EXPLOITING Y REVERSING USANDO HERRAMIENTAS GRATUITAS - PARTE 14

S

Como hemos visto en los tutoriales anteriores, crear un rop puede ser facil o muy dificil depende de como es la vulnerabilidad y segun las 4 preguntas que hemos visto nos dicen el nivel de dificultad.

Segun las 4 preguntas el ejemplo de la parte anterior:

1)Tiene módulos que no tengan ASLR? - **SI**

2)Tiene en algun modulo que no tenga ASLR importada la funcion VIrtualAlloc o VirtualProtect? **-SI**

3)Están los datos ya ubicados en el stack para comenzar a ropear? -**SI**

4)Puedo pasar cualquier caracter o sea no hay caracteres inválidos? **NO-PERO AFECTA POCO**

Podemos poner un score de 0 a 4: sumando uno por cada **NO**

Por supuesto un ROP de nivel 0 seria a primera vista más facil y uno nivel 4 puede llegar a ser muy dificil o imposible depende del caso, aunque después en la practica puede cambiar un poco.

El anterior vimos que seria nivel 1, aunque en la practica afectan tan poco los caracteres invalidos, así que en el caso ya realizado podriamos sumar solo 0.5 en vez de 1 ya que no fue casi afectado, lo cuál nos daria finalmente un score de 0.5, segun la escala ricnar jeje.

Bueno tenemos los dos ejemplos siguientes en 32 y 64 bits y primero que nada analizaremos la dificultad del ROP antes de ponernos a ROPEAR.

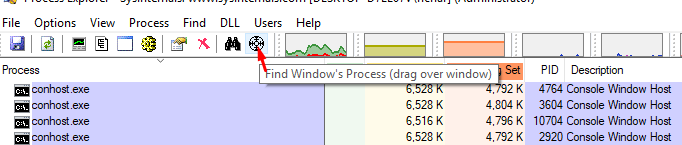
ANALIZANDO LA DIFICULTAD DEL SEGUNDO EJERCICIO DE 32 BIT

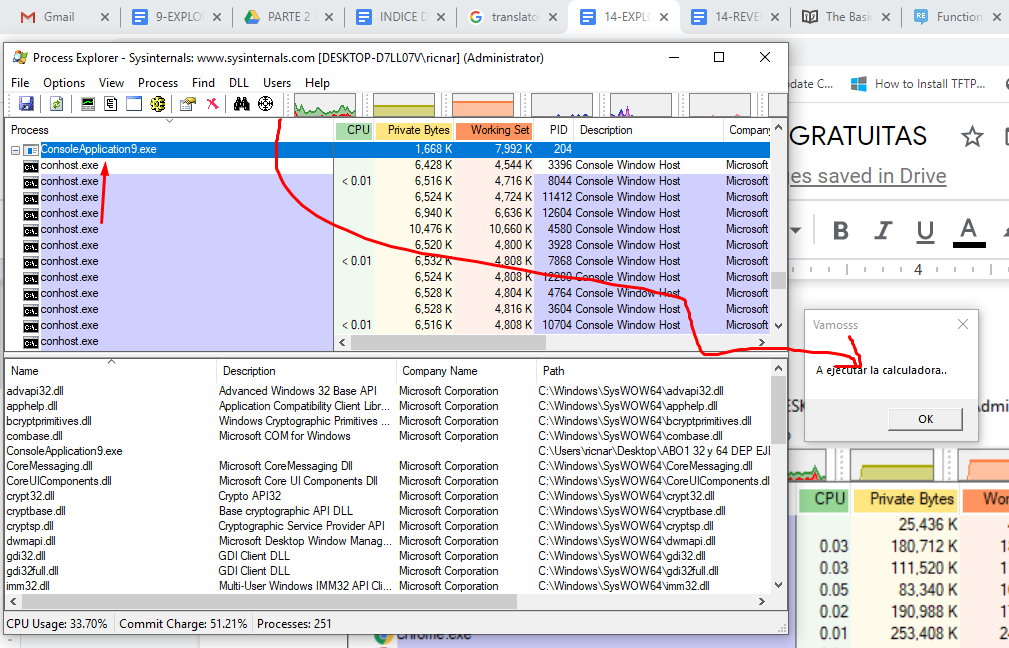
<https://drive.google.com/open?id=1IMiq0AqErEzWOiq78l0hA4Mb9S80RGXb>

Vayamos contestando las preguntas:

