

CyberSecurity (CSL6010)

Lab Assignment 7 : Replay Attack

Soumen Kumar
Roll No-B22ES006

1 Task 1: Understanding Replay Attacks

A replay attack is a cybersecurity threat where an attacker intercepts and records legitimate data packets before resending them to the recipient. This attack exploits the lack of uniqueness in transmitted messages, making the recipient believe that the replayed packets are authentic. As a result, an attacker can bypass authentication mechanisms, manipulate data, or disrupt network operations.

1.1 Key Components of Replay Attacks

Replay attacks generally consist of three major steps:

1. **Packet Capture:** The attacker first captures legitimate network packets using tools like Wireshark.
2. **Packet Replay:** The attacker then resends the intercepted packets using tools like Scapy, making them appear as fresh transmissions.
3. **Deception:** Since the recipient cannot differentiate between the original and replayed packets, the attacker can exploit this to gain unauthorized access or disrupt communication.

1.2 Types of Replay Attacks

Replay attacks can be classified into different categories based on how they are executed:

- **Simple Replay:** The attacker replays captured packets exactly as they were without any modifications.
- **Delayed Replay:** The attacker stores the captured packets and replays them after some time to evade detection.
- **Modified Replay:** The attacker alters certain packet fields before resending them, making them appear different while still being valid to the recipient.

1.3 Impact of Replay Attacks

Replay attacks pose serious threats to cybersecurity, leading to various consequences:

- **Authentication Bypass:** Attackers can gain unauthorized access by replaying authentication-related packets.
- **Data Manipulation:** Sensitive data can be modified or duplicated, affecting the integrity of communications.
- **Denial of Service (DoS):** Replying large volumes of packets can overwhelm network resources, causing disruptions.

1.4 Common Targets

Replay attacks commonly target the following systems:

- **Authentication Systems:** Passwords, security tokens, and biometric authentication data are at risk.
- **Financial Transactions:** Unauthorized replaying of transaction requests can result in duplicate payments or fraudulent activities.
- **Control Systems:** Industrial control networks and IoT devices can be manipulated by replaying control commands, leading to potential operational failures.

2 Task 2: Lab Setup

To set up a lab environment for replay attack simulation, we need to install and configure the necessary software: Wireshark, Python, and Scapy. The following steps outline the setup process on a Kali Linux system.

2.1 Step 1: Install Wireshark

1. Open a terminal and update package lists:

```
sudo apt update
```

2. Install Wireshark:

```
sudo apt install wireshark -y
```

3. During installation, select **Yes** to allow non-root users to capture packets.
4. Verify the installation by running:

```
wireshark
```

2.2 Step 2: Install Python

1. Check if Python is installed:

```
python3 --version
```

2. If Python is not installed, install it with:

```
sudo apt install python3 -y
```

2.3 Step 3: Install Scapy

1. Ensure pip is installed:

```
sudo apt install python3-pip -y
```

2. Install Scapy using pip:

```
pip3 install scapy
```

3. Verify the installation by running:

```
scapy
```

2.4 Step 4: Verify Installations

- Open Wireshark and ensure it runs without errors.
- Check Python installation using:

```
python3 --version
```

- Verify Scapy installation by launching it with:

```
scapy
```

2.5 Step 5: Set Up the Testing Environment

- Ensure Wireshark and Scapy are correctly installed and working.
- Set up a network environment for capturing and analyzing packets.
- Use either a local system (localhost) or two networked devices for testing replay attacks.

3 Task 3: Capturing Network Packets

3.1 Step 1: Open Wireshark

1. **Launch Wireshark:** I open Wireshark from my Applications folder.
2. **Select an Interface:** On the start page, I see a list of network interfaces (Wi-Fi, Ethernet, etc.). Since I'm testing on my own machine, I select lo0 (localhost).

3.2 Step 2: Start Capturing Packets

1. **Begin Capture:** I click on the interface I chose to start capturing packets.
2. **Generate ICMP Traffic:** To generate some network traffic, I open a terminal and type:

```
ping localhost
```

