



**DE MONTFORT  
UNIVERSITY  
LEICESTER**

De Montfort University

# Honeypot for Defence of a Local Network

Final Deliverable Draft



– Kira Leigh Sherriff

## Development Project

Supervisor -



Total Word Count

7810

# Contents

<b>Introduction.....</b>	<b>4</b>
Background .....	5
Problem & Motivation .....	5
Existing honeypots .....	5
Project Objectives .....	5
Project Overview.....	6
<b>Analysis.....</b>	<b>7</b>
Methodology.....	8
Tools For Development.....	8
GitHub .....	8
PHP Server.....	8
Tools For Testing .....	8
Hydra.....	8
Wireshark & Nmap.....	8
Tools for Final Implementation.....	9
MySQL .....	9
Python Packages .....	9
<b>Design .....</b>	<b>10</b>
Code Structure .....	11
Design.....	11
Functional Requirements.....	11
User Case Diagram .....	11
UML Class Diagram .....	11
Database .....	12
<b>Back-End .....</b>	<b>13</b>
Introduction .....	14
Install.exe & Uninstall.exe .....	14
Database .....	15
Database Login.....	15
Creation of the Database .....	15
Logging Data.....	16
Launch Files.....	16
TCP Socket Scripts .....	16

Protocols .....	18
Secure Shell.....	19
Server Script.....	19
Interface Script.....	19
Command Script.....	19
HTTP .....	20
Telnet .....	21
Socket Script.....	21
Server Script.....	22
Command Script.....	22
File Transfer Protocol.....	23
Server Script.....	23
Alert.....	24
Alert Script .....	24
<b>Front-End.....</b>	<b>25</b>
Introduction .....	26
Web Server.....	26
Login & Register page .....	26
Register page.....	26
Login.....	27
Home Page .....	27
View Data Page .....	28
Setting Page .....	28
<b>Testing .....</b>	<b>30</b>
Testing.....	31
Back-end Testing.....	31
Front-end Testing.....	31
<b>Critical Analysis.....</b>	<b>32</b>
Strengths .....	33
Challenges .....	33
Improvements.....	34
<b>References.....</b>	<b>35</b>
<b>Appendix .....</b>	<b>36</b>



# Introduction



Word Count

754

## Background

Honeypots are used to gather information about the attacker's behaviour, to determine the threat landscape. The literature review in the first deliverable provides evidence as to why honeypots are a critical part of defence. Intelligence gathering can help implement the correct mitigations to prevent an attack and act as an early warning system (Ashish Girdhar, 2012). Honeypots can aid in accurate detection and traditional intrusion detection systems by reducing the false positive rate and improving performance.

## Problem & Motivation

In 2023 an IBM report found that it takes over 277 days on average for an organisation to identify and report a data breach which could cost over £5 million. (IBM, 2023) In 227 days, an attacker can do damage by stealing intellectual property, which will have legal and financial consequences. Identifying a data breach earlier can mitigate the impact and reduce the cost of an attack. Honeypots act as an early warning system, helping improve the response time. The information gathered from a honeypot can allow a cyber security technician to apply the correct mitigation for the threat landscape.

The motivation for this project is to create a tool that can help detect when an intruder is in the network, through their interactions with network protocols. This will give insight into the attacker's target and goals while helping mitigate future attacks and keeping critical infrastructure safe from attackers.

## Existing honeypots

There is a wide range of honeypots that are available some are network-based and emulate a whole network, while one like this project emulates network services to detect an attacker in a network. This honeypot was developed on Windows, typically a project like this will be done on Linux due to the simplicity but Windows was chosen due to the lack of open-source honeypots supported for the OS. There are some honeypots, one example being "WinHoneyd", but this is not designed for Windows 11 and may not work. From research, few low interactive honeypots exist for Windows 10/11 systems, and though there is "KFSensor" there is only a free trial available. Most open-source honeypots on GitHub are for Linux systems, which most are not compatible with Windows creating the opportunity to make a low interactive honeypot for Windows that emulates some network protocols.

## Project Objectives

The research conducted for the literature review helped lay out the functional and non-functional requirements as to what makes an effective honeypot. Honeypots should be sensitive to capturing data from the connection attempt, and stealthy so they cannot be fingerprinted. This is why more ports were developed to make it more authentic to a real system. The honeypot should also not be a victim that can be used to launch attacks. (Yogendra Kumar Jain, 2011)

The objective of the project is stated within the contract to create a honeypot for the defence of a local network to detect an intruder. There should also be a front-end web application that can view the data and control the honeypot function by opening and closing ports. The back end is coded in Python, while the front uses PHP. The honeypot will log into a database and ensure that it is sensitive to data. The data that will be logged are IP Addresses, ports accessed, time/date, usernames, passwords, and commands. The honeypot will contain protocols with which the user can interact, such as SSH, Telnet, HTTP and FTP. These four protocols all take credentials, and SSH and Telnet both take commands.

The requirements for the project were:

- Create a database to store the logs generated by the honeypot.
- To accept all connections from port 0, 1024
- Log all connection attempts to those ports.
- Implement SSH
- Implement Telnet.
- Implement FTP
- Implement HTTP.
- Display gathered information with a web application.
- Web application can manage the back end by opening and closing ports

## Project Overview

An attacker can connect to the protocols that are developed, which will log all interactions into the database. The attacker can enter a command and get a coded response in two of the developed protocols. There is no underlying operating system behind the ports, so the honeypot cannot be compromised. However, the code supplying the responses to the command could compromise the honeypot's identity as someone with knowledge would easily notice that SSH or Telnet is not operating normally. The owner of the honeypot can view the data through a secure web application through a single-user login system. There the user can view the most common usernames, and passwords, for each of the protocols, and the most common ports and IP addresses.



# Analysis



Word Count  
804

## Methodology

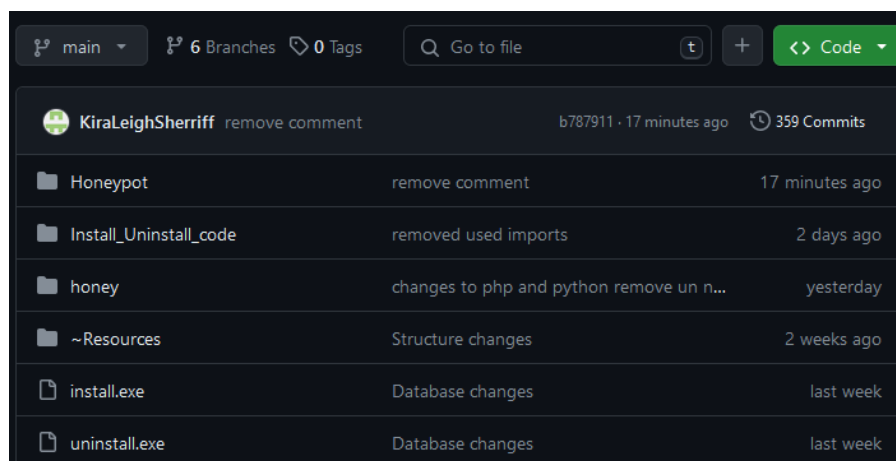
The agile methodology was followed for this project, which involves breaking down the project into tasks that need to be completed through an iterative process. An iterative process can help identify semantic errors within the code as it will be analyzed many times which benefits the project development. The project has multiple different components, the database, the honeypot and its protocols and the web application. When managing this in the given time, splitting them into different mini projects was the most effective way.

The software development life cycle is design -> Code -> Test. Each iteration was to be designed then coded and tested. Each one was reviewed, and rewrites were made to make the code more efficient and improve the logic, then deployed and the cycle started again. Doing this ensured that some of the requirements would have been met if something went wrong with the project and completed to a high standard.

## Tools For Development

### GitHub

Throughout the project, the version control software GitHub was used to track and manage any changes over time when developing the program. GitHub can review the history of the code and when mistakes are made, they can be undone. The ability to undo saved this project many times when a wrong mistake was made during development. GitHub has also been very useful when it comes to developing the project on multiple different devices.



### PHP Server

PHP has a built-in server that can be used for development which makes it faster as no third-party software is needed such as XAMPP. The PHP server allows for quick on-demand access to the web application when needed and will automatically close with Virtual Studios so an underdeveloped web application can no longer be accessed.

## Tools For Testing

### Hydra

The tool Hydra was used to test each protocols response to being brute forced. Hydra was chosen out of the other brute-forcing tools as it is the one that is most familiar to me and have used it throughout my studies at DMU.

### Wireshark & Nmap

The network activity of the protocols can be observed in Wireshark, which can be used to analyse how each protocol is behaving and if changes need to be made to the code. The protocols can



also be tested throughs scanning the device running the honeypot with Nmap to see if they are detected.

## Tools for Final Implementation

When it comes to deploying the honeypot, no third-party services were used such as XAMMP or WampServer, as setting up and configuring my own web server and database on Windows opened an opportunity to learn new skills. PHP, MySQL, and Apache were all installed individually which gives more control.

### Apache lounge

The project was developed on Windows 11 meaning it needs Apache Lounge as its web server to process and send data to the user. Apache Lounge will only process the web application for the front end as the back-end HTTP is controlled by Waitress. Apache is configured to only be accessed by localhost, and not the whole network, which is critical to keeping the identity of the honeypot secret as part of the non-functional requirements.

### MySQL

MySQL was used as the database manager to store the data collected by the honeypot. MySQL was chosen as it can be scaled to need and is secure with a large community which can help when trying to fix errors within the code when relating to accessing the database. MySQL is flexible and can store large amounts of data which the honeypot might generate when the attacker is trying to gain access.

## Python Packages

### *MySQL Connector*

The MySQL connector was used by Python to connect to the database and execute commands to create and delete the database and it's tables. The MySQL connector provides secure communication over TCP/IP using SSL. (MySQL, 2024)

### *Socket*

The socket package is a lower-level networking interface, which can be used to make network connections. In the project, this package was used for all but one protocol to establish the TCP/IP connections over IPv4 and send and receive data from an attacker. (Python, n.d.)

### *Paramiko*

Paramiko was used to build SSH which is implemented over SSHv2 protocol and provides functionality for the server interface to create the SSH channel. (Forcier, n.d.)

### *Flask + Waitress*

Flask was used to create the web page that the attacker will access to enter credentials. Flask was used as it is quick and easy to develop and deploy. (Flask, n.d.) Waitress is the production server which was used to deploy the Flask app (Flask, n.d.)

### *MIMEText & MINEMultiplepart & smtplib*

This package that is used to send the email to the owner of the honeypot and can dynamically create an email based on each of the different protocols that are accessed by the attacker. SMTPlib sends emails over TLS on port 587.



# Design



Word Count

703

## Code Structure

Python and PHP use object-oriented programming (OOP) that helps with the organisation of the code, allowing for some reusability. OOP can allow for better concentration on creating a single object at once instead of trying to do all things at once. OOP can allow for faster development and help maintain the code and is flexible to updates. (databox, 2023) The code follows the best practice when it comes to naming the classes, functions, and variables. The classes and functions use the naming convention pascalcase while variables use snakecase. This way classes, functions, and variables can easily be separated and identified.

## Design

Over the development of the project, there have been some design changes made to improve the functionality. These changes include both the database and the UML diagram. The database only has some small modifications. There is an updated version of the database entity relationship diagram and UML diagram, and the functional requirements and user case diagram can be seen in the appendix.

## Functional Requirements

There has been two changes to the functional requirements all of which can be seen in Appendix 1. There has been a very slight modification to Telnet, so it allows commands, to add more variety for the user. There has also been the removal of 1 and this was HPY13, which was to implement an uncomplicated Firewall (UFW). This was originally in the functional requirements as the system was going to be developed on Linux but later changed to Windows. UFW is not supported by Windows and has its firewall built in making UFW a redundant requirement.

## User Case Diagram

The user case diagram from the first deliverable shows the two actors and how they will interact with the system. There is a direct user, the administrator (Appendix 2) and indirect user the attacker (Appendix 3). The direct user, the administrator manages the setting and is directly interacting with the knowledge that it is a honeypot. While the indirect user, the attacker unlike the administrator does not know they are attacking a honeypot and are therefore indirectly contributing data to the honeypot. The attacker and administrator have two different views of the honeypot, which affects how they interact with it. The attacker has a limited view of the honeypot as they only have access to the development ports, which are SSH, Telnet, FTP, and HTTP. The administrator will have a full view which allows them to access all the protocols and have access to a front-end web application that views the data and controls the ports if they should be open or closed.

## UML Class Diagram

The new UML diagram can be seen in Appendix 5, the changes made include having a socket script that will handle the connection for each protocol. This was done so the protocols are not dependent on a single central file, and each can be run independently. This is the same for the log scripts that are used to insert the data into the database, in the original UML diagram there was only 1 log script which would insert the data. Now there is one for every protocol. This was done to keep each protocol separated and make for better readability. There is also an alert script for each of the protocols, but there is a centralized script that will send the email. A centralized script was chosen to send the email as, it contains the password for a Gmail account and realistically should not be in the code, and the exposure of the password should be limited. Structuring the code this way allows each protocol to be its mini-project. In the future, if new protocols need to be added, they can be done without affecting the functionality of the already-developed protocols.

