EXPLOITING Y REVERSING

USANDO HERRAMIENTAS

GRATUITAS (PARTE 21)

Seguimos con el tema estructuras ahora veremos cómo manejarlas con Radare.

En la parte 20 anterior vimos el ejercicio CON ESTRUCTURAS en IDA FREE/

EJERCICIO CON ESTRUCTURAS

password =a

<https://drive.google.com/file/d/1i_PKAY5ustTHjPdoM75ffwfCmXH9cDb9/view?usp=sharing>

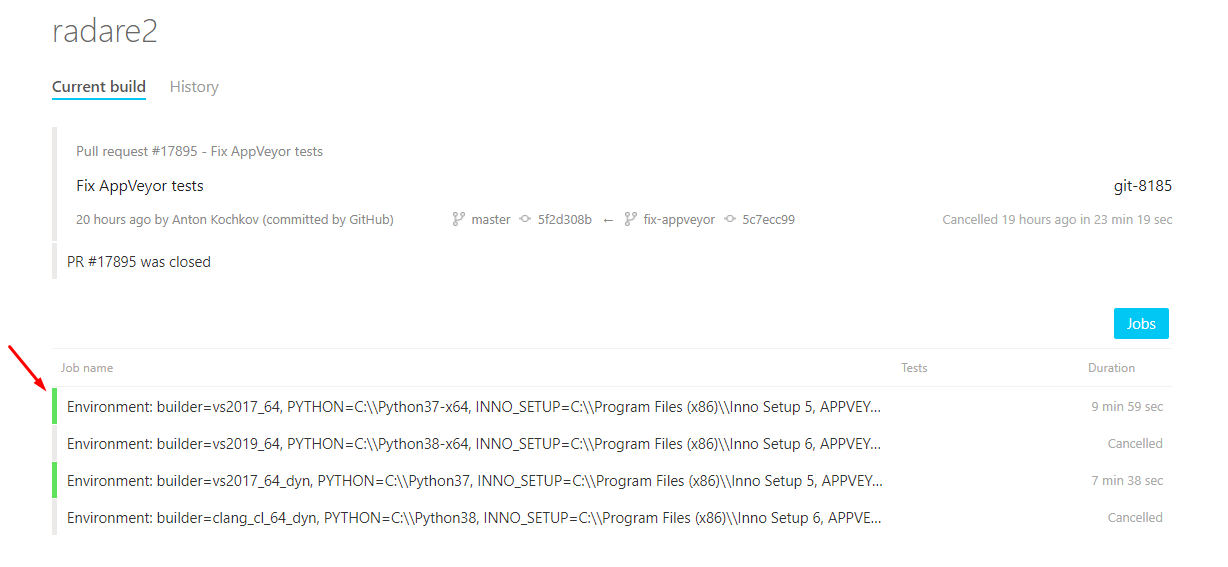
El mismo tiene su código fuente adjuntado.

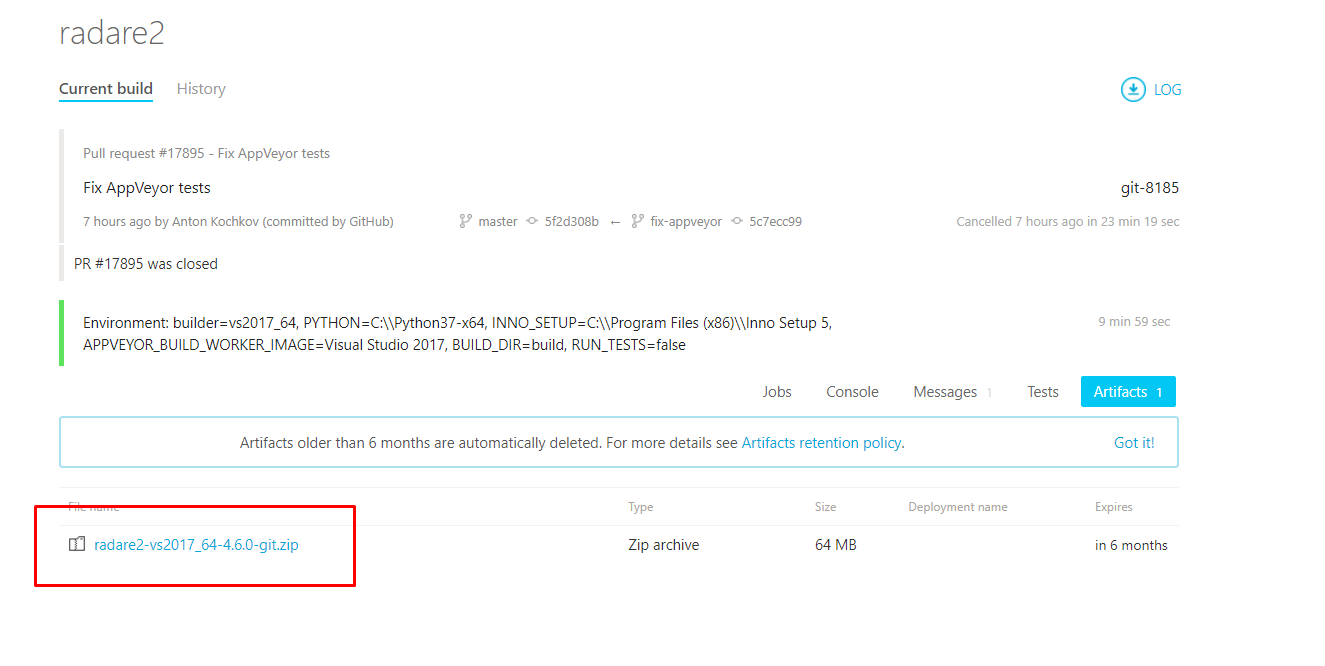


Abrimos una consola de Windows en la carpeta donde esta el ejecutable y los símbolos y cargamos el ejecutable en radare2. (recuerden de tratar de bajar siempre el ultimo build como enseñamos en las partes anteriores)

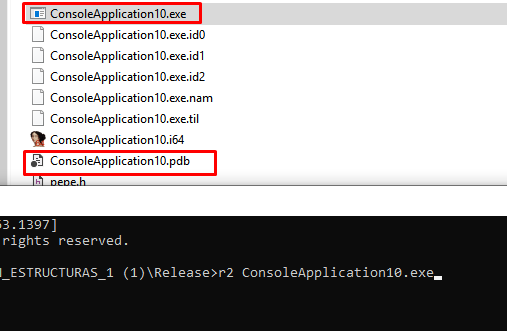
<https://ci.appveyor.com/project/radareorg/radare2>

y buscar el más nuevo BUILD en verde, entrar ir a ARTIFACTS y bajarse el zip con los ejecutables compilados más nuevos.





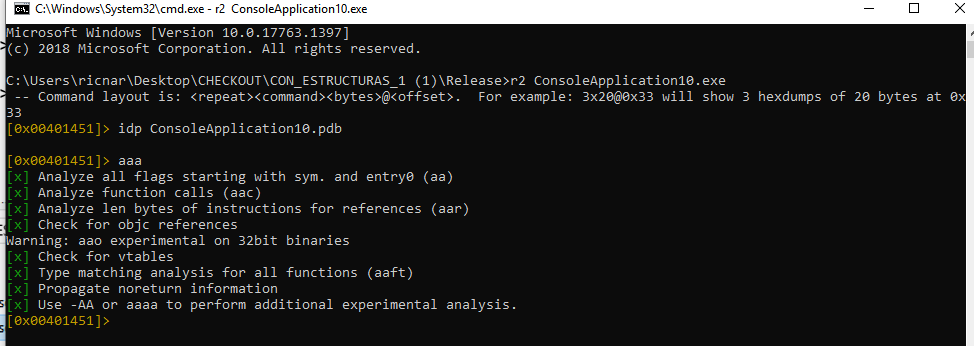
Buscamos el ejecutable y lo cargamos.



