Building Dynamic Honeypots

Selby Dasent

***Abstract*—Traditional honeypots have long served as valuable tools for monitoring and studying attacks on both research and production networks. However, their inherent static nature falls short in accurately reflecting the dynamic landscape of today's networks. Modern networks comprise of changing hosts and devices that frequently connect and disconnect, making honeypot's single-host, open-port setup with its static address conspicuous amidst the network activity. To address this static nature, dynamic honeypots automatically deploy and configure hosts, enabling them to blend harmoniously with the network's existing environment. This paper introduces a dynamic honeypot design that automates the configuration and deployment of Cowrie honeypots within a network environment. Importantly, this system boasts the ability to be deployed without prior knowledge of the network's topology or associated hosts.**

***Index Terms*—Honeypot, Dynamic Honeypots, Cowrie, Fingerprinting**

I. Introduction

With the constant evolution of internet-enabled devices, the world has become more interconnected than ever. The recent pandemic, in particular, has prompted a significant shift toward hybrid work structures, enabling employees to work remotely [9]. However, this increased connectivity has also given rise to a surge in cybercrime [8]. Advanced attackers are now capable of bypassing conventional security measures like firewalls. In response to this, network deception-based methods are being developed to detect intruders within a network, and one of the key strategies in this realm is the deployment of honeypots.

Honeypots represent a vital tool for countering emerging threats and provide an opportunity to study attacker behavior and tools. A more comprehensive understanding of attacker behavior and the tools they employ can empower defenders to better comprehend their own network vulnerabilities. A honeypot is essentially a system intentionally placed within a network with the purpose of being attacked and compromised. Existing honeypot system designs can generally be categorized into two key approaches: improving intelligence gathering and enhancing trapping capabilities. The former aims to make honeypot deployment more adaptable, while the latter concentrates on creating high-fidelity scenarios that divert attackers away from other network devices.

Deployment strategies for honeypots encompass both static and dynamic methods. Traditionally, honeypots are manually configured and deployed on physical devices, and defenders often prefer static approaches due to their accessibility. However, these strategies face scalability issues, making it challenging to mirror the ever-changing dynamics of network environments. The advent of lightweight virtualization technology has facilitated the deployment of multiple honeypots on a single device, propelling the development of dynamic honeypot deployment as a forward-looking approach to network deception.

This project was supervised by Supervisor Masood Mansori

II. The Problem

The effectiveness of honeypots hinges on their ability to emulate real systems, thereby luring potential attackers. To achieve this, honeypots should closely resemble genuine computer systems or services. However, adversaries are becoming increasingly adept at detecting honeypots due to their static characteristics and inherent limitations when compared to live systems. On a network with regular inter-host activity, identifying a low-interaction honeypot can be as simple as observing its lack of outbound network traffic.

A substantial challenge for honeypots lies in their deployment and integration into production networks. Modern networks are dynamic ecosystems with a multitude of server devices, hosts, and diverse operating systems. This amalgamation of systems and services undergoes periodic changes as new systems and users join, while others depart. Devices with prolonged up-time, such as honeypots, may stand out and be recognized as potential honeypots.

Furthermore, it's important to note that while honeypots capture primarily malicious traffic, not all malicious activities on a network necessarily target honeypots. Despite being enticing targets, attackers may refrain from interacting with honeypots, as they can be wary of their online presence, duration, or conspicuousness.

The inherent static nature of honeypots renders them susceptible to exposure through network mapping, traffic monitoring, and operating system/port fingerprinting. This visibility diminishes their efficacy in capturing meaningful data. Additionally, configuring, deploying, and managing honeypots can be a time-consuming process for users, especially when collecting data from individual systems.

This project aims to address the challenges of honeypot deployment and their inherent static nature by developing a management system capable of deploying a set of dynamically initiated honeypots into a production environment.

