# Fuzzeando ACDSee Free 1.1.21 con WinAfl

Este tuto vamos a aprender cómo se usa winafl ya que hay muy poca información en la red y tiene sus cosas.

Para empezar, veamos que es winafl:

**¿Qué es el WinAFL?**

AFL es un fuzzer genético guiado por cobertura, que tiene una implementación sólida como una roca y una heurística inteligente que ha demostrado ser muy exitosa en la búsqueda de errores reales en software real.

WinAFL es una bifurcación de AFL para Windows, creada y mantenida por Ivan Fratric (Google Project Zero).

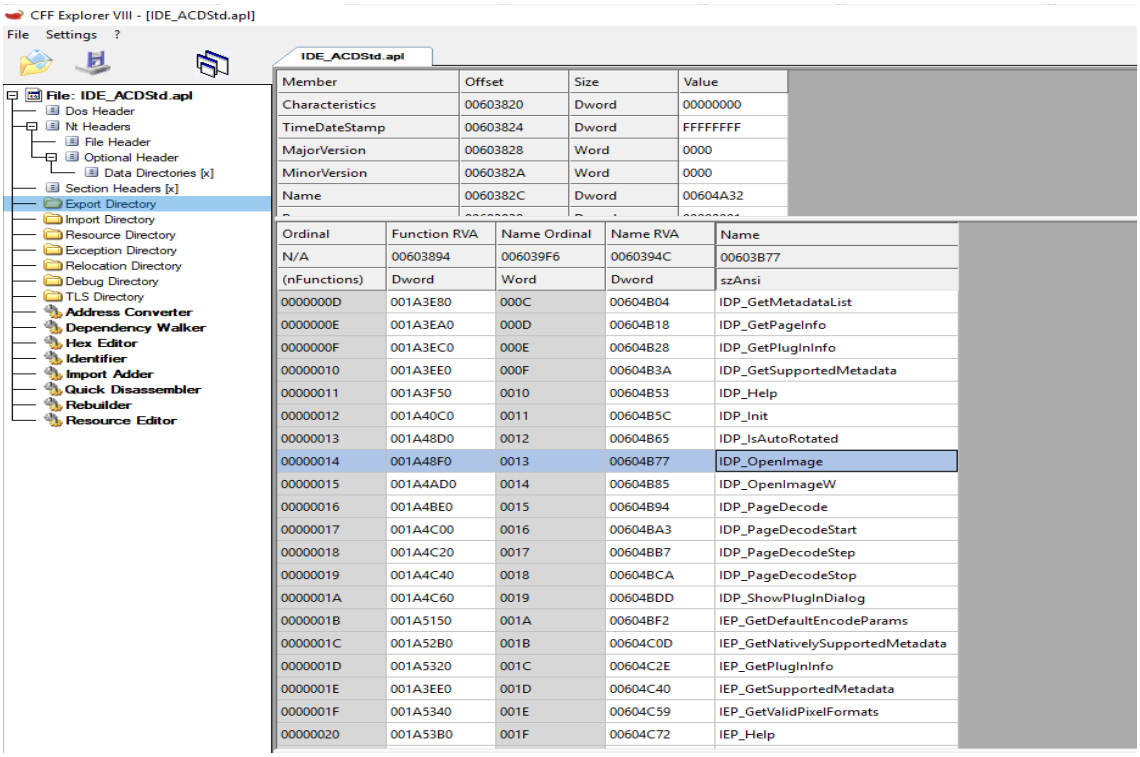
La versión para Windows utiliza un estilo diferente de instrumentación que nos permite fuzzear a binarios de código cerrado, los instrumentadores que se pueden usar son: DynamoRio, Syzygy, Intel PT.

WinAFL es extremadamente efectivo para encontrar errores de formato de archivo, especialmente en formatos binarios comprimidos (imágenes / videos / archivos).

Para más información y descargarlo:
https://github.com/googleprojectzero/winafl

# Atacando ACDsee free

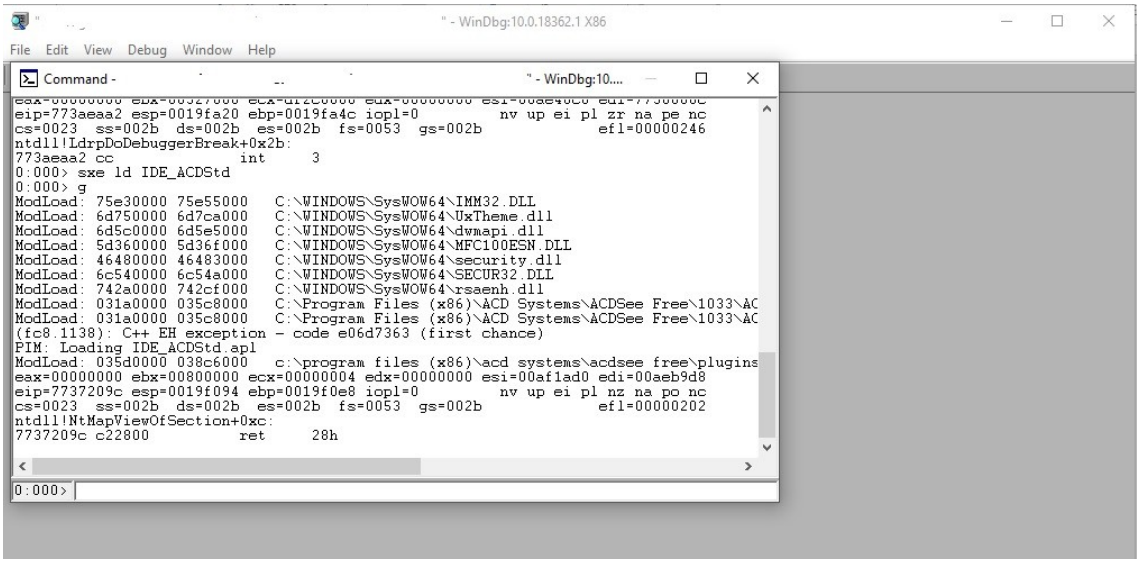
Vamos a atacar este visor de imágenes conocido, buscando encontré el archivo donde contiene las funciones que parsea los distintos formatos de imágenes ese se llama IDE_ACDStd.apl




