



THESIS / THÈSE

MASTER IN COMPUTER SCIENCE

Integration of a honeypot solution into a DevSecOps pipeline in an Edge Computing context

BOUHOU, Abdel-Malek

Award date:
2022

Awarding institution:
University of Namur

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

UNIVERSITY OF NAMUR
Faculty of Computer Science
Academic year 2021-2022

Integration of a honeypot solution into a DevSecOps pipeline in an Edge Computing context

BOUHOU Abdel-Malek



Internship Supervisors: Sébastien DUPONT & Guillaume GINIS

Promoter: Pr. Jean-Noël COLIN (Signature for deposit approval - REE art. 40)



Co-promoter: Christophe PONSARD

Thesis submitted for the award of the degree of
Master in Computer Science.

Acknowledgments

I would like to express my sincere thanks:

To my wife, Sophie, for her support and unconditional love on a daily basis.

To my family for their support and encouragement during those years of study.

To Professor Jean-Noël COLIN, my promoter, for having allowed me to write this thesis on this topic and for his wise advice during these two years of study.

To CETIC members and internship masters, Sébastien DUPONT and Guillaume GINIS for their support, their patience, their follow-up, during these five months of research.

To Christophe PONSARD, also a member of CETIC and Professor of the security course, for making this internship possible and for the follow-up.

To my classmates who took the time to answer my questions and encouraged me to make this work possible.

Thank you all.

Abstract

In the world of information technology and the Internet, the proliferation of connected objects, online services, and the amount of data in transit, has led to the emergence of a new paradigm called Edge computing. This concept brings IT intelligence closer to the periphery to meet the demands of these increases. Through the various layers that make up it, attacks are constant, and devices connected to the Internet are continually scanned by hackers who try to find valuable data, critical applications, and vulnerabilities in network infrastructures. A pot of honey is by vocation, an information system, more or less artificial, whose main purpose is to be probed, attacked, and possibly compromised in order to make a hacker believe that he has penetrated the organization's real network. Thanks to the honey pot, the hacker is removed from the real network and its activities and movements can be captured and monitored. The information collected about attackers allows us to study their methods, motivations, and attack techniques they use to penetrate or corrupt a system. The aim is to develop ways to improve system security and prevent future attacks. One of the areas affected by these security improvements is the software development industry. It is increasingly turning to movements such as DevOps, which improve the performance and quality of software, facilitate collaboration between development, security, and operations teams, but also to reduce the life cycle of development. One of the major problems of this life cycle is the lack of continuous integration of security. Security issues are often considered far too late in the development phases and when they appear a posteriori, they can have significant financial implications. To reduce threats and address this lack of security, DevOps is moving to DevSecOps, which adds a layer of security to the pipeline, integrating solutions throughout the development lifecycle. As part of this work, we propose the implementation of a DevSecOps architecture, in the context of Edge Computing, integrating a honey pot solution in order to respond to a threat. The concept of Edge Computing is materialized by an autonomous car representing a connected object and the threat will be manifested by the creation of a back door allowing the control of the autonomous vehicle. Following the discovery of this threat, a response is provided by the deployment of a honeypot using continuous deployment and integration tools, specific to the DevSecOps pipeline.

Table of Contents

1. Introduction.....	7
1.1. General Introduction	7
1.2. Context	8
1.3. Objectives.....	8
2. DevSecOps Pipeline.....	9
2.1. DevOps.....	9
2.2. DevSecOps	10
2.3. Honeypot in a DevSecOps Pipeline.....	11
3. Edge Computing.....	12
3.1. Honeypot and Edge Computing.....	15
4. Honeybots: State of the art	16
4.1. Definition	16
4.2. Characteristics of honeybots	19
4.2.1. Interaction Level.....	21
4.2.2. Resource level	21
4.2.3. Communication interfaces.....	22
4.2.4. Multi-tier architecture	23
4.2.5. Topology	23
4.2.6. Containment	24
4.2.7. Observability.....	25
4.2.8. Scalability	25
4.2.9. Context	26
4.3. Strength and weakness of characteristics	27
4.4. Honeybot	28
4.5. Threat management	28
4.5.1. Threat Analysis.....	28
4.5.2. Mitre	29
4.5.3. Categorization of threats	30
4.6. Honeybot solutions references	32
5. Proof of concept	34
5.1. Context	34
5.1.1. Autonomous vehicles.....	34
5.1.2. Challenges	34
5.1.3. Scenario.....	35
5.1.4. Integration of a honeybot	37
5.2. Implementation in a use case.....	38
5.2.1. Functional specification	38
5.2.2. Technical Specification	39
5.2.2.1. Technologies and constraints.....	39
5.2.2.2. Solution Architecture.....	47
5.2.3. Deployment.....	53
5.2.3.1. Install OpenShift.....	53
5.2.3.2. Falco and Response Engine deployment.....	53
5.2.3.3. Deploy the autonomous vehicle	55
5.2.3.4. Update the autonomous vehicle with backdoor	57
5.2.3.5. Deploy the honeybot vehicle.....	58
5.3. Results.....	61
5.3.1. Results	61
6. Discussion and Future works	62
7. Conclusion	65

List of Figures

Figure 1 - Phases of DevOps lifecycle.....	9
Figure 2 - DevSecOps lifecycle with sample security activities	11
Figure 3 - Industrial IoT data processing layer stack.....	13
Figure 4 - Representation of the different concepts related to Edge Computing	14
Figure 5 - Basic diagram of different environments using honeypots.....	17
Figure 6 - Diagram of the different characteristics of a honeypot – part 1	19
Figure 7 - Diagram of the different characteristics of a honeypot – part 2.....	19
Figure 8 - Diagram of a wastewater treatment plant honeypot	20
Figure 9 - Table of strengths and weaknesses of honeypots characteristics	27
Figure 10 - Threat Categorization & Frequency Statistics	30
Figure 11 - Cowrie command & MITRE categorization.....	31
Figure 12 - Attack-Defence Trees scenario	35
Figure 13 - Backdoor attack scenario	35
Figure 14 - Backdoor attack demonstration with MetaSploit	36
Figure 15 - Structure of the use case integrating a honeypot	37
Figure 16 - Infrastructure technologies	39
Figure 17 - Autonomous vehicle technology	40
Figure 18 - ROS message send by a publisher illustration diagram	41
Figure 19 - ROS message received by subscribers illustration diagram	41
Figure 20 - Security technologies.....	42
Figure 21 - Honeypot technology.....	44
Figure 22 - Red Team technologies	45
Figure 23 - Technologies architecture.....	46
Figure 24 - 4+1 Architecture diagram	47
Figure 25 - Logical view: Class diagram.....	48
Figure 26 - Process View: Sequence diagram	49
Figure 27 - Development View: Components diagram	50
Figure 28 - Physical View: Deployment diagram	51
Figure 29 - Scenario: Uses Cases view	52
Figure 30 - Falco deployment in OpenShift	53
Figure 31 – Security response illustration diagram	54
Figure 32 - Gitlab: Autonomous vehicle repository - publisher.....	55
Figure 33 - List of YAML files necessary for the deployment of the application	56
Figure 34 - Publisher publishing message.....	56
Figure 35 - Subscriber heard message.....	56
Figure 36 - Reverse shell connection from the autonomous vehicle	57
Figure 37 - Detection of reverse shell events by Falco	58
Figure 38 - OpenShift console: infected POD quarantined	58
Figure 39 -Gitlab: Infected autonomous vehicle repository – publisher	59
Figure 40 - List of YAML files necessary for the deployment of the application	59
Figure 41 - OpenShift console: Honeypot deployed.....	60
Figure 42 - Reverse shell connection from the honeypot.....	60
Figure 43 - Honeypot exported logs files	60

1. Introduction

1.1. General Introduction

In the world of information technology and the Internet, the increase in connected objects, online services, and the amount of data in transit, has led to the emergence of a new paradigm. Cloud computing has evolved into new concepts such as Edge Computing, which brings computational intelligence closer to the periphery to meet the demands of these increases.

Through the various layers that make up Edge Computing, attacks are constant, and devices connected to the Internet are continually scanned by hackers. Their objectives are to find valuable data, critical applications, and vulnerabilities in network infrastructures.

The honey pot is by vocation, an information system, more or less artificial, whose main purpose is to be probed, attacked, and possibly compromised in order to make a hacker believe that he has penetrated the real network of the organization.

Thanks to the honey pot, the hacker is removed from the real network and its activities and movements can be captured and monitored. The information gathered about attackers allows us to study their methods, motivations, and the attack techniques they use to penetrate or corrupt a system. The goal is to develop ways to improve system security and prevent future attacks.

One of the sectors affected by these security improvements is the software development industry. It is turning more and more to movements like DevOps, which makes it possible to improve the performance and quality of software, facilitate collaboration between the development, security, and operating teams, but also to reduce the development life cycle. This methodology is organized around a continuous pipeline, characterized by the use of continuous development technology, continuous deployment, and continuous integration.

One of the major problems of this life cycle is the lack of continuous security integration. Attention to security issues is often given far too late in the development stages, and when they appear a posteriori, they can have significant financial implications.

In order to reduce threats and address this lack of security, DevOps is now moving to DevSecOps, which adds a layer of security to the pipeline, integrating solutions throughout the development lifecycle.

As part of this thesis, we will propose the implementation of a DevSecOps architecture, in the context of Edge Computing, integrating a honeypot solution in order to respond to a threat.

The concept of Edge Computing will be materialized by an autonomous car representing a connected object and the threat will appear by the creation of a backdoor allowing the control of the autonomous vehicle. Following the discovery of this threat, a response is given by the deployment of a honeypot through continuous deployment and integration tools, specific to the DevSecOps pipeline.

1.2. Context

This project is part of a master's degree in computer science at the University of Namur. During the bridge year and as part of the introductory course to the scientific process, I chose the topic “Honey-pot and IoT”. During the master’s year, it was quite natural that I decided to turn to this theme again to propose a subject of master thesis to my promoter, Professor Jean-Noel Colin. After several discussions and reflections with Mr Christophe Ponsard, Research-Innovation-Enterprise Coordinator at CETIC [CETIC] and lecturer in “IT Security”, we discussed the possibility of doing this work in collaboration with CETIC.

CETIC, for “Centre d'Excellence en Technologies de l'Information et de la Communication” is an applied research centre in computer science whose mission is to support regional economic development by transferring to Walloon companies, and SMEs, the most innovative results from applied research in Information and Communication Technologies (ICT) and Digital Technologies.

To this end, meetings were organized with Sébastien Dupont and Guillaume Ginis, research engineers at CETIC in the MBEDIS unit, who mentored me during my research. A topic then emerged around the implementation of a honeypot solution in a DevSecOps pipeline in an Edge Computing context.

Indeed, research on this topic can contribute to two research projects in which CETIC is involved. Both researches carried out within the framework of the SPARTA project, but also within the framework of the CyberExcellence project in which the University of Namur is also involved.

SPARTA [SPARTA], for “Strategic Programs for Advanced Research and Technology in Europe”, is a European research programme whose objective is to develop and implement leading research and innovation actions in collaboration with other European research centres.

CyberExcellence [CyberExcellence] is a research project linked to the activity of Walloon and Brussels cybersecurity research actors whose objective is to establish a single core of tools allowing to implement solutions based on effective and thoughtful cybersecurity with a competitive advantage for Wallonia recognized internationally.

1.3. Objectives

The main objective of this work is to contribute to the advancement of research on projects led by CETIC and UNamur, quoted above [GithubCetic].

First by analysing the DevOps domain and more particularly DevSecOps to determine the possible links with the implementation of a honeypot.

Secondly, by analysing the context of Edge Computing and to what extent the use of a honeypot has an interest in this context.

Then, by continuing the research in the field of honeypots with the drafting of a state of the art and the highlighting of aspects of cyber-security existing through the management of threats.

Finally, proof of concept is proposed, and a case study is set up based on this research and more specifically around the theme of integrating a honeypot solution in the DevSecOps pipeline, in a context of Edge Computing.

This last part will be the subject of a reflection around the results and potential future work.

2. DevSecOps Pipeline

2.1. DevOps

DevOps is a modern philosophy that emerged in 2009 [Pathak2022] that consists in reducing the life cycles of software development. It was born on the one hand of a desire to use and globalize agile methods in development processes. On the other hand, it helps to solve organizational problems between development and operational departments, with emphasis on communication and collaboration, continuous integration, quality assurance and delivery with automated deployment using a set of development practices [Jabbari2016].

DevOps is therefore the combination of Development and Operations. "Dev" refers not only to the development phases, but also to the entire team involved in developing a software solution. It combines practices, people, tools, technologies, and processes from different disciplines, such as planning, testing, quality assurance, etc. Similarly, "Ops" refers to the operational phases and to all members of the operating team: system administrators, system engineers, database administrators, operating personnel, publishing engineers, etc.

DevOps helps to foster the automation and monitoring of each step of application creation shown in Figure 1, from development, integration, testing, delivery to deployment, operation, and maintenance of systems in a continuous pipeline.

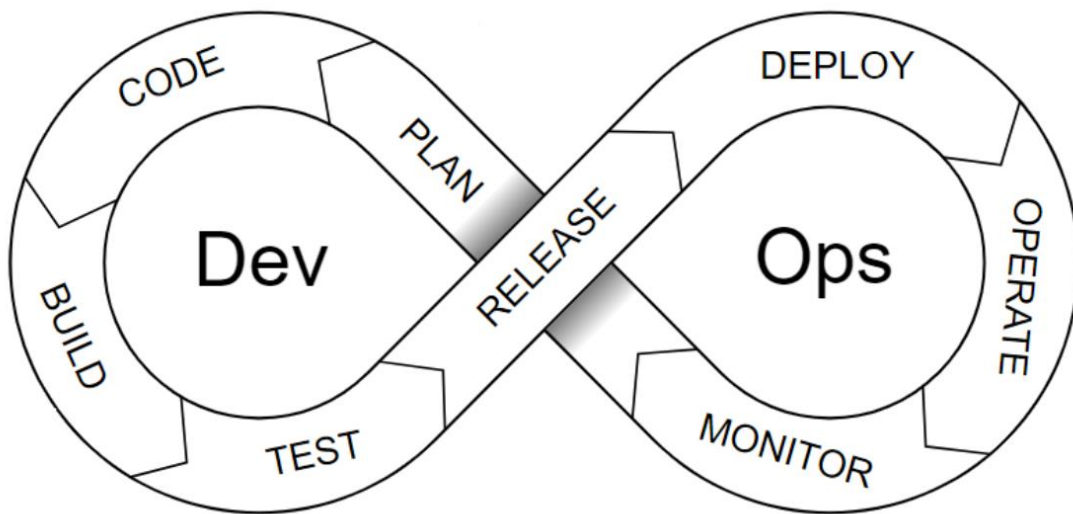


Figure 1 – Phases of DevOps lifecycle [Dupont2021]

2.2. DevSecOps

Security has long been considered an activity that begins after production and has sometimes never been considered until problems arise on the final product.

DevOps principles often do not integrate security teams into their CI/CD pipeline. Adding security to the pipeline translates into secure applications from the design stage (“by design”), which reduces the costs of its implementation, as the security problems and the financial and reputational damage they will cause are much less.

DevSecOps [Benchrifa2021] is therefore the answer to the integration of security in the early design and production phases as it aims to increase speed and reduce the frequency of bugs and security issues and their impact on production through continuous security (CS).

Concretely, DevSecOps solves these problems by placing security earlier in the application development lifecycle to reduce vulnerabilities and bring security closer to objectives through threat modelling, intrusion testing exercises and, to some extent, security monitoring, and the automation of incident responses when considering the SIEM/IDS and SOAR tools.

2.3. Honeypot in a DevSecOps Pipeline

As a response to security incidents, it is interesting to integrate a honeypot solution allowing to evaluate the security of the components during certain phases the life cycle of the development of an application. A honeypot makes it possible to study both the behaviour of attackers in terms of implemented security techniques, but also to highlight possible flaws and vulnerability.

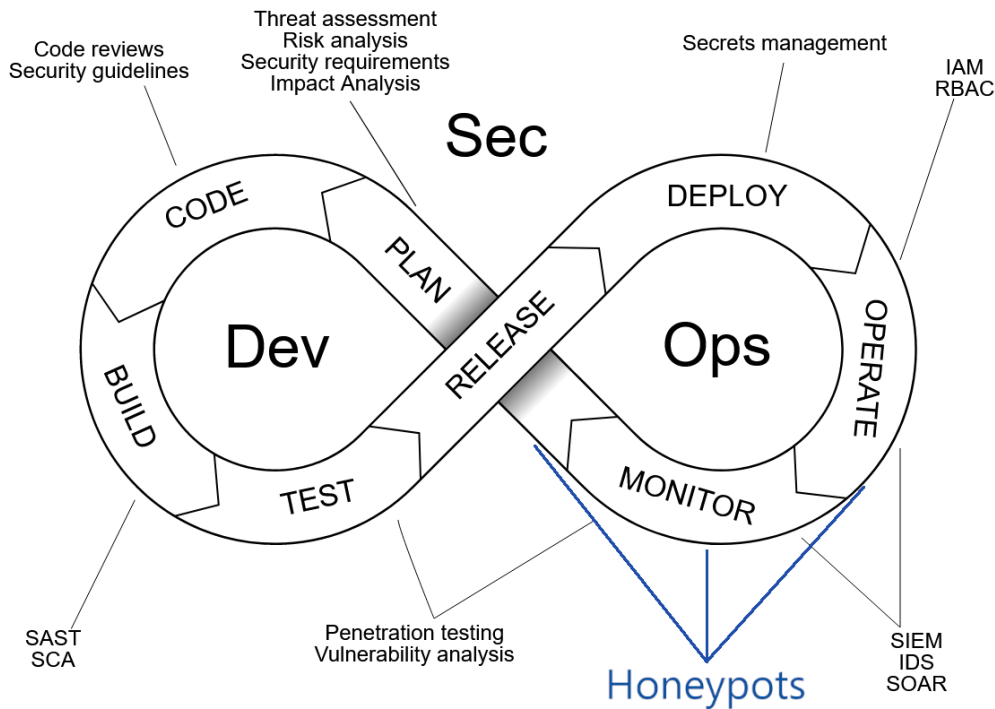


Figure 2 - DevSecOps lifecycle with sample security activities [Dupont2021]

In the DevSecOps lifecycle, the honeypot finds its place in 3 key stages, namely "Plan", "Operate" and "Monitor":

1. The honeypot is initially linked to the "Plan" phase because it is in this phase that its implementation will be decided. Depending on the results obtained using the security tools specific to the planning phase (Threat assessment, Risk analysis, Security requirements, impact analysis), the honeypot deployment strategy and its characteristics will be defined so that it is deployed in the next iterations of the phases.
2. In the "Operate" phase, the honeypot has been deployed on the basis of the "Plan" phase and is in operation. It acts as a decoy and is attacked and probed. It makes it possible to attract the attention of attackers, both to protect the real system in production, but also to study their behaviour.
3. In the "Monitor" phase, the honeypot collects security data using an intrinsic sensor, such as a keylogger, to retrieve keyboard inputs. In addition to using a honeypot, other tools are used. First an IDS, which will identify events that occur in and out of the honeypot. These events are then transmitted to SIEM, which will analyse the data and compare it to vulnerability databases. Finally, a SOAR will respond when a threat is detected. Note that the deployment of a honeypot can be a countermeasure to the detection of a threat in a system in production.

3. Edge Computing

In the Internet world, cloud computing allows businesses to provide IT services online. Users can benefit from various on-demand online services, such as access to software (SaaS), access to platforms (PaaS) and the provision of hardware resources (IaaS). The main objective of setting up cloud computing in companies is to minimize the costs of the IT infrastructure and to be able to adapt its resources according to its needs and the fluctuation of needs.

Beyond the companies themselves, cloud computing is also one of the essential elements for the advent of the Internet of Things. Indeed, IoT and cloud computing are now inseparable and linked to each other. The resulting Internet applications can be related to the world of industry or business, but also for domestic applications. There are applications such as surveillance, virtual reality, traffic monitoring, home automation, smart cities, logistics, etc. which have a purpose to serve the population in their daily lives.

Every year, the number of connected objects increases significantly and could reach 14.4 billion by the end of 2022 [Hasan2022]. With this increase and the growing use of cloud applications and services by these mobile devices, mobility-related issues have emerged, showing the limits of centralized computing that relies on hundreds or thousands of servers running applications in consolidated data centre.

One of the concepts to solve these problems is Edge Computing [Khan2019]. Edge Computing is an optimization concept that consists of processing data and the intelligence of its processing on the periphery of the data source. it makes it possible to meet the application requirements, but it also brings a new set of challenges [Fruehe2020] to the network.

- **Security:** the major challenge of security is the management of paradigm change. Centralized data security enables the standardization of technical and physical security. Edge Computing and its layered structure requires thinking about network models, physical security, traffic models and above all the management of user access rights on a much larger number of devices. Indeed, as shown in Figure 3, each of the layers that make up the Edge and Fog computing has its technical specificities, resulting in a complexity of security management.
- **Network Bandwidth:** The way network bandwidth is distributed is changing. Traditionally, the bandwidth allocated to data centers is greater than the bandwidth allocated to end-of-line devices. For example, the increase in the use of video surveillance leads to a need for more network bandwidth closer to the edge and the cameras.
- **Distributed computing:** The increase in distributed computing and east-west traffic prompts thinking about consolidated compute models and device placement based on use cases due to capacity and capability constraints (e.g., limited compute, available GPU, etc.)
- **Latency:** The application latency and decision cost are reduced thanks to the proximity of computing resources to the edge. By limiting the effects of coming and going from the data centre to the to end-of-line devices, responses and actions are faster. One of the major challenges is managing the replication of data from the data centre to the edge.
- **Backup:** Business backup strategies are dependent on where the data resides. Network bandwidth requirements and storage support are equally critical in Edge Computing. However, network backups may not make sense depending on the application.

- Data accumulation: Data is a key business asset. Storing and accessing this data at the edge creates new challenges in the life cycle of this data.
- Control and management: From company to company, the physical edge location differs. Enterprises need to use newer orchestration tools to consistently manage and control applications regardless of location.
- Scale: Edge Computing requires increasing scalability across all computing disciplines: compute, networking, storage, management, security, licensing, and more. We must not be satisfied with increasing the number of servers, this challenge requires having a global reflection around the scalability of everything related to the edge.

