Comprehensive VPN Configuration Using IP XFRM and Network Namespaces in Kali Linux

Simone Sampognaro s322918

Prof. Fulvio Valenza
01GYLUV - Networks, cloud and application security

September 11, 2024

Contents

1 Introduction

In the realm of modern networking, Virtual Private Networks (VPNs) are crucial for ensuring secure communication over potentially untrusted networks. VPNs create encrypted tunnels through which data travels, safeguarding information from unauthorized access and eavesdropping. They play a vital role in protecting sensitive information, maintaining privacy, and facilitating secure remote access to network resources.

This project is centered on the practical implementation of VPN configurations, utilizing ip xfrm as the primary VPN solution. Additionally, Linux namespaces will be employed to construct and manage a complex network topology within a Kali Linux virtual machine.
The goal is to create a comprehensive example that covers all potential VPN configuration scenarios, demonstrating the versatility and functionality of VPN technologies in a controlled environment.

2 Network topology configuration

For this project, I referenced the network topology shown in Figure 1, where the networks, using IP addresses in the range 192.168.x.0/24, represent geographically dispersed company offices (routers PE shown in orange).
In this scenario, i also considered employees (CE1 and CE5) who need to access the corporate network remotely from their home networks (routers PE shown in light blue). The company reserves the address range 192.168.4.0/24 specifically for remote workers.

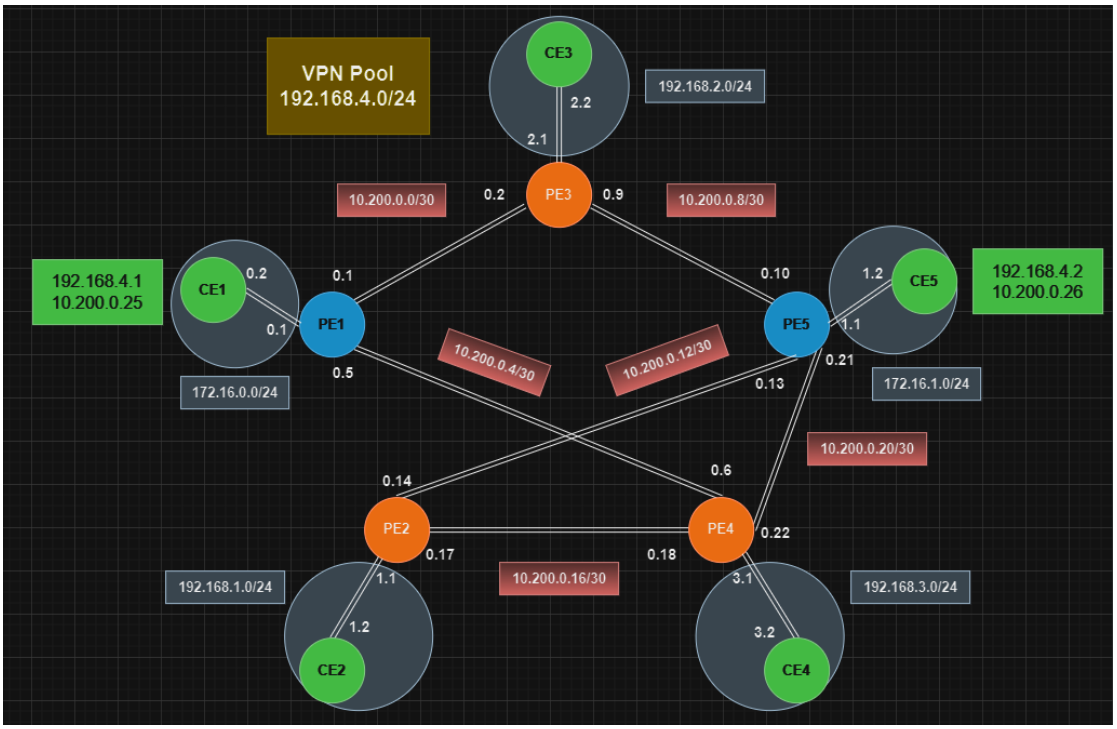

Figure 1: Simulated topology

Network interface	IP address	MAC address
CE1 – eth01	172.16.0.2/24 10.200.0.25/32 192.168.4.1/32	00:00:00:00:00:01
PE1_int - eth02	172.16.0.1/24	00:00:00:00:00:02
PE1_ext – eth03	10.200.0.1/30	00:00:00:00:00:03
PE1_ext – eth04	10.200.0.5/30	00:00:00:00:00:04
CE2 – eth05	192.168.1.2/24	00:00:00:00:00:05
PE2_int – eth06	192.168.1.1/24	00:00:00:00:00:06
PE2_ext – eth07	10.200.0.14/30	00:00:00:00:00:07
PE2_ext – eth08	10.200.0.17/30	00:00:00:00:00:08
CE3 – eth09	192.168.2.2/24	00:00:00:00:00:09
PE3_int – eth10	192.168.2.1/24	00:00:00:00:00:0a
PE3_ext – eth11	10.200.0.2/30	00:00:00:00:00:0b
PE3_ext – eth12	10.200.0.9/30	00:00:00:00:00:0c
CE4 - eth13	192.168.3.2/24	00:00:00:00:00:0d
PE4_int – eth14	192.168.3.1/24	00:00:00:00:00:0e
PE4_ext - eth15	10.200.0.6/30	00:00:00:00:00:0f
PE4_ext - eth16	10.200.0.18/30	00:00:00:00:00:10
PE4_ext – eth22	10.200.0.22/30	00:00:00:00:00:16
CE5 - eth17	172.16.1.2/24 10.200.0.26/32 192.168.4.2/32	00:00:00:00:00:11
PE5_int - eth18	172.16.1.1/24	00:00:00:00:00:12
PE5_ext – eth19	10.200.0.10/30	00:00:00:00:00:13
PE5_ext – eth20	10.200.0.13/30	00:00:00:00:00:14
PE5_ext – eth21	10.200.0.21/30	00:00:00:00:00:15

Figure 2: Configuration of the network interfaces

2.1 Linux namespaces

Linux namespaces are a feature of the Linux kernel that provide isolation for various system resources, such as process IDs, network interfaces, and file systems. Each namespace offers a separate instance of these resources, enabling the creation of isolated environments that can operate independently.
To allow communications between these namespaces, I introduced a Linux bridge, which functions as a virtual network device similar to a physical network switch, to interconnect all the virtual Ethernet interfaces of the ten namespaces.

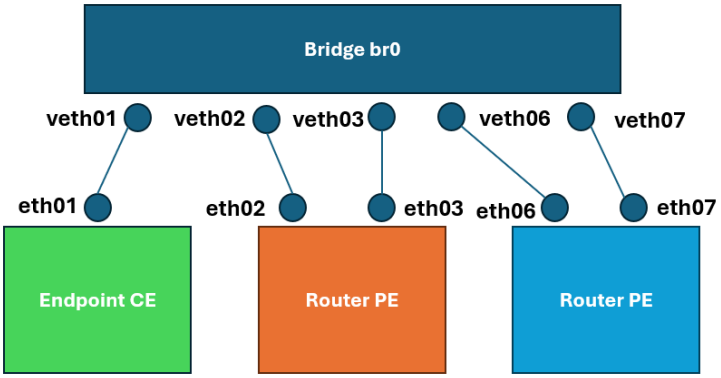

Figure 3: Connecting namespaces through Linux bridge

By combining Linux namespaces with Linux bridges, I was able to create isolated network environments. Here's a brief overview of how I accomplished this:

1. **Namespace Creation**: First, I created separate network namespaces for the endpoints (CE) and gateaways (PE). Since each namespace has its own network stack, including its own interfaces, routing tables, and firewall rules, I defined the routing tables and set up network interfaces with the IP and MAC addresses listed in Figure 2 within each namespace.

2. **Virtual Ethernet Pairs Creation (veth)**: Each veth pair acts as a virtual cable, where data sent through one end of the pair emerges at the other end.

3. **Bridge**: I created a Linux bridge and attached the virtual Ethernet devices to it. The bridge functions as a switch, allowing traffic to pass between the connected virtual devices and, consequently, between different namespaces. One end of each veth pair is placed inside a namespace, while the other end is connected to the bridge.

This setup not only simplified the process of linking the namespaces but also made it easier to capture network traffic and verify the VPN configurations.
Let's focus on the necessary commands.
First, it was important to disable the feature of sending ICMP Redirect packets and to enable the feature of packet forwarding instead:

```
sudo bash -c 'echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects'
sudo bash -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
```

Then, I created the Linux bridge:

```
sudo apt-get install bridge-utils -y
sudo brctl addbr bridge
sudo brctl stp bridge off
sudo ip link set dev bridge up
```

Instead, for the creation of the ten namespaces:

```
sudo ip netns add ce1
sudo ip netns add pe1
sudo ip netns add ce2
sudo ip netns add pe2
...
```

Next, I created all the pairs of network interfaces eth0i-veth0i, with i ranging from 1 to 22:

```
sudo ip link add eth01 type veth peer name veth01
sudo ip link add eth02 type veth peer name veth02
...
```

For all pairs of virtual interfaces, I repeated these steps afterward.
After creating the interface, I connected it to the specific network elements. The interface veth01 was plugged into the Linux bridge:

```
sudo brctl addif bridge veth01
```

The interface eth01, on the other hand, was plugged into the namespace CE1:

```
sudo ip link set eth01 netns ce1
```

I also configured the interface eth01 with the proper MAC and IP addresses (according to Figure 2) using the ifconfig and ip addr commands:

```
sudo ip netns exec ce1 ifconfig eth01 hw ether 00:00:00:00:00:01
sudo ip netns exec ce1 ip addr add 172.16.0.2/24 dev eth01
```

Finally, I activated both interfaces:

```
sudo ip netns exec ce1 ip link set dev eth01 up
sudo ip link set dev veth01 up
```

At that point, I still needed to perform some additional general configuration settings. I activated the loopback interface on all ten created namespaces:

```
sudo ip netns exec ce1 ip link set lo up
sudo ip netns exec pe1 ip link set lo up
sudo ip netns exec ce2 ip link set lo up
sudo ip netns exec pe2 ip link set lo up
...
```

I disabled the feature of receiving ICMP redirect packets for each namespace using a specific directive in the INPUT chain of iptables:

```
sudo ip netns exec ce1 iptables -I INPUT-p icmp--icmp-type redirect-j DROP
sudo ip netns exec pe1 iptables -I INPUT-p icmp--icmp-type redirect-j DROP
sudo ip netns exec ce2 iptables -I INPUT-p icmp--icmp-type redirect-j DROP
sudo ip netns exec pe2 iptables -I INPUT-p icmp--icmp-type redirect-j DROP
...
```

This following command configures ARP (Address Resolution Protocol) filtering for a specific network interface within a network namespace by setting arp_filter to 1. ARP filtering ensures that the interface only responds to ARP requests for IP addresses that it is configured with.
I introduced this configuration because, while monitoring the network with Wireshark, I noticed that some interfaces were responding to ARP requests with IP addresses that they were not configured with, which can lead to ARP spoofing and other network issues, enabling ARP filtering prevents such issues:

```
sudo ip netns exec ce1 sysctl -w net.ipv4.conf.eth01.arp_filter=1
sudo ip netns exec pe1 sysctl -w net.ipv4.conf.eth02.arp_filter=1
sudo ip netns exec pe1 sysctl -w net.ipv4.conf.eth03.arp_filter=1
...
```

Next, I enabled the feature of packet forwarding in the PE namespaces since they served as VPN gateways that needed to forward packets from one interface to another:

```
sudo ip netns exec pe1 sysctl-w net.ipv4.ip_forward=1
sudo ip netns exec pe2 sysctl-w net.ipv4.ip_forward=1
sudo ip netns exec pe3 sysctl-w net.ipv4.ip_forward=1
sudo ip netns exec pe4 sysctl-w net.ipv4.ip_forward=1
sudo ip netns exec pe5 sysctl-w net.ipv4.ip_forward=1
```

Finally, I configured static routes to ensure that each router could communicate with all others. For instance, the following routes were added for gateaway PE2:

```
sudo ip netns exec pe2 ip route add 10.200.0.0/30 via 10.200.0.13 dev eth07
sudo ip netns exec pe2 ip route add 10.200.0.4/30 via 10.200.0.18 dev eth08
sudo ip netns exec pe2 ip route add 10.200.0.8/30 via 10.200.0.13 dev eth07
```

The network created with Linux namespaces consists of ten network namespaces representing the three different elements of the simulated network topology: the remote hosts CE1 and CE5, the office endpoints CE2, CE3, and CE4, and the VPN gateways PE.

Along with the report, I also provide a Bash script that includes the full list of commands for setting up the environment. This covers the creation of network namespaces, bridges, interfaces, static routing, and VPN configurations, offering a complete guide to reproduce the setup.

3 IPSec

IPSec, short for Internet Protocol Security, is a suite of protocols designed to ensure secure communications over IP networks. Primarily used in Virtual Private Networks (VPNs), the goal of IPSec is to protect data during its transmission over public networks, like the Internet.
This includes ensuring that data cannot be accessed by unauthorized parties (confidentiality), verifying that the data has not been modified during transit (integrity), confirming that it originates from a trusted source (data authentication), and protecting against attackers capturing and replaying data packets (anti-replay protection).
IPSec operates in two main modes: Transport mode and Tunnel mode.
In Transport mode, IPSec focuses on securing the payload, the actual content of the IP packet, leaving the original IP header intact. This mode is commonly used for direct host-to-host communication, such as between a client and a server. Since the original IP header is not encrypted, intermediate routers can still read and use this information to route the packet correctly.
Conversely, in Tunnel mode, IPSec encrypts the entire IP packet, including the original header, and encapsulates it within a new IP packet with a new IP header. This mode is typically employed in communications between two networks, such as between two VPN gateways. By encapsulating the entire packet, Tunnel mode protects not only the message content but also the original routing information.
IPSec uses two main protocols to provide its security services: the Authentication Header (AH) and the Encapsulating Security Payload (ESP) protocols.
The AH protocol ensures data integrity and origin authentication by verifying that the content of the packet has not been altered during transmission and that it originates from a legitimate source. However, AH does not provide confidentiality protection, meaning the data is not encrypted and can be read by anyone who intercepts the packet.
The ESP protocol, on the other hand, offers both encryption and data integrity and authenticity. ESP encrypts the payload of the IP packet, ensuring that only authorized recipients can read it.
A fundamental part of IPSec is the concept of a Security Association (SA). An SA is a logical connection that defines the security parameters, such as the encryption and authentication algorithms, the keys used, and the connection's lifetime, used for a specific communication session.
SAs represents the "agreement" between two endpoints on how to secure the data being exchanged and are uniquely identified by a triplet consisting of a Security Parameters Index (SPI), the IP destination address, and the security protocol (AH or ESP).
Security Policies (SP) define the rules for processing IP traffic by specifying the criteria for applying Security Associations (SAs) to specific network traffic. They determine which traffic needs protection and the security measures that should be applied to data packets.

3.1 ip xfrm

ip xfrm (IP Transform) is a framework within the Linux kernel that supports IPsec by managing packet transformations and applying the necessary IPsec policies for secure data communications. It handles the creation, management, and application of Security Associations (SAs) and Security Policies (SPs) as defined in IPsec protocols.
To utilize IPsec with ip xfrm, you must define the **state** and **policies** that dictate how traffic is handled.
The **state** refers to the configuration of Security Associations (SAs), which are essential for encrypting and authenticating traffic. Each SA includes parameters like encryption keys, authentication keys, algorithms used, and other attributes necessary for securing the communication.
Here's an example of how to define a state:

```
ip xfrm state add [src IP_src ] [dst IP_dst ] [proto type_of_protection ] [spi
    SPI ] [ protection_methods ]
```

- **src IPsrc, dst IPdst** : It identifies the origin and the destination of the packets for which this SA will be used.
- **proto type_of_protection**: Defines the type of IPsec protocol used for this SA. Common values are 'esp' and 'ah'.
- **spi SPI**: Security Parameter Index (SPI) is a unique identifier for the SA. It is a 32-bit value used to distinguish between different SAs.
- **protection_methods**: Specifies the protection methods for the SA. This includes the encryption and authentication algorithms and their parameters.

The **policy** dictates how packets should be processed according to the specified rules. It determines which packets require IPsec processing and the direction (inbound, outbound) of the policy. Here's an example of how to define a policy:

```
ip xfrm policy add [src IPsrc ] [dst IPdst ] [proto PROTOCOL] [ dir DIR ] [
    action ACTION] [tmpl POLICY]
```

- **src IPsrc, dst IPdst** : It defines the origin and destination (IP address or range) of the traffic that the policy will affect.
- **proto PROTOCOL**: Defines the protocol to which the policy applies, common values are 'tcp', 'udp', and 'icmp'.
- **dir DIR**: This parameter specifies the direction of traffic to which the policy will be applied. It can be set to 'in' for incoming traffic, 'out' for outgoing traffic and 'fwd' is used for forwarded traffic, which means traffic that is being routed through the device from one network interface to another.
- **action ACTION**: This parameter determines the action to take when a packet matches the policy. The options include 'allow', which permits the traffic that matches the policy to pass through, 'drop', which discards the traffic without further processing, and 'monitor', which tracks the traffic for observation purposes without altering or blocking it.
- **tmpl POLICY**: Specifies a template for the policy, referring to the Security Association that should be applied. This parameter typically includes the details of the SA (e.g. Source and Destination IPs, IPsec protocol and mode).

