DE MONTFORT UNIVERSITY LEICESTER

De Montfort University

# Honeypot for Defence of a Local Network

First Deliverable

– Kira Leigh Sherriff

Development Project

Supervisor –

# Contents

# Literature Review

Word Count

2297

# Honeypot: Defence of Local Network

*Abstract —* This literature review covers the honeypots and their role in defending a local network from cyber-attacks. We will examine how honeypots may improve security by detecting and being a decoy to real network infrastructure. We will also slightly touch upon the diverse array of honeypots that are available. The advantages and disadvantages of deploying a honeypot will be discussed to fully comprehend the impact that they may have on the security of the network. This literature will contain academic articles, websites, and statistics to provide accurate information to ensure a sufficient understanding can be achieved by the reader. This literature review will hopefully shed some light on how honeypots can safeguard a local network.

***Keywords—Honeypot, Defense, Network Security, Detection***

## Introduction

A honeypot is a cyber security tool that is used to deceive an attacker. This deception is done by emulating applications, operating systems, and even a whole network. This will trick the attacker into targeting the honeypot and hopefully be detected before the critical infrastructure is affected. However, honeypots themselves do not provide security mechanisms to defend a network. They do so through diversion, to keep the attacker distracted while the IT team try to solve the security problem.

There is a diverse array of honeypots that can be adapted to everyone's needs. The versatile capability of a honeypot makes it a powerful tool. Honeypots can be used to mimic software to capture malware or to detect bots. The intelligence gathered from the honeypot can help understand emerging threats, and it can act as an early warning system. (Ashish Girdhar, 2012). The number of global cyber-attacks is increasing each year, such as in 2022 with 500 organizations experiencing a data breach (IBM, 2022), compared to 2023 with over 550 organizations. The average cost of a data breach has also increased by 15% over the past 3 years (IBM, 2023). It is important to take steps to help mitigate the effects of a cyber-attack. A honeypot provides some mitigation within a network to divert a hacker away from a critical system and act as an early warning system, which could reduce the cost of an attack. However, for a honeypot to have an effect it must be effective by being properly implemented.

## Main Body

### Types of Honeypots

There are two main types of honeypots; production and research; which have two very different tasks. These honeypots are separated by their various levels of interaction which is how much access an attacker has to the honeypot. A production honeypot is typically used in an organisation as an early warning system and to identify attack patterns exposing vulnerabilities. Production honeypots are usually low interactive meaning full access cannot be gained. Low interactive honeypots also collect less information about an attack, though the data provided can still help build better defences and countermeasures. Though is always good practice to implement other security tools examples being firewalls, IDS, and sufficient security policies. (Iyatiti Mokube, 2007)

A research honeypot is designed to gain as much information as possible about the attack surface. They are used to gather intelligence on threats that any organisation may be exposed to. A research honeypot's main aim is to understand the attacker's motives and improve detection, prevention, and response. This will in turn improve security defences. Research honeypots are typically high-interactive honeypots, these honeypots provide an attacker with a real operating system to interact with, and no action is restricted by the honeypot. Research honeypots are usually exposed to the internet which comes with inherent risks. These risks are associated with a high-interactive honeypot as it allows the attacker to compromise the honeypot. If the honeypot is not secured correctly, it could open a door for the attacker to gain true access to a real operational system that is vulnerable. (Iyatiti Mokube, 2007)

## What Makes an Effective Honeypot?

Three key features are vital to having an effective honeypot, those are sensitivity, countermeasures, and stealth. Sensitivity increases the efficiency of capturing data from various attacks. Countermeasures mean that the honeypot will respond to an attack instead of being a victim. This idea of not being a true victim is essential as they could be a gateway to further attacks if they are not configured correctly. When a honeypot is attacked, it should not lead to further damage to other systems on the network. (Yogendra Kumar Jain, 2011) The last feature is stealth, the honeypot should conceal its functions, evading a hacker from knowing its true nature. (Ashish Girdhar, 2012). A honeypot's effectiveness is based on its ability to capture good-quality data that could be used to help mitigate future attacks. A honeypot should identify attacks and should also identify attacks that have yet to be found. When a honeypot is integrated with an intrusion detection system it can improve the efficiency of the honeypot detection mechanism by lowering the chances of false positives (Babak Khosravifar, 2008). An intrusion detection system is a monitoring tool for network security which holds a database of known malicious vulnerabilities. An intrusion detection system will monitor network traffic, and any matching signature from the IDS database will send an alert to the admins. (IBM, n.d)

## How They Defend a Network

Honeypots that are deployed within a network as part of the security infrastructure can help validate an intrusion detection system and provide an early warning system for an attack. The detection system of a honeypot provides real-time defence. The honeypot can be monitored to spot anomalies from normal network traffic. Moreover, deploying a honeypot within a network can give insight into an attacker's methods. This insight allows an IT technical team to implement security measures and policies based on the threat landscape. (Yogendra Kumar Jain, 2011) This early warning system can help detect port scanning, which is part of the reconnaissance stage in Lockheed Martin cybers kill chain, Reconnaissance is done when an attacker is preparing an attack. (eccouncil, 2022) This can aid in putting in mitigation techniques based on the vulnerability that was exploited. In the event of an attack having a honeypot collecting data can help in incident response, by finding out the motives, and the vulnerabilities exploited.

The early detection function of honeypots combined with other traditional security intrusion techniques can increase the probability that a sophisticated attack will be detected earlier. An advanced persistent threat is one of these attacks. An APT is a well-planned and organised attack that aims to have long-term access to a network. The targets of an APT are typically large enterprises and governments, with the goal of the attacker varying from target to target. (imperva, n.d) A research paper looked at how a honeypot can defend a network from an APT. This is done through the detection of incoming traffic to the honeypot, which will alert an administrator of the event. In the paper is mentioned that if a low interactive honeypot was used APT's such as Stuxnet and Duqu could have been detected earlier. (Saud, 2015) In the case of Stuxnet, the Worm the targeted IR-1 centrifuges could have been detected earlier. If there had been a honeypot acting as a PLC that would have been probed by the worm. (Chrissikopoulos, 2014) With Duqu the main aim was to gather information on a target. Again, if a honeypot was deployed it could have detected the behaviour of the Duqu connecting to the command-and-control server, mitigating the risk. (Chrissikopoulos, 2014)

Honeypots can also be used to identify zero-day exploits through the isolation of malicious network traffic. A zero-day exploit is a vulnerability that is unknown within a software or protocol that was not discovered until an attack occurred. (IBM, n.d) In 2021 the number of recorded zero-days was 66 which was 55% more than the previous year. (purplesec, 2023) A zero-day exploit is more severe than a known vulnerability. This is due to its unknown nature and is harder to detect, with no hash signature available within an intrusion detection systems database. However, as we have also seen Stuxnet and Duqu both used Zero-Day exploits to achieve their goal and research has shown that a honeypot could have detected the attack (Chrissikopoulos, 2014). Further research has shown that a honeypot can be effective when identifying zero days. Unlike traditional intrusion detection systems that use signature-based detection, a honeypot may detect malicious traffic that an IDS could not detect. Honeypots are a powerful tool for detecting zero days by capturing malicious traffic when the zero-day has encountered the honeypot. The captured traffic will be analysed, and it could be used to generate an attack signature that then could be used in intrusion detection systems. (Constantin Musca, 2013).

A honeypot can also help improve the false positive rate in the detection of an event. A false positive is an event that has been incorrectly classified as a positive. This is when a detection system flags that a malicious event has occurred when nothing has happened. A false positive is dangerous as it reduces the credibility of the system and wastes time, money, and resources. Research has shown that if a hybrid approach is taken by combining signature detection and anomaly detection with a honeypot, then the detection rate can be improved. In the research, Pragya Jain conducted an experiment that sent 11234 packets and out of those 7342 were malicious. When only using signature detection, it achieved a detection rate of 32% and anomaly detection with a rate of 34%. The hybrid approach with the honeypot had a detection rate of 81% with a 4% false positive, comparing this to only a hybrid of signature and anomaly detection at a rate of 61% with a 6% false positive. (Pragya Jain, 2012) Babak Khosravifar also investigated reducing false positives by using a honeypot and intrusion detection systems. They also came to the same conclusion that combining both intrusion detection systems and a honeypot can reduce false positives and lead to better performance. (Babak Khosravifar, 2008) As seen in the research a hybrid approach cannot only reduce the false positive rate but also increase the overall detection rate of malicious activity.