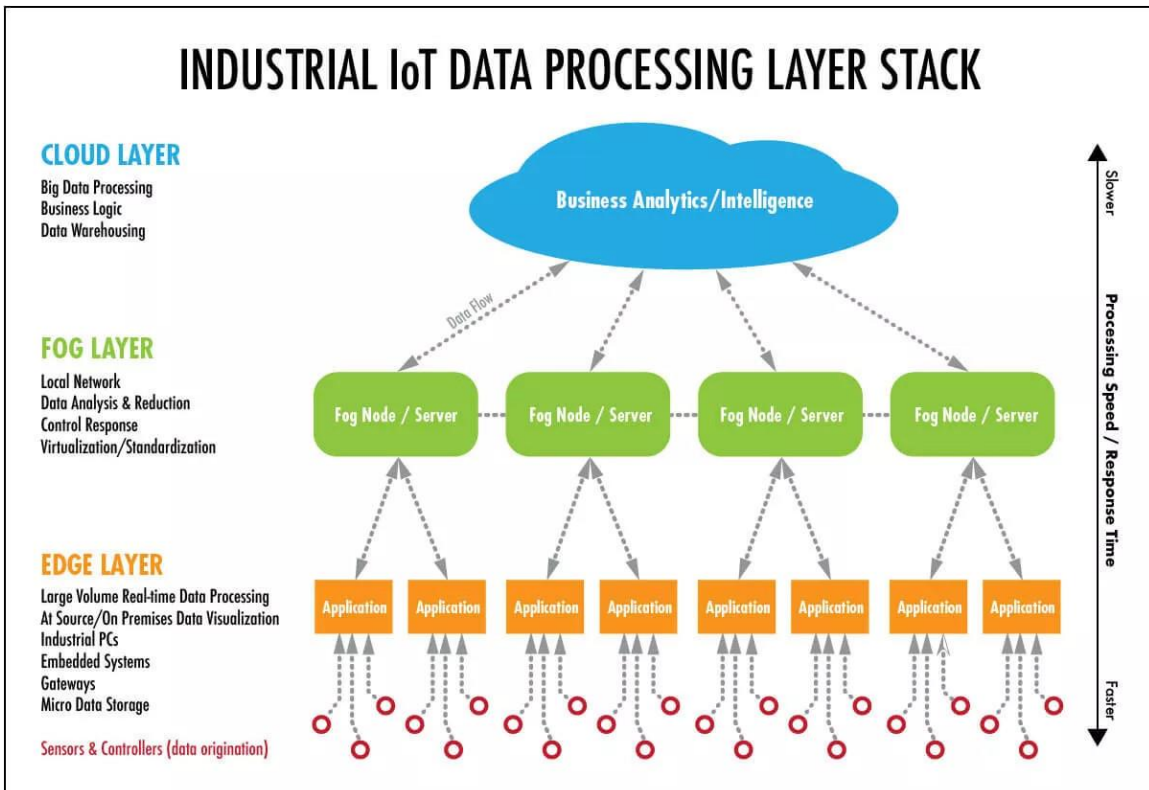


Figure 3 - Industrial IoT data processing layer stack [Winsystems2017]

In practical terms, Edge Computing is an advanced version of cloud computing that solves high latencies constraints and bottlenecks that are not properly managed in the cloud computing paradigm. It meets the requirements and challenges of the evolution of computer networks by bringing services closer to end users. One of the main differences between Edge Computing and cloud computing is the location of the servers. Public Edge Computing services are in the edge network while public cloud services are located on the internet.

There is also the concept of Fog computing which is very similar to Edge Computing. The major difference is that the Edge Computing tends to decentralize processing on devices at the periphery of networks, while Fog computing seeks to distribute processing on local network units using intermediate data aggregation strategies when necessary [Khan2019].

Other concepts exist, such as the one used by the “Open Nebula” application [Opennebula2021]. It offers an even more detailed view of "Edge Computing" by adding specific layers depending on the type of systems being connected. It also adds a notion of distance to categorize them. As shown in Figure 4, five categories emerge:

- The Public Cloud corresponds to resources located in data centres at more than 1000 Km and having a latency of 20 milliseconds.
- Public Edge, corresponding to national or private Data Centres, or on a company's premises. In this case we are talking about distances ranging from 100 Km to 1000 Km and having a latency of 10 to 20 milliseconds.
- 5G/ Near Edge, defining 5G and cellular antenna sites, aggregation nodes used in telecoms (operator's cabin, ...) and Utilities Networks (gas, electricity, water cabin, ...). These resources are between 1 Km and 100 Km and have a latency of 2 to 5 milliseconds.
- Even closer, the On-Premises Edge is, by definition, the resources that are nearby and stored "locally". For example, it is the computer systems of a hospital, a factory or a football stadium that are installed within these infrastructures. In this case, the resources are less than 1 km away and have a relatively low latency of 1 millisecond.
- Finally, the IoT Edge, or FAR Edge, represents all the connected objects of everyday life. These items are smartphones, bicycles, cameras, but also convenience services such as trains, kiosks, etc. These resources are considered "in range" and have very low latencies.

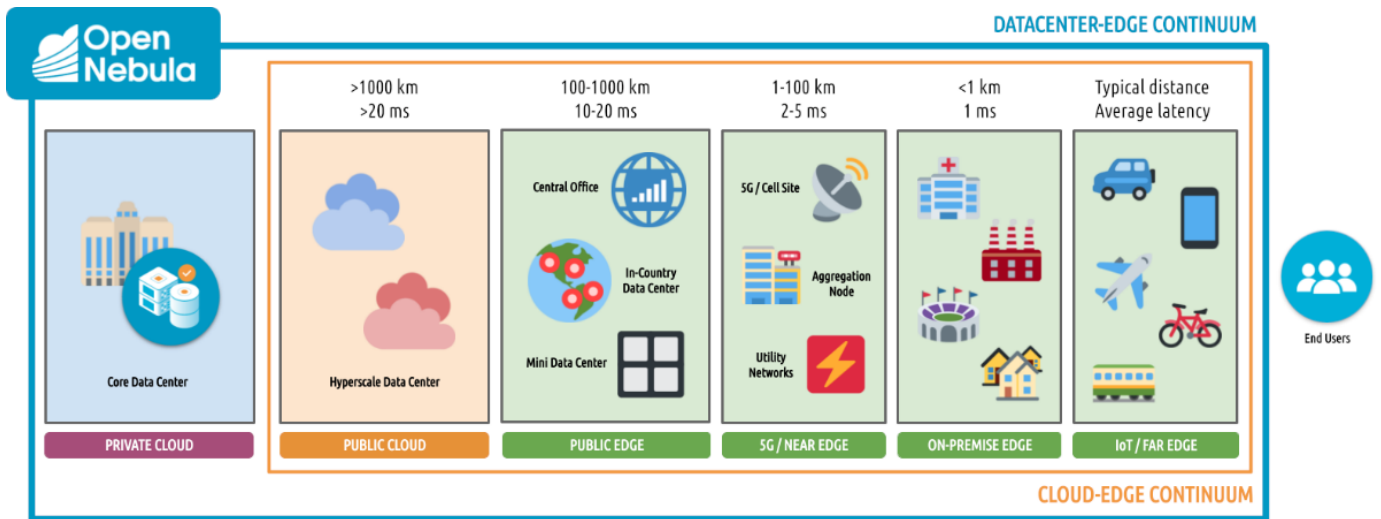


Figure 4 -Representation of the different concepts related to Edge Computing [Opennebula2021]

The availability of cloud services on the Internet depends on the distance of multiple jumps between the end-user and the cloud servers. The significantly high distance between the mobile device and the cloud server induces high latency in cloud computing compared to low latency in Edge Computing.

Beyond performance issues, it is likely that the number of data attacks on the road is higher in cloud computing than in Edge Computing due to the longer path to servers.

However, security issues are just as important in the world of Edge Computing. Gartner predicts, in a report [VanDerMeulen2018], that 75% of data created and processed by companies will be done outside of a traditional centralized data centre or cloud by 2025. In other words, the use of edge technologies is on the verge of enormous growth, and this growth is accompanied by new cybersecurity threats.

First because extending the footprint of enterprises using Edge Computing exponentially increases the surface area for attacks. Then, because a new vendor landscape compounds this risk. Unsecure endpoints are already used in distributed denial-of-service attacks or as entry points to core networks.

Next, Gartner adds in its report that “The network security architectures that put the enterprise data centre at the centre of connectivity requirements are an inhibitor of the dynamic access requirements of digital enterprises”. He also adds: “Digital companies and advanced computing have reverse access requirements, with more users, devices, applications, services and data located outside a company than inside.”

3.1. Honeypot and Edge Computing

Using honeypot in the different layers that make up Edge Computing can be a great way to study attackers, their tools, techniques, and procedures, and how they would try to circumvent the actual security controls that make up these different layers.

One of the solutions proposed by Stolfo [Stolfo2012] is to use decoy and honeypot techniques to limit the damage caused by stolen data by reducing the value of this stolen information. They posit that Cloud Services and Edge Computing, which describe methods for storing documents, files, and media in a remote service accessible anywhere a user can connect to the Internet, can be implemented by profiling user behaviour and using decoys.

Concretely, the behaviour of the user is analysed and if it turns out abnormal, luring information can be returned by the Cloud and delivered to the attacker in such a way as to appear completely legitimate and normal. The use of this decoy information makes it possible to detect malicious activity, to create confusion for the attacker who will have to spend resources to distance the real from the false information, naturally creating a deterrent effect.

This solution could be extended for each of the Edge Computing layers. Indeed, and for example, at the "IoT Edge" layer, using client honeypots imitating smartphone behaviour and accessing resources from the "On-Premises Edge" layer. Or, by using server-type honeypots, providing Web Apps-type services, and returning decoy information in the event of positive profiling.

4. Honeypots: State of the art

4.1. Definition

In common language, the "honeypot" in the figurative sense of the term, designates a tool used to attract wasps thanks to the smell of honey placed at the bottom of the pot. When the wasp enters it, its legs remain stuck to the honey and the insect is then trapped. In the IT sense of the term, we must turn to Lance Spitzner [Spitzner2003] who defines the honeypot as follows: *"A honeypot is an information system resource whose value lies in the unauthorized or illicit use of this resource"*.

This very general definition induces the notions of bait, lure, and surveillance of intruders.

In the information technology world, attacks are constant, and devices connected to the Internet are constantly scanned by hackers. Their goals are to find valuable data, critical applications, and vulnerabilities in network infrastructures. The honeypot is by vocation, an information system, artificial, whose main purpose is to be probed, attacked, and possibly compromised to make a hacker believe that he has penetrated the real network of the organization. Thanks to the honeypot, the hacker is removed from the real network and his activities and movements can be captured and monitored. The information collected on the hackers makes it possible to study their methods, their motivations, and the attack techniques they use to penetrate or corrupt a system. The goal is to develop ways to improve system security and prevent attacks in the future.

The location of the honeypot largely depends on its sophistication, the traffic it aims to attract, and its proximity to sensitive resources within the organization's network. It can be placed inside the DMZ in the middle of the services (web, mail, ...) intended for the public (internet), on decoy servers. It can also be placed outside the external firewall to detect attempts to enter the internal network. Nevertheless, it will always be better to isolate it from the production environment.

There are several advantages of using a honeypot. The first is that it allows collecting data from real attacks, allowing to provide rich information to the cyber security manager. Then, it also helps to discover new attack patterns and new vulnerabilities to define countermeasure strategies. An interesting aspect is that it reduces the volume of false positives from alert systems, since no ordinary users are supposed to connect to them. We can also mention the fact that setting up a honeypot can be quite inexpensive in terms of investment and performance. Nevertheless, some complex systems require specific skills, which can reverse the trend of low implementation cost.

There are also some risks associated with implementing a honeypot that need to be considered. To function properly and produce information, the honeypot must be attacked. It is therefore limited to the analysis of the data collected and dependent on the attacks carried out on it. For the system to be attacked, it must be attractive to hackers. If they are experienced, they can easily recognize that they have a honeypot in front of them and therefore that they are not in the presence of a legitimate system.

Another risk to consider is the fact that a honeypot can generate unwanted alerts and false positive alerts. Indeed, since a honeypot is voluntarily exposed and vulnerable, it is necessary to properly manage alerts so that it is identified by surveillance systems as a system undergoing attacks and that it is not quarantined and becomes useless.

Finally, the biggest risk is the compromise of the honeypot. If the attacker manages to take control of the honeypot, he can use it to his advantage and provide bad information to the administrator to divert his attention from the real objective. He can also use it to perform other attacks or to create his own honeypot. The experienced hacker could even use the honeypot as an entry point to the real system if a connection exists between the two (Logs, synchro, ...).

To prevent the hacker from discovering the deception, the honeypot must be functionally an easy target, but not be perceived as such.

A few clues allow a hacker to identify that he is communicating with a honeypot [Lutkevich2021]:

- The system has all its network ports open.
- The system has unusual ports open (systems connected to the Internet usually only have basic services available).
- The system is a site or a server that is too easy to hack.
- The system consists of directories with extremely literal names indicating something of value, like "social security numbers" or "credit card data".
- The system has very little software installed.
- The system has a lot of free space on the hard disk, which indicates that it is not a well-used production disk.

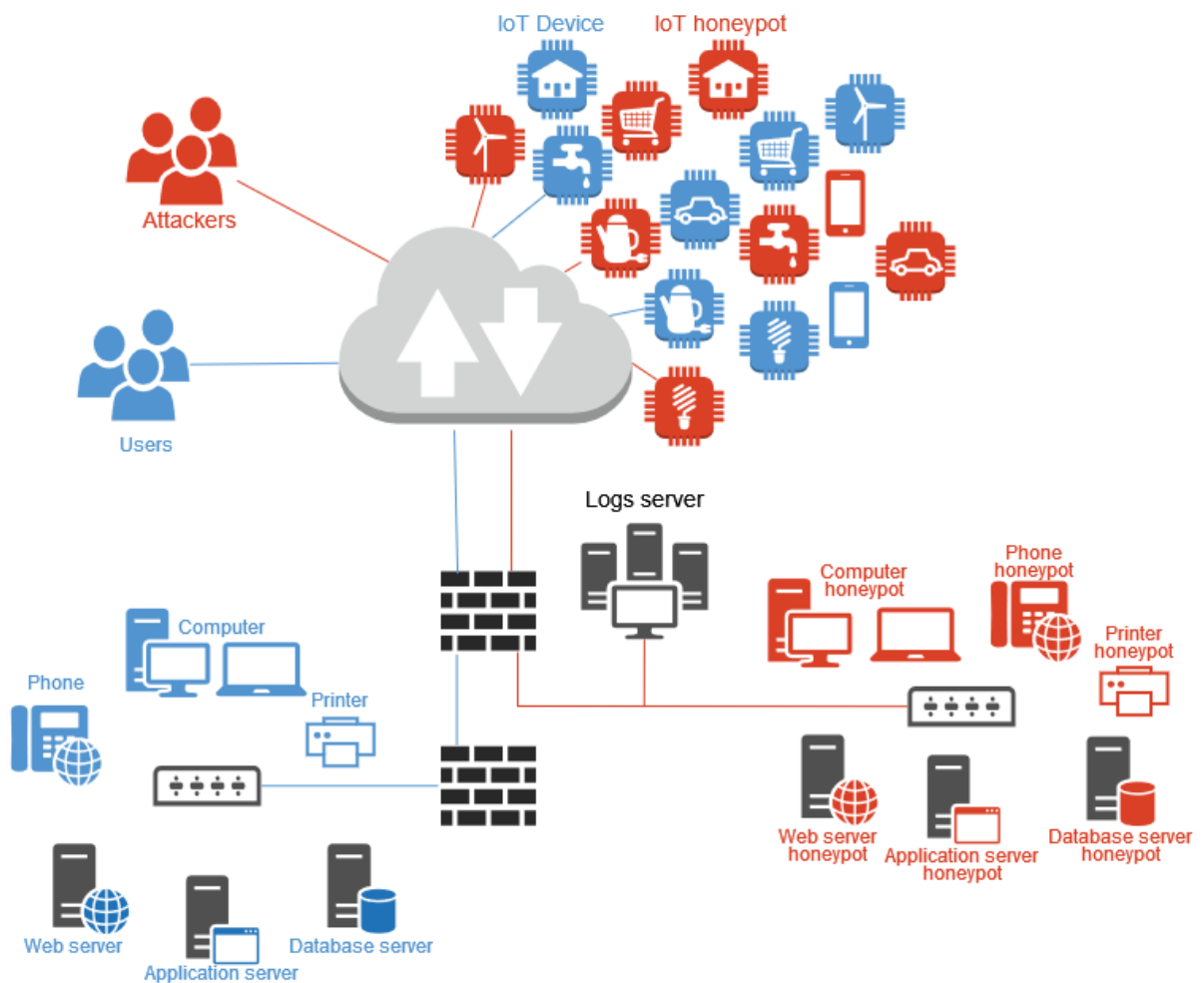


Figure 5 - Basic diagram of different environments using honeypots [Diagram2022]

This figure (Fig.1) shows the extent to which IoT, and production environments coexist with honeypots. On the one hand, "IoT"-type connected objects, which are found in everyday life, and which are required to make life easier for users.

On the other hand, the production environments of companies which allow their legitimate users to connect from the Internet to reach corporate resources. By trying to access any potentially exploitable resource, attackers will also find themselves faced with honeypots.

These honeypots, depending on their characteristics, will adopt behaviours to study attackers' strategies, highlight vulnerabilities and protect legitimate systems.

For example, a honeypot phone will simulate the use of the SIP protocol and allow requests to be sent on port 5060. A "database" type honeypot will simulate the behaviour of a real database and may even contain decoy data that can be traced later.

4.2. Characteristics of honeypots

The honeypots can be classified by means of nine features [Diagram2022]:

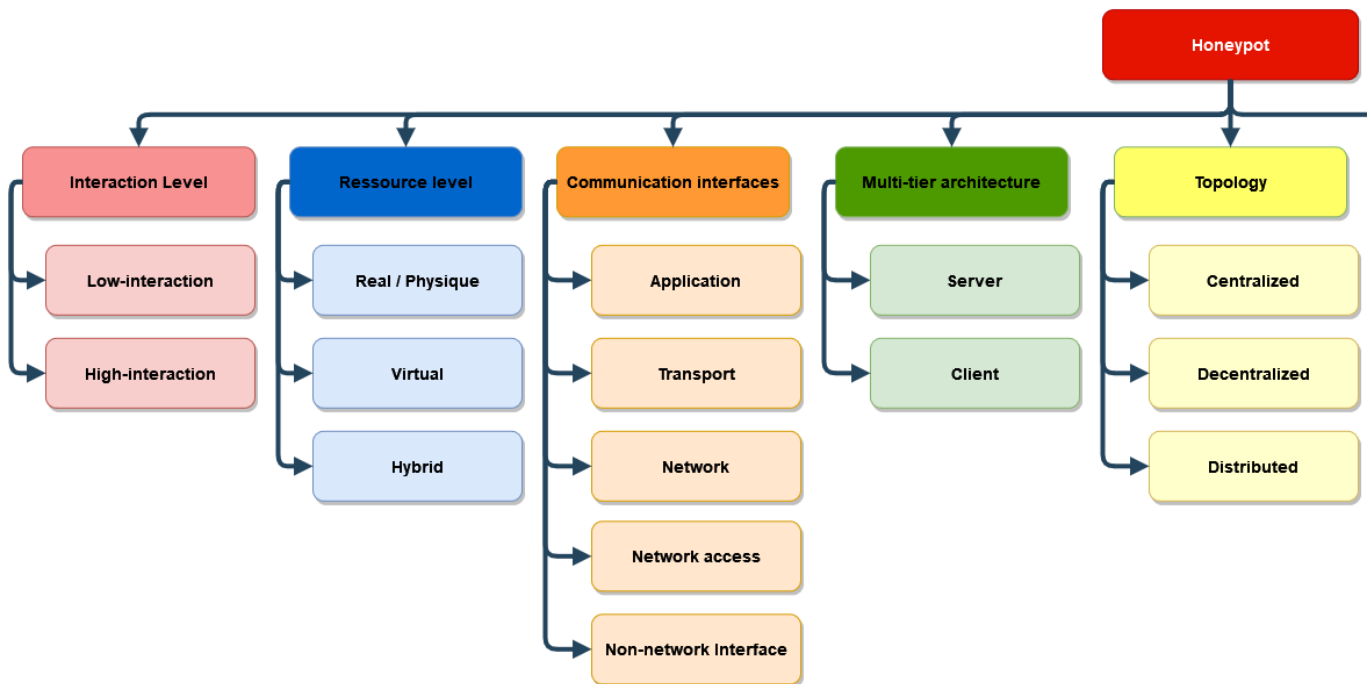


Figure 6 - Diagram of the different characteristics of a honeypot – part 1 [Diagram2022]

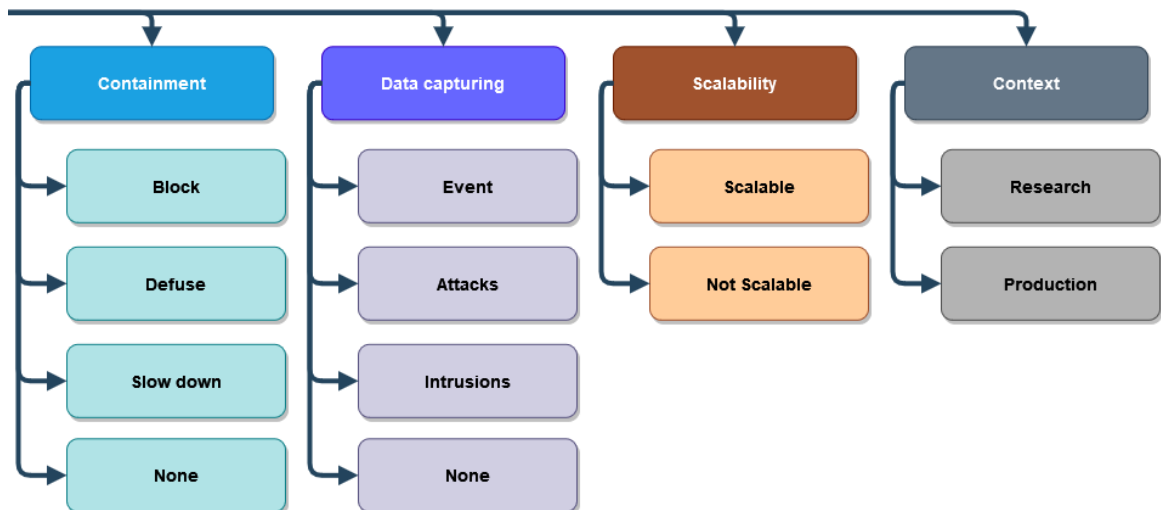


Figure 7 - Diagram of the different characteristics of a honeypot – part 2 [Diagram2022]

To represent and understand the different characteristics of honeypots, a concrete example is used throughout this section. The example honeypot represents a wastewater treatment plant.