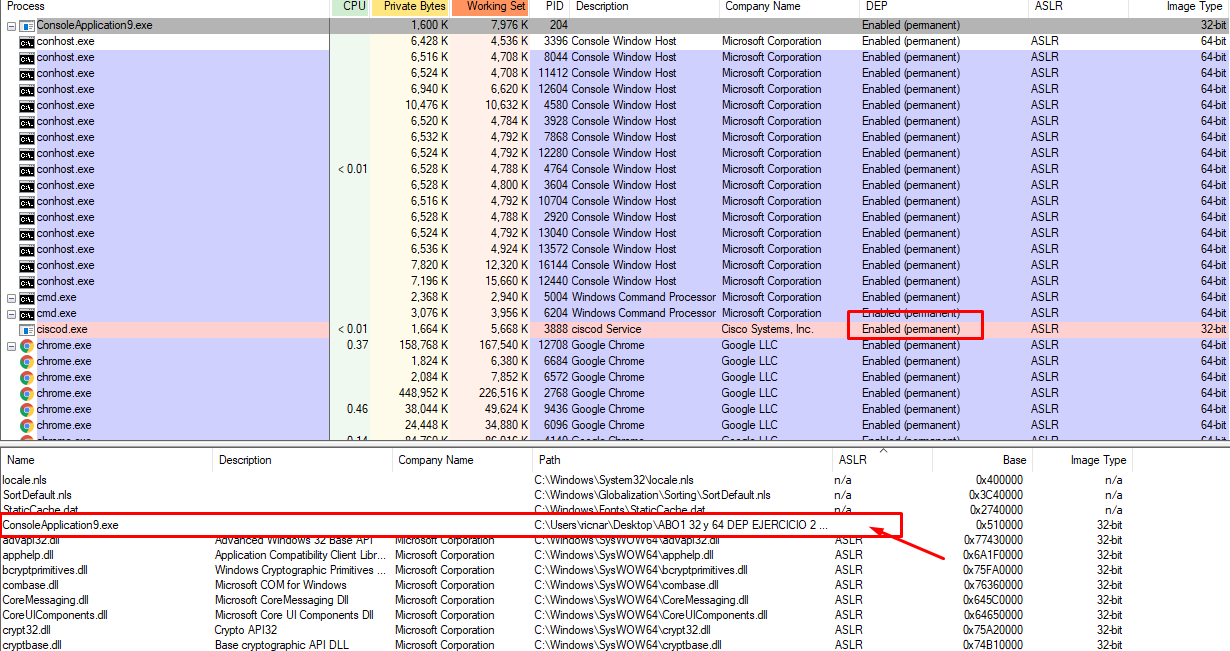
1)Tiene módulos que no tengan ASLR?

Corramoslo y cuando salga el MessageBoxA miremos en Process Explorer si hay modulos sin ASLR.





Hacemos click sin soltar y vamos a la ventana del MessageBoxA, alli soltamos y nos mostrará a que proceso corresponde.

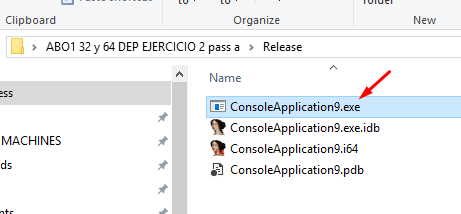


Tiene DEP habilitado por lo que para resolverlo habra que construir un ROP, y tiene un modulo sin ASLR (el del mismo ejecutable), así que la respuesta a la primera pregunta es **SI** por lo cuál no sumamos (recordemos que solo sumamos 1 por cada **NO**)

**SCORE PARCIAL =0**

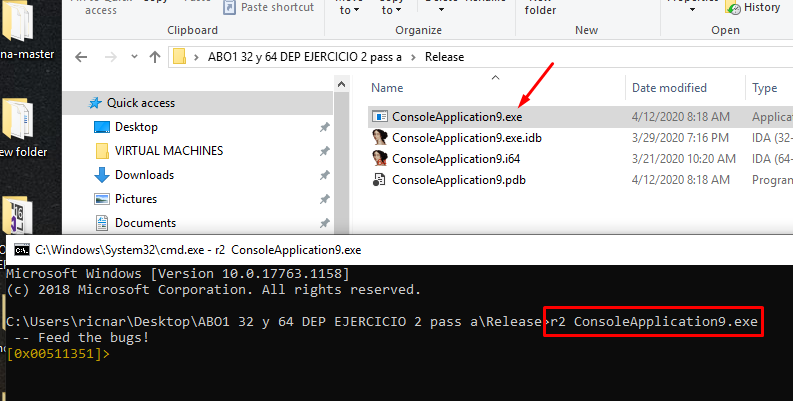
2)Tiene en algun modulo que no tenga ASLR importada la funcion VIrtualAlloc o VirtualProtect?

