GRAMMATECH

Signing Drivers for Windows 10

Disclaimer

This document is a record of our experience signing a device driver for Windows. Use information in this document at your own risk; your mileage may vary.

The Problem

During 2016, we started getting sporadic reports of Windows 10 installations that would not load a device driver component of CodeSonar. According to the reports, the driver did not have a valid digital signature. Users could disable SecureBoot to work around the problem temporarily. We needed to figure out how to reproduce the problem and then solve it once and for all.

Why does CodeSonar need a device driver?

In order to make the software analysis process as easy as possible, CodeSonar observes the user's build system (e.g., make, msbuild, or anything else) at a very low level from inside the kernel on Windows. By observing the build system, we can discover all the source code in a project and identify exactly how it should be compiled. This way, the users don't need to create a second build system for CodeSonar or make any modifications to their existing build system. We tried userland hooking for a while, but some processes were very sensitive to having unfamiliar code injected into them, and some security products didn't like it very much, either. The kernel approach has been reliable and opaque by comparison.

Unfortunately, there is some overhead associated with kernel development. In addition to the obvious costs of learning the APIs and creating a robust implementation, kernel code has to be digitally signed for Windows to trust it. Previous to Windows 10, developers could sign drivers, using a certificate purchased from any of the usual vendors, such as GlobalSign, VeriSign, or Symantec.

Windows 10 Policy

When Windows 10 was coming out, Microsoft announced what the driver signing policy would be: https://blogs.msdn.microsoft.com/windows_hardware_certification/2015/04/01/driver-signing-changes-in-windows-10/

> What about existing drivers? Do I need to re-sign these drivers to get them to work with Windows 10?
> No, existing drivers do not need to be re-signed. To ensure backwards compatibility, drivers which are properly signed by a valid cross-signing certificate that was issued before the release of Windows 10 will continue to pass signing checks on Windows 10.

What a relief! Our CA's cross certificate (the GlobalSign "r1-r3" certificate) was issued before Windows 10 came out and wouldn't be expiring until 2021, so no action was required until it expired.

So why did various Windows 10 users start encountering signature problems during late 2016, a while after the initial release of Windows 10? And why could we not reproduce any of these problems on any Windows 10 machine in our possession, even with SecureBoot enabled?

Unbeknownst to us, Microsoft posted this to their Hardware Certification Blog in July 2016: https://blogs.msdn.microsoft.com/windows_hardware_certification/2016/07/26/driver-signing-changes-in-windows-10-version-1607/

> What are the exact exceptions? Are cross-signed drivers still valid?
>
> Enforcement only happens on fresh installations, with Secure Boot on, and only applies to new kernel mode drivers:
>
> - PCs upgrading from a release of Windows prior to Windows 10 Version 1607 will still permit installation of cross-signed drivers.
> - PCs with Secure Boot OFF will still permit installation of cross-signed drivers.
> - 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.
> - To prevent systems from failing to boot properly, boot drivers will not be blocked, but they will be removed by the Program Compatibility Assistant. Future versions of Windows will block boot drivers.
>
> To summarize, on non-upgraded fresh installations of Windows 10, version 1607 with Secure Boot ON, drivers must be signed by Microsoft or with an end-entity certificate issued prior to July 29th, 2015 that chains to a supported cross-signed CA.

Hmm, this one says "an end-entity certificate" while the previous blurb said "valid cross-signing certificate." In retrospect, I wonder if Microsoft was referring to the end-entity certificate (i.e., the one with subject GrammaTech) when they said “valid cross-signing certificate” and not the cross certificate itself, which is how I interpreted the phrase.

Why wasn’t the problem found during testing? Because we were not using “fresh installs” of Windows 10 1607 but were instead installing older builds and then upgrading with Windows Update. This grandfathered in the lack of enforcement. After installing the OS using Windows 10 1607 on a machine with SecureBoot enabled, we could reproduce the issue. Now about fixing it...

Hardware Dev Center Site

The driver had to be signed differently. All we really knew was that the new signing process involved https://sysdev.microsoft.com/hardware. At least that’s what the MS blog post said. So we started signing up for accounts and used our existing certificate to verify our corporate identity.