Bien ahora veamos cuales son las funciones que se usan de ese plugin para parsear los formatos para eso vamos a windbg

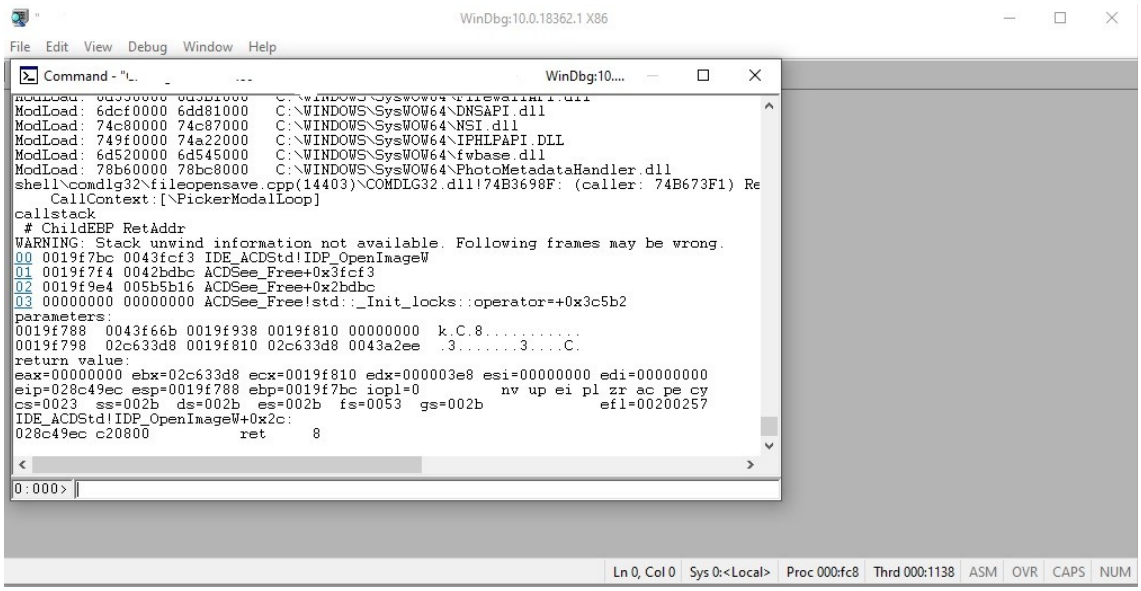
Usamos el comando *sxe ld* IDE_ACDStd para que pare cuando cargue el plugin IDE_ACDStd

Entonces tocamos g corremos el programa, vamos a ver como carga el plugin y para




Ponemos el siguiente comando: *bm /a* IDE_ACDStd*!* "*.echo callstack; k L5; .echo parameters:; dc esp L8; .echo return value: ; pt; "*

*Este comando va a hacer que pare en cada función que plugin ejecute, tocamos g y cargamos la imagen.*




Como vemos ahí paro en la primera función que parsea la imagen que es OpenImageW si seguimos corriendo el programa vemos todas las imágenes que usa.

In BP: IDP_OpenImageW
In BP: IDP_GetImageInfo
In BP: IDP_GetImageInfo
In BP: IDP_GetPageInfo
In BP: IDP_PageDecodeStart
In BP: IDP_PageDecodeStep
In BP: IDP_PageDecodeStep
In BP: IDP_PageDecodeStep
...
In BP: IDP_PageDecodeStep
In BP: IDP_IsAutoRotated
In BP: IDP_IsAutoRotated
In BP: IDP_PageDecodeStop
In BP: IDP_CloseImage

Bien ya tenemos las funciones que usa el programa para parsear los distintos formatos de imágenes.

Ahora con estas funciones vamos a crear el harness.

Un arnés es un programa escrito por nosotros que dispara la funcionalidad que queremos que queremos fuzzear. El arnés incluye una función que será usada como nuestra función objetivo.

