Driver security checklist

# Driver security checklist

This topic provides a driver security checklist for driver developers.

## Driver security overview

Anything an attacker can do that causes a driver to malfunction in such a way that it causes the system to crash or become unusable is a security flaw. In addition, vulnerabilities in driver code can allow an attacker to gain access to kernel creating an avenue to compromise the entire OS. When most developers are working on their driver, their focus is on getting the driver to work properly and not whether a malicious attacker will attempt to exploit holes within their code.

However, after a driver is released, there are attackers who attempt to probe and identify security weaknesses. Developers must consider these issues during the design and implementation phase in order to minimize the likelihood that such vulnerabilities exist. The goal is to eliminate all known security gaps before the driver is released.

Creating secure drivers requires the cooperation of the system architect (consciously thinking of potential threats to the driver), the developer implementing the code (defensively coding common operations that can be the source of exploits), and the test team (pro-actively attempting to find weakness and vulnerabilities). By properly coordinating all of these activities, the security of the driver will be dramatically enhanced.

**Security checklist:** *Complete the security task described in each of these topics.*

[Confirm that a kernel driver is required](#confirmkernel)

[Control access to software only drivers](#controlsoftwareonly)

[Threat analysis and threat model creation](#threatanalysis)

[Driver security code practices](#driversecuritycodepractices)

[Device Guard compatibility](#dgc)

[Technology specific code practices](#technologyspecificcodepractices)

[Managing driver access control](#managingdriveraccesscontrol)

[Code validation tools](#codevalidationtools)

[Use code analysis in Visual Studio to investigate driver security](#use-code-analysis)

[Use Static Driver Verifier to Check for Vulnerabilities](#sdv)

[Use the Device Guard Readiness Tool to evaluate HVCI driver compatibility](#use-the-device-guard-readiness-tool)

[Check code with Binscope Binary Analyzer](#binscope)

[Debugger techniques and extensions](#debugger)

[Enhance device installation security](#enhancedeviceinstallationsecurity)

[Release driver signing](#releasedriversigning)

[Secure coding resources](#securecodingresources)

[Key takeaways](#keytakeaways)

## Confirm that a kernel driver is required

**Security checklist item #1:** *Confirm that a kernel driver is required and that a lower risk approach such as Windows service or app is not a better approach.*

Drivers live in the windows kernel, and having an issue when executing in kernel exposes the entire operating system. If any other option is available, it likley will be lower cost and have less associated risk than a kernel driver.

For more information about the built in Windows drivers, see [Do you need to write a driver?](https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/do-you-need-to-write-a-driver-).

For information on if you can use the lower risk user mode framework driver (UMDF), see [Choosing a driver model](https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/choosing-a-driver-model).

For information on using background tasks, see [Support your app with background tasks](https://docs.microsoft.com/windows/uwp/launch-resume/support-your-app-with-background-tasks).

For information on Windows Services, see [Services](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685141(v=vs.85).aspx).

## Control access to software only drivers

**Security checklist item #2:** *If a software only driver is going to be created addtional access control must be implemented.*

Software only kernel drivers may not use PnP to become associated with specific hardware IDs. Software only kernel drivers contain addtional risk and must be limited to run on specific hardware.

Code signed by a trusted SPC (Sofware Publishers Certificate) or WHQL (Windows Hardware Quality Labs) signature must not facilitate bypass of Windows code integrity and security technologies. Before code is signed by a trusted SPC or WHQL signature, first ensure it complies with guidance from both [Device.DevFund.Reliability.BasicSecurity](https://docs.microsoft.com/en-us/windows-hardware/design/compatibility/1703/device-devfund#device.devfund.reliability) and [Creating Reliable Kernel-Mode Drivers](https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/creating-reliable-kernel-mode-drivers). In addition the code must not contain any dangerous behaviors, described below. For more information about driver signing, see [Release driver signing](#releasedriversigning), later in this topic.

**Examples of dangerous behavior include** - Providing the ability to map arbitrary kernel, physical, or device memory to user mode - Providing the ability to read or write arbitrary kernel, physical or device memory, including Port I/O (inp, outp). - Providing access to storage that bypasses Windows access control - Providing the ability to modify hardware or firmware the driver was not designed to manage.

*Development, testing, and manufacturing kernel driver code*

Kernel driver code that is used for development, testing, or manufacturing might include dangerous capabilities that pose a security risk. This dangerous code should never be signed with a certificate that is trusted by Windows. The correct mechanism for executing dangerous driver code is to disable UEFI Secure Boot, enable the BCD “TESTSIGNING”, and sign the development/test/manufacturing code using an untrusted certificate - generated by makecert.exe for example.

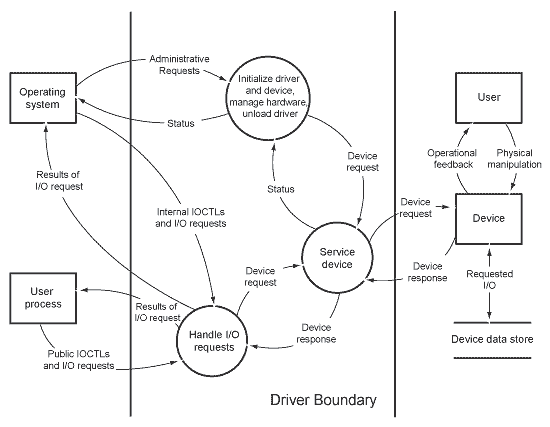
For example, imagine OEM Fabrikam wants to distribute a driver that enables an overclocking utility for their systems. However, if this software only driver were to execute on a system from a different OEM, system instability or damage might result. Fabrikam’s systems should include a unique plug-and-play ID to enable creation of a PnP driver, which is also updatable via Windows Update. If this is not possible, and Fabrikam authors a Legacy driver, that driver should find another method to verify it is executing on a Fabrikam system, for example by examination of the SMBIOS table, prior to enabling any capabilities.

## Threat analysis and threat model creation

**Security checklist item #3:** *Either modify an existing driver threat model or create a custom threat model for your driver.*

In considering security, a common methodology is to create specific threat models that attempt to describe the types of attacks that are possible. This technique is useful when designing a driver because it forces the developer to consider the potential attack vectors against a driver in advance. Having identified potential threats, a driver developer can then consider means of defending against these threats in order to bolster the overall security of the driver component.

This topic provides driver specific guidance for creating a light weight threat model - [Threat modeling for drivers](threat-modeling-for-drivers.md). That topic shows an example driver threat model diagram that can be used as a starting point for your driver.



sample data flow diagram for hypothetical kernel-mode driver

Security Development Lifecycle (SDL) best practices and associated tools can be used by IHVs and OEMs to improve the security of their products. For more information see [SDL recommendations for OEMs](https://msdn.microsoft.com/windows/hardware/drivers/bringup/security-overview#sdl).

## Driver security code practices

**Security checklist item #4:** *Review your code and remove any known code vulnerabilities.*

The core activity of creating secure drivers is identifying areas in the code that need to be changed to avoid known software vulnerabilities. Many of these known software vulnerabilities deal with keeping strict track of the use of memory to avoid issues with others overwriting or otherwise comprising the memory locations that your driver uses.

