line 28 of Listing 3.19, a loop with ssh login attempts to the port 55, that is not the port hosting the SSH service, is performed. No decoder is triggered while, being a loop, the rule at Listing 3.38 is because the frequency of the timeout action is bigger than 6.

Listing 3.38: Rule used for SSH high number timeouts

```
<rule id="5705" level="10" frequency="6" timeframe="360">
  <if_matched_sid>5704</if_matched_sid>
  <description>sshd: Possible scan or breakin attempt (high number of
    login timeouts).</description>
```

Figure 3.20: Wazuh alert for SSH scan

Figure 3.21: Wazuh alert for SSH connection reset

```
4    <mitre>
5      <id>T1190</id>
6      <id>T1110</id>
7    </mitre>
8    <group>gdpr_IV_35.7.d,gpg13_4.12,nist_800_53_SI.4,pci_dss_11.4,tsc_CC6
       .1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
9  </rule>
```

The rule above generates the alert with level 10 of Figure 3.20.
Line 30 and line 31 of Listing 3.19 perform a ssh login that is stopped immediately after. Those events trigger, firstly, the decoder at Listing 3.39, and secondly the rule at Listing 3.40.

Listing 3.39: Decoder used for SSH connection reset

```
1  <decoder name="sshd-reset">
2    <parent>sshd</parent>
3    <prematch>Connection reset</prematch>
4    <regex offset="after_prematch">(\S+) (\S+) port (\d+)</regex>
5    <order>user, srcip, srcport</order>
6  </decoder>
```

Listing 3.40: Rule used for SSH connection reset

```
1  <rule id="5762" level="4">
2    <if_sid>5700</if_sid>
3    <match>Connection reset</match>
4    <description>sshd: connection reset</description>
5  </rule>
```

The result of those triggers is the alert with level 4 shown in Figure 3.21.

Peripherals attachment

Those events concern the detection of new peripherals attached. By default, Wazuh detects when new peripherals are attached and disconnected but, in the case of mounting as a file system the interest is also in the system calls *mount* and *umount*. To achieve that, one auditd rule has been added, as shown in the Listing 3.41.

Listing 3.41: Auditd rules for mount and umount operations

```
## Mount operations
-a always,exit -F arch=b64 -S mount -S umount2 -F auid!=-1 -k audit-wazuh-
    c
```

Test performed About tests, they can be divided in three parts. In the first one, an USB device is attached; in the second step the tests performed are shown in the Listing 3.42; finally, the USB device is disconnected.

Listing 3.42: Tests performed for validate mount and umount operations

```
## Mount the attached USB to a file system partition
mount /dev/sdb1 /media/usb_device
## Disconnect and umount the file system partition
umount /media/usb_device
```

For clarification, */dev/sdb1* is the special device file for the attached device.

Test result At first, the USB device has been attached triggering the decoder at Listing 3.43 and the rule at Listing 3.44.

Listing 3.43: Decoder used for attached device

```
<decoder name="usb-storage-attached">
  <parent>kernel</parent>
  <prematch offset="after_parent">^usb|^[\s*\S+] usb</prematch>
  <regex offset="after_parent">^(usb) |^[\s*\S+] (usb)</regex>
  <order>id</order>
</decoder>
```

Figure 3.22: Wazuh alert for attached device

Figure 3.23: Wazuh alert for umount operation

Listing 3.44: Rule used for attached device

```
<rule id="81101" level="3">
  <if_sid>81100</if_sid>
  <match>New USB device found</match>
  <description>Attached USB Storage</description>
  <group>gpg13_4.8,</group>
</rule>
```

Those decoder and rule generate the alert with level 3 shown in Figure 3.22. After that, the tests shown in Listing 3.42 at line 2 and line 4 are performed. Both trigger the decoders in *0040-auditd_decoders.xml* file and the rule at Listing 3.15, that, of course, concerns the analysis of auditd logs. The alerts generated are shown in Figure 3.24 for the *mount* operation and in Figure 3.23 for the *umount* one.

Statistical resource usage

For that specific event, new decoders and new rules have been added to the Wazuh ruleset trying to exploit the command monitoring offered by Wazuh itself. First of all, the command used is *uptime* that prints the current time, the length of time the system has been up, the number of users online, and the load average. "The load average is the number of runnable processes over the preceding 1-, 5-, 15-minute intervals"[28]. Secondary, a XML tag has been added to the Wazuh server configuration as shown in the Listing 3.45.

Listing 3.45: Configuration for the command monitoring of *uptime*

```
<localfile>
  <log_format>command</log_format>
```

Figure 3.24: Wazuh alert for umount operation

```
  <command>uptime</command>
  <frequency>120</frequency>
</localfile>
```

The *frequency* represents how much time elapses between two consecutive execution of the *command* written above. Then, two new decoders have been added because the output of the command could have two different formats: one for the current time in minutes, the other one for the current time in hours, as shown in Listing 3.46 and Listing 3.47, respectively.

Listing 3.46: Decoder used for *uptime* in minutes format

```
<decoder name="ossec-uptime-min">
  <parent>ossec</parent>
  <type>ossec</type>
  <prematch offset="after_parent">^output: 'uptime':\s+\d+:\d+:\d+\s+up\s
    +\d+\s+min,\s+\d+\s+\S+,\s+load average:</prematch>
  <regex offset="after_prematch">^ (\d+.\d+)</regex>
  <order>extra_data</order>
</decoder>
```

Listing 3.47: Decoder used for *uptime* in hours format

```
<decoder name="ossec-uptime-hour">
  <parent>ossec</parent>
  <type>ossec</type>
  <prematch offset="after_parent">^output: 'uptime':\s+\d+:\d+:\d+\s+up\s
    +\d+:\d+,\s+\d+\s+\S+,\s+load average:</prematch>
  <regex offset="after_prematch">^ (\d+.\d+)</regex>
  <order>extra_data</order>
</decoder>
```

Thus, the *extra_data* is the value of load average in the last minute and it is matched by the rule in Listing 3.48.

