UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAURA MAGISTRALE IN INGEGNERIA INFORMATICA

IoT oriented SIEM tools

Relatore: Prof. Migliardi Mauro

Laureando: Maggiolo Nicola

ANNO ACCADEMICO: 2021-2022

Data di laurea 28 Febbraio 2022

Contents

List of Figures

1 Introduction

From the very beginning, man felt the need to improve his living conditions, looking for solutions to facilitate and improve his quality of life for his survival.

Nowadays, **globalization** has made it easy for people to get in touch with other people, leading to the worldwide dissemination of ideas and issues, thanks to new media of communication. The ever-increasing need to improve the quality of life and the greater possibility of quickly exchanging knowledge and information have led to rapid growth in the technologies available to men.
Generally speaking, technologies help to improve the quality of life, for example by helping to execute complex tasks in a simpler way or by making the environment safer and cleaner.

Technological development has led to the creation of many new and increasingly efficient tools that can help and simplify people's lives. New tools can be applied in many fields, for example, they can allow companies to improve and increase productivity, improve the monitoring of a patient's health, help the environment by reducing waste and pollution such as noise, and light, and much more.

Nowadays, devices can connect and communicate data. An example is **IoT devices**, technological devices that are able to communicate information collected from the environment with a high degree of automation, transferring them through networks. IoT devices that are always connected and increasingly reliable and fast wireless networks make it easy to collect large amounts of data with high accuracy.

In addition to the development of new devices, protocols are being developed to transport data efficiently with low power consumption. IoT devices are generally devices that have a low battery capacity and therefore consumption must be minimized. Examples of protocols that are used, and which are discussed in this paper, are *Message Queuing Telemetry Transport (MQTT)*, *Hypertext Transfer Protocol (HTTP)*, *Constrained Application Protocol (CoAP)*, etc...

The introduction of these new technologies has created new ***vulnerabilities*** within complex systems, allowing an attacker to breach them more easily. Attackers use these devices, which generally don't have important protections because they are composed of minimal hardware. Generally, the attackers' goal is to capture data, create malfunctions, steal sensitive and personal information, and more.

In order to protect and limit the actions of possible attackers, new software has been developed to neutralize or reduce vulnerabilities in a complex system composed of these devices.
An example of software that belongs to this category are **SIEMs**, which are discussed in the following chapters. They make it possible to analyze *real-time data* and *logs* of what is happening within a system.
They give the possibility of creating a ***history*** of the collected information by the system, indexing the information to allow efficient and rapid analysis. They also allow the collected data to be visualized using dashboards.
With the introduction of *artificial intelligence*, these tools have become more precise, allowing the automatic creation of thresholds that generate alerts in critical situations if exceeded. These tools may also be able to autonomously analyze the environment and identify any vulnerability within the system, and even respond to certain situations autonomously.

In this thesis, IoT devices and SIEM software are **combined**. As previously mentioned, there are numerous applications in which IoT devices can be used. In the example described below, they are applied in the ***agricultural*** sector by simulating a greenhouse to which attacks are made on it to interrupt its correct functioning. The SIEM has the task of helping monitor the network packets to identify possible attacks and analyse the data collected, to control the situation inside the greenhouse. In this way, a supervisor can visualise and understand the state of the network and protect it from attackers by identifying any present vulnerabilities; he can also understand trends in environmental factors within the greenhouse, being able to evaluate any situations where values do not remain within the expected range.

Once the tests have been executed by simulating some sensors that collect the state of the greenhouse environment and launching attacks against the system, the use of SIEM in the IoT environment is evaluated to improve the **effectiveness of the security and safety** of the system.

2 Network and Computer Security

The purpose of **computer security** is to protect the valuable parts of an information system, such as hardware, software, data or processes. Some examples may be a computer device, an operating system, and its functions, an installed application, or a self-developed program. The value of each asset is defined by the owner, for example, according to the monetary impact or sentimental value.
The goal of computer security is to protect important assets from attackers, who aim to damage or steal them.
Two definitions must be given:

- **Vulnerability**: a weakness in the system which might be exploited to cause loss or harm. [9]
- **Threat**: a set of circumstances which has the potential to cause loss or harm. [9]

An attacker tries to exploit system vulnerabilities to break into it. *Denial-of-Service* is an example of an attack which takes advantage of weaknesses.

Controls and **countermeasures** can work as protection by creating procedures or using techniques which allow to eliminate or reduce the vulnerabilities of a system.
The relationship between threats, controls, and vulnerabilities can be defined as:

*"A **threat** is blocked by **control** of a **vulnerability**"* [9]

Basic security properties are defined in the ISO 7498-2 and are five:

- ***Availability***: only authorized users can *use* specific assets.
- ***Integrity***: only authorized users can *modify* specific assets.
- ***Confidentiality***: only authorized users can *visualize* specific assets.
- ***Authentication***: ability to *confirm the sender* of a network communication.
- ***Non-repudiation***: a *sender can't deny* its network communications.

The first 3 properties are defined as ***C-I-A triad*** or ***security triad*** (Figure 2.1). The other two properties can be considered an *extension* of the security definition to *network devices*.

[https://blog.jamestyson.co.uk/wp-content/uploads/2019/09/CIA-triad.png]

Figure 2.1: C-I-A triad

The US Department of Defense has introduced an additional property: ***auditability***. It refers to the possibility of tracking all the actions performed on an asset.

The harm can be characterized by 4 acts: interception, interruption, modification, and fabrication.
A threat represents a possible cause of harm. Threats can be:

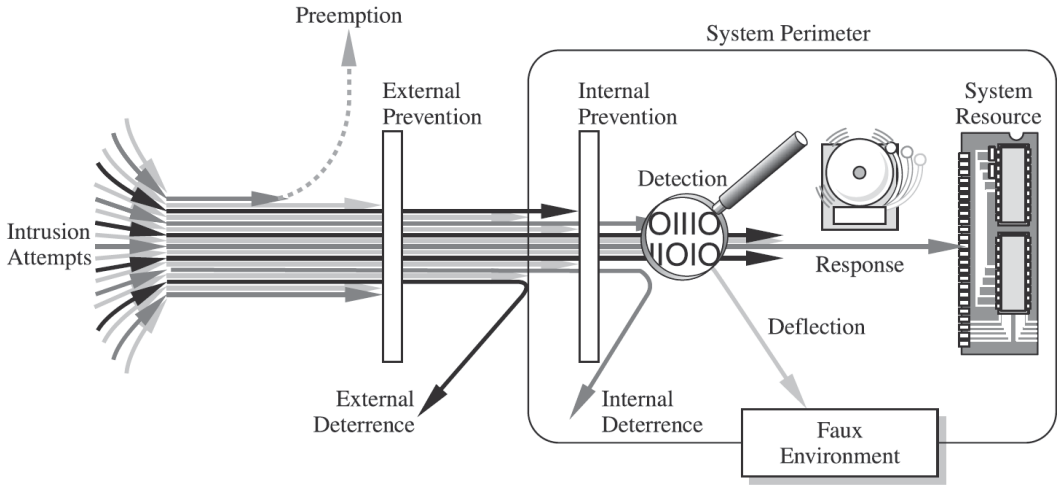
- *Non-human*: not human-caused, such as loss of electrical power or components failure.
- *Human*: threats can be malicious, i.e. intentional damage, or benign, i.e. accidental damage or damage that a user unintentionally causes and generates a dangerous situation for the system, for example, by opening unwanted e-mails or files containing a virus.

In general, an attacker to hack a system needs 3 things:

- *Method*: skill and knowledge to carry out an attack.
- *Opportunity*: possibility to execute an attack.
- *Motive*: a reason to perform an attack.

Vulnerabilities are weaknesses which can lead to attacks. Security analysts identify the set of real and potential vulnerabilities of a system as **attack surfaces** which can be physical hazards, mistakes, data stolen from the system, and more. It is important to take countermeasures and check possible vulnerabilities to stop threats that may arise due to them.
In order to protect a system, it is important to: [9]

- ***prevent***: solve vulnerabilities which can lead to an attack.
- ***deter***: make an attack difficult for an attacker.
- ***deflect***: make other targets more attractive.
- ***mitigate***: control the attacker's damage.
- ***detect***: identify when an attack occurred.
- ***recover***: recover data after an attack.



[Figure 1-12 in the book [9]]

Figure 2.2: Effects of controls

It's important to balance the costs and effectiveness of the countermeasure.

One between software solutions to reduce and control vulnerabilities are SIEMs. These tools can assess a system environment and help security teams to discover attack surfaces and analyse attacks occurring to a system by identifying system weaknesses.

3 Security Information and Event Management

A *Security System (SS)* is a procedure which secures an asset through a computer network.

A similar system is called *Security Monitoring System (SMS)* with different performance. It is an automated method which supervises a system environment by identifying dubious or unauthorized situations, alerting the system owner, or making decisions.
Security Monitoring Systems focus on reducing attacks by decreasing vulnerabilities and threats by elaborating and analysing specific indicators.

3.1 SIEM Composition

The first appearance of the term SIEM was in 2005 in a Gartner's SIEM report written by Amrit T. Williams and Mark Nicolett.
The acronym ***SIEM*** stands for *Security Information and Event Management*. The goal of SIEM is to manage the security of a system using a centralized interface.

It is a composition of two arguments of ITIL (Information Technology Infrastructure Library) security, which are:

- ***Security Event Management (SEM)***: management of event security. SEM monitors and manages real-time events inside a network, providing event correlation and aggregation. It allows to monitor and alert unusual occurrences.
- ***Security Information Management (SIM)***: management of information security. SIM collects and organizes logs from devices which compose a system, but not in real-time. It allows to create customized logs by collecting data.

SIEM is the result of the combination between ***SEM*** and ***SIM***. It represents a complete tool to manage network security and prevent real-time attacks and diagnostic of past events. Thanks to those features, SIEM has become one of the most important software solutions for organizations which want to control their systems in real-time.

3.2 SIEM Generations

3.2.1 Early generations

Early generations of SIEM had the task of reducing false positives of the Network Intrusion Detection System (NIDS) which generated a large number of alerts. First SIEM's had important limitations in terms of scalability, which is an important feature for these tools.

Subsequent generations improved data management. New SIEMs were capable of handling large amounts of historical logs and extracting information using historical logs and real-time events.

These first generations of SIEM had important limitations:

- *Complexity*: difficult to understand how to manage events and logs and set important rules.
- *Time Consuming*: long time to initialize a SIEM.
- *Expensive*: high costs to initialize SIEM solution and specialized technicians were needed to manage the tool.
- *No Cloud-Native*: first generations of SIEM could not communicate with clouds.

3.2.2 Artificial Intelligence and Machine Learning generations

Nowadays SIEMs are more flexible and have lower costs than older generations. In 2017 a multinational company which provides guidance and tools to help organizations in critical fields called Gartner introduced a SIEM tool able to analyze users' and entities' behaviour and automatically respond to anomalies.

The major contributions to these new generations of SIEM are **Artificial Intelligence (AI)** and **Machine Learning (ML)**. Thanks to these innovations, SIEMs can efficiently discover correlations between

data. For example, the tools use heuristic algorithms to execute pattern matching or aggregate logs for advanced research.

A deep insight into data allows SIEMs to choose which countermeasures to take and responds automatically to problems.

New generations are able to create **users' risk profiles**. This feature helps the security team to find the source of problems caused by users and prevent them.
Also, dashboards have been introduced. They allow to visualize and search into logs and data.

3.3 SIEM Structure

SIEMs are responsible for collecting logs and general events from applications and network systems to monitor the situation.

The functional principle of the SIEM is based on a set of rules, which are defined by vendors or security managers, and a software which searches correlations between collected data.
The tools implement different functions:

- ***Collect and analyze data***: are the main functions of the SIEM. The tool collects data in real-time and generates historical logs, which are useful to quickly identify system anomalies.
  Different data suppliers are used. Some examples are:
  - *Security Events*: intrusion detection systems, Antivirus, antimalware, VPN, web filters, firewalls.
  - *Network Logs*: routers, switches, DNS Servers, Wireless Access Points, WAN, data transfers.
  - *Applications and Devices*: databases, web applications, cloud-hosted servers, end-user laptops or desktops, mobile devices.
  - *IT Infrastructure*: configuration, owners, network maps, vulnerability reports, software inventory.

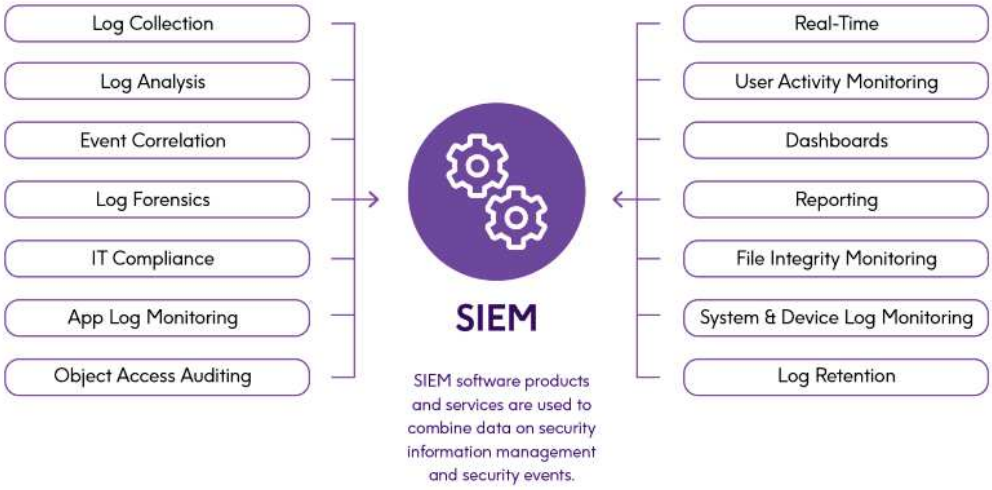
  SIEM monitors real-time data using specific metrics to detect system anomalies. It also allows setting alerts in case of problems detection.
  Analysing historical logs the software helps to understand what happened in the controlled system.
- ***Correlation***: it is based on the metrics used to get information. SIEM not only collects data but also correlates them to discover strange behaviours of the system.
  It correlates historical and real-time data. This feature allows to detect anomalies and set alerts using a threshold based on historical logs. For that reason, SIEM can become more precise.
- ***Alerting***: as previously introduced, an important function is to alert in case anomalies are detected, by specifying critical thresholds. SIEM can send SMS or e-mail to warn.
- ***User-friendly***: SIEM usually allows to use customized dashboards to visualize and consult data collected and correlated. Also, the ability to customize reports is important.

3.3.1 Functionalities

In order to monitor and control a system, a SIEM relies on the following capabilities: [43]

1. *Data aggregation*: using security systems and network devices, it collects and aggregates data.
2. *Threat intelligence feeds*: SIEM combines internal data about threats and vulnerabilities with third-party data.
3. *Correlation and Security Monitoring*: it links data and events with security incidents.
4. *Analytic*: it can identify correlations between data using statistical models and machine learning.
5. *Alerting*: SIEM analyzes log and data to identify issues and immediately alert security team.
6. *Dashboards*: it allows to visualize collected data and help the security team to identify patterns or anomalies.

7. *Compliance*: it collects log data and generates customized reports using them.
8. *Retention*: SIEM stores long-term data, which are useful for investigations.
9. *Forensic Analysis*: it enables analysis of logs and events data to discover security incident details.
10. *Threat Hunting*: SIEM gives the possibility to query logs and events data to uncover threats.
11. *Incident Response*: it allows security teams to quickly identify and solve security incidents.
12. *SOC Automation*: SIEM can take action automatically in order to respond to incidents by orchestrating security systems.



[https://www.precisely.com/app/uploads/2019/11/SIEM-Graphic_2020-800x432.png]

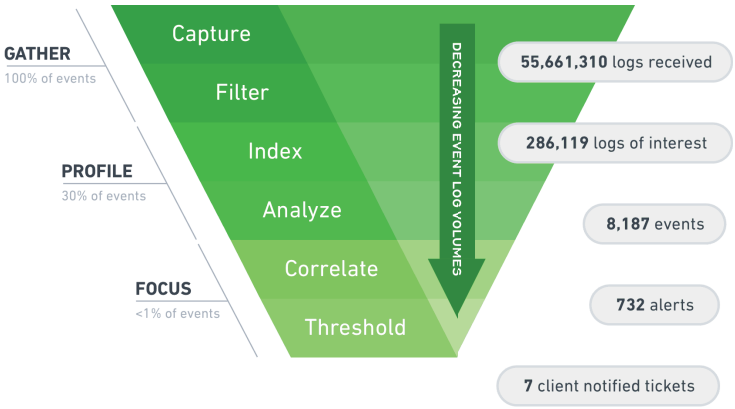
Figure 3.1: SIEM functionality

3.3.2 Architecture

The most important structure of a SIEM is *log management*, which involves collecting, analysing, and storing data.

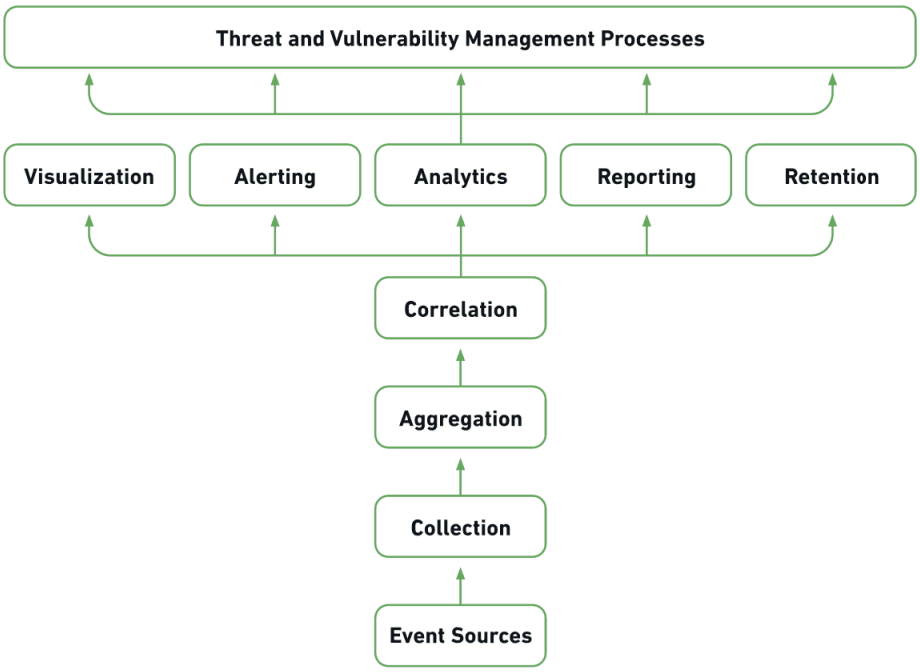
- ***Data Collection***: SIEMs gather logs and events from different data sources. Devices store logs in files or databases. Logs are collected from SIEM, which standardize and store them in a format which allows analysis of data. The SIEM can collect data in different ways:
  1. By using an **agent** which is installed on system devices.
  2. By directly accessing to log files.
  3. Via event streaming protocols.
  4. By directly connecting to the device using a network protocol or API call.
- ***Data Management***: a large amount of data are stored by SIEMs. The stored data needs to be:
  - Stored in local or in cloud or both.
  - Efficiently stored and indexed to make faster analysis.
  - In *high-performance* storage should be stored *important data*, which have to be often investigated for real-time monitoring, and other data in other locations.

- ***Log Retention***: as mentioned in the previous point, a large amount of data are analyzed and stored by SIEMs and a very high volume of logs are stored over time. For this reason, SIEMs use strategies to reduce log volumes:
  - They use a specific format called *Syslog*, which allows to *compress logs* and retain a lot of historical data.
  - SIEMs automatically *delete* logs which are *not useful* for future analysis.
  - *Log filtering*: logs are *filtered* using defined rules by the security team.
  - *Summarization*: data are summarized storing *only important information*.



[https://www.exabeam.com/wp-content/uploads/2018/08/log-flow-diagram@3x.png]

Figure 3.2: SIEM Log Filtering



[https://www.exabeam.com/wp-content/uploads/2018/08/security-events@3x.png]

Figure 3.3: SIEM Architecture

3.3.3 Metrics

In order to study the security of a system, it is important to know how incidents affect an environment. *Metrics* play an important role and must have the following features:

- *Simple*: easy to measure, so it is also easy to explain.
- *Measurable*: must be measurable in a consistent and well-defined way.
- *Actionable*: useful for making a decision.
- *Relevant*: valuable for the decision to make.
- *Time-based*: useful for showing changes over time.

In the case of SIEMs, metrics can be used to monitor:

- Statistics on handled alerts, monitoring for example the number of alerts per day depending on the category.
- Response time to a security incident with a given alert.
- Same incidents that are detected after being notified to security teams using an alert.
- Information on the actors who are most frequently involved in security incidents.

4 Elastic SIEM

4.1 SIEM tools

In the IT market, a large amount of SIEM tools have been developed. Softwares can be open-source or paid, each one with its features.

In order to choose the most suitable SIEM tool, the end-users need to evaluate tasks the software need to perform, such as protection against attacks with notification to users.

To assess the market, researchers of Gartner company evaluate SIEMs implementation in the market, generating a quadrant called *"Magic Quadrant for Security Information and Event Management"* (Figure 4.1), which represents a qualitative evaluation of the tools.

Source: Gartner (June 2021)

[https://images.contentstack.io/v3/assets/bltefdd0b53724fa2ce/bltfaf8b8077c0f9a52/60e47a86979c171ed133e547/gartner-magic-quadrant-security-information-and-event-management-april-2021.png]

Figure 4.1: Gartner magic quadrant for SIEM

The quadrant shows the leaders, which are Exabeam Fusion, IBM-Qradar, Securonix Next-Gen SIEM, Splunk Enterprise, InsightIDR (Rapid7), LogRhythm NextGen SIEM Platform.
In the following paragraphs, some SIEMs are discussed and analysed.

4.1.1 Exabeam Fusion by Exabeam

Exabeam Fusion SIEM is a cloud SIEM solution. The tool can detect, investigate, and respond to threats and allows to gather data from anywhere. It can also detect threats missed by other solutions.

Pros	Cons
Intuitive and easy to use	Default rules can lead to false positives
Customizable dashboard	Could be more flexible

Table 4.1: Pros and cons of Exabeam Fusion [72]

4.1.2 IBM QRadar by IBM

IBM QRadar is one of the most used SIEM on Windows environments. Developed by IBM, it represents a log management, offers analytical functions, and allows to detect intrusions on systems. QRadar can be considered as a near-complete solution and is also available as a cloud platform.

Pros	Cons
Allows to evaluate risks using artificial intelligence	Doesn't have good integration with other SIEM tools
Can simulate attacks to evaluate effects on a network	Analysis tools should be improved
Easy and user-friendly interface	

Table 4.2: Pros and cons of IBM QRadar [58]

4.1.3 Splunk Enterprise by Splunk

Splunk Enterprise is one of the most important SIEM tools. This tool monitors real-time data of network and machine and can identify potential vulnerabilities or wrong behaviours, at the same time. Another important feature of Splunk is the *Asset Investigator* which identifies malicious actions. The user interface is simple.

Pros	Cons
Detects threats analysing behaviours	A quote from the vendor is required to know the price
Simple and customizable user interface	More convenient for large enterprises
Events prioritization	Not efficient processing language which slows learning curve
Available for Linux and Windows	

Table 4.3: Pros and cons of Splunk Enterprise [58]

4.1.4 LogRhythm NextGen SIEM Platform by LogRhythm

LogRhythm NextGen SIEM is considered one of the pioneers of SIEM solutions. It is a complete solution: the tool allows to execute behavioural analysis, identify correlations between logs and use artificial intelligence for machine learning. LogRhythm NextGen SIEM can be considered a good solution for medium-sized organizations thanks to its price.

Pros	Cons
Easy wizards to set up the tool	No free trial option
Customizable interface	No cross-platform support
Uses artificial intelligence to analyse the behaviour	

Table 4.4: Pros and cons of LogRhythm NextGen SIEM [58]

4.1.5 Microsoft Sentinel by Microsoft

Microsoft Sentinel, previously known as Azure Sentinel, is a cloud solution. Using analytics, the tool detects threats and reduces false positives, and using artificial intelligence investigates to find suspicious activities. It can respond automatically to problems.

