DEVELOPMENT OF A LIGHTWEIGHT HONEYPOT SYSTEM FOR DETECTING UNAUTHORISED ACCESS ON A LOCAL NETWORK

BY

OJEH ELLIS CHUKWUKA

PSC2208011

DEPARTMENT OF COMPUTER SCIENCE,

FACULTY OF COMPUTING,

UNIVERSITY OF BENIN, BENIN CITY,

EDO STATE, NIGERIA.

NOVEMBER 2025

DEVELOPMENT OF A LIGHTWEIGHT HONEYPOT SYSTEM FOR DETECTING UNAUTHORISED ACCESS ON A LOCAL NETWORK

BY

OJEH ELLIS CHUKWUKA

PSC2208011

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,

FACULTY OF COMPUTING,

UNIVERSITY OF BENIN, BENIN CITY,

EDO STATE, NIGERIA.

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR OF SCIENCE (B.Sc.) DEGREE IN COMPUTER SCIENCE

NOVEMBER 2025

CERTIFICATION

This is to certify that this project was titled “Development of a Lightweight Honeypot System for Detecting Unauthorized Access on a Local Network."” was carried out by OJEH ELLIS CHUKWUKA with Matriculation number: PSC207963 and submitted to the Department of Computer Science, Faculty of Computing, University of Benin, Benin City, under the supervision of PROF. F. AMADIN.

_____ _____

ARUYA OMUAH DEBBIE Date

Student

_____ _____

PROF. F. AMADIN Date

Supervisor

_____ _____

DR. MAXWELL OSAGIE Date

Project Coordinator

_____ _____

DR. ROSEMARY USIOBAIFO Date

Head of Department

APPROVAL

This project titled “Development of a Lightweight Honeypot System for Detecting Unauthorized Access on a Local Network.” by OJEH ELLIS CHUKWUKA with Matriculation number: PSC2207963 has been approved as meeting the requirements for the award of Bachelor of Science Degree in the Department of Computer Science, Faculty of Computing, University of Benin, Benin city.

_____	_____
PROF. F. AMADIN	Date
Supervisor	
_____	_____
DR. MAXWELL OSAGIE	Date
Project Coordinator	
_____	_____
DR. ROSEMARY USIOBAIFO	Date
Head of Department	
_____	_____
EXTERNAL EXAMINER	Date

DEDICATION

This project is dedicated to Almighty God for His infinite wisdom, guidance, and strength throughout this journey. It is also dedicated to my loving parents Mr. and Mrs. Ojeh whose selfless love and dedication has shaped me to the person I am today. You have been my pillar of strength and inspiration. To my friends and well-wishers, thank you for your encouragement, understanding, unwavering support and encouragement. Your positive energy and constant support have made this journey a fulfilling one. This work is a reflection of collective strength and inspiration I have drawn from each of you.

ACKNOWLEDGEMENT

My profound gratitude goes to Almighty God for His grace and mercy throughout my academic pursuit. I extend heartfelt thanks to my supervisor, PROF. F. AMADIN, for his continuous guidance, constructive feedback, and mentorship during the course of this project. Special appreciation also goes to DR. ROSEMARY USIOBAIFO , the Head of Department, for her outstanding leadership and support. Finally, I wish to thank my parents, Mr and Mrs OJEH, for their immense contribution to my academic growth.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This project focuses on the design and implementation of a lightweight honeypot system aimed at detecting and logging unauthorized access attempts on a local network. With cyberattacks becoming increasingly common and sophisticated, many small organizations lack the tools to properly monitor their networks. Honeypots offer a simple yet effective way to study intrusion behaviour by acting as decoy systems that attract potential attackers. The system developed in this project was built using Python due to its flexibility and strong support for network programming through libraries such as socket, datetime, and logging. It works by listening on a specific network port, recording each incoming connection, and logging details such as the intruder's IP address, timestamp, and input commands into a text file named honey.txt. Using the prototyping methodology, the system was built and tested within a controlled local environment using Telnet and Nmap to simulate intrusion attempts. The results confirmed that the honeypot could efficiently detect connections, capture relevant data, and operate smoothly with minimal resource use. Overall, this project shows that lightweight honeypots can provide valuable insights into network security and help raise awareness of potential threats, especially for small organizations and students. Future improvements could include multi-port monitoring, real-time alert features, and database-based log management to enhance system efficiency and scalability.

CHAPTER ONE

INTRODUCTION

1.1. Background to the Study

In today's digital world, the threat of cyberattacks continues to grow rapidly, affecting individuals, organizations, and entire governments. Attackers are constantly developing new ways to exploit system vulnerabilities, access private data, or disrupt services. As a result, cybersecurity has become a top priority across all sectors, and one of the proactive tools developed to counter these threats is the honeypot.

A honeypot is a decoy system or network resource designed to attract and trap unauthorized users or attackers. It mimics real systems but is isolated and monitored, allowing security professionals to study the behaviour, tools, and techniques used by intruders. Honeypots provide a safe environment where cyber threats can be analysed without putting actual systems at risk.

The concept of honeypots in cybersecurity dates back to the late 1980s and early 1990s, when researchers and security experts began creating fake systems to trick hackers and learn how they operate. One of the earliest well-known honeypot projects was "The Cuckoo's Egg" by Clifford Stoll, which documented how a fake network was used to track a real hacker. Since then, honeypots have evolved from basic traps to highly sophisticated systems capable of detecting advanced persistent threats (APTs) and insider attacks.

Honeypots are now used in corporate networks, academic institutions, and government infrastructures to gain valuable insights into emerging threats. However, many existing honeypot systems are complex, resource-intensive, or difficult to implement in smaller environments. This creates a gap for simpler, lightweight honeypot systems that can still

provide useful detection and logging of suspicious activity—especially on smaller, local networks such as those found in schools, offices, and small businesses.

This project focuses on the development of a lightweight honeypot system that can be deployed on a local network to detect unauthorized access attempts. The aim is to provide a functional and efficient tool for basic threat monitoring, while keeping the system simple enough to run on minimal resources. By simulating vulnerable services and logging suspicious activity, the honeypot system will serve as a practical tool for both learning and early threat detection.

1.2. Statement of the Problem

As cyberattacks become more frequent and advanced, many small networks like those in schools and offices remain vulnerable due to limited access to advanced security tools. Traditional systems such as firewalls and antivirus software focus on blocking threats and often fail to detect how intruders behave once they gain access. Honeypots can solve this problem by acting as decoy systems that attract and log attacker activity.

However, most existing honeypot solutions are either too complex to deploy or require high-performance systems and advanced configuration skills, making them impractical for small environments. This project aims to address the gap by developing a lightweight honeypot system that is easy to deploy and can monitor, log, and analyse suspicious activity on local networks. It is designed to improve network security awareness, especially for small organizations with limited technical resources.

1.3. Aim and Objectives of Study

The aim of this study is to design and develop a lightweight honeypot system. To achieve this aim the following objectives are outlined:

a) To design and implement a simple honeypot that simulates vulnerable network services.

b) To capture and log unauthorized access attempts, including attacker IP addresses and activities.

c) To provide a user-friendly interface for viewing intrusion logs and analysing threat data.

d) To test the honeypot system on a local network environment and evaluate its effectiveness.

e) To ensure the system is lightweight, easy to deploy, and suitable for small-scale networks.