Listing 3.48: Rule used for *uptime*

Figure 3.25: Wazuh alert for *uptime* output

```
<rule id="100300" level="7">
  <if_sid>530</if_sid>
  <match>ossec: output: 'uptime': </match>
  <extra_data type="pcre2">^(?:[5-9]\d*\.\d+)|(?:\d\d+\.\d+)$</extra_data
    >
  <description>Load average reached 5..</description>
</rule>
```

Test performed For testing that event, a custom C script has been coded as shown in Listing 3.49. The *fork* system call creates a new process each time it is called and it has been used to saturate the system resources.

Listing 3.49: Test for *uptime*

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    while(1)
        fork();
}
```

Test result Thus, every 2 minutes, the output of *uptime* is analysed through the decoders and the rule shown above. In particular, if the rule is triggered, an alert like the one shown in Figure 3.25, is generated.

Process states such as debugger attaching

Those events concern the attaching of the debugger to applications or processes. When a debugger is attached, the system call *ptrace* is called. It

Figure 3.26: Wazuh alert for GDB

provides a means by which one process observes and controls the execution of another, examining and changing the registers and the memory of the traced process. For monitoring those events, new auditd rules have been added and they are shown in the Listing 3.50.

Listing 3.50: Auditd rules for debugger attaching

```
1 -a always,exit -F arch=b64 -S ptrace -F a0=0x4 -k audit-wazuh-c
2 -a always,exit -F arch=b64 -S ptrace -F a0=0x5 -k audit-wazuh-c
3 -a always,exit -F arch=b64 -S ptrace -F a0=0x6 -k audit-wazuh-c
4 -a always,exit -F arch=b64 -S ptrace -k audit-wazuh-c
```

Test performed For testing the Wazuh and auditd capacities, the test at Listing 3.51 has been performed. At line 1, a simple program is started with a debugger, such as GDB and, then, it is killed.

Listing 3.51: Tests performed for validate debugger attaching

```
1 gdb -ex=r  /home/weseth/Codes/hello
2 kill -9 $(pidof gdb hello)
```

Test result In that case, the decoders and the rule used are the same as the other auditd logs, therefore the decoders at *0040-auditd_decoders.xml* file and the rule at Listing 3.15. The only difference is the alert that is shown in Figure 3.26.

Power state changes

About power state changes, they mean when the state of the machine changes such as a reboot, a shutdown, a hibernate, or a suspension. Those events are monitored thanks to auditd rules, as shown in the Listing 3.52.

Figure 3.27: Wazuh alert for *ossec* stop

Figure 3.28: Wazuh alert for *ossec* start

Listing 3.52: Auditd rules for power state changes

```
1 -w /sbin/poweroff -p x -k audit-wazuh-x
2 -w /sbin/reboot -p x -k audit-wazuh-x
3 -w /sbin/halt -p x -k audit-wazuh-x
```

Every rule is monitored for execution, due to the permission *-p x*, and Wazuh collects their logs thanks to the key *audit-wazuh-x*.

Test performed Those tests should be performed manually but, for clarity, those are listed in the Listing 3.53.

Listing 3.53: Tests performed for validate power state changes

```
1 reboot
2 power-off
3 halt
```

Test result In line 1 of the Listing 3.53, the system is rebooted and the result is the generation of two alerts that show the stopping and starting of the Wazuh agent as shown in Figure 3.27 and Figure 3.28.
The alert in Figure 3.27 comes from the decoder at Listing 3.54 and the rule at Listing 3.55.

Listing 3.54: Decoder used for *ossec* stop

```
1 <decoder name="ossec-agent-stop">
2   <parent>ossec</parent>
3   <type>ossec</type>
4   <prematch offset="after_parent">^Agent stopped:</prematch>
```

```
  <regex offset="after_prematch">^ '(\S+)'</regex>
  <order>extra_data</order>
  <fts>name, location, extra_data</fts>
</decoder>
```

Listing 3.55: Rule used for *ossec* stop

```
 <rule id="506" level="3">
  <if_sid>500</if_sid>
  <match>Agent stopped</match>
  <description>Ossec agent stopped.</description>
  <mitre>
    <id>T1562.001</id>
  </mitre>
  <group>pci_dss_10.6.1,pci_dss_10.2.6,gpg13_10.1,gdpr_IV_35.7.d,
   hipaa_164.312.b,nist_800_53_AU.6,nist_800_53_AU.14,nist_800_53_AU.5,
   tsc_CC7.2,tsc_CC7.3,tsc_CC6.8,</group>
</rule>
```

Differently, the alert in Figure 3.28 is the result of decoder in Listing 3.56 and rule at Listing 3.57.

Listing 3.56: Decoder used for *ossec* start

```
<decoder name="ossec-agent">
  <parent>ossec</parent>
  <type>ossec</type>
  <prematch offset="after_parent">^Agent started:</prematch>
  <regex offset="after_prematch">^ '(\S+)'</regex>
  <order>extra_data</order>
  <fts>name, location, extra_data</fts>
</decoder>
```

Listing 3.57: Rule used for *ossec* start

