

Fuzzear la librería 7zip de winrar

Como dice el título vamos a fuzzear el formato 7zip de winrar, donde se encuentran todos los métodos y funciones que usa el formato 7zip es en la dll 7zxa.dll.

El fuzzer que vamos a usar es el WinAFL, el primer paso que tenemos que hacer es armar un harness con las funcionalidades que queremos probar.

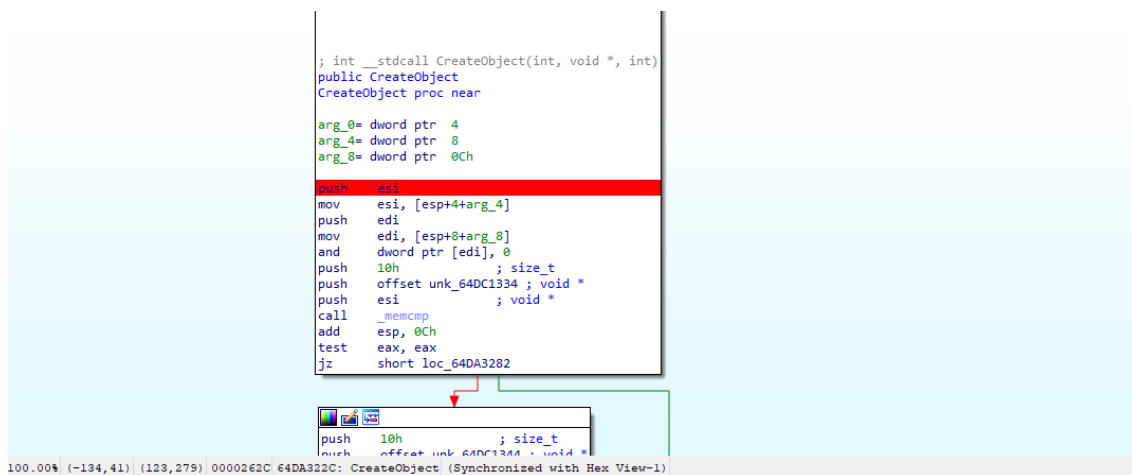
El harness se tiene que diseñar de la siguiente manera:

- Primero abre el archivo.
- Ejecuta las funciones.
- Cierra el archivo y libera memoria.

Para poder armarlo primero tenemos que reversear y ver cómo funciona así elegimos las funciones a fuzzear.

Aclaro que en este tuto no voy a meterme muy de lleno en la parte de reversing sino que voy a dar un pantallazo de cómo funciona.

Bien el primer método que ejecuta la librería 7zip es el CreateObject donde se va a crear el objeto 7zip para luego usar sus métodos de la vtable para trabajar con el archivo.



```
; int __stdcall CreateObject(int, void *, int)
public CreateObject
CreateObject proc near

arg_0= dword ptr 4
arg_4= dword ptr 8
arg_8= dword ptr 0Ch

push esi
mov esi, [esp+4+arg_4]
push edi
mov edi, [esp+8+arg_8]
and dword ptr [edi], 0
push 10h ; size_t
push offset unk_64DC1334 ; void *
push esi ; void *
call _memcmp
add esp, 0Ch
test eax, eax
jz short loc_64DA3282

push 10h ; size_t
push offset unk_64DC1334 ; void *
```

100.00% (-134,41) (123,279) 0000262C 64DA322C: CreateObject (Synchronized with Hex View-1)

Este método básicamente te devuelve el puntero al objeto 7zip

Luego se ejecuta el método open de su vtable, que es el que va a abrir el archivo y va a verificar su header y demás cosas.

```

.rdata:54911590 dd offset sub_548F5D36
.rdata:54911594 dd offset sub_548F77F7
.rdata:54911598 dd offset sub_548F74F0
.rdata:5491159C VTable7zip dd offset sub_548F7704 ; DATA XREF: debug316:06165C18fo
.rdata:5491159C ; sub_548F7676+18fo
.rdata:549115A0 dd offset sub_548F5D28
.rdata:549115A4 dd offset sub_548F77D6
.rdata:549115A8 dd offset Open
.rdata:549115AC dd offset sub_548F634C
.rdata:549115B0 dd offset sub_548F6211
.rdata:549115B4 dd offset sub_548F6BA1
.rdata:549115B8 dd offset sub_548F54E6
.rdata:549115BC dd offset sub_548F684C
.rdata:549115C0 dd offset sub_548F9F03
.rdata:549115C4 dd offset sub_548F9FE2
.rdata:549115C8 dd offset sub_548F6223
.rdata:549115CC dd offset sub_548F6232
0002039C 5491159C: .rdata:VTable7zip: (Synchronized with EIP)

