



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

IT/OT MODULAR HONEYPOT

MODULÁRNÍ HONEYPOT PRO IT A OT

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Martin Nečas

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. Petr Blažek

BRNO 2023

Bachelor's Thesis

Bachelor's study program **Information Security**

Department of Telecommunications

Student: Martin Nečas

ID: 231259

**Year of
study:** 3

Academic year: 2022/23

TITLE OF THESIS:

IT/OT modular honeypot

INSTRUCTION:

The bachelor's thesis aims to design and implement a modular and scalable honeypot focusing on IT and OT communication protocols and to study the low to high-interaction honeypot issues. Subsequently, design and implement a system where the core will be a scalable honeypot, in which modules representing communication protocols and communicating physical and virtual devices can be easily added. Also, focus on the modularity of the system during design and implementation. Then test the functionality and deployment in a simulated or real environment and evaluate the data obtained from the honeypot.

The output of the bachelor's thesis will be a honeypot system focusing on selected protocols, implementation of at least two protocols (one from IT and the other from OT) and simulations of selected devices. A partial output will be the deployment of the honeypot into a real or simulated environment and evaluation of the obtained data.

RECOMMENDED LITERATURE:

[1] SPITZNER, Lance. Honeypots: tracking hackers. Reading: Addison-Wesley, 2003.

[2] NG, Chee Keong; PAN, Lei; XIANG, Yang. Introduction to Honeypot. In: Honeypot Frameworks and Their Applications: A New Framework. Springer, Singapore, 2018. p. 1-5.

**Date of project
specification:** 6.2.2023

**Deadline for
submission:** 26.5.2023

Supervisor: Ing. Petr Blažek

doc. Ing. Jan Hajný, Ph.D.
Chair of study program board

WARNING:

The author of the Bachelor's Thesis claims that by creating this thesis he/she did not infringe the rights of third persons and the personal and/or property rights of third persons were not subjected to derogatory treatment. The author is fully aware of the legal consequences of an infringement of provisions as per Section 11 and following of Act No 121/2000 Coll. on copyright and rights related to copyright and on amendments to some other laws (the Copyright Act) in the wording of subsequent directives including the possible criminal consequences as resulting from provisions of Part 2, Chapter VI, Article 4 of Criminal Code 40/2009 Coll.

ABSTRACT

The bachelor's thesis created a modular and scalable honeypot system focusing on IT and OT. The goal was to design and implement a system that allows the easy addition of modules representing communication protocols and virtual devices while strongly emphasising modularity. The honeypots are deployed in containers, and the system actively monitors the communication between these containers and the attackers. All communication activities are stored in a centralised log for thorough analysis and monitoring.

KEYWORDS

Modular Honeypot System, Honeynet, Containers, Docker, Network Namespaces, Container Network Monitoring, Information Technology, Operational Technology, Cybersecurity, tcpdump, Python, Django, Centralized Logging, syslog, rsyslog, modbus, http, cron

ABSTRAKT

V bakalářské práci byl realizován modulární a škálovatelný systém honeypotů zaměřený na IT a OT. Cílem bylo navrhnout a implementovat systém, který umožní snadné přidávání modulů reprezentujících komunikační protokoly a virtuální zařízení a silně zdůrazňuje modularitu. Honeypoty jsou nasazeny v kontejnerech a systém aktivně monitoruje komunikaci mezi těmito kontejnery a útočníky. Všechny komunikační aktivity jsou uloženy v centralizovaném logu pro důkladnou analýzu a monitorování.

KLÍČOVÁ SLOVA

Modulární systém Honeypotů, Honeynet, Kontejnery, Docker, Síťové jmenné prostory, Monitorování sítě kontejnerů, Informační technologie, Operační technologie, Kyberbezpečnost, tcpdump, Python, Django, Centralizované logování, syslog, rsyslog, modbus, http, cron

ROZŠÍŘENÝ ABSTRAKT

Cílem této bakalářské práce bylo navrhnout a vyvinout modulární, škálovatelný systém honeypotů zaměřený na IT a OT protokoly. Dále cílem bylo vytvořit uživatelsky přívětivý koncept, který by zajistil jednoduchou upravitelnost systému a škálovatelnost z pohledu rozšiřování nových modulů nad rámec této bakalářské práce.

Motivace práce

Honeypoty slouží jako návnada pro kybernetické útočníky, čímž odvádějí pozornost od skutečných systémů. Pomáhají při detekci a analýze útoků, což umožňuje lepší pochopení taktik, technik a postupů kybernetických zločinců. Dále umožňují identifikovat nové druhy útoků a slouží jako první obranná linie před neautorizovaným přístupem. Nakonec, tím, že zpomalují útočníky a nutí je vynakládat více úsilí, mohou také sloužit jako efektivní odstrašující prostředek.

Teoretický rámec

Úvodní kapitola poskytuje přehled o honeypotech, jejich nezbytnosti, typech a klasifikacích. Je zde provedeno srovnání mezi honeypoty a systémy pro detekci průniku a je uveden přehled existujících open-source projektů honeypot, z nichž některé byly využity během nasazení, testování a sběru dat v rámci této práce.

Kapitola dále podrobněji popisuje použití kontejnerů a porovnává je s virtuálními stroji. Představuje koncept Linuxových jmenových prostorů, které tvoří základ kontejnerů a přispívají k zabezpečení hostitelského operačního systému. Dále se zabývá sítěmi kontejnerů a mechanismy, prostřednictvím kterých kontejnery komunikují se sítí hostitele. Kapitola také diskutuje o obrazech kontejnerů, jejich složení a o tom, jak je uživatelé mohou vytvářet a nahrávat do externího registru. Závěr teoretické části popisuje Docker-compose, nástroj pro zjednodušenou správu a orchestraci Docker kontejnerů na hostitelském systému.

Návrh a implementace

Tato sekce popisuje proces návrhu, přičemž zdůrazňuje dvě hlavní fáze vývoje návrhu, rozdělené na počáteční a konečné návrhy.

Segment počátečního návrhu podrobně popisuje první model používající virtuální stroje na hostiteli VMware ESXi místo kontejnerů. Vysvětluje proces vytváření systému honeynet pomocí Ansible a zdůrazňuje problémy návrhu, které vedly ke změně směrem ke kontejnerům.

Část konečného návrhu popisuje konečný design implementovaný pomocí kontejnerů, které umožňují rychlejší nasazení systému a menší nároky na úložný prostor než virtuální stroje. Tento návrh zahrnuje několik propojených komponent.

Systém se skládá z webového serveru s uživatelským rozhraním, pomocí kterého mohou administrátoři interagovat se systémem honeynet. Webový server přijímá zachycenou komunikaci ve formátu pcap prostřednictvím API a ukládá je pomocí cron jobu. Zachycená data jsou poté přenesena na centralizovaný FTP logovací

server. Webový server také funguje jako motor honeynetu, který spravuje a vytváří honeynet a honeypoty.

Při vytváření honeynetu vytváří webový server novou síť Docker s definovanou podsítí. Pokud není specifikováno, je proveden náhodný výběr, což vede k vytvoření jmenového prostoru sítě pro další spuštění kontejneru.

Nejdůležitější aspekt práce byla jednoduchost přidávání nových honeypotů. Administrátoři pouze potřebují určit obraz honeypot kontejneru a porty, které mají být otevřené do sítě hostitele, aby byl honeypot přístupný mimo hostitele a to je vše. Honeynet vytvoří kontejner se získaným obrazem a zveřejní jej na síti hostitele.

Další klíčovou součástí je monitorovací kontejner, který sleduje komunikaci honeypotu s útočníkem. Spuštěný ve jmenovém prostoru sítě hostitele, získává přístup k Linuxovému mostu, čímž umožňuje veškerou komunikaci do jmenového prostoru sítě honeynetu. Tento proces zajišťuje, že monitoring pokračuje bez povšimnutí, i když útočník získá kontrolu nad kontejnerem honeypotu.

Monitorovací kontejner využívá nástroj tcpdump pro zachycení všech příchozích paketů, které filtruje podle IP adresy kontejneru honeypotu. Tcpdump umožňuje ukládání zachycených dat do souborů pcap, které jsou automaticky odeslány na webový server, když je dosažena určitá velikost souboru nebo časový limit, jenž je nastaven administrátorem.

Design také zahrnuje komplexní logování všech běžících kontejnerů pomocí protokolu syslog. Server rsyslog ukládá komunikaci syslog, která je připojena k webovému serveru a přenesena na centralizovaný webový server pomocí naplánovaného cron jobu. Server rsyslog je integrován do každého jmenového prostoru sítě honeynetu, což umožňuje každému honeypotu odesílat logy na rsyslog pomocí protokolu syslog.

Výsledky

Poslední kapitola podrobně popisuje výsledky bakalářské práce, a to ve dvou odđílech - nasazení ve veřejné síti a analýza zachycených dat.

Honeynet byl nasazen na serveru VUT, připojeném k veřejné síti, aby útočníci mohli přistupovat k honeypotům. Honeynet se skládal ze dvou honeypotů - HTTP a Modbus. Sekce "Deployed Honeypots" 3.1 diskutuje o open-source honeypotech použitých v tomto projektu a o procesu sestavení a sdílení jejich obrazů kontejnerů se systémem honeynet.

Honeynet byl nasazen na serveru VUT po dobu 2 týdnů. Poslední sekce ukazuje skutečnou komunikaci útočníků s kontejnery. Bylo zachyceno okolo 277000 paketů na nichž byla udělána základní analýza, která ukazuje potenciál a nutnost honeypotu v sítích.

NEČAS, Martin. *Modular Honeypot for IT and OT*. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2023, 68 p. Bachelor's Thesis. Advised by Ing. Petr Blažek,

Author's Declaration

Author: Martin Nečas
Author's ID: 231259
Paper type: Bachelor's Thesis
Academic year: 2022/23
Topic: Modular Honeypot for IT and OT

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the paper and listed in the comprehensive bibliography at the end of the paper.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno

.....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

Rád bych poděkoval vedoucímu bakalářské panu Ing. Petru Blažkovi, za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Contents

Introduction	14
1 Theory	15
1.1 Honeypots	15
1.1.1 Categorisation Based on Usage	15
1.1.2 Categorisation Based on Interaction	16
1.1.3 Honeypots vs IDS	17
1.1.4 Honeypot Detection	17
1.1.5 Honeypot Solutions	17
1.2 Information Technologies	19
1.3 Operational Technology	20
1.3.1 Communication OT Protocols	21
1.4 Containers	23
1.4.1 Linux Namespaces	23
1.4.2 Container Networking	24
1.4.3 Container Security	25
1.4.4 Container Images	26
1.4.5 Docker Compose	27
2 Design and Implementation	28
2.1 Initial Design	28
2.1.1 Ansible Usage	28
2.1.2 Creation of Honeypot System	29
2.1.3 Design Problems	29
2.2 Final Design	30
2.2.1 Web Server	31
2.2.2 Monitoring	38
2.2.3 Logging	42
2.2.4 Honeypot Creation Workflow	45
3 Results	50
3.1 Deployed Honeypots	50
3.1.1 Deployment of IT Honeypot	50
3.1.2 Deployment of OT Honeypot	51
3.2 Captured Communication	51
3.2.1 HTTP Honeypot	52
3.2.2 Modbus Honeypot	53

Conclusion	54
Bibliography	55
Symbols and abbreviations	59
List of appendices	61
A Captured Data Graphs	62
B User Manual Installation	64
C Contents of the Electronic Attachment	67

List of Figures

1.1	ISO/OSI model	19
1.2	Operation Technology structure	20
1.3	Containers VS Virtual Machines	23
2.1	Linux bridge monitoring	28
2.2	Honeynet design	31
2.3	Web Server Database	33
2.4	Creation of honeynet	36
2.5	Creation of honeypot	36
2.6	Honeynet overview	37
2.7	Linux bridge monitoring	39
3.1	HTTP Honeypot UI	52
3.2	HTML Form of the captured data	52
A.1	Comparing HTTP and Modbus number of captured packets	62
A.2	The communication grouped by source IP	62
A.3	The most common data key in HTTP request	63
A.4	The most common data value in HTTP request	63

List of Tables

1.1	Overview of Existing Honeypot Solutions	18
1.2	Linux Bridge vs Open vSwitch	25

Listings

1.1	Base image dockerfile	26
1.2	Base docker compose	27
2.1	Crontab Example	34
2.2	Web Server Container Image	37
2.3	Monitoring dockerfile	40
2.4	Tcpdump service code	41
2.5	Rsyslog container image	44
2.6	honeypot.yml.j2	47
2.7	monitoring.yml.j2	49
3.1	Honeypots Dockerfile	51
3.2	Captured modbus communication	53
B.1	Initiating Development Mode	64
B.2	Production docker-compose	65

Introduction

In the modern era, Information Technology (IT) and Operational Technology (OT) devices are constantly threatened by cyber-attacks, underscoring the growing necessity to understand these attacks and the attackers themselves. Honey pots, which simulate systems to study attack methodologies, are instrumental in this learning process. The data collected from these attacks are analysed and utilised in Intrusion Prevention Systems (IPS) to thwart future attacks.

This bachelor's thesis focuses on designing a scalable and modular honeypot system for IT and OT communication protocols. The study is divided into three main sections: a theoretical analysis, the design of the modular honeypot system, and the deployment on real networks and gathered data.

The first chapter is a theoretical exploration that provides an overview of different types of honeypots, categorises them based on their usage and level of interaction, and compares them with Intrusion Detection Systems. It also delves into the intricacies of Operational Technology systems, examining how OT devices communicate. The final part of this chapter defines containers and explains their advantages over Virtual Machines.

The second chapter is dedicated to designing and implementing modular honeypots using containers. It elucidates the construction of a web server to gather all data from the honeypots and the communication mechanisms employed to transfer the data from the honeypots to the web server.

The third chapter describes the deployment of the modular honeypot system. It provides an overview of the honeypots which were used and deployed and it shows the gathered data.

1 Theory

This chapter explores the theoretical aspects of the bachelor's thesis in a detailed manner. It focuses on honeypots, their definition, and how they are categorised. The chapter provides an extensive overview of existing solutions in the field, shedding light on the advancements and challenges in honeypot technology.