Here is a diagram:

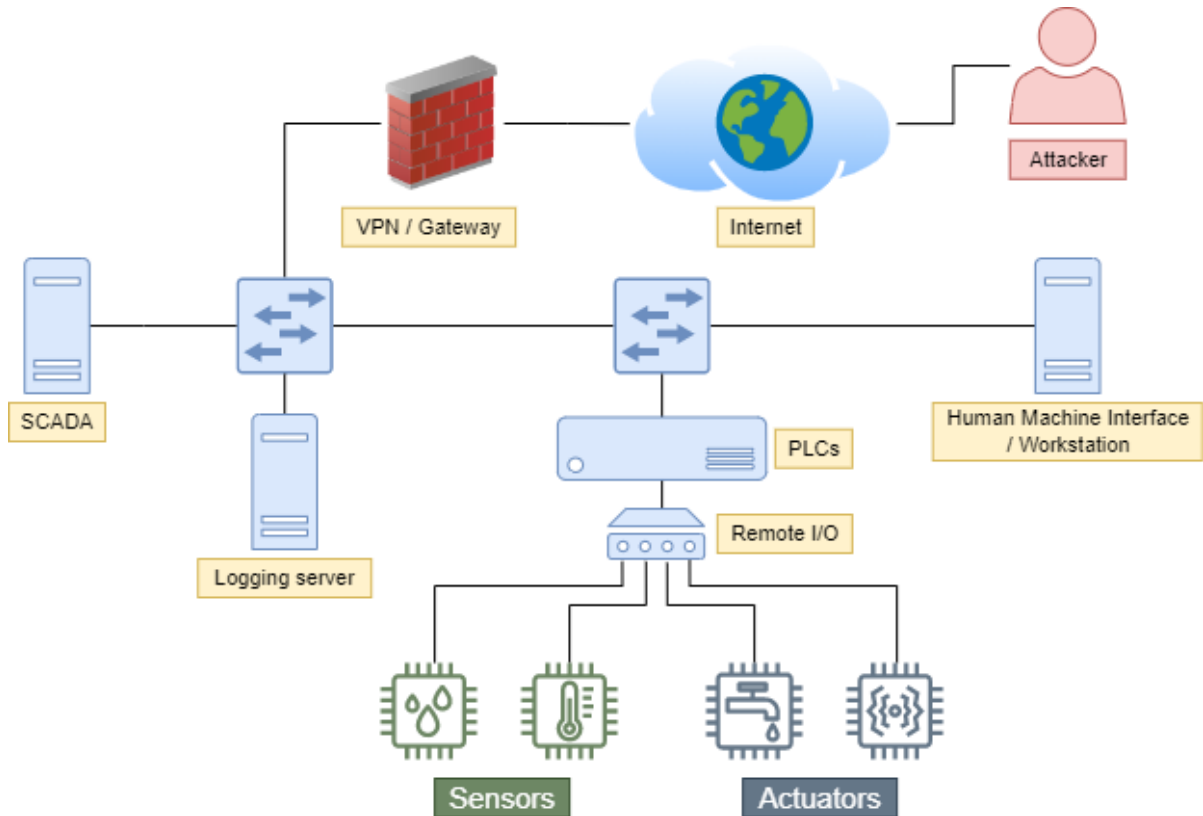


Figure 8 - Diagram of a wastewater treatment plant honeypot [Diagram2022]

The context used as the honeypot example represents the infrastructure of the wastewater treatment plant [Antonioli2016]. Depending on the characteristics and according to the needs, we will consider either a part or the entire infrastructure as being the honeypot.

It consists of a firewall serving as a gateway to and from the outside. Within the network, we find the elements usually constituting this type of industry. There is a SCADA, the control, and data acquisition system, but also a log server, allowing this data to be stored over time. These systems are directly connected to the PLCs. The PLCs make it possible, on the one hand, to recover information from the various sensors (water temperature, flow rates, etc.), and on the other hand, to send orders and commands to the various actuators (valves, bypass, ...). If some of these components (sensors and actuators) are directly connected to a PLC, others can be located in relatively distant geographical positions. The RIOS (Remote Input/Output) are then used to make the link between the PLC and the components.

Finally, all the components are visualized and checked by the engineers and technicians of the Man-Machine Interface. This interface is comparable to a traditional workstation.

4.2.1. Interaction Level

The interaction level of the honeypot indicates to what extent its functionality is exposed. Three values are used to define it [Franco2021]:

- Low interaction: honeypot have low level of realism. The interaction between the Hacker and the system is short and limited. It is generally not a system per se, but rather an emulation of operating systems and services. This type of honeypot can simulate only specific systems services and uses basic techniques like identifying port scans [Ahn2019], generating attack signatures, analysing trends, and collecting malware information. Low-interaction honeypots are particularly useful for monitoring fewer complex attacks, such as robots and malwares.
- High interaction: The honeypot provides the most realistic environment for attackers because it uses a variety of real services running on real operating systems. It imposes few restrictions inside the system, encouraging attackers to interact with it. High interaction honeypots collect information about all the attacker's movements and actions, which is an advantage because the information collected is very faithful. Additionally, it makes them more effective at monitoring more experienced hackers, as they look realistic and are likely to deceive them. This type of honeypot is effective against automated attacks but involves risks of compromise, since it goes so far as to allow the attacker to obtain root access, giving him full power over the system.
- Medium interaction: A pot of honey offering a mix between low and high interaction.

In the case of the proposed example, if the attacker tries to break into a PLC, he will want to retrieve the list of TCP ports open on it. The low interaction honeypot will then return a list of ports and give information about the status of it. For example, when the attacker wants to try to connect and authenticate on the SSH port of the PLC, the response from the honeypot will be simulated by a script and will send back a defined message. This type of honeypot is effective against automated attacks. While the high interaction honeypot allows the attacker to be able to interact with open ports and, for example, login via SSH port using low credential. The connected attackers then have free rein and can retrieve information from the sensors or modify the state of the actuators, as a real PLC would do. In the case of a Medium-interaction honeypot, one can imagine on the one hand, that certain components are "high interaction", for example for the network part and the connection to the PLC via the SSH port, as explained above. On the other hand, components simulated by scripts which can return fictitious information from sensors or to simulate the open/closed position of an actuator.

4.2.2. Resource level

The resource level used by the honeypot is determined by the replication realism of the system. There are three possibilities:

- Real / Physical: is a replication of a target system.
- Virtual: use virtualization technology to reproduce a system. This solution is low-cost.
- Hybrid: include a mixture of real and virtual device. This solution is cost effective.

To better understand, let's take the example of the wastewater treatment plant. Physical would be a carbon copy of a processing wastewater plant in production. Requiring an identical infrastructure at all levels. If the *real* wastewater plant uses 20 PLCs, the honeypot will use 20 real PLCs. One can very easily imagine the difficulty and the costs generated by such a deployment. To provide additional information, if we decided to limit ourselves to the creation of

a honeypot consisting essentially of servers, then the so-called "physical" honeypot would be an exact replication of all these servers. The virtual honeypot is a virtual copy of the wastewater plant. Hosted on a machine capable of virtualizing the layers and services of each component. Virtualization makes it possible to create the 20 PLCs in such a way that they perfectly imitate the behaviour of the real components of the wastewater plant. In the case of hybrid, one could quite imagine that the network infrastructure is completely virtualized, but that the PLCs are real physical components.

4.2.3. Communication interfaces

Communication interfaces are the characteristics that define how the honeypot will be reached and through which interface one will be able to interact with it [1]. To illustrate the different communication interface possibilities of a honeypot, we can use the TCP/IP model as a reference and comparison model. Each interface can be considered as a gateway to a system for attackers. The interest of defining the layers in which these interfaces are located makes it possible to specifically categorize honeypot entrance doors. Indeed, attackers can specifically target a protocol, or an application and this model allows to have a better overview. A honeypot can have several exposed interfaces and each one belongs to one of these characteristics:

- Application: layer specifying the applications used by the honeypot and which will be exposed (HTTP, SMTP, SSH, DNS, ...)
- Transport: layer specifying how end-to-end communications are managed between processes (TCP, UDP, SCTP, ...) and possible
- Network: layer specifying the packet routing method (IP, ICMP, IGMP, IPSec, ...)
- Network access: layer specifying the physical means (cable, radio, ...) and connection (Wi-Fi, Ethernet, ...) used to communicate with the honeypot.
- Non-network Hardware Interface: honeypot interfaces with Non-Network Hardware Interfaces (PLC, USB, etc.).

As an example, in the treatment plant, we can take the case of a honeypot imitating the behaviour of a server providing to client's different resource shares (files and printers) via the SMB (Server Message Block) protocol. The honeypot will receive the "Application: protocol SMB" feature as it is the only "exposed" interface by the honeypot. If we decide to expose flaws on other layers of the model, then we can also add these layers by specifying more precisely the protocol or technology used: -- Application: SMB protocol -- Transport: TCP protocol -- Network: IP v4 -- Physical: Cable via Ethernet We can also take an example of honeypot disconnected from the network. Let's imagine that each USB key introduced in the building must be first connected to the honeypot (sandbox) before being used on a legitimate computer. The goal is to verify that the key does not have malware capable of infecting a system and propagating to other systems. In this case, the honeypot will receive the characteristic of Off-network hardware interface.

4.2.4. Multi-tier architecture

Honeypot can behave in 2 different ways when it comes to its interaction [Mohammadzadeh2011]:

- Server: honeypot acts as a server and passively waits for client requests
- Client: honeypot acts as a client and actively initiates requests to servers

To illustrate the use of a client honeypot, let's imagine that the legitimate workstations of the treatment plant must connect to external resources to send water quality analysis data. The implementation of the client honeypot consists of the deployment of a robot capable of visiting a list of URLs to verify their reliability and legitimacy. The honeypot will send http/https requests to this URL and the responses from the remote servers will be analysed. Servers containing malware will be considered dangerous and will be listed as such. Thus, avoiding a connection from the workstations of the station.

The second is the reverse role, namely the honeypot server, which will allow clients (PLCs, Workstations, control systems) to connect so that they use a service provided by the server. Clients will, for example, connect via the SSH protocol to the server to send analysis data. The server honeypot will then behave like a legitimate machine and send the corresponding responses to the requests initiated by the clients. Customer activity data is collected and analysed to verify that no customer attempts to execute illegal commands and access unauthorized information.

4.2.5. Topology

The topology of a honeypot determines its physical, software or logical architecture. It is generally represented with nodes connected to each other and defining a structure:

- Centralized [Pouget2005]: Centralized honeypots use a client/server architecture where one or more client components (client nodes) are directly connected to a central server (central node).
- Decentralized [Baykara2019]: Like the blockchain, decentralized honeypots also use a client/server architecture, with the difference that the honeypot consists of several servers (central nodes) communicating with each other. These central nodes also communicate with client components (client nodes) which are specific to them.
- Distributed [Shi2019]: Distributed honeypots do not work with a client/server architecture, so there are no client nodes and central nodes (even if some nodes have coordination and arbitration roles). It is a mesh architecture where each node relates to several other nodes. These nodes work together to form a honeypot.

Within the wastewater treatment plant and to illustrate a Centralized topology, one can imagine a virtual honeypot, made up of a virtual machine acting as a central server and containing a RedHat operating system and making it possible to simulate the behaviour of a SCADA. Along with this, the central server contains the logging system and can also simulate the behaviour of a workstation. The client components (PLCs, RIOS, ...) are on other virtual machines and communicate directly with the central server. Client nodes synchronize with the node and retrieve configuration information (NTP, routines, commands, etc.) or send data for analysis. Concretely, all communications pass through the central nodes, if this were to be out of service, the client nodes would become an orphan, resulting in a total failure of the system, which constitutes a Single point of failure.

To explain the Decentralized honeypot, just take the example above and imagine that we duplicate the virtual honeypot 3 times, while keeping the components and simply modifying their

configurations so that they have the appearance to be on another treatment plant site. We get 4 honeypots. Now let's add a connection between these 4 honeypots and a totally independent logging server. The honeypot now has a Decentralized topology. Each of the central nodes is completely independent. If one of the nodes were to go out of service, the other central nodes would continue to operate and communicate with each other, causing a partial failure of the system.

The Distributed topology of a honeypot can be explained by imagining that a network of honeypots has been created consisting of workstations within the treatment plant. Each workstation is a honeypot in its own right, intended to be compromised and for which we will analyse incoming and outgoing traffic. Each of these workstations is independent and aims to report information. If one of them is compromised, the entire infrastructure continues to operate, without degrading the integrity of the system. This type of honeypot is for example used to counter DDoS attacks [Deshpande2015].

4.2.6. Containment

The containment strategy defines the ability of a honeypot to defend itself against attacks, other malicious activities spreading from it. A honeypot can use several of these strategies [Seifert2006]:

- Block: attacker is identified and blocked, he never reaches his target
- Defuse: attacker reaches his target, but his attacks are defused; he is manipulated way to fail
- Slow down: The attacker is slowed down throughout their attack process and the benefits of parallel automated attacks are reduced. He will consume resources and will end up getting discouraged [Ponemon2016].
- None: No action is taken to prevent the attacker, the aim being to study all his actions.

To illustrate this characteristic, one can imagine that a hacker has entered an HMI of the wastewater treatment plant and tries to modify the position of a valve (actuator). Based on this observation, several reaction strategies are possible. In the case of a Block, one can imagine that as soon as the honeypot has detected and identified the pirate, it is automatically blacklisted, and the connection is interrupted. In the case of a Defuse, the pirate has already achieved his objective and the command to modify the position of the valve has been sent. From this moment, the honeypot can detect unauthorized activity. The attack is defused, and the valve has returned to its normal state. In the case of a Slowdown, everything is done to slow down and discourage the attacker. When he thinks he has sent the command to modify the position of the valve, another verification is requested, the process is slowed down or fails. Concretely, he is faced with new pitfalls and his objective is never achieved. Finally, if we decide that the honeypot of our treatment plant is intended to study the behaviour of the hacker, we can decide to let him act with impunity and study his methodology. The attacker can reach his target with or without difficulty, depending on the implementation of the honeypot. In this case, no containment policy (None) is applied.

4.2.7. Observability

This characteristic does not directly concern the technical aspects, but rather its ability to capture types of data ⁴:

- Event: collects data related to an abnormal event that has occurred or linked to a change in the state of a system or a component.
- Attacks: collects activities threatening the security policy.
- Intrusions: collects policy-threatening activities that lead to a security breach.
- None: no data collection.

Returning to the example case, if there is an unexpected event capable of being detected by the honeypot, such as the unwanted opening of a valve, then it will receive the Event data capture characteristic. If, on the other hand, the honeypot can capture the unsuccessful attempts of a hacker trying to modify the position of the valve, then it will receive the Attack data capture characteristic. If the honeypot can capture the events produced by an attack resulting in an intrusion, then it will be assigned the intrusions data capture characteristic. Finally, if the honeypot is not intended to capture information, as is the case for a decoy intended to imprison hackers to prevent them from acting. They can be assigned the characteristic of None.

4.2.8. Scalability

Scalability determines the ability of a honeypot to evolve. By this we mean its ability to be modified to increase the number of decoys and services it offers [Franco2021]:

- Scalable: can increase the number of services and lures offered
- Not Scalable: Only has a set number of services and cannot be changed at that level.

Let's take the example of the PLCs present in the structure of the honeypot of the wastewater treatment plant. If PLCs can be added, then the honeypot is considered scalable. On the other hand, if it is impossible to add more, then it is considered as non-scalable.

4.2.9. Context

This characteristic is used to define in which context the honeypot is used and what is the purpose of its implementation [Franco2021]:

- Research: is used to lure attackers and study their behaviours and the tools they use to develop better protection against these attacks.
- Production: is used to defend and prevent attack on real systems and to protect it. It is important to note that a honeypot can serve both as a means of protection, while collecting data allowing research, therefore, a honeypot can be assigned both values of this characteristic.

To illustrate this characteristic, one can imagine the existence of a virtual honeypot with high interaction identically imitating the behaviour of a real wastewater treatment plant in production. Its deployment can be carried out within the framework of research, we want to highlight the flaws that may exist in the existing system and know how to protect ourselves from attackers. In the case of a production, its purpose is above all to protect the real system and one does not necessarily seek to collect data for study purposes, but rather to divert the attention of the attacker.

4.3. Strength and weakness of characteristics

		Strength	Weakness
Interaction Level	Low-interaction	<ul style="list-style-type: none"> - Low resource Consumption - Limited risk of compromise - Effective against automated attacks 	<ul style="list-style-type: none"> - Easily detectable - Simulation limit - Lower scalability due to their simplicity - Simulation of services and application using scripting - No access to the operating system - High risk of compromise if poorly protected - More difficult to maintain and implement
	High-interaction	<ul style="list-style-type: none"> - Easy to implement - High level of realism - Access to the operating system, use of real service, important for the realism - Effective against non-automated targeted attacks 	<ul style="list-style-type: none"> - High Cost to reproduce the system - May lack realism (Response time too short, too powerful to be real)
Ressource level	Real / Physical	<ul style="list-style-type: none"> - Easy to deploy, maintain and restore - Cost limited to the host machine - Easy scalability 	<ul style="list-style-type: none"> - Requires special attention to integrate "Real" and "Virtual" components
	Virtual	<ul style="list-style-type: none"> - Perfectly mimics the behavior of a real system 	<ul style="list-style-type: none"> - Do not cover more specific protocols
Communication interfaces	Hybrid	<ul style="list-style-type: none"> - Good compromise between Real & Virtual - Allows you to specifically target costs and virtualize what is not essential 	
	TCP/IP model	<ul style="list-style-type: none"> - Covers most standard protocols used. - Provides a good overview of possible vulnerabilities. - Large knowledge base of existing protocols. 	
Multi-tier architecture	Non-network hardware interface	<ul style="list-style-type: none"> - Communication tailored and specific to what you want observe - Limits the risk of propagation ("offline" mode) 	<ul style="list-style-type: none"> - Very few existing security protocols - Requires a particular implementation
	Server	<ul style="list-style-type: none"> - Great way to study customer behavior 	<ul style="list-style-type: none"> - Passivity: Needs to be attacked to be effective. - Adds complexity to network configuration
Topology	Client	<ul style="list-style-type: none"> - Great way to study corrupted server behavior 	<ul style="list-style-type: none"> - High risk of being blacklisted - Requires special permissions - Great way to study client behavior - Single point of failure
	Centralized	<ul style="list-style-type: none"> - Easier to update and secure - Dedicated resources - Efficient topology for small size honeypots - Easy addition or removal of new client nodes 	
Containment	Decentralized	<ul style="list-style-type: none"> - Allows to obtain high availability, even in case of compromise - More autonomy, no single point of failure - Allows for a more diverse and more flexible system 	<ul style="list-style-type: none"> - Not suitable for small systems - Can lead to coordination and synchronization problems - More difficult to manage and configure
	Distributed	<ul style="list-style-type: none"> - High performance (low latency) - Independent failure of component - Easy scalability - High availability 	<ul style="list-style-type: none"> - Requires the establishment of a consensus to agree on the same commands - No global clock (difficulty recording events) - High implementation cost (due to the number of components) - Limits the amount of information collected
Data capturing	Block	<ul style="list-style-type: none"> - Excellent defense - Prevents the attacker from moving through the system and reaching his target 	
	Defuse	<ul style="list-style-type: none"> - Good source of attack strategy data 	<ul style="list-style-type: none"> - The attacker has already reached his target - Jeopardizes the integrity of the system - Risk of compromise - Requires a good forecast of the countermeasures to be applied
Scalability	Slow down	<ul style="list-style-type: none"> - Good source of attack strategy data - Great Deterrent - The attacker consumes his resources unnecessarily 	<ul style="list-style-type: none"> - The attacker consumes the resources of the honeypot by his attempts - Little lack of realism - High risk of compromise - Little lack of realism - Requires special attention in terms of encapsulation - Generates a huge amount of false positives
	None	<ul style="list-style-type: none"> - Limit the number of false positives 	
Context	Event	<ul style="list-style-type: none"> - Good overview of events 	
	Attacks	<ul style="list-style-type: none"> - Good compromise between "Event" and "Intrusion" 	<ul style="list-style-type: none"> - Limits the study of strategies - higher risk of compromise (the attacker already has control of the system)
Scalability	Intrusions	<ul style="list-style-type: none"> - Allows a good study of attack strategies - Allows to highlight security vulnerabilities 	
	None	n/a	<ul style="list-style-type: none"> - Does not capture any information - Requires an external component responsible for capturing information data - Can cause a loss of performance - Requires a specific configuration to remain realistic - Need adequate resources to scale out - Limited to resources and components initially installed
Context	Scalable	<ul style="list-style-type: none"> - Great optimization flexibility - Performance adaptability 	
	Not Scalable	n/a	<ul style="list-style-type: none"> - Not necessarily intended to protect - High risk of compromise due to greater freedom - Requires an efficient logging system
Context	Research	<ul style="list-style-type: none"> - Excellent means for finding vulnerability and threat - Great way to study attack and defense strategy 	<ul style="list-style-type: none"> - Emphasis on the imprisonment of the attacker rather than the study of his behavior, which limits the data collected
	Production	<ul style="list-style-type: none"> - Protects systems more effectively - Precise definition of containment methods 	

Figure 9 - Table of strengths and weaknesses of honeypots characteristics [Characteristics2022]

4.4. Honeynet

The capabilities of a single honeypot are limited to its characteristics. Today, many honeypot systems focus on combining different types of honeypots to achieve an optimized solution.

For Wenjun's Fan [Fan2015], a typical honeynet is "a hybrid system composed of a combination of high-interacting and low-interacting honeypots, which strikes a good balance between scalability, performance, fidelity, and containment. From an attacker's perspective, the honeynet appears to have servers and desktops, many different types of applications, and several different platforms. Therefore, the term "zoo" is also used to name the honey nets, as they allow the capture of the wild hacker in its natural environment".

In his article [Fan2015], proposes a honey network taxonomy that facilitates their classification and study. It is composed of 3 main features:

1. Data control: Set of measures aimed at reducing the risk that a honeypot will be compromised and used to attack other systems. Outbound attacks must be controlled to protect non-honeypot systems
2. Data capture: Its purpose is to capture and record and collect the activities of attackers to be exploited later. 3 critical data capture layers have been identified:
 - o The layer related to incoming and outgoing connections stored in the firewall logs.
 - o The traffic analysis layer, which checks packets and their payloads as they enter and leave the honeynet.
 - o The layer related to activity on the system (system calls, modified files, attacker's keystroke, etc).
3. Data collection: Means necessary and used to guarantee the safe transfer of data collected by honeypots to a centralized point.

4.5. Threat management

To study the behaviour of attackers, it is necessary to make sure that they do not reach the legitimate resources, but the honeypot. The honeypot is placed in a specific IP address range and isolated from the company's resources. All attempts to connect to the honeypot are considered malicious. The honeypot provides services such as SMB, HTTP, SSH, FTP, TFTP, MySQL, MSSQL, SIP, etc, depending on the characteristics defined above. Once the attackers' activities have been collected, this information must be processed to dissect the threats and possible ramifications of the attacks. This collected information can be attack sequences, commands, scripts, and any other malicious payloads. They generate information on threats and thus enrich the organization's risk analyses to mitigate those risks.

4.5.1. Threat Analysis

In his article, Tuma [Tuma2018] tells us that "A threat analysis technique consists of a systematic analysis of the attacker's profile vis-à-vis assets of value to the organization. These activities often take place during the design phase and are repeated later in the product life cycle, if necessary".

Ryandy and Lim [Ryandy2020] propose a first threat analysis cut into 2 categories of artifacts. Network traffic artifacts and payload artifacts.

Network traffic artifacts are key attack information that can be linked to a domain name, an IP address, a URL, and their relationship. Some of these artifacts can be identified as compromise

indicators (IOC) and can then be used for early detection of attack attempts using intrusion detection systems and antivirus software. This key attack information can be linked to a particular service. For example, the ssh service often starts with an authentication process and continues with sending a command or instruction that may involve using a URL to download a load used to exploit the host. In this case, the URL is an indicator of compromise and allows to highlight the importance of network services in the identification of the attacker, his intentions, and his behaviour.