A. Proposed Solution

The primary aim of this project is to develop a system capable of dynamically deploying honeypot servers within a network environment while simultaneously diminishing the detectability of these honeypots by potential attackers. The proposed system is designed to offer a comprehensive solution. It will begin by generating a network visualization, providing a clear and detailed representation of the current network layout. Concurrently, it will create and configure a series of honeypots, tailored to resemble the existing hosts on the network. The

ultimate objective is to enable these honeypots to seamlessly blend into the environment, appearing indistinguishable from genuine network assets.

Once these honeypots are deployed, they will actively capture any interactions initiated by users. This captured data will be collated and presented in a format that is easily accessible for review by analysts. This streamlined process ensures that valuable insights into network threats and attacker behavior can be readily extracted. Beyond the initial deployment, the system will maintain the flexibility to take honeypots offline and subsequently redeploy them. This feature allows the simulation of network changes, effectively mimicking the addition and removal of devices within the network.

The core deliverable of this project is a system with the ability to scan the network and seamlessly deploy honeypots. This scanning component is designed to provide users with a visual representation of the network's current state. Furthermore, it will generate detailed honeypot configurations, expertly customized to ensure the honeypots' seamless integration into the specific deployment environment. The project's completion will also include all-encompassing documentation, offering clear instructions on software setup and usage, empowering users to maximize the system's potential. Moreover, the system has been intentionally designed for adaptability and further development, offering opportunities for scalability and the continuous enhancement of configuration generation

III. Related Work

This section delves into the contemporary concepts for comprehending the project's objectives. Additionally, it sheds light on existing solutions in the realm of dynamic honeypots, offering analysis of their strengths and limitations.

A. Honeypots

At a fundamental level, honeypots are servers or services strategically positioned within a network. Their purpose is to be probed, attacked, or even compromised by potential intruders [7]. Honeypots are designed to mimic real systems, complete with vulnerabilities and configurations, luring attackers into interacting with them.

These honeypots typically fall into one of two categories: research honeypots and production honeypots. Although both share the common goal of collecting valuable data, their objectives differ significantly.

Research honeypots focus on acquiring insights into attacker tactics, motivations, and the tools they employ. While they may not directly contribute to an organization's security, research honeypots are invaluable for building threat intelligence and an understanding of a businesses threat landscape. However, they are often high-interaction and complex to deploy and maintain due to the wealth of data they can capture.

Production honeypots, on the other hand, reside within a business's network and primarily serve for monitoring and threat detection. These honeypots tend to be low-interaction systems, making them easier to manage, but they provide fewer details. In the context of this project, research-type honeypots are chosen due to their extensive data capture capabilities.

Despite significant advancements in honeypot technology since their inception in the 1990s [1], there remains a substantial emphasis on manual configuration and deployment. Honeypots rely on precise technical configurations, as even minor errors can compromise their ability to capture or deceive attackers. Additionally, an advanced intruder may swiftly detect the presence of honeypots or exploit them for further attacks.

B. Dynamic Honeypots

A dynamic honeypot is designed to streamline the traditionally manual processes involved in configuring, deploying, and logging a honeypot system. In this project, the system's key tasks include scanning a network to gather necessary data and generating configurations tailored for the honeypots that will be deployed. Once deployed, these honeypots are expected to actively monitor the network for any changes and autonomously adapt to these changes as needed. This automation not only enhances the efficiency of honeypot deployment but also ensures their ability to stay relevant and responsive in dynamic network environments.

In the realm of dynamic honeypots, two distinct approaches have been explored. The first takes on a "bait and switch" methodology, where the system swiftly detects an intruder and redirects them to a completely fabricated network that mirrors the primary network. Implementing this approach requires not only the rapid detection of intruders but also the immediate deployment of the faux environment and moving the intruder into it without their awareness. However, scalability poses a challenge—the efficiency of the system can be directly impacted by the size of the network being emulated.