It soon became clear that our certificate was not an EV certificate and we would need to purchase an EV certificate to register for file signing, so we did. A colleague signed the nonce binary with his account to demonstrate to MS that we were in possession of an EV certificate. Now it was my turn to start the driver signing process.

Upon navigating to the hardware dashboard, I was told GrammaTech did not have an EV certificate and still needed to sign a nonce binary before I could proceed to sign drivers. This was surprising, because the colleague had already done it. So I tried signing the nonce binary and was told the certificate was already claimed. What’s going on? I emailed Microsoft technical support, and they made some suggestions, which did not lead to success. However, we eventually noticed something odd: Everyone who had signed up for an account with this service actually had created two accounts without realizing it: One account identified by their GrammaTech email address (username@grammatech.com) and another account username@grammatech.onmicrosoft.com. It turns out our EV certificate had been associated with the username@grammatech.onmicrosoft.com accounts, so we needed to use those accounts exclusively from now on for all this activity. So that’s one problem solved.

On to the next step. Once you log in, click “File signing services” and then “Create driver signing submission.” It directs you to a developer.microsoft.com page that looks like a different web application (“Dashboard”). It asks for an hlkx file or a cab file to be uploaded. We don’t use cab files (or inf files) for our driver, so I start looking into what an hlkx file is. It turns out this is a file that can be produced by Hardware Lab Kit. Do we really need to use this, given we don’t produce any hardware? Down the rabbit hole.

Hardware Lab Kit

Installation

I search the internet for Hardware Lab Kit and find the download site. Multiple versions are available, corresponding to different versions of Windows 10. I am highly uncertain about which version to choose. If I choose the HLK 1607 version, will my driver signature be accepted by machines running 1703? The MS blog says the hardware dashboard will stop accepting submissions from HLK 1607 on June 30, 2017, but the download page has dire warnings that I have to use HLK 1607 if I want certification for Windows Server 2016. Is certification the same thing as signing? I am still uncertain. Some of our customers use Windows server 2016, so I hesitantly choose 1607.

I attempt to install HLK and discover that it has server and client components. The server can only be installed on Server editions of Windows, and I will need a Windows 10 machine to serve as the client. We set up a VM with Windows Server 2012 to act as the server and install both the HLK Controller and Studio components. The laptop that was used to reproduce the signature rejection problem earlier can act as the client machine.

Our IT department installed the OSs and HLK on these two systems and added me to the Admin group. Upon trying to start the HLK Studio UI on the server, I get a “Failed to connect to database” error. After some internet searching, I determine that this happens if a user other than the one who installed HLK attempts to run it. I contact the person who installed HLK to follow the instructions under “Configure additional user accounts” here (I can’t do it myself because you need to be able to start HLK to follow the directions): https://msdn.microsoft.com/en-us/library/windows/hardware/dn939925(v=vs.85).aspx

I try again. This time it starts.

Prepare the Driver

I start following the “Getting Started” section in the “Help” dialog (also available at https://msdn.microsoft.com/en-us/library/windows/hardware/dn915002%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396) to install the client software on the laptop. I then proceed to install CodeSonar on the laptop. I prepare a special build of the device driver and sign it, using our normal cross-signing certificates.

Then I plug in the USB token with the EV certificate private key and add a signature, just in case the HLK submission requires it:

```
signtool sign /v /as /sha1
        6927415CF6A3A6AC5C99F450108CDF061A430CF6 /fd sha256
        /td sha256 /tr http://timestamp.globalsign.com/?signature=sha2
        drvhookcsmf_amd64.sys
```

I don't know whether this signing step is necessary or not.

Disable SecureBoot

The laptop won't be willing to load this driver as long as SecureBoot is enabled. I go into the UEFI and disable SecureBoot. For good measure, I also run:

```
bcdedit -set TESTSIGNING ON
```

Now I can copy the driver onto the laptop and install it. Our driver starts on demand the first time you use our software, so I use our software to activate it.

Pool Configuration

Back to HLK Studio on the server. I can see the client machine in the default pool, and I'm trying to use it. It turns out (this isn't obvious) that you cannot use a machine that is in the default pool: It has to be moved to a different pool before it can be used. So I proceed to move the machine into a different pool.

Something is still wrong, as the machine is not ready. As it turns out, on the configuration tab, when viewing the pool, you have to right click the machine and then change its status to ready.