```
<rule id="503" level="3">
  <if_sid>500</if_sid>
  <match>Agent started</match>
  <description>Ossec agent started.</description>
  <group>pci_dss_10.6.1,pci_dss_10.2.6,gpg13_10.1,gdpr_IV_35.7.d,
   hipaa_164.312.b,nist_800_53_AU.6,nist_800_53_AU.14,nist_800_53_AU.5,
   tsc_CC7.2,tsc_CC7.3,tsc_CC6.8,</group>
</rule>
```

In case of *halt*, and *power-off* commands the only alert generated is the one at Figure 3.27.

Discretionary Access Control

DAC refers to a "type of security access control that grants or restricts object access via an access policy determined by an object's owner group and/or subjects"[29]. Those events are monitored thanks to the auditd rules of the Listing 3.58.

Listing 3.58: Auditd rules for DAC

```
1  -a always,exit -F arch=b64 -S chmod -F auid>=1000 -F auid!=-1 -k audit-
       wazuh-c
2  -a always,exit -F arch=b64 -S chown -F auid>=1000 -F auid!=-1 -k audit-
       wazuh-c
3  -a always,exit -F arch=b64 -S fchmod -F auid>=1000 -F auid!=-1 -k audit-
       wazuh-c
4  -a always,exit -F arch=b64 -S fchmodat -F auid>=1000 -F auid!=-1 -k audit-
       wazuh-c
5  -a always,exit -F arch=b64 -S fchown -F auid>=1000 -F auid!=-1 -k audit-
       wazuh-c
6  -a always,exit -F arch=b64 -S fchownat -F auid>=1000 -F auid!=-1 -k audit-
       wazuh-c
7  -a always,exit -F arch=b64 -S fremovexattr -F auid>=1000 -F auid!=-1 -k
       audit-wazuh-c
8  -a always,exit -F arch=b64 -S fsetxattr -F auid>=1000 -F auid!=-1 -k
       audit-wazuh-c
9  -a always,exit -F arch=b64 -S lchown -F auid>=1000 -F auid!=-1 -k audit-
       wazuh-c
10 -a always,exit -F arch=b64 -S lremovexattr -F auid>=1000 -F auid!=-1 -k
       audit-wazuh-c
11 -a always,exit -F arch=b64 -S lsetxattr -F auid>=1000 -F auid!=-1 -k
       audit-wazuh-c
12 -a always,exit -F arch=b64 -S removexattr -F auid>=1000 -F auid!=-1 -k
       audit-wazuh-c
13 -a always,exit -F arch=b64 -S setxattr -F auid>=1000 -F auid!=-1 -k audit-
       wazuh-c
```

All of them are monitored only if the Audit User Identifier is more than 1000 because the values smaller than this threshold are not used while the auid

Figure 3.29: Wazuh alert for *chown* command

should be not equal to -1 because this value is referred to auid not set. The system calls related to *chown* change the owner of files and directories while the ones referred to *chmod* concern the change of permissions. Differently. the system calls related to *setxattr* refer to the extended attributes associated with files and directories.

Test performed The test performed are the ones shown in the Listing 3.59

Listing 3.59: Tests performed for validate DAC

```
1 val = /etc/passwd
2 oldPerms=$(stat -c "%a" $val)
3 oldOwner=$(stat -c "%U" $val)
4 oldGroup=$(stat -c "%G" $val)
5 chown weseth:weseth $val
6 chmod 777 $val
7 chown $oldOwner:$oldGroup $val
8 chmod $oldPerms $val
```

From line 2 up to line 4 of Listing 3.59, the old permissions, the old owner, and the old group are saved into variables in order to restore them after the test performing in line 7 and line 8 of the same Listing. Also those events are monitored and, together with tests at line 5 and line 6, alerts are generated.

Test result The tests performed at line 5 and line 7 of Listing 3.59 trigger the decoders of *0040-auditd_decoders.xml* file and the rule at Listing 3.15 and the alerts generated are the same of the one at Figure 3.29.
Similarly, the tests performed at line 6 and line 8 trigger the same decoders and rule and the alerts generated are as well as the one shown at Figure 3.30.

Figure 3.30: Wazuh alert for *chmod* command

User and Group Management

User and Group management refer to adding, deleting, and updating of user and group profiles. To achieve those detections new auditd rules have been added as shown in Listing 3.60.

Listing 3.60: Auditd rules for user and group management

```
-w /usr/sbin/groupadd -p x -k audit-wazuh-c -k audit-wazuh-x
-w /usr/sbin/groupmod -p x -k audit-wazuh-c -k audit-wazuh-x
-w /usr/sbin/groupdel -p x -k audit-wazuh-c -k audit-wazuh-x
-w /usr/sbin/useradd -p x -k audit-wazuh-c -k audit-wazuh-x
-w /usr/sbin/userdel -p x -k audit-wazuh-c -k audit-wazuh-x
-w /usr/sbin/usermod -p x -k audit-wazuh-c -k audit-wazuh-x
-w /usr/sbin/adduser -p x -k audit-wazuh-c -k audit-wazuh-x
-w /usr/bin/passwd -p x -k audit-wazuh-c -k audit-wazuh-x
-w /usr/sbin/chpasswd -p x -k audit-wazuh-c -k audit-wazuh-x
-w /bin/gpasswd -p x -k audit-wazuh-x
```

All these rules concern the commands to deal with users and groups in Linux, also considering passwords.

Test performed The Listing 3.61 describes what tests have been performed to validate the detections of user and group management events.

Listing 3.61: Tests performed for validate user and group management

