Smoke and Mirrors: Driver Signatures are Optional

Gabriel Landau
Elastic Security

whoami

Low-level Windows [reverse] engineer

Help build Elastic Endpoint Security

Detecting malware tradecraft

Attack & defense of EDR

Presented research at:
- Shmoocon
- Black Hat USA
- Black Hat Asia

Blue, formerly red

Chapter 1 - Windows File Sharing

More than you've ever wanted to know about sharing violations.

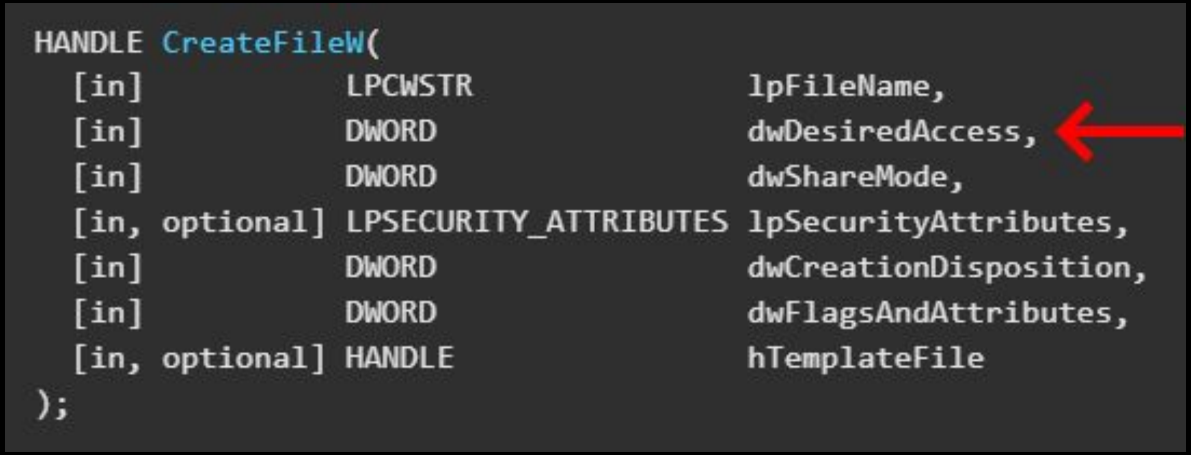
Opening Files - Access Rights

CreateFile - Win32 API to open or create files.

- ntdll analog is **NtCreateFile**.
- Kernel driver analog is **ZwCreateFile**.

Specify desired access rights:

- **FILE_READ_DATA**
- **FILE_WRITE_DATA**
- **DELETE**
- …



```
HANDLE CreateFileW(
  [in]           LPCWSTR               lpFileName,
  [in]           DWORD                 dwDesiredAccess,
  [in]           DWORD                 dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                 dwCreationDisposition,
  [in]           DWORD                 dwFlagsAndAttributes,
  [in, optional] HANDLE                hTemplateFile
);
```

https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew
https://learn.microsoft.com/en-us/windows/win32/fileio/file-security-and-access-rights

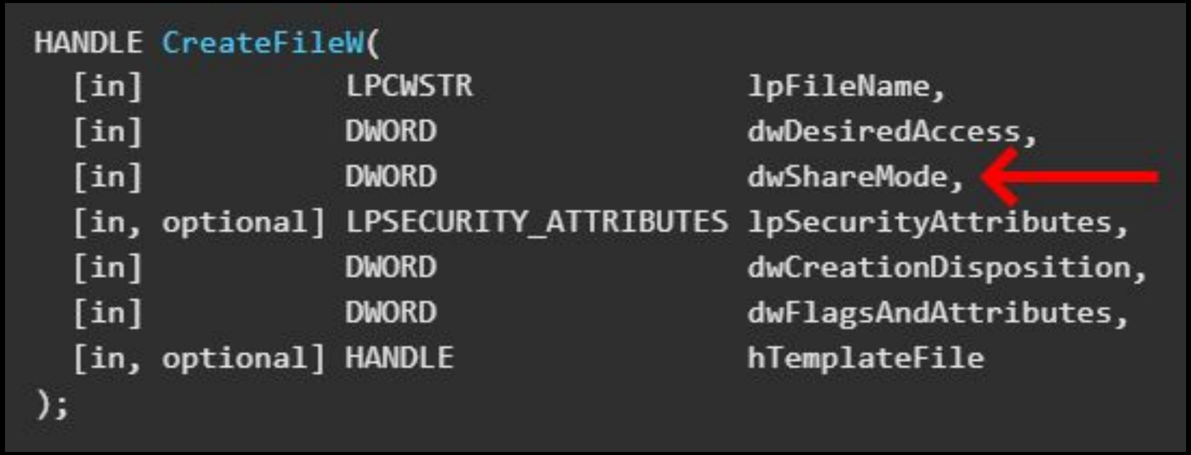
Opening Files - Share Mode

FILE_SHARE_READ / **FILE_SHARE_WRITE** / **FILE_SHARE_DELETE**

“I’m okay with others reading/writing/deleting this file while I’m using it.”

As file is opened:

- **DesiredAccess** is tested against **ShareMode** of all existing file handles
- **ShareMode** is tested against **GrantedAccess** of all existing file handles



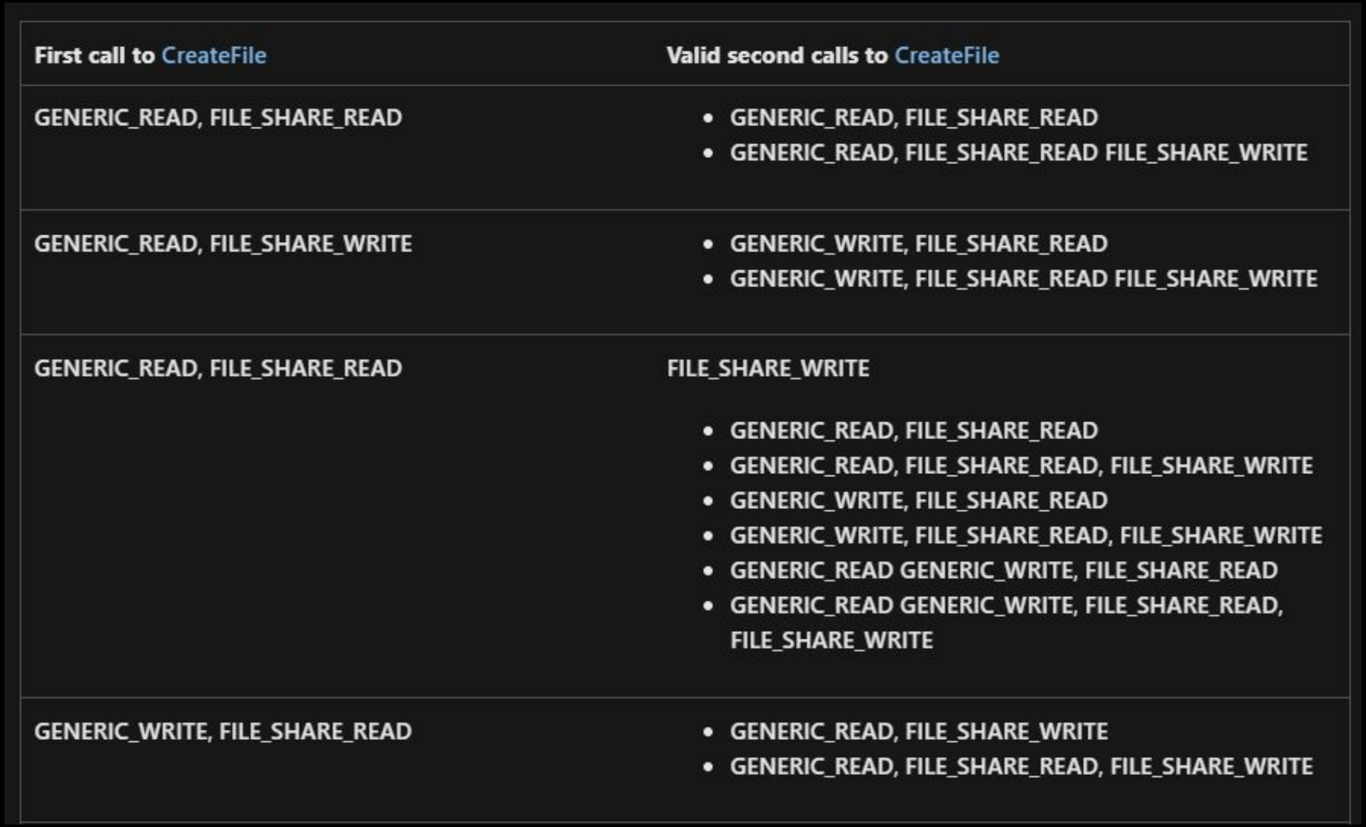
```
HANDLE CreateFileW(
  [in]           LPCWSTR               lpFileName,
  [in]           DWORD                 dwDesiredAccess,
  [in]           DWORD                 dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                 dwCreationDisposition,
  [in]           DWORD                 dwFlagsAndAttributes,
  [in, optional] HANDLE                hTemplateFile
);
```

https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew
https://learn.microsoft.com/en-us/windows/win32/fileio/creating-and-opening-files

Opening Files - Sharing Violation

DesiredAccess/**ShareMode** incompatibilities fail the **CreateFile** call.

- **ERROR_SHARING_VIOLATION** / **STATUS_SHARING_VIOLATION**

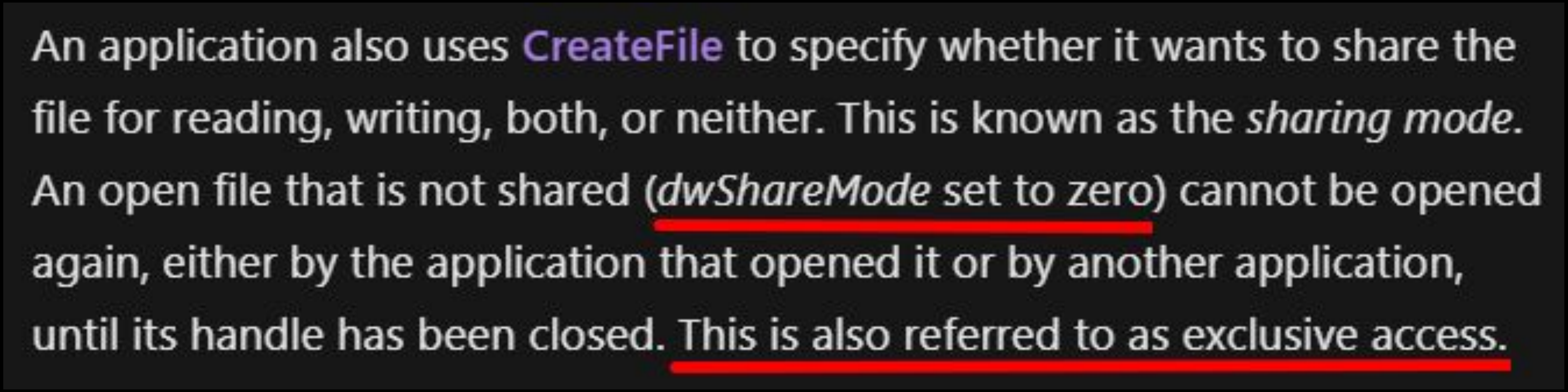


First call to CreateFile	Valid second calls to CreateFile
GENERIC_READ, FILE_SHARE_READ	• GENERIC_READ, FILE_SHARE_READ • GENERIC_READ, FILE_SHARE_READ FILE_SHARE_WRITE
GENERIC_READ, FILE_SHARE_WRITE	• GENERIC_WRITE, FILE_SHARE_READ • GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE
GENERIC_READ, FILE_SHARE_READ	FILE_SHARE_WRITE • GENERIC_READ, FILE_SHARE_READ • GENERIC_READ, FILE_SHARE_READ, FILE_SHARE_WRITE • GENERIC_WRITE, FILE_SHARE_READ • GENERIC_WRITE, FILE_SHARE_READ, FILE_SHARE_WRITE • GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ • GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ, FILE_SHARE_WRITE
GENERIC_WRITE, FILE_SHARE_READ	• GENERIC_READ, FILE_SHARE_WRITE • GENERIC_READ, FILE_SHARE_READ, FILE_SHARE_WRITE

https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew
https://learn.microsoft.com/en-us/windows/win32/fileio/creating-and-opening-files

Opening Files - Exclusive Access

Set **ShareMode**=0 for exclusive access to files until you close the handle.



