Modular Honeypot for IT and OT

Martin Nečas
Department of Telecommunications
The Faculty of Electrical Engineering and Communication
Brno University of Technology
Brno, Czech Republic
xnecas29@vutbr.cz

***Abstract*—This paper presents a design and development of a modular honeypot for information and operation technology (OT) that can be easily expandable and scalable. The honeypots send captured data to a web server. The honeypots can capture and archive all communication or specific data. The measured data are processed and shown to the user via a user-friendly website.**

***Index Terms*—containers, honeypots, IT, network monitoring, OT**

I. Introduction

This article focuses on designing modular honeypots for Information Technology (IT) and Operational Technology (OT) environments, which can enhance cybersecurity in critical infrastructure.

Several OT protocols lack the security to encrypt or authenticate communications. Consequently, they are exposed to various cyberattacks, such as virus infections and denial of service attempts [1]. In addition, many OT systems use outdated hardware that is challenging to secure and does not have the newest security features. Critical infrastructure systems which are susceptible to cyberattacks could lead to hinder or even endanger public safety. Some OT protocols started slowly implementing Transport Layer Security (TLS) for security propose. It is worth noting that the implementations of Modbus TCP with TLS for security were suggested in 2018 [2].

Honeypots can be used to detect new and emerging attack techniques that may not be detectable by traditional security measures. They can also provide insights into attacker behaviour and help organizations improve incident response processes [3]. Furthermore, honeypots can divert attacks from critical systems, providing additional protection. Using honeypots, organisations can identify vulnerabilities in their OT systems and take proactive measures to prevent cyber attacks that could significantly impact public safety and national security.

The honeypots are designed to simulate legitimate systems and services, comprising modules of network services such as web applications, Industrial Control Systems (ICS), and Supervisory Control And Data Acquisition (SCADA) systems. Creating an effective modular honeypot requires a comprehensive understanding of the simulated environment and regular monitoring of the honeypot's activity. Furthermore, honeypots should be isolated from production environments to prevent attackers from accessing real systems.

II. State of the art

In the article [4], the authors presented a modular honeypot system that uses containers to simulate protocols as high-interaction modules, enabling attackers to interact with the system. The system features a flexible software architecture that integrates new protocol modules. It comprises a core module for configuration and management, a packet capture and noise filter module, low-interaction modules, high-interaction modules, and an attack database.

Likewise, the article [5] outlines a proposed honeynet server architecture that utilises virtual machines and load balancing to monitor and intercept attacks on honeypots. The data is stored on a shared storage device. The honeynet uses software agents to manage and monitor traffic on the RTC monitoring cluster server and then forwards the communication to honeynet servers.

The [6] article describes a system architecture that uses containers to deploy a honeynet in a monitored network and collect data from it. The system comprises three modules: Honeynet Deployment, Intrusion Detection, and System Management. The Honeynet Deployment module uses Conpot to configure several honeypots to form a honeynet, and the Intrusion Detection module includes data collection. The System Management module enables users to manage honeypot information and intrusion detection parameters through a graphical user interface.

This article draws inspiration from previous designs by using containers for easy modularity, a load balancer for distributing the workload across servers, and shared storage for data storage. The primary focus of this article is to capture communication in the absence of preexisting honeypot services on physical or virtual networks.

III. Design

Numerous factors must be considered to ensure a honeypot system's efficacy. For the honeypots to work at their best, they must first be easy to install on any system and use few resources. Virtual Machines and Linux containers are only a few possibilities available to do this. But, Linux containers were chosen due to their simple installation on any system, scalability, and low resource needs.

Unlike Virtual Machines, Linux containers use fewer resources since they don't need a whole operating system. They are more effective and less resource-intensive since they share

a kernel with the host operating system. Linux containers are the best choice for creating a modular honeypot system that is simple to build and resource-effective because of this capability [7].

The accessibility of honeypot services, which would mimic some services, is another crucial factor that needs to be considered when creating a honeypot system. The honeypot system may not operate well for some network services if no honeypots can mimic their communication. In these situations, physical or virtual services might be used as honeypots to lure attackers.

A. Honeypot system with preexisting honeypot service

The design is made of multiple parts, as shown in Figure 1. The main honeypot application comprises the Container image and the Data server. The Container image contains the Honeypot service to mimic a networking service that will communicate with the attacker.

The Base container image utilises the tcpdump to monitor all incoming or outgoing communication to the container and stores it in a pcap file. When the pcap file reaches the maximum size or the specified timeout expires, the captured communication is sent to the data server. After sending the data, the honeypot removes it and resumes listening for new communication. These parameters are intended to prevent long-term storage and ensure prompt transmission to the data server.