Cargaremos los símbolos y analizamos

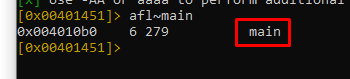
**idp ConsoleApplication10.pdb**

**aaa**



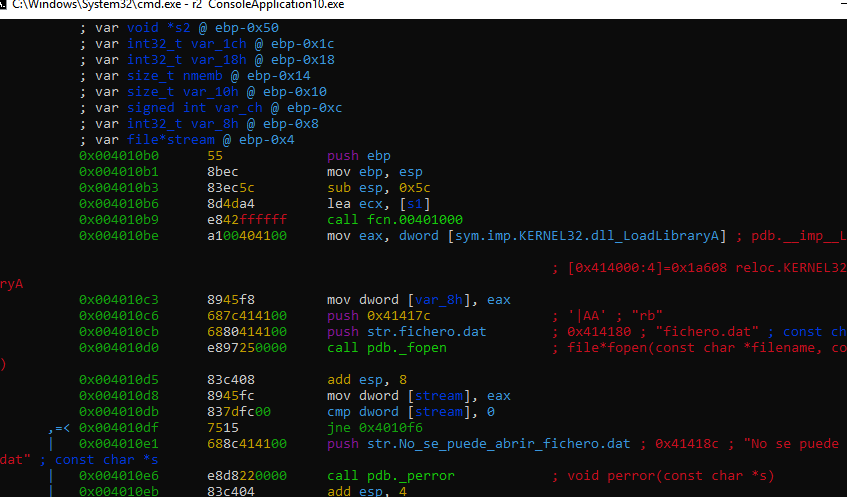
Buscamos el main

**afl~main**



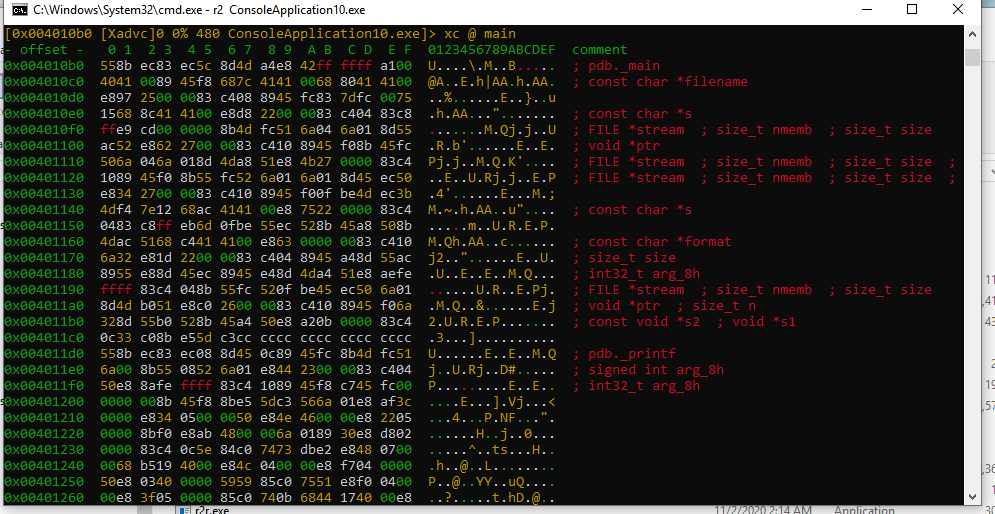
**s main**

**pdf**



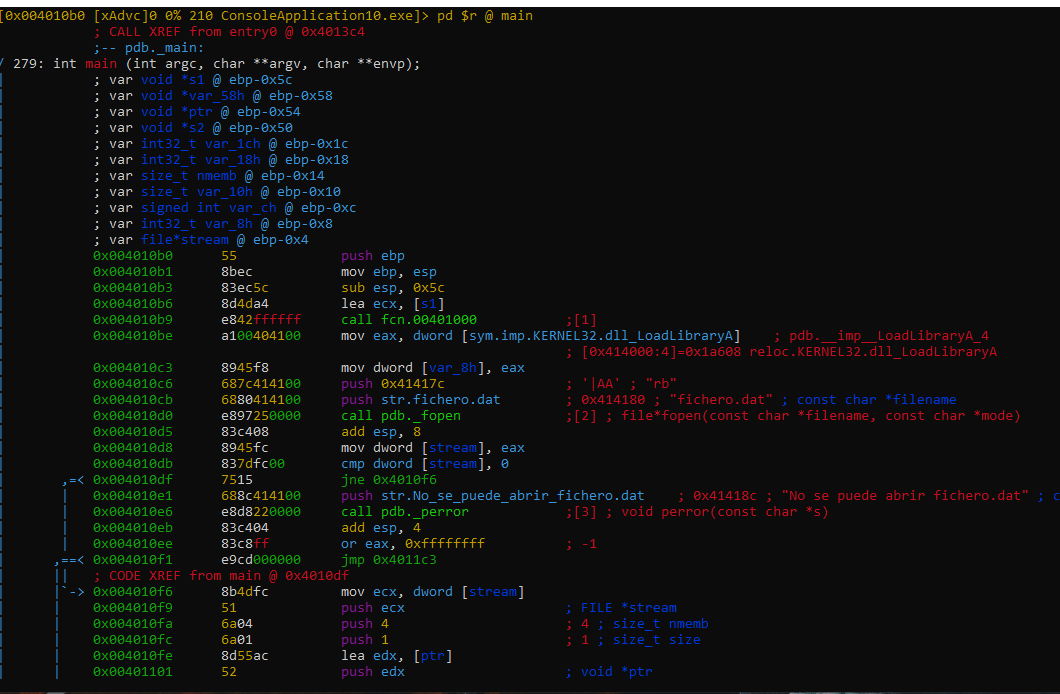
Hemos visto cómo usar radare en vista clásica,usando CUTTER, y las opciones de VISUAL MODE y VISUAL MODE PANELS, creo que para mí la más cómoda es VISUAL MODE sin los paneles, en este tutorial vamos a usar ese modo que se entra con **V** solamente y si tenemos que usar algún comando de radare se ingresa con dos puntos “**:” .**

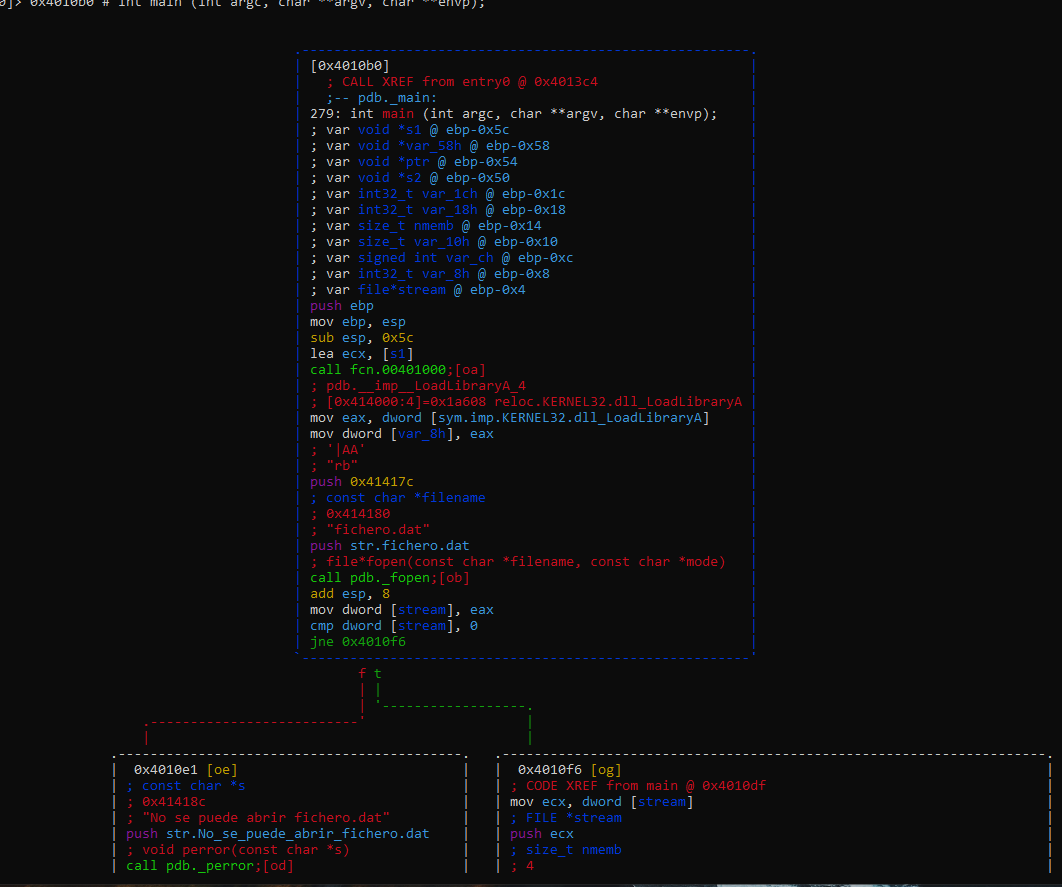
Podemos ir con V al modo visual sin paneles.



Vemos que primero nos muestra la visualización hexadecimal vamos rotando, apretando “**p”** entre las distintas vistas hasta llegar al listado desensamblado.

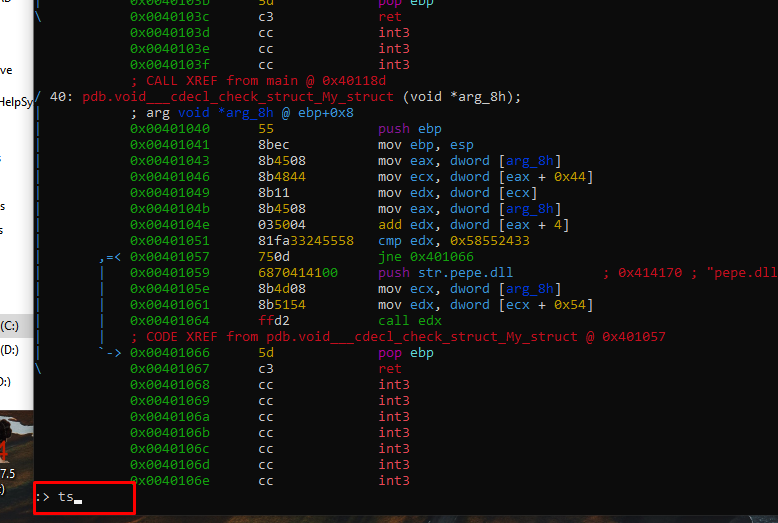
Una vez que estoy en el listado desensamblado, acá mismo si apreto la barra espaciadora veo los gráficos de la función.

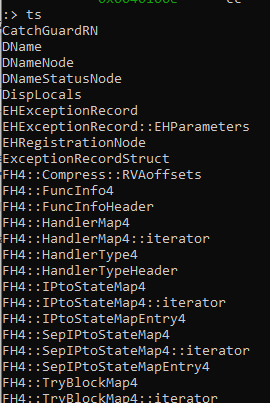




Lo cual está bueno después de estar en modo gráfico, con p se puede cambiar el modo de visualizar hay varios diferentes.