```
groupadd newGroup
groupmod -n Group newGroup
useradd -g Group newUser
usermod -u 10000 newUser
usermod -g weseth newUser
userdel newUser
groupdel Group
adduser secondNewUser --gecos "First Last,RoomNumber,WorkPhone,HomePhone"
    --disabled-password --force-badname --no-create-home
```

```
9  set -e
10 PASSWD=$(date | md5sum | cut -c1-8)
11 echo "secondNewUser:$PASSWD" | chpasswd
12 userdel secondNewUser
```

The command *groupadd* in line 1 creates a new group in the system, the command *groupmod* in line 2 modifies the name of the just created group, the commands *useradd* in line 3 and *adduser* in line 8 create new users in the system but if in the first case no password is set by default, in the second one the command is interactive and it waits for password insertion by the user; for that reason the option *–disable-password* is used while the option *–force-badname* is used to avoid checks in the input name of the user; moreover, the option *–gecos* is used to avoid interactive request for the gecos information, while the option *–no-create-home* is used to avoid the creation of the home directory for that new user. The command *usermod* at line 4 and line 5 changes, respectively, the group and the user identifier, the command *userdel* at line 6 and line 12 deletes just created users, the command *groupdel* at line 7 deletes the group while in line 9 and line 10 a new password is created and then in line 11 the password has been set for a specific user: that has been done to insert the command in a bash script but also the *passwd* command is monitored.

Test result Because logs come from auditd rules, all those tests trigger the decoders in *0040-auditd_decoders.xml* file and the 3.15 rule but some of those commands are also monitored by Wazuh by default and other decoders and rules are triggered, also considering that the commands above write files monitored by the auditd rules of Listing 3.2. For command *groupadd*, the decoder used is shown in Listing 3.62 and the rule is at Listing 3.63.

Listing 3.62: Decoder used for *groupadd* command

```
1 <decoder name="groupadd-fields">
2     <parent>groupadd</parent>
3     <prematch offset="after_parent">new group: name=</prematch>
4     <regex offset="after_prematch">(\S+), GID=(\S+)</regex>
5     <order>user,gid</order>
6 </decoder>
```

Figure 3.31: Wazuh auditd alert for *groupadd* command

Figure 3.32: Wazuh default alert for *groupadd* command

Listing 3.63: Rule used for *groupadd* command

```
<rule id="5901" level="8">
  <match>^new group</match>
  <description>New group added to the system.</description>
  <group>pci_dss_10.2.7,pci_dss_10.2.5,pci_dss_8.1.2,gpg13_4.13,
   gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164.312.b,hipaa_164.312.a.2.I,
   hipaa_164.312.a.2.II,nist_800_53_AU.14,nist_800_53_AC.7,nist_800_53_AC
   .2,nist_800_53_IA.4,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

The alerts created for auditd log analysis and Wazuh rule triggered are shown in Figure 3.32 and Figure 3.31, respectively.
Moreover, the *groupadd* command writes in the */etc/group* file, that is monitored; therefore, the alert in Figure 3.33 is generated.
Besides the auditd alert, shown in Figure 3.34, the command *groupmod* modifies also the */etc/group* file , generating the alert in Figure 3.33.
In addition to the auditd analysis and consequent alert, the command *useradd* generates two other kinds of alerts: default alert, and writing files alerts. The first one is generated by the decoder at Listing 3.64 and the rule at Listing 3.65.

Listing 3.64: Decoder used for *useradd* command

```
<decoder name="useradd-fields">
  <parent>useradd</parent>
  <prematch offset="after_parent">new user: </prematch>
  <regex offset="after_prematch">name=(\S+), UID=(\S+), GID=(\S+), home
   =(\S+), shell=(\S+)</regex>
  <order>user,uid,gid,home,shell</order>
</decoder>
```

Figure 3.33: Wazuh alert for *groupadd* writing in */etc/group*

Figure 3.34: Wazuh auditd alert for *groupmod* command

Listing 3.65: Rule used for *useradd* command

```
<rule id="5902" level="8">
  <match>^new user|^new account added</match>
  <description>New user added to the system.</description>
  <mitre>
    <id>T1136</id>
  </mitre>
  <group>pci_dss_10.2.7,pci_dss_10.2.5,pci_dss_8.1.2,gpg13_4.13,
   gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164.312.b,hipaa_164.312.a.2.I,
   hipaa_164.312.a.2.II,nist_800_53_AU.14,nist_800_53_AC.7,nist_800_53_AC
   .2,nist_800_53_IA.4,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

The alerts generated by auditd log analysis and default Wazuh rule are shown in Figure 3.35 and Figure 3.36, respectively.
As in the *groupadd* case, also the *useradd* command writes some files such as */etc/group*, */etc/passwd*, and */etc/shadow* and the corresponding alerts are shown in Figure 3.37.

The command *usermod* triggers the auditd decoders and rule and, in addition, the update of */etc/passwd*, and */etc/shadow* files. The auditd rule is shown at Figure 3.38, while the alerts for the writings are in Figure 3.39 and Figure 3.40.

The command *userdel* is responsible of triggering the decoder at Listing 3.66 and the rule at Listing 3.67, in addition to auditd ones.

Figure 3.35: Wazuh auditd alert for *userpadd* command

Figure 3.36: Wazuh default alert for *useradd* command

Listing 3.66: Decoder used for *userdel* command

```
<decoder name="open-userdel">
  <program_name>userdel</program_name>
  <regex>user removed: name=(\S+)$|delete user '(\S+\w)'</regex>
  <order>srcuser</order>
</decoder>
```

Listing 3.67: Rule used for *userdel* command

```
<rule id="5903" level="3">
  <match>^delete user|^account deleted|^remove group</match>
  <description>Group (or user) deleted from the system.</description>
  <mitre>
    <id>T1531</id>
  </mitre>
  <group>pci_dss_10.2.7,pci_dss_10.2.5,pci_dss_8.1.2,gpg13_4.13,
   gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164.312.b,hipaa_164.312.a.2.I,
   hipaa_164.312.a.2.II,nist_800_53_AU.14,nist_800_53_AC.7,nist_800_53_AC
   .2,nist_800_53_IA.4,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

The consequent alerts are the auditd one in Listing 3.41, the default one in Figure 3.42, and the writings of */etc/group*, */etc/passwd*, and */etc/shadow* in Figure 3.43.