So far so good. The machine is now ready. I create a new project named after our driver. Proceed to the selection tab. I select the pool with my machine, and then I'm not sure which category to select in the left margin. After a bit of head scratching, I settle on "Software Device" even though we don't register even a software device with Device Manager.

Driver Missing?

Something's wrong. Our driver isn't showing up in the Software Device list, and I have no idea why. "sc query" shows it is running. I download procmon and start trying to reverse engineer how HLK enumerates drivers. Eventually I determine that it only lists drivers configured with start=system, but we use start=demand. I execute this command on the laptop to correct the situation: “sc config drvhookcsmf start=system”. After I force HLK studio to refresh its list of drivers, our driver finally makes an appearance.

Test Selection

There is some confusion about whether or not I need to download an HLK playlist and load it. After some amount of messing about, I determine that all the playlists do is specify a subset of tests that should get run. If I don’t use a playlist, then I might end up running extra tests. But that sounds OK to me since it looks like using the playlist would only save me one uninteresting test. Running extra tests won’t cause my submission to be rejected.

I check the box next to our driver in the list, then right click it, and choose DeviceGuard, FileSystem, Filter.Driver.Fundamentals, and Filter.Driver.Security. Our driver is a file system minifilter, so I’m pretty much just guessing which things to check, based on their names.

Test Prerequisites

Now I proceed to the “Tests” tab and attempt to run all the tests. I am immediately notified that some of these tests have prerequisites. After some internet searching, I determine that I need to run these commands on the laptop:

```
bcdedit.exe /set groupsize 2
bcdedit.exe /set groupaware on
shutdown.exe -r -t 0 -f
```

It also tells me that I need to configure 7 additional partitions on the laptop, and they all need to be on a separate disk. This is a problem: The laptop only has one disk. I try using virtual disks, but it turns out this is unworkable because they do not automatically get mounted when the machine boots, and the HLK likes to automatically reboot the machine a bunch of times as the tests run. Plan B: IT has a bunch of external USB disks, so I go steal a 1TB disk, even though I only need 10GB.

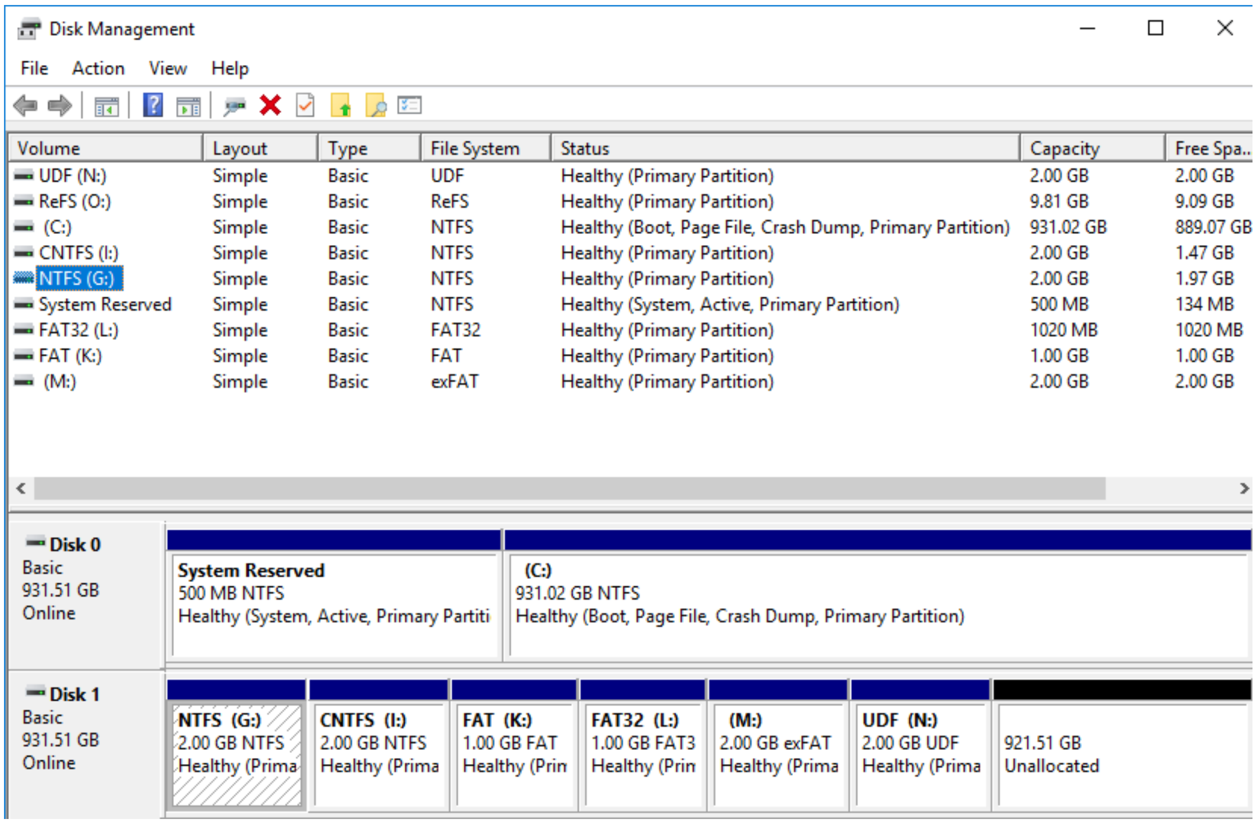
I configure the various partitions, using the “Disk Management” console. It doesn’t know how to format disks using UDF or ExFAT, so I leave those disks unformatted and later format them, using the format command from the console. Then I notice something: The documentation doesn’t say anything about creating a Resilient FS partition, but HLK’s prereq dialog box says I need one on drive O. I’m no stranger to out-of-date documentation, so I try to create this new volume.