Additionally, the chapter discusses the technologies utilised in the design and implementation of the thesis. It covers IT and OT protocols, highlighting their significance and impact. The chapter also explores the use of containers, explaining their scalability and deployment benefits. Furthermore, it touches upon container network communication, explaining how containers interact and exchange information.

1.1 Honeypots

Honeypots are hard to define. They are not a project or solution. The honeypot is a computer system that acts as a decoy to lure cyber attackers and detect, deflect, or study attempts to gain unauthorised access to information systems. Generally, a honeypot consists of a computer that appears to be part of a network but is isolated and monitored and which seems to contain information or a resource of value to attackers. They can provide the attackers' activities to the administrators with information about the attacks. Multiple honeypots in the network create honeynet. [1]

Honeypots can be categorised based on their purpose, including production and research. Additionally, they can be classified based on their interaction level, including low, medium, and high interaction honeypots. [1]

1.1.1 Categorisation Based on Usage

Research Honeypots

Research honeypots are primarily used for gathering information and researching attacker behaviour, new attack techniques, and emerging threats. These honeypots capture detailed data about attackers. Research honeypots often involve high interaction, emulating various services and vulnerabilities to attract attackers. [1]

Production Honeypots

Production honeypots, also known as deployment honeypots or strategic honeypots, are designed to be integrated into an organisation's production environment. These honeypots detect and divert real attacks from critical systems, applications, and

data. Production honeypots are deployed alongside legitimate systems and services, giving the appearance of genuine targets to attackers [1].

1.1.2 Categorisation Based on Interaction

The honeypot categorisation based on interaction refers to the level of engagement a honeypot has with an attacker. This typically falls into three main categories: low-interaction, medium-interaction, and high-interaction honeypots. [1]

Low-interaction

Low-interaction honeypots are designed to emulate the services and vulnerabilities of real systems while providing limited functionality. They are typically lightweight and easy to deploy. Low interaction honeypots simulate a few specific services, such as open ports for common protocols like FTP or HTTP, and respond with emulated responses when accessed by attackers. Since they have minimal functionality, they can capture only basic information about the attacker's activities, such as the source IP address and the type of attack. Low-interaction honeypots are relatively simple and require less maintenance and monitoring than higher-interaction honeypots. [1]

Medium-interaction

Medium interaction honeypots provide a more realistic environment for attackers by simulating a broader range of services and vulnerabilities. They are designed to offer a higher level of interaction, allowing attackers to interact with various emulated services in a controlled environment. These honeypots may provide limited functionality and responses to attackers, enabling them to gather more detailed information about attacker behaviour, techniques, and tactics. Medium interaction honeypots balance realism and security, providing a more enticing target for attackers without exposing the underlying systems. [1]

High-interaction

High-interaction honeypots are the most complex and resource-intensive type of honeypots. They are designed to fully simulate real production systems and provide attackers with an environment that closely mimics genuine network services and applications. High-interaction honeypots are typically deployed using real operating systems and applications, making them highly authentic targets for attackers. The attackers can access them but do not affect the internal network. [1]

1.1.3 Honeypots vs IDS

Honeypots and Intrusion Detection Systems (IDS) are both cybersecurity tools used to enhance network security, but they serve different purposes and have distinct characteristics.

Honeypots are primarily focused on detecting attacks and gathering information about attackers. They act as decoy systems designed to deceive attackers and simulate vulnerable systems or services. By luring attackers away from production systems, honeypots provide valuable insights into attacker techniques, tools, and motives. [1]

On the other hand, IDS is designed for real-time intrusion detection. It monitors network traffic or system events to identify suspicious or malicious activities. IDS employs a combination of signature-based and behaviour-based detection methods [2]. However, IDS may suffer from false negatives when it fails to detect an attacker utilising a new exploit. False positives can also occur in IDS when regular network traffic is mistakenly flagged as an attack, leading to the dropping of legitimate packets. [1]

Honeypots address the limitations of IDS by being isolated from the production network. Since all communication with honeypots is with potential attackers, any activity detected can be assumed to be malicious, reducing the likelihood of false positives. Honeypots provide a complementary approach to IDS, capturing and analysing attacker interactions rather than real-time alerting. [1]

1.1.4 Honeypot Detection

Attackers use Honeypot detection tools to identify and avoid traps designed to deceive and gather information about their activities, thereby minimizing their risk of detection and countermeasures. One of these tools is the Honeypot Hunter.

The Honeypot Hunter is a tool designed to evaluate open proxy connectivity by performing a series of tests, including setting up a false mail server and attempting to proxy back to it to classify proxies as safe, failed, or honeypots [3].

To prevent honeypot detection, measures can be taken to make the honeypot appear as realistic as possible. This includes ensuring the system has a believable fingerprint, mimicking normal network traffic patterns, and employing defensive mechanisms to deter attackers from further investigating the honeypot. [4]

1.1.5 Honeypot Solutions

Table 1.1 contains already-made honeypot solutions, which are open-source and can be found on GitHub. This bachelor's thesis focuses on IT and OT protocols.

From the OT honeypots, the thesis implementation uses the **conpot** project, which has a GPL-2.0 license which allows copying the code and making changes but does not allow changing the original license. From the IT honeypots, the thesis implementation uses the **honeypots** project, which has an AGPL-3.0 license which also allows us to modify the code and can emulate up to 25 IT protocols such as dns, ftp, httpproxy, http, https, imap, mysql, pop3, postgres, redis, smb, smtp, socks5, ssh, telnet etc.

Table 1.1: Overview of Existing Honeypot Solutions

Name	Interaction	Function
conpot ¹	Low	Simulates OT devices
GasPot ²	Low	Simulates a Veeder Root Gaurdian AST
dicompot ³	Low	Simulates Digital Imaging and Communications in Medicine (DICOM) Honeypot
medpot ⁴	Low	Honeypot that simulates HL7 / FHIR
HoneyPLC ⁵	High	Hooneypot designed to simulate multiple PLC models from different vendors
mailoney ⁶	Low	Simulates SMTP server
honeypots ⁷	Medium	Simulates up to 25 IT protocols
cowrie ⁸	Medium to high	SSH and Telnet honeypot
Heralding ⁹	Low	Honeypot that collects credentials
HoneySAP ¹⁰	Low	Research-focused honeypot specific for SAP services
HellPot ¹¹	Low	Honeypot which sends to attackers endless stream of random data
RDPY ¹²	Medium	Simulates the Microsoft Remote Desktop Protocol (RDP) protocol
WebTrap ¹³	Low	Create deceptive webpages

¹<https://github.com/mushorg/conpot> ²<https://github.com/sjhilt/GasPot>

³<https://github.com/nsmfoo/dicompot>, ⁴<https://github.com/schmalle/medpot>,

⁵<https://github.com/sefcom/honeyplc>, ⁶<https://github.com/phn3has/mailoney>,

⁷<https://github.com/qeeqbox/honeypots>, ⁸<https://github.com/cowrie/cowrie>,

⁹<https://github.com/johnnykv/heralding>, ¹⁰<https://github.com/OWASP/HoneySAP>,

¹¹<https://github.com/yunginnanet/HellPot>, ¹²<https://github.com/citronneur/rdpy>,

¹³<https://github.com/IllusiveNetworks-Labs/WebTrap>,

1.2 Information Technologies

Information Technology (IT) uses computers, storage, networking and physical devices to create, process, store and transfer data. It is the backbone of modern digital systems, enabling various industries and sectors to operate efficiently and effectively [5].

IT protocols are a set of rules, either formatting or processing data. The protocols help to standardise the IT world. Despite differences in hardware or software configurations, IT protocols provide a common language that enables different devices and platforms to interact and exchange information [5].

This communication is described in Open Systems Interconnection (OSI) model shown in Figure 1.1. OSI is a model which splits communication into seven layers. Each layer describes a part of the IT network communication system.

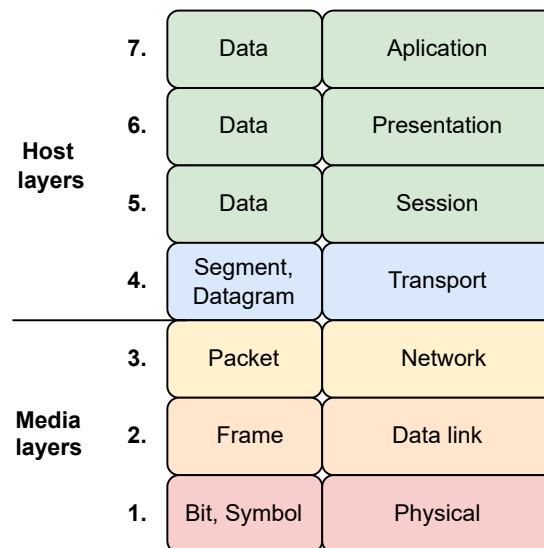


Figure 1.1: ISO/OSI model

1. Physical layer transmits raw bits or symbols over medium [5].
2. Data link layer transfers data in frames between connected hosts with physical links. Example protocols **ARP**, **Frame relay**, **L2TP**, etc [5].
3. Network layer bundles data in packets, handles host addressing, and determines paths, logical addressing and routing. Example protocols **IP**, **ICMP**, **RIP**, **OSPF**, etc [5].
4. Transport layer allows a reliable data transfer; it establishes and maintains the connections. Example protocols **TCP**, **UDP**, etc [5].
5. Session layer maintains connections between applications. Example protocols **NetBEUI**, **RTCP**, **RPC**, etc [5].

6. Presentation layer standardises application communication and provides cryptography and data compression services. Example protocols **GIF, ASCII, JPEG, MIDI**, etc [5].
7. Application layer allows interfacing the user application with network flows. Example protocols are **HTTP, HTTPS, DNS, SSH, SMTP, SFTP, SNMP**, and more [5].

This bachelor's thesis focuses on the Application layer protocols.

1.3 Operational Technology

Operational Technology (OT) refers to the hardware and software used to change, monitor or control the enterprise's physical devices, processes, and events. This technology is found across various industries, such as manufacturing, oil and gas, and utilities. The Operation Technology networks are made of Industrial Control Systems (ICS), Supervisory Control And Data Acquisition (SCADA), Programmable Logic Controllers (PLC), Discrete Process Control systems (DPC), and Remote Terminal Units (RTU). This structure is shown in Figure 1.2. [6]

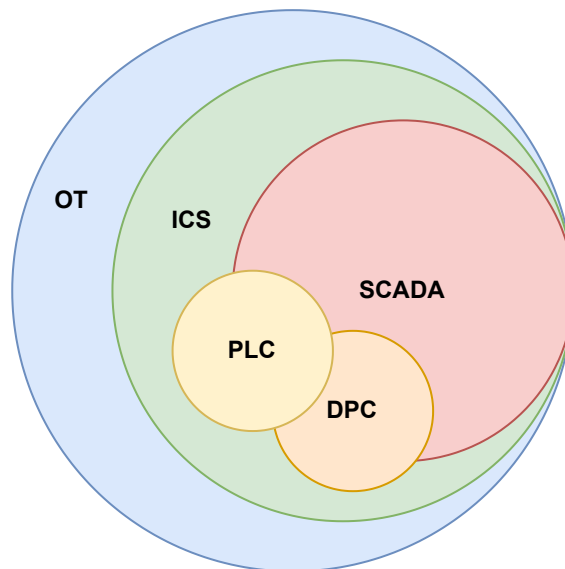


Figure 1.2: Operation Technology structure

The ICS are the centre of Operational Technologies. They are systems of monitoring and controlling industrial processes, for example, electrical grids and building alarm information systems. Industrial control systems mainly focus on high-availability and mission-critical applications. Since operational technologies are used in these areas, the most important is control and safety of the systems. [7]

The SCADA systems display the system under control and allow control of each unit. These units comprise programmable logic controllers and discrete process control systems. SCADA systems provide a graphical interface that presents real-time data, such as temperature, pressure, and flow rates, enabling operators to make informed decisions and take necessary actions for optimal system performance. [7]

The PLC is a controller with digital or analogue inputs, digital or analogue outputs and a communication protocol describing how the controller communicates. PLCs play a crucial role in process automation by collecting sensor data, processing it based on programmed instructions, and triggering appropriate outputs to maintain desired system behaviour and operational efficiency. [8]

The RTU are devices used in industrial automation systems to collect and transmit data from remote locations to a central control system. They act as intermediaries, gathering information from sensors and equipment, and relaying it via communication networks, enabling efficient monitoring and control of remote operations. [7]

1.3.1 Communication OT Protocols

There are many communication protocols in operational technology, such as **Modbus**, **EtherNet/IP**, **Profibus**, **Modbus**, **Interbus**, **ProfiNet**, and more [9]. This Bachelor thesis focuses on the Modbus protocol.

Modbus

Modbus is a data communication protocol that uses a request-response model. The Modbus is based on server and client topology. The server sends the requests to the clients to do something, and the clients respond to the request. It is one of the most widely used protocols in operational technology, especially manufacturing. The Modbus is primarily used in communication between supervisory control, data acquisition systems, sensors, and programmable logic controllers. The Modbus protocol is open source, easy to use and reliable for transferring data. [10]

Modbus RTU

Modbus Remote Terminal Unit (RTU) is the most common serial transmission protocol. There are two types, the Modbus RTU, which transmits the data in binary and the Modbus American Standard Code for Information Interchange (ASCII), which sends data in plain text. That makes the Modbus ASCII less secure than Modbus RTU. The Modbus RTU should be used unless the communication can not support the binary transfer. In that case, the Modbus ASCII should be used. The

Modbus RTU and Modbus ASCII are connected with a point-to-point connection, which provides a channel for communication between two ports. Modbus RTU can only have one client and up to 247 server devices. The Modbus RTU networks are made of EIA-approved RS-485, RS-422, or RS-232 physical standards describing serial communications. [10]

Modbus TCP

Modbus TCP is an industrial Ethernet protocol that uses TCP/IP at the transport layer, as shown in Figure 1.1. It solves the problems of Modbus RTUs by using the ethernet instead of serial links. Communication is faster, can be sent over longer distances and can be used in the IT infrastructure. It also solves the issue of multiple devices in one network. [10]

There are two types of Modbus TCP. First, there is the Modbus TCP and Modbus over TCP. The Modbus TCP uses the Modbus packets in the TCP layer. Modbus over TCP is Modbus RTU packet over TCP, and this Bachelor thesis focuses on the Modbus TCP packets. [10]

The Modbus TCP comprises of Modbus application protocol (MBAP) header, a 7-byte header. The header identifies the Modbus Application Unit (ADU) that is used. The Modbus standard uses TCP port 502. [10]

Secure Modbus

In 2018, the Modbus Security protocol was published. The Modbus Security protocol does not change any specification of Modbus but defines the usage of Modbus with Transport Layer Security (TLS) and with certifications [11].