The command *groupdel* triggers only the auditd decoders and rule and the writing of the */etc/group* file, as shown in Figure 3.44 and 3.45, respectively.

The command *useradd* used in line 3 of Listing 3.61 generates many alerts due to the operations it performs: at first, it creates a new group as shown for

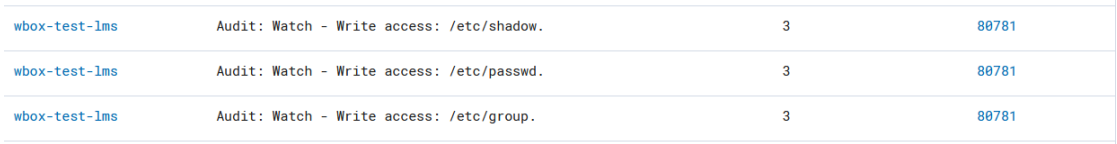

Figure 3.37: Wazuh alerts for *groupadd* writings

Figure 3.38: Wazuh auditd alert for *usermod* command

the *groupadd* command, the command *chfn*[1] is called, the user is added, and, finally, it is also modified with the command *usermod*. All those commands trigger the decoders in *0040-auditd_decoders.xml* file and the rule in Listing 3.15. Moreover, the *chfn* command triggers both an auditd rule and a Wazuh default: the decoder is in Listing 3.68 while the rule is in Listing 3.69.

Listing 3.68: Decoder used for *chfn* command

```
<decoder name="chfn-fields">
  <parent>chfn</parent>
  <prematch offset="after_parent">changed user</prematch>
  <regex offset="after_prematch">'(\S+)'</regex>
  <order>user</order>
</decoder>
```

Listing 3.69: Rule used for *chfn* command

```
 <rule id="5904" level="8">
  <match>^changed user</match>
  <description>Information from the user was changed.</description>
  <mitre>
    <id>T1098</id>
  </mitre>
  <group>pci_dss_10.2.7,pci_dss_10.2.5,pci_dss_8.1.2,gpg13_4.13,
   gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164.312.b,hipaa_164.312.a.2.I,
   hipaa_164.312.a.2.II,nist_800_53_AU.14,nist_800_53_AC.7,nist_800_53_AC
   .2,nist_800_53_IA.4,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

[1]This command is in charge to show the user information, in the field GECOS

Figure 3.39: Wazuh alert for *usermod* writing of */etc/passwd* file

Figure 3.40: Wazuh alert for *usermod* writing of */etc/shadow* file

As a consequence, the auditd alerts of the above commands have been shown previously while the auditd alert about the *chfn* command is in Figure 3.46, and the alert generated by the default Wazuh alert for the same command is shown in Figure 3.36.
The last command tested, *chpasswd*, triggers both auditd rule and Wazuh default one. That is shown in Listing 3.70.

Listing 3.70: Rule used for *chpasswd* command

```
<rule id="5555" level="3">
  <match>: password changed for</match>
  <description>PAM: User changed password.</description>
  <group>pci_dss_8.1.2,pci_dss_10.2.5,gpg13_4.13,gpg13_7.10,gdpr_IV_35.7.
    d,gdpr_IV_32.2,hipaa_164.312.a.2.I,hipaa_164.312.a.2.II,hipaa_164.312.
    b,nist_800_53_AC.2,nist_800_53_IA.4,nist_800_53_AU.14,nist_800_53_AC
    .7,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,</group>
</rule>
```

Alerts generated are the auditd one, shown in Figure 3.48, and the Wazuh default one shown in Figure 3.49.

Firewall violations

The firewall used in the Weseth boxes is UFW, that has been created to deal in a simpler way *iptables*. Its logs are analysed by default by Wazuh, the only things that it needs is the forwarding of the logs and that is achieved thanks to the addition of a XML tag in the configuration of each Wazuh agent as shown in Listing 3.71.

Listing 3.71: Configuration for the forwarding of ufw.log

Figure 3.41: Wazuh auditd alert for *userdel* command

Figure 3.42: Wazuh default alert for *userdel* command

```
1 <localfile>
2   <log_format>syslog</log_format>
3   <location>/var/log/ufw.log</location>
4 </localfile>
```

Test performed About the test performed, firstly a new rule has been appended to the firewall blocking all the connection that arrives at port 1234 and to add that rule the command line 1 command of Listing 3.72 has been run.

Listing 3.72: UFW command for adding a new rule

```
1 sudo ufw deny in from any port 1234
2 nc -l 127.0.0.1 1234
```

Right after the port 1234 in localhost is opened and it is ready for receiving new requests, as specified in line 2 of Listing 3.72. From another device, the request *nc <boxIpAddress> 1234* is run but the connection is never opened.

Test result The decoders triggered are shown in Listing 3.73 while the rule triggered is shown in Listing 3.74.

Listing 3.73: Decoders used for UFW

