VU VISHWAKARMA UNIVERSITY

Maximising Human Potential

Activity based

Project 2 Report on

Application Security

Submitted to Vishwakarma University, Pune

Name – Atharva shevate

SRN No : 202201727

Roll No : 02

Div : E

Second Year Engineering

Department of Computer Engineering

Faculty of Science and Technology

Academic Year

2023-2024

Application Security

Project Statement:

Demonstrate a Masquerading attack to try and expose the lack security of a communication channel of Pizza House which doesn't use Public/Private Key encryption. Implement effective secutiry measues against such an attack

1. Introduction:

Pizza House relies on a communication channel between its servers and clients for various operations, including order placement and processing. However, without proper encryption, this channel is vulnerable to attacks, such as masquerading, where unauthorized entities can intercept and manipulate data.

2. Objective:

The objective of this project is to demonstrate a masquerading attack on Pizza House's communication channel, highlight its security implications, and implement effective security measures to prevent such attacks.

3. Implementation:

3.1. Vulnerable Communication Setup:

- Develop a simple server-client communication system using Python socket programming.
- Implement a basic messaging system without any encryption.
- Run the server and client scripts on separate machines within the same local network.

3.2. Masquerading Attack Demonstration:

- Utilize Wiershark or similar tools to scan the local network for active devices.
- Identify the IP address and port associated with the Pizza House server.
- Use a custom Python script to intercept and manipulate messages between the client and server.
- Demonstrate how an attacker can masquerade as a legitimate client or server to intercept, modify, or inject malicious data into the communication channel.
- Showcase the potential risks and consequences of such an attack, including unauthorized access to sensitive information or manipulation of orders.

3.3. Implementation of Encryption:

- Integrate encryption algorithms (e.g., AES, RSA) into the communication channel to secure data transmission.

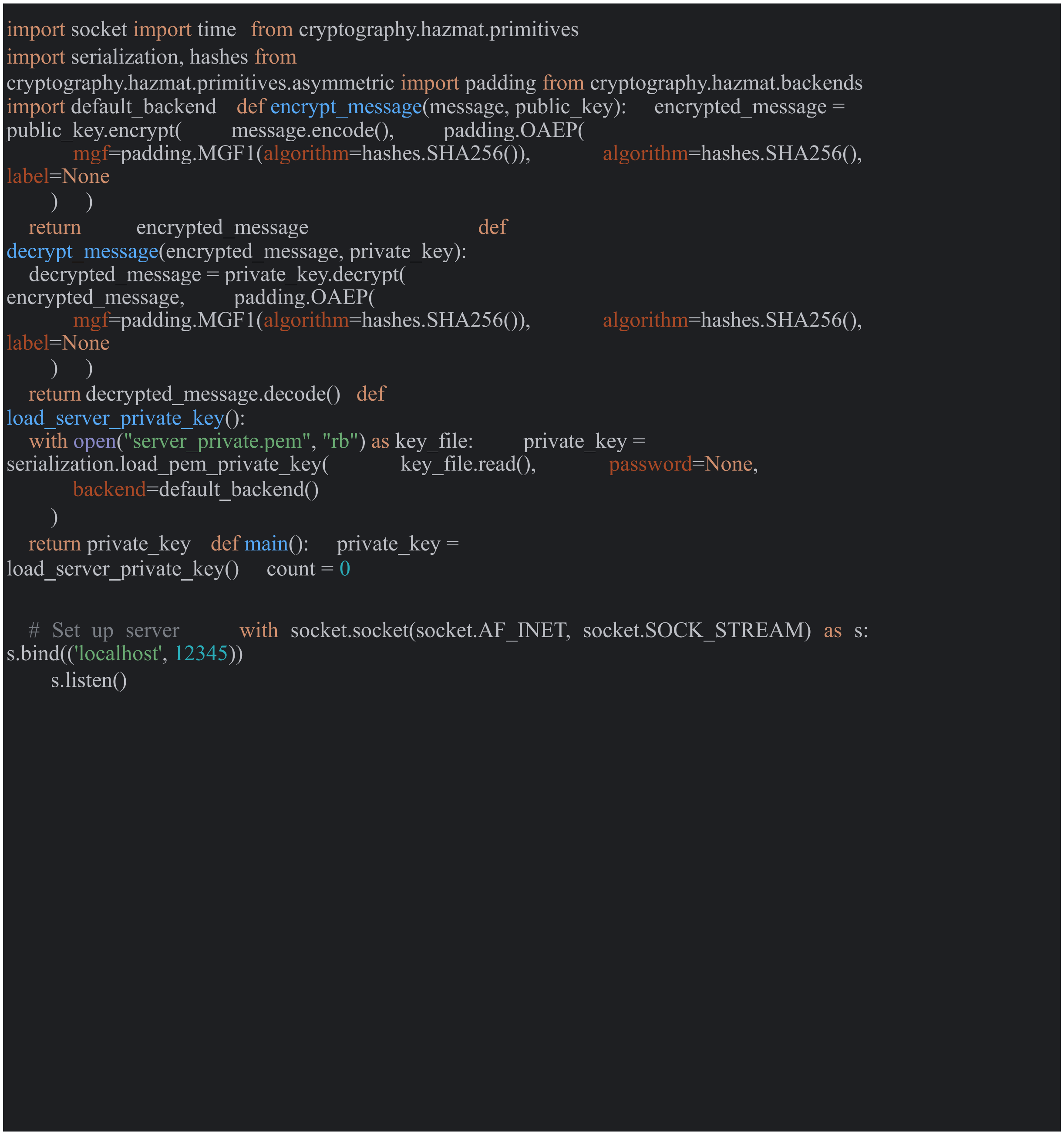
    - Modify the server and client scripts to encrypt outgoing messages and decrypt incoming messages using the chosen encryption algorithm.
    - Ensure that keys are securely exchanged between the server and client to establish a secure communication channel.