Industry 4.0

Lately, OT and IT are slowly overlapping each other. This overlap creates OT systems which can run on IT-based networks, creating Industry 4.0. One of the examples is the Modbus TCP protocol, as mentioned in Section 1.3.1, which is used for industrial operations but uses the ethernet and can communicate over the IT infrastructure [12].

1.4 Containers

A container represents software operating on an existing operating system [13]. This software introduces an independent layer that doesn't interfere with the host operating system but instead encapsulates resources like storage, CPU, and networking capabilities [14].

Containers are designed to execute programs and bundle all essential dependencies, ensuring no disruption to the host operating system on which they operate. Docker represents a commonly used command-line interface for managing containers, providing functionalities such as creating new containers, building container images, and interacting with these containers. [13]

When comparing containers to Virtual Machines (VMs), as demonstrated in Figure 1.3, distinct differences become apparent. VMs generate an abstraction layer over physical hardware, facilitating the operation of multiple servers on one machine. However, each VM requires a full operating system layered over the host operating system, resulting in potentially excessive storage use. [13, 14]

The containers still need some host operating system. Instead of the guest operating system, it uses a container engine such as the Docker Engine, which allows the programs to run on top of the host operating system but with an extra separation layer. [13]

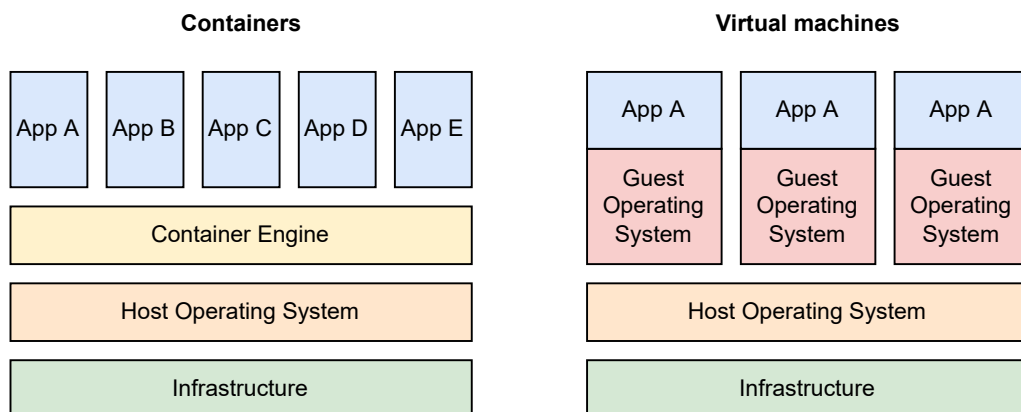


Figure 1.3: Containers VS Virtual Machines

1.4.1 Linux Namespaces

Linux namespaces offer a method for abstracting global system resources, creating an illusion that processes within a namespace have their isolated instances of those resources. As a result, modifications to a resource are visible to processes within the

same namespace but not to those outside it. Linux namespaces are widely used in container implementation. [15]

Several types of Linux namespaces exist, with the most common ones listed below:

- Cgroup – Limit and monitor the resources
- IPC – System V IPC, POSIX message queues
- Network – Network devices, stacks, ports, etc.
- Mount – Mount points
- PID – Process IDs
- Time – Boot and monotonic clocks
- User – User and group IDs
- UTS – Hostname and NIS domain name

Each namespace type offers distinct isolation levels, enabling containerised applications to operate independently from one another and the host system. [15]

1.4.2 Container Networking

A key aspect of containerisation is the capacity to isolate networking resources and functions using Network namespaces. Network namespaces provide isolation from the host network, which is crucial for security and stability. However, they do not offer any additional capabilities beyond basic network isolation. [16]

A virtual switch is required to manage communication between network namespaces to facilitate packet forwarding and other advanced networking functionalities. The Linux bridge is a widely-used virtual switch that functions as a network switch and oversees packet forwarding between connected interfaces. Its primary use involves forwarding packets on routers, gateways, and between VMs and network namespaces on a host. [17]

This bachelor's thesis concentrates on container networking and the capture of communication between containers and attackers. Containers are situated in isolated network namespaces, ensuring they remain separate from the host networks.

Veth devices are virtual Ethernet devices capable of acting as tunnels between network namespaces, forming a bridge to a physical network device in another namespace. They can also function as independent network devices. [18]

Linux Bridge vs Open vSwitch

An alternative to the Linux bridge is Open vSwitch (OVS), which supports the Switched Port Analyzer (SPAN). OVS is particularly appealing for this thesis due to its SPAN port compatibility. [19]

Table 1.2 contrasts the Linux bridge with OVS. OVS supports SPAN ports, which mirror incoming or outgoing communication from the switch, making it an ideal tool

for monitoring incoming packets from attackers [19]. However, since Docker does not support OVS by default but uses bridge [20], manually creating an OVS bridge for each container is the only option, rendering it impractical.

Table 1.2: Linux Bridge vs Open vSwitch

Support	Linux Bridge	Open vSwitch
Default implementation in Docker	yes	no
Switched Port Analyzer	no	yes
Adding extra connections	yes	yes
Kernel module	yes	no

Consequently, the decision was made to use the Linux bridge, which lacks SPAN port support. An alternative is to monitor communication passing through the veth on the host machine, the default gateway inside the namespace. This way, tcpdump collects all communication and filters it by destination IP within the namespace.

Bridge Network vs Host Network

The Bridge network is the default network driver for Docker. When an application runs in a container and needs to be accessible on a network, it's commonly deployed in a bridge network. This network is isolated from the host, meaning that the application won't have access to its network but can communicate with other applications in the same network. This adds a layer of security and allows better control over network resources. [20]

On the contrary, the Host network driver removes the network isolation between the Docker host and the Docker containers. This means a container can directly access the host's network. However, this also means that the container can read and send all traffic to the host, which can be a security risk. [21]

1.4.3 Container Security

Container security relies on various components to ensure the overall security of containers. These components include Control Groups (cgroups), Linux Kernel Capabilities, Linux namespaces, and more. Control Groups allow Docker to manage resources and isolate containers, thereby enabling efficient allocation and preventing any potential abuse of resources. Linux Kernel Capabilities play a crucial role in limiting container privileges, thereby reducing the attack surface and restricting access to critical operations. Additionally, Linux namespaces provide isolation for various aspects of a container's environment, including processes, networking, and file systems. Through the utilisation of these technologies, Docker achieves strong

isolation, precise resource control, and fine-grained privilege management, resulting in enhanced security for containerised environments. These measures contribute to an overall secure ecosystem. [22]

This bachelor's thesis uses the Linux Kernel Capabilities to restrict the capabilities of honeypots, limiting the actions that containers can perform in the case that an attacker gains access to the honeypot container. By implementing these restrictions, the thesis aims to mitigate the potential damage caused by unauthorised access, ensuring that the containers within the honeypot environment remain contained and do not pose further risks. The use of Linux Kernel Capabilities serves as a valuable security measure, providing an additional layer of protection and control within the honeypot system.

1.4.4 Container Images

A container image is a lightweight, standalone, executable package that includes everything needed to run the software, including the code, a runtime, libraries, environment variables, and config files [13].

Docker images are built from Dockerfiles. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image [23]. The following list describes the Listing 1.1, which shows an example of a simple application.

- **FROM** initialises a new build stage and sets the Base Image for subsequent instructions [23].
- **COPY** copies new files or directories from `<src>` and adds them to the filesystem of the container at the path `<dest>` [23].
- **RUN** will execute any commands in a new layer on top of the current image and commit the results [23].
- **CMD** provides defaults for an executing container [23].

Listing 1.1: Base image dockerfile

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

The Docker image can be built using the command:

`docker build -t image_name` [24]. Once built, the Docker image can be pushed using the command: `docker push registry:port/name:tag`. The image must be tagged with the registry path name, as shown in the command:

`docker image tag image_name registry:port/image_name:tag` [25].

Container Registries

A container registry is a server application that stores and distributes Docker images [26]. These registries can be accessed via the Docker Command Line Interface (CLI) or other tools using the `docker push` and `docker pull` commands to upload and download images [25, 27].

1.4.5 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services and performs the creation and start-up process of all the containers with a single command. [28]

The Docker Compose uses file `docker-compose.yml`. The `docker-compose.yml` file is a YAML file defining services, networks, and volumes for a Docker application. [28]

Figure 1.2 shows a basic structure of a `docker-compose.yml` file.

Listing 1.2: Base docker compose

```
version: '3.1'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
  database:
    image: postgres:latest
    environment:
      POSTGRES_PASSWORD: example
```

The Docker Compose file defines two services: `web` and `database`. The `web` service uses the latest nginx image and maps the container's port 80 to the host's port 80. The 'database' service uses the latest PostgreSQL image and sets an environment variable with the key `POSTGRES_PASSWORD` and value `example`.

2 Design and Implementation

This chapter describes the design development stages of the bachelor’s thesis. The initial design used VMware ESXi hosts but encountered challenges regarding using Virtual Machines (VMs) and licensing. The chapter’s second section describes the final design using containers, and it shows all components, such as the web server, monitoring and logging.

2.1 Initial Design

The first version of the bachelor’s thesis design did not integrate containers and went through various design phases. The initial design shown in Figure 2.1 used VMware ESXi hosts to manage VMs. One of the reasons to use VMware was the default integration of the Open vSwitch [29], which can create SPAN ports that mirror the communication [19]. The design consisted of a management VM creating honeypots, monitoring attackers’ communication, and sharing the information with the administrator.

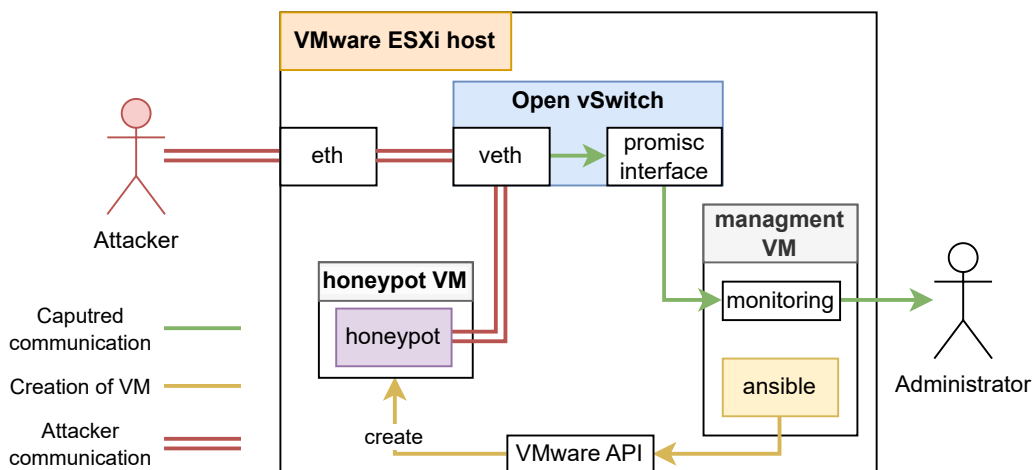


Figure 2.1: Linux bridge monitoring

2.1.1 Ansible Usage

Administrators use Ansible playbooks to define the desired state of the server. These playbooks contain tasks, each of which includes modules, and each module is essentially a Python script that manages a specific action. The characteristic of Ansible that ensures the server remains in the state defined in the playbook is its idempotency. This allows for consistency and reproducibility in the management of servers. [30]

The VMware ESXi server communicates with an API (Application Programming Interface) via the VMware Software Development Kit (SDK). Ansible interacts with this API, sending requests to create the VM, upload the disk, and perform other tasks. [31]

2.1.2 Creation of Honeypot System

The initial step in the honeypot creation process involved the management Virtual Machine (VM). This VM would manage the whole honeynet system and the ESXi host. All captured data would be stored on this VM.

After deploying the honeynet from the VM web server, the web server would start the Ansible playbook to create the necessary virtual switch in promiscuous mode and the one promiscuous interface attached to the web server VM through which the communication would be monitored. Once everything is configured, the playbook will upload a pre-constructed disk containing all necessary honeypot dependencies. This disk upload process was time-consuming as it required the upload of the entire operating system, highlighting the advantages of using containers in the final design. After the VM is ready, Ansible would add the VM to its inventory and start another playbook inside it, deploying the honeypot.

2.1.3 Design Problems

The VMware ESXi was chosen for its free license, which led to some compromises, such as creating a virtual switch in promiscuous mode instead of the SPAN ports because the SPAN ports are supported in the vCenter, which would require additional licences [32, 33]. The VMware ESXi free license does not support the API communication needed to deploy the honeypot system with Ansible.

An alternative could be the free and open-source virtualisation oVirt, which uses the Open vSwitch to switch virtualisation. It also supports the monitoring interfaces for communication, making it a better fit than VMware ESXi. [34]

Ultimately, the design changed to use containers instead of Virtual Machines, removing the extra necessary operating system overhead and making it faster to deploy.

2.2 Final Design

This bachelor's thesis aims to design and implement a modular honeypot system to collect vast amounts of data from attacks akin to the Research Honeypots discussed in Section 1.1.1. The system must capture all communication with the honeypots to achieve this. The design needs to be modular to enable easy extension and installation. The containers solve both of these criteria. The final design, illustrated in Figure 2.2, consists of several components.

The final designs took inspiration from the "A dynamic honeypot design for intrusion detection" article [35]. The design share usage of the Administrator Web Interface to manage the honeynet system, log the honeypot events and monitor the ethernet segment communication. This design also takes the design step further, implementing the network namespaces for running multiple honeynets on one host, separating all components into containers and gathering all information on a centralized remote server.

