Using Scapy for Man-in-the-Middle Attacks

Ana Rita Martinho
Department of Electrical and Computer Engineering
Faculty of Engineering of University of Porto
up201709727@fe.up.pt

Gonçalo Xavier
Department of Electrical and Computer Engineering
Faculty of Engineering of University of Porto
up201604506@fe.up.pt

***Abstract*—A Man-in-the-Middle(MITM) attack is a type of cyberattack where an attacker secretly relays and possibility alters the communication between 2 unsuspecting parties. Most recent cryptographic protocols include some form of endpoint authentication specifically to prevent MITM attacks.**

The goal of this project is to demonstrate, using scapy, why such measurements are needed. Scapy is a python module that enables a user to send, sniff and forge network packets, as such it provides easy methods to the types of functionalities needed to run a MITM attack.

4 different protocols were analyzed (FTP, SNMP, Telnet and SMTP).

I. Introduction

This paper will focus on the work developed for the main project of the Security for Systems and Networks course.

This projects aims to demonstrate the different procedures used to carry out a MITM attack and their potential outcomes.

3 Linux Virtual Machines (VM) were used, 2 simulating normal users of the analyzed protocols - victims - and 1 running scapy, simulating the potential attacker.

The general flow of a MITM attack is as follows: **Sniffing/Evesdropping** - the attacker detects some kind of vulnerable communication or victims - **Positioning** - the attacker position itself between both victims - **Exploitation** - the attacker stores or even alters the collected information for nefarious actions.

A. Analyzed Protocols

1) FTP: FTP stands for File Tranfer Protocol and is a standard network protocol used in computer file transfer between 2 nodes in a network, using a hierarchical architecture in a client-server model fashion (the server being the node storing the files and the client the node requesting it). Clients can authenticate themselves in an *anonymous* way - if the server is prepared for that - or using a username and password that is sent in **clear text**. With that in mind and knowing that FTP server uses well-known reply codes, (such as *230 - User logged in, proceed.*) an attacker can easily store and use valid credentials to access the server, impersonating a valid client.

2) SNMP: SNMP stands for Simple Network Management Protocol and it is used for collecting, organizing and changing information about managed devices on Layer 3 networks. SNMP defines structured management data using a Management Information Base (MIB). This database is hierarchical (tree-structured) and each entry is addressed through an object identifier (OID). SNMP follows a client-server model in which the servers (*managers*) collect and process information about devices on a network. The clients (*agents*) are any type of device in a network that can be monitored, such as computers, switches, routers, printers, etc. There are 3 main SNMP commands: - *set*: the manager sets some value to an OID value available on the agent; *get*: the manager queries the agent about some OID value and *trap*: the agent sends unsolicited information about some OID value to the manager. These commands require the use of a *community string*, representing somewhat off a password. In version 1 and 2 of the protocol this community string is sent in **clear text** and so, any attacker, sniffing a SNMP communication can *set* or *get* any OID value using the retrieved community string. SNMPv3 corrects this lack of security.

3) Telnet: Telnet is a server-client application protocol that is typically used to open a command line on a remote computer. It uses a bidirectional interactive text-oriented communication facility. When a user tries to access the remote computer, they are prompted to enter their username and password combination. Telnet, by default, **doesn't encrypt** any type of data, neither the pass and user combination nor the data sent over the connection, presenting a huge lack of security, especially when this protocol is deployed over the Internet - an attacker can eavesdrop on the communications and use the gathered information for malicious purposes. In fact, Telnet is becoming deprecated in favor of the SSH protocol.

4) SMTP: SMTP stands for Simple Mail Transfer Protocol and it is used for sending and receiving e-mail. SMTP also follows a client-server model, with the server being the application that its primary purpose is to send, receive, and/or relay outgoing mail between email senders and receivers. In other works, a client email sender, writes the emails body, subject and specifies the receiver. This information is sent to the SMTP server which is responsible for relaying this email to the appropriate receiver, if possible. This information is sent using **clear text** and any attacker in the middle of the client-server communication can access the contents of the email, compromising a mostly desired privacy. Similarly to FTP, SMTP server also uses reply codes, which increases the ease of the attacker to know which type of data is being sent.

II. Used tools - Related concepts

1) Scapy:

Scapy is a powerful python interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It can replace hping, arpspoof, arp-sk, arping, p0f and even some parts of Nmap, tcpdump, and tshark.

