**Mini Project**

**Title :**

Man In Middle Attack.

**Objective :**

* Man In Middle project written in python using sockets, threading and object-oriented programming.
* Illustrate the concept of convertly intercepting and modifying messages between two parties in aclient-server architecture.

**Problem Statement** :

To demonstrate how Man in Middle works using python through sockets and object oriented programming.

**Software and Hardware requirement** :

Spyder/Pycharm and Windows/Linux python installed on it.

**Theory** :

**Definition of Man In Middle** :

In cryptography and computer security, a man-in-the-middle attack (MITM) is an attack where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other.

**Example :**

Suppose [Alice](https://en.wikipedia.org/wiki/Alice_and_Bob" \o "Alice and Bob) wishes to communicate with [Bob](https://en.wikipedia.org/wiki/Alice_and_Bob" \o "Alice and Bob). Meanwhile, [Mallory](https://en.wikipedia.org/wiki/Alice_and_Bob" \l "Mallory" \o "Alice and Bob) wishes to intercept the conversation to eavesdrop and optionally to deliver a false message to Bob.

First, Alice asks Bob for his [public key](https://en.wikipedia.org/wiki/Public-key_cryptography" \o "Public-key cryptography). If Bob sends his public key to Alice, but Mallory is able to intercept it, an MITM attack can begin. Mallory sends Alice a forged message that appears to originate from Bob, but instead includes Mallory's public key.

Alice, believing this public key to be Bob's, encrypts her message with Mallory's key and sends the enciphered message back to Bob. Mallory again intercepts, deciphers the message using her private key, possibly alters it if she wants, and re-enciphers it using the public key she intercepted from Bob when he originally tried to send it to Alice. When Bob receives the newly enciphered message, he believes it came from Alice.

1. Alice sends a message to Bob, which is intercepted by Mallory:

Alice *"Hi Bob, it's Alice. Give me your key."* →     Mallory     Bob

1. Mallory relays this message to Bob; Bob cannot tell it is not really from Alice:

Alice     Mallory *"Hi Bob, it's Alice. Give me your key."* →     Bob

1. Bob responds with his encryption key:

Alice     Mallory     ← *[Bob's key]* Bob

1. Mallory replaces Bob's key with her own, and relays this to Alice, claiming that it is Bob's key:

Alice     ← *[Mallory's key]* Mallory     Bob

1. Alice encrypts a message with what she believes to be Bob's key, thinking that only Bob can read it:

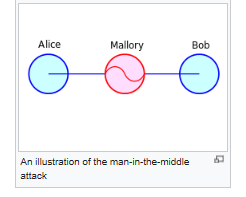
Alice *"Meet me at the bus stop!" [encrypted with Mallory's key]* →     Mallory     Bob

1. However, because it was actually encrypted with Mallory's key, Mallory can decrypt it, read it, modify it (if desired), re-encrypt with Bob's key, and forward it to Bob:

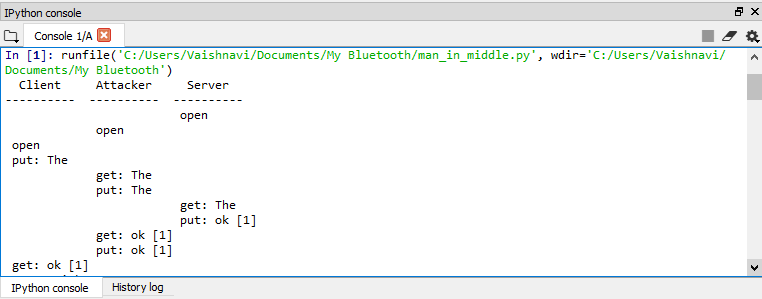
Alice     Mallory *"Meet me at the van down by the river!" [encrypted with Bob's key]* →     Bob

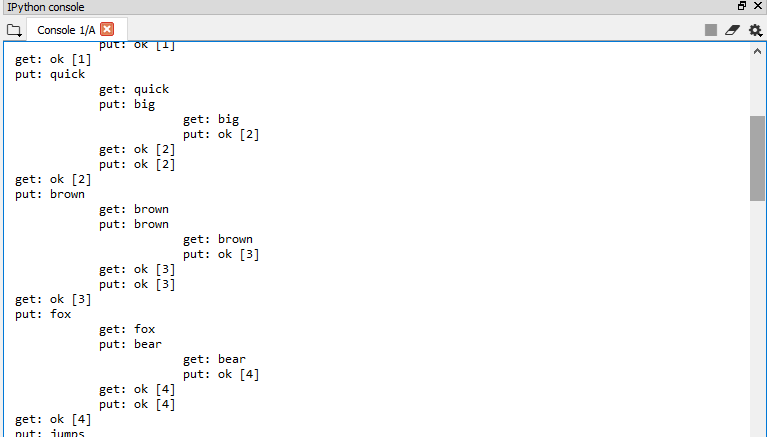
1. Bob thinks that this message is a secure communication from Alice.