Payload artifacts are parts of a program or malware used to perform various malicious activities to achieve the attacker's objectives. These artifacts can be analysed in 2 different ways. First analysed statically, by quickly analysing the file structure of a program without executing it. The binary code is analysed to determine if the signature is known and if it is malware. Or, analysed dynamically, running the code in a sandbox to discover the behaviour of the program at execution.

All these artifacts provide an easy-to-share knowledge base for cybersecurity organizations. One of the best known is Mitre.

4.5.2. Mitre

MITRE is an American non-profit organization whose objective is to work for the public interest. His fields of intervention are systems engineering, information technology, operational concepts, and business modernization.

MITRE has become a reference in terms of sharing information around cybercrime and its 2 essential platforms are MITRE ATT&CK[®] and MITRE CVE.

MITRE ATT&CK [MitreATT] is an open and freely accessible reference knowledge base, known in the world of computer security, as it provides a list of tactics and techniques used by cybercriminals. It describes the adversary behaviours specific to the different environments (Windows, Linux, Mac, cloud, mobiles, etc.) through different phases: Reconnaissance, resource development, initial access, execution, persistence, privilege escalation, defence evasion, credential access, discovery, lateral Movement, collection, command and control, exfiltration, and impact. Organizations tap into this knowledge base to develop offensive and defensive measures to strengthen their security. ATT&CK[®] can help cyber defence managers better understand attacker behaviour and assess risk to a society by classifying attacks. There is even a knowledge base dedicated to Industrial Control Systems, ATT&CK[®] ICS[MitreCVE].

MITRE CVE [MitreICS] is a public dictionary that records all vulnerabilities and security flaws that have been discovered. CVE identifiers are notably used by IDSs and IPSs.

The use of these 2 sources of information by a honeypot can be interesting insofar as they provide interesting techniques to feed the attacker's kill chain or attack tree (ADT).

Cyber kill chain is a model to describe cyber-attacks so as to develop incident response and analysis capabilities. Cyber kill chain in simple terms is an attack chain, the path that an intruder takes to penetrate information systems over time to execute an attack on the target [Yadav2015].

Attack trees provide a formal and methodical way of describing system security, based on various attacks. Basically, attacks against a system are represented in a tree structure, with the goal as the root node and different ways to achieve that goal as leaf nodes.

4.5.3. Categorization of threats

As part of their research, Ryandy and Lim [Ryandy2020] argue that data collected by honeypot can be processed and analysed, using algorithms to correlate with Mitre's threat scheme to finally categorize the various threats. They installed two well-known honeypots, namely Crowie and Dionaea. Over a two-month period in 2020, they were able to observe 547 million attacks. Of these attacks, only 1% were successful in logging in and executing commands or delivering a payload. Nevertheless, 96.1% were able to connect without executing any commands. They concluded that attackers spend most of their time scanning ports rather than trying to exploit the system.

It is interesting to note that they were able to observe a much higher number of attacks during working days than during the weekend. He suggests that this is a strong indicator that source IP addresses would come from desktop computers used as bots.

These two honeypots emulate a long service list, namely FTP, SMB, EpMapper, HTTP, Memcache, MQTT, MSSQL, MYSQL, PPTP, SIP, TFTP, UPNP. Analysis of the data allowed them to define a taxonomy of threats that are covered by these honeypots. Once correlated with Mitre, they were able to summarize in a table the categories of attacks with the number of occurrences:

Threat Category	Total	Threat Category	Total
Bruteforce	397719	Download tools	667
Profiling hardware	13072	Covering track	371
Profiling system	10054	Removing previously used tools	314
Profiling Linux tools	4471	Security bypass	56
Execution of tools	3291	Setup/ Modify env PATH	40
Profiling file system	2742	Leaving mark/ Narcissm	28
Enumerating task/ process	2615	Copy of file	25
Enumeration login user	2614	Linux tool execution	15
SSH account config	2614	Setup persistence on boot	8
Privilege modification	2614	Silent run of tools	4

Figure 10 - Threat Categorization & Frequency Statistics [Ryandy2020]

In the case of Cowrie, more focused on artifacts and networks, the categorization of these threats could be done through the recovery of the various commands entered by the attackers:

SCS005 – System Profiling & Persistence			
No	Commands	Threat Categories	MITRE Techniques
1	cat /proc/cpuinfo grep name wc -l	Profiling hardware	T1082
2	echo \root:AcpF15CDkgMb\ chpasswd bash	Privilege modification	T1098
3	cat /proc/cpuinfo grep name head -n 1 awk '{print \$4,\$5,\$6,\$7,\$8,\$9;}'	Profiling hardware	T1082
4	free -m grep Mem awk '{print \$2 , \$3, \$4, \$5, \$6, \$7}'	Profiling hardware	T1082
5	ls -lh \$(which ls),	Profiling file system	T1518, T1083
6	which ls,	Profiling Linux tools	T1083
7	crontab -l,	Linux tool execution	T1204
8	w,	Enumeration Login user	T1033
9	uname -m,	Profiling system	T1082, T1518
10	cat /proc/cpuinfo grep model grep name	Profiling hardware	T1082
11	top,	Enumerating task	T1057
12	uname,	Profiling system	T1082, T1518
13	uname -a,	Profiling system	T1082, T1518
14	lscpu grep Model,	Profiling hardware	T1082
15	cd ~&& rm -rf .ssh && mkdir .ssh && echo \ssh-rsa	SSH account config	T1098

Figure 11 - Cowrie command & MITRE categorization [Ryandy2020]

When combined with the MITRE techniques, these commands highlighted three categories of attacks, namely:

- SCS005 (System Profiling & Persistence)
- SCS029 (System Profiling)
- SCS007 (Shell, Tool Exec. & System Profiling)

In the case of Dionaea, more focused on payload artifacts, they were able to observe the use of malware of which here are the top 7:

- TrojanDownloader: Win32/Small (183)
- Ransom: Win32/CVE-2017-0147.A (121)
- TrojanDownloader: Win32/ZombieBoy.A!bit (16)
- Trojan: Win32/Occamy.CBO (12)
- Trojan: Win32/Tiggrelr.fn (8)
- Ransom: Win32/WannaCrypt.A!rsm (6)
- Trojan: Win32/Eqtonex.F!rfn (2)

They found that most of the malware encountered is ransomware or trojans/ TrojanDownloader.

4.6. Honeypot solutions references

There are as so many honeypots as there are services and type of application. It would therefore be difficult to create an exhaustive list, but here is one that includes some of the most popular honeypot tools for SSH, HTTP and IoT services, but also an all-in-one solution

SSH honeypots:

- Kippo [Kippo]: This SSH honeypot written in Python has been designed to detect and log brute force attacks and, most importantly, the complete shell history performed by the attacker. It's available for most modern Linux distros, and offers both cli-command management and configuration, as well as web-based interface. Kippo offers a fake file system and the ability to offer fake content to attackers (such as user password files, etc.), as well as a powerful statistics system called Kippo Graph.
- Cowrie [Cowrie]: This medium interaction SSH honeypot works by emulating a shell. It offers a fake file system based on Debian 5.0, letting you add and remove files as you wish. This application also saves all the downloaded and uploaded files in a secure and quarantined area, so you can perform later analysis if needed. Apart from the SSH emulated shell, it can be used as an SSH and Telnet proxy and allows you to forward SMTP connections to another SMTP honeypot. Coupled with an event-driven networking engine like Twisted [Twisted], it helps provide a bespoke environment.

HTTP honeypots

- Glastopf [Glastopf]: This HTTP-based honeypot lets you detect web-application attacks effectively. Written in Python, Glastopf can emulate several types of vulnerabilities, including local and remote file insertion as well as SQL Injection (SQLi) and using a centralized logging system with HPFeeds.
- Nodepot [Nodepot]: This web-app honeypot is focused on Node.js, and even lets you run it in limited hardware such as Raspberry Pi / Cubietruck. If you're running a Node.js app and are looking to get valuable information about incoming attacks and discover how vulnerable you are, then this is one of the most relevant honeypots for you. Available on most modern Linux distros, running it depends on only a few requirements. –
- Google Hack Honeygot [GHHoneygot]: Commonly known as GHH, this honeypot emulates a vulnerable web app that can be indexed by web crawlers but remains hidden from direct browser requests. The transparent link used for this purpose reduces false positives and prevents the honeypot from being detected. This lets you test your app against ever-so-popular Google dorks. GHH offers an easy configuration file, as well some nice logging capabilities for getting critical attacker information such as IP, user agent and other header details.

IoT honeypots:

- HoneyThing [HoneyThing]: Created for the Internet of TR-069 enabled services, this honeypot works by acting as a full modem/router running the RomPager web server and supports TR-069 (CWMP) protocol. This IoT honeypot is capable of emulating popular vulnerabilities for Rom-0, Misfortune Cookie, RomPager and more. It offers support for TR-069 protocol, including most of its popular CPE commands such as GetRPCMethods, Get/Set parameter values, Download, etc. Unlike others, this honeypot offers an easy and polished web-based interface. Finally, all the critical data is logged and stored in a secured part.
- Kako [Kako]: The default config will run several service simulations to capture attack information from all incoming requests, including the full body. It includes Telnet, HTTP and HTTPS servers. Kako requires the following Python packages to work properly: Click, Boto3, Requests and Cerberus. Once you're covered with the required packages, you can configure this IoT honeypot by using a simple YAML file called kako.yaml. All the data is recorded and is exported into AWS SNS, and flat-file JSON format.

All in one honeypot

- T-Pot [Tpot]: Since 2016, Telekom-security has been offering the community, under the GPL 3.0 license, an all-in-one multi-arch (amd64, arm64) honeypot platform, "The All In One Multi Honeypot Platform" can be optionally distributed and supports over 20 honeypots as well as countless viewing options using the elastic stack but also live animated attack maps and also plenty of security tools to further enhance the deception experience.

A reference list of honeypots

- Awesome Honeypots [AwesomeHP]: Going further, there is a curated list of honeypots, along with related components. This list is divided into categories such as web, services, and others, with a focus on free and open-source projects.

5. Proof of concept

5.1. Context

As part of the SPARTA [SPARTA] project and the integration of security solutions into a DevSecOps pipeline, CETIC worked on a case based on the operation of a platoon of connected and autonomous vehicles using ROS [ROS].

The objective of the use case of this work, is to demonstrate the value of integrating a honeypot solution into an Edge Computing environment while promoting the value of continuous deployment, integration, and security in the DevSecOps pipeline.

5.1.1. Autonomous vehicles

Autonomous vehicles are primarily computer systems consisting of an operating system, software, a variety of sensors and components generating a large amount of data to be processed quickly as necessary for decision making [Jo2014].

These vehicles can connect to both the other vehicles in the platoon, but also to a more global management system. These connections are made through Edge Computing to improve security and efficiency, but also to reduce accidents and congestion. Nevertheless, these different connections use unreliable communication protocols and have security vulnerabilities.

Edge Computing includes compute, storage, data management, data analysis and networking technologies, and enables real-time data processing, ensuring vehicles react instantly to collected data.

5.1.2. Challenges

CETIC has demonstrated, using a test bench deployed in a DevSecOps lifecycle, that exploitable threats exist and that attacks can lead to disasters if these threats are not addressed. To counter these threats, CETIC researchers have integrated IDS, SIEM and SOAR solutions into the DevSecOps lifecycle. They were also able to demonstrate that the integration of security solutions makes it possible to put in place countermeasures to the dangers of compromises.

Even though these security solutions have undeniable interests in countering these threats, they serve primarily to protect systems from known techniques and vulnerabilities.

It is therefore interesting to ask how to discover new security flaws, techniques, and vulnerabilities without this taking place in a system in production and in order to limit the risks related to computer systems and cybersecurity. The solution lies in integrating a honeypot into the DevSecOps pipeline.

5.1.3. Scenario

The attack scenario defined by CETIC is based on the construction of an ADT.

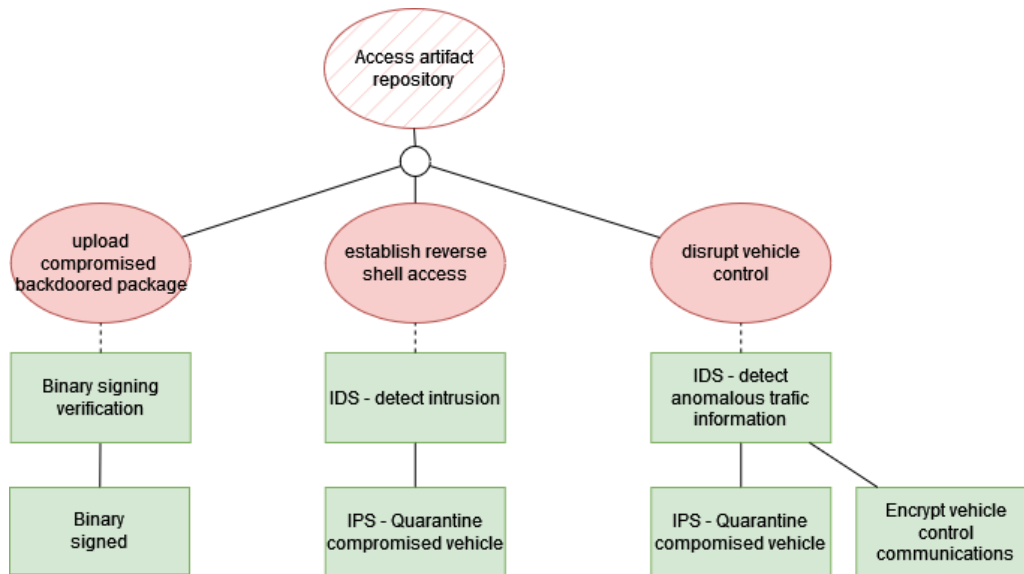


Figure 12 - Attack-Defence Trees scenario [Diagram2022]

It describes injecting a backdoor into a vehicle's firewall via an update from the project repository.

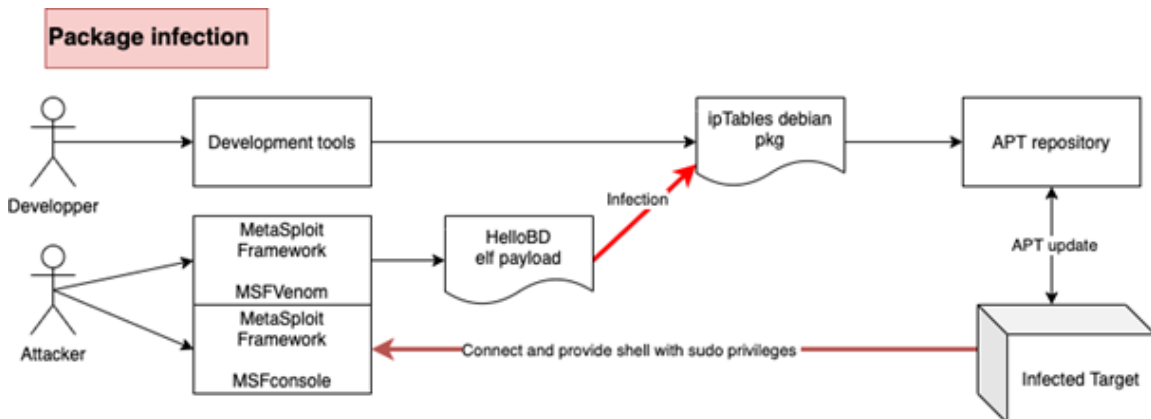


Figure 13 - Backdoor attack scenario [Ginis2021]

After discovering a vulnerability in the firewall, the attacker will exploit it using MSFVenom [MSFvenom]. It will inject a payload via a HelloBD file into the legitimate Ubuntu package, IpTables [Iptables]. This package contains the backdoor that will force the vehicle to connect using a reverse shell to the attacker's Metasploit console. It will then be uploaded to the repository containing the project code. The vehicle will then update its firewall to the new version of IpTable and install the malicious binary. Once installed, the victim vehicle will connect to the attacker's console and provide him with administrator access rights (sudo). The attacker then has his hand on the vehicle and can control it as he pleases.

```
    =[ metasploit v6.0.43-dev ]
+ -- --=[ 2129 exploits - 1139 auxiliary - 363 post ]
+ -- --=[ 592 payloads - 45 encoders - 10 nops ]
+ -- --=[ 8 evasion ]

Metasploit tip: When in a module, use back to go
back to the top level prompt

msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload linux/x64/shell_reverse_tcp
payload => linux/x64/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.134.2.100
LHOST => 10.134.2.100
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.134.2.100:4444
[*] Command shell session 1 opened (10.134.2.100:4444 -> 10.134.2.95:56702) at 2021-07-23 10:49:14 +0200

ls
bin
boot
cdrom
dev
```

Figure 14 - Backdoor attack demonstration with MetaSploit [Ginis2021]

5.1.4. Integration of a honeypot

The integration of security tools (IDS, SIEM, SOAR) has been carried out in previous internships [Benchrifia] [Amraoui]. The detection and analysis steps have been implemented so that an initial response to this vulnerability is given. Indeed, the existing response allows downgrading the firewall of the target vehicle to return to a reliable version.

This work consists of adding an additional layer of security by adding a honeypot imitating as much as possible the behaviour of an infected vehicle in production. The intention is to add a response from SOAR, which will communicate with a sandbox environment to deploy a dummy vehicle replicating the infected one but sandboxed and monitored.

Here is a diagram representing the structure integrating a honeypot:

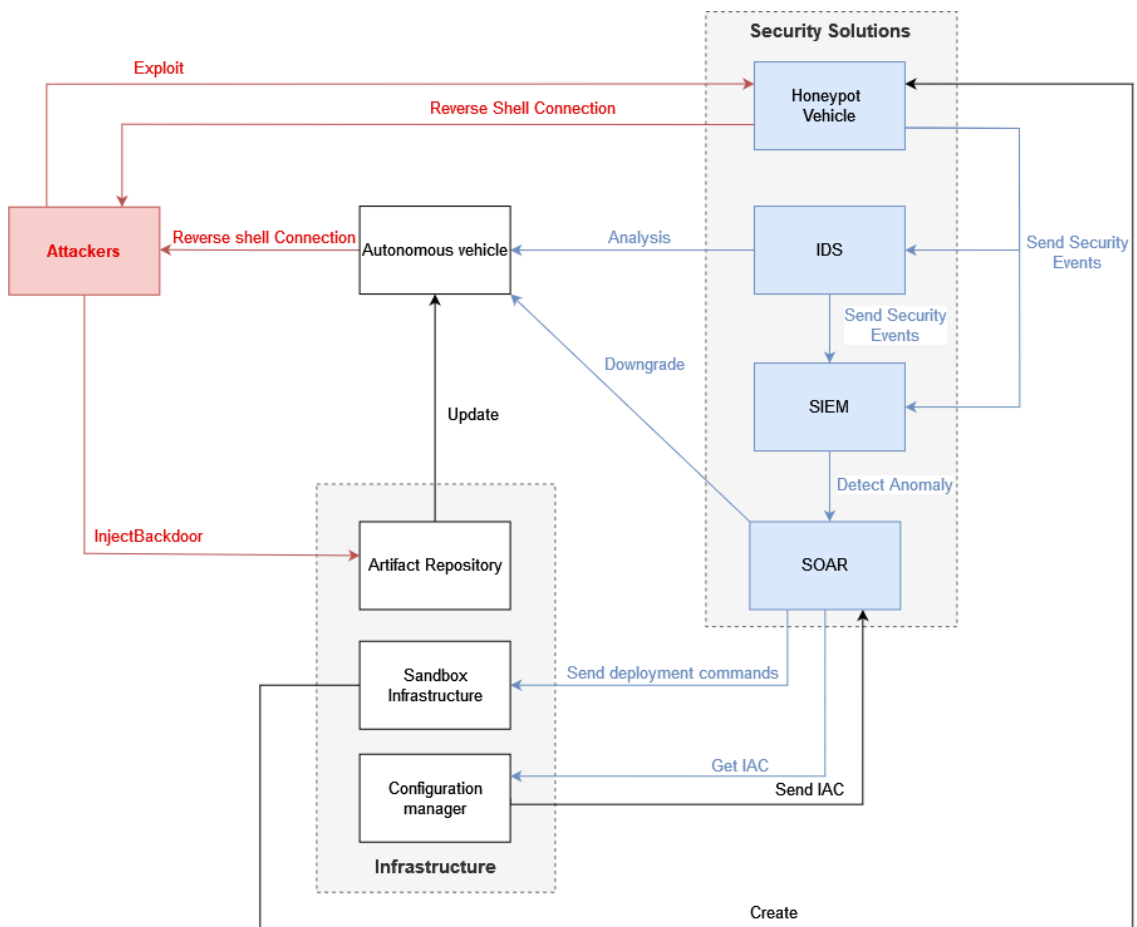


Figure 15 - Structure of the use case integrating a honeypot [Diagram2022]

5.2. Implementation in a use case

Several technical aspects and requirements must be taken into consideration to respect the infrastructure and solutions implemented in previous implementations around the SPARTA project.

As a result, constraints have been set on the tools used to build and deploy the DevSecOps pipeline. Then, a detailed architecture allows to understand the different components and tools used and the interactions between them.

5.2.1. Functional specification

To determine a solution that can be integrated into the DevSecOps pipeline and with features specific to Edge Computing and in this case meeting the requirements of the use case, we have determined selection criteria both for the infrastructure to be deployed, but also for the honeypot used.

From a functional perspective, the infrastructure must:

- Use DevSecOps tools that meet the criteria of continuous development, deployment, and integration:
 - Use of containers
 - Use of containers images
 - Using orchestration and replication tools
 - Using versioning tools
 - Use of monitoring tools
- Enable centralized management of infrastructure components
- Provide management interface and command line access

Also, from a functional point of view, the honeypot must be able to:

- Capture the commands typed by the attacker
- Record and/or transmit the collected data to an IDS
- Be deployed by SOAR

From a non-functional perspective, the tools used for infrastructure implementation and honeypot must:

- Open-source licenses
- Active community
- Extensibility
- Well Documented
- Easy to install and take in hand
- Easy to maintain
- Sustainability

5.2.2. Technical Specification

From a technical point of view, the specification of the infrastructure and the choice of technologies are based on the requirements of a deployment in a DevSecOps pipeline. For each of the components used, the technological choice was detailed and argued.

The honeypot must be able to integrate into the DevSecOps pipeline. By this, we mean that it is necessary to take into consideration its ability to be technically compatible with orchestration tools and mechanisms.

As explained above, one of the objectives of the project is to use exclusively open-source solutions, possessing large communities and sufficiently documented to guarantee a certain sustainability, scalability, and maintainability level of the tools.

5.2.2.1. Technologies and constraints

In this case, decisions at the level of technological solutions had to be made to carry out the proof of concept. Some of these solutions have been discarded for ease of deployment, research, but also because equivalent solutions existed intrinsically in the higher-level solutions.

Infrastructure

Infrastructure is a key element in building a DevSecOps pipeline. Their primary role is to enable the Ops Operations Team to manage the deployment and continuous integration aspects. To do this, we use orchestration mechanisms to manage solutions deployed on the cloud and Edge.