1.4. Significance of the Study

This study provides a simple and affordable security tool for small organizations that lack advanced protection. The honeypot system helps detect unauthorized access and improves awareness of network threats. It can also be useful for students and researchers interested in learning how attacks happen and how to monitor them.

1.5. Scope of the Study

This study focuses on the design and implementation of a lightweight honeypot system for detecting unauthorized access on a local network. The system will simulate basic network services to attract intruders, log their activities, and present the information in a readable format. The project will cover only small-scale network environments, such as those found in schools, offices, or personal setups. It will not include large enterprise security systems or advanced threat response mechanisms.

1.6. Limitations of the Study

This study focuses on developing a lightweight honeypot for small network environments, meaning its deployment and performance in large-scale enterprise networks were not within the project's scope. The system monitors specific ports and does not detect all types of attacks,

especially those using encrypted channels or targeting unrelated services. It functions as a passive monitoring tool without active defence capabilities, and testing was conducted in a controlled environment due to time and resource constraints.

1.7. Definition of Terms

1) **Honeypot:** A security tool designed to attract attackers by simulating a vulnerable system, used to detect, study, and log unauthorized access attempts.
2) **Unauthorized Access:** Any attempt to enter or use a computer system or network without permission.
3) **Intrusion Detection:** The process of identifying and recording suspicious or malicious activity on a network.
4) **Network:** A group of connected computers or devices that share resources and data.
5) **Local Network:** A network that connects devices within a limited area such as a home, office, or school.
6) **Simulation:** The act of creating a fake version of a real system or service to observe how it is used or attacked.
7) **Lightweight System:** A system that is simple, easy to run, and does not require a lot of computing resources.
8) **Logging:** The process of recording events or actions that happen within a system for monitoring and analysis.
9) **Cybersecurity:** The practice of protecting computer systems, networks, and data from digital attacks or unauthorized access.
10) **Threat Monitoring:** The continuous observation of a network or system to detect signs of potential cyberattacks.
11) **Secure Shell (SSH) Service:** A program used for authentication and secure communication, and can also measure all SSH attempts.

CHAPTER TWO

LITERATURE REVIEW

This chapter reviews related works and previous research on honeypot technology, including methodologies and models used in designing intrusion detection systems. It defines the purpose, operations, and limitations of existing honeypot systems and introduces a proposed lightweight honeypot system designed to overcome these limitations and support monitoring of local network threats in low-resource environments.

2.1 Introduction

In today's digitally connected world, securing computer networks from unauthorized access and malicious activity has become a central focus of cybersecurity research. As hackers find smarter ways to break into systems, traditional defence mechanisms such as firewalls and intrusion detection systems (IDS) are no longer sufficient on their own. This has led to the emergence and growing importance of honeypot systems; a controlled decoy environments that are deliberately designed to appear vulnerable in order to attract, observe, and analyse attacker behaviour (Spitzner, 2003).

The concept of honeypots has evolved significantly over the past two decades, both in functionality and deployment scale. They are now widely used for intrusion detection, malware analysis, threat intelligence gathering, and behavioural research (Bailey et al., 2009). Unlike typical security systems that aim to block or prevent intrusions, honeypots take a proactive approach by inviting interactions from potential attackers. These interactions offer deep insights into attack patterns, tools, and motives that would otherwise remain hidden. Honeypots are generally categorized into two types based on the level of interaction they offer: Low-Interaction Honeypots, which emulate basic services and protocols to detect suspicious activity

without giving full access; and High-Interaction Honeypots, which simulate entire systems and allow attackers to engage more freely in a controlled environment (Zhu et al., 2015).

Each category serves different research and operational purposes, but they come with their own difficulties, resource demands, and data quality. Several notable honeypot systems have been developed to meet various security needs. For example, Honeyd, which provides low-level interaction and is known for simulating multiple virtual hosts on a single machine (Provos, 2004). Meanwhile, there's Kippo and its more advanced fork Cowrie focus on capturing detailed SSH-based attacks. Dionaea specializes in malware collection through network service emulation, and Glastopf targets web-based attacks by imitating vulnerable web applications. A recent innovation in honeypot systems is the 'SemanteCT IoT Honeypot', which integrates semantic web technologies with traditional honeypot mechanisms to simulate interactions with IoT devices. This model improves detection accuracy in smart environments and aids in modelling attacker behaviour using linked data techniques (Pa, Baldini, & Steri, 2021).

Despite the advancement in honeypot technology, a significant challenge remains: most of the existing systems are either too resource-intensive for smaller organizations or lack the adaptability required for modern threats. This creates a research opportunity to develop lightweight, effective honeypots tailored to local networks, especially in educational institutions or small-scale enterprise environments with limited technical infrastructure. This literature review explores the historical evolution, technical architecture, strengths, and limitations of existing honeypot systems. By analysing key contributions from previous research and tools, it lays the groundwork for the design of a new lightweight honeypot system that balances simplicity, effectiveness, and real-world applicability.

2.2 Historical Background

The concept of honeypots in cybersecurity dates back to the late 1980s, rooted in the principle of deceptive defence — a strategy that involves intentionally deploying vulnerable-looking systems to lure malicious actors into revealing their tools and behaviour. One of the earliest well-documented real-world applications of this strategy was in The Cuckoo's Egg by Clifford Stoll (1989), where an accountant-turned-astronomer helped uncover a Soviet hacker infiltrating U.S. military systems through the use of intentional traps and monitoring.

However, honeypots as a formalized cybersecurity toolset began to emerge in the late 1990s, particularly with the establishment of The Honeynet Project in 1999, which was founded by Lance Spitzner. The project brought global attention to honeypots as a structured method for studying cyberattacks, especially security threats that no one has seen before and malware that works without human control. Spitzner's own work, including the widely cited Honeypots: Tracking Hackers (2003), laid the conceptual foundation for many of today's honeypot technologies.

The first honeypots were simple setups, like fake Telnet or FTP servers with easy-to-guess passwords, designed solely to observe unauthorized access. Over time, these systems evolved in sophistication and purpose. By the early 2000s, tools like Backoffice Friendly and Honeyd began to simulate entire virtual networks, giving researchers the ability to observe attackers who believed they had accessed real systems (Provos, 2004). This marked the beginning of low-interaction honeypots, which were easy to deploy and configure, though limited in how deeply attackers could interact.

Soon after, high-interaction honeypots emerged. Full systems or virtual machines that attackers could compromise entirely, providing deep insights into post-exploitation behaviour. This led to the development of solutions like Argos, T-Pot, and the increasing use of virtual machines

and lightweight software containers to build fake networks for trapping hackers. (Bailey et al., 2009).

By the 2010s, honeypots began expanding into specialized roles:

- Kippo and Cowrie focused on SSH and brute-force attack logging
- Dionaea was optimized for malware collection through fake SMB/HTTP services
- Glastopf targeted web-based threats like SQL injection and remote file inclusion
- Distributed platforms like T-Pot integrated multiple honeypots with dashboards (Telekom Security, 2018)

This historical development reflects an ongoing trend: from simple decoys to powerful research tools capable of capturing real-time threats, aiding digital forensics, and improving intrusion detection systems. The rapid advancement of IoT technologies, cloud infrastructure, and automated attack tools in recent years has only heightened the need for honeypots that are more flexible, easier to deploy, and tailored for local or resource-constrained environments.