4 Site-to-site VPN

4.1 Understanding Site-to-Site VPN

What is a Site-to-Site VPN?
A site-to-site virtual private network is a connection between two or more networks, frequently used by companies with multiple offices in different geographic locations.
With a site-to-site VPN, a company can securely connect its corporate network with remote offices to communicate and share resources as a single network.
It works by creating a secure, encrypted tunnel between two networks located at different sites through the public Internet. The tunnel acts as a direct link through which data can be securely transmitted, ensuring that it cannot be intercepted or read by unauthorized parties.
The process involves establishing a VPN gateway at each network end, where data entering and exiting the tunnel is encrypted and decrypted.

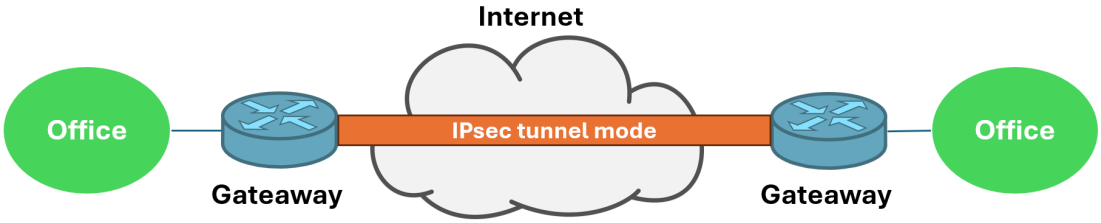

Figure 4: Site-to-Site VPN

Advantages of Site-to-Site VPN

- **Simplified Administration**: It is not needed to manage the configurations for all the end nodes, but only on the gateways.
- **Scalability**: Adding new branch offices or remote sites to the network is relatively straightforward.
- **Seamless Connectivity**: It allows different office networks to function as one unified network, enabling users to access resources across locations without manual setup. This continuous, transparent access enhances collaboration and productivity by simplifying workflows and ensuring consistent resource availability.

Disadvantages of Site-to-Site VPN

- **Lack of Endpoint Authentication**: It is not possible to authenticate the actual endpoints of the communication.
- **No End-to-End Security Against Internal Network Attacks**: While the VPN provides secure communication between gateways, it does not protect against attacks originating from within the internal networks at either end of the VPN connection.
- **Internal Traffic Analysis**: Once inside the internal networks, the data may be susceptible to monitoring and analysis by malicious insiders.
- **Single Point of Failure**: All tunneling is performed by the gateway, making it a critical component of the network. This can impact overall network performance, and any misconfiguration or failure at the gateway can expose the network to attacks and disrupt connectivity.

4.2 Step-by-Step Configuration on Linux

Considering the network topology in Figure 1, before defining the policies and states, it is necessary to add routes in the namespaces of the CEi hosts and the corporate PEi routers (i = 2, 3 and 4).
This ensures that the corporate networks, which are not directly connected to the corporate hosts (CE2, CE3, CE4), are reachable through the interface of the VPN gateway (Figure 2) located within the same IP network:

```
sudo ip netns exec ce2 ip route add 192.168.2.0/24 via 192.168.1.1 dev eth05
sudo ip netns exec ce2 ip route add 192.168.3.0/24 via 192.168.1.1 dev eth05
sudo ip netns exec ce3 ip route add 192.168.1.0/24 via 192.168.2.1 dev eth09
sudo ip netns exec ce3 ip route add 192.168.3.0/24 via 192.168.2.1 dev eth09
sudo ip netns exec ce4 ip route add 192.168.2.0/24 via 192.168.3.1 dev eth13
sudo ip netns exec ce4 ip route add 192.168.1.0/24 via 192.168.3.1 dev eth13
```

When defining routes for the PE gateaways, I had to keep in mind that the sequence of events for routing and applying IPsec policies in the Linux kernel follows a specific flow:

1. Routing using the routing tables
2. Application of IPsec policies
3. Packet transformation

Consistent with how the kernel functions, a packet coming from host CE2 and destined for host CE3 (e.g., IPsrc 192.168.1.2, IPdst 192.168.2.2) must have its destination determined by the routing table of the PE2 gateaway before it can enter the tunnel. In my static routing configuration, gateway PE2 connects to gateway PE3 through the intermediary gateway PE5. Conversely, gateway PE3 reaches PE4 via gateway PE1. Finally, gateways PE2 and PE4 are directly connected. Therefore, I needed to define these routes within the PE namespaces:

```
sudo ip netns exec pe2 ip route add 192.168.2.0/24 via 10.200.0.13 dev eth07
sudo ip netns exec pe2 ip route add 192.168.3.0/24 via 10.200.0.18 dev eth08
sudo ip netns exec pe3 ip route add 192.168.1.0/24 via 10.200.0.10 dev eth12
sudo ip netns exec pe3 ip route add 192.168.3.0/24 via 10.200.0.1 dev eth11
sudo ip netns exec pe4 ip route add 192.168.1.0/24 via 10.200.0.17 dev eth16
sudo ip netns exec pe4 ip route add 192.168.2.0/24 via 10.200.0.5 dev eth15
```

Even though the PE namespaces are the endpoints of the VPN tunnels, which use the public IP addresses of the gateaway interfaces in the outer header, the tunnel is not established if the kernel cannot determine the destination of the packet with private IP addresses in the header.
Next, i could proceed with the actual IPSec configuration.
For each PE namespace, four Security Associations (SAs) are required to securely connect with the other two offices. Specifically, for each link, two SAs are needed, one for each tunnel direction, since SAs are unidirectional.
Let's consider one link at a time, assuming that confidentiality, data authentication, and integrity are required by the company's security policies, starting with the tunnel between PE2 and PE3. With the following commands, i Set up IPsec ESP tunnel mode states for namespaces PE2 and PE3:

```
sudo ip netns exec pe2 ip xfrm state add src 10.200.0.14 dst 10.200.0.9 proto esp
     spi 0x1000 mode tunnel \
    enc 'aes' 0x72d3b1535a04bb0a0a0d9c7ce4f40b39 \
    auth hmac\(sha256\)
0x9bdacd6c4223355ed46684ee345439ce3c98b499c4fcd716c2b737b6b62f3ecf

sudo ip netns exec pe2 ip xfrm state add src 10.200.0.9 dst 10.200.0.14 proto esp
     spi 0x2000 mode tunnel \
    enc 'aes' 0x4255637618d950ff217825366ae4d43d \
    auth hmac\(sha256\)
0x7c1b3bef2d0ff2c232760f320abbda941067c87de4ecbb4377ed47ba844efb5a

sudo ip netns exec pe3 ip xfrm state add src 10.200.0.14 dst 10.200.0.9 proto esp
     spi 0x1000 mode tunnel \
    enc 'aes' 0x72d3b1535a04bb0a0a0d9c7ce4f40b39 \
    auth hmac\(sha256\)
0x9bdacd6c4223355ed46684ee345439ce3c98b499c4fcd716c2b737b6b62f3ecf

sudo ip netns exec pe3 ip xfrm state add src 10.200.0.9 dst 10.200.0.14 proto esp
     spi 0x2000 mode tunnel \
  enc 'aes' 0x4255637618d950ff217825366ae4d43d \
  auth hmac\(sha256\)
0x7c1b3bef2d0ff2c232760f320abbda941067c87de4ecbb4377ed47ba844efb5a
```

Here, a brief explanation of the different command components:

- **src 10.200.0.14 dst 10.200.0.9**: This indicates the source and destination IP addresses of the traffic that needs to be protected by this Security Association (SA). Since encapsulation is involved, it refers to the outer IP header.
- **aes**: This defines the encryption algorithm for this SA, along with a 32-character hexadecimal string representing the key, which is 16 bytes long.

- **hmac-sha256**: This outlines the authentication algorithm used for this SA (in this case, HMAC with SHA-256). It ensures data authentication and integrity using hash-based message authentication codes, followed by a 64-character hexadecimal string that represents the key (32 bytes).
- **mode tunnel**: This identifies the IPsec mode being used. In this case, it's tunnel mode, meaning the original IP packet is entirely encapsulated within a new IP packet.
- **spi 0x1000, spi 0x2000**: This specifies the Security Parameters Index (SPI) for the SA, which is used, along with the destination IP address and security protocol, to uniquely identify the SA.

Next, i created the IPsec ESP tunnel mode policies for namespaces PE2 and PE3, specifically, I needed two SPs in each namespace:

```
sudo ip netns exec pe2 ip xfrm policy add src 192.168.1.0/24 dst 192.168.2.0/24
    dir out \
    tmpl src 10.200.0.14 dst 10.200.0.9 proto esp mode tunnel

sudo ip netns exec pe2 ip xfrm policy add src 192.168.2.0/24 dst 192.168.1.0/24
    dir fwd \
    tmpl src 10.200.0.9 dst 10.200.0.14 proto esp mode tunnel

sudo ip netns exec pe3 ip xfrm policy add src 192.168.2.0/24 dst 192.168.1.0/24
    dir out \
    tmpl src 10.200.0.9 dst 10.200.0.14 proto esp mode tunnel

sudo ip netns exec pe3 ip xfrm policy add src 192.168.1.0/24 dst 192.168.2.0/24
    dir fwd \
    tmpl src 10.200.0.14 dst 10.200.0.9 proto esp mode tunnel
```

Here, a brief explanation of the different command components:

- **src 192.168.1.0/24 dst 192.168.2.0/24**: This defines the range of source and destination IP addresses for which the security policy (SP) applies.
- **tmpl src 10.200.0.14 dst 10.200.0.9 proto esp mode tunnel**: This template specifies the source and destination IP addresses, the protocol (ESP in this case), and the mode (tunnel). It defines the characteristics of the Security Association (SA) used to secure traffic between the specified IP address ranges.
- **dir fwd, dir out**: 'dir fwd' refers to the forwarding direction, meaning the policy applies to packets being forwarded by the router, not generated by or destined for the local machine. 'dir out' refers to the outgoing direction, meaning the policy applies to packets originating from the machine. To properly configure policies in a network, if a policy has 'out' in one device, the same policy with 'fwd'/'in' must be configured in the receiving host to ensure traffic is handled correctly.

Now, let's move on to the tunnel link between PE2 and PE4. As before, I used ip xfrm to define the SAs:

```
sudo ip netns exec pe2 ip xfrm state add src 10.200.0.17 dst 10.200.0.18 proto
    esp spi 0x5000 mode tunnel \
   enc 'aes' 0xeaeafd578e4308c057c79768c43e4a29 \
   auth hmac\(sha256\)
0xdbe07dced4d7e7808f88bc3975be46c25f8b2ad857bd6920c785bf722c5efdee

sudo ip netns exec pe2 ip xfrm state add src 10.200.0.18 dst 10.200.0.17 proto
    esp spi 0x6000 mode tunnel \
   enc 'aes' 0xe0a108253dee8e457df8969e61cb43ad \
   auth hmac\(sha256\)
0x04b35aa360f450fed7f9402de8661941624886f0ead49afa0000f5566f4b3295

sudo ip netns exec pe4 ip xfrm state add src 10.200.0.17 dst 10.200.0.18 proto
    esp spi 0x5000 mode tunnel \
     enc 'aes' 0xeaeafd578e4308c057c79768c43e4a29 \
```

```
     auth hmac\(sha256\)
0xdbe07dced4d7e7808f88bc3975be46c25f8b2ad857bd6920c785bf722c5efdee

sudo ip netns exec pe4 ip xfrm state add src 10.200.0.18 dst 10.200.0.17 proto
    esp spi 0x6000 mode tunnel \
     enc 'aes' 0xe0a108253dee8e457df8969e61cb43ad \
     auth hmac\(sha256\)
0x04b35aa360f450fed7f9402de8661941624886f0ead49afa0000f5566f4b3295
```

Next, proceed with the configuration of the Security Policies:

```
sudo ip netns exec pe2 ip xfrm policy add src 192.168.1.0/24 dst 192.168.3.0/24
    dir out \
    tmpl src 10.200.0.17 dst 10.200.0.18 proto esp mode tunnel

sudo ip netns exec pe2 ip xfrm policy add src 192.168.3.0/24 dst 192.168.1.0/24
    dir fwd \
    tmpl src 10.200.0.18 dst 10.200.0.17 proto esp mode tunnel

sudo ip netns exec pe4 ip xfrm policy add src 192.168.3.0/24 dst 192.168.1.0/24
    dir out \
    tmpl src 10.200.0.18 dst 10.200.0.17 proto esp mode tunnel

sudo ip netns exec pe4 ip xfrm policy add src 192.168.1.0/24 dst 192.168.3.0/24
    dir fwd \
    tmpl src 10.200.0.17 dst 10.200.0.18 proto esp mode tunnel
```

Lastly, I configured the Site-to-Site VPN between the PE3 and PE4 gateways, once again defining IPsec ESP tunnel mode states and policies:

```
# Set up IPsec ESP tunnel mode states for namespace PE3 and PE4
sudo ip netns exec pe3 ip xfrm state add src 10.200.0.2 dst 10.200.0.6 proto esp
    spi 0x3000 mode tunnel \
  enc 'aes' 0x0462f50849ad5b94e1c793c60fac5e62 \
  auth hmac\(sha256\)
0xcf33038c657c25befa554419b35b98b33c6dee4844759c9745e9c13690c4ff2b

sudo ip netns exec pe3 ip xfrm state add src 10.200.0.6 dst 10.200.0.2 proto esp
    spi 0x4000 mode tunnel \
  enc 'aes' 0x63325bc69038456b07c688eb3844866a \
  auth hmac\(sha256\)
0x322ff07641e210fb2d9051534954e740834021fa62ca68c3641100cf87e506bf

sudo ip netns exec pe4 ip xfrm state add src 10.200.0.2 dst 10.200.0.6 proto esp
    spi 0x3000 mode tunnel \
   enc 'aes' 0x0462f50849ad5b94e1c793c60fac5e62 \
   auth hmac\(sha256\)
0xcf33038c657c25befa554419b35b98b33c6dee4844759c9745e9c13690c4ff2b

sudo ip netns exec pe4 ip xfrm state add src 10.200.0.6 dst 10.200.0.2 proto esp
    spi 0x4000 mode tunnel \
   enc 'aes' 0x63325bc69038456b07c688eb3844866a \
   auth hmac\(sha256\)
0x322ff07641e210fb2d9051534954e740834021fa62ca68c3641100cf87e506bf

# Set up IPsec ESP tunnel mode policies for namespace PE3 and PE4
sudo ip netns exec pe3 ip xfrm policy add src 192.168.2.0/24 dst 192.168.3.0/24
    dir out \
    tmpl src 10.200.0.2 dst 10.200.0.6 proto esp mode tunnel

sudo ip netns exec pe3 ip xfrm policy add src 192.168.3.0/24 dst 192.168.2.0/24
    dir fwd \
    tmpl src 10.200.0.6 dst 10.200.0.2 proto esp mode tunnel

sudo ip netns exec pe4 ip xfrm policy add src 192.168.3.0/24 dst 192.168.2.0/24
    dir out \
    tmpl src 10.200.0.6 dst 10.200.0.2 proto esp mode tunnel

sudo ip netns exec pe4 ip xfrm policy add src 192.168.2.0/24 dst 192.168.3.0/24
    dir fwd \
    tmpl src 10.200.0.2 dst 10.200.0.6 proto esp mode tunnel
```

As previously discussed, the primary issue with site-to-site VPNs is the lack of end-to-end security against internal network attacks. Specifically, traffic between corporate hosts is only protected while it is transmitted through the tunnel between the gateways. An attacker inside the corporate network can perform various attacks, such as sniffing, spoofing, replay, and cancellation attacks.
To address this vulnerability, I have added a second layer of defense by defining transport mode Security Associations (SAs) between the corporate endpoints. This approach ensures that traffic remains authenticated within the internal network, providing an additional layer of protection against potential internal threats.
To minimize complexity, I decided to focus on protecting only the traffic to and from the CE4 namespace, which I assumed contains sensitive data. This additional protection is achieved using the data origin authentication and integrity provided by the AH protocol.
Another possible combination of SAs could have been using AH for integrity and ESP for confidentiality. However, in my case, I chose to retain the authentication provided by ESP. This choice allows the gateway to compute and compare the Integrity Check Value (ICV) of packets, enabling rapid detection and rejection of altered packets at the gateway. This approach helps avoid the decryption of these packets, potentially reducing the impact of denial-of-service attacks. Although this method introduces additional computational overhead on the gateway, it results in a more efficient and secure network.
Since gateways are often equipped with specialized hardware for cryptographic algorithms, it is feasible to use more robust algorithms between the gateways. Meanwhile, for the Security Associations (SAs) between the two hosts, which may have less computational power, faster and less resource-intensive algorithms can be employed to strike a balance between performance and security.
With the following commands, i Set up IPsec AH transport mode states and policies between namespaces CE2 and CE4:

```
# Set up IPsec AH transport mode states for namespace CE2 and CE4
sudo ip netns exec ce2 ip xfrm state add src 192.168.1.2 dst 192.168.3.2 proto ah
     spi 0x1000 auth hmac\(sha256\)
0x0d6aac1735278fe8a709716271d0a5117c89cef8e5302d1cb7b39bbf6ac35e22

sudo ip netns exec ce2 ip xfrm state add src 192.168.3.2 dst 192.168.1.2 proto ah
     spi 0x2000 auth hmac\(sha256\)
0x280730d699e985ab62f98d508a001a70cd2ff28db4dd871e890612bd03ad1917

sudo ip netns exec ce4 ip xfrm state add src 192.168.1.2 dst 192.168.3.2 proto ah
     spi 0x1000 auth hmac\(sha256\)
0x0d6aac1735278fe8a709716271d0a5117c89cef8e5302d1cb7b39bbf6ac35e22

sudo ip netns exec ce4 ip xfrm state add src 192.168.3.2 dst 192.168.1.2 proto ah
     spi 0x2000 auth hmac\(sha256\)
0x280730d699e985ab62f98d508a001a70cd2ff28db4dd871e890612bd03ad1917

# Set up IPsec AH transport mode policies for namespace CE2 and CE4
sudo ip netns exec ce2 ip xfrm policy add dir out src 192.168.1.2 dst 192.168.3.2
      tmpl src 192.168.1.2 dst 192.168.3.2 proto ah spi 0x1000

sudo ip netns exec ce2 ip xfrm policy add dir in src 192.168.3.2 dst 192.168.1.2
     tmpl src 192.168.3.2 dst 192.168.1.2 proto ah spi 0x2000

sudo ip netns exec ce4 ip xfrm policy add dir in src 192.168.1.2 dst 192.168.3.2
     tmpl src 192.168.1.2 dst 192.168.3.2 proto ah spi 0x1000

sudo ip netns exec ce4 ip xfrm policy add dir out src 192.168.3.2 dst 192.168.1.2
      tmpl src 192.168.3.2 dst 192.168.1.2 proto ah spi 0x2000
```