The vast amount of stored packets can overwhelm the security analyst. To address this, the Honeypot service can be configured to send specific data to the data server, such as attempted login usernames and passwords in SSH honeypots. Communication tools in the Base container image guarantee secure and encrypted data transmission to the data server.

The Data server is also scalable and made up of several components. The User Interface (UI) allows analysts to download and analyse captured pcap files in analyse tools, e.g. Wireshark, or obtain specific data from the honeypot service in JSON (JavaScript Object Notation) format. Additionally, the Data server includes the Application Programming Interface (API) with two endpoints for specific data and pcap files in binary format. All data is stored in a container and linked to the database or saved directly in the database, which can be located on a remote server. Lastly, the back-end component handles server logic and connects all the other parts.

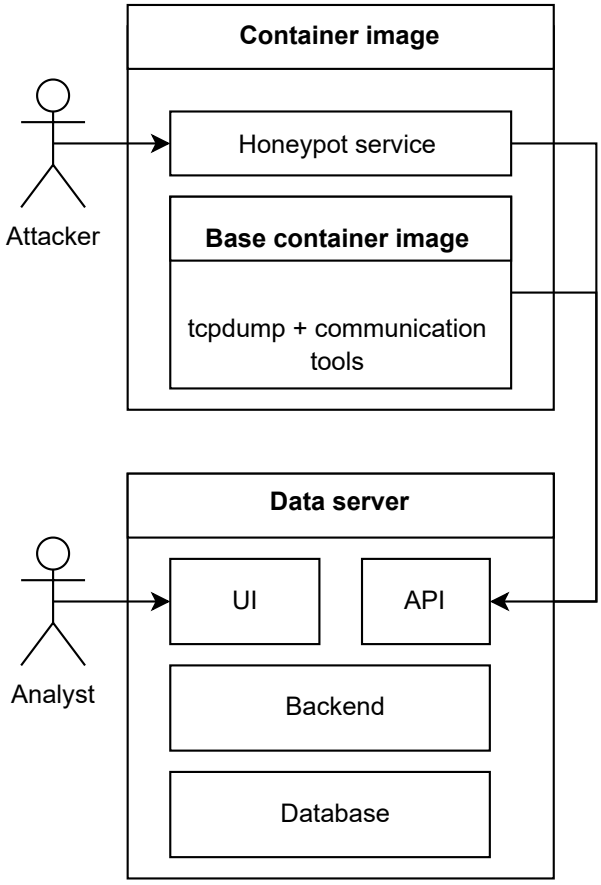

Fig. 1. Honeypot system with preexisting honeypot service design

B. Honeypot system without preexisting honeypot service

This section will discuss the application structure, which includes a container image and a data server, similar to the previous section. However, a container image is different because it does not contain a honeypot service. Instead, external physical or virtual devices communicate with attackers.

This situation is common with the OT protocols because they do not have many existing honeypots to mimic the communication compared to the Information Technology protocols. For this reason, the honeypot needs to monitor the whole communication to gather all the necessary data from the attack.

To achieve this, a network tap or switch with a Switched Port ANalyzer (SPAN) port is placed between the service and the attacker. The sole purpose of this tap or switch SPAN port is to duplicate the communication between the service and the attacker onto another port, as shown in Figure 2. By adding this interface to the container, the container can monitor and record all communication between the attacker and the service and send it to the data server.

In virtual networks, the network tap or switch with a SPAN port can also be added by creating a virtual switch, monitoring specific ports, and forwarding the communication to another port that can be attached to the virtual machine.

This approach allows security analysts to capture and analyze attackers' interactions with the service. Unlike honeypot service, this approach uses real services while capturing attacker behaviour.

It is important to note that the data server is the same as in the previous section. It has a user interface for analyzing the collected data and an API with endpoints for accessing specific data and pcap files. Data can be stored either in containers, linked to the database, or in the database itself, with the backend connecting all parts of the data server and

managing the server logic.