As this review will later explore, most traditional honeypots require significant technical effort to set up and monitor, making them impractical for smaller institutions or individual researchers, and this evolving gap will justify the need for new approaches such as, ‘Lightweight Python-based Honeypots that combine effectiveness with simplicity.

2.3 Classification of Honeypots

Honeypots come in different types, and each one is designed to serve a specific purpose. In general, they can be grouped based on how much attackers can interact with them and what kind of threats they are meant to detect or study. Understanding these categories helps explain why some honeypots are better suited for research, while others work well in real-world security setups.

2.3.1 Based on Level of Interaction

The most common way to classify honeypots is by how much access they give to attackers:

- Low-Interaction Honeypots

Low-interaction honeypots only imitate basic services, like a login screen or a fake webpage. They don't give full control of a system, but they're enough to trick automated attacks or basic scans. Because they don't run a full operating system, they're safer and easier to manage. Tools like Honeyd and Dionaea fall into this group.

For example; a fake FTP server that always accepts "admin" as a password is a low-interaction honeypot. It logs the attempt but doesn't let the attacker actually run real commands. These honeypots are good for collecting general attack patterns and spotting common hacking attempts. But they can't tell you much about what an attacker does after breaking in, since they don't allow deeper interaction.

- High-Interaction Honeypots

High-interaction honeypots give attackers full access to a real system or a virtual machine. The idea is to let them behave as they normally would download files, try commands, install malware all while the system quietly records everything.

Tools like Cowrie, T-Pot, and older research-based setups like Argos are examples. These honeypots give more detailed data, especially about post-intrusion behaviour. However, they also come with risks if not isolated properly, attackers could use the honeypot to harm other systems. Plus, they're more complex and require more time to manage.

2.3.2 Based on Purpose or Function

Some honeypots are used to study specific types of attacks:

- SSH Honeypots

These are designed to catch login attempts over SSH, a secure way of accessing remote systems. Tools like Kippo and Cowrie fall into this group. They log everything from guessed passwords to the exact commands attackers try to run after breaking in.

- Malware Collection Honeypots

These systems pretend to run services that are often targeted by malware (like file sharing or web servers). When a malware script tries to infect them, the honeypot captures the malicious file. An example is Dionaea, which is great for collecting viruses, worms, and trojans for analysis.

- Web Application Honeypots

These imitate vulnerable websites to attract attacks like SQL injection, cross-site scripting (XSS), or file upload exploits. Glastopf is a well-known tool in this category. It lets researchers understand how web-based threats are evolving.

- IoT Honeypots

Newer honeypots like SemanteCT focus on Internet of Things (IoT) devices such as smart home gadgets or cameras because attackers are increasingly targeting them. These honeypots simulate IoT behaviour and trap specialized attacks meant for low-power devices.

Classification Type	**Example**	**Best for**
Low-Interaction	Honeyd, Dionaea	Quick deployment, safe environments
High-Interaction	Cowrie, T-Pot	Deep attack analysis, full behaviour tracking
SSH-focused	Kippo, Cowrie	Password attacks and command monitoring
Malware-focused	Dionaea	Collecting real malware samples
Web App-focused	Glastopf	Testing web app threats like SQL injection
IoT-focused	SemanteCT	Monitoring smart devices and home networks

Classification Table.

This classification makes it clear that no single honeypot can handle every kind of threat. Choosing the right one depends on the goal. Whether it's to gather attack data, collect malware samples, or study how attackers behave after breaking in.

2.4 Review of Key Honeypot Systems

Over the years, since the early 2000's, different honeypot systems have been developed to for different reasons in cybersecurity world and each one has its own features, focus areas, and limitations. Some are made for research, while others are built for real-world use. So, in this section we would review a few of the most widely used honeypots, explain what makes each of them unique and also why they play a key role in helping us study and respond to security threats.

2.4.1 Honeyd

Honeyd is one of the earliest and most widely recognized low-interaction honeypots. It was created by Niels Provos in the early 2000s and is best known for its ability to simulate multiple virtual hosts on a single physical machine (Provos, 2004). Each virtual host can pretend to be a different operating system, complete with fake network services like HTTP, FTP, or SSH.

What made Honeyd stand out was its ability to fake basic network scans by giving the attackers the impression that they were seeing many real machines whereas in reality, it was all fake. For example, a hacker would scan a university network and think they found 30 servers, but it was just one machine running Honeyd with 30 fake identities.

So, Honeyd became very popular in research and education because it was lightweight, flexible, and could run on almost any hardware. However, it also had its limits. Since it is a low-interaction honeypot, attackers couldn't do much after connecting and they would quickly realize the system wasn't real after they tried deeper commands. It also lacks updates and modern logging features, making it less effective against today's more advanced threats.

2.4.2 Kippo

Kippo is an SSH honeypot designed to capture brute-force login attempts and record everything that happens after an attacker successfully logs in. Developed by an open-source contributor under the name “odellin,” Kippo came into the spotlight around 2010 as a tool for monitoring SSH attacks in real time.

What made Kippo powerful was its fake file system. Once a hacker logged in with a guessed username and password, they could “see” a fake Linux environment. They could type commands, try to download malware, or run scripts and Kippo would log every keystroke.

Imagine a hacker logs in and tries to create a new user, install a backdoor, or upload malware. Kippo records all of it like a hidden camera.

Kippo helped researchers understand how real attackers behave once inside a system. But over time, it became outdated. It lacked support for newer protocols, and its logs were hard to manage and these issues led to the creation of its improved version: ‘COWRIE’.

2.4.3 Cowrie

Cowrie is the modern replacement for Kippo, developed by ‘Michel Oosterhof’ in 2015. Like Kippo, it’s an SSH honeypot but it added more realistic features like better logging, and support for Telnet. Cowrie is now one of the most widely used honeypots for SSH monitoring in both research and production environments.

Some of Cowrie’s improvements include:

- A more realistic fake file system (copied from real Linux systems)
- Support for file uploads and downloads (SCP protocol)
- A more detailed logging system

With Cowrie, you don’t just know someone tried to log in, you get a complete picture of what they did afterward. Cowrie is still being actively maintained and is often used in honeynet setups and research environments. It offers a balance between ease of use and deep attack

insight. However, because it's a high-interaction system, it still requires careful setup and system isolation to avoid being misused.

2.4.4 Dionaea

Dionaea honeypot was developed by Markus Koetter and was officially released around 2010. It was designed specifically to collect malware. It simulates vulnerable services like SMB, FTP, and HTTP, and waits for attackers to try and exploit those services. When they do, Dionaea captures the payload which is the actual malware file for later analysis (Koetter, 2010).

Dionaea is often used in large-scale honeynet deployments where researchers want to gather a wide range of malicious samples. It can store data locally or forward it to analysis tools like Cuckoo Sandbox or VirusTotal. For example, a worm tries to infect everything on the network, Dionaea will catch the file, log where it came from, and also record how the attacker tried to break in.

Dionaea is also helpful for antivirus research, botnet tracking, and used to study malware behaviour. But like other honeypots, it needs to be isolated so that the malware it collects doesn't accidentally escape into the real network.

2.4.5 Glastopf

Glastopf is a web application honeypot created by Lukas Rist. It focuses on detecting web-based attacks, like SQL injection, remote file inclusion (RFI), and cross-site scripting (XSS). Glastopf simulates vulnerable PHP applications and responds to attacker input in a way that looks real.

