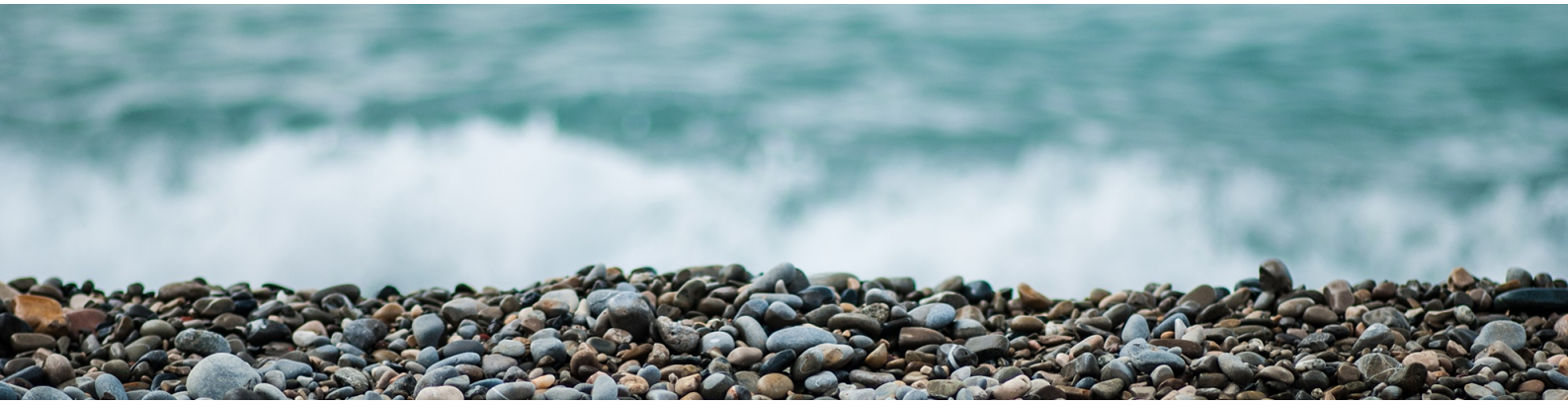
Data Security - 24/25

Dinesh Kumar SHANMUGANATHAN - University of Law
Created: 21-OCT-24 **Last Modified:** 04-FEB-25

PebblePad

Contents

Cover Sheet



FRONT SHEET

Individual Coursework

CANDIDATE NUMBER (C-NUMBER)

C2109911

MODULE NAME

Data Security

WORD COUNT

1796

SUBMISSION DATE

04/02/2025

DECLARATION

I certify that this assessment submission is entirely my work and I have fully referenced and correctly cited the work of others, where required. I also confirm the contents of my submission have not been generated by a third party, or through an Artificial Intelligence generative system*.

I have read the Student Discipline Regulations (Student Discipline Regulations) and understand any Assessment Related Offence/ Academic Misconduct may result penalties being applied.

By submitting this assessment submission, I am confirming that I am fit to sit according to the Assessment Regulations.

I declare that: This is my own unaided work.

Yes

No

The word count stated by me is correct

Yes

No

I’m happy for my work to be retained on the Elite repository and made available to staff and future students**

Yes

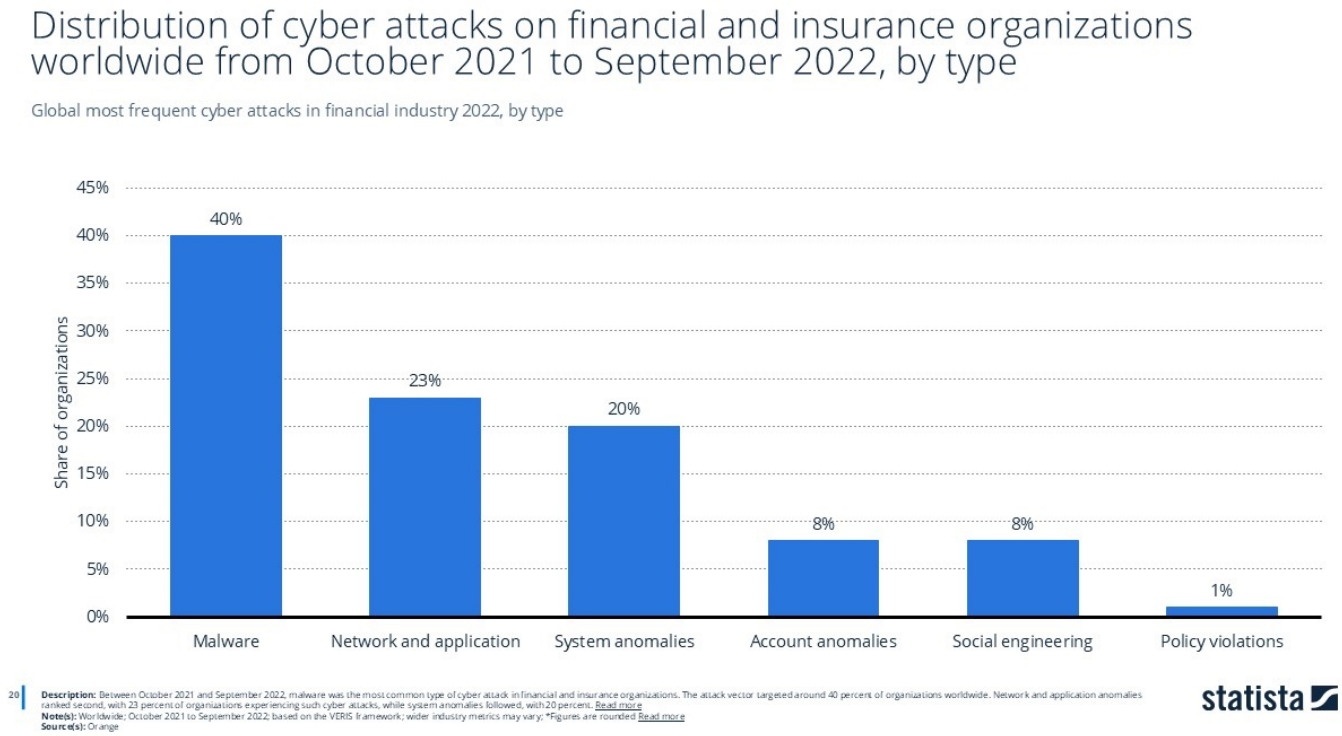
No

*Please note that all the assignments are submitted to Turnitin.

**Please note personal information (such as names) will be deleted

Introduction

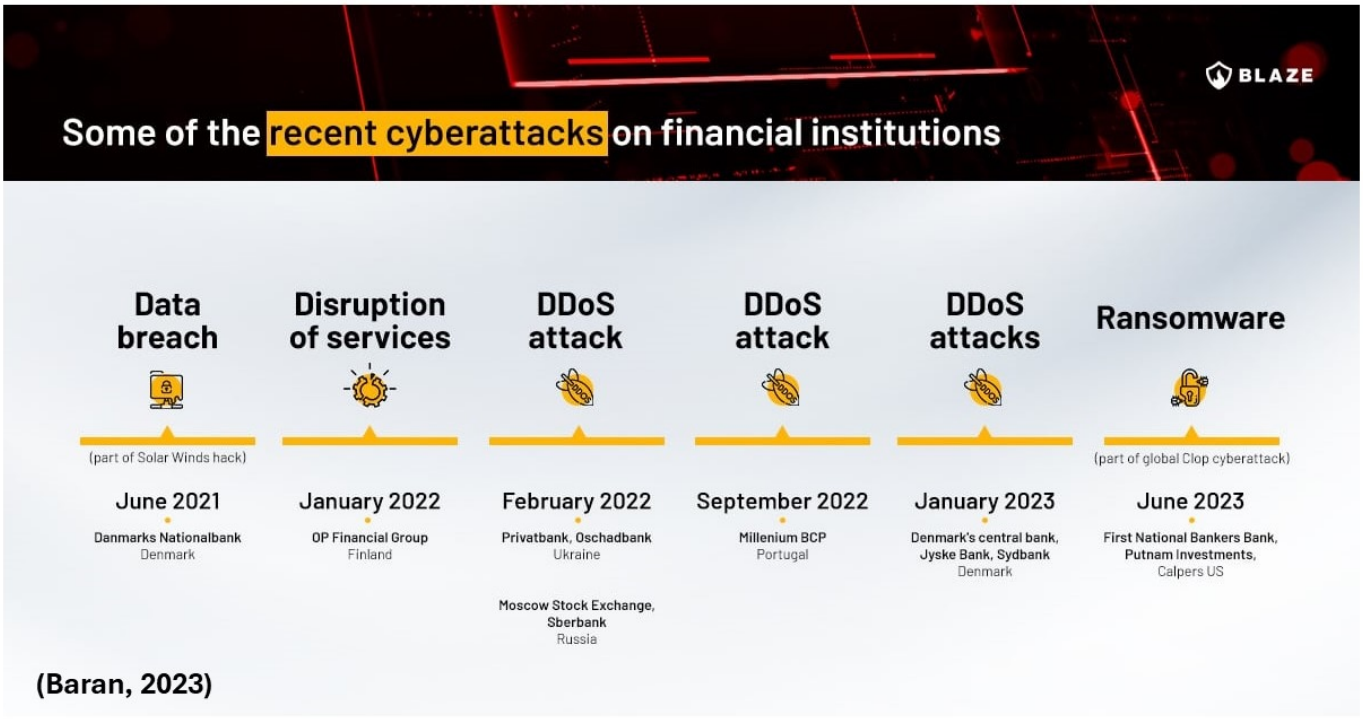
eSecurityFin Limited is a financial services company which collects and manages a significant number of users' personal and financial data. Over the past year, there has been several security incidents like Data Breach, Phishing Attacks, and Insider threat. These incidents suggest several poor security practices.