The web server orchestrates the honeypot system, storing all packets captured by the monitoring container before forwarding them to the central logging server. The web server features a User Interface (UI) that allows administrators to interact with, manage, and monitor the honeypots. The UI enables administrators to create a new honeypot system. Additionally, the web server facilitates CRON export, transmitting captured data to the centralised logging server based on a predetermined CRON schedule. Finally, the web server includes an Application Programmable Interface (API) that accepts captured data from the monitoring containers for storage in the database and on the web server.

The syslog server archives system logs from all honeypot, monitoring, and web server containers. The syslog server logs all data to a file as a container. These stored files are mounted to the web server, enabling the web server and the syslog container to access the logs for automatic export to the centralised logging server. The syslog server container's network interface is incorporated into each honeynet network namespace, allowing all honeypots to send log messages to the server. This setup results in a single syslog server aggregating data from all active containers.

The honeypot container contains a service exposed to the external host network via a specific port, enabling communication with potential attackers. All honeypots are initiated with the syslog server address, to which all standard output will be logged. Administrators only need to specify the container image address, and the web server will integrate it into the container network namespace.

Each honeypot is accompanied by a monitoring container, which initiates tcpdump on the bridge interface. All communication to the network namespace passes through this interface, and the honeypot IP address filters the communication. Tcp-

dump dispatches the captured data either when the volume of captured data surpasses the set limit or when the predetermined timeout expires. After sending the data to the web server, tcpdump is restarted.

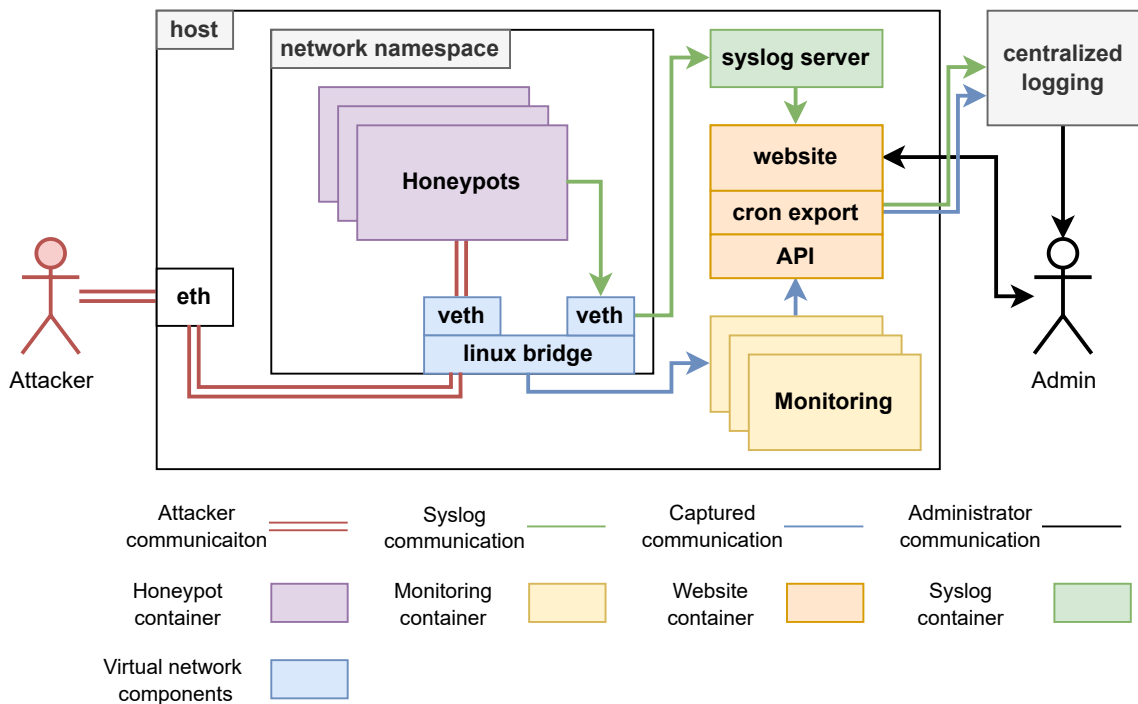


Figure 2.2: HoneyNet design

2.2.1 Web Server

The web server is the centre of the whole honeypot system. It serves multiple purposes, such as the User Interface for the administrator and the database of stored packets. It also creates the whole honeypot system using the docker-compose tool.

The web server needs to manage the host containers and container networks. This was one of the issues encountered during the design. One potential solution was to run the web server directly on the host operating system rather than within a container. However, this solution was not chosen due to the increased complexity it would introduce to installing the honeypot system. Eventually, the issue was resolved by mounting the docker socket to the web server container. This way, the web server container is able to access and manage the host containers and host container networking.

Django Python Framework

The web server was designed to use Django Python Framework to handle web server logic and incoming requests and generate dynamic web pages. Django is a high-level, open-source Python web framework [36].

Key features and capabilities of Django include:

- **Model-View-Controller (MVC) Design Pattern:** Django uses the MVC architectural pattern, a practical system for managing intricate applications. It divides an application into three interlinked components: the Model, the View, and the Controller. This separation allows the design to modify individual elements without impacting others [36].
- **Object-Relational Mapping (ORM):** Django's ORM enables developers to interact with the database as SQL, providing a Pythonic interface for creating, retrieving, updating, and deleting records in the database. The ORM helps prevent SQL injection by providing parameterised queries, where user input is treated as data rather than executable code [36].
- **Admin Interface:** Django offers a built-in administrative interface that is highly customised and ready to use. This admin interface offers a user-friendly UI for managing the data within your application [36].

Database

One crucial server component is the database, which organises and stores various data. The database consists of seven tables, as depicted in Figure 2.3, each serving a specific purpose:

- **Honeynet:** This table represents multiple honeypots within the same network namespace. It allows the administrator to specify the subnetwork of the network namespace.
- **Honeypot:** The Honeypot table contains essential data for each honeypot, including details such as the image used to start the container and the exposed port on the host network. Additionally, it specifies tcpdump parameters such as filters, the maximum size of the pcap file, timeout for sending pcaps to the web server API, and any extra arguments required by the administrator.
- **HoneypotSyslog:** Each honeypot in the network namespace has an assigned address for the Syslog server, which is stored in this table.
- **HoneypotExport:** This table specifies the address, username, password, and path where captured data should be stored. It also includes a crontab entry to define the frequency at which the data should be sent.
- **AttackDump:** The AttackDump table maintains the paths to the captured pcap files received by the server. It stores the location of the pcap file within

the web server container and the timestamp indicating when it was received.

- **HoneyPotAttack:** Honey pots can send specific data to the web server in JSON format. This table stores the received JSON data from the honeypots.
- **Attacker:** When a honeypot sends data via the HoneyPotAttack, it may include the IP and MAC address of the attacker. The Attacker table stores this information.

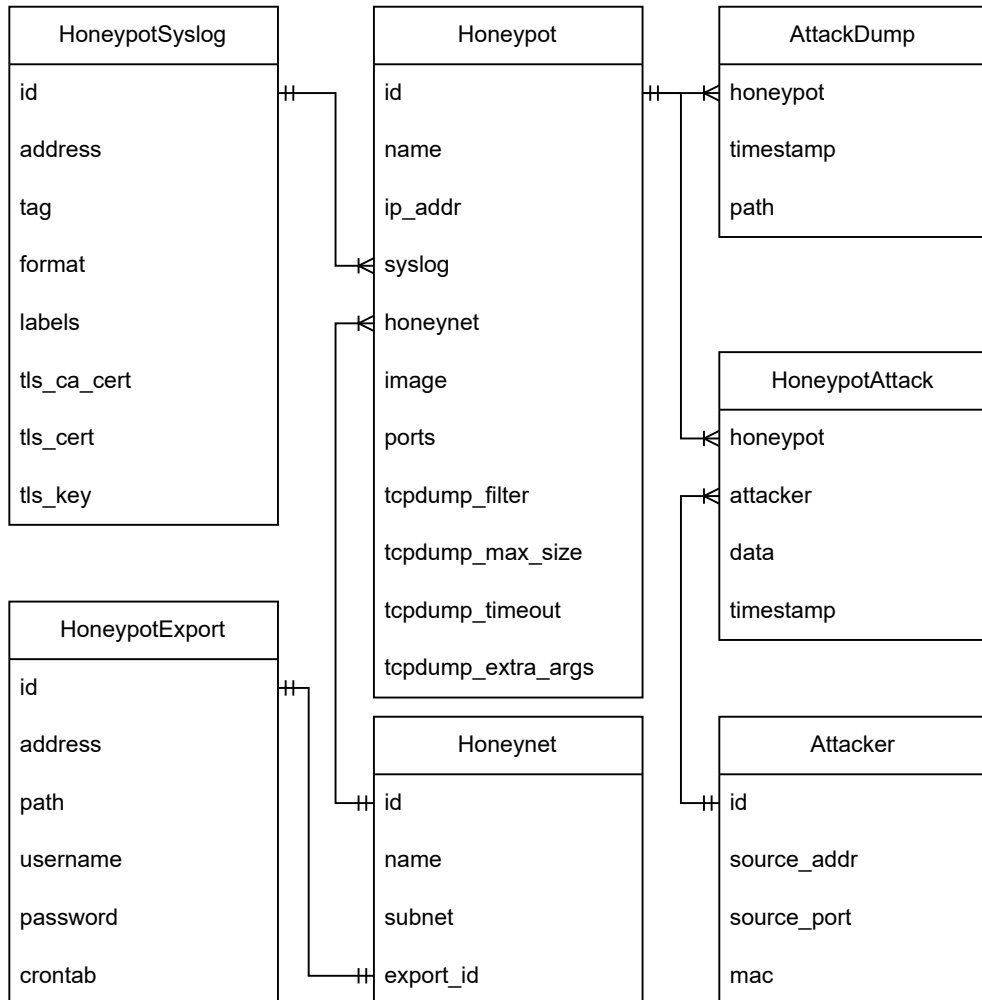


Figure 2.3: Web Server Database

By default, Django uses the SQLite database, which stores all data within a file. This feature is beneficial during web application development as it requires no prerequisites or database setup. However, it poses a setback during deployment with a container. If a container utilising the SQLite database is restarted, all data could be wiped out. [37]

A remote database, specifically the PostgreSQL database, was selected to circumvent this issue. PostgreSQL is a free, open-source relational database management system. [38]

The PostgreSQL database server can operate externally to the host, requiring network access from the web server. However, PostgreSQL was set up within another container the Django web server could connect to for this specific implementation. This arrangement allows the Django web server to be restarted and updated with new code while ensuring data persistence.

Cron Export

In this bachelor's thesis, Cron has been used to automate exports to external storage. Administrators specify the crontab format to determine the frequency of data exports, and the web server starts this operation as a background process [39]. The export sends all captured pcap files, honeypot syslogs, and specific data from the database. Once all data are successfully sent to the external storage, they are deleted from the web server to free up storage for more data.

The Cron is a fundamental technology within UNIX. Operating as a background system daemon, it executes commands according to a predetermined schedule [39]. To define the timing for script execution, Cron uses files referred to as 'crontabs' [40]. The crontab uses the format `minute hour day month day script`. Examples of this format are displayed in Listing 2.1.

Listing 2.1: Crontab Example

```
5 4 * * * script.sh # AT 04:05.  
0 */4 * * * script.sh # At minute 0 past every 4th hour.
```

Application Programming Interface

The API is crucial for facilitating interactions with honeypots. Each monitoring container communicates with the API by sending data to its web service endpoints. The API is designed with multiple endpoints, each serving a different purpose. The list of the API endpoints utilised by the honeypot:

- **honeypots/<uuid>/upload** – This endpoint is used for uploading pcap files from tcpdump.
- **honeypots/<uuid>/attack** – This endpoint is used for uploading specific data from the honeypot.

The API uses the Django Rest Framework (DRF), a powerful and flexible toolkit for building Web APIs. It's a modular, flexible, customised API framework built on Django and Python. [41] Key features of Django Rest Framework include:

- **Web browsable API:** DRF provides a user-friendly web interface to navigate the API, making it easy to understand and test [41].

- **Authentication & Authorisation:** DRF comes with multiple authentication methods and provides a robust and customised system for managing API access [41].
- **Serialisation:** DRF provides a powerful serialisation engine compatible with ORM and non-ORM data sources, effortlessly handling complex and nested JSON structures [41].

Authentication and Authorisation

The API uses Authentication and Authorisation to allow only existing users to upload data to the web server. The web server creates a user for each monitoring container. This way, even if the attacker gets access to the monitoring container, they could only upload data but not get, remove or change any data. So the honeypot would still know the attacker's steps and how he got to the system.

In the design, every monitoring container initiates with a secret token passed in an environment variable named `TOKEN`. This token is critical for authentication, acting as a unique identifier to validate the honeypot's identity when interacting with the API.

Once authenticated, authorisation rules come into play. In the system, each honeypot user is granted permission only to send data to the honeypot associated with the same UUID as the user. This restriction ensures that each user can interact only with the intended honeypot, enhancing the security and data integrity of the system.

User Interface

The User Interface (UI) is created for administrators' easy usage of the honeypot system infrastructure. The UI of the system uses the Bootstrap framework for its responsive design capabilities.

The administrators can create the honeypot system in the UI. They can specify the name of the honeynet, the network namespace's subnetwork and the auto export to the remote storage server. The UI of this action is shown in Figure 2.4.

In the honeynet, the administrators can create a honeypot with a name and container image containing the honeypot itself. The administrator can specify the networking of the honeypot, such as the IP address with which the container should be started and the exposure of ports, which creates a tunnel to the host network on a specific port. The last parameters the administrators can specify for the honeypot are the `tcpdump` parameters. They can specify the filter of the listening packets. The timeout at which the capturing packets should be stopped, sent to the web server API and started again. The max file size specifies the threshold on the size

Honeynets

Signout

Add

Name: honeynet-1 **Subnet:** 192.168.100.0/24

Logging

Auto export

Address: 127.0.0.1:/ftp/alpineftp/test **Crontab:** 0 0 * * *

Username: alpineftp **Password:**

Submit

Figure 2.4: Creation of honeynet

of the log/network-trace files that when reached, the file is sent to the web service API. The last parameter of tcpdump is the tcpdump extra arguments for any extra parameters that allow the monitoring container to run. The UI of this action is shown in Figure 2.5.