## Advantages

As discussed, a honeypot can help defend a network through detection. When a honeypot is paired with other traditional intrusion detection systems it reduces the number of false positives. (Babak Khosravifar, 2008)A honeypot could also distract an attacker from a critical system. When attacks occur, the information captured can give insight into a cyber-attack. This can help a cyber security technician uncover what happened. A honeypot would not overload the admin with a large amount of data but would still provide high-value information. (Abdulrazaq Almutairi, 2012) Compared to other security tools a honeypot can be considered lightweight. They have no complex algorithms, and depending on the type of honeypot may take up little computing resources. (Iyatiti Mokube, 2007). Since honeypots require very limited resources they can easily be implanted on old underpowered hardware, which has the advantage of being very cheap and needs little maintenance.

### Disadvantages

There are risks associated with the use of honeypots. A honeypot could be used as a staging ground for an attack against an organization, which could compromise the network. As stated, a honeypot will only work if an attacker interacts with the honeypot. This limited vision could have consequences. (Abdulrazaq Almutairi, 2012) If an attacker does not interact with the honeypot, it will not be able to detect an intrusion and the job will be left to the traditional intrusion detection systems. If a honeypot is not stealthy as it should be an attacker may be able to fingerprint that honeypot. Fingerprinting is when an attacker can identify the true nature of the honeypot, without detection. (Iyatiti Mokube, 2007) Due to this, an attacker will not interact with the honeypot and will not be detected by it, this leaves the local network more vulnerable to risks.

## Limitations and Future Research

Further research could be done on how an AI-powered honeypot could help improve efficiency. With AI, it may be possible to create a dynamic honeypot that could shift to fit into its environment and adapt on its own based on the current network traffic. A honeypot of this nature could be more realistic and have much more functionality such as being able to detect deep fakes and other deception techniques. AI could automatically check the data and create predictions on what the attacker's goal may have been, along with the methods used. This could further enhance threat intelligence. AI-powered honeypots could increase the detection rate of APT, and Zero-days compared to a normal honeypot. There is research about machine learning honeypots being trained on data, combining this with AI could make honeypots a very powerful tool for the defence of a network.

## Conclusion

In conclusion, a honeypot can be an effective way of defending a local network. They are versatile and can be tailed for different purposes for research or production for protection. They play a crucial part in diverting and detecting a potential threat. A honeypot collects data from the attacks which can then be used to find vulnerabilities within the local network, based on the attack pattern. However, a honeypot must not be the only defence mechanism as they are not a definitive security solution. The most effective honeypot is combined with traditional security mechanisms such as a firewall, intrusion detection systems, and intrusion prevention. Combining all these cyber security tools can result in a better detection rate of 81% and decrease the false positive rate to 4%. (Pragya Jain, 2012)A honeypot does have its limitations as it requires an attacker to interact with the honeypot to be detected. Though overall, a honeypot is a cheap way to enhance the security of a local network, by combining it with existing security tools.

# Functional Requirements

Word Count
1047

## Introduction

The honeypot will require multiple functions for it to work effectively and securely. These requirements are necessary to ensure the honeypot's proper function. Proper function in this project is defined as secure, and easy-to-use software, while also having the capability of safeguarding the integrity of the network and data. This part of the document will identify the functional requirements.

## Honeypot Overview

To understand the requirements, the aim of the project needs to be identified first. The honeypot will mimic a computer offering services, this will attract a hacker away from a real system. The honeypot will operate on well-known ports from 0-1023 assigned by the IANA. The focus on well-known ports is due to services that lie behind them, and their ability to draw an attacker. When it comes to the most attacked ports Secure Shell (SSH) is high on the list, as a remote protocol that can execute commands, modify files, and change configurations, this is very desirable to an attacker. For this reason, SSH will be a developed port on the honeypot containing a vulnerable service. When the honeypot is inevitably attacked logs are collected from the hackers' attempts. The logs can provide insight into the hackers' goals while also giving mitigation techniques. While a log file is useful it is also vital to know when a hacker is in the network, for this reason, an alert system within the honeypot will need to be set up. The logs will be stored in a database so they can be displayed on the web interface.  The alert system will send a message when an attempt on the honeypot is made, this can increase the response time to an attack. To control the honeypot, a web interface should be provided that allows the owner to close/open the port, this will also be accompanied by an Uncomplicated Firewall (UFW), on the implanted device to provide more security.

## Users

There are two classes of users these are direct and indirect. A direct user is an administrator that has control over the whole honeypot and directly interacts with it, by changing settings and viewing the logs in the database. The indirect users are interacting with the honeypot but are not directly impacting the function, or altering it. These are the attacker who can interact with the honeypot and "use" it through port access but nothing more.

### Direct User

*Administrator:*

The administrator has control over the whole honeypot, they can manage the setting and change the way that the attacker will interact with the honeypot by closing and opening ports. The administrator can see all the log files and interact with the web application freely. They Should be the only ones with full access, the access should be secure and not be interrupted by the attacker.

### Indirect User

*Attacker:*

The attacker has been classed as an Indirect user; they are not direct users like the admin, but the honeypot is still providing them with a physical port. The indirect user has no control over how they interact with the honeypot and should not know they are interacting with it. The reason the attacker has been classed as a user is because they are the main target, they contribute data and intelligence to the honeypot. They will only have access to the physical ports and should not be able to access the web application or any other function that must be kept secure.

## Use Case Diagram

A case diagram can summarise the direct actors of the system, a honeypot has two actors who interact with the system, the attacker, and the administrator. The attacker and administrator must have a completely different view of the honeypot and how they will interact with it. An attacker should only have access to the front end of a honeypot and should not be able to access the log files of recorded events. The administrator should have a full view of the honeypot and have complete control over it. The user case diagram will help see the scope of the system and how each of the users should interact with the honeypot.