Analysis of Security Risks:

Weak credentials and poorly secured passwords expose systems to brute force and credential-stuffing attacks, compromising data confidentiality. Vulnerable software, including unpatched systems, poses risks like Remote Code Execution (RCE) and SQL Injection, enabling malware, ransomware, and DDoS attacks (America's Cyber Defense Agency, 2024). Social engineering exploits human error, with phishing attacks targeting employees and bypassing technical defenses. Insider threats, including disgruntled or compromised employees, increase the risk of unauthorized access or accidental damage. These risks undermine the organization's ability to protect sensitive financial data, necessitating robust technical controls, proactive monitoring, and comprehensive employee training to address the gaps effectively.

The above artifact, "Cyber Crime and the Financial Industry in the United States," highlights the widespread prevalence of cyberattacks within the financial sector. It visually underscores how vulnerabilities such as weak credentials, unpatched software, and social engineering are frequently exploited, aligning with the risks identified in this analysis. By illustrating the significant impact of these risks, this artifact emphasizes the urgency of implementing robust controls and proactive measures.



Critically Analyzing Current Security Practices and Their Effectiveness

Recent incidents reveal eSecureFin's inadequate security measures. Weak encryption methods expose sensitive data, with OWASP ranking cryptographic failures as a leading breach cause (OWASP, 2024). Poor employee training leaves staff vulnerable to phishing attacks targeting personal channels (Lenaerts-Bergmans, 2023). Reliance on single-factor authentication enables attackers to exploit weak passwords for critical access (Anon., 2024).

The artifact outlines major cyberattacks on financial institutions, including data breaches, DDoS attacks, and ransomware, highlighting vulnerabilities from weak encryption and human error. These recurring threats reinforce the urgency for enhanced encryption protocols, continuous training, and multi-factor authentication to strengthen eSecureFin's defenses and prevent similar breaches

Security gaps and recommendations

Identifying security gaps and recommendations based on CIA Triad
Confidentiality:

Security gap:

Weak credentials and poor access control practices lead to risks in accessing financial data to threat actors.

Recommended mitigations:

- Strong credentials which are long, with random characters would make them unique and hard to brute force into. For example, using a password generator or a passphrase (America's Cyber Defense Agency, 2024).
- Multi-factor Authentication (MFA) paired with Single Sign-On (SSO) provides an additional layer of security for passwords.
- Use password throttling and password deny lists as a technical measure (National Cyber Security Centre, 2018).
- Encrypting financial information and passwords using a strong algorithm like AES ensures confidentiality event in the event of a data breach.

Integrity:

Security gap:

Vulnerable and outdated software poses the risk of sensitive financial and personal data to unauthorized modifications.

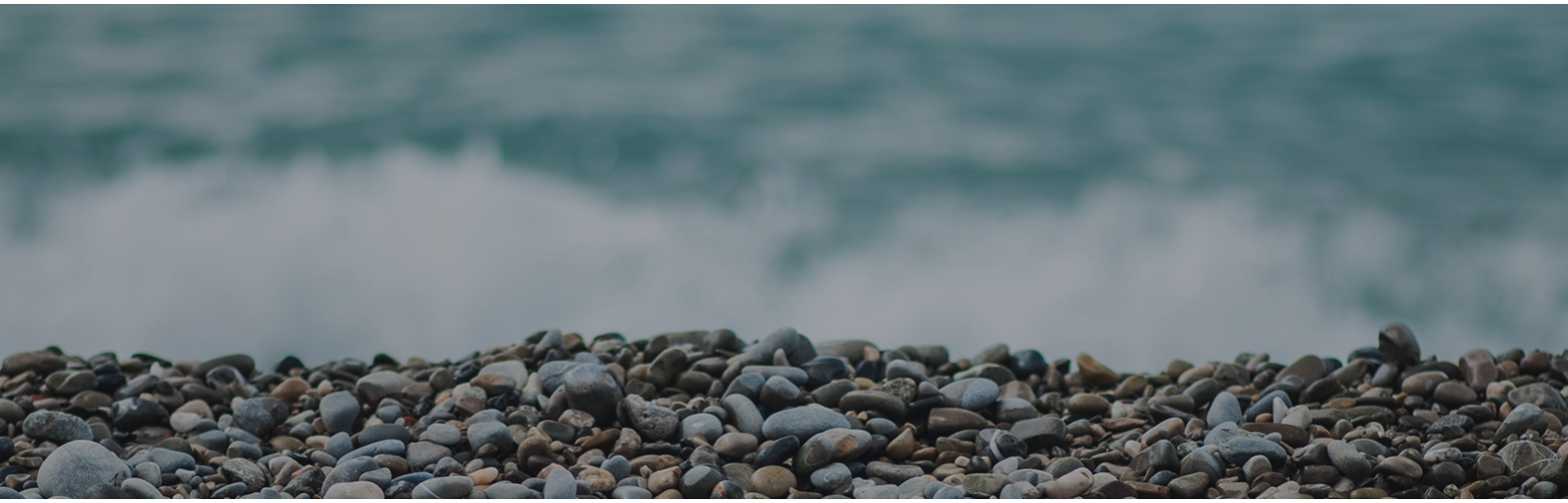
Recommended mitigations:

- Use hashing algorithms like SHA-256 validates that data is unchanged by unauthorized users.
- Audit trials help keep track of log files of who accessed or modified that data helping identify unauthorized access.
- Strict access control policies allowing only authorized personnel to access the data ensure mitigate unauthorized access (Anyanwu, et al., 2024).


```
import hashlib
import os
import base64
import hmac
import json
from cryptography.fernet import Fernet
import pyotp


#Generate and save a secret key for encryption
def generate_key():
    key = Fernet.generate_key()
    with open("secret.key","wb") as key_file:
    key_file.write(key) # Save the key securely return key

#Load the secret key

def load_key():
     return open("secret.key","rb").read()

# Encrypt data

def encrypt_data(data, key):
    f = Fernet(key)
    encrypted_data = f.encrypt(data.encode()) # Convert string to bytes before encryption                                    return
encrypted_data

#Decrypt data

def decrypt_data(encrypted_data, key):
    f = Fernet(key)
    decrypted_data = f.decrypt(encrypted_data).decode() # Convert bytes back to string after decryption                      return
decrypted_data

# Source input validation

def validate_input(data):
    if not isinstance(data, str) or any(char in data for char in [';','--','"',\'"]):
        raise ValueError("Invaid Input detected") # Prevent malicious characters                                                 return
```