I open up “storage spaces” to create a new storage space, but there’s a problem. It turns out pools won’t share disks with any other partitions. I fetch an additional 1TB USB disk, plug it in, and use “Disk Management” to remove the existing NTFS partition.

Then I open up the "storage spaces" dialog and try to create a new storage pool. Cryptic error message says the disk is in use. I close Disk Management and try again. No such luck. After much messing about, I reboot and try again. It works this time. Takeaway: You've got to reboot between deleting the partitions and creating a storage pool. I don't know why.

I select "No mirroring" and ReFS from the drop down menus and request a 10GB volume. Upon trying to create the volume, it tells me "The parameter is incorrect." Thanks. After trying many random things, I decide to try 2-way mirroring in the drop down menu. Ugh. There's a problem: It refuses to have the mirror on the same physical disk. Luckily, IT has a lot of these 1TB disks, so I go steal another one.

After much disk wiping and rebooting, I create storage spaces on these last two disks. I try to create a 10GB ReFS volume mirrored across these two 1TB disks. Finally, it works. Takeaway: ReFS can't be used without mirroring, but the GUI makes no attempt to prevent one from asking it to do so. Something to look forward to in Windows 11. Disk Management should look about like this:



Volume	Layout	Type	File System	Status	Capacity	Free Spa..
UDF (N:)	Simple	Basic	UDF	Healthy (Primary Partition)	2.00 GB	2.00 GB
ReFS (O:)	Simple	Basic	ReFS	Healthy (Primary Partition)	9.81 GB	9.09 GB
(C:)	Simple	Basic	NTFS	Healthy (Boot, Page File, Crash Dump, Primary Partition)	931.02 GB	889.07 GB
CNTFS (I:)	Simple	Basic	NTFS	Healthy (Primary Partition)	2.00 GB	1.47 GB
NTFS (G:)	Simple	Basic	NTFS	Healthy (Primary Partition)	2.00 GB	1.97 GB
System Reserved	Simple	Basic	NTFS	Healthy (System, Active, Primary Partition)	500 MB	134 MB
FAT32 (L:)	Simple	Basic	FAT32	Healthy (Primary Partition)	1020 MB	1020 MB
FAT (K:)	Simple	Basic	FAT	Healthy (Primary Partition)	1.00 GB	1.00 GB
(M:)	Simple	Basic	exFAT	Healthy (Primary Partition)	2.00 GB	2.00 GB

Disk 0
Basic
931.51 GB
Online

System Reserved
500 MB NTFS
Healthy (System, Active, Primary Partiti

(C:)
931.02 GB NTFS
Healthy (Boot, Page File, Crash Dump, Primary Partition)