A notable design employing the "bait and switch" concept was presented by I. Beres [2]. This design extensively detailed the use of the cloud to generate mirrored honeypot servers on demand. It relied on Amazon's Cloudwatch to alert the system to an intruder's presence. Subsequently, new instances were spawned as honeypots, and malicious traffic was redirected to them. This paper demonstrates the effectiveness of the "bait and switch" honeypot approach in cloud environments. These environments can swiftly generate new services, demanding additional resources without affecting users. However, the honeypots used in this method were low-interaction and did not capture attack analytics, primarily due to scalability concerns. A large network with high-interaction honeypots could consume considerable resources and potentially extend the environment generation process. The unexpected delay experienced by an intruder when being shifted to the simulated environment might alert them to the honeypot's presence.

Y. Gao and colleagues [4] proposed an approach similar to the "bait and switch" strategy. They utilized attack path prediction to strategically deploy honeypots within a network. The primary goal of this method was to anticipate an intruder's likely path and position the honeypot directly in their way. This approach offered the advantage of reducing the resources required to capture an intruder while also minimizing the risk

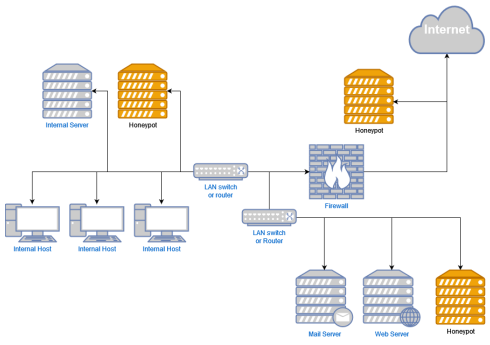

Fig. 1. Traditional static deployment of honeypots

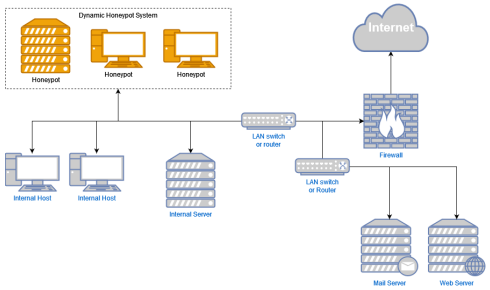

Fig. 2. Dynamic deployment of honeypots

of detection. Although effective, it is worth noting that there is still a possibility that the intruder might choose an alternate path, thereby evading the honeypot entirely. This approach highlights the ongoing cat-and-mouse game between defenders and attackers in the realm of cybersecurity.

The second category of dynamic honeypots, which serves as the central focus of this project, offers seamless integration into the existing network environment. This dynamic honeypot approach emulates the behavior of adding and removing itself periodically, effectively mirroring the typical actions of new hosts joining a network. In stark contrast to traditional honeypots, which are deployed into a network and remain static (Figure 1), dynamic honeypots demonstrate unique adaptability. In the face of changing network topologies, traditional honeypots remain unaltered, while dynamic honeypots maneuver within the network (Figure 2).

In this methodology, intruders are still required to attempt accessing the deployed honeypots. It's important to note that this form of dynamic honeypots doesn't primarily aim to safeguard the network, as observed in "bait and switch" honeypots. Instead, its primary objective revolves around data collection and alerting defenders to an intruder's activities.

C. Network Scanning

To perform comprehensive network scanning, two primary methods are typically employed: passive scanning and active scanning. Passive scanning entails deploying a scanner on a router to intercept outgoing traffic. While this approach can capture all the necessary information to map the network, it relies on the active transmission of packets by network hosts. In contrast, active scanning involves the use of tools like nmap to send packets to every available host on the network to collect data. Active scanning is a faster method that can identify machines, even those unlikely to transmit traffic.

A significant study conducted by M. Mansoori and colleagues [3] relied on passive scanning as the primary method for network mapping. Their hypothesis was that passive scanning could match the accuracy of active scanning. While the passive method offered certain advantages such as reduced network traffic, it took longer to detect hosts within the network. Despite the effectiveness of passive scanning, this project opts for the active scanning approach, primarily due to the need to generate large-scale network maps efficiently.