```
1 <decoder name="ufw">
2   <parent>kernel</parent>
```

wbox-test-lms	Audit: Watch - Write access: /etc/shadow.	3	80781
wbox-test-lms	Audit: Watch - Write access: /etc/passwd.	3	80781
wbox-test-lms	Audit: Watch - Write access: /etc/group.	3	80781

Figure 3.43: Wazuh alerts for *userdel* writings

wbox-test-lms	Audit: Command: /usr/sbin/groupdel.	3	80792

Figure 3.44: Wazuh auditd alert for *groupdel* command

```
  <type>firewall</type>
  <prematch>^[\s*\d+.\d+] [\.*] IN=</prematch>
  <regex>^[\s*\d+.\d+] [(\.*)] \.+ SRC=(\S+) DST=(\S+)</regex>
  <regex> \.+ PROTO=(\w+) </regex>
  <order>action,srcip,dstip,protocol</order>
</decoder>

<decoder name="ufw">
  <parent>kernel</parent>
  <type>firewall</type>
  <regex offset="after_regex">^SPT=(\d+) DPT=(\d+) </regex>
  <order>srcport,dstport</order>
</decoder>
```

Listing 3.74: Rule used for UFW

```
<rule id="100400" level="7">
  <if_sid>4100</if_sid>
  <dstip>!224.0.0.251</dstip>
  <action>UFW BLOCK</action>
  <description>A ufw rule has been violated</description>
</rule>
```

About the rule it has been added in the *local_rules.xml* where the destination IP cannot be 224.0.0.251 because it is a broadcast address and the interest is only on the block action of the firewall. The consequent alert is the one shown in Figure 3.50.

Figure 3.45: Wazuh alerts for *groupdel* writings

Figure 3.46: Wazuh auditd alert for *chfn* command

Dangerous and suspicious activities

Those event concern the monitoring of dangerous and suspicious activities that are run in the Weseth box.

Test performed The test have been performed are shown in Listing 3.75 and in Listing 3.76.

Listing 3.75: Tests performed for validate danger and suspicious activities

```
nft list ruleset
traceroute 192.168.5.106
printenv
arp -a
capsh --print
chroot /home/weseth/jail /bin/bash
start-stop-daemon --start --exec --name auditd
vipw -s&
vigr -s&
kill -9 $(pidof vipw)
kill -9 $(pidof vigr)
busctl&
kill -9 $(pidof busctl)
nice -10 busctl&
renice -n 0 -p $(pidof busctl)
kill -9 $(pidof busctl)
ln -s /home/weseth/testLms/testLms.sh /home/weseth/testLms/test
rm /home/weseth/testLms/test
loginctl list-sessions
nohup ping 192.168.5.106&
```

Figure 3.47: Wazuh default alert for *chfn*

Figure 3.48: Wazuh auditd alert for *chpasswd* command

```
kill -9 $(pidof ping)
rm nohup.out
openssl genrsa -out private.key 2048
rm private.key
run-parts --list --regex '^p.*d$' /etc
timedatectl status
ls -al  /etc | xargs
dbus-send --help
service sshd status
```

Listing 3.76: Tests performed for validate detection of vulnerable binaries

```
cp -p /usr/bin/w /usr/bin/w.copy
# Replace the content of the original binary with the content of Listing
    3.77
/usr/bin/w.copy
```

Listing 3.77: Script that replace the original binary

```
#!/bin/bash
echo "`date` this is evil" > /tmp/trojan_created_file
echo 'test for /usr/bin/w trojaned file' >> /tmp/trojan_created_file
```

Test result Each one of the commands in the above Listing triggers the decoders in *0040-auditd_decoders.xml* file and the rule at Listing 3.78.

Listing 3.78: Rule used for suspicious commands

```
<rule id="80789" level="3">
  <if_sid>80700</if_sid>
```

wbox-test-lms	PAM: User changed password.	3	5555

Figure 3.49: Wazuh default alert for *chpasswd*

wbox-test-SEi	A ufw rule has been violated	7	100400

Figure 3.50: Wazuh alert for UFW rule violation