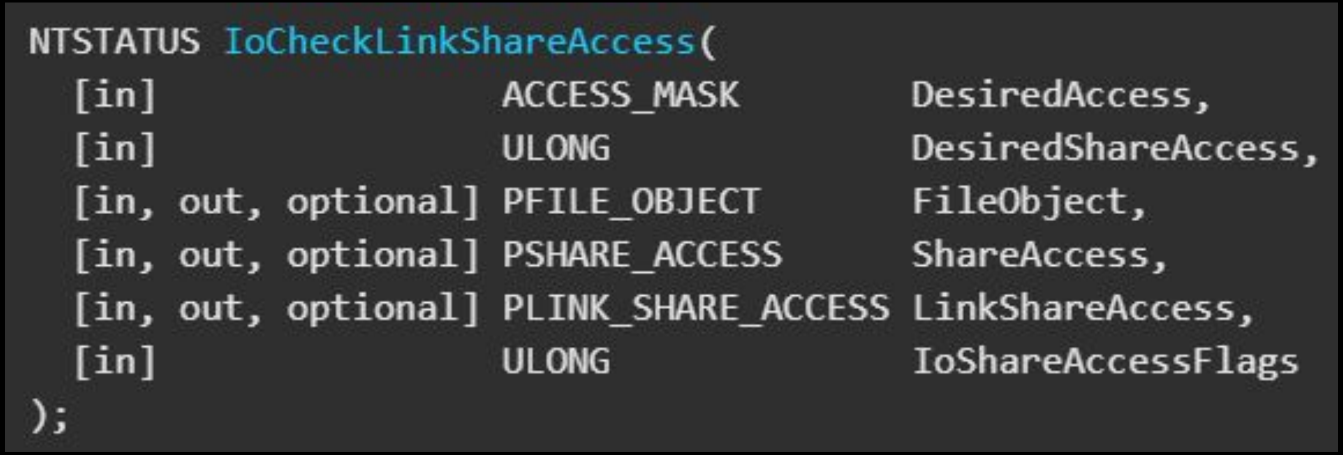
An application also uses CreateFile to specify whether it wants to share the file for reading, writing, both, or neither. This is known as the *sharing mode*. An open file that is not shared (*dwShareMode* set to zero) cannot be opened again, either by the application that opened it or by another application, until its handle has been closed. This is also referred to as exclusive access.

https://learn.microsoft.com/en-us/windows/win32/fileio/creating-and-opening-files

Sharing Enforcement - I/O Manager

Filesystems call **IoCheckLinkShareAccess** to see whether **DesiredAccess**/**ShareMode** is compatible with existing handles.

```
NTSTATUS NtfsCheckShareAccess(FileObject, DesiredAccess, ShareAccess)
{
    ntStatus = IoCheckLinkShareAccess(
        FileObject, DesiredAccess, ShareAccess);
    if (!NT_SUCCESS(ntStatus))
    {
        return ntStatus;
    }
    ...
}
```

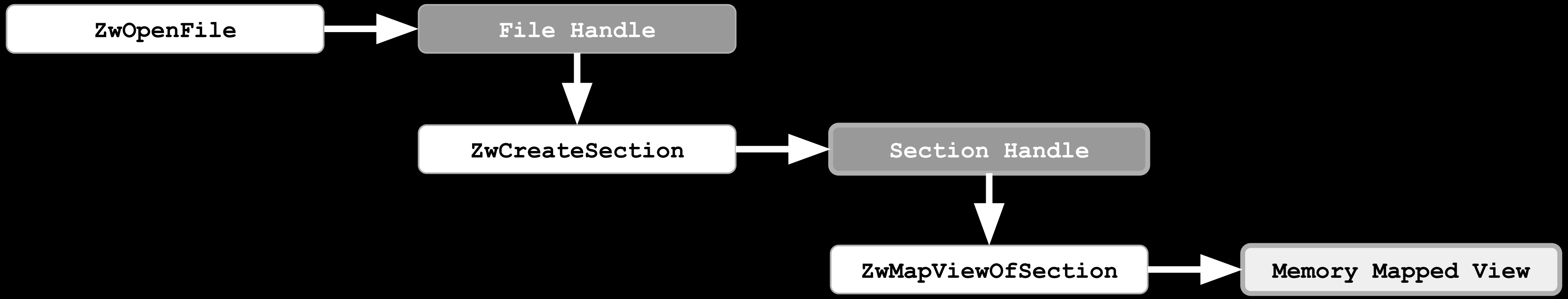


```
NTSTATUS IoCheckLinkShareAccess(
  [in]                     ACCESS_MASK        DesiredAccess,
  [in]                     ULONG              DesiredShareAccess,
  [in, out, optional] PFILE_OBJECT       FileObject,
  [in, out, optional] PSHARE_ACCESS      ShareAccess,
  [in, out, optional] PLINK_SHARE_ACCESS LinkShareAccess,
  [in]                     ULONG              IoShareAccessFlags
);
```

https://github.com/Microsoft/Windows-driver-samples/blob/622212c3fff587f23f6490a9da939fb85968f651/filesys/fastfat/create.c#L6822-L6884

Sharing Enforcement - File Mapping

File mappings (section objects) allow files to be readable/writable after handles are closed.



```
NTSTATUS NtfsOpenAttributeCheck(...)
{
    if (!FlagOn(ShareMode, FILE_SHARE_WRITE) &&
        MmDoesFileHaveUserWritableReferences(FileObject->SectionObjectPointer))
    {
        return STATUS_SHARING_VIOLATION;
    }
    ...
}
```

https://github.com/Microsoft/Windows-driver-samples/blob/622212c3fff587f23f6490a9da939fb85968f651/filesys/fastfat/create.c#L6858-L6870

Sharing Enforcement - Executables

Files mapped as executable images (EXEs/DLLs/etc) **must be immutable** while in use.

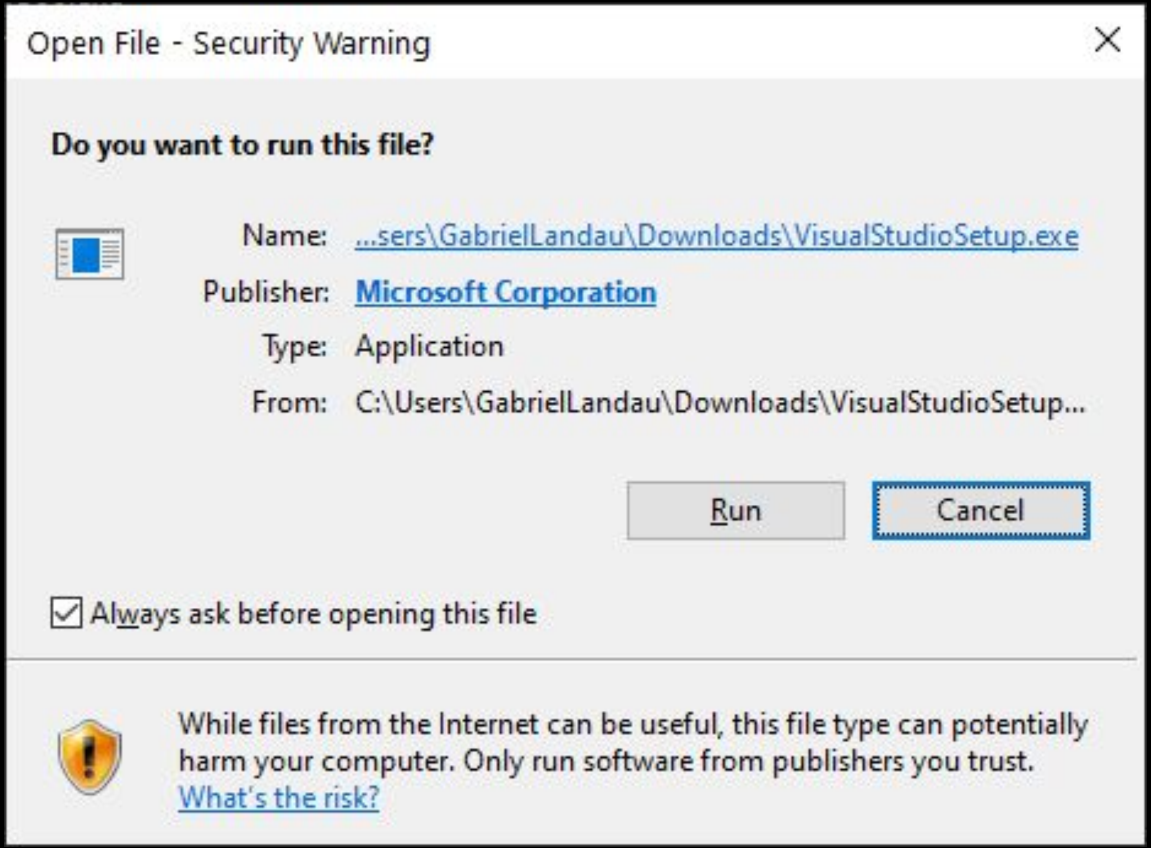
In other words, **ZwMapViewOfSection(SEC_IMAGE)** implies no-write-sharing.

```
NTSTATUS NtfsOpenAttributeCheck(...)
{
    // Block writes to active image section objects
    if (FlagOn(DesiredAccess, FILE_WRITE_DATA) &&
        FileObject->SectionObjectPointer.ImageSectionObject &&
        !MmFlushImageSection(FileObject->SectionObjectPointer), MmFlushForWrite)
        {
            return STATUS_SHARING_VIOLATION
        }
    }
    ...
}
```

https://github.com/Microsoft/Windows-driver-samples/blob/622212c3fff587f23f6490a9da939fb85968f651/filesys/fastfat/create.c#L3572-L3593

Chapter 2 - Code Integrity

How do you trust the code that's running on your system?



Open File - Security Warning

Do you want to run this file?

Name: ...sers\GabrielLandau\Downloads\VisualStudioSetup.exe

Publisher: **Microsoft Corporation**

Type: Application

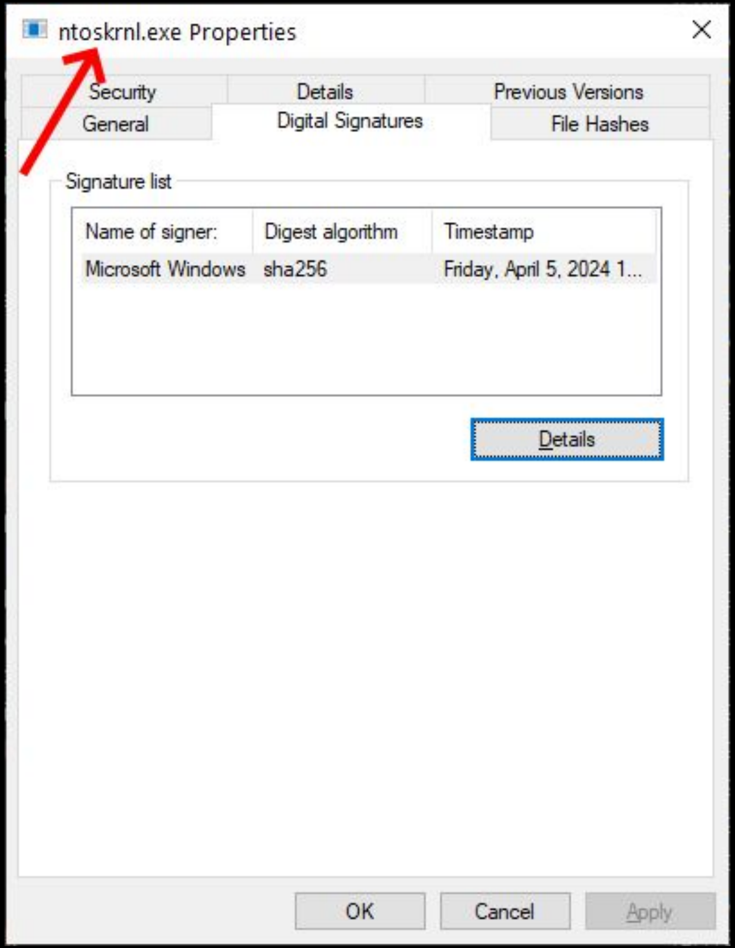
From: C:\Users\GabrielLandau\Downloads\VisualStudioSetup...