Scapy also performs a lot of other specific tasks that most other tools can't handle, like sending invalid frames, injecting your own 802.11 frames, combining techniques (VLAN hopping+ARP cache poisoning, VOIP decoding on WEP encrypted channel, . . .), etc. As such, it allows its users to easily craft the exact packets they would like, send them out into the network and analyse their responses.

In the context of this project, the Scapy module was used as the primary tool of the developed code, specially its custom packet building functionalities.

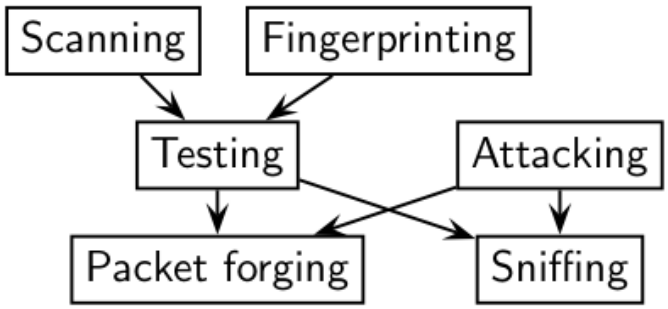

Fig. 1. Scapy's taxonomy as depicted in [2]

2) ARP spoofing:

As mentioned in the section I, the general MITM attack goes through 3 main steps: Evesdropping, Positioning and Exploiting. *ARP spoofing* played a major role in this project's first and middle step.

The ARP protocol is normally used to resolve the specific MAC address associated with a particular IP, so that packets can reach the intended device - translating Layer-3 addresses into Layer-2. On normal circumstances, the sending device sends an ARP Request to discover the MAC address of a particular IP. This request is then broadcasted to the network and the device with the specified IP sends an ARP Response announcing its MAC address.

This process can be bypassed by sending an unsolicited ARP Response to a specific user, announcing that a particular IP actually belongs to the attacker's MAC address. In the most common form of *ARP spoofing*, the attacker forges an ARP Response packet declaring that its MAC address is the one associated with the active Access Point (AP) IP. The attacker then informs the actual AP (in a similar way) that its MAC address is the one associated with the victims IP. This way the attacker inserts itself in the middle of all communications between the victim and the network, and is able to analyse all the victim's traffic.

3) TCP hijack:

An alternative method to accomplish the **Positioning** step of section I is *TCP hijacking*.

During TCP's three-hand handshake, the client initially sends a random value to be used as its sequence number (Seq.A), and the server replies with an acknowledgment number (ACK) of Seq.A + 1 and a random value value of its own, sequence number (Seq.B). This values are then used to re-arrange out of order packets and ensure that the correct packet is received during communication.

An attacker can impersonate one of the involved parties by forging an IP packet with the correct ongoing Seq. and ACK values. If that happens, the victim's own Seq. and ACK numbers became offsetted, and their packets are dropped when reaching the other side of the conversation. As such, the attacker is able to "hijack" the ongoing connection.

This approach faces some problems:

- It requires a ongoing connection - upon receiving a forged SYN packet, the server will reply with a SYN ACK to the IP of said packet. Since the real device of this IP did not start a connection, it will respond with a TCP packet whose RST bit is set, immediately closing the connection.
- The server, and client became aware of the hijack - it's possible that the large quantity of sudden dropped packets trigger some kind of security alert and this may cause the connection to be dropped.

Both these problems can be somewhat dealt with by conditioning not only when this attack may occur, but also by closing the real client's connection before the hijack, without the server realizing it.

4) RST attack:

Each TCP packet has a header, and each header contains a bit known as the "reset" (RST) flag. In usual TCP traffic, this bit is set to 0 and has no effect. However, if this bit is set to 1, it indicates to the receiving device that it should immediately stop using the TCP connection; it should not send any more packets using the connection's identifying numbers, called ports, and discard any further packets it receives with headers indicating they belong to this connection. A TCP reset basically kills a TCP connection instantly.

As seen previously, it's possible to inject a forged TCP packet into a ongoing TCP connection, if the forged Seq. and ACK numbers are correct. As such, an attacker can prematurely terminate a TCP connection, without the willing consent of the parties involved.