Bueno ahora cargaremos las estructuras que no tenemos, podemos listar los tipos que acepta con **ts**

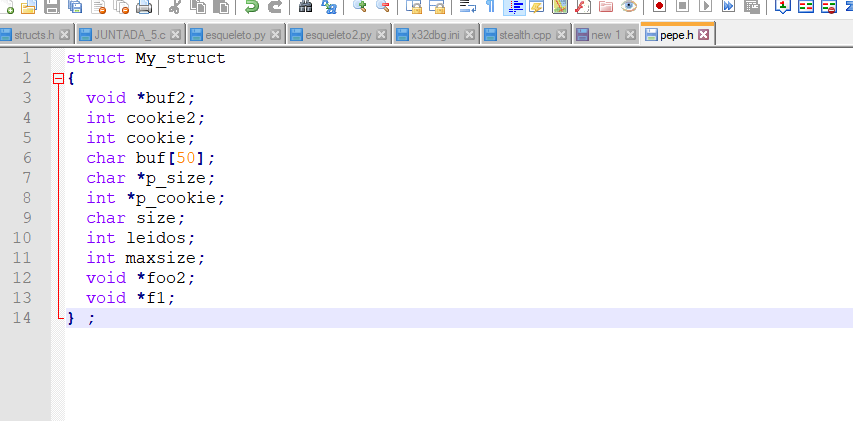


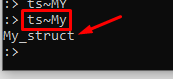


Creo un archivo .h con la estructuras que quiero agregar cuando lo agrego con **to nombre del archivo.**

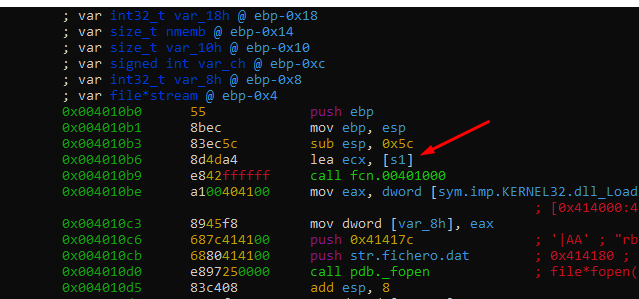
**to pepe.h**

A veces me da algún error por no usar los tipos que acepta radare, así que lo voy acomodando y pasando los errores hasta que lo acepta.

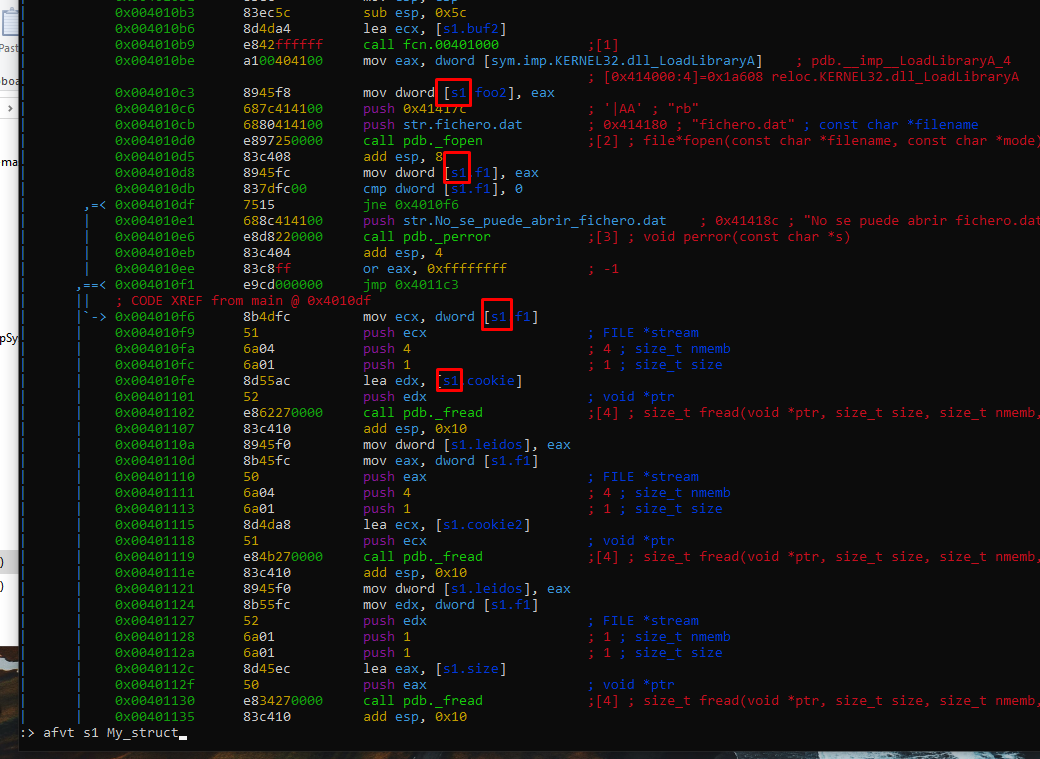




Ahí lo agrego, voy a cambiar el tipo de la variable s1 que habíamos visto en el reversing anterior que era del tipo **My\_struct**.



**afvt s1 My\_struct**

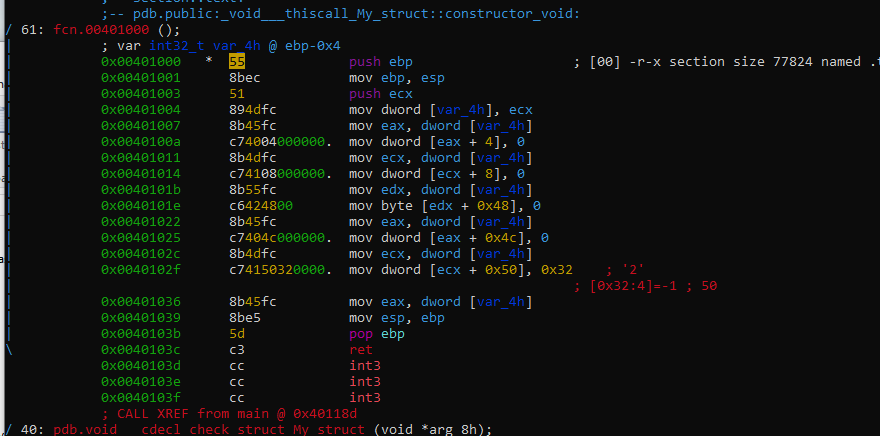


Vemos que ahí asignó los campos de las estructuras correctamente.

Si no estamos en MODO CURSOR (al que se entra y sale con C) apretamos la tecla / podemos resaltar alguna variable o lo que queramos.



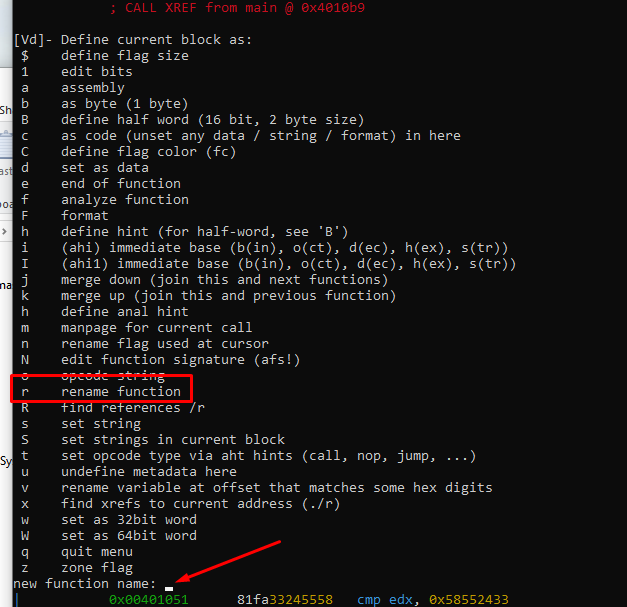
Entramos al modo cursor con C bajamos con las flechas hasta la función para mirar el CALL al que miramos con ENTER.

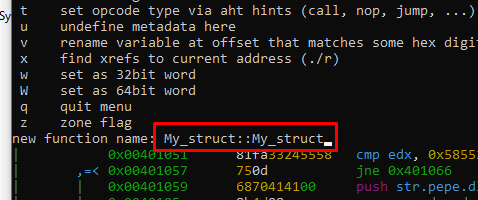


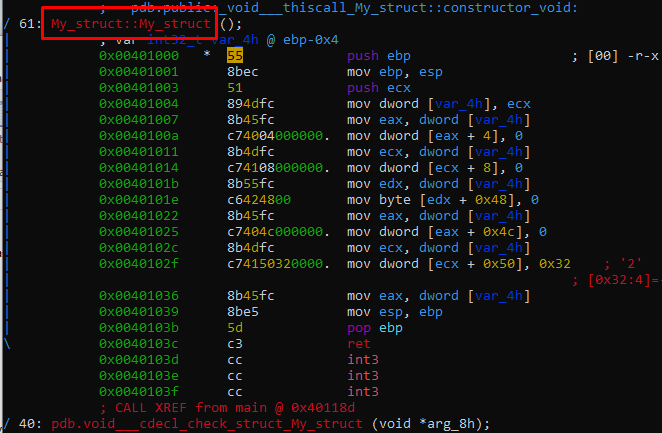
Renombrábamos la función actual con

**afn My\_struct::My\_struct**

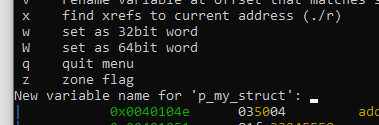
pero también en visual mode se puede hacer directo apretando d y luego entre las opciones elegimos r y ponemos el nuevo nombre



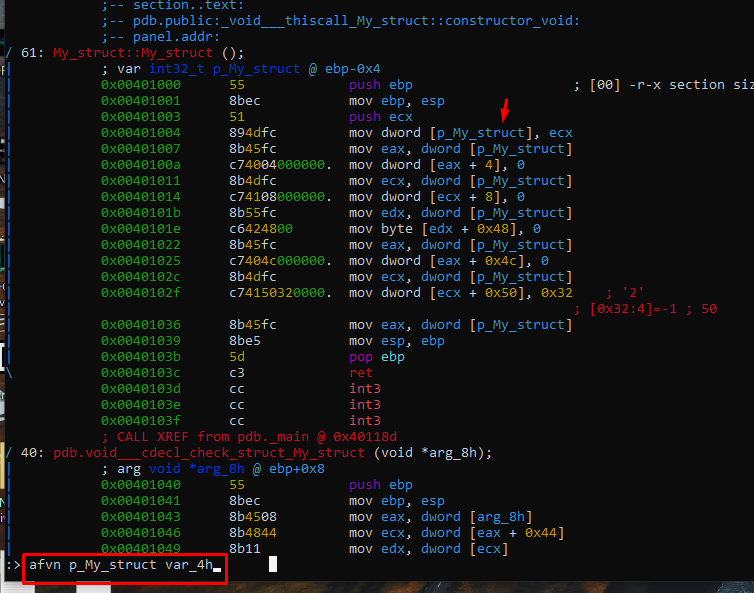




Y la variable también la puedo renombrar llevando el cursor a esa línea y apretando **d** y luego **n**.