Disk 1
Basic
931.51 GB
Online

NTFS (G:)
2.00 GB NTFS
Healthy (Prima

CNTFS (I:)
2.00 GB NTFS
Healthy (Prima

FAT (K:)
1.00 GB FAT
Healthy (Prin

FAT32 (L:)
1.00 GB FAT3
Healthy (Prin

(M:)
2.00 GB exFAT
Healthy (Prima

UDF (N:)
2.00 GB UDF
Healthy (Prima

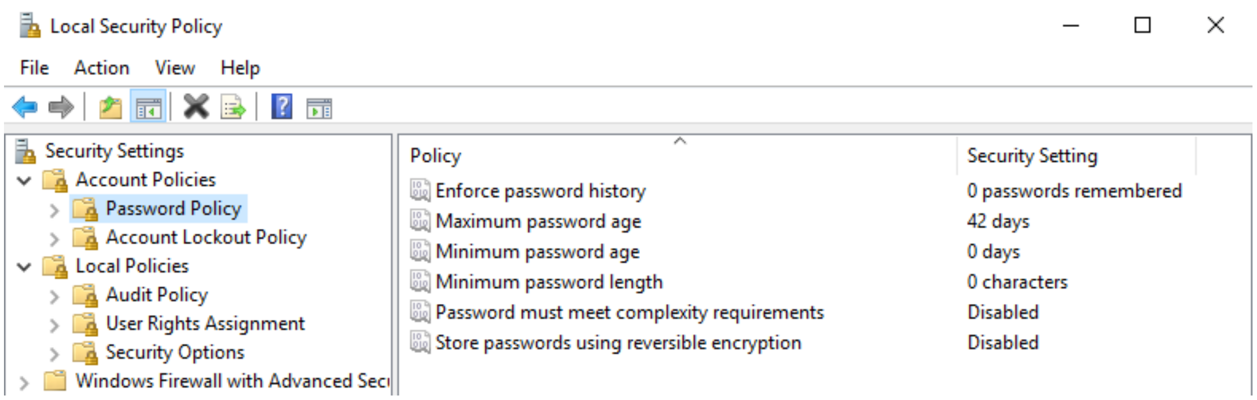
921.51 GB
Unallocated

Password Policy Problems

At this point, I'm pretty sure we are in compliance with all the requirements stipulated by the documentation and the in-application dialog, so I proceed to let the tests start running. After some time, I see that a few tests have failed.

The first failure says:
"Failed to create user account (IFSTEST). Status = 0x8c5"

Thanks. It also reports the filename and line number in the source code of the test case where things went wrong. However, Microsoft doesn't share that source code, so it doesn't do me any good. That's all we have to go on. After trying many things, I eventually start looking into disabling the company password policy on the laptop. I don't have permission to do this, but with some help from IT and a couple reboots, the password policy is eventually disabled:



Policy	Security Setting
Enforce password history	0 passwords remembered
Maximum password age	42 days
Minimum password age	0 days
Minimum password length	0 characters
Password must meet complexity requirements	Disabled
Store passwords using reversible encryption	Disabled

Takeaway: This test creates a synthetic user with a weak password. We can't change the password it chooses, so we have to disable the password policy.

I do not recommend removing the computer from the domain since I suspect it would disrupt connectivity between the HLK server and client. I believe IT removed the machine from the "password policy OU" (I have yet to discover what this stands for) to disable the password policy.

USB Hard Disk Problems

Now I attempt the test again. Good news and bad news: The test gets further but still fails. The new failure mode is as follows:

```
1690.1F40 : +TEST+SEV2 : Test :CreatePagingFileTest
Group :OpenCreateGeneral
Status :C000001E (IFSTEST_TEST_NTAPI_FAILURE_CODE)
LastNtStatus :C00000BB STATUS_NOT_SUPPORTED
ExpectedNtStatus :C0000010
STATUS_INVALID_DEVICE_REQUEST
Description :{Msg# OpCreatG!crpagef!11b} NtCreatePagingFile
should
return STATUS_NOT_SUPPORTED on UDFS and
STATUS_INVALID_DEVICE_REQUEST on DAX or a surprise
removal media.RemovableMedia 0, RemovalPolicy 3,
VirtualDisk 0
WDKTUID: A72C4D26-977B-19CB-2E11-D6F93917D718
```