IV. Design

The scope of this project encompasses the development of a system capable of managing multiple honeypots and configuring them to adapt to the dynamic nature of a network environment. This section explores the design decisions that will guide the development of this project.

A. Project Considerations

The core implementation, as highlighted in the proposed solution, can be divided into four key considerations. The first involves selecting the programming language to be used in developing and deploying the entire system. The second consideration pertains to the type and design of the honeypot that will be deployed. The third aspect focuses on the method and environment in which the honeypots will be deployed. Finally, the fourth consideration revolves around the network scanner to be used for mapping the network and developing the necessary configuration details. These aspects are crucial for the successful execution of the project.

B. Design Considerations

The purpose of this section is to explore the design decisions made regarding the development language, scanning and the deployment of the honeypots.

1) Development Language: Considering the programming language for system development was the first step in this project, particularly for creating an environment capable of deploying Cowrie honeypots within a network. This selection involved evaluating various aspects such as scalability, budget, complexity, time constraints, and available resources.

Two languages were evaluated for this project: Java and Python. While Java is recognised for its speed, Python was chosen due to its abundance of libraries tailored for managing Docker containers and network scanning. Furthermore, since the Cowrie honeypot itself is built using Python, choosing Python would facilitate future improvements and integration into the system. Another advantage of Python is its versatility in running on a wide array of base operating systems with ease, reducing the system's reliance on a particular operating environment.

TABLE I
COMPARISON OF HONEYPOTS

	Cowrie	Honeyd	Kippo
Interaction Level	Medium	Low	Medium
Docker Images	✓	✗	✓
Configurable	✓	✓	✓
Frequently Updated	✓	✗	✗

2) Honeypot: Another design aspect impacting the project was the selection of the Honeypot system. A plethora of open-source honeypots are available, each designed to fulfill distinct objectives, as documented in [11].

For this project, we evaluated several key considerations, not ranked in any particular order: the level of interaction, ease of deployment and configuration, and the frequency of updates. We conducted a comparative analysis of three different honeypots, detailed in I, in order to determine the most suitable choice.

The ultimate selection was Cowrie, a medium interaction honeypot known for its user-friendliness in deployment and configuration. Additionally, the involvement of the Owhiti cybersecurity group at Victoria University of Wellington, who have contributed to and employed Cowrie in their research, further influenced this decision.

3) Deployment Environment: To achieve the rapid deployment of honeypots, we must address the challenge of running multiple honeypots within a single device. When Cowrie honeypots are deployed individually, they assume the host's IP address and bind themselves to a port. While this works well for single honeypots, it becomes problematic when you aim to run multiple simultaneously.

Initially, I contemplated having the system interact with a hypervisor like Virtualbox to initiate and deploy machines with Cowrie running. However, this approach raised concerns regarding the substantial resource requirements it would entail. A more efficient solution emerged: transitioning to a containerized system, such as Docker and Kubernetes. During our selection process for the honeypot to run within this system, we considered its compatibility with such an environment. Cowrie, our chosen honeypot, offers readily available Docker images that can be swiftly downloaded and executed. Moreover, these images can be dynamically customized at runtime, enabling a variety of Cowrie configurations to be established.

4) Network Scanner: The final critical design decision revolved around how to scan the network and identify the connected devices and their respective operating systems. We considered two primary approaches: leveraging the capabilities of an existing tool, such as Nmap, or developing our own network scanner. The most challenging aspect of creating our own scanner was determining the devices' operating systems, which can be achieved through methods like identifying operating system signatures or analyzing response packet time-to-live values.

For the successful configuration of the honeypots in this project, it was imperative to have precise knowledge of the operating systems. With this requirement in mind, we opted to utilize existing tools. An analysis conducted by J. M. Pittman [10] revealed no discernible difference in effectiveness or accuracy between Nmap, Zmap, and Massscan. However, we chose Nmap as the primary scanner for this project due to the availability of readily accessible Python libraries tailored for Nmap integration.