As can be seen from the commands, since this is End-to-End security, the source and destination IP addresses are not ranges but unique addresses. Additionally, the 'dir in' specifies that the traffic is destined for the machine itself, unlike 'fwd' as in site-to-site VPNs.
Lastly, I protected the traffic between CE3 and CE4 by defining IPsec AH transport mode states and policies again:

```
# Set up IPsec AH transport mode states for namespace CE3 and CE4
```

```
sudo ip netns exec ce3 ip xfrm state add src 192.168.2.2 dst 192.168.3.2 proto ah
    spi 0x3000 auth hmac\(sha256\)
0x831de8fbad2a8306c1d835f946565d8fd168bdfba2092055afe245034ec06b52

sudo ip netns exec ce3 ip xfrm state add src 192.168.3.2 dst 192.168.2.2 proto ah
    spi 0x4000 auth hmac\(sha256\)
0x3f797aec8e7684ddd7b6074da347b351fcebe273e3be32b32ba845eb36138a3b

sudo ip netns exec ce4 ip xfrm state add src 192.168.2.2 dst 192.168.3.2 proto ah
    spi 0x3000 auth hmac\(sha256\)
0x831de8fbad2a8306c1d835f946565d8fd168bdfba2092055afe245034ec06b52

sudo ip netns exec ce4 ip xfrm state add src 192.168.3.2 dst 192.168.2.2 proto ah
    spi 0x4000 auth hmac\(sha256\)
0x3f797aec8e7684ddd7b6074da347b351fcebe273e3be32b32ba845eb36138a3b

# Set up IPsec AH transport mode policies for namespace CE3 and CE4
sudo ip netns exec ce3 ip xfrm policy add dir out src 192.168.2.2 dst 192.168.3.2
     tmpl src 192.168.2.2 dst 192.168.3.2 proto ah spi 0x3000

sudo ip netns exec ce3 ip xfrm policy add dir in src 192.168.3.2 dst 192.168.2.2
    tmpl src 192.168.3.2 dst 192.168.2.2 proto ah spi 0x4000

sudo ip netns exec ce4 ip xfrm policy add dir in src 192.168.2.2 dst 192.168.3.2
    tmpl src 192.168.2.2 dst 192.168.3.2 proto ah spi 0x3000

sudo ip netns exec ce4 ip xfrm policy add dir out src 192.168.3.2 dst 192.168.2.2
     tmpl src 192.168.3.2 dst 192.168.2.2 proto ah spi 0x4000
```

4.3 Testing and Verification

I then verified the correct functionality of the configured IPsec tunnel (with ESP) and transport (with AH) SAs by having the office endpoints ping each other. To capture the packets, I monitored the virtual interfaces of the Linux bridge — veth05, veth09, veth13, veth07, veth08, and veth11 — using Wireshark, with a separate instance of the packet sniffer running on each interface (Figure 2 provides a summary of the interface information).
Starting with the namespace CE2 initiating ICMP pings between the offices 2 and 3:

```
sudo ip netns exec ce2 ping 192.168.2.2 -c 5
```

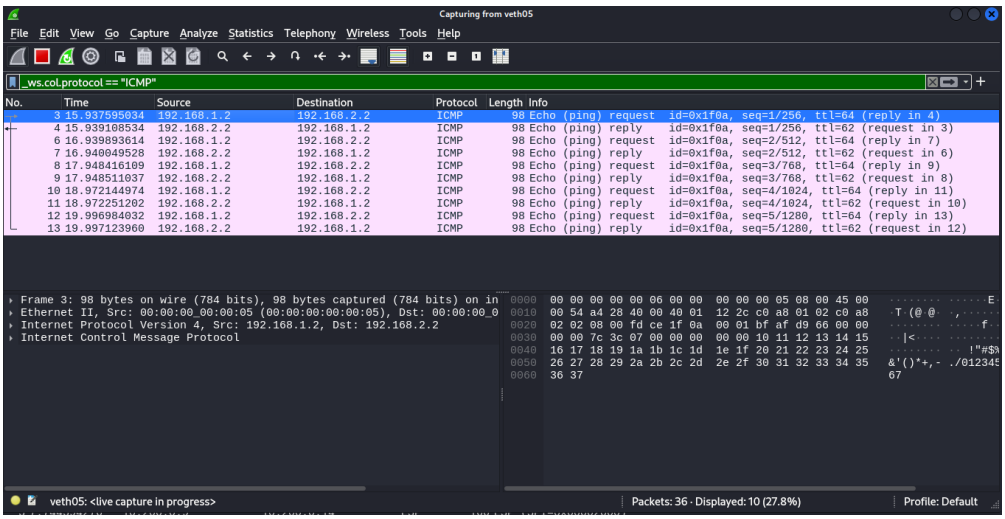

Figure 5: Capturing on veth05 - Namespace CE2

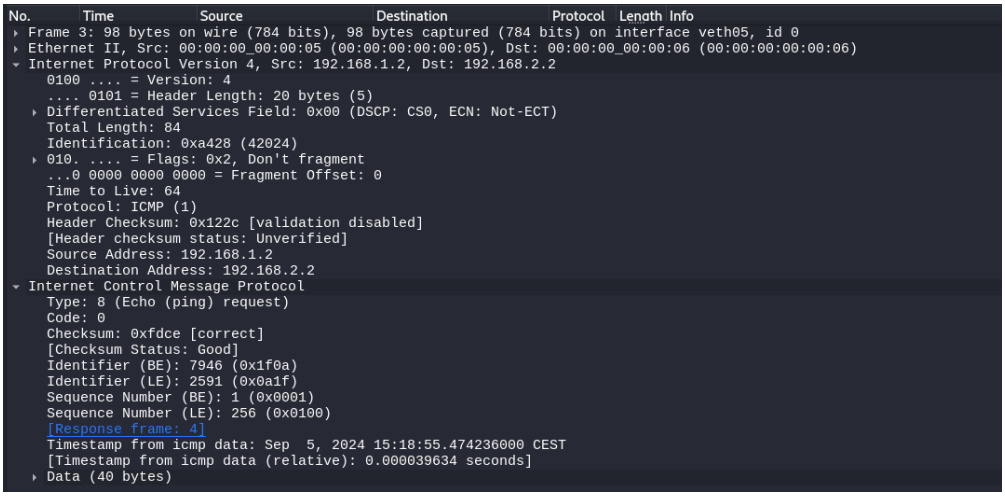

Figure 6: Capturing on veth05 - Namespace CE2

Figures 5 and 6 show the Wireshark capture of an ICMP Echo Request on the bridge interface veth05 (which connects to the eth05 interface in the CE2 namespace) during a ping from CE2 to CE3. Key observations from the captured packet are as follows:

- The Ethernet header shows the source MAC address as 00:00:00:00:00:05 and the destination MAC address as 00:00:00:00:00:06. The destination MAC corresponds to the interface of the VPN gateway (namespace PE2), located within the same IP network, as the other corporate networks are not directly connected.
- The source IP address is 192.168.1.2, associated with the eth05 interface in the CE2 namespace, and the destination IP address is 192.168.2.2, associated with the eth09 interface in the CE3 namespace.
- As expected, the packet is unencrypted and in plaintext until it reaches the IPsec tunnel for protection.

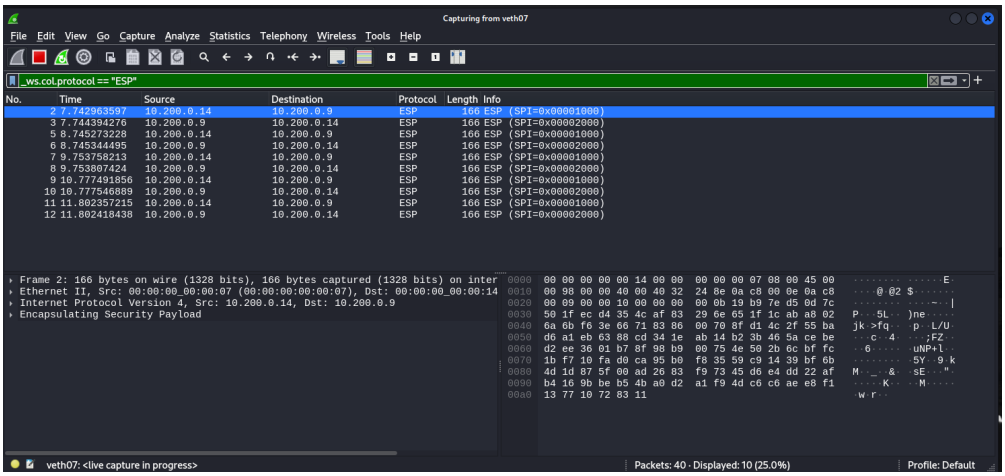

Figure 7: Capturing on veth07 - Namespace PE2

When the ICMP Echo Request arrives at the VPN gateway (namespace PE2), the packet undergoes the IPsec tunnel processing. The system performs a lookup in the Security Policy Database (SPD) to determine if the packet requires protection. Upon finding a match, the Security Association Database (SAD) is consulted to retrieve the correct Security Association (SA) parameters.

The original IP packet is encapsulated into the ESP Payload field and then encrypted using the specified parameters. After encryption, an integrity algorithm is applied. This computes the Integrity Check Value (ICV) over the ESP packet minus the ICV field. The resulting packet, which is encrypted and integrity-protected, is then encapsulated in a new IP header and sent through the tunnel.
Figure 7 displays a Wireshark capture of ESP packets observed on the bridge interface veth07 (which connects to the eth07 interface in the PE2 namespace). Key observations from the captured packet include:

- The Ethernet header reveals a source MAC address of 00:00:00:00:00:07 and a destination MAC address of 00:00:00:00:00:14, that is the MAC address of the next hop, interface eth13 in the namespace PE5.
- The packet contains an IP header with a source IP address of 10.200.0.14 and a destination IP address of 10.200.0.9, corresponding to the eth07 interface in the PE2 namespace and the eth12 interface in the PE3 namespace, respectively. These IP addresses are used for routing across a larger shared network infrastructure rather than the private IP addresses of the endpoints.
- Following the IP header, the ESP header includes the Security Parameters Index (SPI) and a sequence number. While the ESP payload and additional IP headers are present, they are encrypted, preventing Wireshark from displaying their contents without proper decryption keys.

Even though the payload of an ESP packet is encrypted, it is possible to configure Wireshark to decrypt the packet and reveal its contents.
Navigate to Edit → Preferences → Protocols → ESP → Edit. After entering the correct encryption and authentication keys, and specifying the relevant Security Parameters Index (SPI) for the IPsec connection, I configured the menu as follows:

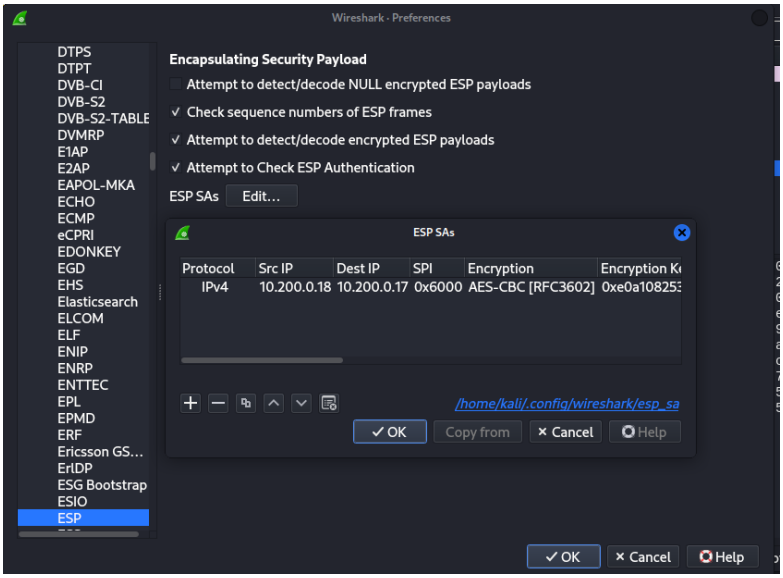

Figure 8: How to decrypt IPSec Packets - Wireshark

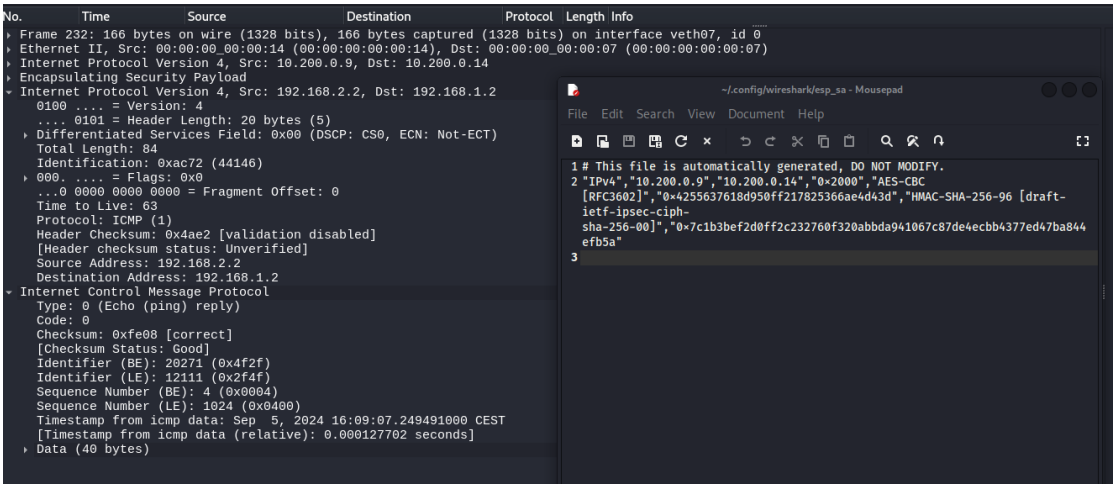

Figure 9: Capturing on veth07 - Decrypted packet

Once these details were properly configured (as it can be seen in the right in Figure 9), Wireshark successfully decrypted the captured ESP packets. As shown in Figure 9, the inner IP header displayed the source and destination IP addresses of the office endpoints, allowing me also to confirm that the packet was an ICMP Echo Reply.

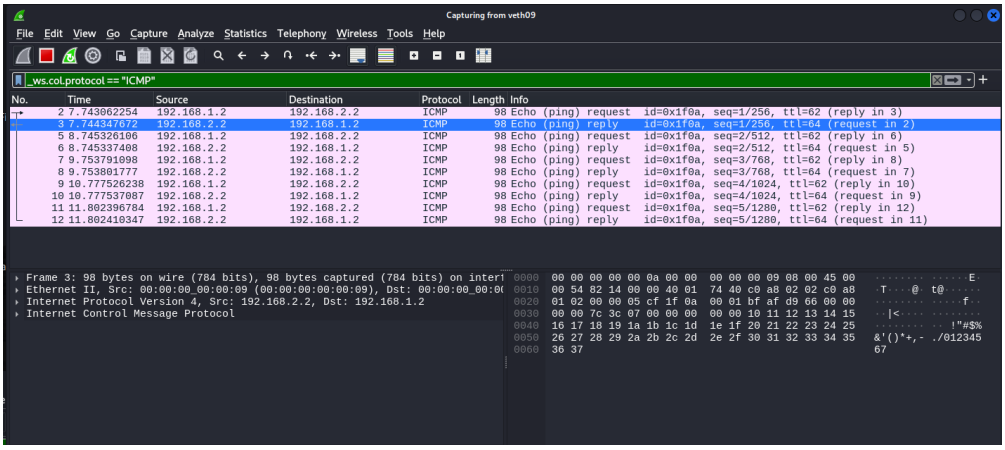

Figure 10: Capturing on veth09 - Namespace CE3

After the tunneled ICMP Echo Request reaches the eth12 interface of PE3, the namespace removes the encapsulation and forwards the decapsulated packet to CE3. Figure 10 shows the Wireshark capture of the ICMP Echo Requests and Replies on the bridge interface veth09, which connects to the eth09 interface in the CE3 namespace.
Key details from the captured packet include:

- The Ethernet header displays a source MAC address of 00:00:00:00:00:09 and a destination MAC address of 00:00:00:00:00:0a, or vice versa. Notably, 00:00:00:00:00:0a corresponds to the interface of the VPN gateway (PE3), which is within the same IP network.
- The IP header displays a source IP address of 192.168.1.2 and a destination IP address of 192.168.2.2, or vice versa.
- As expected, the packet is unencrypted and in plaintext after exiting the IPsec tunnel.