Pros	Cons
Self-explanatory	Reports are not easy to read
Analyses any threat, including those not yet discovered	Depends on Microsoft Azure software
Reduces false positives	
Automatically responds to problems	

Table 4.5: Pros and cons of Microsoft Sentinel [71]

4.1.6 Elastic SIEM by Elastic

Elastic SIEM is an open-source tool. It has a modular architecture, allows to easily gather data, and generates shareable analytics. The user can interact with the data using a SIEM dashboard. Elastic SIEM evolves with threats.

Pros	Cons
Free to get started	Scaling challenges
Distributed in different programming languages	Stability problem when data volume increases, due to indices definition
Easy to administrate	
Data can be visualized and analysed in real-time	

Table 4.6: Pros and cons of Elastic SIEM [49] [71]

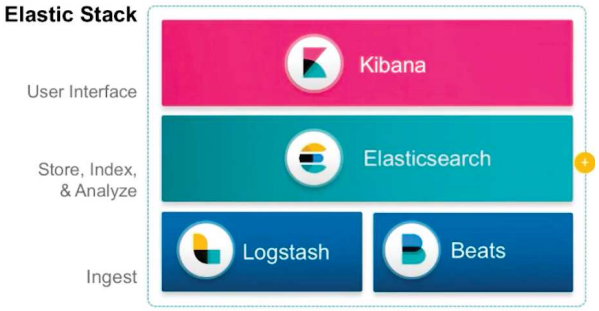
In the next paragraph, Elastic SIEM is detailed because is the tool which is used in the applied section. The SIEM was chosen because it is open-source, which is a good requirement for education purposes, and only essential parts can be installed. It has few paid-for functionalities.

4.2 Elastic Stack

The company *Elastic* has developed three open-source products:

1. *Elasticsearch*: is an open-source, search and analysis engine;
2. *Logstash*: transforms and parses data collected from different sources;
3. *Kibana*: allows users to visualize and analyze data.

These software modules compose the acronym, **ELK Stack**. This project is used to monitor, troubleshoot and secure IT environments. It allows to collect event logs or metrics and analyse information to solve challenges for complex environments.
In 2015, a fourth module called **Beats** has been added to the ELK stack project. This program is a lightweight data-shippers, which works as an *agent* installed on devices to collect information.
Taking everything to account, Beats and Logstash gather and process data, Elasticsearch optimizes and stores logs, and Kibana allows to visualize data.
This new project structure has been renamed **Elastic Stack**.



[https://www.itzgeek.com/wp-content/uploads/2020/06/Log-Monitoring-With-ELK-Stack.jpg]

Figure 4.2: Elastic Stack (1)

[https://collabnix.com/]

Figure 4.3: Elastic Stack (2)

The *Elastic Stack* is popular because provides a robust platform which allows to elaborate data from different sources, store them in centralized storage, and can scale.
The tool is *open-source* and allows to understand why the stack is so popular. For this reason, organizations can keep away from "vendor lock-in".
In addition, ELK is a cheaper solution than others vendors.

4.2.1 Elasticsearch

Elasticsearch is considered as *"The heart of the free and open Elastic Stack"* [19]. It is a distributed system which stores data and allows fast search and analytics, using a NoSQL database.

Figure 4.4: Elasticsearch logo

Elasticsearch lets perform a variety of searches, exploring trends and patterns in data. The search is really fast and rapidly gives results, thanks to implemented inverted indices and BKD trees.
It also ranks search results to improve the way results are shown to users and tries to correct human errors such as typos.

The heart of the Elastic Stack can run on a cluster or a single node independently. It can identify failures on cluster devices and maintain the operational structure. Elasticsearch can also decide where store data: locally for fast queries or remotely for historical data.

Elasticsearch plays such an important role that its name is used as a **synonym** for *ELK stack*.

4.2.2 Logstash

Logstash performs data processing. It collects data from different sources, like logs, web applications, metrics, and then parses and transforms the data, building structures which can be easily analysed.

Figure 4.5: Logstash logo

Logstash elaborates and converts any type of data:

- identifies structure from unstructured data;
- finds GEO coordinates using IP addresses;
- hides sensitive information;
- independent simply process.

Once the data have been processed, they are sent to different supported output destinations.

4.2.3 Kibana

Kibana is a free user interface, which allows to visualize and analyse Elasticsearch data. It allows to customize the data display creating your screen to represent thousands of data. Kibana gives the possibility to create alerts, such as sending emails, by checking the value of indices against thresholds.

Figure 4.6: Kibana logo

There are different categories of customization:

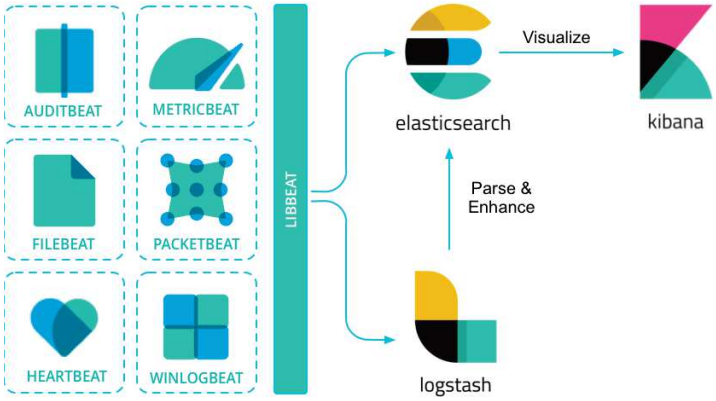
- *Basics*: classic shapes of data such as histograms, line graphs, pie charts, and more.
- *Location analysis*: data are represented on a map to explore location data.
- *Time-series*: graphics which uses time series to visualize data.
- *Machine learning*: find anomalies in data using unsupervised machine learning features.
- *Graphs and networks*: data are represented using graphs to discover relationships on data.

4.2.4 Beats

Beats is a free software which gathers data from devices. It is a lightweight shipper of data, which works as an ***agent*** installed on devices: it collects logs or metrics from a device and sends them to Logstash, which transforms and parses data, or Elasticsearch, which centralizes the data.
There are different types of Beats which allow to collect various data:

- *Filebeat*: allows to collect log files and forward them to Elasticsearch or Logstash for indexing.
- *Metricbeat*: gathers operating system metrics and information from services, such as Apache, MongoDB, MySQL, PostgreSQL.
- *Packetbeat*: sniffs network traffic and allows to monitor applications and performance.
- *Winlogbeat*: is a shipper of Windows events log, such as application, hardware, security, system events.
- *Auditbeat*: gathers data from users' activities and systems' processes, and allows to identify security branches.
- *Heartbeat*: is installed on a remote server and is used to check the status of services on another server, for example, if the latter is reachable.

- *Functionbeat*: allows to collect data from a serverless environment and ship them to ELK stacks.



[https://www.elastic.co/guide/en/beats/libbeat/current/images/beats-platform.png]

Figure 4.7: Beats Diagram

4.3 Elastic SIEM

Elastic Stack is an efficient log management and analysis, which can collect information from a large number of sources. These features are important to implement a SIEM which is why Elastic has presented a *Security Information & Event Management with the Elastic Stack*.
Elastic SIEM allows to analyze environment behaviours, detect and solve threats, using real-time events information.
Thanks to Kibana's ability to customize visualization, users can monitor data and analyze information using dashboards. ML jobs are implemented to control potential problems which are or are not expected. Due to intelligence, alerts can notify possible threats to the security team.

5 Internet of Things

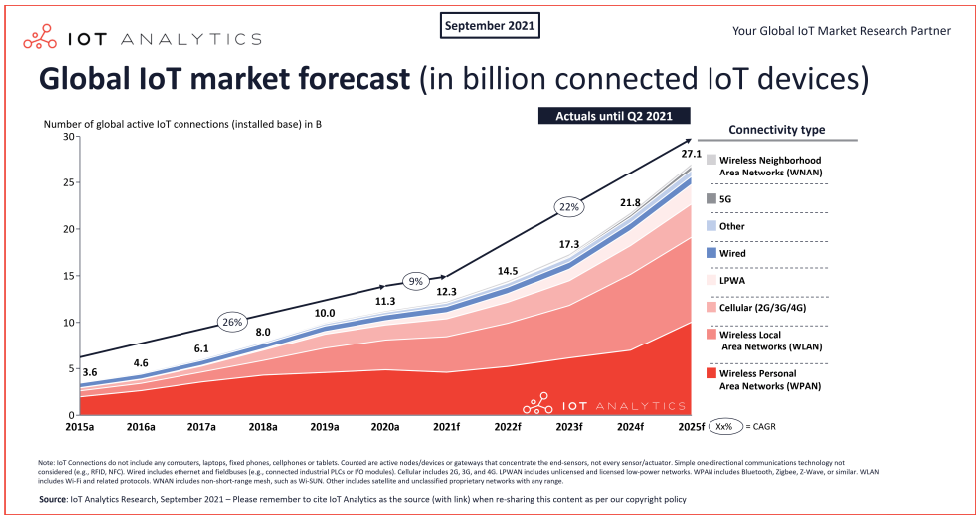
The terms **Internet of Things (IoT)** is used to describe a network of physical devices which are connected to others to share data over the Internet.

In the 1980s and 1990s began debating about the idea of adding sensors and intelligence to objects, but the idea was impracticable due to technology limitations: chips were too big and expensive for objects. In 1999 Kevin Ashton coined the phrase "*Internet of Things*" to describe this idea.
In the beginning, IoT was related to business and manufacturing, where its application was known as machine-to-machine (M2M), but today the Internet of Things is also widely used in our homes and offices thanks to smart devices that have become more and more important in our lives.

Thanks to the lower cost of chips and the increasing number of wireless networks, it is now possible to connect a large number of devices, from the smallest to the biggest device. Some examples of objects that usually people connect every day are sensors, smart-TVs, smartphones, smart speakers, appliances, thermostats, home security systems, cameras, cars, also baby monitors.
Recent improvements that have enabled the implementation of the IoT idea are:

- *Affordable and reliable sensors* with low-cost and low-power consumption.
- *Connectivity*: the devices can be easily connected with other devices using the Internet with efficient data exchange.
- *Conversational artificial intelligence (AI)*: thanks to natural language processing (NLP) using neural networks and applying it to IoT devices (e.g. digital personal assistants Alexa, Cortana, and Siri), made it possible to use IoT devices in the home.

Nowadays there are more than 10 billion IoT devices and experts expect this number to grow to 22 billion by 2025 and 125 billion by 2030.



[https://iot-analytics.com/wp/wp-content/uploads/2021/09/Global-IoT-market-forecast-in-billion-connected-iot-devices-min.png]

Figure 5.1: Growth of IoT devices

The increase of IoT devices has led to the introduction of a new version of the Internet Protocol, the **IPv6**, which is the successor of *IPv4*.

Total Number of Addresses	
IPv4	**IPv6**
$2^{32} = 256^4 \approx 4.310^9$	$21^{28} = 16^{32} \approx 3.410^{38}$

Connecting devices allow real-time data communication with minimal human intervention. IoT allows automating systems, like factories, creating smarter and more responsive environments. Digital systems allow to monitor and record the sensor data. In this way, the physical and digital world meets and cooperate.

Not only advantages from the IoT introduction but also disadvantages. The greatest problems are due to the ***privacy and security of users' data***. Industry and governmental moves to develop standards and guidelines to avoid those issues.

5.1 IoT Data Protocols

IoT devices need protocols to transfer data and communicate. In the IoT world, the devices are connected through a *wired*, a *cellular*, or a *Wi-Fi network*.
The general goal of IoT data protocols is to manage large and often unreliable communication networks due to the increasing number of small, cheap, and lower-power IoT devices that appear in the networks.

Different data protocols have been developed to allow IoT devices to communicate with others.
These protocols can be divided into two main categories:

- *machine-to-machine (M2M)*: allows communication to devices with limited capacity in low-bandwidth networks. Examples are:
    1. **MQTT** - Message Queuing Telemetry Transport
    2. **CoAP** - Constrained Application Protocol
- *client-to-server*: manages data transmission from device to server. Examples are:
    1. **HTTP** - Hypertext Transfer Protocol
    2. **AMQP** - Advanced Message Queuing Protocol
    3. **Websocket**

Another protocol of interest is the **DDS** - *Data Distribution Service* protocol, which is considered the first open international IoT middleware standard.

In the following paragraphs, the protocols are discussed in more detail.

5.1.1 Machine to Machine Protocols

5.1.1.1 MQTT - Message Queuing Telemetry Transport

The **Message Queuing Telemetry Transport (MQTT)** protocol is a lightweight protocol and is based on the *publish/subscriber model*. This protocol allows low power consumption for devices and also works on TCP/IP protocol.

The MQTT protocol has been adopted in many fields. Despite this, the data doesn't have a specific representation and are platform or vendor-specific.

5.1.1.2 CoAP - Constrained Application Protocol

In order to communicate, the IoT devices can use the HTTP protocol which is used in the World Wide Web, but it's a heavy and power-consuming protocol.

The **Constrained Application Protocol (CoAP)** translates the HTTP protocol. The protocol reduces overall consumption making it useful for devices with limited battery and resources.

The protocol is generally used for IoT microcontrollers or wireless sensors network nodes.

5.1.2 Machine to Server Protocols

5.1.2.1 HTTP - Hypertext Transfer Protocol

Another possible approach is the **HyperText Transfer Protocol (HTTP)**, which is a *request/response protocol* and is born for web communication. The drawbacks of this protocol are the costs, the huge consumption of battery and it isn't a lightweight protocol.

The protocol is used in some industries like 3-D printing because a large amount of data must be published during communication.

5.1.2.2 AMQP - Advanced Message Queuing Protocol

The **Advanced Message Queuing Protocol (AMQP)** is used to manage the communication with servers and is a secure protocol. The protocol receives messages and places them in queues, stores messages, and creates a relationship between components.

This protocol is generally used in server-based analytical environments.

5.1.2.3 Websocket

WebSocket is born as part of HTML5. The protocol allows to send messages between the client and the server using a single TCP connection.

The protocol is used in IoT networks where data are continuously communicated between different devices.

5.1.3 International IoT middleware standard

5.1.3.1 DDS - Data Distribution Service

DDS (Data Distribution Service) is an IoT protocol which allows high-quality communication and works with the *publish/subscribe model*, like MQTT. The protocol is used in real-time and embedded systems.

The advantage over MQTT is that DDS allows to exchange data ***independently of hardware and software platforms***. This feature allows DDS to be considered as the *first open international middleware IoT standard*.

5.2 IoT and SIEM integration

Nowadays, IoT devices can be considered as the main vulnerability of any system, because of their low security, generally due to the few resources available in these devices that do not allow the execution of energy-expensive software or other security measures that can be used.

The increasing development of IoT devices and their introduction into complex systems in various sectors, including industry, has led to an increased focus on the security of these systems. This is done to increase the security of these systems by protecting sensitive information and data and reducing the risk of these systems being hacked, which can lead to serious damage to businesses such as business failure. The monitoring of data and device logs becomes important in order to analyse and supervise their behaviour.

For these reasons, networks with **IoT devices have been combined with SIEMs**. In this way, the network is continuously analysed and the devices are supervised in order to detect undesired behaviour, incorrect accesses, and more. Notification of any issue allows security teams to work on making the network more secure. The combination of SIEMs and IoT, therefore, allows security problems to be managed more efficiently. Using SIEMs as supervisors of the environment can help decreasing vulnerabilities improving the security of IoT networks.

Integrating the IoT with the SIEM can put the tool under pressure because the devices generate a large amount of data. In order to reduce the amount of information, Big Data repositories could be used, which collect only significant information from IoT devices. Then, SIEM works on the collected data and analyses only the significant IoT information.

In the application section of the paper, a network of IoT sensors that are able to detect information about the environment of a greenhouse is combined with a SIEM. This allows supervisors to simplify the management and supervise the operations inside the greenhouse.

6 MQTT - Message Queuing Telemetry Transport

As introduced in the previous paragraphs, MQTT is one of the most used protocols in the IoT environment. In this section, the protocol is discussed in-depth because it is used in the application part of the paper. This protocol is aimed at minimising the resource requirements of the devices guaranteeing reliability and message delivery.
These characteristics make the protocol suitable for environments where available resources and network bandwidth are limited, and where there are remote devices with small memory and low computing capacity. For those reasons, the MQTT is chosen in the applicative section between the protocols previously described. Having to simulate a network of IoT devices, which are generally devices with little memory, little battery power and low computing capacity, that have the task of collecting information about a greenhouse, explains the use of this protocol.

The **Message Queuing Telemetry Transport** was invented in 1999 to manage communication between machines efficiently, to minimise network traffic while requiring minimal energy consumption from the devices involved. MQTT is a *lightweight* and energy-efficient application layer protocol, is simple to implement, capable of efficiently distributing messages and allows to exchange information even when the networks are not stable where the devices are connected.

The MQTT protocol is a protocol based on the **publish/subscribe model** using **topics** to organise messages. It is based on *TCP/IP* using the following ports:

- **1883** for ***non-encrypted*** communication;
- **8883** for ***encrypted*** communication.

Two *main entities* are important for the MQTT protocol:

- **Client**: produces and receives data by publishing or subscribing to a *topic*. MQTT protocol can be considered as a bidirectional protocol because the devices can publish and also receive messages.
- **Broker**: also known as server, can be considered as a *post office*. It receives the published messages from the clients and sends them to others which subscribe to a topic.

In the previous lines, the MQTT protocol has been defined as a **publish/subscribe protocol**. Devices publish messages in one topic and receive messages from other devices only from the topics they are subscribers, all in an ***asynchronous*** way.

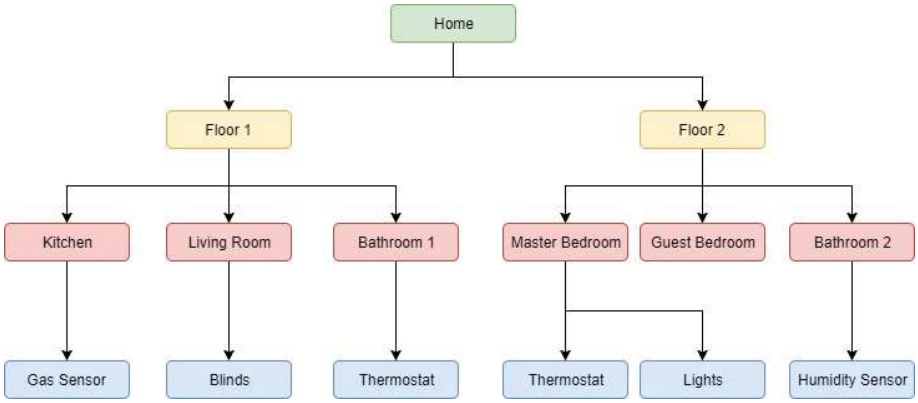
The following paragraph describes the structure of the *topics*.

6.1 Topics

The **topics** are UTF-8 strings which are used by the broker to filter messages. The topics may consist of several levels. The character which allows creating a new level is the forward-slash (`/`).
Some examples of topics are:

- `locker_room`
- `car/position`
- `home/livingroom/lights`

The topics must consist of ***at least one character*** and are ***case-sensitive***.
This structure of topics allows MQTT to be lightweight because doesn't need to use a message queue.
The following image represents an example of an *expansion tree* of some topics.



[https://smarthomeblog.net/wp-content/uploads/2018/01/MQTTTopics.jpg]

Figure 6.1: Example MQTT Topics

6.1.1 Wildcards

In order to subscribe to a topic, a client subscribes to the exact topic and publishes messages. The client can also use special characters called **wildcards**. These special characters allow to select different levels of topics, but can't be used to publish messages, only subscribe.
There are two different types of wildcards:

- **Single Level (+)**: this wildcard replaces one level of topic. Every topic containing an arbitrary string in +'s space is selected

`home/+/lights`	
Topics Selected	**Topics Not Selected**
`home/livingroom/lights`	`home/livingroom/temperature`
`home/kitchen/lights`	`garage/lights`

- **Multi-Level (#)**: this wildcard allows to select more than one topic level. The subscriber receives all the messages related to topics starting with the string before the wildcard character.

`home/livingroom/#`	
Topics Selected	**Topics Not Selected**
`home/livingroom/lights`	`home/bedroom/temperature`
`home/livingroom/temperature`	
`home/livingroom/lights/shelf`	

6.2 Packet Structure and Control Packet

According to the MQTT protocol specifications, the packet has a *fixed header* of *two bytes*. The header is followed by a *variable header* which depends on *topic name length*.

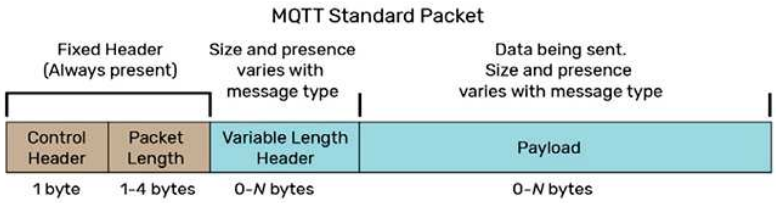


Figure 2. The size of an MQTT packet can vary from 256 MB to as little as 2 bytes.

[https://www.automation.com/getmedia/fcd1f361-9e71-4f70-ab60-03eb68e5e401/IIot-figure-2-March-10-2021-web.png]

Figure 6.2: MQTT Packet

The header is composed of five fields:

- ***Packet Type*** *(4 bit)*: it indicates the type of the packet, which represents the different behaviours of the protocol.

  1. *CONNECT*: when the client requests to connect to the server.
  2. *CONNACK*: the server sends an acknowledgement to the connection request.
  3. *PUBLISH*: to publish a message.
  4. *PUBACK*: acknowledgement to message publication.
  5. *PUBREC*: confirmation of receipt of the packet.
  6. *PUBREL*: when a message is released.
  7. *PUBCOMP*: publication of a message completed.
  8. *SUBSCRIBE*: client requests to the server to subscribe to a topic.
  9. *SUBACK*: the server sends an acknowledgement to subscribe request.
  10. *UNSUBSCRIBE*: client requests to the server to unsubscribe a topic.
  11. *UNSUBACK*: the server sends an acknowledgement to unsubscribe request.
  12. *PINGREQ*: generates a request PING.
  13. *PINGRESP*: it is used to PING response.
  14. *DISCONNECT*: client disconnects from the server.

  The numbers 0 and 15 are ***reserved*** values.

- ***DUP Flag*** *(1 bit)*: it represents a duplicate flag and is used to mark messages which are duplicated. This field is always 0 for all messages where the quality of service is 0 (described in the following paragraph).

- ***Quality of Service*** *(2 bit)*: it is used to choose QoS (described in the following paragraph). The allowed values are:

  00. QoS0
  01. QoS1
  10. QoS2
  11. Not allowed

- ***Retain*** *(1 bit)*: if this flag is set to 1, the server stores the message and the QoS to deliver it to future subscribers.

- ***Remaining Length*** *(8 bit)*: indicates the packet length

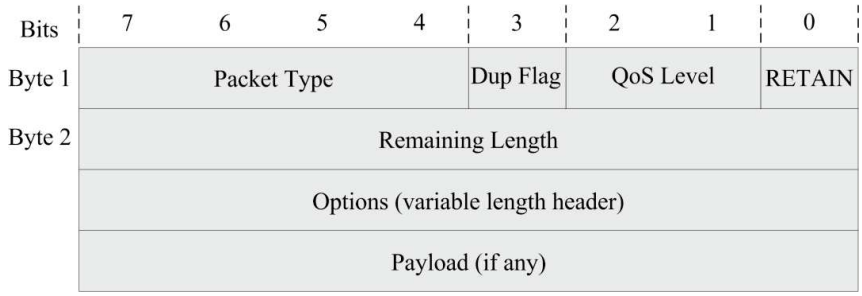


Figure 6.3: Fields of MQTT Packet

The **payload** field contains the message sent via the MQTT packet. It can contain every type of message with a maximum size of 256MB.

Using MQTT protocol, a device can leave a message if it suddenly crashes or can't connect to the server. It is called the **"Last Will and Testament"** message. This special message can be specified when the client sends CONNECT request.

