electronics

MDPI

Article

HoneyFactory: Container-Based Comprehensive Cyber Deception Honeynet Architecture

Tianxiang Yu *, Yang Xin and Chunyong Zhang

School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China; yangxin@bupt.edu.cn (Y.X.); chertish_xxd@bupt.edu.cn (C.Z.)
* Correspondence: yutianxiang@bupt.edu.cn

Abstract: Honeynet and honeypot originate as network security tools to collect attack information during the network being compromised. With the development of virtualization and software defined networks, honeynet has recently achieved many breakthroughs. However, existing honeynet architectures treat network attacks as interactions with a single honeypot which is supported by multiple honeypots to make this single one more realistic and efficient. The scale and depth of existing honeynets are limited, making it hard to capture complicated attack information. Existing honeynet frameworks also have low-level simulation of protected network and lacks test metrics. To address these issues, we design and implement a novel container-based comprehensive cyber deception honeynet architecture that consists of five modules, called HoneyFactory. Just like factory producing products according to customer preferences, HoneyFactory generates honeynet using containers based on business networks under protection. In HoneyFactory architecture, we propose a novel honeynet deception model based on hmm model to evaluate deception stage. We also design other modules to make this architecture comprehensive and efficient. Experiments show that HoneyFactory performs better than existing research in communication latency and connections per second. Experiments also show that HoneyFactory can effectively evaluate deception stage and perform deep cyber deception.

Keywords: honeynet; honeypot; container network

check for updates

Citation: Yu, T.; Xin, Y.; Zhang, C. HoneyFactory: Container-Based Comprehensive Cyber Deception Honeynet Architecture. *Electronics* **2024**, *13*, 361. https://doi.org/10.3390/electronics13020361

Academic Editor: Krzysztof Szczypiorski

Received: 24 November 2023
Revised: 7 January 2024
Accepted: 12 January 2024
Published: 15 January 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

To deal with increasingly complicated network security threats, honeypot and honeynet [1] technologies have been proposed to enhance cyber deception defense, mainly serving as early warnings. The honeypot exposes services or vulnerabilities to attackers, attracting them to explore and exploit vulnerabilities. The main purpose of honeypots is to collect information on the attacker's behavior and attack data during the exploitation. This information can provide an early warning of potential attacks on the network, or identify potential vulnerabilities in advance to better protect the real system and network.

Honeynet is a network composed of honeypots aimed at improving the interaction between honeypots and attackers. The development of honeynets has passed through a long period, and the previous generations of honeynets enhanced the data control and capture capabilities of honeypots using customized gateway facilities, improving utilization of honeypot capabilities. However, there is still no correlation between different honeypots in this stage. With the development of SDN, container, and cloud technology, more honeynet architectures have been proposed to enable different types of honeypots, such as high-interaction honeypots and low-interaction honeypots, to work together. But essentially, from an attacker's perspective, they can still only launch attacks on a single device at one time, so the current honeynet is unable to capture deep-level network attacks. This process of attack is completely different from the current common network threats, so the role that cyber deception can play is also very limited.

Besides this, current honeynet and protected network are not similar in terms of services and network structure, so honeynet is not highly attractive to attacker. The attack information captured by this honeynet is also not very useful for analyzing potential threats and vulnerabilities of the protected network. According to surveys of honeynets in cloud [2] and IoT [3], most existing honeynets are manually configured, and there was a lack of unified definition for the content in honeypots, which makes it difficult to simulate protected network efficiently.

Therefore, we think that there are still four main issues in the current research on honeynet deception defense architecture: (1) attackers can only access one single honeypot at the same time, which is inconsistent with the general network attack process; (2) honeynet has a low simulation level and lacks attraction to attackers; (3) honeynet cannot capture deep-level attacks and can only engage in preliminary interaction with the attacker; (4) honeynet lacks testing metrics for deception effects. Existing research only qualitatively discusses how to switch deception modes, but lacks quantitative experiments.

To address the above issues, we argue that the cyber deception scenario simulated by honeynet should be more similar to real network environment under protection, and the interaction between honeypot and attacker should be more similar to the multi-step process of penetration attacks. In the experiment, we divided honeypots into three categories: simulation honeypot, traditional honeypot, and vulnerability honeypot. We constructed multiple container networks containing these honeypots based on docker, achieving a high simulation degree with protected network. We summarize network attack process and propose a novel honeynet deception model based on state model theory to evaluate attack and deception stage. Different types of honeypots are deployed and launched at different stages, allowing attackers to interact with more types of honeypots and achieve the goal of capturing deeper and more types of attack information.

The contribution of this paper is listed below.

- We propose a container-based cyber deception honeynet architecture, called HoneyFactory, which is more comprehensive than existing honeynet architectures and can provide a more realistic network deception environment. This architecture can capture deeper and more types of attack information. HoneyFactory also performs better than existing honeynet architecture in current honeynet test metrics.
- In HoneyFactory, we propose an environment learning and honeynet generation mechanism that can dynamically generate simulation honeynets based on business network under protection.
- In HoneyFactory, we propose a honeynet deception model based on the Gaussian Hidden Markov theory. Compared to previous honeynet that maintains the connection between attacker and a single honeypot, the honeynet deception implementation technology proposed in this paper evaluates attack stage, automatically arranges deception honeynet, and performs deep cyber deception. In stage evaluation problem, this model performs better than the existing models.
- We evaluate the performance of honeynet systems from multiple perspectives and propose novel test metrices for honeynet architecture.

2. Research Status of Honeynet

Honeynet is a network architecture that utilizes various honeypots, other network assets, and network environments to deceive attackers and analyze their malicious behavior.

The first generation honeynet [4] was proposed by the HoneyProject team, which created a network from honeypots and forwarded traffic to the nearby IDS system through port mirror at switch for traffic analysis and recording. Afterward, the HoneyProject team successively proposed the second and third-generation honeynets, which were based on virtual machine technology and integrated with kernel capture technology, abnormal traffic detection technology, covert transmission technology, data analysis technology, and other technologies. A comprehensive honeynet with data capture, data analysis, and data control capabilities was built on Honeywall [5], the integrated honeynet gateway.

The Honeyfarm further expands the application capability of honeynets. By deploying probes in the business network, suspicious traffic in the business network can be redirected to a managed honeynet. This redirection mechanism can be used to perceive and prevent worm virus [6]. However, the redirection of Honeyfarm requires access detection or anomaly detection of the network flow before forwarding, and attackers may recognize changes in the network environment.

Subsequent honeynet architecture was improved based on framework proposed by the HoneyProject team, further enhancing data capture, data analysis, and data control capabilities of honeynet.

The Honeybird [7] honeynet architecture enhances the data interaction and control capabilities of honeypots. This architecture divides attacker interaction into two parts: initial scanning and subsequent stages, with low-interaction honeypots and high-interaction honeypots used for processing in each stage.

Collapsar [8] honeynet architecture is based on SDN technology, which enhances the data capture capability of the honeynet. This study proposes a new architecture that redirects attackers' traffic in business networks to SDN-based virtual machine networks, enabling overall management of traffic within virtual networks.

Honeymix [9] honeynet architecture introduces proxy technology based on SDN technology and proposes multiple SDN controller modules to control communication between attackers and multiple honeypots containing the target service, and selects appropriate responses that eliminate fingerprint features to pass to the attacker. Subsequently, the HoneyProxy [10] honeynet architecture was proposed based on Honeymix. The architecture divided the interaction between attackers and honeypots into three stages: transparent mode, multicast mode, and relay mode, corresponding to initial attacks such as login attempts, fingerprint attacks, and complex interactions. Through deep packet inspection technology, the types of attacks were analyzed, allowing attackers to communicate with corresponding levels of honeypots.

Honeynet and honeypot have been used in many scenarios. There are many surveys about honeynet [11] and its applications on IoT [12] and industry [13].

In recent years, with the development of cloud and container technologies, many researchers focus on using cloud and container technologies to build honeynets. These studies dynamically manage honeypot containers, achieving efficient resource utilization, and utilizing SDN and proxy technology to enable attackers to access honeypots. HoneynetAIoT [14] proposed a container honeynet for collecting attack information to supplement attack data and improve detection model performance. IoT Honeynet [15] used multiport honeypots to capture IoT attacks. HoneyChart [16] proposed using container technology to create honeypots and collect data.

As the scale of honeynets increases, some studies are focusing on strategies for deploying honeynets [17].

However, current research only focuses on the communication relationship between attackers and a single honeypot in the honeynet, without comprehensively utilizing the honeynet network system and multiple honeypots to deceive attackers. The functions of honeypots are relatively independent, and the honeynet is not similar to the increasingly complex business network, with low simulation, and therefore has low attraction to attackers. The type of network attack expected to deceive is significantly different from popular network threats.

3. HoneyFactory Framework Design

3.1. Overview

We propose a novel honeynet architecture, HoneyFactory, to address the issues of existing honeynet frameworks that do not consider the specificity of application scenarios and can only maintain one single session to communicate with attackers at one time, making it unable to deal with complicated network attacks. The architecture of HoneyFactory is shown in Figure 1.

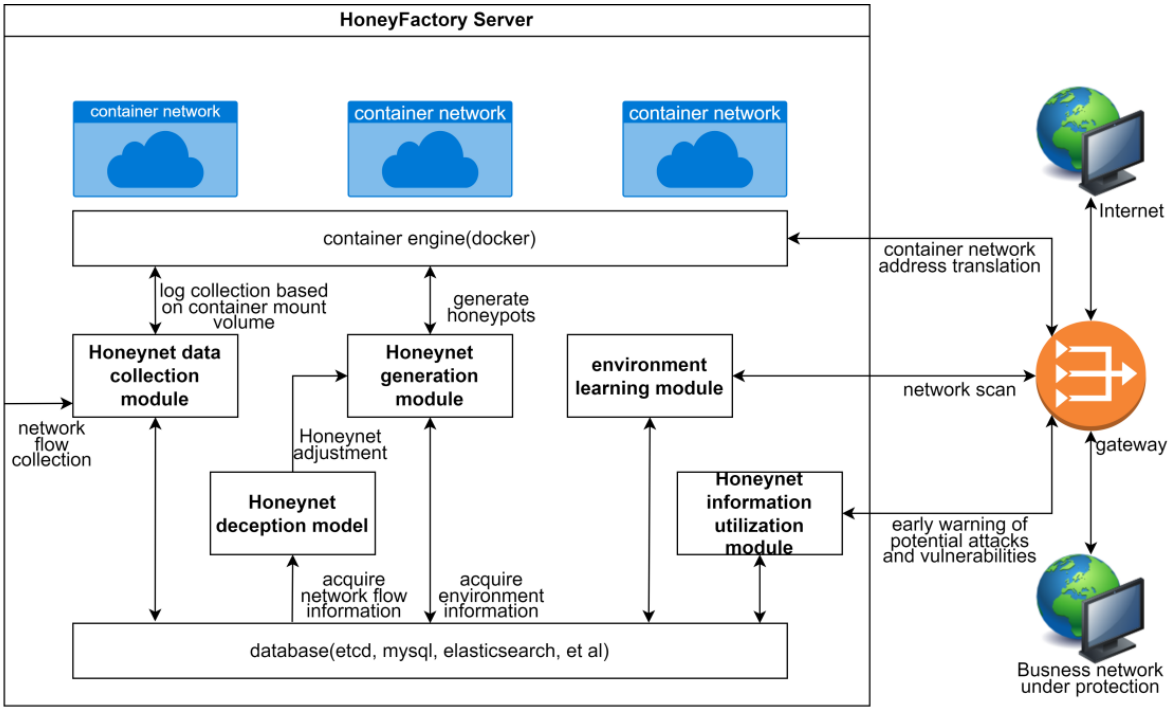


Figure 1. The architecture of HoneyFactory.

The HoneyFactory architecture creates multiple honeynets in the server, each one contains multiple honeypots. The server itself does not provide external services. The honeynet architecture assigns external IPs to some honeypots through gateways, enabling them to be accessed by Internet. Other honeypots collaborate with Internet-accessible honeypots to simulate intranet structure and perform deep cyber deception.

The HoneyFactory architecture consists of five parts.

