# Contain Yourself: Staying Undetected Using the Windows Containers Isolation Framework

DEF CON 31 2023

# About Me

@daniel_Avinoam

Security Researcher @ deep instinct

Interested in Windows internals, reverse engineering and low-level programming

Motorsport fan and weightlifter

# Today’s Agenda

01

## Overview

Windows containers basics

02

## wcifs.sys

Analyzing the main filter driver responsable for containers FS isolation

03

## Utilization

Bypassing security products using our findings

04

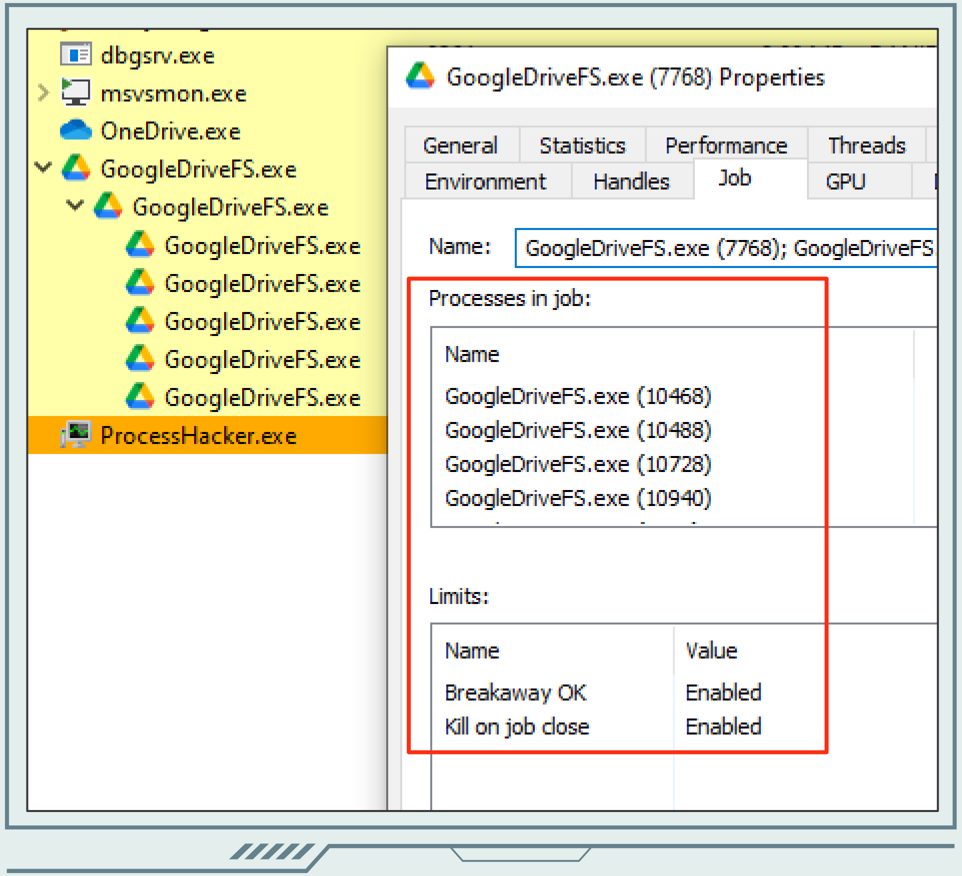
## Summary

Summary, mitigation and future research

# Windows Containers – The Basics

## Jobs

- Objects designed to group several processes and manage them as one unit
- Allows limitation of CPU usage, I/O bandwidth, virtual memory usage, and network activity of managed processes
- Often used by multi-processed applications



Name: GoogleDriveFS.exe (7768); GoogleDriveFS

Processes in job:

| Name |
|---|
| GoogleDriveFS.exe (10468) |
| GoogleDriveFS.exe (10488) |
| GoogleDriveFS.exe (10728) |
| GoogleDriveFS.exe (10940) |