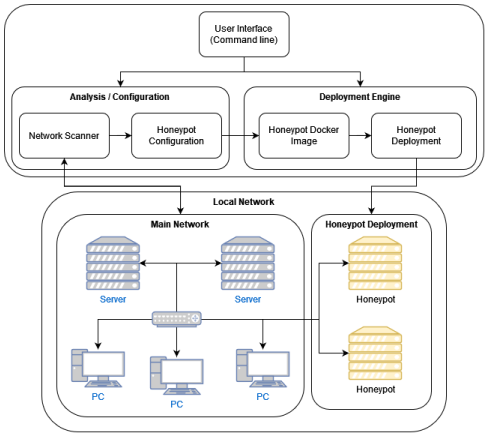

Fig. 3. Proposed system modules

C. System Architecture

The system architecture outlines how the proposed solution is dissected into smaller, interconnected segments. As depicted in Figure 3, the design comprises two core components: the analysis/configuration engine and the deployment engine. These core components can be further subdivided into distinct functions, including the network scanner, honeypot configuration, Docker image management, and the deployment of honeypots.

Additional facets of the system encompass the local network, including its topology, and the user interface. The user interface has been intentionally tailored for command-line interaction, enabling seamless remote system access and operation. While the local network's design may not be within our direct control, it remains a crucial consideration during the solution's implementation phase.

V. IMPLEMENTATION

In the following sections, we delve into the project's implementation process. This project unfolded in three distinct stages: firstly, the development of the network scanner, followed by the deployment of honeypots in the second stage. The third and final stage involved the seamless integration of these components to accurately configure the deployed honeypots.

A. Scanner

In the initial development phase, our primary focus was on the network scanner, which played a crucial role in scanning and identifying hosts along with their respective operating systems. To accomplish this, we designed a solution that harnessed the capabilities of Nmap. We initially considered two libraries, namely nmap3 and nmapthon. Our initial design incorporated nmapthon, which allowed us to customize the scan types and store results as a dedicated object within the

system. However, we encountered challenges related to the completeness of scan results, leading us to explore alternative options.

Ultimately, we decided to work with a different library that met the core requirements of our scans and delivered results in a manageable JSON format. This format provided us with more control over data handling within the system. Consequently, the system was configured to save scan results in a JSON-formatted file, enabling access to historical network data and facilitating the identification of network changes over time.

Ensuring that the scanner was targeting the correct network required user input of a network range or, alternatively, the system needed to determine its connected network automatically. If the user did not specify a network range, the system would attempt to retrieve its own IP address from its network adapter, alongside the netmask of the network. This process occurred in two stages before consolidating the results. In the first stage, the system identified the device's network adapter and surveyed the networks to which it was connected. It excluded the local host (127.0.0.1) and extracted the device's outward-facing IP address. Subsequently, it gathered the netmask for this IP. This involved obtaining the binary response from a socket connection request and converting it into an integer representation of the netmask. Armed with the device's IP address and netmask, the system generated a subnet for scanning.

B. Deployment Engine

For the deployment of Cowrie honeypots, we chose to employ Docker containers. Cowrie conveniently offers its own Docker image hosted on Docker's platform, enabling us to effortlessly pull the latest image and execute it within a Docker container. To interact with Docker seamlessly from our system, we harnessed the Docker SDK for Python, which empowered us to programmatically access and manage Docker's comprehensive set of functionalities.

Our initial objective was to download and initiate the Docker image. To achieve this, we created a Docker environment object in Python and instructed it to fetch the most recent Cowrie image from the Docker library. When combined with the Docker image's run function, this process enabled us to swiftly deploy a Cowrie honeypot within the Docker network. These active containers were then cataloged in a list within the system, affording us the capability to monitor, update, or shut down each active honeypot as needed.