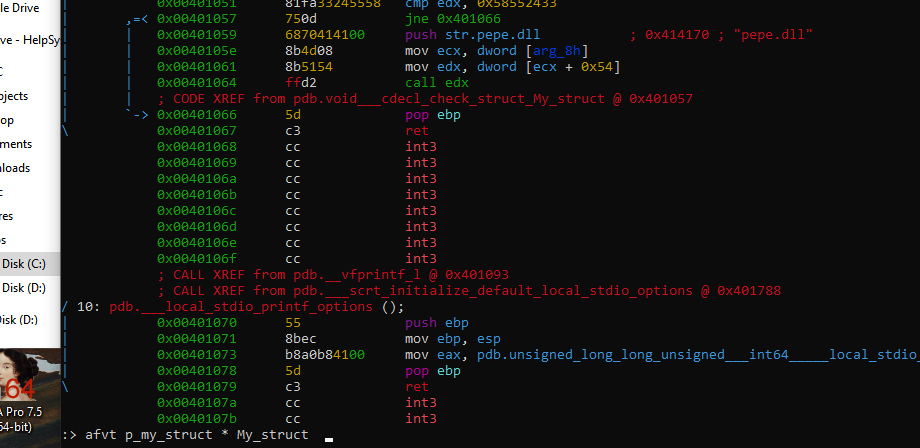


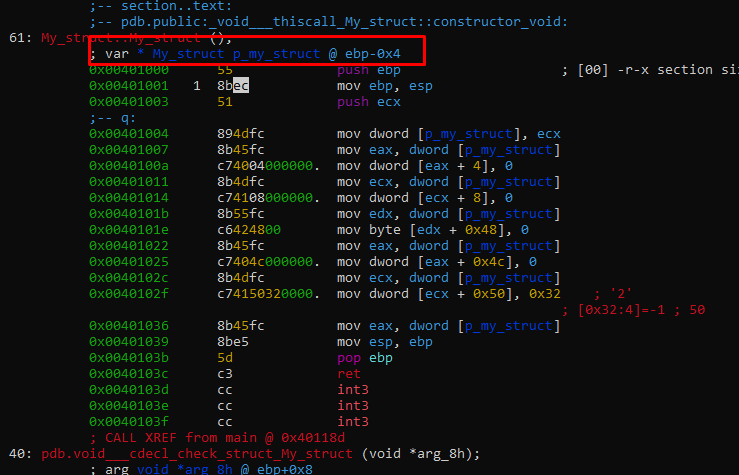




Ahora falta cambiarle el tipo con no encontré como hacerlo directamente en visual mode así que tipeamos **:** y luego.

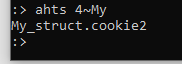
**afvt p\_My\_struct \* My\_struct**





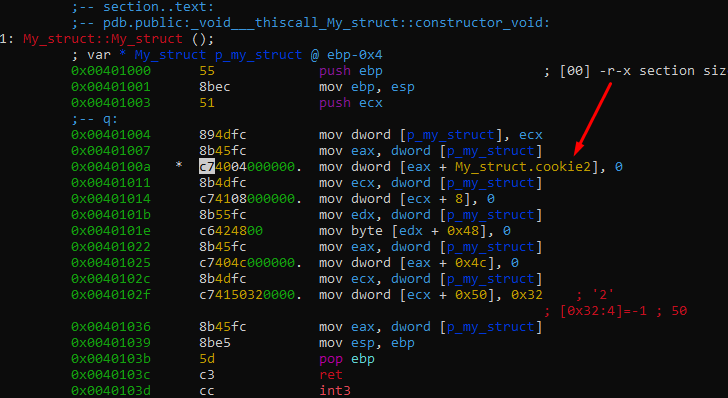
Equivalente al comando T de IDA es **ahts** offset para que muestre el listado de estructuras que tienen por ejemplo offset 4 y de ahí elegimos la que queremos asignar

**ahts 4**



Le puse filtro porque si no salen muchos resultados, ahí veo que el campo de **My\_struct** correspondiente al offset 4 es **cookie2**.

**aht My\_struct.cookie2 @ 0x0040100a**

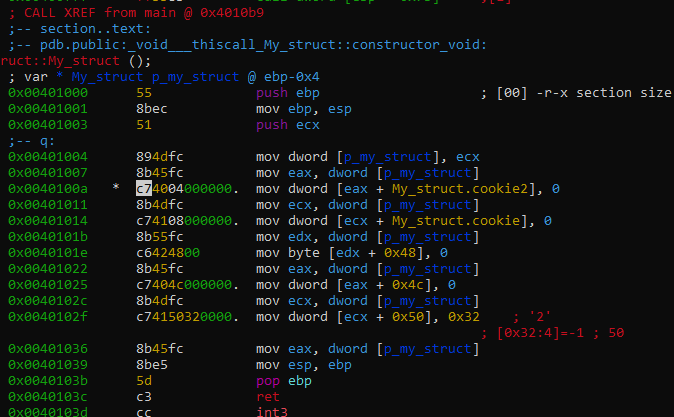


De esa forma podemos asignar todos los offset correspondientes a las estructuras.

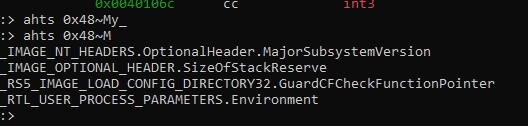
:> **ahts 8**



:> **aht My\_struct.cookie @ 0x00401014**

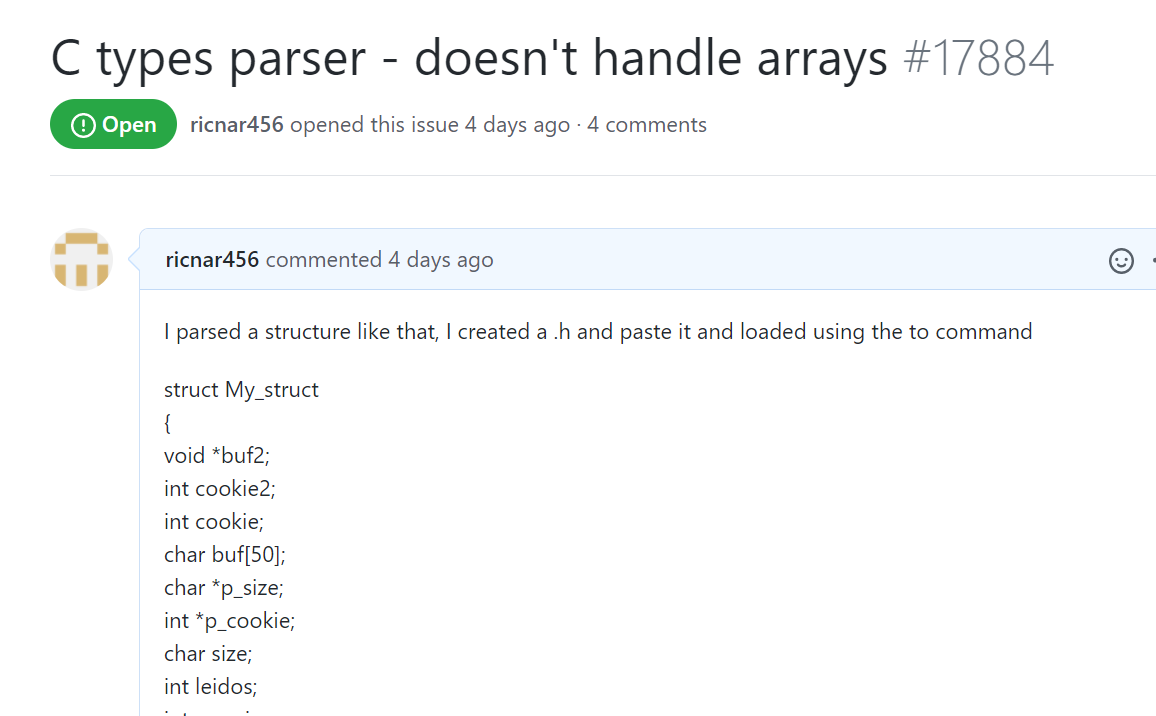


Veo que el 0x48 no me lo lista hay un bug que ya comuniqué.



Bueno abrí un ticket por eso

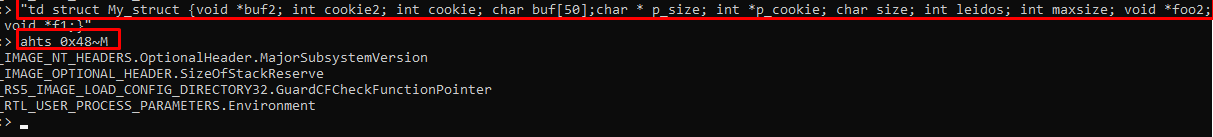
https://github.com/radareorg/radare2/issues/17884



También se puede ingresar la estructura con un comando de una sola línea con el comando **td** delante y entre comillas.