4. Security Measures:

- Implement authentication mechanisms, such as username/password or token-based authentication, to verify the identity of clients and servers.
- Utilize SSL/TLS protocols for secure socket communication to prevent eavesdropping and man-in-the-middle attacks.
- Regularly update encryption algorithms and keys to mitigate the risk of cryptographic attacks.
- Monitor network traffic using intrusion detection systems (IDS) or network monitoring tools to detect suspicious activities.
- Educate Pizza House staff on cybersecurity best practices, including the importance of encryption and the risks associated with unsecured communication channels.

Code -

New server.py



```
import socket import time   from cryptography.hazmat.primitives
import serialization, hashes from
cryptography.hazmat.primitives.asymmetric import padding from cryptography.hazmat.backends
import default_backend    def encrypt_message(message, public_key):     encrypted_message =
public_key.encrypt(         message.encode(),         padding.OAEP(
           mgf=padding.MGF1(algorithm=hashes.SHA256()),             algorithm=hashes.SHA256(),
label=None
       )     )
    return          encrypted_message                              def
decrypt_message(encrypted_message, private_key):
    decrypted_message = private_key.decrypt(
encrypted_message,         padding.OAEP(
           mgf=padding.MGF1(algorithm=hashes.SHA256()),             algorithm=hashes.SHA256(),
label=None
       )     )
    return decrypted_message.decode()   def
load_server_private_key():
    with open("server_private.pem", "rb") as key_file:         private_key =
serialization.load_pem_private_key(             key_file.read(),             password=None,
           backend=default_backend()
       )
    return private_key    def main():     private_key =
load_server_private_key()     count = 0


    # Set up server           with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
s.bind(('localhost', 12345))
       s.listen()
    print("Server listening...")        while True:
try:                conn, addr = s.accept()                with
conn:
            print('Connected by', addr)
if count == 0:

                # Receive client's public key                        client_public_key_pem = conn.recv(4096)
                client_public_key = serialization.load_pem_public_key(                        client_public_key_pem,
backend=default_backend()                        )

                print("Client's Public Key:")                        print(client_public_key.public_bytes(
encoding=serialization.Encoding.PEM,


format=serialization.PublicFormat.SubjectPublicKeyInfo                        ).decode())

                # Send server's public key to client                        with open("server_public.pem", "rb") as key_file:
                   server_public_key =
serialization.load_pem_public_key(                                key_file.read(),
backend=default_backend()
                   )                        conn.sendall(server_public_key.public_bytes(
                            encoding=serialization.Encoding.PEM,


format=serialization.PublicFormat.SubjectPublicKeyInfo
                ))                        count = 1
while True:
                # Receive length of data first                        encrypted_client_message_length =
int.from  bytes(conn.recv(4), 'big')
                encrypted_client_message =
conn.recv(encrypted_client_message_length)                        if not
encrypted_client_message:                        break
                 decrypted_client_message =
decrypt_message(encrypted_client_message, private_key)


                print("Client message: ", decrypted_client_message)
if decrypted_client_message.lower() == "exit":
break


                time.sleep(2)
                server_message = input("Server message: ")                        encrypted_server_message =
encrypt_message(server_message, client_public_key)
                print("-----------:Encrypted message:--------------")                        print(encrypted_server_message)
print("------------------:end:--------------------")
```

```
                    conn.sendall(len(encrypted_server_message).to_bytes(4,
'big'))
                    time.sleep(2)
                    conn.sendall(encrypted_server_message)                    if decrypted_client_message.lower()
== "exit":                    break            except Exception as e:
             print(f"An error occurred: {e}")                continue    if __name__ == "__main__":
   main()
```