## Database

The database is managed by MySQL, used to store, and display the data collected from the honeypot. There have been slight changes to the tables, ssh\_login\_successfull and ssh\_login\_fail have been combined into one table called ssh\_login. Once more table has been added called "Telnet\_command" that stores the commands entered to the honeypot through telnet. The Time\_Date field has also been separated for better readability on the web application. A schema diagram has been created that can be seen in Figure 1.

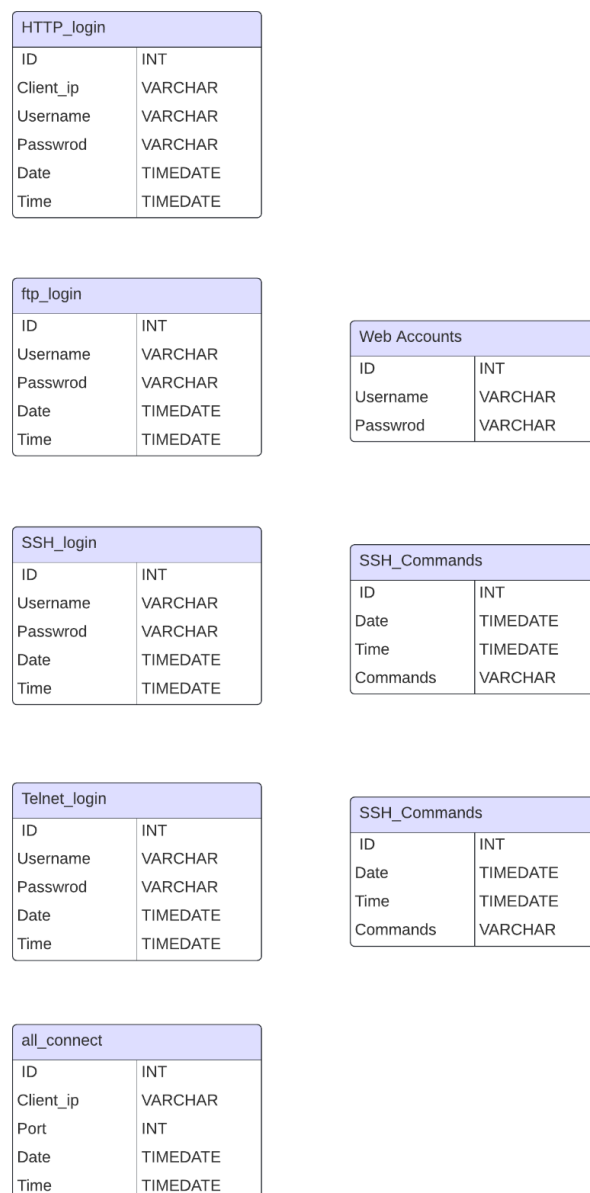


Figure 1: schema Database of honeypot tables



# Back-End



Word Count

3449

## Introduction

The back end of the program consists of the database and honeypot. The code for the back end has been structured in a way that puts each requirement in a separate folder (Figure 3) and contain scripts for each function (Figure 2). JSON is used to store information for the back end, such as the login credentials for the database connection and the credentials for Telnet and SSH. The back end is the critical part of the project as it contains the honeypot, which is the main objective of this project.

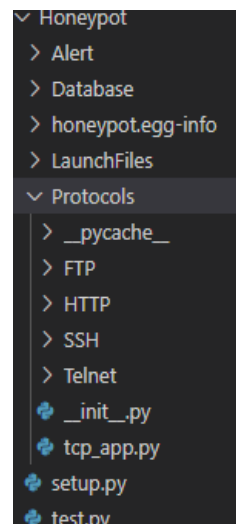


Figure 3: File structure

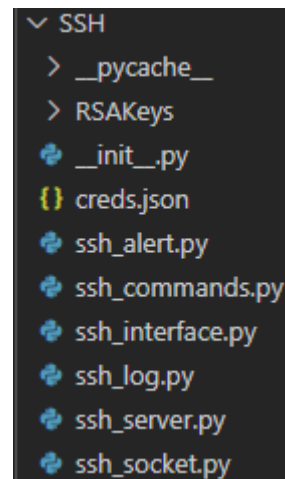


Figure 2: Example of code structure

## Install.exe & Uninstall.exe

The install and uninstall are Python scripts that have been converted to Windows executable files, so the user does not interact with Python. The install script is used to create the database and put all the honeypot folders into the correct file locations so they can be accessed. The location chosen for the back end was "C:\Users\name", as it does not need administrative permission and is easily accessible. The name of the Windows account is needed which was obtained through the function "os.getlogin()". The Windows account name was put into a JSON file inside the "honey" folder as PHP has no way of getting the Windows account name, which is needed to create the path to execute the scripts and access back end JSON files. The web application folder "honey" is placed into the "C:\apache24\htdocs\" making it accessible through the web browser. The uninstall.exe simply removes the honeypot files from their stored location and will drop the database, making for a quick install and clean-up. Figure 4 shows a flow diagram of where the file will be put for better visualizations.

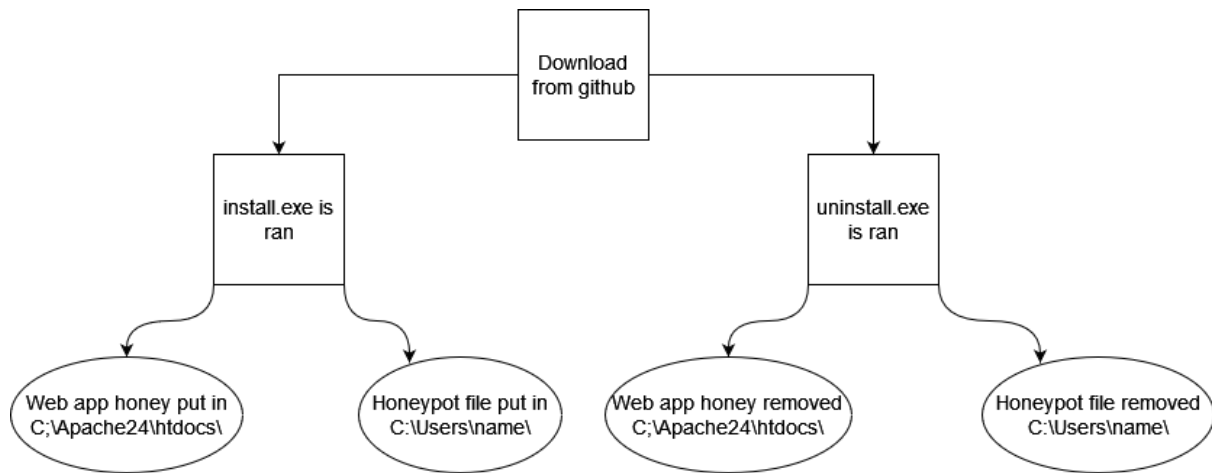


Figure 4: Flow diagram of how the exe will work

## Database

The database is used to store information that is collected by the honeypot and is displayed on the web application. The database manager used for this project is MySQL, as it is flexible. To access the database through the Python code the library MySQL-connector was chosen, which is developed by ORACLE. (ORACLE, n.d.)

### Database Login

A JSON file was used to store the login credentials of the account, this is accessed by both Python and PHP. This approach avoids putting the database login credentials into the code and offering a central location that when changed will affect the whole program. The JSON file contains the database and table names used for when they need to be created and access to insert data. A file in Python is created to access the data in the JSON file seen in Appendix 7. The script accesses the JSON file by getting the current directory and combining it with the name. The data in the JSON file is then stored in a Python variable that is then imported into the other scripts so they can access the database. This Python script was created as it removed the need to reuse the code, in other scripts reducing workload.

### Creation of the Database

The Python script is seen in Appendix 8 automatically creates the database. The database needs to be automatically created as it removes errors or any misconfiguration of the tables which could affect the program's functionality. The code connects to the MySQL account and creates a cursor that can execute queries. It is also important that the script can check if the table and database exist, so it does not give errors when the database is already created.

The script that creates the database as seen in Appendix 8 is called by install.exe. The first function called is "CreateDatabase" which uses the cursor to execute the "SHOW DATABASES" command to fetch all databases and check if the database exists and if not it will be created. Once the database is created it can then be accessed and the function "TableCreate" is called and will iterate through the list of tables in the JSON file and create them. In the "TableCreate" function it will compare the table to a string in an if statement so the table can be created with the correct data types. Once all of this is completed the last step is to close the connection, so it is not left hanging, which could eat up resources and be a security risk.

## Logging Data

The last element of the database is how the program logs data from the different protocols into the correct table. This was originally going to be done by one single file that manages all the data insertion but was changed for a better structure. Since there could be a large amount of data being inserted into the database, having a pool of connections made the most sense. Pooling the connections allows for better performance when inserting data. Once the data is inserted the connection will be closed and returned to the pool. Pooling MySQL connections can help reduce the latency of the program, as it does not have to create a new connection to insert data. The only problem was that my code needed to exceed the 32 maximum connections that were available with the MySQL connector described in the documentation. (ORACLE, n.d.) This is why every different protocol within the honeypot has its log file to combat this problem and is a better way of managing data insertion.

In Appendix 9 the code for an insert script can be seen that logs all the connections made by the socket scripts. This log script is the only one that is accessed by all protocols to remove the need to repeat it. Each of the log files contains the same code, the only difference is that they output the data into different tables. In the code there are 20 pooled connections, this was done as if the system is being scanned it will need to be able to keep up with the tool. Each log script has its own number of pooled connections based on need.

## Launch Files

There are launch files for each of the protocols, the code for the main launch file to start all protocols can be seen in Appendix 6. This was done to control which port can be opened from the web application. The script starts by calling the function "GetProcessID", which gets the Process ID of the script and stores it into a JSON file. This was done so that the script could be terminated by the PHP in the front end, as it can kill the script with the PID in the JSON file. Once the PID is stored in the JSON file the launch script will call the socket scripts for each protocol with a thread, which creates a separate flow for the execution. The thread is critical as it opens all ports concurrently which can speed up the program.

## TCP Socket Scripts

The socket scripts are key aspects of the honeypot, they open ports into a listening state. The socket is coded only to accept TCP connection on IPv4, this was done as a UDP connection is not suitable for a honeypot. Each of the protocols has its own socket script, which is just a duplicate code of each other with a slight change in the ports that are bound to the IP address. Having duplicate code is not that effective, but in the case of this project, it is beneficial as it allows the separation of the protocols meaning they are not dependent on a single socket script. A socket script for each of the protocols makes the program easier to read and they will not alter each other's functions. This allows for better maintenance in the future and the ability to easily add protocols to the honeypot project without affecting the already existing protocols.

In Appendix 10 the socket script that opens the ports in the range of 1-1024 can be seen. In this script, it can be noted that ports 22, 80, 23, and 21 are not being opened as they are being handled by a different socket script, that contains the same code just that the port is different. In the script that opens all ports from 1-1024, the script starts with the function called "Openall", which iterates through the well-known ports and creates a thread when calling the function "HoneyConnect" in Figure 5. This function will create the socket by taking the IP address 0.0.0.0 for all interfaces on the machine and the port number of that given iteration, together to make the socket. Once the socket is created it will be placed in a state of listening and has a window size of 250. When a connection is accepted, the three-way handshake will take place and the data collected such as the IP address and port will be



logged into the database. All other socket scripts for each of the protocols follow the same format other than they don't have a for loop for the ports. The "port" variable in the bind function (Figure 5) is replaced by the port number for that protocol.

```
def HoneyConnect(self, port):  
    try:  
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
        server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)  
        server_socket.bind(('0.0.0.0', port))  
        server_socket.listen(250)  
  
        while True:  
            client_socket, client_addr = server_socket.accept()  
            self.con_insert.OtherPorts(client_addr[0], port)  
            client_socket.close()  
  
    except Exception as e:  
        print(f"Error: {e}")  
        server_socket.close()
```

Figure 5: Creating the socket snippet code

## Protocols

Before discussing the protocols, a flow diagram has been created (Figure 6) to help better visualize how the code functions. The flow diagram is generic such as not having individual alert or login scripts but can still aid with understanding how it works.

