# Task 1: Idle Host Scan

## Introduction

Task 1 involves conducting an **Idle Host Scan** using a Windows 7 machine as a "zombie," Kali2 as the attacker, and Kali3 as the victim. A step-by-step breakdown will be shown as well as analysis of the results.

The first step to take for this is verify if the windows machine is a suitable zombie. To test this, we can ping the windows and capture the packets for analysis. To confirm if it is a suitable zombie, we can observe the IP ID increments in the echo replies, if it is increasing by 1, it means it is a suitable zombie in an idle scan.



**Fig 1.1 Task1a.pcap**

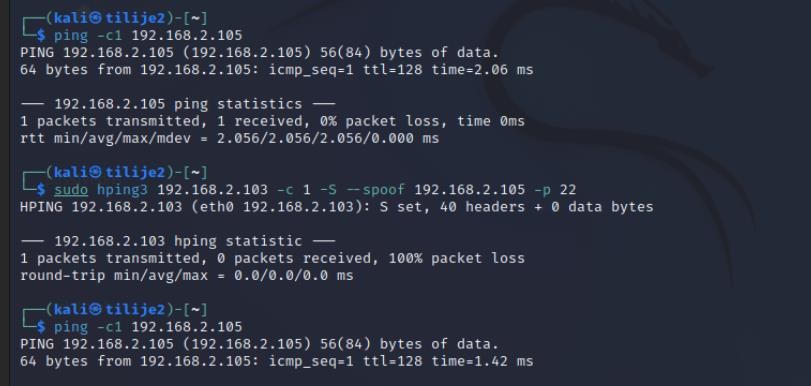
The next step is to go to the victim’s machine to start ssh. The purpose of this is to open port 22, which will be the port the idle scan will target.



**Fig 1.2 Starting SSH**

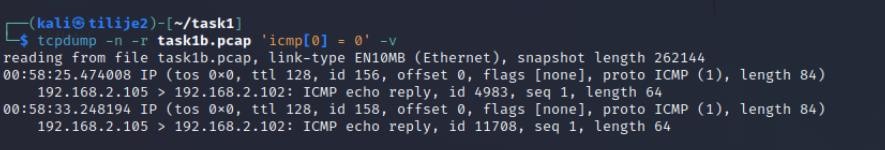
Now, start another packet capture to analyze the idle host scan (open port) about to be done. First, we send a baseline ping to the zombie to get the current IP ID value, then we send a SYN packet to the victim on port 22 with the source address being spoofed as the zombie (192.168.2.105).

The purpose of the second ping is to allow us check if the IP ID has gone up by 2 (one increment for the ping reply and the other for response from the victim). This shows that the port on the victim’s machine is opened.



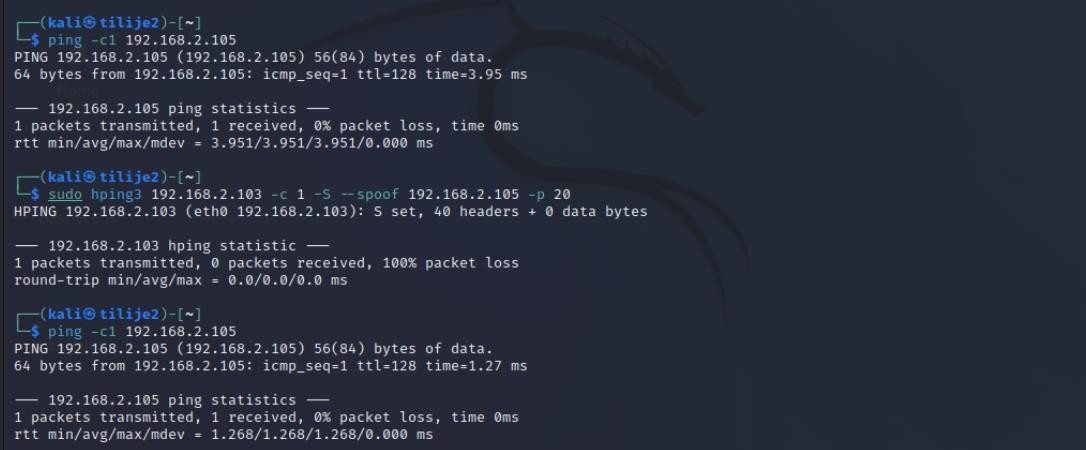
**Fig 1.3 Pinging in open port**

We kill the capture and we can see that the IP ID values increased by 2 after sending the spoofed packet, indicating that the victim reponded to the SYN packet sent to port 22.

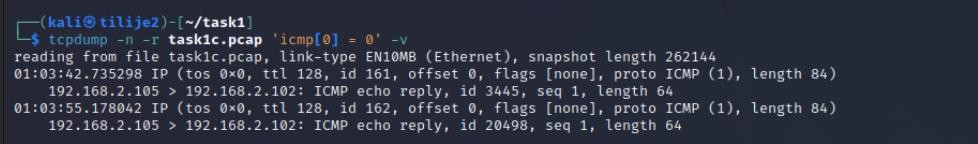


**Fig 1.4 Result from the packets sent.**

Now, we repeat the same steps in a closed port and we can see from fig 1.6 that IP ID in the zombie reply only increases by 1. This indicates that the victim that the victim did not respond to the SYN packet sent because it is a closed port.



**Fig 1.5 Ping from a closed port.**



**Fig 1.6 Results.**

## Summary and Significance of the Idle Host Scan

Idle Host Scan is a stealth scanning technique that allows attackers to know the status of ports on a victim’s device without directly interacting with it. This scan leverages “zombie” hosts with predictable IP ID behavior to mirror victim responses, which allows the attacker to assess whether a particular port is open or closed based on the IP ID.

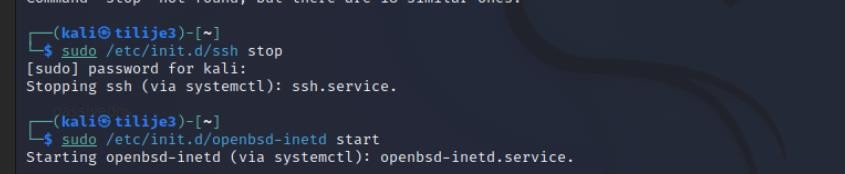
**Significance:**

Stealth: Scanning does not directly engage the target. This reduces the chance of detection.

Effectiveness: By observing IP ID increments, an attacker can reliably determine the port status on the victim.

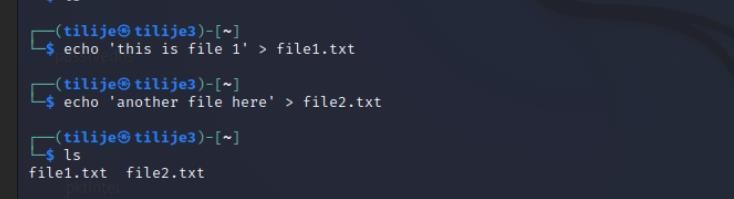
# Task 2: Sniffing in a switched environment: ARP Cache Poisoning

The first step in this task was to enable telnet, which is because Telnet sends data in plain text, which is ideal for observing sensitive information.



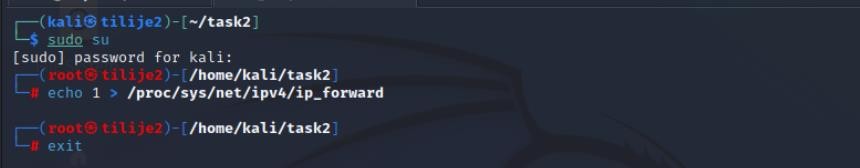
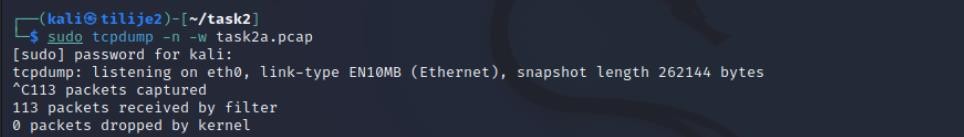
**Fig 2.1 Starting telnet.**

Now, we create a new user and add some files that will be accessed via Telnet later on. The attacker will be able to view the communication between the two victims in plain text.



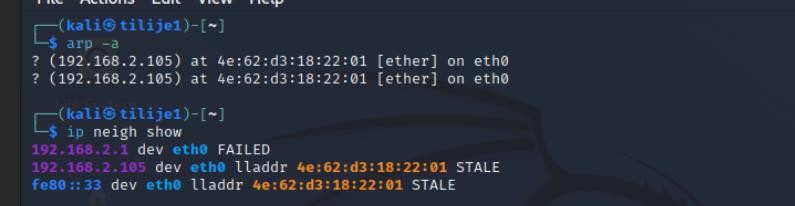
## Fig 2.2 Creating files in kali3

We return to the attacker machine and start capturing packets. Enabling IP forwarding is essential because it allows the attacker to be able to perform ARP spoofing.



**Fig 2.3 Enabling Ip Forwarding.**

Before we begin to send the spoofed ARP, we run these commands in the screenshot below to ensure we have the baseline MAC addresses before the attack is carried out

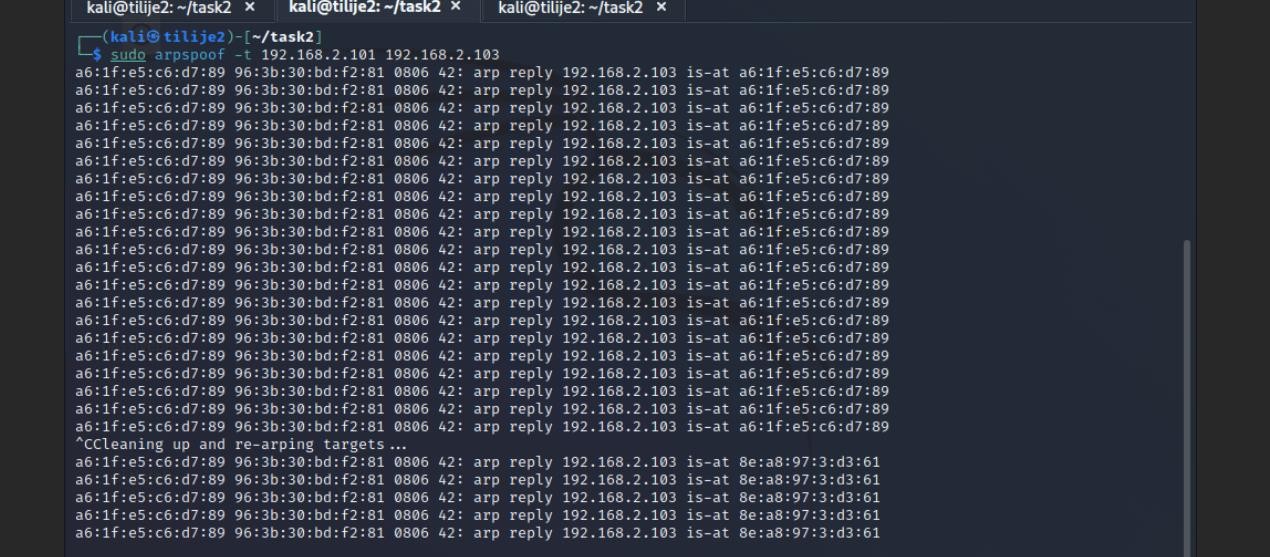


## Fig 2.4 MAC Addresses

Now we send the spoofed ARP Replies from the attacker’s machine. Using the command;

**sudo arpspoof -t 192.168.2.101 192.168.2.103**

The goal of this ARP spoofing command is to trick the target device Kali1 (192.168.2.101) into believing that Kali3's IP (192.168.2.103) is associated with the attacker's MAC address (a6:1f:e5:c6:d7: 89) ) . This trick allows an attacker (Kali2) to intercept, modify, or redirect any traffic. that is intended for Kali3.



The displayed result above (Fig 2.5) is a set of ARP replies sent multiple times by the attacker to maintain control of the victim's ARP cache, each row represents a spoofed ARP reply packet.

Every part here has meaning, these include;

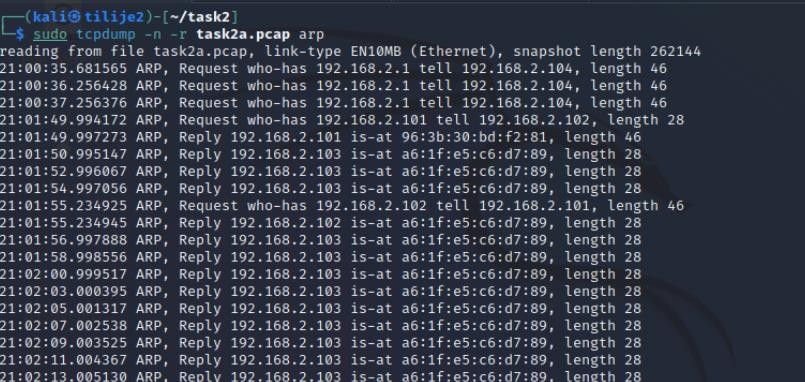
* ARP reply message: Each line shows an ARP response to **192.168.2.103 is-at a6:1f:e5:c6:d7:89.** This is a web message claiming that Kali3's IP

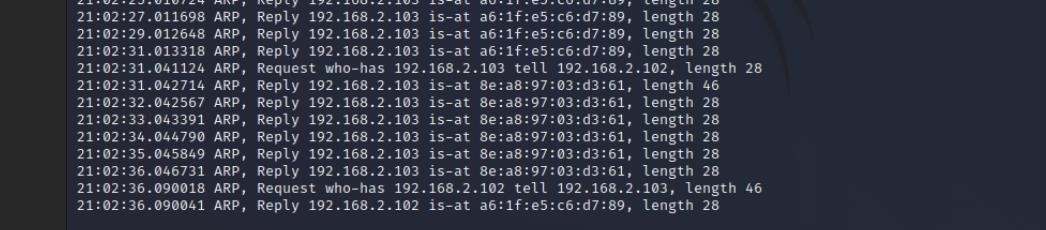
(192.168.2.103) is associated with the attacker's MAC address (a6:1f:e5:c6:d7:89).

