
Semester Learning Portfolio

Alexander Schandorf Sumczynski

September 29, 2025

ABSTRACT

this is the apprsikt

Contents

0.1 Introduction for typst	2
0.2 Theorems	2
1 Authentication + Link Layer Security: Lecture One	2
1.1 Notes leacure one	2
1.2 Network Security Assignment part 1	2
1.3 Network Security Assignment part 2	3
2 TCP/IP Internet Layer Security	5
2.1 Assignment Experiencing IPsec (Group) part one	5
2.2 Assignment VPN part two	7
2.3 Appendix section	8
Appendix A Bash code Network Layer Security part one	9

0.1 Introduction for typst

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

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

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

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

1 Authentication + Link Layer Security: Lecture One

1.1 Notes leacure one

Notes. (Remote User - Authentication using Asymmetric Encryption):

$A \rightarrow AS : ID_A \parallel ID_B$

$AS \rightarrow A : E(PR_{as}, [ID_A \parallel PU_a \parallel T]) \parallel E(PR_{as}, [ID_B \parallel PU_b \parallel T])$

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

$A \rightarrow KDC : ID_A \parallel ID_B$

$KDC \rightarrow A : E(PR_{auth}, [ID_B \parallel PU_b])$

$A \rightarrow B : E(PU_b, [N_a \parallel ID_A])$

$B \rightarrow KDC : ID_A \parallel ID_B \parallel E(PU_{auth}, N_a)$

$KDC \rightarrow B : E(PR_{auth}, [ID_A \parallel PU_a]) \parallel E(PU_b, E(PR_{auth}, [N_a \parallel K_s \parallel ID_A \parallel ID_B]))$

$B \rightarrow A : E(PU_a, [N_b \parallel E(PR_{auth}, [N_a \parallel K_s \parallel ID_A \parallel ID_B])])$

$A \rightarrow B : E(K_s, N_b)$

1.2 Network Security Assignment part 1

Assignment.

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. (ARP Spoofing Mitigation):

- 1) **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
- 2) **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.
- 3) **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. (STP Attacks Mitigation):

- 1) **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
- 2) **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.

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. (Experiencing Layer 2 attacks):

- 1) 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: <BRADACAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    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 300sec preferred_lft 300sec
    inet6 fe80::a00:27ff:fe57:e30b/64 scope link
        valid_lft forever preferred_lft forever
root@vboxClone:~# arp -a
root@vboxClone:~# ping -c
root@vboxClone:~# ping -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
? (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
? (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:~# ping -c
root@vboxClone:~#

```

```

File Machine View Input Devices Help
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 = 0.555/0.690/1.641/0.150 ms
root@vboxClone:~# 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=0.68 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.

- 2) 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.