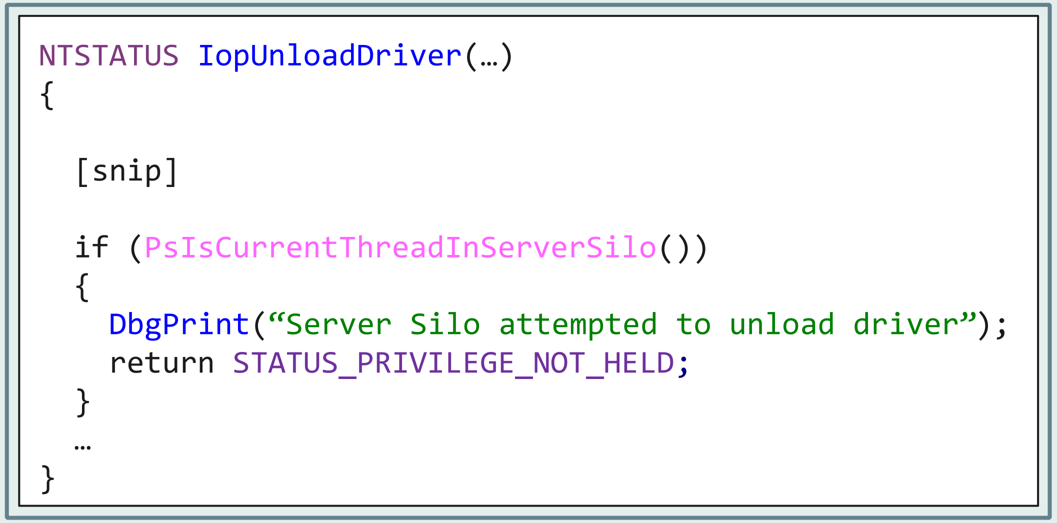
Limits:

| Name | Value |
|---|---|
| Breakaway OK | Enabled |
| Kill on job close | Enabled |

# Windows Containers – The Basics

## Silos

- Like traditional jobs, these objects are used for process grouping with additional features
- Server silo - provides separation of various system resources like the registry, networking, and the object manager
- The Windows kernel detects processes assigned to silos using APIs like *PsIsCurrentThreadInServerSilo* and *PsIsProcessInSilo*

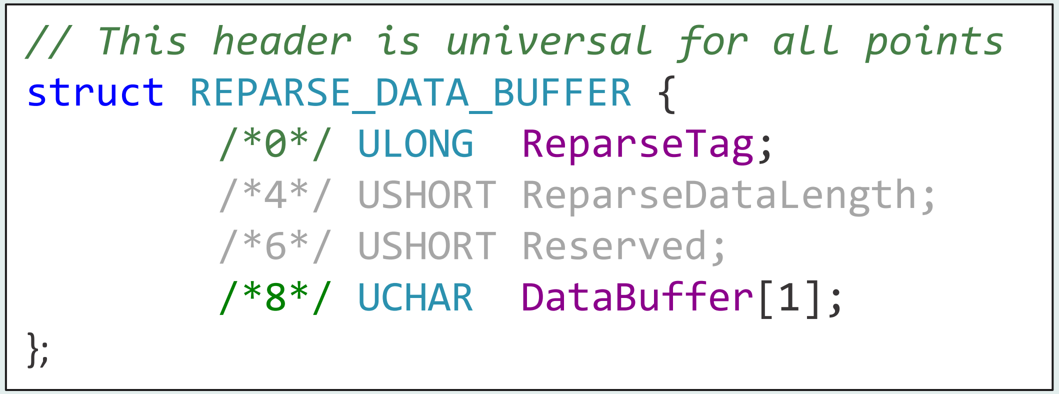


```
NTSTATUS IopUnloadDriver(…)
{

  [snip]

  if (PsIsCurrentThreadInServerSilo())
  {
    DbgPrint(“Server Silo attempted to unload driver”);
    return STATUS_PRIVILEGE_NOT_HELD;
  }
  …
}
```