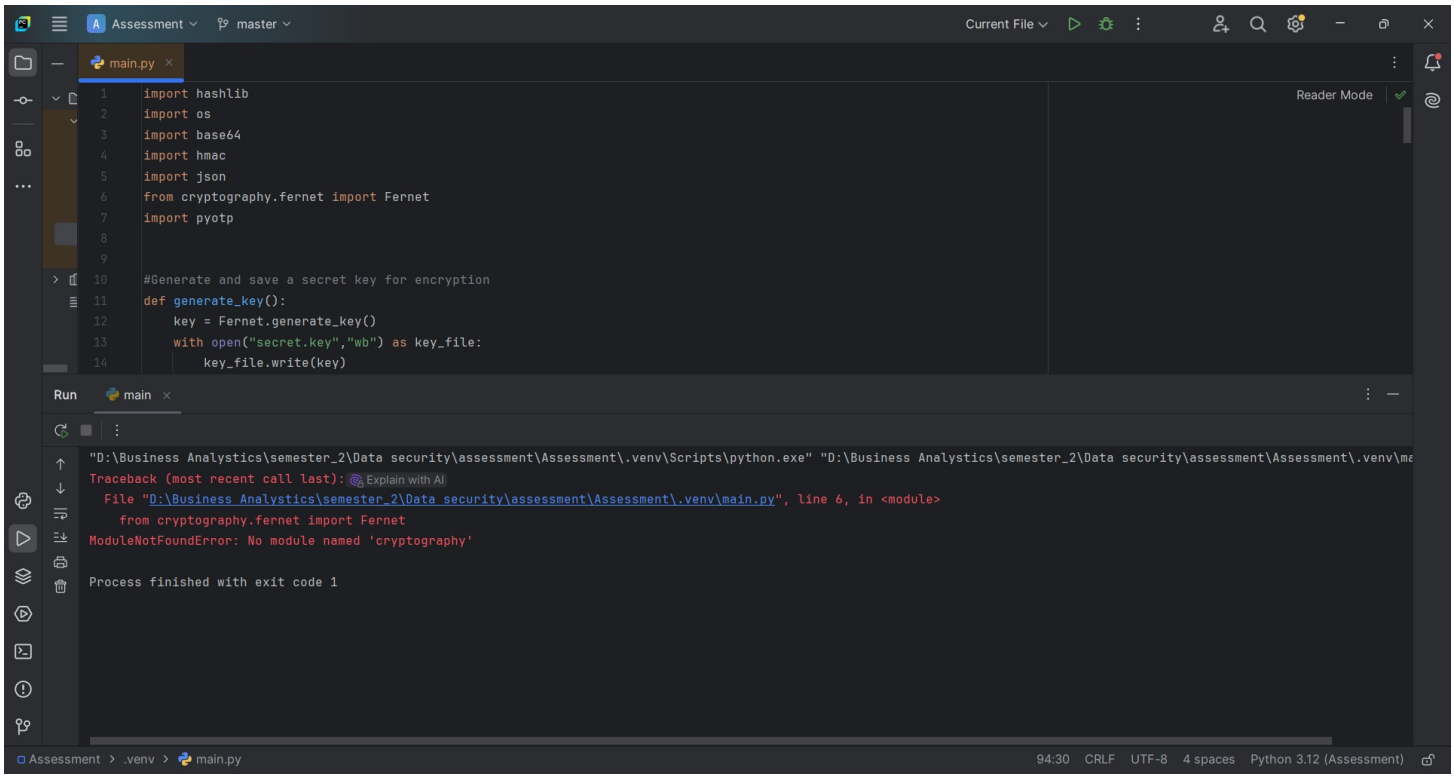
```
data

# Hash password securely

def hash_password(password):
    salt = os.urandom(32)
    hashed_pw = hashlib.pbkdf2_hmac('sha256', password.encode(), salt,1000000) # Apply hashing                    return
base64.b16decode(salt+hashed_pw).decode() # Encode to a storable format



# verify hashed passwords

def verify_password(stored_hash, password):
    stored_hash = base64.b64decode(stored_hash) # Decode stored hash                                                salt,
stored_pw = stored_hash[:32], stored_hash[32:] # Extract salt and hash                                          new_hash =
hashlib.pbkdf2_hmac('sha256',password.encode(), salt, 100000)
    return hmac.compare_digest(stored_pw,new_hash)  # Securely compare hashes



# Generate OTP secret for MFA

def generate_otp_secret():
    return pyotp.random_base32()

# Generate OTP code

def get_otp_code(secret):
    totp = pyotp.TOTP(secret)
    return totp.now()

# Verify otp code

def verify_otp(secret, otp_code):
    totp = pyotp.TOTP(secret)
    return totp.verify(otp_code)



if __name__=="__main__":

# Key management: Ensure an encryption key is available

    if not os.path.exists("secret.key"):
        key = generate_key()
    else:
        key = load_key()



#Secure data processing

    sensitive_data = "Client Financial data" encrypted = encrypt_data(sensitive_data, key)
    decrypted = decrypt_data(encrypted,key)


    print(f"Encrypted: {encrypted}")
    print(f"Decrypted: {decrypted}")
```

```
#MFA implementation

    otp_secret = generate_otp_secret()
    print("Your OTP secret (store securely): ",otp_secret)
    otp_code = get_otp_code(otp_secret)
    print("Generated OTP Code (for testing):", otp_code)


    user_otp = input("Enter OTP: ")
    if verify_otp(otp_secret, user_otp):
        print("OTP verified successfully!")
    else:
        print("Invalid OTP!")
```

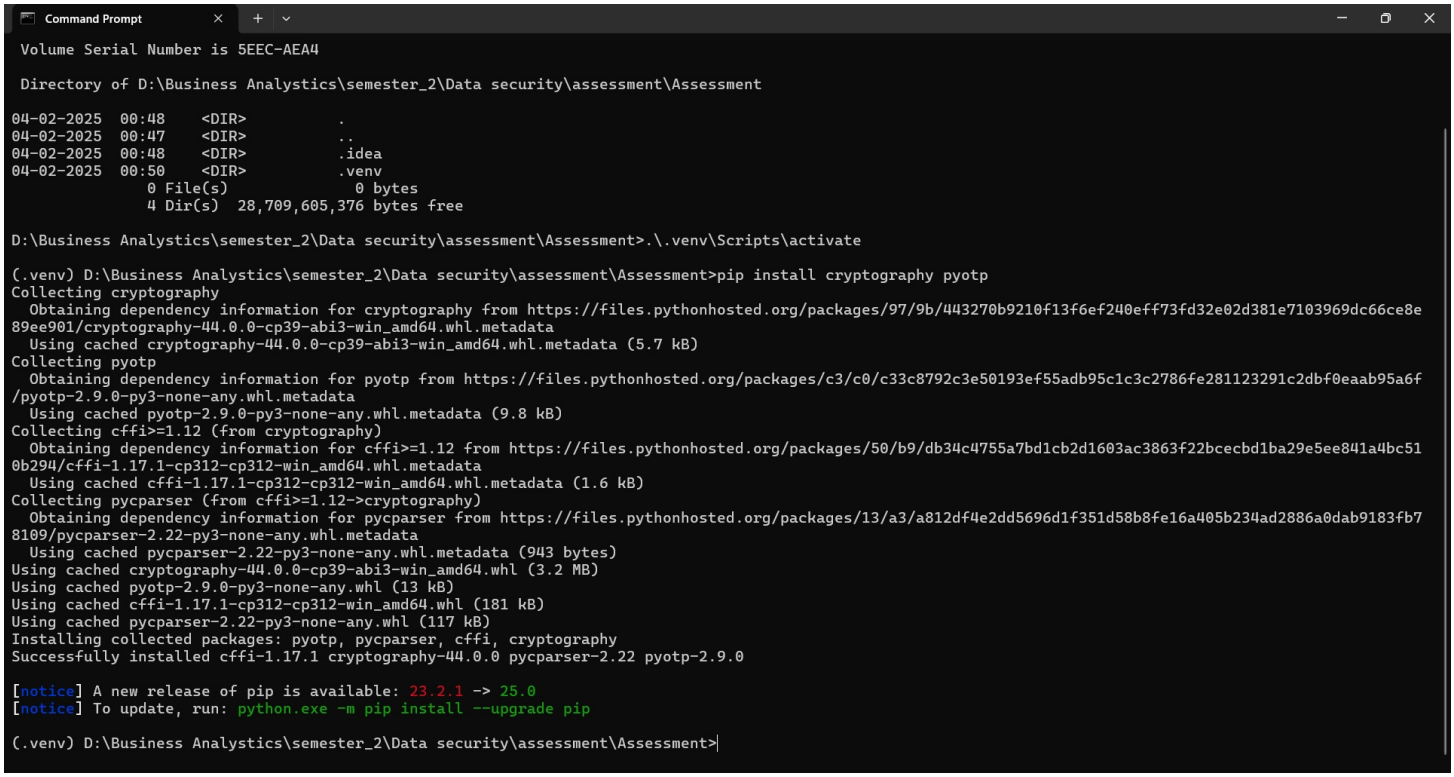


Errors and correction processes:

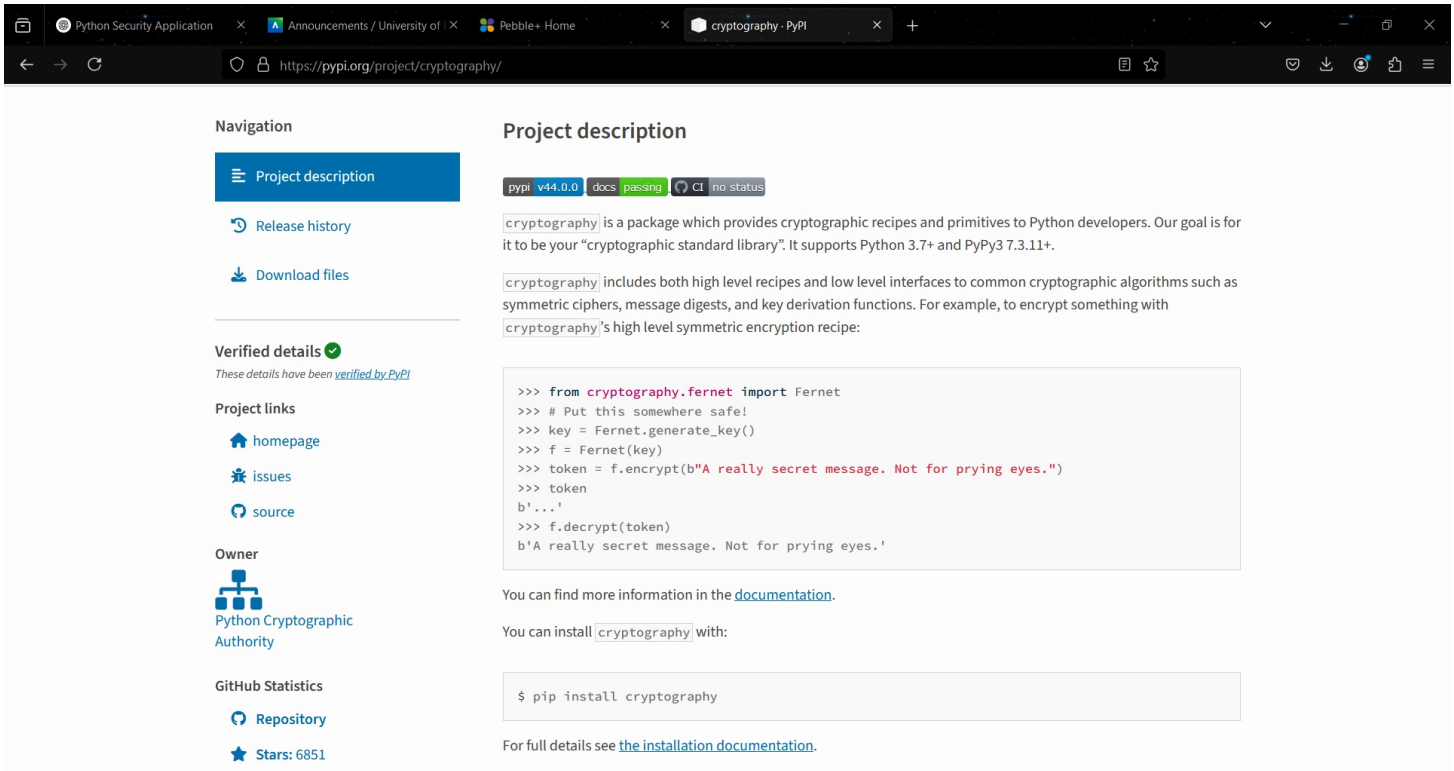
1. Error: Cryptography module not installed