The environment learning module learns the network, nodes, and application services information of business network under protection, and analyses learning results to output the optimized honeynet structure. The corresponding attack and defense methods are completely different in different network scenarios. Therefore, if the services in honeynet are significantly different from those of the business network, the attack information captured by honeynet is meaningless to the business network.

The honeynet generation module obtains environment learning results from a database, uses a container engine to generate honeypots, and creates a container honeynet. This module also receives results of honeynet deception model to adjust honeynet, such as adding or deleting honeypots.

The honeynet deception model evaluates the attack status based on collected attack information of honeynet, adjusts the honeynet situation, and provides more lateral movement space for the attacker. This module aims at capturing more types of attacks and alleviating potential internal network security threats in the business network.

Honeynet data collection module obtains logs inside the honeypot through container volume mount mechanism, and monitors server network card to collect all traffic sent to the honeynet, to collect the behavior and traffic information of the attackers.

The design motivation of the honeynet information utilization module comes from some related research on intelligent honeynet architecture. Existing research proposes to use the information collected in honeynet for applications such as vulnerability mining and cyber threat intelligence, but there is no specific implementation and design yet.

The goal of the honeynet information utilization module is to collect attack information and utilize it, such as attack tools and means of attack information. Therefore, the utilization of information collected by honeynet is also an important component of honeynet architecture. Some research studies also mention honeynet information utilization to

generate threat intelligence [18], improve intrusion detection [19], generate attacker profiles, and perform fuzz tests [20]. Most of them do not mention its implementation. However, this module involves too many technical fields, and its implementation is also not in the scope of this paper. We will focus on the design and implementation of other modules and the deception performance of HoneyFactory in the following content.

The following chapters will introduce modules of HoneyFactory architecture separately, including design, algorithm innovation, and model innovation. It should be noted that the design of each specific module in the HoneyFactory architecture not only refines the functions and algorithms but also points to some specific technologies and tools used to prove the feasibility of HoneyFactory architecture. The design of a module has a certain degree of universality, but subsequent research can also replace specific components, algorithms, and functions according to application scenarios to form a new honeynet architecture.

3.2. Environment Learning Module

Most current honeynet architectures are directive and guiding, defining the data flow between honeypots, attackers, and business networks. In practical applications, defenders often need to investigate their business network environment, collect honeypots themselves, and build honeynet according to the guidance of honeynet architecture. This unclear application scope limits the effectiveness of honeynets. To make the honeynet architecture more efficient in practical applications, we propose the first step of honeynet architecture to be dedicated to learning the business system [21], including hosts in the business network and its application services, to improve the authenticity of honeynet.

The honeynet server scans the network segment of protected business network and obtains the IP addresses of active hosts. After discovering TCP or UDP ports through port scanning, service scanning will query these ports to obtain the service type and version. The system integrates the collected information about network segments, hosts, and services on hosts. In this way, the honeynet system automatically collects and perceives service network information. The HoneyFactory environment learning module is shown in Figure 2.

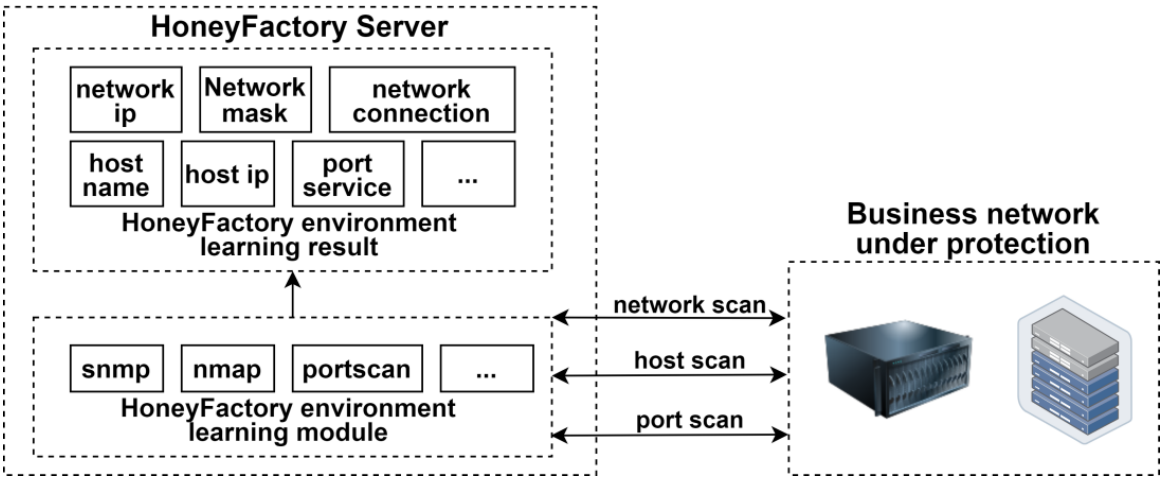


Figure 2. HoneyFactory environment learning module.

At present, the honeynet server has obtained overall environment information of a business network, and can customize honeypot and honeynet according to this information. The honeynet server can automatically generate docker containers and deploy them based on the nodes of the service network and their applications.

3.3. Honeynet Generation Module

Based on the result of environment learning, we can generate honeynets that are highly correlated with the protected network, and attack information collected by honeynet can better reflect the potential vulnerabilities and threats faced by the protected network.

The HoneyFactory architecture suggests using docker container technology to orchestrate and construct honeynets. Recently, with the rapid development of container technology and virtualization technology, there have been some studies on the orchestration and construction of honeynets based on related technologies, such as containers and microservices [22]. However, most container honeynets only focus on how to use microservices and containers to collectively manage and launch honeypots and use SDN, proxy, and other technologies to redirect attackers to just one specific honeypot and maintain connections between the attacker and that honeypot.

Some cyber security research studies have focused on building experimentation platforms based on containers [23]. However, the existing research on container honeynets only focuses on the proxy application of multiple honeypots and does not consider the impact among honeypots. The network deception in these studies is just one-step and does not consider the multi-step and long-term characteristics of penetrating a network from the perspective of attackers. Therefore, we propose the HoneyFactory honeynet generation module, which simulates the entire business network and constructs several container honeynets. The HoneyFactory honeynet generation module is shown in Figure 3.

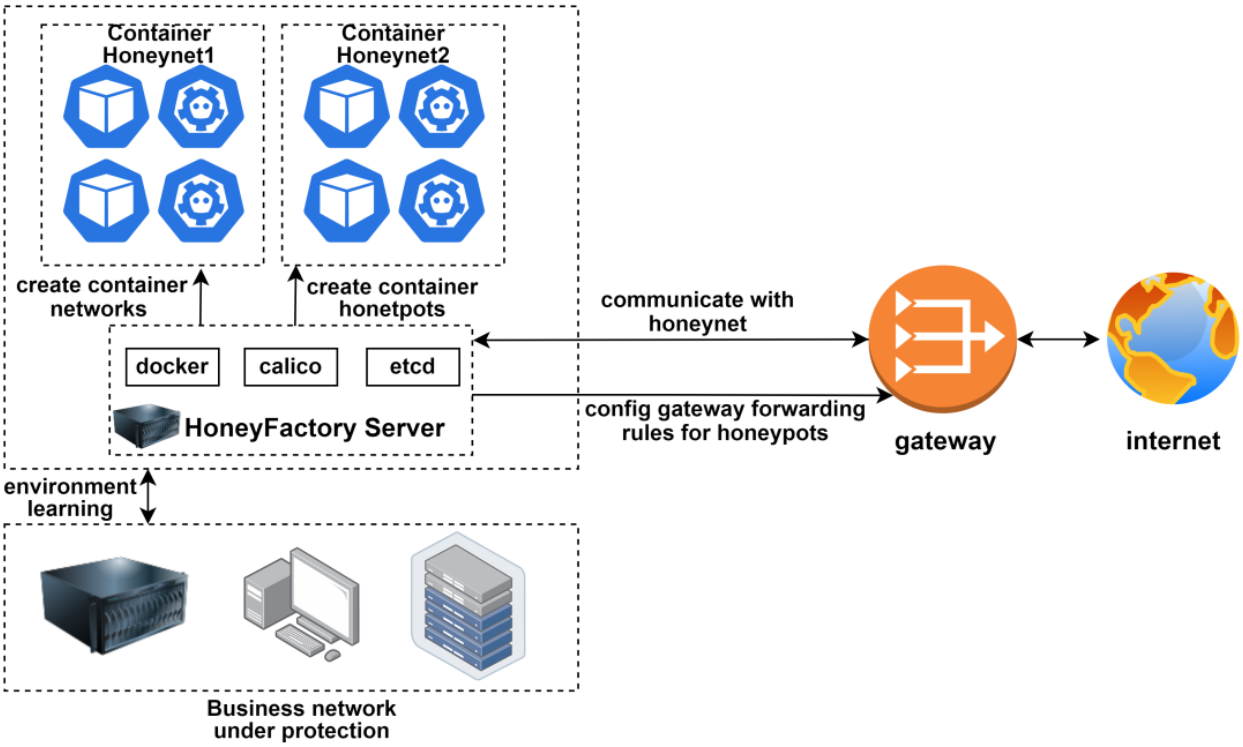


Figure 3. HoneyFactory honeynet generation module.

HoneyFactory detects and learns the business network through tools such as Nmap, portscan, and SNMP. Based on the learning result, honeynet generation module generates container information of honeypots in the generated honeynet and stores it into a database. At the same time, the honeynet generation module receives the deception action of the honeynet deception model and converts the action into modifying the container information of honeypots.

The honeynet generation module continuously monitors the database and updates honeynet using docker engine and calico component, including generating honeypots and

adjusting the network relationships between honeynets. The honeynet generation module also sends network information to the gateway for networks to address the translation configuration of honeynets.

Some honeynets can be accessed by attackers directly, while other honeynets can only be accessed by honeynets. The honeynet generation module combines container honeypot technology, container network technology, and NAT technology to create a multi-level comprehensive honeynet. There exists an intranet security structure within the honeynet, which can deeply deceive attackers.

To achieve this, the honeynet generation module needs to record honeypot information, network information, and network connection information in detail, and generate or adjust honeynets based on this information.

As part of this information, the honeypot information can be divided into multiple categories based on the type of honeypot. The first type is service simulation honeypot that imitates the business network host. This type of honeypot is lightweight and used to simulate services within the protected network, enhancing the authenticity of honeypot. However, it does not have real service functions and only simulates service with minimal resource consumption. The second type of honeypot is traditional honeypot, which uses container technology to deploy traditional web honeypots, SSH honeypots, and database honeypots and places them in the network to interact with attackers. This type of honeypot does not provide a foothold for further and internal attacks, regardless of whether they have vulnerabilities exploited by the attackers or not. The third type of honeypot is vulnerability honeypot, which contains applications with real vulnerability. This type of honeypot is disguised as an experimental testing server for various applications, used to interact with various attacks during the vulnerability exploitation and lateral movement stages of attackers, providing a foothold for further internal attacks so that honeypot can capture potentially related attacks. The honeypot itself is still a container, and the network environment that attackers can detect with this honeypot is still within container network, without posing a threat to the host.

This honeypot classification can be implemented based on file configuration and container configuration in container technology. For example, for the first type of honeypot, service simulation honeypot, honeynet generation module prepares some service files or crawler files for honeypot to simulate the business network through file configuration. For example, the module can configure the Apache. conf or nginx. conf files, and configure computer instructions for service start-up and log collection, which will be executed when the container is created. In this way, the network composed of these honeypots is similar to protected networks. For the second type and the third type of honeypots, the module can configure container images containing related deceptive interactive applications or vulnerability applications.

The honeypot information storage architecture is shown in Figure 4, which can also serve as the directory structure of the etcd database. Based on this architecture, the content of honeypots can be easily modified to cope with different types of attacks. In this paper, to demonstrate the feasibility of the system, we selected some existing honeypots and transformed them into container images, and constructed some honeypot images for specific services with vulnerabilities. In future use and further research, these honeypots can be modified according to the scene. The honeynet system can also integrate other cyber deception techniques through file configuration and command configuration, such as honeytoken [24] and camouflage network flow [25], making the system flexible and scalable.

Network information contains the honeypot ID and network segment IP. The connection information records the connection relationship between network segments.