**Code review**

Seek out knowledgeable code review to look for issues, that you may have missed. A second set of eyes will often see issues that you may have missed.

The [Code Validation Tools](#codevalidationtools) section of this topics describes software tools that can be used to help locate known software vulnerabilities.

**Memory buffers**

Always check the sizes of the input and output buffers to ensure that the buffers can hold all the requested data.

For more information, see [Failure to Check the Size of Buffers](https://msdn.microsoft.com/library/windows/hardware/ff545679)

Properly initialize all output buffers with zeros before returning them to the caller.

For more information, see [Failure to Initialize Output Buffers](https://msdn.microsoft.com/library/windows/hardware/ff545693)

Validate variable-length buffers

For more information, see [Failure to Validate Variable-Length Buffers](https://msdn.microsoft.com/library/windows/hardware/ff545709).

For more information about working with buffers and using [**ProbeForRead**](https://msdn.microsoft.com/library/windows/hardware/ff559876) and [**ProbeForWrite**](https://msdn.microsoft.com/library/windows/hardware/ff559879) to validate the address of a buffer, see [Buffer Handling](https://msdn.microsoft.com/library/windows/hardware/ff539004).

**Use the appropriate method for accessing data buffers with IOCTLs**

One of the primary responsibilities of driver stacks is transferring data between user-mode applications and a system's devices. There are three methods for accessing data buffers.

|  |  |  |
| --- | --- | --- |
| IOCTL Buffer Type | Summary | For more information |
| METHOD\_BUFFERED | Recommended for most situtations | [Using Buffered I/O](https://docs.microsoft.com/windows-hardware/drivers/kernel/using-buffered-i-o) |
| METHOD\_IN\_DIRECT or METHOD\_OUT\_DIRECT | Used in some high speed HW I/O | [Using Direct I/O](https://docs.microsoft.com/windows-hardware/drivers/kernel/using-direct-i-o) |
| METHOD\_NEITHER | Avoid if possible | [Using Neither Buffered Nor Direct I/O](https://docs.microsoft.com/windows-hardware/drivers/kernel/using-neither-buffered-nor-direct-i-o) |

In general buffered I/O is recomended as it provides the most secure buffering methods. But, even when using buffered I/O there are risks such as embedded pointers that must be mitigated.

For more information about the working with buffers in IOCTLs, see [Methods for Accessing Data Buffers](https://docs.microsoft.com/windows-hardware/drivers/kernel/methods-for-accessing-data-buffers)

**Errors in use of IOCTL buffered I/O**

Check the size of IOCTL related buffers. For more information, see [Failure to Check the Size of Buffers](https://msdn.microsoft.com/library/windows/hardware/ff545679).

Properly initialize output buffers. For more information, see [Failure to Initialize Output Buffers](https://msdn.microsoft.com/library/windows/hardware/ff545693).

Properly validate variable-length buffers. For more information, see [Failure to Validate Variable-Length Buffers](https://msdn.microsoft.com/library/windows/hardware/ff545709).

**Errors in IOCTL direct I/O**

Handle zero-length buffers correctly. For more information, see [Errors in Direct I/O](https://msdn.microsoft.com/library/windows/hardware/ff544300).

**Errors in referencing user-space addresses** Pointers embedded in buffered I/O requests must be validated. For more information, see [Errors in Referencing User-Space Addresses](https://msdn.microsoft.com/library/windows/hardware/ff544308).

Validate any address in user space before trying to use it, using APIs such as [**ProbeForRead**](https://msdn.microsoft.com/library/windows/hardware/ff559876) and [**ProbeForWrite**](https://msdn.microsoft.com/library/windows/hardware/ff559879) when appropriate.

**Driver code must make correct use of memory**

All driver pool allocations must be in NX pool. Using non-executable memory pools, it is inherently more secure as compared to executable non-paged (NP) pool, and provides better protection against overflow attacks. For more information about the related device fundamentals test, see [Device.DevFund.Memory.NXPool](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund#devicedevfundmemory).

Device driver must properly handle various user-mode as well as kernel to kernel I/O requests. For more information about the related device fundamentals test, see [Device.DevFund.Reliability.BasicSecurity](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund#devicedevfundreliability).

To allow drivers to support HVCI virtualization, there are additional memory requirements. For more information, see [Device Guard Compatibility](#dgc) later in this topic.

**Handles**

Validate handles passed between user-mode and kernel-mode memory. For more information, see [Handle Management](https://msdn.microsoft.com/library/windows/hardware/ff547382).

Validate object handles. For more information, see [Failure to Validate Object Handles](https://msdn.microsoft.com/library/windows/hardware/ff545703).

**Device objects**

Secure device objects. For more information, see [Securing Device Objects](https://msdn.microsoft.com/library/windows/hardware/ff563688).

Validate device objects. For more information, see [Failure to Validate Device Objects](https://msdn.microsoft.com/library/windows/hardware/ff545700).

**IRP**

**Validate IRP input values**

Validate all values that are associated with an IRP, such as buffer addresses and lengths. The following topics provide information about validating IRP input values.

[DispatchReadWrite Using Buffered I/O](https://msdn.microsoft.com/library/windows/hardware/ff543388)

[Errors in Buffered I/O](https://msdn.microsoft.com/library/windows/hardware/ff544293)

[DispatchReadWrite Using Direct I/O](https://msdn.microsoft.com/library/windows/hardware/ff543393)

[Errors in Direct I/O](https://msdn.microsoft.com/library/windows/hardware/ff544300)

[Security Issues for I/O Control Codes](https://msdn.microsoft.com/library/windows/hardware/ff563700)

[Errors in Referencing User-Space Addresses](https://msdn.microsoft.com/library/windows/hardware/ff544308)

**Handle IRP completion operations properly**

A driver must never complete an IRP with a status value of STATUS\_SUCCESS unless it actually supports and processes the IRP. For information about the correct ways to handle IRP completion operations, see [Completing IRPs](https://msdn.microsoft.com/library/windows/hardware/ff542018).

**Manage driver IRP pending state**

The driver should mark the IRP pending before it saves the IRP and should include both the call to [**IoMarkIrpPending**](https://msdn.microsoft.com/library/windows/hardware/ff549422) and the assignment in an interlocked sequence. For more information, see [Failure to Check a Driver's State](https://msdn.microsoft.com/library/windows/hardware/ff545672).

**Handle IRP cancellation operations properly**

Cancel operations can be difficult to code properly because they typically execute asynchronously. Problems in the code that handles cancel operations can go unnoticed for a long time, because this code is typically not executed frequently in a running system. Be sure to read and understand all of the information supplied under [Canceling IRPs](https://msdn.microsoft.com/library/windows/hardware/ff540748). Pay special attention to [Synchronizing IRP Cancellation](https://msdn.microsoft.com/library/windows/hardware/ff564531) and [Points to Consider When Canceling IRPs](https://msdn.microsoft.com/library/windows/hardware/ff559700).

One way to avoid the synchronization problems that are associated with cancel operations is to implement a [cancel-safe IRP queue](https://msdn.microsoft.com/library/windows/hardware/ff540755).