Since Docker containers operate independently of the system, it was imperative to implement a proper shutdown sequence to halt the running containers. Failure to do so could result in a buildup of unused honeypots. This was accomplished by utilizing the stop function on any active containers.

Once the containers were deployed, we needed to place them on the appropriate network to be effective. When Docker images are deployed, they automatically become part of a network created by the Docker system. To expose Docker containers to the internet, there are three primary methods.

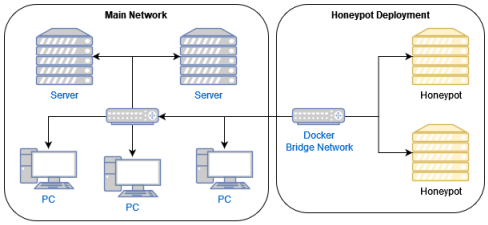

Fig. 4. Bridged network adapter

The most commonly accepted approach is to create a firewall rule that maps a container's port to a port on the Docker host machine. However, this approach would not suffice for our system, as the honeypots required their own IP addresses. To address this, we utilized a Docker network bridge, which allowed us to configure a network adapter that the Docker images could connect to, providing them with their unique IP addresses.

To configure the network adapter, we need input similar to what the scanner requires. If the user knows the subnet address, they can input it. Otherwise, the system will use the network to which the device is connected, a piece of information obtained through the scanning module. Once the subnet is set, we can proceed to create the network adapter using Docker. This action yields a network adapter object, allowing us to iterate through a list of containers and connect them to the network as seen in Figure 4

Similar to managing the containers, it's crucial to establish an appropriate shutdown sequence for the network adapter. Leaving networks operational can potentially interfere with the functioning of the running program. We accomplished this by first removing all containers connected to the adapter and subsequently removing the adapter from Docker.

C. Cowrie Configuration

The next step in the process involved configuring the Cowrie honeypots. To achieve this, we needed to transform the data generated from the network scan into a suitable configuration file for Cowrie. The scan produced a comprehensive JSON object, but for the project's specific needs, we extracted essential information. We retained the IP addresses, open ports, and hostnames of the devices. Additionally, we selected the most accurate operating system information.

Nmap's OS scan produces a list of potential operating systems for a device, derived from two sources. The first is a unique signature determined by Nmap using various factors such as TCP ISN sampling, IP ID sampling, and initial window size checks. If Nmap cannot establish an operating system with this signature, it resorts to TCP sequence predictability classification, measuring the difficulty of establishing a forced TCP connection against the host. These techniques have proven to be highly accurate, as substantiated by studies [3] [10], making them a reliable method for configuring the honeypots.

Once the scan results have been parsed, the system generates a list and incorporates it into the Cowrie configuration file, which is then injected into the Docker image during the startup process. When Cowrie determines the operating system to

emulate, it randomly selects from the list of available systems within the configuration file. With this in mind, we construct a list of all the operating systems, including duplicates, to create a weighted list for Cowrie to choose from.

D. User Interface

User interaction with the system is facilitated through a command-line interface (CLI) that was purpose-built for the project. This CLI offers a range of commands enabling users to interact with the system, including tasks like starting and stopping honeypots, initiating network scans, and accessing the results of these scans. Although the consideration of developing a Graphical User Interface (GUI) was entertained initially, it was deemed incompatible with the system's intended use and requirements for remote access. Consequently, the decision was made to revert to a command-line interface, which aligns better with the project's requirements.

The user interface was designed with simplicity and ease of use in mind. Users can issue commands to perform various tasks, and the system provides feedback and status updates to keep users informed about ongoing processes. The CLI is structured to accommodate both novice and experienced users, with detailed help documentation available to guide users through the commands and their usage.

E. Integration and Testing

The integration of the various components of the system was a critical phase in ensuring the functionality and reliability of the entire solution. Extensive testing was conducted at each stage of the development to identify and resolve any issues, bugs, or inconsistencies.