Honeynets

Signout

honeynet-1

Add

Create honeypot

***Honeypot name:** honeypot-http ***Docker image:** quay.io/mnecas0/honeypots

Networking

IP: **Expose ports:** 80:8080

Tcpdump

Tcpdump filter: dst port 80 **Tcpdump timeout:** 3600

Tcpdump max file size: 100 **Tcpdump extra args:**

Submit

Figure 2.5: Creation of honeypot

Once the honeynet with the honeypot is created, it has an overview of the honeypots in the graph shown in Figure 2.6. The graph is made with the 'AnyChart' JavaScript library. The AnyChart is a JavaScript library that provides a wide range of powerful and customisable charts. In the graph are shown honeypots with their IP in the network namespace. The administrators can click on the honeypot in the graph to view its details as shown in Figure 2.5 and to show all its captured data.

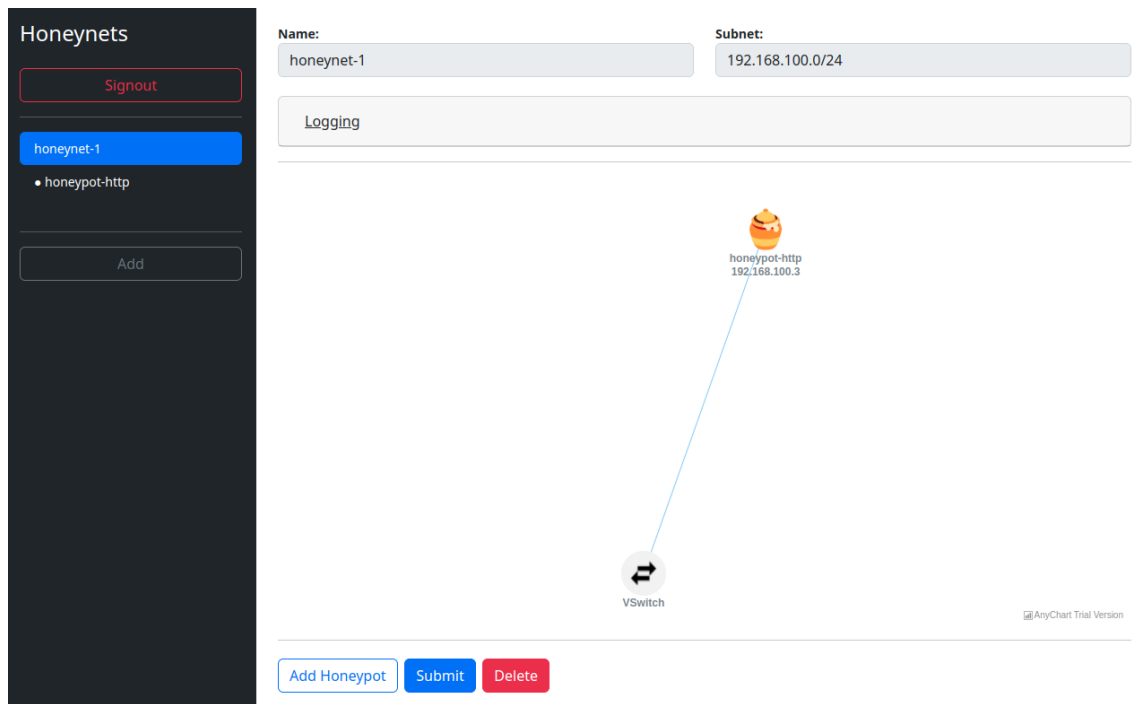


Figure 2.6: Honeynet overview

Container Image

The web server is running in a docker container, so it needs to have the docker file to specify how it should be installed and started. The docker file also specifies the dependencies of the whole project. This provides easy installation on any system-supporting containers. The administrators do not need to install any dependencies but simply start the container. The Listing 2.2 shows the docker file.

Listing 2.2: Web Server Container Image

```
FROM ubuntu:latest

COPY . /code/
WORKDIR /code

ENV DEBIAN_FRONTEND=noninteractive
RUN apt update -y
RUN apt install python3 python3-pip cron libpq-dev -y
RUN pip3 install -r requirements.txt

# Install docker and docker-compose
RUN apt install docker docker-compose -y
```

The docker file is made of multiple parts:

- The Docker file starts by using the latest version of the Ubuntu base image using the `FROM` instruction.
- The `COPY` instruction is then used to copy the honeynet project directory (`.`) to the `/code/` directory in the container.
- The `WORKDIR` instruction sets the working directory to `/code/`.
- The `ENV` instruction sets three environment variables. The `DEBIAN_FRONTEND` is set to `noninteractive`, sets the front end to non-interactive mode, and disables any prompts during package installations.
- The `RUN` instruction updates the package index and installs several packages including `python3`, `python3-pip`, `cron`, and `libpq-dev`. The `python3` package is needed for running the Django web server. The `python3-pip` is a Python package manager needed for installing the Django framework and its Python dependencies. The `cron` package is needed to export the gathered data automatically. The last required system package is the `libpq-dev`, which contains the required PostgreSQL system libraries. These are needed for the web server connection and communication to the remote database.
- The `RUN` instruction then installs the packages listed in the `requirements.txt` file using `pip3`. Which installs all the web server Python dependencies such as Django web server, Django REST API, `jinj2` for the jinja templating and more.
- Finally, the `RUN` instruction installs Docker and Docker Compose by installing the `docker` and `docker-compose` packages using `apt`. Which are needed management of the host containers and the creation of the honeypot system.

2.2.2 Monitoring

The monitoring needs to capture all incoming and outgoing communication to the honeypot. The communication needs to be sent to the web server API. This monitoring is represented in Figure 2.7.

The monitoring is also a container as all the honeynet components. The container must monitor the Linux bridge interface connected to each honeynet network namespace. By default, the docker containers are started in their network namespace. The administrators can specify the network namespace the container should be started with.

One solution could be adding the mirroring container to the network namespace of the honeynet. This solution was not chosen for security reasons. If the attackers could get inside the honeypot container, they could scan the network namespace and see the mirroring container. Another reason why this solution was not chosen

is that the monitoring container needs to send the captured data to the web server API. For this, the monitoring container needs to have network access to the web server, and the web server would need to be inside the network namespace. This would also expose the web server to attackers and allow further attacks.

The solution which was chosen was that the monitoring container would run in the host network mode. The mode allows the container to access the host network stack. This allows the container to monitor the host bridge interface, which is connected to the network namespace through which all communication goes to the network namespace. This way, the monitoring container will not be visible to the honeypot container inside the network namespace. The web server must also run with the host network mode so the monitoring container can easily send data to the web server API.

There is one running monitoring container for every honeypot, which filters the communication by the honeypot IP address. This splits the communication across multiple containers. Each container sends the captured data to its corresponding API endpoint.

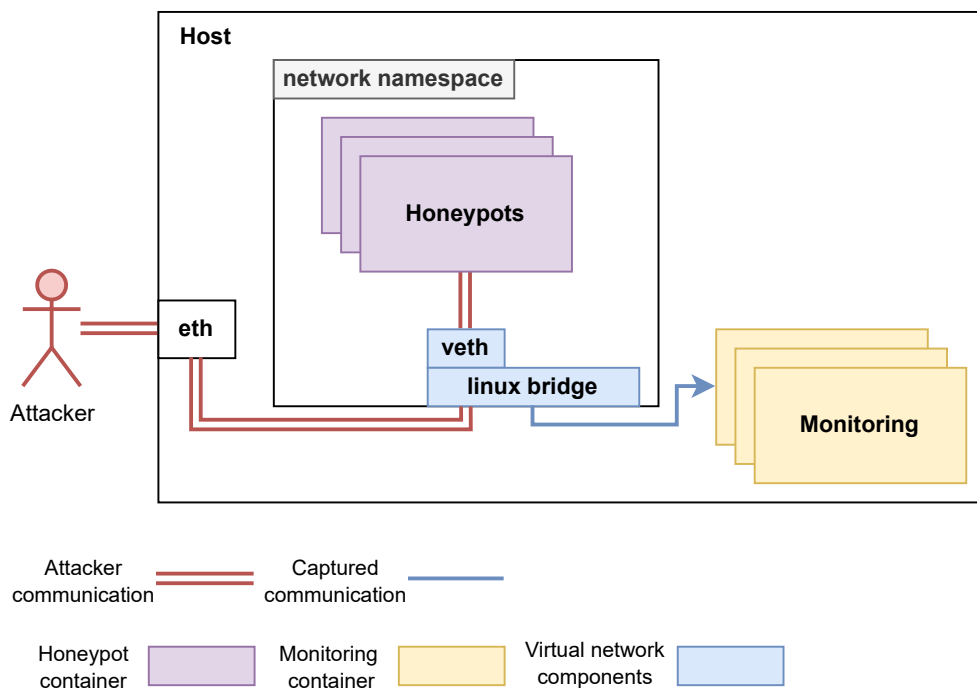


Figure 2.7: Linux bridge monitoring

Container Image

Like all the honeynet components, the monitoring component is a container, so it also needs its container image, which describes what the container should consist of

and what it should do when running. The container images docker file is specified in Listing 2.3.

Listing 2.3: Monitoring dockerfile

```
FROM ubuntu:latest
WORKDIR /code/
COPY . .
RUN apt-get update -y
RUN apt-get install tcpdump inotify-tools \
        util-linux curl libcap2-bin -y
RUN groupadd -r tcpdumpuser
RUN useradd -r -g tcpdumpuser tcpdumpuser
RUN groupadd pcap
RUN usermod -a -G pcap tcpdumpuser
RUN chgrp pcap /usr/bin/tcpdump
RUN chmod 750 /usr/bin/tcpdump
RUN setcap cap_net_raw,cap_net_admin=eip /usr/bin/tcpdump
USER tcpdumpuser
CMD ["/honeypot", "start"]
```

The docker file is made of multiple parts:

- The Docker file starts by using the latest version of the Ubuntu base image using the `FROM` instruction.
- The `WORKDIR` instruction sets the working directory to `/root/`.
- The `COPY` instruction is then used to copy the honeynet project directory (`.`) to the `WORKDIR`, which is the root directory in the container.
- The `RUN` instruction updates the package index and installs several packages including `tcpdump`, `inotify-tools`, `util-linux`, `curl` and `libcap2-bin`.
- The following `RUN` instructions create the `tcpdumpuser` user and `pcap` group which is added to the user and the group grants access to the `tcpdump` binary file, with these instructions, the `tcpdump` will not require the root privileges.
- The `USER` instruction specifies the user with which the container will be started, in this case, the created user `tcpdumpuser`.

Tcpdump

The `Tcpdump` is used to monitor the Linux bridge that is connected to the Honeynet network namespace.

The `tcpdump` is a powerful tool in the UNIX system. It allows capturing all incoming and outgoing communication packets [43]. This tool also allows filtering

communication by specified filters [43]. For example, `tcpdump -i eth0 tcp` for filtering all TCP packets on interface eth0.

The `tcpdump` program can be initiated with various arguments that impact its functionality. The honeypot `tcpdump` service utilises the following parameters.

- The option `-w` specifies the file path where the captured packets should be stored. The file is in the format `pcap`, which can be opened in data analytic software such as Wireshark [42, 43].
- The option `-W` specifies the number of files created per rotation. In the honeypot, the base is used for just one file. The `tcpdump` rotation represents a start of capturing packets and is ended by one of the triggers, either by the `-C`, which specifies the maximum size of the file or by the `-G`, which specifies the timeout. [43]
- The option `-C` specifies the maximum file size in Megabytes before finishing the capture, starting a rotation script specified by `-z`, and starting the new capture again. [43]
- The option `--packet-buffered` with option `-w` causes packets to be stored in the file instead of waiting for the buffer to be filled. This is needed in honeypots in case the honeypot stops or failed still has all captured packets stored in the file instead of volatile RAM. [43]
- The option `-G` specifies the timeout after which the `tcpdump` should finish and start, which clears the captured files from `-w` and start capturing packets again. This separates the captured communication into smaller parts. [43]
- The option `-z` specifies which script should be started after the rotation. For example, `-z gzip` would compress the captured data in the `gzip` format file. In the honeypot use case, it starts the script, which sends the stored `pcap` file to the web server [43].

Most options are used with the environment variables to make the `tcpdump` more configurable. The environment variables can be set during the container startup, which makes the container more modular since the image can be the same, and the only thing that changes are the variables. The final `tcpdump` service command with all the options used together is shown in Listing 2.4.

Listing 2.4: `Tcpdump` service code

```
tcpdump -W 1 -C $TCPDUMP_MAX_SIZE -G $TCPDUMP_TIMEOUT \  
  --packet-buffered -i any -w $TCPDUMP_FILE \  
  $TCPDUMP_FILTER -z $PWD/send_data.sh
```

Container Runtime

As with all other components, the monitoring is also made modular, fast and easy to install. The monitoring comprises one container image, which can be started with various environment variables. The monitoring container needs to capture all incoming and outgoing communication to the honeypot, and the attacker can not see the monitoring container. By default, all docker containers are started in the network namespace, so they can not interact with the host networking devices; this could cause a huge security risk to the host, but the monitoring containers need to see the host network to access the virtual ethernet interface connected to the honeynet system.

The monitoring container is started with the host network mode, which does not add the container to its separate network namespace but uses the host network resources. Due to this, the web server container can monitor the host network interfaces, such as the virtual ethernet interface connected to the network namespace from the honeynet. Because of this, the monitoring container can capture all incoming packets to the honeypot container and filter by the honeypot IP address.

The monitoring container is started with multiple environment variables. The following list describes the most important variables for the monitoring container.

- **SERVER** – The address to the webserver to which the captured pcap files should be sent.
- **TOKEN** – The token the monitoring container will use to authenticate and authorize itself to the web server.
- **ID** – The ID of the honeypot for which it is listening to the data. The ID is used to specify to which API endpoint the captured data should be sent.
- **TCPDUMP_EXTRAARGS** – The container starts with the tcpdump extra arguments to specify which virtual ethernet to start listing the communication.
- **TCPDUMP_FILTER** – The honeypot container is started before the mirroring container. This way, the monitoring container can get the assigned IP address of the container. This address filters incoming communication to the network namespace through the virtual ethernet connected to the Linux bridge.