Figure 15 illustrates the infrastructure. Several components can be distinguished:

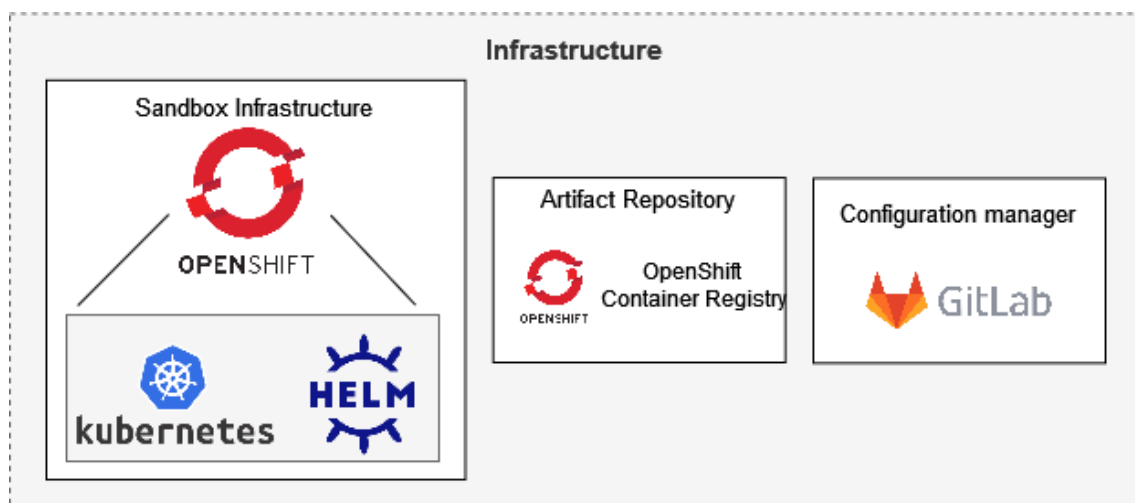


Figure 16 - Infrastructure technologies [Diagram2022]

First, the sandbox infrastructure, which will provide a virtual environment and host the various containers used. For this use case, we will use Red Hat OpenShift and more particularly its version usable on a local machine, namely MiniShift. This version allows to run a containerized single node cluster to be able to develop for Kubernetes solutions.

OpenShift is a Kubernetes distribution provided by Red Hat and is a platform as a service (PaaS), which automates the deployment, scaling, and management of containerized applications.

It is also composed of Helm [HELM], which automates the creation, packaging, configuration and deployment of applications and services on Kubernetes clusters using YAML files for manifest maintenance.

As shown in Figure 15, the artifact repository is also discussed. This allows to provide a repository to the infrastructure on which are stored the images that will allow to build the containers used. Initially, it was planned to use Nexus Repository [Nexus], already used by CETIC, but OpenShift has a repository component integrated and its uses are more appropriate in this use case as it avoids the use of an external component.

Finally, the last component used by the infrastructure layer is the configuration manager. It aims to host and provide access to the different configurations needed for container deployment, via a versioning tool. CETIC currently uses GitLab [GitLab], in its community version, for configuration management.

Autonomous vehicles

The autonomous vehicles used in the SPARTA project are deployed under ROS 1, in its latest version: Noetic. ROS1 exists since 2007 and the multiple technological evolutions related to robotics have led to many versions and updates, making ROS 1 quite unstable for more complex tasks [ROS1vs2].

In order to address the lack of security of ROS1 and the absence of some industry-related requirements, such as real-time, security and certification, has led developers to create a new version of ROS. ROS version 2. It has therefore been completely developed from scratch and allows to meet the requirements of compatibility with industrial applications, but also to use newer versions of packages and components.

As the ROS1 Noetic EOL is announced for 2025, the use of ROS 2 in this use case is part of the requirements of the requirements of this case study.



Figure 17 - Autonomous vehicle technology [Diagram2022]

ROS, in its version 1 and 2, has the particularity of working with nodes (different from the nodes used in OpenShift). Each node is composed of a service/ an application and the communication between the nodes is done by means of message.

One of the main architectural differences between ROS1 and ROS2 is the functioning of these nodes, since ROS1 requires the definition of a “Master” node and “slave” nodes, while ROS2 uses the concept of publisher and subscriber. The concept is relatively similar [ROS1vs2], but the main difference lies in the flexibility of relations between publishers and subscribers.

[19] <https://roboticsbackend.com/what-is-a-ros-topic/>

The principle of message exchange is that a publisher sends a message through a topic (channel) that will distribute this message to the different subscriber using port “11311”:

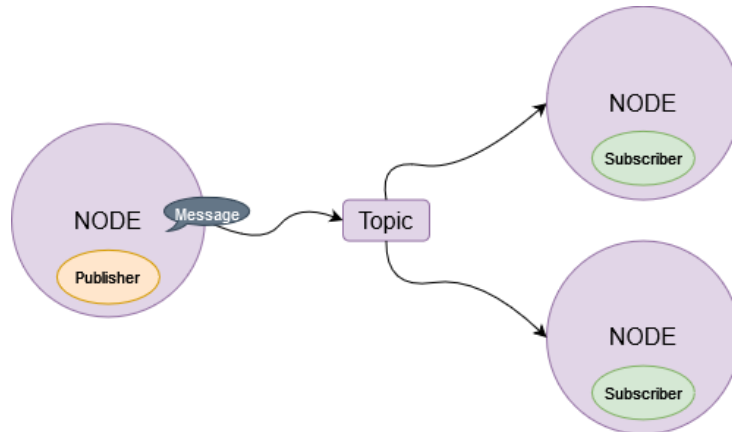


Figure 18 - ROS message send by a publisher illustration diagram [Diagram2022]

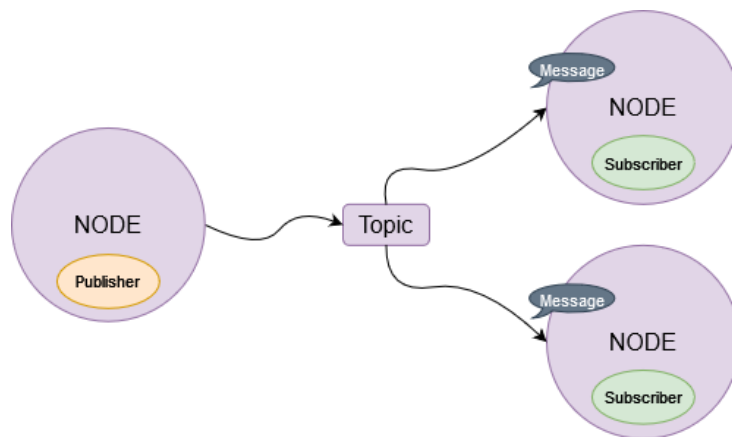


Figure 19 - ROS message received by subscribers illustration diagram [Diagram2022]

In this use case and to get as close as possible to the reality of a context in production, the autonomous vehicle consists of a publisher node and a subscriber node. Message exchanges between vehicles will be illustrated during the deployment phase.

Security Solutions – Blue Team

The term “Blue team” describes the team that uses their skills to defend a system. It focuses on identifying potential vulnerabilities and its purpose is to improve defence mechanisms. She is aware of the defences already in place and is continuously involved in the analysis of suspicious activity.

In this use case, the blue team has several roles to play since it must:

- Detect the reverse shell connection between the autonomous vehicle and the attacker.
- Respond to this security event by:
 - Disconnecting the attacker’s autonomous vehicle and downgrading it to a stable version.
 - Deploying a honeypot, identical to the autonomous vehicle, capable of connecting using the reverse shell to the attacker.
 - Analyse network artifact data and data collected by the honeypot.

In short, the IDS aims to detect security anomalies. SIEM's objective is to analyse and identify threats. Finally, SOAR makes it possible to orchestrate automated responses to these threats. Although these systems have shown their effectiveness, they have not been able to detect a connection made in reverse shell inside an infrastructure using containers in a first approach.

One of the constraints is therefore to integrate a solution capable of detecting and identifying (not as well as a real SIEM), to threats within a deployment and continuous integration infrastructure. This will be done using Falco security [Falco].

Falco security is a native cloud execution security project and is a threat detection engine that integrates well with Kubernetes. It can detect threats during execution by observing the behaviour of applications and containers in cloud and Edge Computing environments. It acts as a security camera detecting unexpected behaviours, intrusions, and real-time data theft.

Falco in the version available for deployment under OpenShift, provides 3 components [Roy2021]:

1. Falco [Falco]: the core framework where policy rules are configured into.
2. Falco Sidekick [FalcoSidekick]: a daemon that extends several possible Falco outputs to interface with external response engines.
3. Falco Sidekick UI [FalcoSidekick-UI]: a simple web UI where all those container security events are displayed in a dashboard.

Specifically, Falco uses system calls to secure and monitor this system by:

- Analysing Linux kernel during execution.
- Affirming the flow in relation to a powerful rule engine.
- Alerting when a rule is violated.

As part of this use case, Falco will also play the role of SIEM since it will identify the backdoor as a threat thanks to its rule engine. He will then send an alert to the OpenShift system that will simulate the role of SOAR and deploy the honeypot.



Figure 20 - Security technologies [Diagram2022]

Honeypot

The honeypot is the cornerstone of this use case. Its role is mainly to be a decoy capable of imitating the behaviour of an autonomous vehicle to deceive the attacker and recover as much information as possible about its operating methodology.

To enable honeypot to play this role, it is essential to define its technical specification. To do this, it must be classified using the characteristics defined in the state of the art (Chapter 4.2):

- Interaction Level: High Interaction

In this use case, the attacker is already well advanced in his attack tree, because the vehicle has already been infected through the repository and the backdoor is open. This means that it is not a BOT, but rather an attack targeted by a decided attacker.

It is necessary that the honeypot is of “high-interaction”. First to avoid that it is identified too quickly by the attacker as being a honeypot. Second, because the specificity of the autonomous car, using ROS, would require the creation of a low interaction honeypot with custom-made environment specific to the ROS functions and commands.

- Resource level: Hybrid

To justify the honeypot’s level of credibility, it must be composed of both a publisher and a subscriber. These components are real since they come from the artifact repository and created based on the configuration of the autonomous vehicle present on the configuration manager. This can therefore be considered a "real" resource level.

Nevertheless, to remain credible, the honeypot must be composed of different sensors and actuators allowing decision making. These sensors are virtual and simulated by the honeypot. For this reason, we can consider this one as having a virtual resource level.

The honeypot being made of real and virtual elements, its characteristic of the resource level is “hybrid”.

- Communication interface: TCP/IP, UDP, reverse shell

The main protocol used for communication with the honeypot is TCP/IP. It is used for the reverse shell.

In addition, the honeypot must be able to simulate the communication between the publisher and the listener, as an autonomous car would do. The default protocol used by ROS is the TCPCROS on the 11311 port.

- Multi-tier architecture: Server & Client

The multi-level architecture has the particularity of being both client and server. Indeed, the honeypot is client, because it connects in reverse shell to its attacker, but also server, since once the connection is initiated, it passively awaits for the exploitation of the attacker and records its actions.

- Topology: Centralized

In this use case, only one honeypot is deployed and no communication with another honeypot exists. The topology is therefore centralized.

- Containment: None

Although actions are taken by SOAR to downgrade the infected vehicle and cut the connection with the attacker, it is not the responsibility of the honeypot.

The honeypot is configured to give the attacker “carte blanche”. It is for this reason that it is deployed in a cluster different from that of the autonomous vehicle.

- Observability: Intrusions

The IDS, Falco, plays the observer role and it is at the origin of the discovery of backdoor and the deployment of the honeypot. Nevertheless, its role is mainly to observe the incoming and outgoing communications and system calls of the containers present in the infrastructure.

The honeypot must be able to retrieve the attacker's keystrokes, which are not systematically observed by Falco. As a result, at least one local sensor must be deployed on the honeypot, resulting in its observability at "intrusion".

- Scalability: Scalable

Being deployed on OpenShift and thus in an environment of continuous deployment and continuous integration, the container is intrinsically scalable since it will adapt the available resources according to needs. In addition, the honeypot, as such, will adapt and update based on the image of the autonomous vehicle. It is therefore defined as "scalable".

- Context: Research & Protection

The honeypot can be defined by the 2 characteristics of the context. By the characteristic of "protection", then that it aims to protect the autonomous vehicle and deploys to play the roles of decoy and shield, but also by the characteristics of "research" since it must allow the study of the attacker's exploitation strategies.

Other technical specialties and constraints are highlighted to allow the implementation of the chosen scenario for the proof of concept. The honeypot must:

- Allow the use of a "payload" artifact to initialize the reverse shell connection.
- Be deployed in a container that can be used under OpenShift.
- Allow the use of a YAML file for its deployment and configuration.
- Allow the use of a recent version of python for script execution.
- Be able to mimic the behaviour of an autonomous vehicle deployed under ROS2, namely the use of a publisher and a subscriber.
- Ability to capture bash commands typed by an attacker

Due to the technical specificity mentioned above and more particularly to the established classification, the honeypot is a true copy of the autonomous vehicle, namely that it is based on a ROS system to which the possibility of exporting the keyboards of the attacker.



Figure 21 - Honeypot technology [Diagram2022]

Attacker – Red Team

The term “Red Team” describes the team that imitates attackers using their techniques. It is focused on the "Pentest & Ethical Hacking". Their goal is to detect, prevent and eliminate vulnerabilities. To do this, it imitates the role of an attacker by finding exploitable backdoors and vulnerabilities.

Kali Linux is a GNU/Linux distribution based on Debian which aims to provide a set of tools necessary for the security tests of an information system, in particular the intrusion test.

The attacker is represented in the use case by a virtual machine on which a Kali Linux is deployed. On this Kali linux, is installed the essential elements for the creation and deployment of the backdoor, namely MetaSploit, and more particularly MSFVenom for the creation of the Payload, and MSFConsole that provides the attacker with reverse shell access with super user privileges.

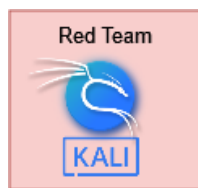


Figure 22 - Red Team technologies [Diagram2022]

Technologies conclusion

To represent the different technologies used and to better understand how they are organized, here is the complete diagram:

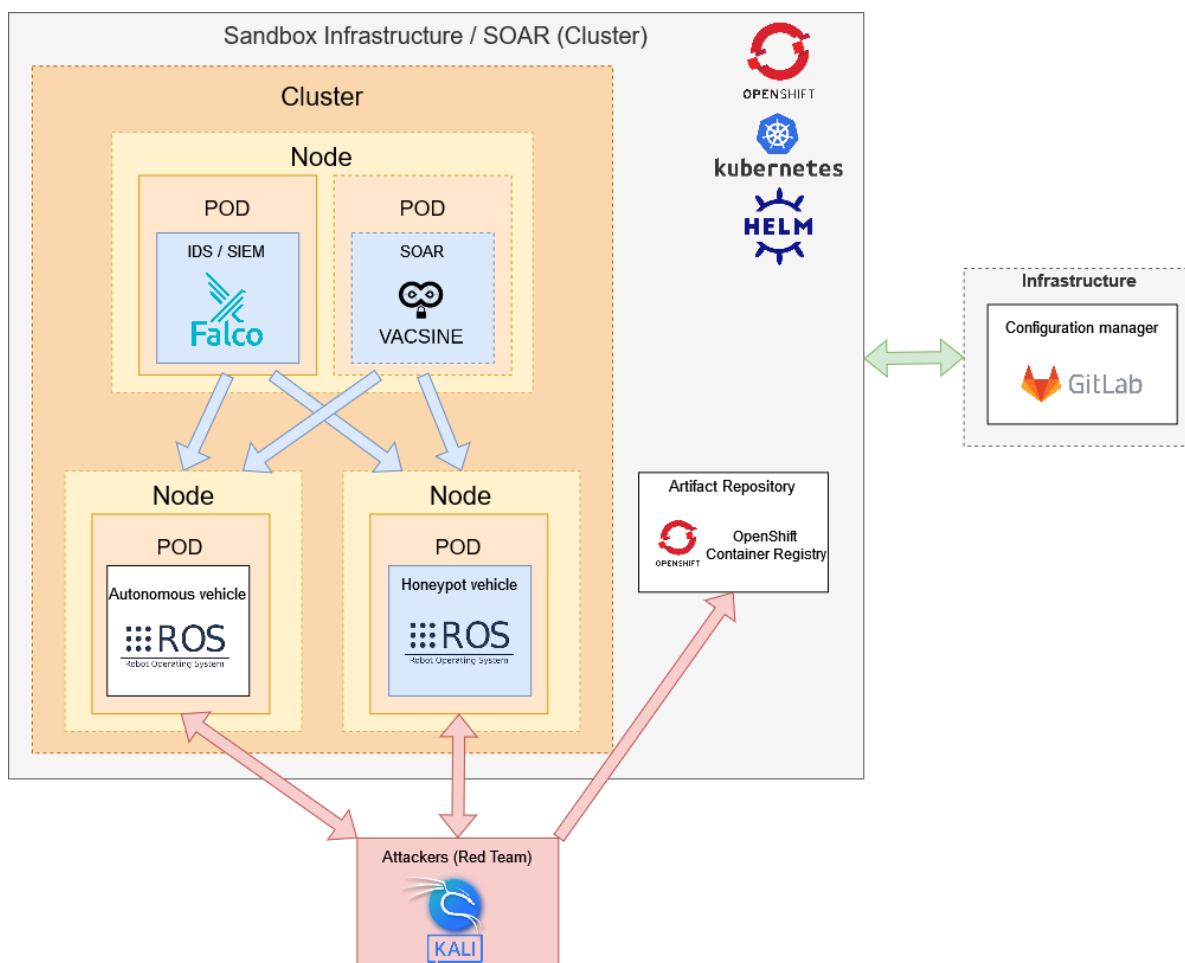


Figure 23 - Technologies architecture [Diagram2022]

OpenShift uses HELM and Kubernetes to deploy, organize and orchestrate the different containers used. The containers of each application are deployed in basic Kubernetes application execution units called “POD”. Thanks to the Kubernetes architecture, pods and containers natively can communicate with each other if they are in the same cluster, even if it is in separate nodes [Palmer].

If we want to allow access to certain applications from the outside, we just need to create services and expose the ports we want to use by routes.

For example, in this use case, it will be necessary to open the TCP port 2802, corresponding to the http service of the “falco-falcosidekick-ui” web application. Simply create a route pointing to this service to allow access to the Falco dashboard from a browser outside the cluster.

These technical aspects will be detailed in the “Deployment” section.

5.2.2.2. Solution Architecture

To describe the architecture of the proposed system, an architecture view model “4+1” [4+1Archi] is used, based on the use of multiple simultaneous views.

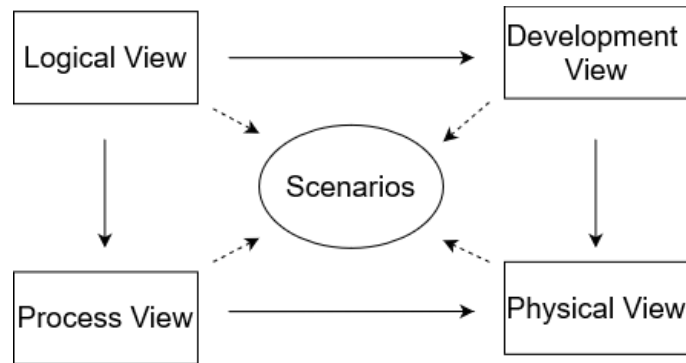


Figure 24 - 4+1 Architecture diagram [Diagram2022]

This model consists of a logical view, a process view, a development view, a physical view, and a use case diagram.

Logical view: Class diagram

The class diagram provides a more structural view of the architecture of the deployed system:

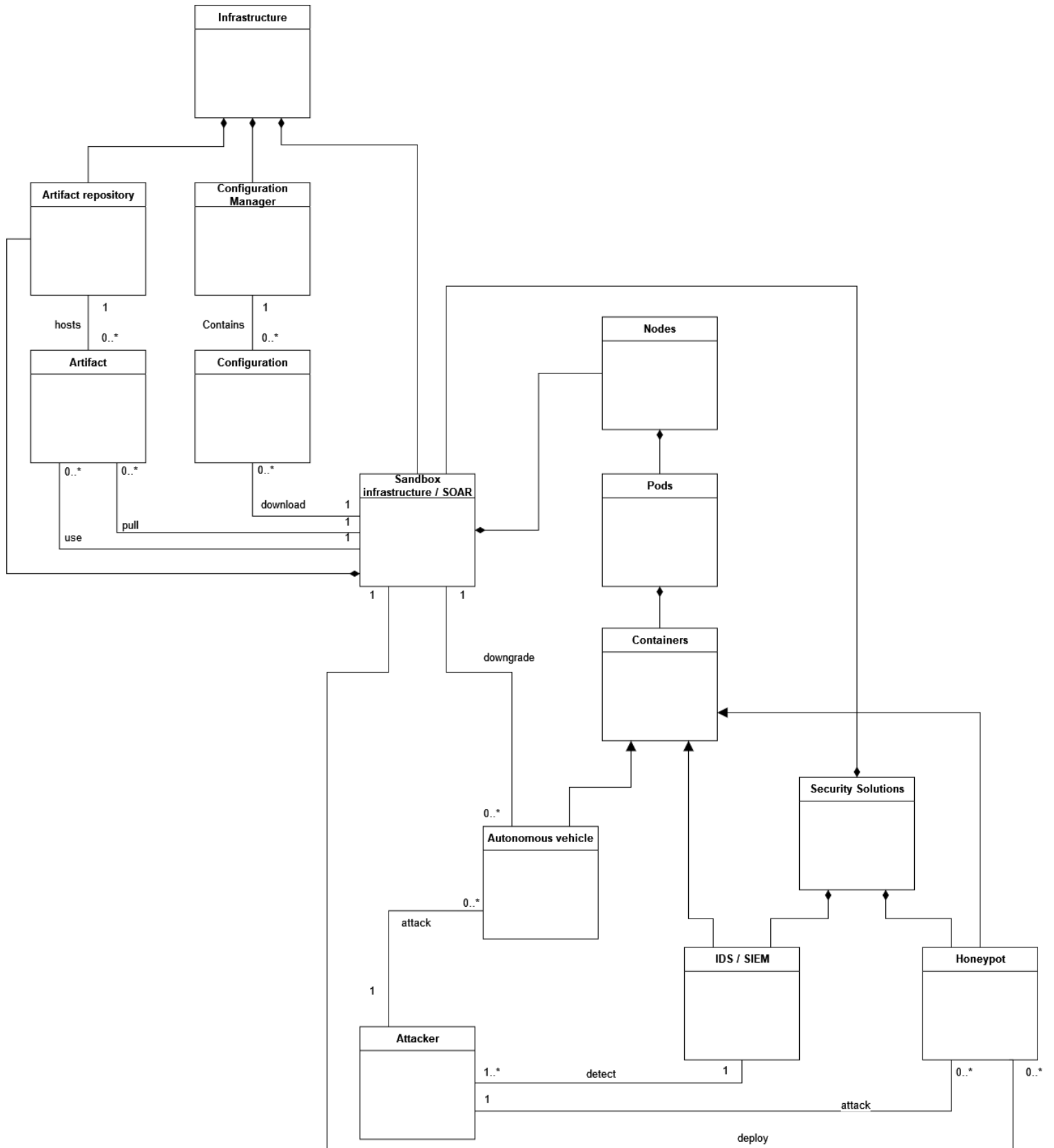


Figure 25 - Logical view: Class diagram [Diagram2022]