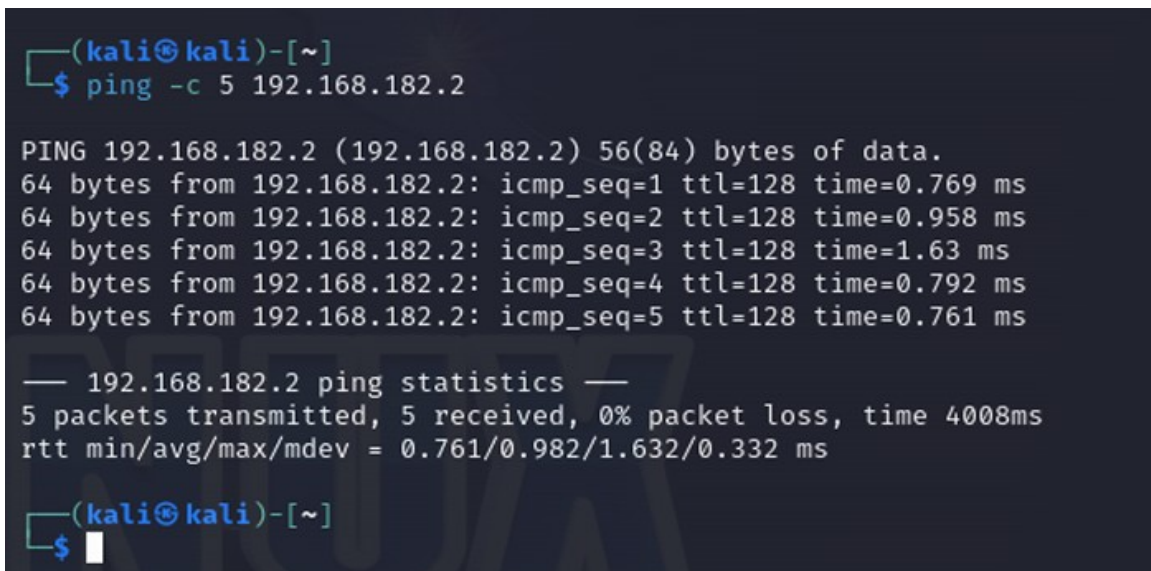
3. If I want to capture packets on a different interface (e.g., Wi-Fi), I ping another device on my network.

3.3 Step 3: Stop and Save the Capture

1. **Stop Capture:** After capturing enough packets (10-20 is good), I click the red square button in Wireshark to stop recording.
2. **Save the Capture:** I go to File > Save As, choose a location, and save my capture in .pcapng format.

3.4 Step 4: Analyze the Capture

1. **Filter ICMP Packets:** In the Wireshark filter bar, I type icmp and press Enter. This helps me focus only on ICMP packets.
2. **Examine Packets:** I click on individual packets to see their details in the middle and bottom panes of Wireshark.

A terminal window on a Kali Linux system. The prompt is (kali@kali)-[~]. The user enters the command \$ ping -c 5 192.168.182.2. The output shows five successful ping responses with varying times. Below the responses, it shows the ping statistics: 5 packets transmitted, 5 received, 0% packet loss, time 4008ms, and rtt min/avg/max/mdev = 0.761/0.982/1.632/0.332 ms. The prompt returns to (kali@kali)-[~].

```
(kali@kali)-[~]  
$ ping -c 5 192.168.182.2  
  
PING 192.168.182.2 (192.168.182.2) 56(84) bytes of data.  
64 bytes from 192.168.182.2: icmp_seq=1 ttl=128 time=0.769 ms  
64 bytes from 192.168.182.2: icmp_seq=2 ttl=128 time=0.958 ms  
64 bytes from 192.168.182.2: icmp_seq=3 ttl=128 time=1.63 ms  
64 bytes from 192.168.182.2: icmp_seq=4 ttl=128 time=0.792 ms  
64 bytes from 192.168.182.2: icmp_seq=5 ttl=128 time=0.761 ms  
  
— 192.168.182.2 ping statistics —  
5 packets transmitted, 5 received, 0% packet loss, time 4008ms  
rtt min/avg/max/mdev = 0.761/0.982/1.632/0.332 ms  
  
(kali@kali)-[~]  
$
```