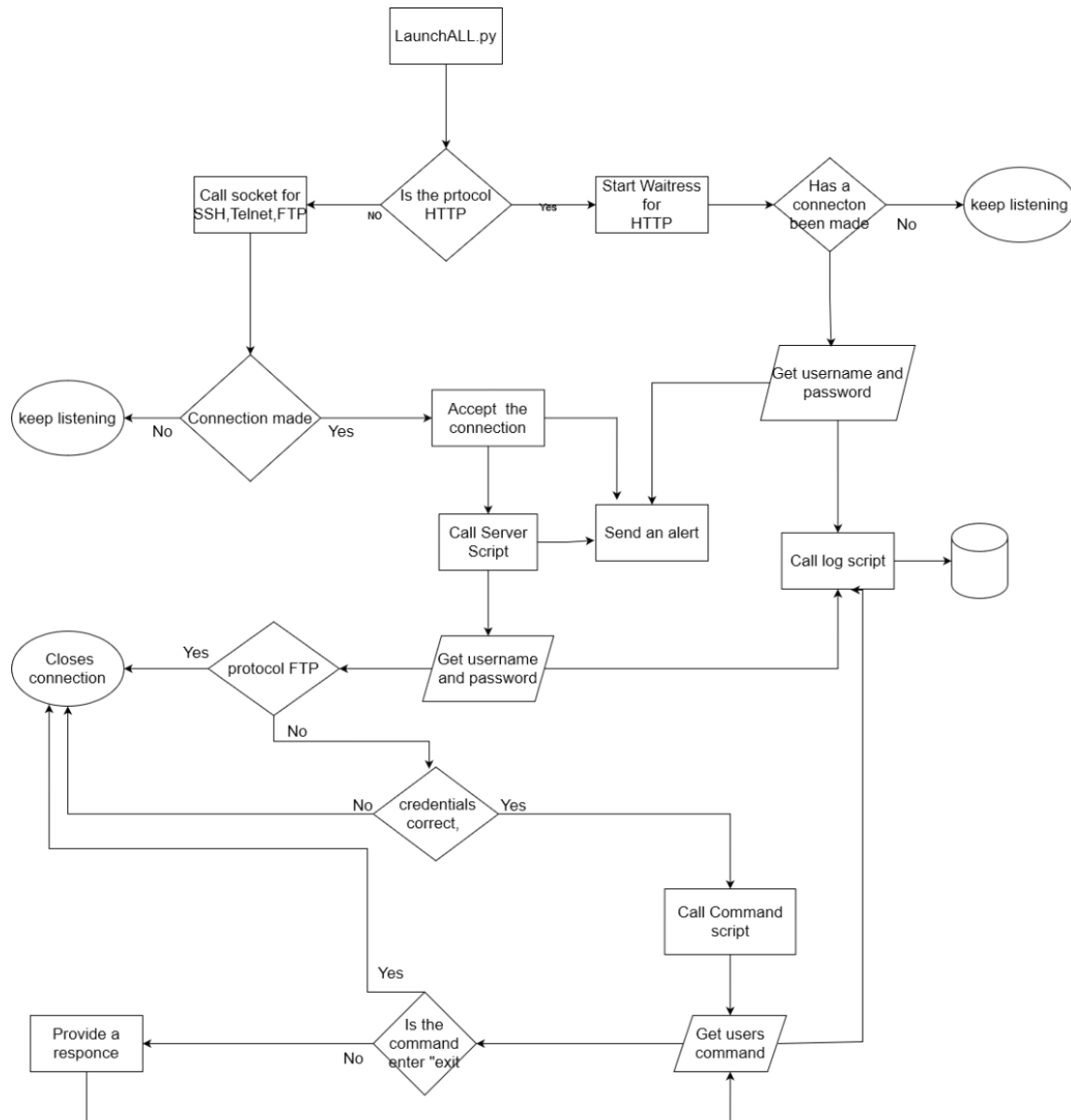


Figure 6: Flow diagram of the code for all protocols

## Secure Shell

Secure shell protocol (SSH) is used to remotely connect to devices through a command line terminal and can execute commands on the system, a diagram of the connection process can be seen in Figure 7. SSH was chosen as one of the developed ports as it is one of the most targeted ports as many users leave it vulnerable by using weak passwords or misconfigured. The SSH honeypot can take usernames and passwords and also commands which are logged into the database. An attacker may try to brute force SSH, which will keep them distracted from critical systems. Behind the SSH login, there is no operating system, instead, the code will respond to the commands that are sent down the SSH channel. This was done as it still allowed monitoring of what commands users might try to enter once they first access SSH, but without providing a staging ground that the attacker could use for lateral movement in a network.

### Server Script

The SSH server script can be seen in Appendix 11 is called by one of the socket scripts that is explained in the above section. An RSA key is needed to create a secure connection between the SSH server and the client. The RSA is generated on my host machine and is part of the project, so the RSA key is not generated each time SSH is launched. If the RSA key was generated each time for each launch of the script, then the user will get an error, as the client should have detected the RSA key change, and they would not be able to access SSH until they remove it from their "known\_host" file. The script starts by calling the function "ConnectSSH" which will transport the RSA key into the Paramiko class "Transport" which negotiates the encryption and authentication, and then the Paramiko "ServerInterface" class will be called. This will create the SSH channel and control the user login, by taking the username and password detailed in the next section. If the credential is correct a welcome message will be sent, and the server script will take commands and log them into the database. The user command will be sent to another script to be processed. If the credentials are incorrect the channel will be terminated, and the attacker will try to log in again.

### Interface Script

The SSHInterface class in Appendix 12 is called by the server script, and it holds the paramiko code that will configure the behaviour of SSH. It has been configured to only allow passwords and any attempts to access SSH through the RSA keys will be disallowed. Since one of the requirements of the project is to allow the user to interact with the SSH a correct username and password in a JSON file is stored. If the entered credentials match with the username and password stored in the JSON file, the attacker will log in. A JSON file was used as it can be modified by the front-end web application, which allows the user to have more control. The JSON file is being accessed in the interface script instead of another script like with the database login. Doing this allows automatic updates of the credentials every time the function is called so the program does not need to be restarted improving usability. If the user enters the correct credentials, then a channel will be created, and an alert will be sent notifying a connection was successful.

### Command Script

When the attacker enters the command in the server script, the command will be sent to the command script seen in Appendix 13 that will process and send a response. There are only a few commands that the code will respond to, and these commands were chosen based on their popularity and how likely an attacker will be to enter them into the terminal.

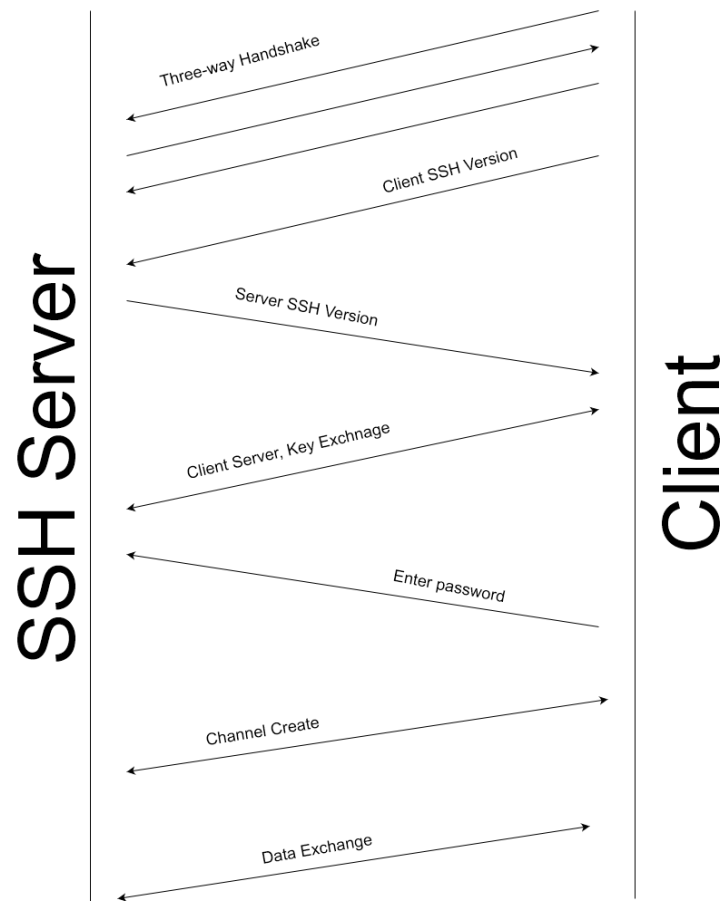


Figure 7: SSH connections

## HTTP

To create the web page which allows an attacker to enter login credentials the Python packet Flask was used. Flask was chosen over other packets such as Django for its simplicity. Django is used for larger-scale web applications, the smaller scale of this project makes Flask a better fit. Flask is also more flexible and allows the simple creation and rapid development with a single file. A single HTML file is needed to create the login page which will allow the attacker to enter credentials that can then be put into the database. The HTML page was reused from the front end of the web application; the only difference is that the title has been removed. Reusing the HTML from the front end allows me to save time when developing the HTTP honeypot. The HTML file for Flask needs to be stored in a template folder so it can be rendered. When deploying the Flask application there is a production server that will handle the communication between the Flask application and the client.

HTTP has a different structure from the other protocols and has no socket or server script needed as the whole protocol is controlled by the Flask code seen in Appendix 14. There is a single file which controls the application, that handles both GET and POST HTTP methods to send and receive data from the client, a diagram of this can be seen in Figure 8. In the first deliverable, the network architecture was explained, and a local DNS is in the network meaning the honeypot can have a domain name instead of being accessed by an IP address, the domain name map to the honeypot is "privatecloud.com".

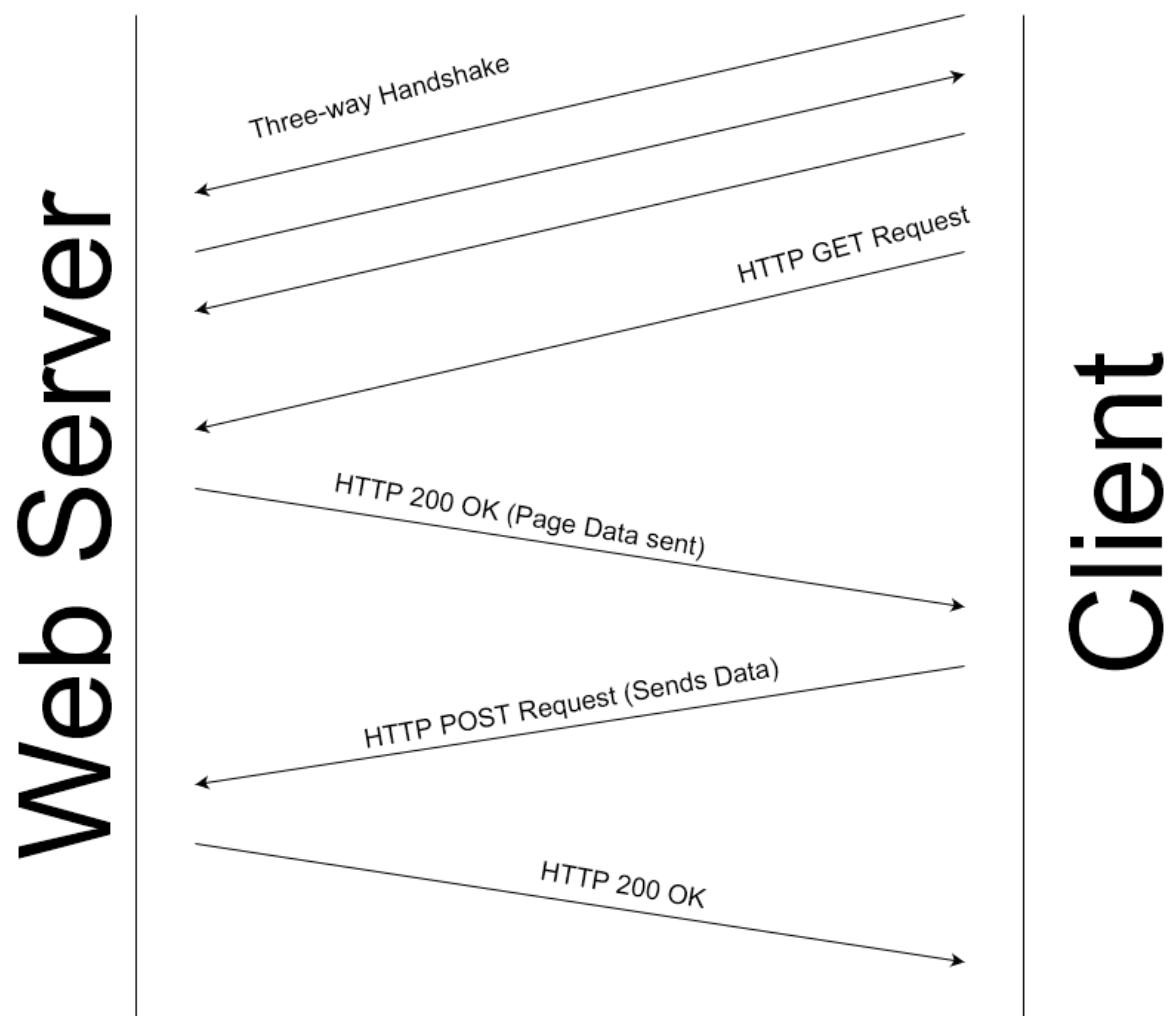


Figure 8: HTTP honeypot diagram

## Telnet

Telnet was chosen as a developed port as it is a vulnerable service and does not provide end-to-end encryption. Even though telnet is not used within my network it provides more flexibility to the user if they wish to have Telnet instead of SSH. Telnet has been coded to be in character-by-character mode meaning it will only transmit 1 byte at a time, this is because Telnet is an 8-bit connection as defined in RFC 854 (Postel, 1983). When creating Telnet lower-level socket programming was used to send and retrieve data to and from the client. The lower-level socket was chosen as after researching Python packages on “pypi” none was found that would fit the needs of the project.

## Socket Script

There is a slight difference in this socket script as there is a variable called “attempts” which is set to 0 and will be sent as a parameter within the “start” function to the server script. This was done to count how many times the attacker can attempt to log into telnet in a single connection.