```

El cuarto argumento de esta función open va a ser un puntero a unos punteros que apuntan a funciones de winrar.exe, el 3er es argumento es un puntero a 0x100000, el segundo argumento va a ser puntero a funciones de winrar que esas funciones van a usarse para trabajar el archivo posteriormente ya que son Readfile, Setfilepointer, etc. También este argumento contiene nuestro el HFile de nuestro archivo.

```

debug438:093549FC db 7Bh ; {
debug438:093549FD db 0D0h ; D
debug438:093549FE db 3
debug438:093549FF db 18h
debug438:09354A00 dd offset p_funciones_winrar_TrabajaConArchivo
debug438:09354A04 dd offset p_funciones_winrar_0
debug438:09354A08 dd 1
debug438:09354A0C dd offset off_3C7A34
debug438:09354A10 dd 0BAAD0001h
debug438:09354A14 dd 730h
debug438:09354A18 dd offset unk_1162AE8
debug438:09354A1C dd 0BAADF000h
debug438:09354A20 db 0
debug438:09354A21 db 0
debug438:09354A22 db 0
debug438:09354A23 db 0
debug438:09354A24 db 0Dh
UNKNOWN 09354A18: debug438:09354A18 (Synchronized with EIP)

```

Esta es la imagen del argumento 2.

Y como primer argumento es el puntero a nuestro Objeto 7zip.

Nuestro harness va quedando de la siguiente manera.

```

int Winrar_Base, zip7_Base;

int flag;

CargarLibrerias(&Winrar_Base, &zip7_Base);

HANDLE hFile=CreateFile(argv[1],GENERIC_READ | GENERIC_WRITE,0,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);

if (hFile == INVALID_HANDLE_VALUE)
{
    printf("error al abrir archivo");
}

// Armado de argumentos y ejecucion de funcion CreateObject//
char Retorno_Objeto7zip[8];
memset(Retorno_Objeto7zip,0,8);

char Argumento2_CreateObject[70]="\x69\x0f\x17\x23\xc1\x40\x8a\x27\x00\x00\x00\x06\x00\x60\x00\x69\x0f\x17\x23\xc1\x40\x8a\x27\x00\x00\x00\x06\x00\xa0\x00";
char Argumento1_CreateObject[28]="\x69\x0f\x17\x23\xc1\x40\x8a\x27\x10\x00\x00\x01\x10\x07\x00\x00";

int (*CreateObject)(char*,char*,char*) = (int (__cdecl*)(char*,char*,char*))GetProcAddress((HMODULE)zip7_Base,"CreateObject");

if(CreateObject==NULL){
    printf("error al cargar CreateObject");
}

int Retorno_CreateObject=(*CreateObject)(Argumento1_CreateObject ,Argumento2_CreateObject ,Retorno_Objeto7zip );

```

Este es el armado de createObject, como vemos abro el archivo con CreateFile porque así lo abre winrar.

p_funciones_winrar_TrabajaConArchivo que contiene los punteros a las funciones ReadFile, Setfilepointer,etc hay que reprogramarlos a mano en nuestro harness, porque se preguntaran?

```

LPDWORD v14; // edi
bool v15; // bl
int v16; // edi
int v17; // eax
int v18; // ecx
int v19; // edx
int v20; // esi
DWORD v21; // [esp+Ch] [ebp-14h]
int v22; // [esp+10h] [ebp-10h]
int v23; // [esp+14h] [ebp-Ch]
int v24; // [esp+18h] [ebp-8h]

v4 = thi;
if ( *(BYTE *)(thi + 16) )
{
}
else
{
    v14 = lpNumberOfBytesRead;
    v15 = ReadFile(*(HANDLE *)(thi + 0x14), lpBuffer, nNumberOfBytesToRead, lpNumberOfBytesRead, NULL) != 0;

    result = v15;
}
return result;
}

DWORD __stdcall readfile_1(void *thi, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, int a2)
{

```

[illegible]

```
int p_MetodoOpen=(int)Vtable*0xC;  
int *p_MetodoOpen1=(int*)p_MetodoOpen;  
int (*Open)(int *,Argumento2_Open*,int*,Vtable_Winrar *)=(int (__cdecl *))(int *,Argumento2_Open*,int *,Vtable_Winrar*))p_MetodoOpen1;  
int ret=(Open)(obj7zip,arg2,p_size,VTwinrar);
```