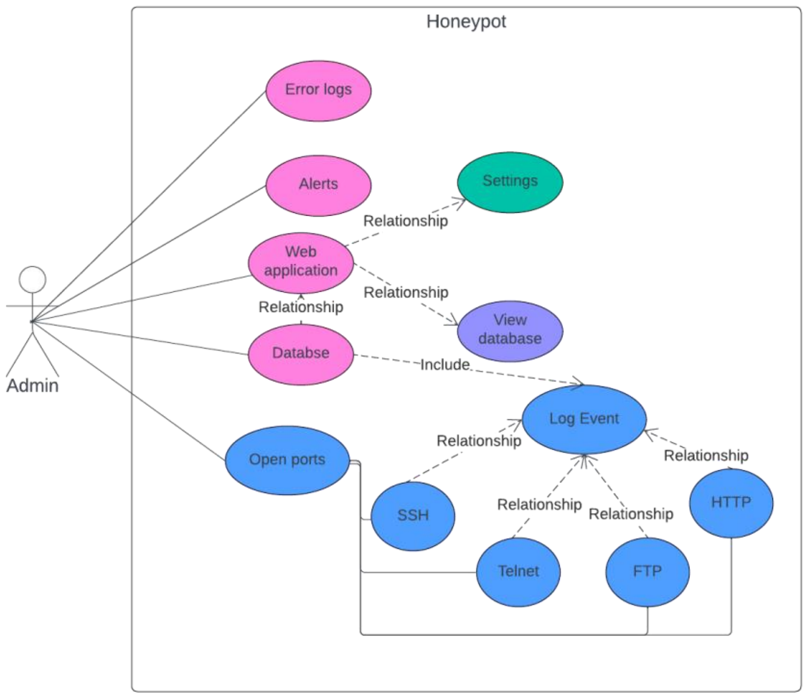


*Figure 1:Use Case Admin*

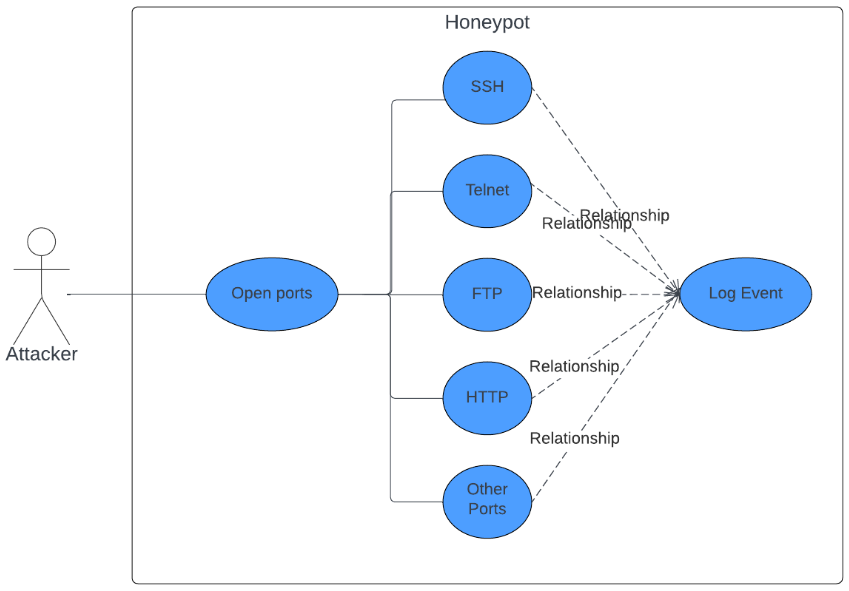


*Figure 2: User Case Attacker*

## Functional Requirement

The Functional requirement is necessary for the honeypot to work, some of the requirements are dependent on others, so those requirements will have to be completed first. One example is the connection to the honeypot. This should be done first as the whole project depends on this function.

| **ID:** | **HYP01** |
|---|---|
| **Title:** | Honeypot Accepting Connection |
| **Description:** | Accept the well-known ports, with TCP only, allow multiple connections at once, and should be a threaded program. |
| **Dependencies:** | N/A |
| **Resources:** | IDE, GitHub, Desktop/Laptop, Python library. |
| **Actor:** | Hacker/Admin |
| **Priority:** | High |
| **Notes:** | Should only allow TCP and IPv4 |

*Table 1*

| **ID:** | **HYP02** |
|---|---|
| **Title:** | Honeypot SSH Development |
| **Description:** | SSH development, allows an attacker to log into SSH and can execute commands. This honeypot will be vulnerable, and easy to exploit. |
| **Dependencies:** | HYP01 |
| **Resources:** | IDE, GitHub, Desktop/Laptop, Python library. |
| **Actor:** | Hacker/ Admin |
| **Priority:** | High |
| **Notes:** | Should have username and password which can be brute forced |

*Table 2*

| **ID:** | **HYP03** |
|---|---|
| **Title:** | Honeypot HTTP Development |
| **Description:** | HTTP development allows an attacker to attack a vulnerable web app. |
| **Dependencies:** | HYP01 |
| **Resources:** | IDE, GitHub, Desktop/Laptop, Python library. |
| **Actor:** | Hacker/ Admin |
| **Priority:** | Medium |
| **Notes:** | Should only monitor for login attempts |

*Table 3*

| **ID:** | **HYP04** |
|---|---|
| **Title:** | Honeypot Telnet Development |
| **Description:** | Telnet development, to study login attempts |
| **Dependencies:** | N/A |
| **Resources:** | IDE, GitHub, Desktop/Laptop, Python library. |
| **Actor:** | Hacker/ Admin |
| **Priority:** | Medium |
| **Notes:** | Should only monitor for login attempts |

*Table 4*

| ID: | HYP05 |
|---|---|
| Title: | Honeypot FTP Development |
| Description: | FTP development, to study login attempts |
| Dependencies: | HYP01 |
| Resources: | IDE, GitHub, Desktop/Laptop, Python library. |
| Actor: | Hacker/ Admin |
| Priority: | High |
| Notes: | Should only monitor for login attempts |

*Table 5*

| ID: | HYP06 |
|---|---|
| Title: | Honeypot Alert |
| Description: | An alert system sends a message when an attempt against a port is made, this alert could be sent by email or on the web application. |
| Dependencies: | HYP01, HYP02, HYP03, HYP04, HYP05 |
| Resources: | IDE, GitHub, Desktop/Laptop, Python library, account to send alerts too |
| Actor: | Admin |
| Priority: | High |
| Notes: | Should be alerted through email or phone number |

*Table 6*

| ID: | HYP07 |
|---|---|
| Title: | Honeypot Log Capture |
| Description: | Capturing attempts made to connect to the port, the data, and time. |
| Dependencies: | HYP01, HYP011 |
| Resources: | IDE, GitHub, Desktop/Laptop, Python library, MySQL |
| Actor: | Admin |
| Priority: | High |
| Notes: | N/A |

*Table 7*

| ID: | HYP08 |
|---|---|
| Title: | Login Functions |
| Description: | User should have a username and password to login to the web page for security |
| Dependencies: | N/A |
| Resources: | IDE, GitHub, Desktop/Laptop, PHP, HTML, Apache. |
| Actor: | Admin |
| Priority: | Medium |
| Notes: | N/A |

*Table 5*

| ID: | HYP09 |
|---|---|
| Title: | Honeypot Web Application Port Control |
| Description: | Honeypot must successfully allow connection before the control system can be set up |
| Dependencies: | HYP08 |
| Resources: | IDE, GitHub, Desktop/Laptop, PHP, HTML, Apache. |
| Actor: | Admin |
| Priority: | Medium |
| Notes: | This should be created after the login page |

*Table 6*

| **ID:** | **HYP10** |
|---|---|
| **Title:** | Honeypot Web Application change user settings |
| **Description:** | Users should be able to change email/phone numbers for the alerts, through the web application. |
| **Dependencies:** | HYP08 |
| **Resources:** | IDE, GitHub, Desktop/Laptop, PHP, HTML, Apache. |
| **Actor:** | Admin |
| **Priority:** | Medium |
| **Notes:** | This should be created after the login page |

| **ID:** | **HYP11** |
|---|---|
| **Title:** | Honeypot Web Application Log View |
| **Description:** | To view all the data that is stored in the database. |
| **Dependencies:** | HYP12 |
| **Resources:** | IDE, GitHub, Desktop/Laptop, PHP, HTML, Apache |
| **Actor:** | Admin |
| **Priority:** | Medium |
| **Notes:** | Should be created after the login page |

*Table 7*

| **ID:** | **HYP12** |
|---|---|
| **Title:** | Database |
| **Description:** | To store the honeypot logs for long-term access |
| **Dependencies:** | N/A |
| **Resources:** | IDE, GitHub, Desktop/Laptop, Python library, MySQL |
| **Actor:** | Admin |
| **Priority:** | High |
| **Notes:** | Should automatically create without user input |

*Table 8*

| **ID:** | **HYP13** |
|---|---|
| **Title:** | Configuring UFW |
| **Description:** | To have port control, to open and close, as UFW is an easy way to configure the iptables for IPV4. |
| **Dependencies:** | HYP* |
| **Resources:** | IDE, GitHub, Desktop/Laptop, Python library, Device implementation. |
| **Actor:** | Admin |
| **Priority:** | Low |
| **Notes:** | Configuring UFW can only be done when the honeypot is implemented on the device preferably a Linux machine. |

*Table 9*

## Non-Functional Requirements

Non-functional requirements are defined as how the honeypot system should behave; these are indirect to the functionalities but focus on how the function affects the system. The system can still work if none of these requirements are met, but if they are it will make the user experience better and the honeypot more efficient.

Stealth:

The honeypot should contain the element of stealth. The attacker should not know that it is a honeypot and should try to interact with it as if it were a normal device. stealth is critical to the success of a honeypot; if this does not contain this element the honeypot will not be able to deceive the attacker.