As previously explained, the traffic to and from the office endpoint CE4 is protected by a dual layer of defense. Specifically, I have established an ESP tunnel mode between the gateways and

an AH transport mode between the hosts to safeguard against certain internal network attacks. Now, I will highlight the main differences compared to the results obtained in the previous case, using the following command to have hosts CE2 and CE4 ping each other:

```
sudo ip netns exec ce4 ping 192.168.2.2 -c 5
```

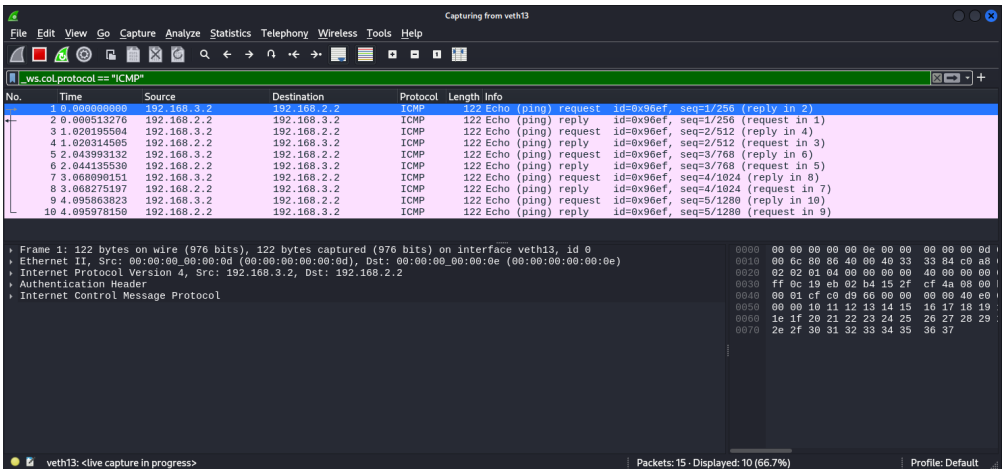

Figure 11: Capturing on veth13 - Namespace CE3

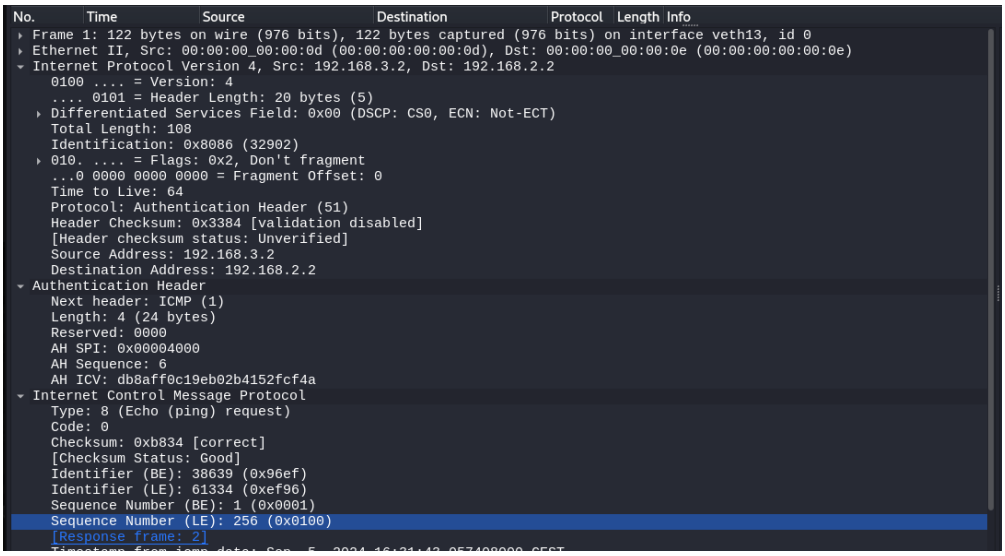

Figure 12: Capturing on veth13 - AH Capture

As shown in Figures 11 and 12, an ICMP echo request with source IP address 192.168.3.2 (eth13 interface in the CE4 namespace) and destination IP address 192.168.2.2 (eth09 interface in the CE3 namespace) includes an additional header, the Authentication Header, which contains information such as SPI, sequence number, and ICV.

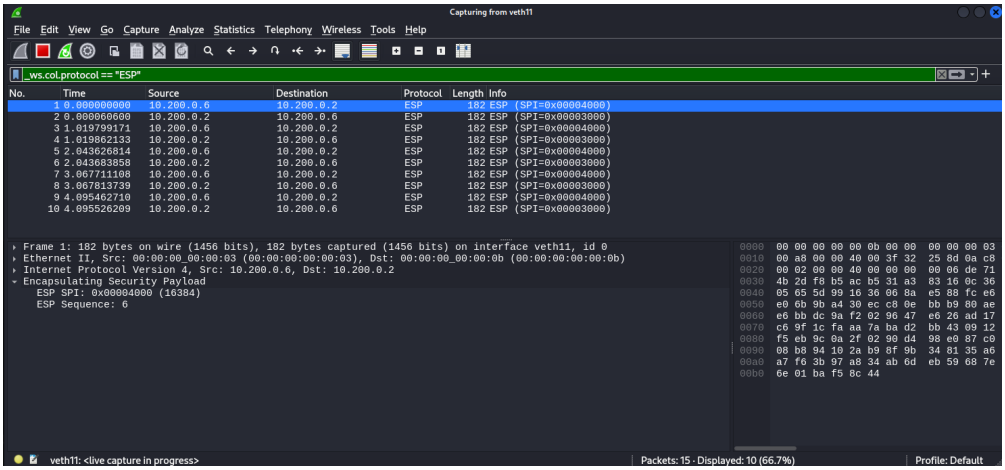

Figure 13: Capturing on veth11 - Namespace PE3

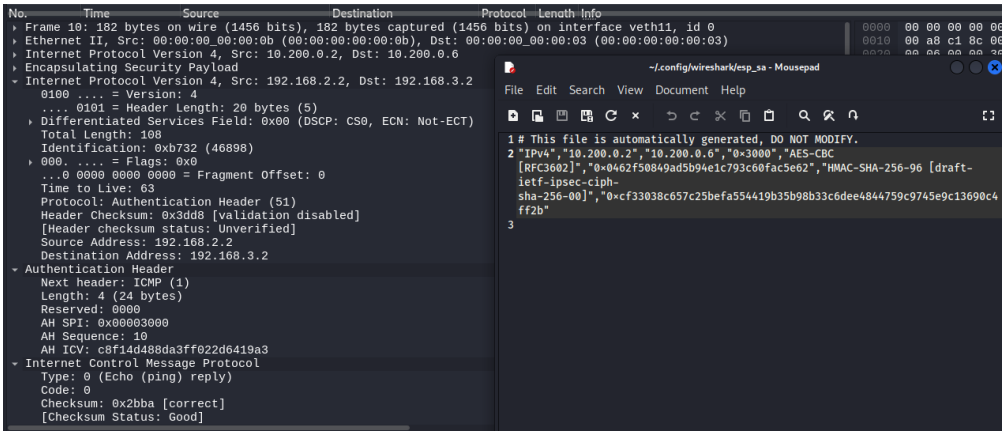

Figure 14: Capturing on veth11 - Decrypted packet

Capturing traffic on the virtual interface veth11, we can observe that the VPN tunnel is established (Figure 13), and the ESP payload is encrypted. I then configured Wireshark, as before, with the SA details for the traffic from PE3 to PE4.
Figure 14 shows all the packet headers:

- The outer IP header, which contains the source and destination IP addresses of the VPN gateway interfaces. In this case, the source IP is 10.200.0.2 (PE3), and the destination IP is 10.200.0.6 (PE4), indicating the endpoints of the tunnel between the two VPN gateways.
- The ESP header, which includes the Security Parameter Index (SPI) with a value of 0x3000 and the sequence number, which helps in maintaining the order of the encrypted packets and preventing replay attacks.
- The inner IP header, which contains the source and destination IP addresses of the office hosts. In this case, the source IP is 192.168.2.2 (CE3), and the destination IP is 192.168.3.2 (CE4).
- The Authentication Header, which also has an SPI value of 0x3000, and includes the ICV.
- The ICMP header, specifically for an ICMP echo reply message, confirming that the original ping (echo request) has been received and processed correctly.

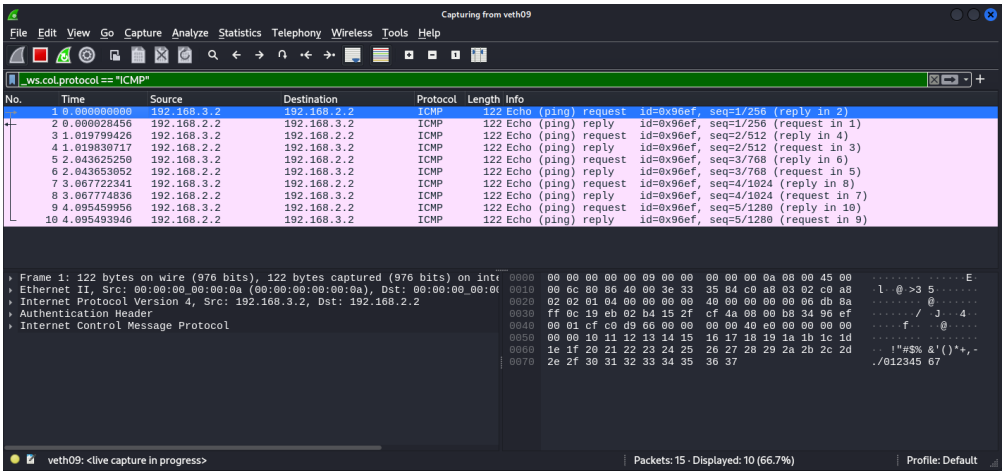

Figure 15: Capturing on veth09 - Namespace CE4

To complete the testing and verification of the site-to-site VPN, I provide the images of the ping between offices 2 and 4, whose results are similar to the previous case:

```
sudo ip netns exec ce2 ping 192.168.3.2 -c 5
```

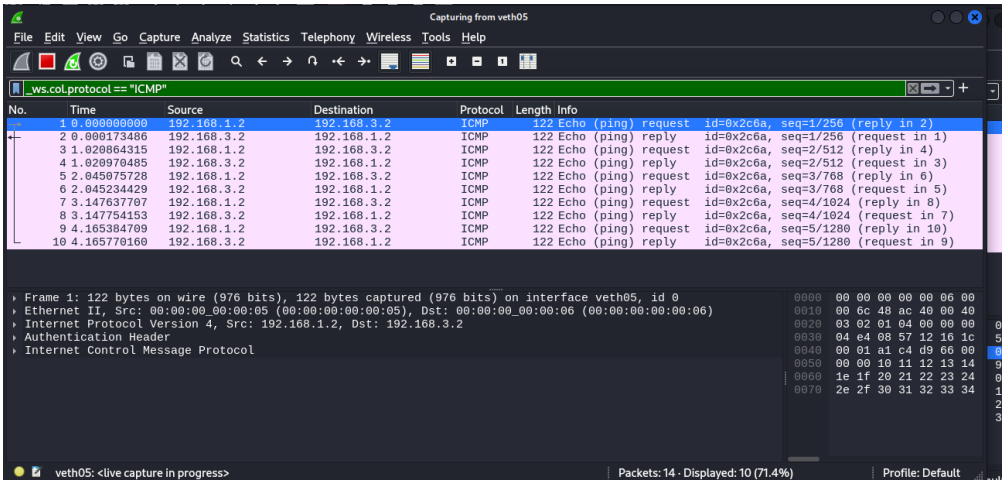

Figure 16: Capturing on veth05 - Namespace CE2

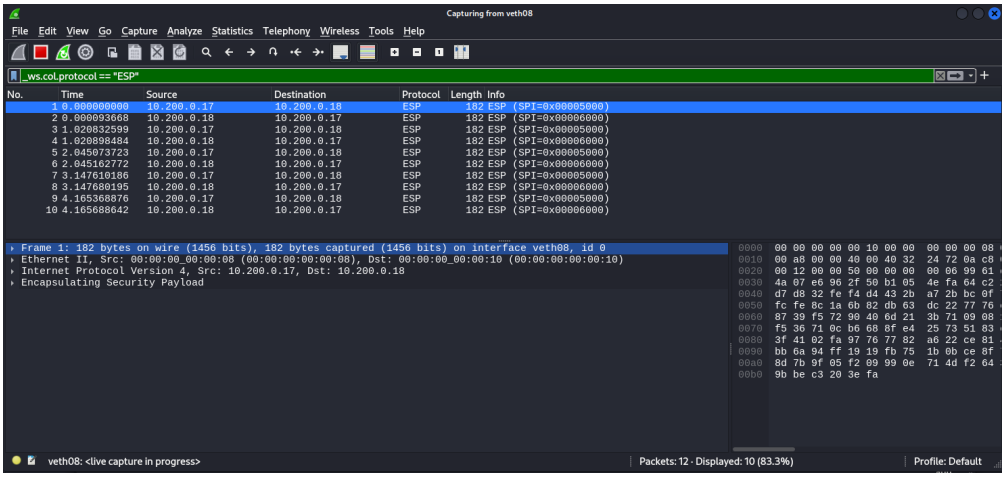

Figure 17: Capturing on veth08 - Namespace PE2

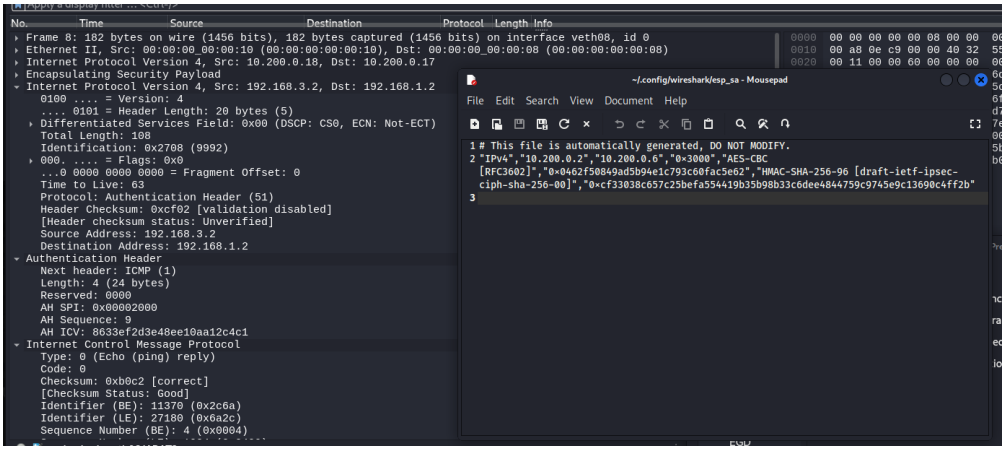

Figure 18: Capturing on veth08 - Decrypted packet

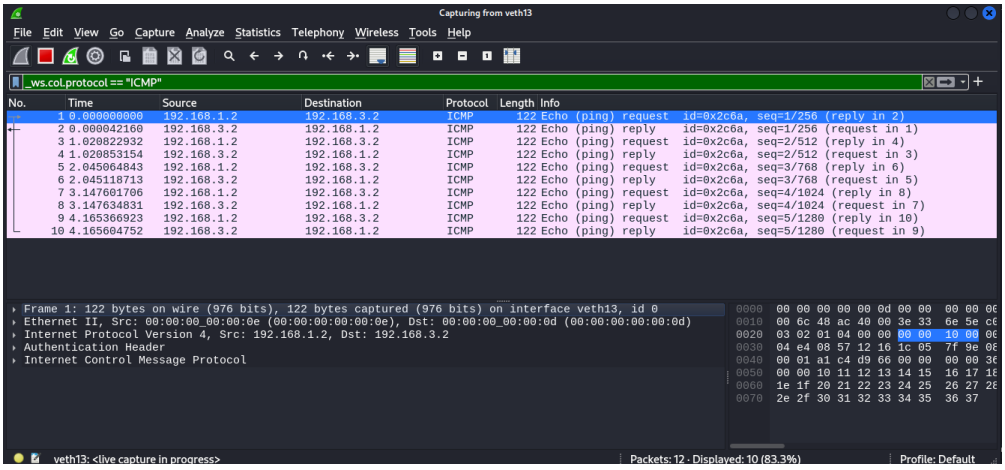

Figure 19: Capturing on veth13 - Namespace CE4

5 End-to-End VPN

5.1 Understanding End-to-End VPN

What is a End-to-End VPN?
An End-to-End (e2e) VPN is particularly useful for securing communications between two specific devices or applications, ensuring that even intermediate devices or networks cannot intercept or tamper with the data.
This type of VPN is often implemented in transport mode because this mode only encrypts the payload of the IP packets, leaving the original IP headers intact, allowing routers and intermediate devices to handle the packet without needing to decrypt it.

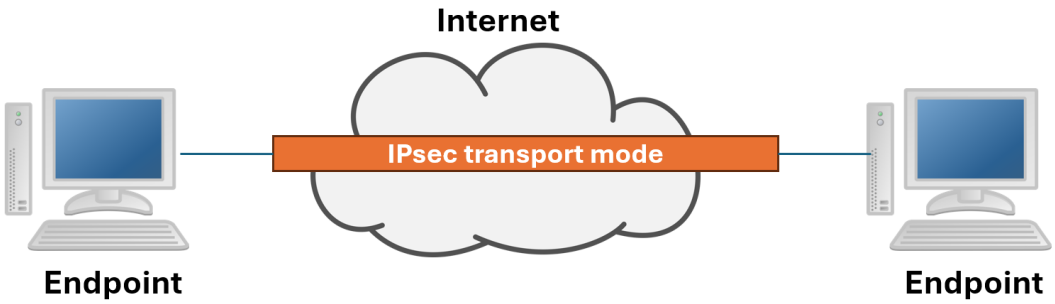

Figure 20: End-to-End VPN

Advantages of End-to-End VPN

- **Endpoint Authentication**: Ensures that the devices at both ends of the VPN connection are verified, enhancing security.
- **Protection Against Untrusted Networks**: Prevents traffic from being intercepted, even within the company LAN.