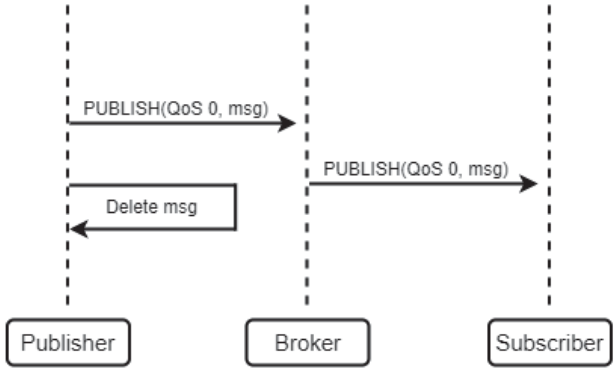
Packet Name	Packet Value	Flag Direction	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client → Server	Client request to connect to Server
CONNACK	2	Server → Client	Connect acknowledgment
PUBLISH	3	Client → Server Or Server → Client	Publish message
PUBACK	4	Client → Server Or Server → Client	Publish acknowledgment
PUBREC	5	Client → Server Or Server → Client	Publish received
PUBTREL	6	Client → Server Or Server → Client	Publish release
PUBCOMP	7	Client → Server Or Server → Client	Publish complete
SUBSCRIBE	8	Client → Server	Client subscribe request
SUBACK	9	Server → Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client → Server	Unsubscribe request
UNSUBACK	11	Server → Client	Unsubscribe acknowledgment
PINGREQ	12	Client → Server	PING request
PINRESP	13	Server → Client	PING response
DISCONNECT	14	Client → Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Figure 6.4: MQTT control packet

6.2.1 Quality of Service

The MQTT protocol reserves a field for setting the **Quality of Service** level. It allows to choose how to deliver a message to the broker by an agreement between client and server.
There are 3 levels of QoS:

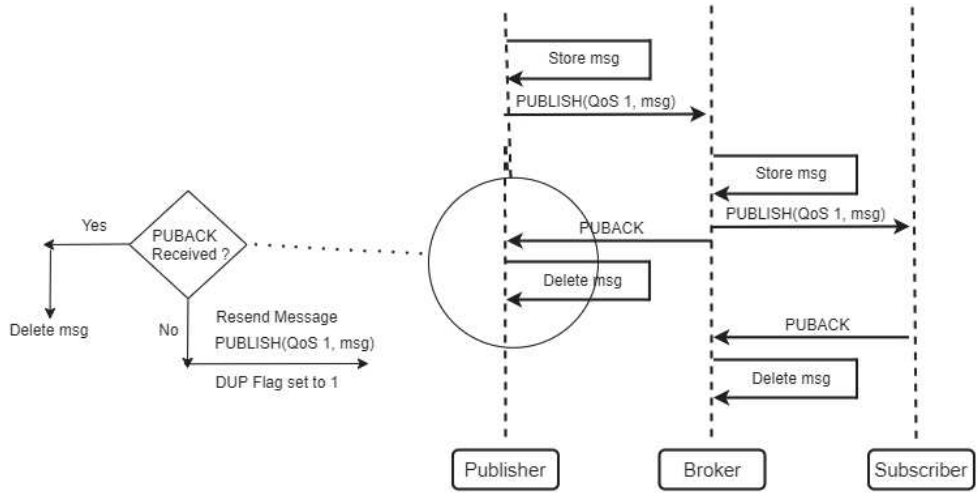
- **QoS 0 (at most once)**: it is fast and reliable and guarantees which message is delivered at least once. With this level of quality of service, the client doesn't receive any confirmation when the message is published (as shown in Figure 6.5).
  This level of QoS is generally used when data is not important and when devices have a stable connection, such as a wired connection.



[Article [7] page 5 - Stress Testing MQTT Brokers]

Figure 6.5: MQTT control packet sequence QoS 0

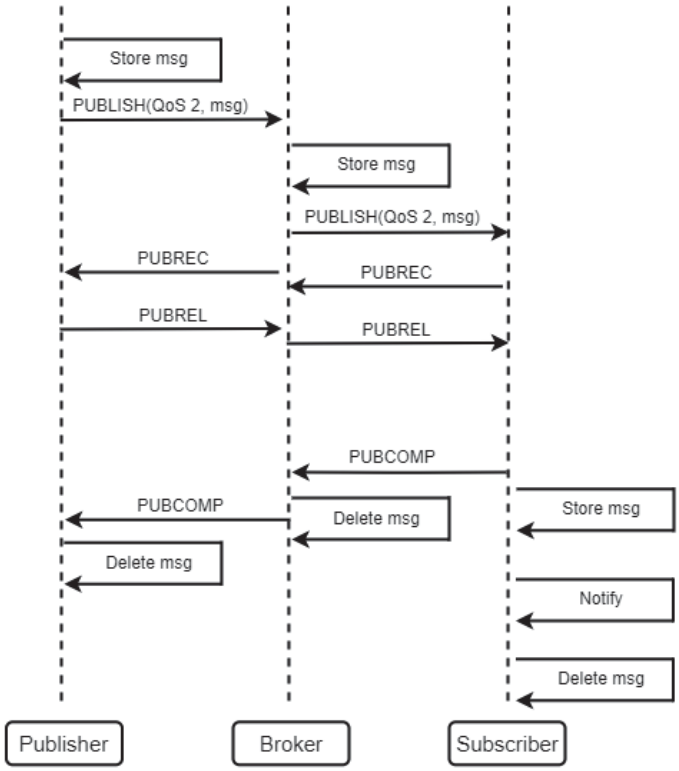
- **QoS 1 (at least once)**: this level guarantees that a message is sent to the receiver at least once, but duplicates may be received. If the sender doesn't receive a *PUBACK* message after publication in a certain time, the submitter repeats the message and sets *DUP* field to indicate that it is a duplicate (as shown in Figure 6.6).
  This level of QoS is used to guarantee that the message is received and quickly sent.



[Article [7] page 5 - Stress Testing MQTT Brokers]

Figure 6.6: MQTT control packet sequence QoS 1

- **QoS 2 (exactly once)**: it ensures that each message sent is received only once. It is the safest and most reliable level but it is complex and bandwidth-consuming. The message is published, then the receiver sends a *PUBREC* to confirm. The submitter sends *PUBREL* and the receiver replies with *PUBCOMP* (as shown in Figure 6.8).
  This level of QoS is used in applications where every message must be received and duplication must be avoided. QoS 3 generates a lot of network traffic.

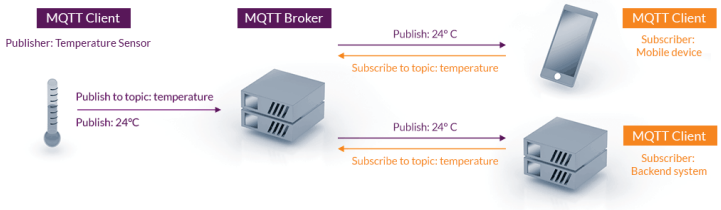


[Article [7] page 5 - Stress Testing MQTT Brokers]

Figure 6.7: MQTT control packet sequence QoS 2

6.3 MQTT Broker

The **MQTT Broker** (or server) is an important entity of the MQTT protocol. It is responsible for managing the messages it receives from clients and forward them to subscribers using topics as filters. It also has the task of authorizing and authenticating senders and subscribers.



[https://mqtt.org/assets/img/mqtt-publish-subscribe.png]

Figure 6.8: Example of communication between a temperature sensor, the MQTT Broker, and other output clients

Different software of MQTT brokers have been developed, some examples are:

- Mosquitto
- Bevywise MQTT Route
- ActiveMQ
- HiveMQ CE
- VerneMQ
- EMQ X

Scalability means the ability to expand when the environment changed in size or scale and is a very important factor to consider to build an MQTT Broker.
In order to manage scalability, MQTT brokers can be developed taking into account:

- *Single system*: improve Broker using an event-driven mechanism to manage TCP communications
- *Clustering*: the idea is to use a distributed system. Users observe that it behaves as a single broker, but in reality, there are multiple MQTT brokers which share the workload.

MQTT broker software can be separated into two families, depending on implementations.
There are two types of message broker implementations:

- Single or fixed number of threads ***non-scalable broker*** which cannot use all system resources
- Multi-process ***scalable broker*** which uses all the available system resources

Using this logic MQTT Brokers can be divided into:

Non-scalable broker	**Scalable broker**
Mosquitto and Bevywise	ActiveMQ, HiveMQ, VerneMQ, and EMQ X

Mosquitto has a "bridge mode" which allows to create a cluster of message brokers with the drawback of communication overhead inside the cluster.

In the article "*Stress-Testing MQTT brokers: A Comparative Analysis of Performance Measurements*" [7], authors compared scalable and non-scalable brokers implementation in single-core and multi-core CPU in stress conditions. They evaluated the latency of the systems, scalability, increasing the network traffic, and availability, to control systems failure.

Results showed that *Mosquitto* and *Bevywise*, which don't scale automatically to use available resources, performed better in resource-constraint environments. However, in a distributed/multi-core environment, *ActiveMQ* performed the best. It scaled well and showed better results than all other scalable brokers.

In general, to choose which broker to install must be considered environmental conditions.

6.4 MQTT Gateway

In order to improve the performance and efficiency of MQTT brokers, can be used devices which behaves as *MQTT Gateway*. The purpose of these entities is to reduce the data which are sent to a broker. For example, a temperature sensor is able to send hundreds of readings in a short time, and not always are useful to humans. The goal of the MQTT Gateway is to reduce the network traffic, resample the sensor data and send them to the Broker.

The **general chain between MQTT entities** can be represented in the following way:

1. Sensors detect information from the environment and send them to the MQTT Gateway
2. MQTT Gateway filters and reduces data volume
3. Reduced network traffic is sent to the MQTT Broker

6.5 Pros and Cons

The greatest advantage of the MQTT protocol is the ***simplicity of implementation***.

Small and medium-sized enterprises are approaching the IoT world to take advantage of its great potential. Softwares used for industrial IoT are usually expensive and protocols used are sophisticated. For resources, costs reasons and thanks to its flexibility, small and medium-sized enterprises mostly use the MQTT protocol.

Other advantages are provided by the publish/subscribe architecture because it allows to manage and process data in real-time.

The MQTT protocol was invented in the '90s and its goal was to manage the machine to machine communication. Nowadays not only M2M communication is required, but also machine to human. Some examples are the voice assistants like Amazon Echo or Google Home, which are more present in our homes.

Another problem is related to ***security***. When the protocol was invented, cybercrime was a small phenomenon, and cryptography and authentication protocols weren't enough developed. Some researches have shown that lots of MQTT messages are usually stolen from MQTT brokers which are exposed to attacks. Hackers may be able to control IoT endpoints or execute DDOS attacks.

7 IoT applications using MQTT protocol

A large number of IoT devices are available today. As discussed in the previous chapters, IoT devices generally use the MQTT protocol to transmit and collect data computed or measured by the devices.
These devices, such as sensors, electric sockets, and more, can simplify everyday life, and they can also be applied in any field by helping and facilitating processes.
IoT devices can be used in sectors such as:

- *Healthcare*
- *Agriculture*
- *Smart home*
- *Smart city*
- *Smart metering*
- *Industry*
- *Robotics*
- *Biomedical engineering*
- *Video surveillance*

In the following paragraphs, some IoT applications in different fields and their contributions are analyzed.

7.1 Smart Home

The most common application of IoT devices concerns smart homes. Electrical sockets, light bulbs, thermostats, smart boxes, and more are devices connected to others to control and manage your home. The introduction of voice assistants has accelerated the use of IoT devices in homes.

Usually, devices communicate using the MQTT protocol. Smart home devices data are collected to an MQTT server and the user, using mobile applications, can control the information of the house.

For example, it is possible to gather the states of the lights (ON - OFF) in the house, collecting them in specific topics with the MQTT protocol, and the user can check if he has turned off the lights when he already left the house. Moreover, using the voice assistants, the user can turn on the lights with a voice command.

Another application can be a soil sensor which activates irrigation as soon as the soil is not sufficiently wet.

The main problem is related to security. According to a study by Avast researchers, around 49,000 MQTT servers are public due to incorrect configurations, including around 32,000 without any password [53]. These MQTT servers can be hacked using requests which contain wildcards to subscribe to any topic, or an attacker can visualize the server dashboard without any impediment.
MQTT Server if incorrectly configured could expose personal information about a person's home.

For this reason, it is important to protect the data in Smart Home, as in any other application in the IoT world.

7.2 Smart City

Monitoring the city environment through sensors has become of interest to improve the level of well-being in the city. IoT devices are geographically distributed around the city to collect environmental data such as traffic in the streets, parking, traffic lights, air pollution levels (NO_2, SO_2), weather, and more. Once the data are collected, they are processed and managed to take action to manage the city and citizens.

The MQTT protocol plays an important role, because it has good performance even in conditions of low connectivity, and allows the transmission of data even with bad connectivity.

For example, using a sensor which measures the air quality, the information can be used to decide where to direct traffic to reduce pollution levels in order to have an acceptable air quality.
Another application concerns the monitoring of traffic lights and traffic: using this information the green light duration of traffic lights can be changed, or cars can be redirected within the city by reducing city traffic.

In conclusion, the use of devices which are able to gather data from the environment and monitoring them allows to achieve more secure and safer cities for citizens.

7.3 Industry

The industry sector is a competitive world. The introduction of *Industry 4.0* aims to optimize production and save time by collecting factory data. Real-time and dynamic data analysis can help to solve problems efficiently, prevent system maintenance, as well as create precise statistics on product quality.

IoT devices can be used to monitor the environment by reporting and collecting diagnostic information.

The MQTT protocol helps medium and small enterprises which can't afford to buy expensive programs, because it is an excellent compromise, in order to improve their business operations.

In the oil and petroleum industry SCADA systems, which are distributed systems that use PLCs to monitor and supervise physical systems, are used. IoT devices can replace these expensive systems, reducing hardware and software costs, even by 50%, and operations and maintenance personnel by 70%.

IoT devices can be applied in logistics by giving more services to customers.

Thanks to its easy implementation, the MQTT protocol allows to quickly and effectively update facilities, helping companies to reduce costs and to increase production.

7.4 Healthcare

IoT devices are also used in the health field. They allow to collect patient information, such as temperature, oxygenation, blood pressure, sugar levels, and more. By gathering this data in a non-invasive way and without visiting a user, a doctor can remotely monitor a patient and make a decision for his health, or can collect information which will be used in the hospital.
An example of a device which collects information is a smartwatch.

A sample application was developed by IBM. In this project, an implanted defibrillator communicates with a hospital, providing patient and device data. Collected data are securely transferred to the hospital using the MQTT protocol. The system reduces patient visits and can alert the hospital in case the device or patient needs attention.

IoT devices allow to improve the quality of care, especially in the management of chronic conditions, by reducing emergencies, and improving patient outcomes.

7.5 Agriculture

In the agriculture field, IoT devices help in precision agriculture. An example of the application of the MQTT protocol is hydroponic greenhouses.
Greenhouses help farmers to cultivate plants by protecting them from extreme weather events and insects. The hydroponic technique consists in cultivating plants without soil, feeding them through aqueous solutions. In hydroponic greenhouses, is important to monitor environmental factors, such as oxygenation, light intensity, air humidity, pH, temperatures, and more. These values allow to evaluate how to feed plants also based on further information, such as the age of the plant. The data are collected and visualized by experts to manage the greenhouse.

The use of IoT devices allows to be more efficient than a human, that executes repetitive work, optimize plant nutrition, and reduce any waste.

8 SIEM supervises a hydroponic greenhouse

8.1 Introduction

Nowadays, IoT devices are applied in many fields including agriculture. New tools are created all the time, and they are capable of taking increasingly precise measurements of the surrounding environment, allowing a user to monitor it precisely. These instruments are also often able to make decisions on their own.

An interesting example is **greenhouses**. Greenhouses are environments in which plants can grow and protect from adverse weather conditions and attacks by unwanted insects. Controlling the environment inside greenhouses gives the possibility to optimize plant growth: monitoring the values of the soil or other factors can improve the actions to be taken to obtain the best results from cultivation. Sensors allow to collect data from the environment and make them available to supervisors who can understand them and, if necessary, make decisions.

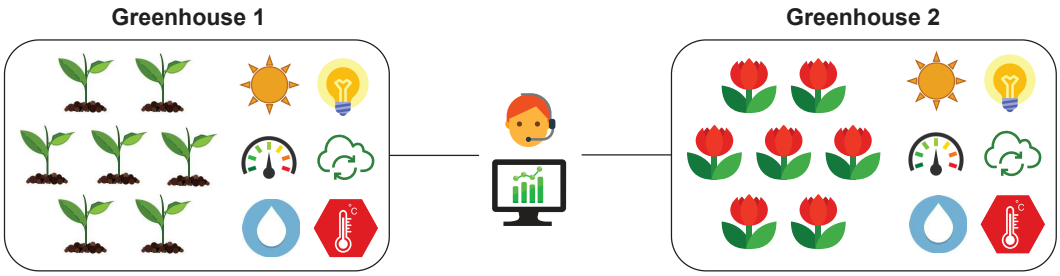


Figure 8.1: Supervised hydroponic greenhouse

In the project that has been developed, a hydroponic greenhouse is monitored, which is a greenhouse in which plants are fed with nutrient-rich water solutions. Sensors monitor the environment by taking measurements and can take actions autonomously in certain circumstances. The data collected by the sensors are communicated to a central server using the **MQTT protocol**, which is easy to implement and is suitable for any farm. In the project, the measurements are not real but are simulated through a **Python script** which is described in the following paragraphs. Furthermore, the hydroponic greenhouse that is analyzed is simplified, using a reduced number of sensors. In the following paragraphs the MQTT Broker that is used, **Mosquitto**, is also described.

A large amount of data are collected via sensors. Analyzing those data and monitoring them all together can be difficult for a human. To monitor and control them efficiently, a SIEM is used. It allows to organize, index, visualize, control through the use of alerts the collected information. In the project, **Elastic SIEM** is used as a tool to analyze the data. Some graphics are constructed to allow a supervisor to monitor the environment continuously. Colors are used to identify values that may be problematic or irregular. In this way, the supervisor can immediately understand what the problems are and try to solve them. The sending of e-mails is not managed because this is a function that is activated with a paid version of the tool.

To simulate the described environment, virtual machines are used to reproduce the behaviour of the greenhouse's sensors and supervise it. In the following sections, their configurations are described.

In addition to managing the **safety** of the greenhouse, it is also very important to handle the **security** of the environment.
The network structure can also be subject to **external or internal attacks**. The goal of an attacker may be to damage the network or environment devices, create slowdowns and more. In this way, for example, the attacker can sabotage production, creating damage to company productivity and economy.

In the following paragraphs, the simulation that is performed also contains the execution of some attacks. An attacking host tries to violate network security by damaging the main server where the MQTT broker is installed and interrupting the functioning of data collection by blocking the port listening to MQTT messages.

In this way, the SIEM behavior can be evaluated, and consequently the security of the network can be improved.

8.1.1 Eclipse Mosquitto

In the project concerning the hydroponic greenhouse, sensors communicate their measurements through the MQTT protocol. A fundamental entity which allows to collect and analyse collected data is the *MQTT Broker*. Its role is to gather messages sent by clients and forward them to subscribers, using topics as filters. In the previous chapters, some MQTT Brokers have been compared.

A message broker from the Eclipse Foundation is used for the project: **Mosquitto**. It is an open-source and lightweight tool which can be used in any device, thanks to its efficient implementation which allows low energy consumption. Moreover, Mosquitto is a portable software available on different platforms.

[https://it-obey.com/wp-content/uploads/2021/02/mosquitto-logo.png]

Figure 8.2: Mosquitto logo

It also provides two operations via command line:

1. *mosquitto_pub*: to post messages in a topic;

```
mosquitto_pub -h host -t topic -m "I'm a message"
```

2. *mosquitto_sub*: to subscribe to a topic and receive messages.

```
mosquitto_sub -h host -t topic -d
```

In the project, Mosquitto is installed on a host using Ubuntu Server as its operating system.

In the hydroponic greenhouse project, the MQTT Broker has the task of collecting the information from the sensors and making it available to SIEM, which allows a supervisor to analyze it.

8.1.2 Paho-MQTT

Paho is an Eclipse Foundation project. **Paho-MQTT** represents a library which implements the MQTT protocol. It is implemented in different programming languages, such as Java, Python, C, C#, JavaScript, and others.

Figure 8.3: Paho-MQTT Eclipse logo

The project manages machine-to-machine (M2M) communication by implementing the functions which characterize the MQTT protocol. The project provides a client class and functions which allow to *publish one-time messages* to an MQTT Broker and *subscribe topics*.

In the hydroponic greenhouse project, the Paho-MQTT project is used to create a sensor simulator in Python, which sends sensors data to MQTT Broker. In order to install Paho-MQTT can be used the *pip* tool:

```
pip install paho-mqtt
pip3 install paho-mqtt
```

8.2 Simulation analysis

8.2.1 Intent

In the application section, a SIEM is combined with a network of IoT sensors that are used to collect information in a greenhouse. The goal is to evaluate the behaviour of the tool in an IoT environment with devices composed of elementary hardware, and whether it is an effective solution to solve vulnerabilities in a system making it more secure.

The **attack surfaces** of the IoT, which are the set of real and potential vulnerabilities of a system, can be: [8]

1. **Devices**: attackers can exploit vulnerabilities in devices to perform an attack. They can use outdated components, insecure settings or updates, exploit available interfaces, network services and more.
2. **Communication channels**: IoT systems can be attacked by exploiting system composition and protocols, for example by creating a denial of service (DoS) and spoofing attacks.
3. **Applications and software**: attackers can use software flaws, for example, to steal credentials or make malicious firmware updates.

Attackers can therefore exploit flaws within the system to perform attacks. Examples are:

- **Denial-of-Service (DoS)**: attacks whereby can be interrupted services.
- **Spoofing**: attackers collect information passing through the network in order to exploit it.
- In the case of the MQTT protocol, an attacker can hack into the system and subscribe to a topic to collect data or publish malicious data to create malfunctions.
- Exploit application vulnerabilities in order to obtain information that should only be accessible to authorised persons.
- **Ransomware** with which important system data are scanned and encrypted.

The purpose of the attacks is to slow down or block the system operation, and, in the case of the greenhouse, can lead to a decrease in production creating also economic problems.

Three types of attack are performed in the application section:

- **ICMP Flood**: this is an attack in which an attacker loads a device with requests using the ICMP protocol in order to carry out a denial of service attack. The aim is to interrupt the function of the server.
- **DoS MQTT port attack**: another DoS attack is also carried out on the server. In this attack, innumerable requests are opened on port *1883* of the MQTT broker, causing the block of the service and interrupting the function of the entire system.
- **DoS attack on Kibana**: the attacker's goal is to interrupt the smooth operation of the *Kibana* service by blocking the display of data to the supervisor.

The SIEM, therefore, has the task of supervising the system and its behaviour is analysed.

In addition, a **simulator** is created to simulate IoT devices using a host on which a Python script is run. The simulation creates measurements of the devices in the greenhouse allowing them to be monitored.

Data are communicated using the **MQTT protocol** which allows sharing information in a lightweight way. A *QoS level 1* is used because the data must be guaranteed to be sent quickly without saturating the network traffic.
The information collected is structured using a *JSON object*. The MQTT messages are then sent to a host that works as a broker, where *Mosquitto* is installed and run on port *1883*.

The SIEM that is used is Elastic SIEM. All 4 modules of Elastic are used in the simulation.

- **Beats**: two modules are installed: **Packetbeat** and **Filebeat**.

  1. *Packetbeat*: is used to *sniff network traffic* about the data visualisation host and the MQTT broker, allowing applications and performance to be monitored. The data collected are then directly sent to the Elasticsearch engine, because the data do not need parsing.
  2. *Filebeat*: using this module to *collect logs*, MQTT messages are collected and managed by the Broker by filtering them using a topic. The information is then forwarded to Logstash.

  These modules are installed on the hosts where the broker and Elasticsearch run.

- **Logstash**: is responsible for managing the payload of MQTT messages. Logstash receives the content of the MQTT messages and extracts the JSON object. Then the object is forwarded to the Elasticsearch engine.

- **Elasticsearch**: receives the sniffed network packets and the JSON objects representing the state of the greenhouse and indexes them to be efficiently analysed and stored.

- **Kibana**: allows the supervisor of viewing historical and real-time data, and also of setting alerts.

Dashboards are created to keep track of packets traversing the network. Different types of graphs are shown, allowing to check the flow of network packets. The bitstreams occurring in the network server can be monitored by filtering information through hosts and IPs.
The graphs used are histograms and pie charts. Each one is used to represent a specific *type of network packet* and a *specific host*.
The information is automatically updated every few seconds.