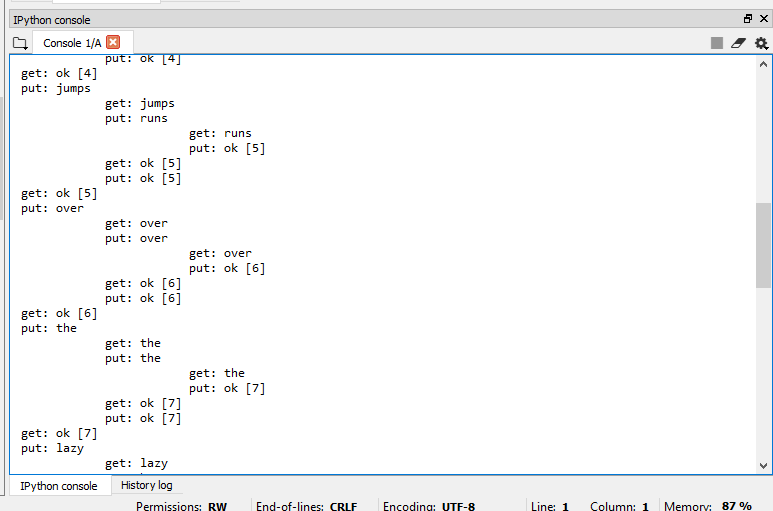
This example[[4]](https://en.wikipedia.org/wiki/Man-in-the-middle_attack" \l "cite_note-4) shows the need for Alice and Bob to have some way to ensure that they are truly each using each other's [public keys](https://en.wikipedia.org/wiki/Public-key_cryptography" \o "Public-key cryptography), rather than the public key of an attacker. Otherwise, such attacks are generally possible, in principle, against any message sent using public-key technology. A variety of techniques can help defend against MITM attacks.

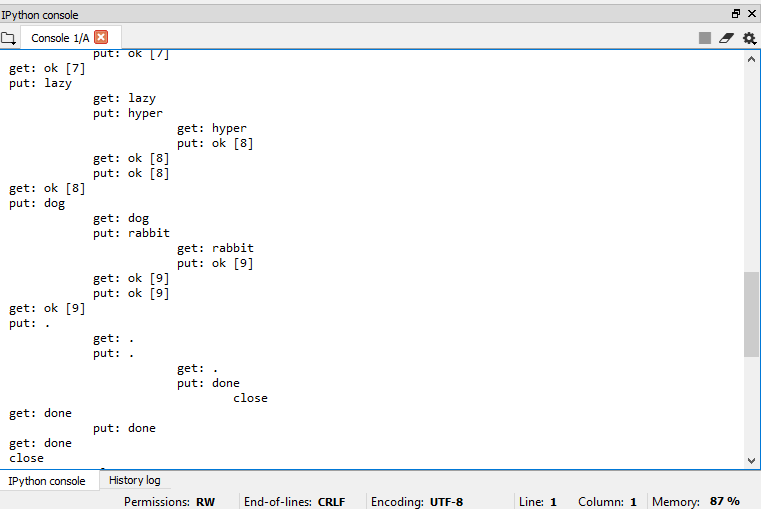
****

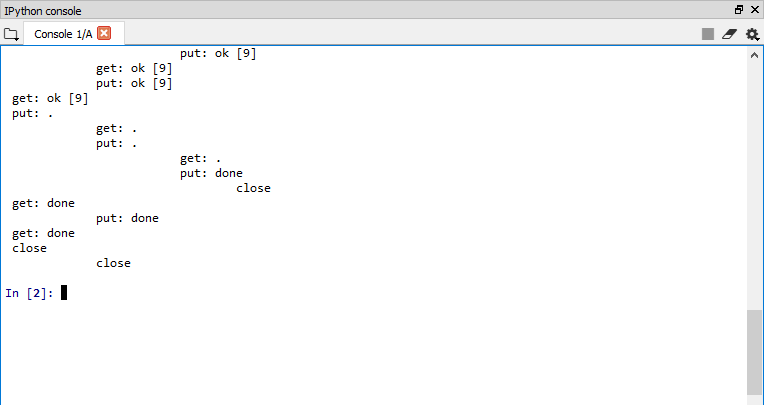
**Program codes with output screenshots** :