**Handle IRP cleanup and close operations properly**

Be sure that you understand the difference between [**IRP\_MJ\_CLEANUP**](https://msdn.microsoft.com/library/windows/hardware/ff550718) and [**IRP\_MJ\_CLOSE**](https://msdn.microsoft.com/library/windows/hardware/ff550720) requests. Cleanup requests arrive after an application closes all handles on a file object, but sometimes before all I/O requests have completed. Close requests arrive after all I/O requests for the file object have been completed or canceled. For more information, see the following topics:

[DispatchCreate, DispatchClose, and DispatchCreateClose Routines](https://msdn.microsoft.com/library/windows/hardware/ff543279)

[DispatchCleanup Routines](https://msdn.microsoft.com/library/windows/hardware/ff543242)

[Errors in Handling Cleanup and Close Operations](https://msdn.microsoft.com/library/windows/hardware/ff544304)

For more information about handling IRPs correctly, see [Additional Errors in Handling IRPs](https://msdn.microsoft.com/library/windows/hardware/ff540543).

**Other security issues**

Properly handle the cleanup and close sequence of operations. For more information, see [Errors in Handling Cleanup and Close Operations](https://msdn.microsoft.com/library/windows/hardware/ff544304).

Use a lock or an interlocked sequence to prevent race conditions. For more information, see [Errors in a Multiprocessor Environment](https://msdn.microsoft.com/library/windows/hardware/ff544288).

Device driver must properly handle various user-mode as well as kernel to kernel I/O requests. For more information, see [Device.DevFund.Reliability.BasicSecurity](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund#devicedevfundreliability).

No TDI filters or LSPs are installed by the driver or associated software packages during installation or usage. For more information about the related driver fundamentals test, see [Device.DevFund.Security](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund#devicedevfundsecurity).

**Use safe functions**

* Use safe string functions. For more information, see [Using Safe String Functions](https://msdn.microsoft.com/library/windows/hardware/ff565508).
* Use safe arithmetic functions. For more information, see [Arithmetic Functions](https://msdn.microsoft.com/library/windows/hardware/hh406348) in [Safe Integer Library Routines](https://msdn.microsoft.com/library/windows/hardware/hh406570)
* Use safe conversion functions. For more information, see [Conversion Functions](https://msdn.microsoft.com/library/windows/hardware/hh450942) in [Safe Integer Library Routines](https://msdn.microsoft.com/library/windows/hardware/hh406570)

**Additional code vulnerabilities**

In addition to the possible vulnerabilities covered here, this topic provides additional information about enhancing the security of kernel mode driver code - [Creating Reliable Kernel-Mode Drivers](https://msdn.microsoft.com/library/windows/hardware/ff542904).

For additional information about C and C++ secure coding, see [Secure coding resources](#securecodingresources) and the end of this topic.

## Device Guard compatibility

**Security checklist item #5:** *Validate that your driver uses memory so that is Device Guard Compatible.*

**Memory usage and Device Guard compatibility**

Device Guard uses hardware technology and virtualization to isolate the Code Integrity (CI) decision-making function from the rest of the operating system. When using virtualization-based security to isolate Code Integrity, the only way kernel memory can become executable is through a Code Integrity verification. This means that kernel memory pages can never be Writable and Executable (W+X) and executable code cannot be directly modified.

To implement Device Guard, make sure your driver code does the following.

* Opt-in to NX by default
* Use NX APIs/flags for memory allocation – NonPagedPoolNx
* Do not use sections that are both writable and executable
* Do not attempt to directly modify executable system memory
* Do not use dynamic code in kernel
* Do not load data files as executable
* Section alignment must be a multiple of 0x1000 (PAGE\_SIZE). E.g. DRIVER\_ALIGNMENT=0x1000

The following list of DDIs that are not reserved for system use may be impacted:

|  |
| --- |
| DDI name |
| [**ExAllocatePool**](https://msdn.microsoft.com/library/windows/hardware/ff544501) |
| [**ExAllocatePoolWithQuota**](https://msdn.microsoft.com/library/windows/hardware/ff544506) |
| [**ExAllocatePoolWithQuotaTag**](https://msdn.microsoft.com/library/windows/hardware/ff544513) |
| [**ExAllocatePoolWithTag**](https://msdn.microsoft.com/library/windows/hardware/ff544520) |
| [**ExAllocatePoolWithTagPriority**](https://msdn.microsoft.com/library/windows/hardware/ff544523) |
| [**ExInitializeNPagedLookasideList**](https://msdn.microsoft.com/library/windows/hardware/ff545301) |
| [**ExInitializeLookasideListEx**](https://msdn.microsoft.com/library/windows/hardware/ff545298) |
| [**MmAllocateContiguousMemory**](https://msdn.microsoft.com/library/windows/hardware/ff554460) |
| [**MmAllocateContiguousMemorySpecifyCache**](https://msdn.microsoft.com/library/windows/hardware/ff554464) |
| [**MmAllocateContiguousMemorySpecifyCacheNode**](https://msdn.microsoft.com/library/windows/hardware/ff554469) |
| [**MmAllocateContiguousNodeMemory**](https://msdn.microsoft.com/library/windows/hardware/jj602795) |
| [**MmCopyMemory**](https://msdn.microsoft.com/library/windows/hardware/dn342884) |
| [**MmMapIoSpace**](https://msdn.microsoft.com/library/windows/hardware/ff554618) |
| [**MmMapLockedPages**](https://msdn.microsoft.com/library/windows/hardware/ff554622) |
| [**MmMapLockedPagesSpecifyCache**](https://msdn.microsoft.com/library/windows/hardware/ff554629) |
| [**MmProtectMdlSystemAddress**](https://msdn.microsoft.com/library/windows/hardware/ff554670) |
| [**ZwAllocateVirtualMemory**](https://msdn.microsoft.com/library/windows/hardware/ff566416) |
| [**ZwCreateSection**](https://msdn.microsoft.com/library/windows/hardware/ff566428) |
| [**ZwMapViewOfSection**](https://msdn.microsoft.com/library/windows/hardware/ff566481) |
| [**NtCreateSection**](https://msdn.microsoft.com/library/windows/hardware/ff556473) |
| [**NtMapViewOfSection**](https://msdn.microsoft.com/library/windows/hardware/ff556551) |
| [**ClfsCreateMarshallingArea**](https://msdn.microsoft.com/library/windows/hardware/ff541520) |
| NDIS |
| [**NdisAllocateMemoryWithTagPriority**](https://msdn.microsoft.com/library/windows/hardware/ff561606) |
| Storage |
| [**StorPortGetDataInBufferSystemAddress**](https://msdn.microsoft.com/library/windows/hardware/jj553720) |
| [**StorPortGetSystemAddress**](https://msdn.microsoft.com/library/windows/hardware/ff567100) |
| [**ChangerClassAllocatePool**](https://msdn.microsoft.com/library/windows/hardware/ff551402) |
| Display |
| [*DxgkCbMapMemory*](https://msdn.microsoft.com/library/windows/hardware/ff559533) |
| [**VideoPortAllocatePool**](https://msdn.microsoft.com/library/windows/hardware/ff570180) |
| Audio Miniport |
| [**IMiniportDMus::NewStream**](https://msdn.microsoft.com/library/windows/hardware/ff536701) |
| [**IMiniportMidi::NewStream**](https://msdn.microsoft.com/library/windows/hardware/ff536710) |
| [**IMiniportWaveCyclic::NewStream**](https://msdn.microsoft.com/library/windows/hardware/ff536723) |
| [**IPortWavePci::NewMasterDmaChannel**](https://msdn.microsoft.com/library/windows/hardware/ff536916) |
| [**IMiniportWavePci::NewStream**](https://msdn.microsoft.com/library/windows/hardware/ff536735) |
| Audio Port Class |
| [**PcNewDmaChannel**](https://msdn.microsoft.com/library/windows/hardware/ff537712) |
| [**PcNewResourceList**](https://msdn.microsoft.com/library/windows/hardware/ff537717) |
| [**PcNewResourceSublist**](https://msdn.microsoft.com/library/windows/hardware/ff537718) |
| IFS |
| [**FltAllocatePoolAlignedWithTag**](https://msdn.microsoft.com/library/windows/hardware/ff541762) |
| [**FltAllocateContext**](https://msdn.microsoft.com/library/windows/hardware/ff541710) |
| WDF |
| [**WdfLookasideListCreate**](https://msdn.microsoft.com/library/windows/hardware/ff548694) |
| [**WdfMemoryCreate**](https://msdn.microsoft.com/library/windows/hardware/ff548706) |
| [**WdfDeviceAllocAndQueryProperty**](https://msdn.microsoft.com/library/windows/hardware/ff545882) |
| [**WdfDeviceAllocAndQueryPropertyEx**](https://msdn.microsoft.com/library/windows/hardware/dn265599) |
| [**WdfFdoInitAllocAndQueryProperty**](https://msdn.microsoft.com/library/windows/hardware/ff547239) |
| [**WdfFdoInitAllocAndQueryPropertyEx**](https://msdn.microsoft.com/library/windows/hardware/dn265612) |
| [**WdfIoTargetAllocAndQueryTargetProperty**](https://msdn.microsoft.com/library/windows/hardware/ff548585) |
| [**WdfRegistryQueryMemory**](https://msdn.microsoft.com/library/windows/hardware/ff549920) |