Luego yo llamo a dos métodos más de la Vtable 7zip que son el metodo8 y metodo6 que a grandes rasgos son funciones que van preparando al objeto7zip para posteriormente mandarlo a la función de extracción pero la función de extracción no la hice porque había que reprogramar una función demasiado compleja y hablando con Boken me dijo que no hacia falta, lo principal es que mande la función open que ahí es donde comprueba el header y demás cosas.

```

.rdata:5446158C off_5446158C dd offset sub_5444777F ; DATA XREF: sub_54447676+21to
.rdata:54461590 dd offset sub_54445D36
.rdata:54461594 dd offset sub_544477F7
.rdata:54461598 dd offset sub_544474F0
.rdata:5446159C VTable7zip dd offset sub_54447704 ; DATA XREF: debug352:03815C18to
.rdata:5446159C ; sub_54447676+18to
.rdata:544615A0 dd offset sub_54445D28
.rdata:544615A4 dd offset sub_544477D6
.rdata:544615A8 dd offset Open
.rdata:544615AC dd offset sub_5444634C
.rdata:544615B0 dd offset sub_54446211
.rdata:544615B4 dd offset metodo6
.rdata:544615B8 dd offset func
.rdata:544615BC dd offset metodo8
.rdata:544615C0 dd offset sub_54449F03
.rdata:544615C4 dd offset sub_54449FE2
.rdata:544615C8 dd offset sub_54446223
000203C4 544615C4: .rdata:544615C4 (Synchronized with EIP)

```

Bien primero ejecuta el metodo8, su 1er argumento es un puntero al objeto7zip, 2do manda un 13, y el tercero un puntero a un buffer seteado a cero.

```

int p_Metodo8=(int)Vtable+0x20;
int *p_Metodo81=(int*)p_Metodo8;

int (*Metodo8)(int* , signed int , int *)=(int (__cdecl *)(int* , signed int , int *))p_Metodo81;
int retMetodo8=(Metodo8)(obj7zip,13,pbuffecero);

if(retMetodo8==0){

```

Luego ejecuta el metodo6, sus argumentos son los siguiente: el primero es el puntero al objeto7zip, 2do cero, 3ero una constante que es el 3 y el 4to es un puntero a un buffer creado con Malloc y seteado en cero.

```

int p_Metodo6=(int)Vtable+0x18;
int *p_Metodo61=(int*)p_Metodo6;

int (*Metodo6)(int*,int,int,int*)=(int (__cdecl *)(int*,int,int,int*))p_Metodo61;
int retMetodo61=(Metodo6)(obj7zip,0,3,buffer1);

```

Y por último el método close que libera el objeto7zip y en nuestro harness liberamos los objetos que allocamos.

```

    int p_MetodoClose=(int)Vtable+0x8;

    int *p_MetodoClose1=(int*)p_MetodoClose;
    int (*Close)(int *)=(int (__cdecl *)(int *))p_MetodoClose1;
    (*Close)(obj7zip);

    free(buffer1);
    free(pbuffecero);
    free(arg4);
    free(VTwinrar);
    free(arg2);
    free(MArchivo);

```

Luego que tenemos listo nuestro harness vamos a fuzzear y probar suerte a ver que sale.

Preparando Fuzzing

Minimización

Primero necesitamos un corpus de archivos 7zip y luego minimizarlo con herramientas del mismo winaf1, ¿para qué es esto?

La minimización de los archivos hace que en nuestro corpus queden solamente los archivos que generen rutas nuevas en el código, así ampliamos la cobertura de código y podemos llegar más lejos.

Con winaf1-cmin.py lo minimizamos

```

python winaf1-cmin.py --skip-dry-run -D
C:\DynamoRIO-Windows-7.1.0-1\bin32 -t 100000 -i corpus -o C:\result
-coverage_module harness7zip.exe -target_module harness7zip.exe -target_offset
0x1529 -nargs 2 -- harness7zip.exe @@

```

en la carpeta result es donde va a devolver todos los archivos que generen nuevas rutas.

Luego que hacemos esto nos ponemos a fuzzear.

Ejecutando WinAFL

Algo que aclarar vamos a usar dos comandos, vamos a paralelizar en dos uno maestro y otro esclavo, el maestro se va a ocupar de tareas deterministas como bit flips, byte flips y el esclavo de la tarea sucia y aleatoria.

Conviene hacer esto porque va a hacer varias fuzzing estrategias al mismo tiempo.

Primero ejecutamos al master con -M

```
afl-fuzz.exe -i in -o out -M master -D C:\DynamoRIO-Windows-7.1.0-1\bin32 -t 20000 --  
-coverage_module harness7zip.exe -fuzz_iterations 5000 -target_module harness 7zip.exe  
-target_offset 0x1529 -nargs 2 -- harness 7zip.exe @@
```

En el target_offset hay que poner el offset donde comienza nuestra función del harness.