The idea behind Glastopf is simple: attackers scan websites looking for weak spots. Glastopf pretends to be one of those weak sites. When an attacker sends a malicious request, the system records it giving researchers insight into how attackers are trying to exploit the web.

While Glastopf is very effective at gathering attack patterns, it doesn't emulate full web servers or allow deep interaction, which is still an important part of honeynet setups focused on defending web apps.

2.4.6 SemanteCT IoT Honeypot

SemanteCT is one of the more recent honeypots and focuses on the growing number of attacks targeting Internet of Things (IoT) devices like smart home gadgets, cameras, and sensors. Developed by cybersecurity researchers in 2022, SemanteCT uses semantic deception meaning it doesn't just fake a device, it behaves in ways that seem human or natural to fool attackers even more convincingly (Vanhoenshoven et al., 2022).

Unlike traditional honeypots that only respond to basic input, SemanteCT uses intelligent response systems. For example, it might delay a reply or use context-aware messaging, so attackers believe they're interacting with a real IoT device. A hacker might try to access a fake smart thermostat, and SemanteCT lets them log in, adjust "settings," and receive responses all while logging everything.

SemanteCT is still more academic than commercial, but it shows where honeypot research is heading, which is toward smarter, more believable, and more adaptive traps. Its lightweight design also makes it suitable for smaller environments similar to the goals of this project.

2.5 Overview and Analysis of Existing Honeypot Systems and Limitations

Honeypot Systems	**Purpose**	**Operation**	**Limitations**
Honeyd (Provos, 2004)	Simulate OS/services.	Makes use of configuration files to emulate devices.	Complex setup and scripting required.
Kippo/Cowrie (Rist et al., 2010).	Secure Shell (SSH) honeypots.	Logs login attempts and commands.	Requires VM; limited to SSH.
Dionaea (Baecher et al., 2010).	Malware capture.	Emulates vulnerable network services.	Heavy setup; limited to specific protocols.
Glastopf (The Glastopf Project, 2013).	Web application honeypot.	Simulate vulnerable web interfaces.	Focused on HTTP; lacks versatility.
SemanteCT IoT Honeypot (Pa et al., 2021).	IoT-focused interactions.	Makes use of semantic web for threat modelling.	Requires semantic infrastructure.

Overview of Related Systems

These systems generally operate by simulating vulnerable services or environments, waiting for unauthorized access, logging malicious activities, and storing the data for later review.

While each of these systems addresses specific threat types and environments, their deployment is often hindered by high resource consumption, complex configurations, and the need for advanced user expertise, limiting their application in smaller, less technical businesses, academic institutions or environments.

2.6 Related Researches and Implementations

Honeypots have been widely studied in cybersecurity research, education, and even national defences. Many organizations, from universities to industry labs have used honeypots not only to detect threats but also to understand how attackers behave once inside a system. In this section I will highlights some key research efforts and past real-world implementations that have helped shape the field, while also supporting the relevance of this project.

Academic Research and Case Studies

One of the most well-known efforts in honeypot research is The Honeynet Project, a global non-profit group founded in 1999. Its goal has always been to study cyberattacks in real time and share findings openly. Over the years, it has developed tools like Sebek for keystroke logging, Honeywall for traffic control, and supported honeypots like Dionaea and Glastopf.

In a study by Bailey et al. (2009), researchers deployed high-interaction virtual honeypots across a university network. Their goal was to monitor worm activity and automated scanning. The results showed that even simple honeypots attracted thousands of connection attempts per day, proving how frequently random attacks happen online.

Theres another paper, by Antunes et al. (2013) focused on using Kippo in a university setting to teach students about cybersecurity. They found that Kippo helped students understand SSH-based attacks and raised awareness about weak passwords, command logging, and system hardening. This proved that honeypots are not just research tools but valuable educational resources, because in both of these studies, honeypots were not used to stop attacks, but to learn from them, which is the same core idea behind this project.

Industry and Government Use

Outside of academics, honeypot systems have also been deployed by companies and governments. For example, Telekom Security released T-Pot, a full honeynet platform that combines multiple honeypots (like Cowrie, Dionaea, and others) with a web-based dashboard using ELK (Elasticsearch, Logstash, Kibana). This makes it easier to visualize threats and analyse data in real time.

We also have 'CERTs' (Computer Emergency Response Teams) which in many cases and in several countries have used honeypots to track national-level threats, including botnet behaviour, ransomware infections, and large-scale scanning campaigns. While these setups are often complex and require strong IT skills, they've shown that honeypots are a reliable tool for early warning and investigation.

Honeypots in IoT and Modern Networks

With the rise of IoT (Internet of Things) devices, honeypot research has broadened its focus, devices like smart home assistants, cameras, and routers are often poorly secured which makes them easy targets. So, tools like IoTPOT and the newer SemanteCT honeypot were built to simulate these devices and capture how attackers try to take control of them.

In a recent study by Vanhoenshoven et al. (2022), SemanteCT was used to explore how attackers respond to semantic deception, like fake IoT settings, time delays, and user behaviour simulations. The study showed that attackers took longer to realize they were interacting with a trap, which in turn gave researchers more time to gather data. This is an important step toward more intelligent honeypots, and directly supports the idea behind this project.

2.7 Research Gaps and Limitations of Existing Honeypots

Many honeypots have been developed to help detect and analyse cyber threats, but none are perfect. This section highlights key limitations in existing systems.

- **Complex Setup and Configuration**

Many honeypots require multiple installations and tools to function well. Systems like Cowrie or T-Pot demand experience and infrastructure, which smaller organizations often lack.

- **Heavy Resource Usage**

High-interaction honeypots need a lot of memory, storage, and processing power. Tools like Dionaea, while powerful, can overload basic systems.

- **Limited Flexibility**

Most honeypots focus on one type of attack (SSH, HTTP, etc.). They may miss other attack types completely.

- **Lack of Modern Behaviour Simulation**

Older honeypots don't adapt to attacker behaviour. They're easy to spot once someone interacts with them deeply.

- **Not Beginner-Friendly**

Most tools are hard to install or understand for new users or students.

Because of these issues, there is a clear need for a lightweight honeypot that runs easily on basic systems, doesn't require advanced setup, and can still gather useful threat information. And that the exact aim of this project, to build a simple, python-based honeypot that helps users monitor local networks effectively without complex tools or high-end equipment.

CHAPTER THREE

SYSTEMS ANALYSIS AND METHODOLOGY

This chapter presents the methodology and system analysis approach adopted in the development of the proposed lightweight honeypot system. The aim is to explain how the project was planned, the steps followed during development, and the reasons behind the chosen method. Since this project involves both research and practical implementation, it was important to use a step-by-step development process that allows gradual improvement based on testing and feedback.

3.1 Methodology Adopted

For this project, the Prototyping methodology was adopted. This model was chosen because it allowed me, the developer to build a basic version of the system early, test it, and gradually improve it based on results and observations. Since the project involves creating a simple honeypot tool for detecting suspicious activity on a local network, this method made it easier to develop and refine the system in manageable stages.

The Prototyping model is particularly suitable for projects where the full system requirements may not be clear at the start, or when the goal is to try different ideas and learn from how they perform. In this case, the initial prototype focused on building a basic TCP server that listens on a chosen port and logs connection attempts.