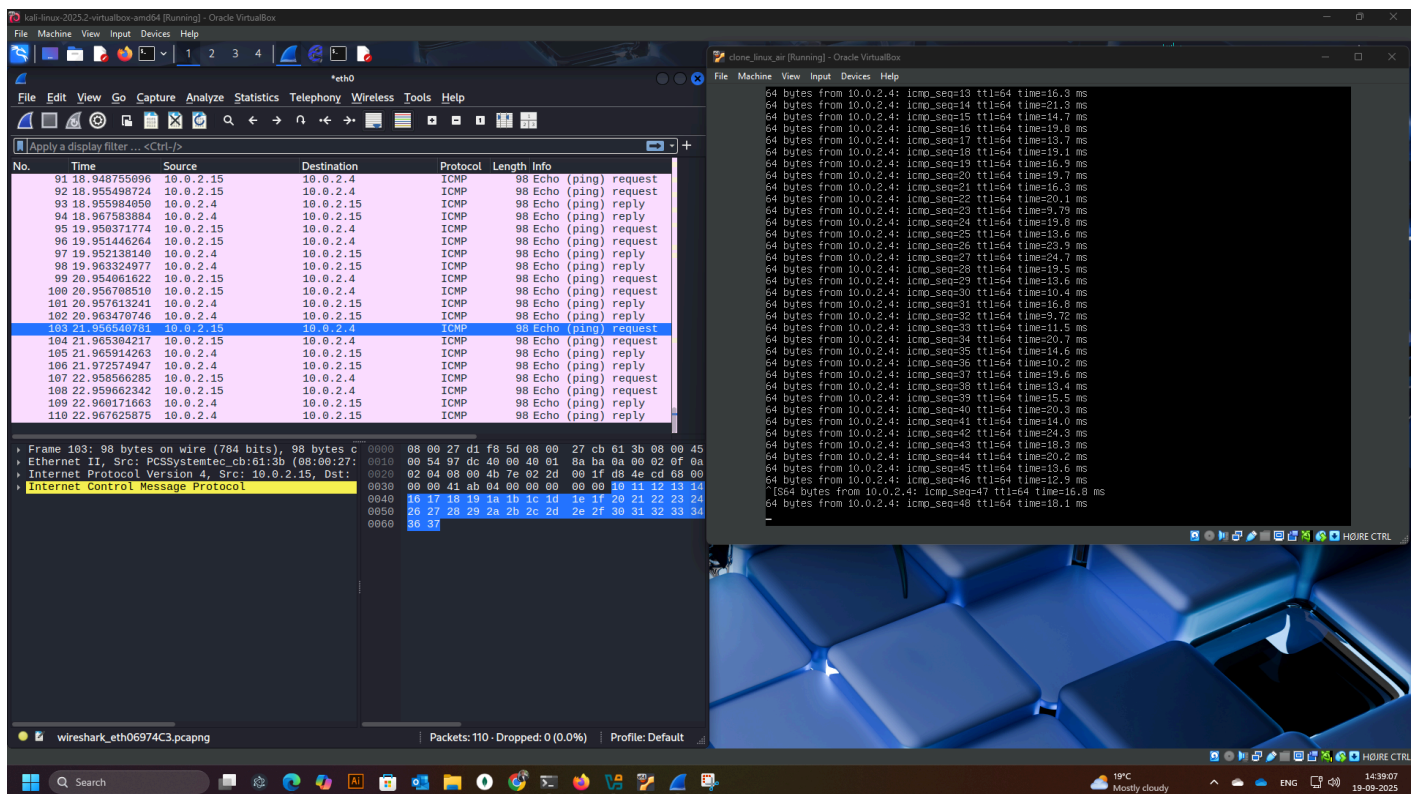


Figure 2: The two Lubuntu machines

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 first part of the start is setting up the two system ubuntu servers that should communicate together,

```
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 Ubuntu machines: alice and bob

In Figure 3 shows how after setting up the config files that alice machine can ping the bob virtual machine

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. (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. (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. (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.

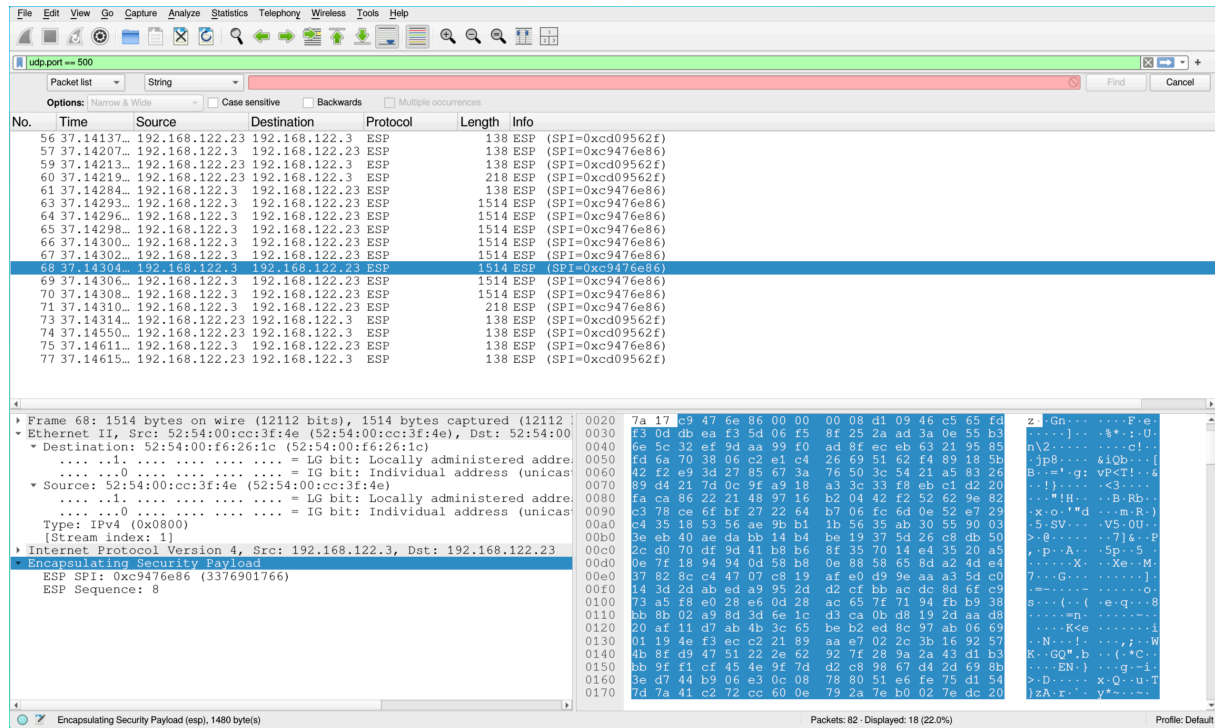


Figure 4: ESP traffic

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.

2.3 Appendix section

Appendix A Bash code Network Layer Security part one