Y luego ejecutamos al esclavo con -S.

```
afl-fuzz.exe -i in -o out -S Slave01 -D C:\DynamoRIO-Windows-7.1.0-1\bin32 -t 20000 --  
-coverage_module harness7zip.exe -fuzz_iterations 5000 -target_module harness 7zip.exe  
-target_offset 0x1529 -nargs 2 -- harness 7zip.exe @@
```

Lo dejamos un par de horas fuzzeando y estos son los resultados.


```
Administrador: Símbolo del sistema - afl-fuzz.exe -i in -o out [redacted]

+-- process timing -----+
| run time      : 0 days, 14 hrs, 31 min, 6 sec |
| last new path  : 0 days, 7 hrs, 11 min, 21 sec |
| last uniq crash : 0 days, 2 hrs, 59 min, 52 sec |
| last uniq hang  : 0 days, 6 hrs, 30 min, 54 sec |
+-- cycle progress -----+
| now processing : 0 (0.00%) |
| paths timed out : 0 (0.00%) |
+-- stage progress -----+
| now trying : arith 8\8 |
| stage execs : 8771/19.3k (45.38%) |
| total execs : 18.4k |
| exec speed  : 0.32/sec (zzzz...) |
+-- fuzzing strategy yields -----+
| bit flips : 51/2464, 13/2463, 3/2461 |
| byte flips : 0/308, 1/307, 1/305 |
| arithmetics : 0/0, 0/0, 0/0 |
| known ints  : 0/0, 0/0, 0/0 |
| dictionary  : 0/0, 0/0, 0/0 |
| havoc       : 0/0, 0/0 |
| trim        : 0.00%/142, 0.00% |
+-- overall results -----+
| cycles done : 0 |
| total paths : 126 |
| uniq crashes : 45 |
| uniq hangs  : 26 |
+-- map coverage -----+
| map density : 5.76% / 11.63% |
| count coverage : 1.66 bits/tuple |
+-- findings in depth -----+
| favored paths : 73 (57.94%) |
| new edges on  : 96 (76.19%) |
| total crashes : 3789 (45 unique) |
| total tmouts  : 243 (26 unique) |
+-- path geometry -----+
| levels : 2 |
| pending : 126 |
| pend fav : 73 |
| own finds : 37 |
| imported  : 0 |
| stability : 99.88% |
+-----+
Correcto: se terminó el proceso con PID 112.-----+
1 processes nudged

[cpu: 0%]

Símbolo del sistema - afl-fuzz.exe -i in -o out [redacted]

+-- process timing -----+
| run time      : 0 days, 13 hrs, 41 min, 56 sec |
| last new path  : 0 days, 0 hrs, 1 min, 32 sec |
| last uniq crash : 0 days, 0 hrs, 14 min, 36 sec |
| last uniq hang  : 0 days, 0 hrs, 27 min, 40 sec |
+-- cycle progress -----+
| now processing : 30 (5.17%) |
| paths timed out : 0 (0.00%) |
+-- stage progress -----+
| now trying : splice 11 |
| stage execs : 320/512 (62.50%) |
| total execs : 25.7k |
| exec speed  : 0.60/sec (zzzz...) |
+-- fuzzing strategy yields -----+
| bit flips : n/a, n/a, n/a |
| byte flips : n/a, n/a, n/a |
| arithmetics : n/a, n/a, n/a |
| known ints  : n/a, n/a, n/a |
| dictionary  : n/a, n/a, n/a |
| havoc       : 68/2900, 568/13.8k |
| trim        : 0.53%/3842, n/a |
+-- overall results -----+
| cycles done : 0 |
| total paths : 580 |
| uniq crashes : 188 |
| uniq hangs  : 63 |
+-- map coverage -----+
| map density : 6.17% / 11.85% |
| count coverage : 1.91 bits/tuple |
+-- findings in depth -----+
| favored paths : 117 (20.17%) |
| new edges on  : 140 (24.14%) |
| total crashes : 13.4k (188 unique) |
| total tmouts  : 155 (63 unique) |
+-- path geometry -----+
| levels : 2 |
| pending : 560 |
| pend fav : 105 |
| own finds : 458 |
| imported  : 33 |
| stability : 99.94% |
+-----+
Correcto: se terminó el proceso con PID 2804.-----+
1 processes nudged

[cpu: 0%]
```

Ahora toca analizar si algunos de esos crash son explotables.

Quería agradecer a Boken, Ricardo y a toda la comunidad de cracklatinos que gracias a ellos estoy haciendo estos tutos jeje.

Un saludo Gabriel Durdiak.