**"td struct My\_struct {void \*buf2; int cookie2; int cookie; char buf[50];char \* p\_size; int \*p\_cookie; char size; int leidos; int maxsize; void \*foo2; void \*f1;}"**

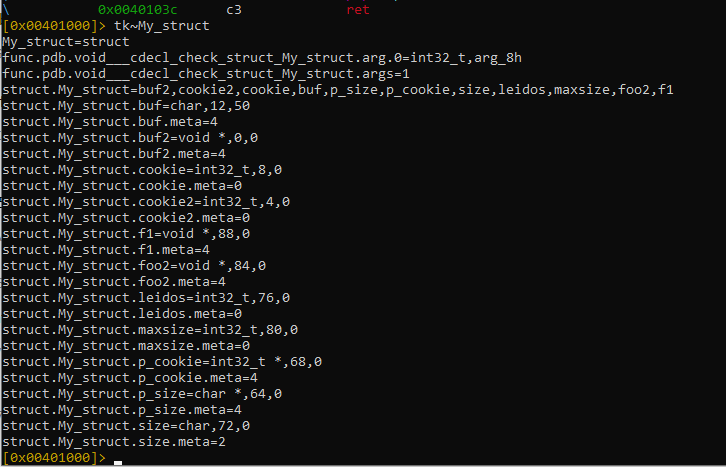
Vemos que la acepta, aunque el bug esta también allí



El comando **tk** usado de esta forma

**tk~My\_struct**

Nos permite ver la estructura interna, como fue parseada y veamos que allí esta correcta el problema es que no lo deja asignar.



Eso significa que el campo buf es del tipo **char** que esta en el offset 12 y que su largo es de 50 elementos o sea 50 chars, así que esta bien, por lo cual hasta que fixeen el bug, podemos seguir usando este listado ya que provee los nombres, offset y size correctos de cada campo.

Quitando las líneas que dicen meta que no nos importan queda así

My\_struct=struct

struct.My\_struct.buf=char,12,50

struct.My\_struct.buf2=void \*,0,0

struct.My\_struct.cookie=int32\_t,8,0

struct.My\_struct.cookie2=int32\_t,4,0

struct.My\_struct.f1=void \*,88,0

struct.My\_struct.foo2=void \*,84,0

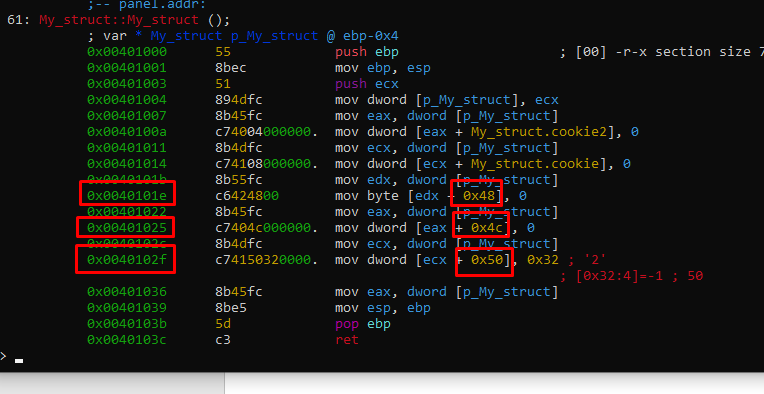
**struct.My\_struct.leidos=int32\_t,76,0**

**struct.My\_struct.maxsize=int32\_t,80,0**

struct.My\_struct.p\_cookie=int32\_t \*,68,0

struct.My\_struct.p\_size=char \*,64,0

**struct.My\_struct.size=char,72,0**



Así que los campos que debemos asignar son 0x48, 0x4c y 0x50 pasados a decimal **72, 76, 80**.

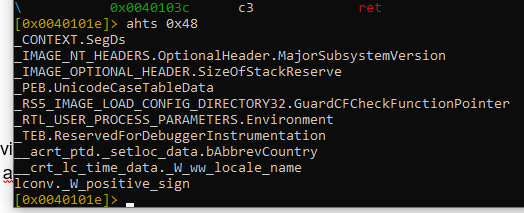
**struct.My\_struct.size=char,72,0**

**struct.My\_struct.leidos=int32\_t,76,0**

**struct.My\_struct.maxsize=int32\_t,80,0**

Bueno como quiero terminar el tutorial y ya vimos la forma que debería funcionar, realmente solo con **ahts** offset nos debería listar las estructuras elegibles para ese offset y el nombre de los campos y con **aht** se asigna, mientras arreglan el bug agregare como comentarios los nombres de los campos que no me parsea.

O sea, para asignar, si no hubiera bug, en el campo en 0x40101e debería hacer



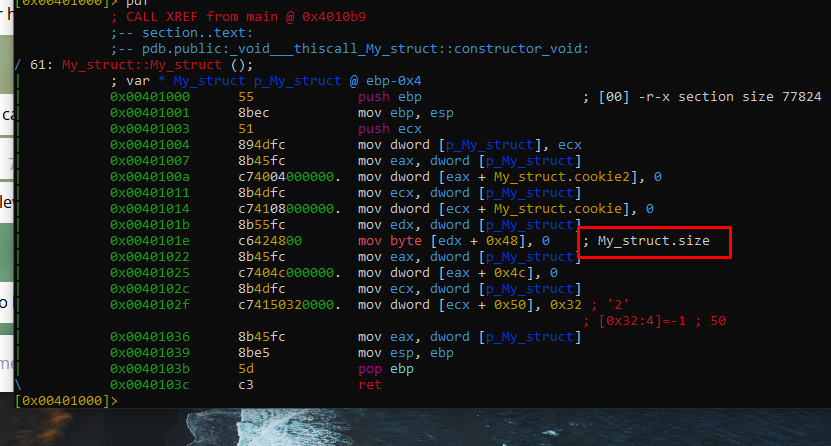
Y ahí debería salir listado el campo **My\_struct.size** que esta en el offset **72.**

Y luego para asignarlo

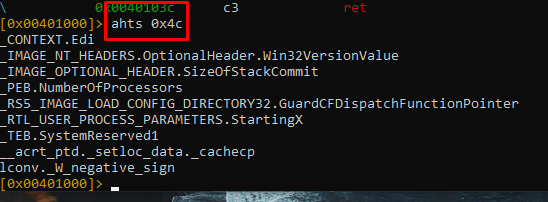
**aht My\_struct.size @ 0x0040101e**

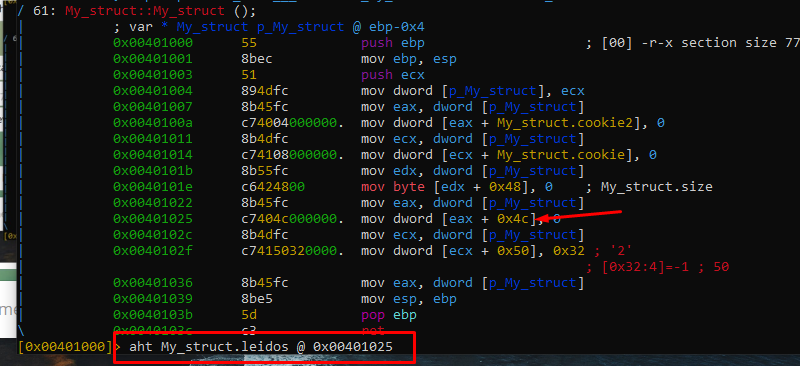
Pondremos un comentario en esa línea solo cambiando **aht** por **CCu** que es el comando para agregar comentarios.

**CCu My\_struct.size @ 0x40101e**



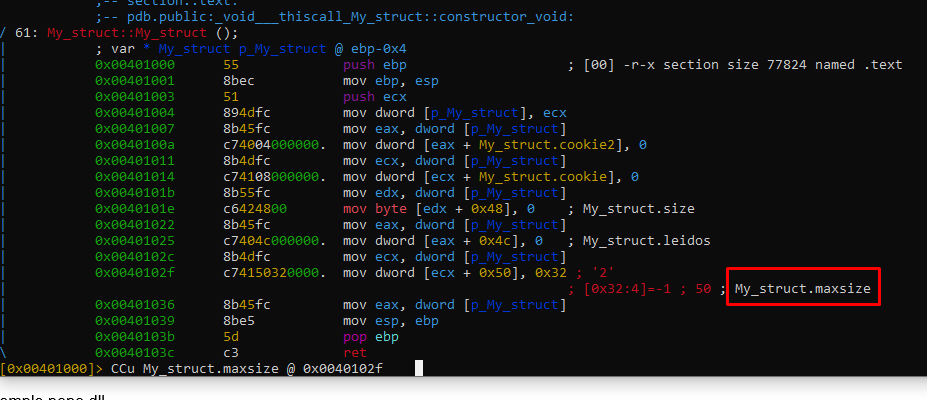
Y bueno al menos vemos que campo es y hacemos lo mismo con los otros dos que faltan, la forma correcta seria.





Agregamos el comentario

**CCu My\_struct.leidos @ 0x401025**

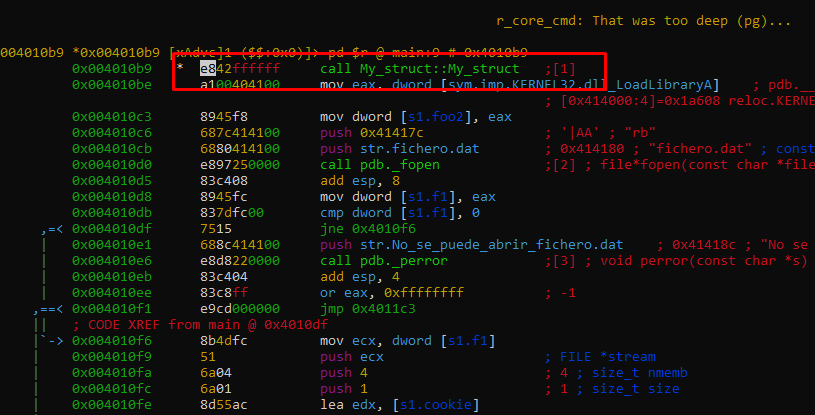


**CCu My\_struct.maxsize @ 0x40102f**

Bueno como ya lo habíamos reverseado, ya sabemos que allí inicializa dichos campos y lo vemos aquí también que esta función es el constructor de **My\_struct**.