  For more information about using the tool, see [Use the Device Guard Readiness Tool to evaluate HVCI driver compatibility](#use-the-device-guard-readiness-tool) later in this topic.

For more information about the related device fundamentals test, see [Device.DevFund.DeviceGuard](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund#devicedevfunddeviceguard).

For general information about Device Guard, see [Windows 10 Device Guard and Credential Guard Demystified](https://blogs.msdn.microsoft.com/windows_hardware_certification/2015/05/22/driver-compatibility-with-device-guard-in-windows-10/) and [Device Guard deployment guide](https://docs.microsoft.com/en-us/windows/device-security/device-guard/device-guard-deployment-guide)

## Technology specific code practices

**Security checklist item #6:** *Review the following technology specific guidance for your driver.*

*File Systems*

For more information, about file system driver security see the following topics.

[Security Considerations for File Systems](https://msdn.microsoft.com/windows/hardware/drivers/ifs/security-considerations-for-file-systems)

[File System Security Issues](https://msdn.microsoft.com/windows/hardware/drivers/ifs/file-system-security-issues)

[Security Features for File Systems](https://msdn.microsoft.com/windows/hardware/drivers/ifs/security-features-for-file-systems)

[Security Considerations for File System Filter Drivers](https://msdn.microsoft.com/windows/hardware/drivers/ifs/security-considerations-for-file-system-filter-drivers)

*NDIS - Networking*

For about NDIS driver security, see [Security Issues for Network Drivers](https://msdn.microsoft.com/windows/hardware/drivers/network/security-issues-for-network-drivers)

*Display*

For information about display driver security, see <Content Pending>.

*Bluetooth*

For information about Bluetooth driver security, see <Content Pending>.

*Printers*

For information related to printer driver security, see [V4 Printer Driver Security Considerations](https://msdn.microsoft.com/library/windows/hardware/jj863679).

*Security Issues for Windows Image Acquisition (WIA) Drivers*

For information about WIA security, see [Security Issues for Windows Image Acquisition (WIA) Drivers](https://msdn.microsoft.com/windows/hardware/drivers/image/security-issues-for-wia-drivers)

## Managing driver access control

**Security checklist item #7:** *Review your driver to make sure that your are properly controlling access.*

**Managing Driver Access Control**

Drivers must help to prevent users from inappropriately accessing a computer's devices and files. To prevent unauthorized access to devices and files, you must:

* Name device objects only when necessary.
* Provide security descriptors for device objects and interfaces.

For more information, review these topics.

[Controlling Device Access in KMDF Drivers](https://msdn.microsoft.com/windows/hardware/drivers/wdf/controlling-device-access-in-kmdf-drivers)

[SDDL for Device Objects](https://msdn.microsoft.com/library/windows/hardware/ff563667)

[SID Strings](https://msdn.microsoft.com/en-us/library/windows/desktop/aa379602(v=vs.85).aspx)

[Controlling Device Access](https://msdn.microsoft.com/library/windows/hardware/ff542063)

[Controlling Device Namespace Access](https://msdn.microsoft.com/library/windows/hardware/ff542068)

"Names, Security Descriptors and Device Classes " on page 6 of the *January February 2017 The NT Insider Newsletter* published by [OSR](http://www.osr.com).

**Additional general Windows security model information**

[Windows security model: what every driver writer needs to know](windows-security-model--what-every-driver-writer-needs-to-know.md)

[Windows security model scenario: creating a file](windows-security-model-scenario--creating-a-file.md)

## Enhance device installation security

**Security checklist item #8:** *Review driver inf creation and installation guidance to make sure you are following best practices.*

For more information, review these topics.

[Creating Secure Device Installations](https://msdn.microsoft.com/windows/hardware/drivers/install/creating-secure-device-installations)

[Guidelines for Using SetupAPI](https://msdn.microsoft.com/windows/hardware/drivers/install/guidelines-for-using-setupapi)

[Using Device Installation Functions](https://msdn.microsoft.com/windows/hardware/drivers/install/using-device-installation-functions)

[Device and Driver Installation Advanced Topics](https://msdn.microsoft.com/windows/hardware/drivers/install/device-and-driver-installation-advanced-topics) ## Release driver signing

**Security checklist item #9:** *Use the Windows partner portal to sign your driver for distribution.*

Before you release a driver package to the public, we recommend that you submit the package for certification. For more information, see [Test for performance and compatibility](https://msdn.microsoft.com/windows/hardware/commercialize/test/index), [Get started with the Hardware program](https://msdn.microsoft.com/windows/hardware/drivers/dashboard/get-started-with-the-hardware-dashboard), [Hardware Dashboard Services](https://msdn.microsoft.com/windows/hardware/drivers/dashboard/dashboard-services), and [Attestation signing a kernel driver for public release](https://msdn.microsoft.com/windows/hardware/drivers/dashboard/attestation-signing-a-kernel-driver-for-public-release).

