Semester Learning Portfolio

Alexander Schandorf Sumczynski

October 06, 2025

Abstract

this is the firewall-friendly

Contents

0.1 Introduction for typst

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \tag{0.1}$$

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

As we can see, it is not numbered.

0.2 Theorems

The template uses `great-theorems` for theorems. Here is an example of a theorem:

Theorem 0.1. (Example Theorem): *This is an example theorem.*

Proof. This is the proof of the example theorem. □

We also provide `definition`, `lemma`, `remark`, `example`, and `questions` among others. Here is an example of a definition:

Definition 0.2. (Example Definition): *This is an example definition.*

***Question* 0.3.** (Custom mathblock?): How do you define a custom mathblock?

1 Authentication + Link Layer Security: Lecture One

1.1 Notes leacure one

Notes 1.1. (Remote User - Authenticationusing Asymmetric Encryption):

$$\begin{aligned} \text{A} \rightarrow \text{AS} &: ID_A \parallel ID_B \\ \text{AS} \rightarrow \text{A} &: \text{E}(PR_{as}, [ID_A \parallel PU_a \parallel T]) \parallel \text{E}(PR_{as}, [ID_B \parallel PU_b \parallel T]) \\ \text{A} \rightarrow \text{B} &: \text{E}(PR_{as}, [ID_A \parallel PU_a \parallel T]) \parallel \text{E}(PR_{as}, [ID_B \parallel PU_b \parallel T]) \parallel \text{E}(PU_b, \text{E}(PR_a, [K_s \parallel T])) \end{aligned}$$

$$\begin{aligned} \text{A} \rightarrow \text{KDC} &: ID_A \parallel ID_B \\ \text{KDC} \rightarrow \text{A} &: \text{E}(PR_{\text{auth}}, [ID_B \parallel PU_b]) \\ \text{A} \rightarrow \text{B} &: \text{E}(PU_b, [N_a \parallel ID_A]) \\ \text{B} \rightarrow \text{KDC} &: ID_A \parallel ID_B \parallel \text{E}(PU_{\text{auth}}, N_a) \\ \text{KDC} \rightarrow \text{B} &: \text{E}(PR_{\text{auth}}, [ID_A \parallel PU_a]) \parallel \text{E}(PU_b, \text{E}(PR_{\text{auth}}, [N_a \parallel K_s \parallel ID_A \parallel ID_B])) \\ \text{B} \rightarrow \text{A} &: \text{E}(PU_a, [N_b \parallel \text{E}(PR_{\text{auth}}, [N_a \parallel K_s \parallel ID_A \parallel ID_B])]) \\ \text{A} \rightarrow \text{B} &: \text{E}(K_s, N_b) \end{aligned}$$

1.2 Network Security Assignment part 1

Assignment 1.2.

Objective: Research and write a concise paragraph about techniques used to mitigate ARP spoofing and Spanning Tree Protocol (STP) attacks (Layer 2 attacks). Please write details on how the chosen technique detects and prevents the attack, and any potential limitations they may have in a network environment. Please also mention your opinion about the complexity of the techniques you found.

Answer 1.3. (ARP Spoofing Mitigation):

(i) **Static ARP entries:** Using static entries in the ARP table means the IP–MAC mapping cannot be altered by ARP spoofing. The limitation is that when a new device joins the network, its IP–MAC pair must be manually added to the ARP tables of the relevant devices. its nice that i can setup this in a static mac adreses but let say that i ahve to do this for a capnut and maitnign this so alle vesties are update date and

(ii) **Dynamic ARP Inspection (DAI):**Is a technique where the switches are configured to map each device in the network to a specific IP–MAC pair. If an ARP spoofing attack occurs, then the switch detects that there is an unauthorized ARP request. The limitation of this method is that the switch must be set up with DAI and must be a supported type of switch. This is a better solution than the static assigning since there is a dynamic system in the switches that can help manage the ARP spoofing attacks instead of manually setting each device.

(iii) **XArp:** Is an anti-spoofing software that can detect if an ARP spoofing attack is being performed on a target system that has installed the XArp on the system, and this is the limitation—that I have to install the XArp and make sure that it's up to date and has no vulnerabilities in this program.

Answer 1.4. (STP Attacks Mitigation):

(i) **BPDU Guard** is a security feature that automatically puts a PortFast-configured access port into an error-disabled state when it receives any BPDU, protecting the STP domain from rogue switches or misconfiguration

(ii) **Root Guard** is a security feature that prevents non-root ports from becoming root ports by placing them into a root-inconsistent state if they receive superior BPDUs, ensuring the STP topology remains stable and protecting the network from rogue root bridge elections.