At this point, HoneyFactory has completed the learning of protected business network and the construction of honeynet. In the early stage of HoneyFactory system operation, after environment learning, several container networks are created. Based on the services of hosts in the business network, simulation honeypots are created from basic images and

allocated to each subnet. In the initial interaction with attackers, for example, the attacker uses various reconnaissance tools. The system creates containers from various traditional honeypot images and configures them to run. These honeypots are allocated to container subnets for interactions with attackers. As the attacker gradually delves into the honeynet intranet environment, the system creates containers from various vulnerability honeypot images, allocates these honeypots to internal container subnets, and interacts with the attacker to capture more types of attacks.

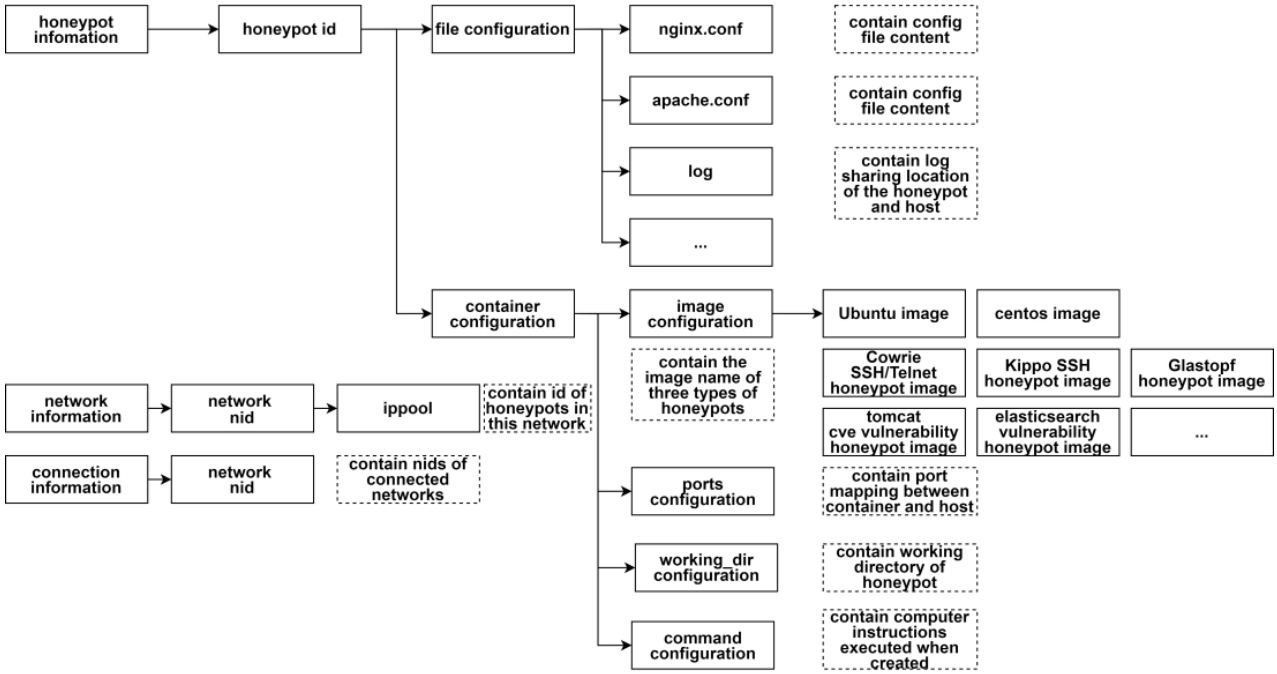


Figure 4. Honeypot and honeynet information storage architecture.

Some other honeypot research also mentioned honeypot content architecture to make them easy to design and use, like HoneyDoc [26], without considering the relationship between honeypots and honeynets. They also did not consider a unified definition of honeypot content based on some container orchestration techniques [27]. Some other honeynet research considers using the cloud-native orchestration technique [28], but it is still in its early stage.

Most of the existing honeynet research is used to maintain the connection between attackers and just one single honeypot at one time, naturally without considering deployment efficiency. HoneyFactory needs to deploy honeypots in multiple network segments and maintain network connections. The system should be able to quickly adjust and modify these honeypots and honeynets when an attacker's behavior changes. Therefore, we use above honeynet representation method to quickly update honeypot and honeynet content in a lightweight manner.

3.4. Honeynet Deception Model

In HoneyFactory, the honeynet deception model is responsible for adjusting cyber deception strategies based on the behavior of the attackers in honeynet. In this way, on the one hand, computing resources can be saved when the attacker does not perform an exploit; on the other hand, cyber deception actions can be targeted to the attack stage and the effectiveness and success rate of cyber deception will be improved.

Technologies related to the perception and detection of attacker behavior are relatively complex, including intrusion detection, network situation analysis, etc. These technologies are beyond the scope of cyber deception defense discussed in this paper. However, network

attacks and network traffic in honeynets are quite distinctive. Normal service networks contain a large amount of normal traffic. The traffic in honeynet is generally malicious.

To better generate specific cyber deception actions, we propose a honeynet deception model in HoneyFactory. This model will be used to evaluate the current deception stage and adjust honeypots and honeynets.

HoneyFactory is aimed at capturing multi-stage network attacks, collecting as much information as possible from attackers during various stages of penetration and intrusion. Therefore, the honeynet deception model first needs to define attack stages.

According to the definitions of cyber kill chain and ATT&CK framework for the attack and defense stages, this model selects some stages where both attacker and defender are involved and where traffic has obvious features. The attack stage and corresponding deception stage are divided into four stages. The first stage is the Reconnaissance stage, where attackers use some scanning tools to find network, host, and service information about honeynet. The second stage is the Exploitation and Persistent stage, where attackers exploit service vulnerabilities and maintain long-term access to hosts. The third stage is the Exfiltration and Control stage, where attackers establish transmission and control channel based on covert communication, such as dns tunnel. The fourth stage is the Discovery and Lateral Movement stage, where attackers begin to explore and attack the deep internal network.

The stages summarized above have removed the weaponization stage in the cyber kill chain that the defender did not participate in, and have integrated some stages with similar characteristics. During the Reconnaissance stage, attackers mainly interact with some open IPs in the network and service resources. The interaction is frequent, lasting for a long time with a small size of payload. The Exploitation and Persistent stage is the integration of execution phase, persistence phase, and privilege escalation phase in the ATT&CK framework. The number of packets sent during this stage is small, the duration is short, and the size of payload varies greatly depending on the type of attack. The attack at this stage is based on the vulnerabilities in specific applications or system services, and the location of the attacked host is still at the edge of honeynet. During the Exfiltration and Control stage, attackers mainly obtain sensitive files and create backdoors on the attacked machine. At this stage, there are a large number of traffic packets generated, with a large payload size, few service types, and a strong degree of traffic encryption. During the Discovery and Lateral Movement stage, attackers further penetrate the internal network based on controlled machines. The source and destination address of traffic generated during this stage are all in honeynet.

After the honeynet deception model obtains the attack stage based on traffic features in honeynet, the honeynet deception model only needs to select deception honeypots in the corresponding stage and allocate these honeypots to honeynet to generate effective and targeted cyber deception against attackers.

For example, during the Reconnaissance stage, the honeynet deception model can deploy some traditional honeypots of specific services to capture the attacker's basic information and generate threat intelligence. These traditional honeypots do not provide real services to attackers but is only used to collect attack information. This type of honeypot has some available samples on the network, such as the web service honeypot Glastopf [29] and the SSH service honeypot kippo [30]. This stage is also the main deception stage of the other existing cyber deception research, which only conducts preliminary deception on attackers and can collect the attackers' IP, port, and some types of attacks (botnets, spam, etc.), but cannot interact with deep-level attacks that exploit vulnerabilities and collect information. Even if these honeypots are not recognized by the attacker, the attacker will give up further attacks after consuming a certain amount of energy on these fake services.

When the interaction traffic of attackers in the Reconnaissance stage gradually decreases, the honeynet deception model can assume that the attacker is about to lose interest in the Reconnaissance stage and may enter the Exploitation and Persistent stage, conducting vulnerability detection and exploitation attacks on specific services. At this moment, the

honeynet model can deploy some vulnerability honeypots that contain real vulnerability applications, such as SSH vulnerability container honeypots, HTTP web vulnerability container honeypots, and MySQL vulnerability container honeypots. These honeypots are based on container technology to run real vulnerable service applications, built on various CVE vulnerabilities, and can be used to interact with real attacks and obtain attack samples, such as spring cve_2020_5410, Elasticsearch cve_2015_3337, log4j2-cve_2021_4104, etc. The services in these real vulnerability honeypots are mainly various application services, such as web services, database services, and log services. The appearance of these services in the network boundary is reasonable.

When the interaction traffic of attackers in the Exploitation and Persistent stage suddenly increases, the honeynet deception model can assume that the attacker is in the Exfiltration and Control stage. During this stage, the honeynet deception model will save the data in the attacked container to create a new container image, capturing the backdoor tools, and attack scripts uploaded by the attacker.

When attack traffic begins to appear inside honeynet, the honeynet deception model can assume that attacker has obtained control permissions of the boundary container honeypot and launched attacks inside the network. At this time, the attacker is in the Discovery and Lateral Movement stage. At this moment, the honeynet model can also deploy some vulnerability honeypots that contain real vulnerability applications. In contrast to the deployed honeypots in the Exploitation and Persistent stage, honeypot includes both application service vulnerabilities and system service vulnerabilities at this stage, like Linux cve. These operating system service vulnerabilities are reasonable to occur in the internal network and can capture more types of attack information.

We summarize the deception stages and actions of the honeynet deception model mentioned above and obtain the honeynet deception model, as shown in Figure 5.

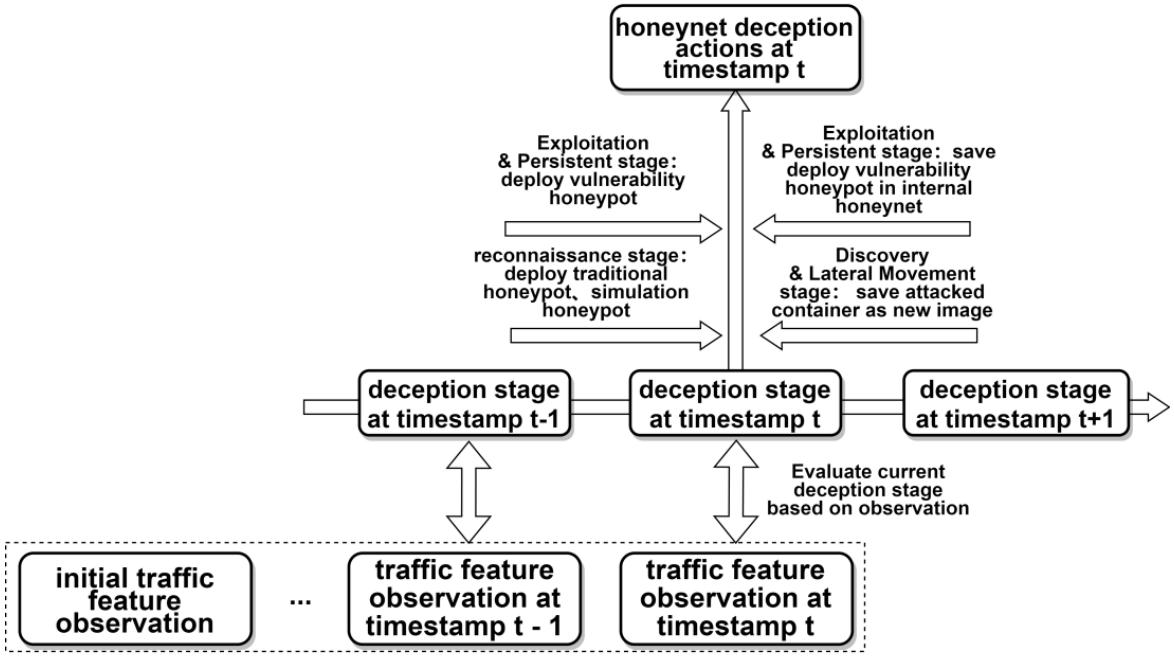


Figure 5. Honeynet deception model.