Another reason why I choose the prototyping model is because honeypot research is highly practice-oriented. Building and testing a prototype offers more reliable insights compared to relying on theoretical analysis alone. It also ensures that the proposed system is not just a conceptual model but a functional tool that can help small organizations understand and monitor network intrusions.

3.2 Analysis of the Existing System

3.2.1 Overview of Existing Honeypot Systems

Over the years, honeypot systems have become a valuable part of cybersecurity strategies, especially in detecting and analysing unauthorized access attempts. Many of these systems have been designed to monitor specific types of threats, such as SSH brute-force attacks, malware payload delivery, or web application vulnerabilities. While these tools are effective in certain environments, they are often built for large-scale deployments and can be too complex or resource-intensive for small networks, schools, or individual users.

For starters, one of the earliest honeypots, Honeyd, was developed to simulate multiple virtual systems from a single machine. It allowed researchers to create fake services like FTP or Telnet on fake IP addresses. However, while it was lightweight and flexible, it is now outdated and no longer maintained making it less reliable for modern use.

Tools like Kippo and its improved version, Cowrie, were developed to simulate SSH servers and capture attacker behaviour after login. These tools offer rich logs and fake file systems to interact with, which is valuable for deep analysis. However, setting them up requires installing additional services, configuring virtual environments, and sometimes integrating log analysis dashboards all of which may be too demanding for users with limited technical experience.

There are also more advanced platforms, such as T-Pot, bundle several honeypots into one full-featured system, complete with dashboards and real-time monitoring. While powerful, these platforms are designed for enterprise use and require significant hardware resources and system knowledge to deploy correctly.

Furthermore, recent tools like SemanteCT have introduced the concept of semantic deception for IoT honeypots which allowed attackers to interact with a fake smart device in a realistic way. Although this approach represents a modern shift in honeypot technology, tools like SemanteCT are still experimental and more suited for academic research than everyday use.

3.2.2 Strengths of the Existing Systems

Existing honeypot and IDS solutions have several strengths that make them valuable in cybersecurity research and defence:

- **Attack Visibility** – Honeypots provide detailed insight into attacker behaviour. For example, systems like Cowrie log every keystroke from an attacker during SSH sessions, which helps researchers understand attack patterns.
- **Threat Intelligence** – By capturing IP addresses, commands, and malware samples, honeypots generate real-world threat intelligence that can be shared across security communities.
- **Early Warning System** – Honeypot tools such as Snort act as an early warning mechanism, alerting administrators when suspicious activities are detected.
- **Resource Efficiency** – Low-interaction honeypots like Honeyd require minimal hardware resources and are easy to deploy, making them suitable for small networks.
- **Specialization** – Some honeypots are tailored for specific environments. For example, the SemanteCT IoT Honeypot focuses on IoT devices, providing a unique advantage in detecting threats targeting smart homes and industrial IoT systems.

3.2.3 Weaknesses and Limitations of Existing Honeypot Systems

Although honeypot systems have proven useful in detecting and studying malicious activity, they do have their own weaknesses and limitations;

One key limitation is high resource demand. Advanced honeypots, especially high-interaction ones, often require powerful hardware, large storage, and constant monitoring. This makes them unsuitable for small organizations or local networks with limited resources. Another challenge is ease of detection by attackers. Experienced attackers can sometimes identify honeypots based on unusual responses or restricted system behaviour. Once discovered, attackers may avoid the honeypot altogether, reducing its effectiveness.

There is also the problem of data overload. Many honeypots generate a massive amount of logs, including irrelevant or repetitive information. Without proper filtering, this can overwhelm administrators and make it difficult to quickly spot real threats.

Finally, limited adaptability to modern attack methods is a concern. Some existing honeypot systems were designed years ago and have not kept pace with threats targeting cloud platforms, IoT devices, or automated malware campaigns. As a result, they may miss or poorly capture newer attack techniques.

3.2.4 Justification for a Lightweight Honeypot

From the weaknesses of existing honeypot systems, it is clear that there is a need for a solution that is both effective and practical. Large-scale honeypots, while powerful, demand significant resources and constant administration. On the other hand, older lightweight solutions are easier to deploy but lack the ability to capture modern threats in a meaningful way.

This project seeks to close that gap by developing a lightweight honeypot system that is simple enough for small organizations or individuals to use, yet capable of capturing valuable information about unauthorized access attempts. By focusing on easy deployment, low

resource usage, and straightforward logging, the system can provide practical security benefits without the complexity of larger honeypots.

The justification for this approach lies in its relevance to environments that are often overlooked — small networks, campus labs, or businesses without dedicated cybersecurity staff. A lightweight honeypot gives such users the ability to detect intrusions, understand possible attack patterns, and raise awareness of security risks, all without heavy investment in infrastructure.

3.3 System Requirements Analysis

A honeypot system must be carefully planned before it is implemented. To achieve the project's goals, it is important to first outline the requirements that guide its development. These requirements are divided into functional requirements (what the system must do) and non-functional requirements (how the system should perform).

3.3.1 Functional Requirements

The functional requirements describe the specific tasks the honeypot should carry out. They include:

- **Listening for connections:** The system must open a port (e.g., SSH or Telnet) and wait for unauthorized connection attempts.
- **Capturing data:** Any input or commands sent by an attacker should be logged for analysis.
- **Recording metadata:** Details such as IP address, timestamp, and session activity should be stored.
- **Log storage:** All captured data must be saved in a log file for later review.

3.3.2 Non-Functional Requirements

These requirements focus on qualities that make the system practical and reliable. They include:

- **Lightweight operation:** The system should use minimal resources, making it deployable on standard computers without special hardware.
- **Simplicity:** Deployment and usage should be straightforward so that even users with limited technical knowledge can run it.
- **Scalability:** The design should allow for expansion, such as monitoring additional ports if needed.
- **Reliability:** The honeypot must remain active for long periods without frequent crashes or manual restarts.
- **Security of logs:** Logs must be stored securely to prevent tampering, ensuring accurate records of intrusion attempts.

3.4 System Design

System design explains how the honeypot will be structured and how its different parts will work together. Since this project is focused on building a lightweight honeypot, the design is kept simple but effective, ensuring that it captures unauthorized access attempts while remaining easy to deploy.

3.4.1 Architectural Design of the Proposed System

The proposed honeypot consists of three main components:

- **Trap Module (Listener):**
    - Opens a network port and waits for incoming connections.
    - Simulates a vulnerable service (e.g., SSH or Telnet).

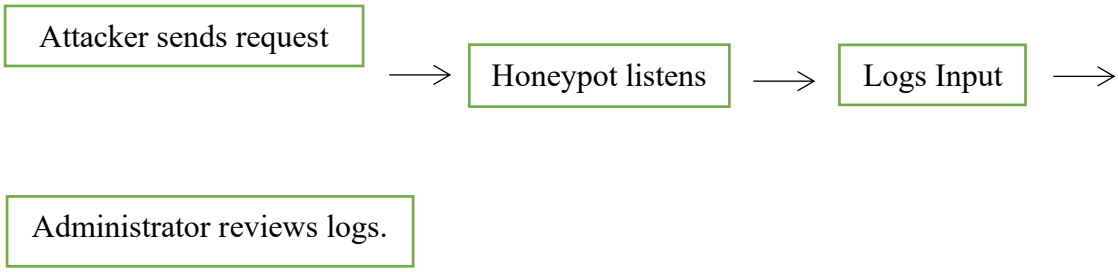
- Accepts any data sent by the attacker.