Figure 1: PING check

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
2	0.023639498	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
3	0.507988840	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
4	7.501817897	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
5	8.643818846	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
6	9.508920441	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
7	10.504747019	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
8	14.661131018	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
9	15.498920253	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
10	16.497367349	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
11	17.687064295	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
12	18.505689126	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
13	19.496650881	Vmware_c0:00:00:00:00:00	Broadcast	ARP	60	Who has 192.168.182.2? Tell 192.168.182.1
14	42.386271568	192.168.182.128	192.168.182.2	ICMP	98	Echo (ping) request id=0x797e, seq=1/256, ttl=64 (reply in 17)
15	42.386651089	Vmware_ff:08:70	Broadcast	ARP	60	Who has 192.168.182.128? Tell 192.168.182.2
16	42.386657657	Vmware_ff:08:70	Vmware_ff:08:70	ARP	42	192.168.182.128 is at 00:0c:29:bf:86:4d
17	42.386970436	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) reply id=0x797e, seq=1/256, ttl=128 (request in 14)
18	43.388547697	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) request id=0x797e, seq=2/512, ttl=64 (reply in 19)
19	43.389459895	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) reply id=0x797e, seq=2/512, ttl=128 (request in 18)
20	44.390374554	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) request id=0x797e, seq=3/768, ttl=64 (reply in 21)
21	44.391959198	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) reply id=0x797e, seq=3/768, ttl=128 (request in 20)
22	45.391876454	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) request id=0x797e, seq=4/1024, ttl=64 (reply in 23)
23	45.392622539	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) reply id=0x797e, seq=4/1024, ttl=128 (request in 22)
24	46.393777989	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) request id=0x797e, seq=5/1280, ttl=64 (reply in 25)
25	46.394491849	192.168.182.2	192.168.182.128	ICMP	98	Echo (ping) reply id=0x797e, seq=5/1280, ttl=128 (request in 24)
26	47.612726236	Vmware_ff:08:70	Vmware_ff:08:70	ARP	42	Who has 192.168.182.2? Tell 192.168.182.128
27	47.613407902	Vmware_ff:08:70	Vmware_ff:08:70	ARP	60	192.168.182.2 is at 00:50:56:ff:08:70

Figure 2: Wireshark Capturing ICMP Packets

4 Task 4: Performing a Replay Attack

4.1 Step 1: Capture ICMP Packets

1. **Capture ICMP Traffic:** I use Wireshark to capture ICMP packets by pinging a target (e.g., localhost or google.com). I apply the icmp filter to view only ICMP packets.
2. **Save the Capture:** I save the captured packets as a .pcapng file.

4.2 Step 2: Replay Captured Packets with Scapy

1. **Write a Scapy Script:** I use the following Python script to replay the captured packets:

```
from scapy.all import rdpcap, send
# Loading captured packets
packets = rdpcap("before_attack.pcapng")
# Replaying each packet
print("Replaying packets...")
for packet in packets:
    send(packet)
    print(f"Replayed: {packet.summary()}")
```

2. **Run the Script:** I execute the script using Python:

```
python lab7_attack.py
```

4.3 Step 3: Analyze the Replay in Wireshark

1. **Start a New Capture:** Before running the Scapy script, I start a new capture in Wireshark on the same interface (lo0 for localhost).
2. **Run the Scapy Script:** I execute the script to replay the packets.
3. **Stop the Capture:** I stop the capture in Wireshark after the replay is complete.
4. **Analyze the Replay:** I apply the icmp filter and examine the packets. I look for duplicate packets or any abnormalities in the sequence numbers or timestamps.

508	5.531657444	10.40.0.244	10.40.0.249	ICMP	98 Echo (ping) request	ic
509	5.532392031	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
510	5.532883170	10.40.0.244	10.40.0.249	ICMP	98 Echo (ping) request	ic
511	5.533097689	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
512	5.533289130	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
513	5.533746634	10.40.0.244	10.40.0.249	ICMP	98 Echo (ping) request	ic
514	5.533803557	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
515	5.534164442	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
516	5.534626559	10.40.0.244	10.40.0.249	ICMP	98 Echo (ping) request	ic
517	5.534653436	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
518	5.535075668	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
519	5.535473711	10.40.0.244	10.40.0.249	ICMP	98 Echo (ping) request	ic
520	5.535884216	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
521	5.536299234	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
522	5.536299322	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
523	5.536559532	10.40.0.244	10.40.0.249	ICMP	98 Echo (ping) request	ic
524	5.537033513	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
525	5.537441272	10.40.0.244	10.40.0.249	ICMP	98 Echo (ping) request	ic
526	5.537863497	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
527	5.538064763	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic
528	5.538064867	10.40.0.249	10.40.0.244	ICMP	98 Echo (ping) reply	ic

```

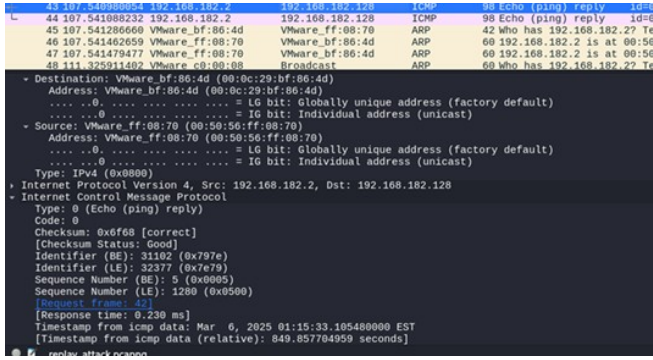
Frame 508: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
Ethernet II, Src: VMware_8b:fb:9e (00:0c:29:8b:fb:9e), Dst: VMware_11:be:88 (00:0c:29:11:be:88)
Internet Protocol Version 4, Src: 10.40.0.244, Dst: 10.40.0.249
Internet Control Message Protocol