Two more dashboards are created for a supervisor representing the information collected by the sensors in the greenhouse. The displays update automatically according to the data received.
In the two dashboards, the data are represented with the same types of graphs, but the information is different depending on the plantation.
The created dashboards allow highlighting the data using the following structure:

1. *Table* containing all measured data sorted by the hour and day. The values are colored to allow the supervisor a clearer view, especially if the values are outside the correct ranges.

2. *Line graphs* showing the sensors' value.

3. *Display* of current measured values.

This system makes it possible to create a network of IoT devices supervised by an Elastic SIEM. The structure allows to evaluate the ***effectiveness of a SIEM in a complex system with IoT devices*** in which ***safety*** and ***security*** are to be monitored.

The appendix to paragraph A.2 explains in a more detailed way how the whole system is configured.

8.2.2 Limitations

The system consisting of several IoT devices supervised by a SIEM which is simulated has some *limitations* compared to a real environment.

First of all, the Python script has the task to simulate only the values collected by the sensors from the environment. In this way, the system is not affected by **sensor failures** or **incorrect readings**.
Other network problems that may arise during data communication are not considered in the simulation, e.g. **malfunctioning network devices** such as a switch, or **network congestion** due to high traffic generated by IoT devices.

Due to this shortcoming, using the SIEM tool to monitor and detect information in case problems arise to the sensors, intrusions or attacks to the devices, or problems with some component of the system is not possible. Thus, the functionality of the tool is ***not fully exploited***.

Another limitation is due to some **paid features** of the Elastic SIEM. In fact, for example, the sending of emails is not implemented because it is a paid feature. This function would be useful when the values detected by the sensors exceed a certain threshold, and the supervisor is automatically notified by email, allowing him to take immediate action.

Other simulations can be developed to simulate possible problems with the sensors and devices in the network.

8.3 Sensors

The peculiarity of ***hydroponic greenhouses*** compared to classic structures is that the plants grow without using the soil: aqueous solutions are used to feed plants.
Environmental factors, such as *oxygenation*, *light intensity*, *air humidity*, *pH*, *temperatures*, and more, must be monitored to minimize waste and optimize plant development by creating the best environment.
These parameters can be monitored using sensors which may be able to perform operations. For example, if the CO_2 level in the greenhouse is too high, the environment is ventilated through specific air pumps.

In the project, the information collected by sensors is communicated through the *MQTT protocol*, and the greenhouse data are supervised using a SIEM tool.

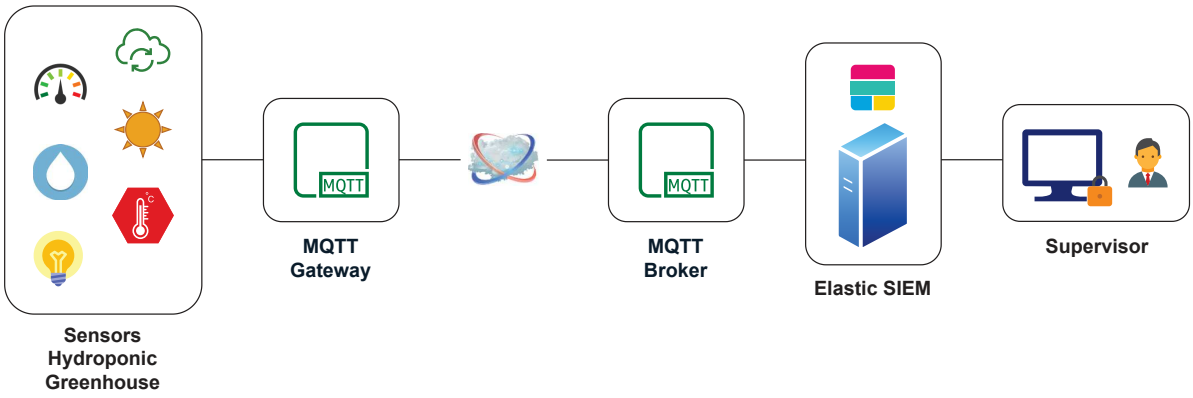


Figure 8.4: Structure of hydroponic greenhouse

Sensors are not real sensors but are simulated using a *Python script* which uses the Paho-MQTT library. The hydroponic greenhouse project which is simulated is a simplified greenhouse. The simulated sensors are represented and described in the following table.

	Description	**Range Value**	**Unit of Measure**
Temperature	Temperature inside the greenhouse	18 - 26	°C
Light Intensity	Number of hours during which plants are exposed to the sun	12 - 18	h
Humidity	Humidity in the air	65 - 70	%
CO_2	CO_2 is used in plants photosynthesis and allows to convert solar energy into chemical energy: • under normal circumstances • for tomatoes, lettuce, cucumbers, and peppers • for violets and Gerbera varieties	 1.000–1.300 800–1.000 500-800	ppm

EC	Electrical Conductivity measures the concentration of salts by measuring small currents in an aqueous solution	500 - 2000	$\mu S/cm$
Soil Moisture	Humidity in the soil	50 - 60	%
pH	Potential hydrogen which measures aqueous solution acidity and alkalinity	5.5 - 6	pH

The simulation script also works as *MQTT Gateway*. It generates and sends data sensors every second. These data represent the set of measurements taken by the sensors and collected by the gateway, which aggregates the information and send it to the broker.

The content of MQTT messages is structured in **JSON format**. It is a simple format which allows to exchange data and is independent of the language. The messages contain information about the readings of the sensors and the timestamp which represents the sensor event.

The following JSON structure is used.

```
{
    "@timestamp": "2021-12-19T00:00:00",    // timestamp of sensor data
    "temperature": 20,                      // temperature in Celsius
    "lightState": true,                     // light OFF or ON
    "minutesLightON": 120,                  // minutes light ON
    "minutesLightOFF": 140,                 // minutes light OFF
    "humidity": 67,                         // greenhouse humidity
    "CO2": 950,                             // CO2 level
    "EC": 900,                              // EC value
    "soil": 55,                             // percentage of soil moisture
    "pH": 6                                 // pH value
}
```

The MQTT topic is used to distinguish the various plantations in the greenhouse. Each plantation message has `/greenhouse` as the base level, and the next level represents the type of plantation, e.g. tomatoes, cucumber, lettuce, strawberry, Gerbera, etc.

The task of the script is to simulate the sensors of the hydroponic greenhouse and send them to the MQTT Broker, using a **level 1 QoS** that guarantees that a message is sent to the receiver at least once. The data sent will be available to the SIEM, with which a dashboard will be created allowing a supervisor to oversee the greenhouse.

8.3.1 Simulator

The Python script has the task of simulating the sensors which are used in a hydroponic greenhouse. It generates and sends MQTT messages to the MQTT Broker, containing sensor measurements which are aggregated by the MQTT Gateway. The content of the message is structured using the JSON format with the fields described in the previous paragraph.

The Python script can have different behaviours depending on the **command line arguments**. The only mandatory argument is the ***plant type***; using this value the simulator returns sensor values which conform to the plantation as in the table described in the previous paragraph.

```
python simulator.py plantation [options]
```

In addition to this argument, the script has ***13 other arguments***. *Seven* of them allow to set the values generated by the simulator, which generates correct values if the argument is not present, otherwise, it sends incorrect ones.

1. `-t` or `--temp`: for temperature.
2. `-l` or `--light`: for the duration of the light on.

3. `-u` or `--hum`: for the humidity of the air.
4. `-c` or `--co2`: for CO_2 levels.
5. `-e` or `--ec`: for EC values.
6. `-m` or `--soil`: for soil moisture.
7. `-p` or `--ph`: for the pH value.

Five other arguments give the possibility to set the simulations:

8. `-n SIM` or `--sim=SIM`: number of simulations which are executed. The default value is 144, one simulation every 10 minutes daily.
9. `-s SEED` or `--seed=SEED`: allows setting the seed used to generate the sensor values in order to repeat the tests.
10. `-D DAY` or `--day=DAY`: day number of the timestamp. The default value is 1.
11. `-M MONTH` or `--month=MONTH`: month number of the timestamp. The default value is 12.
12. `-Y YEAR` or `--year=YEAR`: year number of the timestamp. The default value is 2021.

In addition, there is the argument `-h` (or `--help`) which allows to show the helper. The OptParse library is used to parse command-line arguments.

The *libraries* used to generate the data and carry out the simulation are imported into the script.

```
import time                             #import time to sleep process
from datetime import datetime           #import to build timestamp
from datetime import timedelta
import json                             #import to parse json objects and strings
import numpy.random as rnd              #numbers generation
import paho.mqtt.client as mqtt         #import MQTT client
from optparse import OptionParser       #parse command-line arguments
```

The simulation script consists mainly of two classes:

1. **Simulator**: class for simulating the greenhouse's sensors.
2. **SensorSimulation**: executes the simulation of the greenhouse using the simulator and sends messages to the Broker.

The **Simulator** class is used to simulate the sensors.
The ***constructor method*** receives as parameters the plant type and a flag for each data measured by the sensors, which allows the simulator to decide whether to generate values in the correct range or not. As an optional parameter, it can receive the seed to make the tests repeatable. The ranges used are those described in the table in the previous paragraph:

```
# range sensors' value
__range_temp = (18, 26)
__range_hours_light = (12, 18)
__range_hum = (65, 70)
__range_co2 = [(800, 1000), (500, 800)]       # depends on plantation
__range_ec = (500, 2000)
__range_soil = (50,60)
__range_ph = (5.5, 6)
```

The initial value of the sensors is chosen randomly among the range values. Using the value representing the hours when the lighting is on, the photoperiod is calculated and constructed to represent when the greenhouse lights are on or off. In the constructor method, the timestamp is also initialized.

The class provides a method called ***simulate()***. The method simulates a measurement made by the sensors, representing the aggregation of the measurements made in *10 minutes* inside the greenhouse. It returns the values of the fields used in the MQTT message and the timestamp which indicates the date and time of the measurement.
The sensor values are updated randomly with a variation obtained from a *Gaussian distribution* when the method is called. If the sensor value remains within the range and the update takes it out of it, the variation is reversed.

```
# Update value using Gaussian distribution
def __gauss_update(self, value, rng, mu=0, sigma=0.1, r=2, check_range=True):
    # compute gaussian value
    var_value = round(rnd.normal(mu, sigma), r)
    # update value
    new_value = round(value + var_value, r)
    # check value within range
    if check_range and (new_value > rng[1] or new_value < rng[0]):
        return round(value - var_value, r)
    return new_value
```

In addition, the timestamp is updated by increasing it by 10 minutes. In case the timestamp represents the start of a new day, the photoperiod is also updated as done for the constructor.

```
# Method which simulates sensors
def simulate(self):
        # Method which computes light state using photoperiod
        def compute_light_state(hours_simulation)

        # Method which calculates daily hours spent in the simulation
        def compute_hours_simulation(count_day_simulation)

        # Method which computes minutes
        def compute_minutes(date_minutes)

        # if 24 hours have passed, update the photoperiod
        hours_simulation = compute_hours_simulation(self.__count_day_simulation)
        if(self.__count_day_simulation > self.one_day_simulations):
                self.__tot_light_on, self.__act_light, self.__photoperiod =
                ↪  self.__photoperiod_builder()
                self.__minutes_light_on = self.__minutes_light_on.replace(hour=0,
                ↪  minute=0)
                self.__minutes_light_off = self.__minutes_light_off.replace(hour=0,
                ↪  minute=0)
                self.__count_day_simulation = 0

        # store sensors value
        timestamp, temperature, light_state, humidity, CO2, EC, soil_moisture, pH =
        ↪  self.__timestamp, self.__act_temp, compute_light_state(hours_simulation),
        ↪  self.__act_hum, self.__act_co2, self.__act_ec, self.__act_soil, self.__act_ph
        minutes_light_on = compute_minutes(self.__minutes_light_on)
        minutes_light_off = compute_minutes(self.__minutes_light_off)
```

```
        # update sensors value
        self.__act_temp = self.__gauss_update(self.__act_temp, self.__range_temp,
        ↪  sigma=0.25, r=1, check_range=(not self.__warning_temp))
        self.__act_hum = self.__gauss_update(self.__act_hum, self.__range_hum, sigma=0.2,
        ↪  r=2, check_range=(not self.__warning_hum))
        self.__act_soil = self.__gauss_update(self.__act_soil, self.__range_soil,
        ↪  sigma=0.2, r=2, check_range=(not self.__warning_soil))
        self.__act_ph = self.__gauss_update(self.__act_ph, self.__range_ph, sigma=0.03,
        ↪  r=1, check_range=(not self.__warning_ph))
        self.__act_co2 = self.__gauss_update(self.__act_co2,
        ↪  self.__range_co2[self.__idx_plantation], sigma=5, r=0, check_range=(not
        ↪  self.__warning_co2))
        self.__act_ec = self.__gauss_update(self.__act_ec, self.__range_ec, sigma=15,
        ↪  r=0, check_range=(not self.__warning_ec))

        #update timestamp
        self.__timestamp = self.__timestamp +
        ↪  timedelta(minutes=self.__minutes_sensor_data)
        self.__minutes_light_on = self.__minutes_light_on +
        ↪  timedelta(minutes=(self.__minutes_sensor_data if light_state else 0))
        self.__minutes_light_off = self.__minutes_light_off +
        ↪  timedelta(minutes=(self.__minutes_sensor_data if not light_state else 0))

        # update counter of simulation
        self.__count_day_simulation += 1

        return timestamp, temperature, light_state, minutes_light_on, minutes_light_off,
        ↪  humidity, CO2, EC, soil_moisture, pH
```

Finally, the method returns all the fields which are used to construct the JSON.

The other **SensorSimulation** class uses the simulator to generate the data and send it to the MQTT broker using the *Paho-MQTT library*. The IP address of the broker is that of the host where Mosquitto is installed: *192.168.10.50*. The MQTT messages are sent using a QoS of 1 and the topic uses as prefix `/greenhouse` and as suffix the plantation, for example, `/greenhouse/tomatoes`. The constructor method receives as parameters the type of plantation, the flags related to the simulation, the topic, the id value used to connect to the Broker, for example, `"TomGreenhouse"` and initializes the simulator.

The class has a method called ***simulate()*** which receives as a parameter the number of simulations to be carried out. It first connects to the MQTT server and subscribes to the topic. This allows checking if the messages have been correctly sent and received by the Broker. After opening the connection and enabling the reading and writing of messages, the method starts the simulation. Within a loop the `simulate` method of the `SensorSimulation` class object is executed, the message in JSON format is constructed and sent to the Broker. After waiting a few seconds, these operations are repeated. Once the simulation is complete, the connection to the server is closed.

```
def simulate(self, num_simulation=Simulator.one_day_simulations):
        # given parameters the function builds JSON object
        def __json_builder(timestamp, temperature, light_state, minutes_light_on,
        ↪  minutes_light_off, humidity, CO2, EC, soil_moisture, pH)

        # Callback function which allows to print received messages of a topic
        # It is used to check if messages have been correctly sent
        def __on_message(client, userdata, message):
```

```
8                 print("Received message:")
9                 print("\tMessage:", str(message.payload.decode("utf-8")))
10                print("\tTopic:", message.topic)
11
12        # create client connection
13        print("Creating", self.__plantation, "client connection...")
14        client = mqtt.Client(client_id=self.__client_id)
15        # set callback which allows to received messages of a topic
16        client.on_message = __on_message
17        # connect to MQTT broker
18        print("Connecting to MQTT broker...")
19        client.connect(broker_address)
20
21        # enable reading and writing data
22        client.loop_start()
23
24        # start to send messages.
25        # It stops the simulation using the number of simulations
26        for i in range(num_simulation):
27
28                # subscribe topic to read messages
29                print("Subscribing to topic:", self.__topic)
30                client.subscribe(self.__topic)
31
32                # publish messages to topic
33                print("Publishing message to topic:", self.__topic)
34
35                # simulate sensor value and build JSON object
36                timestamp, temperature, light_state, minutes_light_on, minutes_light_off,
                  ↪  humidity, co2, ec, soil_moisture, ph = self.__simulator.simulate()
37
38                # build JSON object
39                json_obj = __json_builder(timestamp, temperature, light_state,
                  ↪  minutes_light_on, minutes_light_off, humidity, co2, ec,
                  ↪  soil_moisture, ph)
40
41                # publish message
42                client.publish(self.__topic, json.dumps(json_obj), qos=qos)
43
44                # wait to simulate
45                time.sleep(sleep_time)
46
47        # stop the loop
48        client.loop_stop()
49        client.disconnect()
```

The main method has the task of parsing the command line, initializing the simulation, and executing it.

The complete Python code of the simulator with comments can be found in the appendix in paragraph A.1.

8.4 Network Structure

In order to run the simulation of the hydroponic greenhouse, virtual computers are created with different tasks such as simulating sensors, collecting data, and supervising through Elastic SIEM.
Computers are virtualized using **Oracle Virtualbox 6.1.30**, an open-source software which offers a performing system and allows to virtualize a large number of operating systems.

To implement the structure of a hydroponic greenhouse supervised by a SIEM, 3 different PCs with different tasks are used.
The hosts' structure to simulate the hydroponic greenhouse can be schematized as follows:

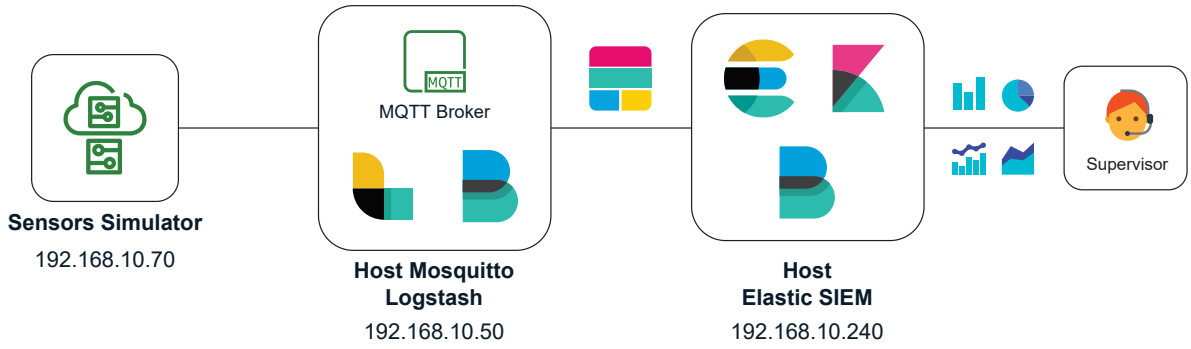


Figure 8.5: Architecture of hydroponic greenhouse simulation

A fourth host is used to execute *attacks* using Kali Linux, which is usually used for ethical hacking. The host is located within the network.

8.4.1 Sensors Simulator

The operating system which is used in the first computer is ***Lubuntu***. This OS is a lightweight system with small consumption of hardware resources. The computer which has installed this operating system has the **task of simulating the sensors** of the greenhouse and the MQTT gateway. It is used to run the *Python - Paho-MQTT script* described in the previous chapter. The gathered JSON data are sent to the MQTT server.
In the following table, there are virtual machine characteristics.

Sensors Simulator	
# Processor:	2
Disk Size:	10 GB
RAM Memory:	4 GB
Network adapters:	2 1. Bridged networking 2. Internal networking: *greenhousenet*
IP Address:	192.168.10.70
Operating System:	Lubuntu 21.10

8.4.2 Host Mosquitto - Logstash

A second PC works as **MQTT Broker** and ***Ubuntu Server*** is used as OS. Its role is to receive the MQTT messages sent by the simulator, save and forward them. In this PC the Mosquitto software is installed which is an implementation of the MQTT Server.
Also, **Logstash** and **Beats** which are part of Elastic Stack are installed. The Beats installed are:

- **Filebeat**: is used to collect information from messages sent via the MQTT protocol
- **Packetbeat**: is used to sniff network traffic and monitor applications and performance.

Data gathered through *Packetbeat* are sent directly to the Elasticsearch engine, while information collected through *Filebeat* is transformed and parsed with Logstash and sent to SIEM.
In the appendix paragraph A.2, the installation and configuration are described.
The following table shows the virtual machine characteristics.

Host Mosquitto - Logstash	
# Processor:	3
Disk Size:	40 GB
RAM Memory:	4 GB
Network adapters:	2 1. Bridged networking 2. Internal networking: *greenhousenet*
IP Address:	192.168.10.50
Operating System:	Ubuntu Server 21.10

8.4.3 Host Elastic SIEM

Also, in the third computer is installed ***Ubuntu Server*** as the operating system. On this PC are installed:

- **Elasticsearch**: the heart of Elastic SIEM;
- **Kibana**: allows to visualize the information received from hosts;
- **Packetbeat**: used to sniff network traffic about the Elastic SIEM host.

Thanks to this host, a supervisor can see some dashboard filled with information about the network traffic generated by the devices in the network and the values of the simulated sensor.
In the following table, there are virtual machine characteristics.

Host Elastic SIEM	
# Processor:	4
Disk Size:	80 GB
RAM Memory:	8 GB
Network adapters:	2 1. Bridged networking 2. Internal networking: *greenhousenet*
IP Address:	192.168.10.240
Operating System:	Ubuntu Server 21.10

8.4.4 Attacker host

A host on which is installed ***Kali Linux*** as an operating system it's used to execute attacks. The OS is a GNU/Linux distribution, generally used in the field of computer security, in particular for penetration testing.
Virtual machine characteristics are described in the following table.

Attacker host	
# Processor:	2
Disk Size:	30 GB
RAM Memory:	4 GB
Network adapters:	2 1. Bridged networking 2. Internal networking: *greenhousenet*
IP Address:	192.168.10.10
Operating System:	Kali 2021.4

8.5 Simulation of Hydroponic Greenhouse supervised by SIEM

After completing the configuration of the hosts and creating the dashboards, the entire system is activated. First, the virtual machines are started, and then the simulation script on the host at address *192.168.10.70* is executed.

8.5.1 Network Attacks

In the application section, various types of **attacks** are performed. The purpose is to use the SIEM to supervise the system by analysing the network packets passing through the network. In this way, the behaviour of the SIEM is evaluated in order to evaluate and improve the security of the network.

Several attacks are performed against the two most important hosts of the simulation: the MQTT Broker and the host where the Elasticsearch engine is installed. The attackers try to stop the main services running in these hosts. The attacks that are performed are:

- **ICMP Flood**: this is a DoS attack using the ICMP protocol with the aim of stopping the operation running on the Elasticsearch hosts.
- **DoS attack on MQTT port**: the attacker blocks the MQTT Broker's service on port 1883 thus interrupting the operation of the data collection system.
- **DoS attack on Kibana**: the attack stops the operation of the Kibana service, blocking the dashboard shown to the supervisor.

The attacks are executed using **hping3**. This software generates packets for the TCP/IP protocol, allowing attacks to be launched on victim hosts.
The main parameters that are used in the application section: [57]

- `--flood`: mode in which messages are sent as fast as possible.
- `--icmp`: to activate the ICMP mode.
- `-S`: set SYN flag.
- `-p <port>`: destination port.
- `-i <packet sending time interval>`: message sending time interval.

- `<attacked host IP address>`: IP address of the attacked host.

The dashboards available to the security team show the traffic generated by the devices in the network, representing the source and destination. The main host generating network traffic is the one simulating the sensors.