Process View: Sequence diagram

To understand step by step the operation from the attack to the response, here is a sequence diagram:

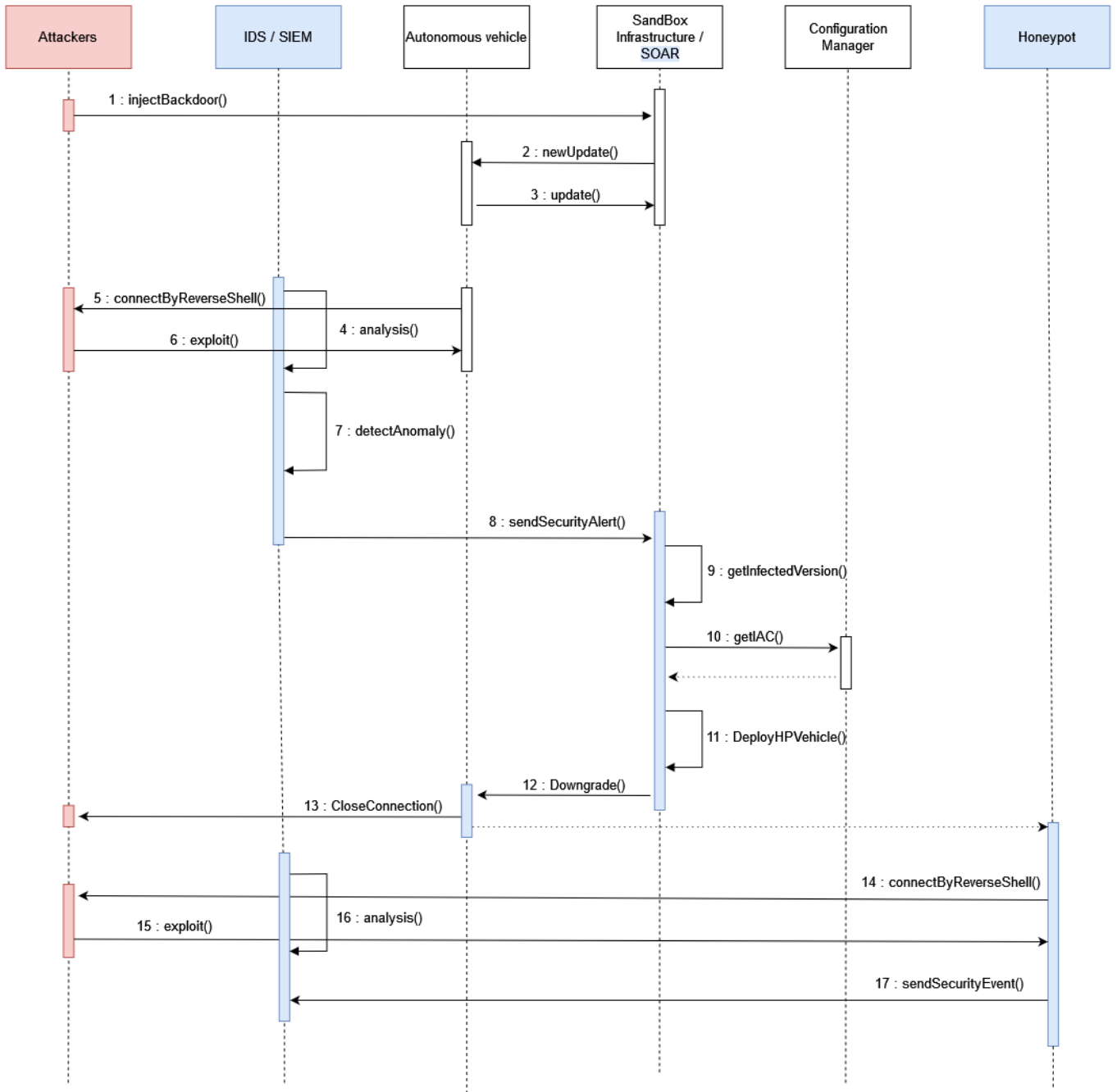


Figure 26 - Process View: Sequence diagram [Diagram2022]

The steps shown in this diagram are:

1. The attacker injects a backdoor into the application repository by modifying IpTable.
2. Sandbox infrastructure informs the autonomous vehicle that an update is available on the repository.
3. Autonomous vehicle downloads the new update containing the payload and backdoor.
4. IDS analyses the traffic of the autonomous vehicle.
5. Autonomous vehicle connects to the attacker via reverse shell.
6. Attacker exploits the autonomous vehicle.
7. IDS detects an event, the SIEM analyses events and detects an anomaly.
8. SIEM sends an alert to SOAR to address the vulnerability.
9. SOAR retrieves the infected version from the artifact repository.
10. SOAR retrieves the configuration (IAC) of infected vehicles from the configuration manager
11. SOAR/Sandbox infrastructure executes the honeypot vehicle creation command, and it is created and deployed.
12. SOAR/Sandbox infrastructure downgrades the infected vehicles.
13. Infected vehicles close connection to the attacker
14. Honeypot vehicle mimics the behaviour of an autonomous vehicle and connects to the attacker via the reverse Shell.
15. Attacker exploits the honeypot vehicle.
16. / 17. Events and artifacts (network traffic and payloads) are collected and analysed by the IDS/SIEM

Development View: Components diagram

The component diagram illustrates the architecture of the different components used in this use case:

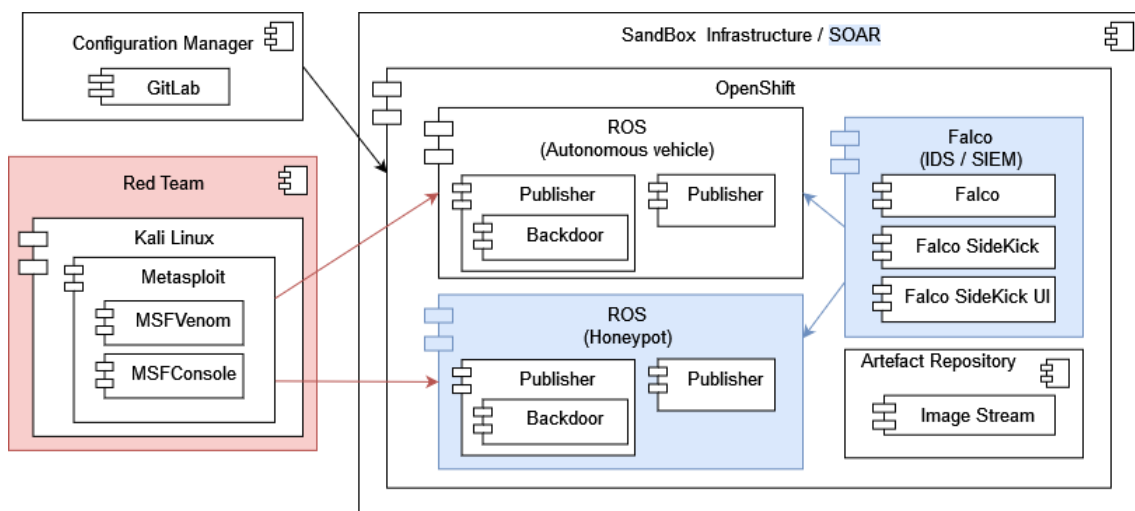


Figure 27 - Development View: Components diagram [Diagram2022]

Physical View: Deployment diagram

The deployment diagram shows the configuration of the different execution processing nodes and the components that reside there.

It models the physical aspects and the static deployment view of the system.

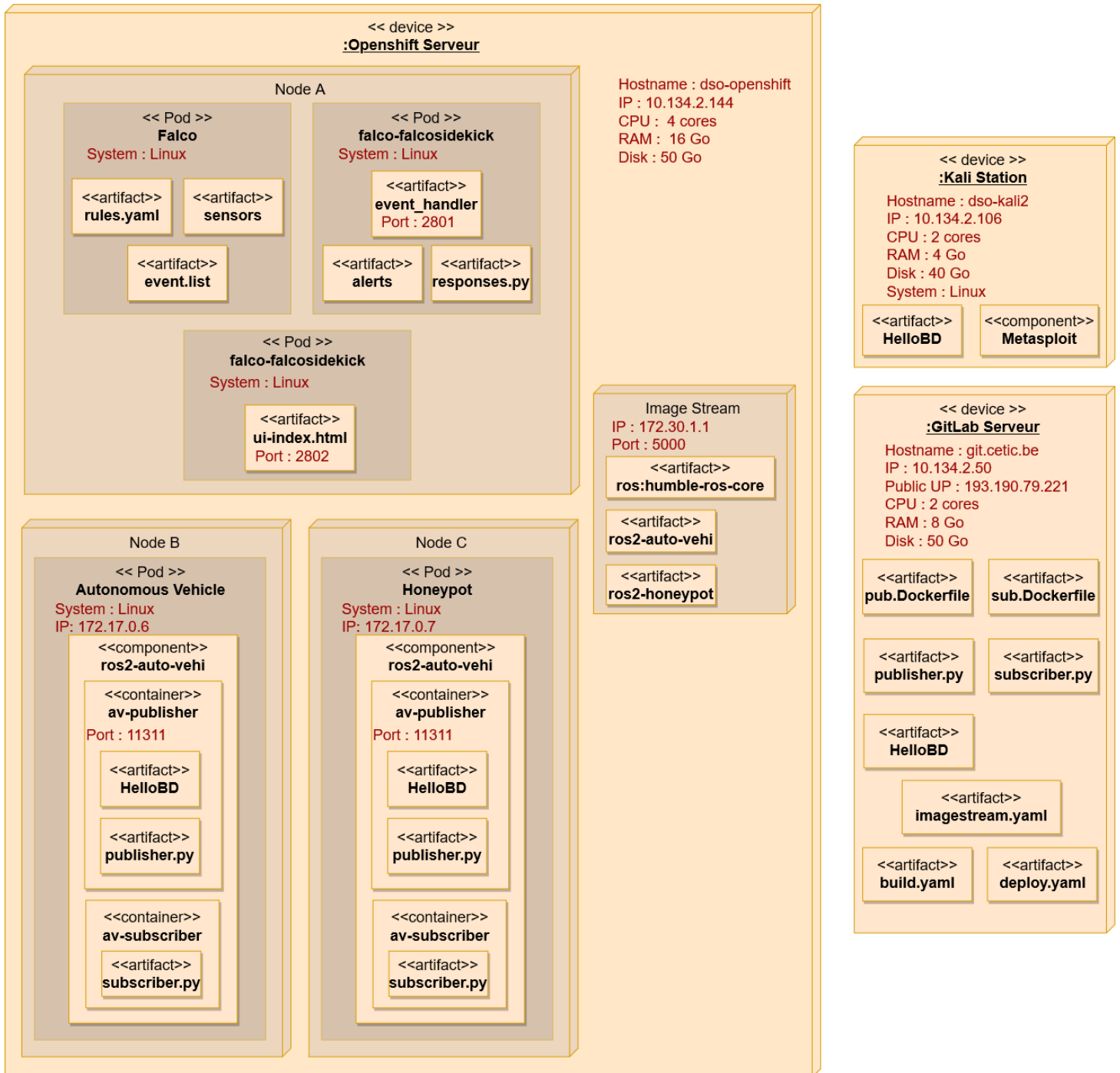


Figure 28 - Physical View: Deployment diagram [Diagram2022]

Scenario: Uses Cases view

The use case view describes the different scenarios and actors' identifiers in this use case:

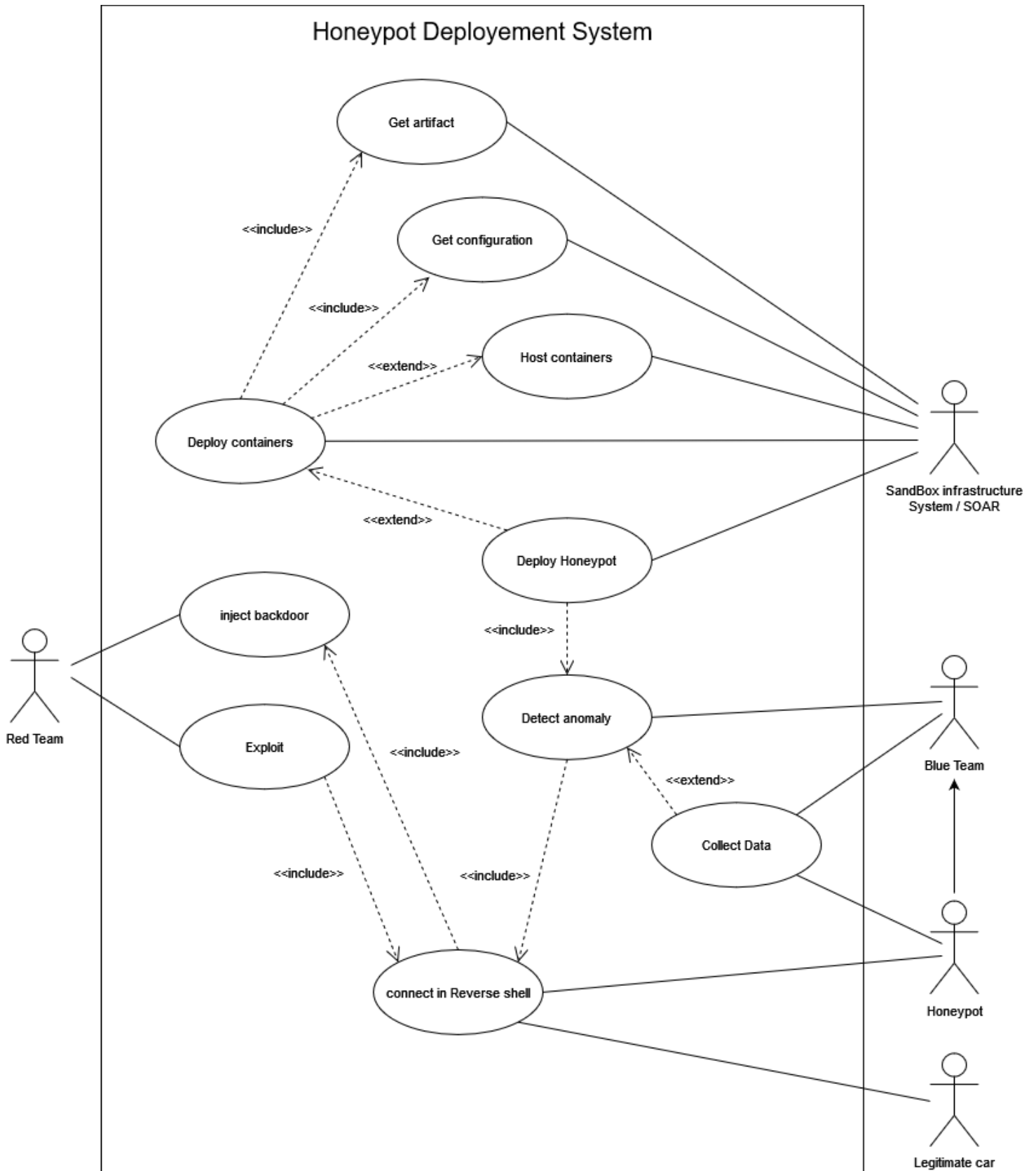


Figure 29 - Scenario: Uses Cases view [Diagram2022]

5.2.3. Deployment

5.2.3.1. Install OpenShift

Installing OpenShift, a MiniShift version, is relatively standard and was made in accordance with the architecture defined in the technical specifications. Its deployment was done in accordance with the instructions and directives of the official documentation. OpenShift can be deployed under Linux, Windows, Mac and in a virtual machine with Virtual Box. As the steps and details of installations are specific to each of these systems, they will not be detailed here.

5.2.3.2. Falco and Response Engine deployment

Falco, FalcoSidekick, FalcoSidekick-Ui

As explained above, Falco consists of 3 components that will be installed:

- Falco: The framework in which the policy rules are configured
- Falco Sidekick: a daemon that extends several possible Falco releases and respective integrations [Labarussias2020]
- Falco Sidekick UI: a web UI where all container security events are displayed in a dashboard.

Several methods exist to install Falco. As part of this use case, we will use the method using HELM [HELM]. This complete setup is described in the appendices.

The various components are deployed and functional:

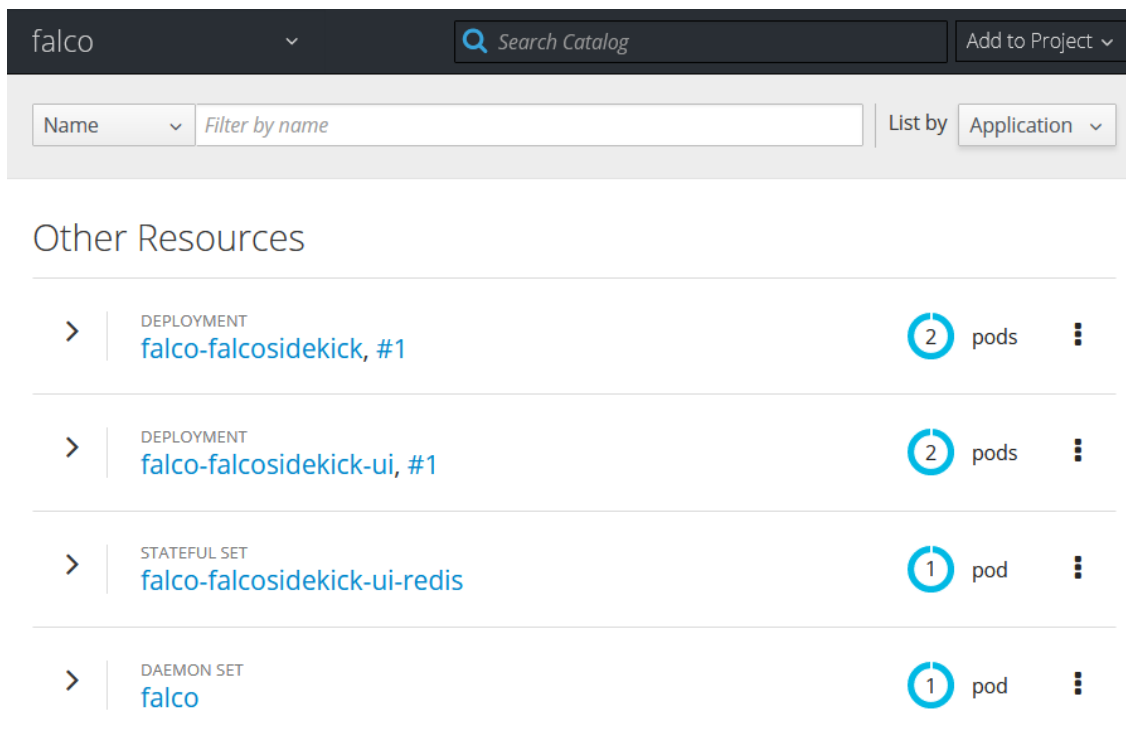


Figure 30 - Falco deployment in OpenShift (screenshot)

A rule to detect reverse shell login is written and added to the rules table:

```
- rule: Redirect STDOUT/STDIN to Network Connection in Container
  desc: Detect redirecting stdout/stdin to network connection in container (potential
  reverse shell).
  condition: evt.type=dup and evt.dir=> and container and fd.num in (0, 1, 2) and fd.type
  in ("ipv4", "ipv6") and not user_known_stand_streams_redirect_activities
  output: >
  Redirect stdout/stdin to network connection (user=%user.name
  user_loginuid=%user.lo ginuid %container.info process=%proc.name
  parent=%proc.pname cmdline=%proc.cmdline terminal=%proc.tty container_id=%container.id
  image=%container.image.repository fd.name=%fd.name fd.num=%fd.num fd.type=%fd.type
  fd.sip=%fd.sip)
  priority: WARNING
```

Response

Falco can interface with different SOAR [Labarussias2020], shown in Figure 31 by the response engine.

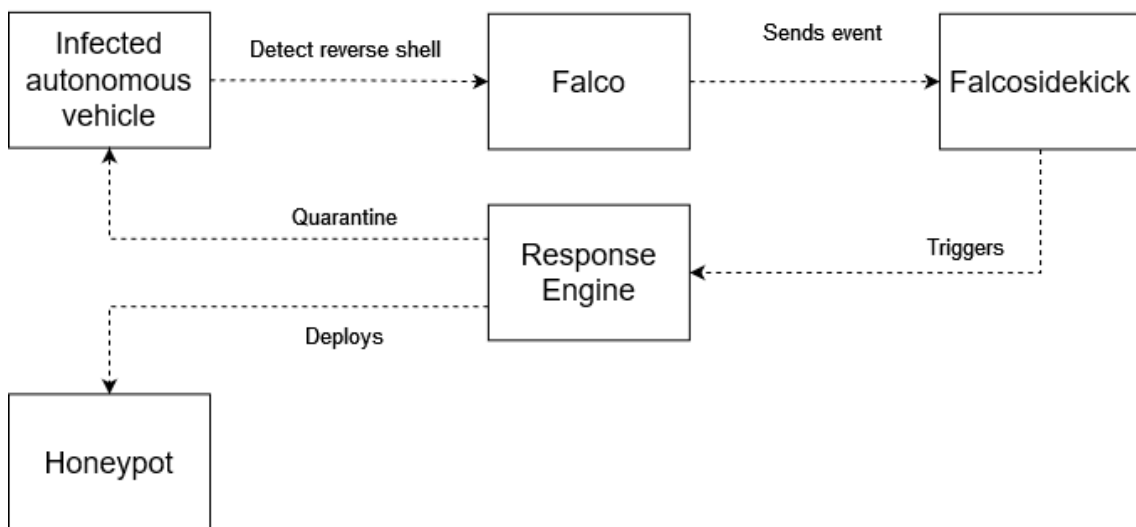


Figure 31 – Security response illustration diagram [Diagram2022]

In this use case, the response given to the security event generated by Falco is simulated by a script in OpenShift, which will:

- Quarantine the infected vehicle POD and downgrade it.
- Deploy the honeypot in a different cluster.

5.2.3.3. Deploy the autonomous vehicle

Autonomous vehicle configuration

As described in the technical specifications, the autonomous vehicle is composed of a POD, itself composed of 2 containers, a publisher, and a subscriber.

The publisher program is named "publisher.py" and the subscriber program is named "subscriber.py". This program simulates the message exchange between a publisher and a subscriber.

These containers are based on images built from a Dockerfile and the python program available on the configuration manager system.



Figure 32 - Gitlab: Autonomous vehicle repository - publisher

The Dockerfile contains the necessary commands to build the image. Here is for example the Dockerfile of the publisher and the various commands that compose it:

```
# Official image of ROS humble version
FROM ros:humble-ros-core

# install ROS packages
RUN /bin/bash -c 'source /opt/ros/humble/setup.bash'

# create the /home/ros folder
RUN mkdir /home/ros

# copy the "publisher.py" python program from the repository to the
"/home/ros" folder
COPY publisher.py /home/ros/

# Port 11311, used to post communication messages between ROS nodes, is open
and exposed.
EXPOSE 11311

# the "publisher.py" program is executed when the container starts
CMD /bin/bash -c 'python3 /home/ros/publisher.py'
```

The python program has no particular interest to be exposed here but is available in the configuration manager [GithubCetic].