- **Logging Module:**
  - Records attacker inputs, IP addresses, timestamps, and session details.
  - Stores the data in a structured log file for later analysis.

- **Alert/Monitoring Module:**
  - Provides simple console output or log updates to notify the user when suspicious activity is detected.
  - Allows network administrators to review attempted attacks in real time.

3.4.2 Activity Diagram

To explain how the honeypot works, an Activity Diagram can show the flow of actions between the system and its users:



The attacker tries to connect to the honeypot, while the System Administrator: Reviews logs and checks for suspicious activity

3.4.3 Data Flow Diagram (DFD)

The Data Flow Diagram (DFD) illustrates how data moves through the Python Honeypot System. It provides a high-level view of the system's internal processes, external entities, and data stores, showing how attacker interactions are captured, stored, and communicated to the system administrator.

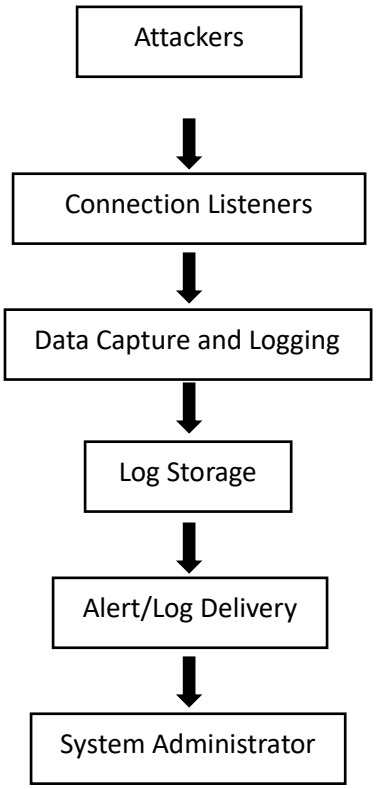


Fig. 1. Data Flow Diagram (DFD)

The process begins with the attacker, who sends connection requests, commands, or probes to the network. These requests are first received by the Connection Listener, which monitors specific ports for incoming traffic. The listener forwards any detected activity to the Data Capture & Logging process, where important details such as the attacker's IP address, the time of access, and any commands issued are recorded.

This information is stored in the Log Storage, which acts as a secure repository for all captured data. When necessary, the Alert/Log Delivery process retrieves the relevant logs and sends them to the System Administrator. The administrator can then analyse these reports to identify security threats, understand attack patterns, and take appropriate preventive measures.

3.5 Choice of Tools and Technologies

The success of this project depends on selecting the right tools and technologies that balance simplicity, functionality, and ease of use. Since the aim is to build a lightweight honeypot, the chosen tools are minimal but powerful enough to achieve the objectives.

- **Programming Language – Python**

  Python was selected because of its simplicity, readability, and large library support. It is widely used in cybersecurity research and allows rapid development of networking applications. Modules such as socket for network communication and logging for activity recording make it ideal for implementing a honeypot.

- **Development Environment – Visual Studio Code (VS Code)**

  VS Code provides a user-friendly environment for writing, editing, and running Python code. It supports extensions for Python debugging, syntax highlighting, and code formatting, making development smoother.

- **Operating System – Windows**

  The honeypot is designed to run on both Windows and Linux systems. But I would be using a Windows Operating System.

- **Ports Used**

  The honeypot listens on configurable ports, like 2222 for SSH-like services or 23 for Telnet and these ports are chosen because attackers frequently target them.

- **Logging System**

  The system uses plain text log files to store attacker activity. Each entry includes the attacker's IP address, timestamp, and the data sent. This ensures logs can be easily reviewed without requiring a database.

- **Testing Tools – Telnet**

Telnet is used for manual testing by connecting to the honeypot and sending input.

This combination of tools was chosen because it is lightweight, beginner-friendly, and effective enough to meet the project's objectives while still being easy to deploy in small organizations or local networks.

3.6 Justification of Methodology

The choice of methodology and tools for this project is guided by the need to design a honeypot that is simple, lightweight, and practical for small organizations or local networks.

First, I adopted the prototyping approach because it allows for building a small but functional version of the honeypot early on, testing it, and then improving it based on performance. This approach fits the project because the main goal is not to create a large-scale enterprise honeypot but a minimal system that can demonstrate how unauthorized access attempts are detected and logged.

Then, I selected Python as the implementation language due to its ease of learning, wide community support, and strong networking libraries. Unlike other languages that may require complex setups, Python provides a direct way to build socket-based applications, which is at the heart of a honeypot.

Third, the system design diagrams (Activity Diagram, and DFD) provide a structured way to represent how the honeypot works logically. They make it easy to explain the system to non-technical stakeholders and also act as a guide for implementation.

Finally, using simple text-based logging instead of a database ensures that the project remains lightweight and easy to run even on resource constrained systems. This is important since the target users are individuals, students, or small organizations with limited technical resources.

In summary, the methodology and tools were chosen because they ensure the project is achievable within the given timeframe, easy to understand, and effective in meeting the stated objectives.

CHAPTER FOUR

SYSTEMS IMPLEMENTATION AND RESULTS

4.1 System Implementation

This chapter will explain how the proposed honeypot system is designed and implemented. It covers both the logical and physical design of the system, starting with a general overview of its structure and then presents diagrams such as a data flow diagram, and flowchart to visually show how the system works. These diagrams will help describe how data flows from the point of connection to where it is logged. After the design phase, we would outline how the system was built using Python, and that includes a breakdown of different key components, code explanations, and testing procedures used to confirm the system's effectiveness. Still, the overall aim is to create a lightweight and easy-to-deploy honeypot that can be used in small networks for monitoring unauthorized access.

4.1.1 Development Environment

The honeypot system was designed and implemented using tools chosen for simplicity, cross-platform support, and ease of testing.

All development was carried out on a Windows 10 computer using Visual Studio Code (VS Code) as the integrated development environment (IDE).

The implementation language was Python 3.13.5, executed with the default Python interpreter in VS Code's integrated terminal.

The system relies entirely on Python's standard library; therefore, no external installations were required.

Some Key Modules I Used Include:

No	Library	Purpose
1.	socket	Creates the network listener that accepts incoming TCP connections.
2.	datetime	Generates accurate timestamps for each log entry.
3.	Operating system	Handles file creation and path operations for the log file.
4.	signal	Enables clean shutdown of the honeypot when Ctrl + C or system termination occurs.
5.	sys	Allows graceful exit of the program.
6.	alert	Provides real-time visual feedback when new connection detected.

The honeypot script is saved as honeypot.py, and all connection logs are stored in a text file named honey.txt located in the same directory.

The system is executed from VS Code's terminal with the command:

```
python honeypot.py
```

Testing and simulation were performed locally using the loopback address (127.0.0.1) and external network-scanning tools such as Telnet and Nmap to mimic unauthorized access attempts.

This environment provided a stable and controlled platform for development and evaluation.

4.1.2 System Modules

The honeypot program is built as a small set of cooperating modules inside the single script honeypot.py. The main modules are:

1. Trap (Listener) Module

2. Session Handler / Input Normaliser
3. Logging Module
4. Alert Module
5. Shutdown Handler

Each module is explained below with reference to the actual code.