* Repeat pattern: The results show a repeatable ARP response pattern. This is intentional in ARP spoofing because the ARP tables on network devices are periodically refreshed or may be overwritten by legitimate ARP responses. By continuously sending these replies, the attacker will ensure that the victim's ARP table remains poisoned, by maintaining relationships with 192.168.2.103 with the attacker's MAC address.
* "Cleaning and re-targeting...": This message indicates that arpspoof stopped the attack and attempted to "clear" the victim's ARP cache by restoring it to its original state. The tool sends a valid ARP response to resolve the MAC-IP connection back to the actual MAC address of 192.168.2.103 (in this case **8e:a8:97:03:d3:61**), effectively canceling the ARP poisoning efficiency.
* New ARP response from a valid MAC: After the cleanup, you will see a new ARP response saying 192.168.2.103 is already associated with the correct MAC address

**8e:a8:97:03:d3:61.** This final step is Arpspoof's attempt to leave the network in its original state. and restore normal operation after the spoofing attack ends.

Furthermore, if we observe the pcap captured below, we can observe many items show that 192.168.2.103 is associated with the attacker's MAC (a6:1f:e5:c6:d7:89), which is a key indicator of ARP spoofing. Normally, each IP should have a unique MAC address in the ARP table, so looking at duplicate IPs indicates a potential attack. There is also no valid ARP response for 192.168.2.103 or it is immediately replaced with a spoofed response. This will cause any device that queries 192.168.2.103 to incorrectly communicate with the attacker's MAC.





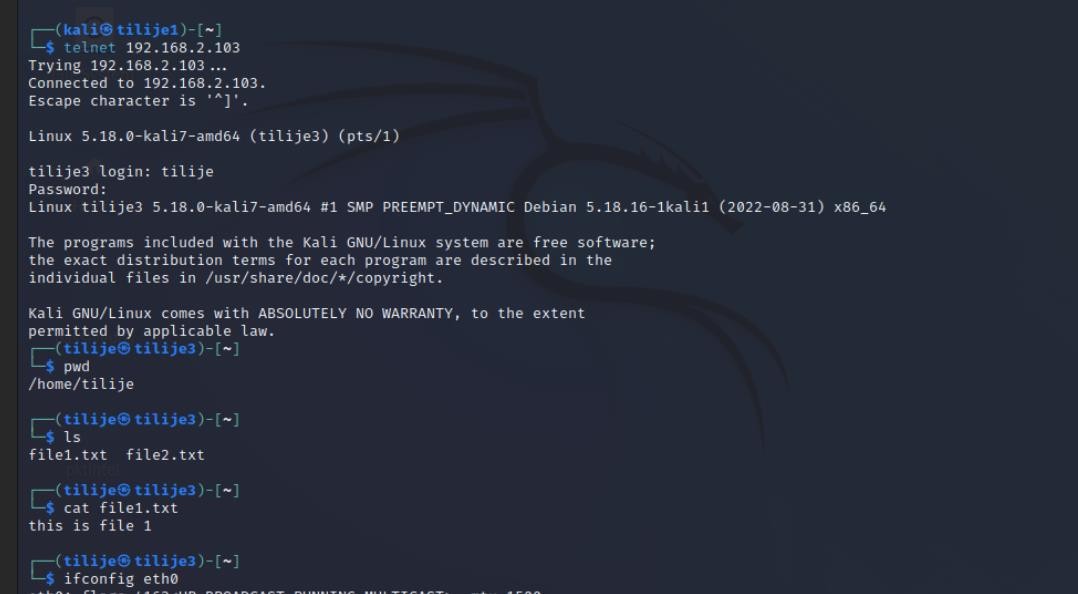
## Fig 2.6 Packet analysis

Now let us initiate a two-way ARP spoofing attack by tricking kali1 to associate kali3’s IP with the attacker’s MAC address and vice versa. This poisons the ARP tables on the victim’s machines.



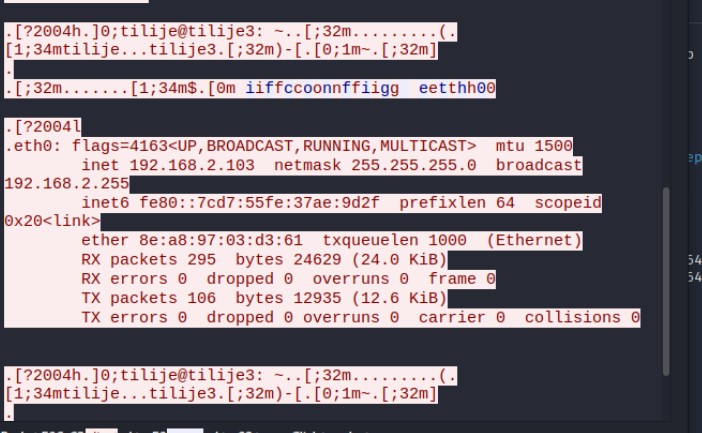
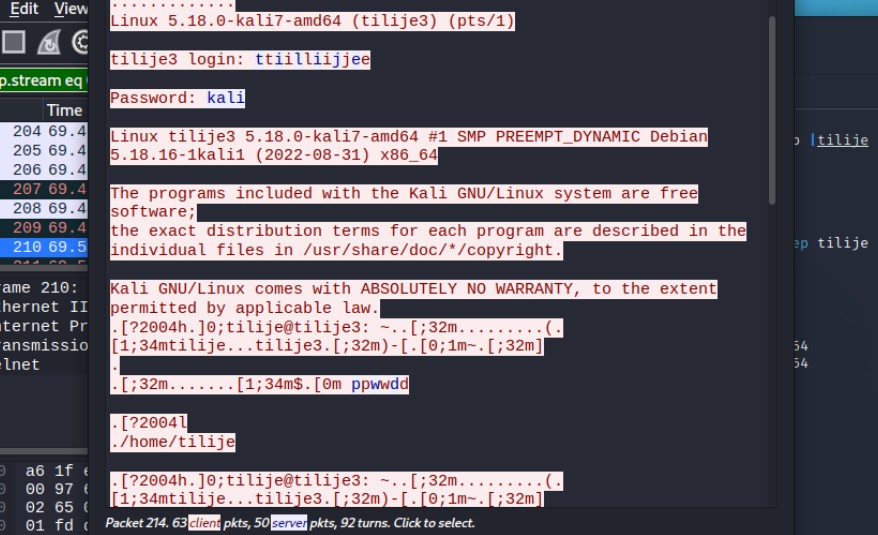
## Fig 2.7 Spoofing

Let us establish Telnet connection from kali1 to kali3. This connection will go through kali2 due to ARP poisoning, hence, allowing the attacker to capture all the telnet traffic. Also remember that Telnet transmits in plaintext, so all the information will be seen in the clear.



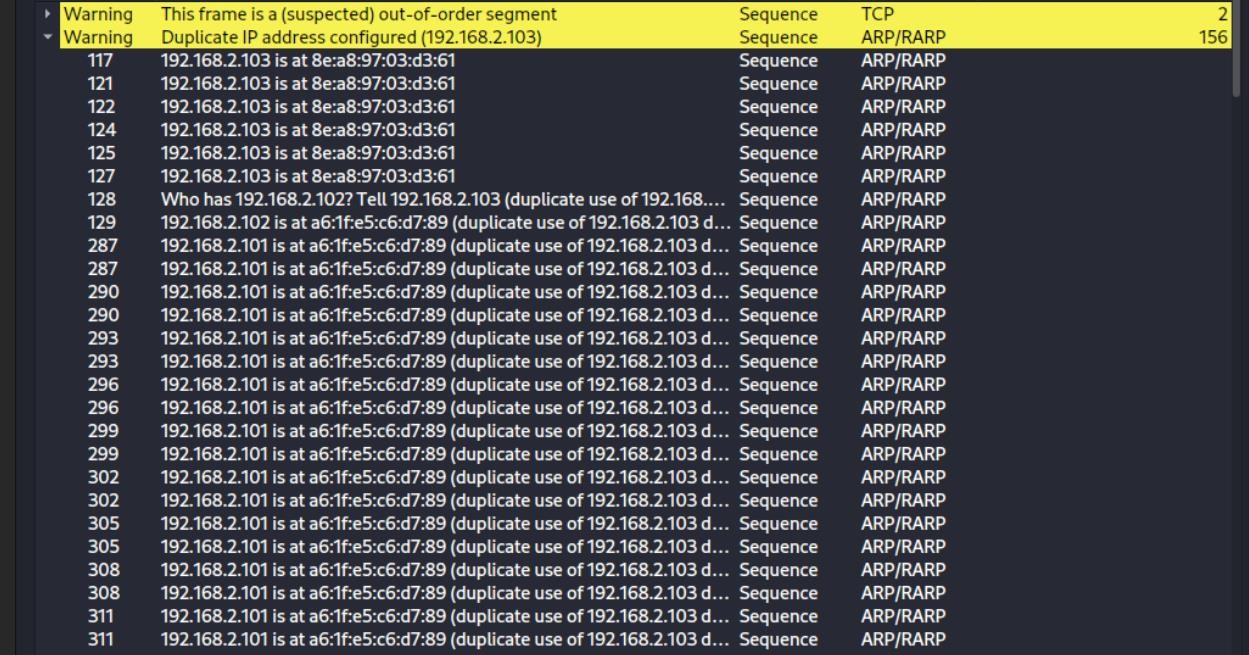
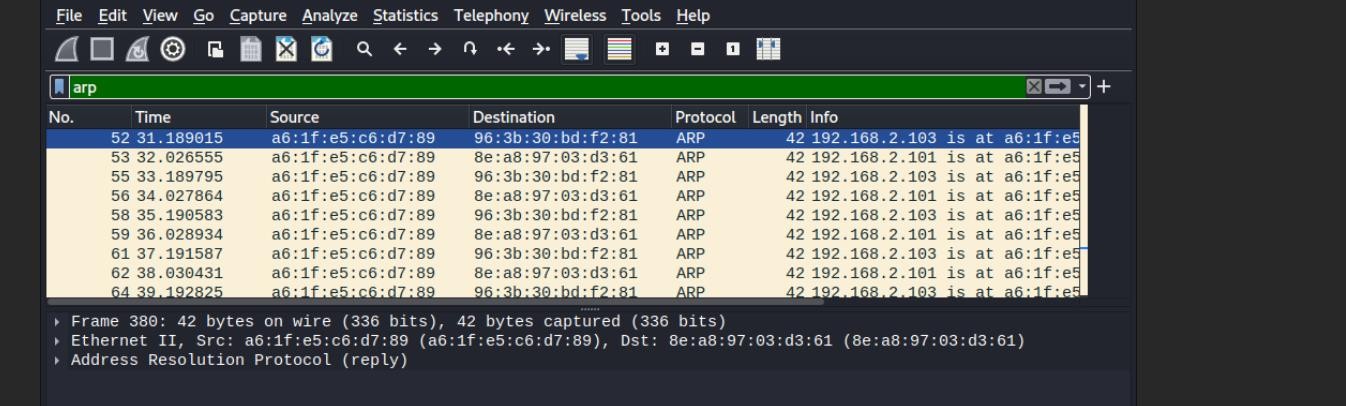
## Fig 2.8 Telnet Session

Using wireshark, we can observe the telnet session that was captured. We can see the credential exposure because it is in plaintext.



## Fig 2.9 Telnet In Wireshark

Lastly, we will use expert information to analyze how the attack happened. The expert information feature in Wireshark provides a quick overview of potential problems or important events in packet capture. It categorizes these into different severity levels, such as Notes, Warnings, and Errors. This tool is useful in identifying anomalies in network traffic, potential security issues or incorrect configuration. For security analysts, it can quickly reveal signs of network attacks such as ARP spoofing.



**Fig 2.10 Expert information**

## Explanation of Fig 2.10

1. Duplicate IP Address Configurations: The messages indicate that 192.168.2.103 and 192.168.2.101 have duplicate uses or conflicts in MAC that it is associated with, this happens when two or more devices claim the same IP address, which typically should not occur in a normal network environment. In this case, Wireshark detects that

192.168.2.103 is associated with two different MAC addresses: one being the legitimate

MAC address (e.g., 8e:a8:97:03:d3:61) and the other being the attacker’s MAC address (e.g., a6:1f:e5:c6:d7:89). Similarly, 192.168.2.101 also has duplicate MAC associations, showing that ARP replies are associating this IP with multiple MAC addresses, which indicates ARP spoofing.

1. Sequence of duplicate IP messages: The repetition of duplicate IP warnings indicates that this is not a one time anomaly ,but it is an ongoing attack. The attacker continues sending spoofed ARP replies to maintain control of the target machine's ARP cache entries. This ensures that they will continue to route traffic through the attacker's machine.
2. Out-of-Order Segment Warning: Although not directly related to ARP attacks, out-oforder blocking warnings can indicate network instability or packet delivery problems. This can reduce network performance due to potential ARP cache poisoning from ARP spoofing attacks and attacker packet delivery issues. This can happen if traffic is redirected or blocked.

# Task 3 : Port Stealing (Paper based task)

**What is port stealing?** Port stealing is a network interception technique used in switched

Ethernet networks. Where the attacker disrupts traffic intended for a legitimate host (the "victim") by changing the switch's MAC address table, unlike ARP spoofing, which changes the victim's ARP table, port stealing targets at the switch's internal MAC address table (this is also known as the CAM table). It blocks traffic at the data link layer without directly affecting the target device.

How Switching and the CAM Table Work

Ethernet switches maintain a MAC address table inside a switched network, by mapping each port to a port associated with a MAC address. This allows the switch to send packets to the desired destination port. This can be beneficial in aiding the network efficiency and limit traffic.

Each entry in the table is dynamic and can change when the MAC address appears on a different port.

Port stealing mechanism Port stealing takes advantage of the switch's habit of updating its MAC table when a MAC address appears on a new port. By periodically spoofing the victim's source and destination MAC addresses. An attacker can temporarily "hijack" the connection between the victim's MAC address and the source port. The switch mistakenly associates the victim's MAC address with the attacker's port. by sending all packets intended for the victim to the attacker.