Security:

A honeypot by nature must be vulnerable to being attacked but must not be a victim. The vulnerability of the honeypot if not correctly secured could lead to further breaches in security. The attacker should not be able to launch an attack from the honeypot to compromise other systems. The code for the honeypot will follow good practices to ensure the security of the honeypot.

Scalability:

The honeypot should be able to handle heavy or light amounts of internet traffic. Its resources should not limit it, nor should the traffic volume affect its performance at logging into the database.

Customization:

The honeypot should have some level of customization, that allows the user to tailor the honeypot to their needs. When the honeypot is customized by the user the performance and functionality should not be affected.

Reliable:

The honeypot should be reliable at detecting those who interact with it and should correctly log the events that have occurred. It should not crash and should have error-handling abilities and an event log if errors with the honeypot occur.

Ease of Use:

The web interface of the honeypot should be user-friendly and inclusive of those with some form of visual impairment.

# System Design

Word Count
2129

## System Architecture Overview

The program will be written in 2 different languages Python3 and PHP. The program itself should work on both Windows and Linux operating systems. Each aspect of the program will have its separate folder, database, config, web app, and honeypot, doing this will allow for better management of the code. In the web app folder, there will be a folder that will hold all the HTML files, to be used as templates for the PHP code. Each file will contain an __init__.py file that will create packages. This is done so the files can access other files in another folder. The program will be threaded to allow for multiple connections at once, improving the efficiency of the honeypot. In the honeypot, sockets will only specify the use of TCP and IPv4. TCP is the only transport protocol used as UDP is a connectionless protocol making it harder to detect activity on the ports. The library Paramiko will be used to implement an SSH that provides a server to establish a connection. (Forcier, n.d) Paramiko will be used to build the SSH server from the bottom up so that it's fully interactive with the attacker. Since SSH uses an RSA key file it will need to be generated for the code, so it doesn't generate a new key each time it runs. This is important as the SSH client will detect an RSA key change from the known host file and flag a warning about a man-in-the-middle attack. On top of SSH other protocols will be developed to make the honeypot more believable for the attacker, but not in as much depth as SSH. Another library that will be used for the program is Flask, which will handle the web application of the honeypot, to capture incoming credentials. (Flask, 2010) The web application for the admin will manage the honeypot. The web application will contain a login page to keep the data secure that will be displayed.

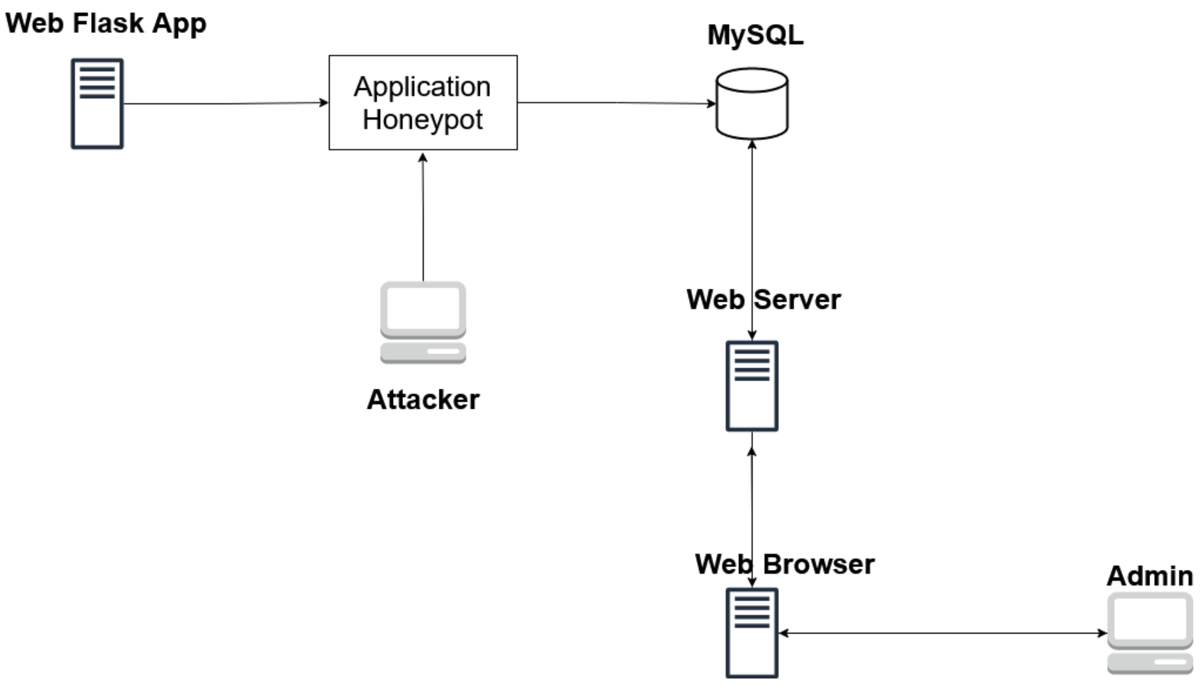


*Figure 3: System Architecture Overview*

## UML Class Diagram

A UML class diagram designs the structure of that code, what classes are needed, their attributes and functions, and the relationship with other classes if they are dependent on each other or not. The attributes contain their data type to understand what data is needed. A UML diagram provides insight into what is needed for the system to function, by giving an overview which can help when it comes to its implementation of the code. (lucidchart, n.d)

In the UML diagram, the code is dependent on the ServerPort class, which is used to create the listening port, which the client can connect to. The services SSH, Telnet, HTTP, and FTP, are all dependent on this class. Another heavy dependency is the database manager class which is how the program creates the database and its tables, without this class information will not be logged into the database. Most of the attributes in the class are private, the only attributes which are public are the Client_IP, client_port and database login information. The username and password have been used multiple times, and all the classes are private as each service needs its credentials. Having them public would cause a conflict within the code.