Disadvantages of End-to-End VPN

- **Potential Compatibility Issues**: Challenges may arise related to the availability of the IPsec software module and the necessary computational hardware power.
- **Configuration Complexity**: Setting up and managing end-to-end VPNs for each host pair can be intricate and time-consuming.

5.2 Step-by-Step Configuration on Linux

To establish an end-to-end VPN in ESP transport mode between hosts CE1 and CE5, I started by assigning public IP addresses as aliases to the network interfaces in the namespaces CE1 (eth01) and CE5 (eth17). These public IPs acted as secondary addresses, meaning that both interfaces managed traffic for the public and private IPs assigned to them.
Choosing to use public IPs was crucial because integrating Network Address Translation (NAT) with IPsec can present several challenges, primarily due to IPsec's security mechanisms. IPsec depends on preserving the integrity of packet headers and payloads, which can be compromised by NAT's alterations to IP addresses.
Even though ESP does not extend its protection to the IP headers, it is not designed to handle NAT address modifications without NAT Traversal (NAT-T). NAT-T is a widely accepted standard that addresses NAT-related issues with IPsec by encapsulating ESP packets within UDP packets. This approach ensures compatibility across various network setups and NAT devices,

and also resolves issues related to Port Address Translation.
However, implementing a solution like NAT-T with ip xfrm would be complex, by using public IP addresses directly, I simplified the setup and avoided potential issues with NAT and IPsec interoperability. Therefore, I configured the network as follows:

```
sudo ip netns exec ce1 ip  addr add 10.200.0.25/32 dev eth01
sudo ip netns exec ce5 ip  addr add 10.200.0.26/32 dev eth17
```

First, I added default routes to direct traffic between CE1 and CE5, ensuring that the public IPs were used as the source addresses:

```
sudo ip netns exec ce1 ip route add default via 172.16.0.1
        dev eth01 src 10.200.0.25
sudo ip netns exec ce5 ip route add default via 172.16.1.1
        dev eth17 src 10.200.0.26
```

Next, to make sure the gateways in the PE1 and PE5 namespaces could reach the public host IPs, I added static routes through their interfaces in the private networks 172.16.x.0/24 (as shown in Figure 1):

```
sudo ip netns exec pe1 ip route add 10.200.0.25/32 dev eth02
sudo ip netns exec pe5 ip route add 10.200.0.26/32 dev eth18
```

I also configured static routes on the gateways PE1, PE5, and PE3 to route the end-to-end VPN traffic via PE3. These routes ensured that traffic between hosts CE1 and CE5 flowed correctly through the network:

```
sudo ip netns exec pe1 ip route add 10.200.0.26/32 via 10.200.0.2 dev eth03
sudo ip netns exec pe5 ip route add 10.200.0.25/32 via 10.200.0.9 dev eth19
sudo ip netns exec pe3 ip route add 10.200.0.26/32 via 10.200.0.10 dev eth12
sudo ip netns exec pe3 ip route add 10.200.0.25/32 via 10.200.0.1 dev eth11
```

Once the routing was set up, I configured the Security Associations to use ESP in transport mode. I then added the necessary policies to define how packets should be processed. This configuration ensured that the packets exchanged between CE1 and CE5 were encrypted using AES in CBC mode, while their integrity was protected by HMAC with SHA-256:

```
# Set up IPsec ESP transport mode states and policies for namespace CE1
sudo ip netns exec ce1 ip xfrm state add src 10.200.0.25 dst 10.200.0.26 proto
    esp spi 0x1000 \
  enc cbc\(aes\) 0x3c75754276e73d0131551e367bfdcef8 \
  auth hmac\(sha256\)
0xfa74f3627eacf2ba06155c9e0e99073099178ec042e099a1970a98c840f46e2b

sudo ip netns exec ce1 ip xfrm state add src 10.200.0.26 dst 10.200.0.25 proto
    esp spi 0x2000 \
  enc cbc\(aes\) 0x3a7b9c2f5e8d1a64b7f9e0c2a3d5f8e1 \
  auth hmac\(sha256\)
0xd0f4c86a89d2cf02ccd3ff473204407030c0dc8804fba5d522617b56f00aa24f

sudo ip netns exec ce1 ip xfrm policy add dir out src 10.200.0.25 dst 10.200.0.26
     tmpl src 10.200.0.25 dst 10.200.0.26 proto esp spi 0x1000

sudo ip netns exec ce1 ip xfrm policy add dir in src 10.200.0.26 dst 10.200.0.25
    tmpl src 10.200.0.26 dst 10.200.0.25 proto esp spi 0x2000

# Set up IPsec ESP transport mode states and policies for namespace CE5
sudo ip netns exec ce5 ip xfrm state add src 10.200.0.25 dst 10.200.0.26 proto
    esp spi 0x1000 \
  enc cbc\(aes\) 0x3c75754276e73d0131551e367bfdcef8 \
  auth hmac\(sha256\)
0xfa74f3627eacf2ba06155c9e0e99073099178ec042e099a1970a98c840f46e2b

sudo ip netns exec ce5 ip xfrm state add src 10.200.0.26 dst 10.200.0.25 proto
    esp spi 0x2000 \
  enc cbc\(aes\) 0x3a7b9c2f5e8d1a64b7f9e0c2a3d5f8e1 \
  auth hmac\(sha256\)
0xd0f4c86a89d2cf02ccd3ff473204407030c0dc8804fba5d522617b56f00aa24f

```

```
sudo ip netns exec ce5 ip xfrm policy add dir in src 10.200.0.25 dst 10.200.0.26
    tmpl src 10.200.0.25 dst 10.200.0.26 proto esp spi 0x1000

sudo ip netns exec ce5 ip xfrm policy add dir out src 10.200.0.26 dst 10.200.0.25
     tmpl src 10.200.0.26 dst 10.200.0.25 proto esp spi 0x2000
```

5.3 Testing and Verification

To ensure the IPsec transport mode (using ESP) SAs were functioning correctly, I tested the connectivity between remote hosts CE1 and CE5 by initiating pings. I captured the packet data by monitoring the virtual interfaces of the Linux bridge—veth01, veth12, and veth17—using Wireshark, with a dedicated packet sniffer instance on each interface (Figure 2 provides a summary of the interface information):

```
sudo ip netns exec ce1 ping 10.200.0.26 -c 5
```

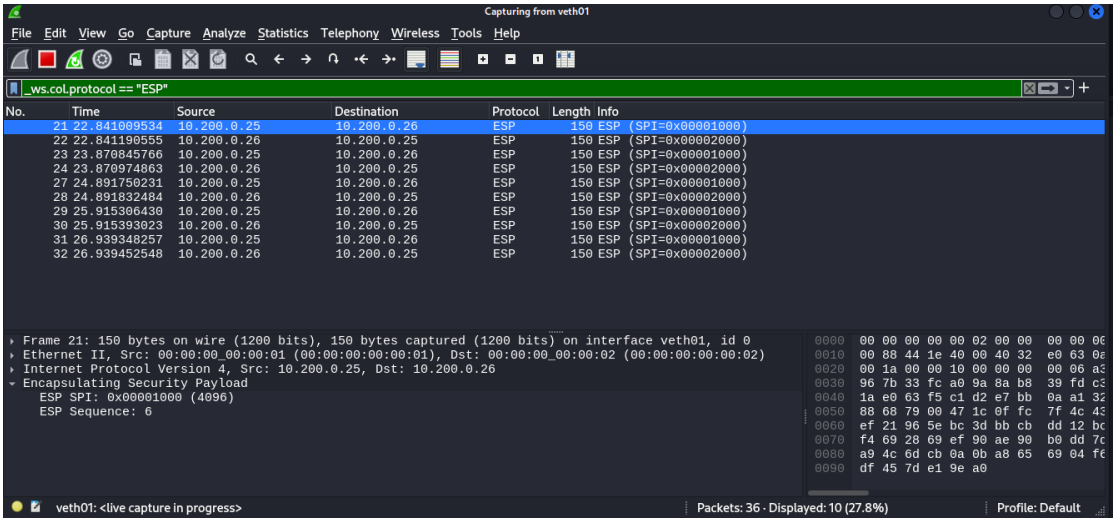

Figure 21: Capturing on veth01 - Namespace CE1

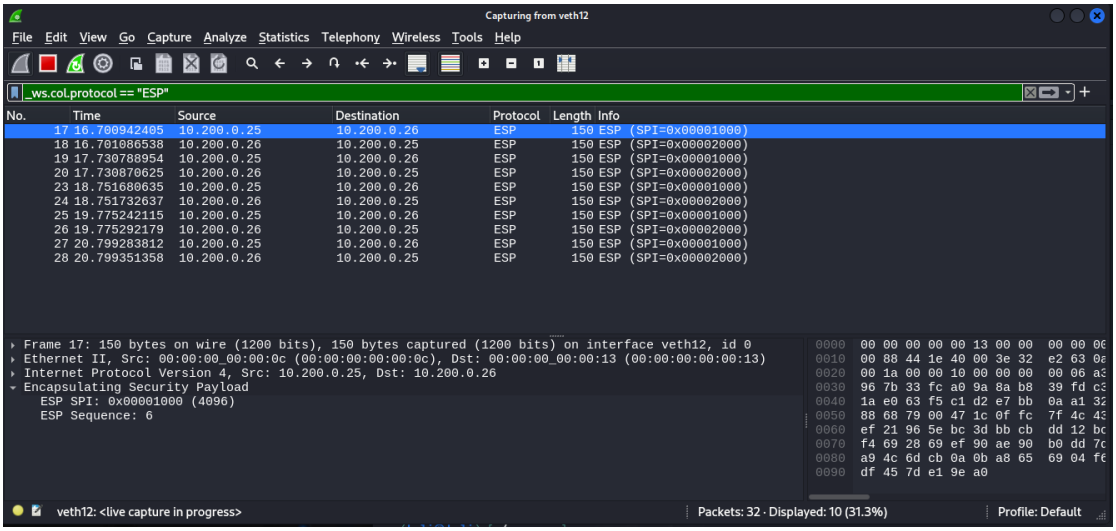

Figure 22: Capturing on veth12 - Namespace PE3

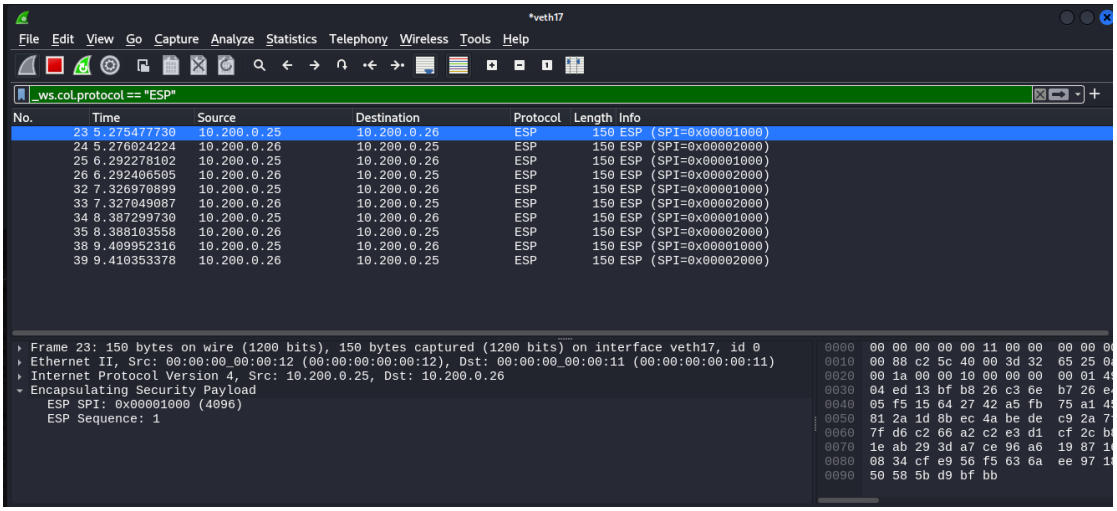

Figure 23: Capturing on veth17 - Namespace CE5

On the three monitored interfaces, the correct configuration of the end-to-end VPN can be clearly observed. An ICMP echo request originating from CE1 reaches the PE1 gateway through its eth02 network interface (with destination MAC address 00:00:00:00:00:02, Figure 21), and then, protected by the ESP protocol, it passes through the eth12 interface of the PE3 gateway, heading towards the eth19 interface of the PE5 gateway (with destination MAC address 00:00:00:00:00:13, Figure 22).
Finally, it reaches the CE5 host through the private network interface eth18 of the PE5 gateway (with source MAC address 00:00:00:00:00:12, Figure 23).
The packet arrives encrypted at the CE5 host, but since I defined specific IPsec states and policies for the traffic between the two remote endpoints using ip xfrm, CE5 is able to verify the integrity of the packet, decrypt the echo request (ESP in transport mode provides integrity and authenticity for the encrypted payload) and securely send an echo reply back to CE1.

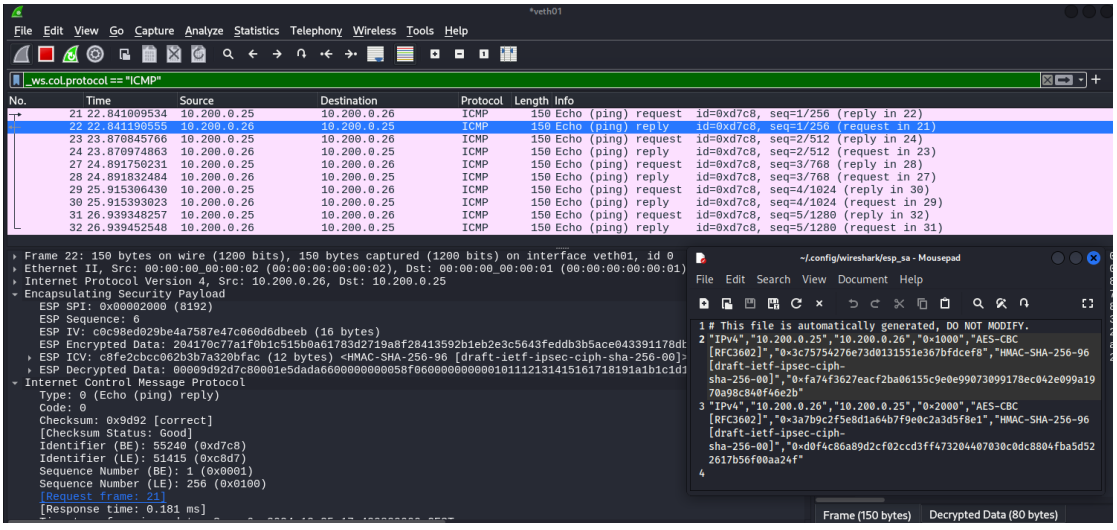

Figure 24: Capturing on veth01 - Decrypted packet

I configured Wireshark again to confirm the results, specifying the encryption and authentication algorithms used, along with their corresponding keys and the SPI. The packet sniffer instance on the virtual interface veth01 (Figure 24) shows that the ESP payload contains ICMP echo requests and replies as expected.

6 Remote Access VPN

6.1 Understanding Remote Access VPN

What is a Remote Access VPN?
A remote access VPN is a type of virtual private network that allows individual users to connect securely to a private network from a remote location.
Commonly used by employees working from home, traveling, or accessing the network from any location outside the office, it works by establishing an encrypted tunnel between the user's device and the corporate network.
The secure tunnel extends the network's perimeter to the remote user, essentially placing them within the corporate network. This process allows for safe access to internal resources like applications, file servers, and databases.

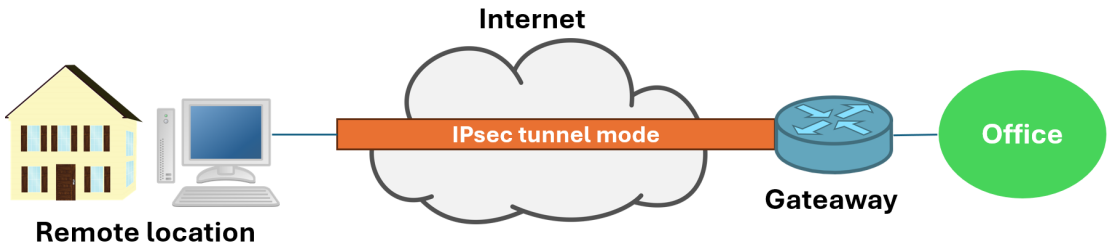

Figure 25: Remote Access VPN

Advantages of Remote Access VPN

- **User authentication**: Strong authentication methods ensure that only authorized users can establish a VPN connection.
- **Simplified Management**: Simplify network management by providing a single point of control for user access and security policies.
- **Cost-Effectiveness**: Reduced Need for Physical Infrastructure.

Disadvantages of Remote Access VPN

- **Inconsistent User Experience**: Connectivity issues and the need for manual logins can hinder productivity and frustrate remote employees.
- **Endpoint security**: If endpoint devices are compromised, they can inadvertently become conduits for malware or other cyber threats.
- **Performance Issues**: The encryption and decryption processes on the gateaway can introduce latency, potentially reducing the speed of data transmission and overall network performance.
- **Internal traffic analysis**: Data is securely transmitted only between remote host and gateaway.