So the test expected NtCreatePagingFile to fail with STATUS_NOT_SUPPORTED, but it failed with STATUS_INVALID_DEVICE_REQUEST instead. Now the big question is, is our driver responsible for causing this problem, or is it something else? It seems pretty unlikely to me that our driver could be responsible for this particular problem, so I proceed to uninstall our driver from the test system. I restart this whole process and test the “null.sys” driver that comes with Windows. I recommend anyone using HLK should try to verify a known-good driver that comes with Windows before attempting to run any tests on her own software. This will suss out any configuration problems before you actually start testing your code. Not surprisingly, null.sys also fails this test in the same way, so it isn’t my fault.

Microsoft isn’t going to accept my submission, even if it is their test or USB disk driver that’s buggy. So we still need to figure out how to fix this problem ourselves. My hunch based on the error message is that NtCreatePagingFile will always fail to create paging files on removable media, because, if the device hosting the paging file were removed, the system would have no choice but to blue screen. My guess is that the HLK developers wanted this test to be able to pass with removable media, so they made an allowance for NtCreatePagingFile to fail. But there is a disagreement about exactly which status code NtCreatePagingFile should fail with. Running this test on a removable disk is probably a corner case that didn’t get tested very well. In any case, we can’t modify Windows or the test case. All we can change is the hardware--let’s make sure we aren’t exercising the removable media corner case.

I return to IT and declare that I will be needing “an incredibly average consumer grade desktop PC similar to what Microsoft probably used to test their tests.” Who knows how many other quirks we have yet to run into? Fast forward a couple weeks: we have a new machine with 4 internal SATA disks (1 system, 1 for everything but ReFS, and 2 for ReFS) 2 of which are already installed.

I decide to upgrade the HLK server to the 1703 build, given that the deadline for using HLK 1703 is fast approaching.

I open up the system and start plugging in the disks. The machine doesn't have enough mounts for these disks, so they just hang out the side. There's a problem, though: I can only find 2 SATA ports on the motherboard. Eventually I locate a third one tucked under the CPU and determine that I will need to swap out the optical drive to get a fourth.

I turn the machine on with 3 hard disks and the optical drive plugged in. Windows thinks there are two disks. What's going on? Hardware RAID is what's going on. Not exactly the average consumer grade PC I was hoping for. I bite the bullet and delete all the RAID volumes, which of course destroys the OS install, necessitating a reinstall. We use Windows 10 1703 in order to match the HLK version (I don't know if this was necessary or not).

Now there are three disks. Install Windows to the first one, boot up, enlist IT's assistance to get it on the domain, swap out the optical drive, and follow all the directions from above w.r.t. configuring the HLK client system.

Eventually I reach the point where I was with the laptop and run the tests again, this time just on null.sys. The tests fail because we forgot to disable the password policy. I try again the next day, and finally, the tests succeed (on null.sys).

NX Bit Problems

Now it's time to test our software. I install it and run the tests. The DeviceGuard test fails. It is readily apparent that I need to change our driver so it uses the No Execute bit for all memory allocations:
https://msdn.microsoft.com/en-us/library/windows/hardware/hh920402(v=vs.85).aspx

I follow said directions and run into an Undefined Reference for ExInitializeDriverRuntime. I was afraid of this, but it was going to happen sooner or later. We were still using the venerable wdk7600 toolchain to build our kernel code, even though we had switched to the Enterprise WDK 10 (https://docs.microsoft.com/en-us/windows-hardware/drivers/develop/installing-the-enterprise-wdk) for all our userland code ages ago. The solution to the Undefined Reference was to switch to EWDK10 for kernel code too.

Why do we use EWDK10 instead of Visual Studio? Ages ago, before SCons or cmake existed, we did use Visual Studio. The biggest problem was that there were two build systems: Windows, and everything but Windows. The Windows build was second class and frequently broken. Different machines had different versions of Visual Studio or different patch levels, which, as it turns out, causes quite a bit of pain for a large software project. The “new” build system is an SCons-based system that downloads the entire build toolchain, at least on Windows where there is good binary compatibility between different OS versions. This way, we use exactly the same compiler and environment for building our software on every Windows computer at the company.