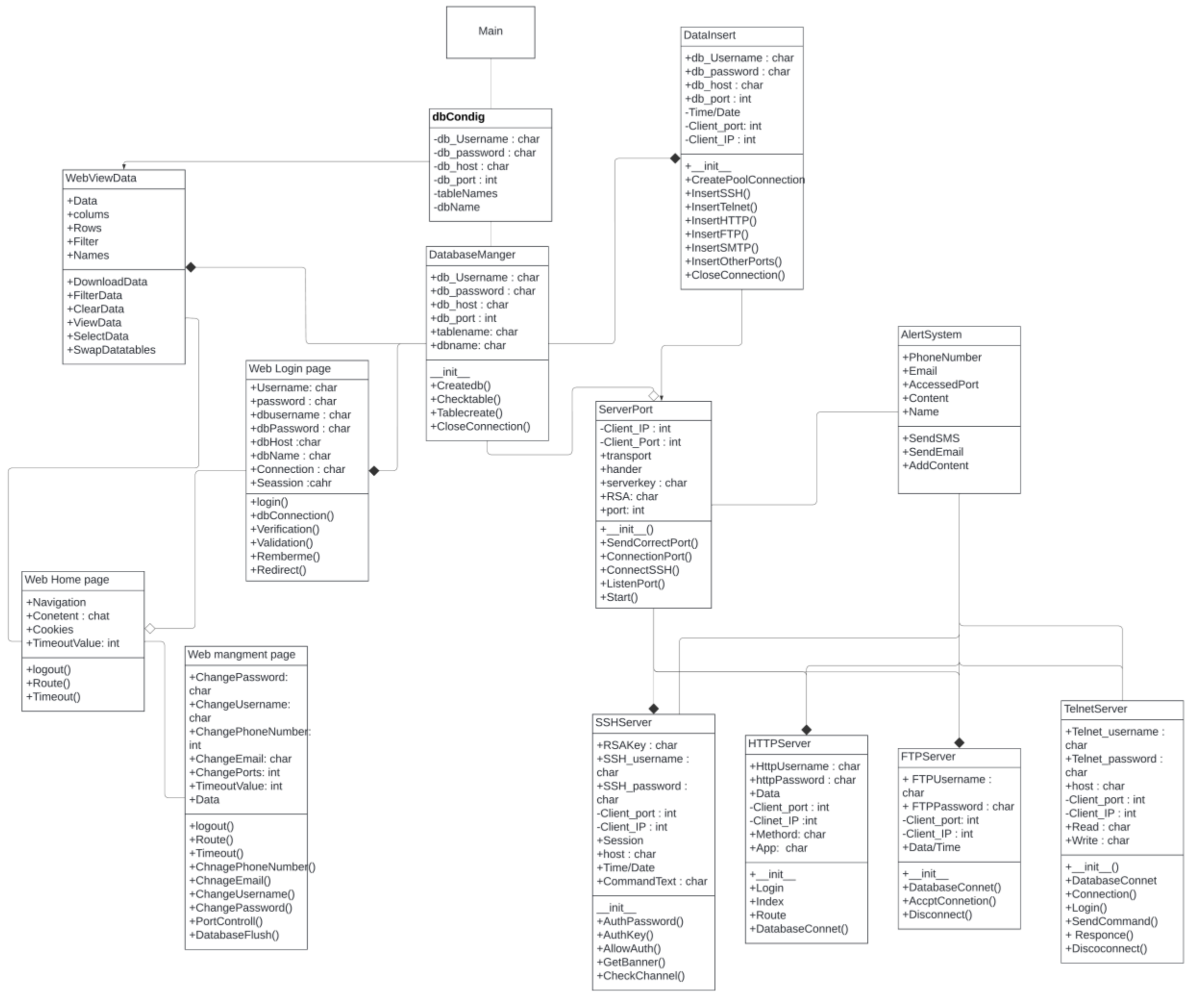


*Figure 4: UML Honeypot Class*

## Local Network Design

The local network used in this project will be a basic home network set-up, which contains both wireless and wired connections, all the internet traffic comes and goes to the same gateway at 192.168.1.254. The local network contains a Raspberry Pi Zero W which is running Pi-VPN with wireguard, which allows for a certain device with the matching wire guard client to connect to the network. The Raspberry Pi Zero W also contains a local DNS. Within the local network, there is a home server which runs a private cloud server that can be accessed by all devices within the network. I own a domain that is used for the cloud server, this is what provides the SSL certificate for the web application. The server also is running SSH for remote configuration, the honeypots' goal is to distract an attacker from the server.

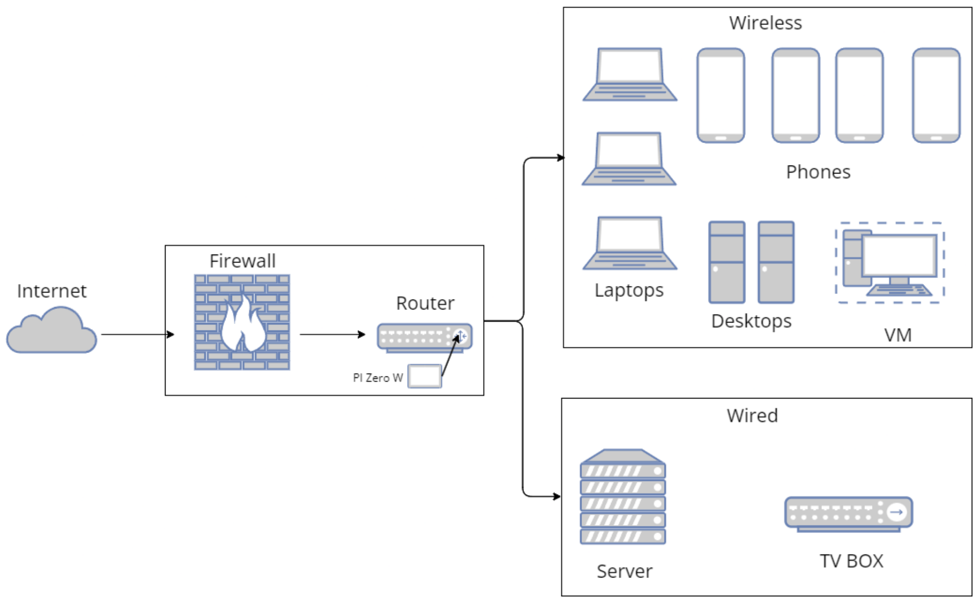


*Figure 5: Local Network*

## Network

The honeypot works by network protocols, to establish a connection with TCP with ports in the range of 0-1024. Network architecture is necessary for this project as it's the backbone of the honeypot. The Honeypot will only accept TCP segments as UDP is not suitable for detection as it is a connectionless protocol. Transmission control protocol (TCP) is part of the transport layer in the TCP/IP model that handles the communication to ensure the delivery of segments. TCP is a connection-orientated protocol meaning that a stable connection must be established for transmission. The connection-oriented state of TCP is for handling data recovery if a segment is not received by the device. Error detection is done with a sequence number which is 32 bits long, to keep track of how much data is sent. The sequence number is added to each segment that is being transmitted. When a segment is sent the receiver will send an ACK number to inform the sender that the segment has been received successfully. (W. Eddy, 2022)

A TCP connection is established with a series of handshakes in a three-step conversation between two devices that wish to communicate. In the case of the project, the attacker will start the conversation with the honeypot by sending a SYN packet short for synchronization. The honeypot will reply with an SYN-ACK packet to confirm that it has received the request for communication, the attacker machine will then send an ACK packet to acknowledge and complete the three-way handshake. When the three-way handshake is completed the connection information will be stored in the database. Once the connection information is logged the connection is terminated with a FIN ACK

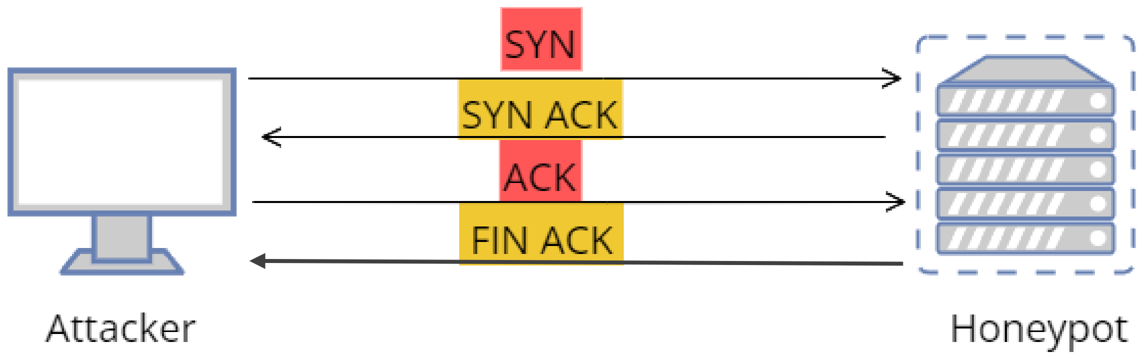


*Figure 6: TCP Connection*

## Web Application

A web interface will be provided with the honeypot which will have multiple functional purposes. The web application will also need to be on another port that is not 80 or 443, as these ports will be used by the honeypot. Since there is a local DNS on the network the web application can be encrypted with SSL to improve security. The SSL can be created with a DNS challenge meaning no forwarding is needed. A web server will be needed to interpret the PHP code so that it can be displayed on the web browser. The web server used for this project will be Apache2 as it is fast, secure, and highly reliable. A web application is easy to use for the administrator, so they can easily manage the settings of the honeypot without the code needing to be modified. The web interface will of course be connected to the back end of the program which will gather the information from the database. Each function associated with the web interface will have its web page for better management of information, so the user is not overloaded with data.