Usaremos RADARE para abrir primero el ejemplo de 32 bits.



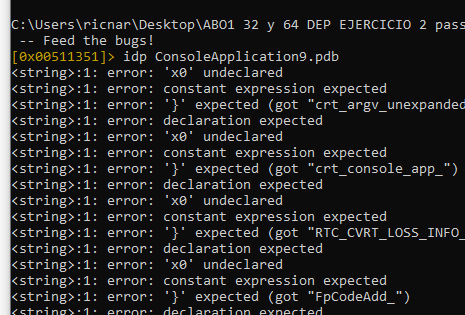
Abro un **cmd** en la carpeta donde está ubicado y tipeo

**r2 ConsoleApplication9.exe**

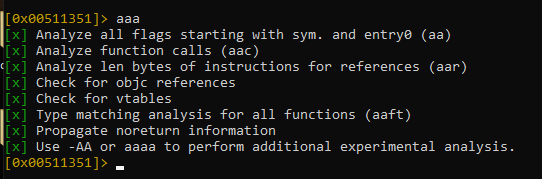


Cargo los símbolos con el comando **idp**

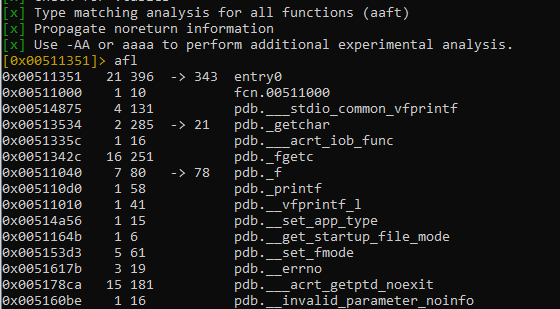
**idp ConsoleApplication9.pdb**

****

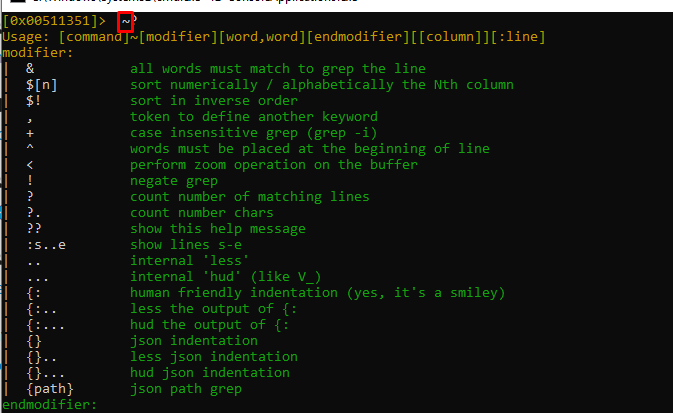
y luego con **aaa** analizo el programa.



Con **afl** listo las funciones.