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Authenticode

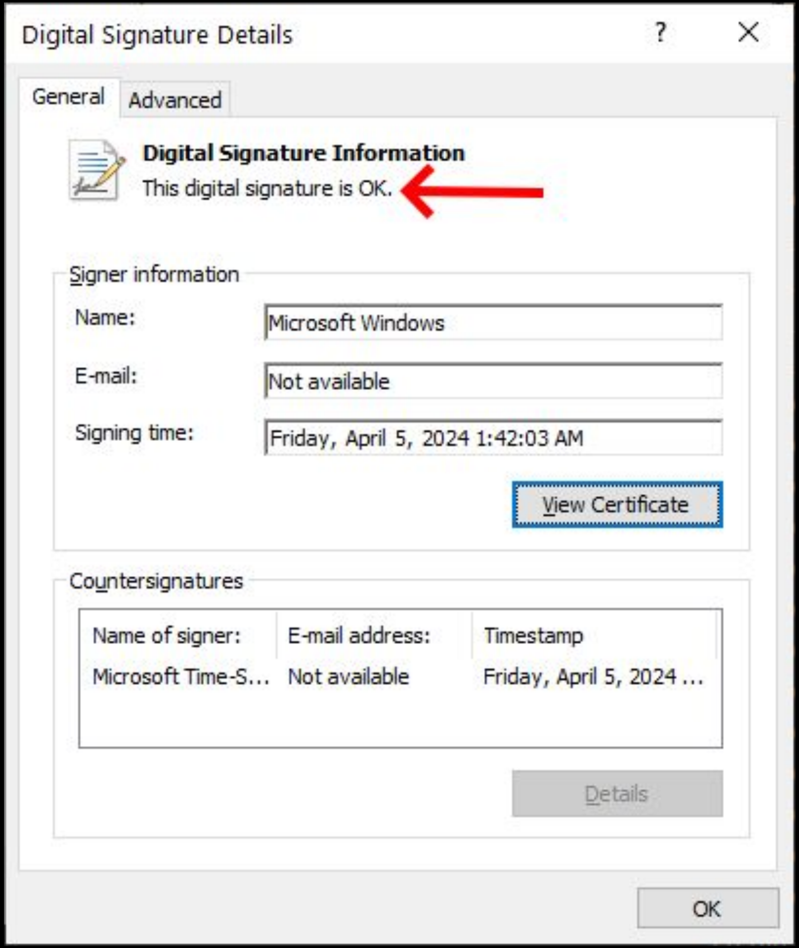
Microsoft specification to digitally sign Portable Executable (PE) files.



ntoskrnl.exe Properties

Signature list

Name of signer:	Digest algorithm	Timestamp
Microsoft Windows	sha256	Friday, April 5, 2024 1...



Digital Signature Details

Digital Signature Information

This digital signature is OK.

Signer information

Name: Microsoft Windows

E-mail: Not available

Signing time: Friday, April 5, 2024 1:42:03 AM

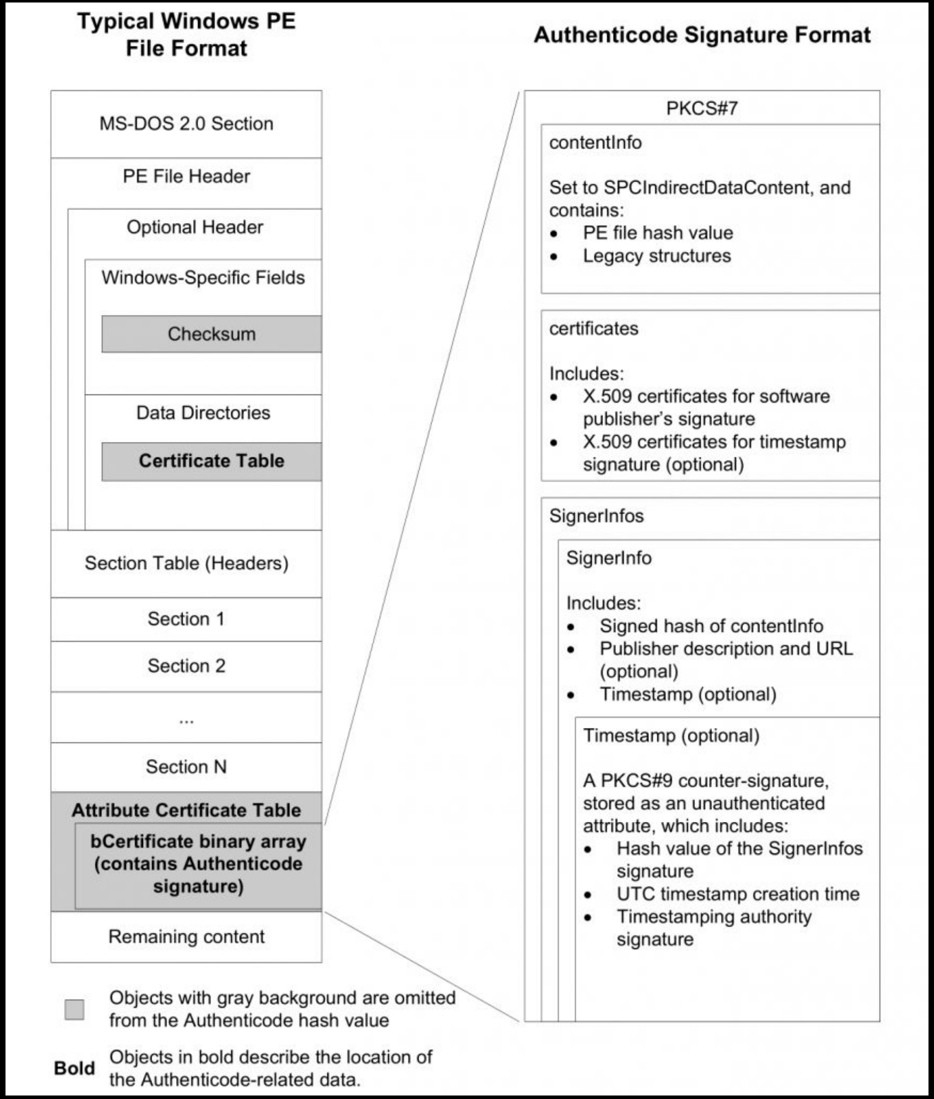
Countersignatures

Name of signer:	E-mail address:	Timestamp
Microsoft Time-S...	Not available	Friday, April 5, 2024 ...

Authenticode Signing

Authentihash algorithm computes hash over most (but not all) of the PE file.

Authentihash is signed using PKCS #7 and appended to PE as Security Directory (aka Certificate Table).



Typical Windows PE File Format

MS-DOS 2.0 Section

PE File Header

Optional Header

Windows-Specific Fields

Checksum

Data Directories

Certificate Table

Section Table (Headers)

Section 1

Section 2

...

Section N

Attribute Certificate Table

bCertificate binary array (contains Authenticode signature)

Remaining content

Authenticode Signature Format

PKCS#7

contentInfo

Set to SPCIndirectDataContent, and contains:

- PE file hash value
- Legacy structures

certificates

Includes:

- X.509 certificates for software publisher's signature
- X.509 certificates for timestamp signature (optional)

SignerInfos

SignerInfo

Includes:

- Signed hash of contentInfo
- Publisher description and URL (optional)
- Timestamp (optional)

Timestamp (optional)

A PKCS#9 counter-signature, stored as an unauthenticated attribute, which includes:

- Hash value of the SignerInfos signature
- UTC timestamp creation time
- Timestamping authority signature

▨ Objects with gray background are omitted from the Authenticode hash value

Bold Objects in bold describe the location of the Authenticode-related data.

https://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/authenticode_pe.docx

Authenticode Implementations

User and kernel implementations to validate signatures.

The user implementation is out of scope for this talk.

The kernel implementation is the **Code Integrity** (CI) subsystem.

CI.dll protected from tampering by Secure Boot and Trusted Boot systems.

Code Integrity

Kernel Mode Code Integrity (KMCI)

- Validates signatures on drivers before allowing them to load.
- Enforces Driver Signing Enforcement and Vulnerable Driver Blocklist.

User Mode Code Integrity (UMCI)

- CI validates the signatures of EXEs and DLLs before allowing them to load.
- Enforces Protected Processes and Protected Process Light signature requirements.
- Enforces Microsoft Signer process mitigation (**SetProcessMitigationPolicy**).
- Enforces **/INTEGRITYCHECK** for FIPS 140-2 modules.
- Exposed to consumers as **Smart App Control**.
- Exposed to businesses as **App Control for Business** (formerly WDAC).

KMCI and UMCI implement different policies for different scenarios.

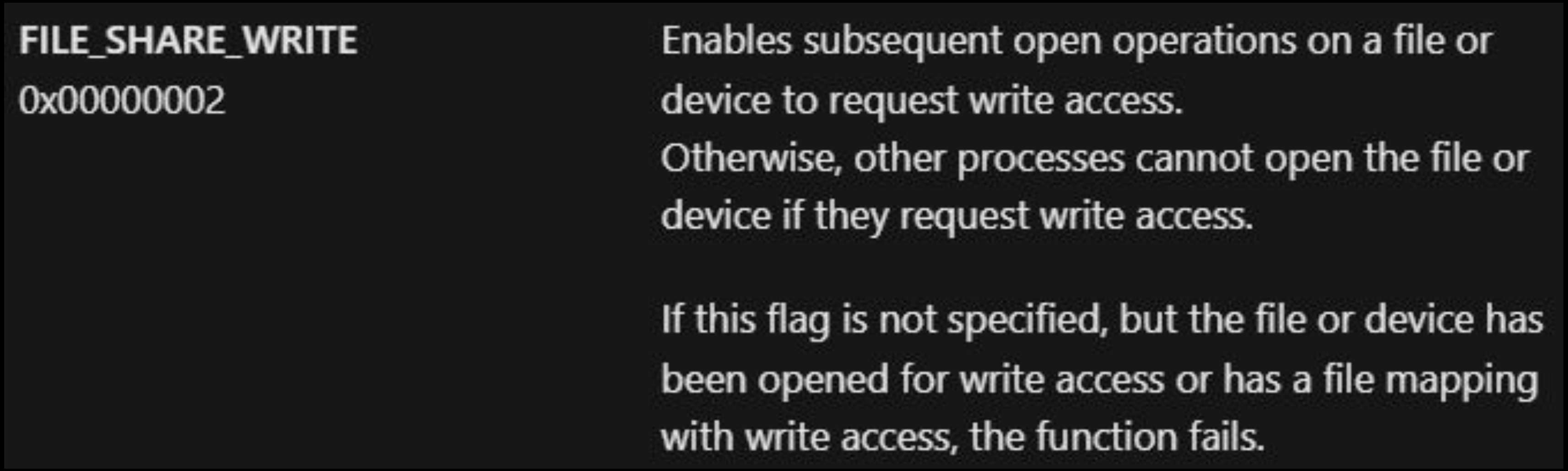
https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/design/select-types-of-rules-to-create
https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-setprocessmitigationpolicy
https://x.com/GabrielLandau/status/1668353640833114131
https://learn.microsoft.com/en-us/windows/apps/develop/smart-app-control/overview
https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/wdac
https://learn.microsoft.com/en-us/windows/security/application-security/application-control/windows-defender-application-control/design/microsoft-recommended-driver-block-rules

Chapter 3 - Incorrect Assumptions

Let’s discuss a class of vulnerabilities resulting from incorrect assumptions.

Incorrect Assumptions

Microsoft docs imply that files successfully opened without write sharing can’t be modified under you.



FILE_SHARE_WRITE 0x00000002	Enables subsequent open operations on a file or device to request write access. Otherwise, other processes cannot open the file or device if they request write access. If this flag is not specified, but the file or device has been opened for write access or has a file mapping with write access, the function fails.

What if the filesystem doesn’t know that the file’s been modified?

Executable Image Section Paging

Executable image sections originate from PE files.

MM can page these out if memory is needed:

- Never modified? Discard it. We already have a copy in the original PE.
- Modified? Save it to the pagefile.
  - Example: ntdll was detoured. MM copy-on-write created private copy.

Upon page fault:

- Never modified*? Read the page from the original PE file.
- Modified? Grab the private copy from the pagefile.

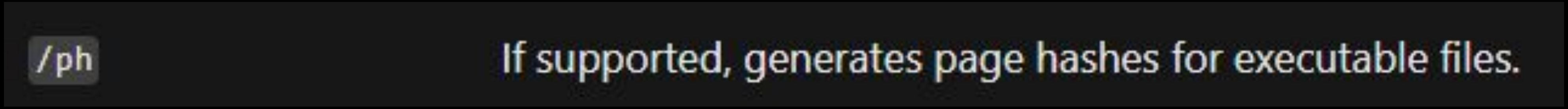
* Exception: The memory manager may treat PE-relocated pages as unmodified, dynamically reapplying relocations during page faults.

Page Hashes

Optional list of hashes of each 4KB page of PE. Allows MM to validate hashes of individual pages during page faults.

Static page hashes

- Stored within signature when file is signed.
- **signtool.exe /ph**



/ph If supported, generates page hashes for executable files.

Dynamic page hashes

- Computed on the fly by CI when **SEC_IMAGE** is created and validated.
- Enables page hash enforcement even if signature does not include them.

Page hashes are not free - they use CPU and slow down page faults.

https://learn.microsoft.com/en-us/windows/win32/seccrypto/signtool

Attacking Code Integrity

Scenario:

1. Orphanage administrator enables macros in email attachment containing ransomware.
2. Ransomware employs UAC bypass to instantly elevate to Admin.
3. Ransomware fails to terminate AV running as Protected Process Light (PPL).
4. Ransomware author wants PPL rights so it can kill AV and ransom orphanage.

Can it launch itself directly as PPL?

❌ UMCI prevents improperly-signed EXEs and DLLs from loading into PPL.

CreateFile(FILE_WRITE_DATA) to inject code into already-in-use DLL?