However, for the honeynet deception model, obtaining attack stage through traffic features within the honeynet is relatively difficult. Random phenomena are common in network security. The interaction traffic does have certain characteristics in terms of the overall distribution when attackers are in various stages of network attacks, but specific attackers may display completely different characteristics from each other. Therefore, the honeynet deception model needs to measure the uncertainty between attack stage and

traffic features, comprehensively considering features in multiple timestamps, and making the optimal reference of current attack stage.

The honeynet deception model includes four attack stages mentioned earlier. We use Equation (1) to represent the collection of stages.

$$Q = \{q_1, q_2, \cdots, q_N\} \tag{1}$$

At different timestamps, attackers may be in different stages, and their observation features are various features of traffic. The honeynet deception model can evaluate attack stage of attacker based on traffic features and thus execute corresponding honeynet deception actions to complete deep-level network deception. These traffic features include the number of packets, average size of packets, destination ports, flow duration, protocol types, etc. We use Equation (2) to represent traffic features during timestamp t.

$$V = \{v_1, v_2, \cdots, v_M\} \tag{2}$$

Due to the presence of continuous variables in the observed features, the honeynet deception model is a model with discrete state space but continuous observed features. Different states naturally show different observational features. We assume that the distribution of these continuous features can be fitted using the Gaussian distribution. We use Equation (3) to represent the probability of observation is V_t, while state i_t is q_n at timestamp t.

$$p(V_t|i_t = q_n) = \frac{1}{\sqrt{(2\pi)^M \det(C_n)}} \exp\left(-\frac{1}{2}(V_t - \mu_n)^T C_n^{-1}(V_t - \mu_n)\right) \tag{3}$$

In Equation (3), μ_n represents the expect of observation distribution when the state is q_n. C_n represents the covariance matrix of observation when the state is q_n.

$$\mu = E(V) \tag{4}$$

$$C = Cov(V) \tag{5}$$

The expect and covariance matrixes of multidimensional Gaussian distributions are parameters of the honeynet deception model, which can be learned from actual traffic data and corresponding attack stages. Some datasets can be used to train the model, such as the CICIDS2017 and DAPT2020 datasets, which contain attack traffic data and corresponding network attack stages.

At present, the honeynet deception model can be regarded as a multidimensional Gaussian Hidden Markov Model (Gaussian HMM). The state may also change over time. The probability of transition among states can be expressed by Equations (6) and (7).

$$K = \begin{bmatrix} k_{11} & \cdots & k_{1N} \\ \vdots & \ddots & \vdots \\ k_{N1} & \cdots & k_{NN} \end{bmatrix} \tag{6}$$

$$k_{ij} = p(i_{t+1} = q_j | i_t = q_i) \tag{7}$$

At present, the honeynet deception model has the following parameters: state transition matrix K, initial state probability π, multi-dimensional Gaussian distribution expect μ, and covariance C.

$$\theta = (K, \pi, \mu, C) \tag{8}$$

The first step is the construction of training data, and the dataset has already indicated the specific stages corresponding to the traffic. We extract traffic features from attack traffic

at different stages to form a sequence of attack stages. These sequences are training data. The training data representation is shown in Equation (9).

$$Data = \{((\mathrm{i_t}, \mathrm{V_t})|\mathrm{t} = 1 : \mathrm{T})\} \tag{9}$$

The set of states and observation features at different times within one sequence is shown in Equations (10) and (11).

$$Q' = \{\mathrm{i}_1, \mathrm{i}_2, \cdots, \mathrm{i_T}\} \tag{10}$$

$$O = \{V_1, V_2, \cdots, V_T\} \tag{11}$$

After the construction of training data, their parameters can be calculated iteratively using the Expectation Maximum (EM) algorithm [31].

$$\theta^{(k+1)} = \max_{\theta} \sum_{Q'} \log p(O, Q'|\theta) \log p\left(O, Q' \middle| \theta^{(k)}\right) \tag{12}$$

Take the model parameter π as an example to illustrate the calculation process.

$$\sum_{Q'} p(O, Q'|\theta) = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_T} \pi_{i_1} \prod_{t=2}^{T} K_{i_{t-1}, i_t} \prod_{t=1}^{T} \mathrm{p}(V_\mathrm{t}|i_t) \tag{13}$$

$$\pi^{(k+1)} = \max_{\pi} \sum_{i_1} \sum_{i_2} \cdots \sum_{i_T} \log \pi_{i_1} p\left(O, Q' \middle| \theta^{(k)}\right) \tag{14}$$

$$\pi^{(k+1)} = \max_{\pi} \sum_{i_1} \log \pi_{i_1} p\left(O, i_1 \middle| \pi^{(k)}\right) \tag{15}$$

After the parameters are solved, the honeynet deception model can transform the problem of estimating the attacker's current state based on long-term observation into a state model filtering problem, which can be solved using forward algorithms. The problem can be transformed based on the Bayesian formula.

$$p(i_T|O) = \frac{p(O|i_T)p(i_T)}{p(O)} = \frac{p(V_T|i_T)p(V_{1:T-1}|i_T, V_T)p(i_T)}{p(O)} \tag{16}$$

We can simplify Equation (16) based on the assumption of state model independent observation.

$$p(i_T|O) = \frac{p(V_T|i_T)p(i_T|V_{1:T-1})}{p(V_T|V_{1:T-1})} \tag{17}$$

Based on the homogeneous Markov assumption of the state model, we can further simplify and normalize Equation (17) to obtain its recursive calculation formula.

$$p(i_t|V_{1:T}) = normalize(p(V_T|i_T)p(i_{T-1}|V_{1:T-1})K) \tag{18}$$

Using the above algorithms, the honeynet deception model describes the uncertainty of a network attack. It can evaluate the probability of the current attack state based on an indefinite-length feature sequence. Based on the result, the honeynet deception model can take corresponding honeynet deception actions.

3.5. Honeynet Data Collection Module

In HoneyFactory, the honeynet data collection module is responsible for collecting log and traffic information of attackers within honeynet. The architecture of the honeynet data collection module is shown in Figure 6.

HoneyFactory does not limit the components used in honeynet data collection modules and databases. To better illustrate the effectiveness of the framework, we have provided a suitable combination of data collection tools and databases. Some other honeypot data collection research [32] has also mentioned these data collection tools, such as Elasticsearch. The data collected usually include the IP address, timestamp, destination port, and binary code [33]. Because all honeynet traffic passes through the host, we can use Packetbeat and tcpdump to directly capture traffic on the network card. This capturing collection is transparent to the attacker.

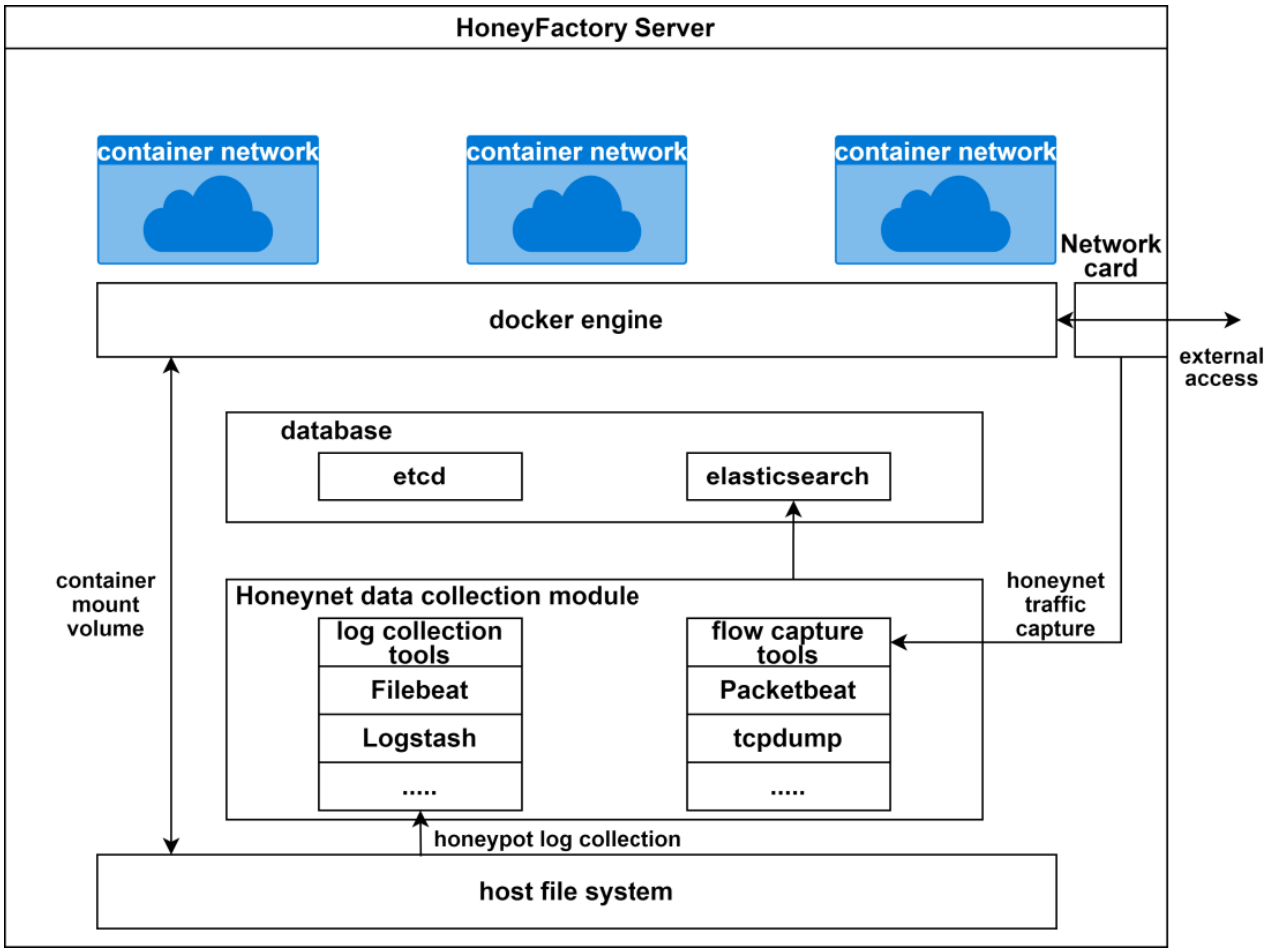


Figure 6. Honeynet data collection module.

The content of logs in the honeypot is not hidden from attackers. Previous research on Honeynethas proposed using kernel information collection methods and bypassing the original socket to directly use network cards to send data packets to prevent attackers from discovering that their information is being collected. We do not deny the progressiveness of this technology, but the core of HoneyFactory is the simulation of the real network environment to cope with more types, longer duration, and more complex network attacks. Therefore, the design of log collection module does not consider how not to be discovered by attackers. It is normal for server systems to have log collection systems. From the perspective of an attacker, these logs are only saved locally on the server and have not been collected remotely. Log collection relies on volume mount in container technology to collect log files of the operating system in the container, such as auth logs and console logs. The collected log files do not have execution permission, ensuring security of host and transparency to attackers.

4. Implementation

To verify the feasibility of HoneyFactory and test its performance, we implement a prototype system using some basic software, such as docker 18.09, docker-compose 1.27, calico 2.6.10, Java 8, and Python3.9. The software architecture of the HoneyFactory prototype system is shown in Figure 7.

The application running inside the backend system container in the major function containers is implemented by Java Spring Boot 2.6 and Vue 3 frameworks, which implement the main function of the HoneyFactory deception honeynet prototype system, including user interaction, data display, and honeynet management.

The honeynet deception model application running inside the main function container is implemented by Python. It used the GaussianHMM module of the hmmlearn library to model the problem of evaluating the hidden state which represents the honeynet deception stage. This container is used to generate specific deception defense actions and send results to the backend system.