The web application itself should be able to handle errors in response to HTTP requests, such as 400 and 500 error codes. If an error does occur during the runtime of the web application, it should be able to catch it so that no information is leaked to the user. The web application should not allow a user who does not have an open session to view data on the web application. This is done to keep the administrative web application secure from any potential attacker in the network. For the security of the database, all SQL queries should be prepared in the code to prevent SQL injections. Along with safeguarding from SQL Injections, preparing the statements can help with lessening the overhead. (Oracle, n.d)

### Web Login Page

The web application will contain a login page for the security of the application. However, before the login page can be accessed by the user, they must register an account. Due to security reasons, the register page should only appear once, this should be when the database is empty of login credentials. If a login credential does exist, then the user should be directed to the login page. When the login is successful the web server will redirect the user to the main dashboard where they can navigate through the web application. The login page will contain validation and sanitation to prevent SQL injections to the database so an attacker will not gain access to sensitive information. The user should be forced to enter both username and password for registering an account, the password length should also be equal to or greater than 8. It should also have a CSRF token to prevent unwanted execution in the web application. A CSRF token can also prevent the brute force of the web application. There should also be some kind of punishment for getting an incorrect password. An example is not being able to enter a password for 30 seconds after 5 incorrect passwords. There will only be one HTML page for both the login and the register functions. The

HTML code will be dynamic so it can update and relay information to the user when certain criteria are met. For example, if no users are found in the database on the form the title will be "register", if accounts are found it will be "Honeypot Login". This will also be used to alter the information if they have entered the wrong login or used an incorrect password format.

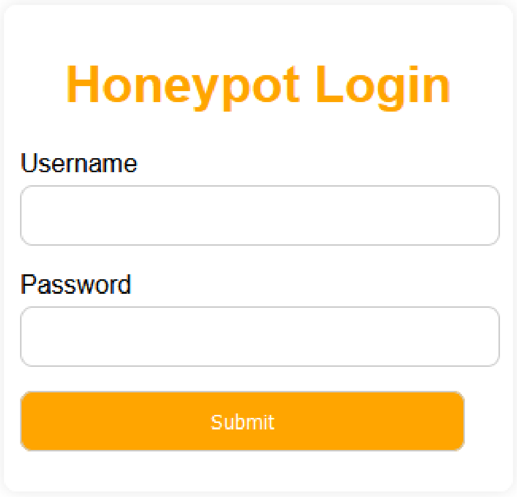


*Figure7: Login Page*

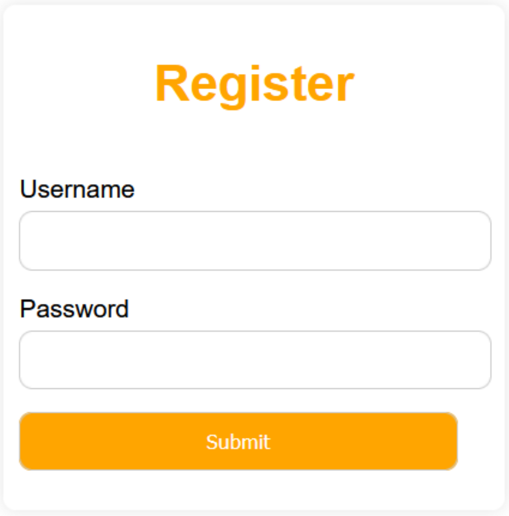


*Figure 8: Register page.*

## Management Page

Within the management page, there should be a way for the user to change the login settings such as username and password. With the alert system, an email address and phone number are needed and there should be a way to modify and add this information. Within the management settings, the ports can be controlled. This input from the user will need to be directed to the back end of the honeypot within the Python code. The user should be able to open and close all ports with a single button. There should also be a button for each of the services, so if the user wants only SSH to be open they will have the choice. Along with the button, the user should have the ability to input a range of port numbers to be opened. This functionality of opening and closing the ports allows the user to have more control to fit the honeypot to their needs. There should also be a single button to flush all the database tables, though this should exclude the web accounts. Since the settings page has different functions, the UI needs to be created to ensure that the user does not get confused and click on the wrong button. It will be good practice to have a notification to alert the user when a serious change takes place such as flushing the database.

## Database

The database used for the program will be MySQL. MySQL database is used as it can be scaled to demand with high performance and security. Using a database can help with storing the data long term. The database login information should be modifiable so the administrator does not need to store the data on the local machine, instead can be stored in a centralised database. This use of a database can make the honeypot more flexible for the administrator's needs. The database information will be stored in a JSON file that can be modified by the user, which will allow for ease of use, so only one file needs to be modified to update MySQL login information.

There will be one database within the program, this database will contain all the tables needed to fully support the program. There will be multiple tables within the program, this is to ensure good management of the data. Some tables will contain the login credentials attempts for the protocols operating on the honeypot. There will be a table that will hold the connection information of the client, this will have the port number, IP address and the time and date it was accessed. The reason for the network information to be on different tables is for better management of data. If all protocols were on the same table, it would be hard to distinguish between them. SSH will need an extra table so the commands entered by the attacker will be correctly stored in the database. The last table needed is to store the web application account information of the user. The passwords must be hashed before entering the table.

### Database Relationships

The tables within the database will have limited relationships between them due to the complexity they are not easily linked together. An example is SSH, I want the Client_port and Client_ip to be viewed with the ssh_login, doing this with a foreign key could be complex, as the table for the connections will hold all port attempts. Also, SSH allows for multiple password attempts per connection. In a foreign key ID column within SSH 2 out of 3 login attempts will have NULL value. For simplicity, I believe having no relationship between the tables will be more beneficial for the program. It would also be simpler to view them separately in the web application.