new client.py

```
import socket   from cryptography.hazmat.primitives import serialization from
cryptography.hazmat.primitives.asymmetric import padding from
cryptography.hazmat.primitives import hashes from cryptography.hazmat.backends
import default_backend     def encrypt_message(message, public_key):
encrypted_message = public_key.encrypt(         message.encode(),
padding.OAEP(
          mgf=padding.MGF1(algorithm=hashes.SHA256()),
algorithm=hashes.SHA256(),            label=None
       )    )
   return       encrypted_message                            def
decrypt_message(encrypted_message, private_key):
   decrypted_message = private_key.decrypt(
encrypted_message,        padding.OAEP(
          mgf=padding.MGF1(algorithm=hashes.SHA256()),
algorithm=hashes.SHA256(),            label=None
       )    )
   return       decrypted_message.decode()
def load_client_private_key():
   with open("client_private.pem", "rb") as key_file:        private_key =
serialization.load_pem_private_key(           key_file.read(),
password=None,
          backend=default_backend()
       )
   return private_key
```

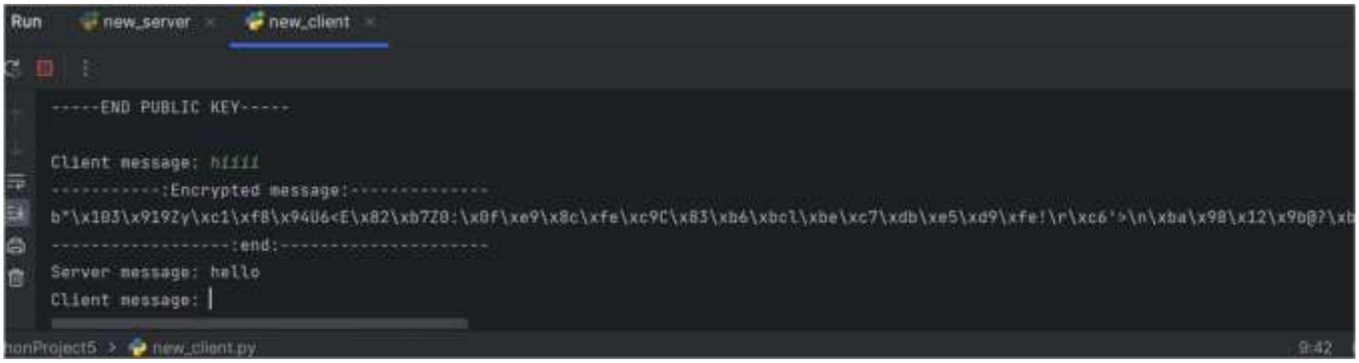
```
 def main():
   private_key = load_client_private_key()
   # Establish connection with server      with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as s:
       s.connect(('localhost', 12345))
       # Send client's public key to server          with open("client_public.pem", "rb") as
key_file:              client_public_key = serialization.load_pem_public_key(
key_file.read(),                backend=default_backend()
           )
       s.sendall(client_public_key.public_bytes(                                   encoding=serialization.Encoding.PEM,
format=serialization.PublicFormat.SubjectPublicKeyInfo         ))


       # Receive server's public key         server_public_key_pem = s.recv(4096)
server_public_key = serialization.load_pem_public_key(            server_public_key_pem,
backend=default_backend()
       )
       print("Server's Public Key:")         print(server_public_key.public_bytes(
encoding=serialization.Encoding.PEM,
           format=serialization.PublicFormat.SubjectPublicKeyInfo
).decode())            while True:              client_message = input("Client message:
")


           encrypted_client_message = encrypt_message(client_message, server_public_key)


           print("-----------:Encrypted message:--------------")           print(encrypted_client_message)             print("--
----------------:end:---------------------")


           s.sendall(len(encrypted_client_message).to_bytes(4, 'big'))
           s.sendall(encrypted_client_message)             if client_message.lower()
== "exit":
               break


               encrypted_server_message_length = int.from_bytes(s.recv(4), 'big')           encrypted_server_message =
                                                  s.recv(encrypted_server_message_length)
            if not encrypted_server_message:
break


           decrypted_server_message = decrypt_message(encrypted_server_message,
private_key)             print("Server message:", decrypted_server_message)    if __name__ ==
"__main__":
   main()
```