5) Reverse Shell:

In a typical remote access scenario, the user is the client and the target machine is the server. The user initiates a remote shell connection and the target system listens for such

connections. With a reverse shell, the roles are the opposite. The target machine initiates the connection to the user, and the user's device listens for incoming connections on a specified port. This can be used to bypass firewall restrictions on open ports, as most restrictions are imposed on incoming traffic, not on outgoing one. Reverse shells can usually be used after some initial code or packet is injected in the target machine and allow an attacker to obtain a interactive shell session on the victim's machine.

III. Solution

The final product of this projects accomplishes the 1st step mentioned on section I through a *ping broadcast* command, identifying all available hosts present in the network. It then positions itself using *ARP spoofing*, communicating to both users that the IP to which they intend to communicate with belongs to its MAC address. Finally, it preforms 1 type of exploitation (active and passive), depending on the service being used.

Given the nature of MITM attacks, an emphasis is given to the continuous forwarding of packets between both victims, as to not alert the users that the communication has been compromised. This usually requires a fast forwarding process to simulate a normal traffic flow.

The proposed final system overcomes this difficulty using the Linux IP forwarding mechanism and deactivating ICMP redirecting in the attacker machine.

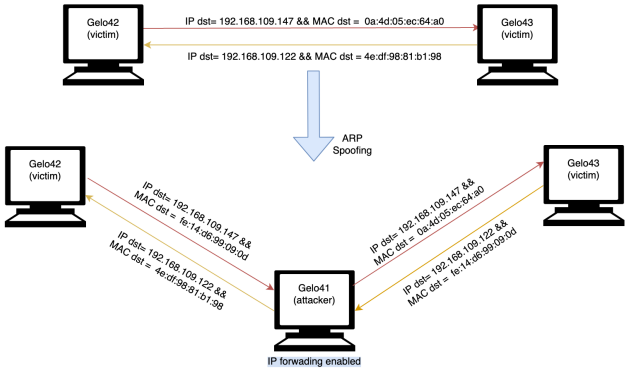

Fig. 2. Communication flow before and after ARP Spoofing

A. FTP Passive Attack

Using the built-in Scapy's *sniff* function, the attacker starts by sniffing the on-play interface filtering packets by the TCP port (port **21** in the context) and checking whether specific keywords (like USER or PASS) are present on the packet's payload. The server's replies to these packets are then evaluated to see whether the previous credentials are valid or not. Those credentials are stored for possible further usage.

B. SNMP Passive/Active Attack

Using the built-in Scapy's *sniff* function, the attacker starts by sniffing the on-play interface filtering packets by the UDP port (port **161** in the context).

After detecting SNMP packets, the information below is printed to the stdout:

- IP destination of the packet
- IP source of the packet
- OID and OID value
- Community String

Since that information is sent using ASN.1 protocol, the system asks the user to enter manually the displayed community string and IP dst to perform an active SNMP attack. This attack consists of sending a craft SNMP GET packet to the device, querying it for its system description. The answer for that packet is then outputted to the stdout.

C. Telnet Passive/Active Attack

1) Passive Attack: The attacker sniffs the network traffic, through Scapy's built-in *sniff* function, filtering packets by the Telnet associated port (**23**). These packets don't have any type of encryption, and as such can be easily analyzed and outputted to stdout.

The developed system also detects which parties are involved in the communication and what their roles are (who's the client and who's the server). It achieves this by analyzing the contents of the exchanged packets and correlating them with the expected Telnet's mode of operation. On a Telnet connection the server acknowledges every keystroke inputted by the user - meaning every action performed by the user is then repeated back by the server to confirm the received input. By analyzing the origin of the repeated messages, is then possible to determine which of the involved parties plays the server role.

2) Active Attack: After both roles in the communication are determined the system waits for an "appropriate" time to inject a nefarious packet in the communication. (Note: The need for this "appropriate" time is debatable, the main idea behind it was to find a small time window where no immediate packets are exchanged and inject the packet there. This way there's a better chance for the attacker to "guess" the correct Seq. and ACK numbers being used. Since the attacker is placed already in-between both parties this sort of timing procedure is unnecessary.) This nefarious packet launches a reverse shell targeting the server and returning a valid shell connection back to the attacker's 1337 port using:

```
/bin/bash -i >& /dev/tcp/192.168.109.138/1337 0>&1
```

The attacker, waiting for this reverse shell using netcat, then gains shell access to the server, impersonating the legitimate client. Finally, a forged RST TCP packet is sent to the server closing the previous Telnet client-server communication, effectively shutting down the user's connection.