1.3 Network Security Assignment part 2

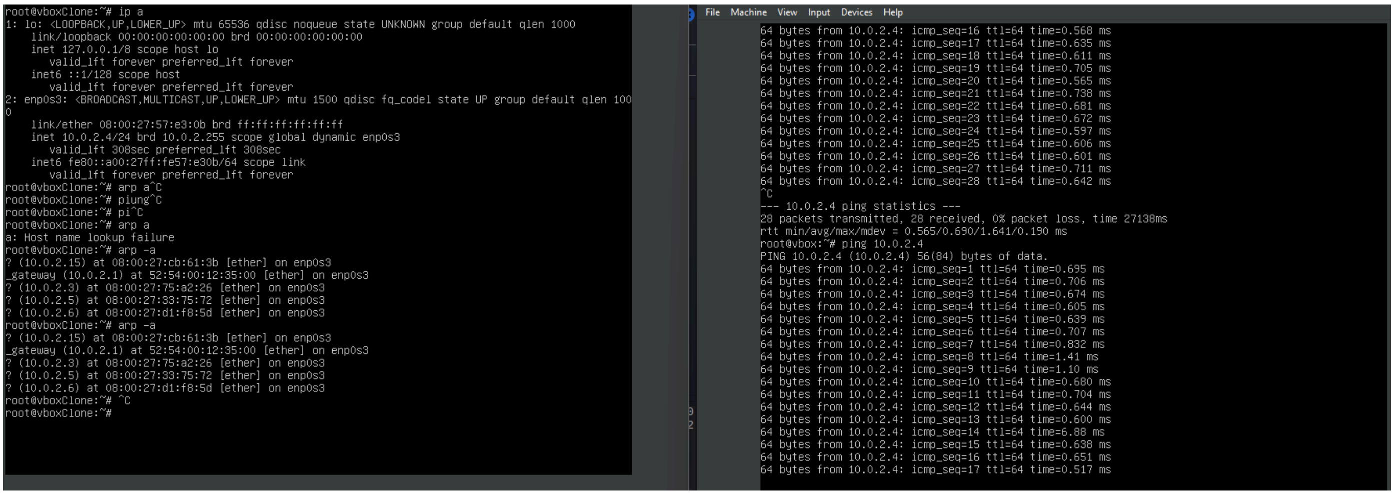
Assignment 1.5.

Objective: In this assignment we are going to emulate a Man-in-the-Middle (MITM) attack using this network topology.

As an attacker we should connect to the switch to be able to communicate with the target/ victim hosts. From now on, we refer to our two targets hosts as victims.

Answer 1.6. (Experiencing Layer 2 attacks):

(i) The setup I have is two lightweight Lubuntu systems and a Kali Linux where the Man-in-the-Middle attack will be performed. The network is connected to a NAT network through my local machine.