## Code validation tools

**Security checklist item #10:** *Use these tools to help validate that your code follows security recommendations and to probe for to look for gaps that were missed in your development process.*

**Code analysis for drivers**

The code analysis feature in Visual Studio to check for security vulnerabilities in your code. The Windows Driver Kit installs rule sets that are designed to check for issues in native driver code.

For more information see [Code Analysis for drivers overview](https://msdn.microsoft.com/library/windows/hardware/hh698231.aspx). For additional background on code analysis see [Visual Studio 2013 Static Code Analysis in depth](https://blogs.msdn.microsoft.com/hkamel/2013/10/24/visual-studio-2013-static-code-analysis-in-depth-what-when-and-how/)

Lastly, see this walkthrough later in this topic: [Use Code Analysis in Visual Studio to Investigate Driver Security](#use-code-analysis)

**Static Driver Verifier**

Static Driver Verifier (SDV) uses a set of interface rules and a model of the operating system to determine if the driver interacts correctly with the Windows operating system. SDV finds defects in driver code that could point to potential bugs in drivers.

For more information [Introducing Static Driver Verifier](https://msdn.microsoft.com/windows/hardware/drivers/devtest/introducing-static-driver-verifier)

See this walkthrough later in this topic, [Use Static Driver Verifier to Check for Vulnerabilities](#sdv)

**Driver Verifier**

Driver Verifier allows for live testing of the driver. Driver Verifier monitors Windows kernel-mode drivers and graphics drivers to detect illegal function calls or actions that might corrupt the system. Driver Verifier can subject the Windows drivers to a variety of stresses and tests to find improper behavior. For more information, see [Driver Verifier](https://msdn.microsoft.com/windows/hardware/drivers/devtest/driver-verifier).

**Device Guard Readiness Tool**

The Device Guard Readiness Tool is used to evaluate HVCI driver compatibility. See this walkthrough later in this topic: [Use the Device Guard Readiness Tool to Evaluate Driver HVCI compatibility](#use-the-device-guard-readiness-tool)

**Hardware compatibility program tests**

The hardware compatibility program tests can be used on the command line to exercise driver code and probe for weakness. The Device Fundamentals tests can be used on the command line to exercise driver code and probe for weakness. For general information about the device fundamentals tests and the hardware compatibility program, see [Hardware Compatibility Specifications for Windows 10, version 1607](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/index).

The following tests are examples of tests that may be useful to check driver code for some behaviors associated with code vulnerabilities.

Device driver must properly handle various user-mode as well as kernel to kernel I/O requests. For more information, see [Device.DevFund.Reliability.BasicSecurity](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund#devicedevfundreliability)

The Device Fundamentals Penetration tests perform various forms of input attacks, which are a critical component of security testing. Attack and Penetration testing can help identify vulnerabilities in software interfaces. Some basic fuzz testing as well as the IoSpy and IoAttack utilities are included. For more information, see [Penetration Tests (Device Fundamentals)](https://msdn.microsoft.com/windows/hardware/drivers/devtest/penetration-tests--device-fundamentals-) and [How to Perform Fuzz Tests with IoSpy and IoAttack](https://msdn.microsoft.com/windows/hardware/drivers/devtest/how-to-perform-fuzz-tests-with-iospy-and-ioattack).

The CHAOS (Concurrent Hardware and Operating System) tests run various PnP driver tests, device driver fuzz tests, and power system tests concurrently. For more information, see [CHAOS Tests (Device Fundamentals)](https://msdn.microsoft.com/windows/hardware/drivers/devtest/chaos-tests--device-fundamentals-).

Device Path Exerciser runs as part of Device.DevFund.Reliability.BasicSecurity. For more information see [Device.DevFund.Reliability](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund).

All driver pool allocations must be in NX pool. Using non-executable memory pools, it is inherently more secure as compared to executable non-paged (NP) pool, and provides better protection against overflow attacks. For more information see [DevFund.Memory.NXPool](https://msdn.microsoft.com/ie/dn932575#device-devfund-memory-nxpool).

Use the [Device.DevFund.DeviceGuard](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund#device-devfund-deviceguard) test, along with the other tools described in this topic, to confirm that your driver is device guard compatible.

**BinScope Binary Analyzer**

See this walkthrough later in this topic: [Check code with Binscope Analyzer](#binscope)

For more information, see [New Version of BinScope Binary Analyzer](https://blogs.microsoft.com/microsoftsecure/2014/11/20/new-binscope-released/) and the user and getting started guides that are included with the tool download.

**Custom and domain specific test tools**

Consider the development of custom domain specific security tests. To develop additional tests gather input from the original designers of the software, as well as unrelated development resources familiar with the specific type of driver being developed, and one or more people familiar with security intrusion analysis and prevention.

## Use Static Driver Verifier to check for vulnerabilities

**Security checklist item #11:** *Follow these steps to use Static Driver Verifier (SDV) in Visual Studio to check for vulnerabilities in your driver code.*

For more information, see [Static Driver Verifier](https://msdn.microsoft.com/windows/hardware/drivers/devtest/static-driver-verifier). Note that only certain types of drivers are supported by SDV. For more information about the drivers that SDV can verify, see [Supported Drivers](https://msdn.microsoft.com/windows/hardware/drivers/devtest/supported-drivers).

1. Open the targeted driver solution in Visual Studio.

To become familiar with SDV, you can use one of the sample drivers - for example the featured toaster sample. <https://github.com/Microsoft/Windows-driver-samples/tree/master/general/toaster/toastDrv/kmdf/func/featured>

1. In Visual Studio, change the build type to *Release*. Static driver verifier requires that the build type is release, not debug.
2. In Visual Studio, select Build >> Build Solution.
3. In Visual Studio, select Driver >> Launch Static Driver Verifier.
4. In SDV, on the "Rules" tab select *Default* in the pull down under Rule Sets.

Although the default rules find many common issues, consider running the more extensive *All driver rules* rule set as well.

1. In the "Main" tab of SDV, click *Start*.
2. When SDV is complete, review any warnings in the output. The *Main* tab displays the total number of defects found.
3. Click on each warning to load the SDV Report Page and examine the information associated with the possible code vulnerability. Use the report to investigate the verification result and to identify paths in your driver that fail a SDV verification. For more information, see [Static Driver Verifier Report](https://msdn.microsoft.com/library/windows/hardware/ff552834).

## Use code analysis in Visual Studio to investigate driver security

**Security checklist item #12:** *Follow these steps to use the code analysis feature in Visual Studio to check for vulnerabilities in your driver code.*

For more information, see [How to run Code Analysis for drivers](https://msdn.microsoft.com/windows/hardware/drivers/devtest/how-to-run-code-analysis-for-drivers).

1. Open the driver solution in Visual Studio.

To become familiar with code analysis, you can use one of the sample drivers - for example the featured toaster sample.

<https://github.com/Microsoft/Windows-driver-samples/tree/master/general/toaster/toastDrv/kmdf/func/featured>

1. In Visual Studio, for each project in the solution change the project properties to use the desired rule set. For example: Project >> Properties >> Code Analysis >> General, select *Recommended driver rules*.