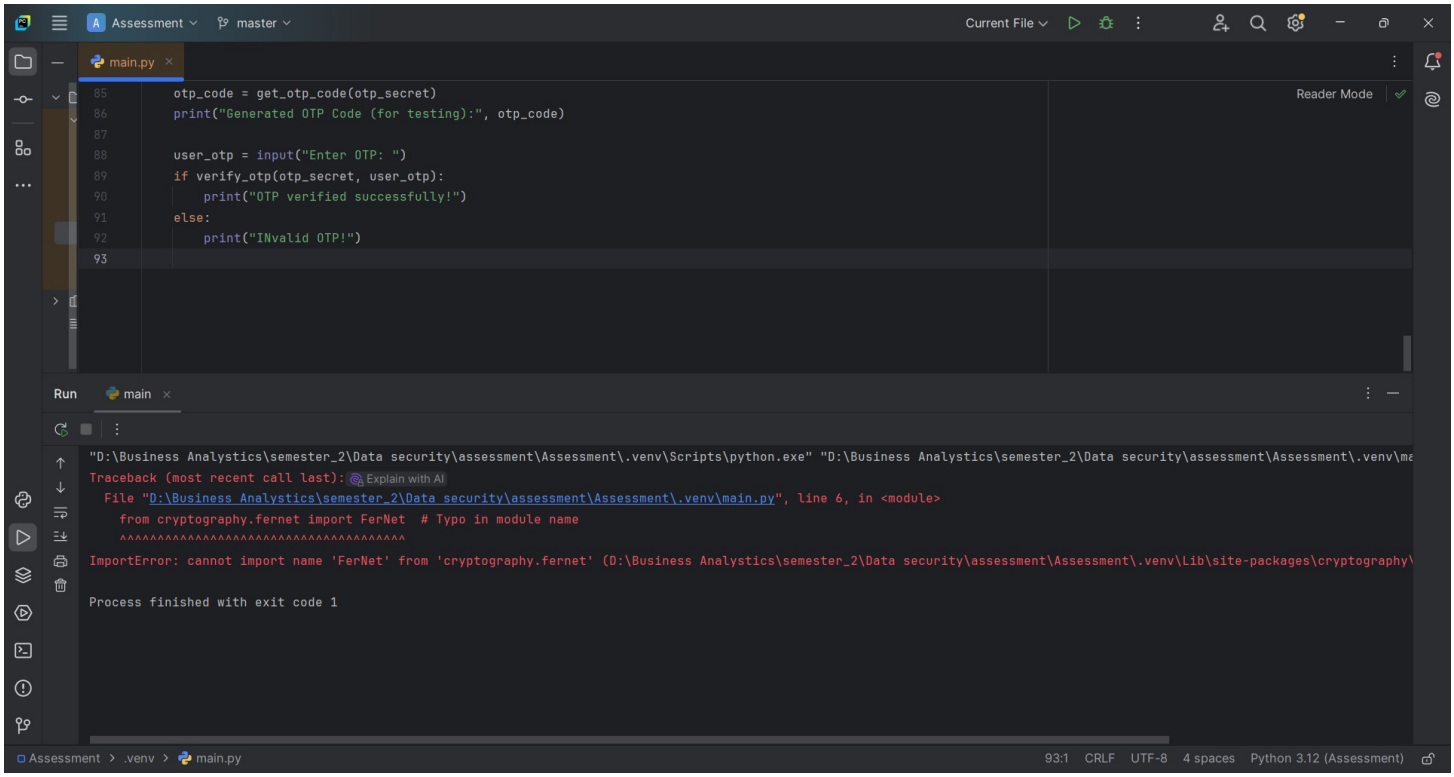
Solution: Module installation process.



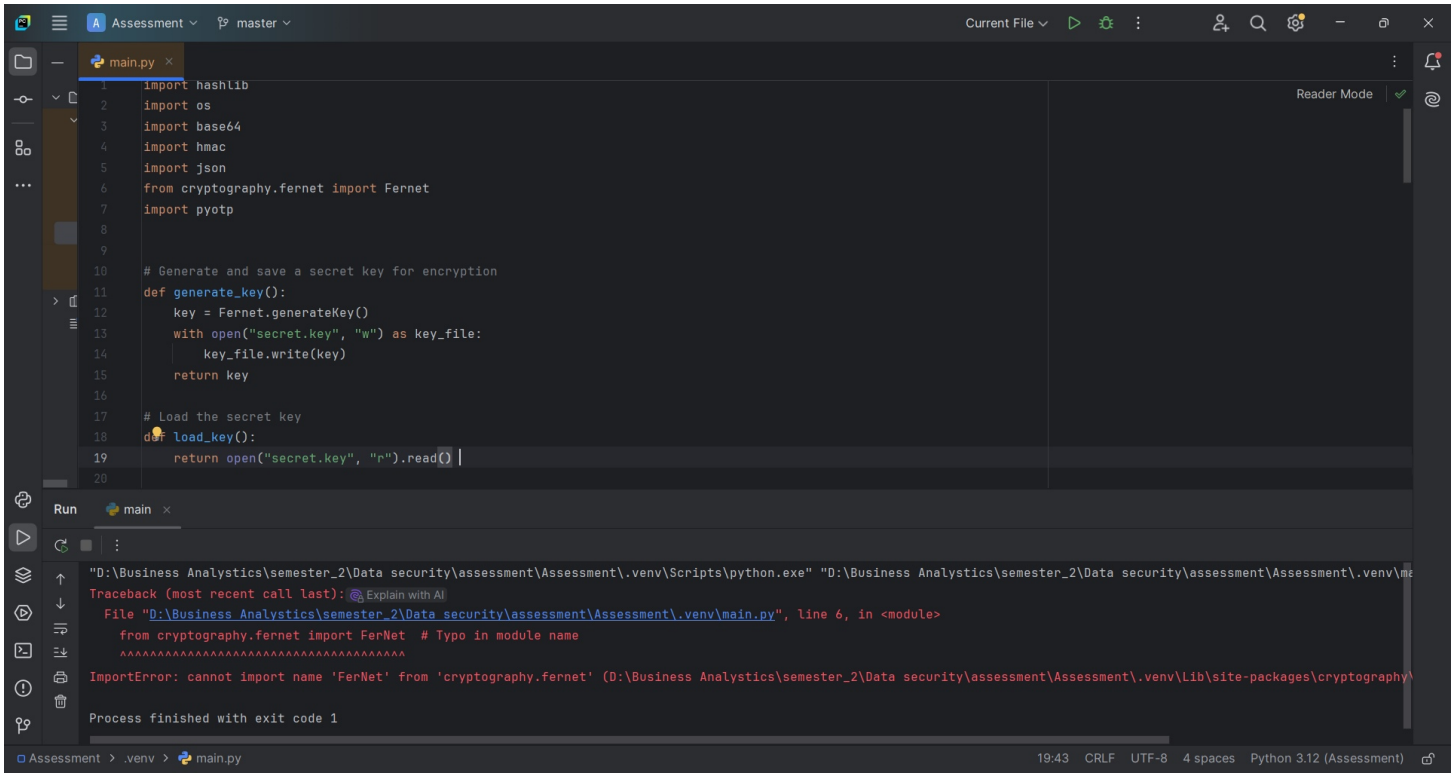
Source used to consult for solutions.