❌ NTFS checks prevent **CreateFile(FILE_WRITE_DATA)** to in-use image sections.

  - Aforementioned **MmFlushImageSection** check.

FILE_WRITE_DATA check is in NTFS. What if we move the filesystem to another machine?

- SMB server could be a Samba server, or even a python script.

Attacker can modify a DLL server-side, bypassing sharing restrictions.

- DLLs are incorrectly assumed to be immutable.
- **False File Immutability**

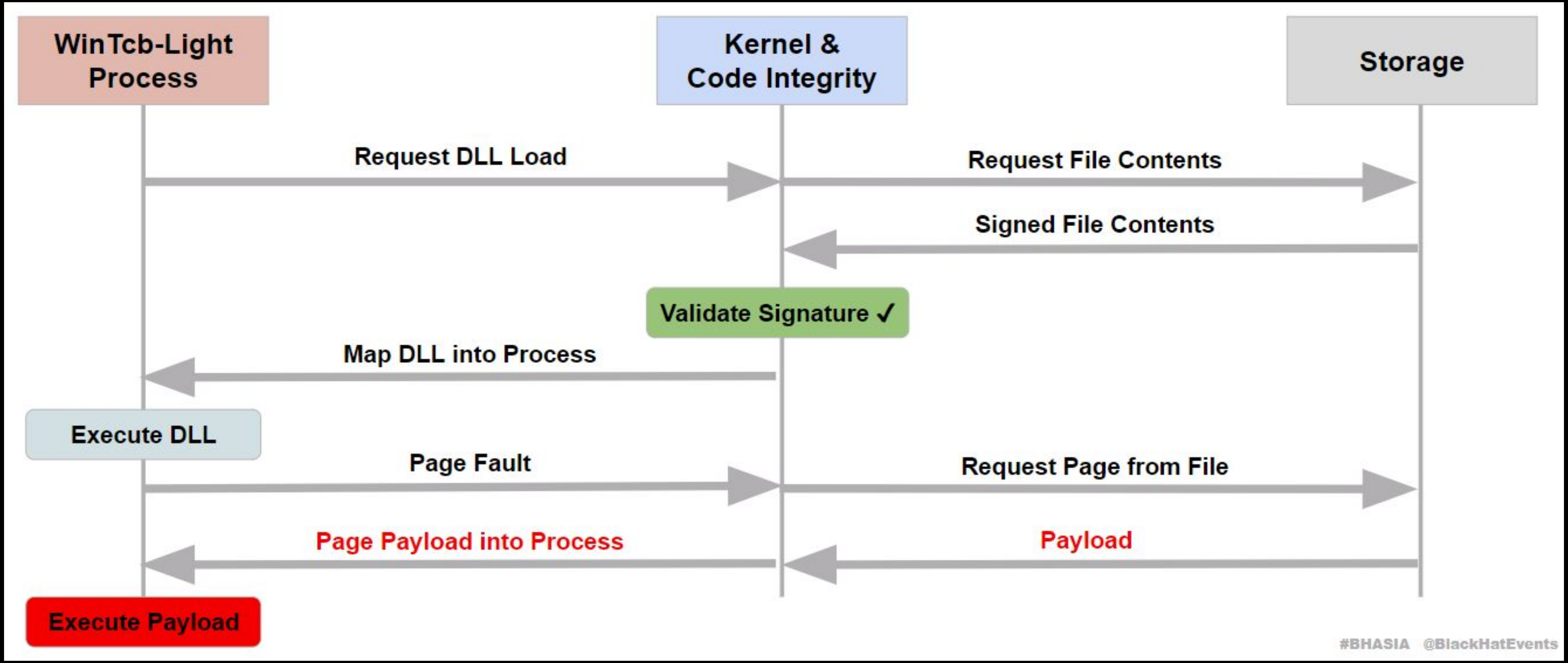
Can Attacker Exploit Paging?

Even if attacker successfully exploits **false file immutability** to inject code into a PE, won’t page hashes catch this attack?

	Authenticode	Page Hashes
Kernel Drivers	✅	✅
Protected Processes	✅	✅
Protected Process Light (PPL)	✅	❌

Admin->PPL Exploit: PPLFault

Disclosed by me at Black Hat Asia 2023.

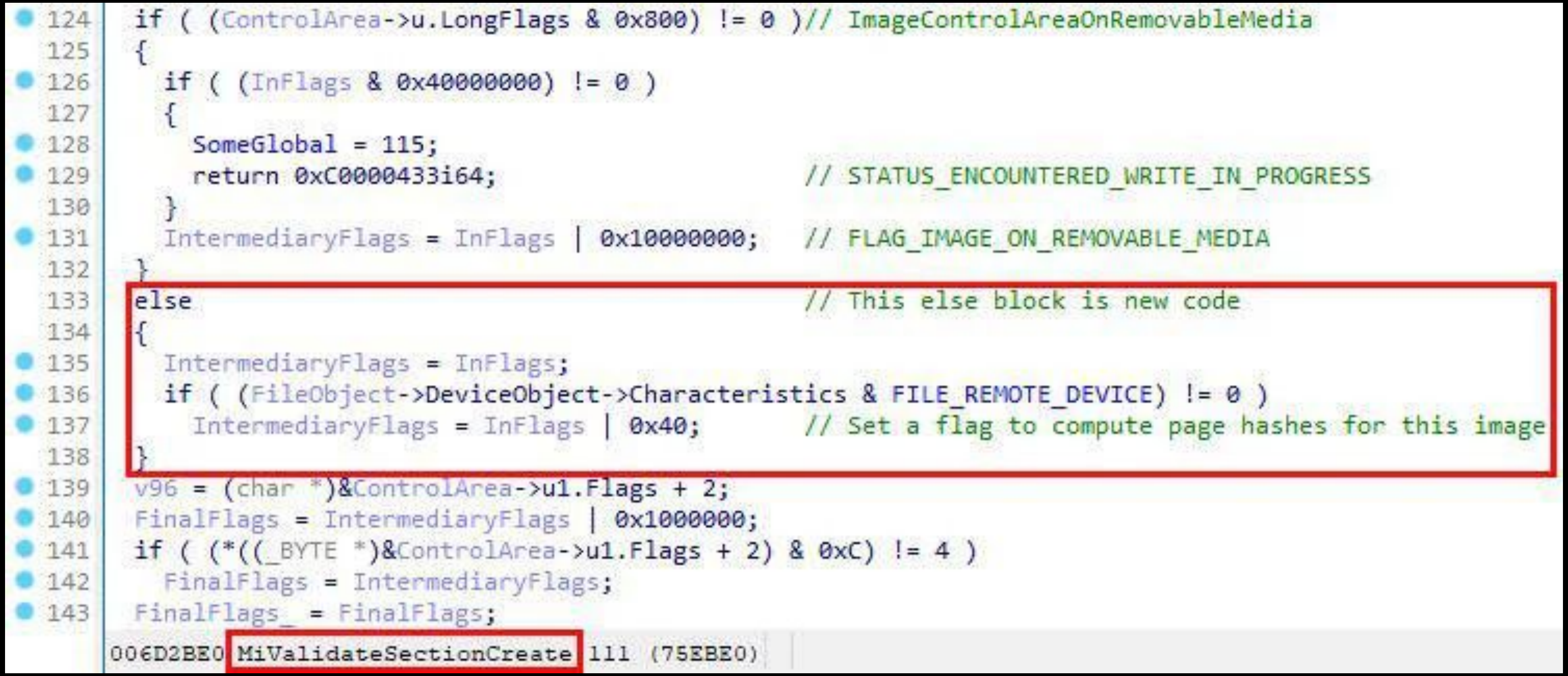


https://github.com/gabriellandau/PPLFault
https://www.youtube.com/watch?v=5xteW8Tm410
https://i.blackhat.com/Asia-23/AS-23-Landau-PPLdump-Is-Dead-Long-Live-PPLdump.pdf

Mitigating PPLFault

In February 2024, Microsoft added a check to mitigate PPLFault.

MM sets a flag requiring dynamic page hashes for images that originate from remote devices such as network redirectors like SMB.



```
if ( (ControlArea->u.LongFlags & 0x800) != 0 )// ImageControlAreaOnRemovableMedia
{
  if ( (InFlags & 0x40000000) != 0 )
  {
    SomeGlobal = 115;
    return 0xC0000433i64;                      // STATUS_ENCOUNTERED_WRITE_IN_PROGRESS
  }
  IntermediaryFlags = InFlags | 0x10000000;   // FLAG_IMAGE_ON_REMOVABLE_MEDIA
}
else                                           // This else block is new code
{
  IntermediaryFlags = InFlags;
  if ( (FileObject->DeviceObject->Characteristics & FILE_REMOTE_DEVICE) != 0 )
    IntermediaryFlags = InFlags | 0x40;       // Set a flag to compute page hashes for this image
}
v96 = (char *)&ControlArea->u1.Flags + 2;
FinalFlags = IntermediaryFlags | 0x1000000;
if ( (*((_BYTE *)&ControlArea->u1.Flags + 2) & 0xC) != 4 )
  FinalFlags = IntermediaryFlags;
FinalFlags_ = FinalFlags;
```

```
006D2BE0 MiValidateSectionCreate 111 (75EBE0)
```

https://www.elastic.co/security-labs/inside-microsofts-plan-to-kill-pplfault

PPLFault - Takeaways

What did we learn?

PPLFault successfully exploited bad assumptions in CI about DLL immutability, achieving unsigned WinTcb-Light PPL code execution. For reasons out-of-scope, it was easy to chain this to full physical memory read/write,**compromising the entire OS in a few seconds.**

The mitigation was narrow in scope - targeting images loaded from remote devices.

Chapter 4 – New Research

Can we exploit false file immutability in other ways?

Let’s look beyond executable image sections.

What about attacks against data files?

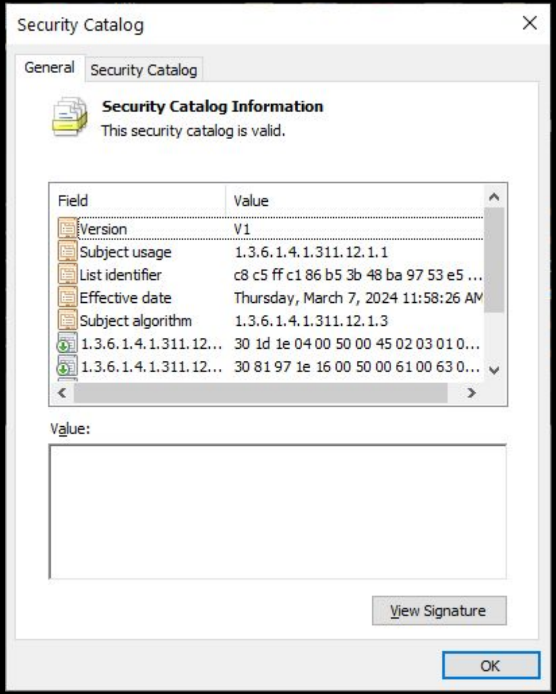
Authenticode – Security Catalogs

Security catalogs - detached Authenticode signatures.

Signed array of Authentihashes in .cat files in **C:\Windows\System32\CatRoot**

Every PE with Authentihash in list is considered to be signed by that signer.

Hash	**Hash**	**Hash**	**Hash**	**Hash**	**Hash**	**Hash**	**Hash**	**Hash**	**Hash**	**Hash**	**Hash**	**Hash**	**Signature**



Security Catalog

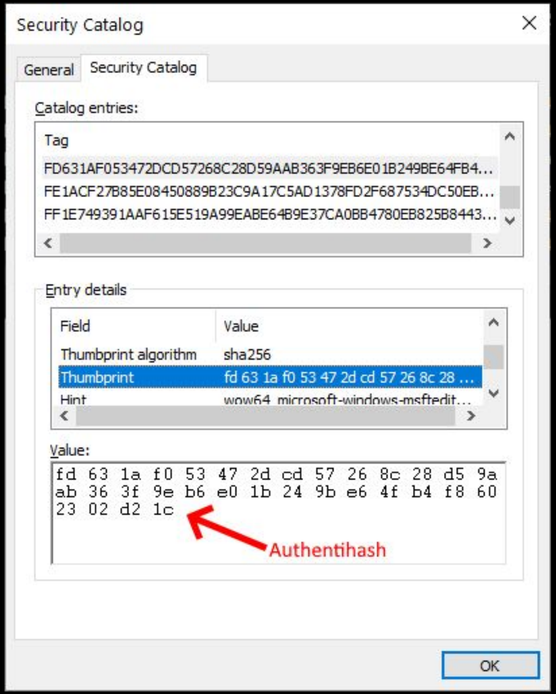
General | Security Catalog

Security Catalog Information

This security catalog is valid.