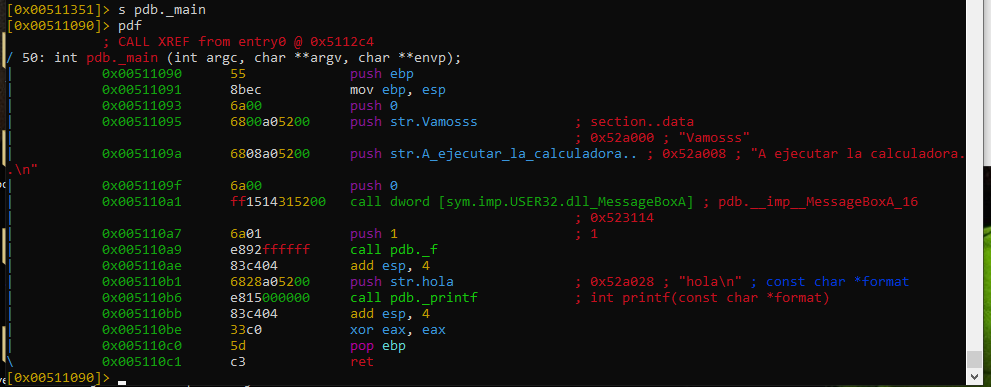


Si queremos filtrar radare tiene un grep incluido usando el simbolo **~**





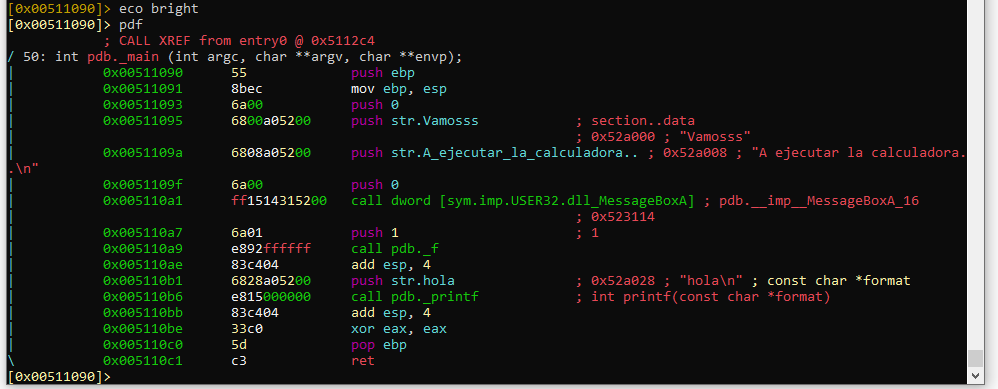
Vayamos al main con **s pdb.\_main**



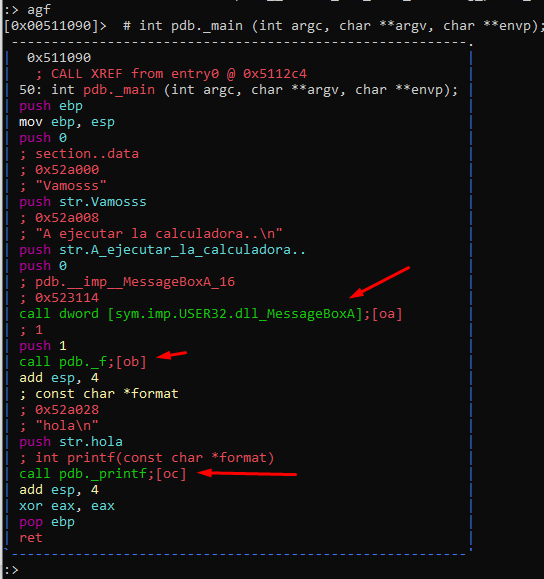
Creo un proyecto con **Ps ROP32**

****

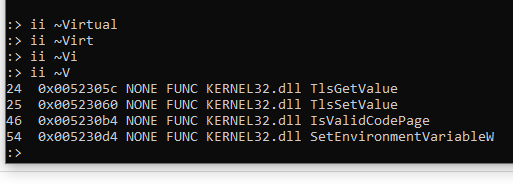
Más visible con **eco bright**.

****

Usando **agf** veré los bloques de la función con eso me alcanza.



Veo que no está la llamada a VirtualAlloc que teniamos en el ejercicio anterior, lo cuál nos facilitaba el trabajo ya que estaba importada dicha funcion, listemos las funciones importadas a ver si la tenemos, usando el comando **li** y filtrando con **~**.



Vemos que no está importada, por lo cuál la respuesta es NO.