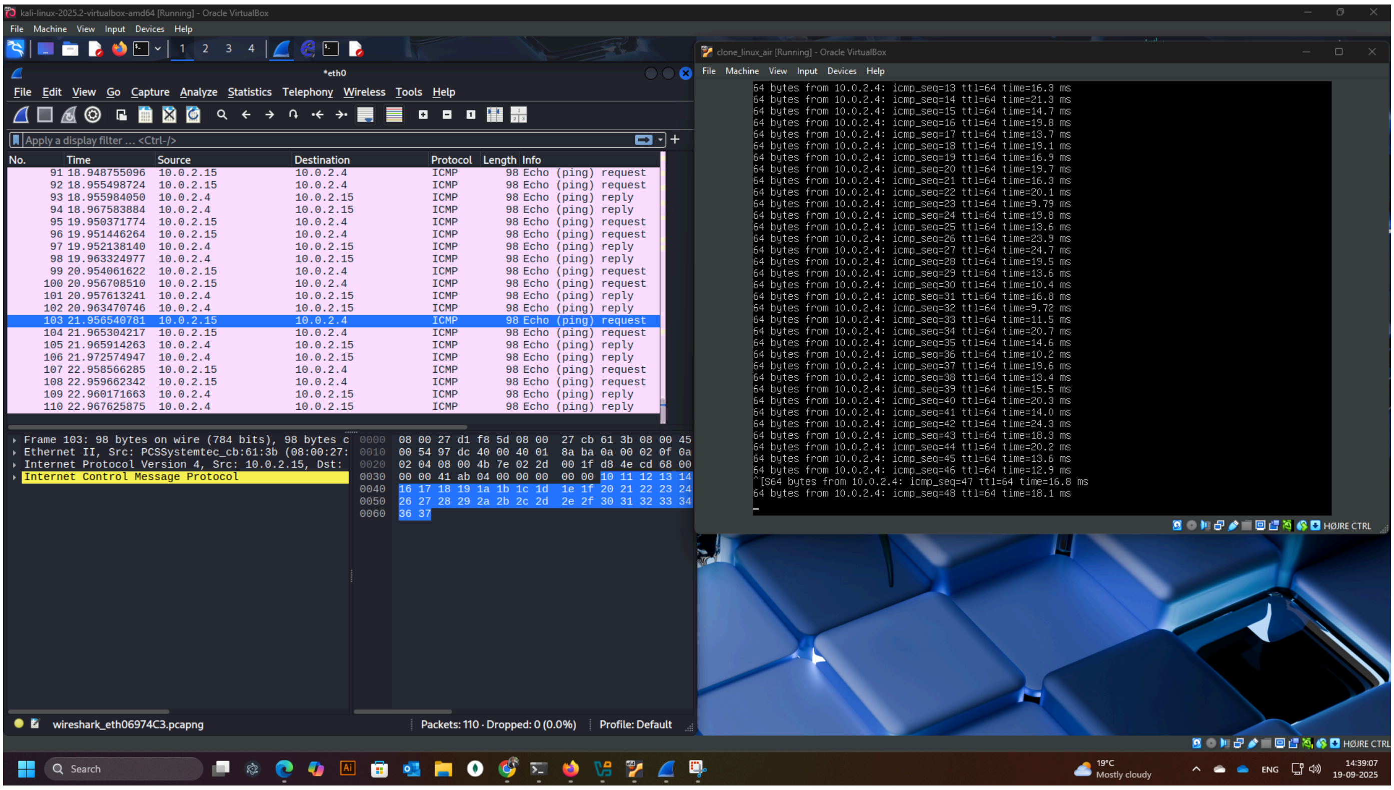
```
root@vboxClone:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 100
0
    link/ether 08:00:27:57:e3:0b brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 308sec preferred_lft 308sec
    inet6 fe80::a00:27ff:fe57:e30b/64 scope link
       valid_lft forever preferred_lft forever
root@vboxClone:~# arp a^C
root@vboxClone:~# piung^C
root@vboxClone:~# pi^C
root@vboxClone:~# arp a
a: Host name lookup failure
root@vboxClone:~# arp -a
? (10.0.2.15) at 08:00:27:cb:61:3b [ether] on enp0s3
_gateway (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
? (10.0.2.3) at 08:00:27:75:a2:26 [ether] on enp0s3
? (10.0.2.5) at 08:00:27:33:75:72 [ether] on enp0s3
? (10.0.2.6) at 08:00:27:d1:f8:5d [ether] on enp0s3
root@vboxClone:~# arp -a
? (10.0.2.15) at 08:00:27:cb:61:3b [ether] on enp0s3
_gateway (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
? (10.0.2.3) at 08:00:27:75:a2:26 [ether] on enp0s3
? (10.0.2.5) at 08:00:27:33:75:72 [ether] on enp0s3
? (10.0.2.6) at 08:00:27:d1:f8:5d [ether] on enp0s3
root@vboxClone:~# ^C
root@vboxClone:~#
```

```
64 bytes from 10.0.2.4: icmp_seq=16 ttl=64 time=0.568 ms
64 bytes from 10.0.2.4: icmp_seq=17 ttl=64 time=0.635 ms
64 bytes from 10.0.2.4: icmp_seq=18 ttl=64 time=0.611 ms
64 bytes from 10.0.2.4: icmp_seq=19 ttl=64 time=0.705 ms
64 bytes from 10.0.2.4: icmp_seq=20 ttl=64 time=0.565 ms
64 bytes from 10.0.2.4: icmp_seq=21 ttl=64 time=0.738 ms
64 bytes from 10.0.2.4: icmp_seq=22 ttl=64 time=0.681 ms
64 bytes from 10.0.2.4: icmp_seq=23 ttl=64 time=0.672 ms
64 bytes from 10.0.2.4: icmp_seq=24 ttl=64 time=0.597 ms
64 bytes from 10.0.2.4: icmp_seq=25 ttl=64 time=0.606 ms
64 bytes from 10.0.2.4: icmp_seq=26 ttl=64 time=0.601 ms
64 bytes from 10.0.2.4: icmp_seq=27 ttl=64 time=0.711 ms
64 bytes from 10.0.2.4: icmp_seq=28 ttl=64 time=0.642 ms
^C
--- 10.0.2.4 ping statistics ---
28 packets transmitted, 28 received, 0% packet loss, time 27138ms
rtt min/avg/max/mdev = 0.565/0.690/1.641/0.190 ms
root@vbox:~# ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.695 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=0.706 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=0.674 ms
64 bytes from 10.0.2.4: icmp_seq=4 ttl=64 time=0.605 ms
64 bytes from 10.0.2.4: icmp_seq=5 ttl=64 time=0.639 ms
64 bytes from 10.0.2.4: icmp_seq=6 ttl=64 time=0.707 ms
64 bytes from 10.0.2.4: icmp_seq=7 ttl=64 time=0.832 ms
64 bytes from 10.0.2.4: icmp_seq=8 ttl=64 time=1.41 ms
64 bytes from 10.0.2.4: icmp_seq=9 ttl=64 time=1.10 ms
64 bytes from 10.0.2.4: icmp_seq=10 ttl=64 time=0.680 ms
64 bytes from 10.0.2.4: icmp_seq=11 ttl=64 time=0.704 ms
64 bytes from 10.0.2.4: icmp_seq=12 ttl=64 time=0.644 ms
64 bytes from 10.0.2.4: icmp_seq=13 ttl=64 time=0.600 ms
64 bytes from 10.0.2.4: icmp_seq=14 ttl=64 time=6.88 ms
64 bytes from 10.0.2.4: icmp_seq=15 ttl=64 time=0.638 ms
64 bytes from 10.0.2.4: icmp_seq=16 ttl=64 time=0.651 ms
64 bytes from 10.0.2.4: icmp_seq=17 ttl=64 time=0.517 ms
```

Figure 1: The two Lubuntu machines

In Figure 1 there are the two lightweight Lubuntu machines. The right machine is performing a ping to the other machine (on the left), and the left machine is running the arp -a command to show the devices that are running on this NAT network.