I used CodeSonar’s build monitoring capability to discover exactly what flags the Visual Studio IDE passes to the Visual C++ compiler to compile kernel code. Using these flags, I was able to produce an equivalent SCons construction environment for building device drivers using SCons’ SharedLibrary primitive (a driver is basically just a shared library loaded by the kernel executable).

Having done this, we could finally stop using wdk7600 (well, not really, because we still use ATL headers that wdk7600 has and EWDK10 doesn’t have). I proceed to install the drivers compiled using the new toolchain on the test machine and rerun the tests. All tests pass. Hooray!

Static Driver Verifier

A Static Driver Verifier (SDV) log has to be included with “applicable” HLK submissions. As it turns out, “applicable” does not include MiniFilter drivers because SDV isn’t capable of analyzing them, but I didn’t know that. EWDK10 doesn’t include SDV, so I had to find a machine with Visual Studio installed, create a VS project that could build our driver code, and attempt to activate SDV. It refused to run, due to the driver type.

I did have the opportunity to run PreFast and eliminate every last warning it produced. The most annoying false positive was a warning about the OutputBuffer output parameter of our CommPortMessage callback not being initialized in cases where *ReturnOutputBufferLength was 0 because no output data needed to be returned. After reading a bit about Microsoft’s Static Annotation Language (SAL), I added the annotation below to communicate the relationship between OutputBufferLength and *ReturnOutputBufferLength to PreFast.

```
static NTSTATUS
CommPortMessage(
      __in PVOID PortCookie,
      __in_opt PVOID InputBuffer,
      __in ULONG InputBufferLength,
      _Out_writes_bytes_to_opt_(OutputBufferLength, *ReturnOutputBufferLength) PVOID
OutputBuffer,
      __in ULONG OutputBufferLength,
      __out PULONG ReturnOutputBufferLength
      );
```

Another false positive that struck me as unexpected (my primary job is to develop static analysis software, so I know something about how these things work) was a buffer overrun on the memset below:

```
oht->data = ExAllocatePoolWithTag(
      PagedPool, sizeof( CELL ) * oht->size, drvhookcsmf_STRING_TAG );
 if( cs_unlikely( !oht->data ) )
     return CS_ERROR_OUT_OF_MEMORY;
 if( oht->size > 1024*1024 )
 {}
 memset( oht->data, 0, sizeof( CELL ) * oht->size );
```

Clearly, oht->data is exactly as big as it needs to be for this memset, since it was just allocated. If we comment out the "if(oht->size > 1024*1024)" statement, then the false positive goes away.

32-bit Driver

We also need to ship a 32-bit version of the driver. I won't go into much detail, but you basically want to set up a 32-bit HLK client machine and repeat the whole process with that system. As of June 2017, this system only needs 2 disks, because Microsoft hasn't enabled Resilient FS for 32-bit systems. So it can't be tested. Otherwise, the hardware configuration can be the same.

Submission Package

Now we need to provide a submission package for Microsoft. HLK wants me to add the "Driver Directory" to the package. I wish I had any idea what a "Driver Directory" was, but I try the obvious thing: Create an empty directory and put the 64-bit and 32-bit .sys files in it. This, as it turns out, does not work. A .inf file also needs to be placed inside the driver directory. We have an old .inf file on the shelf that we don't use (because we use CreateService and friends to install the driver). I make some modest updates to the .inf file, which is for the most part exactly the same as the inf file generated by Visual Studio when you create a new MiniFilter project.

Once the directory with the two sys files and the inf file is added as the driver directory, right click it in the HLK, and click "Add Symbols." Add pdb files for the drivers. No supplemental folder is necessary. Now plug the USB token with the EV certificate private key into the HLK server machine, if it isn't a VM. Click "Create Package" and sign the package if it's not a VM. If it is a VM, then don't sign the package.

If the package isn't signed, you need to install HLK Studio on a physical computer. Copy the package there, open it in HLK Studio, plug in the USB token, and then sign the package.

Finally, I submit the package to the hardware dashboard site. In my case, the submission was accepted, and I had signed drivers in under ten minutes.