The attack has three main components:

MAC spoofing: The attacker sends a packet with the source MAC address set to the victim's address. Forces the switch to connect the victim's MAC to the attacker's port.

Time and Frequency: Since the MAC table of the switch is dynamic, the victim’s real traffic will bring back the MAC address association to the correct port. The attacker must repeatedly spoof the victim's MAC to intercept packets.

Packet Capture: When a switch sends a packet to an attacker's port, attackers capture and analyze captured traffic using packet capture tools such as tcpdump or Wireshark.

Part (a): Detailed steps to steal ports using hping3

To steal the port An attacker can spoof packets with source and destination MAC addresses matching the victim's MAC without using advanced packet shaping tools such as Scapy. Detailed instructions on port hijacking using hping3:

First step is to identify the victim’s MAC address by using the command ‘arp -a’

Let us assume that the victim’s IP address is 192.168.2.101 and the MAC address is 00:11:22:33:44:55.

The next step will be to use hping3 to send the spoofed packets, it may not have the same MAC spoofing capabilities as scapy, but it can send packets using custom IP addresses. Here is a command we can use to send a spoofed SYN packet with victim’s MAC as both the source and destination.

**sudo hping3 --ether-src 00:11:22:33:44:55 --ether-dst**

**00:11:22:33:44:55 --spoof 192.168.2.101 -p 80 -S 192.168.1.1**

Where;

* -src and -dst sets both the source and destination MAC addresses to the victim’s MAC address.
* --spoof changes the IP source to the victim’s IP, making it look as if the packet came from the victim.
* -p sets the destination port to 80 (HTTP)
* -S this sets the SYN flag, and this is done to resemble an actual connection request to ensure the packet looks real.

After sending it, we have to maintain the attack. Since the switch will eventually bring back the MAC associated to the correct port after the victim sends traffic, the attack must loop the process periodically. This be done via bash shell scripting, using the command;

**while true; do**

**sudo hping3 --ether-src 00:11:22:33:44:55 --ether-dst 00:11:22:33:44:55 --spoof 192.168.2.101 -p 80 -S 192.168.1.1 sleep 2 done**

The last step will be to capture the packets for analysis via tcpdump, this can also be done in separate tab before you send any of the spoofed packets, so as to analyze any sensitive data.

## Part (b): Appraisal of Port Stealing

**Stealthier than ARP Spoofing:** Because it doesn't directly alter the victim's ARP cache Port stealing can therefore bypass some ARP spoofing detection mechanisms. Network monitoring tools usually focus on detecting ARP anomalies, so port stealing goes unnoticed.

**No direct impact on the victim's system:** Port stealing does not require any changes with the victim's machine unlike ARP spoofing. The attack focuses on the entire switch. This makes the victim less likely to notice any changes.

**Useful in VLAN environments:** Port stealing can work within a VLAN, allowing attackers to capture traffic on hosts in the same VLAN that might otherwise be isolated by a switch.

## Limitations of Port Stealing

**Temporary control:** The CAM table on the switch is dynamic. Therefore, the victim's legitimate traffic eventually recovers its MAC port. This requires continuous spoofing, which makes the risk of detection higher.

**High traffic noise:** Repeated spoofing causes network noise. If the refresh rate of the switch's CAM table is high, the attacker will have to send a continuous number of spoofed packets, which can be detected by intrusion detection systems (IDS).

**Restricted to local VLANs:** Port stealing only works within a single VLAN or broadcast domain. Because it requires a Layer 2 MAC address table, this is ineffective for blocking traffic across VLANs or subnets.

**Switch security mechanisms:** Many modern switches have security features such as port security, Dynamic ARP Inspection (DAI), and MAC flood mitigation, which can block or mitigate port theft attempts, for example, port security allows only a limited number of MAC addresses on each port and can restrict unknown MAC address. This prevents attackers from spoofing the victim's MAC address on a different port.

## Advanced Countermeasures Against Port Stealing

**Port security:** Managed switches can limit the number of MAC addresses associated with each port. If an attacker's MAC address exceeds this limit, the switch can disable the port. notify the administrator or prevent new MAC connections

**MAC Address Aging Timer**: Many enterprise switches can be configured to have longer aging for known MAC addresses, which means the switch is less likely to connect a valid MAC address to another port.

**Dynamic ARP Inspection (DAI):** DAI inspects ARP packets on the network and prevents unwanted MAC table updates, this can indirectly prevent port stealing. This is because the switch is less likely to change its MAC table based on packets from unauthorized sources.

**Network Monitoring and Logging:** Network administrators can check for unusual MAC repeat patterns, for example, if the MAC address changes ports frequently in a matter of seconds. It may indicate a port-stealing attack or any attempt to flood the MAC.

# Task 4 : Use Wireshark to answer this task

(a) Use ExampleA.pcap

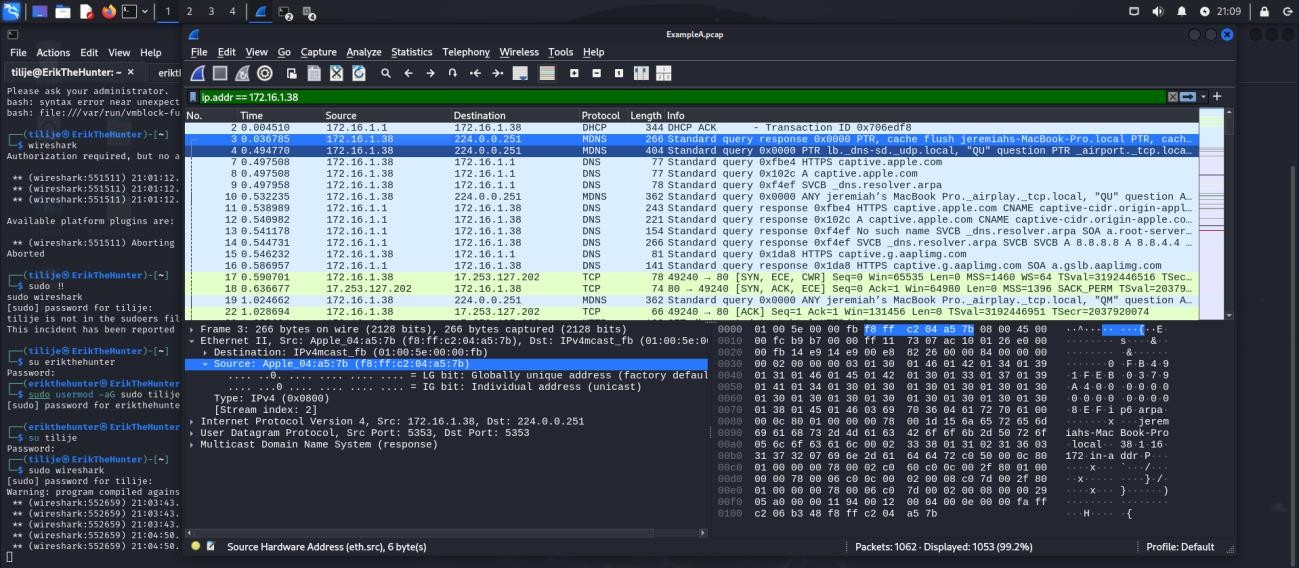
What is the MAC address of the machine with IP address 172.16.1.38 ?

Use DHCP to find the hostname that is using IP address 172.16.1.38

To answer this, we open the pcap file and apply a filter that displays all the packets invovling IP 12.1.38

We use **ip.addr == 172.16.1.38** in this case and it looks at the ethernet layer in any packet where 172.16.1.38 is the source or destination.

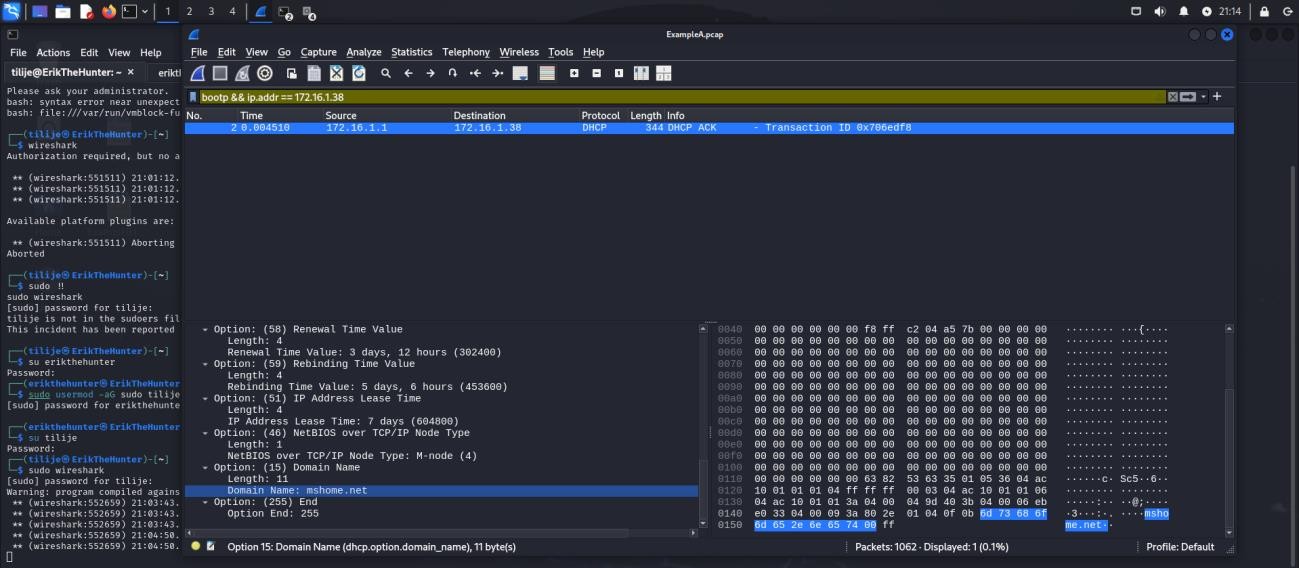
From the screenshot below you can see that the MAC address of the machine with IP 172.16.1.38 is **f8:ff:c2:04:a5:7b**.



**Fig 4.1 MAC Address of 172.16.1.38**

To use DHCP to find the hostname associated with 172.16.1.38 . we use the filter **bootp && ip.addr == 172.16.1.38.** “bootp” displays packets associated with DHCP. In this case, the DHCP traffic look for the DHCP ACK or REQUEST Packet. This is where items like “Hostname” or “Domain name” will be seen in the packets.

From the screenshot below, The hostname associated with IP 172.16.1.38 is **mshome.net**.



## Fig 4.2 Hostname of 172.16.1.38

(b) Use ExampleB.pcap

The Domain is www.pcapworkshop.net and the Domain Controller is at 172.16.1.16

The Domain controller hostname is PCAPWORKSHOP-DC

What user account name is being used in traffic between the domain controller at 172.16.1.16 and

the Windows client at 172.16.1.141 Hint:

Filter on kerberos.CNameString.

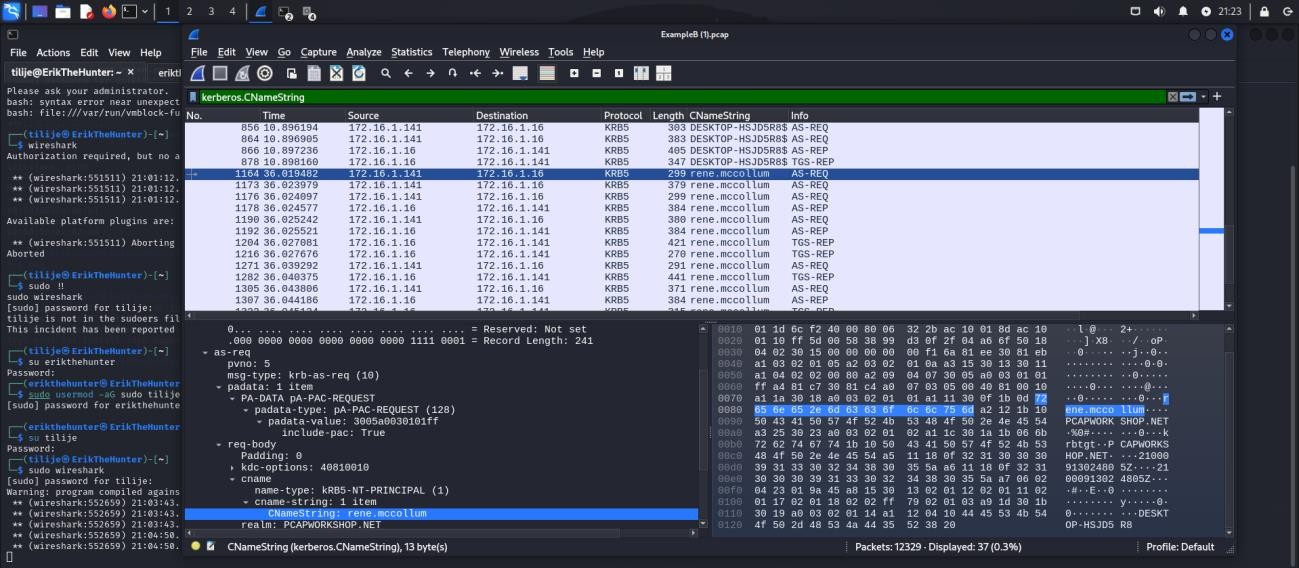
Select the first frame.

Go to the frame details section and expand the lines.

Select the line with CNameString: desktop-hsjd5r8$ and apply it as a column

Using the hint above we apply kerberos.CNameString. We select the frame details section and expand the keberos protocol section to find the CNameString. This field shows the username or cilent involved in the Keberos Authentcation Service Request.

The user account name involved in Kerberos traffic between the Domain Controller (172.16.1.16) and the client (172.16.1.141) is **rene.mccollum**.