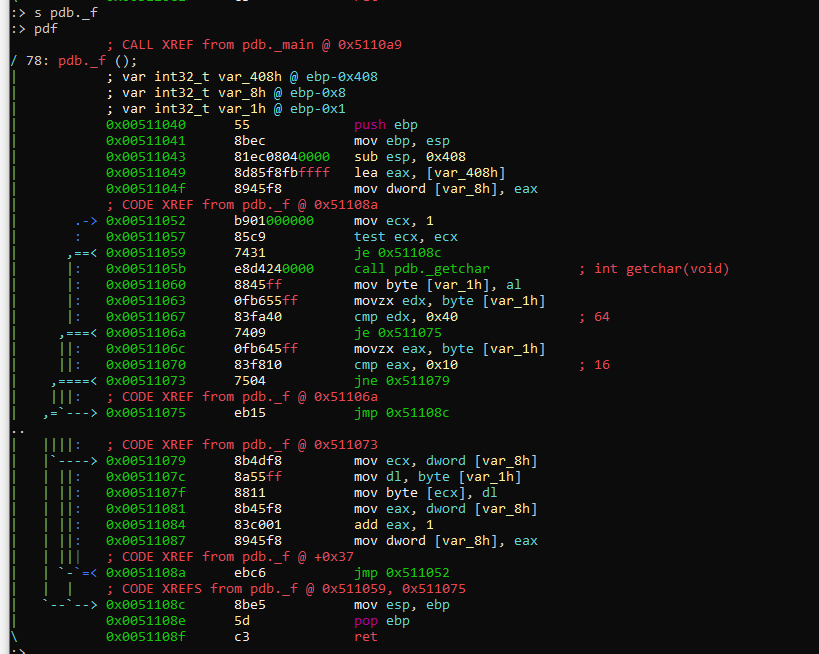
**SCORE PARCIAL= 1**

3)Están los datos ya ubicados en el stack para comenzar a ropear?

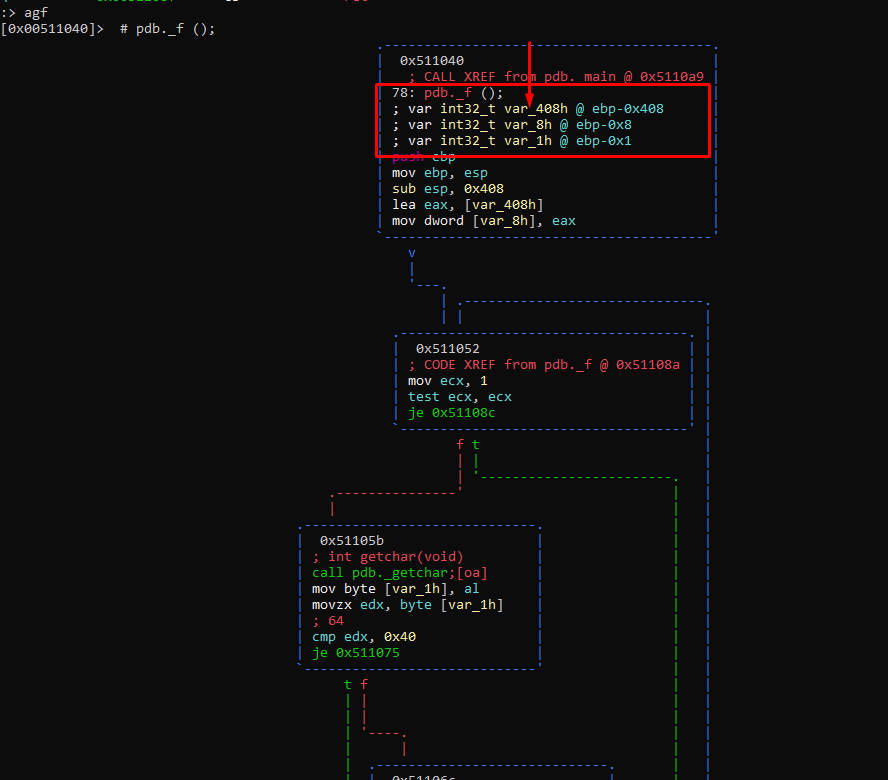
Para ver eso debemos analizar la vulnerabilidad, en la funcion **main** no se ve nada, quizas este dentro de la funcion llamada **f**.

Podiamos ver la funcion f con **s pdb.\_f**

Pero mejor usemos el modo visual, entrando en el mismo con **v,** apretando **c** controlamos el cursor y con las flechas de direccion bajamos hasta el call a la funcion **f** y apretamos **ENTER**.



En modo visual salimos del modo cursor con **c** y apretamos la barra espaciadora para ver los bloques. (barra espaciadora de nuevo para volver)



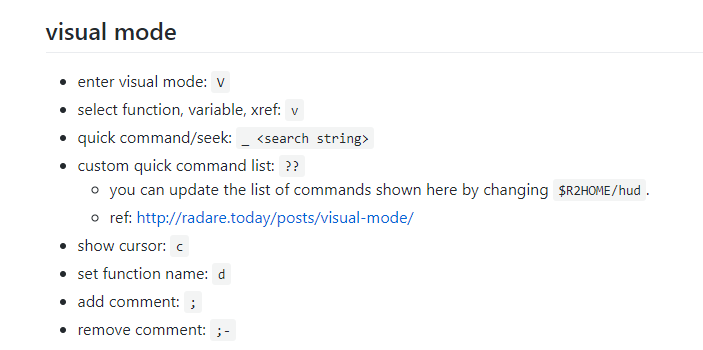
Vemos tres variables, la superior tiene un espacio considerable con la siguiente así que debe ser un buffer en el stack.