(ii) The next step is to perform the ARP spoofing attack on the two targets. To do that, on the Kali machine I use the program Ettercap to scan for the two targets and select them as victims, where it will then perform the spoofing attack.



```
No.  Time          Source       Destination  Protocol Length Info
  91 18.948755096  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
  92 18.955498724  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
  93 18.955984050  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
  94 18.967583884  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
  95 19.950371774  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
  96 19.951446264  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
  97 19.952138140  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
  98 19.963324977  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
  99 20.954061622  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
 100 20.956708510  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
 101 20.957613241  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
 102 20.963470746  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
 103 21.956540781  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
 104 21.965304217  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
 105 21.965914263  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
 106 21.972574947  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
 107 22.958566285  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
 108 22.959662342  10.0.2.15    10.0.2.4     ICMP     98 Echo (ping) request
 109 22.960171663  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
 110 22.967625875  10.0.2.4     10.0.2.15    ICMP     98 Echo (ping) reply
```

```
› Frame 103: 98 bytes on wire (784 bits), 98 bytes c    0000  08 00 27 d1 f8 5d 08 00  27 cb 61 3b 08 00 45
› Ethernet II, Src: PCSSystemtec_cb:61:3b (08:00:27:    0010  00 54 97 dc 40 00 40 01  8a ba 0a 00 02 0f 0a
› Internet Protocol Version 4, Src: 10.0.2.15, Dst:     0020  02 04 08 00 4b 7e 02 2d  00 1f d8 4e cd 68 00
› Internet Control Message Protocol                     0030  00 00 41 ab 04 00 00 00  00 00 10 11 12 13 14
                                                        0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24
                                                        0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34
                                                        0060  36 37
```

```
wireshark_eth06974C3.pcapng        Packets: 110 · Dropped: 0 (0.0%)   Profile: Default
```

```
64 bytes from 10.0.2.4: icmp_seq=13 ttl=64 time=16.3 ms
64 bytes from 10.0.2.4: icmp_seq=14 ttl=64 time=21.3 ms
64 bytes from 10.0.2.4: icmp_seq=15 ttl=64 time=14.7 ms
64 bytes from 10.0.2.4: icmp_seq=16 ttl=64 time=19.8 ms
64 bytes from 10.0.2.4: icmp_seq=17 ttl=64 time=13.7 ms
64 bytes from 10.0.2.4: icmp_seq=18 ttl=64 time=19.1 ms
64 bytes from 10.0.2.4: icmp_seq=19 ttl=64 time=16.9 ms
64 bytes from 10.0.2.4: icmp_seq=20 ttl=64 time=19.7 ms
64 bytes from 10.0.2.4: icmp_seq=21 ttl=64 time=16.3 ms
64 bytes from 10.0.2.4: icmp_seq=22 ttl=64 time=20.1 ms
64 bytes from 10.0.2.4: icmp_seq=23 ttl=64 time=9.79 ms
64 bytes from 10.0.2.4: icmp_seq=24 ttl=64 time=19.8 ms
64 bytes from 10.0.2.4: icmp_seq=25 ttl=64 time=13.6 ms
64 bytes from 10.0.2.4: icmp_seq=26 ttl=64 time=23.9 ms
64 bytes from 10.0.2.4: icmp_seq=27 ttl=64 time=24.7 ms
64 bytes from 10.0.2.4: icmp_seq=28 ttl=64 time=19.5 ms
64 bytes from 10.0.2.4: icmp_seq=29 ttl=64 time=13.6 ms
64 bytes from 10.0.2.4: icmp_seq=30 ttl=64 time=10.4 ms
64 bytes from 10.0.2.4: icmp_seq=31 ttl=64 time=16.8 ms
64 bytes from 10.0.2.4: icmp_seq=32 ttl=64 time=9.72 ms
64 bytes from 10.0.2.4: icmp_seq=33 ttl=64 time=11.5 ms
64 bytes from 10.0.2.4: icmp_seq=34 ttl=64 time=20.7 ms
64 bytes from 10.0.2.4: icmp_seq=35 ttl=64 time=14.6 ms
64 bytes from 10.0.2.4: icmp_seq=36 ttl=64 time=10.2 ms
64 bytes from 10.0.2.4: icmp_seq=37 ttl=64 time=19.6 ms
64 bytes from 10.0.2.4: icmp_seq=38 ttl=64 time=13.4 ms
64 bytes from 10.0.2.4: icmp_seq=39 ttl=64 time=15.5 ms
64 bytes from 10.0.2.4: icmp_seq=40 ttl=64 time=20.3 ms
64 bytes from 10.0.2.4: icmp_seq=41 ttl=64 time=14.0 ms
64 bytes from 10.0.2.4: icmp_seq=42 ttl=64 time=24.3 ms
64 bytes from 10.0.2.4: icmp_seq=43 ttl=64 time=18.3 ms
64 bytes from 10.0.2.4: icmp_seq=44 ttl=64 time=20.2 ms
64 bytes from 10.0.2.4: icmp_seq=45 ttl=64 time=13.6 ms
64 bytes from 10.0.2.4: icmp_seq=46 ttl=64 time=12.9 ms
^[S64 bytes from 10.0.2.4: icmp_seq=47 ttl=64 time=16.8 ms
64 bytes from 10.0.2.4: icmp_seq=48 ttl=64 time=18.1 ms
_
```