In addition to using recommenced driver rules, use the *Recommended native rules* tool set as well.

1. Select Build >> Run Code Analysis on Solution.
2. View warnings in **Error List** tab of build output window in Visual Studio.

Click on the description for each warning to see the problematic area in your code.

Click on the linked warning code to see additional information.

Determine if your code needs to be changed, or if an annotation needs to be added to allow the code analysis engine to properly follow the intent of your code. For more information on code annotation, see [Using SAL Annotations to Reduce C/C++ Code Defects](https://msdn.microsoft.com/library/ms182032.aspx) and [SAL 2.0 Annotations for Windows Drivers](https://msdn.microsoft.com/windows/hardware/drivers/devtest/sal-2-annotations-for-windows-drivers).

## Use the Device Guard Readiness Tool to evaluate HVCI driver compatibility

**Security checklist item #13:** *Follow these steps to use Device Guard Readiness Tool to evaluate HVCI driver compatibility of your driver code.*

**Overview**

The Device Guard Readiness tool is designed to check a number of requirements for creating a PC that supports a variety of security enhancement features. This section describes how to use the tool to evaluate the ability of a driver to run in a Hypervisor Code Integrity (HVCI) environment.

OS and Hardware requirements for testing HVCI driver Device Guard compatibility

1. Windows SKUs: Available only on these Windows SKUs - Enterprise, Server and Enterprise IoT
2. Hardware: Recent hardware that supports virtualization extension with SLAT.

To use the readiness tool to evaluate the additional requirements such as secure boot, refer to the readme.txt file included in the readiness tool download.

For more information about the related device fundamentals test, see [Device.DevFund.DeviceGuard](https://msdn.microsoft.com/windows/hardware/commercialize/design/compatibility/device-devfund#devicedevfunddeviceguard).

**Using the tool**

To use the Device Guard Readiness Tool to evaluate complete the following steps.

* **Prepare the test PC**
* *Enable Virtualization Based Protection of Code Integrity* - Run the System Information app (msinfo32). Look for the following item: “Device Guard Virtualization based security”. It should show: “Running”.
* Alternatively, there is also a WMI interface for checking using management tools that can be used to display information in PowerShell.
* Get-CimInstance –ClassName Win32\_DeviceGuard –Namespace root\Microsoft\Windows\DeviceGuard
* For information on how to interrupt the output displayed, see [Deploy Device Guard: enable virtualization-based security](https://technet.microsoft.com/itpro/windows/keep-secure/deploy-device-guard-enable-virtualization-based-security).
* *Disable Device Guard* - Note that while running the Readiness Tool, Device Guard must be disabled on the PC under test, as Device Guard might prevent the driver from loading, and the driver won’t be available for the Readiness Tool to test.
* *Optionaly Enable Test Signing* - To allow for the installation of unsigned development drivers, you may want to enable test signing using BCDEdit.
* bcdedit /set TESTSIGNING ON
* **Install test drivers**
* Install the desired test driver(s) on the target test PC.
* **Important**  After you have tested the development driver and worked through any code issues, retest the final production driver. In addition use the HLK to test the driver. For more information, see [HyperVisor Code Integrity Readiness Test](https://msdn.microsoft.com/library/windows/hardware/dn955152).
* **Install the Device Guard Readiness Tool**
* **Warning**   As the Device Guard Readiness Tool changes registry values and may impact features such as secure boot, use a test PC that doesn't contain any data or applications. After the tests have been run, you may want to re-install Windows to re-establish your desired security configuration.
* 1. Download the tool from here: [Device Guard and Credential Guard hardware readiness tool](https://www.microsoft.com/download/details.aspx?id=53337)
  2. Unzip the tool on the target test machine.
* **Configure PowerShell to allow for the execution of unsigned scripts.**
* The readiness tool is a PowerShell script. To work with the readiness tool script open an Administrator PowerShell script.
* If Execution-Policy is not already set to allow running script, then you should manually set it as shown here.
* Set-ExecutionPolicy Unrestricted
* **Run the readiness tool to enable HVCI**
  1. In Powershell, move to the directory that you unzipped the readiness tool into.
  2. Run the readiness tool to enable HVCI.
* DG\_Readiness\_Tool.ps1 -Enable HVCI
  1. When directed, reboot the PC.
* **Run the script to evaluate HVCI capability**
  1. Run the readiness tool to evaluate the ability of the drivers to support HVCI.
* DG\_Readiness\_Tool.ps1 -Capable HVCI
* **Evaluate the output**
* The output to the screen is color coded.

|  |  |
| --- | --- |
| * Red - Errors | * Elements are missing or not configured that will prevent enabling and using DG/CG. |
| * Yellow - Warnings | * This device can be used to enable and use DG/CG, but additional security benefits will be absent. |
| * Green - Messages | * This device is fully compliant with DG/CG requirements. |

* In addition to the output to the screen, by default the log file with detailed output is located at C:\DGLogs
* There are five sections in the output of the device guard readiness tool output. Step 1 contains the is the driver compatibility information.
* ====================== Step 1 Driver Compat ======================
* Drivers displayed in green have no identified HVCI compatibility issues. If you are interested in evaluating a specific driver, if the driver name is displayed in green and is active and loaded, it has passed the HVCI compatibility test.
* Locate the "InCompatible HVCI Kernel Driver Modules" section shown below, towards the end of the log.
* InCompatible HVCI Kernel Driver Modules found  
    
  Module: TestDriver1.sys  
   Reason: section alignment failures: 9  
  Module: TestDriver2.sys  
   Reason: execute pool type count: 3
* In the sample shown above, two drivers are identified as incompatible. TestDriver1.sys has a memory section alignment failure and TestDriver2.sys has a pool that is configured to use executable memory area.
* The statistics for the seven types of device driver incompatibilities are also available using the !verifier debugger extension. For more information on the !verifier extension, see [**!verifier**](https://msdn.microsoft.com/library/windows/hardware/ff565591).
* Execute Pool Type Count: 3  
   Execute Page Protection Count: 0  
   Execute Page Mapping Count: 0  
   Execute-Write Section Count: 0  
   Section Alignment Failures: 0  
   Unsupported Relocs Count: 0  
   IAT in Executable Section Count: 0
* Use the following table to interpret the output to determine what driver code changes are needed to fix the different types of HVCI incompatibilities.

Warning

Redemption

Execute Pool Type

The caller specified an executable pool type. Calling a memory allocating function that requests executable memory.

Be sure that all pool types contain a non executable NX flag.

Execute Page Protection

The caller specified an executable page protection.

Specify a "no execute" page protection mask.

Execute Page Mapping

The caller specified an executable memory descriptor list (MDL) mapping.

Make sure that the mask that is used contains MdlMappingNoExecute. For more information, see [MmGetSystemAddressForMdlSafe](https://msdn.microsoft.com/en-us/library/windows/hardware/ff554559.aspx)

Execute-Write Section

The image contains an executable and writable section.

Section Alignment Failures

The image contains a section that is not page aligned.