La misma esta ubicada en **-0x408** con respecto a ebp y la siguiente variable está ubicada en **-0x8** respecto a ebp, hay **0x400** bytes de diferencia que es el largo del buffer.

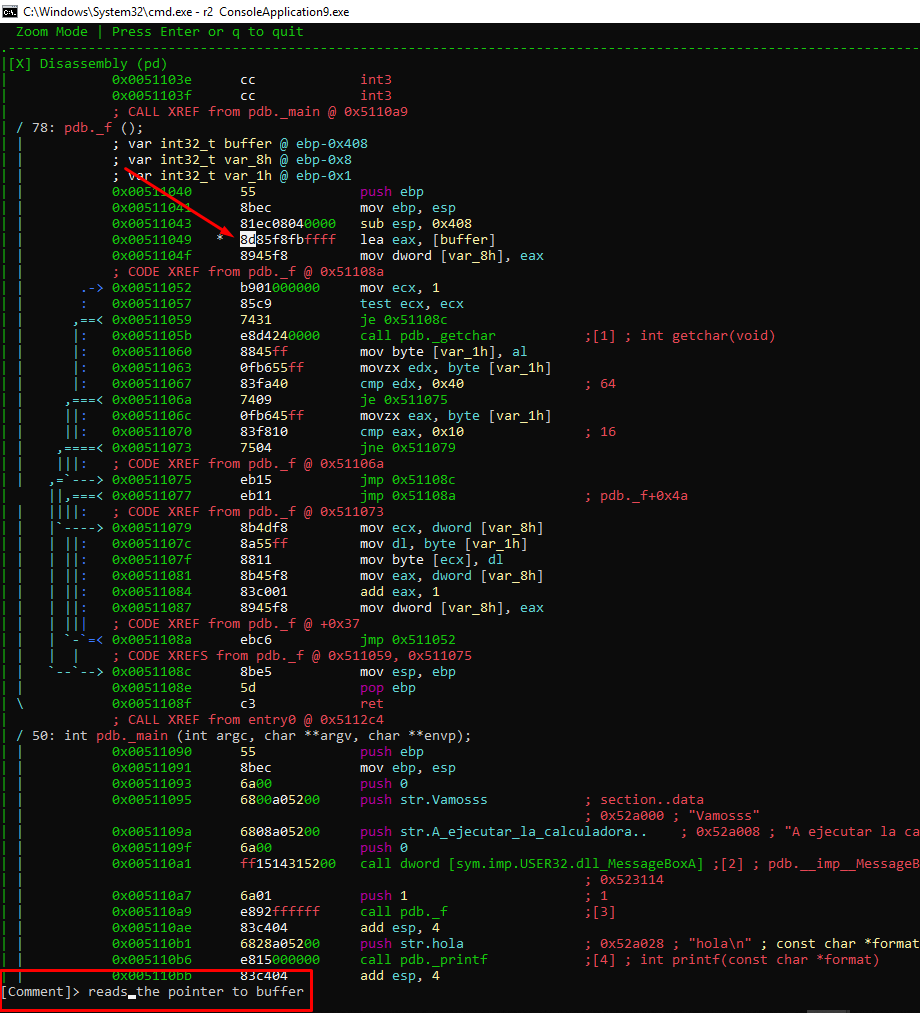
En modo visual puedo también apretar **:** y luego renombrar con el comando **afvn buffer var\_408h.**



Y luego **v** de nuevo para que refresque.

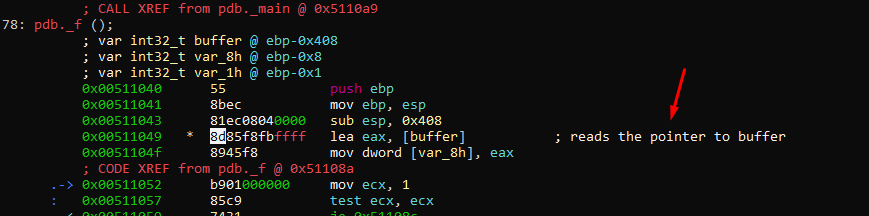


Como vimos, apretando la tecla **c** controlamos el cursor con las flechas de dirección (c para salir del modo cursor)

****

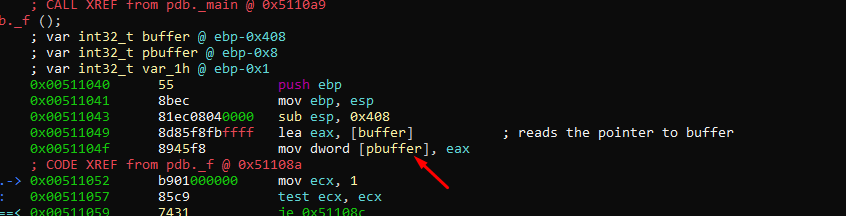
Voy con las flechas de direccion a la línea que quiero comentar y apreto **;** y agrego el comentario y ENTER.