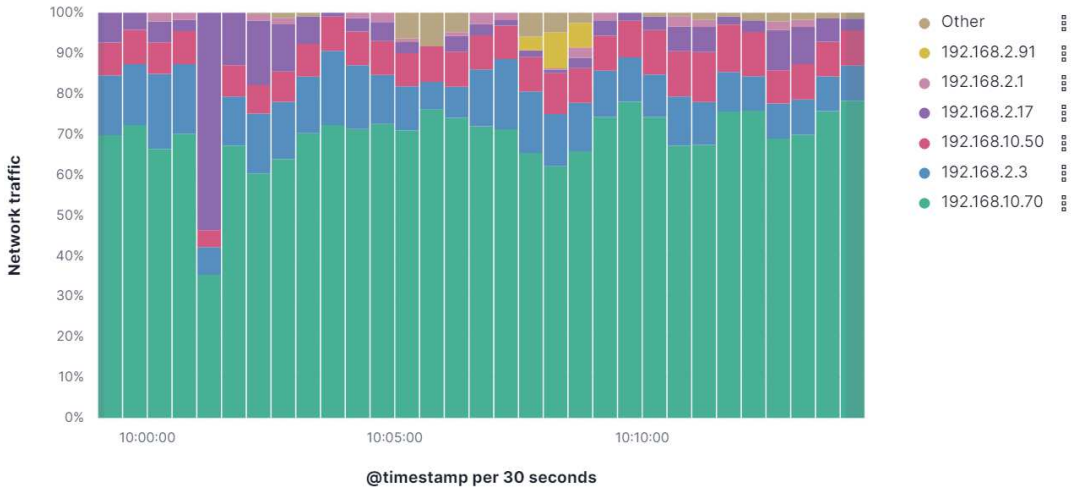


Figure 8.6: Percentage graph of network traffic in normal situation

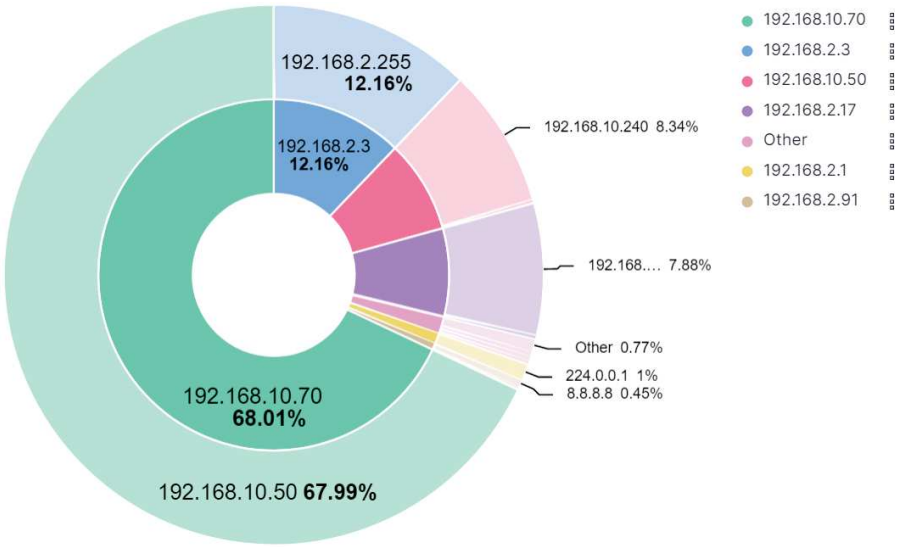


Figure 8.7: Pie chart of network traffic in normal situation

In addition, the status of the MQTT Brokers is displayed while the sensors are sending messages.

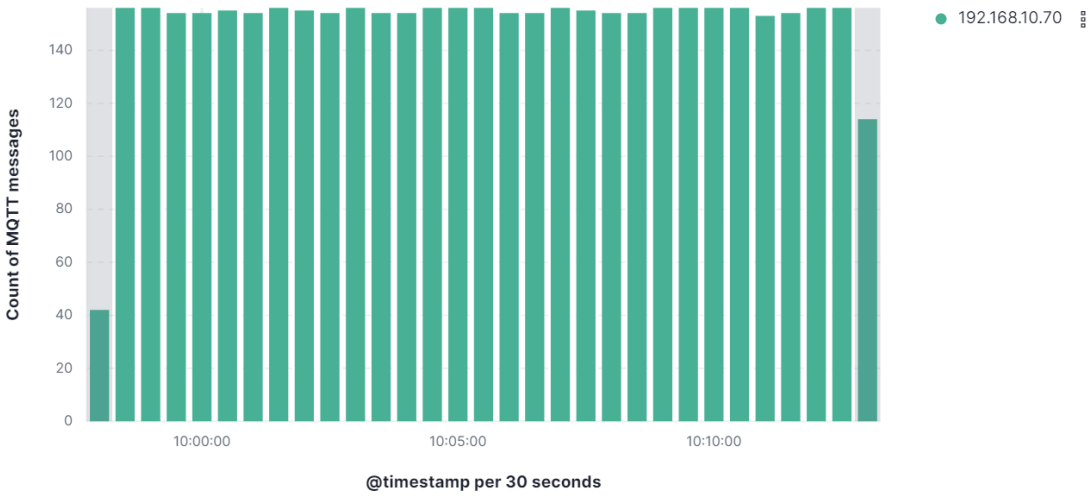


Figure 8.8: Histogram of MQTT packets in normal situation

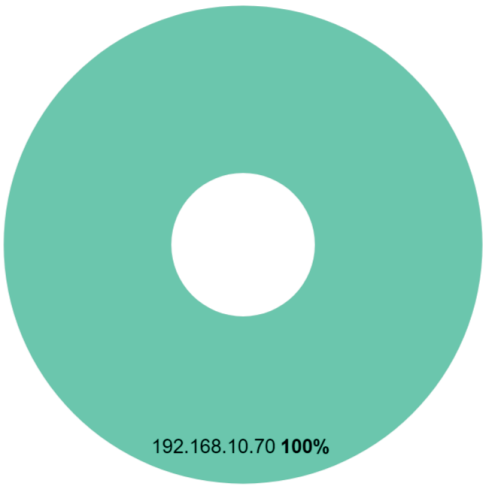


Figure 8.9: Pie chart of MQTT packets in normal situation

A portion of the dashboard is used to show the status of the server. Specifically, ICMP packets and requests to port *5601* of Kibana are shown.

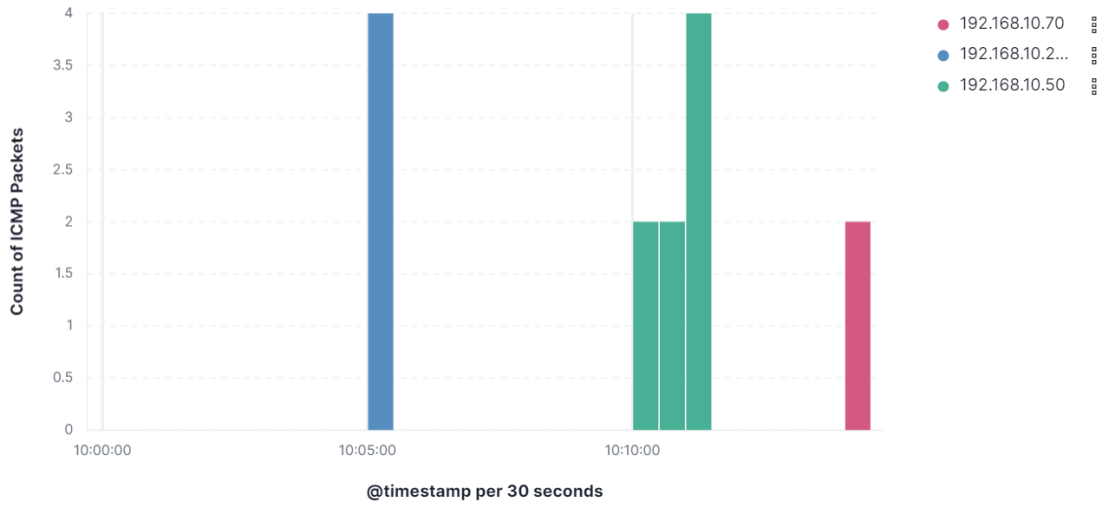


Figure 8.10: Histogram of ICMP packets in normal situation

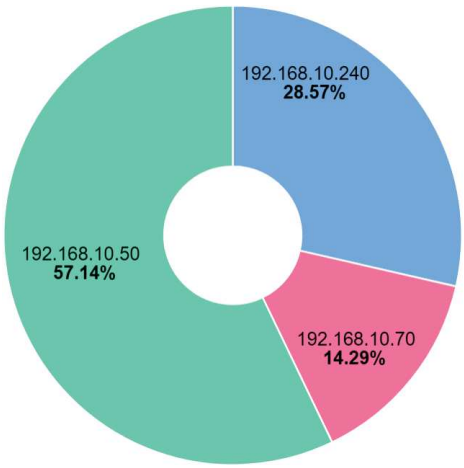


Figure 8.11: Pie chart of ICMP packets in normal situation

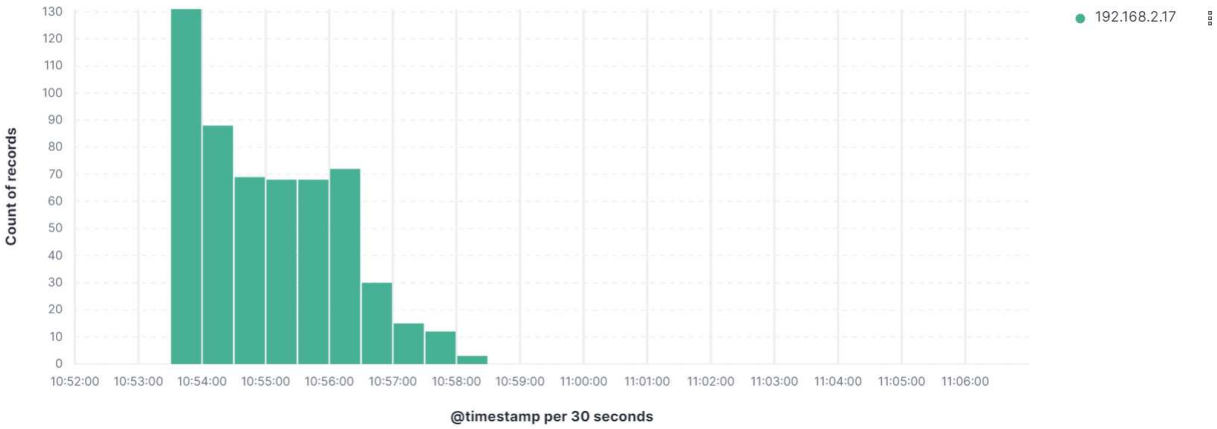


Figure 8.12: Histogram of HTTP requests in normal situation

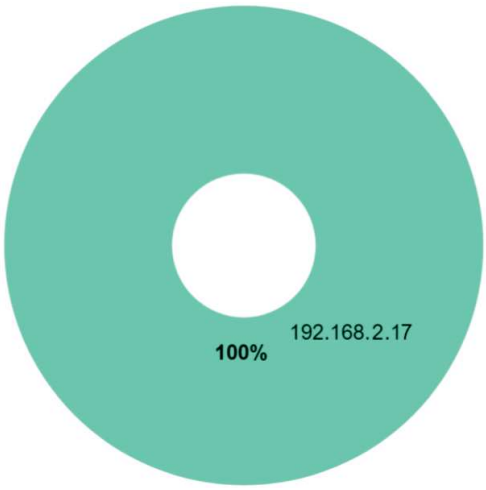


Figure 8.13: Pie chart of HTTP requests in normal situation

Attacks lead to alterations of these situations.

8.5.1.1 ICMP Flood

The first attack that is performed at the network is the **ICMP Flood**. In this attack, the attacker sends a large number of requests to the host where the heart of the SIEM is installed: Elasticsearch, using the attacking host with Kali Linux installed as the operating system. In this way, the attacker succeeds in blocking the operation of the system.

The following command is used to launch the attack:

```
hping3 --flood --icmp -S 192.168.10.240
```

The situation that can be observed is the following.
Network traffic shows that a new host with IP address *192.168.10.10* is generating a large amount of traffic into the network, disrupting the following graphs.

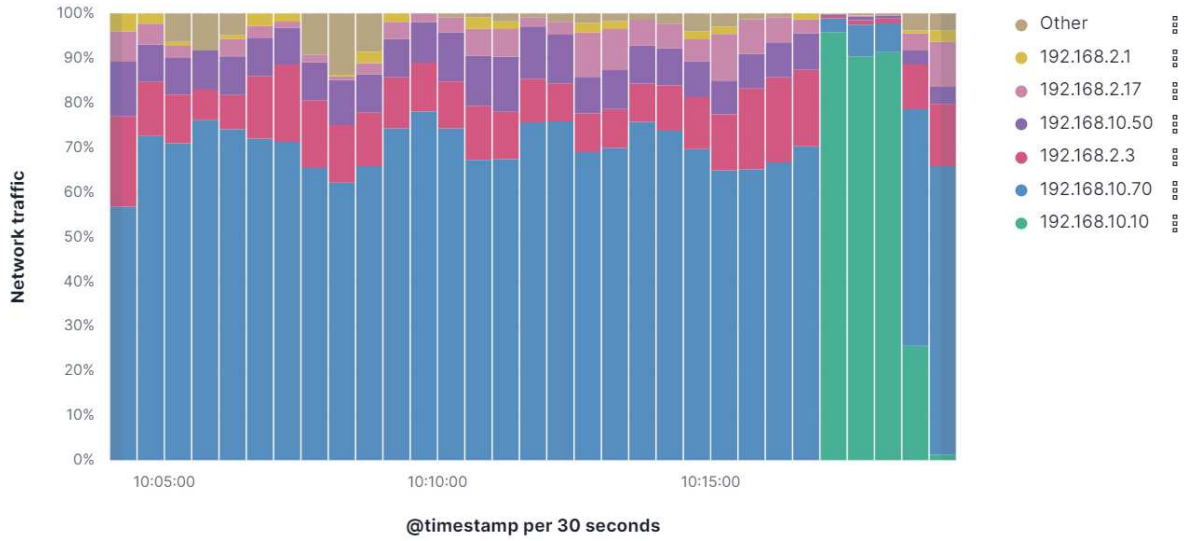


Figure 8.14: Percentage graph of network traffic during the attack

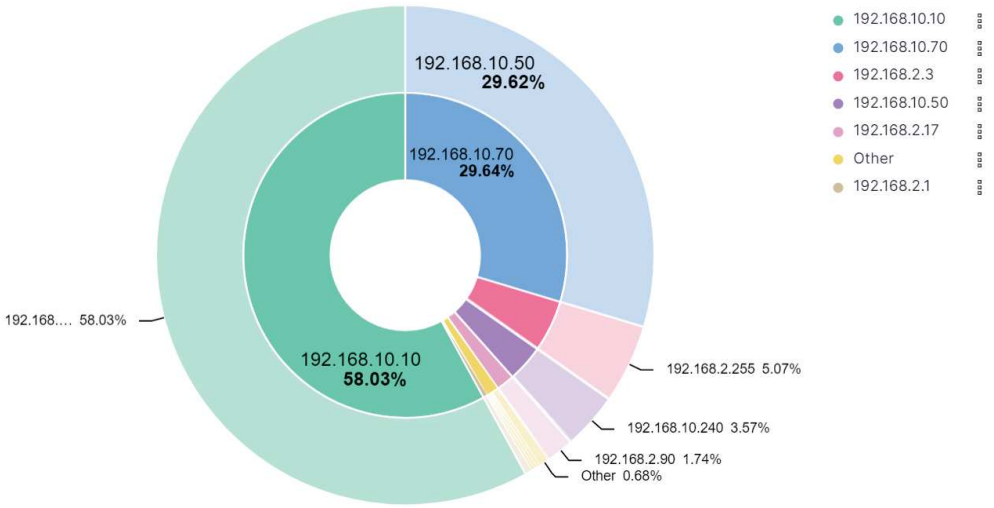


Figure 8.15: Pie chart of network traffic during the attack

The traffic generated by the attacking host is sent to the host with Elasticsearch installed. This can be observed in the following graphs, which show that the packets are directed to the host.

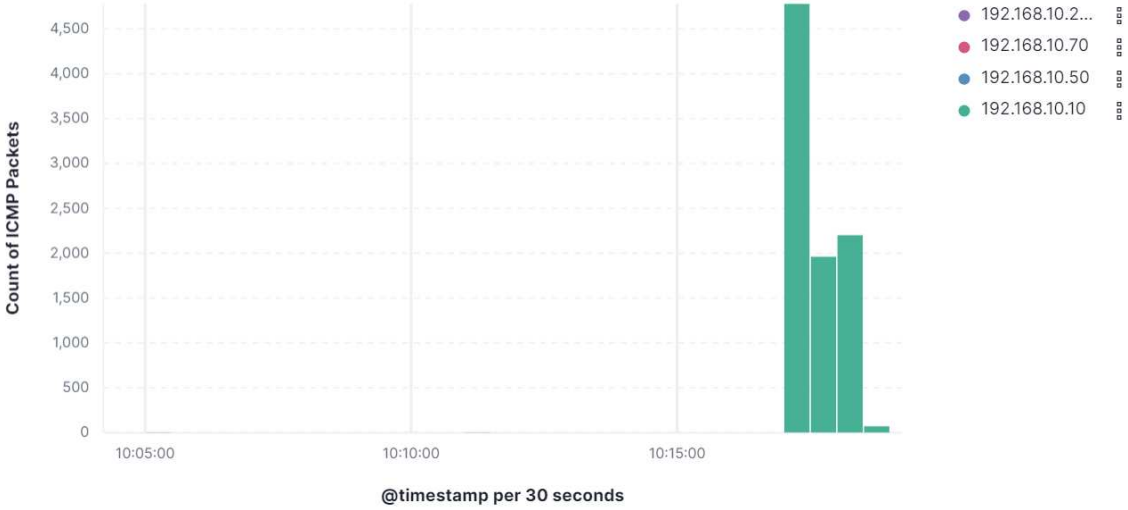


Figure 8.16: Histogram of ICMP packets during the attack

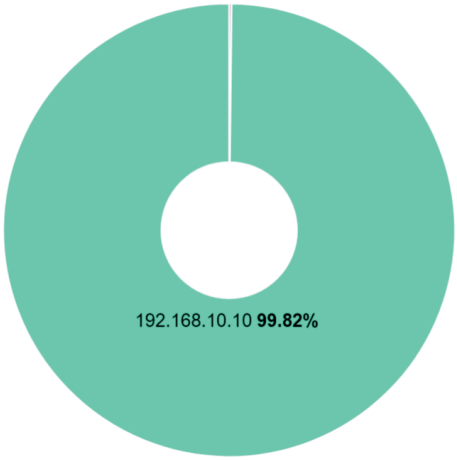


Figure 8.17: Pie chart of ICMP packets during the attack

One idea for blocking this type of attack is to interrupt the process of the ICMP protocol. However, this solution makes the host no pingable.

8.5.1.2 DoS attack on MQTT port

The Kali host is also used in this attack. A large number of requests are sent to **port 1883** of the MQTT broker, blocking the service and thus interrupting the functioning of the system's data collection.

To perform the attack, the following command is executed, flooding the port *1883* where MQTT runs:

```
hping3 --flood -S -p 1883 192.168.10.50
```

It can be seen that the attacking host generates high traffic as in the previous attack, and the graphs that can be observed are similar to those of the ICMP flood.

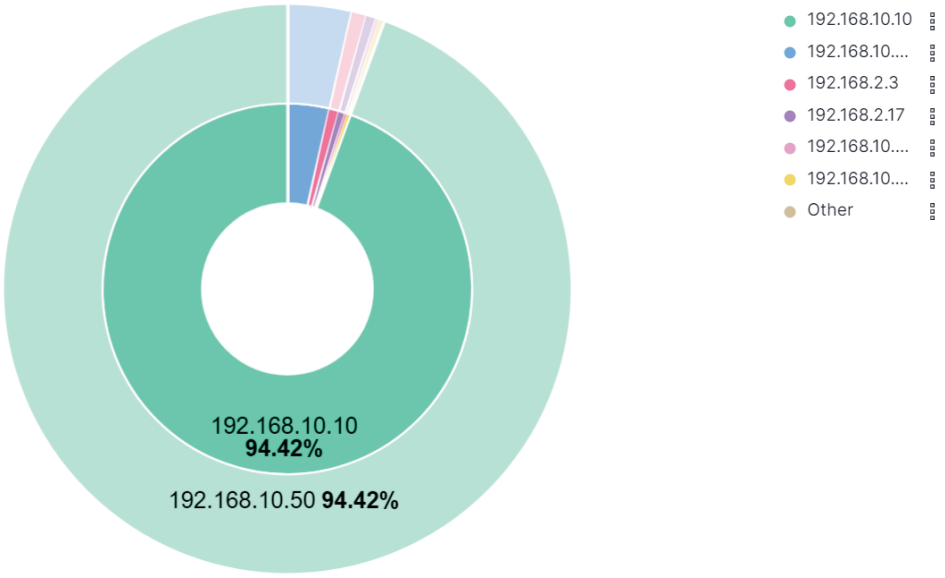


Figure 8.18: Pie chart of network traffic during the attack

Concerning the graphs representing the MQTT packets sent to the broker, a very high increase is observed for those transmitted by the attacker.

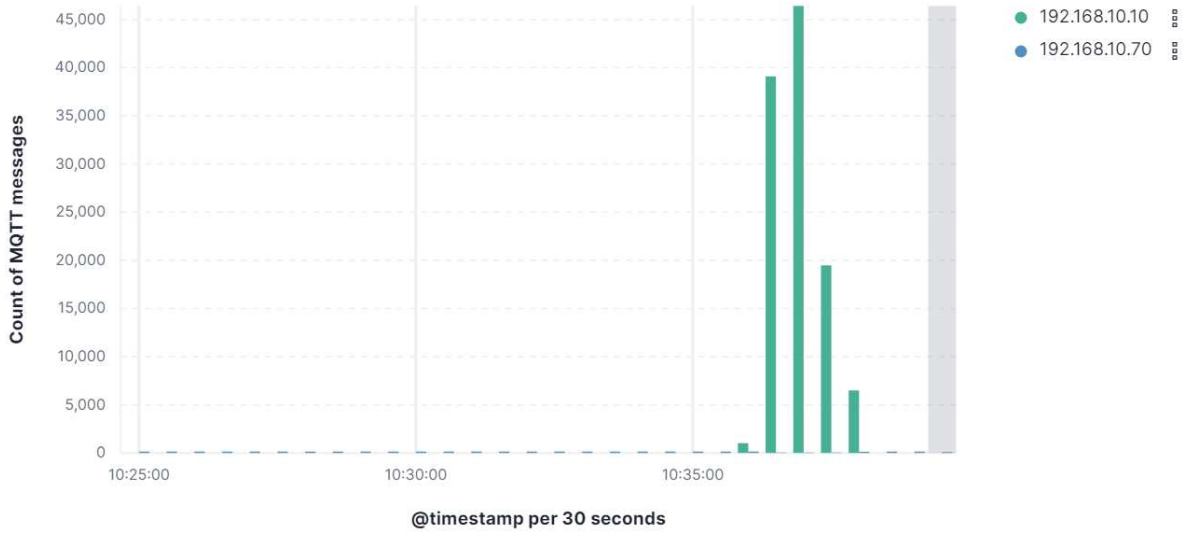


Figure 8.19: Histogram of MQTT packets during the attack

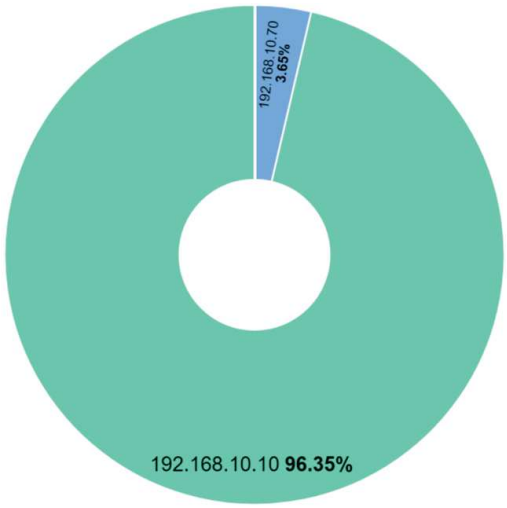


Figure 8.20: Pie chart of MQTT packets during the attack

8.5.1.3 DoS attack on Kibana

The last DoS attack that is launched targets **Kibana**. The attacker tries to interrupt the normal operation of the service by blocking the data dashboards to the supervisor.

The attack is launched with the following command:

```
hping3 -i u1 -S -p 5601 192.168.10.240
```

The situation that can be observed concerning network traffic is the same as the two attacks previously analysed, with the attacker's traffic suddenly increasing.

The graphs showing the packets received by host *192.168.10.240* on Kibana port *5601*, highlight the attack by displaying the high number of packets received during the attack.