**Fig 4.3 User account name.**

## TASK 5: Simple iptables rules

**(a) On one of the Kalis, write an iptables rule to silently drop TCP PUSH packets from another Kali.**

**Write a rule to first LOG the packet using the string “Dropped PUSH Packet by John”**

**(and change John to your name).**

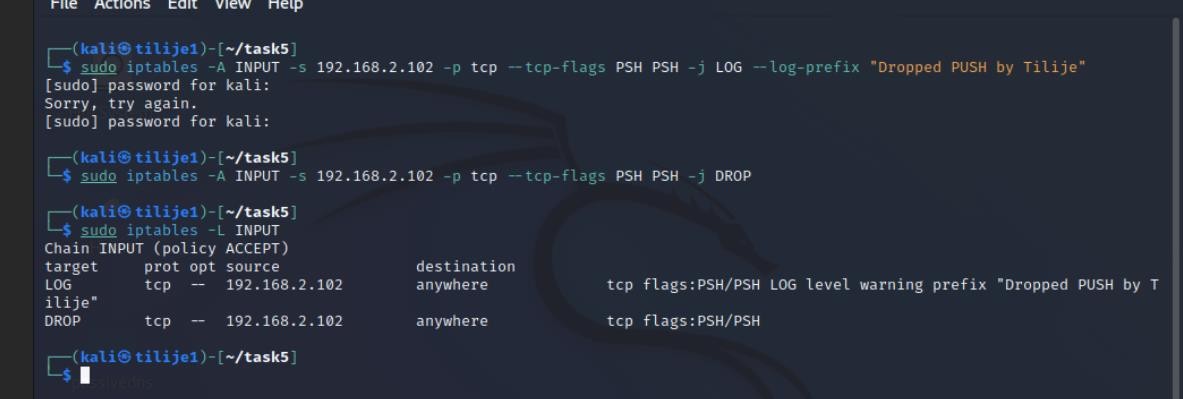
**Test your rule by setting up a netcat listener on Kali, and connecting to it via netcat from the other Kali, and sending some strings across.**

The first step is to writes by iptables rules which are;

1. sudo iptables -A INPUT -s 192.168.2.102 -p tcp --tcp-flags PSH PSH -j LOG --log-prefix "Dropped PUSH by Tilije"
2. sudo iptables -A INPUT -s 192.168.2.102 -p tcp --tcp-flags PSH PSH -j DROP

The first command appends (-A) a rule to the INPUT chain for the local device. It specifies the source IP address (192.168.2.102) and fits just the TCP packets (-p tcp) where the PSH flag is set (--tcp-flags PSH PSH). When such packets are encountered, they are logged (-j LOG) with a custom prefix "Dropped PUSH by means of Tilije". This permits the administrator to see entries related to these packets in device logs (e.g /var/log/messages).

The second command adds another rule to the input chain. The matching criteria are the same as the first order. Instead, it uses the DROP (-j DROP) target, which means that any TCP packets with the PSH flag from the specified source are silently discarded. No response to sender. It effectively ignores these packets.



Now, we will test the rule by starting a netcat session and we use kali2 to connect to it.

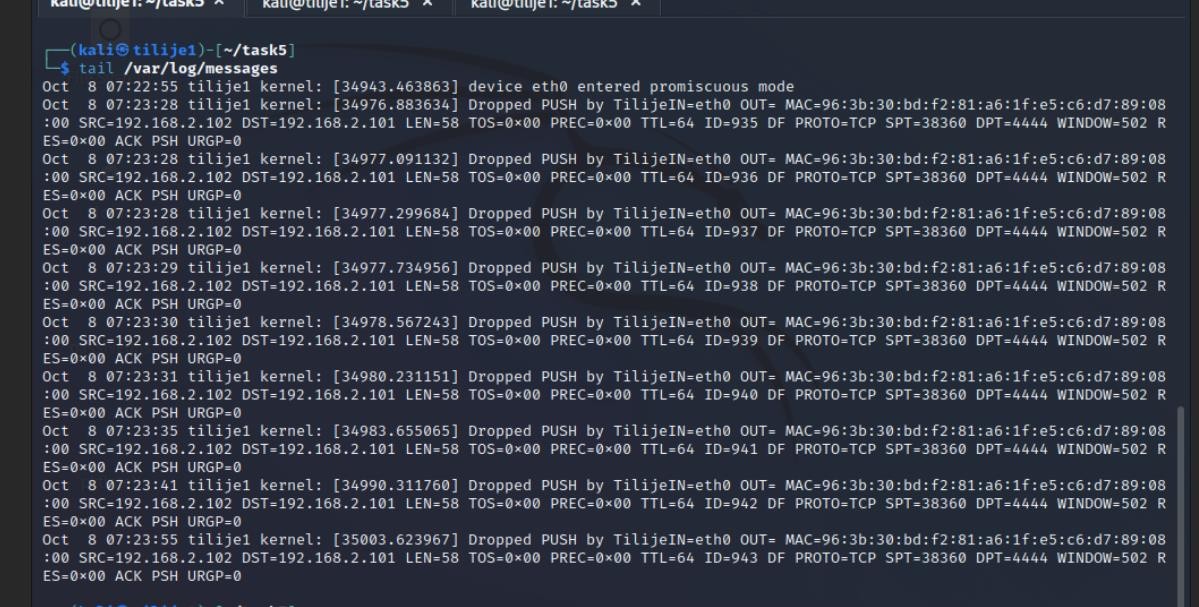


**Fig 5.1 Netcat Session.**

Now we check last lines of the log messages. The screenshot below shows a list of logs from **/var/log/messages** that shows TCP packets dropped when flagged by iptables rules. The first line logs the network interface (eth0) into promiscuous mode. which is required for packet sniffing This setting allows the interface to log all traffic on the network segment and not just the packets that are needed. This mode is usually enabled when using tools like tcpdump or for certain network applications.

Subsequent lines in the log show entries related to TCP packets that were blocked and logged by iptables rules. Each log entry contains details about the packet, such as:

* Timestamp: For example, Oct 8, 07:23:28 indicates when the packet was processed.
* Host: tilije1 Specifies the machine that records packets.
* Kernel message: The kernel log indicates that the packet was dropped. The "Dropped PUSH by Tilije" prefix marks these entries as the result of specific iptables logging rules.
* Network interface: The interface involved in data transmission. which is annotated as eth0 MAC Address: For example, MAC=96:3b:30:bd:f2:81 (source) and a6:1f:e5:c6:d7:89 (destination) show the Ethernet address of the communication device.
* Source and destination IP: e.g. SRC=192.168.2.102 and DST=192.168.2.101 Specifies the source and destination of the packet.
* Protocol: PROTO=TCP Specifies the protocol used. Port: SPT=38360 and DPT=4444 specify the source and destination ports.
* Flags: ACK The PSH flag indicates that a TCP packet has the PUSH flag set, signaling that the data in the packet should be immediately pushed to the receiving machine.
* Other packet metadata: Fields such as TOS=0x00 (Type of Service), TTL=64 (Time To Live), ID=936 (Packet Identifier) further describe the nature of the packet.

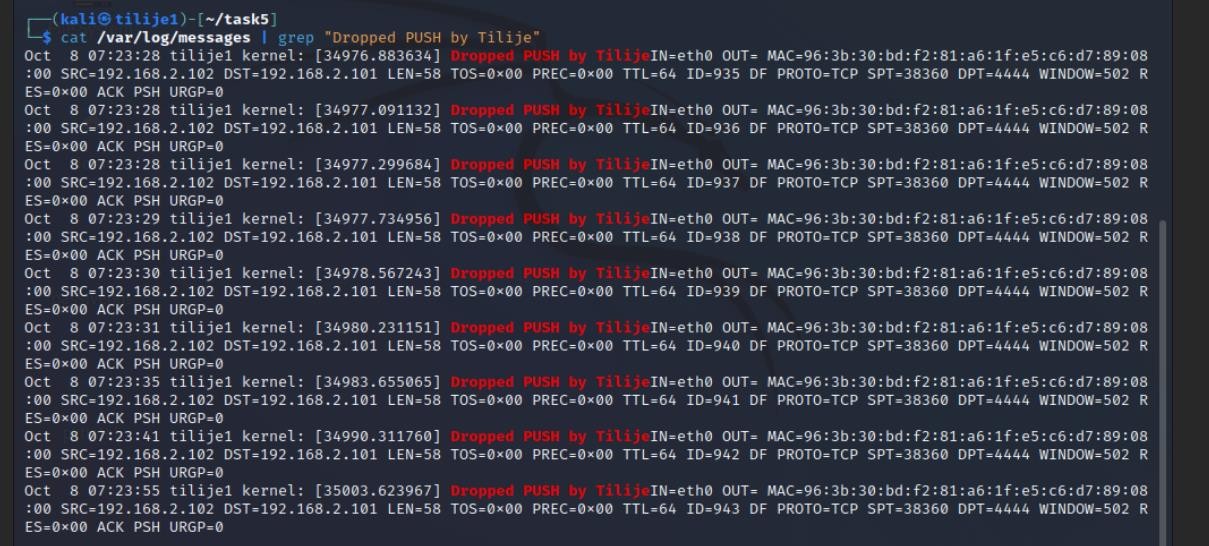


**Fig 5.2 Tail of Log messages.**

Now we filter our search by ‘Dropped by Tilije’ and we can see that the output shows successful execution of iptables rules. The rule saves packets that meet the specified criteria before discarding them, making sure that it does not reach its intended destination. The presence of these entries confirms that traffic with the TCP PSH flag originates from 192.168.2.102 and it is set for port 4444 on 192.168.2.101 and it was blocked and managed according to configured rules. The DROP operation with logging, allows for traffic control while making an audit trail for blocked packets.

The list that came out is the result of a test scenario where Netcat (nc) is used to send data from one host to another. A listener was set up on the destination host (192.168.2.101), while

(192.168.2.102) acts as the client. The firewall is configured to block filters for such packets that are captured and dropped before they reach their target. This demonstrates the importance of iptables in monitoring and controlling specific types of network traffic. The detailed log shows the usefulness of this in monitoring and troubleshooting, because they provide all the necessary information about blocked packets. This helps ensure that the firewall doesn't just enforce security policies, but it also records firewall activity for future reference or analysis.



**Fig 5.3 Filter of Log messages.**

The command in the screenshot below counts the number of lines in the system log of /var/log/messages.

Output 9 indicates that there are nine entries in the log file containing this specific string.

The reason it is nine is because the sequence of packets matches with the iptable rules created. The iptables rules are designed to:

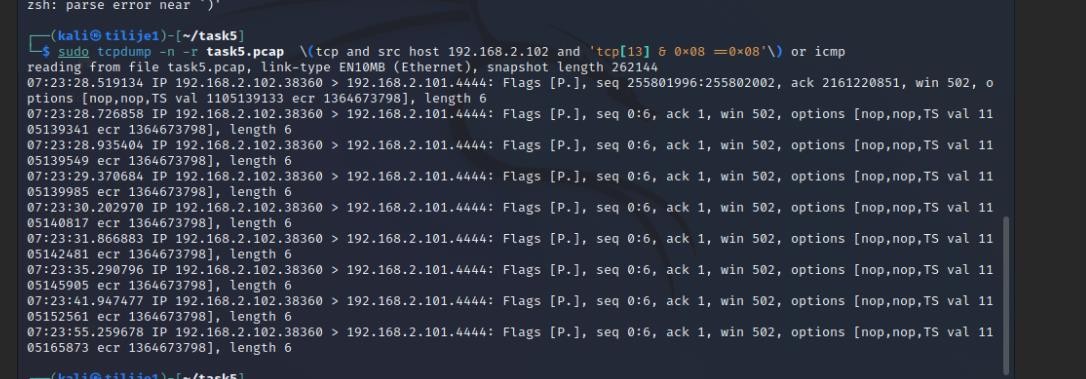
* Record and drop TCP packets: specifically from source IP 192.168.2.102 to destination IP 192.168.2.101 with the PUSH (PSH) flag set on port 4444.
* Create a log for each packet: whenever a packet matches a rule, the kernel saves the packet to /var/log/messages With a custom prefix of "Dropped PUSH by Tilije"

In this case, nine packets are captured and processed by the iptables rule. Each matching packet receives a log entry. The grep command filters the log file for lines containing "Dropped PUSH by Tilije" and wc -l specifies the number of those lines.



**Fig 5.4 Filter of Log messages with number of lines.**

Now finally, we analyze the packets that we captured while the netcat session was going on.



### Fig 5.5 Packet Analysis

The output in Fig 5.5 lists each matching packet. It shows the source IP address 192.168.2.102 and destination IP address 192.168.2.101 that TCP sends over port 4444. Other TCP header details in the fields shown in the output include the sequence number and the acknowledgment number. Each packet in the capture shows an increasing sequence number as expected for TCP communications. However, the acknowledgment number is continually set to 1.

The reason the acknowledgment number (ACK) remains 1 throughout the output is because, In TCP communications, the ACK field acknowledges the receipt of the last packet sent by the other machine. In this case, the destination device at 192.168.2.101 sent an initial packet with sequence number 0, and the source at 192.168.2.102 still acknowledges receipt of that first packet.

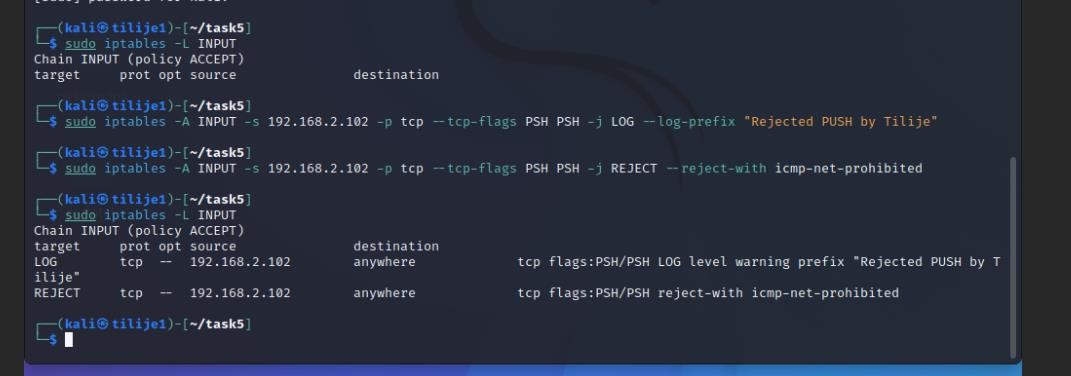
Since there is no additional information sent by the destination to 192.168.2.102, therefore the acknowledgment number does not change. Where the source device primarily sends PSH flagged packets to the destination. and the ACK field remains constant due to the lack of two way data exchange. This type of analysis is useful for confirming the behavior of TCP flags and understanding how specific hosts communicate within the network.