```

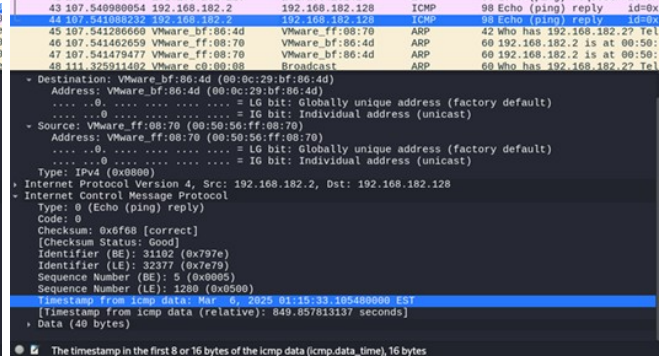
Figure 3: Wireshark Capturing ICMP Packets during attack

4.4 Abnormalities to Look For

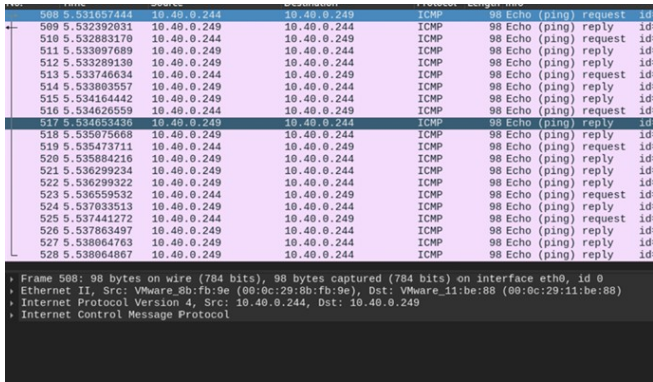
- **Identical Timestamps:** If the packets are replayed quickly, they may have the same or very close timestamps, which is unusual in a normal session.
- **Sequence Number Discontinuity:** If packets are replayed out of order, I might see sequence numbers that are not consecutive.
- **Identical Payload:** The replayed packets may have exactly the same payload as the original ones, which is a strong indicator of a replay attack.



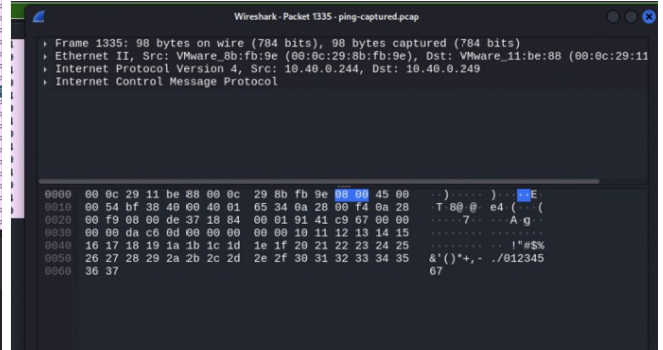
(a) timestamp43



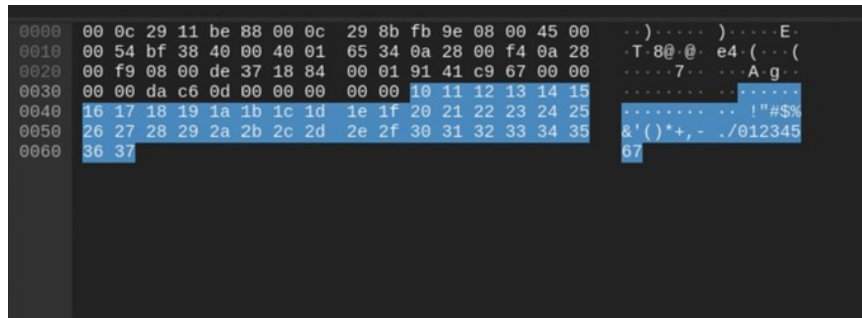
(b) timestamp44



(c) SN Discontinuity



(d) IdenticalPayload1



(e) IdenticalPayload2

Figure 4: Grouped figures showing various network abnormalities

5 Task 5: Detecting a Replay Attack

5.1 Step 1: Write a Python Script to Detect Replay Attacks

To detect replayed packets, I use a Python script that checks for repeated sequence numbers or identical payloads in a packet capture file.