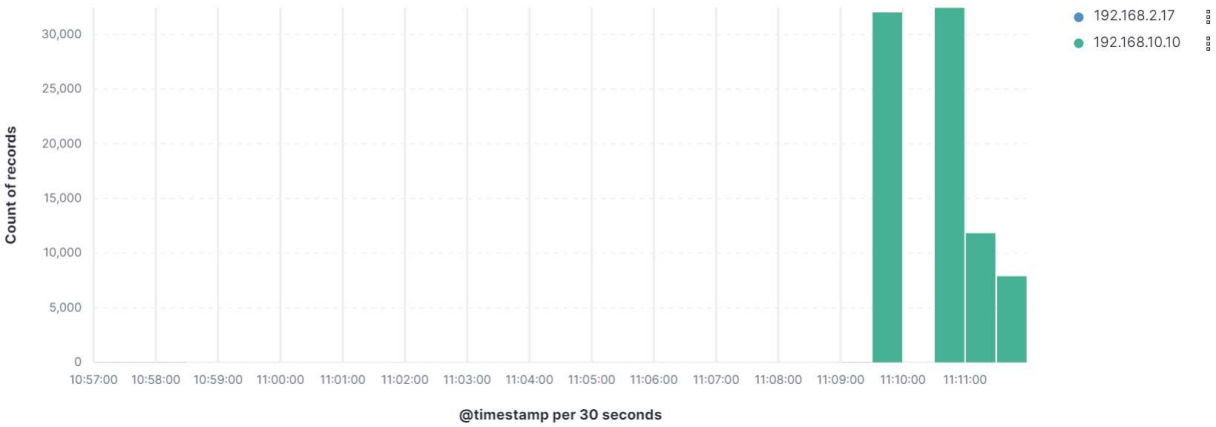


Figure 8.21: Histogram of packets sent to Kibana port during the attack

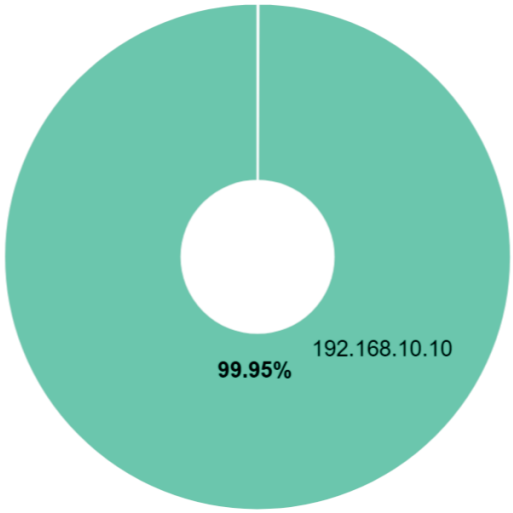


Figure 8.22: Pie chart of MQTT packets sent to Kibana port during the attack

8.5.2 Sensors Data

Concerning the information collected by the sensors, in the first run, a simulation is performed without any simulated values going out of the correct ranges. In order to do this, no parameters are added to the command line while the script is run, except for the seed to repeat the tests. Both the *tomato* and *gerbera* greenhouse simulators are executed simultaneously.

```
python3 simulator.py tomatoes --seed 10 -D 1 -M 12 -Y 2021
python3 simulator.py gerbera --seed 15 -D 1 -M 12 -Y 2021
```

Then, starting timestamp is changed and it is started with all parameters active in order to observe the behaviour of the tool if the values are not correct.

```
python3 simulator.py tomatoes --seed 20 -n 1440 -D 5 -M 12 -Y 2021 -t -l -u -c -e -m -p
python3 simulator.py gerbera --seed 25 -n 1440 -D 5 -M 12 -Y 2021 -t -l -u -c -e -m -p
```

The dashboard that is analyzed concerns a greenhouse in which **Gerberas** are grown.
In the first part of the dashboard, all numerical metrics representing the last detected value by the greenhouse sensors are displayed.

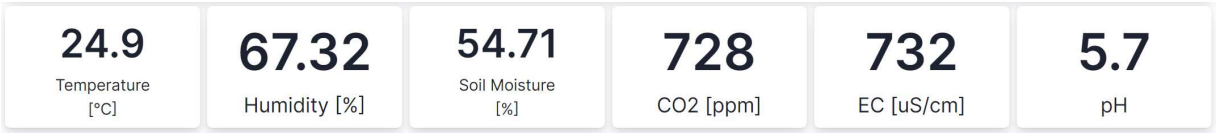


Figure 8.23: Sensors value of Gerberas' greenhouse

Successive, a table is displayed showing all the values measured by the sensors in temporal order using the timestamp. By looking at the table, the supervisor can observe in real-time the status of all the parameters that are controlled by the IoT devices.

↓ Timestamp	Temperature [°C]	Light State	Light ON [min]	Light OFF [min]	Humidity [%]	Soil Moisture [%]	CO2 [ppm]	EC [uS/cm]	pH
23:50	25.6	true	830	540	67.74	53.49	715	662	5.7
23:40	25.4	true	820	540	67.83	53.48	715	665	5.7
23:30	25.4	true	810	540	68.22	53.47	715	649	5.7
23:20	25.2	true	800	540	68.3	53.67	722	628	5.7
23:10	25.1	true	790	540	68.35	53.24	726	634	5.7
23:00	24.8	true	780	540	68.64	53.64	724	619	5.7
22:50	25.1	true	770	540	68.71	53.6	718	596	5.7
22:40	25	true	760	540	69.11	53.89	723	605	5.8
22:30	25.1	true	750	540	69.06	54.11	723	611	5.8

Figure 8.24: Table of sensors value

In this table, the data are shown in chronological order and are highlighted using colors in this way the supervisor can quickly see if there are any anomalies in the measurements. Examples are:

1. temperature too low or too high

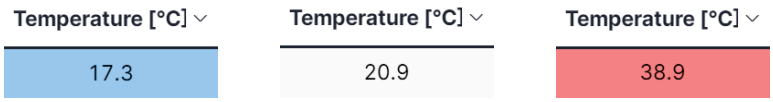


Figure 8.25: Color measured temperature displayed in the table

2. solutions too acidic or alkaline

Figure 8.26: Color measured pH level displayed in the table

3. environment too bright or too dark

Figure 8.27: Color measured EC displayed in the table

4. CO_2 levels too high or low

Figure 8.28: Color measured EC displayed in the table

5. electrical conductivity too high

Figure 8.29: Color measured EC displayed in the table

6. environment too humid

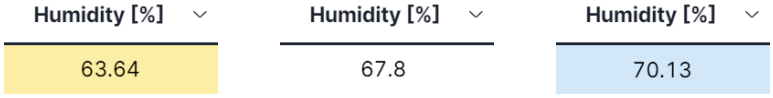


Figure 8.30: Color measured humidity displayed in the table

7. too high or low soil moisture

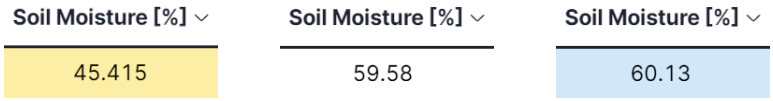


Figure 8.31: Color measured soil moisture displayed in the table

The next section of the dashboard shows the *temperature* details. Using a line graph, the temperatures measured by the sensors over a user-defined period are displayed. In addition, there are two horizontal lines in the graph describing the minimum and maximum temperature. The last value measured in the greenhouse is also displayed.
This visualization allows the user to observe a history of the measurements and at the same time see the current temperature in the greenhouse.

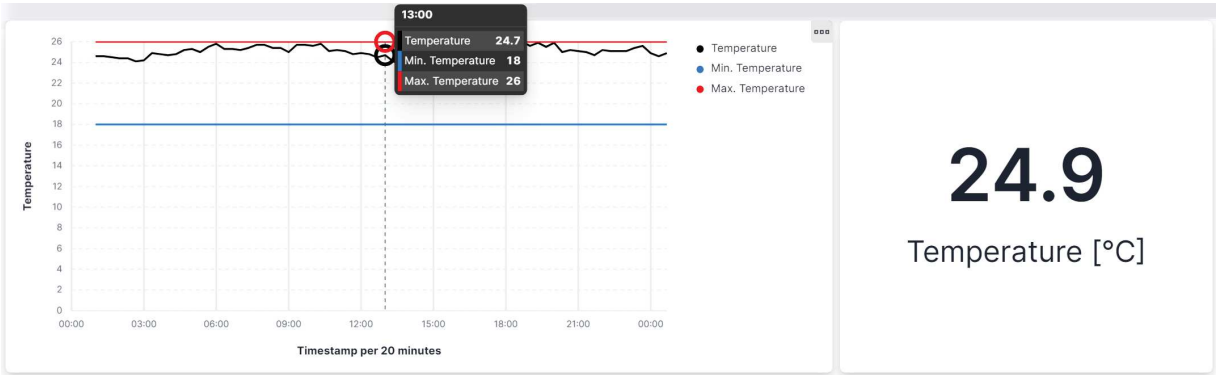


Figure 8.32: Temperatures graph

In case the sensor values are out of range, the line representing the temperature measurements doesn't remain between the horizontal lines.

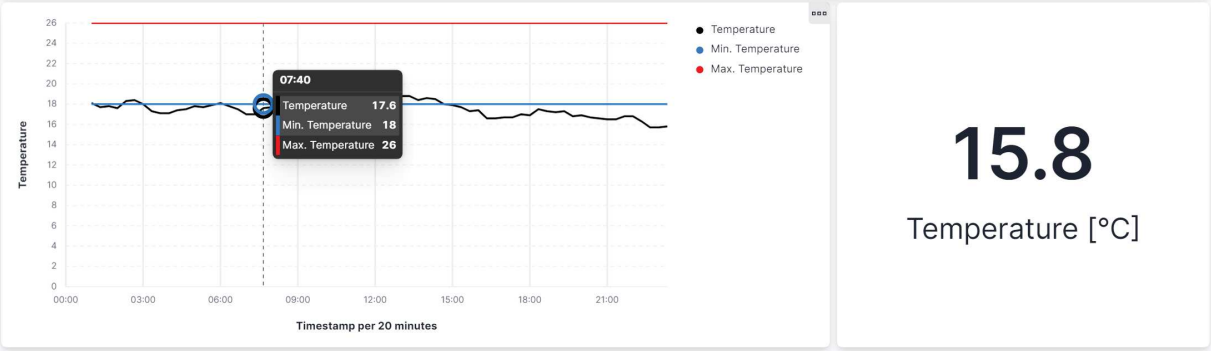


Figure 8.33: Temperatures graph out of range

Similarly, a line graph representing the *humidity* values and its real-time measured value are displayed. Also for this measurements, two horizontal lines are indicating the range of values for the greenhouse containing the Gerberas.

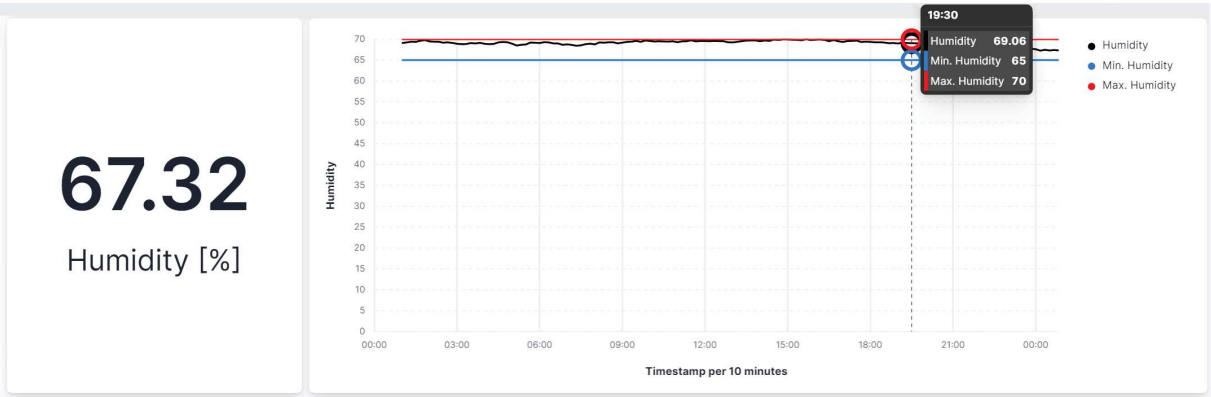


Figure 8.34: Humidity graph

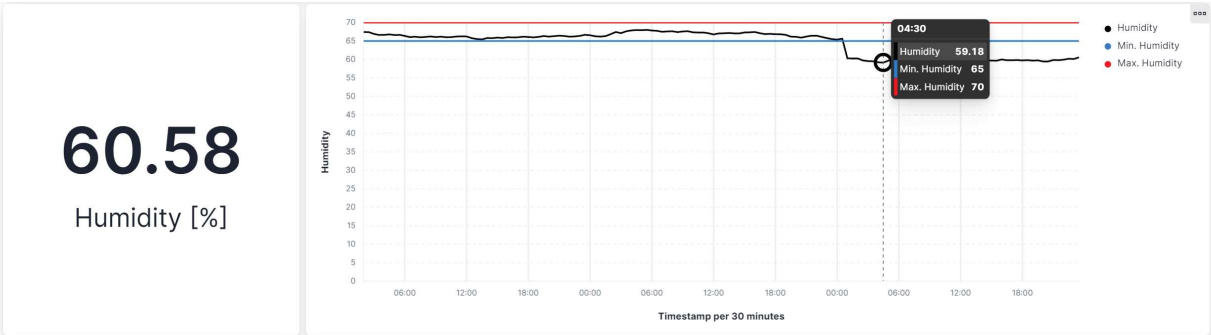


Figure 8.35: Humidity graph out of range

The photoperiod is represented using a bar graph. In this graph, the different bars represent the minutes of light and dark in the greenhouse since midnight. At the end of the day, this graph represents the total hours of light and dark.

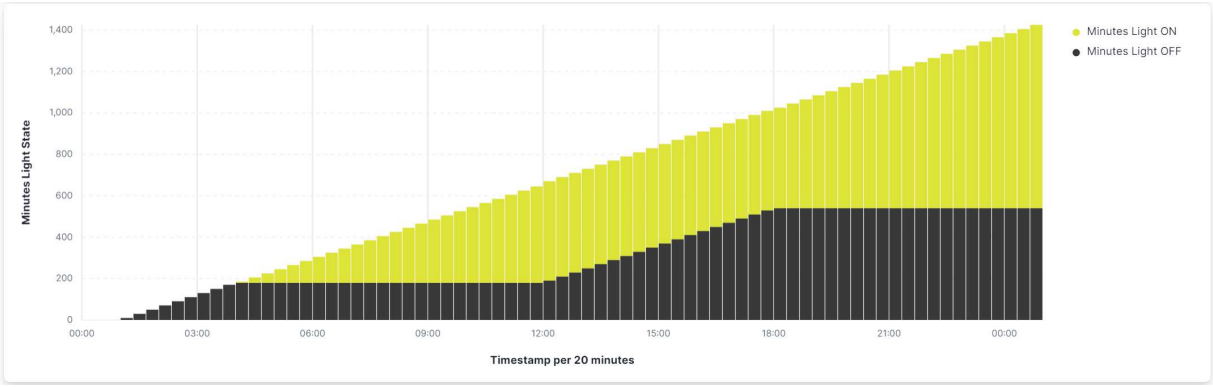


Figure 8.36: Photoperiod graph

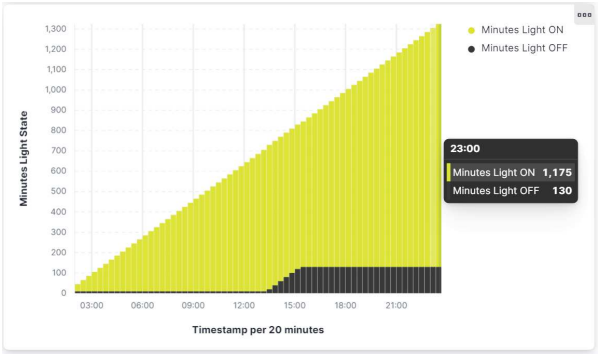


Figure 8.37: Overly bright photoperiod graph

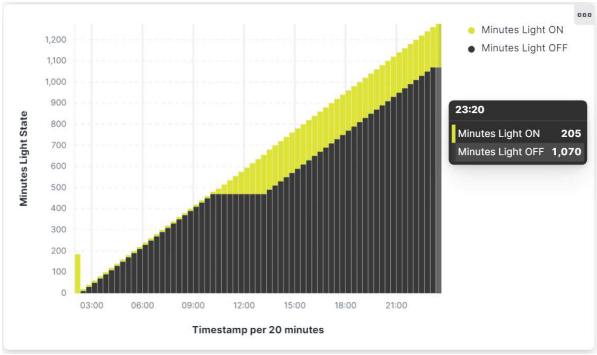


Figure 8.38: Photoperiod graph too dark

CO_2 levels must also be monitored. This is because CO_2 is used by plants for photosynthesis and helps them to grow. The CO_2 values are represented using a line graph with the limits and the last measured value is also displayed.

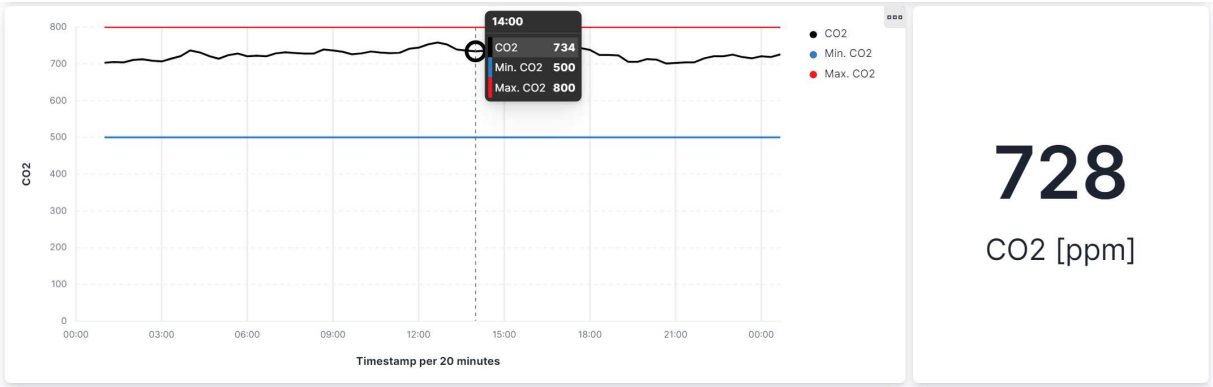


Figure 8.39: CO_2 graph

Figure 8.40: CO_2 graph out of range

In the same way as temperature, humidity, and CO_2 levels, *electrical conductivity*, *pH*, and *soil moisture* are represented using a line graph and the value is the last value measured by the sensors.

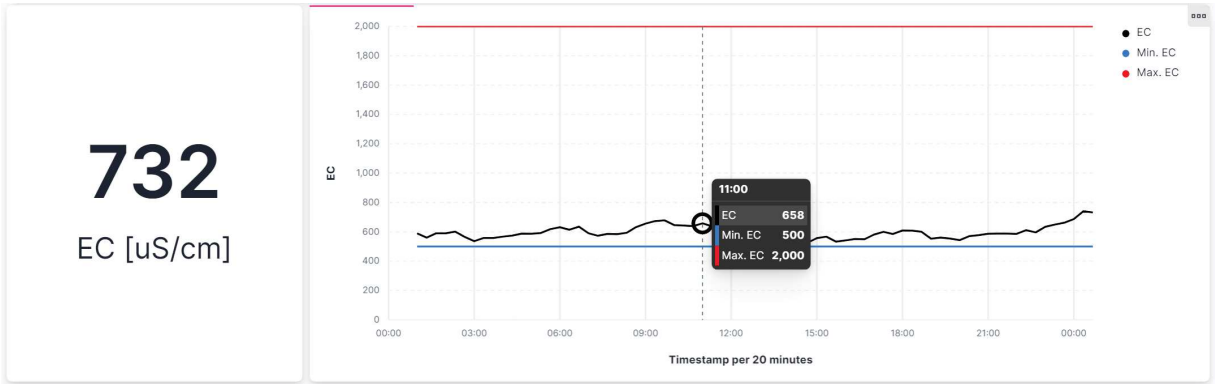


Figure 8.41: Electrical Conductivity graph

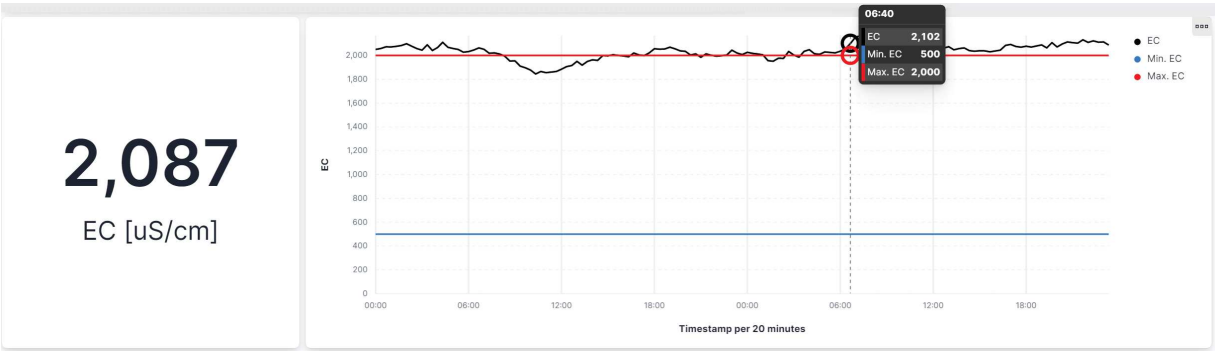


Figure 8.42: Electrical Conductivity graph out of range

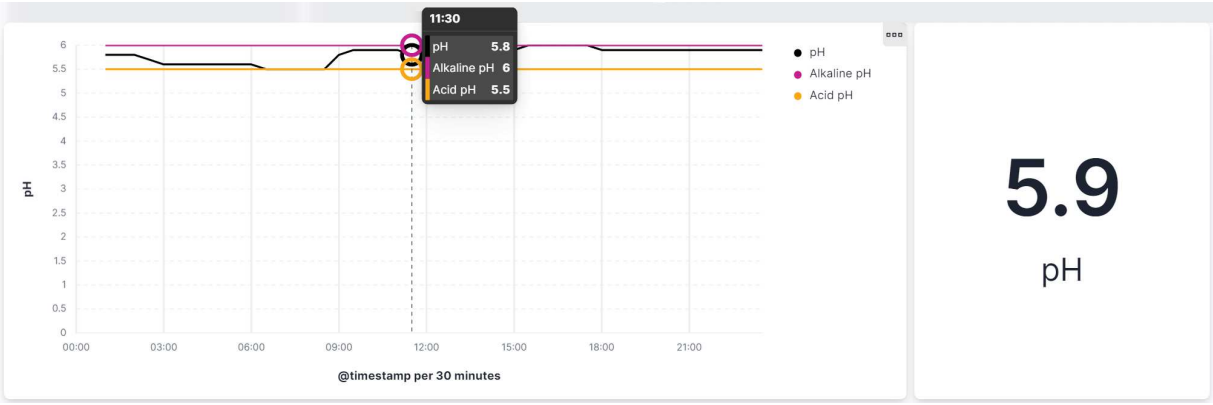


Figure 8.43: pH graph

Figure 8.44: pH graph out of range

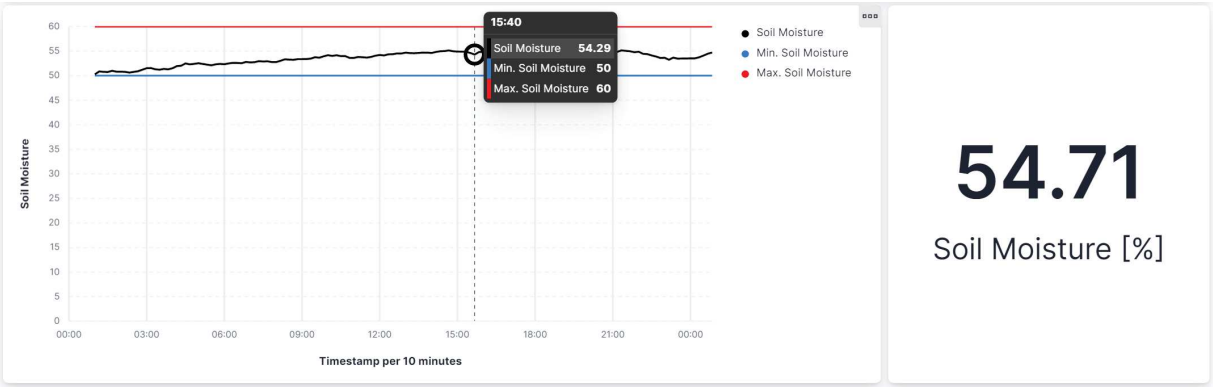


Figure 8.45: Soil moisture graph

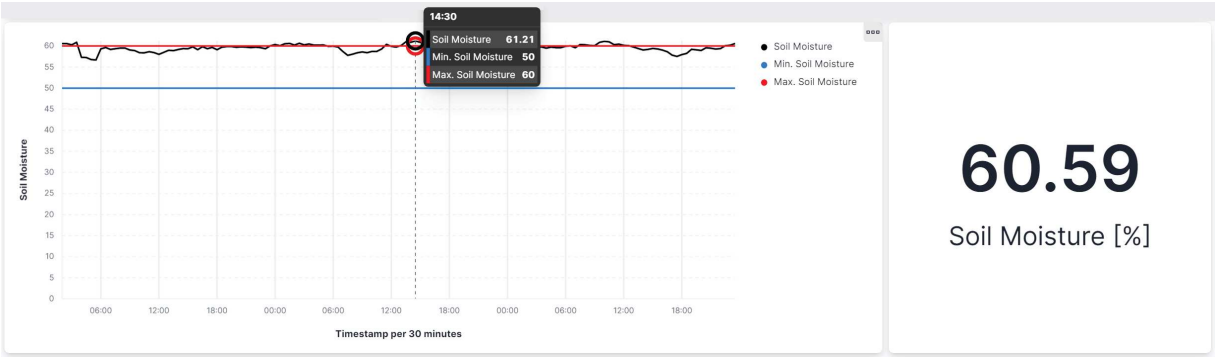


Figure 8.46: Soil moisture graph out of range

The dashboard for the **tomatoes** simulation is the same as the one for the Gerbera cultivation, but the data are filtered using the topic `/greenhouse/tomatoes` and the ranges are adapted to the values, especially in the case of CO_2.