Fake_client.py

```
import   socket      from   cryptography.hazmat.primitives   import
serialization    from    cryptography.hazmat.primitives.asymmetric
import padding from
cryptography.hazmat.primitives import hashes from
cryptography.hazmat.backends import default_backend    def
encrypt_message(message, public_key):     encrypted_message =
public_key.encrypt(         message.encode(),         padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),             algorithm=hashes.SHA256(),
label=None
        )     )
    return encrypted_message    def
load_server_public_key():     try:
        with open("hacked_key.pem", "rb") as key_file:  # Load the correct server's public key
public_key = serialization.load_pem_public_key(                 key_file.read(),
backend=default_backend()
            )
        return public_key     except Exception as e:
        print(f"Error loading public key: {e}")         return None
def main():      public_key = load_server_public_key()     if
not public_key:          return  # Exit if public key loading
failed


    # Establish connection with server     with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as s:
        s.connect(('localhost', 12345))           while True:                 client_message =
input("Client   message:   ")                                   encrypted_client_message   =
encrypt_message(client_message, public_key)


            print("-----------:Encrypted message:--------------")
print(encrypted_client_message)              print("------------------:end:--------------------")
# Send length of data before sending actual data
            s.sendall(len(encrypted_client_message).to_bytes(4, 'big'))
            s.sendall(encrypted_client_message)              if client_message.lower()
== "exit":
                break


            # Receive and print response from server
encrypted_server_message_length = int.from_bytes(s.recv(4), 'big')
```

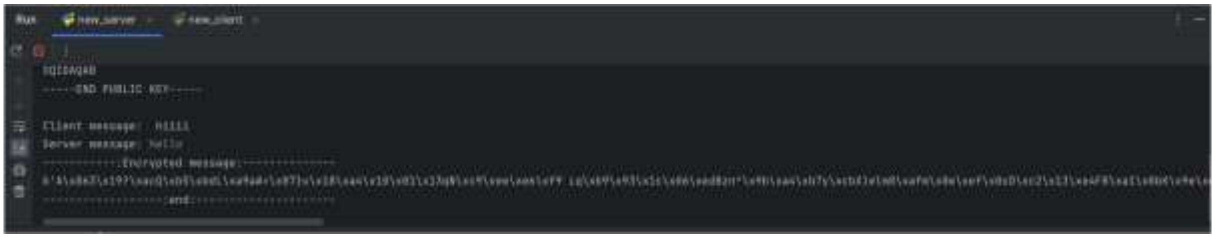
```
        encrypted_server_message =
s.recv(encrypted_server_message_length)            if not
encrypted_server_message:                    break
        print("Encrypted Server message:", encrypted_server_message)
if __name__ == "__main__":
  main()
```