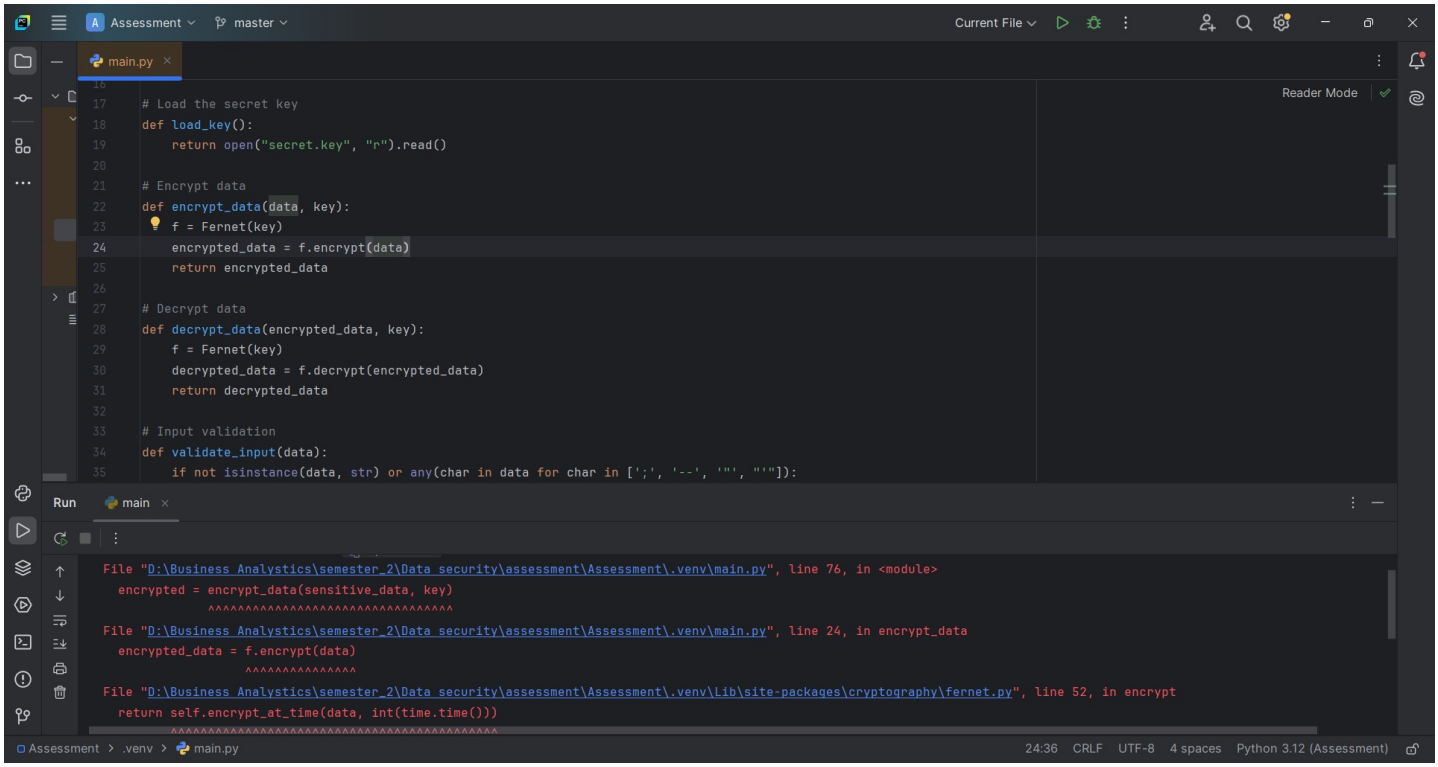
2. Error: Typo in module name


Assessment
master
Current File
main.py
Reader Mode
otp_code = get_otp_code(otp_secret)
print("Generated OTP Code (for testing):", otp_code)
user_otp = input("Enter OTP: ")
if verify_otp(otp_secret, user_otp):
print("OTP verified successfully!")
else:
print("INvalid OTP!")
Run
main
"D:\Business Analystics\semester_2\Data security\assessment\Assessment\.venv\Scripts\python.exe" "D:\Business Analystics\semester_2\Data security\assessment\Assessment\.venv\ma
Traceback (most recent call last):
Explain with AI
File "D:\Business Analystics\semester_2\Data security\assessment\Assessment\.venv\main.py", line 6, in <module>
from cryptography.fernet import FerNet # Typo in module name
ImportError: cannot import name 'FerNet' from 'cryptography.fernet' (D:\Business Analystics\semester_2\Data security\assessment\Assessment\.venv\Lib\site-packages\cryptography\
Process finished with exit code 1
Assessment
.venv
main.py
93:1
CRLF
UTF-8
4 spaces
Python 3.12 (Assessment)

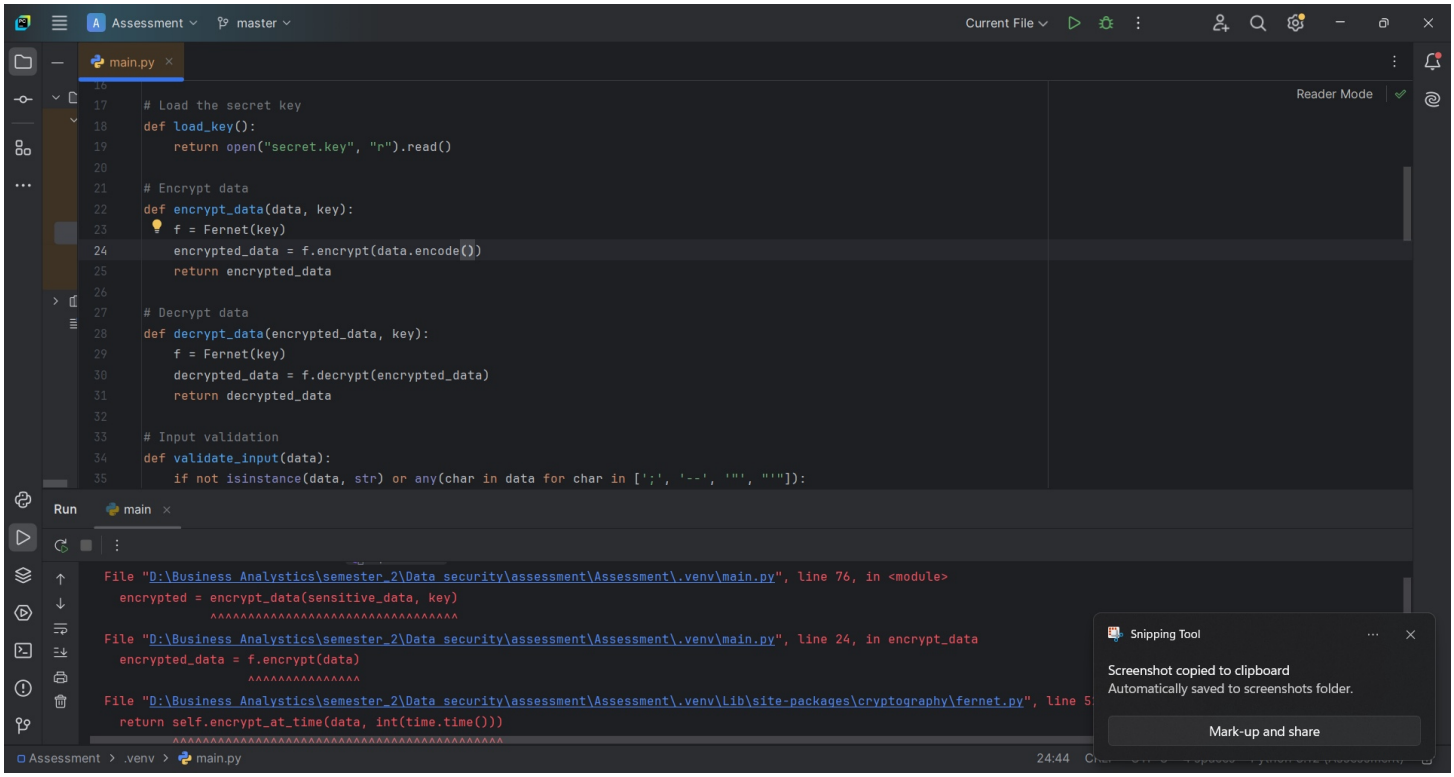
Solution:



3. Error: The encrypted data wasn't properly encoded.



Solution: Use of encode() method to encode the encrypted data.



A personal project to demonstrate python knowledge:
This code implements a **simple chat room** with both a **server** and multiple **clients** using Python's socket and threading modules.