Integration testing involved coordinating the interaction between the network scanner, Docker-based honeypot deployment, and network simulation. The network scanner would provide data to configure honeypots, and the network simulation component would continually monitor the network to maintain the dynamic nature of the honeypot deployment.

User interface testing ensured that the CLI provided a smooth and user-friendly experience, with a clear and intuitive set of commands. Extensive documentation was created to guide users in using the system effectively.

VI. Evaluation

A survey was conducted to assess the usability of the system and gather user satisfaction feedback regarding the deployment of honeypots on a network. The survey involved participants associated with the Owhiti Security Group and was administered during two live demonstrations of the system. The survey questions primarily focused on evaluating the system's ease of use and its effectiveness in providing relevant information.

The following section provides a summary of the questions posed in the survey and outlines the rationale behind each question:

1) **Were the instructions provided for using the command line interface clear and easy to understand?** It helps determine if the system's instructions were sufficient, comprehensible, and user-friendly. Moreover, it indicates whether additional information or clarifications may be required to improve the overall usability of the system.
2) **Did you encounter any difficulties in navigating or using the command line interface?** This question is vital for gaining insights into the user's overall experience with the system, especially concerning its layout, functionality, and ease of navigation. Understanding any difficulties or challenges users faced in navigating or using the command-line interface helps in identifying specific areas of improvement in the system's design and layout.
3) **Were you provided with sufficient information about the network scanning and honeypot deployment process?** This question provides insights into the adequacy of information within the system, which is vital for user understanding and successful operation. If users indicate that they were not provided with sufficient information, it highlights potential areas for improvement.
4) **Have you ever deployed honeypots using a command line interface before using this system?** This question helps in gauging whether the user's prior experience influences their assessment of the new system's ease of use. Understanding the user's level of expertise and familiarity with the domain is essential for obtaining a comprehensive and context-specific evaluation of the system.
5) **If you have prior experience, how does this system compare to your previous experiences with deploying honeypots?** This question is essential as it encourages users to provide insights based on their past experiences, allowing for a contextual comparison between the new system and their previous honeypot deployment tools. Users with prior experience offer valuable feedback on the system's strengths and weaknesses in relation to existing solutions, aiding in the assessment of its usability.
6) **What aspects of the system do you think could be improved?** By directly asking users to identify aspects of the system that could be enhanced, it provides an open platform for users to express their thoughts and suggestions for refinement.

During the first demonstration, feedback was gathered from four participants, with overall positive impressions regarding the system's usability. However, a valuable suggestion emerged: the availability of commands should print back to the command line more frequently. This constructive feedback was incorporated into the system before the second demonstration. Subsequently, during the second demonstration, participants expressed satisfaction with the interface, reporting that they could efficiently navigate the system.

In response to questions 4 and 5, two out of the four participants had prior experience in honeypot deployment. Their feedback regarding the ease of honeypot deployment was positive, with suggestions revolving around providing more detailed information about the deployed honeypots and offering users additional options for deployment, such as

specifying the network range for honeypot connections. There was also a suggestion to have the honeypots deploy to a more pseudo-random set of addresses within the network to enhance their deceptive nature, although this suggestion could not be implemented before the project's completion.

VII. Conclusion and Future Work

In conclusion, this project aimed to address the challenges associated with the deployment of honeypots and their inherent static nature. It sought to create a management system capable of deploying dynamically initiated honeypots into a production environment, ensuring that they integrate with the existing network assets. The project involved multiple key components, including a network scanner, honeypot deployment engine, and a user-friendly command-line interface.

Throughout the implementation process, the project underwent several phases, including the development of the network scanner, the deployment of Cowrie honeypots using Docker containers, and the configuration of honeypots based on scan results. A user survey was conducted to assess the usability and effectiveness of the system, with valuable feedback gathered from participants associated with the Owhiti Security Group.