Field	Value
Version	V1
Subject usage	1.3.6.1.4.1.311.12.1.1
List identifier	c8 c5 ff c1 86 b5 3b 48 ba 97 53 e5 ...
Effective date	Thursday, March 7, 2024 11:58:26 AM
Subject algorithm	1.3.6.1.4.1.311.12.1.3
1.3.6.1.4.1.311.12...	30 1d 1e 04 00 50 00 45 02 03 01 0...
1.3.6.1.4.1.311.12...	30 81 97 1e 16 00 50 00 61 00 63 0...

Value:



Security Catalog

General | Security Catalog

Catalog entries:

Tag
FD631AF053472DCD57268C28D59AAB363F9EB6E01B249BE64FB4...
FE1ACF27B85E08450889B23C9A17C5AD1378FD2F687534DC50EB...
FF1E749391AAF615E519A99EABE64B9E37CA0BB4780EB825B8443...

Entry details

Field	Value
Thumbprint algorithm	sha256
Thumbprint	fd 63 1a f0 53 47 2d cd 57 26 8c 28 ...
Hint	wow64_microsoft-windows-msftedit...

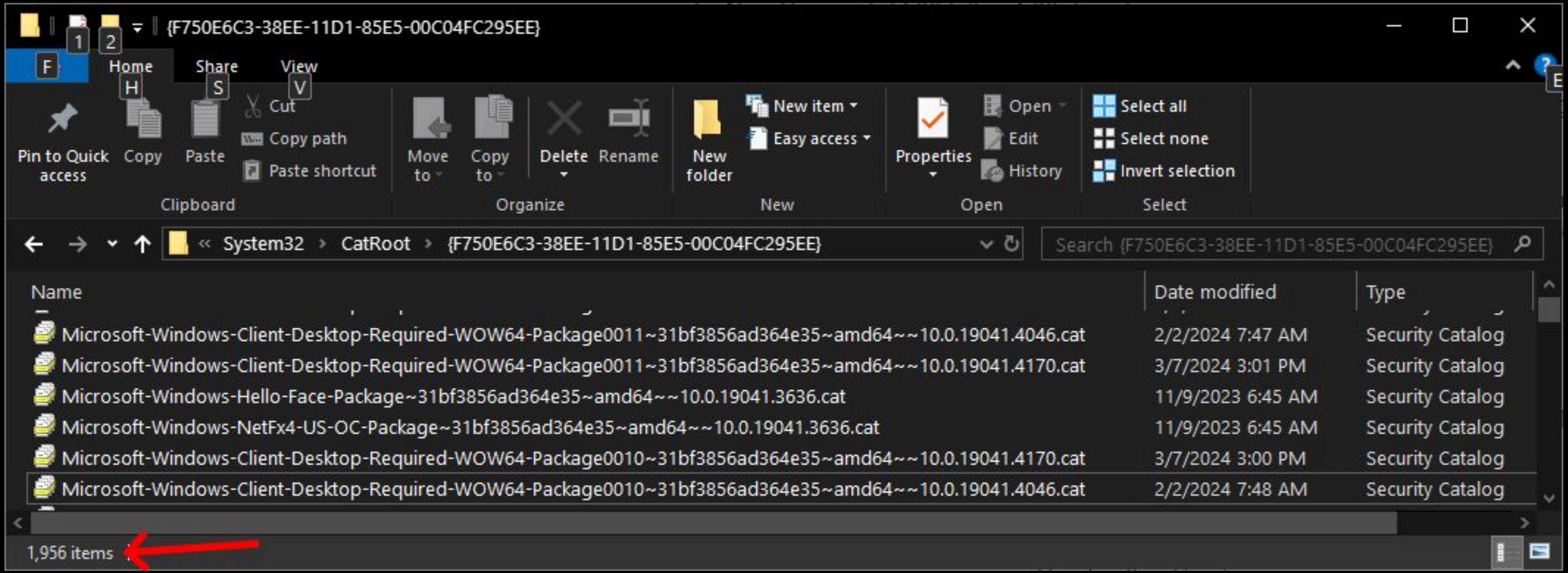
Value:

```
fd 63 1a f0 53 47 2d cd 57 26 8c 28 d5 9a
ab 36 3f 9e b6 e0 1b 24 9b e6 4f b4 f8 60
23 02 d2 1c
```

Authentihash

Authenticode - Security Catalogs

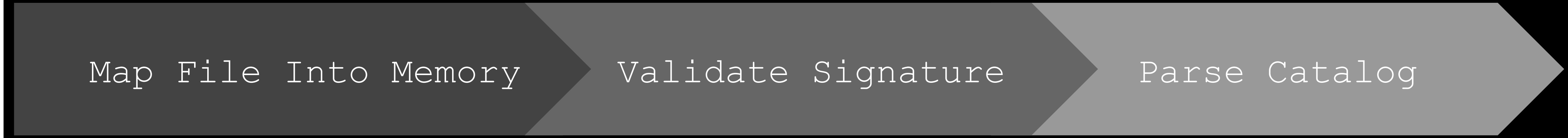
Large list of catalogs. CI loads them into kernel pool for fast lookup.



Name	Date modified	Type
Microsoft-Windows-Client-Desktop-Required-WOW64-Package0011~31bf3856ad364e35~amd64~~10.0.19041.4046.cat	2/2/2024 7:47 AM	Security Catalog
Microsoft-Windows-Client-Desktop-Required-WOW64-Package0011~31bf3856ad364e35~amd64~~10.0.19041.4170.cat	3/7/2024 3:01 PM	Security Catalog
Microsoft-Windows-Hello-Face-Package~31bf3856ad364e35~amd64~~10.0.19041.3636.cat	11/9/2023 6:45 AM	Security Catalog
Microsoft-Windows-NetFx4-US-OC-Package~31bf3856ad364e35~amd64~~10.0.19041.3636.cat	11/9/2023 6:45 AM	Security Catalog
Microsoft-Windows-Client-Desktop-Required-WOW64-Package0010~31bf3856ad364e35~amd64~~10.0.19041.4170.cat	3/7/2024 3:00 PM	Security Catalog
Microsoft-Windows-Client-Desktop-Required-WOW64-Package0010~31bf3856ad364e35~amd64~~10.0.19041.4046.cat	2/2/2024 7:48 AM	Security Catalog

1,956 items

Code Integrity - Catalog Parsing



```
nt!ZwOpenFile(
  GENERIC_READ,
  FILE_SHARE_READ)

nt!ZwCreateSection(
  SEC_COMMIT)

nt!ZwMapViewOfSection
```

```
CI!MinCrypK_
 VerifySignedDataKModeEx
```

```
CI!I_MapFileHashes
```

Catalog Parsing - Key Insights

ZwOpenFile(GENERIC_READ, FILE_SHARE_READ)

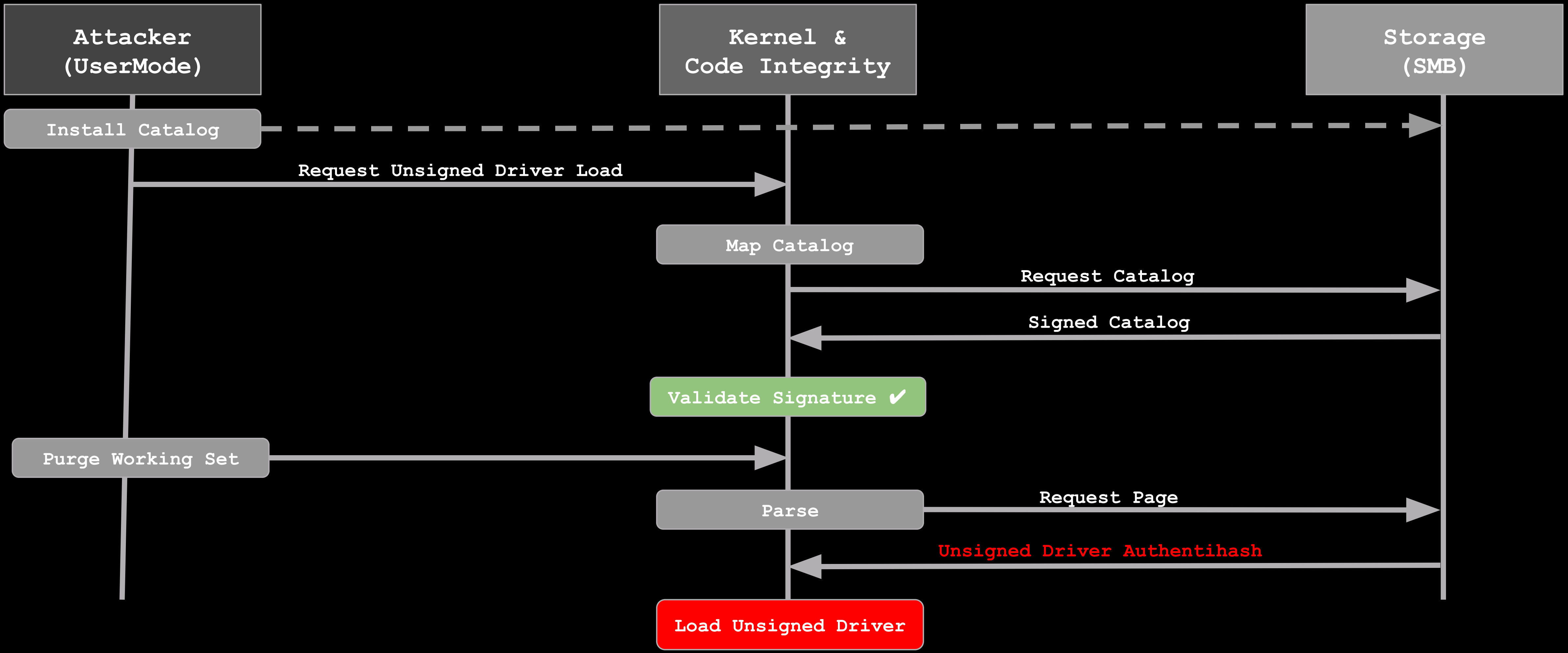
- Denies write sharing to prevent catalog modifications during processing.
- Bad assumption - **false file immutability**.

ZwCreateSection(SEC_COMMIT)

- Creates a data section.
- Not an image section - no page hashes.

Can we perform a PPLFault-style attack on security catalogs?

Exploiting Security Catalogs



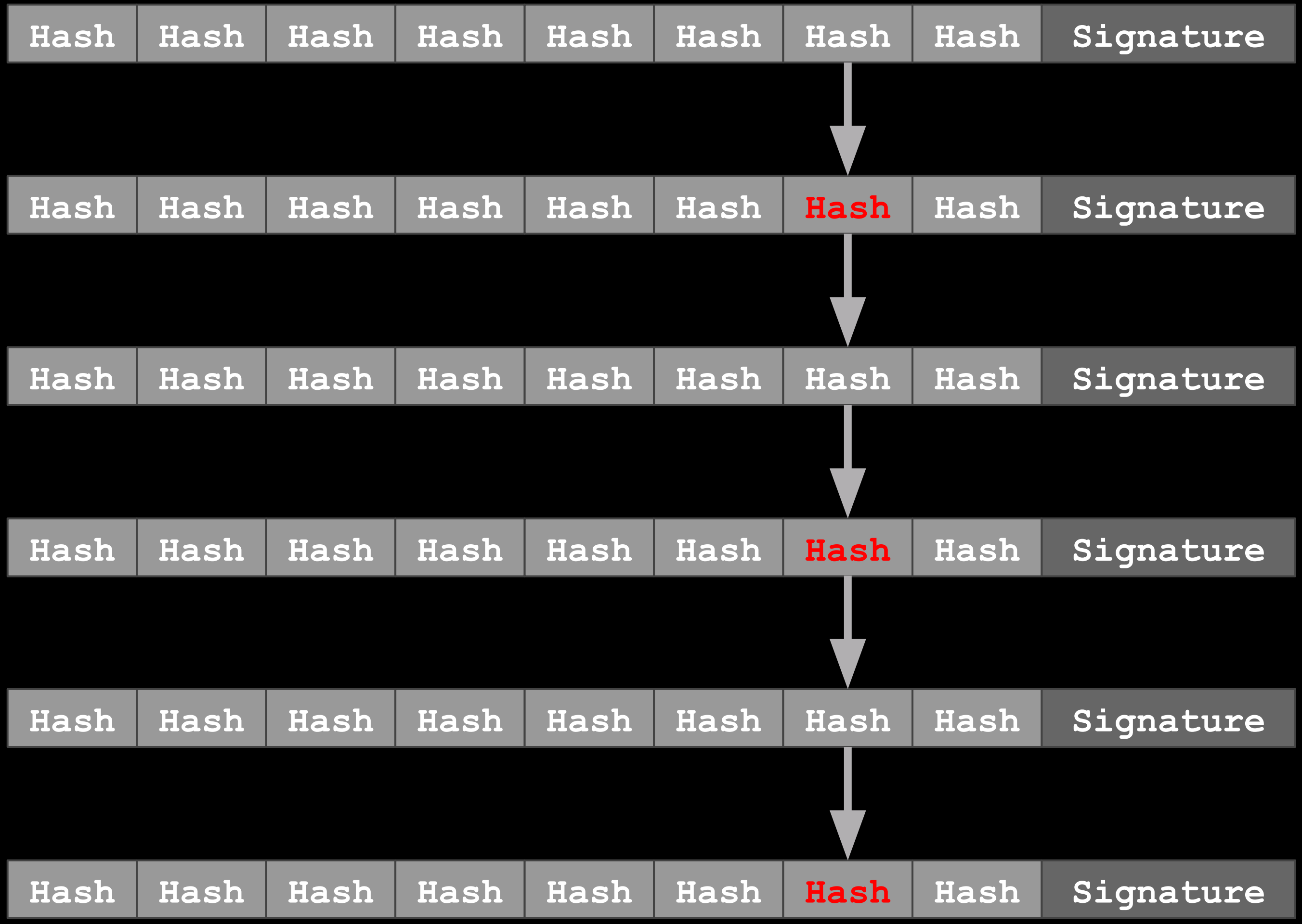
Exploit - Toggling the Catalog

PPLFault used an oplock to deterministically pause the victim process then switch to the payload DLL contents.