```
3   <list field="audit.key" lookup="match_key_value" check_value="execute">
      etc/lists/audit-keys</list>
4   <description>Audit: Watch - Execute access: $(audit.file.name).</
      description>
5   <group>audit_watch_execute,gdpr_IV_30.1.g,</group>
6 </rule>
```

The next part is organized as follows: alert figure followed by the description of each command and where, in the test, it is run.
The command *nft* run in line 1 of Listing 3.75 provides the alert at Figure 3.51 and it is in charge to set up, maintain, and inspect packet filtering and classification rules in Linux systems.

The command *traceroute* run in line 2 of Listing 3.75 provides the alert at Figure 3.52 and it is in charge to track the route packets taken from an IP network on their way to a given host.
The command *printenv* run in line 3 of Listing 3.75 provides the alert at Figure 3.53 and it is in charge to display the values of environment variables.

The command *arp* run in line 4 of Listing 3.75 provides the alert at Figure 3.54 and it is in charge to manipulate and dump the ARP cache.
The command *capsh* run in line 5 of Listing 3.75 provides the alert at Figure 3.55 andi it is a wrapper that allows to explore the Linux capability, to create environment, and to provide debugging features.
The command *chroot* run in line 6 of Listing 3.75 provides the alert at Figure 3.56 and it is in charge to change the root directory.

The command *start-stop-daemon* run in line 7 of Listing 3.75 provides the

Figure 3.51: Wazuh alert for *nft* command

Figure 3.52: Wazuh alert for *traceroute* command

alert at Figure 3.57 and it is in charge to control the creation and termination of system-level processes.
The command *vipw* run in line 8 of Listing 3.75 provides the alert at Figure 3.58 and it is in charge to edit the */etc/passwd* file or, with the *-s* flag, its shadow version */etc/shadow*.
The command *vigr* run in line 9 of Listing 3.75 provides the alert at Figure 3.59 and it is in charge to edit the */etc/group* file or, with the *-s* flag, its shadow version */etc/gshadow*.

The command *busctl* run in line 12 of Listing 3.75 provides the alert at Figure 3.60 and it is in charge to introspect and monitor the D-Bus bus, that is an Inter-Process communication protocol in Linux.
The command *nice* run in line 14 of Listing 3.75 provides the alert at Figure 3.61 and it is in charge to start a process with a different priority respect to the default one.
The command *renice* run in line 15 of Listing 3.75 provides the alert at Figure 3.62 and it is in charge to change the priority of an already run process.

The command *ln* run in line 17 of Listing 3.75 provides the alert at Figure 3.63 and it is in charge to create hard or soft links between files.
The command *loginctl* run in line 19 of Listing 3.75 provides the alert at Figure 3.64 and it is in charge to introspect and control the state of the *systemd* login manager service.
The command *nohup* run in line 20 of Listing 3.75 provides the alert at Figure 3.65 and it is in charge to keep processes running even if the signal Hang Up is received..
The command *openssl* run in line 23 of Listing 3.75 provides the alert at Figure 3.66 and it is an open source implementation of the SSL and TLS protocols.

Figure 3.53: Wazuh alert for *printenv* command

Figure 3.54: Wazuh alert for *arp* command

The command *run-parts* run in line 25 of Listing 3.75 provides the alert at Figure 3.67 and it is in charge to run all the executables inside a specified directory.
The command *timedatectl* run in line 26 of Listing 3.75 provides the alert at Figure 3.68 and it is in charge to control date, time, and time zones of the Linux system. Moreover, it could be used to enable automatic system clock synchronization.
The command *xargs* run in line 27 of Listing 3.75 provides the alert at Figure 3.69 and it allows to execute another command specifying as parameters what is written in standard input.
The command *dbus-send* run in line 28 of Listing 3.75 provides the alert at Figure 3.70 and it allows to send a message to the Dbus message bus.
The command *service* run in line 29 of Listing 3.75 provides the alert at Figure 3.71 and it is in charge to control *SysVinit*[2] services through *SysVinit* scripts.
Anyway, in the Listing 3.76 another test has been performed. As seen in the line 1, a bin is copied and than it is replaced with the content of Listing 3.77. The result is the triggering of the rule at Listing 3.79 and the generation of the alert at Figure 3.72.

Listing 3.79: Rule used for suspicious binary