Figure 2: The two Lubuntu machines

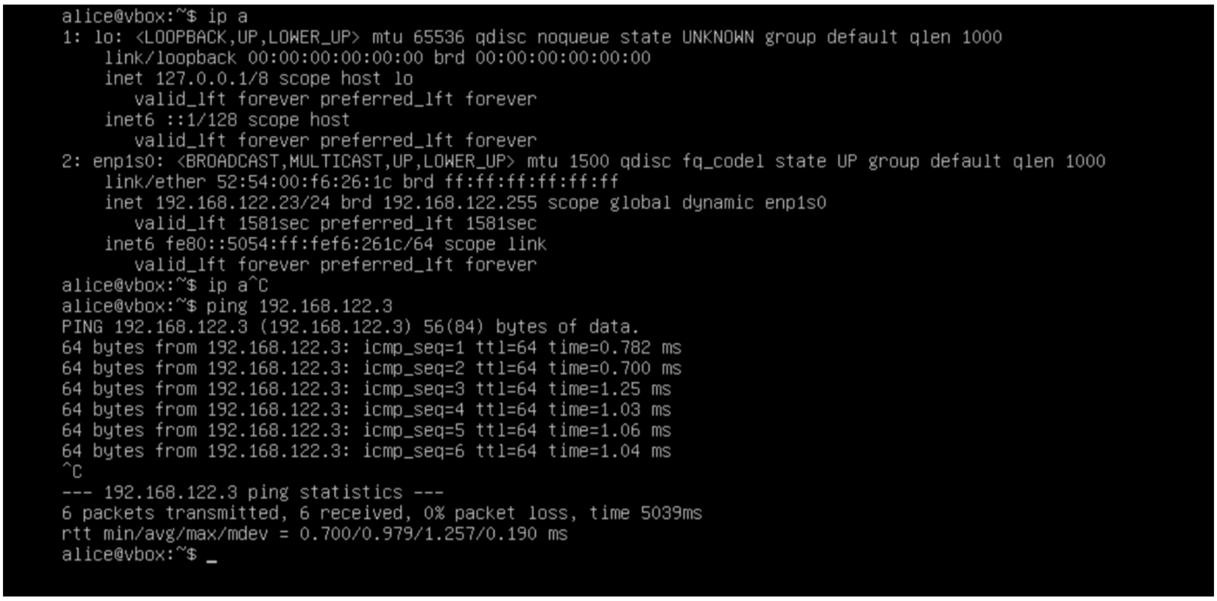
In Figure 2, it shows how the attack is under execution, where on the left is the Lubuntu machine that performs a ping to the other Lubuntu machine (on the right). But since we have created a Man-in-the-Middle between the two targets, the traffic can now be seen on the Kali machine, as shown in the image. In this, Wireshark is capturing the traffic between the two machines.

2 TCP/IP Internet Layer Security

2.1 Assignment Experiencing IPsec (Group) part one

Group:
Alexander Sumczynski, Marcus Kolbe, Luca,

Task 1: In the firt part of the start is setting up the two system ubunto severs that suold communicate toghter,



```
alice@vbox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:f6:26:1c brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.23/24 brd 192.168.122.255 scope global dynamic enp1s0
       valid_lft 1581sec preferred_lft 1581sec
    inet6 fe80::5054:ff:fef6:261c/64 scope link
       valid_lft forever preferred_lft forever
alice@vbox:~$ ip a^C
alice@vbox:~$ ping 192.168.122.3
PING 192.168.122.3 (192.168.122.3) 56(84) bytes of data.
64 bytes from 192.168.122.3: icmp_seq=1 ttl=64 time=0.782 ms
64 bytes from 192.168.122.3: icmp_seq=2 ttl=64 time=0.700 ms
64 bytes from 192.168.122.3: icmp_seq=3 ttl=64 time=1.25 ms
64 bytes from 192.168.122.3: icmp_seq=4 ttl=64 time=1.03 ms
64 bytes from 192.168.122.3: icmp_seq=5 ttl=64 time=1.06 ms
64 bytes from 192.168.122.3: icmp_seq=6 ttl=64 time=1.04 ms
^C
--- 192.168.122.3 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5039ms
rtt min/avg/max/mdev = 0.700/0.979/1.257/0.190 ms
alice@vbox:~$ _
```

Figure 3: The two Lubuntu machines: alice and bob

In Figure 3 shows how after sinnign up the config files that alcie macinge can ping the bob virtuel maicnge
Task 2 Pre-IPsec Capture:

In Task 2, setting up the capture traffic between the two virtual machines will first happen after some traffic has been passed through the system. Observing these packets being sent is just normal traffic that is not encrypted or anything. I can see the GET request to the Bob machine that is hosting an Apache2 service, so all the TCP handshakes and the GET/response is plain text

Task 3 Capturing IKE:

Now starting tshark, then launching the IPsec services. This will allow the capture of the IKE (Internet Key Exchange) packets. The IPsec service is stopped first so that the initial packets can be captured.