No good opportunities here for oplocks.

Rapidly toggle the catalog between benign and malicious - probabilistic approach.

Choose hash near end of catalog because parsing is [probably] linear.



Hash Hash Hash Hash Hash Hash Hash Hash Signature

Hash Hash Hash Hash Hash Hash Hash Hash Signature

Hash Hash Hash Hash Hash Hash Hash Hash Signature

Hash Hash Hash Hash Hash Hash Hash Hash Signature

Hash Hash Hash Hash Hash Hash Hash Hash Signature

Hash Hash Hash Hash Hash Hash Hash Hash Signature

Exploit - Race Condition

Attacker needs CI to trigger a page fault between validation and parsing, but the page is already resident from recent validation. Without a page fault, CI will use the same pages for validation and parsing.

To evict page from kernel memory, attacker must empty working set between **MinCrypK_VerifySignedDataKModeEx** and **I_MapFileHashes**.

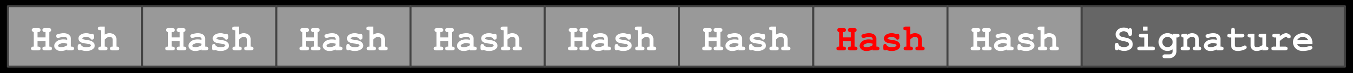
Very short race window. Employ multiple approaches to slow CI and improve chances of winning race:

- Choose large security catalog (4MB).
- Dedicated thread emptying working set.
- Dedicated thread repeatedly loading unsigned driver.
- High-priority dummy threads spinning CPU cores to starve system worker threads.

Fail - Signature Check Failed

If the payload Authentihash is read during the signature check, the catalog will be rejected.

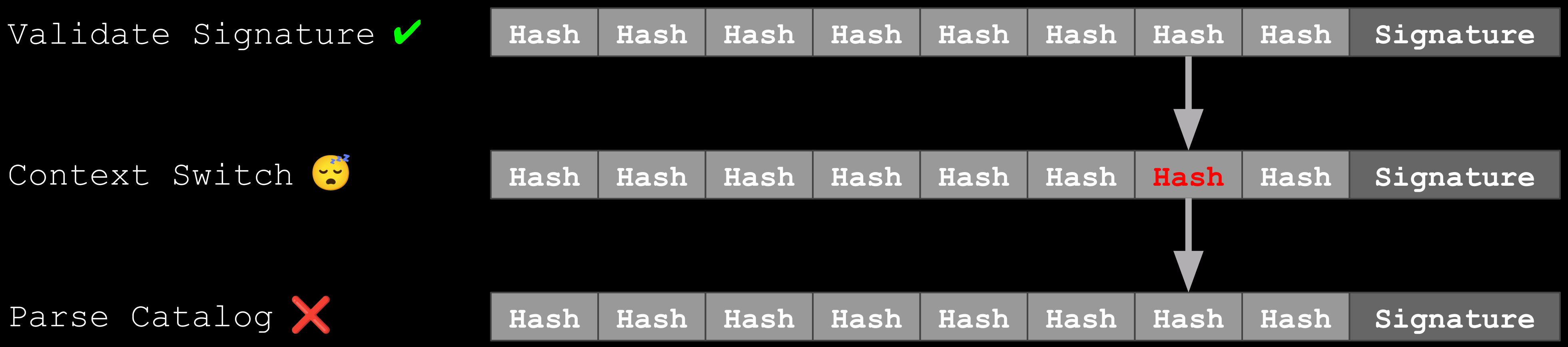
Validate Signature ❌



Hash Hash Hash Hash Hash Hash Hash Hash Signature

Fail - Benign Catalog Parsed

An even number of swaps (including zero) between signature validation and parsing means CI will parse the benign hash and reject our driver.



Validate Signature ✔

Hash	Hash	Hash	Hash	Hash	Hash	Hash	Hash	Signature

Context Switch 😴

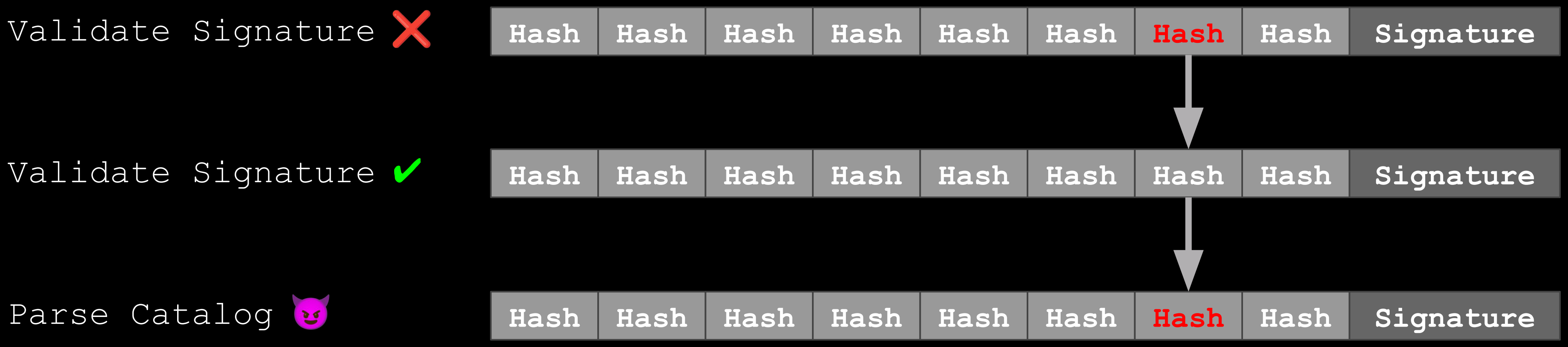
Hash	Hash	Hash	Hash	Hash	Hash	Hash	Hash	Signature

Parse Catalog ❌

Hash	Hash	Hash	Hash	Hash	Hash	Hash	Hash	Signature

Win - Payload Catalog Parsed

CI must validate a benign catalog then parse a malicious one.



Validate Signature ❌ | Hash | Hash | Hash | Hash | Hash | Hash | Hash | Hash | Signature

Validate Signature ✔ | Hash | Hash | Hash | Hash | Hash | Hash | Hash | Hash | Signature

Parse Catalog 😈 | Hash | Hash | Hash | Hash | Hash | Hash | Hash | Hash | Signature

Exploit Demo!

Windows 11 23H2 22631.3447 (April 2024)

Chapter 4 - Avoiding Pitfalls

To avoid this type of bug, we first need to understand it better.

Double Read

Imagine a shared memory mapping for an IPC mechanism. Double Read is a TOCTOU where victim reads a value from attacker-controlled shared memory twice.

Attacker changes memory between the reads, resulting in a unexpected victim behavior.

Example:

- Attacker initially specifies a small length field.
  - **pPacket->length = 16;**
- Victim code allocates a small buffer to hold data.
  - **pBuffer = malloc(pPacket->length);**
- Attacker changes to large length value.
  - **pPacket->length = 32;**
- Victim code uses new length, copying too much data and overflowing buffer.
  - **memcpy(pBuffer, pPacket->data, pPacket->length);** 💥

```
struct IPC_PACKET
{
    SIZE_T length;
    UCHAR data[];
};
```

Windows kernel (and drivers) often operate directly on user mode memory.

- Significant consideration for **METHOD_DIRECT** IOCTL handlers.

Recent example: https://exploits.forsale/24h2-nt-exploit/

Call To Action

Devs must treat attacker-writable files as subject to double-read vulnerabilities.

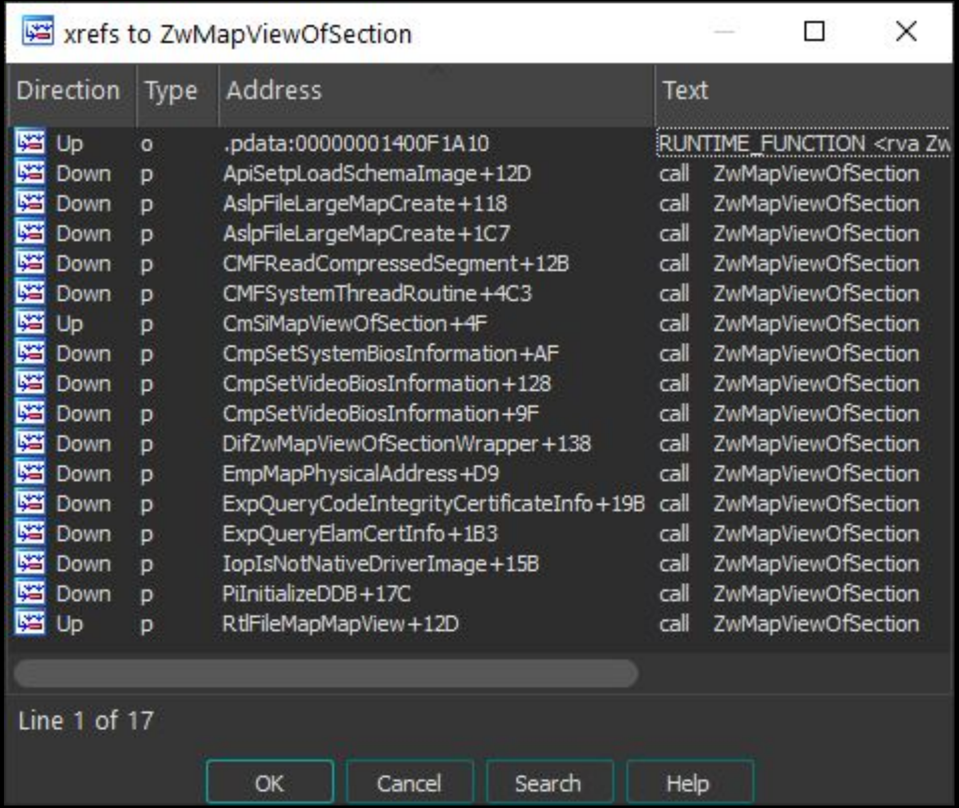
Denying write sharing does not necessarily prevent modification.

Affected Operations

What types of operations are affected by **False File Immutability**?

Operation	API	Mitigations
Image Sections	**CreateProcess** **LoadLibrary**	1. Enable Page Hashes.
Data Sections	**MapViewOfFile**	1. Avoid double reads. 2. Copy the file to a heap buffer before processing. 3. Prevent paging via **MmProbeAndLockPages/VirtualLock**.
Regular I/O	**ReadFile**	1. Avoid double reads. 2. Copy the file to a heap buffer before processing.