Y nuestro harness si o si el trabajo que tiene que hacer es

-Abrir la imagen

-Ejecutar todas las funciones objetivo

-Cerrarse

Nuestro harness va a hacer el siguiente trabajo:

-Carga con LoadLibrary ACDSee.exe y IDE_ACDStd.

-Inicializar con la función IDP_Init.

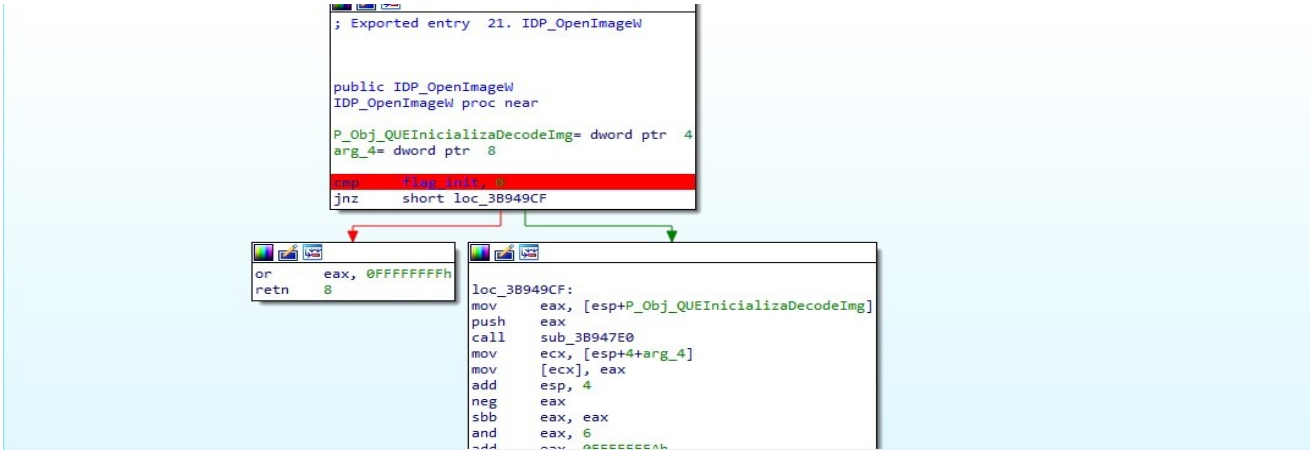
-Abrir el archivo

-Ejecutar IDP_GetImageInfo, IDP_PageDecodeStart

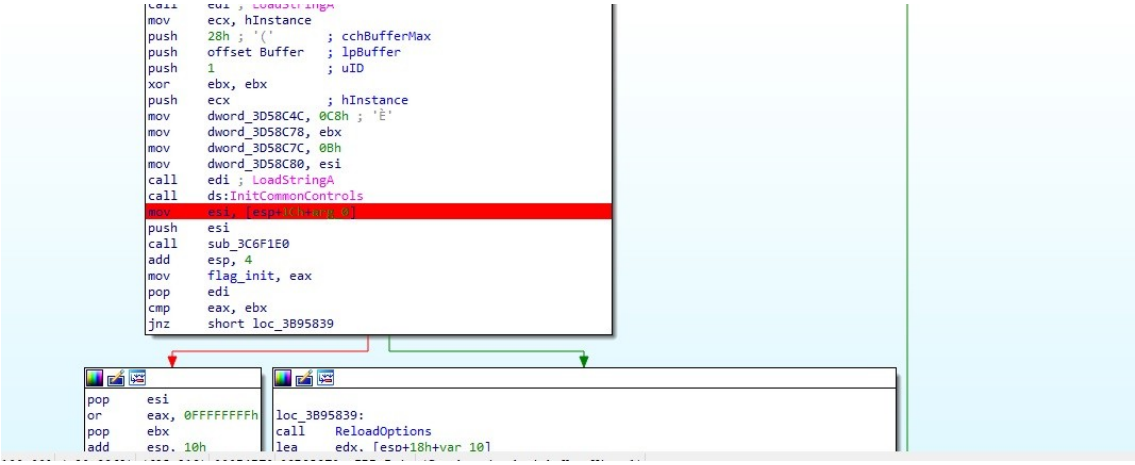
- Ejecutar IDP_PageDecodeStep que es la que va a hacer el trabajo mas pesado esta la vamos a ejecutar en un bucle hasta que devuelva 0xffffe

-Luego ejecutamos IDP_PageDecodeStop IDP_CloseImage.