### Server Script

In the “Start” function seen in Appendix 15 there is a while statement that takes the variable “attempts” and when it exceeds 3 the while statement is terminated, stopping any more login attempts for that connection. The first function called in the while statement is “username” which gets the username from the user, then the “password” function which will get the password. Once both credentials are taken the function “AuthCreds” seen in Appendix 16 is called which will check the username and password against those stored in the JSON file. If the correct credentials are entered access is gained and an alert will be sent. The user will be able to enter commands through the “command” function, which will call another class to process the inputted data.

### Command Script

The “GetCommand” function works very similarly to SSH (Appendix 13) It will take the user input and supply a coded response since there is no operating system. The commands entered will be logged into the database.

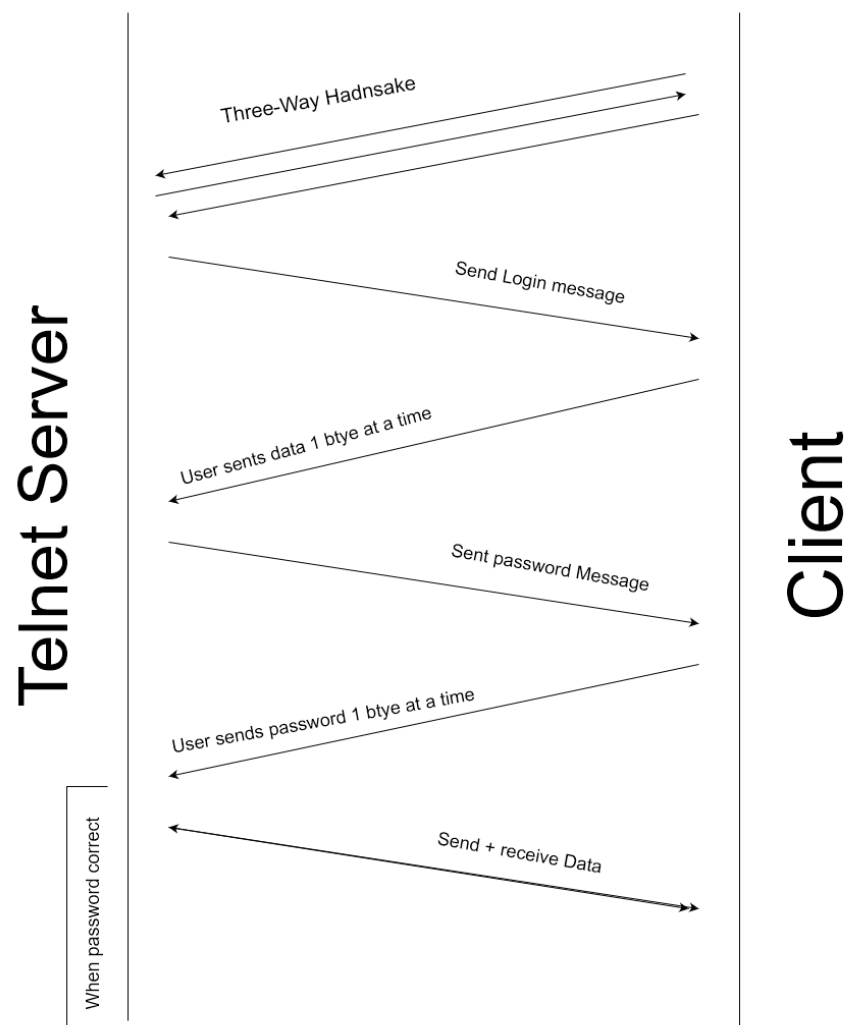


Figure 9: Telnet connection

## File Transfer Protocol

File Transfer protocol was not fully developed so the attacker will not be able to upload or download any files on the system. FTP will only allow the attacker to enter usernames and passwords. This was done as allowing the attacker to upload files through FTP could result in compromising the device. Also, if an attacker were to upload a file there would need to be a place to store the files, which may not be ideal depending on the device that the honeypot is installed on. FTP like Telnet was coded using lower-level socket programming for the same reason that the available Python packages did not suit my needs nor were they well documented. Developing FTP through lower-level sockets made me gain a better understanding of FTP commands that are used when creating a connection.

### Server Script

When the attacker connects to the FTP socket it will call the “login” function in Appendix 17, which allows the attacker to enter a username and password. FTP is handled by control commands that are described in RFC 959, (J. Postel, 1985) these are added to an if statement with the “startswith” function as the user will be the one transmitting the commands through the socket. The FTP commands used in the code are USER (username) and PASS (password), which are part of the authentication process. The USER argument is used to identify the username, and PASS is used to identify that password. There is a diagram that can help better visualize what is happening with the FTP connection. (Figure 10) When an attacker first connects to the FTP server an alert will be sent to the owner and the username and password, will be logged into the database like all the other protocols.

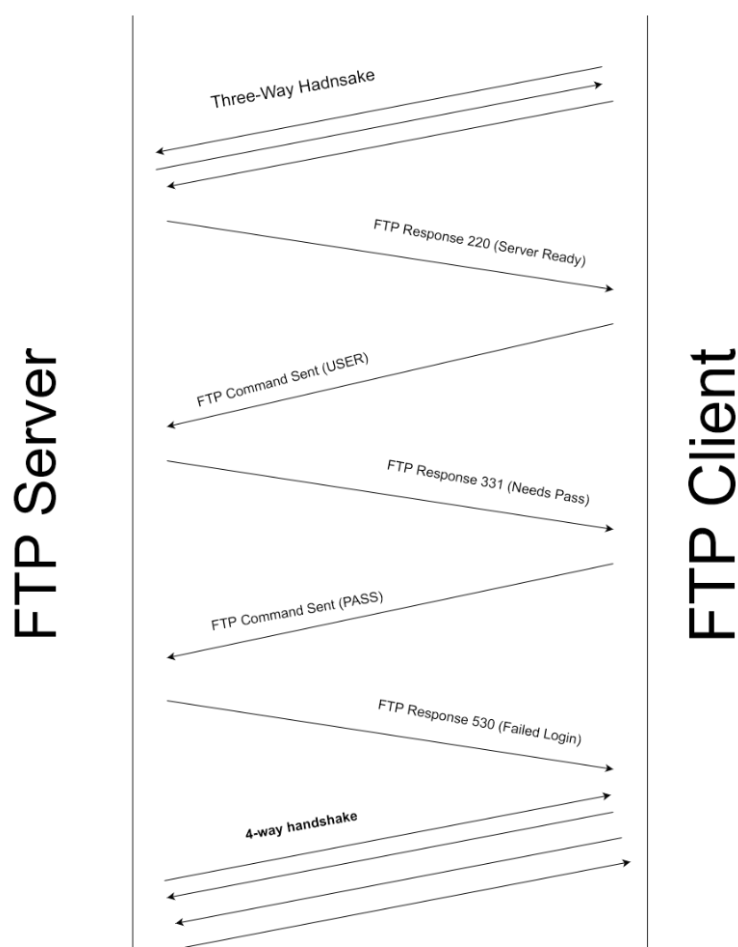


Figure 10: FTP connection when an attacker connects

## Alert

The alert script was created so the owner of the honeypot knows when a login attempt has been made on any of the developed protocols. There is no alert for when non-developed ports are accessed as this would generate many alerts which could overwhelm the user with notifications. The alert is sent to an email address that is found in a JSON file that can be modified by the web application. Email is used over SMS, as it is a simpler approach and is more professional. A single email account was created that will send out the alerts to all the users.

## Alert Script

The alert script can be seen in Appendix 18 and is called multiple times by different scripts within the program. The email is created by using the MIMEMutitPart object which will structure the email with the correct headers. Other content can be added to the email such, as HTML, plaintext, images, and attachments. For this project, the only thing added is plaintext through MIMEText to alert that an attacker is connected or trying to connect to a protocol. Using the MIME packages can allow for better flexibility if a change is needed to the email, it can be made without changing the structure of the code. Once all parts of the email are constructed it will be sent with a simple message transfer protocol server using the package smtplib over port 587 which means it will be transported over TLS.

Each protocol has its alert script which will provide the subject and body that will be sent as part of the email. This was done so the code in Appendix 18 does not have to be repeated more than once, which saves time in development. An example of a protocol alert script can be seen in Appendix 19, each of the scripts has its subject and body based on that protocol. An example of a sent email can be seen in Figure 11 when a connection is successfully made to SSH.

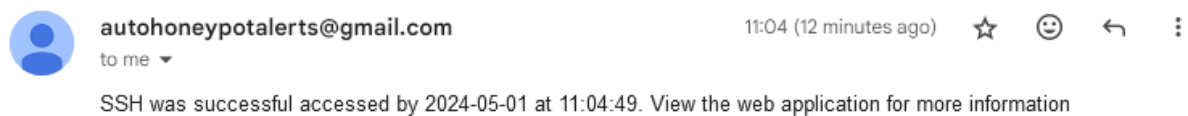


Figure 11: SSH email for a successful login





# Front-End



Word Count

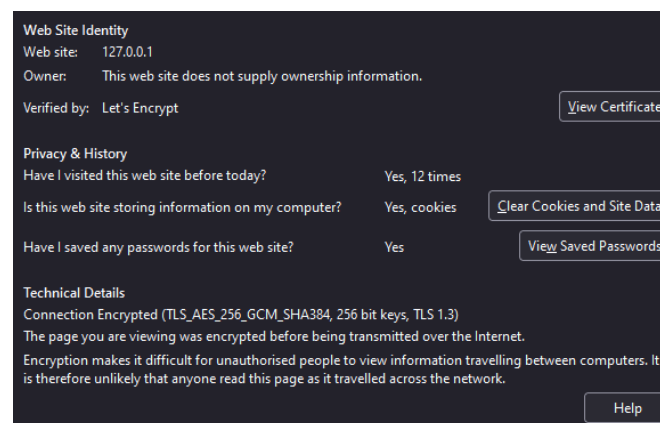
1003

## Introduction

The front-end web application provides a way for the user to interact with the back end of the honeypot without needing to alter the code and view the data. The web application is protected by a login that is a one-user system. The login is monitored by a session, which has a max life of 10 minutes, when expires the user will be logged out. The web application accesses the database with the PDO framework, as it is lightweight and can improve the security of the web application, also statements are prepared to help prevent SQL injections.

## Web Server

The front-end web application is controlled by Apache Lounge and is configured to only be accessible through localhost. If the web application was open to the local network an attacker would be able to see the web application. This will breach the non-functional requirement of the honeypot needing to be stealthy. It was possible to secure the localhost with Let's Encrypt by doing a DNS challenge of my domain and downloading the certificate that can then be configured in the "httpd.conf" file. Even though this is a localhost-only connection it is still good practice to encrypt the traffic.



## Login & Register page

The login and register pages are within the same PHP script, this was for simplicity. The web application is a one-user system so there is no need to have a dedicated register page. The code will count how many accounts there are and if the return is 0 then the register function will be called. When an account is created the value always returns 1 so the login function will always be called making it impossible to register an account. Both the register and login functions validate and sanitize the user input avoiding illegal characters being entered, which can lead to security concerns.

## Register page.

The register function will create a web account that will be entered into the database. Before the password is sent to the database it is hashed, this was done as it is a security risk to transport and store the password in plaintext. When entering the password, it must have at least 1 of the following, a number, and a symbol and with more than 8 characters. This will hopefully prevent a brute force attack against the web application, and it forces the user to have a secure password. If the criteria are not met, then an error message will be returned (Appendix 21).

## Login

The login must provide the correct username and password, which will also be sanitized and validated. When the user provides the correct login, a session will be created that is updated each time the user interacts with the web application, and after 10 minutes of being idle the session will be terminated, and the user will need to log in once again. This was done as if an unauthorized person got access to the machine, they would be able to access the honeypot web application, but terminating the session prevents this from happening (Appendix 20).

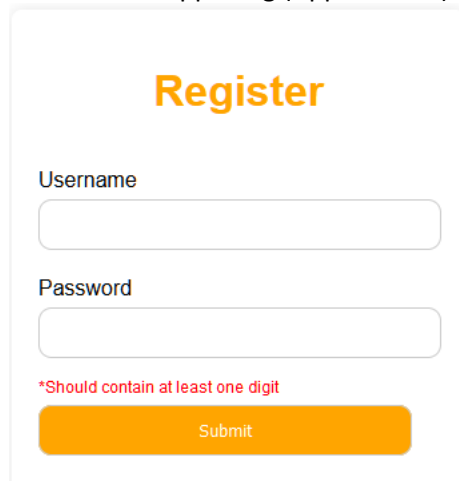


Figure 12: Example of error message.

## Home Page

The home page is the page that is first loaded when the user logs into the web application. The home page is very basic and provides an overview of the most accessed port and IP addresses, along with the most common username and password of each of the protocols. This was done so the user does not need to navigate to all the different pages to get the information that they might want, having all of it in one place provides some convenience to the user.