**(b) Then, change your rule so that, instead of silently dropping the TCP PUSH packet, it replies to**

**a PUSH with an ICMP Network Administratively Prohibited packet. Change the LOG message**

**also, to have the string “Rejected PUSH by John” (and change John to your name). Then set up the netcat session again.**

The same steps used in question a will be used here but with a different log which can be seen in the screenshot below;



### Fig 5.6 New Rules

The command **sudo iptables -A INPUT -s 192.168.2.102 -p tcp -tcp-flags PSH PSH -j LOG --log-prefix "Rejected PUSH by**

**Tilije"** logs all incoming TCP packets with the PSH flag from the source IP

192.168.2.102. The log entry prefixed with "Rejected PUSH by Tilije" is used for auditing purposes and checks in the system log before these packets are rejected.

The command **sudo iptables -A INPUT -s 192.168.2.102 -p tcp -tcp-flags PSH PSH -j REJECT --reject-with icmp-net-prohibited**

rejects TCP packets with PSH flag from source IP 192.168.2.102 and sends an ICMP net prohibited message to the sender. This clearly indicates to the sender that the connection is not allowed. It distinguishes it from silent packets.

After initiating the nc session and sending a message, we check the logs with the filtter shown below



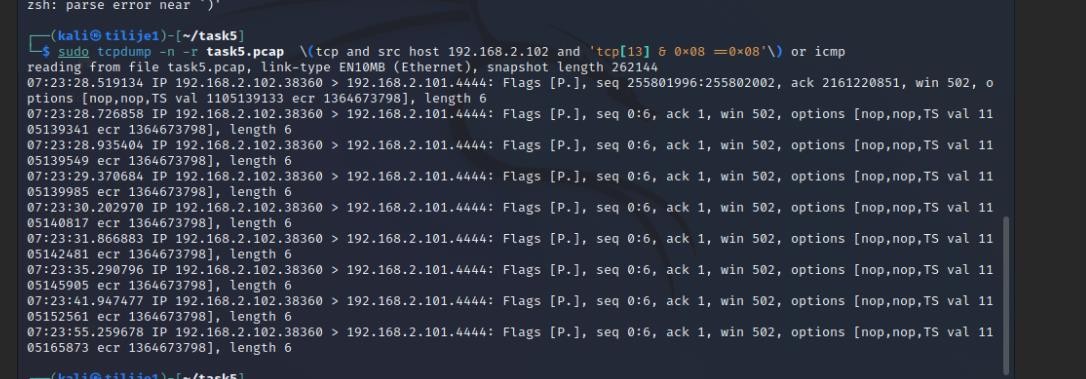
### Fig 5.7 Log Filter

The main point in the screenshot is to demonstrate how iptables rules effectively log and reject "PSH" flagged packets. The logs confirm that specific TCP packets from

`192.168.2.102` to `192.168.2.101` are blocked on port `4444`. The `grep` command filters the logs to show only those preceded by `"Rejected PUSH by Tilije. "` and the command `wc -l` will count all the times it was rejected.

The number 7 indicates that the specific iptables rule rejected seven TCP packets with the PSH flag during the capturing period.

Now we analyze the packet captured by tcpdump.



### Fig 5.7 Packets captured

The command uses a specific filter that checks for the presence of the PSH flag in the 13th byte of the TCP header (tcp[13] & 0x08 == 0x08) and captures all such packets.

The importance of this is to confirm that packets matching this condition have been captured and saved. Each line in the output corresponds to an analyzed TCP packet. All packets have an ACK flag of 1, indicating they are part of an established connection. Meanwhile, the PSH flag specifies that the data in the packet should be pushed to the client application without buffering. This corresponds to the rules set in the iptables configuration.

This analysis directly supports the previous case where iptables rules are rejected or dropped with the PSH flag. By capturing these packets, the task examines the activity recorded by iptables and confirms the conditions under the rules.

**(c) NAT**

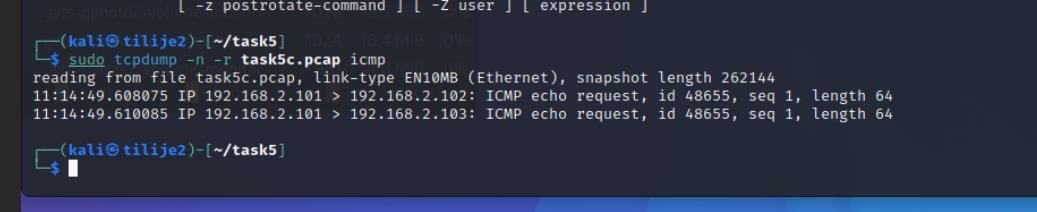
**Use iptables on Kali2 to forward PING packets coming from Kali1 into Kali2 on to Kali3.**

**You may need to turn IP\_FORWARDING on in Kali2. To do that, you may have to become root at the terminal via the command ‘sudo su’. The replies from Kali3 should make their way back to Kali1. (This is not simple.) Capture the network traffic using tcpdump on Kali2 and on Kali3, and analyse it to see what is happening.**

The first step will be to turn on ip forwarding, which can be seen in the screenshot below, the iptables rule used blocks ICMP echo requests from 192.168.2.101 and redirect to 192.168.2.103 This efficiently routes traffic to another destination. It simulates packet redirection or spoofing.



### Fig 5.8 Commands used



### Fig 5.9 pcap results

This result confirms that the previously configured DNAT rules work as intended. The initial ICMP request targeting 192.168.2.102 is blocked and changed tosend as 192.168.2.103.

The redirection is clear because now 192.168.2.101 It appears to be pinging 192.168.2.103. Even if the original goal is different. This proves the efficiency of packet handling using iptables.

**(d) Write your own chain**

**Set up your own chain to deal with TCP packets that have ‘bad flags’.**

**In this chain Log packets that have bad TCP flags at a rate of 20 per minute.**

**Let those packets be**

1. **packets with no flags set,**
2. **packets with all flags set,**
3. **packets with SYN and FIN only and**
4. **packets with SYN, PUSH and RST and possibly others.**

**Use hping to test by sending such packets at a rate of 30 per minute.**

In this task, the goal is to identify and handle TCP packets with "bad flags," which typically indicate malformed or potentially malicious packets. The steps involve defining rules using iptables to catch these packets, log them, and ultimately drop them**.**

**Commands and Explanation:**

* 1. **sudo iptables -A INPUT -p tcp --tcp-flags ALL NONE -j badflags**

This command adds a rule to the INPUT chain to filter out TCP packets that do not have the flag set (NULL packets). These packets are unusual and may indicate scanning or malicious activity**.**

* 1. **sudo iptables -A INPUT -p tcp --tcp-flags ALL ALL -j badflags**

This filters TCP packets with all flags set. This is not a valid state for TCP packets and is often associated with network attacks.

* 1. **sudo iptables -A INPUT -p tcp --tcp-flags ALL SYN,FIN -j badflags**

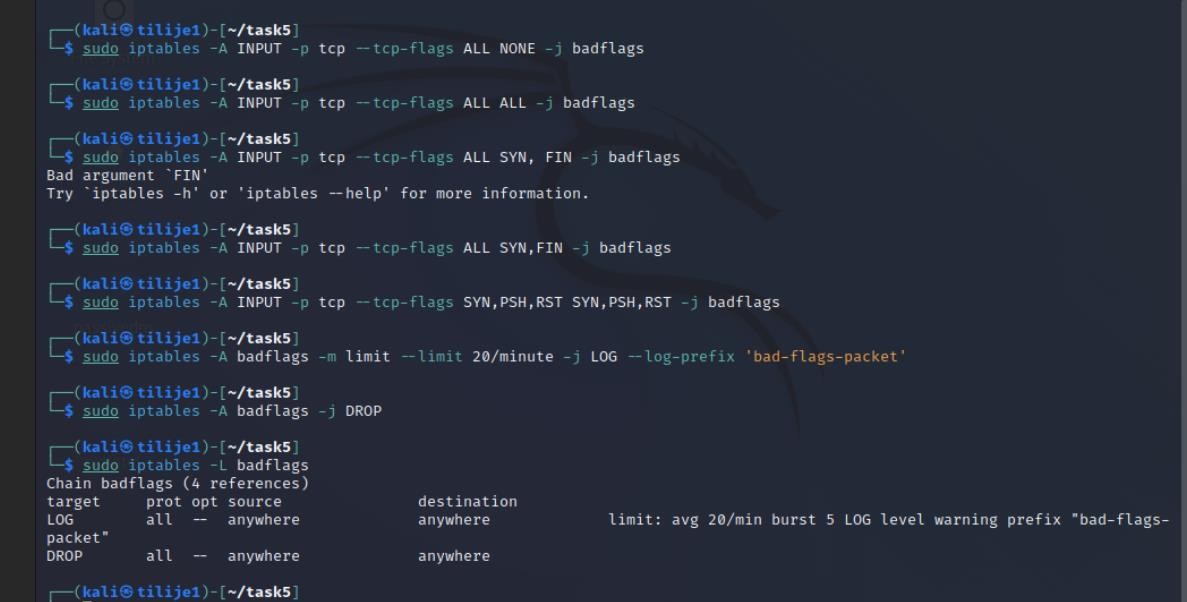
This filters packets by setting the SYN and FIN flags. Such connections are invalid and may indicate malicious searches.

* 1. **sudo iptables -A INPUT -p tcp --tcp-flags SYN,PSH,RST SYN,PSH,RST -j badflags**

This adds logging rules to the badflags chain and log packets matching the above rules. It limits recordings to 20 recordings per minute to prevent excessive logging

* 1. **sudo iptables -A badflags -j DROP**

This drops any packets that match the badflags criteria after logging them.



After we have set the rules, we go to another kali machine to send some commands.

These commands show which iptables rules are configured on the machine and how (192.168.2.101) responds to specific packet types. The previous rules targeting different TCP flag connections effectively identify and block malicious traffic patterns. This test verifies the validity of the security measures implemented on the target machine. This ensures that packets containing suspicious flag combinations do not disrupt normal operations or create vulnerabilities.

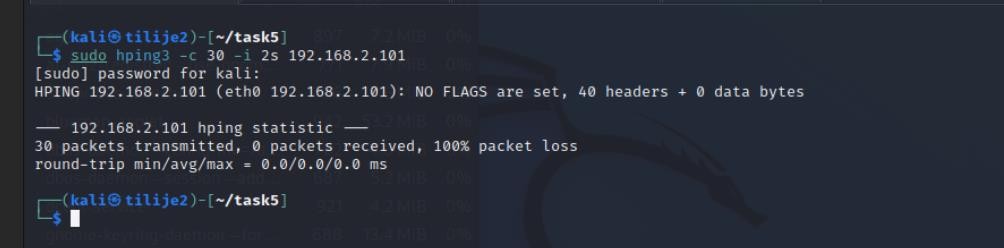
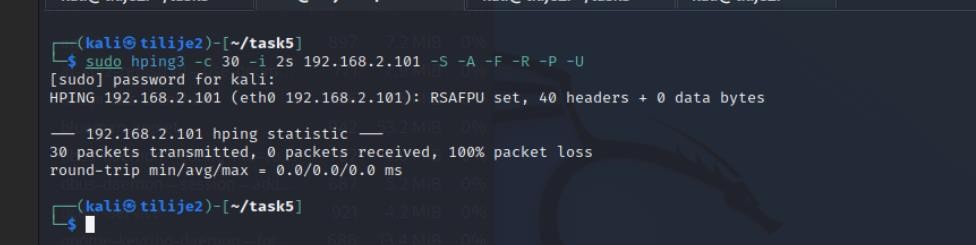
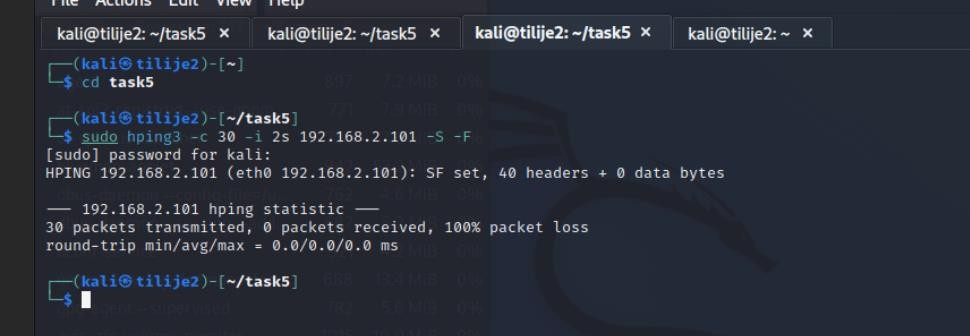
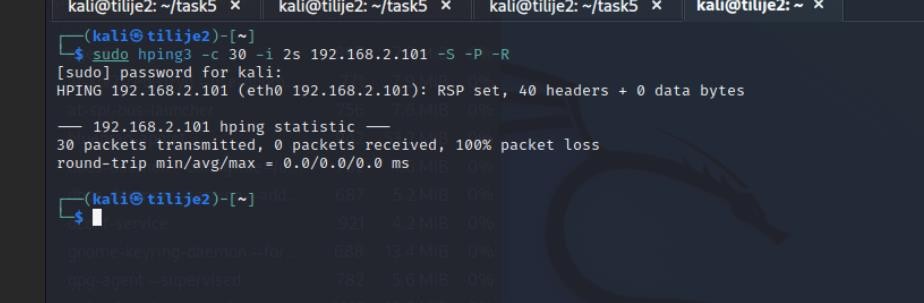


Fig 5.10 Commands sent to kali1

Now we view the logs and these are a few things we have notice from it;

* + **Logged Packets:** Each entry in the log shows packets that were flagged because they met criteria. There are "bad flags" defined in iptables rules. These flags may contain unusual combinations, such as SYN, RST, or PSH, which are not normally present together in a legit traffic.
  + **Source and Destination**: The source IP (192.168.2.102) and destination IP

(192.168.2.101) remain consistent across all records. This indicates that 192.168.2.102 is sending these flagged packets to 192.168.2.101.