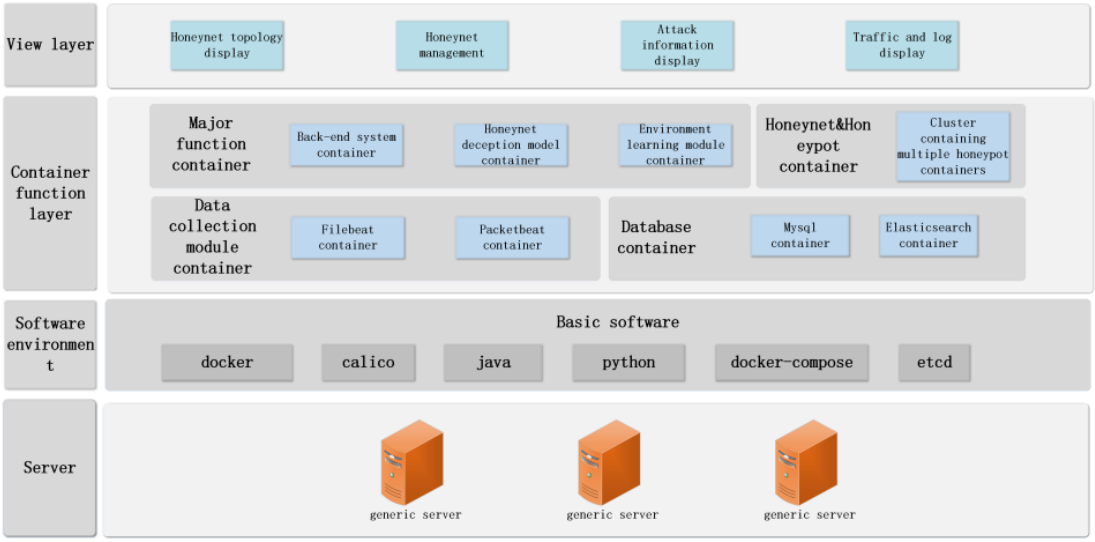


Figure 7. HoneyFactory prototype system software architecture.

The application running in the environment learning container is written in Python, which uses SNMP protocol and the Nmap network mapper tool to learn business network topology information and host service information.

The official 7.7.0 version is selected for the packetbeat container and filebeat container in the honeynet data collection container.

The Elasticsearch container and its data display Kibana container is the official 7.7.0 version, while the MySQL container is the official 8.0 version.

The backend system generates corresponding honeynets based on the environment learning topology and adjusts honeynet according to the result of the honeynet deception model. Honeynet is also divided into multiple network segments, each containing a large number of honeypots. The services generated in honeypot are also managed and maintained by the backend system. The honeypot management information is shown in Figure 8.

The prototype system can be deployed on one or more Linux universal servers according to the required computation resources. Container technology is the most important technology in the HoneyFactory prototype system. The various honeypots that make up the honeynet are containers, and the essence of the honeynet is also configuration of the container network and the control of container data flow. To facilitate a balanced utilization of each server, the prototype system was deployed to three Linux servers; each server was

equipped with CPU Xeon Silver 4210, 64 G memory, and 4T SAS hard disk. The deployment of the HoneyFactory prototype system is shown in Figure 9.

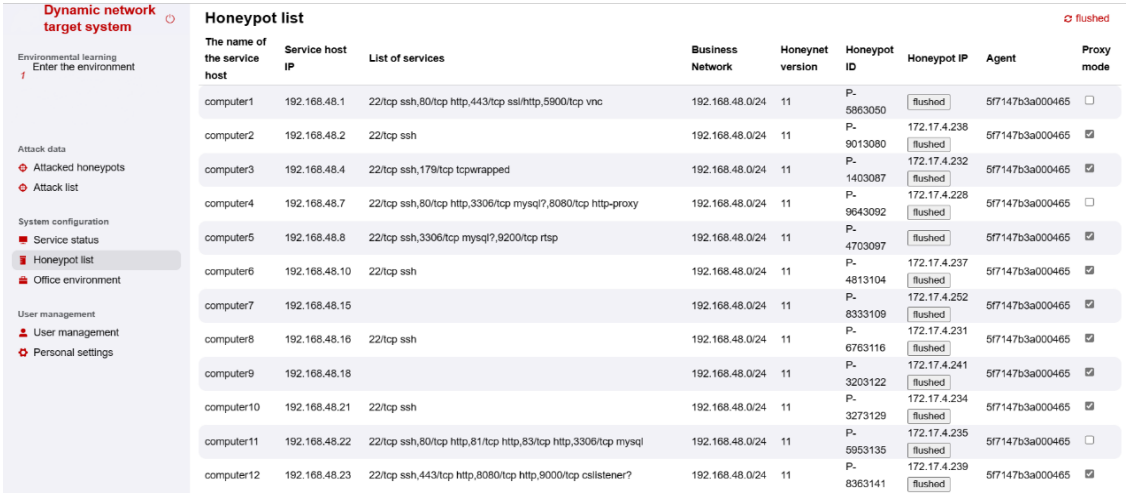


Figure 8. Honeypot management interface.

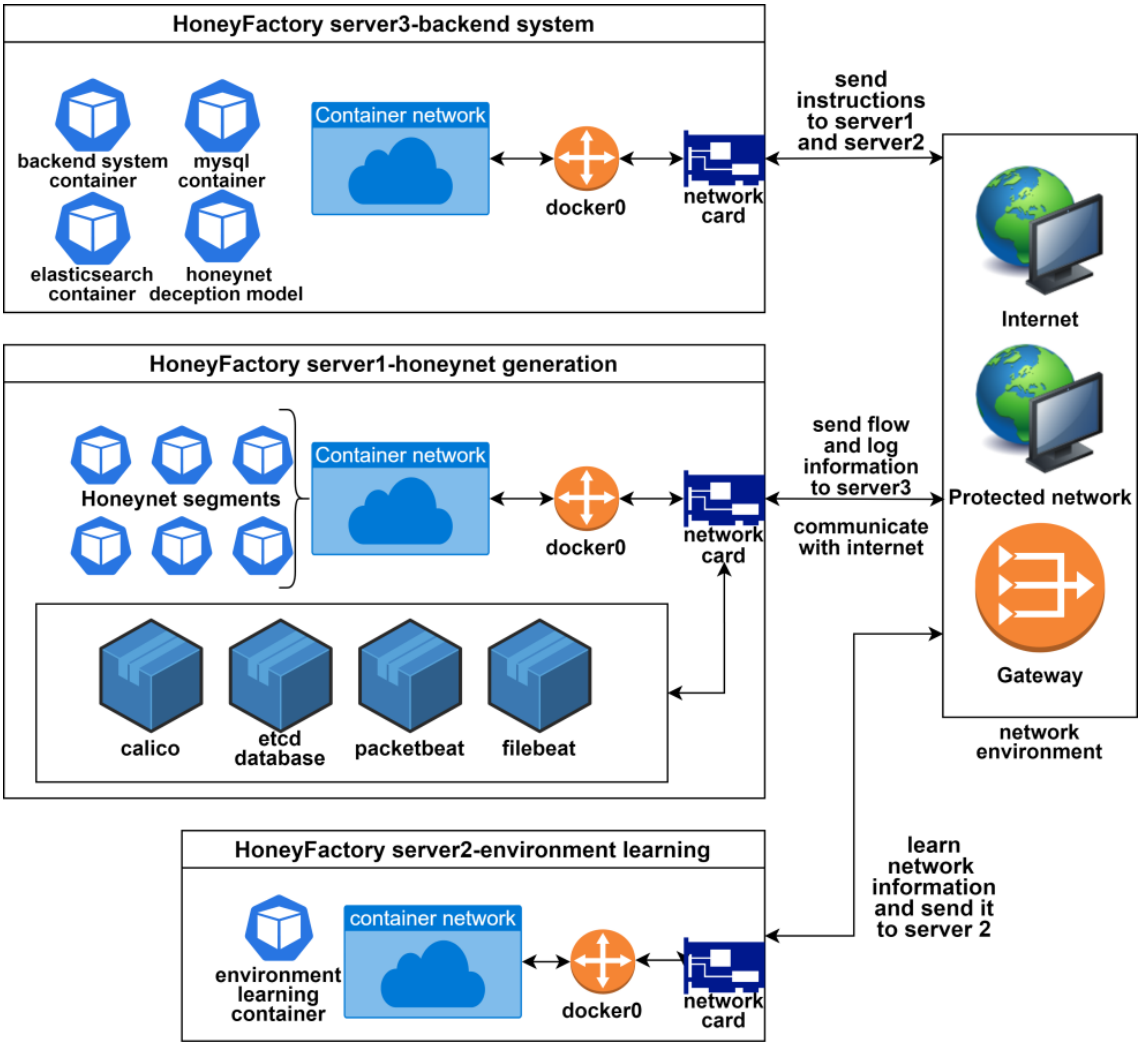


Figure 9. Deployment of HoneyFactory prototype system.

5. Evaluation

In the existing research related to honeynets, the evaluation and test metrics of honeynet mainly focus on the feasibility and communication latency of honeynets. However, honeynet is a cyber deception defense technology, whose core purpose is to deceive attackers, induce them to attack, and collect information. However, few studies have evaluated the effectiveness and performance of the honeynet system from the perspective of the attacker. The network security scenario directly determines the specific tools and methods of network attack and defense. Therefore, to evaluate the deception effectiveness of honeynet from the perspective of an attacker, it is necessary to define the scenario. This work provides new test metrics for evaluating the effectiveness of cyber deception defense.

The previous system implementation part has already provided some basic implementation effects of HoneyFactory. HoneyFactory is now able to generate honeynets based on the protected network, and adjust honeypots and honeynet based on result of the honeynet deception model. This chapter evaluates honeynet performance and compares it with other honeynet architectures.

The test metrics for honeynet vary among existing honeynet architecture studies. In this chapter, we propose a set of test metrics for the effectiveness of honeynet deception based on relevant research on honeynet architectures.

In this chapter, we first introduce the evaluation of honeynet communication, using regular honeynet test metrics from previous honeynet research, including communication latency and connections per second. Then, we introduce the evaluation of honeynet simulation. After this, we will introduce how the honeynet deception model is trained and validated. Then, we will introduce the honeynet deception effect when the honeynet deception model is used in real attack/defense scenario, and we treat the performance of honeynet deception model as a novel test metrics of honeynet system. Finally, we summarize above experiment results and make an overall comparison with other studies.

5.1. Honeynet Communication Evaluation

The first test metrics is the honeynet communication latency time. The previous honeynet architecture uses proxy technology, and HoneyFactory honeynet architecture also uses a container virtual gateway and collects data on the network card. These operations targeting traffic will inevitably cause certain delays and increase the risk of being detected by attackers.

The experiment measured the communication delay of honeynet, and the protected network and the results are shown in Figure 10.

According to the experiment results, the latency caused by the HoneyFactory honeynet architecture is between 0.4 and 0.8 milliseconds, which is smaller than that within the existing research, 0.5 to 1.2 milliseconds.

The second test metrics are connections per second (CPS) from the same research as the first test metrics. The honeynet architecture can affect the forwarding of data packets, thus affecting the number of connections per second. The experiment measures the number of HTTP and HTTPS connections per second in the protected network and HoneyFactory honeynet. The experimental results are shown in Figure 11.

According to the experiment results, honeynet reduces the number of connections per second by about 30%, which is better than the 41–55% reduction in the existing research.