Honeypot Home Page			
Data Summary			
Client IP	Count	Port	Count
192.168.1.222	1035	443	25
192.168.1.224	25	80	9
		22	5
		21	5
		23	4
		3	1
		1	1
		2	1
		5	1
		7	1
SSH Most Common Password		SSH Most Common Username	
root		toor	
Telnet Most Common Password		Telnet Most Common Username	
sdf		sdf	
HTTP Most Common Password		HTTP Most Common Username	
root		root	
FTP Most Common Password		FTP Most Common Username	
test		test	

## View Data Page

These pages view the data collected by the protocols by fetching all the data within the tables and displaying it for the user to view (Appendix 22). The view is designed to be simple and not complex at all, just provides a better way of visualizing the data from the database for those who might not be familiar with MySQL. At the top of the page, the user can view the most common usernames and passwords entered in the protocols, which can help the user, so they do not have to manually check themselves.

Home SSH Telnet HTTP FTP All Ports Settings Logout					
Most Access Port			Most Access IP Address		
80			192.168.1.222		
ID	IP Address	Port	Date	Time	
1	192.168.1.222	22	2024-04-07	17:31:08	
2	192.168.1.222	3	2024-04-08	13:37:33	
3	192.168.1.222	1	2024-04-08	13:37:33	
4	192.168.1.222	4	2024-04-08	13:37:33	
5	192.168.1.222	2	2024-04-08	13:37:33	
6	192.168.1.222	5	2024-04-08	13:37:33	
7	192.168.1.222	6	2024-04-08	13:37:33	
8	192.168.1.222	7	2024-04-08	13:37:33	
9	192.168.1.222	8	2024-04-08	13:37:33	
10	192.168.1.222	9	2024-04-08	13:37:33	
11	192.168.1.222	10	2024-04-08	13:37:33	
12	192.168.1.222	11	2024-04-08	13:37:33	
13	192.168.1.222	13	2024-04-08	13:37:33	
14	192.168.1.222	12	2024-04-08	13:37:33	
15	192.168.1.222	14	2024-04-08	13:37:33	
16	192.168.1.222	15	2024-04-08	13:37:33	
17	192.168.1.222	16	2024-04-08	13:37:33	
18	192.168.1.222	17	2024-04-08	13:37:33	
19	192.168.1.222	18	2024-04-08	13:37:33	
20	192.168.1.222	19	2024-04-08	13:37:33	
21	192.168.1.222	20	2024-04-08	13:37:33	
22	192.168.1.222	23	2024-04-08	13:37:33	
23	192.168.1.222	22	2024-04-08	13:37:33	
24	192.168.1.222	21	2024-04-08	13:37:33	
25	192.168.1.222	24	2024-04-08	13:37:33	
26	192.168.1.222	25	2024-04-08	13:37:33	
27	192.168.1.222	26	2024-04-08	13:37:33	
28	192.168.1.222	27	2024-04-08	13:37:33	
29	192.168.1.222	28	2024-04-08	13:37:33	
30	192.168.1.222	29	2024-04-08	13:37:33	
31	192.168.1.222	30	2024-04-08	13:37:33	
32	192.168.1.222	31	2024-04-08	13:37:33	
33	192.168.1.222	32	2024-04-08	13:37:34	
34	192.168.1.222	33	2024-04-08	13:37:34	
35	192.168.1.222	34	2024-04-08	13:37:34	
36	192.168.1.222	35	2024-04-08	13:37:34	
37	192.168.1.222	36	2024-04-08	13:37:34	
38	192.168.1.222	37	2024-04-08	13:37:34	
39	192.168.1.222	38	2024-04-08	13:37:34	
40	192.168.1.222	39	2024-04-08	13:37:34	
41	192.168.1.222	40	2024-04-08	13:37:34	
42	192.168.1.222	41	2024-04-08	13:37:34	

Figure 13: Page of All connect table

## Setting Page

At the top of the web page, the user can open and close the ports in a range of 21, 22, 80 and 23, and all ports from 1024 by using a checkbox. When the checkbox is ticked and submitted to PHP will run a command using the function “popen” (Appendix 24) which opens a pipe to execute a process through forking the command that will open the Python script. If the user wants to terminate the Python script the PID stored in the JSON file will be used to kill the Python script. Since HTTP is stateless once the web page is reloaded the checkbox will be unticked even if the Python file is running, to fix this the state of the checkbox is stored in a cookie, which will keep the checkbox in the correct state. However, the problem with using a cookie is that they expire after a certain amount of time which will cause the stored data to be lost.

The settings page is also used to modify the information within the JSON file stored on the back end (Appendix 23). The user can modify the content in the JSON files to fit their need such as changing the information to connect to the database or username and password of SSH and Telnet. There is no sanitation and validation for the settings page this is because the data being entered will not compromise the honeypot but will only limit the functionality for the user.

The user can also download all the tables within the database minus the web\_account table. The data is exported as a comma separated values file (CSV). As it is compatible with a simple text editor and Microsoft Excel. The database can be flushed which will prevent the database from getting too large, but this flush will not remove the web\_account table. It's important that web\_account table is not affected as this will affect the users experience, and account data should not be downloaded.

[Home](#) [SSH](#) [Telnet](#) [HTTP](#) [FTP](#) [All Ports](#) [Settings](#) [Logout](#)

[Download Database](#)

[Reset Database](#)

**HoneyPot Controls**

All Ports ☐ SSH ☐ Telnet ☐ FTP ☐ HTTP ☐

**Configure the Database Settings for Login**

Database Host Address

Database Port

Database Username

Database Password

**Email for the alert**

Email

**Configure the Login settings for SSH**

SSH Username

SSH Password

**Configure the Login settings for Telnet**

Telnet Username

Telnet Password

Save Changes

Figure 14: Settings page



# Testing



Word Count

383

## Testing

When testing the honeypot, it is important to make sure the test plan seen in Appendix 4 is met. This was done by interacting with the honeypot to see the response and trying different tools. Throughout each iteration of the agile methodology, the last stage was testing, which all protocols went through.

### Back-end Testing

To test if the honeypot met the test plan criteria, they were connected too through the command line, and in the case of HTTP through the browser. The testing ensured that the protocols were working and entering data into the database with an alert being sent. The protocols should also be visible when an attacker does a scan of the device running the honeypot. This was tested using Nmap, in Figure 15, when all ports are open, they are being correctly scanned by Nmap. This activity was not logged as the honeypot only logs data when the three-way handshake is complete, and the Nmap scan is not making a full connection meaning it will not be logged.

```
Not shown: 989 closed tcp ports (reset)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
80/tcp    open  http
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
443/tcp   open  https
445/tcp   open  microsoft-ds
902/tcp   open  iss-realsecure
912/tcp   open  apex-mesh
3306/tcp  open  mysql
```

Figure 15: Results of Nmap running honeypot

Also, for further testing, each protocol was brute forced to check if it could be attacked through a basic method, and that the database could handle all the connections. Since the protocols can be detected by Nmap, they can be brute-forced. When testing 3 out of 4 of the protocols can be brute force (Figure 16) using “Hydra”, the protocol that didn’t work was Telnet. Telnet was coded to be an 8-bit connection each character is sent over a different packet, meaning that when hydra tries to brute force Telnet it will overload the connection and cause it to crash.

```
h1ragk1ra-7289-5329A2:~$ hydra -L username -P password 192.168.1.222 ssh -t 4
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-04-19 13:54:58
[DATA] max 4 tasks per 1 server, overall 4 tasks, 45 login tries (l:5/p:9), -12 tries per task
[DATA] attacking ssh://192.168.1.222:22/
[22][ssh] host: 192.168.1.222  login: root  password: root
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-04-19 13:55:42
```

Figure 16: SSH brute-forced

### Front-end Testing

The first part tested in the web application was the login, which ensured only one user could create an account, and that they could log in. Viewing the database in the web application was as easy as just checking if the data was being correctly displayed on the page. When testing the settings page, the JSON files in the back-end need to be checked to see if they are being modified. The setting page can also open and close the ports, to test if this was working, Task Manager was used to see if the Python script was launched and if it was killed when the checkbox was unticked.



# Critical Analysis



Word Count

814



## Strengths

A strength of this project was the organisation of the work using the agile methodology that allowed for the program to be created to a good standard and completed on time for submission. The organisation of the project allowed for a simpler and less stressful development of the project as it allowed for the visualisation of how much progress was being made, which kept up the motivation to work on the project every day. The organisation provided by the agile methodology allowed for an increase in productivity and for better performance.

The honeypot can accept all connections and will log them into the database and the attacker can connect to four protocols. All Four protocols allow an attacker to enter a username and password which will be logged into the database, and the owner can see the most common credentials that were entered. Three out of the four protocols were successfully brute-forced. Two of the protocols can also take commands which are logged into the database, allowing insight into what the first move of the attacker might be if they got access to a real system. When the user connects to the honeypot an alert is sent successfully to the owner of the honeypot, helping them stay up to date with any activity that might indicate that an attacker is within the network.

For the web application, my main aim was to make it as simple as possible while still looking professional and displaying the information in an easy-to-read format. The user interface follows a straightforward design of containing a table and headers with the most common username and password. This does not overload the user with information as each of the protocols has its own page to view its data. The setting page for the web application also follows the same principles of being simple, with all configurations on a single page making it easier to manage. The settings page can also be used to start and close the protocols which makes the program more user-friendly as the terminal will not be needed to start the launch of Python scripts.

The web applications are also secure with a login in which the user's password will need to contain, 1 number, and 1 symbol with more than 8 characters to be accepted, this will hopefully help prevent any brute force attacks on the web application. Once the user is logged in a session will be made which will last 10 minutes this is done to stop someone from accessing the web application when the device hosting the honeypot might be unattended.

## Challenges

One of the challenges faced during the development was researching and finding the correct Python packages that were suitable for this project on "pypi". Python has many packages which could help development but most of them have poor documentation, making them hard to follow and implement into the code. This is why FTP and Telnet protocols were created using lower-level programming of the sockets, as it provides good documentation.

The honeypot data it is logging in the SSH, Telnet, FTP and HTTP do not contain the client's IP address when they are connecting to the honeypot, which means that the user may have a hard time correlating the connection attempts to the username and, passwords. This can be done by examining the time and date of the connection and comparing it to the time and date of when the username and passwords were entered. This approach is not user-friendly as it requires the user to go between pages to get that information and work out the IP address that entered the credentials. The same problem occurs with the commands that are entered into the database by Telnet and SSH.

Another challenge of the project was getting the two protocols that allow commands to look authentic. Since no operating system is running behind the port of the honeypot the code will need to respond to the client. An attacker who knows SSH and Telnet after a few minutes of interacting will

notice the protocols are not behaving as they should be. This will compromise the identity of the honeypot which breaks one of the non-functional requirements, stealth that was described within the first deliverable.

### Improvements

If this project is redone instead of having a web application, to manage the honeypot it would be better to have a Windows GUI application, that controls the protocol and views the data. This is because if the web application is misconfigured and is open to the network, the attacker will see the honeypot login which will compromise the honeypot. This does go against the non-functionality requirements of needing the honeypot to be stealthy. The Windows GUI will only be accessible on the host the honeypot is installed upon, which makes for better security and useability for the user.

## References

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.

databox, s. (2023) *11 Advantages of OOPS*. Available at: <https://sparkdatabox.com/blog/11-advantage-of-oops/> (Accessed: 24 March 2024).

Flask (n.d.) *Flask*. Available at: <https://flask.palletsprojects.com/en/3.0.x/> (Accessed: 13 March 2024).

Flask (n.d.) *Waitress*. Available at: <https://flask.palletsprojects.com/en/3.0.x/deploying/waitress/> (Accessed: 13 March 2024).

Forcier, J. (n.d.) *Server implementation*. Available at: <https://docs.paramiko.org/en/latest/api/server.html> (Accessed: 13 March 2024).

IBM (2023) *Cost of a Data Breach Report*. IBM.

J. Postel, J.R. (1985) 'RFC 959 File Transfer Protocol', *ISI* pp.24-25. Available at: <https://datatracker.ietf.org/doc/html/rfc959> (Accessed: 28 March 2024).

KFSensor (2024) *Why use KFSensor?* Available at: <https://www.kfsensor.net/kfsensor/> (Accessed: 27 April 2024).

MySQL (2024) *6.1 Introduction to MySQL Connector/Python*. Available at: <https://dev.mysql.com/doc/connectors/en/connector-python-introduction.html> (Accessed: 13 March 2024).

ORACLE (n.d.) *9.4 Connector/Python Connection Pooling*. Available at: <https://dev.mysql.com/doc/connector-python/en/connector-python-connection-pooling.html> (Accessed: 14 March 2024).

Postel, J.a.J.R. (1983) *Telnet Protocol Specification RFC 854*. Available at: <https://www.rfc-editor.org/rfc/rfc854.txt> (Accessed: 28 April 2024).

Python (n.d.) *Low-level networking interface*. Available at: <https://docs.python.org/3/library/socket.html> (Accessed: 13 March 2024).

Satheesh, V. (2023) 'The Evolution Of Honeypot Technologies and Future Directions', *irjmets*, 5(5).