* + **Dropped Packets**: The presence of these entries helps verify that iptables rules work as intended by detecting and logging packets with unusual flag combinations. This mechanism ensures that such packets do not harm the network or exploit potential vulnerabilities.
  + **Packet Count**: The wc -l command counts 38 entries, which indicates that 38 packets were recorded by "bad flag." This corresponds to crafted traffic sent in a previous commands using hping3.

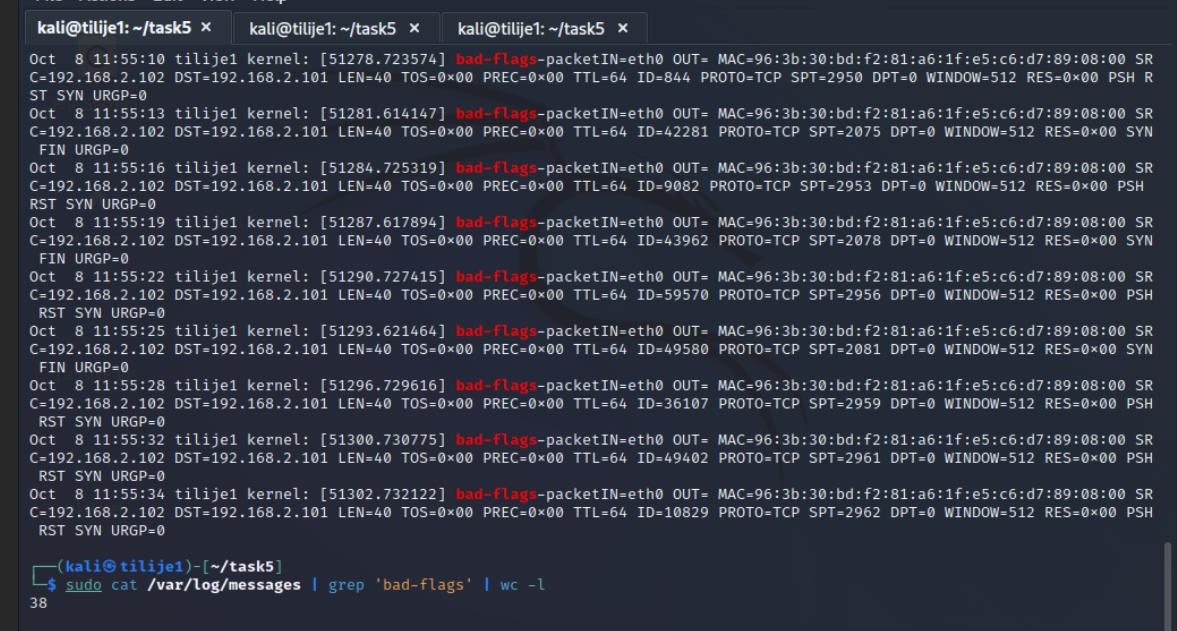


Fig 5.11 Viewing the logs.

# Task 6: Rate Limiting using iptables

(a) **Rate Limiting using -m limit**

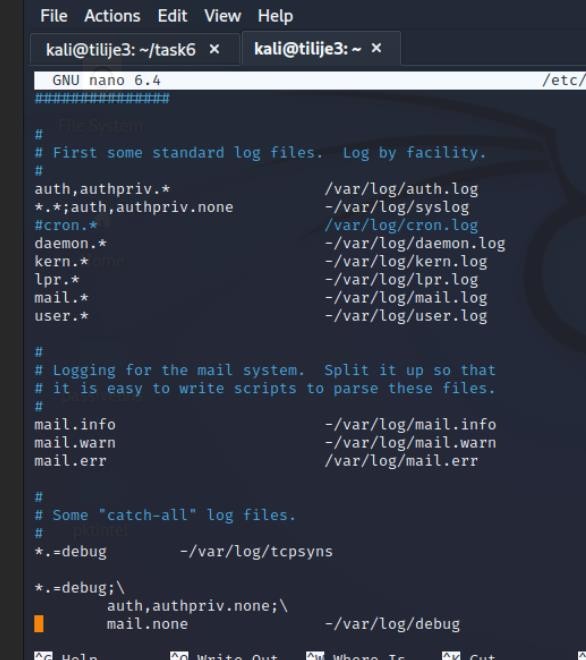
Write iptables rules involving **-m limit** to allow only 5 TCP SYNs per minute to ports 80 or 443.

Then, use another host to send 10 SYNs per minute to these ports.

Write your iptables rule(s) in such a way that the extra packets are logged in the file /**var/log/tcpsyns** with a suitable message, and dropped.

As part of this, look up the “limit burst” and add this to your **-m limit** rule (use man iptables).

First thing was to make sure the packets are logged in /var/log/tcpsysns , this is done by editing the rsyslog.conf file. It ensures that logs related to TCP SYN packets that exceed the rate limit are stored there.



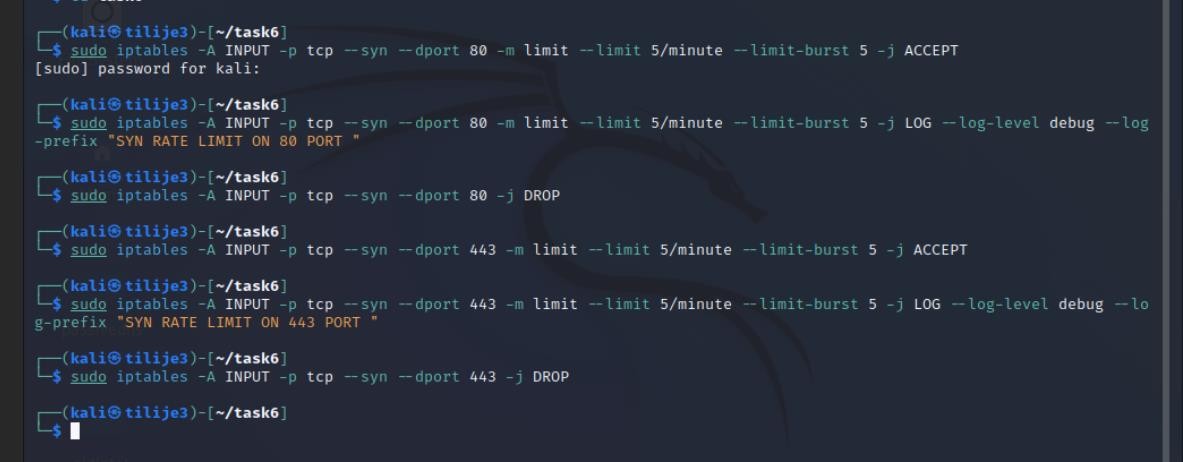
This task focuses on rate limiting for SYN packets sent to specific ports (80 or 443). The various commands in the screenshot below implements rate-limiting rules to control the number of SYN packets arriving per minute. This ensures that the firewall is resilient to SYN flood attacks, helping to protect against denial of service (DoS) attempts that target these ports.

The first `iptables` command adds a rule to accept incoming TCP SYN packets on port 80, but at a maximum rate of 5 packets per minute, with only 5 packets allowed to be sent (burst). This ensures that valid traffic is allowed while excessive SYN requests are sent

The second command records any SYN packets on port 80 that exceed the configured rate limit. By using the option `--log-prefix` to tag logs with "SYN RATE LIMIT ON 80 PORT," it provides visibility into potential malicious activity targeting the port.

The third command drops any SYN packets on port 80 that exceed the rate limit. This prevents such packets from using resources on the server or affecting legitimate connections.

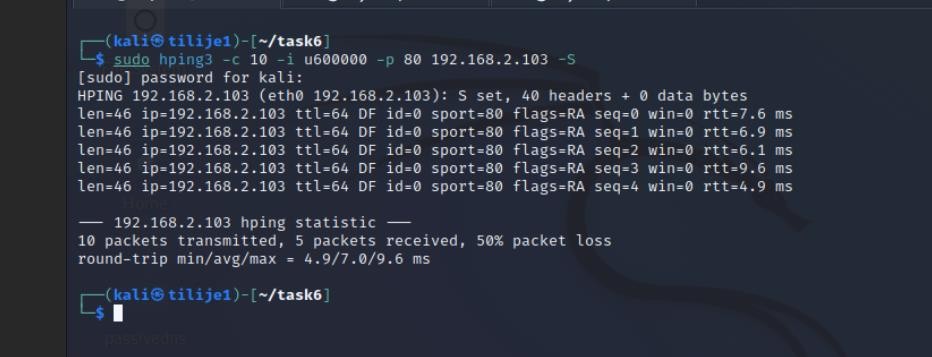
This process is repeated for port 443, where the fourth and fifth commands are the same used in port 80.



## Fig 6.1 LIMIT RULES

After that is done, we go to another kali to test the rate-limiting rules done in kali3, for this case we are using just port 80. The command send 10 TCP SYNS to port 80 at a rate of 0.6 milliseconds. This tries to simulates a flood of SYN packets to analyze how the rule responds.

The results indicate that 10 packets were sent, but only 5 were received, resulting in 50% packet loss. This loss occurs because the rate limiting rules in the firewall only allow up to 5 SYN packets per minute where 5 packet bursts cause extra packets to exceed the limit.

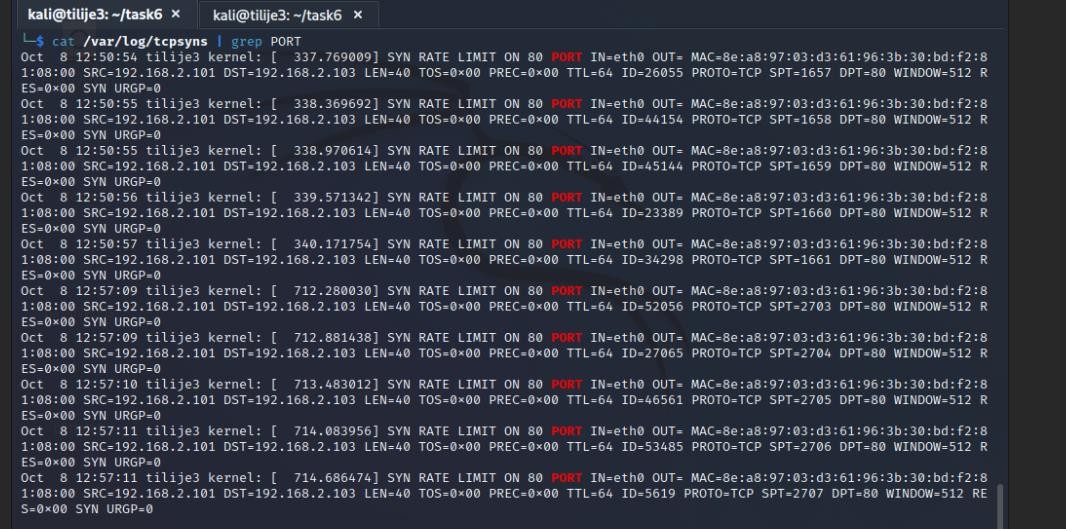


## Fig 6.2 Packets sent

Now we view the logs in tcpsyns and we can see the log below shows repeated attempts from source IP address 192.168.2.101 to destination 192.168.2.103 on TCP port 80. The SYN RATE LIMIT ON 80 PORT prefix verifies that these packets were marked and processed by the rate limit rule. Source MAC address , TTL, ID and other details such as packet level details (e.g.

SYN URGP=0) confirms the structure of blocked packets.

This behavior is exactly the intended purpose of the rate limiting rule, to reduce potential SYN flooding attacks by denying or dropping additional requests. and allows a limited number of connection attempts during a given period of time.



## Fig 6.3 Viewing the logs

(b) **Rate Limiting using hitcount/recent**

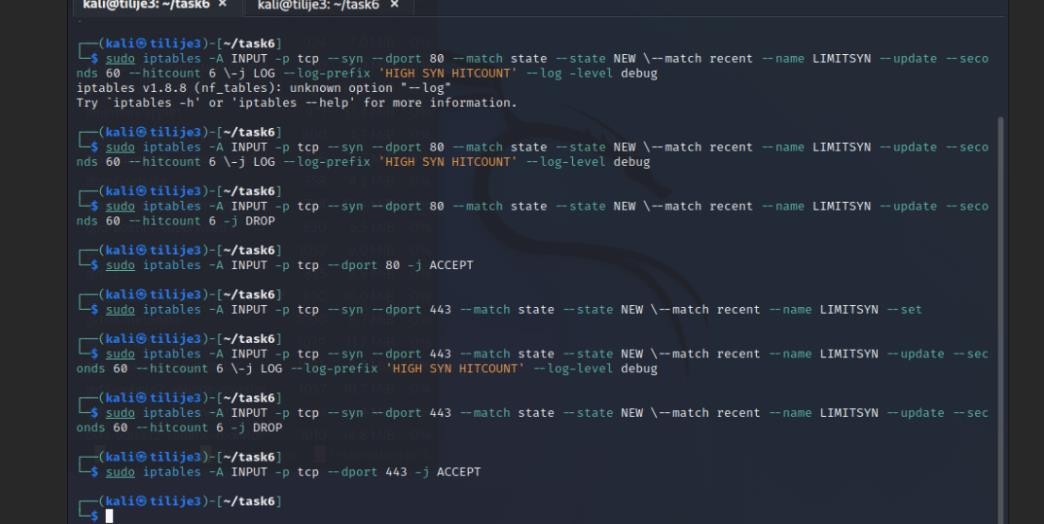
On kal1, use the ‘hitcount/recent’ approach to allow only 5 TCP SYNs per minute to ports 80 or

443 from a single IP address.