5.1.1 Python Script for Replay Detection

```
from scapy.all import rdpcap

# Loading the captured packets
packets = rdpcap('captured_packets.pcapng') # Replace with your .pcapng file path
# Initializing sets to track seen sequence numbers and payloads
seen_seq_numbers = set()
seen_payloads = set()
# Tracking replayed packets
replayed_packets = []
# Iterating through each packet
for pkt in packets:
    if pkt.haslayer('ICMP'): # Check if it's an ICMP packet
        icmp_layer = pkt['ICMP']
        # Getting sequence number and payload (if available)
        seq_number = icmp_layer.seq if hasattr(icmp_layer, 'seq') else None
        payload = bytes(pkt[ICMP].payload)
        # Checking for duplicates
        if seq_number in seen_seq_numbers or payload in seen_payloads:
            replayed_packets.append(pkt)
        else:
            seen_seq_numbers.add(seq_number)
            seen_payloads.add(payload)

# Outputting results
print(f"Total Packets: {len(packets)}")
print(f"Replayed Packets Detected: {len(replayed_packets)}")

if replayed_packets:
    print("\nDetails of Replayed Packets:")
    for rpkt in replayed_packets:
        print(rpkt.summary())
else:
    print("No replayed packets detected.")
```

5.2 How It Works

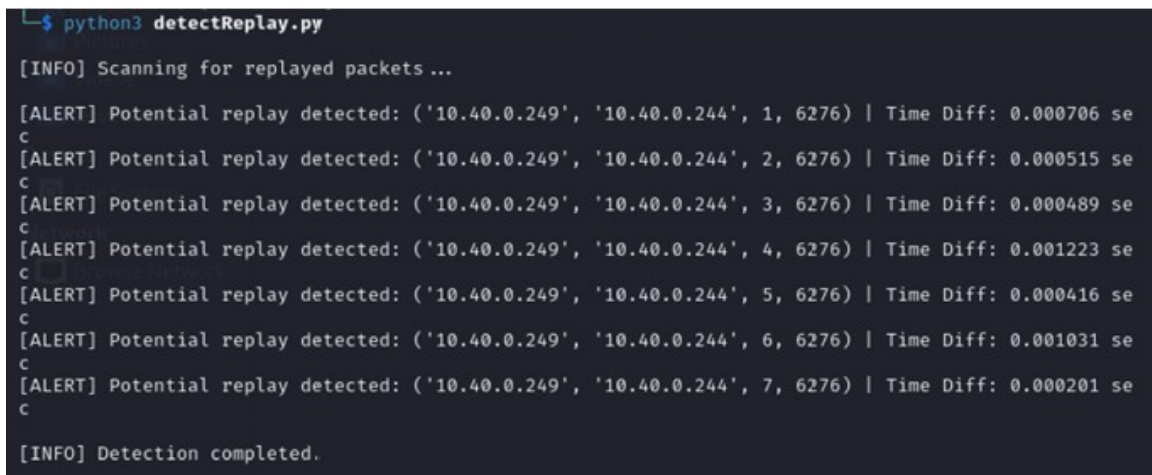
1. **Input:** The script processes a .pcapng file containing captured ICMP traffic.
2. **Detection:**
 - It checks for duplicate sequence numbers (`icmp.seq`) or identical payloads.
 - If duplicates are found, the packet is flagged as a replayed packet.
 - Extracts key identifiers such as source IP, destination IP, ICMP sequence number, and identifier.
 - Compares timestamps, sequence numbers, and payloads to check for repeated occurrences.
 - Prints alerts for potential replay attacks.
3. **Output:**
 - Displays the total number of packets analyzed.
 - Reports the number of replayed packets detected.
 - Provides a summary of replayed packets (if any).