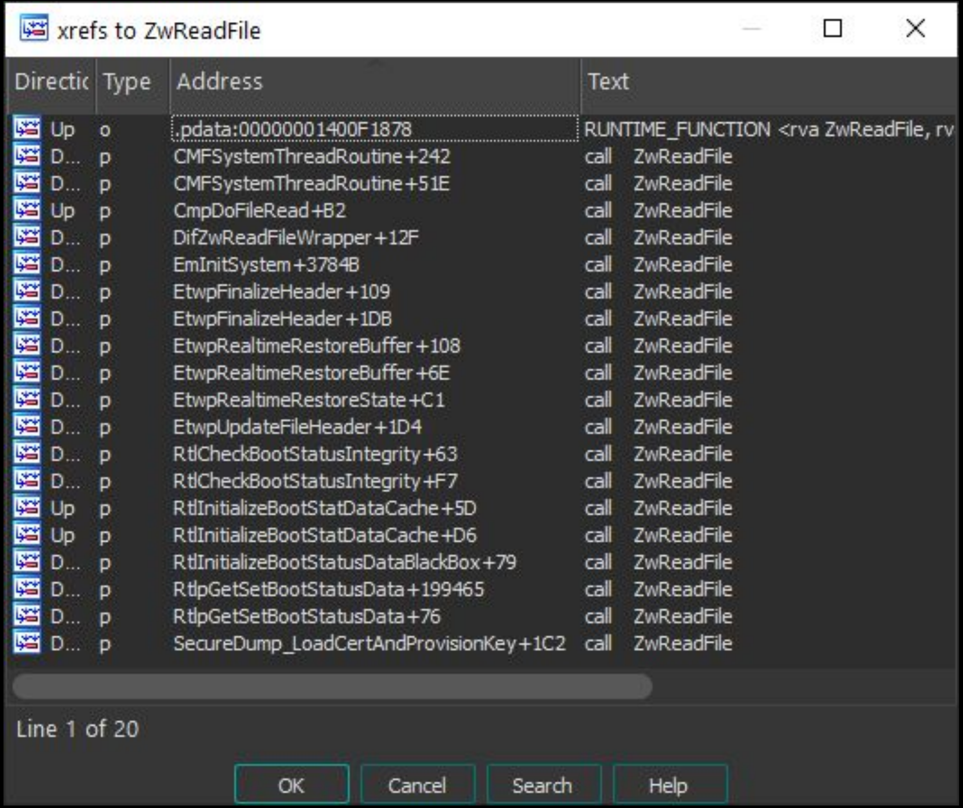
What Else Could Be Vulnerable?



xrefs to ZwMapViewOfSection

Direction	Type	Address	Text
Up	o	.pdata:00000001400F1A10	RUNTIME_FUNCTION <rva Zw
Down	p	ApiSetpLoadSchemaImage+12D	call ZwMapViewOfSection
Down	p	AslpFileLargeMapCreate+118	call ZwMapViewOfSection
Down	p	AslpFileLargeMapCreate+1C7	call ZwMapViewOfSection
Down	p	CMFReadCompressedSegment+12B	call ZwMapViewOfSection
Down	p	CMFSystemThreadRoutine+4C3	call ZwMapViewOfSection
Up	p	CmSiMapViewOfSection+4F	call ZwMapViewOfSection
Down	p	CmpSetSystemBiosInformation+AF	call ZwMapViewOfSection
Down	p	CmpSetVideoBiosInformation+128	call ZwMapViewOfSection
Down	p	CmpSetVideoBiosInformation+9F	call ZwMapViewOfSection
Down	p	DifZwMapViewOfSectionWrapper+138	call ZwMapViewOfSection
Down	p	EmpMapPhysicalAddress+D9	call ZwMapViewOfSection
Down	p	ExpQueryCodeIntegrityCertificateInfo+19B	call ZwMapViewOfSection
Down	p	ExpQueryElamCertInfo+1B3	call ZwMapViewOfSection
Down	p	IopIsNotNativeDriverImage+15B	call ZwMapViewOfSection
Down	p	PiInitializeDDB+17C	call ZwMapViewOfSection
Up	p	RtlFileMapMapView+12D	call ZwMapViewOfSection

Line 1 of 17



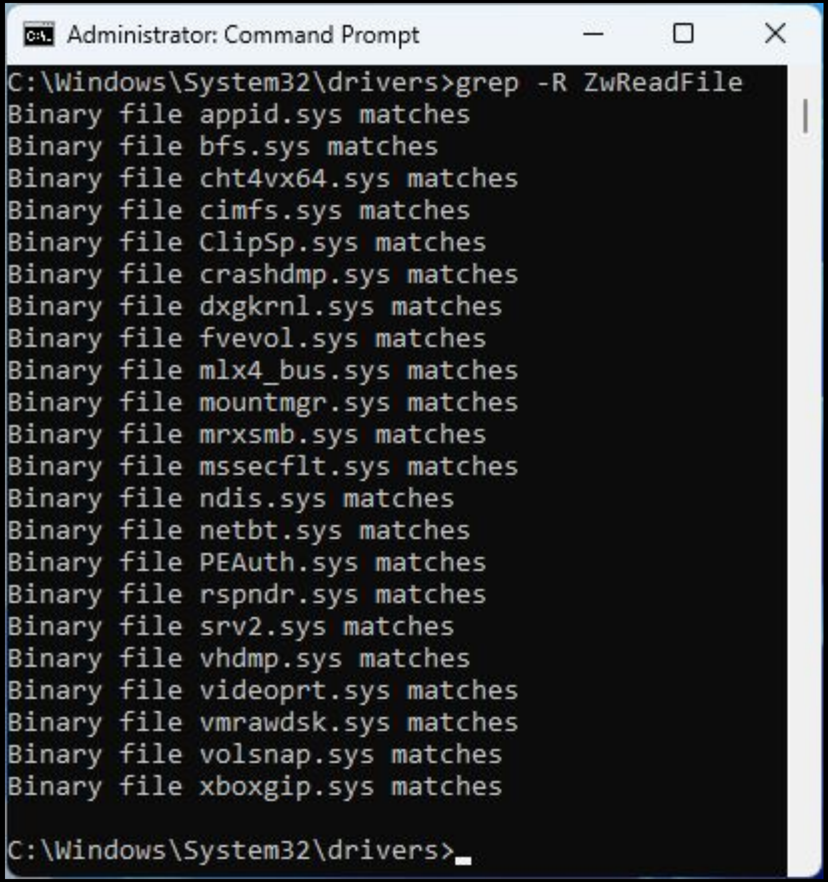
xrefs to ZwReadFile

Directic	Type	Address	Text
Up	o	.pdata:00000001400F1878	RUNTIME_FUNCTION <rva ZwReadFile, rv
D...	p	CMFSystemThreadRoutine+242	call ZwReadFile
D...	p	CMFSystemThreadRoutine+51E	call ZwReadFile
Up	p	CmpDoFileRead+B2	call ZwReadFile
D...	p	DifZwReadFileWrapper+12F	call ZwReadFile
D...	p	EmInitSystem+3784B	call ZwReadFile
D...	p	EtwpFinalizeHeader+109	call ZwReadFile
D...	p	EtwpFinalizeHeader+1DB	call ZwReadFile
D...	p	EtwpRealtimeRestoreBuffer+108	call ZwReadFile
D...	p	EtwpRealtimeRestoreBuffer+6E	call ZwReadFile
D...	p	EtwpRealtimeRestoreState+C1	call ZwReadFile
D...	p	EtwpUpdateFileHeader+1D4	call ZwReadFile
D...	p	RtlCheckBootStatusIntegrity+63	call ZwReadFile
D...	p	RtlCheckBootStatusIntegrity+F7	call ZwReadFile
Up	p	RtlInitializeBootStatDataCache+5D	call ZwReadFile
Up	p	RtlInitializeBootStatDataCache+D6	call ZwReadFile
D...	p	RtlInitializeBootStatusDataBlackBox+79	call ZwReadFile
D...	p	RtlpGetSetBootStatusData+199465	call ZwReadFile
D...	p	RtlpGetSetBootStatusData+76	call ZwReadFile
D...	p	SecureDump_LoadCertAndProvisionKey+1C2	call ZwReadFile

Line 1 of 20

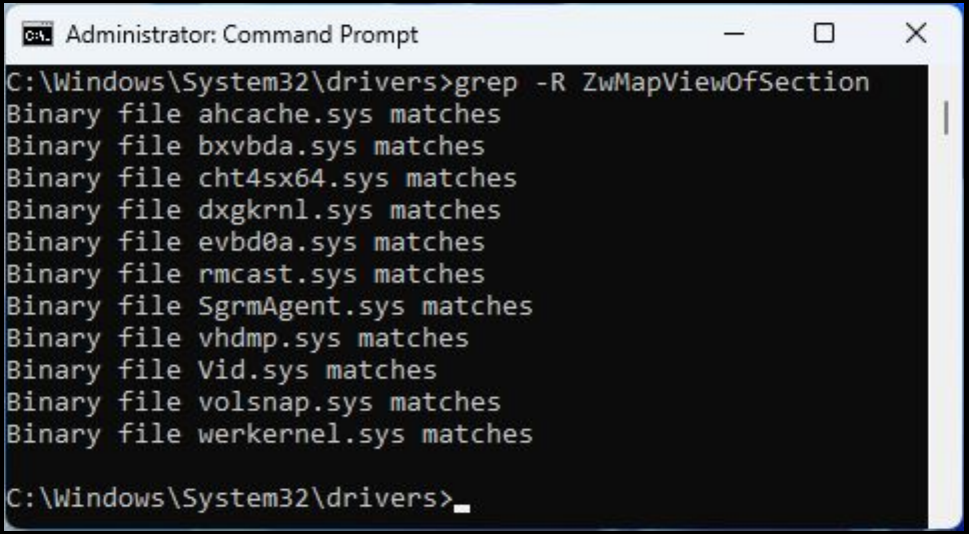
Note: **ZwReadFile** may be used for more than just files. Only uses on files (or those which could be coerced into operating on files) could be vulnerable.

What Else Could Be Vulnerable?



```
C:\Windows\System32\drivers>grep -R ZwReadFile
Binary file appid.sys matches
Binary file bfs.sys matches
Binary file cht4vx64.sys matches
Binary file cimfs.sys matches
Binary file ClipSp.sys matches
Binary file crashdmp.sys matches
Binary file dxgkrnl.sys matches
Binary file fvevol.sys matches
Binary file mlx4_bus.sys matches
Binary file mountmgr.sys matches
Binary file mrxsmb.sys matches
Binary file mssecflt.sys matches
Binary file ndis.sys matches
Binary file netbt.sys matches
Binary file PEAuth.sys matches
Binary file rspndr.sys matches
Binary file srv2.sys matches
Binary file vhdmp.sys matches
Binary file videoprt.sys matches
Binary file vmrawdsk.sys matches
Binary file volsnap.sys matches
Binary file xboxgip.sys matches

C:\Windows\System32\drivers>
```



```
C:\Windows\System32\drivers>grep -R ZwMapViewOfSection
Binary file ahcache.sys matches
Binary file bxvbda.sys matches
Binary file cht4sx64.sys matches
Binary file dxgkrnl.sys matches
Binary file evbd0a.sys matches
Binary file rmcast.sys matches
Binary file SgrmAgent.sys matches
Binary file vhdmp.sys matches
Binary file Vid.sys matches
Binary file volsnap.sys matches
Binary file werkernel.sys matches

C:\Windows\System32\drivers>
```

Note: **ZwReadFile** may be used for more than just files. Only uses on files (or those which could be coerced into operating on files) could be vulnerable.

Don’t Forget About User Mode

Any user-mode application that calls **ReadFile**, **MapViewOfFile**, or **LoadLibrary** on an attacker-controllable file, denying write sharing for immutability, may be vulnerable.

Hypothetical examples:

- **MapViewOfFile**
  - Auto-elevate installers that apply downloaded patches if correctly signed
- **ReadFile**
  - Memory corruption in file parsers by changing double-read values
    - AV engines
    - Search indexers
- **LoadLibrary**
  - RPC server relying on **SetProcessMitigationPolicy(ProcessSignaturePolicy)** to prevent DLL injection via impersonation system drive remapping attacks.

https://bugs.chromium.org/p/project-zero/issues/detail?id=2451

Chapter 5 - Mitigating the Exploit

MSRC won’t service Admin -> Kernel vulnerabilities by default.

- “service” means “fix via security update.”

As a third-party AV dev, I can’t fix CI.dll. How can I protect my customers?

What can Microsoft do to fix it?

https://www.microsoft.com/en-us/msrc/windows-security-servicing-criteria

Third-Party Mitigation

To mitigate **ItsNotASecurityBoundary**, I wrote **FineButWeCanStillEasilyStopIt.sys**

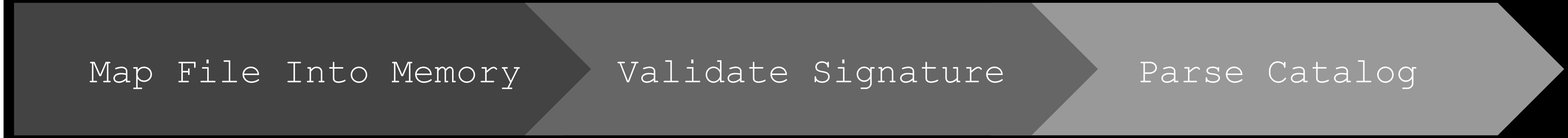
Filesystem Minifilter. In Pre **IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION** callback invoked during **ZwCreateSection**, if:

- **SyncType == SyncTypeCreateSection &&**
- **PageProtection == PAGE_READONLY &&**
- **FlagOn(TargetFileObject->DeviceObject->Characteristics, FILE_REMOTE_DEVICE) &&**
- **Data->RequestorMode == KernelMode &&**
- **FltGetRequestorProcess(Data) == PsInitialSystemProcess &&**
- **IsCalledByCodeIntegrity() && // Check caller via RtlWalkFrameChain**
- **Contains catalog magic bytes and Certificate Trust List PKCS #7 OID.**

then deny the operation.

Messy, right? It’s likely imperfect too. Compare that to a three-line fix in CI.