output:- client server



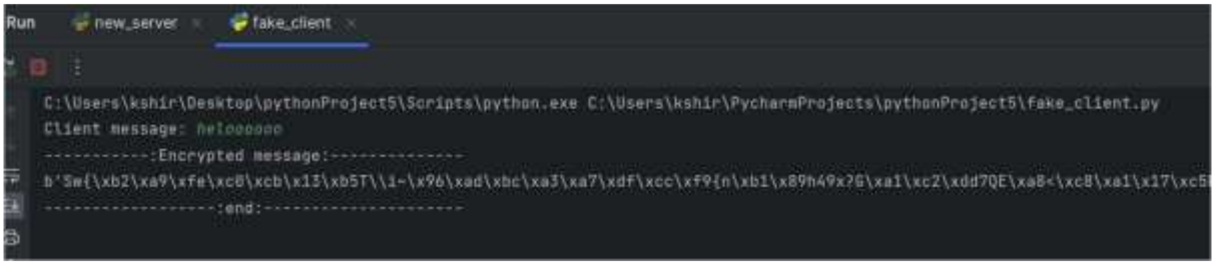
```
-----END PUBLIC KEY-----

Client message: hiiii
-----------:Encrypted message:--------------
b"\x103\x919Zy\xc1\xf8\x94U6<E\x82\xb7Z0:\x0f\xe9\x8c\xfe\xc9C\x83\xb6\xbcl\xbe\xc7\xdb\xe5\xd9\xfe!\r\xc6'>\n\xba\x98\x12\x9b@?\xb
------------------:end:---------------------
Server message: hello
Client message: 
```

Server



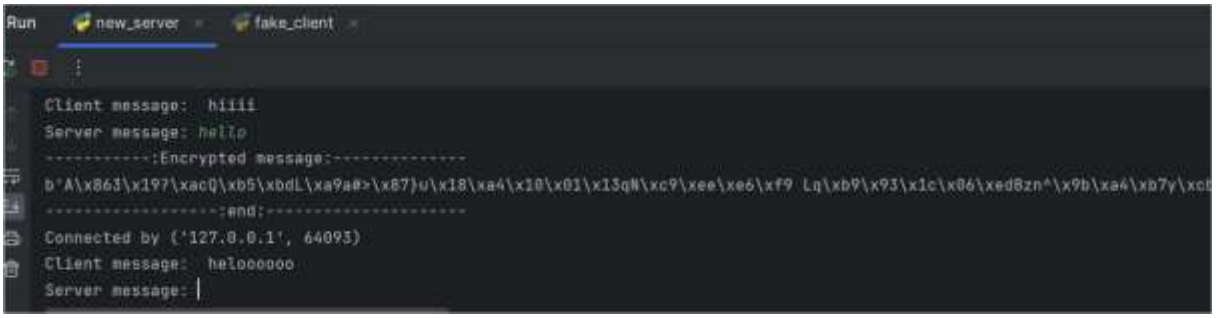
```
IQIDAQAB
-----END PUBLIC KEY-----

Client message:  Hiiii
Server message: hello
-----------:Encrypted message:--------------
b'[illegible]
------------------:end:---------------------
```

Fake client



```
C:\Users\kshir\Desktop\pythonProject5\Scripts\python.exe C:\Users\kshir\PycharmProjects\pythonProject5\fake_client.py
Client message: helooooo
-----------:Encrypted message:--------------
b'Sw{\xb2\xa9\xfe\xc8\xcb\x13\xb5T\\i-\x96\xad\xbc\xa3\xa7\xdf\xcc\xf9{n\xb1\x89h49x?G\xa1\xc2\xdd7QE\xa8<\xc8\xa1\x17\xc5
------------------:end:---------------------
```

Server



```
Client message:  hiiii
Server message: hello
------------:Encrypted message:---------------
b'A\x863\x197\xacQ\xb5\xbdL\xa9a#>\x87}u\x18\xa4\x10\x01\x13qN\xc9\xee\xe6\xf9 Lq\xb9\x93\x1c\x06\xed8zn^\x9b\xa4\xb7y\xc
------------------:end:--------------------
Connected by ('127.0.0.1', 64093)
Client message:  helooooo
Server message: 
```

5. Conclusion:

By demonstrating a masquerading attack and implementing effective security measures, this project aims to raise awareness about the vulnerabilities of unsecured communication channels and emphasize the importance of encryption and other security practices in safeguarding sensitive data for businesses like Pizza House.