5.3 Step 2: Run the Script

1. Save the captured packet file from Wireshark (e.g., `captured_packets.pcapng`).
2. Run the script using Python:

```
python detect_replay.py
```

3. Review the output to check for any detected replayed packets.



```
$ python3 detectReplay.py
[INFO] Scanning for replayed packets ...
[ALERT] Potential replay detected: ('10.40.0.249', '10.40.0.244', 1, 6276) | Time Diff: 0.000706 se
c
[ALERT] Potential replay detected: ('10.40.0.249', '10.40.0.244', 2, 6276) | Time Diff: 0.000515 se
c
[ALERT] Potential replay detected: ('10.40.0.249', '10.40.0.244', 3, 6276) | Time Diff: 0.000489 se
c
[ALERT] Potential replay detected: ('10.40.0.249', '10.40.0.244', 4, 6276) | Time Diff: 0.001223 se
c
[ALERT] Potential replay detected: ('10.40.0.249', '10.40.0.244', 5, 6276) | Time Diff: 0.000416 se
c
[ALERT] Potential replay detected: ('10.40.0.249', '10.40.0.244', 6, 6276) | Time Diff: 0.001031 se
c
[ALERT] Potential replay detected: ('10.40.0.249', '10.40.0.244', 7, 6276) | Time Diff: 0.000201 se
c
[INFO] Detection completed.
```

Figure 5: Detection

6 Task 6: Preventing Replay Attacks

6.1 Step 1: Understanding Replay Attack Prevention Mechanisms

To prevent replay attacks, I can use several techniques. Here are some effective methods:

- **Timestamps:** I can add timestamps to packets or messages so they are only processed within a specific time window. If a packet is too old (e.g., more than 2 minutes), I should reject it.
- **Nonces (Numbers Used Once):** Each message should include a unique identifier (nonce). If a nonce is reused, the system should flag it. This ensures every packet is unique.
- **Encryption:** Encrypting data with secure protocols like SSL/TLS prevents attackers from intercepting and replaying packets without proper decryption.
- **Session Tokens:** By assigning unique session IDs for communication sessions, I can make sure that tokens expire after use or at the end of the session, preventing replays.
- **Payload Validation:** Using cryptographic hashes or Message Authentication Codes (MACs) helps validate the integrity of the payload. If a replay attempt happens, the hash won't match.
- **Monitoring for Anomalies:** I can analyze network traffic for repeated patterns, duplicate sequence numbers, or identical payloads. Detecting such anomalies helps identify replay attacks early.

6.2 Step 2: Modifying the Detection Script

To improve detection, I can enhance the existing script by adding logging and preventive measures. Below is the updated Python script:

```
from scapy.all import rdpcap
import logging
import time

# Configuring logging to log suspicious activity
logging.basicConfig(filename='replay_attack_log.txt', level=logging.INFO, format='%(asctime)s - %(message)s')
# Loading the captured packets
packets = rdpcap('captured_packets.pcapng')
# Initializing sets to track seen sequence numbers, payloads, and timestamps
seen_seq_numbers = set()
seen_payloads = set()
seen_timestamps = set()

# Tracking replayed packets
replayed_packets = []
# Defining time window for valid packets (e.g., 2 minutes)
TIME_WINDOW = 120 # seconds
# Getting the current time for timestamp validation
current_time = time.time()

# Iterating through each packet
for pkt in packets:
    if pkt.haslayer('ICMP'): # Checking if it's an ICMP packet
        icmp_layer = pkt['ICMP']

        # Getting sequence number, payload, and timestamp (if available)
        seq_number = icmp_layer.seq if hasattr(icmp_layer, 'seq') else None
        payload = bytes(pkt[ICMP].payload)
        pkt_time = pkt.time if hasattr(pkt, 'time') else None

        # Checking for duplicates in sequence numbers and payloads
        if seq_number in seen_seq_numbers or payload in seen_payloads:
            replayed_packets.append(pkt)
            logging.info(f"Replay detected: Sequence Number={seq_number}, Payload={payload}")
```

```

# Checking for outdated timestamps
elif pkt_time and (current_time - pkt_time > TIME_WINDOW):
    logging.info(f"Outdated packet detected: Timestamp={pkt_time}")

else:
    seen_seq_numbers.add(seq_number)
    seen_payloads.add(payload)
    if pkt_time:
        seen_timestamps.add(pkt_time)

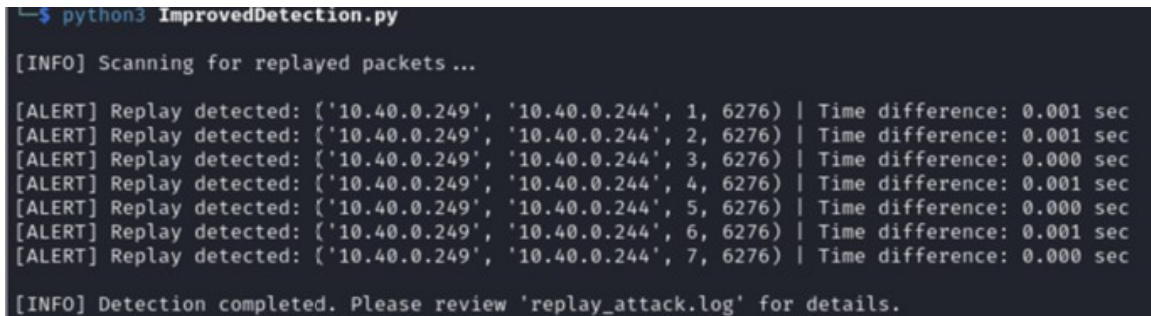
# Outputting results
print(f"Total Packets: {len(packets)}")
print(f"Replayed Packets Detected: {len(replayed_packets)}")

if replayed_packets:
    print("Details of Replayed Packets logged in replay_attack_log.txt")
else:
    print("No replayed packets detected.")