6.2 Step-by-Step Configuration on Linux

A critical element of the configuration is the use of a "VPN pool", a dedicated range of IP addresses (192.168.4.0/24) that is assigned exclusively to remote users. This approach allows each remote user to obtain an IP address from this range when they connect, facilitating communication with the internal corporate network.
By using a separate IP range for the VPN pool, network administrators gain more control over

routing and traffic management. The distinct range makes it easier to identify traffic from remote users and apply routing rules specific to their connections. Additionally, this separation simplifies the implementation of security measures like firewall rules and access controls.
I assigned the IP address 192.168.4.1/32 from the VPN pool as an alias to the network interface in the CE1 namespace, and similarly assigned 192.168.4.2/32 as an alias to the network interface in the CE5 namespace:

```
sudo ip netns exec ce1 ip addr add 192.168.4.1/32 dev eth01
sudo ip netns exec ce5 ip addr add 192.168.4.2/32 dev eth17
```

I then configured routes to direct traffic between the remote hosts (CE1 and CE5) and office networks 2, 3, and 4. For both CE1 and CE5, I ensured that the source IP address for this traffic is assigned from the VPN pool and that the traffic passes through the gateway interface in the internal network:

```
sudo ip netns exec ce1 ip route add 192.168.1.0/24 via 172.16.0.1
    dev eth01 src 192.168.4.1
sudo ip netns exec ce1 ip route add 192.168.2.0/24 via 172.16.0.1
    dev eth01 src 192.168.4.1
sudo ip netns exec ce1 ip route add 192.168.3.0/24 via 172.16.0.1
    dev eth01 src 192.168.4.1

sudo ip netns exec ce5 ip route add 192.168.1.0/24 via 172.16.1.1
    dev eth17 src 192.168.4.2
sudo ip netns exec ce5 ip route add 192.168.2.0/24 via 172.16.1.1
    dev eth17 src 192.168.4.2
sudo ip netns exec ce5 ip route add 192.168.3.0/24 via 172.16.1.1
    dev eth17 src 192.168.4.2
```

Furthermore, static routes were added on the gateways (PE2, and PE4) to enable communication with the public IP addresses previously assigned to the interfaces in the CE1 and CE5 namespaces, as these addresses will be used in the outer headers of the tunnel.
Gateway PE2 reaches remote host CE1 through intermediary gateways PE4 and PE1. Conversely, gateways PE3 and PE4 connect directly to gateway PE1 to reach CE1. Finally, gateways PE2, PE3, and PE4 connect to CE5 via their direct link with gateway PE5. The PE3 namespace has been already configured for the end-to-end VPN case to reach the public IP addresses of CE1 and CE5:

```
# CE1
sudo ip netns exec pe2 ip route add 10.200.0.25/32 via 10.200.0.18 dev eth08
sudo ip netns exec pe4 ip route add 10.200.0.25/32 via 10.200.0.5 dev eth15

# CE5
sudo ip netns exec pe2 ip route add 10.200.0.26/32 via 10.200.0.14 dev eth07
sudo ip netns exec pe4 ip route add 10.200.0.26/32 via 10.200.0.21 dev eth22
```

Additional routes were configured to allow internal company hosts (CE2, CE3 and CE4) to reach the remote hosts through the office gateway:

```
# CE1
sudo ip netns exec ce2 ip route add 192.168.4.0/24 via 192.168.1.1 dev eth05
sudo ip netns exec ce3 ip route add 192.168.4.0/24 via 192.168.2.1 dev eth09
sudo ip netns exec ce4 ip route add 192.168.4.0/24 via 192.168.3.1 dev eth13

# CE5
sudo ip netns exec ce2 ip route add 192.168.4.0/24 via 192.168.1.1 dev eth05
sudo ip netns exec ce3 ip route add 192.168.4.0/24 via 192.168.2.1 dev eth09
sudo ip netns exec ce4 ip route add 192.168.4.0/24 via 192.168.3.1 dev eth13
```

Along with specific routes for the private addresses of the remote hosts on the gateways. These routes are essential due to the sequence of events for routing and applying IPsec policies in the Linux kernel, as explained earlier, even though the packets will ultimately follow a path determined by the public IP addresses:

```
# CE1
sudo ip netns exec pe2 ip route add 192.168.4.1/32 via 10.200.0.18 dev eth08
sudo ip netns exec pe3 ip route add 192.168.4.1/32 via 10.200.0.1 dev eth11
sudo ip netns exec pe4 ip route add 192.168.4.1/32 via 10.200.0.5 dev eth15
```

```
# CE5
sudo ip netns exec pe2 ip route add 192.168.4.2/32 via 10.200.0.14 dev eth07
sudo ip netns exec ce3 ip route add 192.168.4.0/24 via 192.168.2.1 dev eth09
sudo ip netns exec pe4 ip route add 192.168.4.2/32 via 10.200.0.21 dev eth22
```

These commands set up IPsec ESP tunnels between CE1 and the gateways PE2, PE3, and PE4, using AES for encryption and HMAC-SHA256 for integrity. As mentioned previously, the tunnels will have the public IP address 10.200.0.25 in the outer IP header, along with the IP address of the involved VPN gateway interface.

The policies match traffic between the remote host with IP 192.168.4.1/32 and the office networks (192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24). It is worth noting the different values of the dir parameter in the policies, which regulate traffic directed towards either the host or the gateway (e.g., SPI 0x8000 in CE1 namespace and SPI 0x7000 in PE2 namespace). For traffic to the remote hosts, the dir parameter is set to in, whereas for traffic to the gateways, which act as intermediate nodes, the dir parameter is set to out (Figure 1 provides the network topology):

```
# Set up IPsec ESP tunnel mode states and policies between namespaces CE1 and PE2

sudo ip netns exec ce1 ip xfrm state add src 10.200.0.25 dst 10.200.0.17 proto
    esp spi 0x7000 mode tunnel \
    enc 'aes' 0xf06eac104ee52865463f324070717269 \
    auth hmac\(sha256\)
0xcef79c9cb8121b6534f2994448294ec9966b799589dd7401006c49d24de8188e

sudo ip netns exec ce1 ip xfrm state add src 10.200.0.17 dst 10.200.0.25 proto
    esp spi 0x8000 mode tunnel \
    enc 'aes' 0xbcbdcd2f684af3dbe0a336ad20d6b1b2 \
    auth hmac\(sha256\)
0x3a7124edd2fa6023fe83daed0c47779c67b1b84e1dbaa7f43db9286251e4c9d5

sudo ip netns exec ce1 ip xfrm policy add src 192.168.4.1/32 dst 192.168.1.0/24
    dir out \
    tmpl src 10.200.0.25 dst 10.200.0.17 proto esp mode tunnel

sudo ip netns exec ce1 ip xfrm policy add src 192.168.1.0/24 dst 192.168.4.1/32
    dir in \
    tmpl src 10.200.0.17 dst 10.200.0.25 proto esp mode tunnel

sudo ip netns exec pe2 ip xfrm state add src 10.200.0.25 dst 10.200.0.17 proto
    esp spi  mode tunnel \
    enc 'aes' 0xf06eac104ee52865463f324070717269 \
    auth hmac\(sha256\)
0xcef79c9cb8121b6534f2994448294ec9966b799589dd7401006c49d24de8188e

sudo ip netns exec pe2 ip xfrm state add src 10.200.0.17 dst 10.200.0.25 proto
    esp spi 0x8000 mode tunnel \
    enc 'aes' 0xbcbdcd2f684af3dbe0a336ad20d6b1b2 \
    auth hmac\(sha256\)
0x3a7124edd2fa6023fe83daed0c47779c67b1b84e1dbaa7f43db9286251e4c9d5

sudo ip netns exec pe2 ip xfrm policy add src 192.168.1.0/24 dst 192.168.4.1/32
    dir out \
    tmpl src 10.200.0.17 dst 10.200.0.25 proto esp mode tunnel

sudo ip netns exec pe2 ip xfrm policy add src 192.168.4.1/32 dst 192.168.1.0/24
    dir fwd \
    tmpl src 10.200.0.25 dst 10.200.0.17 proto esp mode tunnel

# Set up IPsec ESP tunnel mode states and policies between namespaces CE1 and PE3

sudo ip netns exec ce1 ip xfrm state add src 10.200.0.25 dst 10.200.0.2 proto esp
     spi 0x9000 mode tunnel \
    enc 'aes' 0x11688ded8496b41c6544437e4bfd2a6d \
    auth hmac\(sha256\)
0x9847ad16e8cb429ce7944fed13035bbe83798f2ff9bf78bb5806eca66ff8e5fd

sudo ip netns exec ce1 ip xfrm state add src 10.200.0.2 dst 10.200.0.25 proto esp
     spi 0xa000 mode tunnel \
```

```
    enc 'aes' 0xd58efbd28e4b5dbbe3d00deb58499a49 \
    auth hmac\(sha256\)
0xc32e4825f2150e51bbf4372fec53c8ee3e03d508ad5fee7277fc5080423a4d4d

sudo ip netns exec ce1 ip xfrm policy add src 192.168.4.1/32 dst 192.168.2.0/24
    dir out \
    tmpl src 10.200.0.25 dst 10.200.0.2 proto esp mode tunnel

sudo ip netns exec ce1 ip xfrm policy add src 192.168.2.0/24 dst 192.168.4.1/32
    dir in \
    tmpl src 10.200.0.2 dst 10.200.0.25 proto esp mode tunnel

sudo ip netns exec pe3 ip xfrm state add src 10.200.0.25 dst 10.200.0.2 proto esp
     spi 0x9000 mode tunnel \
    enc 'aes' 0x11688ded8496b41c6544437e4bfd2a6d \
    auth hmac\(sha256\)
0x9847ad16e8cb429ce7944fed13035bbe83798f2ff9bf78bb5806eca66ff8e5fd

sudo ip netns exec pe3 ip xfrm state add src 10.200.0.2 dst 10.200.0.25 proto esp
     spi 0xa000 mode tunnel \
    enc 'aes' 0xd58efbd28e4b5dbbe3d00deb58499a49 \
    auth hmac\(sha256\)
0xc32e4825f2150e51bbf4372fec53c8ee3e03d508ad5fee7277fc5080423a4d4d

sudo ip netns exec pe3 ip xfrm policy add src 192.168.2.0/24 dst 192.168.4.1/32
    dir out \
    tmpl src 10.200.0.2 dst 10.200.0.25 proto esp mode tunnel

sudo ip netns exec pe3 ip xfrm policy add src 192.168.4.1/32 dst 192.168.2.0/24
    dir fwd \
    tmpl src 10.200.0.25 dst 10.200.0.2 proto esp mode tunnel

# Set up IPsec ESP tunnel mode states and policies between namespaces CE1 and PE4

sudo ip netns exec ce1 ip xfrm state add src 10.200.0.25 dst 10.200.0.6 proto esp
     spi 0xb000 mode tunnel \
    enc 'aes' 0xfa92349625d120a73777275230eebe99 \
    auth hmac\(sha256\)
0x227f73861c009352bc0439250fccd83f584065534ae72c1993b26f2500188f07

sudo ip netns exec ce1 ip xfrm state add src 10.200.0.6 dst 10.200.0.25 proto esp
     spi 0xc000 mode tunnel \
    enc 'aes' 0x6fd522b0d4606b88eea6c9838da11a2f \
    auth hmac\(sha256\)
0x5b813ac9afc82409edefe73da8a0920b585a1ddc3fbef1abe66b369769aea67d

sudo ip netns exec ce1 ip xfrm policy add src 192.168.4.1/32 dst 192.168.3.0/24
    dir out \
    tmpl src 10.200.0.25 dst 10.200.0.6 proto esp mode tunnel

sudo ip netns exec ce1 ip xfrm policy add src 192.168.3.0/24 dst 192.168.4.1/32
    dir in \
    tmpl src 10.200.0.6 dst 10.200.0.25 proto esp mode tunnel

sudo ip netns exec pe4 ip xfrm state add src 10.200.0.25 dst 10.200.0.6 proto esp
     spi 0xb000 mode tunnel \
    enc 'aes' 0xfa92349625d120a73777275230eebe99 \
    auth hmac\(sha256\)
0x227f73861c009352bc0439250fccd83f584065534ae72c1993b26f2500188f07

sudo ip netns exec pe4 ip xfrm state add src 10.200.0.6 dst 10.200.0.25 proto esp
     spi 0xc000 mode tunnel \
    enc 'aes' 0x6fd522b0d4606b88eea6c9838da11a2f \
    auth hmac\(sha256\)
0x5b813ac9afc82409edefe73da8a0920b585a1ddc3fbef1abe66b369769aea67d

sudo ip netns exec pe4 ip xfrm policy add src 192.168.3.0/24 dst 192.168.4.1/32
    dir out \
    tmpl src 10.200.0.6 dst 10.200.0.25 proto esp mode tunnel

```

```
sudo ip netns exec pe4 ip xfrm policy add src 192.168.4.1/32 dst 192.168.3.0/24
    dir fwd \
    tmpl src 10.200.0.25 dst 10.200.0.6 proto esp mode tunnel
```

To complete the remote access VPN configurations, I executed the following commands for the remote host CE5, applying the same principles previously discussed:

```
# Set up IPsec ESP tunnel mode states and policies between namespaces CE5 and PE2

sudo ip netns exec ce5 ip xfrm state add src 10.200.0.26 dst 10.200.0.14 proto
    esp spi 0xd000 mode tunnel \
    enc 'aes' 0xc97c42943e15955e3d3b5b6f72728a3d \
    auth hmac\(sha256\)
0x6e0185a1f0e2775db47f00260653489b29b616dc0f49462ce6e007e83e602319

sudo ip netns exec ce5 ip xfrm state add src 10.200.0.14 dst 10.200.0.26 proto
    esp spi 0xe000 mode tunnel \
    enc 'aes' 0x1a50ceb727bdf10d9a5e4be9b8ae9659 \
    auth hmac\(sha256\)
0x0b58e58d43ae0233cc52841747d9d376f58acf07fc5dda73a34f22f8c9c98c09

sudo ip netns exec ce5 ip xfrm policy add src 192.168.4.2/32 dst 192.168.1.0/24
    dir out \
    tmpl src 10.200.0.26 dst 10.200.0.14 proto esp mode tunnel

sudo ip netns exec ce5 ip xfrm policy add src 192.168.1.0/24 dst 192.168.4.2/32
    dir in \
    tmpl src 10.200.0.14 dst 10.200.0.26 proto esp mode tunnel

sudo ip netns exec pe2 ip xfrm state add src 10.200.0.26 dst 10.200.0.14 proto
    esp spi 0xd000 mode tunnel \
   enc 'aes' 0xc97c42943e15955e3d3b5b6f72728a3d \
   auth hmac\(sha256\)
0x6e0185a1f0e2775db47f00260653489b29b616dc0f49462ce6e007e83e602319

sudo ip netns exec pe2 ip xfrm state add src 10.200.0.14 dst 10.200.0.26 proto
    esp spi 0xe000 mode tunnel \
    enc 'aes' 0x1a50ceb727bdf10d9a5e4be9b8ae9659 \
          auth hmac\(sha256\)
0x0b58e58d43ae0233cc52841747d9d376f58acf07fc5dda73a34f22f8c9c98c09

sudo ip netns exec pe2 ip xfrm policy add src 192.168.1.0/24 dst 192.168.4.2/32
    dir out \
    tmpl src 10.200.0.14 dst 10.200.0.26 proto esp mode tunnel

sudo ip netns exec pe2 ip xfrm policy add src 192.168.4.2/32 dst 192.168.1.0/24
    dir fwd \
    tmpl src 10.200.0.26 dst 10.200.0.14 proto esp mode tunnel

# Set up IPsec ESP tunnel mode states and policies between namespaces CE5 and PE3

sudo ip netns exec ce5 ip xfrm state add src 10.200.0.26 dst 10.200.0.9 proto esp
     spi 0xf000 mode tunnel \
    enc 'aes' 0xfa9ad3d8976e6d7ee9a5939762e8a989 \
    auth hmac\(sha256\)
0x6e728e9dd47791d700fa08bec5f7e2f181df5fc5826e2743538ad81fab119741

sudo ip netns exec ce5 ip xfrm state add src 10.200.0.9 dst 10.200.0.26 proto esp
     spi 0x10000 mode tunnel \
    enc 'aes' 0x4f4dea2ac7657081271eaca0b5ca3790 \
    auth hmac\(sha256\)
0xee28e55e557dd6874bb14f943eaf65efeb8e72bb2807510228b203574e64ea30

sudo ip netns exec ce5 ip xfrm policy add src 192.168.4.2/32 dst 192.168.2.0/24
    dir out \
    tmpl src 10.200.0.26 dst 10.200.0.9 proto esp mode tunnel

sudo ip netns exec ce5 ip xfrm policy add src 192.168.2.0/24 dst 192.168.4.2/32
    dir in \
    tmpl src 10.200.0.9 dst 10.200.0.26 proto esp mode tunnel