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



# Appendix



## Appendix 1: Tables for the functional requirements

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, can be brute forced

Table 3

ID:	HYP04
Title:	Honeypot Telnet Development
Description:	Telnet development, to study login attempts and accept commands
Dependencies:	N/A
Resources:	IDE, GitHub, Desktop/Laptop, Python library.
Actor:	Hacker/ Admin
Priority:	Medium
Notes:	Should only monitor for login attempts, can be brute forced

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

<b>Priority:</b>	High
<b>Notes:</b>	Should only monitor for login attempts, can be brute forced

Table 5

<b>ID:</b>	<b>HYP06</b>
<b>Title:</b>	Honeypot Alert
<b>Description:</b>	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.
<b>Dependencies:</b>	HYP01, HYP02, HYP03, HYP04, HYP05
<b>Resources:</b>	IDE, GitHub, Desktop/Laptop, Python library, account to send alerts too
<b>Actor:</b>	Admin
<b>Priority:</b>	High
<b>Notes:</b>	Should be alerted through email or phone number

Table 6

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

Table 7

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

Table 5

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

Table 6

<b>ID:</b>	<b>HYP10</b>
<b>Title:</b>	Honeypot Web Application change user settings
<b>Description:</b>	Users should be able to change email/phone numbers for the alerts, through the web application.
<b>Dependencies:</b>	HYP08
<b>Resources:</b>	IDE, GitHub, Desktop/Laptop, PHP, HTML, Apache.

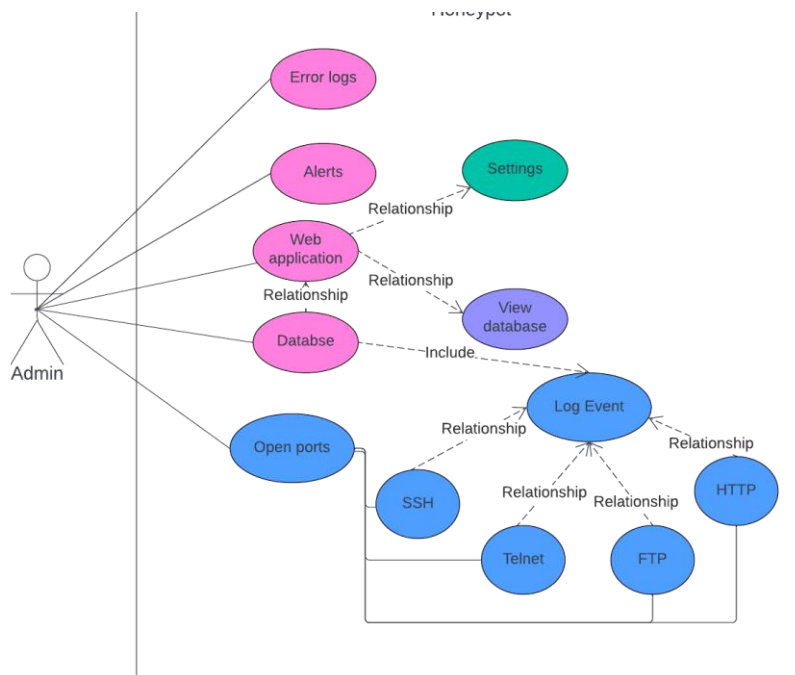
<b>Actor:</b>	Admin
<b>Priority:</b>	Medium
<b>Notes:</b>	This should be created after the login page

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

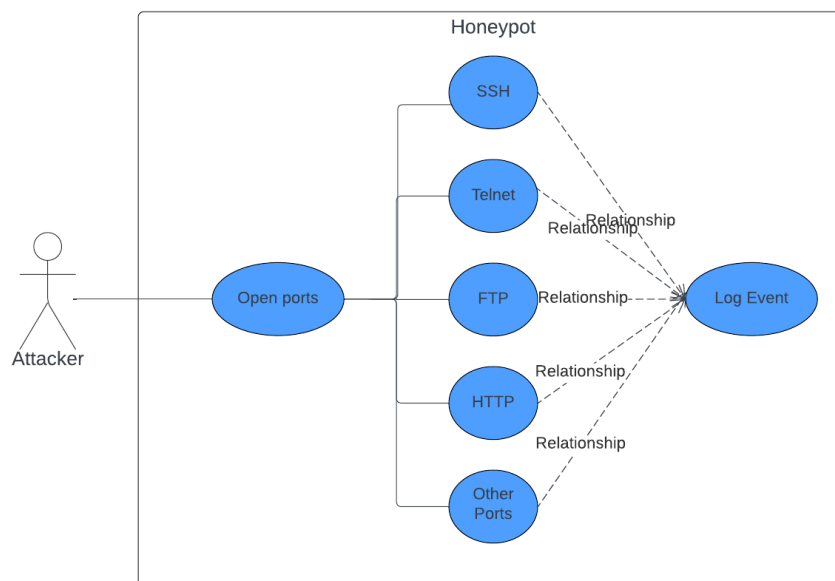
Table 7

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

Appendix 2: Admin user case diagram



Appendix 3: attacker user case diagram



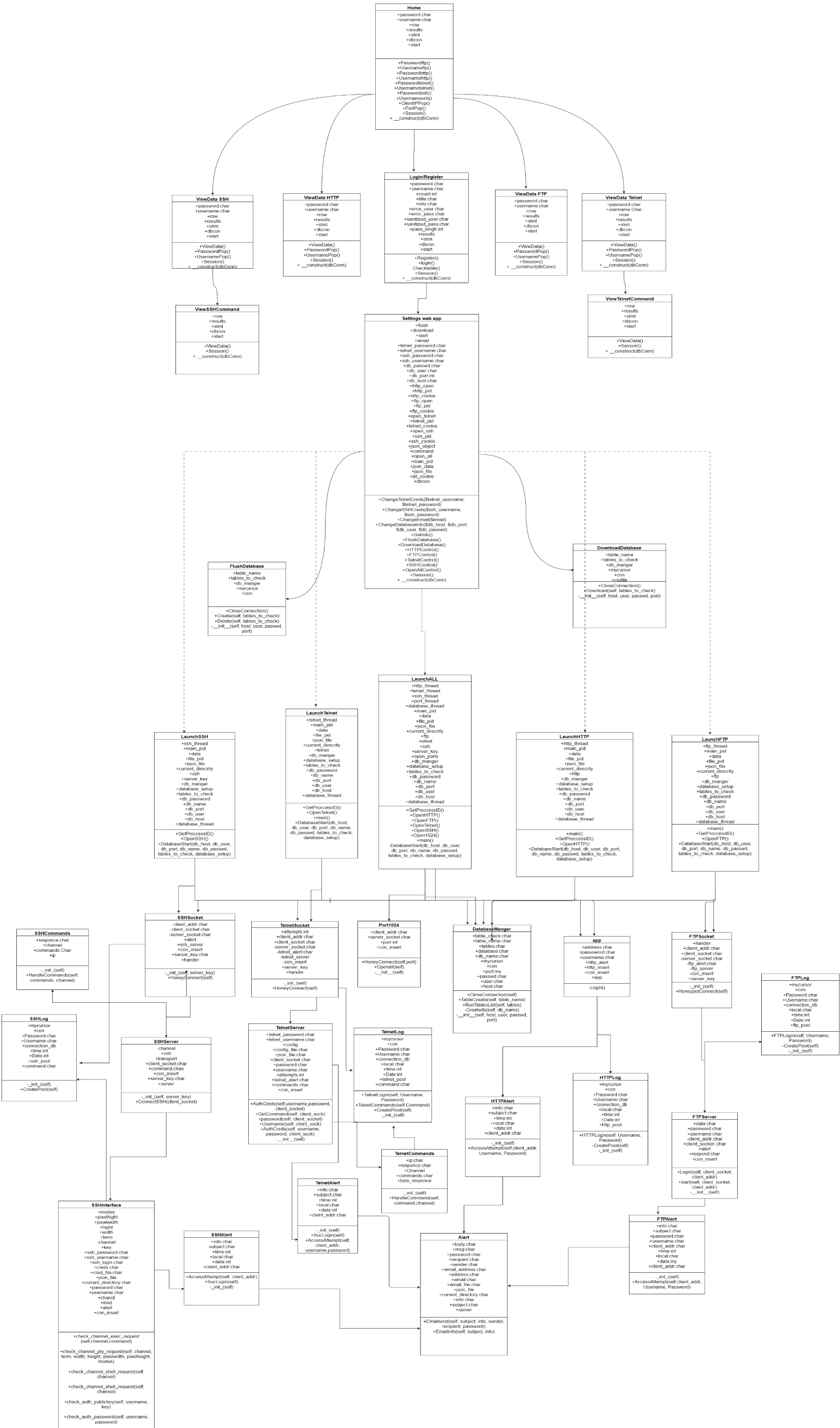


Appendix 4: Test plan table

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	The attacker can enter a command into SSH and get a coded response	Pass
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	The honeypot sends an email when the attacker connects or tries to connect	Pass
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	The attacker can access the port 80 honeypot and enter a username and password	Pass
12	The HTTP attempts should be logged into the database	The access attempts to HTTP are logged into the database	Pass
13	Should alert an attempt made to HTTP	Alert will be sent when an attacker has submitted a web form	Pass
14	Should have some interactive capability with Telnet.	Telnet can interact with commands that provide a code response	Pass
15	All access attempts with Telnet should be logged into the database	All-access attempts are logged into the data	Pass
16	Should alert an attempt made to Telnet	Alerts are sent when the telnet is accessed	Pass
17	should allow a user to attempt to connect to FTP	Alert will be sent when an attacker tries to connect to FTP	Pass
18	Should log the attempt into its database	The database logs all access attempts made by the attacker for FTP	Pass

19	Should have an alert when an attempt is made to FTP	Alert will be sent when an attacker tries to connect to FTP	Pass
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	The user can change the login credentials for all protocols and the database	pass
25	Should have the ability to open/close ports	The user can open and close the ports 21,22,23,80 and 0-1023	Pass
26	The web page view log should be available for the user to view	Can view all logs in the database	Pass
27	The log page for SSH should be available	The user can view the SSH logs	Pass
28	The log view for HTTP should be available	The user can view all logs of HTTP	pass
29	The log view for Telnet should be available	The user can view the logs of telnet, the command, usernames, and password	pass
30	The log view for FTP should be available	The user can view FTP logs, usernames, and passwords	Pass
31	All log views must contain a filter to search for a time a data	The user can not filter the data of the web page	Failed
32	All logs must have a button to flush the database.	All tables can be flushed with a download button on the settings page	Pass
33	All logs should have a download button to download the log file	All data in the tables can be downloaded as a CSV	Pass

Appendix 5: New UML Diagram for the application



## Appendix 6: Launch all protocols

```

1 > Import paramiko --
12
13
14 def Open1024():
15     open_ports = Ports1024()
16     open_ports.Openall()
17
18 def OpenSSH():
19     username = os.getlogin()
20     server_key = paramiko.RSAKey.from_private_key_file(f"C:\\Users\\{username}\\HoneyPot\\Protocols\\SSH\\RSAKeys\\RSAHoney")
21     ssh = SSHSocket(server_key)
22     ssh.HoneyConnect()
23
24 def OpenTelnet():
25     telnet = TelnetSocket()
26     telnet.HoneyConnect()
27
28 def OpenFTP():
29     ftp = FTPServer()
30     ftp.HoneyConnect()
31
32 def OpenHTTP():
33     from Protocols.HTTP.app import app
34     from waitress import serve
35     serve(app, host="0.0.0.0", port=80)
36
37
38 def GetProcessID():
39     current_directory = os.path.dirname(os.path.abspath(__file__))
40     json_file = os.path.join(current_directory, 'pids.json')
41
42     with open(json_file, "r") as file_pid:
43         data = json.load(file_pid)
44
45     main_pid = (os.getpid())
46     data['pythonpid'][0]['allpid'] = main_pid
47
48     with open(json_file, "w") as file_pid:
49         json.dump(data, file_pid, indent=4)
50
51
52 def main():
53
54     port_thread = threading.Thread(target=Open1024)
55     ssh_thread = threading.Thread(target=OpenSSH)
56     telnet_thread = threading.Thread(target=OpenTelnet)
57     http_thread = threading.Thread(target=OpenHTTP)
58     ftp_thread = threading.Thread(target=OpenFTP)
59
60     port_thread.start()
61     ssh_thread.start()
62     telnet_thread.start()
63     http_thread.start()
64     ftp_thread.start()
65
66     port_thread.join()
67     ssh_thread.join()
68     telnet_thread.join()
69     http_thread.join()
70     ftp_thread.join()
71
72 if __name__ == "__main__":
73     GetProcessID()
74     main()

```

## Appendix 7: JSON file to access data

```
Database > Config > db_config.py > ...
import json
import os

current_directory = os.path.dirname(os.path.abspath(__file__))
json_file = os.path.join(current_directory, 'db_config.json')

with open(json_file) as config_file:
    config = json.load(config_file)

database_config = config['database_config']
table_names = config['table_names']

db_host = database_config['db_host']
db_user = database_config['db_user']
db_port = database_config['db_port']
db_passwd = database_config['db_passwd']
db_name = database_config['db_name']

table_1 = table_names['table_1']
table_2 = table_names['table_2']
table_3 = table_names['table_3']
table_4 = table_names['table_4']
table_5 = table_names['table_5']
table_6 = table_names['table_6']
table_7 = table_names['table_7']
table_8 = table_names['table_8']

tables_to_check = config['tables_to_check']
```