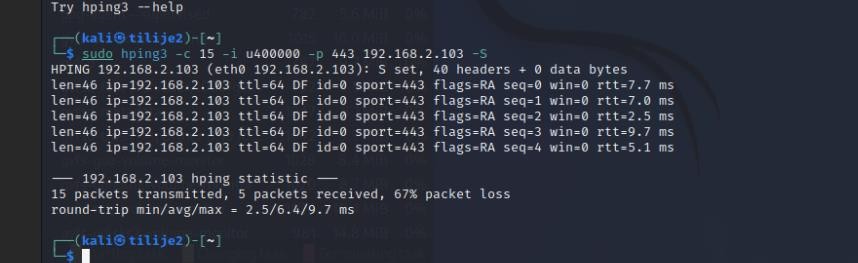
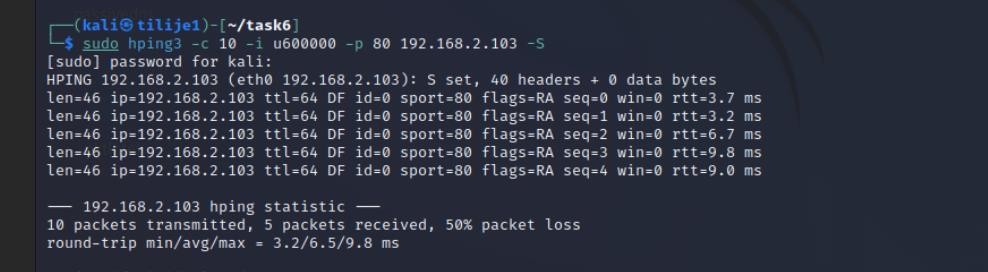
Then, use another host, say kali2, to send 10 SYNs per minute to these ports.

Simultaneously, send 15 SYNs per minute from a third host, say Kali3.

First thing was to input the rules in the commands in the figure below

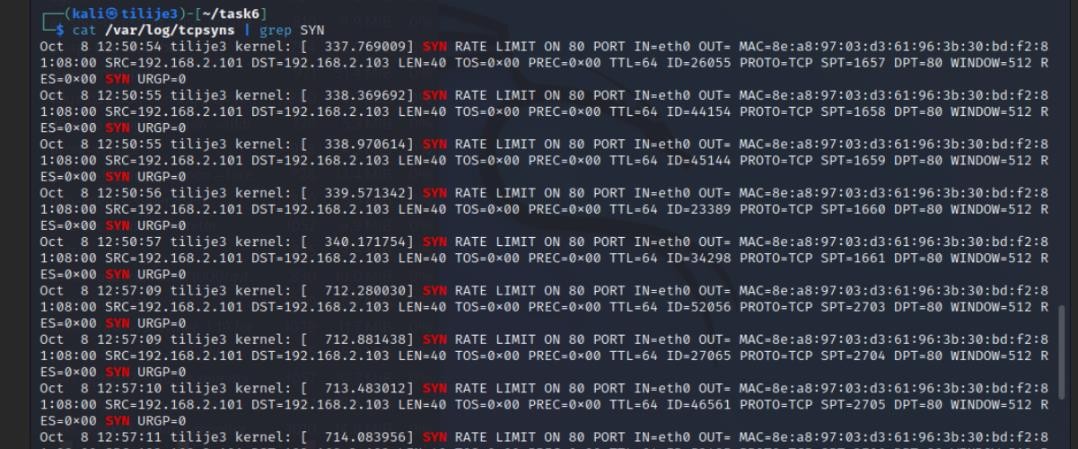


Now we go to another kali to send packets, and we can see that in kali3 only 5 packets were received, leading to a 67% loss. This shows the rate limiting rule configured on the iptables is effective, same thing applies to the 50% loss on kali1



**Fig 6.4 Sending packets from kali1 and 3.**

Now, checking the log file, we can observe the effectiveness of the rate limiting mechanism configured in the iptables to prevent SYN flood attacks targeting port 80. The log shows the number of SYN packets originating from IP 192.168.2.101 and forwarded to 192.168.2.103, these were identified and managed, each providing important details, including timestamp, the origin and destination of the packet. These logs verify that the configured rules are working correctly by enforcing the set limit of 60 SYN packets per minute, with 6 bursts allowed per packet.



# Task 7: Text Based

Look up one of the NGFW products and compare it with a traditional Application and Packet

Filter firewall.

Write approximately 2-4 pages of text.

**Comparison of Palo Alto Networks Next-Generation Firewall (NGFW) with Traditional Application and Packet Filter Firewalls.**

## Introduction to Firewalls

Firewalls are an important component of network security. designed to monitor and control incoming and outgoing network traffic according to predefined security rules. Traditional firewalls, such as application firewalls and packet filters works by inspecting data packets based on protocol, port and headers. But with the increasing use of cyberattacks, Traditional firewalls have proven inadequate in detecting advanced threats that exploit deep layers of network communications. This limitation has led to the development of next-generation firewalls (NGFW), like the Palo Alto Networks firewall, That brings security by providing deeper packet inspection, integrated threat intelligence and improved monitoring capabilities.

## Traditional Firewalls: Application and Packet Filtering

**Application Firewall:** Application firewall works by inspecting traffic at the application layer (OSI Layer 7), filtering traffic based on application data. This means that specific applicationlevel commands or patterns can be blocked within a session. This method is particularly effective at controlling traffic to or from specific applications (e.g., blocking P2P traffic) and can also provide detailed logging based on app activity.

**Packet filtering firewall**: Packet filtering firewall inspects packets at the network and transport layers (layers 3 and 4), focusing on the packet header rather than the data content. They filter the traffic based on source and destination IP addresses, protocols, and ports. They can be set up with rules to allow or deny specific traffic patterns. but this approach limits the firewall's ability to analyze the payload content or detect more sophisticated attacks hidden in seemingly legitimate traffic.

**Limitations of Traditional Firewalls:** Traditional application packet filtering firewalls have limitations when dealing with complex or evolving threats. They cannot:

1. Verify encrypted traffic. which makes it ineffective against attacks hidden within encrypted HTTPS traffic.
2. Look for malware and other threats embedded in the payload packet, which is because only the packet headers are checked.
3. Identify complex attack patterns that exploit application vulnerabilities or involve multiple layers of network communications.

## Next-Generation Firewalls (NGFWs): Palo Alto Networks Firewall

Palo Alto Networks' NGFW is a leader in advanced firewall technology that goes beyond traditional methods to monitor, filter, and control traffic based on application and user content.

Palo Alto NGFW integrates advanced features, which are;

1. **Application Identification (App-ID):** Palo Alto uses App-ID technology to classify all traffic through the app. regardless of ports, protocols, encryption, or other techniques that traditional firewalls may have trouble analyzing. This feature allows administrators to set rules accordingly on specific applications (for example, allowing Chrome but blocking BitTorrent), this provides precise control over network traffic.
2. **Content Inspection and Threat Protection:** The NGFW performs deep packet inspection (DPI) to analyze packet payload metadata. This makes it possible to identify malware attacks and other malicious content that traditional firewalls might miss. Threat protection features include intrusion prevention, anti-malware scanning and URL filtering are integrated within NGFW without the need for additional machine or software.
3. **User Identification (User-ID):** Unlike traditional firewalls that inspect traffic based only on IP addresses, Palo Alto's NGFW uses User-ID to map traffic to specific users, regardless of device or location. This feature improves visibility and control ,which allows you to apply rules based on user identity , group membership or role within the organization
4. **Encrypted Traffic Inspection:** Palo Alto NGFW has SSL decryption capabilities, allowing encrypted HTTPS traffic to be inspected for threats. By decrypting, inspecting, and re-encrypting traffic, it can detect malware or traffic that manipulates commands that might evade detection by traditional firewalls.
5. **Integrated Threat Intelligence**: Palo Alto NGFW uses WildFire, a cloud-based threat analysis service that provides real-time threat intelligence and behavioral analysis of unknown threats. This is done by analyzing suspicious file URLs in WildFire's secure cloud environment.

## Comparison: Detection Capabilities of Palo Alto NGFW vs. Traditional Firewalls

Palo Alto's NGFW detects many types of attacks that traditional application firewalls and packet filtering firewalls cannot:

1. **Advanced malware and zero-day attacks:** Traditional firewalls lack the ability to inspect packet payloads for malware or zero-day attacks. This makes them ineffective against sophisticated malware that masks as legitimate traffic, with Palo Alto's NGFW, integrated WildFire services, and deep content tracking a malicious payload can be caught in real time even if the threat was not previously known.
2. **Command-Control (C2) communication:** Many malware use encrypted channels to communicate with remote command-and-control servers. Traditional firewalls without SSL inspection capabilities cannot detect these communications if they use HTTPS. However, Palo Alto's NGFW can decode and inspect SSL/TLS traffic to identify Suspicious patterns or communications with known C2 servers.
3. **Application Layer Attacks:** Attacks that target specific applications, such as SQL injection, cross-site scripting (XSS), and other web application vulnerabilities, often bypass firewalls. This is because these attacks operate under the allowed ports and protocols Palo Alto's NGFW App-ID is able to detect such attacks through its content inspection feature, this analyzes payloads for detected application-level vulnerabilities.
4. **Lateral movement and insider threats:** Traditional firewalls often lack internal segmentation. This makes it difficult to monitor or control traffic on internal hosts. Palo Alto's NGFW provides granular segmentation and visibility into the internal network, pinpointing lateral movement and protect against insider threats. Alto's NGFW can apply security policies based on user behavior, which prevents unauthorized access and data transfer by malicious insiders.

## Scenario Where the NGFW Detects an Attack Missed by Traditional Firewalls

Imagine a scenario where a user's machine has advanced malware that uses HTTPS to communicate with a command control (C2) server. The malware creates an encrypted channel with the C2 server to receive commands and send data. Traditional firewalls, limited by filtering by IP address and port, cannot inspect encrypted HTTPS traffic. This means that this communication will be allowed without being flagged as suspicious.

However, Palo Alto's NGFW has SSL decryption. When encrypted traffic flows through NGFW, it is decrypted and inspected. NGFW detects unusual behavior patterns, such as communication with malicious IP addresses or connected C2 servers domains. NGFW is also able to identify malware behavior through WildFire analysis and automatically block connections, this prevents data extraction and stops further communication with the C2 server.

## Scenario Where the Attacker’s Capture Is Not Detected

An example of a situation in which NGFW does not record attacker activity might involve network segmentation and encrypted communications in isolated environments. Assuming an attacker attacks a device within a portion of the network that is not monitored by NGFW, the compromised device establishes a VPN connection to an external C2 server, encrypting all outgoing communications.

If this segment's traffic is sent directly to the Internet without passing through NGFW, the firewall will not be able to inspect or detect encrypted VPN traffic. This allows attackers to extract data without detection. To mitigate this risk, organizations can use network design guidelines to ensure that all outbound traffic is routed through NGFW, which can be analyzed for threats by decryption.

# TASK 8: Snort : Setting up, and installing your own rules

**Carry this out on Kali1.**

**First make a bakup of /etc/snort/snort.conf.**

**Do this in case you make an error, but also to save the default snort configuration.**

**cp /etc/snort/snort.conf /etc/snort/snort.bakup.conf**

**Then, edit /etc/snort/snort.conf**

**Uncomment the line which includes $RULE\_PATH/local.rules**

**(if it is commented out).**

**Make sure that the include of the other rules files are all commented out. e.g.**

**# include $RULE\_PATH/exploit.rules etc.**

**Next, edit /etc/snort/rules/local.rules and add in the following rules, one rule per line, (these are not rules that you would normally use ): log icmp any any -> any any (msg:”ping packet”; itype:8; icode:0; sid:22222222;rev:1) alert tcp any 21 -> any any (msg:"FTP Bad login"; content:"530 "; flow:from\_server; sid:33333333;rev:1;)**

**alert tcp any any <> any 23 (flags:S; msg:"Telnet Login Attempt";sid:44444444;rev:1;)**

**(The sid values used in the above rules are random. Snort reserves all sids below one million for itself, so user generated rules should have a sid of one million or greater. ) To run, use snort -c /etc/snort/snort.conf -A full -D (you may need to add -i ethX)**

**(Initially, run without the -D to find errors in the configuration file)**

**Be sure to add the -A full. Otherwise snort will not run as specified in the notes.**

**If you don’t you’ll be using u2spewfoo to read the log files.**

**To verify it is running:**

**ps aux | grep snortThen, from Kali2, send some packets to trigger alerts. View the alerts generated.**

**Check that the rules that you have added fire.**

**To stop snort, use killall -9 snort**

**Note: You can use the -T option to test your configuration/rules:**

**snort -i ethX -T -c /etc/snort/snort.conf -A full**

**No screenshots for this task because it slows my VM, Just stating significance**

Task 8 focuses on configuring Snort to detect specific network activities, Telnet login attempts in particular and also verify its functionality through testing and analysis. This task is important because it shows the signifacnce of Snort rules to detect specific intruders, monitor traffic in real time and analyze generated alerts to understand potential threats.

The process begins by creating a backup of the snort.conf configuration file. This helps ensure that incorrect configurations are reversible during operation. After backing up Snort configuration files are edited to change the behavior.

The important part of this work is creating custom rules in the local.rules file. The rule **alert tcp any any any <> any 23** (flags:S; msg:"Telnet Login Attempt"; sid:4444444; rev:1; ) is added to detect Telnet login attempts on port 23. This rule generates an alert whenever a TCP SYN is going for port 23. The use of ("Telnet login attempt") and unique SID (4444444) help ensure that notifications are easily identified in the log. This step demonstrates how to customize Snort rules to track specific activities related to the network environment.

After we configure snort, when monitoring, it saves the alert in /var/log/snort and we check there to verify the logs and can also be used to test certain rules that has been placed, just like the telnet rule. If a Telnet login is made, snort detects it and logs it.