```

6.3 How This Script Works

1. **Duplicate Detection:** It tracks sequence numbers and payloads to find repeated packets.
2. **Timestamp Validation:** It verifies if packet timestamps are within the allowed time window (e.g., 2 minutes).
3. **Logging:** Any suspicious activity, such as duplicate packets or outdated timestamps, is recorded in `replay_attack_log.txt`.



```

python3 ImprovedDetection.py

[INFO] Scanning for replayed packets ...

[ALERT] Replay detected: ('10.40.0.249', '10.40.0.244', 1, 6276) | Time difference: 0.001 sec
[ALERT] Replay detected: ('10.40.0.249', '10.40.0.244', 2, 6276) | Time difference: 0.001 sec
[ALERT] Replay detected: ('10.40.0.249', '10.40.0.244', 3, 6276) | Time difference: 0.000 sec
[ALERT] Replay detected: ('10.40.0.249', '10.40.0.244', 4, 6276) | Time difference: 0.001 sec
[ALERT] Replay detected: ('10.40.0.249', '10.40.0.244', 5, 6276) | Time difference: 0.000 sec
[ALERT] Replay detected: ('10.40.0.249', '10.40.0.244', 6, 6276) | Time difference: 0.001 sec
[ALERT] Replay detected: ('10.40.0.249', '10.40.0.244', 7, 6276) | Time difference: 0.000 sec

[INFO] Detection completed. Please review 'replay_attack.log' for details.

```

Figure 6: Log1

```
detectReplay.py x ImprovedDetection.py x replay_attack.log x
1 2025-03-06 01:53:40,956 - INFO - Replay detected: ('10.40.0.249',
'10.40.0.244', 1, 6276) | Time difference: 0.001 sec
2 2025-03-06 01:53:40,957 - INFO - Replay detected: ('10.40.0.249',
'10.40.0.244', 2, 6276) | Time difference: 0.001 sec
3 2025-03-06 01:53:40,957 - INFO - Replay detected: ('10.40.0.249',
'10.40.0.244', 3, 6276) | Time difference: 0.000 sec
4 2025-03-06 01:53:40,957 - INFO - Replay detected: ('10.40.0.249',
'10.40.0.244', 4, 6276) | Time difference: 0.001 sec
5 2025-03-06 01:53:40,957 - INFO - Replay detected: ('10.40.0.249',
'10.40.0.244', 5, 6276) | Time difference: 0.000 sec
6 2025-03-06 01:53:40,957 - INFO - Replay detected: ('10.40.0.249',
'10.40.0.244', 6, 6276) | Time difference: 0.001 sec
7 2025-03-06 01:53:40,957 - INFO - Replay detected: ('10.40.0.249',
'10.40.0.244', 7, 6276) | Time difference: 0.000 sec
8 |
```

Figure 7: Log2

6.4 Step 3: Implementing Preventive Measures

To prevent future replay attacks, I need to implement the following safeguards:

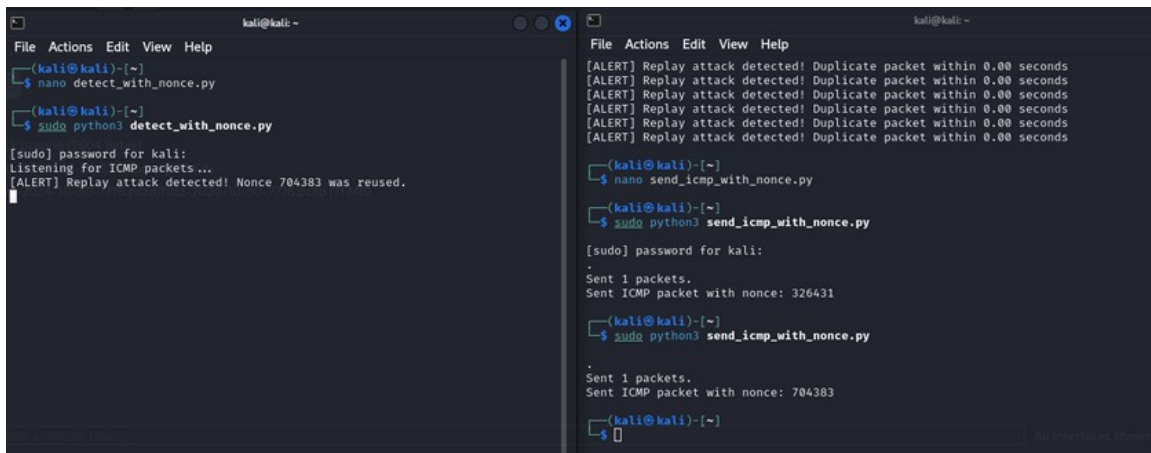
- **Add Timestamps to Packets:** Every packet should have a timestamp. If it's too old, I should reject it.
- **Use Nonces:** I can generate a unique nonce for each packet and check if it has been used before.

6.4.1 Implementing Nonces in Packet Transmission

- A **nonce** is a unique, random number sent with each request to prevent replay attacks. The recipient verifies whether the nonce is new before processing the packet.
- **Concept of Generating and Verifying Nonces:**
 1. **Attacker's Goal:** Capture and resend a packet.
 2. **Defense Mechanism:** Packets should include a random nonce (random value) and reject duplicates.
- **Implementation:** Two scripts were created to implement this mechanism:
 - * `send_icmp_with_nonce.py` – Sends ICMP packets with a unique nonce.
 - * `detect_with_nonce.py` – Detects and verifies nonce values to flag replay attacks.
- **Execution and Results:**
 - * When sending new ICMP packets, each packet had a unique nonce.
 - * When replaying old packets, the script successfully detected and flagged them as replay attacks.
- **ICMP Packets Sent with Unique Nonces:**
 - * The script `send_icmp_with_nonce.py` successfully generated and sent ICMP packets with unique nonce values (e.g., 326431, 704383).
- **Replay Attack Detection Triggered:**
 - * The detection script `detect_with_nonce.py` identified a replay attack when the same nonce (704383) was reused.
 - * The output:

[ALERT] Replay attack detected! Nonce 704383 was reused.

confirms that the detection mechanism is working as intended.



The image shows two terminal windows side-by-side. The left window shows the execution of `detect_with_nonce.py`, which starts listening for ICMP packets and then prints an alert when nonce 704383 is reused. The right window shows the execution of `send_icmp_with_nonce.py`, which sends ICMP packets with nonces 326431 and 704383. The right window also shows a list of alerts from the detection script, indicating that the same nonce was used multiple times.

```
kali@kali: ~  
File Actions Edit View Help  
~(kali@kali):~  
$ nano detect_with_nonce.py  
~(kali@kali):~  
$ sudo python3 detect_with_nonce.py  
[sudo] password for kali:  
Listening for ICMP packets...  
[ALERT] Replay attack detected! Nonce 704383 was reused.  
  
kali@kali: ~  
File Actions Edit View Help  
~(kali@kali):~  
$ nano send_icmp_with_nonce.py  
~(kali@kali):~  
$ sudo python3 send_icmp_with_nonce.py  
[sudo] password for kali:  
Sent 1 packets.  
Sent ICMP packet with nonce: 326431  
~(kali@kali):~  
$ sudo python3 send_icmp_with_nonce.py  
Sent 1 packets.  
Sent ICMP packet with nonce: 704383  
~(kali@kali):~  
$  
[ALERT] Replay attack detected! Duplicate packet within 0.00 seconds  
[ALERT] Replay attack detected! Duplicate packet within 0.00 seconds  
[ALERT] Replay attack detected! Duplicate packet within 0.00 seconds  
[ALERT] Replay attack detected! Duplicate packet within 0.00 seconds  
[ALERT] Replay attack detected! Duplicate packet within 0.00 seconds  
[ALERT] Replay attack detected! Duplicate packet within 0.00 seconds
```

Figure 8: Nonce Check

- **Encrypt Communication:** Using SSL/TLS protocols ensures that intercepted packets cannot be modified or replayed.
- **Validate Payload Integrity:** I should apply cryptographic hashes or MACs to detect tampered packets.

By following these steps, I can minimize the risk of replay attacks and keep my network secure.