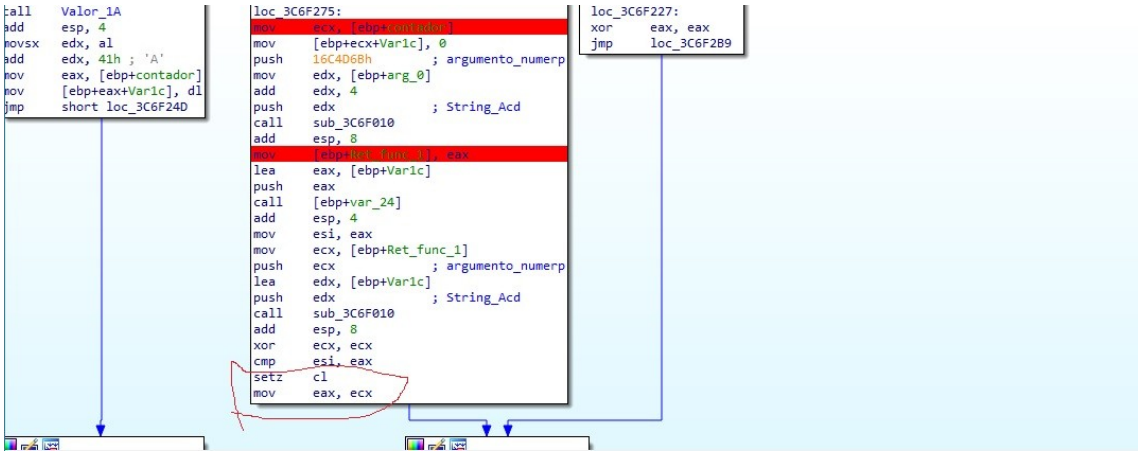
Bien empecemos en IDP_init necesitamos que se ejecute esta función para inicializar variable globales y demás cosas, pero lo más importante poner en 1 la variable global flag_ini que se va a usar en cada función del plugin y que va a comprobar que este en 1 para poder continuar ejecutando como vemos en la imagen.




Bien para poner en 1 la variable global flag_init vamos a ir a la función IDP_init



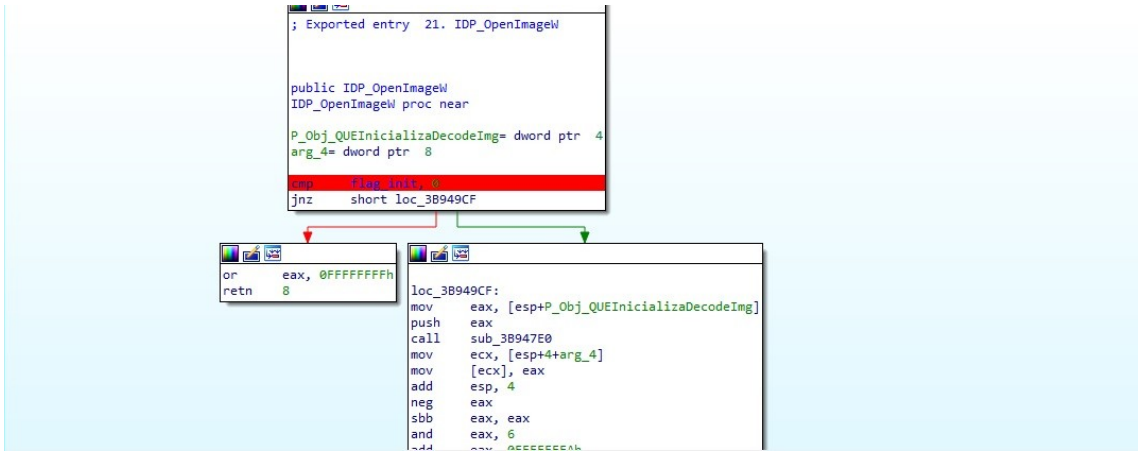

Como ven el valor que devuelve sub_3c6f1e0 se va a pasar a la variable global, vayamos dentro de esa función




Como ven ahí donde esta remarcado con rojo esas dos instrucciones son las encargadas de generar el valor que va a devolver la función, esta función parecería ser una verificación de hash, nosotros tenemos el control para que nos devuelva 1.

Instrucción setz cl setea pone en 1 al registro cl si la comparación esi,eax son iguales, yo para eso en el calle var_24 en el harness arme una función simple que devuelve 0 y guarda el retorno en esi, luego para la segunda función tenemos control para de un argumento para que devuelva cero también y retorna el valor cero a eax y más abajo como vemos compara esi con eax como ve que es igual el flag z se pone en 1 y pone en 1 a cl, luego le pasa ese 1 a eax y que va a ser el valor de retorno que se le va a asignar a flag_init

Luego esa función esta lista, voy a mostrar un poco lo que hace también open_imagew, lleva dos argumentos el primero




Esta imagen como vemos primero comprueba que el flag_init este en 1 luego le pasa 1 argumento que yo le llame objeto que inicializa a la instancia que se va a usar para decodificar