Section Alignment must be a multiple of 0x1000 (PAGE\_SIZE). E.g. DRIVER\_ALIGNMENT=0x1000

Unsupported Relocs

In Windows 10 version 1507 through version 1607, because of the use of Address Space Layout Randomization (ASLR) an issue can arise with address alignment and memory relocation. The operating system needs to relocate the address from where the linker set its default base address to the actual location that ASLR assigned. This relocation cannot straddle a page boundary. For example, consider a 64-bit address value that starts at offset 0x3FFC in a page. It’s address value overlaps over to the next page at offset 0x0003. This type of overlapping relocs is not supported prior to Windows 10 version 1703.

This situation can occur when a global struct type variable initializer has a misaligned pointer to another global, laid out in such a way that the linker cannot move the variable to avoid the straddling relocation. The linker will attempt to move the variable, but there are situations where it may not be able to do so, for example with large misaligned structs or large arrays of misaligned structs. Where appropriate, modules should be assembled using the [/Gy (COMDAT)](https://docs.microsoft.com/en-us/cpp/build/reference/gy-enable-function-level-linking) option to allow the linker to align module code as much as possible.

#include <pshpack1.h>  
  
typedef struct \_BAD\_STRUCT {  
 USHORT Value;  
 CONST CHAR \*String;  
} BAD\_STRUCT, \* PBAD\_STRUCT;  
  
#include <poppack.h>  
  
#define BAD\_INITIALIZER0 { 0, "BAD\_STRING" },  
#define BAD\_INITIALIZER1 \  
 BAD\_INITIALIZER0 \  
 BAD\_INITIALIZER0 \  
 BAD\_INITIALIZER0 \  
 BAD\_INITIALIZER0 \  
 BAD\_INITIALIZER0 \  
 BAD\_INITIALIZER0 \  
 BAD\_INITIALIZER0 \  
 BAD\_INITIALIZER0   
  
#define BAD\_INITIALIZER2 \  
 BAD\_INITIALIZER1 \  
 BAD\_INITIALIZER1 \  
 BAD\_INITIALIZER1 \  
 BAD\_INITIALIZER1 \  
 BAD\_INITIALIZER1 \  
 BAD\_INITIALIZER1 \  
 BAD\_INITIALIZER1 \  
 BAD\_INITIALIZER1   
  
#define BAD\_INITIALIZER3 \  
 BAD\_INITIALIZER2 \  
 BAD\_INITIALIZER2 \  
 BAD\_INITIALIZER2 \  
 BAD\_INITIALIZER2 \  
 BAD\_INITIALIZER2 \  
 BAD\_INITIALIZER2 \  
 BAD\_INITIALIZER2 \  
 BAD\_INITIALIZER2   
  
#define BAD\_INITIALIZER4 \  
 BAD\_INITIALIZER3 \  
 BAD\_INITIALIZER3 \  
 BAD\_INITIALIZER3 \  
 BAD\_INITIALIZER3 \  
 BAD\_INITIALIZER3 \  
 BAD\_INITIALIZER3 \  
 BAD\_INITIALIZER3 \  
 BAD\_INITIALIZER3   
  
BAD\_STRUCT MayHaveStraddleRelocations[4096] = { // as a global variable  
 BAD\_INITIALIZER4  
};

There are other situations involving the use of assembler code, where this issue can also occur.

IAT in Executable Section

The import address table (IAT), should not be an executable section of memory.

This issue occurs when the IAT, is located in a Read and Execute (RX) only section of memory. This means that the OS will not be able to write to the IAT to set the correct addresses for where the referenced DLL.

One way that this can occur is when using the [/MERGE (Combine Sections)](https://docs.microsoft.com/en-us/cpp/build/reference/merge-combine-sections) option in code linking. For example if .rdata (Read-only initialized data) is merged with .text data (Executable code), it is possible that the IAT may end up in an executable section of memory.

**Script customization**

Below is the list of Regkeys and its values for customization of the script to Device Guard and Credential Guard without UEFI Lock.

For RS1 and RS2 – to enable HVCI and CG without UEFI Lock:  
&#39;REG ADD "HKLM\SYSTEM\CurrentControlSet\Control\DeviceGuard" /v "EnableVirtualizationBasedSecurity" /t REG\_DWORD /d 1 /f&#39;   
&#39;REG ADD "HKLM\SYSTEM\CurrentControlSet\Control\DeviceGuard" /v "RequirePlatformSecurityFeatures" /t REG\_DWORD /d 1 /f&#39;

**Driver Verifier code integrity**

Use the Driver Verifier code integrity option flag (0x02000000) to enable extra checks that validate compliance with this feature. To enable this from the command line, use the following command.

verifier.exe /flags 0x02000000 /driver <driver.sys>

To choose this option if using the verifier GUI, choose Create custom settings (for code developers), choose Next, and then choose *Code integrity checks*.

You can use the verifier command line /query option to display the current driver verifier information.

verifier /query

## Check code with BinScope Binary Analyzer

**Security checklist item #14:** *Follow these steps to use BinScope to double check compile and build options are configured to minimize known security issues.*

Use BinScope to examine application binary files to identify coding and building practices that can potentially render the application vulnerable to attack or to being used as an attack vector.

For more information, see this [BinScope Binary Analyzer TechNet Video](https://technet.microsoft.com/video/binscope-binary-analyzer.aspx) and the word documents available as part of the tool download.

Follow these steps to validate that the code you are shipping has includes common security compile options properly configured.

1. Download BinScope Analyzer and related documents from here: <https://www.microsoft.com/download/details.aspx?id=44995>
2. Review the *BinScope Getting Started Guide* that you downloaded.
3. Use the MSI file to install BinScope on the target test machine that contains the compiled code you wish to validate.
4. Open a command prompt window and execute the following command to examine a compiled driver binary. Update the path to point to your complied driver .sys file.

C:\Program Files\Microsoft BinScope 2014>binscope "C:\Samples\KMDF\_Echo\_Driver\echo.sys" /verbose /html /logfile c:\mylog.htm

1. Use a browser to review the BinScope report to confirm that all checks are marked (PASS).

By default, the HTML report is written to \users\<username>\BinScope\

There are three categories that may be output into a log file:

* Failed checks [Fail]
* Checks that didn’t complete [Error]
* Passed checks [Pass]

Note that passed checks are not written to the log by default and must be enabled by using the /verbose switch.

Results for Microsoft BinScope 2014 run on MyPC at 2017-01-28T00:18:48.3828242Z  
  
Failed Checks  
No failed checks.   
Passed Checks  
  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - ATLVersionCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - ATLVulnCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - CompilerVersionCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - DBCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - DefaultGSCookieCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - ExecutableImportsCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - GSCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - GSFriendlyInitCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - GSFunctionSafeBuffersCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - HighEntropyVACheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - NXCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - RSA32Check (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - SafeSEHCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - SharedSectionCheck (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - VB6Check (PASS)  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys - WXCheck (PASS)  
  
Checks Executed:  
• ATLVersionCheck  
• ATLVulnCheck  
• CompilerVersionCheck  
• DBCheck  
• DefaultGSCookieCheck  
• ExecutableImportsCheck  
• GSCheck  
• GSFriendlyInitCheck  
• GSFunctionSafeBuffersCheck  
• HighEntropyVACheck  
• NXCheck  
• RSA32Check  
• SafeSEHCheck  
• SharedSectionCheck  
• VB6Check  
• WXCheck  
  
All Scanned Items  
  
• C:\Samples\KMDF\_Echo\_Driver\echo.sys

## Debugger techniques and extensions

**Security checklist item #15:** *Review these debugger tools and consider their use in your development debugging workflow.*

**!exploitable Crash Analyzer**

The !exploitable Crash Analyzer is a Windows debugger extensions that parses crash logs looking for unique issues. It also examines the type of crash and tries to determine if the error is something that could be exploited by a malicious hacker.