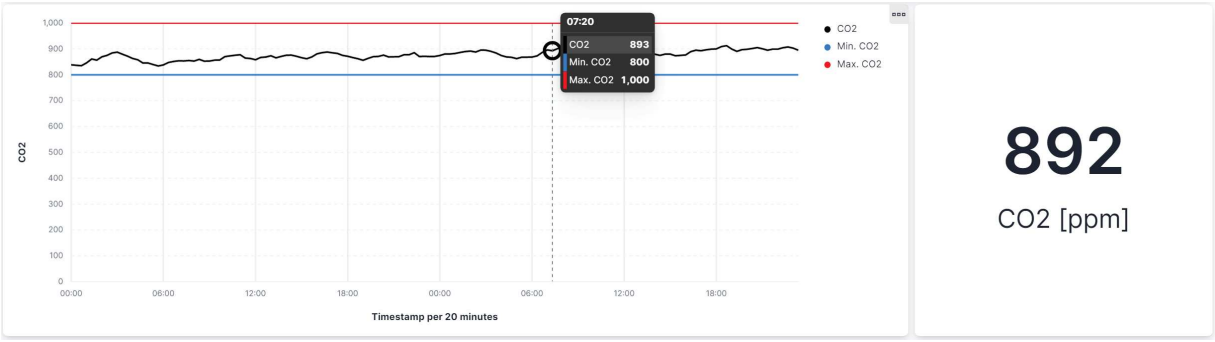


Figure 8.47: CO_2 Graph of tomatoes' greenhouse

9 Conclusion

The **growth of intelligent devices**, able to collect information from the surrounding environment and share it using a network, and able to simplify and help everyday life, has led to the use of these devices in the everyday life of every person and within complex systems. However, this has led to new security issues within the systems due to security problems, such as authentication and authorization, because the devices do not have important protection from attackers due to the fact that they often are composed of elementary hardware.
For these reasons and the high diffusion of these devices in many fields, they become of interest at the security level.

One of the possible applications is in the agricultural sector.
In the presented **project**, IoT devices and SIEMs are combined. Some attacks that can be performed in the system are simulated. The attacks are **DoS attacks**, aiming at blocking the main working services of the system: Elasticsearch, the heart of the SIEM, and the MQTT Broker.
The sensors inside the greenhouse are also simulated. Simulating the sensors and monitoring them through the SIEM, allow control the environment and the IoT devices using the functionalities provided by the SIEM. The SIEM in fact allows collecting historical and real-time information from the system making them easily accessible for analysis.

In the presented application the SIEM allows identifying attacks by collecting network packets involving the main hosts of the system.
In addition, the IoT sensors inside the greenhouse communicate with a server using the MQTT protocol; this is a lightweight and easy to implement data protocol. The data collected inside the greenhouse is sent to an MQTT Broker, which collects the information according to topics and distributes it to subscribers of these topics using a publish and subscribe model.
The collected network packets and data are filtered and analysed using a SIEM, which makes the gathered information available to a supervisor in order to evaluate the security and safety of the system. The first one happens when the SIEM helps the security team to identify attacks and resolve any problems, while safety occurs when supervisors can check that sensor values are not too far out of range.

The ease of representation and analysis of the information collected makes the SIEM an **effective and functional tool** for controlling devices that are used within a more complex system and any attacks that may be performed to damage the system. It allows the end-user to analyse and observe the flow of network packets and all events that are detected by the sensors. These functions allow security teams to take effective countermeasures or actions by also evaluating the history of network packets and data collected. Another advantage in terms of data collection is the MQTT protocol, which is lightweight and easy to implement, so it can be easily applied in any environment.

The SIEM could be able to help a system that is also composed of IoT devices. It would also be able to:

- *evaluate system vulnerabilities* that occur due to the sensors, and alert if occur unwanted intrusions into the system
- monitor continuously and *autonomously* not only the collected values but also the status of the sensors, i.e. unexpected intrusions or possible malfunctions
- send *alerts automatically*, like emails, using rules, for example, based on sensor values
- possibility to connect the SIEM to *cloud platforms* such as Amazon Web Services (AWS), Google Cloud, or Microsoft Azure.

The combination of IoT devices and SIEM is **effective** and **easy to implement**, thanks to the use of the MQTT data protocol. This provides the end-user with a tool that allows them to easily evaluate the actions taking place within the network and the information collected by the sensors. They allow security teams

to respond and take action effectively and quickly. In addition, they can detect and resolve vulnerabilities that may appear within a complex system relating to security, authentication and authorisation. Thanks to the low cost of implementation, and the ease and intuitiveness of deployment, this combination can also be easily used by end-users without high budgets, and in any field, becoming a tool accessible to anyone.

A Appendix

A.1 Simulator Code

```
import time                             #import time to sleep process
from datetime import datetime           #import to build timestamp
from datetime import timedelta
import json                             #import to parse json object and strings
import numpy.random as rnd              #numbers generation
import paho.mqtt.client as mqtt         #import MQTT client
from optparse import OptionParser       #parse command line arguments

# Class which allows to simulate greenhouse's sensor
class Simulator:
    # plantations simulated
    plantations = ["tomatoes", "gerbera"]

    # range sensors' value
    __range_temp = (18, 26)
    __range_hours_light = (12, 18)
    __range_hum = (65, 70)
    __range_co2 = [(800, 1000), (500, 800)]
    __range_ec = (500, 2000)
    __range_soil = (50,60)
    __range_ph = (5.5, 6)

    # sensors send information every 10 minutes
    __minutes_sensor_data = 10
    # - 6 simulations in one hour
    __simulation_in_hour = 60 / __minutes_sensor_data
    # hours in one day: 24
    __hours_one_day = 24
    # value which represents simulations in 1 day
    one_day_simulations = int(__simulation_in_hour * __hours_one_day)

    # Initialization of sensors and warnings
    def __init__(self, plantation, day, month, year, warning_temp, warning_light,
    ↪  warning_hum, warning_co2, warning_ec, warning_soil, warning_ph, seed=None):
        if not (plantation in self.plantations):
            raise Exception("Wrong plantation")
        self.__idx_plantation = self.plantations.index(plantation)
        self.__warning_temp = warning_temp
        self.__warning_light = warning_light
        self.__warning_hum = warning_hum
        self.__warning_co2 = warning_co2
        self.__warning_ec = warning_ec
        self.__warning_soil = warning_soil
        self.__warning_ph = warning_ph
        self.__count_day_simulation = 0                 # count day simulations
        self.__timestamp = datetime(year, month, day)   # timestamp sensor data
        self.__minutes_light_on = datetime.min          # count minutes light ON
        self.__minutes_light_off = datetime.min         # count minutes light OFF
        rnd.seed(seed)                                  # set seed to repeat tests
```

```
49          self.__set_starting_values()                    # method which set starting value
            ↪  of sensors
50
51      # Method which build day photoperiod of greenhouse
52      def __photoperiod_builder(self):
53          # Method which allows to rotate a list
54          def rotate(l, n):
55              return l[n:] + l[:n]
56
57          # create range light on based on warning
58          range_hours_light = self.__range_hours_light
59          if self.__warning_light:
60              range_hours_light = (6, 11) if rnd.choice(a=[False, True]) else (19, 21)
61          # total of light on
62          tot_light_on = rnd.randint(range_hours_light[0], range_hours_light[1])
63
64          # light off -> 24 hours - tot_light_on
65          hours_light_off = self.__hours_one_day - tot_light_on;
66          # two period of light on
67          hours_light_on_step1 = rnd.randint(3, tot_light_on - 2)
68          hours_light_on_step2 = tot_light_on - hours_light_on_step1
69          # two period of light off
70          hours_light_off_step1 = rnd.randint(2, hours_light_off)
71          hours_light_off_step2 = hours_light_off - hours_light_off_step1
72          # choose if light on or off in first hours of day
73          start_light_on = rnd.choice(a=[False, True])
74          # build photoperiod with tuples which represent
75          # hours and light state
76          photoperiod = [
77              (hours_light_on_step1, True),
78              (hours_light_off_step1, False),
79              (hours_light_on_step2, True),
80              (hours_light_off_step2, False)
81          ]
82          # sort photoperiod depending on starting choice
83          if not start_light_on:
84              photoperiod = rotate(photoperiod, 1)
85          return tot_light_on, start_light_on, photoperiod
86
87      # Generate random starting value
88      def __set_starting_values(self):
89          # temperature
90          self.__act_temp = round(rnd.uniform(self.__range_temp[0], self.__range_temp[1]),
            ↪  1)
91          # light period
92          self.__tot_light_on, self.__act_light, self.__photoperiod =
            ↪  self.__photoperiod_builder()
93          # air humidity
94          self.__act_hum = round(rnd.uniform(self.__range_hum[0], self.__range_hum[1]), 2)
95          # co2 level
96          self.__act_co2 = rnd.randint(self.__range_co2[self.__idx_plantation][0],
            ↪  self.__range_co2[self.__idx_plantation][1])
97          # ec
98          self.__act_ec = rnd.randint(self.__range_ec[0], self.__range_ec[1])
```

```
99          # soil moisture
100         self.__act_soil = round(rnd.uniform(self.__range_soil[0], self.__range_soil[1]),
            ↪ 2)
101         # pH value
102         self.__act_ph = round(rnd.uniform(self.__range_ph[0], self.__range_ph[1]), 2)
103
104     # Update value using Gaussian distribution
105     def __gauss_update(self, value, range, mu=0, sigma=0.1, r=2, check_range=True):
106         # compute gaussian value
107         var_value = round(rnd.normal(mu, sigma), r)
108         # update value
109         new_value = round(value + var_value, r)
110         # check value within range
111         if check_range and (new_value > range[1] or new_value < range[0]):
112             return round(value - var_value, r)
113         return new_value
114
115     # Method which simulates sensors
116     def simulate(self):
117         # Method which computes light state using photoperiod
118         def compute_light_state(hours_simulation):
119             hours_spent = 0
120             for i in range(len(self.__photoperiod)):
121                 hours_spent += self.__photoperiod[i][0]
122                 if(hours_simulation <= hours_spent):
123                     return self.__photoperiod[i][1]
124             raise Exception("Wrong photoperiod")
125
126         # Method which calculates daily hours spent in the simulation
127         def compute_hours_simulation(count_day_simulation):
128             return int(count_day_simulation * (60 / self.__simulation_in_hour) / 60)
129
130         # Method which computes minutes
131         def compute_minutes(date_minutes):
132             return date_minutes.hour * 60 + date_minutes.minute
133
134         # if 24 hours have passed, update the photoperiod
135         hours_simulation = compute_hours_simulation(self.__count_day_simulation)
136         if(self.__count_day_simulation > self.one_day_simulations):
137             self.__tot_light_on, self.__act_light, self.__photoperiod =
                ↪ self.__photoperiod_builder()
138             self.__minutes_light_on = self.__minutes_light_on.replace(hour=0, minute=0)
139             self.__minutes_light_off = self.__minutes_light_off.replace(hour=0, minute=0)
140             self.__count_day_simulation = 0
141
142         # store sensors value
143         timestamp, temperature, light_state, humidity, CO2, EC, soil_moisture, pH =
            ↪ self.__timestamp, self.__act_temp, compute_light_state(hours_simulation),
            ↪ self.__act_hum, self.__act_co2, self.__act_ec, self.__act_soil, self.__act_ph
144         minutes_light_on = compute_minutes(self.__minutes_light_on)
145         minutes_light_off = compute_minutes(self.__minutes_light_off)
146
147         # update sensors value
```

```
148             self.__act_temp = self.__gauss_update(self.__act_temp, self.__range_temp,
                ↪  sigma=0.25, r=1, check_range=(not self.__warning_temp))
149             self.__act_hum = self.__gauss_update(self.__act_hum, self.__range_hum, sigma=0.2,
                ↪  r=2, check_range=(not self.__warning_hum))
150             self.__act_soil = self.__gauss_update(self.__act_soil, self.__range_soil,
                ↪  sigma=0.2, r=2, check_range=(not self.__warning_soil))
151             self.__act_ph = self.__gauss_update(self.__act_ph, self.__range_ph, sigma=0.03,
                ↪  r=1, check_range=(not self.__warning_ph))
152             self.__act_co2 = self.__gauss_update(self.__act_co2,
                ↪  self.__range_co2[self.__idx_plantation], sigma=5, r=0, check_range=(not
                ↪  self.__warning_co2))
153             self.__act_ec = self.__gauss_update(self.__act_ec, self.__range_ec, sigma=15,
                ↪  r=0, check_range=(not self.__warning_ec))
154
155             #update timestamp
156             self.__timestamp = self.__timestamp +
                ↪  timedelta(minutes=self.__minutes_sensor_data)
157             self.__minutes_light_on = self.__minutes_light_on +
                ↪  timedelta(minutes=(self.__minutes_sensor_data if light_state else 0))
158             self.__minutes_light_off = self.__minutes_light_off +
                ↪  timedelta(minutes=(self.__minutes_sensor_data if not light_state else 0))
159
160             # update counter of simulation
161             self.__count_day_simulation += 1
162
163             return timestamp, temperature, light_state, minutes_light_on, minutes_light_off,
                ↪  humidity, CO2, EC, soil_moisture, pH
164
165     class SensorSimulation():
166
167         def __init__(self, plantation, client_id, topic, day, month, year, temp, light, hum,
            ↪  co2, ec, soil, ph, seed=None):
168             if not (plantation in Simulator.plantations):
169                 raise Exception("Wrong plantation")
170             self.__plantation = plantation
171             self.__client_id = client_id
172             self.__topic = topic
173             # create simulator
174             print("Creating", plantation, "simulator...")
175             self.__simulator = Simulator(plantation, day, month, year, temp, light, hum, co2,
                ↪  ec, soil, ph, seed=seed)
176
177         def simulate(self, num_simulation=Simulator.one_day_simulations):
178
179             # given parameters the function builds JSON object
180             def __json_builder(timestamp, temperature, light_state, minutes_light_on,
                ↪  minutes_light_off, humidity, CO2, EC, soil_moisture, pH):
181                 # check parameter's value
182                 if(EC < 0):
183                     raise ValueError('EC: the value must be positive')
184                 if(CO2 < 0):
185                     raise ValueError('CO2: the value must be positive')
186                 if(pH < 0 or pH > 14):
187                     raise ValueError('pH: invalid value')
```

```
188             if(soil_moisture < 0 or soil_moisture > 100):
189                 raise ValueError('soil_moisture: the value must be a percentage')
190             if(minutes_light_on < 0 or minutes_light_on > 1440):
191                 raise ValueError('minutes_light_on: the value must be between 0 and
                    ↪ 1440')
192             if(minutes_light_off < 0 or minutes_light_off > 1440):
193                 raise ValueError('minutes_light_off: the value must be between 0 and
                    ↪ 1440')
194             # build and return JSON object
195             json_obj = {
196                             "@timestamp": timestamp.isoformat(),    # timestamp of sensor
                                ↪ data
197                             "temperature": temperature,             # temperature in
                                ↪ Celsius
198                             "lightState": light_state,              # light OFF or ON
199                             "minutesLightON": minutes_light_on,     # minutes light ON
200                             "minutesLightOFF": minutes_light_off,   # minutes light OFF
201                             "humidity": humidity,                   # greenhouse humidity
202                             "CO2": CO2,                             # CO2 level
203                             "EC": EC,                               # EC value
204                             "soil": soil_moisture,                  # percentage of soil
                                ↪ moisture
205                             "pH": pH                                # pH value
206                         }
207             return json_obj
208
209         # Callback function which allows to print received messages of a topic
210         # It is used to check if messages have been correctly sent
211         def __on_message(client, userdata, message):
212             print("Received message:")
213             print("\tMessage:", str(message.payload.decode("utf-8")))
214             print("\tTopic:", message.topic)
215
216         # create client connection
217         print("Creating", self.__plantation, "client connection...")
218         client = mqtt.Client(client_id=self.__client_id)
219
220         # set callback which allows to received messages of a topic
221         client.on_message = __on_message
222
223         # connect to MQTT broker
224         print("Connecting to MQTT broker...")
225         client.connect(broker_address)
226
227         # enable reading and writing data
228         client.loop_start()
229
230         # start to send messages
231         # It stops the simulation using num_simulation
232         for i in range(num_simulation):
233
234             # subscribe topic to read messages
235             print("Subscribing to topic:", self.__topic)
236             client.subscribe(self.__topic)
```

```
237
238             # publish messages to topic
239             print("Publishing message to topic:", self.__topic)
240             # simmulate sensor value and build JSON object
241             timestamp, temperature, light_state, minutes_light_on, minutes_light_off,
                ↪  humidity, co2, ec, soil_moisture, ph = self.__simulator.simulate()
242             json_obj = __json_builder(timestamp, temperature, light_state,
                ↪  minutes_light_on, minutes_light_off, humidity, co2, ec, soil_moisture,
                ↪  ph)
243             # publish message
244             client.publish(self.__topic, json.dumps(json_obj), qos=qos)
245             # wait to simulate
246             time.sleep(sleep_time)
247
248         # stop the loop
249         client.loop_stop()
250         client.disconnect()
251
252 # Class which allows to parse command line arguments
253 class CmdLineParser(object):
254     def __init__(self):
255         # command line
256         self.parser = OptionParser(usage='usage: python plantation simulator.py
            ↪  [options]')
257         # optional arguments
258         self.parser.add_option("-t", "--temp", dest="temp", action="store_true",
            ↪  default=False, help="correct (default) or wrong temperature")
259         self.parser.add_option("-l", "--light", dest="light", action="store_true",
            ↪  default=False, help="correct (default) or wrong light time")
260         self.parser.add_option("-u", "--hum", dest="hum", action="store_true",
            ↪  default=False, help="correct (default) or wrong air humidity")
261         self.parser.add_option("-c", "--co2", dest="co2", action="store_true",
            ↪  default=False, help="correct (default) or wrong CO2 level")
262         self.parser.add_option("-e", "--ec", dest="ec", action="store_true",
            ↪  default=False, help="correct (default) or wrong EC value")
263         self.parser.add_option("-m", "--soil", dest="soil", action="store_true",
            ↪  default=False, help="correct (default) or wrong soil moisture")
264         self.parser.add_option("-p", "--ph", dest="ph", action="store_true",
            ↪  default=False, help="correct (default) or wrong pH value")
265         #number of simulations
266         self.parser.add_option("-n", "--sim", dest="sim", action="store", type="int",
            ↪  default=Simulator.one_day_simulations, help="number of simulations")
267         #seed to generate random values
268         self.parser.add_option("-s", "--seed", dest="seed", action="store", type="int",
            ↪  default=None, help="seed value")
269         #set initial timestamp
270         self.parser.add_option("-D", "--day", dest="day", action="store", type="int",
            ↪  default=1, help="day of timestamp. Default: 1")
271         self.parser.add_option("-M", "--month", dest="month", action="store", type="int",
            ↪  default=12, help="month of timestamp. Default: 12")
272         self.parser.add_option("-Y", "--year", dest="year", action="store", type="int",
            ↪  default=2021, help="year of timestamp. Default: 2021")
273
274     def addOption(self, *args, **kwargs):
```

```
275         self.parser.add_option(*args, **kwargs)
276 
277     def parseArgs(self):
278         (options, args) = self.parser.parse_args()
279         options.plantation = args[0]
280         return options
281 
282 # address of MQTT Broker
283 broker_address = "192.168.10.50"
284 # MQTT QoS value
285 qos = 1
286 # topic's prefix of greenhouse project
287 topic_prefix  = "/greenhouse"
288 # topic of tomatoes greenhouse
289 topic_tomatoes  = topic_prefix + "/tomatoes"
290 # topic of Gerbera greenhouse
291 topic_gerbera  = topic_prefix + "/gerbera"
292 # client id of tomatoes greenhouse
293 client_id_tomatoes = "TomGreenhouse";
294 # client id of Gerbera greenhouse
295 client_id_gerbera = "GerGreenhouse";
296 # sleep time to next sensor measurements
297 sleep_time = 1
298 
299 def main():
300 
301     # parse and read arguments
302     parser = CmdLineParser()
303     opt = parser.parseArgs()
304 
305     if not (opt.plantation in Simulator.plantations):
306         raise Exception("Wrong plantation")
307 
308     # based on plantation set client and topic
309     plantation = opt.plantation.lower()
310     if plantation == "tomatoes".lower():
311         client_id, topic = client_id_tomatoes, topic_tomatoes
312     if plantation == "gerbera".lower():
313         client_id, topic = client_id_gerbera, topic_gerbera
314 
315     # enable simulation
316     simulation = SensorSimulation(opt.plantation.lower(), client_id, topic, opt.day,
    ↪  opt.month, opt.year, opt.temp, opt.light, opt.hum, opt.co2, opt.ec, opt.soil,
    ↪  opt.ph, opt.seed)
317     # start simulation
318     simulation.simulate(opt.sim)
319 
320 main()
```

A.2 Hosts Configuration

A.2.1 Sensors Simulator

A virtual machine is created on VirtualBox with the characteristics previously described. It is the host on which the simulation script is launched. In the host ***LUbuntu 21.10*** is installed. Python 3 is automatically installed with OS installation.

Once the general configuration of the host is completed, Python libraries are installed and will be used to run the script which simulates the sensors of the hydroponic greenhouse with different plantations:

1. ***Numpy***: provides math functions, random number generation and more.

```
pip3 install numpy
```

2. ***OptParse***: allows performing an efficient and fast parse of the script's command line arguments.

```
pip3 install optparse-pretty
```

3. ***Paho-MQTT***: allows sending MQTT messages using a Python script. It is described in the previous chapters.

```
pip3 install paho-mqtt
```

The simulation script is saved in any position of the machine and it is launched through the terminal, simulating various plantations.

Moreover, IP address *192.168.10.70* is set to the network card connected to *greenhousenet* by editing the file:

```
/etc/netplan/00-installer-config.yaml
```

and is applied using the following command.

```
sudo netplan apply
```

A.2.2 Host Mosquitto - Logstash

In the second host Mosquitto, Logstash and Beats are installed.
First, create a virtual machine with **Ubuntu Server 21.10** and set the IP address to *192.168.10.50*.

Then install the *Java Runtime Environment (JRE)* to run Elastic programs:

```
sudo apt-get install default-jre
```

Once Java is installed, add *FTP service* to the server:

```
sudo apt-get install vsftp
```

Then, import the PGP key - *Elasticsearch Signing Key* which allows to download the Elastic software:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
```

Then install *apt-transport-https* which allows to download https repositories and add the repository which allows to download Elasticsearch, Kibana, Logstash, and Beats:

```
sudo apt-get install apt-transport-https

echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" |
        sudo tee /etc/apt/sources.list.d/elastic-7.x.list
```

Set the IP address to *192.168.10.50*.

A.2.2.1 Mosquitto

An MQTT Broker, Mosquitto, is installed on the host. To install **Mosquitto** can be used the default installation provided by *Ubuntu Server* or the instructions of the *APT package manager*:

```
sudo apt-get update && sudo apt install mosquitto mosquitto-clients
```

Once it is installed, create configuration file `/etc/mosquitto/conf.d/mosquitto.conf` to open port 1883:

```
listener 1883        # mosquitto port
allow_anonymous true
```

Then, activate the service.