### Appendix 8: The script that is used to create the database

```

1 > import mysql.connector
2
3 class DatabaseManager():
4     def __init__(self):
5         try:
6             self.db_connect = mysql.connector.connect(
7                 host=db_host,
8                 user=db_user,
9                 port=db_port,
10                passwd=db_passwd
11            )
12            self.mycursor = self.db_connect.cursor()
13        except mysql.connector.Error:
14            pass
15
16    def CreateDatabase(self, db_name):
17        try:
18            self.mycursor.execute("SHOW DATABASES")
19            database = self.mycursor.fetchall()
20            if db_name in database:
21                pass
22            else:
23                self.mycursor.execute(f"CREATE DATABASE {db_name}")
24        except mysql.connector.Error:
25            pass
26
27    def TableCreate(self, tables_to_check):
28        try:
29            for table_name in tables_to_check:
30                self.mycursor.execute(f"USE {db_name}")
31                self.mycursor.execute(f"SHOW TABLES")
32                table_exists = self.mycursor.fetchall()
33                if table_name in table_exists:
34                    pass
35                else:
36                    if table_name == 'all connect':
37                        self.mycursor.execute(f"CREATE TABLE {table_name} (ID INT AUTO_INCREMENT PRIMARY KEY, Client_IP VARCHAR(50), Port VARCHAR(50), Date DATE, Time TIME)")
38                    elif table_name == "http_login":
39                        self.mycursor.execute(f"CREATE TABLE {table_name} (ID INT AUTO_INCREMENT PRIMARY KEY, Client_IP VARCHAR(50), Username VARCHAR(50), Password VARCHAR(255), Date DATE, Time TIME)")
40                    elif table_name == "web_account":
41                        self.mycursor.execute(f"CREATE TABLE {table_name} (ID INT AUTO_INCREMENT PRIMARY KEY, username VARCHAR(50), password VARCHAR(255))")
42                    elif table_name == "ssh_command" or table_name == "telnet_command":
43                        self.mycursor.execute(f"CREATE TABLE {table_name} (ID INT AUTO_INCREMENT PRIMARY KEY, Date DATE, Time TIME, Command VARCHAR(250))")
44                    else:
45                        self.mycursor.execute(f"CREATE TABLE {table_name} (ID INT AUTO_INCREMENT PRIMARY KEY, Username VARCHAR(50), Password VARCHAR(50), Date DATE, Time TIME)")
46        except mysql.connector.Error:
47            pass
48
49    def CloseConnection(self):
50        try:
51            self.mycursor.close()
52            self.db_connect.close()
53        except mysql.connector.Error as e:
54            pass
55
56    manager = DatabaseManager()
57    manager.CreateDatabase(f"{db_name}")
58    manager.TableCreate(tables_to_check)
59    manager.CloseConnection()

```

*Appendix 9: Script to log the data into the database*

```
import mysql.connector
from datetime import date
import time

from .Config.db_config import db_host, db_user, db_passwd, db_name, db_port, table_1

class InsertData:
    def __init__(self):
        self.honeypot_pool = self.CreatePool()
        self.Date = date.today()
        local = time.localtime()
        self.Time = time.strftime("%H:%M:%S", local)

    def CreatePool(self):
        connection_db = {"host": db_host,
                        "user": db_user,
                        "port": db_port,
                        "passwd": db_passwd,
                        "database": db_name}

        honeypot_pool = mysql.connector.pooling.MySQLConnectionPool(
            pool_name = "honey_pool",
            pool_size = 18,
            pool_reset_session=True,
            autocommit=True,
            **connection_db )

        return honeypot_pool

    def OtherPorts(self, Client_IP, Port):
        try:
            db_connect = self.honeypot_pool.get_connection()
            mycursor = db_connect.cursor()
            mycursor.execute(f"USE {db_name}")
            mycursor.execute(f"INSERT INTO {table_1} (Client_IP, Port, Date, Time) VALUES (%s, %s, %s, %s)", (Client_IP, Port, self.Date, self.Time))
            mycursor.close()
            db_connect.close()
        except mysql.connector.Error:
            pass
```

## Appendix 10: Socket script to open all connections

```
Honeypot > tcp_app.py > Ports1024
1  import socket
2  import threading
3  from Database.InsertData import InsertData
4  from datetime import date
5  import time
6
7  # Class to handle connections
8  class Ports1024:
9      def __init__(self):
10         self.con_insert = InsertData()
11
12     def Openall(self):
13         for port in range(1, 1024):
14             if (port != 22) and (port != 80) and (port != 23) and (port != 21):
15                 honeypot_thread = threading.Thread(target=self.HoneyConnect, args=(port,))
16                 honeypot_thread.start()
17
18     def HoneyConnect(self, port):
19         try:
20             server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
21             server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
22             server_socket.bind(('0.0.0.0', port))
23             server_socket.listen(250)
24
25             while True:
26                 client_socket, client_addr = server_socket.accept()
27                 self.con_insert.OtherPorts(client_addr[0], port)
28                 client_socket.close()
29
30         except Exception as e:
31             print(f"Error: {e}")
32             server_socket.close()
```



## Appendix 11: SSH Server script, called by SSH socket script

```

HoneyPot > Protocols > SSH > ssh_server.py > ...
1  import paramiko
2
3  from .ssh_interface import SSHInterface
4  from .ssh_commands import SSHCommands
5  from .ssh_log import SSHInsert
6
7  class SSHServer():
8      def __init__(self, server_key):
9          self.server_key = server_key
10         self.Command = SSHCommands()
11         self.insert = SSHInsert()
12
13     def ConnectSSH(self, client_socket):
14         transport_data = paramiko.Transport(client_socket)
15         transport_data.add_server_key(self.server_key)
16         ssh = SSHInterface()
17
18         try:
19             transport_data.start_server(server=ssh)
20         except paramiko.SSHException:
21             pass
22
23         channel = transport_data.accept(30)
24
25         if channel is None:
26             raise Exception("No channel")
27
28         try:
29             channel.send("Welcome to Ubuntu 17.04 (GNU/Linux 4.10.0.21-generic X86-64)\r\n")
30             channel.send("\r\n")
31             channel.send("0 Packages can be updated.\r\n")
32             channel.send("0 Updates are security updates.\r\n")
33             channel.send("\r\n")
34
35             while True:
36
37                 channel.send("guest@ssh:/ $ ")
38                 commands = ""
39
40                 while not commands.endswith("\r"):
41                     transport_data = channel.recv(1024)
42                     channel.send(transport_data)
43                     commands += transport_data.decode("utf-8")
44
45                 channel.send("\r\n")
46                 commands = commands.rstrip().lower()
47
48                 self.insert.SSHCommands(commands)
49
50                 if commands == "exit":
51                     channel.send("logout guest\r\n")
52                     channel.shutdown()
53                     channel.close()
54                 else:
55                     self.Command.HandleCommands(commands, channel)
56
57         except Exception:
58             transport_data.close()
59
60
61
62 if __name__ == "__main__":
63     server_key = paramiko.RSAKey.from_private_key_file('RSAkeys/RSAHoney') # RSA key SSH
64     server = SSHServer(server_key)
65     server.ConnectSSH()

```

## Appendix 12: SSH Interface script

```

1  import os
2  import json
3  import paramiko
4
5  from .ssh_log import SSHInsert
6  from .ssh_alert import SSHAlert
7
8  class SSHInterface(paramiko.ServerInterface):
9      def __init__(self):
10         self.insert = SSHInsert()
11         self.alert = SSHAlert()
12
13
14     def check_channel_request(self, kind, chanid):
15         if kind == 'session':
16             return paramiko.OPEN_SUCCEEDED
17
18     def get_allowed_auths(self, username):
19         return "password"
20
21     def check_auth_password(self, username, password):
22         self.insert.SSHLogin(username, password)
23
24         current_directory = os.path.dirname(os.path.abspath(__file__))
25         json_file = os.path.join(current_directory, 'creds.json')
26
27         with open(json_file) as config_file:
28             config = json.load(config_file)
29
30         ssh_login = config['ssh_login']
31         json_username = ssh_login['ssh_username']
32         json_password = ssh_login['ssh_password']
33
34         if (username == json_username) and (password == json_password):
35             self.alert.SucLogin()
36
37             return paramiko.AUTH_SUCCEEDED
38         else:
39             return paramiko.AUTH_FAILED
40
41     def check_auth_publickey(self, username, key):
42         return paramiko.AUTH_FAILED
43
44     def check_channel_shell_request(self, channel):
45         return True
46
47     def check_channel_pty_request(self, channel, term, width, height, pixelwidth, pixelheight, modes):
48         return True

```

Appendix 13: SSH command script used to process commands entered by the user

```

class SSHCommands():
    def __init__(self):
        pass

    def HandleCommands(self, commands, channel):

        response = " "

        if commands == "whoami":
            response = "guest\r\n"

        elif 'cd' in commands:
            response = "permission denied\r\n"

        elif 'rm' in commands:
            response = "permission denied\r\n"

        elif 'mv' in commands:
            response = "permission denied\r\n"

        elif 'cp' in commands:
            response = "permission denied\r\n"

        elif 'cat' in commands:
            response = "permission denied\r\n"

        elif commands == "touch":
            response = "permission denied\r\n"

        elif commands == "mkdir":
            response = "permission denied\r\n"

        elif commands == "mkdir":
            response = "permission denied\r\n"

        elif commands == "pwd":
            response = "/" \r\n"

        elif 'sudo' in commands:
            response = "guest is not in the sudoers file. This incident will be reported.\r\n"

        elif commands == "ls":
            response = " . \r\n .. \r\n bin/ \r\n dev/ \r\n home/ \r\n lost+found/ \r\n mnt/ \r\n pro

        elif commands == "ls -la":
            response = ""total 68 \r
drwxr-xr-x 18 root root 4096 Jan 30 06:16 . \r
drwxr-xr-x 18 root root 4096 Jan 30 06:16 .. \r
lrwxrwxrwx 1 root root 7 Dec 11 01:03 bin -> usr/bin \r
drwxr-xr-x 3 root root 4096 Jan 30 06:42 boot \r
drwxr-xr-x 16 root root 3640 Jan 22 08:41 dev \r
drwxr-xr-x 100 root root 4096 Feb 7 21:33 etc \r
drwxr-xr-x 3 root root 4096 Dec 11 01:07 home \r
lrwxrwxrwx 1 root root 7 Dec 11 01:03 lib -> usr/lib \r
drwx----- 2 root root 16384 Dec 11 01:23 lost+found \r
drwxr-xr-x 2 root root 4096 Dec 11 01:03 media \r
drwxr-xr-x 2 root root 4096 Dec 11 01:03 mnt \r
drwxr-xr-x 4 root root 4096 Jan 2 18:56 opt \r
dr-xr-xr-x 149 root root 0 Jan 1 1970 proc \r
drwx----- 4 root root 4096 Jan 22 21:34 root \r
drwxr-xr-x 25 root root 820 Feb 10 18:59 run \r
lrwxrwxrwx 1 root root 8 Dec 11 01:03/sbin -> usr/sbin \r
drwxr-xr-x 2 root root 4096 Dec 11 01:03 srv \r
dr-xr-xr-x 12 root root 0 Jan 1 1970 sys \r

```

## Appendix 14: Flask application for HTTP protocol

```
HoneyPot > HTTP > app.py > login
1  from flask import Flask, render_template, request
2  from waitress import serve
3
4  from Database.InsertData import InsertData
5  from .http_log import HTTPInsert
6  from .http_alert import HTTPAlert
7
8  app = Flask(__name__)
9  app.secret_key = 'secret'
10
11  con_insert = InsertData()
12  http_insert = HTTPInsert()
13  http_alert = HTTPAlert()
14
15  @app.route('/', methods=['GET', 'POST'])
16  def login():
17      if request.method == 'POST':
18          username = request.form['username']
19          password = request.form['password']
20          address = request.remote_addr
21
22          http_alert.AccessAttempt(address, username, password)
23
24          http_insert.HTTPLogin(address, username, password)
25          con_insert.OtherPorts(address, 80)
26
27      return render_template('attacker_login.html')
28
29  if __name__ == "__main__":
30      serve(app, host="0.0.0.0", port=80)
```

## Appendix 15: Telnet server script 1

```
56         return password.strip()
57
58     def GetCommand(self, client_socket):
59
60         while True:
61             command = ""
62             client_socket.send(b"guest@telnet: $> ")
63             while not command.endswith("\n"):
64
65                 data = client_socket.recv(1024)
66                 if not data or data == b'\r\n':
67
68                     command = command.lower().strip()
69                     self.insert.TelnetCommands(command)
70
71                     if command == 'exit':
72                         data = ''
73                         return False
74                     else:
75                         self.commands.HandleCommands(command, client_socket)
76
77             command += data.decode("utf-8")
78
79     def start(self, client_socket, attempts):
80
81         self.attempts = attempts
82
83         while self.attempts < 3:
84
85             username = self.Username(client_socket)
86             password = self.password(client_socket)
87             self.insert.TelnetLogin(username, password)
88             self.AuthCreds(username, password, client_socket)
89             self.attempts += 1
90         else:
91
92             client_socket.send(b"Connection Close")
93             client_socket.close()
```