```
1 <rule id="510" level="7">
2   <if_sid>509</if_sid>
3   <description>Host-based anomaly detection event (rootcheck).</
      description>
4   <group>rootcheck,pci_dss_10.6.1,gdpr_IV_35.7.d,</group>
5   <!-- <if_fts />  -->
6 </rule>
```

[2]sysvinit is a collection of System V-style init programs

wbox-test-lms	Audit: Watch - Execute access: /usr/sbin/capsh.	3	80789

Figure 3.55: Wazuh alert for *capsh* command

wbox-test-lms	Audit: Watch - Execute access: /usr/sbin/chroot.	3	80789

Figure 3.56: Wazuh alert for *chroot* command

Indeed, the command injection written in line 2 of Listing 3.77 has been detected by Wazuh as a root check, as seen in the generated alert.

Figure 3.57: Wazuh alert for *start-stop-daemon* command

Figure 3.58: Wazuh alert for *vipw* command

Figure 3.59: Wazuh alert for *vigr* command

Figure 3.60: Wazuh alert for *busctl* command

Figure 3.61: Wazuh alert for *nice* command

Figure 3.62: Wazuh alert for *renice* command

Figure 3.63: Wazuh alert for *ln* command

Figure 3.64: Wazuh alert for *loginctl* command

Figure 3.65: Wazuh alert for *nohup* command

Figure 3.66: Wazuh alert for *openssl* command

wbox-test-lms	Audit: Watch - Execute access: /usr/bin/run-parts.	3	80789

Figure 3.67: Wazuh alert for *run-parts* command

wbox-test-lms	Audit: Watch - Execute access: /usr/bin/timedatectl.	3	80789

Figure 3.68: Wazuh alert for *timedatectl* command

wbox-test-lms	Audit: Watch - Execute access: /usr/bin/xargs.	3	80789

Figure 3.69: Wazuh alert for *xargs* command

wbox-test-lms	Audit: Watch - Execute access: /usr/bin/dbus-send.	3	80789

Figure 3.70: Wazuh alert for *dbus-send* command

wbox-test-lms	Audit: Watch - Execute access: /usr/sbin/service.	3	80789

Figure 3.71: Wazuh alert for *service* command

wbox-test-SEi	Host-based anomaly detection event (rootcheck).	7	510

Figure 3.72: Wazuh alert for *rootcheck* detected

Conclusion

The paper has aimed to find preventive solutions against cyber-attacks. One of them has been explored trying to consider what are its peculiarities and its challenges. The Host-based Intrusion Detection System has been explained as a good solution to achieve the main purpose of the thesis. Basically, the source information, or logs, are the most important basic knowledge for those tools. Moreover, the Intrusion Detection System is the answer to detect new attacks, starting from information coming from logs, and analyzing those using rules and decoders, as in the case of the chosen tool: Wazuh. Therefore, in the first chapter, all families of Intrusion Detection Systems have been presented matching their source information going in deeply for the Host-based one, such as Wazuh is of that category. In the second chapter, Wazuh has been compared with its competitors such as Sagan, Samhain, and Event Log Manager. Above all, the best comparison has been made with its similar: OSSEC. Those comparisons have been performed based on different discriminants such that the multiplatform feature, File Integrity Monitoring, Log Analysis, and Rootkit detection modules, and, in the end, the chosen solution has been Wazuh.
The third chapter describes how Wazuh has been implemented in the case study, the Weseth platform. Thereby, a test environment has been built up with two Weseth boxes where one acts as Weseth Server while the other one is the device to be monitored. To fit with the platform needs, Wazuh has been implemented with the Linux service *Auditd* integrating its functionality in the tool. For the monitoring purpose, certain security events have been monitored such as the integrity of files and directories, kernel module and parameters tuning, privilege escalation attempts, statistical resource usage, and so on.
Although the great capabilities of Wazuh, it alone is not able to deal with

all cyber-security problems and, for those reasons, some improvements are possible. Firstly, an alert correlation engine could be added to Wazuh trying to extract more useful information from the alert created by the Intrusion Detection tool; moreover, Wazuh is a Host-based IDS and it does not covers the threats coming from networks, thus, for that reason, the introduction of Network-based tool could improve the Wazuh capabilities. Finally, the introduction of new decoders and rules in the Wazuh ruleset and the integrations with other anti-virus tools could help the discovery capability of Wazuh of new attack patterns and vulnerabilities.

Bibliography

[1] *Logging vs Monitoring: Best Practices for Integration*. https://www.appdynamics.com/product/how-it-works/application-analytics/log-analytics/monitoring-vs-logging-best-practices.

[2] Fanny Rivera-Ortiz and Liliana Pasquale. «Automated modelling of security incidents to represent logging requirements in software systems». In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 2020, pp. 1–8.

[3] *Logging Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html.

[4] *Security Logging*. https://owasp.org/www-project-proactive-controls/v3/en/c9-security-logging.html.

[5] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. «Intrusion detection system: A comprehensive review». In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24.

[6] Hervé Debar, Marc Dacier, and Andreas Wespi. «Towards a taxonomy of intrusion-detection systems». In: *Computer networks* 31.8 (1999), pp. 805–822.

[7] Animesh Patcha and Jung-Min Park. «An overview of anomaly detection techniques: Existing solutions and latest technological trends». In: *Computer networks* 51.12 (2007), pp. 3448–3470.

[8] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. «Anomaly-based network intrusion detection: Techniques, systems and challenges». In: *computers & security* 28.1-2 (2009), pp. 18–28.

[9] Juan M Estevez-Tapiador, Pedro Garcia-Teodoro, and Jesus E Diaz-Verdejo. «Anomaly detection methods in wired networks: a survey and taxonomy». In: *Computer Communications* 27.16 (2004), pp. 1569–1584.

[10] Nong Ye, Yebin Zhang, and Connie M Borror. «Robustness of the Markov-chain model for cyber-attack detection». In: *IEEE transactions on reliability* 53.1 (2004), pp. 116–123.

[11] Seyed Ali Mirheidari, Sajjad Arshad, and Rasool Jalili. «Alert correlation algorithms: A survey and taxonomy». In: *International Symposium on Cyberspace Safety and Security*. Springer. 2013, pp. 183–197.

[12] Huwaida Tagelsir Elshoush and Izzeldin Mohamed Osman. «Alert correlation in collaborative intelligent intrusion detection systems—A survey». In: *Applied Soft Computing* 11.7 (2011), pp. 4349–4365.

[13] *Drivesec Weseth*. `https://www.drivesec.com/weseth/`.

[14] Leian Liu, Zuanxing Yin, Yuli Shen, Haitao Lin, and Hongjiang Wang. «Research and design of rootkit detection method». In: *Physics Procedia* 33 (2012), pp. 852–857.

[15] *Aide*. `https://aide.github.io/`.

[16] *Event Log Analyzer*. `https://www.manageengine.com/products/eventlog/help/StandaloneManagedServer-UserGuide/Introduction/about-eventlog-analyzer.html`.

[17] *IBM MaaS360*. `https://www.ibm.com/docs/en/maas360?topic=guide-getting-started-maas360-portal`.

[18] *Lacework Threat Detection*. `https://docs.lacework.com/onboarding/lacework-overview`.

[19] *Ossec*. `https://www.ossec.net/docs/docs/manual/non-technical-overview.html`.

[20] *Sagan*. `https://sagan.readthedocs.io/en/latest/what-is-sagan.html`.

[21] *Samhain*. `https://www.la-samhna.de/samhain/manual/intro.html`.

[22] *Solarwinds SEM*. `https://documentation.solarwinds.com/en/success_center/sem/content/getting_started_guide/gsg-introduction.htm`.

[23] *Tripwire*. https://www.tripwire.com/state-of-security/tripwire-news/tripwire-products-reference-guide/.

[24] *XDR Definition*. https://www.trellix.com/en-us/security-awareness/endpoint/what-is-xdr.html.

[25] *Wazuh*. https://documentation.wazuh.com/current/index.html.

[26] *Filebeat*. https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html.

[27] *Auditd*. https://man7.org/linux/man-pages/man8/auditd.8.html.

[28] *Uptime*. https://www.ibm.com/docs/en/aix/7.2?topic=u-uptime-command.

[29] *DAC*. https://www.techopedia.com/definition/229/discretionary-access-control-dac.