Semester Learning Portfolio

Alexander Schandorf Sumczynski

September 29, 2025

Abstract

this is the apprsikt

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):

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

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

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.

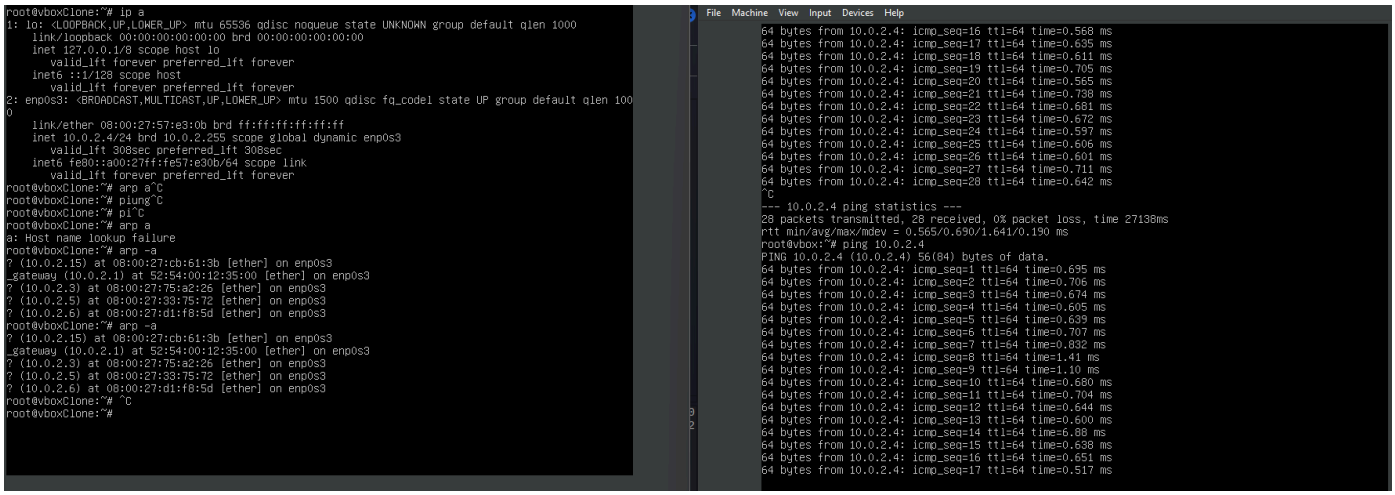

Figure 1: The two Lubuntu machines

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.

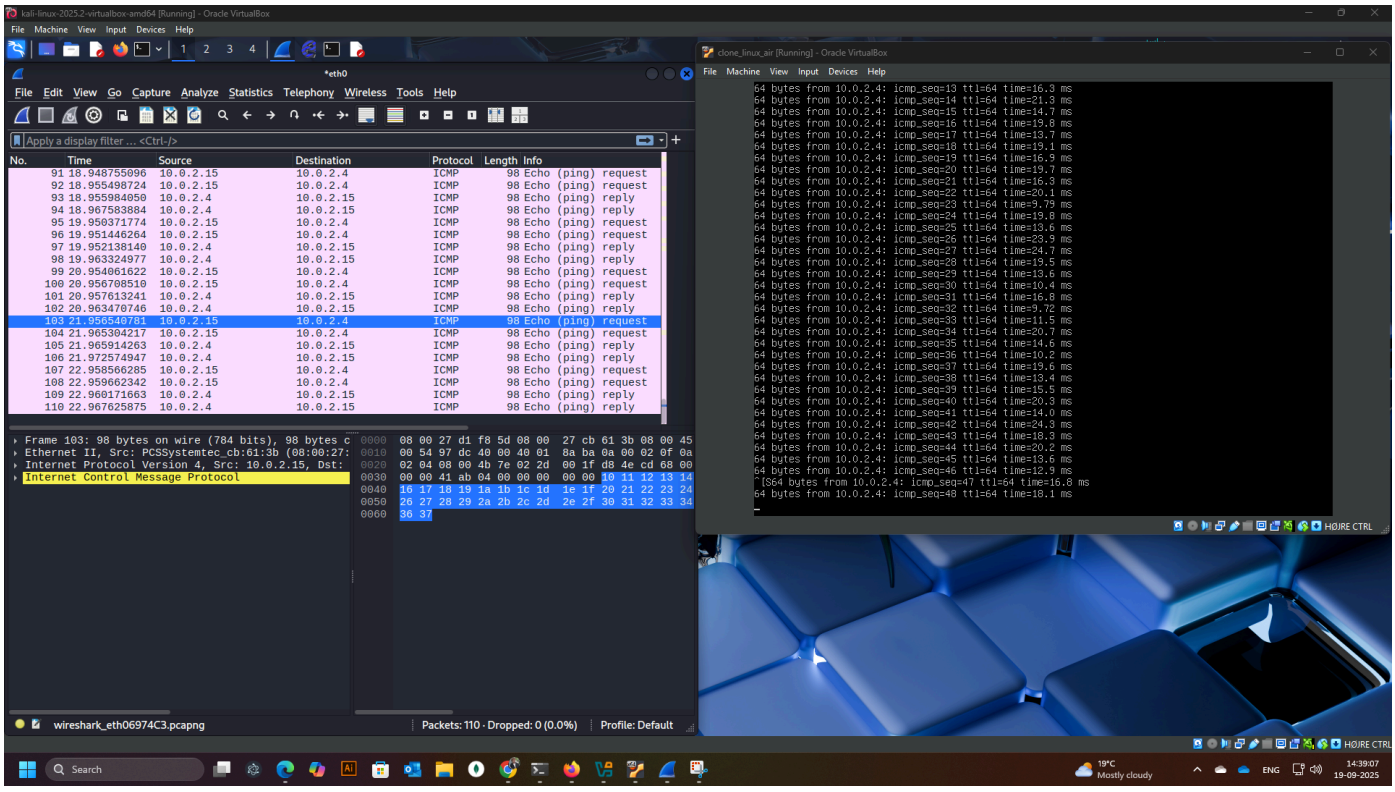

Figure 2: The two Lubuntu machines

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,

We did the this exesice in bash and asnwed with a scpript whre we cateuted alle the files thing in the one singen scpript, to see the scpript that go to Section Appendix A