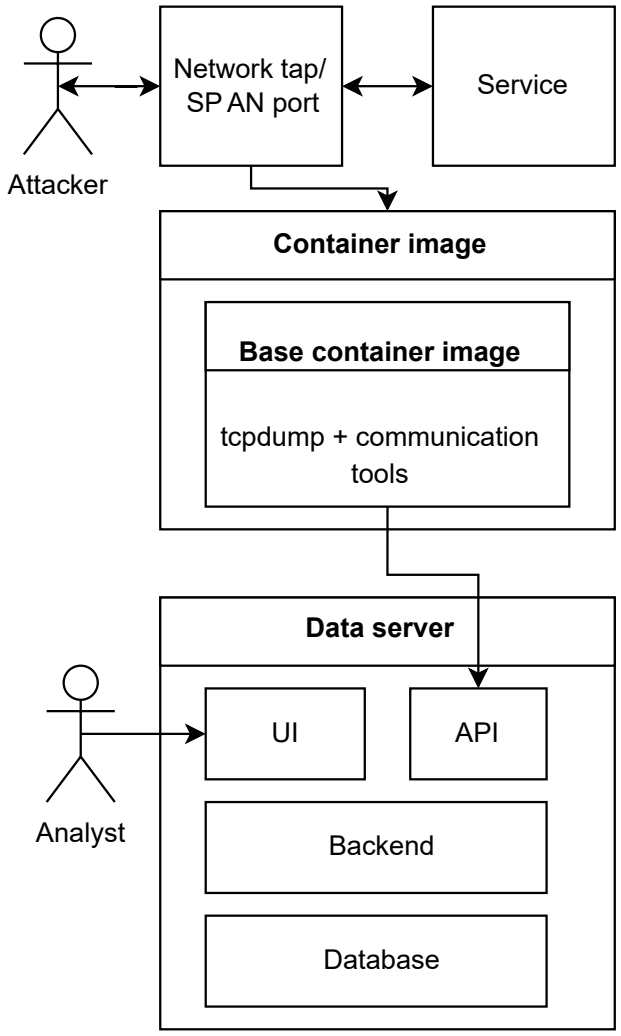

Fig. 2. Honeypot system without preexisting honeypot service design

C. Scalability

Containers are one of the most popular solutions for making applications easily installable and scalable [8]. A common practice to scale containerised applications is to have a load balancer that disperses the load between different workloads and scale-out the system by deploying more containers.

Load balancing ensures incoming requests are split evenly among several servers to maximise system performance. Diverting traffic to healthy servers in the event of a failure also helps prevent server overload and guarantees high availability.

A remote database can be added to store captured communication and data to increase the system's scalability further. As a result, each container is no longer required to maintain its database, simplifying system scaling. All data servers may direct data to the precise location with a single database, simplifying management and maintenance.

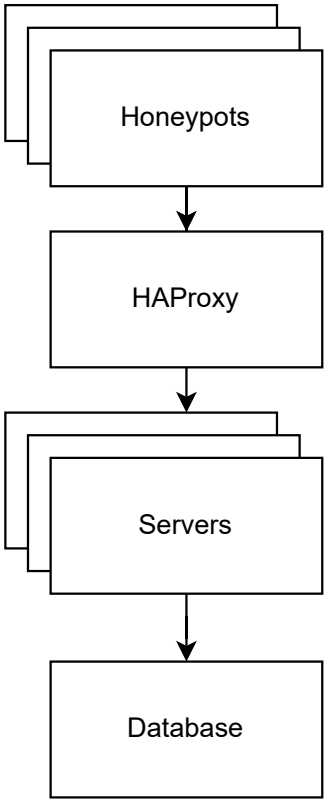

Fig. 3. Scalable design of honeypot system

IV. Implementation

Two container images were created to implement the proposed model: one for the honeypot and the other for data collection. The honeypot container was designed to simulate vulnerable systems and attract potential attackers. Meanwhile, the data collection container is responsible for collecting data on the attacks detected by the honeypot. This information is then sent to the data server container for storage and analysis.

A. Data server

The data server container was implemented using the Django Framework, well-known in the open-source community for its reliability and ease of use. One of the key advantages of using Django is that it comes with an easy-to-use model database running by default with SQLite. However, with small configuration changes, the database can be changed to any remote database running on a remote server to support the scalable design described in the previous section. The API for accessing the data collected by the data server is made using the Django Rest Framework, which simplifies the creation of RESTful APIs and provides a range of tools and features to enhance the functionality and security of the API. The Bootstrap Framework was used to create the UI for the data server to ensure that the user interface is responsive and user-friendly.

B. Container image

The container image used for the honeypot container is based on a base container image, which contains all the necessary tools for communication with the data server and monitoring the communication on the container image. The base container tools are written in bash to be as modular as possible and require only tcpdump and curl binaries. This design decision ensured the base container image could run on the most common Linux distributions, such as Fedora, Ubuntu, Debian, etc.