D. SMTP Passive Attack

Using the built-in Scapy's *sniff* function, the attacker starts by sniffing the on-play interface filtering packets by the TCP port (port **25** in the context) and checking for specific keywords in the packet's payload (sent from a client to the SMPT server), such as MAIL FROM/RCPT TO/DATA. That

information is parsed and then outputted to the stdout. This passive attack ends checking if the server replies with the indication that the previous email was successfully queued.

IV. Results

Before any type of attack, the attacker is prompted by the program with the choice of the victims' IP - based on the ARP table - and the type of attack desired, based on FTP, SNMP, Telnet or SMTP . This workflow is shown in figure 3.

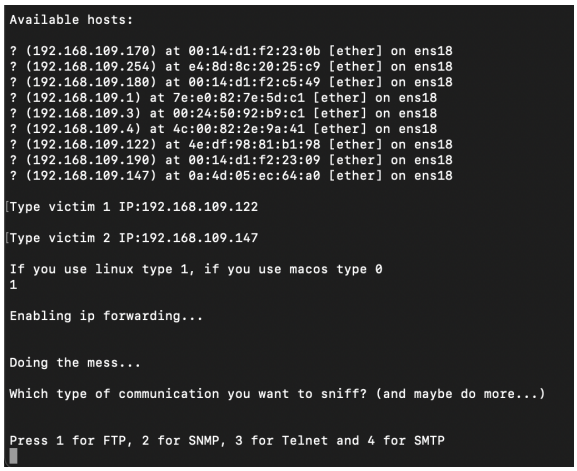

Fig. 3. Choice of victims and desired attack

The demonstration of the FTP passive attack was accomplished by creating a user and password associated to a FTP account that the FTP server recognizes as being valid. As depict in figure 4, one of the victims (*192.168.109.122*) is accessing the FTP server located at *192.168.109.147* and the attacker (left in the figure) has real-time access to the user and pass as valid credentials.

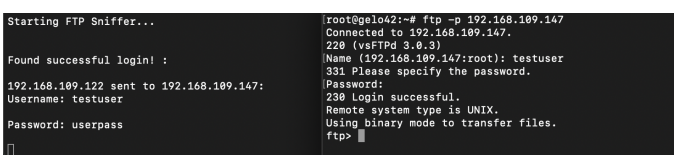

Fig. 4. FTP passive attack

For the demonstration of both SNMP active and passive attack, the below described environment was created:

As shown in figure 5, a remote user is querying the machine at *192.168.109.147* about its System Up Time, using a *snmpget* command. The attacker (left in the figure) is prompted with real-time data about the response to that command, including the community string *veryprivate*. To perform the active attack, the user must type *Ctrl-C* and enter, as described in section **??**, the necessary information. As depicted in figure 6, the victim of this attack in this created environment is running Debian with kernel version 4.19, it's a 64bit OS and the hostname is "gelo43". Those types of information can be useful for any other type of attacks.