### Entity Relationship Diagram

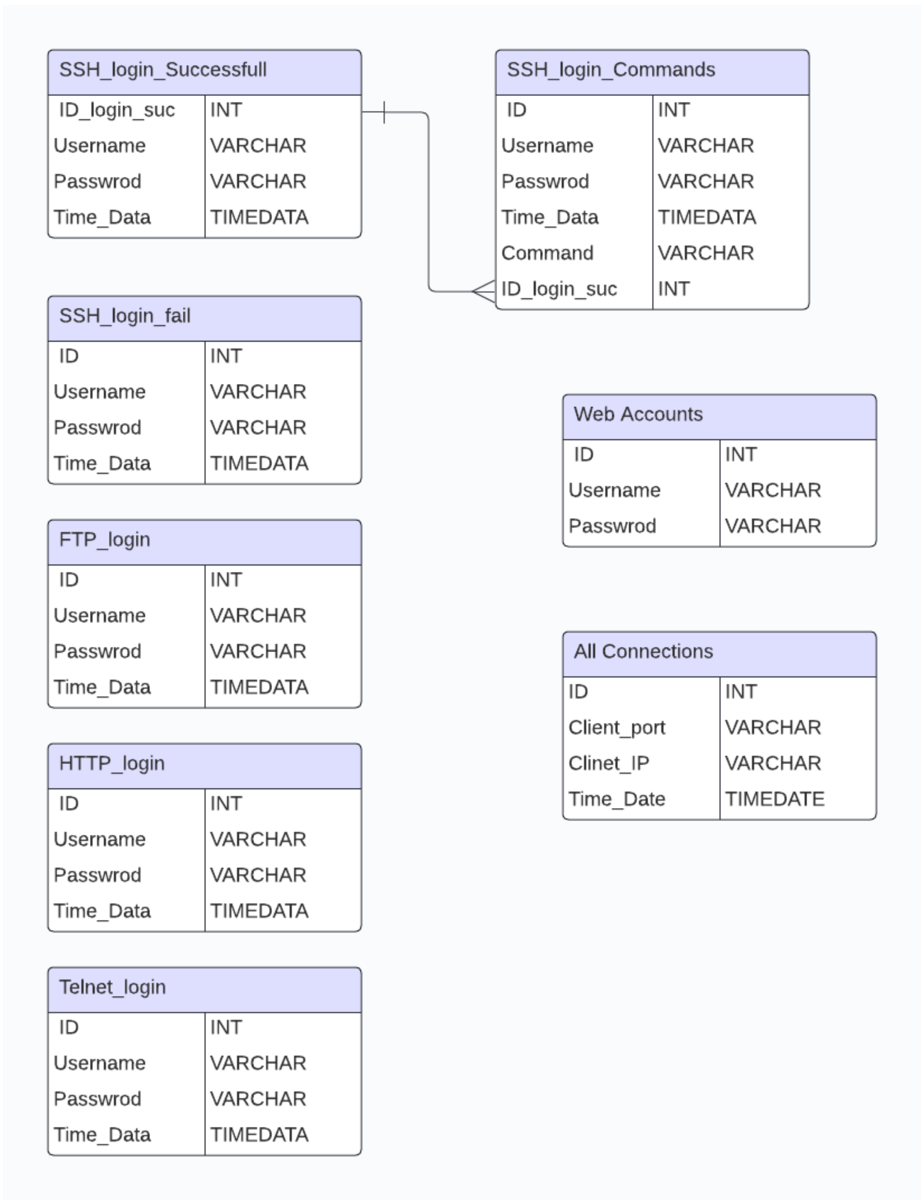


*Figure 7: ERD*

# Indicative Test Plan

Word Count
566

## Methodology

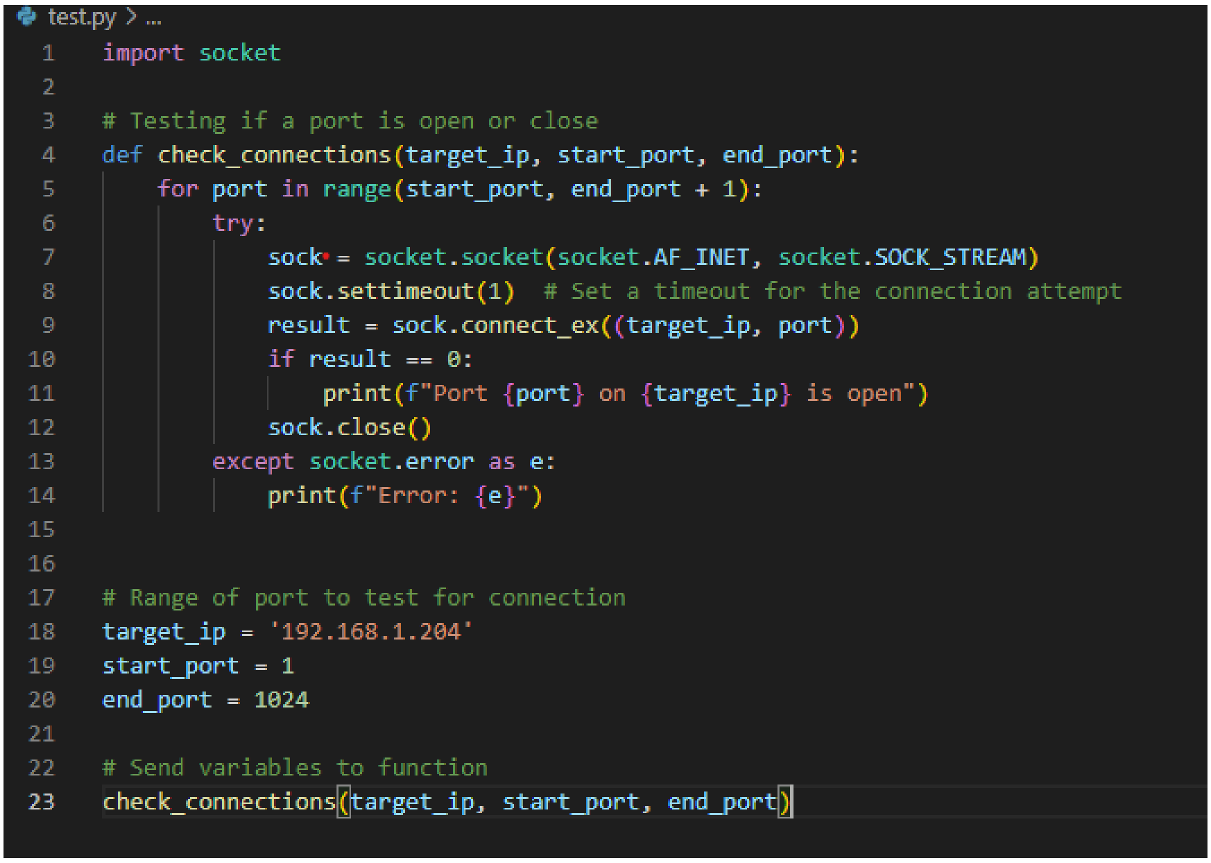
The project will follow the software development life cycle which is design -> Code -> Test. The methodology followed will be agile which is an iterative process. The agile methodology is more flexible than the traditional waterfall model. In agile testing can be done after every requirement, which can help better identify the errors within the code. This error could be a basic syntax error which will be picked up by the debugging tool or a code interpreter. A semantic error where the code has flaws in its logic can affect the output of the program. Semantic errors are harder to find and to fix. If semantic errors are identified early, it can be beneficial to the project development, as the errors will not negatively impact the future of the project.  The waterfall model does the testing when the whole project is complete, if an error does occur within the code, it will be much harder to identify compared to the agile approach. Though following the agile methodology can be more time-consuming than the waterfall models, the time dedicated to testing ensures that the requirements are being met to a good standard. The testing will also include non-functional elements of the project to ensure that the honeypot works effectively.

## Testing Strategy

Creating a testing strategy for the project will ensure that all the requirements are met to a good standard. The methodology that has been chosen is the agile model which allows for more freedom in the development of the project. Agile allows the requirements to change and for new ones to be added, which can have a positive impact on the development of the project.  There will also be a decision about how the honeypot will be tested. This would be to confirm the status of the ports that will need to be connected, which requires a client. The testing strategy will help improve the quality of the honeypot, ensuring that it meets its functional and non-functional requirements.

Testing the functionality of the honeypot requires a client that can interact with the honeypot. This system for testing will only work on the local host, in the development stage of the honeypot it won't be exposed to the network and will only be accessed by the local host. The client should be able to make any connection and will act as a potential attacker in the network. The client should be able to open multiple connections at once to test the threading capability of the program and its ability to handle different types of traffic volume.  When testing the results should not conform to any basis and could only be reported on how they behave and if they have done what is expected. A small amount of code has been made to check all ports to ensure that they are open. There will need to be some manual testing in the case of the web application and developed protocols, to ensure that all features are working correctly.

To test if the honeypot is accepting connections, I wrote a simple code that will check if the ports are open or closed. This program will also allow me to test if the honeypot can handle a large volume of traffic without crashing the MySQL server will all its insertions. The test case will detail all the requirements that should be met by the end of this module.



test.py > ...

```python
import socket

# Testing if a port is open or close
def check_connections(target_ip, start_port, end_port):
    for port in range(start_port, end_port + 1):
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(1)  # Set a timeout for the connection attempt
            result = sock.connect_ex((target_ip, port))
            if result == 0:
                print(f"Port {port} on {target_ip} is open")
            sock.close()
        except socket.error as e:
            print(f"Error: {e}")


# Range of port to test for connection
target_ip = '192.168.1.204'
start_port = 1
end_port = 1024

# Send variables to function
check_connections(target_ip, start_port, end_port)
```

*Figure 8: Test Code*

## *Testing Case for Full Program*

| Test ID | Expected results | Results | Fail/Pass |
|---|---|---|---|
| 01 | TCP should be allowed to connect. | All the ports between 1 – and 1023 have been opened and confirmed with a port scan | Pass |
| 02 | Should display when an attacker has attempted to connect with the port | The Port has displayed the attempts of port access on the terminal | Pass |
| 03 | Create the database and tables | When running the code should make the database a table, and should check if a table or database already exists | Pass |
| 04 | The log should successfully be sent to the database and tables | The log for connections is successfully being added to the database | Pass |
| 05 | Should get the login request of SSH | Can attempt to login into SSH | Pass |
| 06 | SSH should log the connection in the terminal | SSH has displayed the attempts of port access on the terminal | Pass |
| 07 | Should allow the user to enter commands to the SSH Root access should not be allowed | | |

| | | | |
|---|---|---|---|
| 08 | The SSH username and password attempts should be logged into the database | The username and password are being added to the database | Pass |
| 09 | The honeypot should alert, and attempts made to SSH | | |
| 10 | The SSH command attempts should be logged into the database | The Database stores the login attempts with the password and username, combined with the client's IP and port. | Pass |
| 11 | The attacker should be able to interact with an HTTP connection | | |
| 12 | The HTTP attempts should be logged into the database | | |
| 13 | Should alert an attempt made to HTTP | | |
| 14 | Should have some interactive capability with Telnet. | | |
| 15 | All-access attempts with Telnet should be logged into the database | | |
| 16 | Should alert an attempt made to Telnet | | |
| 17 | should allow a user to attempt to connect to FTP | | |
| 18 | Should log the attempt into its database | | |
| 19 | Should have an alert when an attempt is made | | |
| 20 | Register page | The user should be able to make an account for the website | Pass |
| 21 | Web applications should have a login page | The user can log into the web application page | Pass |
| 22 | Should also be logged out of an account | Allows the user to log out of an account. Once logged out they cannot access the home page | Pass |
| 23 | Should have a home page to navigate to the other web pages | The web application as a navigation bar works to direct the user. | Pass |
| 24 | The web page to manage honeypot settings should work | | |
| 25 | Should have the ability to open/close ports | | |
| 26 | The web page view log should be available for the user to view | | |
| 27 | The log page for SSH should be available | | |

| | | | |
|---|---|---|---|
| 28 | The log view for HTTP should be available | | |
| 29 | The log view for Telnet should be available | | |
| 30 | The log view for FTP should be available | | |
| 31 | All log views must contain a filter to search for a time a data | | |
| 32 | All logs must have a button to flush the database. | | |
| 33 | All logs should have a download button to download the log file | | |