Veamos que contiene ese argumento

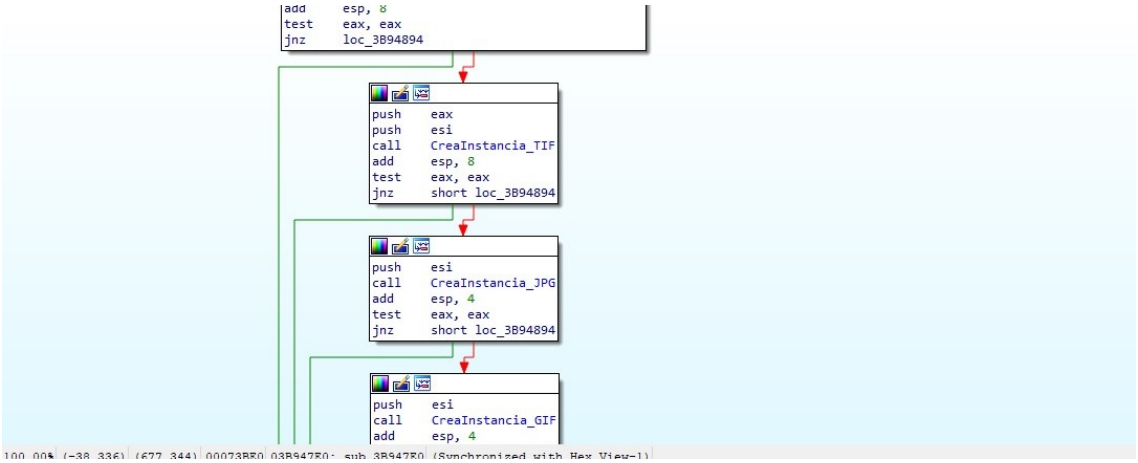
```
Stack[00000E54]:0019F934 db    0
Stack[00000E54]:0019F935 db    0
Stack[00000E54]:0019F936 db    0
Stack[00000E54]:0019F937 db    0
Stack[00000E54]:0019F938 dd 0
Stack[00000E54]:0019F93C dd offset aCUsersGabyDesk_0            ; "C:\\                    .\     \\12322.j"...
Stack[00000E54]:0019F940 dd offset p_imagen
Stack[00000E54]:0019F944 dd 21804h
Stack[00000E54]:0019F948 dd offset Func_Acdsee
Stack[00000E54]:0019F94C dd offset P_ObjetoACDSee
Stack[00000E54]:0019F950 db    0
Stack[00000E54]:0019F951 db    0
Stack[00000E54]:0019F952 db    0
Stack[00000E54]:0019F953 db    0
Stack[00000E54]:0019F954 db    0
Stack[00000E54]:0019F955 db    0
Stack[00000E54]:0019F956 db    0
```

Vemos que contiene el path de la imagen, la variable que llame p_imagen ahí contiene los opcodes de la imagen o como se diga jeje, abajo el size de la imagen, luego una función de ACDsee free y un objeto que se va a usar para comprobar el tamaño de la imagen.

Esto se va a usar en la siguiente función para crear la instancia y se le va a pasar toda esa información a ella.

Cuando entra a la función pasa el argumento 1 y va comprobando de que formato es la imagen

Cuando encuentra de que formato es se crea la instancia a esa imagen para después usarla para decodificar



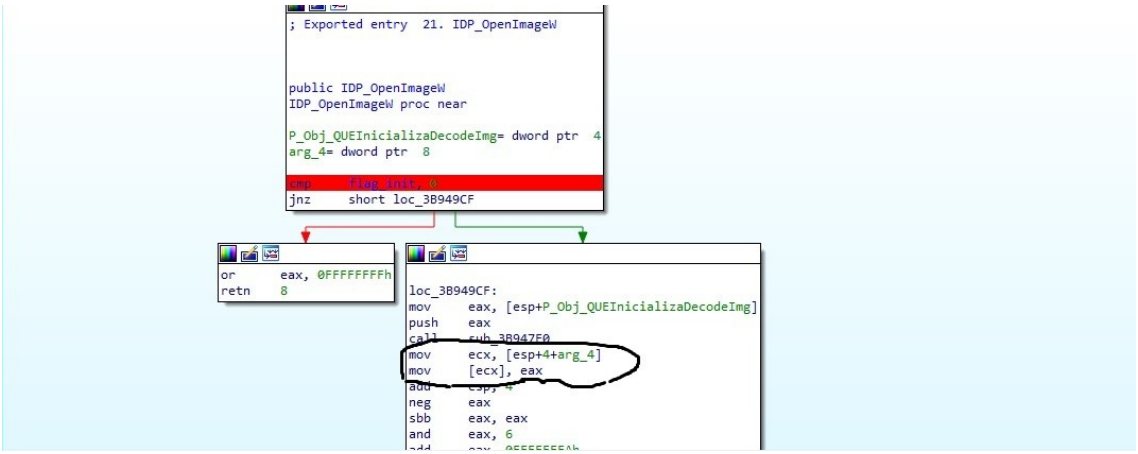

La instancia quedaría asi:

```
debug194:03C8DAAC db 0AFh ; ¯
debug194:03C8DAAD db  46h ; F
debug194:03C8DAAE db    0
debug194:03C8DAAF db  1Ch
debug194:03C8DAB0 dd offset ??_7DecodeJPG@@6B@          ; const DecodeJPG::`vftable'
debug194:03C8DAB4 dd 0
debug194:03C8DAB8 dd offset aCUsersGabyDesk_0           ; "C:\\U      `\12322.j"...
debug194:03C8DABC dd offset p_imagen
debug194:03C8DAC0 dd 21804h
debug194:03C8DAC4 dd offset Func_Acdsee
debug194:03C8DAC8 dd offset P_ObjetoACDSee
debug194:03C8DACC dd offset p_imagen
debug194:03C8DAD0 db 0
debug194:03C8DAD1 db    0
debug194:03C8DAD2 db    0
debug194:03C8DAD3 db    0
debug194:03C8DAD4 db    0
```

Va a contener la Vtable que contiene las funciones que se va a usar de ahora en más para parsear la imagen, y como vemos el primer argumento le paso los punteros que nombre arriba

Ahora el argumento 2 es donde se va a guardar el puntero al Objeto Decoder para posterior usarlo en las demás funciones.