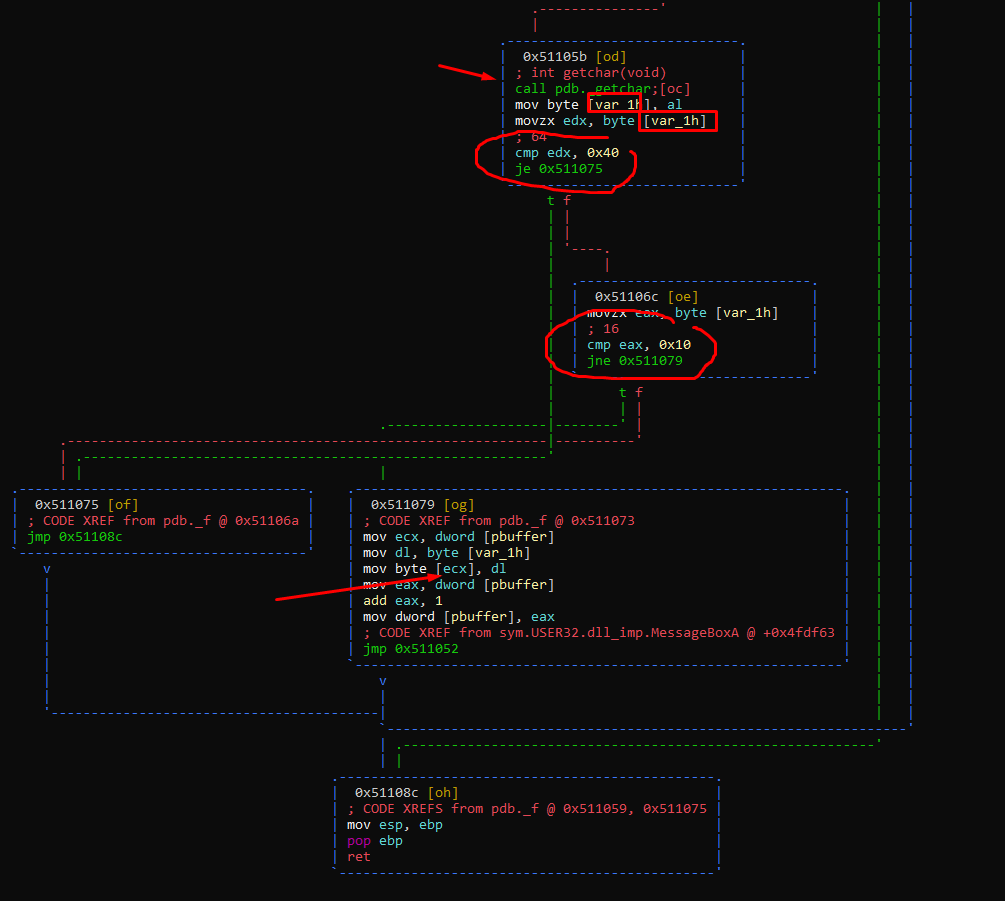
En estos comandos de modo visual no es necesario refrescar.



Vemos que guarda en **var\_8h** el puntero a **buffer**, la renombrare esa variable como **pbuffer**.

Apreto **:** y luego **afvn pbuffer var\_8h** y luego **v.**

****



Vemos que hay un loop dentro del cuál va leyendo caracteres en cada repeticion con la funcion getchar, lo guarda en la variable **var\_1h** que es una variable de un solo byte, y luego los compara con **0x40** y **0x10** si es alguno de esos dos valores sale, sino lo guarda en buffer ya que lee pbuffer a ECX y luego guarda en su contenido o sea en buffer.

Vemos que no hay cookie de proteccion, así que deberia llenar bufffer y pisar el return address (jejeje) hasta que encuentre un byte **0x40** o **0x10** y salga por lo tanto la respuesta a la pregunta es SI ya que mis datos estaran en el stack y podre ropear directamente desde alli.