The survey results indicated that the system provided clear instructions and was relatively easy to navigate. Participants with prior honeypot deployment experience found the system to be user-friendly. The feedback also led to improvements, including more frequent command-line feedback and considerations for enhancing the randomness of honeypot deployment.

In summary, this project addressed the challenges of honeypot deployment and offered a dynamic solution that integrates honeypots into a network environment. While there is room for further improvement and development, this system represents a step towards enhancing the effectiveness of honeypots in network security and threat detection.

A. Future Work

While the outcome of this project was successful, there are several ways to extend this system to increase its value. Four key areas for improvement have been proposed:

1) Enhanced Honeypot Variety: Expanding the range of supported honeypot types, each emulating different services and systems, can improve the system's ability to represent various operating systems. This diversity enhances the system's effectiveness in detecting different types of attacks and attackers, making it a more versatile tool in network security.

2) Real-time Monitoring and Alerts: Implementing real-time monitoring and alerting mechanisms allows security analysts to receive immediate notifications of suspicious activity. Customisable alert triggers provide flexibility in configuring the system to respond to specific threats promptly.

3) Machine Learning Integration: Incorporating machine learning models can enhance the system's ability to identify operating systems and network configurations. Machine learning aids in classifying and analyzing network traffic, allowing the system to adapt honeypots based on evolving attack patterns.

4) Comprehensive Reporting: Implementing a system that gathers and presents logs from deployed honeypots in a user-friendly manner can enhance the system's usability as a research honeypot. This comprehensive reporting feature provides users with valuable information for studying attacks and analyzing security threats.

By focusing on these development areas, the dynamic honeypot management system can become more versatile, responsive, and effective in detecting cyber threats. It will adapt to changing attack techniques, offer real-time insights, and provide a wider range of emulated services, ultimately strengthening network security and threat detection capabilities.

VIII. References Section

References

[1] S. Clifford *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*. New York, NY, USA, Doubleday, 1989.

[2] Beres, Ivan, Hurley-Smith, Darren. (2022). *Dynamic honeypot deployment in the cloud.* 10.13140/RG.2.2.18384.58883.

[3] MOHAMMADZADEH, H., MANSOORI, M., AND WELCH, I. *Evaluation of fingerprinting techniques and a windows-based dynamic honeypot.* [Online]. Available: https://dl.acm.org/doi/pdf/10.5555/2525483.2525490

[4] D. Fraunholz, M. Zimmermann, H. Schotten. (2017). *An adaptive honeypot configuration, deployment and maintenance strategy.* 53-57. 10.23919/ICACT.2017.7890056.

[5] Y. Gao, G. Zhang, C. Xing, *A Multiphase Dynamic Deployment Mechanism of Virtualized Honeypots Based on Intelligent Attack Path Prediction*. Security and Communication Networks. [Online]. Available: https://doi.org/10.1155/2021/6378218

[6] ML. Kiah, W. Zakaria. (2013). *A review of dynamic and intelligent honeypots*. ScienceAsia. 39s. 1 - 5. 10.2306/scienceasia1513-1874.2013.39S.001.

[7] Kaspersky. *What is a honeypot*. [Online]. Available: https://www.kaspersky.com/resource-center/threats/what-is-a-honeypot

[8] CERT NZ. *Cyber Security Insights* [Online]. Available: https://www.cert.govt.nz/about/quarterly-report/

[9] M. Mutebi, A. Hobbs. *The impact of remote and hybrid working on workers and organisations*. UK Parliament, London, England, October 2022. [Online]. Available: https://post.parliament.uk/research-briefings/post-pb-0049/

[10] J. M. Pittman. *A Comparative Analysis of Port Scanning Tool Efficacy*. arXiv. Cornell University. [Online]. Available: https://doi.org/10.48550/arXiv.2303.11282

[11] J. Nazario. *Awesome Honeypots* [Online]. Available: https://github.com/paralax/awesome-honeypots