La función retorna el puntero al objeto como vemos.




Luego tenemos GetImage_info que no es muy relevante, lleva dos argumentos el puntero al Instancia Decoder y un buffer puesto en cero que ahí va a retornar los valores de información.

```
    int (*IDP_GetImageInfo)(int  ,obj_GetImageInfo*) = (int (__cdecl *)(int  ,obj_GetImageInfo *))GetProcAddress(ide1,"IDP_GetImageInfo");

    if(ptr==NULL){
        printf("error al carga IDP_GetImageInfo");
}

int RetImgInfo=(*IDP_GetImageInfo)(Buf[0],&get_img_arg);
```

Buf[0] es donde está el puntero a el objeto decoder, en ese  if  recién veo que me equivoque de nombre de variable jaja.

Luego ejecutamos la función IDP_PageDecodeStart que inicializa lo que es el proceso de decoder cargando lo necesario.

```
int (*IDP_PageDecodeStart)(int,Objeto_DecodeStart*,char*) = (int (__cdecl *)(int  ,Objeto_DecodeStart *,char*))GetProcAddress(ide1,"IDP_PageDecodeStart");

    if(IDP_PageDecodeStart==NULL){
        printf("error al carga IDP_PageDecodeStart");
}


char *buffer_vacio=(char*)malloc(0x2710);

memset(buffer_vacio,0,0x2710);
(*IDP_PageDecodeStart)(Buf[0],&obj_DecodeStart,buffer_vacio);
```

Luego ejecuta la función IDP_PageDecodeStep que es la función mas pesada la encargada de hacer el decoder esta función se va a ejecutar hasta que devuelva FFFFFFFE

```
char *obj_DecodeStep=(char*)malloc(0x2710);

memset(obj_DecodeStep,0,0x2710);


 int (*IDP_PageDecodeStep)(int,char*) = (int (__cdecl *)(int  ,char*))GetProcAddress(ide1,"IDP_PageDecodeStep");

    if(IDP_PageDecodeStep==NULL){
        printf("error al carga IDP_PageDecodeStep");
}

while(retorno!=0xFFFFFFFE){


    retorno=(*IDP_PageDecodeStep)(Buf[0],obj_DecodeStep);
   __asm("pushl %ecx");
   __asm("pushl %ecx");
   printf("IDP_PageDecodeStep cargo con exito\n");


}
```

Esos push ecx los puse porque si no cada vez que ejecutaba el bucle dentro de la función DecodeStep llegaba un momento en que me sobrescribía las demás variables y crasheaba, con esos push los solucioné.

Aclaro el argumento que se ve ahí Buf[0] que está en esta y las demás funciones es el puntero a la instancia decoder que se crea en la función OpenImageW.

Luego ejecutamos las funciones IDP_PageDecodeStop y IDP_CloseImage

Bien ya tenemos el harness ahora tenemos que fuzzear.

Lo primero que necesitamos para fuzzear son los test cases que afl nos proporciona un corpus de imágenes https://lcamtuf.coredump.cx/afl/demo/

Luego con esas imágenes hay que utilizar la herramienta que nos proporciona winafl que es afl-cmin que sirve para minimizar el corpus.

# Minimizando el corpus de imágenes

para que se minimiza el corpus y que diferencia existe en mandar el corpus de imágenes sin minimizar?

Voy a poner una explicación de Boken que está bastante bien explicada:

los testcases sirven para hacer que el binario pase por ciertos basic blocks, es decir que se ejecute cierto código. Ese testcase base se va mutando para ver si se provoca una excepción por esas porciones de código.

un fuzzer pretende cubrir cuanta más cantidad de código mejor, por lo que te interesan testcases que hagan que se ejecuten cuantas más porciones de código.

por ejemplo...

si estas fuzzeando un file parser, como es tu caso con el acdsee,

sí metes como testcase un fichero con:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAa

seguramente no pase del primer basic block, que comprueba el magic del file

vera que AAAA no coincide con GIF, JPG, ni ningún otro magic que se usa como identificador en los primeros bytes de los ficheros, y te tirara un error, ese testcase a modo de cobertura de código (hacer que pase por cuanto más código posible) es igual de bueno/malo que otro testcase que sea:

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBb, otro testcase, que comience por

GIF89a.....

conseguirá que se ejecute más código y más basic blocks, ya que pasara la primera comprobación, llegara a la zona que identifica que es un GIF y saltara a la parte de parser de header de ficheros GIF. Esto ha conseguido más cobertura y es un testcase mejor para fuzzing

afl-minimize, lo que hace es eliminar todos los testcase que no aumenten el número de basic blocks recorridos por lo que con los AAAAAAAAAAAAAA ,BBBBBBBBBBBBB, cogerá el primero y el resto los elimina, DE ESTA FORMA evitas fuzzear cientos de miles de veces un testcase, que ya ha sido fuzzeado y evitándote perder N veces más de tiempo.

en caso de que tengas 3 testcase como estos:

AAAAAAAAAAAAAAAAAA

BBBBBBBBBBBBBBBBBB

CCCCCCCCCCCCCCCCCCCC

pues si haces afl-minimize, te dejara solo uno, y habrás ahorrado 3 veces más de tiempo en tu fuzzing, tiempo que empleara en otros 2 testcase que hacen una cobertura de código distinta y aumenta las probabilidades de que provoques un crash.