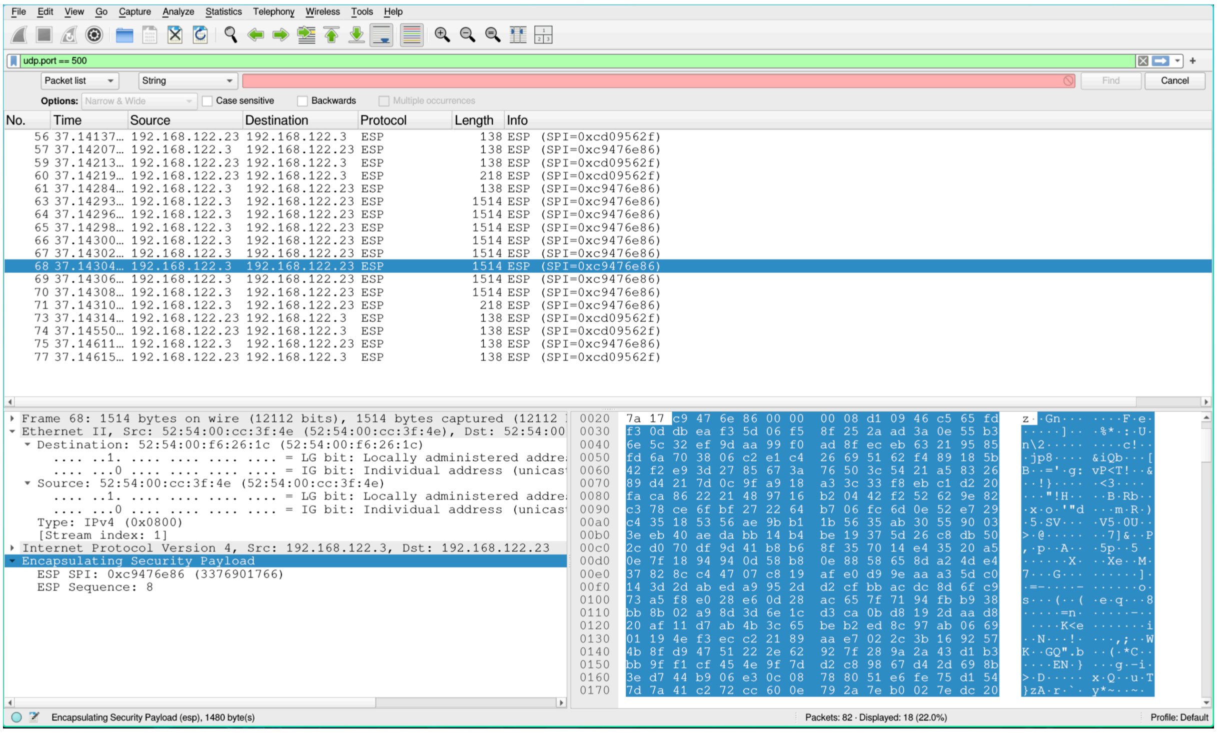
***Question* 2.1.** (What parameters are negotiated during the IKE exchange?): While observing the negotiation, several parameters are mentioned: an integrity algorithm, pseudo-random function, and the Diffie-Hellman key exchange. These different values can be seen in the payload packed

Task 4 Capturing ESP:

***Question* 2.2.** (What differences do you notice between the captured ESP packets and the plaintext packets from Task 2?): Observing the packets from Task 2 that are in plaintext, and then the packets that are encapsulated inside an ESP packet, the information is encrypted and scrambled.

***Question* 2.3.** (Why is the payload data not visible in the ESP packet? (put screenshots on your report to show that)):

The payload data is not visible in the ESP packet because IPsec's Encapsulating Security Payload (ESP) protocol encrypts it.