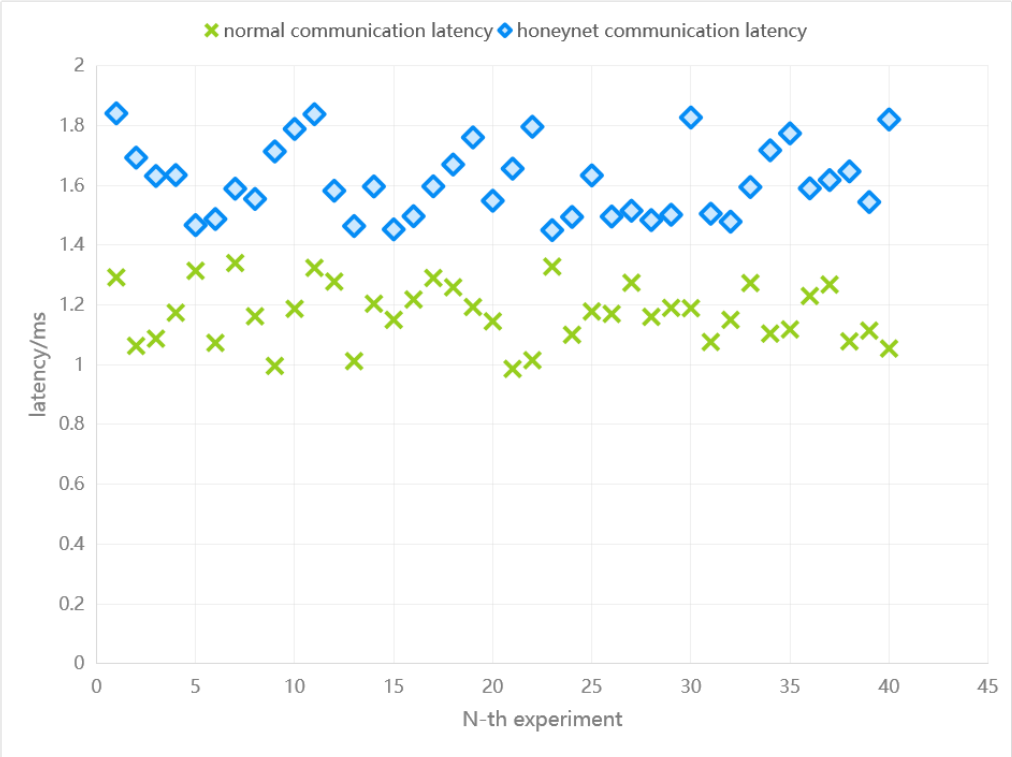


Figure 10. Network communication latency experiment.

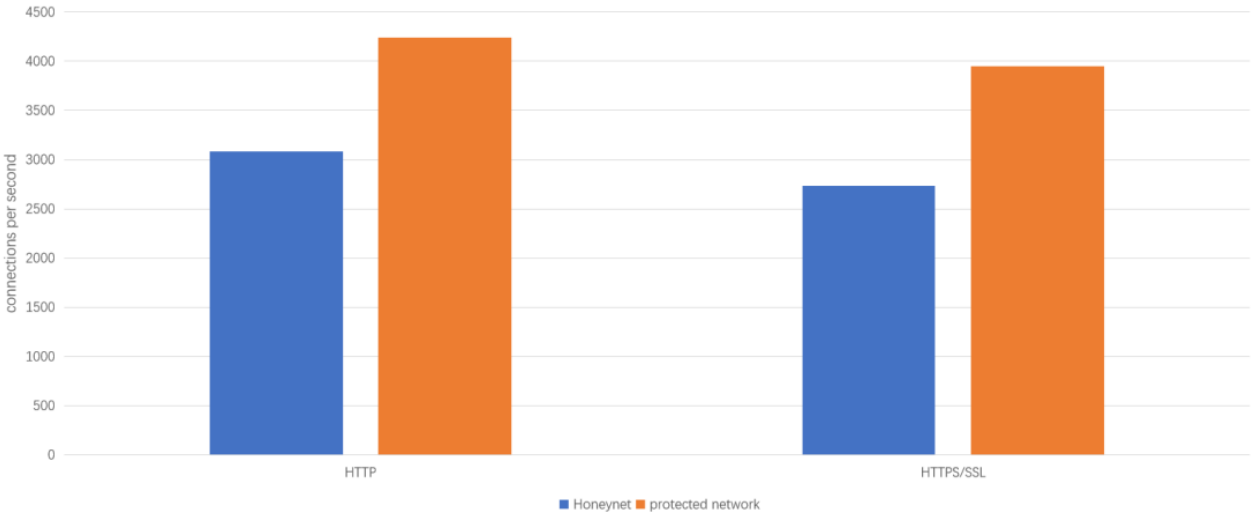


Figure 11. Connections per second experiment.

5.2. Honeynet Simulation Evaluation

The environment learning module and honeynet generation module are responsible for simulating the protected network. Regarding the honeynet simulation, different honeynet research has adopted different methods to complete this task. Some studies, such as HoneyProxy and Honeymix, view honeypots as attackers' access points with few internal services, resulting in lower simulation levels. Other studies, such as Honeychart and Honeywall, view honeypots as fake hosts to lure attackers, resulting in a high level of simulation. However, most studies require manual configuration of honeypots, making it difficult to apply to large-scale networks and evaluate their simulation effectiveness. In this

section, we evaluate the effectiveness of our honeynet simulation from three perspectives: network simulation, host simulation, and service simulation.

The environmental learning module detects the network segment information, host information, and service information of the protected network, and the obtained information examples are shown in Figure 12.


```
{
    "network_segment": "192.168.60.0/24",
    "link": [
        "192.168.4.0/24"
    ],
    "computers": [
        {
            "name": "computer1",
            "IP": "192.168.60.1",
            "service": "22/tcp ssh,53/tcp domain,80/tcp http,
            88/tcp kerberos-sec?,389/tcp ldap,443/tcp ssl/https,
            514/tcp shell?,636/tcp ssl/ldap,902/tcp iss-realsecure,
            2020/tcp xinupageserver?,6502/tcp ssl/http,8010/tcp xmpp,8084/tcp ssl/unknown",
            "web": "http-192.168.60.1-80,https-192.168.60.1-443"
        },
        {
            "name": "computer2",
            "IP": "192.168.60.100",
            "service": "22/tcp ssh,80/tcp http,427/tcp svrloc?,
            443/tcp ssl/https,902/tcp ssl/vmware-auth,5988/tcp wbem-http,
            5989/tcp wbem-https,8000/tcp http-alt?,8300/tcp tmi?,9080/tcp ssl/soap",
            "web": "http-192.168.60.100-80,https-192.168.60.100-443,http-192.168.60.100-5988"
        }
    ]
},
```


Figure 12. Environment learning sample result.

The honeynet topology generated based on environmental learning results is shown in Figure 13.

We use an internal office network as the protected network for honeynet simulation, which includes 3 network segments and 60 hosts. The honeynet simulation result is shown in Table 1. All network segments and hosts can be simulated successfully. Some services fail to be simulated due to their inability to detect or their difficulty with performing automatic configurations and simulations.

Table 1. Honeynet simulation evaluation result.

Model	Network Simulation	Host Simulation	Service Simulation
HoneyFactory	100% (3/3)	100% (60/60)	91.6% (870/950)

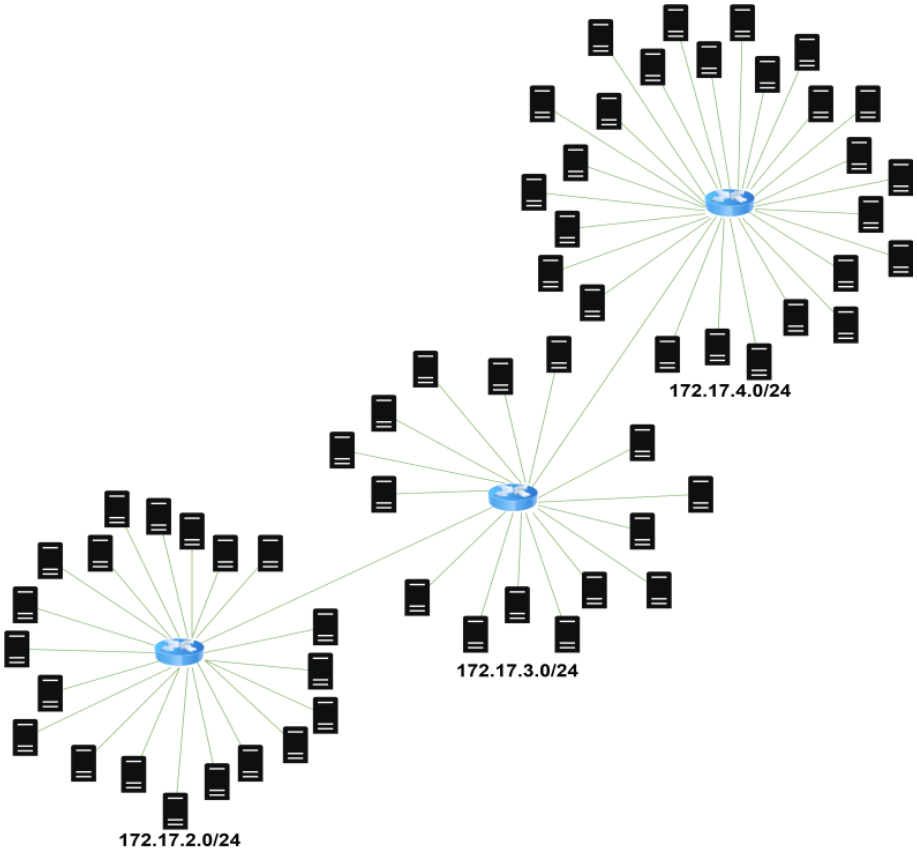


Figure 13. Generated honeynet topology.

5.3. Honeynet Deception Model Training and Validation

Due to the honeynet deception model having four labels, conventional test metrics are not fully applicable to multi-classification models. The test metrics of the honeynet deception model are redefined. Accuracy is the proportion of identifying input data as correct labels. Each type of label calculates its F1 score, with its own samples as positive samples and the remaining samples as negative samples. The overall F1 score of the model is the average F1 score of each of the four types of labels.

The honeynet deception model experiment uses the attack traffic data of the DAPT2020 dataset and its corresponding attack stage label data. The attack traffic in this dataset can be divided into four types based on the attack stage: investigation stage, vulnerability exploitation stage, data exfiltration stage, and lateral movement stage. This is consistent with the four attack stages in the honeynet deception model.

A single row of data in the DAPT2020 dataset corresponds to the entire process of a single session, with IP and port number, and a large distribution of duration. The model trained directly using a single piece of data and its labels will be more difficult to apply in real-time scenarios, such as deep learning models with fixed input vectors, because the attack traffic information recorded in the dataset is the information after the attack is fully completed, not the information during the attack. Therefore, to correctly use this dataset, it is necessary to preprocess the data and average the traffic features of the same type within a certain period to obtain the final traffic features.

During training, the model sets several equally spaced sampling points, obtains several traffic data of the same attack type in the DAPT2020 dataset at each timestamp, and averages the features of these data, recording the observed features at each timestamp.

When applied in real-time scenarios, the corresponding traffic pcap files are saved regularly according to time intervals, and the same types of feature data are extracted using the same feature extraction tool CICFlowMeter as DAPT2020. The observed feature data

at this time are recorded and are input into the model together with historical observed feature data for inference of the current attack stage.

DAPT2020 is the most suitable dataset for the honeynet deception model, as it has classified network attack traffic into different attack stages, perfectly matching these deception stages defined in the honeynet deception model. To supplement this single dataset, we also investigated other network attack traffic datasets. However, most datasets are difficult to use because their data labels are only attack-or-not labels or attack type labels, not including attack stage labels. We finally chose CICIDS2017 as our supplementary dataset, which includes 15 types of attacks, and we manually labeled their attack stages to use this data in the model. For data with "portscan" and "Dos" labels, we labeled these data as the Reconnaissance stage; for data with "web attack", "ssh" and "ftp" labels, we labeled these data as the Exploitation and Persistent stage; for data with "infiltration" label, we labeled these data as the Exfiltration and Control stage; for data with "bot" label, we labeled these data as the Discovery and Lateral Movement stage.

The data flow of model training and practical application is shown in Figure 14. We trained and validated the honeynet deception model based on the DAPA2020 and CICIDS2017 datasets to demonstrate its feasibility and show its experimental performance. We also tested the honeynet deception model based on the attack data collected in the honeynet environment to evaluate its real deception effect. We will talk about this honeynet deception effect evaluation in the next subsection.