**Vamos a minimizar el corpus de imágenes con este comando**

python winafl-cmin.py --skip-dry-run -D C:\DynamoRIO-Windows-7.1.0-1\ bin32 -t 100000 -i corpus -o C:\Users\nex\Desktop\result  -covtype edge -coverage_module IDE_ACDStd.apl -target_module  HarnessOpimizado.exe -target_offset 0x1529  -nargs 2 – HarnessOpimizado.exe @@

en el -i se pone la dirección de la carpeta donde esta nuestro corpus en -o la dirección y nombre de la carpeta que se va a crear y almacenar los test case minimizados en -D la ruta de dynamorio donde se encuentre el drrun.exe.

# Fuzzeando con Win-afl

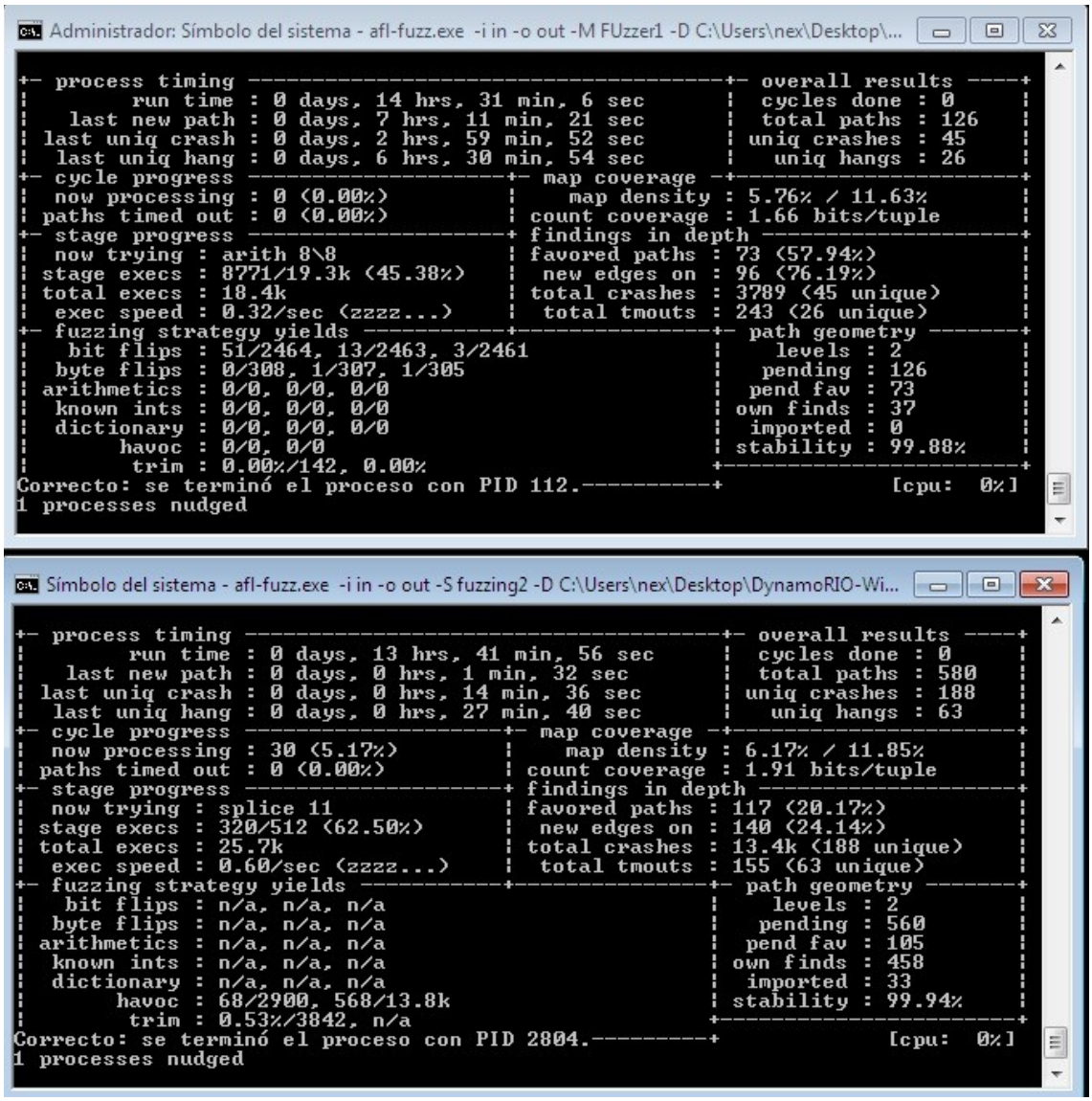
una vez que ya tenemos los casos minimizados vamos a fuzzear con win-afl con el siguiente comando

afl-fuzz.exe -i in -o out -D C:\DynamoRIO-Windows-7.1.0-1\bin32 -t 20000 -- -coverage_module IDE_ACDStd.apl -fuzz_iterations 5000 -target_module HarnessOpimizado.exe  -target_offset 0x1529 -nargs 2 -- HarnessOpimizado.exe @@