Server Side:

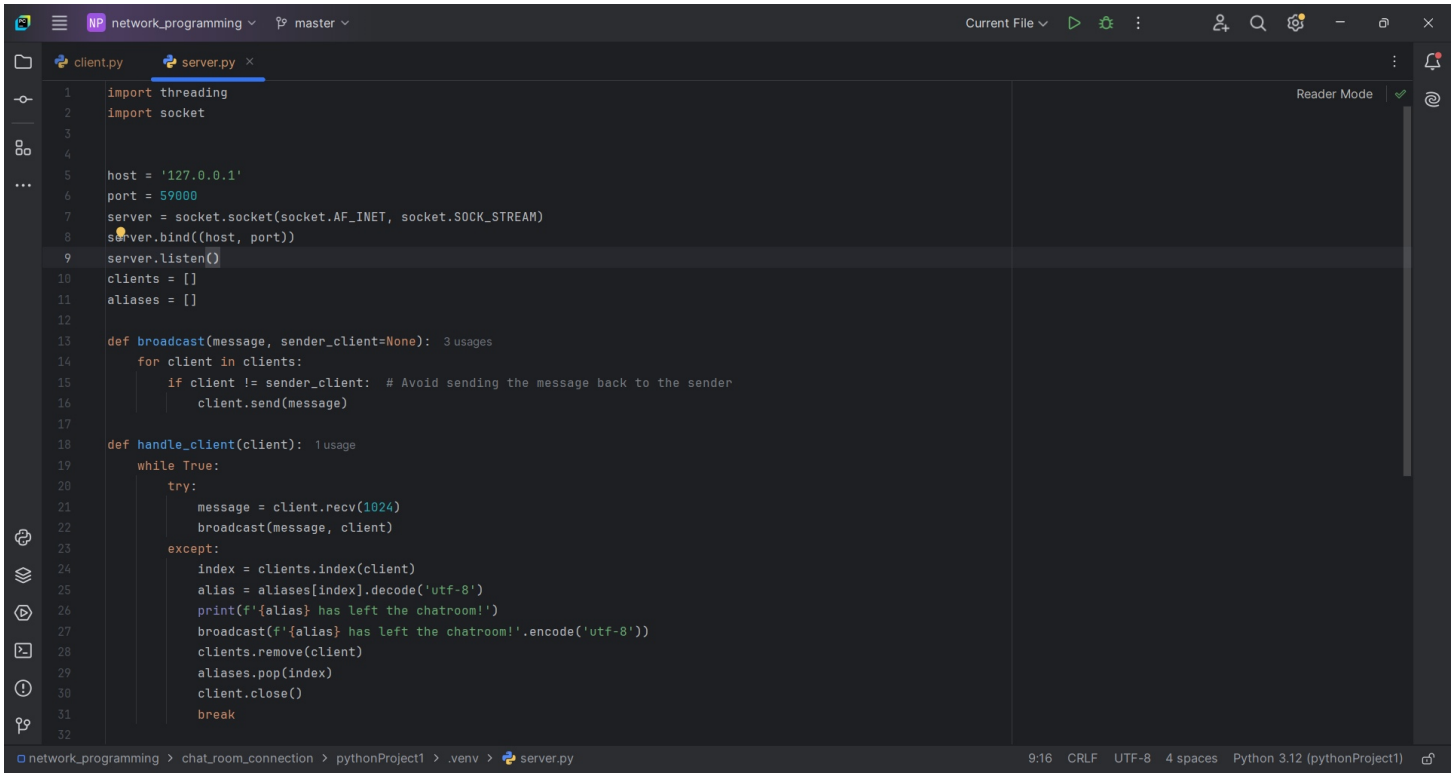
- Listens for incoming client connections.
- Asks each client for an alias (username).
- Stores connected clients and their aliases.
- Broadcasts messages to all clients except the sender.
- Removes disconnected clients and notifies others.

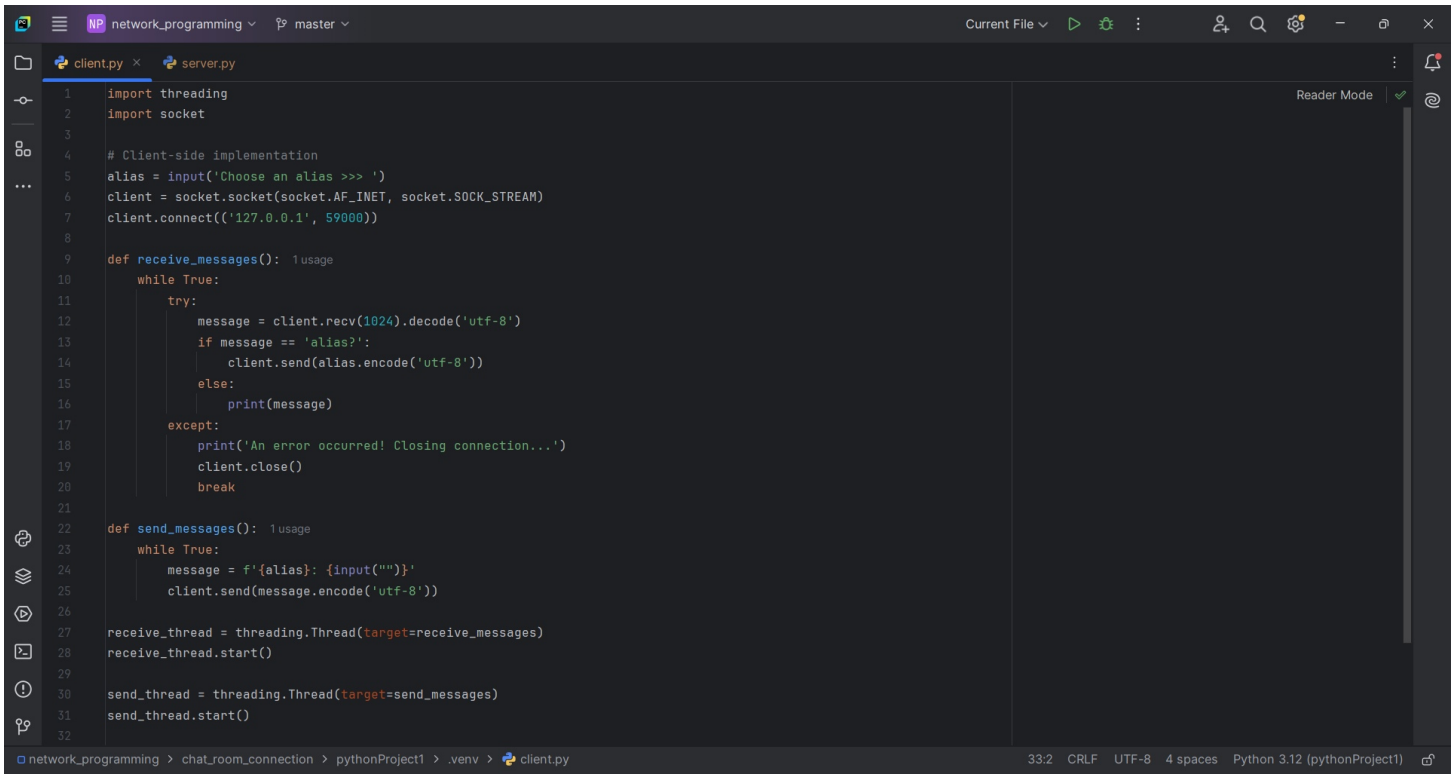
Client Side:

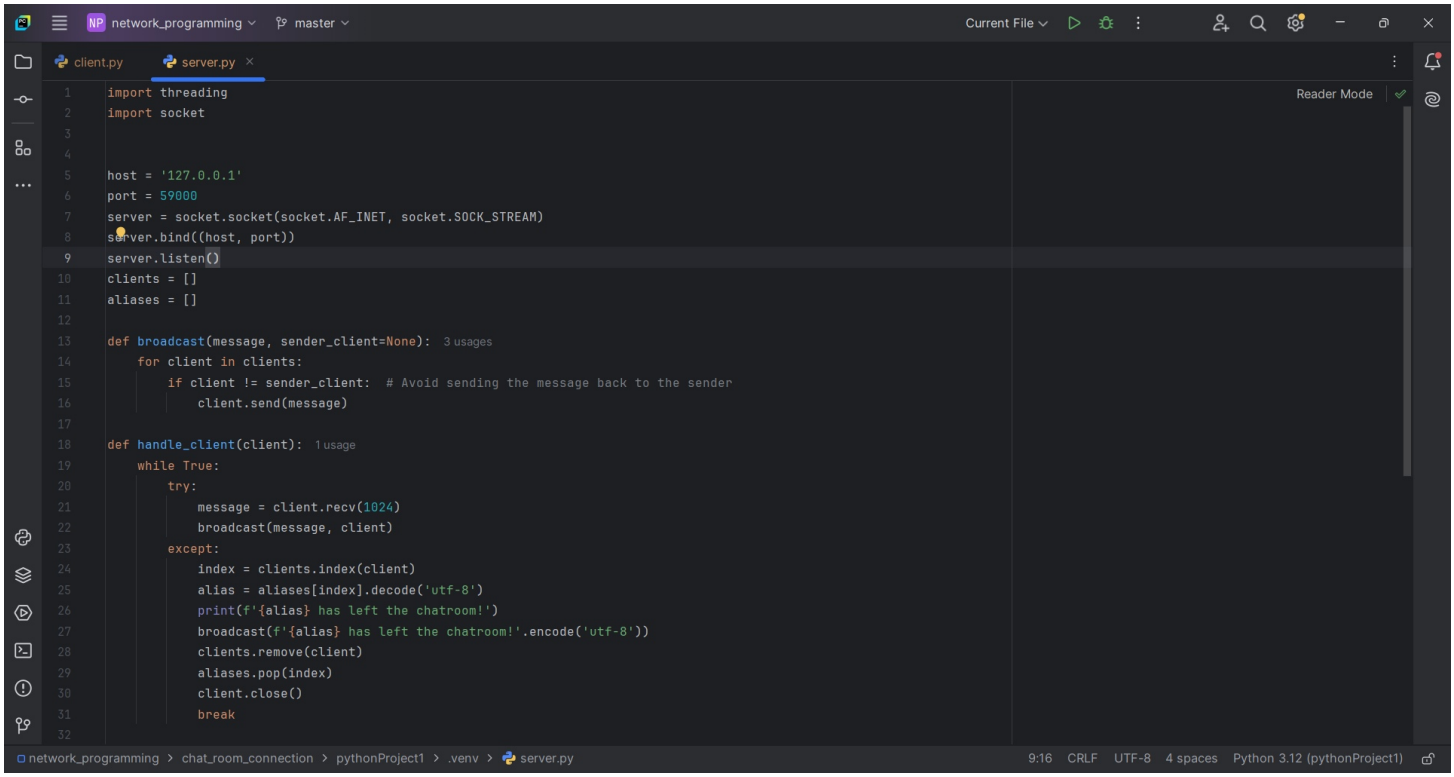
- Connects to the server and sends an alias.
- Runs two threads:
  - **Receiving Thread:** Listens for incoming messages from the server.
  - **Sending Thread:** Takes user input and sends messages to the server.
- Handles server disconnects.