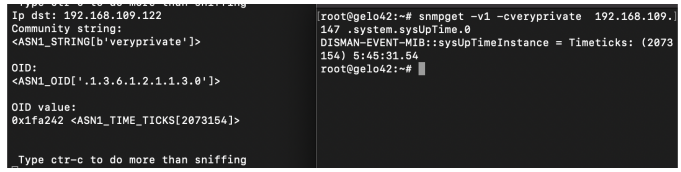

Fig. 5. SNMP passive attack

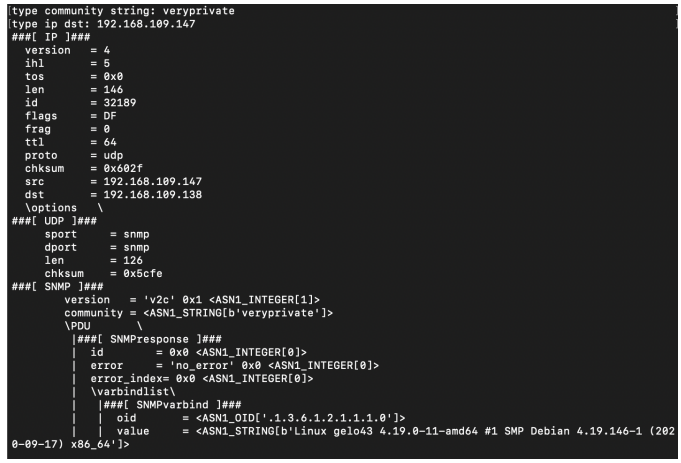

Fig. 6. SNMP active attack

In order to demonstrate both the passive and active Telnet attacks, the setup present in 7 was implemented. One VM was used as a legitimate client for the Telnet service while the attacker simultaneously intercepted and sniffed its connection - allowing the attacker to passively collect all the client's traffic. After both victims were identified (Telnet's client and server), the attacker launches the mentioned reverse shell and waits for it on a running netcat instance - which can be seen in the right part of figure 7. Finally, the user's legitimate connection is shutdown due to the aforementioned RST forged packet sent by the attacker to the server.

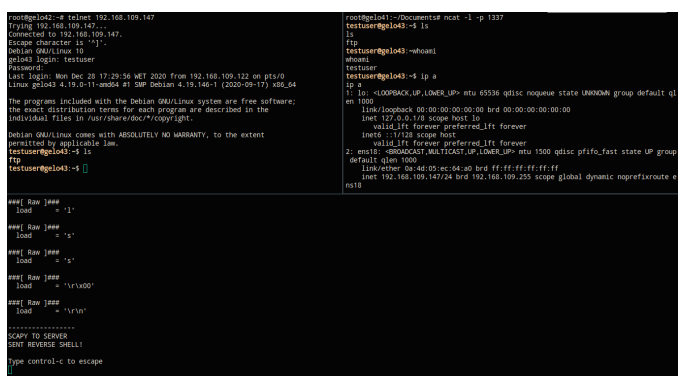

Fig. 7. Telnet active attack - user in top-left, attacker reverse shell in top-right, attacker Scapy bottom

For the demonstration of the SMTP passive attack, one of the victims is *telnetting* to the SMTP Postfix server located at *192.168.109.147*. The victim sends a regular email to the

server and, as shown left in figure 8, the attacker has access to the user sending email, the receiver and the email body.

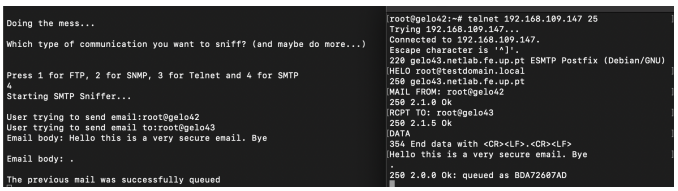

Fig. 8. SMTP passive attack

V. Conclusion

Through this project we were able to have a 1st hand experience with the procedures involved in a MITM attack, the security flaws present in the types of services that fall victim to them and their glaring consequences.

In a general fashion, most of the compromised services analysed would have been safe if their communications were implemented on top of encrypted channels - rendering the attackers interpretation of the detected information useless. In fact, encryption also plays an extremely important role against the **Evesdropping** step mentioned in section I, since no identifying content can be detected by the attacker if all traffic is encrypted.

Prevention of the **Positioning** attacker's step can be done by not accepting any unsolicited ARP responses or by some sort of cross-checking of ARP responses involving the DHCP server such that both dynamic and static IP addresses are verified.

In summary, this project showed us how important concepts like encryption really are in a practical way. A special high note should be given to Scapy, not only for its ease of features (and implementation), but also its documentation and available resources.

References

[1] *ARP Spoofing*. URL: https://www.ionos.com/digitalguide/server/security/arp-spoofing-attacks-from-the-internal-network/ (visited on 11/01/2020).

[2] Philippe Biondi. *Scapy Documentation*. URL: https://scapy.readthedocs.io/en/latest/ (visited on 12/28/2020).

[3] *Postfix - Debian Wiki*. URL: https://wiki.debian.org/SNMP (visited on 12/04/2020).

[4] Lubos Rendek. *How to configure FTP server on Debian 9 Stretch Linux*. URL: https://linuxconfig.org/how-to-configure-ftp-server-on-debian-9-stretch-linux (visited on 11/05/2020).

[5] *SNMP - Debian Wiki*. URL: https://wiki.debian.org/SNMP (visited on 11/06/2020).

[6] *TCP hijacking*. URL: https://www.techrepublic.com/article/tcp-hijacking/ (visited on 12/04/2020).