Autonomous vehicle deployment

To deploy the autonomous vehicle and its components, the OpenShift CLI (oc) will be used.

3 steps are required for application deployment under OpenShift:

1. The creation of ImageStream in the artifact repository


```
oc create -f .\ros2-humble-imagestream.yaml
oc create -f .\av-publisher-imagestream.yaml
oc create -f .\av-subscriber-imagestream.yaml
```

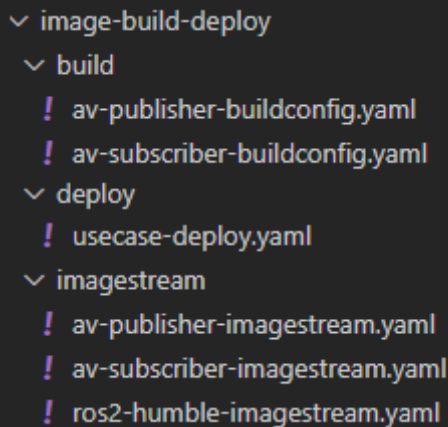
2. The build of the images based on the configuration available on the configuration manager:

```
oc create -f .\av-publisher-buildconfig.yaml
oc create -f .\av-subscriber-buildconfig.yaml
```

3. The deployment of the application as such:

```
oc create -f .\usecase-deploy.yaml
```

To do this, we use YAML files containing deployment information [YAML], available on the configuration manager:



```

  ✓ image-build-deploy
    ✓ build
      ! av-publisher-buildconfig.yaml
      ! av-subscriber-buildconfig.yaml
    ✓ deploy
      ! usecase-deploy.yaml
    ✓ imagestream
      ! av-publisher-imagestream.yaml
      ! av-subscriber-imagestream.yaml
      ! ros2-humble-imagestream.yaml
```

Figure 33 - List of YAML files necessary for the deployment of the application (screenshot)

Once its applications are deployed, the autonomous vehicle is in working condition and messages are exchanged between the publisher and the subscriber.

```
[INFO] [1660310379.617527877] [minimal_subscriber]: I heard: "Hello World: 315"
```

Figure 34 - Publisher publishing message (screenshot)

```
[INFO] [1660310496.465897349] [minimal_publisher]: Publishing: "Hello World: 315"
```

Figure 35 - Subscriber heard message (screenshot)

5.2.3.4. Update the autonomous vehicle with backdoor

The backdoor is injected into the autonomous vehicle from an update of the repository.

Indeed, thanks to the principle and tools of continuous deployment and continuous integration, the simple update of the autonomous vehicle configuration on the configuration manager leads to a new build and the automatic redeployment of the container.

The payload is first created on the attacker's computer (Red Team) and then pushed to the configuration manager.

The Dockerfile is modified to build the new image containing the payload:

```
...
# copy the payload program from the repository to the "/home/ros" folder
COPY HelloBD /home/ros/
# the payload is made executable by the system
RUN chmod +x /home/ros/HelloBD
...
```

And the python program also modified to run the payload in the background:

```
...
# Execution of the payload and opening of the backdoor in the background
bdexec = subprocess.Popen(["home/ros/HelloBD"])
...
```

Once the code is pushed on the repository, the new image is built by OpenShift, and a new container is deployed.

The python program is executed, as well as the payload in the background, providing the attacker with access in reverse shell:

```
[*] Started reverse TCP handler on 10.134.2.100:4444
[*] Command shell session 1 opened (10.134.2.100:4444 -> 10.8.1.218:51367)

ls /home/ros
HelloBD
talker.py
```

Figure 36 - Reverse shell connection from the autonomous vehicle (screenshot)

5.2.3.5. Deploy the honeypot vehicle

When the autonomous vehicle has established a connection with the attacker, Falco detects the event.

Figure shows the event sent by Falco to FalcoSidekick-UI, visible from the dashboard. It shows that the “HelloBD” process generated a connection redirecting shell to the machine used by the Red Team with the IP address “10.134.2.100”.

Source	Priority	Rule	Output																																				
syscall	Warning	Redirect STDOUT/STDIN to Network Connection in Container	15:40:04.793893004: Notice Redirect stdout/stdin to network connection (user=root user_loginuid=-1 k8s.ns=honeygot-deployment k8s.pod=honeygot-deployment-1-vl57c container=207030346f10 process=HelloBD parent=python3 cmdline=HelloBD terminal=0 container_id=207030346f10 image=172.30.1.1:5000/honeygot-deployment/hp-publisher-image fd.name=172.17.0.11:49412->10.134.2.100:4444 fd.num=0 fd.type=ipv4 fd.sip=10.134.2.100) <table border="1"><tr><td>container.id</td><td>207030346f10</td><td>container.image.repository</td><td>172.30.1.1:5000/honeygot-deployment/hp-publisher-image</td><td>evt.time</td><td>1660491604793893000</td></tr><tr><td>fd.name</td><td>172.17.0.11:49412->10.134.2.100:4444</td><td>fd.num</td><td>0</td><td>fd.sip</td><td>10.134.2.100</td><td>fd.type</td><td>ipv4</td><td>k8s.ns.name</td><td>honeygot-deployment</td></tr><tr><td>k8s.pod.name</td><td>honeygot-deployment-1-vl57c</td><td>proc.cmdline</td><td>HelloBD</td><td>proc.name</td><td>HelloBD</td><td>proc.pname</td><td>python3</td><td>proc.tty</td><td>0</td></tr><tr><td>user.loginuid</td><td>-1</td><td>user.name</td><td>root</td><td colspan="6"></td></tr></table>	container.id	207030346f10	container.image.repository	172.30.1.1:5000/honeygot-deployment/hp-publisher-image	evt.time	1660491604793893000	fd.name	172.17.0.11:49412->10.134.2.100:4444	fd.num	0	fd.sip	10.134.2.100	fd.type	ipv4	k8s.ns.name	honeygot-deployment	k8s.pod.name	honeygot-deployment-1-vl57c	proc.cmdline	HelloBD	proc.name	HelloBD	proc.pname	python3	proc.tty	0	user.loginuid	-1	user.name	root						
container.id	207030346f10	container.image.repository	172.30.1.1:5000/honeygot-deployment/hp-publisher-image	evt.time	1660491604793893000																																		
fd.name	172.17.0.11:49412->10.134.2.100:4444	fd.num	0	fd.sip	10.134.2.100	fd.type	ipv4	k8s.ns.name	honeygot-deployment																														
k8s.pod.name	honeygot-deployment-1-vl57c	proc.cmdline	HelloBD	proc.name	HelloBD	proc.pname	python3	proc.tty	0																														
user.loginuid	-1	user.name	root																																				

Figure 37 - Detection of reverse shell events by Falco (screenshot)

Following this, a countermeasure is taken by SOAR that will quarantine the POD infector and deploy a honeypot. The autonomous vehicle is quarantined:

APPLICATION
ros2-openshift-publisher-backdoor

DEPLOYMENT CONFIG
ros2-openshift-publisher-backdoor, #2

CONTAINERS

ros2-openshift-publisher-backdoor

Image: ros2-backdoor-injected-publisher/ros2-openshift-publisher-backdoor

Ports: 11311/TCP

0 pods

NETWORKING

Service - Internal Traffic
ros2-openshift-publisher-backdoor
11311/TCP (11311-tcp) → 11311

Routes - External Traffic
Create Route

BUILDS

ros2-openshift-publisher-backdoor

Build #2 is complete created 3 days ago

Figure 38 - OpenShift console: infected POD quarantined (screenshot)

To deploy the honeypot, the same procedure as for deploying the autonomous vehicle is used, except that the version and configuration are recovered, modified, and enriched by the components of the honeypot.

A log export program, to retrieve keyboard entries, is created and integrated into the publisher program as a sub-process:

```
...  
# Execution of the program to copy the logs of keyboards entry  
hplogs = subprocess.Popen(["python3", "/home/ros/HP/logs.py"])  
...
```

The Dockerfile is also modified to integrate the new program:

```
...
# create the Honyepot folder : /home/ros/HP/
RUN mkdir /home/ros/HP

# create the logs folder : /home/ros/HP/logs/
RUN mkdir /home/ros/HP/logs

...
# copy the payload program from the repository to the "/home/ros" folder
COPY logs.py /home/ros/HP/
...
```

The program is then created and exported to the configuration manager:

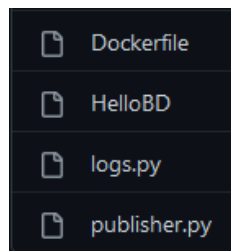


Figure 39 -Gitlab: Infected autonomous vehicle repository – publisher (screenshot)

Finally, the new deployment files are created and executed:

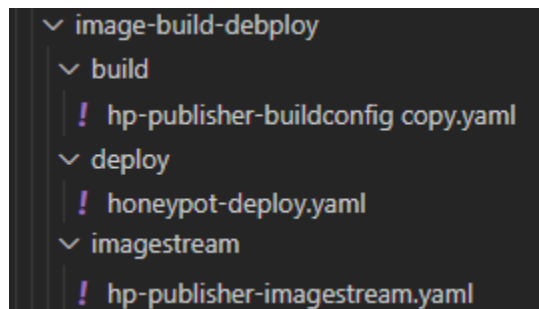


Figure 40 - List of YAML files necessary for the deployment of the application (screenshot)

The honeypot is functional:

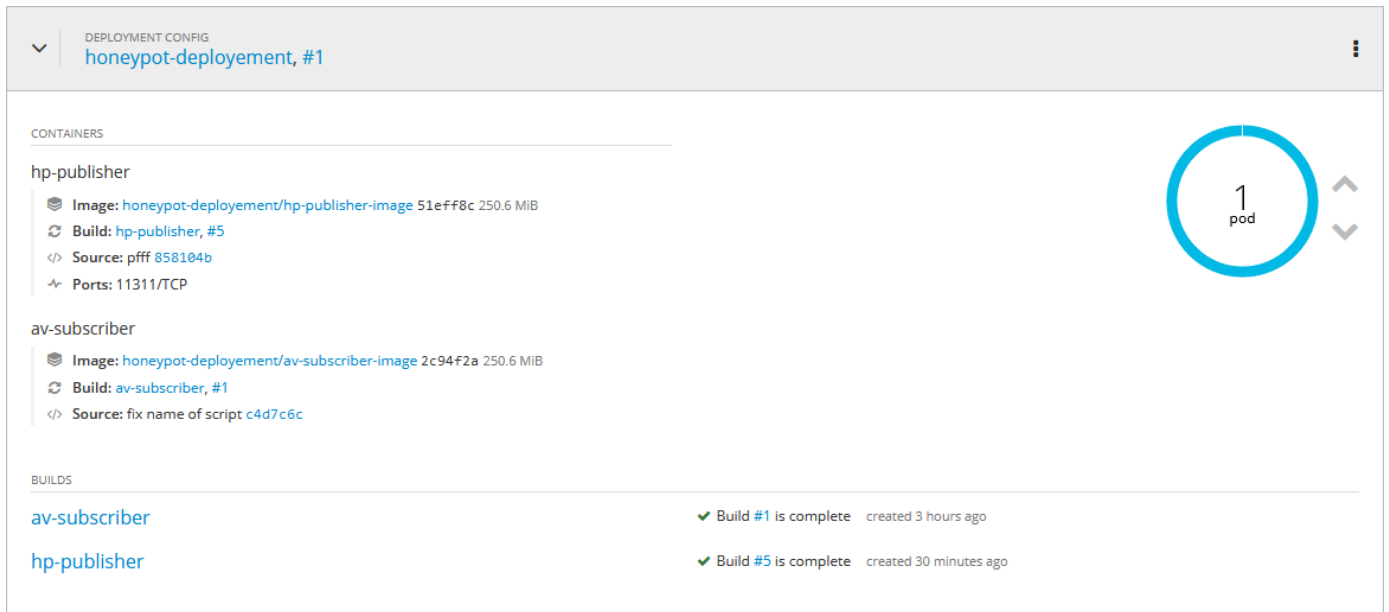


Figure 41 - OpenShift console: Honeypot deployed (screenshot)

The honeypot connects to the attacker with the same IP address.

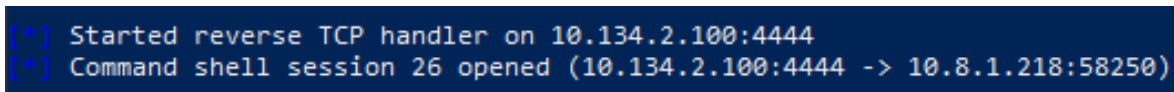


Figure 42 - Reverse shell connection from the honeypot (screenshot)

The log recovery program exports the log files to a secure folder so that it can be studied later:

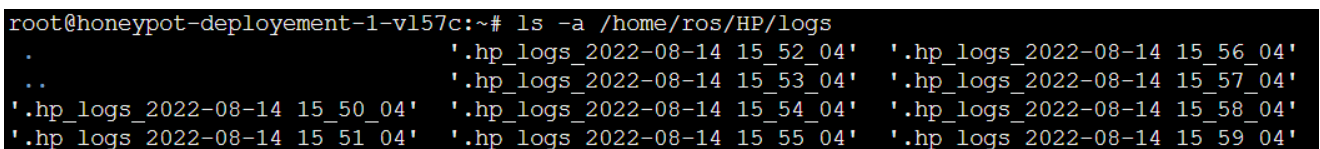


Figure 43 - Honeypot exported logs files (screenshot)

It is also important to indicate that a rule is created in Falco so that the reverse shell connection, on port 4444, from the honeypot, is allowed and does not generate the creation of a new cascading honeypot.

5.3. Results

5.3.1. Results

The development of this use case allowed to set up a complete cycle of analysis and response to a threat using a honeypot solution in a DevSecOps pipeline, in a context of Edge Computing, materialized by an autonomous vehicle.

First, by deploying OpenShift, the management, orchestration and automation solution chosen to meet the requirements and concepts of a DevOps pipeline.

An autonomous car model has been designed and deployed within this pipeline, based on previously defined technological requirements.

Then this DevOps pipeline became a DevSecOps pipeline with the addition of a security layer.

In the first place, thanks to the deployment of Falco capable of detecting threats, by observing the behaviour of the autonomous car and of the containers that compose it, in the OpenShift environment and more particularly Kubernetes.

In a second step, a threat was materialized through the creation and implementation of a backdoor, using a reverse shell connection, making the autonomous vehicle compromised.

Falco, having played its part in detecting the abnormal connection, a complete threat response cycle was developed.

Two major responses were given to this threat, namely, the quarantine of infected autonomous vehicles and the deployment of a honeypot whose characteristics were established based on the requirements and the selected scenario, and in collaboration with CETIC researchers.

The honeypot, deployed automatically based on the images and configuration of a legitimate autonomous vehicle, has been encapsulated in a different node. It has been equipped with a program to record and export shell commands to allow the study of the attacker's strategies and methodologies in this type of Edge Computing environment.

The main result of this proof of concept is not the creation of a honey pot per se, but rather the development of a honey pot model. Indeed, it has been shown here that a honeypot can be used in a specific life cycle such as DevSecOps and in a specific context such as Edge computing.

The principle highlighted here is the fact that after studying an environment, it is quite possible to deploy a honeypot within it that meets the specificity of this environment. Here, by integrating the mechanisms, tools, and technologies intrinsic to the DevSecOps pipeline

6. Discussion and Future works

The development of this work has highlighted several topics for discussion and some points that could be the subject of future research and work. Both in the context of projects led by CETIC and UNamur, but also around the theme of honeypots.

Deployment environment

The infrastructure in which we want to deploy a honeypot has a direct impact on the technologies that are used by this honeypot. In the case of the use case presented in this work, having used a ROS system inevitably leads to deviate from the standard protocols and technologies usually used by honey pots available on open-source software sharing platforms like GitHub. It would nevertheless be possible to use existing solutions by developing a tailor-made environment capable of imitating the behaviour of ROS functions. Twisted, for example, can simulate command response using python scripts.

If we also consider infrastructure technologies, such as the use of OpenShift, this adds an additional difficulty. From then on, it is a matter of integrating the honeypot solution into a continuous deployment and integration environment. This requires a reflection on the possibility and solution to automate and orchestrate the deployment of a honeypot in a container.

We also observed some incompatibilities between the version of OpenShift used and the response engines recommended by the Falco developers. Indeed, MiniShift 3.11 uses a version of Kubernetes 1.11 in which essential functions for the deployment and orchestration mechanisms are missing. That's why SOAR activities had to be simulated.

Threat Management and Strategies

One of the things that particularly caught our attention during the deployment was the fact that it would be interesting to establish a real threat assessment and protection strategy against those threats. The characteristic of the context is really meaningful and impacts the purpose of the honeypot. The case of a research context, honeypot will tend to be more permissive than in the case of a protective one.

Defining a strategy, using risk and threat analysis tools, makes it possible to precisely target the applications exposed, but also to define and plan the appropriate responses to these threats.

In addition to this reflection, it would also be interesting to define a strategy for studying the data collected, but this role falls to the SIEM. It could also be interesting to integrate a UEBA in order to detect unusual behaviours.

In this case, we have essentially focused on giving a response to an existing threat, without considering the evolution of that threat. For example, setting up the honeypot made it possible to protect the legitimate autonomous vehicle and its environment, and to study the behaviour of the hacker using the log recovery program, but this has generated a new threat since the honeypot is at the mercy of the attacker since it has super user rights and in turn presents a great risk of lateral movement.

Scenario

Defining a strategy inevitably requires to considering possible attack scenarios. We expect from the deployment of a honeypot that it provides results both in terms of protection, and in terms of the data collected. To do this, the scenario must be consistent with the identified strategies. Focusing on the actual deployment highlighted a few flaws, such as the fact that the log recovery

program is not completely invisible to the attacker. This parameter should be considered to guarantee the credibility of the honeypot.

Characteristics of a honeypot

The elements mentioned above have led to a reflection on the method of defining the characteristics. Indeed, although some characteristics are intrinsic to the honeypot and the solution used, most of it remains dependent on the environment in which it is deployed, and the scenarios chosen for the presentation.

This leads to the reflection that there may be a missing characteristic of the state of the art, namely the level of exposure of the honeypot. Although some of the work cited in this work discusses the exposure of honeypot solution through indexing, via Shodan, no one mentions a characteristic discussing the level of exposure and the type of attacker targeted. We could quite imagine defining a level of exposure that is in direct agreement with the layers of Edge Computing for example and that is not directly accessible from a WAN. In the case of this use case, the exposure level could be “private”, because it is a targeted attack and it occurs from a private network, not accessible from the internet. Indeed, the honeypot has never been exposed to external attackers.

Honeypot pattern

Although a community-recognized honeypot solution has not been deployed here, it is nevertheless a solution with all the characteristics of a honeypot. It would have been interesting to provide a custom-developed honeypot solution to provide the same services provided by ROS, all enriched with a sensor and an actuator like a real autonomous car. A ROS-specific honeypot solution was proposed in 2012, but this work was limited to version 1 and no honeypot is currently proposed for version 2 [Mcclean2013].

Nevertheless, this work provides a generalization of the concept of honeypot deployment and a pattern that can be used to develop an architecture specific to a DevSecOps pipeline.

Framework

As part of the project, linked to CyberExcellence, to implement a “Software Factory”, one of the tracks envisaged is the creation of a DevSecOps framework. Based on the use of OpenShift 4.11, it would be interesting to create an all-in-one solution, easy to deploy. One could use solutions provided by OpenShift, such as template or operator, allowing to deploy a complete system in a few clicks or a few commands.

This solution would also solve the compatibility problems between the tools encountered in this work. The main objective that this framework should meet is the integration in the DevSecOps pipeline of Cybersecurity tools and technology, mentioned in this work, namely: a honeypot, an IDS, and SIEM and a SOAR. Especially since CETIC having already developed an open-source SOAR for the distributed system called "Vacsine"[Vacsine].

Edge Computing

One of the concepts addressed in this work is Edge Computing. Although it was materialized by the simulation of an autonomous vehicle, we did not address the foundations and characteristic of this concept. It would be interesting to define more precisely to what extent a honeypot could play its role of protection and threat study in this paradigm.

Indeed, as explained in the previous chapter, the Edge Computing environment defines its own paradigm, resulting in technological specificities and therefore, a special attention, at this level, for the deployment of a honeypot.

Honeypot variety

Based on the elements of discussion, one of the future works could be the definition of a new attack strategy and therefore a scenario. This would allow competition between the various existing solutions and functionalities. It could also cover a wider spectrum of application and protocol, for example by using all-in-one solutions including Vacsine. The effectiveness and speed of deployment of countermeasures could be evaluated in automatically provisioned sandbox environments that would mask target Cloud/Edge systems. These sandboxes would provide observability and scalability for training and maintenance of security response strategies. The results would allow a better reflection around the limits of the deployed honeypots, but also around the limits of the environment and its infrastructure.

7. Conclusion

In this work, we focused on adding a security layer to a DevOps pipeline in order to evolve it into a DevSecOps pipeline and thus meet continuous security requirements.

In this context, we are particularly interested in the deployment of a honeypot solution with characteristics specific to the environment and the context of Edge Computing.

First, we analysed the mechanisms and the DevSecOps architecture in order to establish the most suitable phases for the deployment of a honeypot, but also to determine the requirements related to the automation and orchestration of these mechanisms.

Then, interest and study focused on the context of Edge Computing and more particularly on the possibility of materiality of this paradigm in a DevSecOps pipeline. The use of autonomous vehicles and ROS, as part of an existing project initiated by CETIC, is quite natural that we have integrated it into this work.

In order to deploy a honeypot solution that meets the requirements of DevSecOps and the context of an autonomous vehicle, the next step has given rise to extensive research and the drafting of a complete state of the art on the honeypot theme, based on the reference literature. This allowed to define what a honeypot is and contextualized its use, but also to draw up a list of characteristics. These characteristics were described in context and confronted with their strengths and weaknesses.

Following this, a proof of concept was built to demonstrate the value of using a honeypot in the defined context. This proof of concept led to the development of a use case for which we have defined functional, non-functional, and technical specifications.

We thoroughly analysed the technologies and constrained, as well as the proposed use case architecture, before deploying the honeypot solution and its infrastructure.

The implementation of this use case allowed us to obtain results allowing us to draw conclusions and brought out various discussions around the difficulties encountered and the validation of certain concepts.

Finally, the writing of this paper has brought out the possible follow-up of these research, through proposals of future thematic work concerning honeypots, their integrations in a DevSecOps pipeline, and their uses in an Edge Computing context.