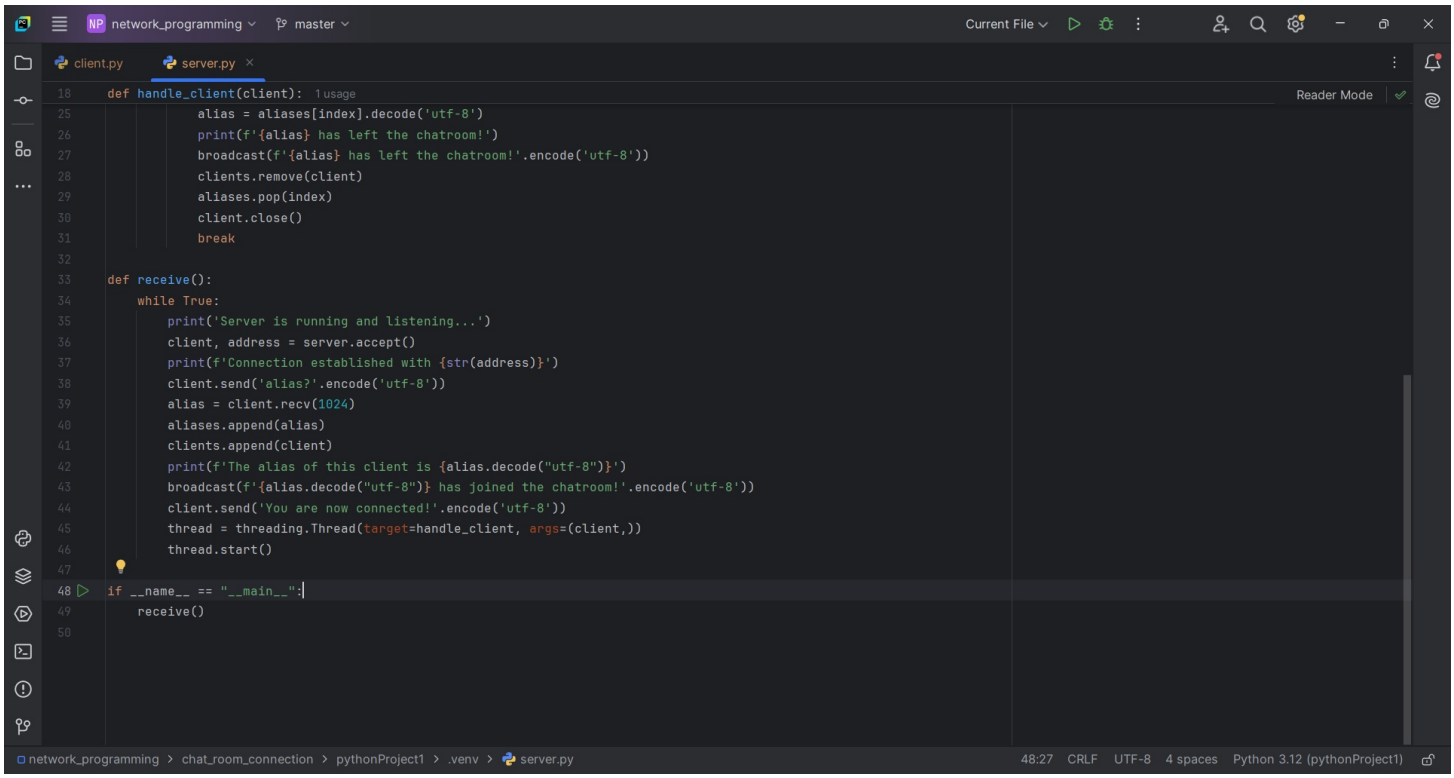
Key Features:

Multi-threading: Allows multiple clients to chat simultaneously.
Message Broadcasting: Messages are sent to all connected clients except the sender.
Error Handling: Prevents crashes when clients disconnect unexpectedly


network_programming
master
Current File
client.py
server.py
Reader Mode
import threading
import socket
host = '127.0.0.1'
port = 59000
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()
clients = []
aliases = []
def broadcast(message, sender_client=None): 3 usages
for client in clients:
if client != sender_client: # Avoid sending the message back to the sender
client.send(message)
def handle_client(client): 1 usage
while True:
try:
message = client.recv(1024)
broadcast(message, client)
except:
index = clients.index(client)
alias = aliases[index].decode('utf-8')
print(f'{alias} has left the chatroom!')
broadcast(f'{alias} has left the chatroom!'.encode('utf-8'))
clients.remove(client)
aliases.pop(index)
client.close()
break
network_programming > chat_room_connection > pythonProject1 > .venv > server.py
9:16 CRLF UTF-8 4 spaces Python 3.12 (pythonProject1)


network_programming
master
Current File
client.py
server.py
Reader Mode
import threading
import socket
# Client-side implementation
alias = input('Choose an alias >>> ')
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('127.0.0.1', 59000))
def receive_messages(): 1 usage
while True:
try:
message = client.recv(1024).decode('utf-8')
if message == 'alias?':
client.send(alias.encode('utf-8'))
else:
print(message)
except:
print('An error occurred! Closing connection...')
client.close()
break
def send_messages(): 1 usage
while True:
message = f'{alias}: {input("")}'
client.send(message.encode('utf-8'))
receive_thread = threading.Thread(target=receive_messages)
receive_thread.start()
send_thread = threading.Thread(target=send_messages)
send_thread.start()
network_programming > chat_room_connection > pythonProject1 > .venv > client.py
33:2 CRLF UTF-8 4 spaces Python 3.12 (pythonProject1)


network_programming
master
Current File
client.py
server.py
Reader Mode
import threading
import socket
host = '127.0.0.1'
port = 59000
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()
clients = []
aliases = []
def broadcast(message, sender_client=None): 3 usages
for client in clients:
if client != sender_client: # Avoid sending the message back to the sender
client.send(message)
def handle_client(client): 1 usage
while True:
try:
message = client.recv(1024)
broadcast(message, client)
except:
index = clients.index(client)
alias = aliases[index].decode('utf-8')
print(f'{alias} has left the chatroom!')
broadcast(f'{alias} has left the chatroom!'.encode('utf-8'))
clients.remove(client)
aliases.pop(index)
client.close()
break
network_programming > chat_room_connection > pythonProject1 > .venv > server.py
9:16 CRLF UTF-8 4 spaces Python 3.12 (pythonProject1)


network_programming
master
Current File
client.py
server.py
Reader Mode
def handle_client(client): 1 usage
alias = aliases[index].decode('utf-8')
print(f'{alias} has left the chatroom!')
broadcast(f'{alias} has left the chatroom!'.encode('utf-8'))
clients.remove(client)
aliases.pop(index)
client.close()
break
def receive():
while True:
print('Server is running and listening...')
client, address = server.accept()
print(f'Connection established with {str(address)}')
client.send('alias?'.encode('utf-8'))
alias = client.recv(1024)
aliases.append(alias)
clients.append(client)
print(f'The alias of this client is {alias.decode("utf-8")}')
broadcast(f'{alias.decode("utf-8")} has joined the chatroom!'.encode('utf-8'))
client.send('You are now connected!'.encode('utf-8'))
thread = threading.Thread(target=handle_client, args=(client,))
thread.start()
if __name__ == "__main__":
receive()
network_programming > chat_room_connection > pythonProject1 > .venv > server.py
48:27 CRLF UTF-8 4 spaces Python 3.12 (pythonProject1)

Critical Evaluation of Encryption methods

1. Critical Evaluation of Encryption Methods

- **Issue:** Currently, the code only uses **Fernet symmetric encryption**.
- **Improvement:** Consider implementing **asymmetric encryption (RSA)** for key exchange or **AES-GCM** for better security.