## Appendix 16: Telnet server script 2

```

10 class TelnetServer():
11     def __init__(self):
12         self.insert = TelnetInsert()
13         self.commands = TelnetCommands()
14         self.alert = TelnetAlert()
15         self.attempts = 0
16
17     def AuthCreds(self, username, password, client_socket):
18
19         current_directory = os.path.dirname(os.path.abspath(__file__))
20         json_file = os.path.join(current_directory, 'creds.json')
21
22         with open(json_file) as config_file:
23             config = json.load(config_file)
24
25         telnet_login = config['telnet_login']
26
27         json_username = telnet_login['telnet_username']
28         json_password = telnet_login['telnet_password']
29
30         if (username == json_username) and (password == json_password):
31             client_socket.send(b"Welcome to Ubuntu 17.04 (GNU/Linux 4.10.0.21-generic X86-64)\r\n")
32             self.GetCommand(client_socket)
33             self.alert.SucLogin()
34             self.attempts = 5
35             client_socket.send(b"\n\rIncorrect Login\n\r")
36
37     def Username(self, client_socket):
38
39         username = ""
40         client_socket.send(b"Login: ")
41         while True:
42             data = client_socket.recv(1024)
43             if not data or data == b'\r\n':
44                 break
45             username += data.decode("utf-8")
46         return username.strip()
47
48     def password(self, client_socket):
49
50         password = ""
51         client_socket.send(b"Password: ")
52         while True:
53             data = client_socket.recv(1024)
54             if not data or data == b'\r\n':
55                 break
56             password += data.decode("utf-8")
57         return password.strip()

```

Appendix 17: FTP server script to login

```
1  from .ftp_log import FTPInsert
2  from .ftp_alert import FTPAlert
3
4  class FTP:
5      def __init__(self):
6          self.con_insert = FTPInsert()
7          self.alert = FTPAlert()
8
9      def Login(self, client_socket, client_addr):
10         client_socket.sendall(b'220 Welcome to the FTP!\r\n')
11
12         username = ''
13         password = ''
14
15         while True:
16             data = client_socket.recv(1024).decode('utf-8')
17
18             if not data:
19                 break
20
21             if data.startswith('USER'):
22                 client_socket.sendall(b'331 Password required for user.\r\n')
23                 username = data.strip()
24                 username = username.replace("USER", "")
25             elif data.startswith('PASS'):
26                 password = data.strip()
27                 password = password.replace("PASS", "")
28             else:
29                 client_socket.sendall(b'\r\n')
30
31
32             if username != '' and password != '':
33                 self.con_insert.FTPLogin(username, password)
34                 client_socket.sendall(b'530 Login authentication failed\r\n')
35                 client_socket.close()
36                 self.alert.AccessAttempt(client_addr, username, password)
```

## Appendix 18: Script to send an email using Mmetext

```

Alert > alert.py > Alert > Emailsend
1  import smtplib
2  import os
3  import json
4
5  import smtplib
6  from email.mime.text import MIMEText
7  from email.mime.multipart import MIMEMultipart
8
9  class Alert():
10     def __init__(self):
11         pass
12
13     def EmailInfo(self, subject, info):
14         current_directory = os.path.dirname(os.path.abspath(__file__))
15         json_file = os.path.join(current_directory, 'Email.json')
16
17         with open(json_file) as email_file:
18             email = json.load(email_file)
19
20             address = email['EmailAddress']
21             email_address = address["user_email"]
22             sender = "autohoneypotalerts@gmail.com"
23             recipient = email_address
24             password = "cogm qcpi bsky kvcz"
25             self.Emailsend(subject, info, sender, recipient, password)
26
27     def Emailsend(self, subject, info, sender, recipient, password):
28         msg = MIMEMultipart()
29         msg['Subject'] = subject
30         msg['From'] = sender
31         msg['To'] = recipient
32         msg.preamble
33
34         body = MIMEText(info)
35         msg.attach(body)
36         try:
37             server = smtplib.SMTP('smtp.gmail.com', 587)
38             server.starttls()
39             server.login(sender, password)
40             server.sendmail(sender, recipient, msg.as_string())
41             server.quit()
42         except:
43             pass

```



*Appendix 19: Info and subject that is sent to the alert script*

```
HoneyPot > SSH > ssh_alert.py > ...
1  from datetime import date
2  import time
3
4  from Alert.alert import Alert
5
6  class SSHAlert():
7      def __init__(self):
8          self.alert = Alert()
9          self.Date = date.today()
10         local = time.localtime()
11         self.Time = time.strftime("%H:%M:%S", local)
12
13     def SucLogin(self):
14         subject = "SSH Successful Accessed"
15         info = info = (f"SSH was successful accessed by {self.Date} at {self.Time}. View the web application for more information")
16         self.alert.EmailInfo(subject, info)
17
18     def AccessAttempt(self, client_addr):
19         subject = "SSH Attempt Accessed"
20         info = info = (f"SSH was accessed by {client_addr} on {self.Date} at {self.Time}")
21         self.alert.EmailInfo(subject, info)
```

## Appendix 20: PHP login function when an account is already created

```

116
117 public function LoginPage()
118 {
119     $title = "<h1> HoneyPot Login </h1>";
120     $info = "";
121     $error_username = "";
122     $error_password = "";
123
124     if ($_SERVER["REQUEST_METHOD"] == "POST") {
125
126         $username = $_POST['username'] ?? '';
127         $password = $_POST['password'] ?? '';
128
129         if (empty($username) || empty($password)) {
130             $info = "<p1>Username and Password Required</p1>";
131         }
132         else{
133             $sanitized_username = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_SPECIAL_CHARS);
134             $sanitized_password = filter_input(INPUT_POST, 'password', FILTER_SANITIZE_SPECIAL_CHARS);
135
136             if ($sanitized_username !== $username || $sanitized_password !== $password) {
137                 $info = "<p1> Invalid Credentials </p1>";
138                 return [$info, $title, include('HTML/login.html')];
139             }
140             else {
141                 try{
142                     $stmt = $this->dbConn->prepare("SELECT * FROM web_account WHERE username =:username");
143                     $stmt->bindParam(':username', $username, PDO::PARAM_STR);
144                     $stmt->execute();
145                     $result = $stmt->fetch(PDO::FETCH_ASSOC);
146                 } catch (PDOException $e) {
147                     echo 'Command execution failed: ' . $e->getMessage();
148                 }
149
150                 if (!isset($result['password'])) {
151                     $info = "<p1>Incorrect Credentials</p1>";
152                     return [$info, $title, include('HTML/login.html')];
153                 }
154                 if (password_verify($password, $result['password'])) {
155                     header("Location: index.php");
156                     $_SESSION['user_id'] = $result['ID'];
157                 }
158                 else{
159                     $info = "<p1>Incorrect Credentials</p1>";
160                 }
161             }
162         }
163     }
164 }
165 include('HTML/login.html');
166
167 }
168

```

## Appendix 21: register account when there is no account

```

43 }
44 public function Register(){
45     $title = "<h1> Register </h1>";
46     $info = "";
47     $error_username = "";
48     $error_password = "";
49 }
50 if ($_SERVER["REQUEST_METHOD"] == "POST") {
51
52     $username = $_POST['username'] ?? '';
53     $password = $_POST['password'] ?? '';
54
55     if (empty($username) || empty($password)) {
56         $info = "<p1>Username and Password Required</p1>";
57         return [$info, $title, include('HTML/login.html')];
58     }
59     else{
60         $sanitized_username= filter_input(INPUT_POST,'username',FILTER_SANITIZE_SPECIAL_CHARS);
61         $sanitized_password= filter_input(INPUT_POST,'password',FILTER_SANITIZE_SPECIAL_CHARS);
62         $password_length = strlen($sanitized_password);
63
64         if ($sanitized_username != $username || $sanitized_password != $password ){
65             $error_password = "<p1> *Use of illegal Characters</p1>";
66             return [$error_password, $title, include('HTML/login.html')];
67         }
68         elseif($password_length <= 8){
69             $error_password = "<p1> *Password Should be more then 8 characters</p1>";
70             return [$error_password, $title, include('HTML/login.html')];
71         }
72         elseif(!ctype_alnum($sanitized_username) ){
73             $error_username = "<p1> *Username Should only contain Characters & Numbers </p1>";
74             return [$error_username, $title, include('HTML/login.html')];
75         }
76         elseif(!preg_match("/\d/", $sanitized_password)) {
77             $error_password = "<p1> *Should contain at least one digit</p1>";
78             return [$error_password, $title, include('HTML/login.html')];
79         }
80         elseif(!preg_match("/\W/", $sanitized_password)) {
81             $error_password = "<p1> *Should contain at least one Special Character</p1>";
82             return [$error_password, $title, include('HTML/login.html')];
83         }
84         else {
85             $hashed_password = password_hash($password, PASSWORD_DEFAULT);
86             try{
87                 $stmt = $this->dbConn->prepare("INSERT INTO web_account (Username, password) VALUES (:username, :password)");
88                 $stmt->bindParam(':username', $username, PDO::PARAM_STR);
89                 $stmt->bindParam(':password', $hashed_password);
90             } catch (PDOException $e) {
91                 echo 'Command execution failed: ' . $e->getMessage();
92             }
93             $stmt->execute();
94             header("Location: login.php");
95         }
96     }
97 }
98 include('HTML/login.html');
99 }

```

Appendix 22: Display the content to a web page

```
75
76 public function ViewData(){
77
78     try {
79         $stmt = $this->dbConn->prepare("SELECT * from ssh_login");
80         $stmt->execute();
81         $result = $stmt->fetchAll(PDO::FETCH_ASSOC);
82         foreach ($result as $row) {
83             echo "<tr>
84                 <td>".htmlspecialchars($row['ID'])."</td>
85                 <td>".htmlspecialchars($row['Username'])."</td>
86                 <td>".htmlspecialchars($row['Password'])."</td>
87                 <td>".htmlspecialchars($row['Date'])."</td>
88                 <td>".htmlspecialchars($row['Time'])."</td>
89             </tr>";
90         }
91         echo "</table>";
92     }
93     catch (PDOException $e) {
94         echo 'Command execution failed: ' . $e->getMessage();
95     }
96 }
97
98 }
99
```

Appendix 23: Modify the information in the JSON file

```
239
240 public function ChangeDatabaseInfo($db_host, $db_port, $db_user, $db_passwd){
241     $json_file = file_get_contents('../Database/Config/db_config.json');
242     $json_data = json_decode($json_file, true);
243
244     if (!empty($db_host)){
245         $json_data['database_config']['db_host'] = $db_host;
246         $json_object = json_encode($json_data, JSON_PRETTY_PRINT);
247         file_put_contents('../Database/Config/db_config.json', $json_object);
248     }
249
250     if (!empty($db_port)){
251         $json_data['database_config']['db_port'] = $db_port;
252         $json_object = json_encode($json_data, JSON_PRETTY_PRINT);
253         file_put_contents('../Database/Config/db_config.json', $json_object);
254     }
255
256     if (!empty($db_user)){
257         $json_data['database_config']['db_user'] = $db_user;
258         $json_object = json_encode($json_data, JSON_PRETTY_PRINT);
259         file_put_contents('../Database/Config/db_config.json', $json_object);
260     }
261
262     if (!empty($db_passwd)){
263         $json_data['database_config']['db_passwd'] = $db_passwd;
264         $json_object = json_encode($json_data, JSON_PRETTY_PRINT);
265         file_put_contents('../Database/Config/db_config.json', $json_object);
266     }
267
268 }
```

## Appendix 24:: Open and close the python scripts

```

90 public function SSHControl(){
91     $current_user = $this->user_account;
92
93     $ssh_cookie = isset($_COOKIE['sshcookie']) && $_COOKIE['sshcookie'] == 'on' ? 'checked' : '';
94
95     $json_file = file_get_contents("C:\\Users\\{$current_user}\\HoneyPot\\LaunchFiles\\pids.json");
96     $json_data = json_decode($json_file, true);
97     $ssh_pid = $json_data['pythonpid']['sshpipid'];
98
99     if ($_SERVER["REQUEST_METHOD"] == "POST") {
100         $open_ssh = isset($_POST['SSHOpen']) ? $_POST['SSHOpen'] : 'off';
101         setcookie("sshcookie", $open_ssh, time()+3600);
102         $ssh_cookie = $open_ssh == 'on' ? 'checked' : '';
103
104         if ($ssh_cookie == 'checked'){
105             if ($ssh_pid == -1){
106                 $command = 'start /b python.exe "C:\\Users\\'.$current_user.'\\HoneyPot\\LaunchFiles\\LaunchSSH.py" > log2.txt 2>&1';
107                 pclose(popen($command, 'r'));
108             }
109         }
110         elseif($ssh_pid != '-1'){
111             exec("taskkill /PID $ssh_pid /F");
112             $json_data['pythonpid']['sshpipid'] = -1;
113             $json_object = json_encode($json_data, JSON_PRETTY_PRINT);
114             file_put_contents("C:\\Users\\{$current_user}\\HoneyPot\\LaunchFiles\\pids.json", $json_object);
115         }
116     }
117
118     return $ssh_cookie;
119 }

```