El único valor importante es el de **My\_struct.maxsize** que lo inicializa a 0x32.

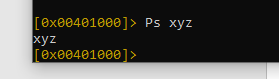
Volviendo al main con **u** volvemos al lugar desde donde habíamos entrado al CALL.

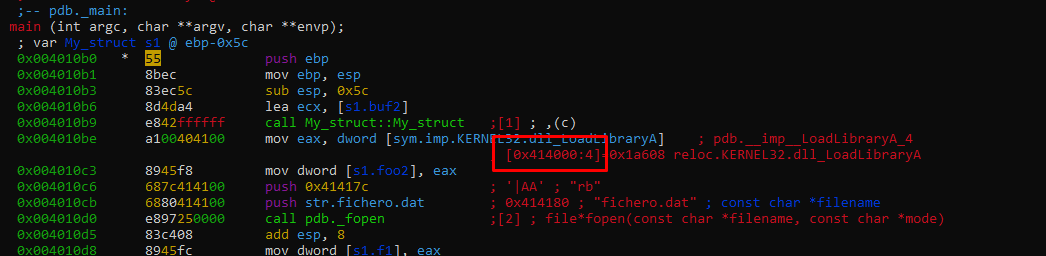


.

Cada tanto voy guardando en el proyecto, a veces le cambio el nombre para tener varios estados distintos para restaurar

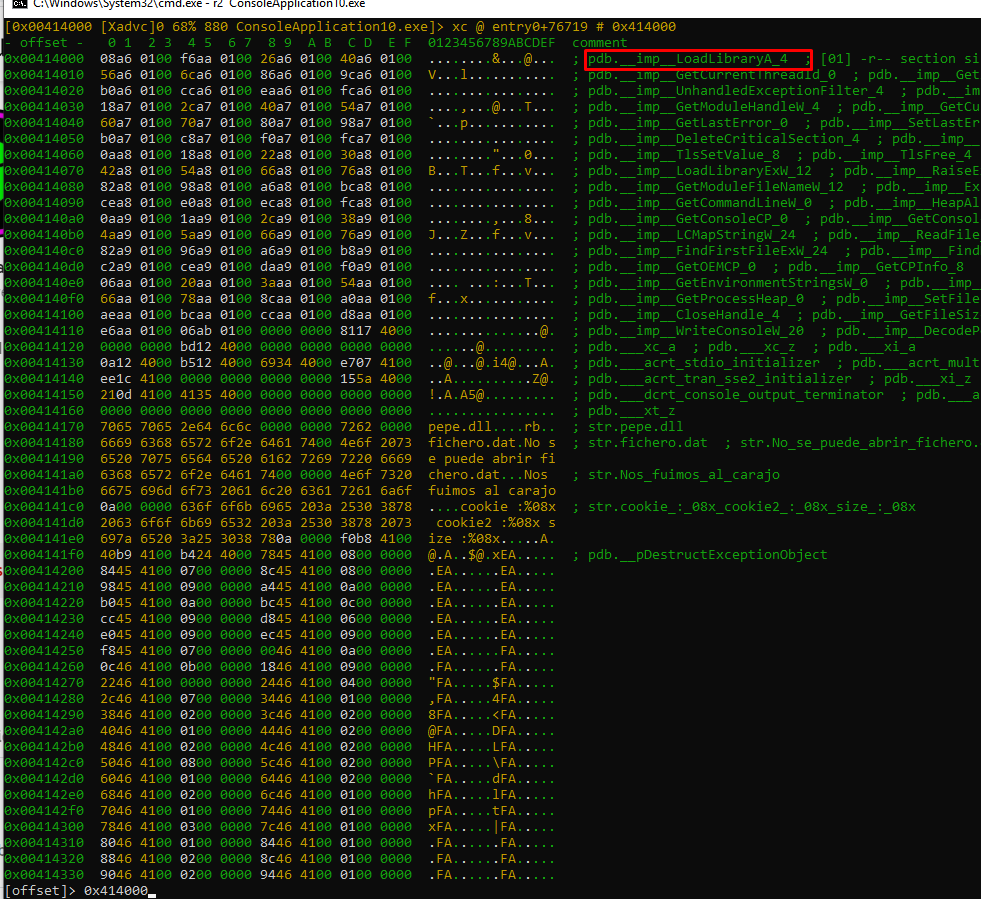
**Ps xyz**





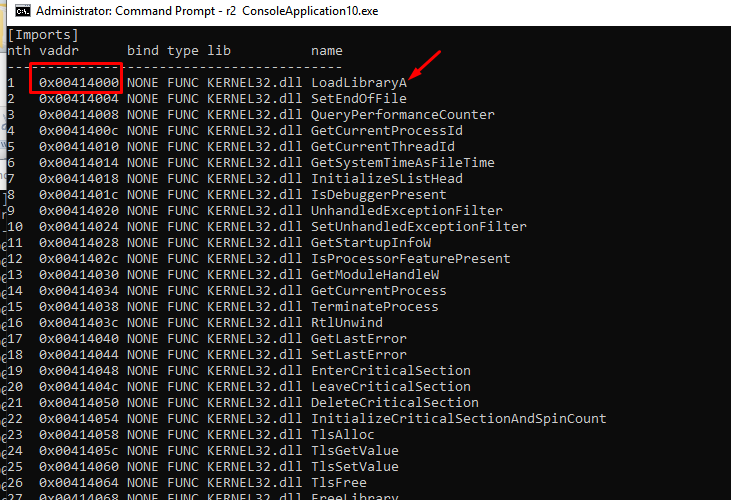
Vemos que carga la dirección de LoadLibrary en EAX.

Si salimos del modo cursor con C y rotamos las visualización con p hasta ver el hex dump, allí apretamos G y vamos a 0x414000 y vemos que es la entrada de la IAT para **LoadLibrary**.

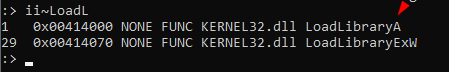


Confirmo que es la entrada de la IAT con

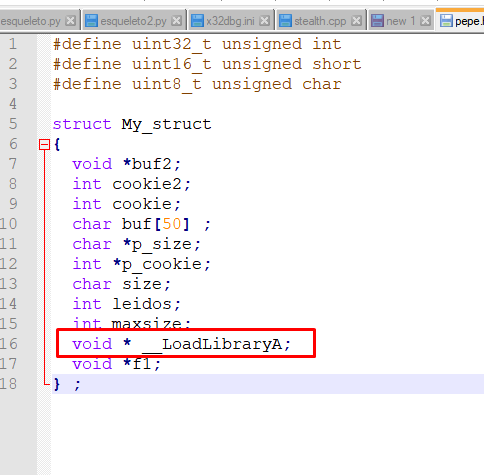
**ii~..**



Si filtro para buscar esa sola entrada.



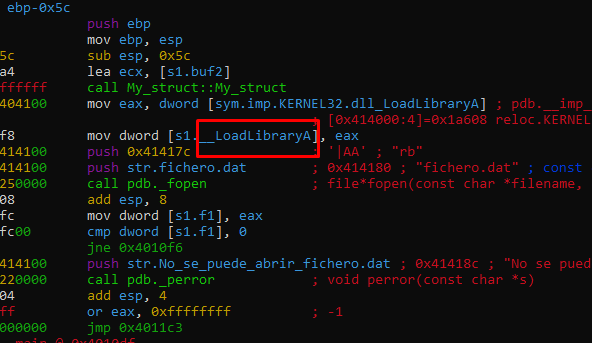
Renombro en el archivo que cree con la estructura, es una de las formas de renombrar campos de la misma.



Luego hago

**to pepe.h**

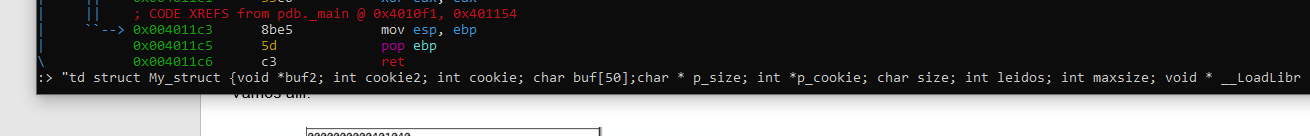
y el nombre ya cambio.