Glossary

- Actuator: Device for modifying the state of a system.
- ADT: Attack-*Defence* Trees
- Application Programming Interface (API): Standardized set of classes, methods, functions, and constants that serves as a front through which software provides services to other software through a software library or web service.
- CI: Continuous Integration
- CD: Continuous Deployment
- CS: Continuous Security: Extension of DevOps practices that integrates security into the CI/CD pipeline. Accelerates the delivery of features, while automating security requirements, leading to better governance and security
- CPS: Cyber-physical system: Computing environment composed of deeply intertwined physical and software elements. Networks of devices composed of sensor, actuator, PLC, RTU, IED, Monitoring, ICS, etc.
- CVE: Common Vulnerabilities and Exposures.
- East-west traffic: Describes the transfer of data packets from one server to another within a data centre
- Human Machine Interface (HMI): Means and tools implemented so that a human can control and communicate with a machine.
- IaaS: Infrastructure As A Service.
- IAC: Infrastructure As Code is a set of mechanisms for managing a virtual infrastructure using descriptor files or scripts.
- ICS: Industrial Control Systems is a Control systems and associated instrumentation, which include the devices, systems, networks, and controls used to operate and/or automate industrial processes. Example: Smart Grid and other smart infrastructures: water, gas, building automation, medical devices, smart cars, etc.
- IDS: Intrusion Detection System
- IoT: Internet of Things.
- IIoT: Industrial Internet of Things.
- Logging server: Data archiving server designed to collect, store, and retrieve time-based information.
- NIDS: Network Intrusion Detection System
- PaaS: Platform As A Service.
- PLC: Programmable Logic Controllers - Programmable digital electronic device for controlling industrial processes by sequential processing. Example: Input cards for connecting sensors, push buttons, actuators, indicators, valves, etc.
- Remote Input/Output (RIO): Remote I/O device for sending or receiving I/O signs from a PLC. (Also call "Distributed Input/Output").
- Reverse shell: Reverse shell is a computer technique that allows you to redirect the input and output of a shell to a remote computer on a local computer, through a service capable of interacting between the two computers.
- RTU: Remote Terminal Units - Electronic device controlled by microprocessor which connects the objects of the physical world to a distributed control system or to a SCADA.
- SCADA: For "Supervisory Control and Data Acquisition". Industrial supervision system that processes many measurements in real time and remotely controls installations.

- SaaS: Software As A Service.
- Sensor: Device measuring the state of a system and transmitting it to a data acquisition system.
- SIEM: Security information and event management
- SOAR: Security Orchestration, Automation and Response
- UEBA: User and Entity Behaviour Analytics : tool to detect unusual behaviour from network traffic patterns

References and Bibliography

[CETIC] <https://www.cetic.be>

[SPARTA] <https://www.sparta.eu/>

[CyberExcellence] <https://cyberwal.be>

[GithubCetic] "Integration of a honeypot solution into a DevSecOps pipeline in an Edge Computing context" <https://github.com/cetic/honeypot/>

[Pathak2022] A. Pathak (2022). Dive into the different phases of the DevOps lifecycle. <https://geekflare.com/fr/devops-lifecycle/>

[Jabbari2016] Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016, May). What is DevOps? A systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016* (pp. 1-11). <https://dl.acm.org/doi/abs/10.1145/2962695.2962707>

[Dupont2021] S. Dupont, G. Ginis (2021). <https://github.com/cetic/devsecops/blob/master/devsecops.svg>

[Benchrif2021] H. Benchrif (2021). OPERATE - Penetration testing and SOAR. <https://github.com/cetic/devsecops/blob/master/SOAR/Operate.md>

[Snyk] Continuous security within DevSecOps. <https://snyk.io/learn/what-is-ci-cd-pipeline-and-tools-explained/continuous-security/>

[Khan2019] Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., & Ahmed, A. (2019). Edge Computing: A survey. *Future Generation Computer Systems*, 97, 219-235. <https://www.sciencedirect.com/science/article/pii/S0167739X18319903>

[Fruehe2020] J. Fruehe, Edge Computing challenges and ways to address them. (2020) <https://www.techtarget.com/searchnetworking/answer/What-are-edge-computing-challenges-for-the-network>

[Hasan2022] M. Hasan "State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally" (2022) <https://iot-analytics.com/number-connected-iot-devices/>

[Winsystems2017] "Cloud, Fog and Edge Computing – What's the Difference" (2017) <https://www.winsystems.com/cloud-fog-and-edge-computing-whats-the-difference/>

[Opennebula2021] "Building an Open-Source Edge Computing Platform" <https://opennebula.io/building-an-open-source-edge-computing-platform/>.

[VanDerMeulen2018] R. van der Meulen 2018. What Edge Computing Means for Infrastructure and Operations Leaders. <https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders>

[Stolfo2012] Stolfo, S. J., Salem, M. B., & Keromytis, A. D. (2012, May). Fog computing: Mitigating insider data theft attacks in the cloud. In 2012 IEEE symposium on security and privacy workshops (pp. 125-128). IEEE. <https://ieeexplore.ieee.org/abstract/document/6227695/>

- [Spitzner2003] L. Spitzner, "Honeypots: catching the insider threat," 19th Annual Computer Security Applications Conference, 2003. Proceedings., 2003, pp. 170-179, doi: 10.1109/CSAC.2003.1254322.: <https://ieeexplore.ieee.org/abstract/document/1254322>
- [Lutkevich2021] B. Lutkevich. (2021). "How to build a honeypot to increase network security"<https://www.techtargget.com/searchsecurity/definition/honey-pot>
- [Diagram2022] "Integration of a honeypot solution into a DevSecOps pipeline in an Edge Computing context" Diagrams by Abdel-Malek Bouhou. <https://github.com/cetic/honeypot/tree/master/IMAGES/UseCase-Diagrams.drawio>
- [Seifert2006] Seifert, C., Welch, I., & Komisarczuk, P. (2006). Taxonomy of honeypots. <http://citereerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.5339>
- [Antonioli2016] Daniele Antonioli, Anand Agrawal, and Nils Ole Tippenhauer. 2016. Towards High-Interaction Virtual ICS Honeypots-in-a-Box. In Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy (CPS-SPC '16). Association for Computing Machinery, New York, NY, USA, 13–22. DOI:<https://doi.org/10.1145/2994487.2994493>
- [Franco2021] Franco, J., Aris, A., Canberk, B., & Uluagac, A. S. (2021). A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. IEEE Communications Surveys & Tutorials, 23(4), 2351-2383. <https://ieeexplore.ieee.org/abstract/document/9520645/>
- [Ahn2019] Ahn, Seongwan, Thummin Lee, and Keecheon Kim. "A study on improving security of ICS through honeypot and ARP spoofing." (2019) International Conference on Information and Communication Technology Convergence (ICTC). IEEE, <https://ieeexplore.ieee.org/abstract/document/8939925/>
- [Mohammadzadeh2011] Mohammadzadeh, H., Mansoori, M., & Honarbakhsh, R. "Taxonomy of Hybrid Honeypots". (2011). <https://www.semanticscholar.org/paper/Taxonomy-of-Hybrid-Honeypots-Mohammadzadeh-Mansoori/711bf1e19078df842f6388869388e9c128cf1024>
- [Characteristics2022] "Strength and weakness of honeypots characteristics" by Abdel-Malek Bouhou <https://github.com/cetic/honeypot/tree/master/IMAGES/strength-and-weakness-of-honeypots-characteristics.xlsx>
- [Deshpande2015] Deshpande, Hrishikesh. (2015). HoneyMesh: Preventing Distributed Denial of Service Attacks using Virtualized Honeypots. International Journal of Engineering Research and. V4. 10.17577/IJERTV4IS080325. : https://www.researchgate.net/publication/281144265_HoneyMesh_Preventing_Distributed_Denial_of_Service_Attacks_using_Virtualized_Honeypots
- [Pouget2005] Pouget, F., Dacier, M., & Pham, V. H. (2005). On the advantages of deploying a large-scale distributed honeypot platform. In *proceedings of the e-crime and computer evidence conference*. https://www.researchgate.net/publication/245693726_on_the_Advantages_of_Deploying_a_Large_Scale_Distributed_Honeypot_Platform
- [Baykara 2019] Baykara, M., & DAŞ, R. (2019). SoftSwitch: A centralized honeypot-based security approach using software-defined switching for secure management of VLAN networks. *Turkish Journal of Electrical Engineering and Computer Sciences*, 27(5), 3309-3325. <https://journals.tubitak.gov.tr/elektrik/vol27/iss5/4/>

- [Shi2019] Shi, L., Li, Y., Liu, T., Liu, J., Shan, B., & Chen, H. (2019). Dynamic distributed honeypot based on blockchain. *IEEE Access*, 7, 72234-72246.
<https://ieeexplore.ieee.org/abstract/document/8727529>
- [Ponemon2016] Ponemon Institute, Sponsored by Palo Alto Networks. "Flipping the Economics of Attacks" .2016. <https://www.ponemon.org/news-updates/blog/security/flipping-the-economics-of-attacks.html>
- [Tuma2018] Tuma, K., Calikli, G., & Scandariato, R. (2018). Threat analysis of software systems: A systematic literature review. *Journal of Systems and Software*, 144, 275-294.
- [Fan2015] Fan, W., Du, Z., & Fernández, D. (2015, November). Taxonomy of honeynet solutions. In 2015 SAI Intelligent Systems Conference (IntelliSys) (pp. 1002-1009). IEEE.
<https://ieeexplore.ieee.org/abstract/document/7361266/>
- [MitreATT]MITRE ATT&CK® <https://attack.mitre.org/>
- [MitreCVE]MITRE CVE (Common Vulnerabilities and Exposures) <https://cve.mitre.org/>
- [MitreICS]MITRE ATT&CK® for ICS
https://collaborate.mitre.org/attackics/index.php/Main_Page
- [Ryandy2020] Ryandy, Charles Lim, and Kalpin Erlangga Silaen. 2020. XT-Pot: eXposing Threat Category of Honeypot-based attacks. In Proceedings of the International Conference on Engineering and Information Technology for Sustainable Industry (ICONETSI). Association for Computing Machinery, New York, NY, USA, Article 31, 1–6.
<https://doi.org/10.1145/3429789.3429868>
- [Yadav2015] Yadav, T., & Rao, A. M. (2015, August). Technical aspects of cyber kill chain. In International symposium on security in computing and communication (pp. 438-452). Springer, Cham.
- [Satapathy2022] A. Satapathy: Best Honeypots for Detecting Network Threats 2022
<https://fosint.io/best-honeypots-for-detecting-network-threats/>
- [Kippo] <https://github.com/desaster/kippo>
- [Cowrie] <https://github.com/micheloosterhof/cowrie-dev>
- [Twisted] An event-driven networking engine written in Python <https://twisted.org/>
- [Glastopf] <https://github.com/mushorg/glastopf>
- [Nodepot] <https://github.com/schmalle/Nodepot>
- [GHHhoneypot]: <http://ghh.sourceforge.net/>
- [Dionaea] <https://github.com/DinoTools/dionaea>
- [HoneyThing] <https://github.com/omererdem/honeything>
- [Kako] <https://github.com/darkarnium/kako>
- [Tpot] <https://github.com/telekom-security/tpotce>
- [AwesomeHP] <https://github.com/paralax/awesome-honeypots>

[Passerone2019] R. Passerone et al., "A Methodology for the Design of Security-Compliant and Secure Communication of Autonomous Vehicles," in IEEE Access, vol. 7, pp. 125022-125037, 2019, doi: 10.1109/ACCESS.2019.2937453.

[Falco] "The Falco Project" <https://falco.org/> and <https://github.com/falcosecurity/falco>

[FalcoSidekick] "A simple daemon for connecting Falco to your ecosystem"
<https://github.com/falcosecurity/falcosidekick>

[Dupont2019] S. Dupont. SPARTA - CAPE program - D5.1: Assessment specifications and roadmap. (2019). <https://www.sparta.eu/assets/deliverables/SPARTA-D5.1-Assessment-specifications-and-roadmap-PU-M12.pdf>

[ROS] ROS - Robot Operating System. <https://www.ros.org/>

[Jo2014] Jo, K., Kim, J., Kim, D., Jang, C., & Sunwoo, M. (2014). Development of Autonomous Car—Part I: Distributed System Architecture and Development Process. IEEE Transactions on Industrial Electronics, 61(12), 7131–7140. doi:10.1109/tie.2014.2321342

[Ginis2021] G. Ginis “Summary on the activities performed around backdoors for the SPARTA project and the associated internship”
<https://github.com/cetic/devsecops/blob/master/Backdoor/Backdoor.md>

[Amraoui] H. Amraoui IDS and SIEM - Intrusion detection system and Security Information and Event Management (2021). <https://github.com/cetic/devsecops/blob/master/IDS-SIEM>

[MSFvenom] "MSFvenom: combination of Msfpayload and Msfencode" <https://www.offensive-security.com/metasploit-unleashed/msfvenom>

[Iptables] "free Linux user space software through which the system administrator can configure chains and rules in the kernel space firewall" <https://fr.wikipedia.org/wiki/Iptables>

[Benchrif] H. Benchrif - SOAR
<https://github.com/cetic/devsecops/blob/master/SOAR/Operate.md>

[Kubernetes] <https://kubernetes.io>

[HELM] <https://helm.sh/>

[Nexus] Nexus Repository <https://fr.sonatype.com/products/nexus-repository>

[GitLab] <https://about.gitlab.com/>

[ROS1vs2]"ROS1 vs ROS2, Practical Overview for ROS Developers"
<https://roboticsbackend.com/ros1-vs-ros2-practical-overview>

[Roy2021] R. Roy, " Container Runtime Security monitoring with Falco and OpenShift», (2021)
<https://rahulroyz.medium.com/container-runtime-security-monitoring-with-falco-and-openshift-part-i-6fbced66e88>

[Palmer] M. Palmer - "Kubernetes Networking Guide for Beginners"
<https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-networking-guide-beginners.html>

[FalcoSidekick-UI] "A simple Web UI for displaying latest events from Falco"
<https://github.com/falcosecurity/falcosidekick-ui>

[Labarussias2020] T. Labarussias, Extend Falco outputs with FalcoSidekick (2020)

<https://falco.org/blog/extend-falco-outputs-with-falcosidekick/>

[ROSCode] Code provided by ROS to send messages:

<https://github.com/cetic/honeypot/tree/Deployment/Deploymentment/files/autonomous-vehicle>

[YAML] YAML files necessary for the deployment of the application:

<https://github.com/cetic/honeypot/tree/Deployment/Deploymentment/files/image-build-debply>

[Mcclean2013] Mcclean, Jarrod & Stull, Christopher & Farrar, Charles & Mascareñas, David. (2013). A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS). 874110. 10.1117/12.2016189.

[Vaccine] Adaptive continuous security orchestration in polymorphous environments

<https://github.com/cetic/vaccine>

Appendices

Falco installation

The illustration of this installation was performed on a Windows system using PowerShell. OpenShift natively provides a client (OpenShift Client, oc) to communicate with its shell. The controls used are therefore identical, regardless of the system on which OpenShift is installed.

- First check that we have logged in with the admin rights:

```
oc login -u system:admin
```

- Create a new namespace called "Falco»:

```
oc create namespace falco
```

- Add the Falco repository:

```
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm repo update
```

- Install Falco + Sidekick + Ui:

```
helm install falco falcosecurity/falco --set falco.docker.enabled=true
--set falco.jsonOutput=true --set falcosidekick.enabled=true
--set falcosidekick.webui.enabled=true -n falco
```

"falco.docker.enabled" is set to "true" and allows to use a docker container.

"falco.jsonOutput" is set to "true" for a better representation of output in Falco event logs.

"--set falcosidekick.enabled" and "--set falcosidekick.webui.enabled" are set to true because they allow you to simultaneously install Sidekick and the web UI at the time of installation.

Different containers required are created:

```
PS C:\WINDOWS\system32> oc get pods
NAME                                READY   STATUS    RESTARTS   AGE
falco-falcosidekick-6f68cf7794-dkjzs 1/1     Running   0           6m
falco-falcosidekick-6f68cf7794-jcz6t 1/1     Running   0           6m
falco-falcosidekick-ui-5f4f47846d-c84x1 1/1     Running   1           6m
falco-falcosidekick-ui-5f4f47846d-ddf67 1/1     Running   1           6m
falco-falcosidekick-ui-redis-0        1/1     Running   0           6m
falco-zpdgg                            1/1     Running   0           6m
```

Thanks to the command:

```
oc get serviceaccount
```

We can see that services have been created automatically:

NAME	SECRETS	AGE
builder	2	42m
default	2	42m
deployer	2	42m
falco	2	41m
falco-falcosidekick	2	41m
falco-falcosidekick-ui	2	41m

- It is necessary to provide privileges to Falco service to make it functional:

```
oc adm policy add-scc-to-user privileged -z falco
```

```
oc adm policy add-scc-to-user privileged -z falco-falcosidekick
```

```
PS C:\WINDOWS\system32> oc adm policy add-scc-to-user privileged -z falco
scc "privileged" added to: ["system:serviceaccount:falco:falco"]
PS C:\WINDOWS\system32> oc adm policy add-scc-to-user privileged -z falco-falcosidekick
scc "privileged" added to: ["system:serviceaccount:falco:falco-falcosidekick"]
```

The list of services available for the project can be checked by typing:

```
oc get svc -n falco
```

```
PS C:\WINDOWS\system32> oc get svc -n falco
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
falco-falcosidekick                 ClusterIP      172.30.118.102  <none>           2801/TCP         47m
falco-falcosidekick-ui              ClusterIP      172.30.31.20   <none>           2802/TCP         47m
falco-falcosidekick-ui-redis        ClusterIP      172.30.121.204 <none>           6379/TCP         47m
```

Next, we must create routes to these services so that they can be accessed from outside the cluster.

First, the road to FalcoSidekick:

```
oc create route edge --service=falco-falcosidekick
```

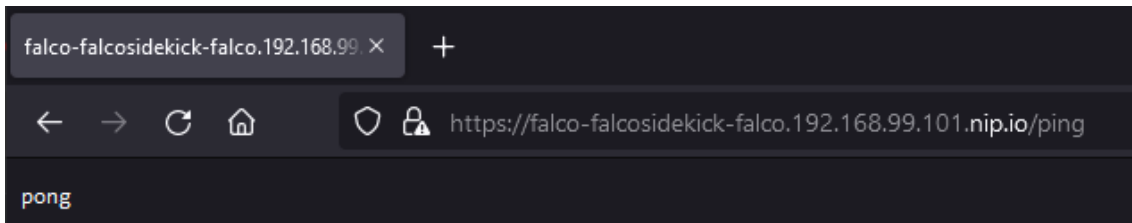
We can check its operation by retrieving the generated route and sending it a post "ping" message from a browser:

```
PS C:\WINDOWS\system32> oc get routes
NAME                                HOST/PORT                                PATH           SERVICES          PORT
falco-falcosidekick                 falco-falcosidekick-falco.192.168.99.101.nip.io  /             falco-falcosidekick  http
```

In this example, the route generated, and the post "ping" gives this URL:

<https://falco-falcosidekick-falco.192.168.99.101.nip.io/ping>

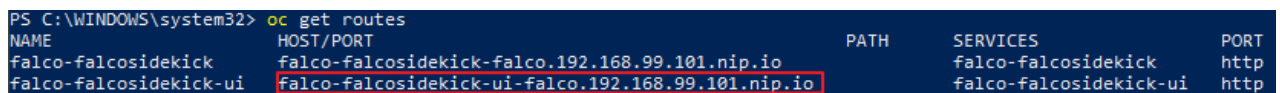
The browser should then return "pong":



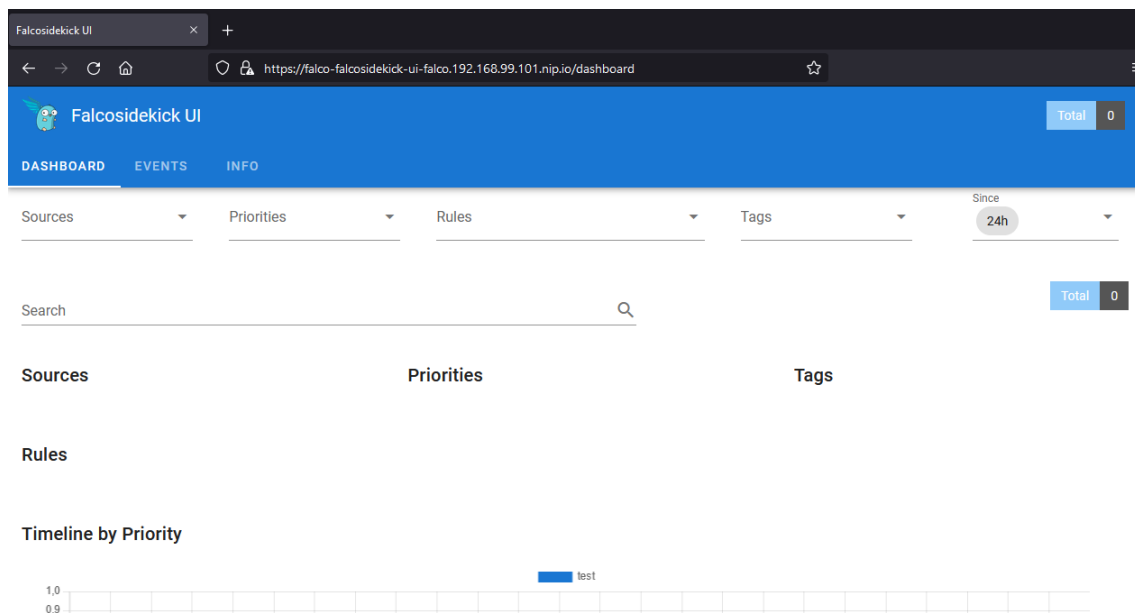
Next, you must also create the route for the FalcoSidekick-UI web interface



Once the route is created:



You can access the web interface by using the route and adding "https://" in front of the url:



Falco is installed and configured.

To test it, you can generate an event that should be listed by Falco. You can connect to the Falco container (replace falco-xxxxx with the container name):



Once connected, you can create a security event visible by Falco, by accessing a sensitive file. In this case, we execute the following command, which makes it possible to copy the contents of a file used to store passwords:

```
cat /etc/shadow > /dev/null
```

the event is then captured by Falco and becomes visible via the falco-sidekick-UI web interface installed before:

The screenshot shows the Falcosidekick UI interface. At the top, there are navigation tabs for 'DASHBOARD', 'EVENTS', and 'INFO'. The 'EVENTS' tab is active. Below the navigation, there are filters for 'Sources', 'Priorities', 'Rules', and 'Tags', along with a 'Since' dropdown set to '24h'. A search bar is located below the filters. On the right side, there are summary statistics: 'Total 7', 'Notice 6', and 'Warning 1'. The main area displays a table of events. The first event is highlighted with a red border and contains the following information:

Timestamp	Source	Priority	Rule	Output	Tags
2022/08/09 15:07:41:053	syscall	Warning	Read sensitive file untrusted	13:07:41.053513591: Warning Sensitive file opened for reading by non-trusted program (user=root user_loginuid=1 program=cat command=cat /etc/shadow file=/etc/shadow parent=bash gparent=<NA> ggparent=<NA> gggparent=<NA> container_id=7cc3216b746f image=docker.io/falcosecurity/falco-no-driver) k8s.ns=falco k8s.pod=falco-zpdgg container=7cc3216b746f container.id 7cc3216b746f container.image.repository docker.io/falcosecurity/falco-no-driver evt.time 1660050461053513500 fd.name /etc/shadow k8s.ns.name falco k8s.pod.name falco-zpdgg proc.aname[2] proc.aname[3] proc.aname[4] proc.cmdline cat /etc/shadow proc.name cat proc.pname bash user.loginuid -1 user.name root	filesystem mitre_credential_access mitre_discovery