DSE Exploit Mitigation #1



Map File Into Memory

```
nt!ZwOpenFile(
  GENERIC_READ,
  FILE_SHARE_READ)

nt!ZwCreateSection(
  SEC_COMMIT)

nt!ZwMapViewOfSection
```

nt!ExAllocatePool2

nt!RtlCopyMemory

Copy the file to a heap buffer before processing.

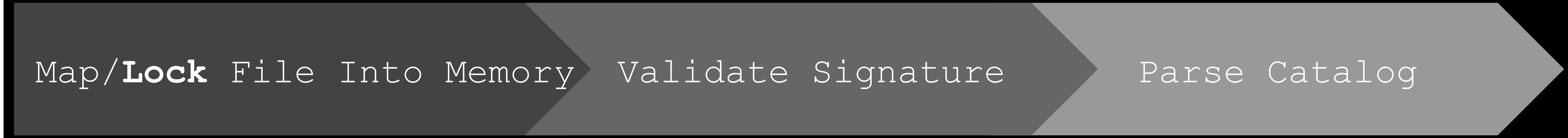
Validate Signature

```
CI!MinCrypK_
 VerifySignedDataKModeEx
```

Parse Catalog

```
CI!I_MapFileHashes
```

DSE Exploit Mitigation #2



Map/**Lock** File Into Memory

Validate Signature

Parse Catalog

```
nt!ZwOpenFile(
  GENERIC_READ,
  FILE_SHARE_READ)

nt!ZwCreateSection(
  SEC_COMMIT)

nt!ZwMapViewOfSection

nt!IoAllocateMdl

nt!MmProbeAndLockPages
```

```
CI!MinCrypK_
 VerifySignedDataKModeEx
```

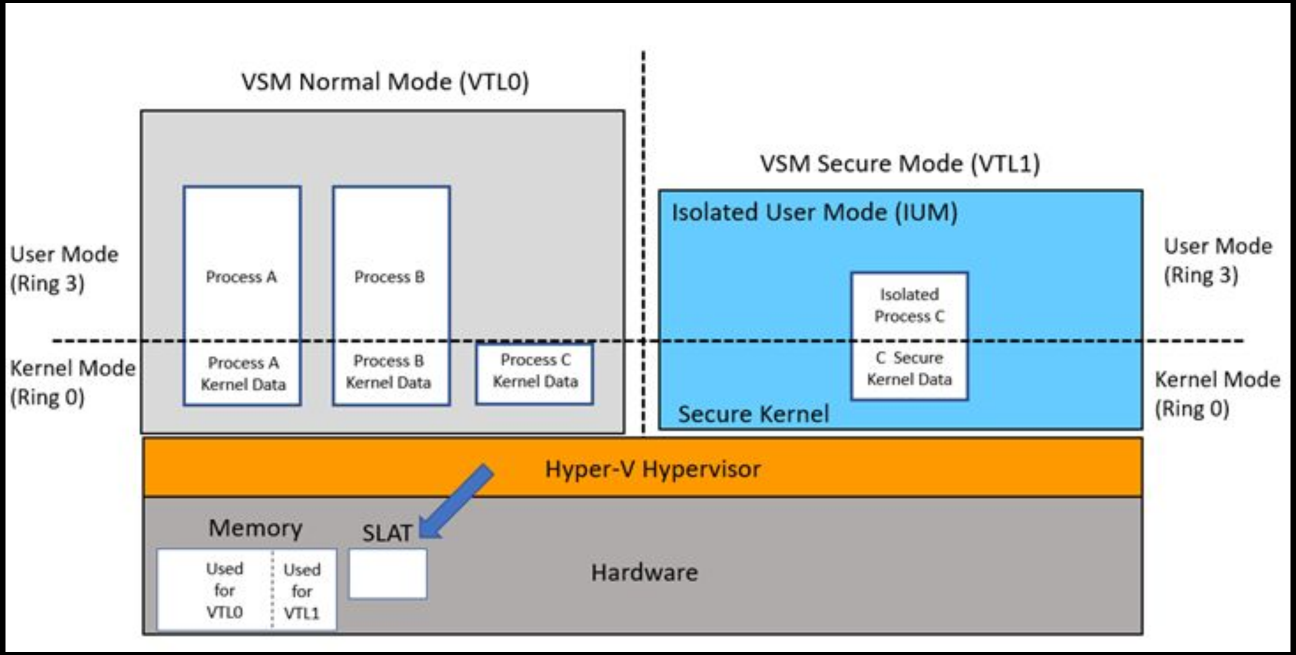
```
CI!I_MapFileHashes
```

Lock pages into RAM to block working set eviction.

Mitigating the Exploit - HVCI

If HVCI is enabled, CI.dll doesn’t do catalog parsing.

- CI sends the catalog contents to the Secure Kernel (SK)
- SK runs in a separate virtual machine.
- SK puts catalog contents in its own secure allocation.
- Signature validation and parsing are done from this secure allocation.
- Attack is mitigated because file changes have no effect on the secure allocation.



https://learn.microsoft.com/en-us/windows/win32/procthread/isolated-user-mode--ium--processes

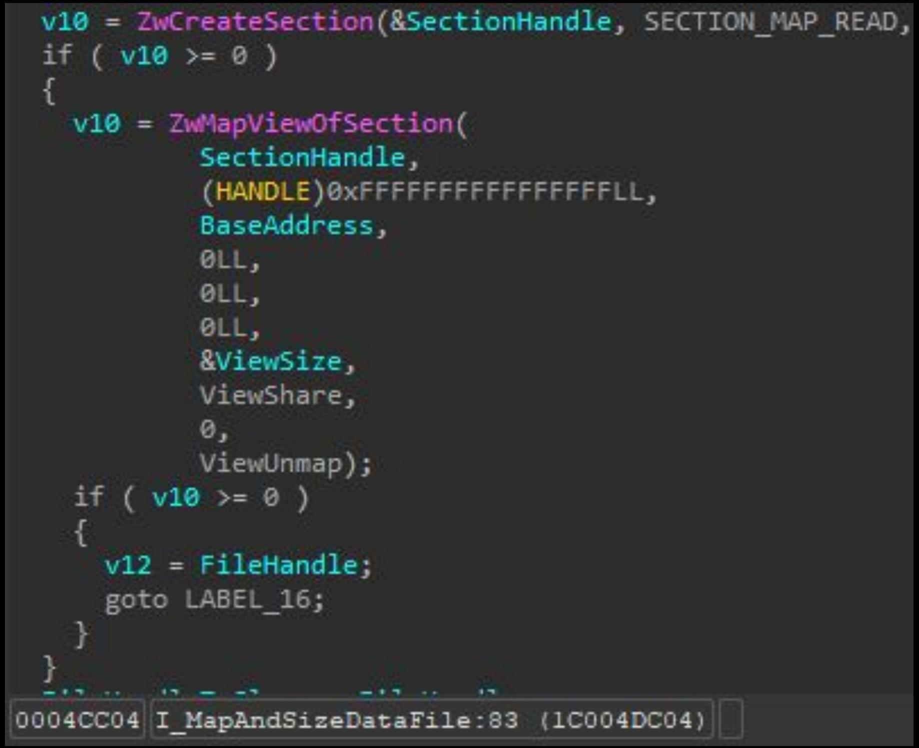
Disclosure Timeline

- 2024-02-14 Reported **ItsNotASecurityBoundary** and **FineButWeCanStillEasilyStopIt** to MSRC as VULN-119340, suggesting **ExAllocatePool** and **MmProbeAndLockPages** as fixes.
- 2024-02-29 Windows Defender team reached out to coordinate disclosure.
- 2024-04-23 Microsoft releases KB5036980 preview with **MmProbeAndLockPages** fix.
- 2024-05-14 Fix reaches GA for desktop releases.

https://support.microsoft.com/en-us/topic/april-23-2024-kb5036980-os-builds-22621-3527-and-22631-3527-preview-5a0d6c49-e42e-4eb4-8541-33a7139281ed

Inside The Mitigation

I_MapAndSizeDataFile is the legacy vulnerable code.



```
v10 = ZwCreateSection(&SectionHandle, SECTION_MAP_READ,
if ( v10 >= 0 )
{
  v10 = ZwMapViewOfSection(
          SectionHandle,
          (HANDLE)0xFFFFFFFFFFFFFFFFLL,
          BaseAddress,
          0LL,
          0LL,
          0LL,
          &ViewSize,
          ViewShare,
          0,
          ViewUnmap);
  if ( v10 >= 0 )
  {
    v12 = FileHandle;
    goto LABEL_16;
  }
}
```

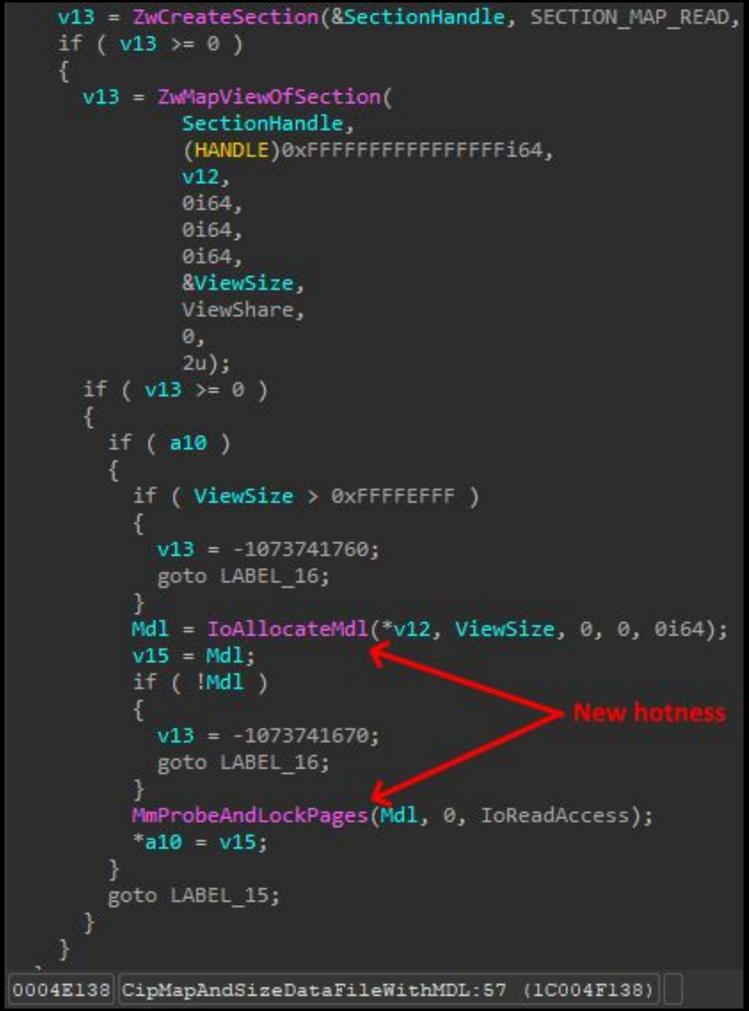
0004CC04 I_MapAndSizeDataFile:83 (1C004DC04)

https://www.youtube.com/watch?v=ha-uagjJQ9k

Inside The Mitigation

CipMapAndSizeDataFileWithMDL contains the fix.

https://www.youtube.com/watch?v=ha-uagjJQ9k



```
v13 = ZwCreateSection(&SectionHandle, SECTION_MAP_READ,
if ( v13 >= 0 )
{
  v13 = ZwMapViewOfSection(
          SectionHandle,
          (HANDLE)0xFFFFFFFFFFFFFFFFi64,
          v12,
          0i64,
          0i64,
          0i64,
          &ViewSize,
          ViewShare,
          0,
          2u);
  if ( v13 >= 0 )
  {
    if ( a10 )
    {
      if ( ViewSize > 0xFFFFEFFF )
      {
        v13 = -1073741760;
        goto LABEL_16;
      }
      Mdl = IoAllocateMdl(*v12, ViewSize, 0, 0, 0i64);
      v15 = Mdl;
      if ( !Mdl )
      {
        v13 = -1073741670;
        goto LABEL_16;
      }
      MmProbeAndLockPages(Mdl, 0, IoReadAccess);
      *a10 = v15;
    }
    goto LABEL_15;
  }
}
```

Summary

Bug class: False File Immutability

PPLFault: Admin -> PPL [-> Kernel via GodFault]

- Exploits bad immutability assumptions about image section in CI/MM
- Reported September 2022
- Patched February 2024

ItsNotASecurityBoundary: Admin -> Kernel

- Exploits bad immutability assumptions about data sections in CI
- Reported February 2024
- Patched May 2024

More exploits: TBA 😀

https://x.com/GabrielLandau/status/1757818200127946922

Conclusion

Exploit PoC to be released in late June. Announcement on Twitter.

Thanks to the Windows Defender team for collaborating on disclosure and fixes!

Gabriel Landau at Elastic Security

Twitter/𝕏: @GabrielLandau