In conclusion, task 8 shows us how important snort is in intrusion detection in our networks.

# TASK 9 : Write your own signature

**(a) Linux traceroute**  **traceroute** is a tool on Kali.

To run it, simply use **traceroute <ip-address>** and use the IP address of the host whose OS you want to determine.

Run traceroute and capture the packets that it generates using tcpdump.

Examine these packets using wireshark or tcpdump.

1. explain how the linux traceroute works.
2. Write a snort signature to alert on seeing the linux traceroute and test it.

**(b) Windows traceroute**

Start a packet capture on one of the Kali’s.

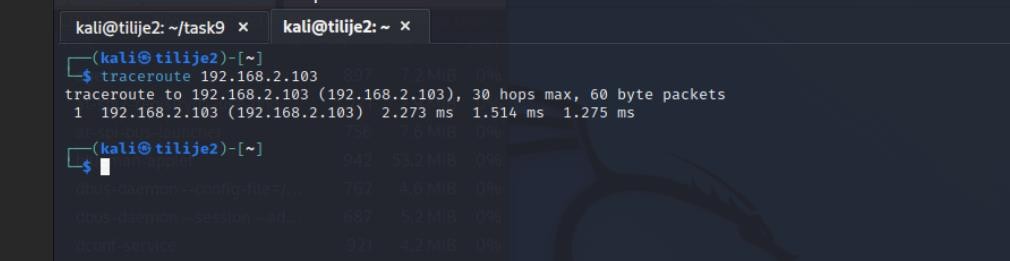
From the Win10 machine, traceroute to the Kali.

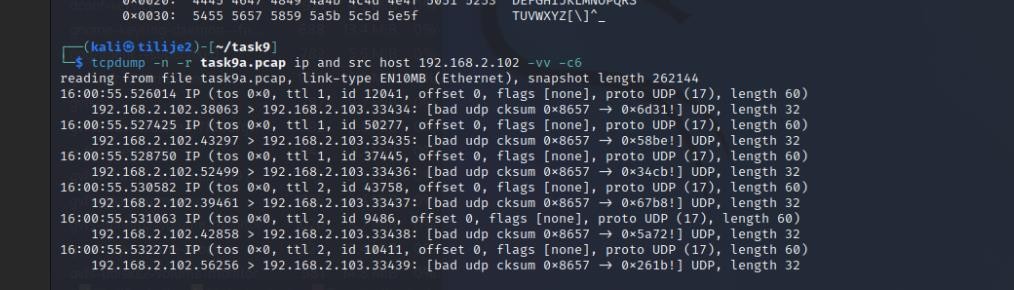
The command is **tracert**.

Stop the capture.

1. explain how the Windows traceroute works.
2. Write a snort signature to alert on seeing the Windows traceroute and test it.

Our first step will be to capture the packets and linux traceroute





## Fig 9.1 Linux Traceroute

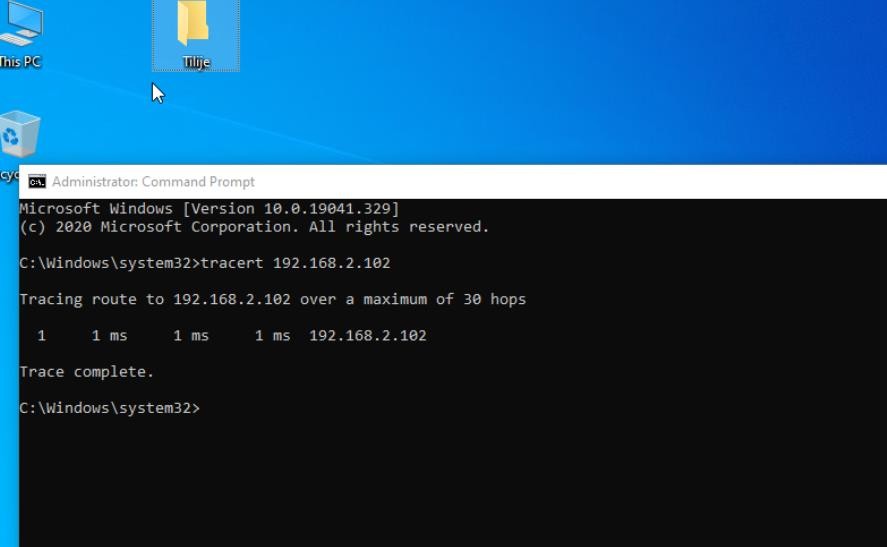
Linux's Traceroute works by increasing the TTL (Time to Live) value by sending packets to the destination IP address starting from 1. Each hop along the way reduces the TTL by 1, and when the TTL reaches zero, the router sends a ICMP “Time Exceeded" message back, showing the IP address. This process continues until the packet reaches its destination, at which the target host returns a response indicating the end of the route.

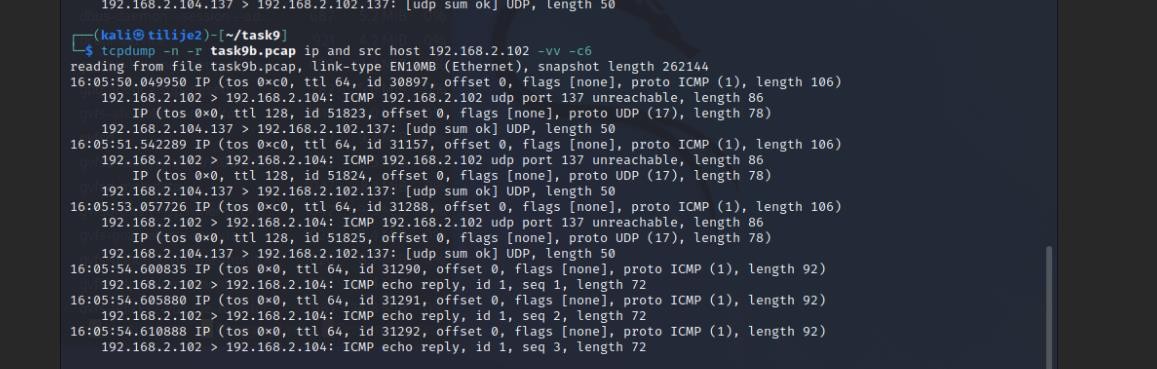
In the output (Fig 9.1), the Traceroute command displays the direct route to 192.168.2.103 with a single hop. The first line summarizes the target IP, maximum number of hops allowed and the size of the packets sent. The results indicate that the packet reached its destination within one hop. This is because the single hop IP address is the same as the destination address. (192.168.2.103).

The second screenshot shows using tcpdump to capture packets generated during the traceroute. The recorded output shows the UDP packets sent to kali3 with a different ID and offset. This shows how Linux Traceroute uses UDP packets by default, by sending it too high and to a unused port number. The ID and offset are changed for each packet, It reflects the progress of the TTL value and the responses that follow.

The recorded output confirms the traceroute behavior. It shows the set of packets sent to the target destination and the responses that were returned. It shows that Linux's traceroute efficiently maps paths through the network.

Now, we do windows traceroute.





## Fig 9.2 Windows Traceroute

The Windows Traceroute (tracert) command works by sending ICMP echo requests with incremental TTL. It is used to trace the path a packet took from source to destination on a hop by hop basis in the network **Process:**

* ICMP Echo Request with Incremental TTL: The Traceroute tool starts by sending an

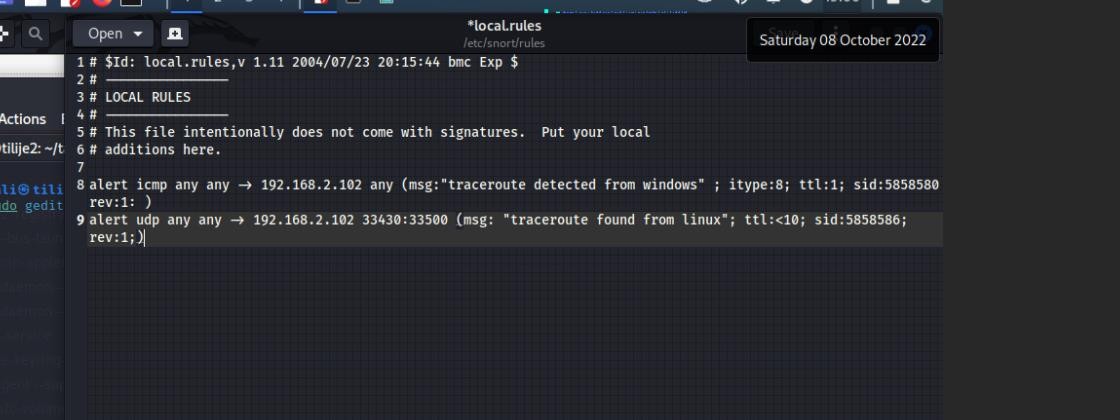
ICMP packet with a TTL value of 1. This packet is initially forwarded to the router which

reduces the TTL duration to 0. When the TTL expires, the router responds with an ICMP "Time Exceeded" message introducing itself to the source system.

* Next hop: The TTL value for the next ICMP packet is increased by 2 to reach the next router This process is repeated, increasing the TTL by 1, until the packet reaches its destination or the maximum TTL value is reached.
* Destination Response: When the packet finally reaches its destination, the destination host sends an ICMP Echo Reply, indicating the completion of the traceroute process.

The tcpdump output provides a detailed view of the network traffic generated by Windows Traceroute to the Kali machine, showing the use of UDP packets for detection and ICMP messages for replies. This is standard behavior for Traceroute, UDP packets are initially sent from the source IP (192.168.2.102) to a high port (such as 33434) on the destination IP (192.168.2.103). These packets are part of the traceroute process, increasing the TTL for each step. This is because there are no intermediate hops in this network, the destination device then responds directly to the with ICMP's "port not reachable" messages, indicating that the UDP packet has reached its target even when the port is closed. This response can confirm the path with traceroute. The last line of output shows the ICMP echo replies as part of the completion process, with Round Trip Times (RTT) indicating the minimum latency. This output confirms that there is no intermediate router and shows a direct connection between the source and destination. This ensures there are no routing issues or excessive delays in this.

These are the snort rules for seeing linux and windows traceroute



## TASK 10 : Snort: Using a PreProcessor to detect a port scan

**The task also uses a file of rules which have been put together to detect port scans.**

From another host, port scan your host running snort using 2 different scan types:

**nmap -sS <your-host>** and **nmap -sX <your-host>**  and watch snort's response.

(aside: -sS is a SYN scan and -sX is a XMAS scan)

1. Try this with no rules at all. See if snort detects the scan.

Which of the 2 scans does it detect, if any?

1. Then, try it with the **portscan preprocessor** enabled, and with no rules enabled.

You may need to change the section in snort.conf that specifies the sfportscan preprocessor to

# Portscan detection. For more information, see README.sfportscan

**preprocessor sfportscan: proto { all } memcap { 10000000 } scan\_type { all } sense\_level**

**{ medium } logfile { /var/log/snort/portscan }**  Note you need to have **include classification.config**  **include reference.config**  uncommented in snort.conf

1. Next, disable the portscan preprocessor and include the rules in the file “scan.rules” that comes with snort on Kali.

You do this by simply uncommenting the relevant include line in snort.conf.

Scan the host again using both scans, and notice any differences.

1. With scan.rules enabled, and with the sfportscan preprocessor enabled, from another host try **nmap -sS -T sneaky <your-host> -p 22-23**  and see if the scan is detected.

Note: This scan can take a while, as it involves an evasion attempt.

No screenshots for this task because my VM is too slow and I have only a few screenshots, so I am going to talk about the main significance of the task.

Task 10 was designed to explore how Snort detects port scans using a combination of behavioral detection through the preprocessor and signature-based detection through rules. This task demonstrates snort's ability to detect various scanning techniques such as SYN scanning and XMAS scanning, both commonly used in reconnaissance activities. These scans are critical for detection because they often inform attackers when they search for vulnerabilities in the network.

The task begins with configuring snort's preprocessor, sfportscan, which analyzes traffic patterns to identify scanning behavior based on connections that it has. This configuration allows snort to detect unusual activities without relying on predefined packet signatures. Running Snort with only the preprocessor enabled can generate alert based on behavior. It provides insight into how effectively Snort can detect stealthy or abnormal scanning techniques.

To test Snort's detection capabilities, I can scan from kali2 using a tool such as nmap, initiating a SYN scan (-sS), which involves sending TCP SYN packets to various ports without completing the handshake, it is very quiet. Additionally, an XMAS scan (-sX), which sends packets containing an unusual combination of flags (FIN, PSH, and URG), is performed to see whether the preprocessor can identify this unusual behavior they use.

The final task involves the use of -T sneaky, this tests the sensitivity of snort and if it can detect packets that are sent quietly.

# References

1. Palo Alto Networks, “Next-Generation Firewalls (NGFW),” Palo Alto Networks Documentation. [Online]. Available:

https://docs.paloaltonetworks.com/ngfw. [Accessed: Nov. 11, 2024].

1. M. Roesch, “Snort – The Open Source Network Intrusion Detection System,” Snort Documentation. [Online]. Available: https://www.snort.org. [Accessed: Nov. 15, 2024].
2. The Linux Foundation, “iptables: The Linux Kernel Firewall Tool,” [Online]. Available: https://netfilter.org. [Accessed: Nov. 10, 2024].
3. Intellect IT, “Traditional vs Next-Gen Firewalls,” Intellect IT Blog. [Online]. Available: https://www.intellectit.com.au/traditional-vsnext-genfirewalls/#:~:text=Traditional%20firewalls%20work%20on%20the ,tries%20to%20breach%20the%20system. [Accessed: Nov. 12, 2024].
4. Linux man-pages project, “traceroute(8) - Linux Manual Page,” [Online]. Available: https://man7.org/linux/manpages/man8/traceroute.8.html. [Accessed: Nov. 19, 2024].
5. Quora, “What is port stealing?” Quora. [Online]. Available:

https://www.quora.com/What-is-port-stealing. [Accessed: Nov. 17, 2024].