2.2.3 Logging

Logging is an essential part of this bachelor's thesis, which designs the research honeypots which should gather as much data as possible. The design logs all incoming and outgoing communication and all logs from every container. For this, multiple technologies were used, such as tcpdump for monitoring the communication of the honeypot with the attacker or the syslog for monitoring the container log. All logs

are stored in the web server and later sent with the automatic export to the central logging server, which can hold data from multiple honeynet infrastructures.

Syslog

Syslog is a standard protocol to send system messages across a network. It allows applications and devices to send log messages to a central server for storage, analysis, and reporting. The Syslog protocol defines a standardised message format. [44]

The Syslog format is a standard logging format used for exchanging event messages. It consists of a header with fields like priority, timestamp, hostname, and application name, and optional fields like process ID and message ID. The message part contains the actual log content. Syslog messages are transmitted over UDP or TCP and can be stored locally or sent to a centralised server for analysis. [44]

Rsyslog

The Syslog is used for logging all container logs. Each container is configured to send all its logs to the rsyslog server. The rsyslog stores all syslogs to files and shares it with the web server, which later sends them to the centralised logging server.

Rsyslog is an open-source implementation of a syslog server that provides advanced features and capabilities for receiving, processing, and storing log messages. It is widely used in Linux and Unix systems as the default syslog daemon due to its flexibility and scalability. [45]

The Dockerfile shown in the Listing 2.5 uses an Ubuntu base image using the FROM instruction. The next two RUN instructions update the Ubuntu package index and install the `rsyslog` package.

Following this, the RUN instruction sets the configuration for `rsyslog` by appending a series of commands to `/etc/rsyslog.conf` file. The commands in the echo statement load the `imudp` and `imtcp` modules, configure `rsyslog` to listen for UDP and TCP traffic on port 514, define a custom logging template called `RemoteStore`, and specify that logs received from any source other than `localhost` should be stored in a file path under `/var/log/remote/` directory, named based on the `syslogtag`, and the current date.

Finally, the `ENTRYPOINT` instruction specifies that the container should run the `rsyslogd` daemon in the foreground with the `-n` option. This will allow the container to continuously process incoming logs and store them in the configured location.

Listing 2.5: Rsyslog container image

```
FROM ubuntu
RUN apt update && apt install rsyslog -y
RUN echo '$ModLoad imudp \n\
$UDPServerRun 514 \n\
$ModLoad imtcp \n\
$InputTCPServerRun 514 \n\
$template RemoteStore,
"/var/log/remote/%syslogtag%/_%$day%-%$month%-%$year%.log" \n\
:source, !isequal, "localhost" -?RemoteStore \n\
:source, isequal, "last" ~ ' > /etc/rsyslog.conf
ENTRYPOINT ["rsyslogd", "-n"]
```

Centralized Storage

The centralised storage system is designed to collect all the data gathered from multiple honeynet infrastructures, offering administrators a single point of reference for monitoring collected information. The web server automatically sends the data to the web server using a Linux cron, which executes the files depending on the schedule.

The centralised storage server in this setup operates an FTP server. The web server needs to have communication capabilities with the FTP. So the web server uses a script to manage the connection, log in, create the directory structure, and send the file to the FTP server.

While setting up the honeynet in the web servers UI, administrators can set the remote logging server and the corresponding path for data storage, expressed in the format ADDRESS:PATH. Additionally, the administrator must provide the username and password for the account intended to establish this connection. The last required parameter is the schedule for the data export, defined in the crontab file format, instructing the web server on the frequency of log transfers to the central logging server.

Upon entering all the requisite information and creating the honeynet system, the web server attempts to connect to the FTP server using the provided details. If the connection cannot be established, the web server presents the administrator with an error message, indicating an issue and specifying the nature of the problem. If the connection is successful, the web server generates the honeynet object and adds it to the database. As the administrator adds more honeypots, data collection begins on the web and the rsyslog servers.

The cron job, once initiated, triggers a script responsible for data transfer to the central logging server. This script initiates a connection to the FTP server, logs in using the credentials provided during honeynet setup, and changes the root directory to the path defined in `ADDRESS:PATH`. It then creates a directory structure in the format `honeynet/<HONEYNET_NAME>/<CONTAINER_NAME>`. After establishing the directory structure, it transmits the captured pcap and syslog files from rsyslog. Assuming all processes function correctly, the script will delete the data to free up space for future data capture.

2.2.4 Honeypot Creation Workflow

When the administrator creates a new honeynet, the web server creates a network namespace on the host using the docker command-line interface. After the honeynet is successfully created, the administrator can add new honeypots.

The honeypot creation comprises the honeypot container deployment and the honeypot monitoring container. For both of these deployments, docker-compose is used, which helps to start up multiple containers at once using the docker-compose files. The files specify what services should be started and their configurations. To make the configuration modular so the web server can pass some data to specify how the services should be started, use the Jinja template.

Jinja is a powerful and flexible open-source templating engine for Python, which allows variable substitution to insert values into text documents. These variables are enclosed in double curly braces `{ { }`. It also supports control structures like loops and conditionals. For example, `{% for item in items %}` to start a loop and `{% endfor %}` to end it and the `{% if condition %}` to start the block with the condition and `{% endif %}` to end it. [46]

For clarification between the already existing docker-compose.yml and jinja files in the project, the jinja files have added the .j2 suffix to the name, creating the filenames `honeypot.yml.j2` and `monitoring.yml.j2`. From these files, the docker-compose files will be generated and used to start the honeypot service.

The `honeypot.yml.j2` show in Listing 2.6 is a Jinja docker-compose template. The template creates a `honeypot.yml` for each honeypot, and using the docker-compose CLI starts it. The containers do not need to be started by the root. Any user with the required privileges can start the containers. The core elements of the Docker Compose file are:

- **Services:** This block defines the different services (containers) that comprise the application. In this case, the service is a honeypot, and its name and ID are dynamically filled using Jinja2 templating.

- **Image:** This field specifies the Docker image for the honeypots' container. The administrator provides the image address. It needs to be a URL to the registry image.
- **Ports:** If defined, these are the ports that the service will expose. These ports are specified dynamically through the Jinja2 template. The administrator specifies the ports in format `HOST:CONTAINER,HOST:CONTAINER...` where the `HOST` represents which port the honeypot should expose and the `CONTAINER` specifies the port on which the container is listening. This creates a tunnel to the host network so the attacker can communicate with the honeypot.
- **Logging:** If a syslog IP address is provided, the syslog driver is used for logging. The syslog address and tag are provided dynamically.
- **Networks:** This field specifies the network the service belongs to. In this case, it belongs to an externally defined network created before starting the container.
- **IPv4 address:** If an IP address is provided and the network does not need updating, this IP address is assigned to the service.
- **Capabilities:** The option indicates that all capabilities will be dropped for the container. This means that the container will run without any elevated privileges or permissions beyond the default capabilities provided to all processes.

Listing 2.6: honeypot.yml.j2

```

version: '3.4'
services:
  honeypot-{{ honeypot.name }}-{{ honeypot_id }}:
    image: {{ honeypot.image }}
    container_name: {{ honeypot.name }}
    restart: always
    cap_drop:
      - ALL
{% if honeypot_ports and honeypot_ports[0] %}
    ports:
{% for port in honeypot_ports %}
      - "{{ port }}"
{% endfor %}
{% endif %}
{% if syslog_ip %}
    logging:
      driver: syslog
      options:
        syslog-address: "tcp://{{ syslog_ip }}:514"
        tag: "{{ honeynet.name }}/{{ honeypot.name }}"
{% endif %}
    networks:
      honeynet_bridge:
{% if honeypot.ip_addr and not update %}
        ipv4_address: {{ honeypot.ip_addr }}
{% endif %}
networks:
  honeynet_bridge:
    external: true
    name: {{ honeynet.name }}

```

The `monitoring.yml.j2` is shown in Listing 2.7. This Docker Compose file describes a service for monitoring honeypot communication. The key components of the Docker Compose file are:

- **Services:** In this Docker Compose file, a single service named `honeypot-monitoring` is defined. The exact name is dynamically generated using Jinja2 templates.
- **Container Name:** The container's name, when it is launched, is also specified using Jinja2 templates.
- **Image:** The Docker image used for this service is `quay.io/mnecas0/honeypot-base:latest`. This image contains the monitoring script with the `tcpdump` dependencies.
- **Restart:** The 'always' restart policy means that Docker will restart the container every time it exits, regardless of the exit code.
- **Logging:** If a syslog IP address is provided, the syslog driver is used for logging with a dynamically specified syslog address and tag.
- **Environment:** This block defines environment variables for the container. These variables include debug mode, server address, identification token, honeypot ID, `tcpdump` filtering options, maximum size of `tcpdump`, `tcpdump` timeout, and additional `tcpdump` arguments. These arguments are passed to the `tcpdump` and monitoring script, which logs captured data to the web server.
- **Network Mode:** The network mode is set to 'host', which means the container will use the host's network stack. This is needed to monitor the Linux bridge, which is connected to the host and network namespaces through which the container communicates. Without this, the monitoring container could not see the Linux bridge.
- **Capabilities:** The option indicates that all capabilities will be dropped for the container except the `NET_RAW` and `NET_ADMIN` which are required for the `tcpdump` monitoring.

Listing 2.7: monitoring.yml.j2

```

version: '3.4'
services:
  honeypot-{{ honeypot.name }}-monitoring:
    container_name: {{ honeypot.name }}-monitoring
    image: quay.io/mnecas0/honeypot-base:latest
    restart: always
    cap_drop:
      - ALL
    cap_add:
      - NET_RAW
      - NET_ADMIN
{% if syslog_ip %}
    logging:
      driver: syslog
      options:
        syslog-address: "tcp://{{ syslog_ip }}:514"
        tag: "{{ honeynet.name }}/{{ honeypot.name }}-mirror"
{% endif %}
    environment:
      SERVER: "127.0.0.1:8000"
      TOKEN: "{{ honeypot_token }}"
      ID: "{{ honeypot_id }}"
      TCPDUMP_FILTER: "host $HONEYPOT_ADDR
        {% if honeypot.tcpdump_filter %}
          and {{ honeypot.tcpdump_filter }}
        {% endif %}"
      TCPDUMP_MAX_SIZE: "{{ honeypot.tcpdump_max_size }}"
      TCPDUMP_TIMEOUT: "{{ honeypot.tcpdump_timeout }}"
      TCPDUMP_EXTRAARGS: "$EXTRA_ARGS
        {{ honeypot.tcpdump_extra_args }}"
    network_mode: host

```

3 Results

This chapter presents the outcomes of the bachelor's thesis. The results consist of the final honeypot system deployed on the public network and the analysis of the captured communication.

3.1 Deployed Honeypots

The modular honeynet system designed and implemented in the previous chapter was installed on a web server shared on the public network. The system utilized two containers, one for the IT protocol and another for the OT protocols. The containers need to have a built container image which contains the honeypot project and be uploaded to the container registry from which it will be accessible to the honeynet system. These images will be started in the honeynet system with a specified port tunnel to the host network so the attackers can access the honeypot.

3.1.1 Deployment of IT Honeypot

For the IT honeypot, an existing open-source honeypot called "honeypots" from qeeqbox was chosen. The source code for this honeypot can be found on GitHub¹. Although the GitHub repository does not include the Dockerfile for the honeypot itself, it does provide the Dockerfile for the syslog server used for logging. However, this logging image was unnecessary for this deployment since the honeynet had its syslog server.

To build the honeypot container, a new Dockerfile was created. The created Dockerfile is shown in Listing 3.1. The image is built in the root directory of the qeeqbox/honeypots repository. It uses the base image ubuntu:22.04, onto which the project is copied and installed, along with all the required dependencies. Once the dependencies are installed, the qeeqbox/honeypots is installed in the container. The entry point is set to `/usr/local/bin/honeypots`, and the `CMD` specifies the parameters for starting the honeypot. In this case, it starts the HTTP honeypot on port 80, which will be exposed to the host network. The image is built using the docker CLI command `docker build`, and then it is pushed to the registry to make the built image accessible to the honeypot system.

¹<https://github.com/qeeqbox/honeypots>

Listing 3.1: Honeybots Dockerfile

```
FROM ubuntu:22.04
WORKDIR /root/
COPY . .
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update -y

# Install honeybots dependencies
RUN apt-get install postgresql libpq-dev python3-pip git -y
RUN pip3 install requests==2.20.1
RUN pip3 install .

# Start base honeybot as background process
ENTRYPOINT ["/usr/local/bin/honeybots"]
CMD ["--setup", "http:80", "--termination-strategy", "signal"]
```

3.1.2 Deployment of OT Honeybot

The OT honeybot selection was the open-source honeybot conpot, available on GitHub². Deploying the conpot was simpler than qeeqbox/honeybots as the conpot GitHub repository already includes the Dockerfile for building the honeybot container. To use conpot on the honeybot system, the image was built and pushed to a container registry from which it can be downloaded. The conpot can mimic large amounts of OT protocol. This bachelor's thesis focuses on the Modbus protocol, which runs on the 502 port, which will be exposed to the host network.

3.2 Captured Communication

The honeynet was deployed for two weeks on the public network through which the honeybots captured vast amounts of data. The HTTP honeybot captured around 275000 packets while the Modbus honeybot captured 1500. This comparison is shown in the Figure A.1. The honeybots do not affect this, but the HTTP servers are most likely more targeted than the Modbus servers, which are uncommon in IT networks.

²<https://github.com/mushorg/conpot>

3.2.1 HTTP Honeypot

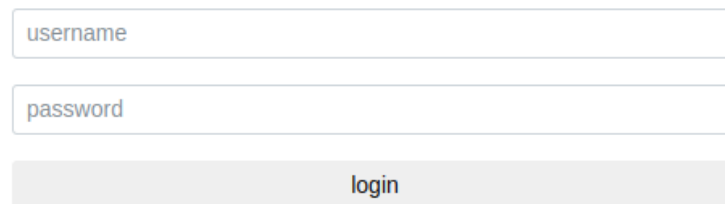
A total of 278 public IP addresses were connected to the HTTP honeypot. The address 100.100.17.23 was the most active, accounting for 96.5% of the recorded HTTP communication. The IP address 100.100.17.23 is reserved for carrier-grade NAT communication between service providers and subscribers³. Figure A.2 shows a graph comparing the number of requests per IP address connected to the honeypot.