**Codes :**

# -\*- coding: utf-8 -\*-

"""

Created on Fri May 1 15:19:15 2020

@author: Sumit

"""

import socket, threading

ENCODING = "utf8"

class Client(threading.Thread):

def run(self):

# initialize necessary variables

spacing = ""

clientHost = "127.0.0.1"

clientPort = 12345

clientAddress = (clientHost, clientPort)

# open connection

connection = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

print(spacing, "open")

requestData = ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog", "."]

for datum in requestData:

# send request to server

print(spacing, "put:", datum)

connection.sendto(datum.encode(ENCODING), clientAddress)

# get response from server

responseData, clientAddress = connection.recvfrom(1024)

print(spacing, "get:", responseData.decode(ENCODING))

# close connection

connection.close()

print(spacing, "close")

class Server(threading.Thread):

def run(self):

# initialize necessary variables

count = 0

spacing = "\t\t\t"

serverHost = "127.0.0.1"

serverPort = 12346

serverAddress = (serverHost, serverPort)

# open connection

connection = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

connection.bind(serverAddress)

print(spacing, "open")

while 1:

# get request from client

requestData, clientAddress = connection.recvfrom(1024)

print(spacing, "get:", requestData.decode(ENCODING))

# process request

count += 1

if requestData.decode(ENCODING) == ".":

responseData = "done"

else:

responseData = "ok [" + str(count) + "]"

# send response to client

print(spacing, "put:", responseData)

connection.sendto(responseData.encode(ENCODING), clientAddress)

# close connection

if requestData.decode(ENCODING) == ".":

break

connection.close()

print(spacing, "close")

class Attacker(threading.Thread):

def run(self):

# initialize necessary variables

spacing = " "

clientHost = "127.0.0.1"

clientPort = 12345

clientAddress = (clientHost, clientPort)

# open connection

clientConnection = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

clientConnection.bind(clientAddress)

# initialize necessary variables

serverHost = "127.0.0.1"

serverPort = 12346

serverAddress = (serverHost, serverPort)

# open connection

serverConnection = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

print(spacing, "open")

while 1:

# get request from client

clientRequestData, clientAddress = clientConnection.recvfrom(1024)

print(spacing, "get:", clientRequestData.decode(ENCODING))

# modify request (if desired)

serverRequestData = clientRequestData.decode(ENCODING)

if clientRequestData.decode(ENCODING) == "quick":

serverRequestData = "big"

if clientRequestData.decode(ENCODING) == "fox":

serverRequestData = "bear"

if clientRequestData.decode(ENCODING) == "jumps":

serverRequestData = "runs"

if clientRequestData.decode(ENCODING) == "lazy":

serverRequestData = "hyper"

if clientRequestData.decode(ENCODING) == "dog":

serverRequestData = "rabbit"

# send request to server

print(spacing, "put:", serverRequestData)

serverConnection.sendto(serverRequestData.encode(ENCODING), serverAddress)

# get response from server

serverResponseData, serverAddress = serverConnection.recvfrom(1024)

print(spacing, "get:", serverResponseData.decode(ENCODING))

# modify response (if desired)

clientResponseData = serverResponseData.decode(ENCODING)

# send response to client

print(spacing, "put:", clientResponseData)

clientConnection.sendto(clientResponseData.encode(ENCODING), clientAddress)

if serverResponseData.decode(ENCODING) == "done":

break

# close connection

serverConnection.close()

clientConnection.close()

print(spacing, "close")

print(" Client Attacker Server")

print("---------- ---------- ----------")

# create all threads

server = Server()

attacker = Attacker()

client = Client()

# start all threads

server.start()

attacker.start()

client.start()

# join all threads

server.join()

attacker.join()

client.join()

**Conclusion/Analysis** : .

In this way we demonstrated how Man In Middle attac works using python through socket and object oriented programming concept.

**Assessor’s sign :**