- **Justification:**
  - **Fernet (AES-CBC mode) lacks authentication** and is vulnerable to padding oracle attacks.
  - **AES-GCM provides authenticated encryption**, ensuring integrity.
  - **RSA is useful for key exchange** in secure communications.
  -

Implementation:

- Add an **AES-GCM-based encryption method**.
- Add an **RSA-based key exchange mechanism (Cryptography, 2021)**.

2. Evaluation of MFA Security

- **Issue:** The MFA implementation is basic and lacks protection against brute-force attacks (National Cyber Security Center, 2024).

- **Improvement:**
  - **Limit OTP attempts** by maintaining a retry counter (SoulFx, 2023).
  - **Use a time-based lockout mechanism** after multiple failed attempts.
  - **Hash OTP secrets before storage** instead of keeping them in plaintext (NIST, 2024).

Implementation:

- Modify verify_otp() to limit retries.
- Store OTP secrets **securely using hashing**.

3. Security Recommendations

- **Issue:** The system lacks logging, access controls, and monitoring (National Cyber Security Center, 2024).

- **Improvement:**
  - Implement **logging** for authentication attempts.
  - Add **role-based access control (RBAC) (Lindemulder & Kosinski, 2024)**.
  - Implement **secure key storage (e.g., HashiCorp Vault)** instead of storing keys in a local file (Owasp Cheat Sheet Series, 2024).

Implementation:

- Use Python's logging module for security monitoring.
- Store encryption keys securely using **environment variables or a vault**.

Summary Report

Introduction

eSecurityFin Limited, a financial services company, manages a vast amount of sensitive personal and financial data. However, a series of security incidents, including data breaches, phishing attacks, and insider threats, have exposed significant vulnerabilities within the organization's security framework. These incidents highlight weak security practices, necessitating a comprehensive evaluation of risks and existing measures.

Security Risks Analysis

Weak Credentials and Access Control

One of the most critical security gaps is the use of weak and easily compromised credentials. Brute force and credential-stuffing attacks exploit these vulnerabilities, threatening data confidentiality (America's Cyber Defense Agency, 2024). The reliance on single-factor authentication exacerbates this issue, making unauthorized access easier (Anon., 2024).

Vulnerable Software

Unpatched software poses risks such as Remote Code Execution (RCE) and SQL Injection, which cybercriminals exploit to launch malware, ransomware, and DDoS attacks (America's Cyber Defense Agency, 2024). The organization's failure to maintain updated systems significantly increases its exposure to these threats.

Social Engineering Attacks

Human error remains a significant security weakness. Phishing attacks, which manipulate employees into disclosing sensitive information, are prevalent. Attackers use personal communication channels like email and social media to deceive employees, emphasizing the need for robust awareness programs (Lenaerts-Bergmans, 2023).

Insider Threats

Current employees or former disgruntled staff pose internal threats. Whether intentional or accidental, insider incidents contribute to unauthorized access and data leaks, undermining security efforts.

Evaluation of Security Practices and Effectiveness

Despite existing security measures, recent incidents demonstrate their ineffectiveness.

- **Weak Encryption:** Sensitive data lacks strong cryptographic protection, with OWASP ranking cryptographic failures as the second-leading cause of data breaches (OWASP, 2024). Implementing stronger encryption protocols is necessary.

- **Poor Employee Training:** A lack of structured cybersecurity training exposes employees to phishing risks (Lenaerts-Bergmans, 2023). Regular training programs are crucial to enhancing awareness.
- **Ineffective Access Controls:** The current system relies on single-factor authentication, making it susceptible to password-related attacks (Anon., 2024). Multi-factor authentication (MFA) is recommended to mitigate unauthorized access.

Recommendations Based on the CIA Triad

To strengthen security, recommendations are structured around Confidentiality, Integrity, and Availability principles.

Confidentiality

- Enforce strong password policies by implementing random, long, and complex passwords (America's Cyber Defense Agency, 2024).
- Implement MFA alongside Single Sign-On (SSO) for enhanced authentication security.
- Utilize password throttling and deny lists as technical countermeasures (National Cyber Security Centre, 2018).
- Encrypt sensitive financial data using advanced encryption standards like AES to maintain confidentiality even during breaches.

Integrity

- Apply hashing algorithms like SHA-256 to verify data integrity and detect unauthorized modifications (Anyanwu et al., 2024).
- Maintain audit logs to track system access and detect anomalies.
- Restrict access based on the principle of least privilege to minimize data modification risks.

Availability

- Strengthen backup systems to mitigate potential data loss.
- Implement DDoS mitigation strategies to ensure system uptime and accessibility.

Technical Improvements

Several enhancements can improve system security.

- **Encryption:** Transitioning from Fernet (AES-CBC) encryption to AES-GCM enhances security by preventing padding oracle attacks. Introducing RSA encryption can further secure key exchanges (Cryptography, 2021).
- **Multi-Factor Authentication (MFA):** Introducing OTP retry limits and using hashed OTP secrets instead of plaintext storage will prevent brute-force attacks (NIST, 2024).
- **Access Control and Monitoring:** Implementing role-based access control (RBAC) and secure key storage solutions (e.g., HashiCorp Vault) will improve system security (Lindemulder & Kosinski, 2024).

Conclusion

eSecurityFin Limited faces multiple cybersecurity challenges due to weak credentials, vulnerable software, social engineering attacks, and insider threats. Current security measures have proven insufficient, emphasizing the need for robust improvements. Strengthening encryption, enforcing access controls, conducting employee training, and adopting advanced security technologies will significantly enhance the organization's cybersecurity posture. By implementing the recommended improvements, eSecurityFin can mitigate existing risks and safeguard its financial and personal data effectively.

References

America's Cyber Defense Agency, 2024. *America's Cyber Defense Agency.* [Online]
Available at: https://www.cisa.gov/secure-our-world/use-strong-passwords
[Accessed 01 01 2025].

America's Cyber Defense Agency, 2024. *CISA.* [Online]
Available at: https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-317a
[Accessed 25 12 2024].

Anon., 2024. *Mitre org.* [Online]
Available at: https://cwe.mitre.org/data/definitions/308.html
[Accessed 27 12 2024].

Anyanwu, A. et al., 2024. *Data Confidentiality and Integrity: A Review of Accounting and Cybersecurity Controls in Superannuation Organizations. Computer Science & IT Research Journal*, 5(1), pp. 237-253.

Baran, E., 2023. *Blaze.* [Online]
Available at: https://www.blazeinfosec.com/post/cyber-threats-for-finance-2023/
[Accessed 30 12 2024].

Cryptography, 2021. *Cryptography.* [Online]
Available at: https://cryptography.io/en/3.4/hazmat/primitives/symmetric-encryption.html
[Accessed 01 02 2025].

Kaspersky, 2024. *Kaspersky.* [Online]
Available at: https://www.kaspersky.com/resource-center/definitions/data-breach
[Accessed 25 12 2024].

Lenaerts-Bergmans, B., 2023. *Crowdstrike.* [Online]
Available at: https://www.crowdstrike.com/en-us/cybersecurity-101/social-engineering/types-of-social-engineering-attacks/
[Accessed 27 12 2024].

Lindemulder, G. & Kosinski, M., 2024. *IBM.* [Online]
Available at: https://www.ibm.com/think/topics/rbac
[Accessed 01 02 2025].

National Cyber Security Center, 2024. *NCSC UK.* [Online]
Available at: https://www.ncsc.gov.uk/blog-post/not-all-types-mfa-created-equal
[Accessed 01 02 2025].

National Cyber Security Center, 2024. *NCSC UK.* [Online]
Available at: https://www.ncsc.gov.uk/collection/device-security-guidance/managing-deployed-devices/logging-and-protective-monitoring
[Accessed 01 02 2025].

National Cyber Security Centre, 2018. *NCSC UK.* [Online]
Available at: https://www.ncsc.gov.uk/collection/passwords/updating-your-approach
[Accessed 01 01 2025].

NIST, 2024. *NIST.* [Online]
Available at: https://pages.nist.gov/800-63-3-Implementation-Resources/63B/Authenticators/
[Accessed 01 02 2025].

OWASP Cheat Sheet Series, 2024. *OWASP.* [Online]
Available at: https://cheatsheetseries.owasp.org/cheatsheets/Key_Management_Cheat_Sheet.html
[Accessed 01 02 2025].

OWASP, 2024. *OWASP.* [Online]
Available at: https://owasp.org/www-project-top-ten/
[Accessed 27 12 2024].

SoulFx, 2023. *SoulFx.* [Online]
Available at: https://soulfx.com/2023/09/best-practices-for-one-time-password-otp-implementation/
[Accessed 01 02 2025].

Appendix

170523_fb_blog_hacker-1961797569.png

PythonToolKit_Banner-133163116.png

advanced-cyber-security-header-02.jpg

th-4097253126.jpg

Appendix item: 170523_fb_blog_hacker-1961797569.png

Show location

170523_fb_blog_hacker-1961797569.png

png (1022 KB)

PythonToolKit_Banner-133163116.png
png (136 KB)

Appendix item: advanced-cyber-security-header-02.jpg

Show location

advanced-cyber-security-header-02.jpg

jpg (92 KB)

Appendix item: th-4097253126.jpg

Show location

th-4097253126.jpg
jpg (35 KB)