```

```
sudo ip netns exec pe3 ip xfrm state add src 10.200.0.26 dst 10.200.0.9 proto esp
     spi 0xf000 mode tunnel \
    enc 'aes' 0xfa9ad3d8976e6d7ee9a5939762e8a989 \
  auth hmac\(sha256\)
0x6e728e9dd47791d700fa08bec5f7e2f181df5fc5826e2743538ad81fab119741

sudo ip netns exec pe3 ip xfrm state add src 10.200.0.9 dst 10.200.0.26 proto esp
     spi 0x10000 mode tunnel \
    enc 'aes' 0x4f4dea2ac7657081271eaca0b5ca3790 \
  auth hmac\(sha256\)
0xee28e55e557dd6874bb14f943eaf65efeb8e72bb2807510228b203574e64ea30

sudo ip netns exec pe3 ip xfrm policy add src 192.168.2.0/24 dst 192.168.4.2/32
    dir out \
    tmpl src 10.200.0.9 dst 10.200.0.26 proto esp mode tunnel

sudo ip netns exec pe3 ip xfrm policy add src 192.168.4.2/32 dst 192.168.2.0/24
    dir fwd \
    tmpl src 10.200.0.26 dst 10.200.0.9 proto esp mode tunnel

# Set up IPsec ESP tunnel mode states and policies between namespaces CE5 and PE4

sudo ip netns exec ce5 ip xfrm state add src 10.200.0.26 dst 10.200.0.22 proto
    esp spi 0x11000 mode tunnel \
    enc 'aes' 0x41b2c59c4256d721521f2b8385322677 \
  auth hmac\(sha256\)
0xf81f2f7fa4a6d653bb3f86aa01034b642203b51c5c562b4bfe7c1a6da46fb273

sudo ip netns exec ce5 ip xfrm state add src 10.200.0.22 dst 10.200.0.26 proto
    esp spi 0x12000 mode tunnel \
    enc 'aes' 0x5b5922481a6e0389645e2e0c9e490adb \
                auth hmac\(sha256\) 0
    x2849dab129abbc9c625f0aeb31abdb61bace168ce91208caaad18cc36eab46c1

sudo ip netns exec ce5 ip xfrm policy add src 192.168.4.2/32 dst 192.168.3.0/24
    dir out \
    tmpl src 10.200.0.26 dst 10.200.0.22 proto esp mode tunnel

sudo ip netns exec ce5 ip xfrm policy add src 192.168.3.0/24 dst 192.168.4.2/32
    dir in \
    tmpl src 10.200.0.22 dst 10.200.0.26 proto esp mode tunnel

sudo ip netns exec pe4 ip xfrm state add src 10.200.0.26 dst 10.200.0.22 proto
    esp spi 0x11000 mode tunnel \
    enc 'aes' 0x41b2c59c4256d721521f2b8385322677 \
  auth hmac\(sha256\)
0xf81f2f7fa4a6d653bb3f86aa01034b642203b51c5c562b4bfe7c1a6da46fb273

sudo ip netns exec pe4 ip xfrm state add src 10.200.0.22 dst 10.200.0.26 proto
    esp spi 0x12000 mode tunnel \
    enc 'aes' 0x5b5922481a6e0389645e2e0c9e490adb \
  auth hmac\(sha256\)
0x2849dab129abbc9c625f0aeb31abdb61bace168ce91208caaad18cc36eab46c1

sudo ip netns exec pe4 ip xfrm policy add src 192.168.3.0/24 dst 192.168.4.2/32
    dir out \
    tmpl src 10.200.0.22 dst 10.200.0.26 proto esp mode tunnel

sudo ip netns exec pe4 ip xfrm policy add src 192.168.4.2/32 dst 192.168.3.0/24
    dir fwd \
    tmpl src 10.200.0.26 dst 10.200.0.22 proto esp mode tunnel
```

6.3 Testing and Verification

To simplify the testing phase, I will analyze only the connectivity between remote host CE1 and office 2 (192.168.1.0/24), and between remote host CE5 and office 4 (192.168.3.0/24) by initiating pings. I captured the packet data by monitoring the virtual interfaces of the Linux bridge, specifically veth01, veth08, and veth05 (for the CE1-CE2 case) and veth17, veth22, and

veth13 (for the CE5-CE4 case) using Wireshark. Each interface was monitored with a dedicated packet sniffer instance (Figure 2 provides a summary of the interface information):

```
sudo ip netns exec ce1 ping 192.168.1.2 -c 5
```

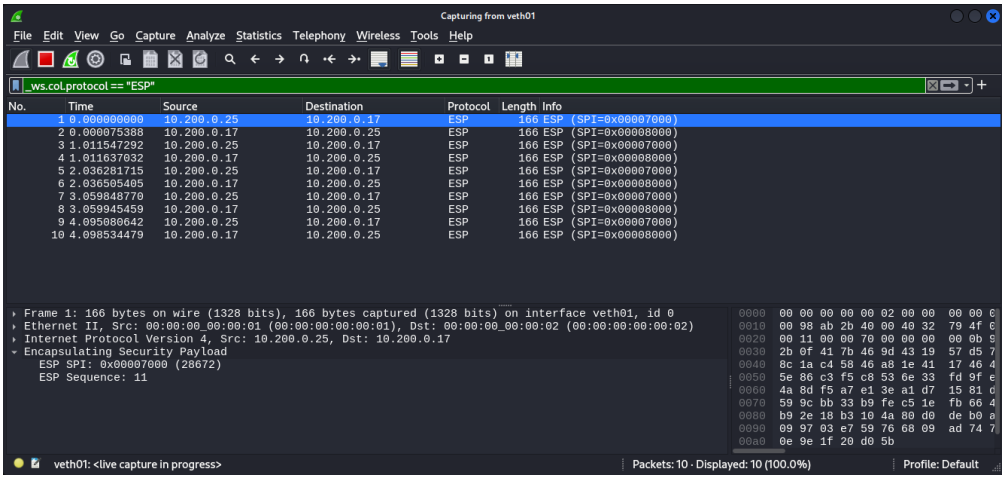

Figure 26: Capturing on veth01 - Namespace CE1

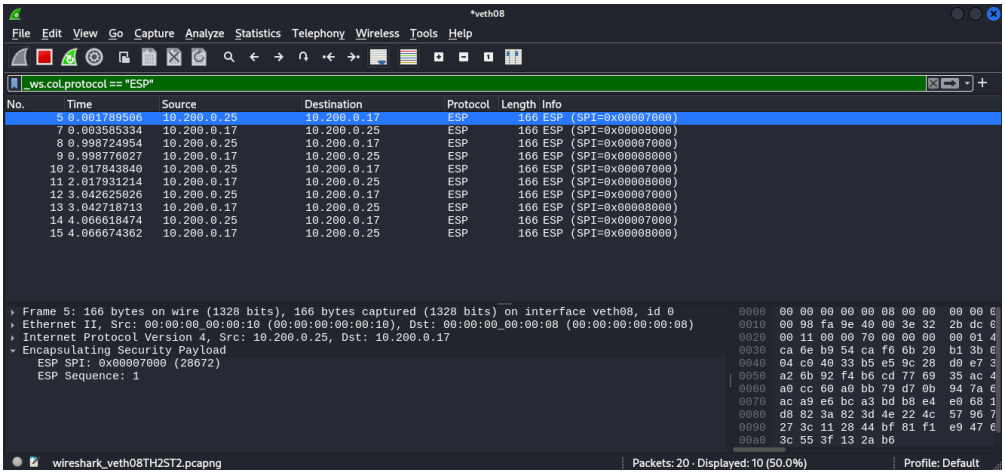

Figure 27: Capturing on veth08 - Namespace PE2

The states and policies have been correctly configured, with the tunnel endpoints being the eth01 interface on host CE1 and the eth08 interface on VPN gateway PE2 (IP address 10.200.0.17). An ICMP echo request originated by CE1 is encapsulated by the new IP header and reaches gateway PE1 via the eth02 interface (with destination MAC address 00:00:00:00:00:02, as shown in Figure 26). The ESP packet reaches the eth08 interface of the corporate gateway PE2 after passing through PE4 (with source MAC address 00:00:00:00:00:10, as shown in Figure 27) and is processed by the kernel to reach the corporate host CE2 in plaintext (with destination MAC address 00:00:00:00:00:05, as shown in Figure 28).
It is worth noting that the remote host's interface uses an IP address 192.168.4.1 assigned from the VPN pool as the source IP to communicate with the company offices, as expected.

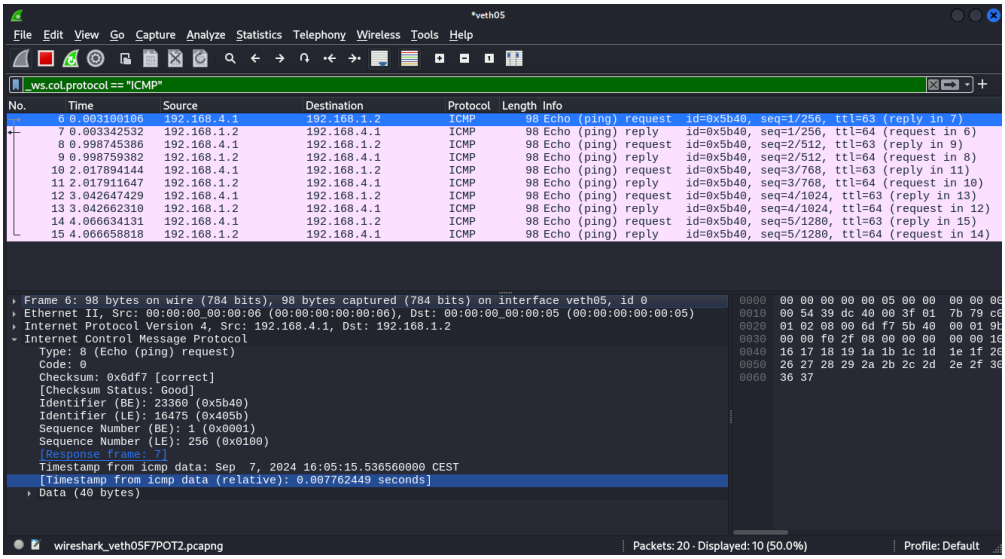

Figure 28: Capturing on veth05 - Namespace CE2

Finally, I analyze what happens when the company host CE4 pings the remote host CE5, which has been assigned the IP address 192.168.4.2:

```
sudo ip netns exec ce4 ping 192.168.4.2 -c 5
```

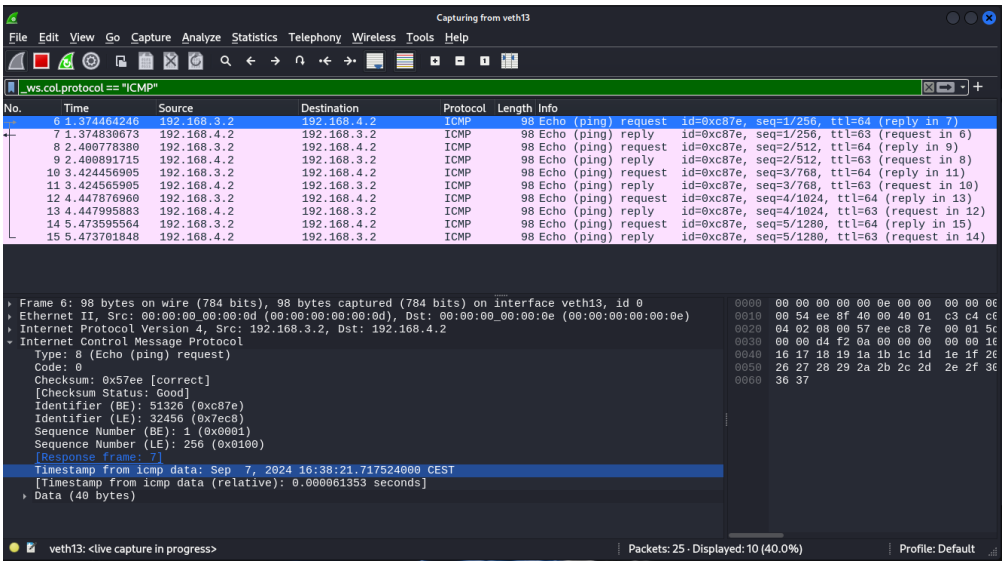

Figure 29: Capturing on veth13 - Namespace CE4

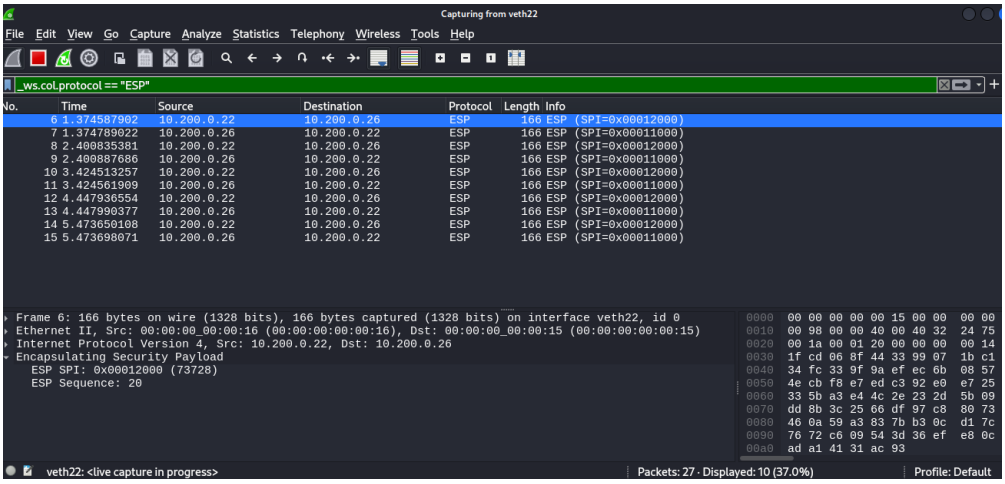

Figure 30: Capturing on veth22 - Namespace PE4

The images again show the correct establishment of the remote access VPN. Let's trace the path of an ICMP echo request originating from host CE4.
After reaching the VPN gateway PE4 through interface eth14 (with destination MAC address 00:00:00:00:00:0e, as shown in Figure 29), the packet is securely directed towards interface eth21 of gateway PE5 (with destination MAC address 00:00:00:00:00:15, as shown in Figure 30).
The latter consults its routing table and forwards the packet with destination IP address 10.200.0.26 to its interface within the private network 172.16.1.0/24, eth18. There, it finally reaches interface eth17 of host CE5 (IP address 172.16.1.2/24), which verifies the packet's integrity and decrypts it using the keys from the defined Security Association (SA), then sends an echo reply back to host CE4, traversing the tunnel again (Figure 31).

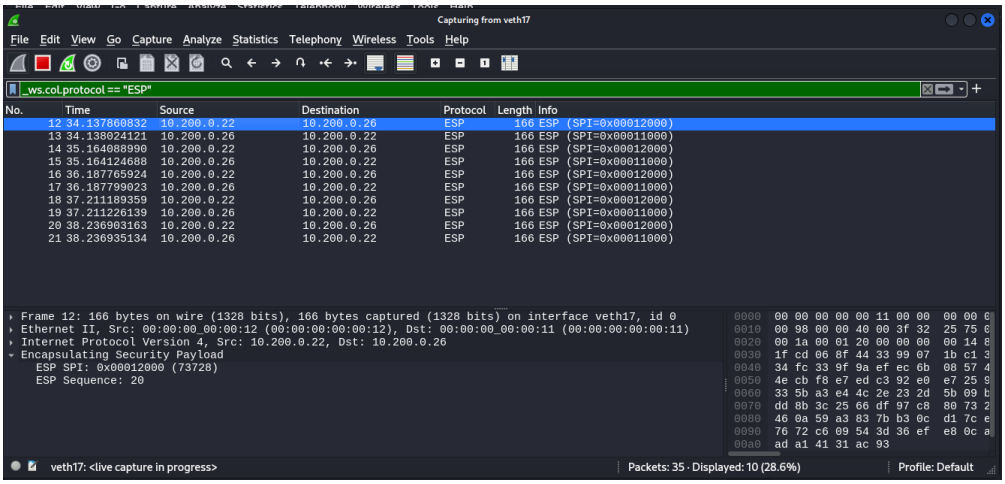

Figure 31: Capturing on veth17 - Namespace CE5

7 strongSwan

7.1 strongSwan in Linux Network Namespaces

strongSwan is an open-source IPsec-based VPN solution that greatly simplifies the setup of secure communication between hosts. It offers advanced features such as automated peer authentication and dynamic key exchange, which are not supported by IP xfrm. In my project, I introduced strongSwan to overcome the limitations of IP xfrm, which requires manual and static configuration of security policies and associations. Unlike IP xfrm, strongSwan uses the

Internet Key Exchange (IKE) protocol to automate the negotiation of encryption and integrity algorithms and secure key exchanges, this automation reduces the complexity and potential for errors.
Running multiple instances of strongSwan in separate network namespaces can be challenging since all namespaces share the same file system. However, the ip netns exec command offers a useful feature: files located in /etc/netns/"namespace"/ are bind-mounted over their counterparts in /etc/ for that specific namespace. This mechanism allows for distinct configuration files for each instance of strongSwan within different namespaces.
A common conflict arises with the default location for PID files and UNIX sockets, which is /var/run. Multiple instances of strongSwan trying to access the same directory would result in conflicts. To solve this, I configured strongSwan with –sysconfdir=/etc and redirect the PID directory using –with-piddir=/etc/ipsec.d/run. After building and installing strongSwan, I set up unique directories for each namespace. Specifically, I configured directories for namespaces PE3 and PE4 to establish a site-to-site VPN:

```
# Download and extract strongSwan
wget https://download.strongswan.org/strongswan-5.9.13.tar.bz2
tar xjf strongswan-5.9.13.tar.bz2
cd strongswan-5.9.13

# Configure strongSwan
./configure --sysconfdir=/etc --with-piddir=/etc/ipsec.d/run

# Compile and install
make
sudo make install

# Verify installation
ipsec --version

# Prepare directories for network namespaces
sudo mkdir -p /etc/netns/pe3/ipsec.d/run
sudo mkdir -p /etc/netns/pe4/ipsec.d/run
```

7.2 Site-to-Site VPN Setup with strongSwan

In the site-to-site VPN setup using strongSwan, three essential configuration files are involved: ipsec.conf, ipsec.secrets, and topology+routing.sh. The ipsec.conf file defines the VPN connections and their specific parameters, while ipsec.secrets manages authentication details, such as pre-shared keys or certificates. Finally, topology+routing.sh configures the network topology and sets up routing for VPN traffic.
I created a Strongswan configuration /etc/netns/pe3/ipsec.conf for gateaway PE3:

```
# PE3 ipsec.conf
config setup
        charondebug="all"
        uniqueids=yes
conn s2s
        type=tunnel
        auto=start
        keyexchange=ikev2
        authby=secret
        left=10.200.0.2
        leftid=@PE3
        leftsubnet=192.168.2.0/24
        right=10.200.0.6
        rightid=@PE4
        rightsubnet=192.168.3.0/24
        ike=aes128-sha256-modp1024!
        esp=aes128-sha256!
        keyingtries=%forever
        ikelifetime=28800s
        lifetime=3600s
```

This file contains important settings for establishing the IPsec tunnel, the following parameters are critical:

- **config setup**: This section includes general strongSwan settings. The charondebug="all" option sets the debug level for troubleshooting, while uniqueids=yes ensures that only one connection per peer is active at a time.
- **conn s2s**: This block defines the specifics of the site-to-site VPN connection.
- **type=tunnel**: Specifies that we are setting up a tunnel mode connection, encapsulating entire IP packets.
- **auto=start**: This instructs strongSwan to automatically bring up the VPN when the service starts.
- **keyexchange=ikev2**: Defines that the IKEv2 protocol will be used for exchanging keys and negotiating security associations.
- **authby=secret**: Authentication will be done using a pre-shared secret, defined in the ipsec.secrets file.
- **left and right**: These refer to the local (PE3) and remote (PE4) IP addresses.
- **left=10.200.0.2 and right=10.200.0.6** are the public IP addresses of eth11 (namespace PE3) and eth15 (namespace PE4), respectively.
- **leftsubnet=192.168.2.0/24 and rightsubnet=192.168.3.0/24**: define the internal subnets behind PE3 and PE4 that will be reachable through the VPN tunnel.
- **ike=aes128-sha256-modp1024!**: Specifies the algorithms for the IKE phase (key exchange), including AES for encryption, SHA256 for integrity, and MODP1024 for Diffie-Hellman group (The exclamation mark indicates that these settings are mandatory and must be supported by both peers).
- **esp=aes128-sha256!**: is the encapsulation security suite of protocols.
- **keyingtries=%forever**: Tells the VPN to retry key negotiation indefinitely in case of failure.
- **ikelifetime=28800s and lifetime=3600s**: Set the lifetimes for IKE and IPsec security associations, after which they will be renegotiated.