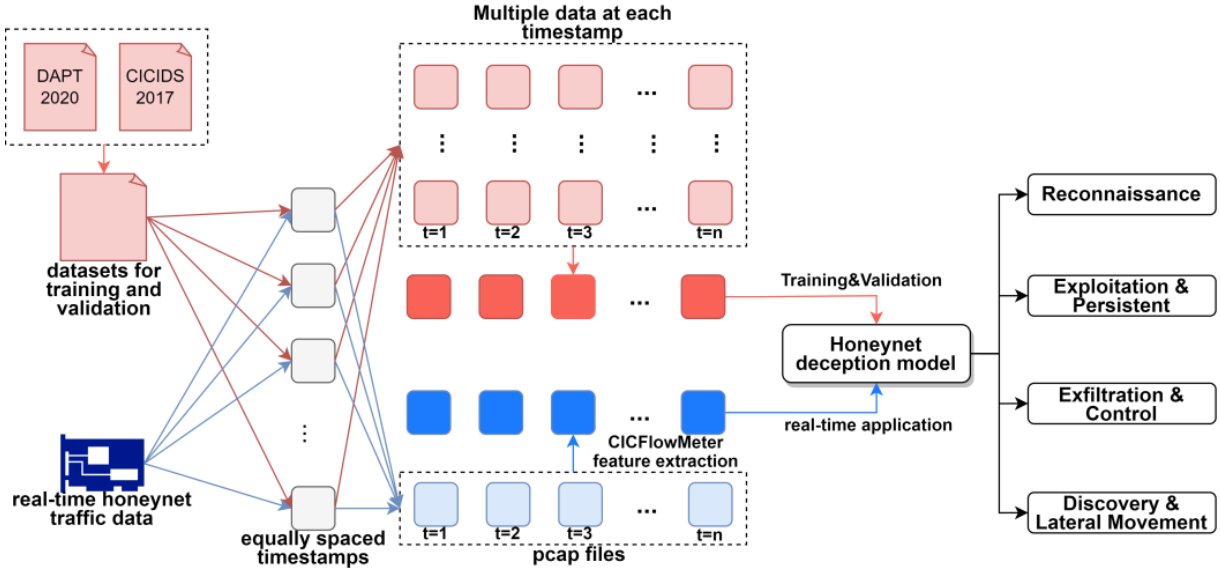


Figure 14. Data flow of honeynet deception model training and practical application.

The model training and validation results are shown in Figure 15. The F1 score of the Reconnaissance stage label is identified by R-F1, the F1 score of the Exploitation and Persistent stage label is identified by P-F1, the F1 score of the Exfiltration and Control stage label is identified by E-F1, and the F1 score of the Lateral Movement Stage label is identified by L-F1.

In the comparison experiment, we compared the honeynet deception model with AI models widely used in network traffic detection and analysis. The other models based on deep learning, due to the small amount of data and the large number of features, have shown some overfitting phenomenon in the experiment, and their overall performance is inferior to the honeynet deception model, and the calculation time is longer. All other deep learning models are built on Python TensorFlow. Unlike the honeynet deception model, all other models cannot handle variable length input data. Regarding LSTM and GRU, we use fixed length network traffic sequences as input data. With regard to CNN, we process the traffic sequence into a graph for input. Concerning MLP, we use single traffic as the input.

The comparison results are shown in Table 2. "ACC" represents accuracy. "F1" represents the average F1 score. "Rec" represents the average recall score. "Time" represents the time required to run these epochs.

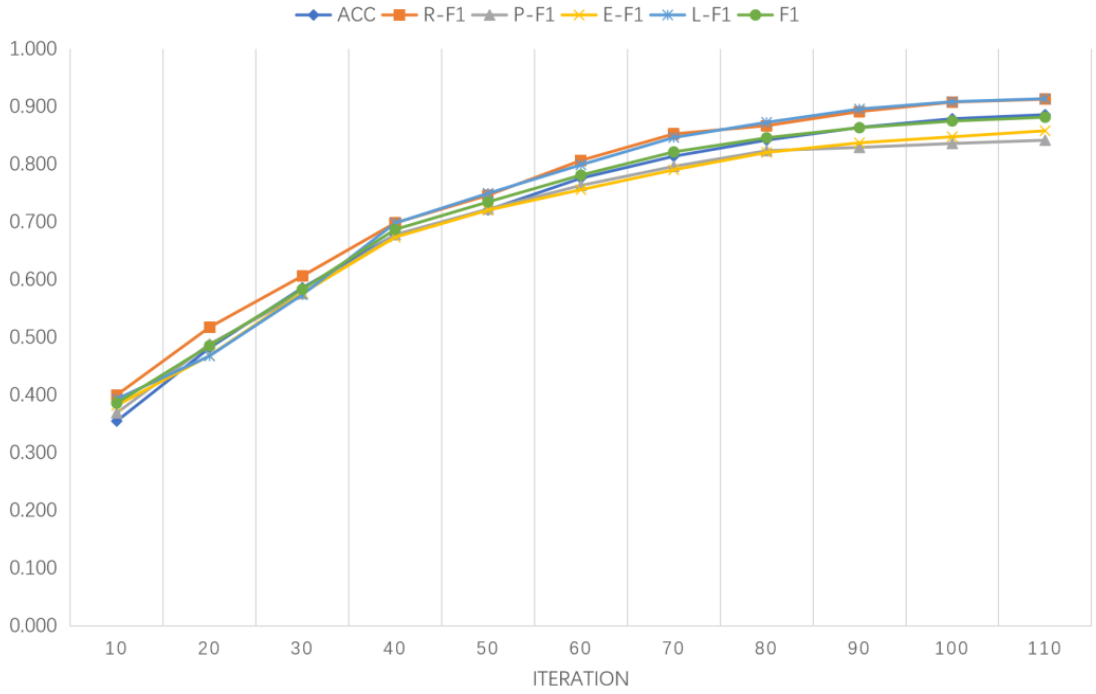


Figure 15. Experiment result of honeynet deception model.

Table 2. Honeynet deception model comparison with other regular network traffic detection models.

Iteration	Honeynet Deception Model				LSTM [34]				GRU [35]				CNN [36]				MLP [37]			
	ACC	F1	Rec	Time	ACC	F1	Rec	Time	ACC	F1	Rec	Time	ACC	F1	Rec	Time	ACC	F1	Rec	Time
10	0.355	0.362	0.366	2 m	0.336	0.315	0.318	7 m	0.308	0.295	0.297	6 m	0.190	0.186	0.210	9 m	0.238	0.219	0.229	5 m
20	0.482	0.486	0.491	4 m	0.434	0.405	0.407	14 m	0.393	0.375	0.377	13 m	0.301	0.297	0.309	20 m	0.398	0.410	0.434	10 m
30	0.586	0.583	0.584	6 m	0.541	0.516	0.517	21 m	0.506	0.490	0.491	20 m	0.422	0.422	0.429	30 m	0.479	0.487	0.500	15 m
40	0.678	0.687	0.690	8 m	0.619	0.596	0.598	28 m	0.587	0.572	0.574	27 m	0.508	0.510	0.516	38 m	0.554	0.554	0.564	20 m
50	0.721	0.735	0.737	9 m	0.708	0.690	0.692	34 m	0.674	0.664	0.667	33 m	0.600	0.605	0.612	49 m	0.611	0.602	0.612	25 m
60	0.776	0.781	0.784	11 m	0.767	0.754	0.758	42 m	0.743	0.735	0.740	40 m	0.668	0.675	0.685	58 m	0.623	0.612	0.621	29 m
70	0.814	0.822	0.825	13 m	0.793	0.781	0.785	50 m	0.774	0.766	0.771	46 m	0.705	0.711	0.720	68 m	-	-	-	-
80	0.842	0.846	0.847	15 m	0.808	0.797	0.798	58 m	0.790	0.781	0.785	52 m	0.719	0.720	0.725	80 m	-	-	-	-
90	0.864	0.864	0.866	16 m	-	-	-	-	-	-	-	-	0.734	0.732	0.736	92 m	-	-	-	-
100	0.879	0.875	0.878	18 m	-	-	-	-	-	-	-	-	0.745	0.740	0.744	104 m	-	-	-	-
110	0.886	0.882	0.884	20 m	-	-	-	-	-	-	-	-	0.753	0.747	0.750	115 m	-	-	-	-
130	-	-	-	-	-	-	-	-	-	-	-	-	0.758	0.751	0.753	140 m	-	-	-	-

5.4. Honeynet Deception Effect Evaluation

Previously, the description of deception effects in honeynet- and honeypot-related studies was mostly qualitative, such as the level of honeypot simulation [38], and different complexity of the computational models for honeynet switching network deception models in Honeymix and Honeyproxy schemes. There are little quantitative descriptions of the effectiveness of honeynet deception in the existing research. The HoneyFactory honeynet architecture uses a honeynet deception model that divides attacks into four stages, with each stage selecting different types of honeypots or performing corresponding deception actions. In the experiment, we conduct attack testing on honeynet and divide our attacks into four stages. We checked the corresponding deception stages and corresponding deception actions of honeynet. If the attack stage is consistent with the deception stage evaluated via honeynet, the deception is considered successful, and the success rate of honeynet deception is calculated based on this.

The honeynet deception environment in the experiment is the same as the environment of the system implementation chapter above. Initially, the protected network is simulated to construct honeynet. Both the honeynet and the protected network have three network segments. Only the honeypot in segment 1 can be accessed via external networks, while segment 1 can access segment 2 and segment 3.

In the experiment, each attack stage contains multiple attacks, and the attack information is shown in Table 3. All attacks are carried out by humans using attack tools, such as Metasploit, Acunetix Web Scanner, Nmap, Nessus, and Iodine. All attacks start and end within three days.

Table 3. Attack information in each stage.

Stage ID	Stage Name	Attack Method
1	Reconnaissance	Nmap, Metasploit ssh_login, Nikto, Dirbuster
2	Exploitation and Persistent	sqlmap, php reverse shell, web component and server vulnerability exploitation by burpsuite
3	Exfiltration and Control	dns covert channel, ftp, scp, dga
4	Discovery and Lateral Movement	Nmap, MySql vulnerability exploitation, vsFTP vulnerability exploitation, Metasploit unix exploitation

Each attack stage corresponds to each deception stage, and the honeypot information of deception containers that can be deployed in different deception stages is shown in Table 4.

Table 4. Cyber deception actions in each stage.

Stage ID	Stage Name	Deception Actions
1	Reconnaissance	Deploy kippo SSH traditional honeypot, cowrie ssh/telnet traditional honeypot, simulation honeypots
2	Exploitation and Persistent	Deploy glastopf web traditional honeypot, honeypots with real web vulnerability
3	Exfiltration and Control	Create image from attacked container
4	Discovery and Lateral Movement	Deploy honeyprint traditional honeypot, honey-FTP traditional honeypot, honeypots with real database vulnerability, honeypots with Linux vulnerability

During the Reconnaissance stage, the attacker scans and detects the network while attempting preliminary attacks, such as a weak password attack. In deception methods, the kippo SSH honeypot container is deployed to deceive a large-scale SSH brute force attack and to record the attacker's dictionary and operations after entering the honeypot. The

cowrie SSH/telnet honeypot is deployed to deceive attacks against SSH and telnet services. Many simulation honeypots are deployed to increase the simulation degree of honeynet and continuously attract attackers, whereas real services are deployed but do not provide deep interactions. For example, HTTP web services only have static pages, while FTP and RTSP services do not have actual content to be transmitted.

During the Exploitation and Persistent stage, the attacker has gained a certain understanding of service types of accessible hosts within the network. At this point, the attacker has realized that some accessible hosts may be traditional honeypots, while other simulation honeypots still have penetration and utilization value from the attacker's perspective. The services may also have potential vulnerabilities. In this stage, the attacker uses specific exploit tools to attack the honeynet. In deception, the defender can deploy web traditional honeypot and honeypots with real web vulnerabilities to collect attack information. For example, a honeypot with real web vulnerability provides the attacker with a possible location for vulnerability exploitation, and collect attack information while providing a foothold for deeper deception in the future.

In the Exfiltration and Control stage, the attacker establishes contact with the Internet through covert channels. At this point, the deception honeynet creates an image from the attacked container to save covert communication tools and targets of the attacker for later analysis.

In the Discovery and Lateral Movement stage, attackers scan other container segments based on the foothold obtained from the Exploitation and Persistent stage and attack various services that appear more frequently in internal networks. At this point, the deception honeynet can set up traditional honeypots for these application services or set up honeypots with some real application vulnerabilities to capture attack methods and tools against such applications, increasing the honeynet's ability to capture more types of attacks.

Compared to deploying honeypots in an intranet that can only collect information on the possible attack after the intranet is truly invaded, HoneyFactory constructs a fake intranet to collect possible attack information. Compared to other honeynet solutions, HoneyFactory deception honeynet has a better simulation and captures more types of attack.