C. Security

The Data server creates a unique token for each honeypot to guarantee the security and integrity of the data collected from the honeypots. This token is used to authenticate and authorize communication between the data server and the honeypot. Incoming requests without a valid token will be denied by the data server, providing a message with the error. The honeypots can only add data to data servers. Even if an attacker succeeds in obtaining the token, they are not permitted to change or modify any already-existing data, but they are permitted to add new information. Although an attacker could still flood the data server with false information, the server will still hold information about how the attacker got on the honeypot.

HAProxy was added to control traffic flow between the honeypots and the data server to enhance the system's security. HAProxy allows for setting anti-DDoS attack parameters, such as limiting the number of requests per second or closing connections for attacks like slowloris [9]. HAProxy restricts the honeypots from communicating with any API endpoints other than those specified, adding a layer of security to the system.

HAProxy offers load-balancing and security features, enabling multiple servers to connect to the HAProxy. By doing this, the load is distributed evenly among the servers and the system is guaranteed to handle an increase in traffic volume [10]. The use of HAProxy enhances the system's scalability and reliability while also adding more security measures. The proposed model can offer a secure and reliable platform for gathering and analyzing attack data by considering these security considerations.

D. Virtual switch with SPAN port

To replicate the communication between an attacker and a virtual machine, it is necessary to have a virtual switch that can control the communication.

The Open Virtual Switch (OVS) was selected because of its compatibility with the most widely used enterprise virtualization platforms, including VMware vSphere and RHV. The OVS can create a virtual switch that can create numerous virtual interfaces, which can be added to Virtual Machines. These virtual interfaces can be mirrored to another virtual interface that can be added to another Virtual Machine or to a container networking namespace for monitoring the communication [11].

V. Conclusion

This article presented a design of a modular honeypot system. The system can simulate various networking protocols and includes monitoring capabilities for virtual and physical communication between services. The systems containers implementation makes it easy to use, modular and scalable design.

The future step of this design and implementation is to deploy the system onto a server to capture and analyse communication for potential security attacks. Using this honeypot system, organizations can understand the attacks against their networks and take proactive actions to prevent future breaches.

Acknowledgment

I would like to thank Ing. Petr Blažek for his expert guidance and helpful thoughts during the writing of this paper.

References

[1] Thomas Morris, Rayford Vaughn, and Yoginder Dandass. A retrofit network intrusion detection system for modbus rtu and ascii industrial control systems. In *2012 45th Hawaii International Conference on System Sciences*, pages 2338–2345, 2012.

[2] Modbus.org, "modbus/tcp security,". 2018.

[3] Iyatiti Mokube and Michele Adams. Honeypots: Concepts, approaches, and challenges. In *Proceedings of the 45th Annual Southeast Regional Conference*, ACM-SE 45, page 321–326, New York, NY, USA, 2007. Association for Computing Machinery.

[4] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. Interaction matters: A comprehensive analysis and a dataset of hybrid iot/ot honeypots. In *Proceedings of the 38th Annual Computer Security Applications Conference*, ACSAC '22, page 742–755, New York, NY, USA, 2022. Association for Computing Machinery.

[5] Ahmed Noaman, Ayman Abdel-Hamid, and Khalid Eskaf. A novel honeynet architecture using software agents. In *2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 1–6, 2019.

[6] Zhiqiang Wang, Gefei Li, Yaping Chi, Jianyi Zhang, Qixu Liu, Tao Yang, and Wei Zhou. Honeynet construction based on intrusion detection. In *Proceedings of the 3rd International Conference on Computer Science and Application Engineering*, CSAE '19, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Amit M Potdar, Narayan D G, Shivaraj Kengond, and Mohammed Moin Mulla. Performance evaluation of docker container and virtual machine. *Procedia Computer Science*, 171:1419–1428, 2020. Third International Conference on Computing and Network Communications (CoCoNet'19).

[8] Lily Puspa Dewi, Agustinus Noertjahyana, Henry Novianus Palit, and Kezia Yedutun. Server scalability using kubernetes. In *2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON)*, pages 1–4, 2019.

[9] Rizgar R. Zebari, Subhi R. M. Zeebaree, Amira Bibo Sallow, Hanan M. Shukur, Omar M. Ahmad, and Karwan Jacksi. Distributed denial of service attack mitigation using high availability proxy and network load balancing. In *2020 International Conference on Advanced Science and Engineering (ICOASE)*, pages 174–179, 2020.

[10] Yassin Abdulkarim Hamdalla Omer, Mohamed Ayman Mohammedel-Amin, and Amin Babiker A. Mustafa. Load balance in cloud computing using software defined networking. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pages 1–6, 2021.

[11] Seyeon Jeong, Doyoung Lee, Jian Li, and James Won-Ki Hong. Openflow-based virtual tap using open vswitch and dpdk. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2018.