Then I configured /etc/netns/pe4/ipsec.conf for gateaway PE4:

```
# PE4 ipsec.conf
config setup
        charondebug="all"
        uniqueids=yes
conn s2s
        type=tunnel
        auto=start
        keyexchange=ikev2
        authby=secret
        left=10.200.0.6
        leftid=@PE4
        leftsubnet=192.168.3.0/24
        right=10.200.0.2
        rightid=@PE3
        rightsubnet=192.168.2.0/24
        ike=aes128-sha256-modp1024!
        esp=aes128-sha256!
        keyingtries=%forever
        ikelifetime=28800s
        lifetime=3600s
```

For both gateaways, I created an ipsec.secrets file to specify the pre-shared key used for authentication (/etc/netns/"namespace"/ipsec.secrets):

```
# PE3 ipsec.secrets
@PE3 @PE4 : PSK "qLGLTVQOfqvGLsWP75FEtLGtwN3Hu0ku6C5HItKo6ac="

# PE4 ipsec.secrets
@PE4 @PE3 : PSK "qLGLTVQOfqvGLsWP75FEtLGtwN3Hu0ku6C5HItKo6ac="
```

Finally, I started the IPsec services in both namespaces:

```
sudo ip netns exec pe3 ipsec start
sudo ip netns exec pe4 ipsec start
```

7.3 Testing and Verification

To analyze the packets exchanged between the two gateways during the IKEv2 phase, I opened a Wireshark instance on the virtual interface veth15 (which connects to the eth15 interface in the PE4 namespace, with IP address 10.200.0.6). Figure 32 shows the details of the IKE_SA_INIT exchange, which establishes the initial Security Association between the peers for the IKE protocol itself (referred to as IKE_SA). This exchange negotiates the encryption and integrity protection algorithms used to secure further communication. Notably, the proposed algorithms in Figure 33 match exactly those specified in the ipsec.conf configuration file (ike = aes128-sha256-modp1024!).
The second, protected exchange, IKE_AUTH, involves the authentication of the peers (in this case, using pre-shared keys) and the creation of the IPsec SAs that will be used to secure the actual IPsec traffic. In the IKE_AUTH packets (Figure 34), the SPIs representing the SAs established during the IKE_SA_INIT phase can be seen for both the initiator and responder. Additionally, as previously mentioned, the payload is encrypted and authenticated.

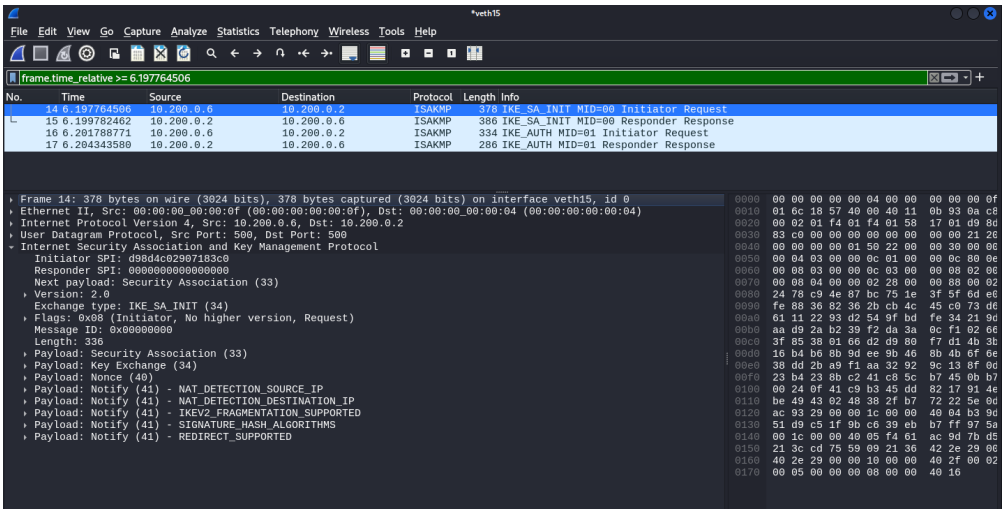

Figure 32: Capturing on veth15 - Namespace PE4

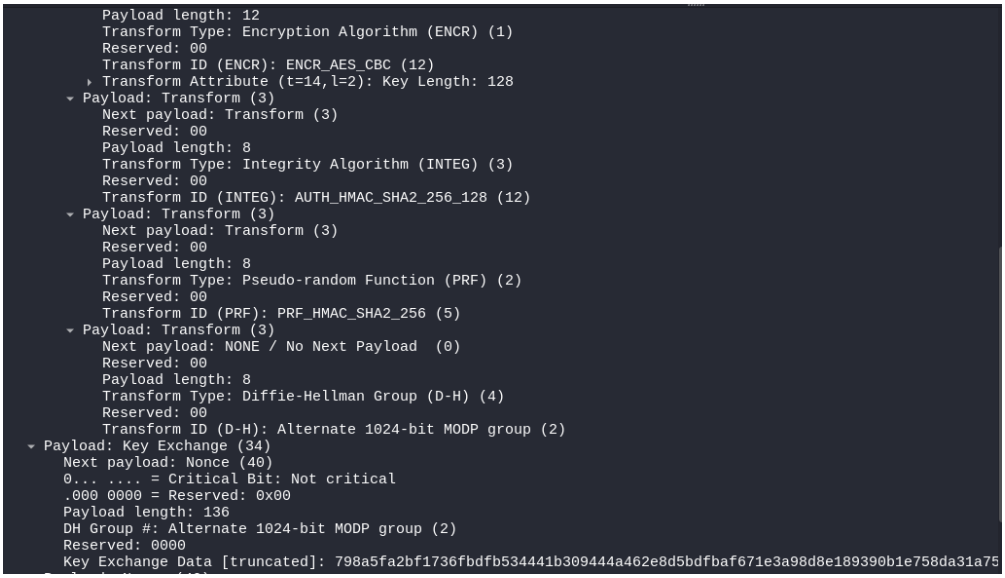

Figure 33: Capturing on veth15 - Namespace PE4 - IKE_SA_INIT

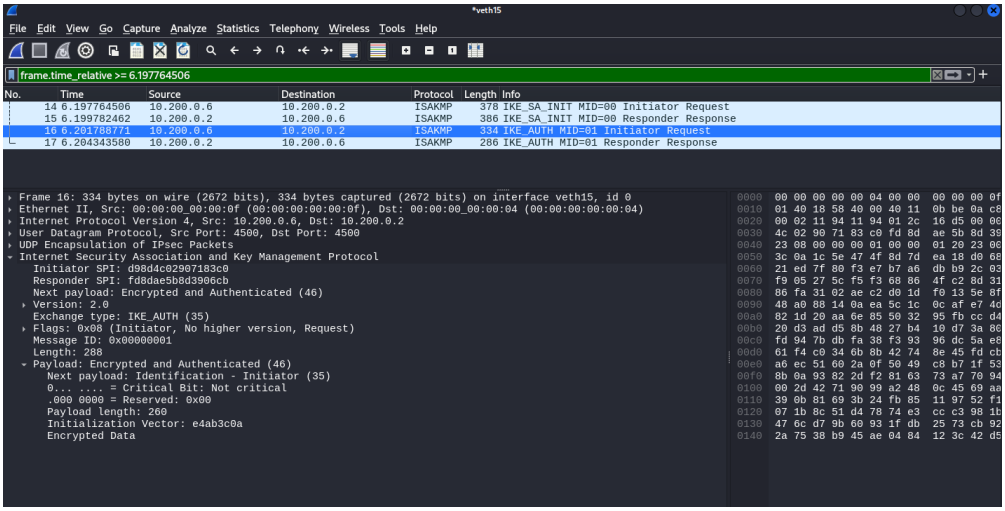

Figure 34: Capturing on veth15 - Namespace PE4 - IKE_AUTH

The output of the *sudo ip netns exec pe4 ipsec statusall* command (Figure 35) offers a detailed overview of the IPsec configuration within the PE4 network namespace:

- It shows the IP addresses of the interfaces within the PE4 namespace that the daemon uses to establish and manage IPsec connections.
- It shows a tunnel connection between the subnets 192.168.3.0/24 and 192.168.2.0/24, with IP addresses 10.200.0.6 and 10.200.0.2, using IKEv2 and pre-shared key authentication.
- It provides comprehensive details about the active Security Associations, including the time of establishment, IKEv2 SPIs, proposal details, ESP SPIs and encryption and integrity algorithms, as well as traffic statistics and the rekeying schedule.

```
┌──(kali㉿kali)-[/etc/netns]
└─$ sudo ip netns exec pe4 ipsec statusall
Status of IKE charon daemon (strongSwan 5.9.13, Linux 6.6.9-amd64, x86_64):
  uptime: 17 seconds, since Sep 09 16:40:56 2024
  malloc: sbrk 2420736, mmap 0, used 468240, free 1952496
  worker threads: 11 of 16 idle, 5/0/0/0 working, job queue: 0/0/0/0, scheduled: 5
  loaded plugins: charon aes des rc2 sha2 sha1 md5 mgf1 random nonce x509 revocation constraints pubkey pkcs1 pkcs
7 pkcs12 pgp dnskey sshkey pem pkcs8 fips-prf gmp curve25519 xcbc cmac hmac kdf gcm drbg attr kernel-netlink resol
ve socket-default stroke vici updown xauth-generic counters
Listening IP addresses:
  192.168.3.1
  10.200.0.6
  10.200.0.18
  10.200.0.22
Connections:
         s2s:  10.200.0.6...10.200.0.2  IKEv2
         s2s:   local:  [PE4] uses pre-shared key authentication
         s2s:   remote: [PE3] uses pre-shared key authentication
         s2s:   child:  192.168.3.0/24 === 192.168.2.0/24 TUNNEL
Security Associations (1 up, 0 connecting):
         s2s[1]: ESTABLISHED 13 seconds ago, 10.200.0.6[PE4]...10.200.0.2[PE3]
         s2s[1]: IKEv2 SPIs: d98d4c02907183c0_i* fd8dae5b8d3906cb_r, pre-shared key reauthentication in 7 hours
         s2s[1]: IKE proposal: AES_CBC_128/HMAC_SHA2_256_128/PRF_HMAC_SHA2_256/MODP_1024
         s2s{2}:  INSTALLED, TUNNEL, reqid 1, ESP SPIs: c3343871_i c26b643c_o
         s2s{2}:  AES_CBC_128/HMAC_SHA2_256_128, 0 bytes_i, 0 bytes_o, rekeying in 44 minutes
         s2s{2}:   192.168.3.0/24 === 192.168.2.0/24
```

Figure 35: Output of ipsec statusall in PE4

```
┌──(kali㉿kali)-[/etc/netns]
└─$ sudo ip netns exec pe4 ip xfrm state
src 10.200.0.6 dst 10.200.0.2
        proto esp spi 0×c26b643c reqid 1 mode tunnel
        replay-window 0 flag af-unspec
        auth-trunc hmac(sha256) 0×7f2017c319fe6a722e682cc0a498997ce26c467e208ff72582a6afdfe92e53cb 128
        enc cbc(aes) 0×2423a402fa35f35d798d4b3487f5f60a
        anti-replay context: seq 0×0, oseq 0×0, bitmap 0×00000000
src 10.200.0.2 dst 10.200.0.6
        proto esp spi 0×c3343871 reqid 1 mode tunnel
        replay-window 32 flag af-unspec
        auth-trunc hmac(sha256) 0×edee1e249a0350bd79cc95b3e7e137724a5cedff128fe0a453c88c4c9c924267 128
        enc cbc(aes) 0×0ca93437988dfd22f7ab8ae049d9d222
        anti-replay context: seq 0×0, oseq 0×0, bitmap 0×00000000

┌──(kali㉿kali)-[/etc/netns]
└─$ sudo ip netns exec pe4 ip xfrm policy
src 192.168.3.0/24 dst 192.168.2.0/24
        dir out priority 375423 ptype main
        tmpl src 10.200.0.6 dst 10.200.0.2
                proto esp spi 0×c26b643c reqid 1 mode tunnel
src 192.168.2.0/24 dst 192.168.3.0/24
        dir fwd priority 375423 ptype main
        tmpl src 10.200.0.2 dst 10.200.0.6
                proto esp reqid 1 mode tunnel
src 192.168.2.0/24 dst 192.168.3.0/24
        dir in priority 375423 ptype main
        tmpl src 10.200.0.2 dst 10.200.0.6
                proto esp reqid 1 mode tunnel
```

Figure 36: Output of ip xfrm in PE4

The commands *ip xfrm state show* and *ip xfrm policy show* within the namespace PE4 (Figure 36) confirm the successful creation of the site-to-site VPN between office 3 and 4, as their outputs match those obtained from previous VPN configurations using IP xfrm. This verification is supported by the observation that a ping between the corporate hosts CE3 and CE4 results in ESP packets being captured on the virtual network interface veth11 of the gateway PE3 (Figure 37). This consistent output and successful packet capture demonstrate that the VPN is functioning correctly.

```
1 sudo ip netns exec ce3 ping 192.168.3.2 -c 3
```

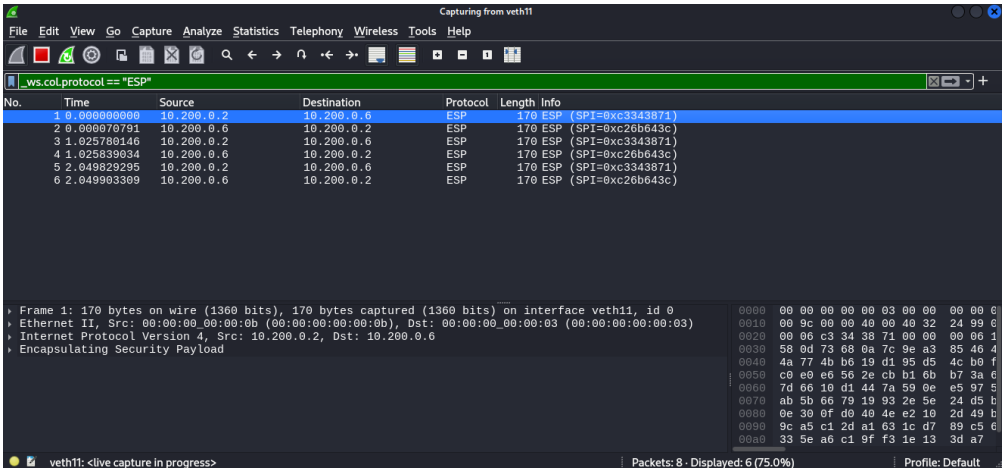

Figure 37: Capturing on veth11 - Namespace PE3

8 Conclusions

In today's digital landscape, cyber threats are evolving rapidly with new methods and technologies emerging daily. It is crucial to protect sensitive information, especially when transmitting data over shared and potentially dangerous mediums like the internet. Virtual Private Network technology serves as a vital defense tool in this regard.
Throughout this project, I have applied my networking knowledge in a practical setting, working with network interfaces, static routing, and IP address assignments. I have gained hands-on experience with creating complex network topologies using veth, Linux bridges, and network namespaces within a Kali Linux VM. This project allowed me to delve deeper into IPsec, understanding its management by the Linux kernel, and exploring the implementation of VPNs via ip xfrm and strongSwan. By implementing various VPN configurations, I have assessed the advantages and disadvantages of each setup, gaining insights into the critical decisions a network administrator must make to safeguard network traffic. This project has significantly enhanced my understanding of VPN technology and its practical applications in securing communication over the internet.