1. Trap (Listener) Module

This module opens a TCP socket and waits for incoming connections. It is the first point of contact for any remote client or attacker:

- The socket is created and configured here:

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((HOST, PORT))
server_socket.listen(5)
```

Fig.4.1. Trap

These lines open a TCP listener bound to the host 0.0.0.0 and port 2222 (the HOST and PORT are set at the top of the code). The listen (5) call allows up to 5 pending connections.

The listener accepts the connection and passes it to the session handler for processing.

2. Session Handler / Input Normaliser

Once a connection is accepted, the script reads incoming bytes, normalises line endings, and extracts complete lines typed by the remote user:

- Immediately after accept (), the script sends a message: “Welcome!”

  This makes the connection feel interactive and encourage the remote users(hackers) or automated bots to send input.

- The code reads data in a loop, decodes it, normalises CRLF/CR to \n, and buffers partial lines. This logic ensures the honeypot records exactly what was typed per line. Partial, non-terminated input is kept in partial until the next recv().
- Completed lines are written immediately to the log and stored in session_data for session-level summary. Any leftover partial data at disconnect is also recorded as a final line.

This design captures human-like typing and bot commands reliably while avoiding broken entries caused by inconsistent newline formats.

3. Logging Module

The logging module records every session into a local text file called honey.txt. Each connection begins with a session header showing the IP address and time of access, followed by any text sent by the intruder, and ends with a closing line to mark the end of the session.

Sample Log:

```
--- Session from ('127.0.0.1', 60969) at 2025-11-02 11:37:19.240523 ---
drop down
My Name is ELLIS.
--- End of session ---
```

```
--- Session from ('127.0.0.1', 49628) at 2025-11-02 10:30:57.976337 ---
(no text received)
--- End of session ---
```

Fig.4.2. Sample Log:

This structure allows an administrator to easily review network activity and identify suspicious behaviour.

4. Alert Module

This module provides real-time visual feedback whenever a new connection is detected. As soon as an incoming connection is accepted, the system displays a warning message on the console:

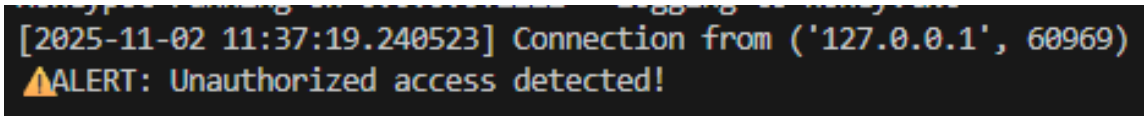


Fig.4.3. Alert Module

This feature helps the administrator immediately recognize potential intrusion attempts while the honeypot continues to log all activity in the background. The alert module was implemented using a simple print statement, which makes it lightweight and reliable. It can later be expanded to include email or SMS notifications for a more advanced alerting system.

5. Shutdown Handler

This module ensures that the honeypot exits safely when the administrator stops it. By using Python's signal and sys libraries, the system can detect when the user presses Ctrl + C or when the program is terminated, and it will then close the socket properly before exiting. This prevents port conflicts and keeps the program stable for future runs.

How these modules interact

- Listener accepts a connection (Trap).
- Welcome message is optionally sent to the client.
- Session handler reads, normalises and splits data per line.

- Each completed line is appended to honey.txt immediately (Logging).
- On disconnect, remaining partial data is saved or a (no text received) note is written.
- Administrator reviews honey.txt to investigate sessions.

4.2 System Testing

System testing was carried out after implementation to ensure that the honeypot functioned as expected. The testing process was designed to check whether the system could successfully detect unauthorized connections, record the attacker's details, and store the captured information in the log file (honey.txt). The testing was conducted in a local environment to avoid exposing the system to real external threats while still simulating realistic attack attempts.

4.2.1 Test Plan

The main goal of the test plan was to validate the core functions of the honeypot. These include:

1. Confirming that the honeypot listens on the correct network port.
2. Ensuring that it accepts incoming connections.
3. Verifying that it logs important details such as the IP address, timestamp, and data entered.
4. Confirming that the honeypot can handle multiple connection attempts without crashing.

The testing was done on the same Windows 10 system used during the development. The following tools were used during testing:

- Telnet: Used to simulate an attacker manually connecting to the honeypot.
- Nmap: Used to perform a port scan to confirm that the honeypot's listening port was visible and open.
- Text Editor (VS Code): Used to review the contents of the honey.txt file after each test to verify if the data was correctly recorded.

During testing, the honeypot was first launched using the command: python honeypot.py

After the system started, connection attempts were made using Telnet and Nmap. Each test was repeated several times to ensure consistent results.

4.2.2 Test Results

The results of the testing confirmed that the honeypot system performed as expected. When the honeypot was running and a connection was initiated using the command: telnet 127.0.0.1 2222, The system immediately responded with the message 'Welcome.' This verified that the honeypot was active and listening on the specified port (2222).

Each command or text entered through Telnet was recorded in the log file, along with the time and IP address of the connection.

A sample section of the log file is shown below:

```
--- Session from ('127.0.0.1', 60969) at 2025-11-02 11:37:19.240523 ---
drop down
My Name is ELLIS.
--- End of session ---
```

Fig.4.4. Test Results

This result confirms that the honeypot successfully captured and stored session data.

Next, a port scan was performed using Nmap with the command: nmap -sV -p 2222 127.0.0.1. The output showed that the port '2222/tcp open', which verified that the honeypot was visible on the network, just like a real server would appear to an attacker.

Overall, the test proved that the system could correctly log unauthorized connections, capture user input, and maintain stable operation without errors.

4.3 Results Presentation

The results obtained from the system testing were carefully reviewed to determine whether the honeypot met its design objectives. The goal of this stage was to present and interpret the captured data in a simple, understandable format that clearly shows the system's effectiveness. After running several connection tests using Telnet and Nmap, all captured data was stored automatically in the log file named honey.txt. Each new connection created a new session entry that included the following details:

- Source IP address of the connection attempt.
- Date and time the attempt occurred.
- Text or commands typed during the session.
- A short line indicating the end of the session.

Below is an example of the log output generated by the system:

```
--- Session from ('127.0.0.1', 49628) at 2025-11-02 10:30:57.976337 ---
(no text received)
--- End of session ---


--- Session from ('127.0.0.1', 60969) at 2025-11-02 11:37:19.240523 ---
drop down
My Name is ELLIS.
--- End of session ---


--- Session from ('127.0.0.1', 60156) at 2025-11-02 12:46:26.622741 ---
hello
setpassword-1245
It's me Admin Ellis
--- End of session ---
```

Fig.4.5. Results Presentation

This output demonstrates that the honeypot successfully captured every line entered by the connecting client. The system accurately recorded the timestamp and IP address, proving that the logging function worked as intended.

When the honeypot was scanned using Nmap, the results showed:

```
2222/tcp open  EtherNetIP-1
```

This confirmed that the honeypot was clearly visible to external scans and behaved like a genuine network service.

If this were a real network environment, such visibility would attract attackers who typically look for open ports to exploit.

From the results, it is clear that:

- The honeypot correctly listens, accepts, and records connections.
- The logging format is clean and easy to interpret.
- The system responds consistently during repeated tests without crashes.

So, the results show that the honeypot system meets its primary objective of detecting and recording unauthorized access attempts on a local network. It provides a simple yet effective way to monitor network activity and gain insight into potential attack patterns.

4.4 Discussion of Findings

The results from the testing and implementation of the honeypot system confirmed that the project achieved its main goal — to design a lightweight and easy-to-deploy tool for monitoring unauthorized access attempts on a local network.

During testing, the honeypot successfully detected and logged all connection attempts made through Telnet and identified as open when scanned with Nmap. This means that the system performed its task of simulating a real network service and capturing essential connection data.

The log file output showed the correct structure, including timestamps, IP addresses, and user input. This level of detail is valuable for network administrators or researchers who want to understand how attackers interact with open systems. It also demonstrates how even a simple honeypot can provide useful information about attempted intrusions without requiring complex or expensive tools.

One key observation was that the honeypot reacted immediately to every connection and logged each attempt without delay or system crashes. This shows that the use of Python's socket library and simple file logging approach provided good reliability and performance for a small-scale system.

However, it was also observed that the honeypot does not record the exact data packets or commands sent by automated bots beyond plain text. It also does not issue any alerts in real time. This means that while it is effective for data collection and monitoring, it would need further enhancement to handle live intrusion detection or automated responses.

In summary, the findings demonstrate that the honeypot system is practical, functional, and fulfills the intended purpose of the study. It effectively captures unauthorized access attempts, provides clear logs for analysis, and offers a simple yet educational tool for small organizations or students learning about network security. Despite its limitations, the system provides a strong foundation for future improvement, such as adding real-time alerts or expanding to multiple monitored ports.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Summary of the Study

This project focused on the design and implementation of a lightweight honeypot system aimed at monitoring and recording unauthorized access attempts within a local network. The goal was to create a simple, easy-to-deploy solution that could help small organizations or individuals understand how network intrusions occur without requiring advanced technical skills or expensive tools.

The system was developed using Python, chosen for its simplicity and built-in libraries that support socket programming and file handling. The honeypot listens on a selected port, accepts incoming connections, and logs key details such as the source IP address, the time of connection, and any input received. All captured information is stored in a text file named honey.txt for later review and analysis.

The design process included both logical and physical system representations, showing how data flows from the attacker to the honeypot and then to the log file. Implementation and testing were carried out in a controlled environment using Telnet and Nmap to simulate real network activity. The results confirmed that the honeypot successfully detected and logged unauthorized access attempts, meeting the main objectives of the project.

Overall, this study demonstrated how an accessible and resource-efficient honeypot system can contribute to better understanding of cybersecurity monitoring and serve as a foundation for future research in network defence mechanisms.

5.2 Contributions to Knowledge

This project contributes to the growing body of knowledge in the field of cybersecurity by providing a practical example of how honeypot systems can be designed and implemented using simple tools. While many existing honeypots are complex and resource-intensive, this project demonstrates that a lightweight version can still offer valuable insights into unauthorized access attempts and network behaviour.

One key contribution of this study is the creation of a basic honeypot framework that can be easily deployed by small organizations, students, or researchers for educational and monitoring purposes. By relying solely on Python's built-in libraries, the system removes the need for external dependencies, making it easier to set up and understand.

The project also helps to promote security awareness, showing how monitoring tools can be used not only to detect attacks but also to study how potential intruders interact with open ports or exposed services. The structure of this honeypot encourages learning by observation, which is valuable for those new to cybersecurity research.

Finally, this study adds to local academic research by focusing on affordable and scalable network defence methods, suitable for smaller environments such as universities, small businesses, or individual users who may not have access to commercial security systems.

5.3 Conclusion

This project set out to design and implement a lightweight honeypot system that could monitor and log unauthorized access attempts in a local network environment. Through the design, implementation, and testing phases, the system proved to be effective in capturing and recording basic intrusion activities, confirming that even simple tools can be useful in enhancing cybersecurity awareness.

The results of the study showed that the honeypot successfully detected every simulated connection and logged the corresponding IP address, timestamp, and input provided by the user. The data collected during testing demonstrated how attackers or automated bots might interact with open ports, providing valuable insights that can help improve network defenses.

Although the honeypot was intentionally built to be simple, it fulfills its intended purpose of showing how proactive monitoring can reveal potential security threats. It also highlights the importance of network surveillance, even in small organizations or home networks where large-scale security systems are not practical.

Overall, the project achieved its stated objectives and contributed meaningfully to cybersecurity research by providing a clear, working model of how honeypots can be developed and used to study network intrusion attempts.

5.4 Recommendations for Future Work

While this project successfully developed a functional honeypot system for monitoring and logging unauthorized access, there are several areas that can be improved or expanded upon in future research.

1. **Multi-Port Monitoring:**

   The current system listens on a single port. Future versions could be extended to monitor multiple ports simultaneously, providing broader visibility into network activities and capturing a wider range of intrusion attempts.

2. **Real-Time Alerts:**

   Adding an alert system that notifies the administrator through email, SMS, or a simple dashboard whenever a new connection is detected would make the honeypot more responsive and practical for active use.

3. **Database Integration:**

   Instead of writing logs to a text file, future implementations could store captured data in a structured database. This would make it easier to search, analyse, and visualize patterns in attacker behaviour.

4. **Automated Threat Analysis:**

   Integrating machine learning or simple heuristic analysis could allow the honeypot to automatically classify suspicious activities or identify recurring attack patterns.

5. **Deployment on Cloud or IoT Environments:**

   The system could be adapted for deployment on cloud servers or Internet of Things (IoT) devices to study attacks targeting those platforms, which are increasingly common today.

6. **Enhanced User Interface:**

   A simple graphical interface could be developed to allow users to monitor logs, view connection statistics, and manage settings without editing code manually.

In summary, future work should aim at making the honeypot system more intelligent, scalable, and user-friendly while maintaining its lightweight design. These improvements would enhance its usefulness both as a research tool and as a practical defence mechanism in real-world network environments.

REFERENCES

Antunes, N., Vieira, M., & Madeira, H. (2013). Using honeypots for teaching secure coding and intrusion detection. Proceedings of the IEEE Frontiers in Education Conference (FIE), 1183–1189.

Bailey, M., Cooke, E., Jahanian, F., Xu, Y., & Karir, M. (2009). A survey of honeypots and honeynets: Issues, challenges, and research opportunities. Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security (CATCH), 370–378. IEEE.

Koetter, M. (2010). Dionaea: A malware capturing honeypot. The Honeynet Project. Retrieved from https://www.honeynet.org

Pa, Y., Baldini, G., & Steri, G. (2021). SemanteCT IoT Honeypot: Using semantic web technologies to enhance IoT threat detection. Journal of Information Security and Applications, 58, 102–111.

Provos, N. (2004). A virtual honeypot framework. Proceedings of the 13th USENIX Security Symposium, 1–14.

Spitzner, L. (2003). Honeypots: Tracking hackers. Addison-Wesley Professional.

Stoll, C. (1989). The Cuckoo's Egg: Tracking a spy through the maze of computer espionage. Doubleday.

Telekom Security. (2018). T-Pot: The all-in-one honeypot platform. Deutsche Telekom AG. Retrieved from https://github.com/telekom-security/tpotce

Vanhoenshoven, F., Pa, Y., & Baldini, G. (2022). Semantic deception in IoT honeypots: Enhancing realism through context-aware interactions. Computers & Security, 112, 102538.

Zhu, Z., Lu, W., & Yu, H. (2015). Research and application of honeypot technology in network security. International Journal of Computer Applications, 115(14), 23–28.