udp.port == 500

```
No.  Time        Source          Destination     Protocol  Length Info
  56 37.14137… 192.168.122.23 192.168.122.3  ESP        138 ESP  (SPI=0xcd09562f)
  57 37.14207… 192.168.122.3  192.168.122.23 ESP        138 ESP  (SPI=0xc9476e86)
  59 37.14213… 192.168.122.23 192.168.122.3  ESP        138 ESP  (SPI=0xcd09562f)
  60 37.14219… 192.168.122.23 192.168.122.3  ESP        218 ESP  (SPI=0xcd09562f)
  61 37.14284… 192.168.122.3  192.168.122.23 ESP        138 ESP  (SPI=0xc9476e86)
  63 37.14293… 192.168.122.3  192.168.122.23 ESP       1514 ESP  (SPI=0xc9476e86)
  64 37.14296… 192.168.122.3  192.168.122.23 ESP       1514 ESP  (SPI=0xc9476e86)
  65 37.14298… 192.168.122.3  192.168.122.23 ESP       1514 ESP  (SPI=0xc9476e86)
  66 37.14300… 192.168.122.3  192.168.122.23 ESP       1514 ESP  (SPI=0xc9476e86)
  67 37.14302… 192.168.122.3  192.168.122.23 ESP       1514 ESP  (SPI=0xc9476e86)
  68 37.14304… 192.168.122.3  192.168.122.23 ESP       1514 ESP  (SPI=0xc9476e86)
  69 37.14306… 192.168.122.3  192.168.122.23 ESP       1514 ESP  (SPI=0xc9476e86)
  70 37.14308… 192.168.122.3  192.168.122.23 ESP       1514 ESP  (SPI=0xc9476e86)
  71 37.14310… 192.168.122.3  192.168.122.23 ESP        218 ESP  (SPI=0xc9476e86)
  73 37.14314… 192.168.122.23 192.168.122.3  ESP        138 ESP  (SPI=0xcd09562f)
  74 37.14550… 192.168.122.23 192.168.122.3  ESP        138 ESP  (SPI=0xcd09562f)
  75 37.14611… 192.168.122.3  192.168.122.23 ESP        138 ESP  (SPI=0xc9476e86)
  77 37.14615… 192.168.122.23 192.168.122.3  ESP        138 ESP  (SPI=0xcd09562f)
```

```
▸ Frame 68: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112
▾ Ethernet II, Src: 52:54:00:cc:3f:4e (52:54:00:cc:3f:4e), Dst: 52:54:00
  ▾ Destination: 52:54:00:f6:26:1c (52:54:00:f6:26:1c)
      .... ..1. .... .... .... .... = LG bit: Locally administered addre
      .... ...0 .... .... .... .... = IG bit: Individual address (unicas
  ▾ Source: 52:54:00:cc:3f:4e (52:54:00:cc:3f:4e)
      .... ..1. .... .... .... .... = LG bit: Locally administered addre
      .... ...0 .... .... .... .... = IG bit: Individual address (unicas
    Type: IPv4 (0x0800)
    [Stream index: 1]
▸ Internet Protocol Version 4, Src: 192.168.122.3, Dst: 192.168.122.23
▾ Encapsulating Security Payload
    ESP SPI: 0xc9476e86 (3376901766)
    ESP Sequence: 8
```

```
0020  7a 17 c9 47 6e 86 00 00  00 08 d1 09 46 c5 65 fd   z··Gn··· ····F·e·
0030  f3 0d db ea f3 5d 06 f5  8f 25 2a ad 3a 0e 55 b3   ·····]·· ·%*·:·U·
0040  6e 5c 32 ef 9d aa 99 f0  ad 8f ec eb 63 21 95 85   n\2····· ····c!··
0050  fd 6a 70 38 06 c2 e1 c4  26 69 51 62 f4 89 18 5b   ·jp8···· &iQb···[
0060  42 f2 e9 3d 27 85 67 3a  76 50 3c 54 21 a5 83 26   B··='·g: vP<T!··&
0070  89 d4 21 7d 0c 9f a9 18  a3 3c 33 f8 eb c1 d2 20   ··!}···· ·<3····
0080  fa ca 86 22 21 48 97 16  b2 04 42 f2 52 62 9e 82   ···"!H·· ··B·Rb··
0090  c3 78 ce 6f bf 27 22 64  b7 06 fc 6d 0e 52 e7 29   ·x·o·'"d ···m·R·)
00a0  c4 35 18 53 56 ae 9b b1  1b 56 35 ab 30 55 90 03   ·5·SV··· ·V5·0U··
00b0  3e eb 40 ae da bb 14 b4  be 19 37 5d 26 c8 db 50   >·@····· ··7]&··P
00c0  2c d0 70 df 9d 41 b8 b6  8f 35 70 14 e4 35 20 a5   ,·p··A·· ·5p··5 ·
00d0  0e 7f 18 94 94 0d 58 b8  0e 88 58 65 8d a2 4d e4   ······X· ··Xe··M·
00e0  37 82 8c c4 47 07 c8 19  af e0 d9 9e aa a3 5d c0   7···G··· ······]·
00f0  14 3d 2d ab ed a9 95 2d  d2 cf bb ac dc 8d 6f c9   ·=-····- ······o·
0100  73 a5 f8 e0 28 e6 0d 28  ac 65 7f 71 94 fb b9 38   s···(··( ·e·q···8
0110  bb 8b 02 a9 8d 3d 6e 1c  d3 ca 0b d8 19 2d aa d8   ·····=n· ·····-··
0120  20 af 11 d7 ab 4b 3c 65  be b2 ed 8c 97 ab 06 69    ····K<e ·······i
0130  01 19 4e f3 ec c2 21 89  aa e7 02 2c 3b 16 92 57   ··N···!· ···,;··W
0140  4b 8f d9 47 51 22 2e 62  92 7f 28 9a 2a 43 d1 b3   K··GQ".b ··(·*C··
0150  bb 9f f1 cf 45 4e 9f 7d  d2 c8 98 67 d4 2d 69 8b   ····EN·} ···g·-i·
0160  3e d7 44 b9 06 e3 0c 08  78 80 51 e6 fe 75 d1 54   >·D····· x·Q··u·T
0170  7d 7a 41 c2 72 cc 60 0e  79 2a 7e b0 02 7e dc 20   }zA·r·`· y*~··~·
```

Encapsulating Security Payload (esp), 1480 byte(s) | Packets: 82 · Displayed: 18 (22.0%) | Profile: Default