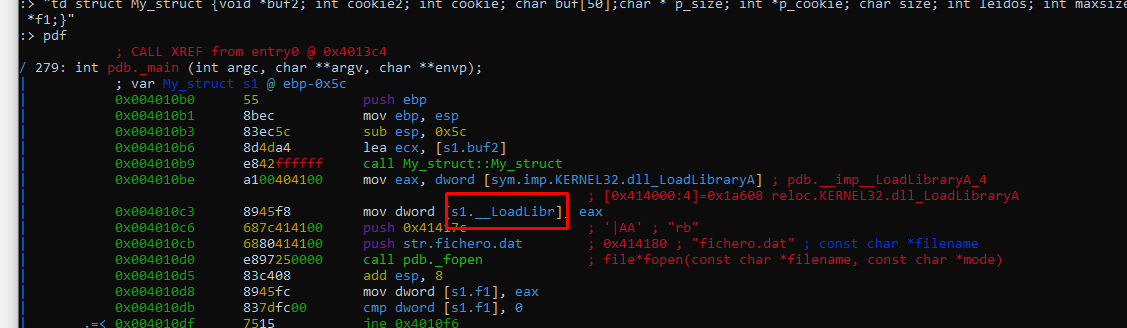
También podría ejecutar el comando **td** con el nombre cambiado

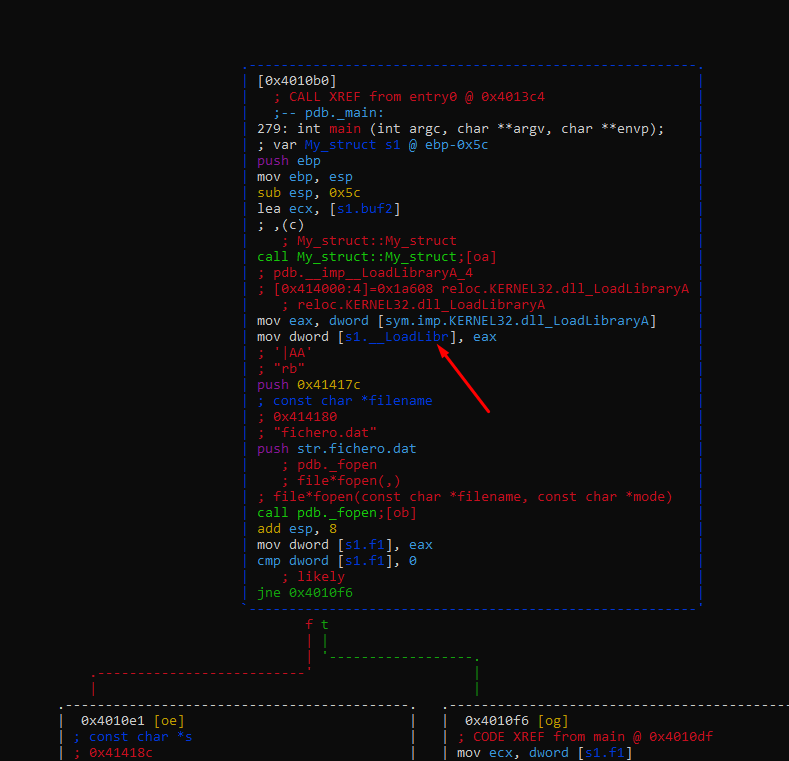
**"td struct My\_struct {void \*buf2; int cookie2; int cookie; char buf[50];char \* p\_size; int \*p\_cookie; char size; int leidos; int maxsize; void \* \_\_LoadLibr ; void \*f1;}"**

Vamos a probar eso a ver si cambia de nuevo

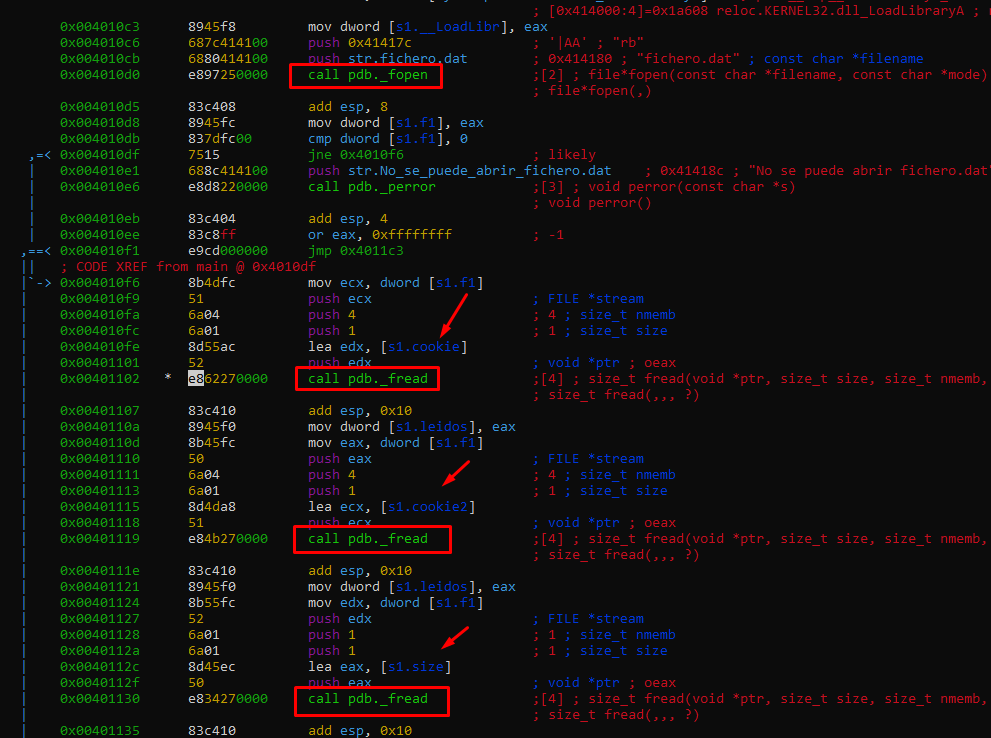


Si, cambio nuevamente.

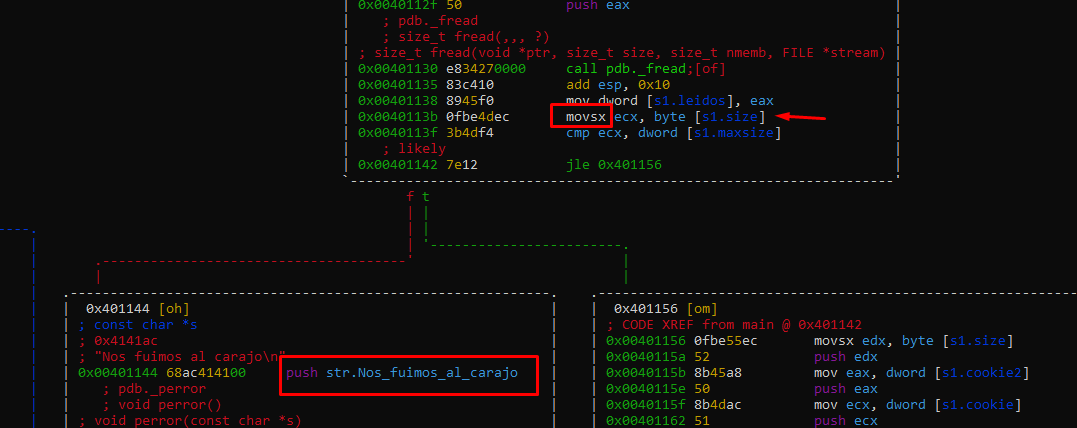




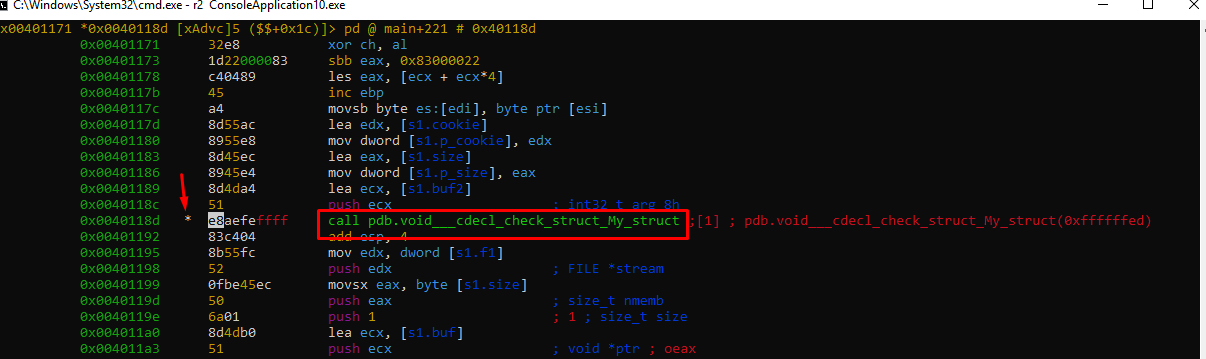
Vemos que es exactamente como habíamos visto la parte anterior, está el llamado a **fopen()** para abrir el **fichero.dat** y si lo puede abrir, lee tres veces del archivo los valores de **cookie, cookie2 y size**



Vamos allí.



Vemos que lee el 9no byte que era size y lo compara contra **maxsize** que era 0x32, si es menor o igual sigue, pero como considera el signo un valor de ese byte negativo termina siendo negativo también allí en la comparación y evita el Nos fuimos al carajo jeje y provocara el overflow más adelante como vimos.



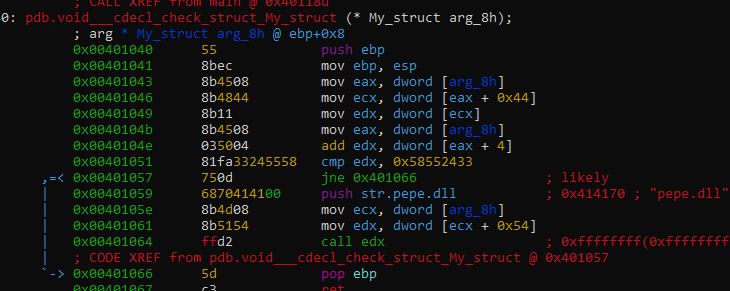
Llegamos al CALL donde chequea y habíamos visto como pasa como argumento un puntero a la estructura **My\_struct**, entramos con ENTER.