Microsoft Security Engineering Center (MSEC), created the !exploitable crash analyzer. You can download the from codeplex. <http://msecdbg.codeplex.com/>

For more information, see this blog post: [!Exploitable crash analyzer version 1.6](https://blogs.microsoft.com/microsoftsecure/2013/06/13/exploitable-crash-analyzer-version-1-6/) and this Channel 9 video [!exploitable Crash Analyzer](https://channel9.msdn.com/blogs/pdcnews/bang-exploitable-security-analyzer).

**Security related debugger commands**

The !acl extension formats and displays the contents of an access control list (ACL). For more information, see [Determining the ACL of an Object](https://msdn.microsoft.com/library/windows/hardware/ff541868) and [**!acl**](https://msdn.microsoft.com/library/windows/hardware/ff561510).

The !token extension displays a formatted view of a security token object. For more information, see [**!token**](https://msdn.microsoft.com/library/windows/hardware/ff565477).

The !tokenfields extension displays the names and offsets of the fields within the access token object (the TOKEN structure). For more information, see [**!tokenfields**](https://msdn.microsoft.com/library/windows/hardware/ff565478).

The !sid extension displays the security identifier (SID) at the specified address. For more information, see [**!sid**](https://msdn.microsoft.com/library/windows/hardware/ff565344).

The !sd extension displays the security descriptor at the specified address. For more information, see [**!sd**](https://msdn.microsoft.com/library/windows/hardware/ff564930).

## Secure coding resources

**Security checklist item #16:** *Review these resources to expand your understanding of secure coding best practices that are applicable to driver developers.*

*Review these resources to learn more about driver security*

**Secure kernel-mode driver coding guidelines**

[Creating Reliable Kernel-Mode Drivers](https://msdn.microsoft.com/library/windows/hardware/ff542904.aspx)

**Secure coding organizations**

[Carnegie Mellon University SEI CERT](http://www.cert.org/)

Carnegie Mellon University SEI CERT [C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems](https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard) (2016 Edition) available as a [PDF download](http://www.sei.cmu.edu/downloads/sei-cert-c-coding-standard-2016-v01.pdf).

CERT - [Secure Coding Professional Certification](https://www.cert.org/go/secure-coding/)

CERT - [Build Security In](https://www.us-cert.gov/bsi)

MITRE - [Weaknesses Addressed by the CERT C Secure Coding Standard](https://cwe.mitre.org/data/definitions/734.html)

**OSR**

[OSR](http://www.osr.com) provides driver development training and consulting services.

[You've Gotta Use Protection -- Inside Driver & Device Security](http://www.osronline.com/article.cfm?article=100)

[Locking Down Drivers - A Survey of Techniques](http://www.osronline.com/article.cfm?article=357)

[Still Feeling Insecure? - IoCreateDeviceSecure( ) for Windows](http://www.osronline.com/article.cfm?article=105)

[Locking Down Drivers - A Survey of Techniques](http://www.osronline.com/article.cfm?article=357)

**Sample Driver Code**

Review these driver samples to review examples of driver projects that illustrate many of the best practices discussed here. Use these samples to become familiar with the code quality tools.

[ELAM - Early Launch Anti-Malware Driver Code Sample](https://github.com/Microsoft/Windows-driver-samples/tree/master/security/elam)

**Books**

*24 deadly sins of software security : programming flaws and how to fix them* by Michael Howard, David LeBlanc and John Viega

*The art of software security assessment : identifying and preventing software vulnerabilities*, Mark Dowd, John McDonald and Justin Schuh

*Writing Secure Software Second Edition*, Michael Howard and David LeBlanc

*Programming the Microsoft Windows Driver Model (2nd Edition)*, Walter Oney

*Developing Drivers with the Windows Driver Foundation (Developer Reference)*, Penny Orwick and Guy Smith

**Training**

Classroom training is available from [OSR](http://www.osr.com) and [Windows Internals](http://www.windows-internals.com/pages/training-services/windows-internals/).

Online training is available from a variety of sources. For example this course is available from coursera. [https://www.coursera.org/learn/software-security](https://www.coursera.org/learn/software-security#faqs)

**Windows 10 Security Feature Updates**

This video from WinHec provides an overview of Windows 10 security features. [Windows 10 - The Safest and Most Secure Version of Windows](https://channel9.msdn.com/Events/WinHEC/WinHEC-December-2016/Windows-10-The-Safest-and-Most-Secure-Version-of-Windows)

## Key takeaways

Driver security is a complex undertaking containing many elements, but here are a few key points to consider.

* Drivers live in the windows kernel, and having an issue when executing in kernel exposes the entire operating system. Because of this, pay close attention to driver security and design with security as top of mind.
* Apply the principle of least privilege:
  1. Use a strict SDDL string to restrict access to the driver
  2. Further restrict individual IOCTL’s
* Create a threat model to identify attack vectors and consider if anything can be restricted further.
* Be careful around embedded pointers being passed in from usermode, they need to be probed, accessed within try except, and they are prone to time of check time of use (ToCToU) issues unless the value of the buffer is captured and compared.
* When not sure use METHOD\_BUFFERED as an IOCTL buffering method.
* Use code scanning utilities to look for known code vulnerabilities and remediate any identified issues.
* Seek out knowledgeable code review to look for issues, that you may have missed.
* Use driver verifier and test your driver with multiple inputs including corner cases.

[Send comments about this topic to Microsoft](mailto:wsddocfb@microsoft.com?subject=Documentation%20feedback%20[hw_design\hw_design]:%20Driver%20security%20checklist%20%20RELEASE:%20%286/16/2017%29&body=%0A%0APRIVACY%20STATEMENT%0A%0AWe%20use%20your%20feedback%20to%20improve%20the%20documentation.%20We%20don't%20use%20your%20email%20address%20for%20any%20other%20purpose,%20and%20we'll%20remove%20your%20email%20address%20from%20our%20system%20after%20the%20issue%20that%20you're%20reporting%20is%20fixed.%20While%20we're%20working%20to%20fix%20this%20issue,%20we%20might%20send%20you%20an%20email%20message%20to%20ask%20for%20more%20info.%20Later,%20we%20might%20also%20send%20you%20an%20email%20message%20to%20let%20you%20know%20that%20we've%20addressed%20your%20feedback.%0A%0AFor%20more%20info%20about%20Microsoft's%20privacy%20policy,%20see%20http://privacy.microsoft.com/default.aspx.)