**SCORE PARCIAL =1**

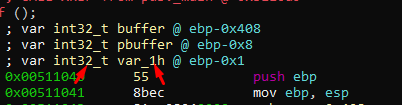
4)Puedo pasar cualquier caracter o sea no hay caracteres inválidos?

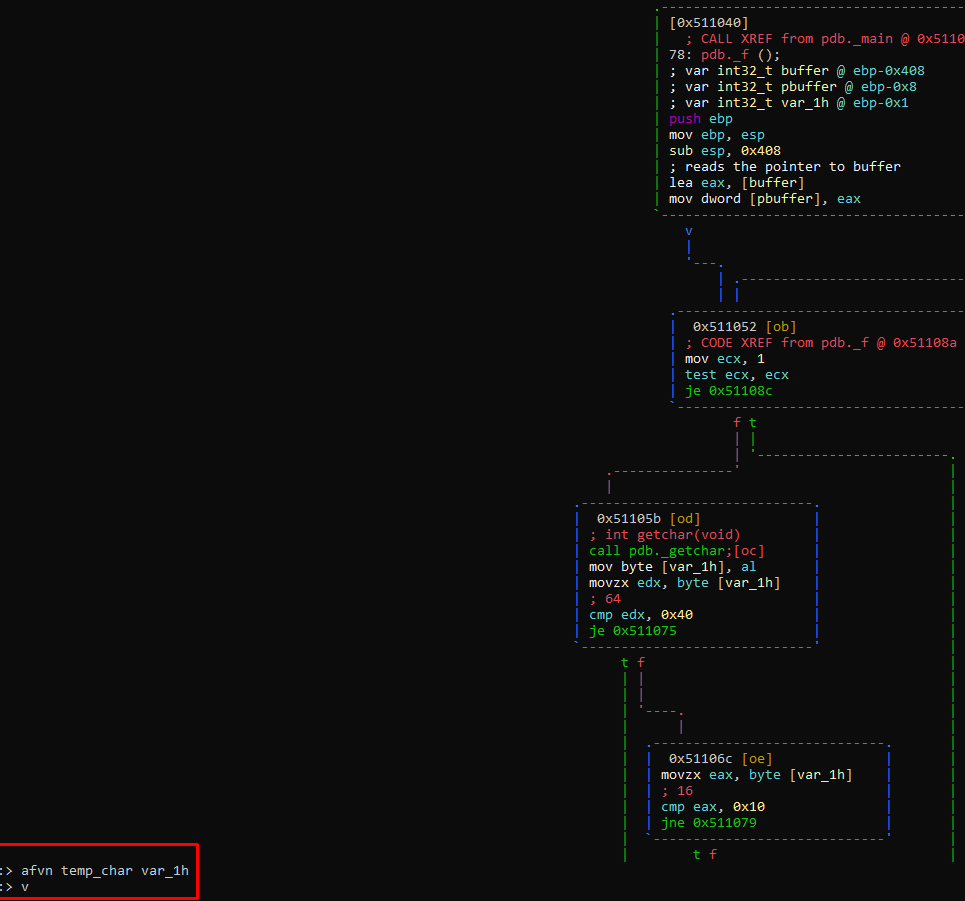
Segun pudimos analizar solo podemos pasar caracteres 0x40 o 0x10 si queremos terminar de copiar, por lo tanto la data copiada en el stack que formara el ROP no podra tener ni 0x40 ni 0x10, lo mismo que el shellcode así que la respuesta es **NO**, ya veremos si afecta mucho o no por ahora le sumamos uno al score.

**SCORE FINAL =2**

Como vemos este analisis parcial me dice que este ROP con SCORE=2, deberia ser más complicado que el que vimos previamente que era de SCORE 0,5.

Renombraremos la variable y le corregiremos el tipo con **afvn temp\_char var\_1h** y luego **v**.

****

****

Para cambiar el tipo a char usaremos el comando **afvt temp\_char char** y luego **v.**

Guardare el proyecto con **ps ROP32** para continuar en la próxima parte y porque me gustaria que lo intenten y no tengan miedo que fallar enseña más que un ejercicio facil.

TIP PARA AYUDAR A QUE LO INTENTEN RESOLVER

Un TIP ----> cuidado al copiar en el stack con **pbuffer** que está debajo de **buffer** y se puede romper y terminar crasheando y no permitirles llegar al return address.

18/04/2020

Seguimos en cuarentena pero con todas las ganas.

Hasta la parte 15.

Ricardo Narvaja