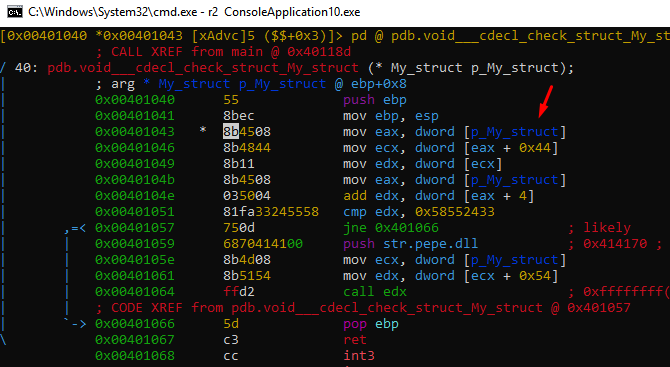
Cambiamos el tipo del argumento a **My\_struct \*** con

**afvt arg\_8h \* My\_struct @0x401040**

Ahi quedó bien



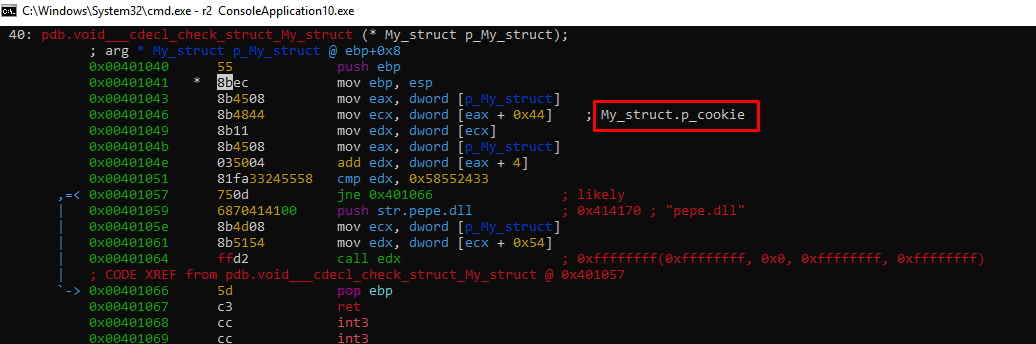
Poniendo el cursor en la línea y apretando d y luego n renombro la variable.



Comentamos los campos que no nos toma

struct.My\_struct.p\_cookie=int32\_t \*,**68**,0

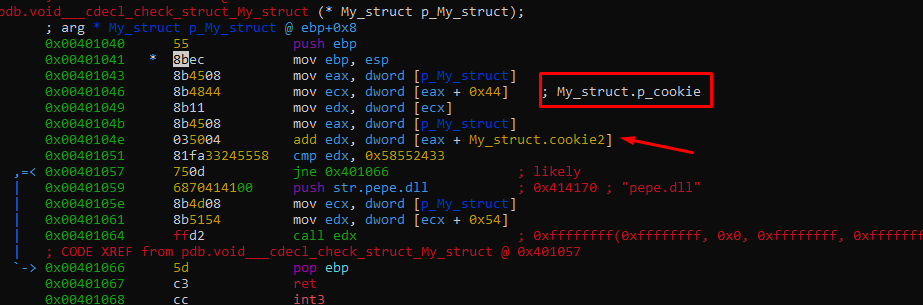
**CCu My\_struct.p\_cookie @ 0x401046**



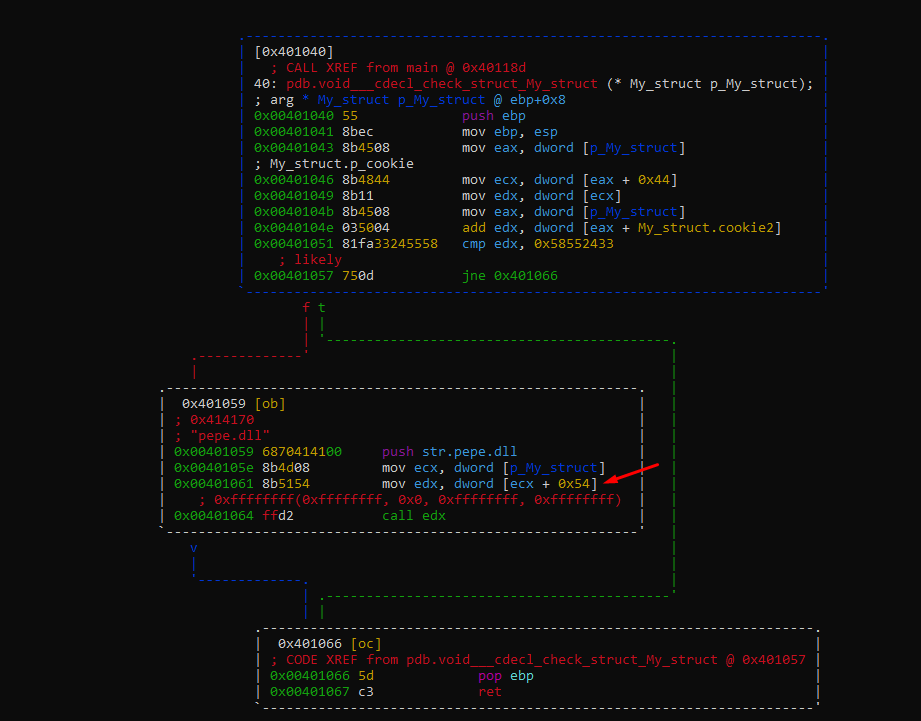
El campo del offset 4 lo tomará.



**aht My\_struct.cookie2 @ 0x40104e**



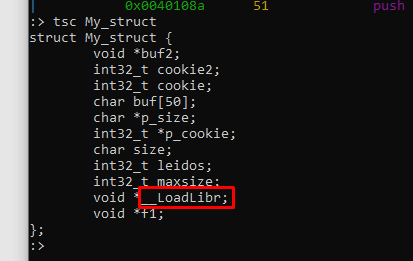
Como habíamos visto, leía el contenido de **p\_cookie** que es **cookie** y lo sumaba a **cookie2** si no es igual a **0x58552433** salta por encima y no carga la **pepe.dll**.



0x54 es 84 decimal.

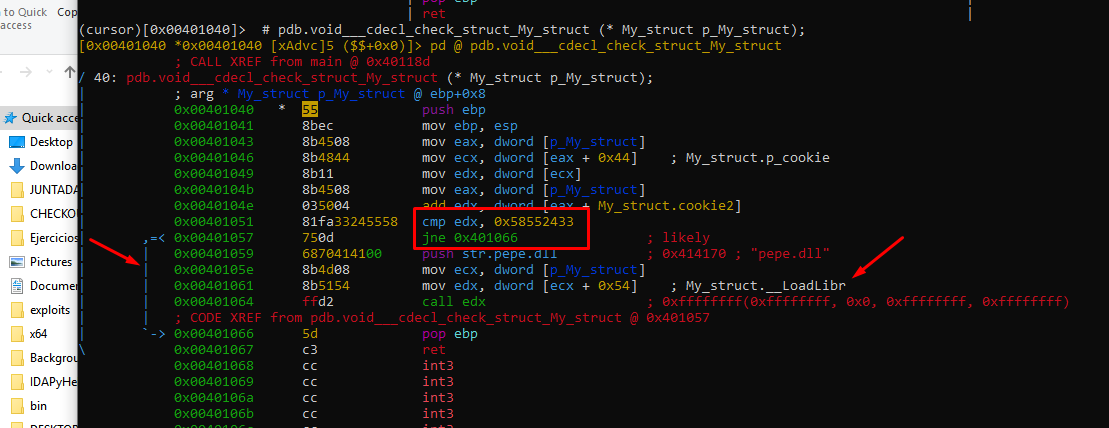
struct.My\_struct**.foo2**=void \*,84,0

que lo habíamos renombrado a **\_\_Loadlibr**



Le puedo poner también el comentario hasta que arreglen el bug.

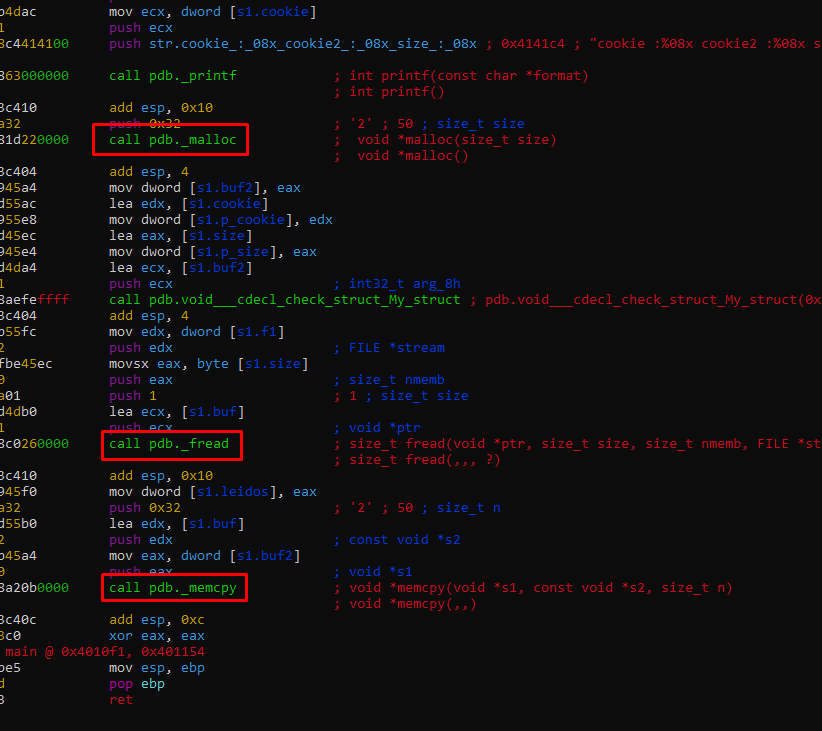
**CCu My\_struct.\_\_LoadLibr @ 0x401061**



Ahí esta el salto y la carga de pepe.dll si la suma es correcta.

Acá la función esta completa, volvamos con u al main.

El resto ya esta explicado en el tutorial anterior no vamos a abundar en ello. Hay un malloc fijo y un fread con un size negativo que desbordara el stack y un memcpy allí abajo que ocurre después del overflow.



Lo importante en este tute era probar la interfase de las estructuras en radare y aprender cómo se usa y a pesar del bug creo que hemos podido hacerlo, ya sabemos cuando se arregle el mismo como podremos manejarnos con las mismas.

Hasta la parte 22

Ricardo Narvaja

11/04/2020 (Aún en cuarentena light jeje)