Figure 4: ESP traffic

As seen in the Figure 4 is the is the screen shot of the ESP filter

2.2 Assignment VPN part two

SSL/TLS VPNs vs IPsec:

SSL/TLS VPNs are a method to establish a VPN connection over the TLS protocol. They use the HTTPS protocol to communicate and encrypt data. The way it works is that the client's packets are encapsulated inside TLS encryption and sent to the VPN server. The VPN server decrypts the packets and forwards the traffic to the final destination on behalf of the client. The response from the destination server is then returned to the VPN server, which re-encapsulates it in TLS and sends it back to the client. Since SSL/TLS VPNs operate over HTTPS, they are firewall-friendly. The SSL/TLS VPN protocol operates at the application layer. Comparing this protocol with IPsec. IPsec operates at the network layer, and therefore the protocol needs to establish a key-exchange method. There are two main methods: Internet Key Exchange (IKEv1) and Internet Key Exchange version 2 (IKEv2). Compared to IPsec, SSL/TLS VPNs are more effective at bypassing normal firewalls, since IPsec traffic can sometimes be blocked or require extra configuration.

WireGuard vs IPsec:

The WireGuard is a more modern VPN. It uses the following protocols:

- ChaCha20 for symmetric encryption, authenticated with Poly1305,
- Curve25519 for key exchange,
- SipHash24,
- BLAKE2s for hashing,
- HKDF for key derivation.

One of the features that WireGuard is primarily designed for is its integration in the Linux kernel, which makes installation and setup easy. WireGuard uses Curve25519 to derive the key-exchange method. Another technique that WireGuard uses is frequent rotation of the session keys, which makes the protocol more secure while still maintaining the fast connection that is one of the key features of WireGuard.

To compare this protocol to IPsec: both operate in the same network stack at Layer 3, but WireGuard has a much smaller code base, whereas IPsec has a much larger code base that makes IPsec more configurable and able to run on most operating systems. This lean design also means WireGuard is easier to audit and maintain, reducing the potential attack surface compared to the more complex IPsec implementation. While IPsec supports a wide range of cipher suites and authentication methods, which contributes to its flexibility, this complexity can also lead to more configuration errors and higher administrative overhead. WireGuard, by contrast, focuses on a fixed set of modern cryptographic primitives, providing strong security with minimal configuration and typically faster connection setup.

3 Transport Layer Security (TLS) + Secure Shell (SSH)

3.1 Assignment TLS Cipher Suite (Individual) + Analyze their components

Review Valid Combinations of TLS Cipher Suites mentioned in the slides

- **Assignment** *'Study the provided list of valid TLS cipher suites'*:

(i) **Key Exchange**
In the 10 valid TLS cipher suites, three different key exchange methods are used: *RSA, DHE, ECDHE.* RSA is a method where the two parties use public keys to encrypt the symmetric key that both sides will use. DHE (Diffie-Hellman Ephemeral) is a technique where the two parties use the Diffie-Hellman algorithm to derive a shared secret key that will be used to encrypt the messages they send to each other. ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) is another method to derive a shared secret key, but this one uses elliptic curve cryptography to achieve stronger security with smaller key sizes.

(ii) **Authentication**

RSA is a public-key algorithms. That can be used for authentication of the server, by signing secure digital messages and certificates. The security of RSA relies on the

ECDSA (Elliptic Curve Digital Signature Algorithm) works in a similar to the RSA way but is based on elliptic-curve cryptography.

(iii) **Encryption Algorithm + Mode**

AES is a symmetric-key encryption algorithm and is the one used most often in the valid TLS cipher suites from the slides. AES is a block cipher that can use three key lengths: 128-bit, 192-bit, or 256-bit. The mode defines how the blocks of data are processed. Common modes in TLS include CBC (Cipher Block Chaining) and GCM (Galois/Counter Mode). CBC mode encrypts each block based on the previous one, while GCM mode provides both encryption and built-in integrity verification.

(iv) **MAC Function**
Message Authentication is a method used to verify that a message truly comes from the sender the client is communicating with. Integrity ensures that the parties in a communication channel can confirm that none of the messages have been tampered with during transmission. In TLS, this is achieved using a MAC function such as SHA256 or SHA384, which creates a unique fingerprint for each message. If the message changes in any way, the fingerprint no longer matches, and the receiver can detect that the data has been altered.

Design 3 "Impossible" Cipher Suites + Justify Each Invalid Combination :

(i) **TLS_DH_DSA_WITH_AES_128_CBC_SHA:**
This combination is invalid because the Diffie-Hellman (DH) and the DSA are not compatible for key exchange and authentication, since DSA is designed only to sign messages. Therefore, there is no authentication in this cipher suite.

(ii) **TLS_AES_RSA_WITH_ECDH_GCM_SHA256:**
AES is a symmetric encryption algorithm and cannot be used for key exchange, while ECDH is an asymmetric key exchange method. Combining them in this order makes the suite structure incorrect and therefore an impossible combination in TLS.

3.2 Assignment SSH MITM attack (Individual)

3.3 Appendix section

Appendix A Bash code Network Layer Security part one