The HTTP honeypot was made of a UI shown in Figure 3.1. The UI form comprises a username and password; these data are then sent to the honeypot server, shown in Figure 3.2.

Figure 3.2 shows two items, and these items are comprised of key and value pairs. These pairs are sent to the server by the attacker. The attacker can create HTTP packets with its HTML form, investigating how the server will reply. These keys and values were analysed and made to the graphs.

Most HTTP requests had no form data, but among those containing some, the most frequently used key was an unfinished PHP script that sent a base64-encoded text, specifically `GBNVT RCE Test`. Figure A.3 shows the most common data key in HTML form.

Figure A.4 shows the most common data value in HTML form. The most common value was the value: `)`) which is finishing the PHP script. The values also contained some SQL injections, an example of one of the injections: `select username from ap_users' or ' 1=1'-- ',x`.



The image shows a simple web form with two input fields. The first field is labeled 'username' and the second is labeled 'password'. Below these fields is a button labeled 'login'.

Figure 3.1: HTTP Honeypot UI

```
▼ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▼ Form item: "username" = "admin"
    Key: username
    Value: admin
  ▼ Form item: "psd" = "Feefifofum"
    Key: psd
    Value: Feefifofum
```

Figure 3.2: HTML Form of the captured data

³<https://www.speedguide.net/ip/100.100.17.23>

3.2.2 Modbus Honeypot

As for the Modbus honeypot communication, there was not as much communication as with the HTTP honeypot, but that does not mean it is not being targeted. The conpot honeypot detected multiple Modbus requests to the server with some data by which the attackers were testing the server's capabilities.

One of these captured request communication is shown in Listing 3.2. The server successfully received the byte sequence "133700000005002b0e0100" in the hexadecimal format. The same byte sequence was mentioned in article [47]. "The attacker read the Modbus variables and subsequently crafted some invalid input that is beyond a specified safe range. The attack appears to be an attempt to cause some sort of damage to the CPS equipment." [47] This incident demonstrates the actual utilisation of the honeypot in a live network environment and shows the value and necessity of honeypots.

Listing 3.2: Captured modbus communication

```
New modbus session from 198.235.24.211
New Modbus connection from 198.235.24.211:59196.
Modbus traffic from 198.235.24.211:
{
    'request': b'133700000005002b0e0100',
    'slave_id': 0,
    'function_code': None,
    'response': b''
}
Modbus connection terminated with client 198.235.24.211.
```

Conclusion

This thesis designed and implemented a modular and scalable honeypot system focused on IT and OT protocols. The initial theoretical part provided valuable information into the operation and necessity of honeypots, their categorisation, and the existing open-source projects in the field. Moreover, it detailed the use and benefits of containers, comparing them to virtual machines.

The project's design phase underwent multiple stages, resulting in a final design that utilises containers for faster system deployment and reduced storage requirements. This system includes a web server for administrator interaction, a monitoring container for tracking honeypot communication, and comprehensive logging capabilities.

Moreover, the system designed in this thesis emphasises the ease of adding new honeypots. Thanks to the modular structure and use of container technology, introducing a new honeypot is as simple as creating and configuring a new container image. This approach streamlines the process and makes the system highly adaptable to evolving cybersecurity threats. Thus, the implemented system provides an efficient and user-friendly platform for cybersecurity defence enhancement.

The honeypot system was deployed in a public network and successfully collected data for analysis, proving its potential for practical application. The deployed honeypots were conpot⁴ for Modbus protocol and honeypots⁵ for HTTP both open-source and available on GitHub. Overall, the thesis demonstrated the effective development and application of a honeypot system using container technology.

This bachelor's thesis was presented at the 2023 EEICT conference under the name "Modular Honeypot for IT and OT". There were few improvements since the writing of the article. The article had one base image for a honeypot with the monitoring tools and on top of this the honeypots would build their own images. If the attackers would get control over the honeypot they could access the monitoring scripts, letting the attackers know that they are being monitored. This bachelor's thesis improved on this by separating the honeypot and monitoring containers.

⁴<https://github.com/mushorg/conpot>

⁵<https://github.com/qeeqbox/honeypots>

Bibliography

- [1] MOKUBE, I.; ADAMS, M. *Honeypots: Concepts, Approaches, and Challenges* In Proceedings of the 45th Annual Southeast Regional Conference [online]. [cit. 2022-02-12]. 2007, pp. 321-326. ISBN 9781595936295. Available from: <https://doi.org/10.1145/1233341.1233399>
- [2] MUKHERJEE, B.; HEBERLEIN, L. T.; and LEVITT, K. N. *Network intrusion detection IEEE Network*, vol. 8, no. 3, pp. 26-41, 1994.
- [3] KRAWETZ, N. *Anti-honeypot technology IEEE Security and Privacy*, vol. 2, no. 1, pp. 76-79, 2004. Available from: <https://doi.org/10.1109/MSECP.2004.1264861>
- [4] TSIKERDEKIS, M.; ZEDADALLY, S.; SCHLESENER, A.; and SKLAVOS, N. *Approaches for Preventing Honeypot Detection and Compromise* In *2018 Global Information Infrastructure and Networking Symposium (GIIS)*, 2018, pp. 1-6.
- [5] ISO OSI Basic reference model. *networkhope* [online]. [cit. 2022-02-12]. Available from: <https://networkhope.in/iso-osi-basic-reference-model/>
- [6] Operational technology (OT) – definitions and differences with IT. *Operational technology* [online]. [cit. 2022-02-12]. Available from: <https://www.i-scoop.eu/industry-4-0/operational-technology-ot/>
- [7] OT, ICS, SCADA – What’s the difference? *kuppingercole*, WILLIAMSON, G. [online]. [cit. 2022-02-12]. Available from: <https://www.kuppingercole.com/blog/williamson/ot-ics-scada-whats-the-difference>
- [8] A Beginner’s PLC Overview, Part 3 of 4: PLC Inputs and Outputs (I/O) *Automation*, GATES, S. [online]. [cit. 2022-02-12]. Available at: <https://www.automation.com/en-us/articles/2018/a-beginners-plc-overview-part-3-of-4-plc-inputs-an>
- [9] What is Communication Protocol? *dipslab*, CHAUDHARI, D. [online]. [cit. 2022-02-12]. Available from: <https://dipslab.com/plc-communication-protocols-used-industry/>
- [10] What is Modbus? *paessler* [online]. [cit. 2022-02-12]. Available from: <https://www.paessler.com/it-explained/modbus>
- [11] How to ensure security in Industrial Protocols. *enigmaedia* [online]. [cit. 2022-02-12]. Available from: <https://enigmaedia.es/2021/09/19/security-industrial-protocols-ot-networks/>

- [12] Communication Protocols for the Industry 4.0 *EEWeb*, GESUALDO, D.D. [online]. [cit. 2022-02-12]. Available from: <https://www.eeweb.com/communication-protocols-for-the-industry-4-0>
- [13] Use containers to Build, Share and Run your applications. *Docker* [online]. [cit. 2022-02-12]. Available from: <https://www.docker.com/resources/what-container/>
- [14] POTDAR, A.M.; NARAYAN D.G.; KENGOND, S., MULLA, M.M. *Performance Evaluation of Docker Container and Virtual Machine* In Procedia Computer Science [online]. [cit. 2022-02-12]. 2020, vol. 171, pp. 1419-1428. ISSN 1877-0509. Available from: <https://doi.org/10.1016/j.procs.2020.04.152>
- [15] namespaces(7) — Linux manual page. *man7* [online]. [cit. 2022-05-10]. Available from: <https://man7.org/linux/man-pages/man7/namespaces.7.html>
- [16] Network namespaces(7) - Linux manual page. *man7* [online]. [cit. 2023-05-10]. Available from: https://man7.org/linux/man-pages/man7/network_namespaces.7.html
- [17] An introduction to Linux bridging commands and features. *Red Hat Developers*, LIU, H. [online]. [cit. 2023-05-11]. Available from: <https://developers.redhat.com/articles/2022/04/06/introduction-linux-bridging-commands-and-features>
- [18] veth(4) — Linux manual page. *man7* [online]. [cit. 2022-05-11]. Available from: <https://man7.org/linux/man-pages/man4/veth.4.html>
- [19] Open vSwitch FAQ - Configuration. *Open vSwitch* [online]. [cit. 2023-05-10]. Available at: <https://docs.openvswitch.org/en/latest/faq/configuration/>
- [20] Docker Documentation: Use bridge networks. *Docker* [online]. [cit. 2023-05-10]. Available at: <https://docs.docker.com/network/bridge/>
- [21] Docker Documentation: Use host networking. *Docker* [online]. [cit. 2023-05-10]. Available at: <https://docs.docker.com/network/host/>
- [22] Docker Documentation: Docker security. *Docker* [online]. [cit. 2023-05-13]. Available at: <https://docs.docker.com/engine/security/>
- [23] Docker Documentation: Dockerfile reference. *Docker* [online]. [cit. 2023-05-10]. Available at: <https://docs.docker.com/engine/reference/builder/>

- [24] Docker Documentation: Docker CLI reference — Docker build. *Docker* [online]. [cit. 2023-05-10]. Available at: <https://docs.docker.com/engine/reference/commandline/build/>
- [25] Docker Documentation: Docker CLI reference — Docker push. *Docker* [online]. [cit. 2023-05-10]. Available at: <https://docs.docker.com/engine/reference/commandline/push/>
- [26] Docker Documentation: Deploying a Registry. *Docker* [online]. [cit. 2023-05-10]. Available at: <https://docs.docker.com/registry/deploying/>
- [27] Docker Documentation: Docker CLI reference — Docker pull. *Docker* [online]. [cit. 2023-05-10]. Available at: <https://docs.docker.com/engine/reference/commandline/pull/>
- [28] Docker Documentation: Docker Compose Features and Uses. *Docker* [online]. [cit. 2023-05-11]. Available at: <https://docs.docker.com/compose/features-uses/>
- [29] Native vSwitch: What is It and How Does It Work? *VMware Network Virtualization Blog*, FORTIER, R. [online]. [cit. 2023-05-11]. Available at: <https://blogs.vmware.com/networkvirtualization/2017/03/native-vswitch.html/>
- [30] Glossary. *Ansible* [online]. [cit. 2023-05-11]. Available at: https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html
- [31] Ansible Documentation: VMware Guide. *Ansible* [online]. [cit. 2023-05-11]. Available at: https://docs.ansible.com/ansible/2.9/scenario_guides/vmware_scenarios/vmware_intro.html
- [32] Monitoring network traffic from within a virtual machine on a VMware vSphere ESX/ESXi server. *VMware* [online]. [cit. 2023-04-06]. Available at: <https://kb.vmware.com/s/article/1038847>
- [33] VMware vSphere Networking Guide. *VMware* [online]. [cit. 2023-05-11]. Available at: <https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.networking.doc/GUID-CFFD9157-FC17-440D-BDB4-E16FD447A1BA.html>
- [34] Port Mirroring. *oVirt*, ASAYAG, M.; OURFALI, O.; HAVIVI, S. [online]. [cit. 2023-05-11]. Available at: <https://www.ovirt.org/develop/release-management/features/network/portmirroring.html>

- [35] KUWATLY, I.; SRAJ, M.; AL MASRI, Z.; and ARTAIL, H. *A dynamic honeypot design for intrusion detection* In *The IEEE/ACS International Conference on Pervasive Services, 2004. ICPS 2004. Proceedings.*, 2004, pp. 95-104.
- [36] Meet Django. *Django* [online]. [cit. 2023-05-13]. Available at: <https://www.djangoproject.com/>
- [37] Django Tutorial Part 2: Creating the Polls App. *Django* [online]. [cit. 2023-05-13]. Available at: <https://docs.djangoproject.com/en/4.2/intro/tutorial02/>
- [38] What is PostgreSQL? *PostgreSQL* [online]. [cit. 2023-05-12], Available at: <https://www.postgresql.org/docs/15/intro-what-is.html>
- [39] cron(8) — Linux manual page. *man7* [online]. [cit. 2023-05-13]. Available from: <https://man7.org/linux/man-pages/man8/cron.8.html>
- [40] crontab(5) — Linux manual page. *man7* [online]. [cit. 2023-05-13]. Available from: <https://man7.org/linux/man-pages/man5/crontab.5.html>
- [41] Django REST framework. *Django REST framework* [online]. [cit. 2023-05-13]. Available at: <https://www.django-rest-framework.org/>
- [42] PCAP(1) MAN PAGE. *tcpdump* [online]. [cit. 2022-02-12]. Available from: <https://www.tcpdump.org/manpages/pcap.3pcap.html>
- [43] TCPDUMP(1) MAN PAGE. *tcpdump* [online]. [cit. 2022-02-12]. Available from: <https://www.tcpdump.org/manpages/tcpdump.1-4.9.2.html>
- [44] The Syslog Protocol. *datatracker.ietf.org* [online]. [cit. 2023-05-14]. Available from: <https://tools.ietf.org/html/rfc5424>
- [45] The Rocket-fast SYStem for LOG processing. *Rsyslog* [online]. [cit. 2023-05-14]. Available at: <https://www.rsyslog.com/>
- [46] Template Designer Documentation. *Jinja* [online]. [cit. 2023-04-06]. Available at: <https://jinja.palletsprojects.com/en/3.1.x/templates/>
- [47] ELIAS, B.; WALTER L.; NICOLA F.; SEAN W., NASIR G.; and BRUNO S., *Cyber Meets Control: A Novel Federated Approach for Resilient CPS Leveraging Real Cyber Threat Intelligence*, *IEEE Communications Magazine*, vol. 55, no. 5, pp. 198-204, 2017. Available from: <https://doi.org/10.1109/MCOM.2017.1600292CM>