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

```
#!/usr/bin/env bash

NODE1="192.168.122.77"
NODE1_USER="alice"
NODE2="192.168.122.122"
NODE2_USER="bob"

TIME=5

## No encryption

# Stop strongswan on machines
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "echo 'password' | sudo -S systemctl
stop strongswan-starter.service"
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "echo 'password' | sudo -S systemctl
stop strongswan-starter.service"

# Wait a moment
sleep 1

# Start capture
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "tshark -w unenc_capture.pcap -i
enp7s0 & sleep ${TIME} && killall tshark" &
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "tshark -w unenc_capture.pcap -i
enp7s0 & sleep ${TIME} && killall tshark" &

# Wait a moment
sleep 1

# Make traffic
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "curl http://192.168.200.153" &

# Wait
sleep $(( $TIME + 2 ))

# Get capture files
sshpass -p "password" scp ${NODE1_USER}@${NODE1}:unenc_capture.pcap
node1_unenc_capture.pcap
sshpass -p "password" scp ${NODE2_USER}@${NODE2}:unenc_capture.pcap
node2_unenc_capture.pcap

## IKE handshake

# Start capture
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "tshark -w ikehandshake_capture.pcap
-i enp7s0 & sleep ${TIME} && killall tshark" &
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "tshark -w ikehandshake_capture.pcap
-i enp7s0 & sleep ${TIME} && killall tshark" &

# Wait a moment
sleep 1

# Start strongswan on machines
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "echo 'password' | sudo -S systemctl
```

```
start strongswan-starter.service" &
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "echo 'password' | sudo -S systemctl
start strongswan-starter.service" &

# Wait
sleep $(( $TIME + 2 ))

# Get capture files
sshpass -p "password" scp ${NODE1_USER}@${NODE1}:ikehandshake_capture.pcap
node1_ikehandshake_capture.pcap
sshpass -p "password" scp ${NODE2_USER}@${NODE2}:ikehandshake_capture.pcap
node2_ikehandshake_capture.pcap

## Get encrypted traffic

# Start capture
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "tshark -w esp_capture.pcap -i
enp7s0 & sleep ${TIME} && killall tshark" &
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "tshark -w esp_capture.pcap -i
enp7s0 & sleep ${TIME} && killall tshark" &

# Wait a moment
sleep 1

# Make traffic
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "curl http://192.168.200.153" &

# Wait
sleep $(( $TIME + 2 ))

# Get capture files
sshpass -p "password" scp ${NODE1_USER}@${NODE1}:esp_capture.pcap
node1_esp_capture.pcap
sshpass -p "password" scp ${NODE2_USER}@${NODE2}:esp_capture.pcap
node2_esp_capture.pcap

## Get Tunnel vs Transport capture (tunnel is default, so we do transport here. (with
fuckery))

# Bring the ipsec tunnel down
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "echo 'password' | sudo -S systemctl
stop strongswan-starter.service"
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "echo 'password' | sudo -S systemctl
stop strongswan-starter.service"

# Wait a moment
sleep 1

# Change the config file
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "echo 'password' | sudo -S sed -i
's/type=tunnel/type=transport/' /etc/ipsec.conf "
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "echo 'password' | sudo -S sed -i
's/type=tunnel/type=transport/' /etc/ipsec.conf"

# Wait a moment
```

```
sleep 1

# Bring up ipsec again
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "echo 'password' | sudo -S systemctl
start strongswan-starter.service" &
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "echo 'password' | sudo -S systemctl
start strongswan-starter.service" &

# Wait a moment
sleep 1

# Start capture
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "tshark -w transport_capture.pcap -i
enp7s0 & sleep ${TIME} && killall tshark" &
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "tshark -w transport_capture.pcap -i
enp7s0 & sleep ${TIME} && killall tshark" &

# Wait a moment
sleep 1

# Make traffic
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "curl http://192.168.200.153" &

# Wait
sleep $(( $TIME + 2 ))

# Get capture files
sshpass -p "password" scp ${NODE1_USER}@${NODE1}:transport_capture.pcap
node1_transport_capture.pcap
sshpass -p "password" scp ${NODE2_USER}@${NODE2}:transport_capture.pcap
node2_transport_capture.pcap

# Stop ipsec again
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "echo 'password' | sudo -S systemctl
stop strongswan-starter.service"
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "echo 'password' | sudo -S systemctl
stop strongswan-starter.service"

# Revert the change
sshpass -p "password" ssh ${NODE1_USER}@${NODE1} "echo 'password' | sudo -S sed -i
's/type=transport/type=tunnel/' /etc/ipsec.conf"
sshpass -p "password" ssh ${NODE2_USER}@${NODE2} "echo 'password' | sudo -S sed -i
's/type=transport/type=tunnel/' /etc/ipsec.conf"
```