```
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

A.2.2.2 Logstash

Logstash is a module of Elastic Stack. After adding the repositories, use the following APT command to install it.

```
sudo apt-get update && sudo apt-get install logstash
```

Once it is installed, proceed with the configuration. Some configuration files are created to customize the data processing pipeline on MQTT messages [35].

1. Set up the Logstash input plugin from which the data are taken. The events' information is collected by *Elastic Beats*. The path of configuration file is `/etc/logstash/conf.d/beats-input.conf`.

```
input {
   beats {
      port => 5044
   }
}
```

2. Filter gathered data by Logstash, collecting JSON messages.
   The path used is `/etc/logstash/conf.d/json-filter.conf`.

```
filter {
   json {
      source => "message"
   }
}
```

3. Logstash sends data to the host where Elasticsearch is installed and create the new index `filebeat-sensor-data`.
   The path of configuration file is `/etc/logstash/conf.d/elasticsearch-output.conf`.

```
output {
   if [@metadata][pipeline] {
      elasticsearch {
         hosts => ["192.168.10.240:9200"]
         manage_template => false
         index => "%{[@metadata][beat]}-sensor-data-%{+YYYY.MM.dd}"
         pipeline => "%{[@metadata][pipeline]}"
      }
   } else {
      elasticsearch {
         hosts => ["192.168.10.240:9200"]
         manage_template => false
         index => "%{[@metadata][beat]}-sensor-data-%{+YYYY.MM.dd}"
      }
   }
}
```

In order to test whether the changes made to the configuration files are correct, can be used the following command:

```
sudo -u logstash /usr/share/logstash/bin/logstash --path.settings /etc/logstash -t
```

After the changes are made, the Logstash service is launched, and set to run automatically when the host starts up:

```
sudo systemctl enable logstash
sudo systemctl start logstash
```

and can be stopped using the `stop` command.

A.2.2.3 Filebeat

Filebeat allows to collect and centralize data and logs from devices and it works as an ***agent*** installed on devices.
Thanks to the imported repository, the Filebeat installation [34] command is:

```
sudo apt-get update && sudo apt-get install filebeat
```

As in the case of Logstash, the next step is the configuration. The file at the following path is edited:

```
/etc/filebeat/filebeat.yml
```

1. Filebeat sends data to Logstash, which will forward them to Elasticsearch. Comment `output.elasticsearch` to not forward data directly to Elasticsearch and set the IP address of the Logstash where the data are sent.

```
# output.elasticsearch:
#    hosts: ["192.168.10.240:9200"]
output.logstash:
    hosts: ["192.168.10.50:5044"]
```

2. Enable the MQTT type and specify the IP address (localhost) of the MQTT Broker [81]. The topic used to filter messages is `greenhouse/#`

```
filebeat.inputs:
- type: mqtt
   hosts:
      - tcp://127.0.0.1:1883
   topics:
      - /greenhouse/#
   tags: ["json"]
   enabled: true
```

In order to show the modules which *Filebeat* provides:

```
filebeat modules list
```

Activate the module `system` to collect data and logs:

```
filebeat modules enable system
```

Then, load the Elasticsearch index model which represents the index settings, and export the model:

```
filebeat setup --index-management -E output.logstash.enabled=false
     -E 'output.elasticsearch.hosts=["192.168.10.240:9200"]'

   filebeat export template > filebeat.template.json
```

Upload the template file to the server where Elasticsearch is installed and load the template using `curl`:

```
curl -XPUT -H 'Content-Type: application/json'
http://localhost:9200/_index_template/filebeat-7.16.3
-d@filebeat.template.json
```

Then activate the Filebeat service.

```
sudo systemctl enable filebeat
sudo systemctl start filebeat
```

A.2.2.4 Packetbeat

Packetbeat sniffs network traffic and allows to monitor applications and performance and it works as an agent installed on devices. The Packetbeat installation [78] command is:

```
sudo apt-get update && sudo apt-get install packetbeat
```

The next step is the configuration editing the file at the following path:

```
/etc/packetbeat/packetbeat.yml
```

1. Enable protocol MQTT to sniff the packets. Other protocols are already activated, such as ICMP, HTTP, and other.

```
packetbeat.protocols:
    - type: mqtt
      ports: [1883]
```

2. Set Kibana host:

```
setup.kibana:
    host: [192.168.10.240:5601]
```

3. Set the output of the collected data to Elasticsearch. The data does not need to be parsed by passing them through Logstash.

```
output.elasticsearch:
    hosts: [192.168.10.240:9200]
```

Then, to test the configuration can be run the following commands:

```
sudo packetbeat test config
sudo packetbeat test output
```

If everything is correct, setup Packetbeat:

```
sudo packetbeat setup
```

Then activate the Packetbeat service.

```
sudo systemctl enable packetbeat
sudo systemctl start packetbeat
```

A.2.3 Host Elastic SIEM

The third virtual host works as Elastic SIEM and displays information collected by the sensors of the greenhouse.
Create the virtual machine and install **Ubuntu Server 21.10**, *Java Runtime Environment (JRE)*, and *FTP*. Then, install *apt-transport-https*, import *Elasticsearch Signing Key*, and add Elastic repository as for *Host Mosquitto - Logstash*. Set the IP address to *192.168.10.240*.

A.2.3.1 Packetbeat

Following the instructions in paragraph A.2.2.4 install **Packetbeat**. The only change to do in the configuration file is the addition of the Kibana port to the HTTP protocol, in this way, the network packets concerning port *5601* can be collected.

```
packetbeat.protocols:
    - type: http
        ports: [80, 8080, 8000, 5000, 8002, 5601]
```

A.2.3.2 Elasticsearch

Proceed with the installation of **Elasticsearch** [24]. Update APT manager and install Elasticsearch.

```
sudo apt-get update && sudo apt-get install elasticsearch
```

After software installation, proceed with the configuration [18] by editing the file located at the path:

```
/etc/elasticsearch/elasticsearch.yml
```

1. Set a meaningful name to the host.

```
node.name: "node-siem"
```

2. By default *localhost* is the only way to access Elasticsearch. The value of this field is changed to enable access from any host.

```
network.host: 0.0.0.0
```

3. The default port to connect to Elasticsearch is 9200.

```
http.port: 9200
```

4. Elasticsearch can scan ports automatically to create an *auto-clustering*. In the project, there is only one node where Elasticsearch is installed: *localhost*.

```
discovery.seed_hosts: ["127.0.0.1"]
```

5. Set up the master node.

```
cluster.initial_master_nodes: ["node-siem"]
```

Once the changes have been made, the Elasticsearch service is launched and it activates automatically when the computer is switched on:

```
sudo systemctl daemon-reload
sudo systemctl enable elasticsearch
sudo systemctl start elasticsearch
```

The service can be stopped using the following command.

```
sudo systemctl stop elasticsearch
```

A.2.3.3 Kibana

After Elasticsearch is installed, **Kibana** is installed [25] using the following commands:

```
sudo apt-get update && sudo apt-get install kibana
```

Once Kibana is installed, proceed by configuring the file at the path: [17]

```
/etc/kibana/kibana.yml
```

1. The default port to connect to Kibana is 5601.

```
server.port: 5601
```

2. Enable Kibana visible from any device.

```
server.host: 0.0.0.0
```

3. Set up Elasticsearch host which is the same host where Kibana is installed: *localhost*.

```
elasticsearch.hosts: ["http://localhost:9200"]
```

Similarly to Elasticsearch, to activate Kibana:

```
sudo systemctl daemon-reload
sudo systemctl enable kibana
sudo systemctl start kibana
```

The service can be stopped:

```
sudo systemctl stop kibana
```

A.2.3.3.1 Dashboard

Once the 3 hosts have been configured with the Elastic modules, proceed to configure the dashboard which is visible to the supervisor user.

After activating the various components (*Mosquitto*, *Kibana*, *Elasticsearch*, *Logstash*, *Filebeat*, and *Packetbeat*) with the correct configuration of the Elasticsearch pipeline, run a test with ping messages and the Python script which simulates the sensors. These operations allow to check whether Packetbeat works and Filebeat extracts the information from the MQTT Broker, and sends it to Logstash which then forwards it to Elasticsearch.

If everything works correctly, the Kibana dashboard is configured. In the project, the host which acts as supervisor is the host on which the virtual hosts run; by configuring the network cards of the virtual hosts set in *Bridged mode* with an IP address which can be pinged by the main host, Kibana is visible from a browser using an address similar to the following:

```
http://192.168.2.90:5601/app/home
```

In the main menu under the heading *Analytics*, open the *Dashboard* item. Here, you can create any type of visualisation using the fields collected in the index `packetbeat-*`, in order to display the information regarding the collected network packets.
The following graphs are used to show the data in the dashboards:

1. ***Histograms*** to show the traffic generated by hosts in the network and the ICMP, MQTT and HTTP messages sent to the most important hosts in the system;

2. ***Pie charts*** to represent the same information as the histograms.

The different data are filtered using the `destination.port` and `destination.ip`.

Then proceed to configure dashboards to show sensors data.
First, open the menu which appears on the left, under *Stack Management*. Then open the *Index Patterns* item and create a **new index pattern** which allows to aggregate all data which are forwarded through Logstash with the index defined in the configuration: `filebeat-sensor-data-{+YYYY-MM-DD}`. In order to configure the pattern is used the `*` symbol as a wildcard to group all information regardless of date:

```
filebeat-sensor-data-*
```

Once the index pattern has been created, the fields representing the sensor measurements, the MQTT topic, and other information relating to the data collected can be observed within it.

After the index pattern has been configured, construct the customized dashboards. In the main menu under the heading *Analytics*, open the *Dashboard* item. Here, can create any type of visualization using the fields collected in the index previously configured.
The dashboards that are created and made available to the supervisor are mainly two:

1. **Greenhouse Tomatoes**: represents the information of a greenhouse where *tomatoes* are cultivated.
2. **Greenhouse Gerbera**: in which the information about the environment where the *Gerberas* are grown is displayed.

In the two dashboards, the data are represented with the same types of graphs, but the information is different.
The created dashboards allow to highlight the data and use the following structure:

1. ***Table*** with all measured data sorted by the hour and day. The values are colored to allow the supervisor a clearer view, especially if the values are outside the correct ranges.
2. ***Line graphs*** showing *temperature*, CO_2, *electrical conductivity*, *humidity*, *light status* indicating minutes on or off, and *soil moisture*.
3. ***Display*** of current measured values.

Filebeat is configured to collect and forward all MQTT messages that are related to the topic `/greenhouse/#`. The data that are collected in Elasticsearch refers to any type of planting in the greenhouse. To filter the data in the dashboards, the `mqtt.topic.keyword` field of the index `filebeat-sensor-data-*` is used. It allows filtering *MQTT messages* through the topic used.
In this way, the data that are displayed represents the JSON related to the specific topic, i.e. `/greenhouse/tomatoes` or `/greenhouse/gerbera`.

Once the dashboards have been created they are saved by activating the flag that automatically updates the displayed values.

References

[1] Python Package Index (PyPi). *paho-mqtt 1.6.1*. https://pypi.org/project/paho-mqtt/. Oct. 2021.

[2] AGrowTronics. *Advanced Monitoring in Hydroponics*. https://www.agrowtronics.com/advanced-monitoring-in-hydroponics/.

[3] AWS Amazon. *ELK Stack*. https://aws.amazon.com/opensearch-service/the-elk-stack/.

[4] ArcSight. *Security Operations Metrics Definitions for Management and Operations Teams*. http://docs.media.bitpipe.com/io_10x/io_100356/item_415627/ArcSight%20Whitepaper-%20Security%20Operations%20Metrics%20Definitions%20for%20Management%20and%20Operations%20Teams.pdf.

[5] Atharva Shewale, Dr. Shwetambari Chiwhane. "Agriculture Using Mqtt Protocol". In: *International Journal of Future Generation Communication and Networking* 13.3s (2020), pp. 1628–1633.

[6] Daniel Berman. *Using the ELK Stack for SIEM*. https://logz.io/blog/elk-siem/. June 2018.

[7] Biswajeeben Mishra and Attila Kersetz. "Stress-Testing MQTT brokers: A Comparative Analysis of Performance Measurements". In: *MDPI* ().

[8] Ziv Chang and Shin Li. *The IoT Attack Surface: Threats and Security Solutions*. https://www.trendmicro.com/vinfo/mx/security/news/internet-of-things/the-iot-attack-surface-threats-and-security-solutions#:~:text=IoT%20systems%20are%20also%20susceptible,can%20lead%20to%20compromised%20systems. May 2019.

[9] Charles P. Pfleeger. "Security in Computing". In: 2015. Chap. 1. ISBN: 9780134085043.

[10] Laura Chartier. *Hydroponics & Nutrient Application*. https://gpnmag.com/article/hydroponics-nutrient-application/. May 2015.

[11] Anton Chuvakin. *On SIEM Tool and Operation Metrics*. https://blogs.gartner.com/anton-chuvakin/2014/06/17/on-siem-tool-and-operation-metrics/. June 2014.

[12] Dipen J. Vyas, Nilesh N. Rudani. "MQTT & IOT Based Control and Monitoring of Smart Green House". In: *International journal of Engineering development and research* 6 (2018).

[13] Onur Dündar. *MQTT — Part II: Smart City Design with MQTT*. https://medium.com/@onur.dundar1/mqtt-part-ii-smart-city-design-over-mqtt-da41018a45c2. Aug. 2018.

[14] Elastic. *2021 Gartner Magic Quadrant for SIEM*. https://www.elastic.co/campaigns/2021-gartner-magic-quadrant-siem?elektra=products-siem&storm=cta1&rogue=SIEM-GIC.

[15] Elastic. *Auditbeat overview*. https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-overview.html.

[16] Elastic. *Beats: lightweight data shippers*. https://www.elastic.co/beats/.

[17] Elastic. *Configure Kibana*. https://www.elastic.co/guide/en/kibana/current/settings.html.

[18] Elastic. *Configuring Elasticsearch*. https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html.

[19] Elastic. *Elasticsearch: the heart of the free and open Elastic Stack*. https://www.elastic.co/elasticsearch/.

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

[21] Elastic. *Functionbeat overview*. https://www.elastic.co/guide/en/beats/functionbeat/current/functionbeat-overview.html.

[22] Elastic. *Heartbeat overview*. https://www.elastic.co/guide/en/beats/heartbeat/current/heartbeat-overview.html.

[23] Elastic. *Important Elasticsearch configuration*. https://www.elastic.co/guide/en/elasticsearch/reference/current/important-settings.html.

[24] Elastic. *Install Elasticsearch with Debian Package*. https://www.elastic.co/guide/en/elasticsearch/reference/current/deb.html.

[25] Elastic. *Install Kibana with Debian package*. https://www.elastic.co/guide/en/kibana/current/deb.html.

[26] Elastic. *Installing Logstash*. https://www.elastic.co/guide/en/logstash/current/installing-logstash.html.

[27] Elastic. *Kibana: your window into the Elastic Stack*. https://www.elastic.co/kibana/.

[28] Elastic. *Logstash: centralize, transform and stash your data*. https://www.elastic.co/logstash/.

[29] Elastic. *Metricbeat overview*. https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html.

[30] Elastic. *Metricbeat quick start: installation and configuration*. https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-installation-configuration.html.

[31] Elastic. *MQTT input*. https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-input-mqtt.html.

[32] Elastic. *Packetbeat overview*. https://www.elastic.co/guide/en/beats/packetbeat/current/packetbeat-overview.html.

[33] Elastic. *Packetbeat quick start: installation and configuratione*. https://www.elastic.co/guide/en/beats/packetbeat/current/packetbeat-installation-configuration.html.

[34] Elastic. *Repositories for APT and YUM*. https://www.elastic.co/guide/en/beats/filebeat/6.8/setup-repositories.html.

[35] Elastic. *Setting Up and Running Logstash*. https://www.elastic.co/guide/en/logstash/current/setup-logstash.html.

[36] Elastic. *SIEM for the modern SOC*. https://www.elastic.co/siem/.

[37] Elastic. *What is the ELK Stack?* https://www.elastic.co/what-is/elk-stack.

[38] Elastic. *Winlogbeat overview*. https://www.elastic.co/guide/en/beats/winlogbeat/current/_winlogbeat_overview.html.

[39] From Wikipedia the free encyclopedia. *Internet of things*. https://en.wikipedia.org/wiki/Internet_of_things. Dec. 2021.

[40] From Wikipedia the free encyclopedia. *MQTT*. https://en.wikipedia.org/wiki/MQTT. Nov. 2021.

[41] From Wikipedia the free encyclopedia. *Security Information and Event Management*. https://it.wikipedia.org/wiki/Security_Information_and_Event_Management. July 2021.

[42] Exabeam. *SIEM Architecture: Technology, Process and Data*. https://www.exabeam.com/siem-guide/siem-architecture/.

[43] Exabeam. *SIEM Tools: Top 6 SIEM Platforms, Features, Use Cases and TCO*. https://www.exabeam.com/explainers/siem/siem-buyers-guide/.

[44] Alberto Ferretti. *Che cos'è un SIEM (Security Information and Event Management)?* https://www.n4b.it/sicurezza-it/sicurezza-che-cose-un-siem-security-information-and-event-management/. Oct. 2020.

[45] Fikret Yalcinkaya, Hüseyin AYDİLEK, Mustafa Yasin Erten, Nihat İNANÇ. "IoT based Smart Home Testbed using MQTT Communication Protocol". In: *International Journal of Engineering Research and Development* 12 (Jan. 2020), pp. 317–324.

[46] Gartner. *Security Information and Event Management (SIEM) Reviews and Ratings*. https://www.gartner.com/reviews/market/security-information-event-management.

[47] Erin Glass. *How To Install Elasticsearch, Logstash, and Kibana (Elastic Stack) on Ubuntu 20.04*. https://www.digitalocean.com/community/tutorials/how-to-install-elasticsearch-logstash-and-kibana-elastic-stack-on-ubuntu-20-04. June 2020.

[48] Carsten Gregersen. *A Complete Guide to IoT Protocols & Standards In 2021*. https://www.nabto.com/guide-iot-protocols-standards/. Dec. 2020.

[49] Thomas Hazel. *5 ELK Stack Pros and Cons*. https://www.chaossearch.io/blog/elk-stack-pros-and-cons. Jan. 2021.

[50] HiveMQ. *Modernizing the Smart Manufacturing Industry with MQTT*. https://www.hivemq.com/solutions/manufacturing/modernizing-the-manufacturing-industry/.

[51] HMS. *MQTT used in production - a use case*. https://www.anybus.com/docs/librariesprovider7/default-document-library/whitepapers/mqtt-used-in-production---a-case-study.pdf. 2019.

[52] Dotan Horovits. *The Complete Guide to the ELK Stack*. https://logz.io/learn/complete-guide-elk-stack/. June 2020.

[53] Martin Hron. *Are smart homes vulnerable to hacking?* https://blog.avast.com/mqtt-vulnerabilities-hacking-smart-homes. Aug. 2018.

[54] IBM. *Telemetry use case: Home patient monitoring*. https://www.ibm.com/docs/en/ibm-mq/8.0?topic=cases-telemetry-use-case-home-patient-monitoring. Dec. 2021.

[55] Imperva. *Security Information and Event Management*. https://www.imperva.com/learn/application-security/siem/.

[56] Appleton Innovations. *Smart Healthcare monitoring System using MQTT protocol*. https://appletoninnovations.medium.com/smart-healthcare-monitoring-system-using-mqtt-protocol-a2818c469f5d. Aug. 2021.

[57] Kali. *hping3 Usage Example*. https://www.kali.org/tools/hping3/.

[58] Tim Keary. *10 Best SIEM Tools for 2021: Vendors & Solutions Ranked*. https://www.comparitech.com/net-admin/siem-tools. Dec. 2021.

[59] Eric Labbate. *FINALLY, KNOW AND UNDERSTAND WHAT YOUR CROP KNOWS!* https://www.climatecontrol.com/blog/greenhouse-soil-ec-moisture-sensors/. Aug. 2018.

[60] LogSign. *Guide for Security Operations Metrics*. https://www.logsign.com/uploads/Guide_for_Security_Operations_Metrics_Whitepaper_2f999f27cc.pdf.

[61] Gianmarco Marcello. *Tutorial – Utilizzare MQTT con Python*. https://antima.it/tutorial-utilizzare-mqtt-con-python-la-classe-client-parte-1/. June 2019.

[62] Monika Bharatbhai Patel, Chintan Bhatt, Hamed Vahdat-Nejad, Hardik B. Patel. "Smart city based on MQTT using wireless sensors". In: *Protocols and Applications for the Industrial Internet of Things* (Apr. 2018), pp. 240–263.

[63] Omafra. *Carbon Dioxide In Greenhouses*. http://www.omafra.gov.on.ca/english/crops/facts/00-077.htm.

[64] Oracle. *What is IoT?* https://www.oracle.com/it/internet-of-things/what-is-iot/.

[65] Mike Poe. *What is MQTT and how does it add value to logistics operations?* https://www.cognex.com/blogs/industrial-barcode-reader/what-is-mqtt-and-how-does-it-add-value-to-logistics-operations. Aug. 2021.

[66] Steve Ranger. *What is the IoT?* https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/. Feb. 2020.

[67] Samer Jaloudi. "MQTT for IoT-based Applications in Smart Cities". In: (Mar. 2019).

[68] Seacom. *I moduli di Elastic Stack*. https://www.seacom.it/elastic-stack-features/.

[69] Seacom. *I vantaggi di una soluzione SIEM basata su Elastic*. https://www.seacom.it/elastic-siem/.

[70] Andrea Sorri. *The potential for MQTT in realizing the smart city vision*. https://www.axis.com/blog/secure-insights/mqtt-smart-cities/. Dec. 2020.

[71] IT Central Station. *Best SIEM Solutions & SIEM Tools*. https://www.itcentralstation.com/categories/security-information-and-event-management-siem.

[72] IT Central Station. *Compare Exabeam Fusion SIEM vs. Securonix Security Analytics*. https://www.itcentralstation.com/products/comparisons/exabeam-fusion-siem_vs_securonix-security-analytics.

[73] The HiveMQ Team. *MQTT Topics and Best Practices - MQTT Essentials: Part 5*. https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/. Aug. 2019.

[74] Dan Tembe. *IoT and SIEM Integration. — Pt.1*. https://medium.com/@dtembe/iot-and-siem-integration-pt-1-6645a012bdc. Sept. 2017.

[75] Celeste Tholen. *What Is a Security System and How Does it Work?* https://www.safewise.com/home-security-faq/how-do-security-systems-work/. Oct. 2021.

[76] Gianluigi Torchiani. *MQTT: cos'è il protocollo e come riesce a far comunicare i dispositivi IoT*. https://www.internet4things.it/iot-library/mqtt-cose-e-come-funziona-il-protocollo-alla-base-delliot/. Mar. 2020.

[77] Vincent de Paul Niyigena Kwizera, Zhanming Li, Victus Elikplim Lumorvie, Febronie Nambajemariya, Xiaowei Niu. "IoT Based Greenhouse Real-Time Data Acquisition and Visualization through Message Queuing Telemetry Transfer (MQTT) Protocol". In: *Advances in Internet of Things* 11.2 (2021).

[78] Gayan Virajith. *Install Packetbeat in Ubuntu*. https://gist.github.com/gayanvirajith/d9682285d9146bbc5ad04ee

[79] Fan Wang. *Application of MQTT protocol in oil & gas industry*. https://www.emqx.com/en/blog/application-of-mqtt-protocol-in-oil-and-gas-industry. July 2021.

[80] Anuradha Wickramarachchi. *Home Automation With MQTT and Home Assistant*. https://medium.com/swlh/home-automation-with-mqtt-and-home-assistant-techtalk-gist-734bc89b5e53. Nov. 2020.

[81] Lasri Yassine. *Track IoT Devices with MQTT and Elastic Stack*. http://www.synapticiel.co/track-iot-devices-with-mqtt-and-elastic-stack/. Oct. 2020.

[82] Laura Zanotti. *SIEM: cos'è e come garantisce la sicurezza delle informazioni*. https://www.cybersecurity360.it/soluzioni-aziendali/siem-cos-e-come-garantisce-la-sicurezza-delle-informazioni/. Dec. 2019.