Bueno acá expliquemos un poco -i es la carpeta donde tenemos nuestro corpus minimizado -o la carpeta donde la va almacenar la campaña y va a guardar nuestros crashes en -D la ruta donde se encuentra ddrun.exe de dynamorio -target_module el nombre de nuestro harness  y -target_offset es el offset donde se encuentra la función a fuzzear.

Importante: la función a fuzzear tiene que mandar la ruta de los test case como argumento para que funcione.

Bueno tiramos el comando y lo dejamos unas horas a ver que sale



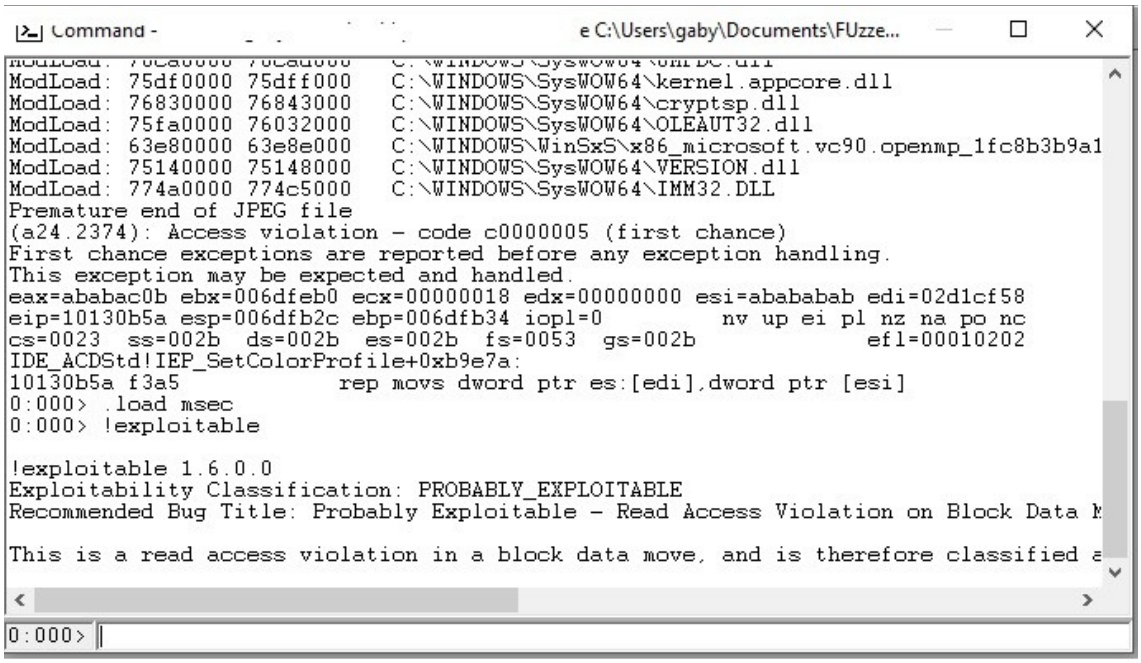

Como vemos el fuzzer descubrió un total de 706 rutas nuevas como vemos en total paths

Y un total de 233 crash únicos como vemos en total crashes

.

Vamos a ver alguno porque son demasiados jeje

Para ver los crashes vamos usar windbg y una herramienta que se llama msec




Como vemos aparentemente este es un bof que esta reportado.

Printer-Friendly View

| CVE-ID | |
|---|---|
| **CVE-2019-13249** | Learn more at National Vulnerability Database (NVD)<br>• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information |
| **Description** | |
| ACDSee Free 1.1.21 has a User Mode Write AV starting at IDE_ACDStd!IEP_SetColorProfile+0x00000000000b9e7a. | |
| **References** | |
| **Note:** References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.<br>• MISC:https://github.com/apriorit/pentesting/blob/master/bugs/acdsee/0x00000000000b9e7a.md | |
| **Assigning CNA** | |
| MITRE Corporation | |
| **Date Entry Created** | |
| **20190704** | Disclaimer: The entry creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE. |
| **Phase (Legacy)** | |
| Assigned (20190704) | |
| **Votes (Legacy)** | |
| **Comments (Legacy)** | |

bueno esto es un poco del uso de winafl yo recién ahora empecé a aprender mientras mas vaya aprendiendo voy a ir compartiéndoles con más tutos.

Quiero agradecer a Boken y a todos CLS que me ayudaron a poner en funcionamiento el fuzzer que la verdad para los nuevos es un poco complicado jeje.

Un saludo Gabriel Durdiak