The results of the deception experiment are shown in Table 5. The attacker is only in one attack stage at a certain time. If the attack stage evaluated by honeynet deception model is correct, it is considered that the deception has been successful.

Table 5. Deception experiment result.

Stage ID	Stage Name	Deception Success Rate	Overall Rate
1	Reconnaissance	82.3% (56/68)	71.3%
2	Exploitation and Persistent	53.2% (33/62)	
3	Exfiltration and Control	66.1% (39/62)	
4	Discovery and Lateral Movement	85.4% (53/62)	

The experiment result shows that the success rate of deception is slightly lower than the theoretical success rate of the deception honeynet model, which may be due to the differences between the training dataset and the actual deception experimental traffic.

Other honeynet studies have also mentioned the determination of attack stages and the success rate of deception but with only qualitative analyses. Honeymix calculates the connection weight based on the duration and number of modifications to determine whether to adjust the response content to the attacker. HoneyProxy switches between different deception stages based on the duration and attack records. However, existing research has not discussed the actual deception effect of the deception stage switching its mechanism. We supplement these test metrics in this paper.

The result also indicates that HoneyFactory can capture different types of attacks and collect their information. For example, HoneyFactory can capture an attacker's profile

information during the Reconnaissance Deception stage, including their IP and targeted port; HoneyFactory can capture an attacker's exfiltration tools, such as iodine, in the Exfiltration and Control Deception stage; HoneyFactory can capture attack traffic samples during the Exploitation Deception stage, such as web vulnerability exploit traffic. Other existing honeynet research only pays attention to one stage of deception and corresponding attack information collection. For example, Honeymix and Honeyproxy can only collect scans and few exploit attack data due to their small network scale; Other SDN-based honeynets, such as Honeychart and Collapsr, only detail the network structure without discussing how to use it to perform deception and can only capture a few kinds of attacks. Based on the above content, we conclude that the data collection capability of HoneyFactory is relatively high, while the data collection capability of other studies is relatively low.

5.5. Comparison with Other Studies

The comparison with other studies is shown in Table 6.

Table 6. Comparison with other honeynet research.

Honeynet Name	Honeynet Communication Latency	Honeynet Connection Reduction	Honeynet Deception Effect Evaluation	Honeynet Simulation	Honeypots Accessible to Attacker	Honeypot Scale	Honeynet Data Collection
HoneyFactory	0.4–0.8 ms	30%	71.3%	High, automatic learning	Multiple	About 100	High
Honeymix [9]			Qualitative description	Low	Single	5–10	Low
Honeyproxy [10]	0.5–1.2 ms	41–55%	Qualitative description	Low	Single	5–10	Low
Honeychart [16]	-	-	-	High, manual configuration required	Multiple	-	Low
Honeywall [5]	-	-	-	High, manual configuration required	Multiple	3	Low

From the comparison results, it can be seen that the HoneyFactory has a larger honeynet scale and can therefore capture deeper attacks. The honeynet has a high simulation level, and attackers can access multiple honeypots at the same time; thus, attacks that happened in the honeynet are highly similar to real network attacks. HoneyFactory performs better in honeynet latency and connection, proposing new test metrics for honeynet deception performance.

6. Conclusions

In this paper, we summarize the limitations of existing honeynet architectures: (1) attackers can only access one honeypot at one time, (2) architectures have a low degree of simulation, (3) architectures are unable to capture deep-level attacks, and (4) there is a lack of test metrics. We propose a novel honeynet architecture called HoneyFactory. We describe the design and architecture of five modules in HoneyFactory and provide feasible technology selection, including environment learning, honeynet deception model, honeynet generation, honeynet data collection, and honeynet information utilization. We propose a honeynet deception model to evaluate the current deception stage based on the Gaussian Hidden Markov theory. We implement the prototype of HoneyFactory. Experimental results demonstrate that HoneyFactory performs better in communication latency and connections per second. We also propose novel test metrics for honeynet to measure

deception effectiveness. The experiment shows that HoneyFactory can simulate protected networks and capture deep-level attacks.

Author Contributions: Conceptualization, T.Y.; Methodology, T.Y.; Software, T.Y.; Validation, C.Z.; Investigation, Y.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under Grant 62233003, the National Key R&D Program of China No. 2020YFB1708602.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lackner, P. How to Mock a Bear: Honeypot, Honeynet, Honeywall & Honeytoken: A Survey. In Proceedings of the ICEIS, Online, 26–28 April 2021.
2. Krishnaveni, S.; Prabakaran, S.; Sivamohan, S. A survey on honeypot and honeynet systems for intrusion detection in cloud environment. *J. Comput. Theor. Nanosci.* **2018**, *15*, 2949–2953. [CrossRef]
3. Oza, A.D.; Kumar, G.N.; Khorajiya, M. Survey of snaring cyber attacks on IoT devices with honeypots and honeynets. In Proceedings of the 2018 3rd International Conference for Convergence in Technology (I2CT), Pune, India, 6–8 April 2018.
4. Spitzner, L. The honeynet project: Trapping the hackers. *IEEE Secur. Priv.* **2003**, *1*, 15–23. [CrossRef]
5. Abbasi, F.H.; Harris, R.J. Experiences with a generation iii virtual honeynet. In Proceedings of the 2009 Australasian Telecommunication Networks and Applications Conference (ATNAC), Canberra, ACT, Australia, 10–12 November 2009.
6. Jain, P.; Sardana, A. Defending against internet worms using honeyfarm. In Proceedings of the CUBE International Information Technology Conference, Pune, India, 3–6 September 2012.
7. Hybrid Honeypot Framework. Available online: http://honeybrid.sourceforge.net (accessed on 1 September 2023).
8. Jiang, X.; Xu, D.; Wang, Y.M. Collapsar: A VM-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention. *J. Parallel Distrib. Comput.* **2006**, *66*, 1165–1180. [CrossRef]
9. Han, W.; Zhao, Z.; Doupe, A.; Ahn, G. Honeymix: Toward sdn-based intelligent honeynet. In Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, New Orleans, LA, USA, 11 March 2016.
10. Kyung, S.; Han, W.; Tiwari, N.; Dixit, V.; Srinivas, L.; Zhao, Z.; Doupe, A.; Ahn, G. HoneyProxy: Design and implementation of next-generation honeynet via SDN. In Proceedings of the 2017 IEEE Conference on Communications and Network Security (CNS), Las Vegas, NV, USA, 9–11 October 2017.
11. Silva, D.V.; Rafael, G.D.R. A review of the current state of Honeynet architectures and tools. *Int. J. Secur. Netw.* **2017**, *12*, 255–272. [CrossRef]
12. Franco, J.; Aris, A.; Canberk, B.; Uluagac, A. A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2351–2383. [CrossRef]
13. Dalamagkas, C.; Sarigiannidis, P.; Ioannidis, D.; Iturbe, E.; Nikolis, O.; Ramos, F.; Rios, E.; Sarigiannidis, A.; Tzovaras, D. A survey on honeypots, honeynets and their applications on smart grid. In Proceedings of the 2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, 24–28 June 2019.
14. Tan, L.; Yu, K.; Ming, F. Secure and resilient artificial intelligence of things: A HoneyNet approach for threat detection and situational awareness. *IEEE Consum. Electron. Mag.* **2021**, *11*, 69–78. [CrossRef]
15. Zhang, W.; Zhang, B.; Zhou, Y.; He, H.; Ding, Z. An IoT honeynet based on multiport honeypots for capturing IoT attacks. *IEEE Internet Things J.* **2019**, *7*, 3991–3999. [CrossRef]
16. Kokolakis, G.; Ntousakis, G.; Karatsoris, I. HoneyChart: Automated Honeypot Management over Kubernetes. In *European Symposium on Research in Computer Security*; Springer: Cham, Switzerland, 2022.
17. Hecker, C.; Hay, B. Automated honeynet deployment for dynamic network environment. In Proceedings of the 2013 46th Hawaii International Conference on System Sciences, Wailea, HI, USA, 7–10 January 2013.
18. Meng, X.; Zhao, Z.; Li, R.; Zhang, H. An intelligent honeynet architecture based on software defined security. In Proceedings of the 2017 9th International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, China, 11–13 October 2017.
19. Wang, Z.; Li, G.; Chi, Y.; Zhang, J.; Liu, Q.; Yang, T.; Zhou, W. Honeynet construction based on intrusion detection. In Proceedings of the 3rd International Conference on Computer Science and Application Engineering, Sanya, China, 22–24 October 2019.
20. Krueger, T.; Krämer, N.; Rieck, K. ASAP: Automatic semantics-aware analysis of network payloads. In *Privacy and Security Issues in Data Mining and Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2010.
21. Herzberg, A.; Shulman, H.; Ullrich, J.; Weippl, E. Cloudoscopy: Services discovery and topology mapping. In Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop, Berlin, Germany, 8 November 2013.
22. Gupta, C. *HoneyKube: Designing a Honeypot Using Microservices-Based Architecture*; University of Twente: Enschede, The Netherlands, 2021.

23. Yin, Y.; Shao, Y.; Wang, X.F.; Su, Q. A flexible cyber security experimentation platform architecture based on docker. In Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofia, Bulgaria, 22–26 July 2019.
24. Bercovitch, M.; Renford, M.; Hasson, L.; Shabtai, A.; Rokach, L.; Elovici, Y. HoneyGen: An automated honeytokens generator. In Proceedings of the 2011 IEEE International Conference on Intelligence and Security Informatics, Beijing, China, 10–12 July 2011.
25. Chaddad, L.; Chehab, A.; Elhajj, I.H.; Kayssi, A. Optimal packet camouflage against traffic analysis. *ACM Trans. Priv. Secur. (TOPS)* **2021**, *24*, 1–23. [CrossRef]
26. Fan, W.; Du, Z.; Smith-Creasey, M.; Fernandez, D. Honeydoc: An efficient honeypot architecture enabling all-round design. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 683–697. [CrossRef]
27. Casalicchio, E. Container orchestration: A survey. *Syst. Model. Methodol. Tools* **2019**, *1*, 221–235.
28. A Cloud-Native Honeynet Automation and Orchestration Framework Hybrid Honeypot Framework. Available online: https://osf.io/xkqzr/download (accessed on 1 September 2023).
29. Glastopf. Available online: https://github.com/mushorg/glastopf (accessed on 1 September 2023).
30. Kippo. Available online: https://github.com/desaster/kippo (accessed on 1 September 2023).
31. Xuan, G.; Zhang, W.; Chai, P. EM algorithms of Gaussian mixture model and hidden Markov model. In Proceedings of the 2001 International Conference on Image Processing, Thessaloniki, Greece, 7–10 October 2001.
32. Almohannadi, H.; Awan, I.; Hamar, A.J.; Cullen, A.; Disso, P.J.; Armitage, L. Cyber threat intelligence from honeypot data using elasticsearch. In Proceedings of the 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), Krakow, Poland, 16–18 May 2018.
33. Moore, C.; Nemrat, A.A. An analysis of honeypot programs and the attack data collected. In Proceedings of the Global Security, Safety and Sustainability: Tomorrow's Challenges of Cyber Security: 10th International Conference, London, UK, 15–17 September 2015.
34. Hwang, R.H.; Peng, M.C.; Nguyen, V.L.; Chang, Y.L. An LSTM-based deep learning approach for classifying malicious traffic at the packet level. *Appl. Sci.* **2019**, *9*, 3414. [CrossRef]
35. Agarap, A.F.M. A neural network architecture combining gated recurrent unit (GRU) and support vector machine (SVM) for intrusion detection in network traffic data. In Proceedings of the 2018 10th International Conference on Machine Learning and Computing, Macau, China, 26–28 February 2018.
36. Vinayakumar, R.; Soman, K.P.; Poornachandran, P. Applying convolutional neural network for network intrusion detection. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13–16 September 2017.
37. Wang, M.; Lu, Y.; Qin, J. A dynamic MLP-based DDoS attack detection method using feature selection and feedback. *Comput. Secur.* **2020**, *88*, 101645. [CrossRef]
38. Qassrawi, M.T.; Hongli, Z. Deception methodology in virtual honeypots. In Proceedings of the 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing, Wuhan, China, 24–25 April 2010.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.