Table 10: Test Case

# Implementation

Word Count
726

## Introduction

In the current state of the program, code has been written for both the front end and the back end, to ensure that the base level of the program is working correctly. This base-level program of the honeypot can be viewed as the prototype and can be built upon in the future to add the full functionality of the honeypot. The prototype will be shown to my supervisor to ensure that I am on target.

All configuration settings for the database are stored in a JSON file for convenience, if the database credentials or port number changes, the only modification to be made will be in the JSON file. This was done to make the program setting simpler to manage, so there is no need to alter all files when there is a password or username change for the database.

## Front-end

The register, login and logout functions for the web application have been developed. The web application has its table which it uses to store the credentials of the user. When the user enters the username and password, they will be sanitized by removing characters that are below ASCII 32. To connect to the database PDO was used for its simplicity and being easy to read. The SQL statements consist of two stages prepare and execute; the statements bind parameters to them so they can be sent to the database. Preparing SQL statements can help protect against SQL injections. If no accounts are found in the database, the user will be directed to register an account. The password from the registered account will be hashed before being inserted into the database. The hashing of the password is done with the bcrypt algorithm, in PHP this is designed to change over time and adapt new and stronger algorithms. (PHP, n.d). The user will be directed to the login page where they will need to enter their correct credentials. If successful a session will be created and will be redirected to the home page. The session is used so that only the user who is logged in can access the other pages that may contain sensitive information. The user can navigate to all the different pages that currently contain no information, they can also log out, which will terminate their session and they will need to log in again.

## Back-end

In the back end of the program, the database, and tables to store the data have been created. In the program, I have created a file that will automatically create the database and tables that are needed for the program to run. This file was created so the program can be moved to another computer, without the need to manually create the databases and all its tables. The only thing that needs to change is the credentials in the JSON file. Along with the database file, there is also a file that handles the insertion of the data from the honeypot to the database. The main app file has been created and is listening on all ports between 1 and 1023. On occasion, some ports will not open due to forbidden access permission, though I believe this is due to an open application on my desktop that is using that port. When I tested the program with the test code that I had created, all connections were established and then instantly closed for security. The access attempts by the test program were correctly logged into the database. However, the database connections had to be pooled, as only having one database connection could not keep up with the amount of data that needed to be inserted. When a user accesses port 22 they will be transferred to paramiko so it can handle the connection. The current state of SSH allows the user to enter any credentials and they will be logged into the database. There is a username and password, which can be brute force as the password can be found on the RockYou wordlist. In its current state if the user were to get the right credentials it would be accepted but that does not matter as an SSH channel has not been coded so the SSH connection will terminate automatically. No other protocols have been coded, but the HTML for the honeypot fake web page has been created as it's just a clone of the normal login page.

# References

# References

Abdulrazaq Almutairi, D. P. R. P., 2012. Survey of High Interaction Honeypot Tools: Merits. *Proceedings of the 13th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting, PGNet2012. PGNet..*

Ashish Girdhar, S. K., 2012. Comparative Study of Different Honeypots System. *International Journal of Engineering Research and Development,* Volume 2( Issue 10 ), pp. 23-27.

Babak Khosravifar, J. B., 2008. An Experience Improving Intrusion Detection Systems False Alarm Ratio by Using Honeypots. *IEEE.*

Bedell, C., 2022. *computer worm.* [Online]
Available at: https://www.techtarget.com/searchsecurity/definition/worm
[Accessed 12 December 2023].

Chrissikopoulos, D. F. a. E. M. a. V., 2014. Evaluating Low Interaction Honeypots and their Use against Advanced Persistent Threats. *Proceedings of the 18th Panhellenic Conference on Informatics.*

Constantin Musca, E. M. R. D., 2013. Detecting and Analyzing Zero-day Attacks using Honeypots. *IEEE,* pp. 543-548.

eccouncil, 2022. *The Cyber Kill Chain: The Seven Steps of a Cyberattack.* [Online]
Available at: https://www.eccouncil.org/cybersecurity-exchange/threat-intelligence/cyber-kill-chain-seven-steps-cyberattack/
[Accessed 11 December 2023].

Flask, 2010. *Flask.* [Online]
Available at: https://flask.palletsprojects.com/en/3.0.x/
[Accessed December 28 2023].

Forcier, J., n.d. *Welcome to Paramiko's documentation!.* [Online]
Available at: https://docs.paramiko.org/en/latest/
[Accessed 28 December 2023].

IBM, 2022. *IBM Report: Consumers Pay the Price as Data Breach Costs Reach All-Time High.* [Online]
Available at: https://newsroom.ibm.com/2022-07-27-IBM-Report-Consumers-Pay-the-Price-as-Data-Breach-Costs-Reach-All-Time-High
[Accessed 07 December 2023].

IBM, 2023. *Cost of a Data Breach Report 2023,* n.d: IBM.

IBM, n.d. *What is a zero-day exploit?.* [Online]
Available at: https://www.ibm.com/topics/zero-day
[Accessed 11 December 2023].

IBM, n.d. *What is an intrusion detection system (IDS)?.* [Online]
Available at: https://www.ibm.com/topics/intrusion-detection-system
[Accessed 09 December 2023].

imperva, n.d. *Advanced persistent threat (APT).* [Online]
Available at: https://www.imperva.com/learn/application-security/apt-advanced-persistent-threat/
[Accessed 12 December 2023].

Iyatiti Mokube, M. A., 2007. Honeypots: Concepts, Approaches, and Challenges. *Proceedings of the 45th Annual Southeast Regional,* p. 321–326.

lucidchart, n.d. *https://www.lucidchart.com/blog/types-of-UML-diagrams.* [Online]
Available at: https://www.lucidchart.com/blog/types-of-UML-diagrams
[Accessed 29 December 2023].

Oracle, n.d. *13.5 Prepared Statements.* [Online]
Available at: https://dev.mysql.com/doc/refman/8.0/en/sql-prepared-statements.html
[Accessed 03 January 2024].

PHP, n.d. *password_hash.* [Online]
Available at: https://www.php.net/manual/en/function.password-hash.php
[Accessed 30 December 2023].

Pragya Jain, A. S., 2012. Defending against Internet Worms using Honeyfarm. *Association for Computing Machinery,* p. 795–800.

purplesec, 2023. *Cyber Security Statistics.* [Online]
Available at: https://purplesec.us/resources/cyber-security-statistics/#ZeroDay
[Accessed 11 December 2023].

Saud, Z. a. I. M. H., 2015. Towards Proactive Detection of Advanced Persistent Threat (APT) Attacks Using Honeypots. *Association for Computing Machinery,* p. 154–157.

W. Eddy, E., 2022. *[RFC 9293] Transmission Control Protocol (TCP).* [Online]
Available at: https://datatracker.ietf.org/doc/html/rfc9293
[Accessed 03 January 2024].

Yogendra Kumar Jain, S. S., 2011. Honeypot based Secure Network System. *IJCSE.*