Symbols and abbreviations

ADU Application Unit
AGPL Affero General Public License
API Application Programming Interface
ARP Address Resolution Protocol
ASCII American Standard Code for Information Interchange
CLI Command-Line Interface
CPU Central Processing Unit
CRON Command Run On
DPC Discrete Process Control
DNS Domain Name System
DMZ DeMilitarized Zone
DRF Django Rest Framework
EIA Environmental Impact Assessment
ESX Elastic Sky X
FTP File Transfer Protocol
GIF Graphics Interchange Format
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
ICMP Internet Control Message Protocol
ICS Industrial Control Systems
IDS Intrusion Detection System
IP Internet Protocol
IT Information Technology
JPEG Joint Photographic Experts Group
L2TP Layer 2 Tunneling Protocol
MAC Media Access Control address
MBAP Modbus Application Protocol
MIDI Musical Instrument Digital Interface
NetBEUI NetBIOS Extended User Interface
OT Operation Technology
OSPF Open Shortest Path First
OVS Open vSwitch
PCI Payment Card Industry
PLC Programmable Logic Controllers
POSIX Portable Operating System Interface
RAM Random Access Memory
REST REpresentational State Transfer

RIP Routing Information Protocol
RPC Remote Procedure Call
RS Recommended Standard
RSYSLOG Rocket-fast SYstem for LOG
SFTP Secure File Transfer Protocol
SMTP Simple Mail Transfer Protocol
SNMP Simple Network Management Protocol
SSH Secure Shell
SPAN Switched Port Analyzer
SCADA Supervisory Control And Data Acquisition
SQL Structured Query Language
SYSLOG System Logging Protocol
TCP Transmission Control Protocol
TLS Transport Layer Security
UDP User Datagram Protocol
UI User Interface
UUID Universally Unique Identifier
VM Virtual Machine

List of appendices

A	Captured Data Graphs	62
B	User Manual Installation	64
C	Contents of the Electronic Attachment	67

A Captured Data Graphs

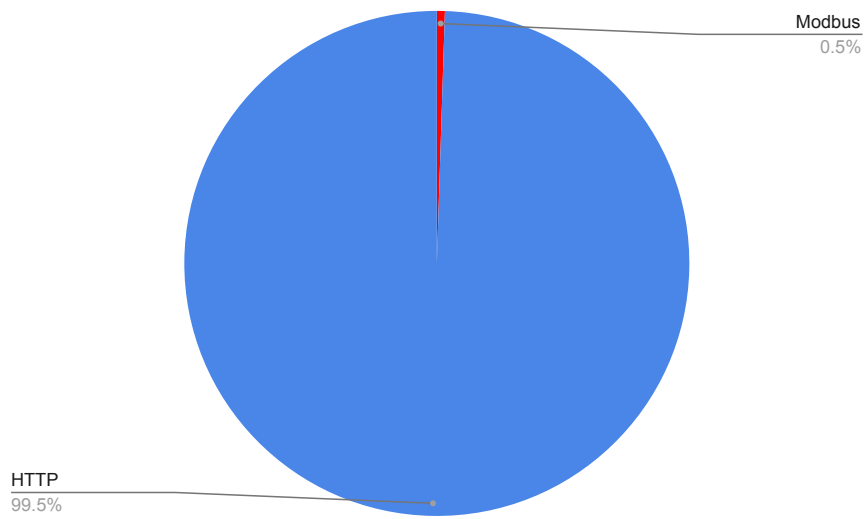


Figure A.1: Comparing HTTP and Modbus number of captured packets

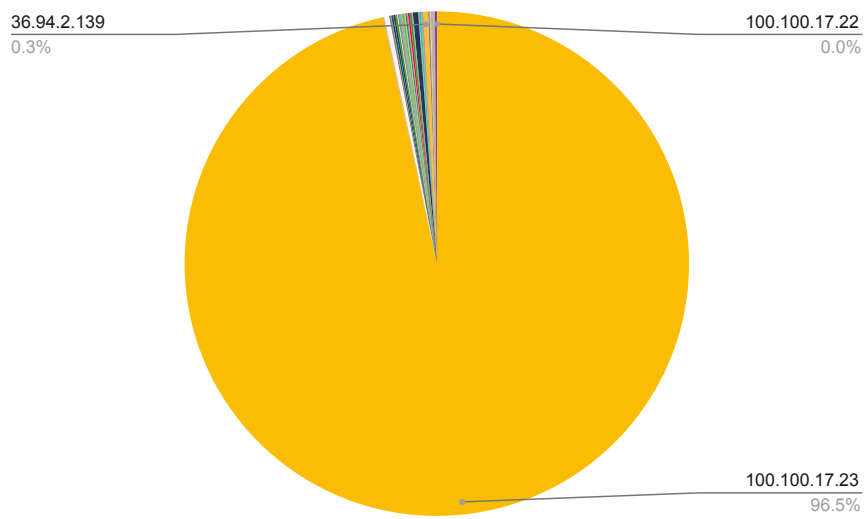


Figure A.2: The communication grouped by source IP

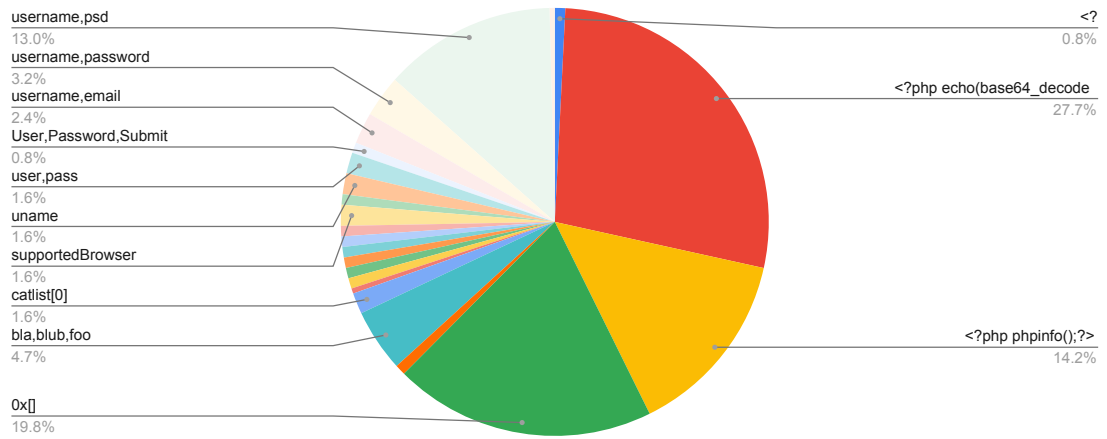


Figure A.3: The most common data key in HTTP request

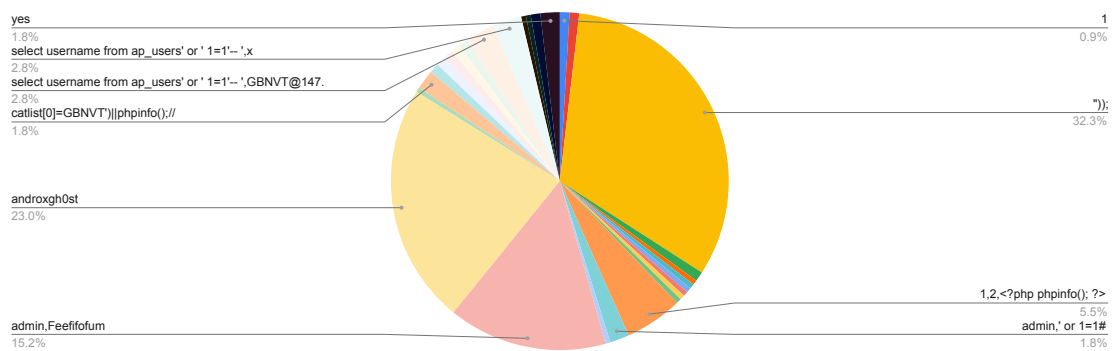


Figure A.4: The most common data value in HTTP request

B User Manual Installation

The web server has two operational modes: production and development.

The default mode is development, the steps to install it are show in Listing B.1. This mode initiates a basic web server integrated with an SQLite database. It does not have any extra components such as a syslog server. These additional components need to be manually installed. The syslog configuration is show in the production deployment Listing B.2 and the administrator needs to use just the rsyslog config and run the `docker-compose up -d`. The `-d` tells to docker that the containers should start as a background process. Despite the absence of a syslog server, the honeynet will remain functional, but it will not keep a record of container logs.

Listing B.1: Initiating Development Mode

```
$ pip3 install -r ./requirements.txt
$ python3 manage.py makemigrations
$ python3 manage.py migrate
$ python3 manage.py runserver
```

For production deployment of the honeynet is used a predefined docker-compose file. This file is shown in Listing B.2. To install the honeynet on a clean system, administrators must install docker and docker-compose. No additional installations are necessary; the docker-compose takes care of the rest.

The production docker-compose file comprises three containers: the web server, a Postgres database, and a syslog server. All of these containers' images can be obtained from the quay registry. Docker will fetch these images and start them. The containers are designed to restart automatically in case of an error. Administrators must execute the `docker-compose up -d` command in the directory with the specified docker-compose file to start the deployment. If the administrator wants to make any changes and deploy the project from the source code, they must add the build parameter to the docker-compose file and run `docker compose build`.

The web server is initiated with a default administrator user called `mnecas` and password `mnecas`. This user can be modified on the Django admin page, to which this user has access.

To initiate a new honeypot, the administrator must create a honeynet network by specifying its name and subnet address. After creating the honeynet, the administrator can continue to create the honeypot, which requires a container image from the registry. The image example¹ is made from qeeqbox/honeypots. To enable access to the honeypot from the host network, the administrator needs to specify the `Expose ports`. In this case, it should be `80:80`.

¹quay.io/mnecas0/honeypots:latest

Listing B.2: Production docker-compose

```

version: '3'
services:
  web:
    container_name: honeynet_web
    image: quay.io/mnecas0/honeynet:latest
    # build: .
    command:
      - /bin/bash
      - -c
      - |
        python3 manage.py collectstatic --noinput
        python3 manage.py migrate \
          --settings web_server.settings.production \
          --noinput
        python3 manage.py create_admin \
          --settings web_server.settings.production
        python3 manage.py runserver \
          --settings web_server.settings.production \
          127.0.0.1:8000
    network_mode: host
    restart: always
    volumes:
      - /var/log:/var/log
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    depends_on:
      db:
        condition: service_healthy
  db:
    image: postgres:latest
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s

```

```
    retries: 5
network_mode: host
volumes:
  - ./10-init-db.sql:/docker-entrypoint-initdb.d/
    10-init-db.sql
  - db:/var/lib/postgresql/data
syslogserver:
  image: quay.io/mnecas0/honeypot-rsyslog:latest
  # build: deployment/logging/
  container_name: syslog
  restart: always
  volumes:
    - /var/log:/var/log
  cap_add:
    - SYSLOG
volumes:
  db:
    driver: local
```

C Contents of the Electronic Attachment

honeynet	Root dir
├── api	API directory
│ ├── apps.py	
│ ├── authentication.py	The authentication file
│ ├── examples.py	
│ ├── serializers.py	The serialisation of the Django Database Objects
│ ├── tests.py	The tests of the API
│ ├── urls.py	The list of API endpoints
│ └── views.py	The logic of the API
├── deployment	Directory for honeynet management scripts
│ ├── delete-honeypot.sh	Removing honeypot with monitoring container
│ ├── deploy-honeypot.sh	Creating honeypot with monitoring container
│ ├── export-data.sh	Export script started by the corn job
│ ├── honeypot-examples	The honeypot example directory, testing/example usage
│ │ ├── Dockerfile	The ssh honeypot example, testing purpose
│ │ ├── http.yml	Example of docker-compose of HTTP honeypot (honeypots)
│ │ └── modbus.yml	Example of docker-compose of Modbus honeypot (conpot)
│ ├── logging	
│ │ └── Dockerfile	The rsyslog server dockerfile
│ └── templates	
│ ├── honeypot.yml.j2	The honeypot docker-compose jinja template file
│ └── monitoring.yml.j2	The monitoring docker-compose jinja template file
├── docker-compose.yml	The docker-compose which starts the whole project
├── Dockerfile	The Dockerfile to build the web server container image
├── 10-init-db.sql	The init script of the Postgres database
├── LICENSE	The Apache License
├── main	The main web server directory
│ ├── admin.py	The configuration of the Django admin page
│ ├── apps.py	
│ ├── forms.py	The Django forms used for user input validation
│ ├── management	The directory of custom Django management commands
│ │ ├── commands	
│ │ │ ├── config.py	
│ │ │ ├── create_admin.py	Command to create new specified administrator
│ │ │ ├── remove_logs.py	Command to remove logs of the honeynet
│ │ │ └── send_logs.py	Command to send the honeynet logs
│ ├── migrations	The Django database migration
│ ├── models.py	Database description
│ └── static	Static files used in for the UI
│ ├── anychart	Anychart library, used for honeynet visualisation
│ ├── bootstrap	Bootstrap library, used for UI
│ ├── dashboard.css	
│ ├── dashboard.js	
│ └── favicon	

```

├── jquery
├── main.css
├── switch.png
├── templates. .... Directory of the HTML templates
│   ├── base.html
│   ├── honeynet.html
│   ├── honeypot_form.html
│   ├── honeypot.html
│   ├── honeypots.html
│   ├── index.html
│   ├── logging_card.html
│   ├── login.html
│   └── sidebar.html
├── templatetags
│   └── get_item.py
├── tools
│   └── deployment.py
├── urls.py
├── views.py. .... The backed logic of the Django main web server
├── manage.py
├── monitoring. .... The scripts for monitoring honeypot continaer
│   ├── docker-compose.example.yml
│   ├── Dockerfile. .... The monitoring Dockerfile used
│   ├── honeypot. .... Main monitoring script
│   ├── Makefile
│   ├── README.md
│   ├── send_data.sh. .... Script executed by tcpdump to send data to API
│   └── tools.sh. .... Common tools used in both scripts
├── README.md
├── requirements.txt. .... All python requirements for web server
├── .dockerignore
├── .editorconfig
├── .gitignore
├── .flake8
├── .pre-commit-config.yaml
├── web_server
│   ├── asgi.py
│   ├── settings
│   │   ├── base.py
│   │   ├── production.py. .... The setting used for production environment
│   │   └── development.py. .... The setting used for development environment
│   ├── urls.py
│   └── wsgi.py

```