# Windows Containers – The Basics

## File System Redirection Using Reparse Points

- Reparse points are MFT attributes that can be given to files or directories
- Stores a reparse tag and user-defined data



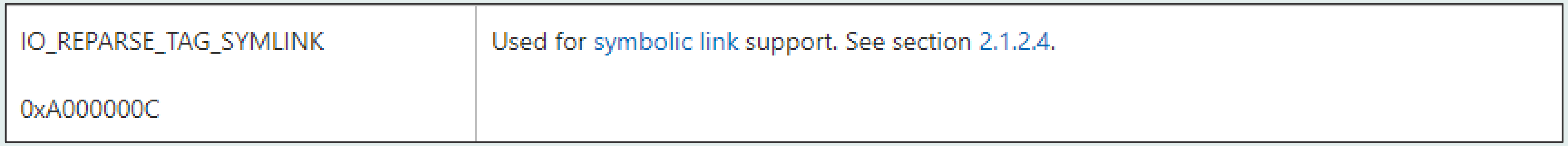
```
// This header is universal for all points
struct REPARSE_DATA_BUFFER {
        /*0*/ ULONG  ReparseTag;
        /*4*/ USHORT ReparseDataLength;
        /*6*/ USHORT Reserved;
        /*8*/ UCHAR  DataBuffer[1];
};
```

- Can be set using `DeviceIoControl` + `FSCTL_SET_REPARSE_POINT` (`WRITE` primitive needed)

# Windows Containers – The Basics

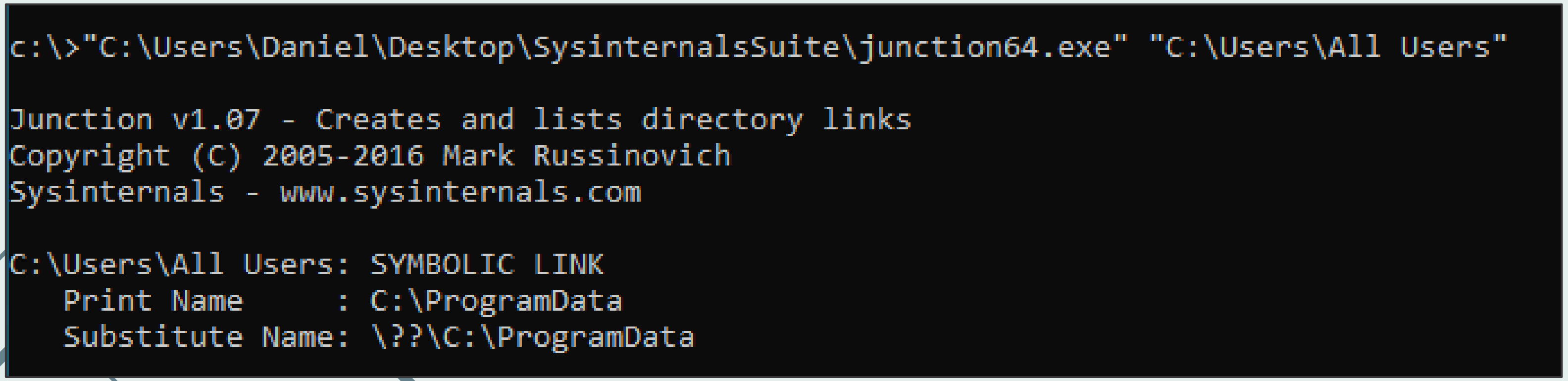
## File System Redirection Using Reparse Points

- When a file with a reparse point is opened, it is handled by a file system mini-filter driver according to its reparse tag
- An example of a common tag is `IO_REPARSE_TAG_SYMLINK`, which is how symbolic links work behind the scenes.



| | |
|---|---|
| IO_REPARSE_TAG_SYMLINK<br>0xA000000C | Used for symbolic link support. See section 2.1.2.4. |

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/c8e77b37-3909-4fe6-a4ea-2b9d423b1ee4

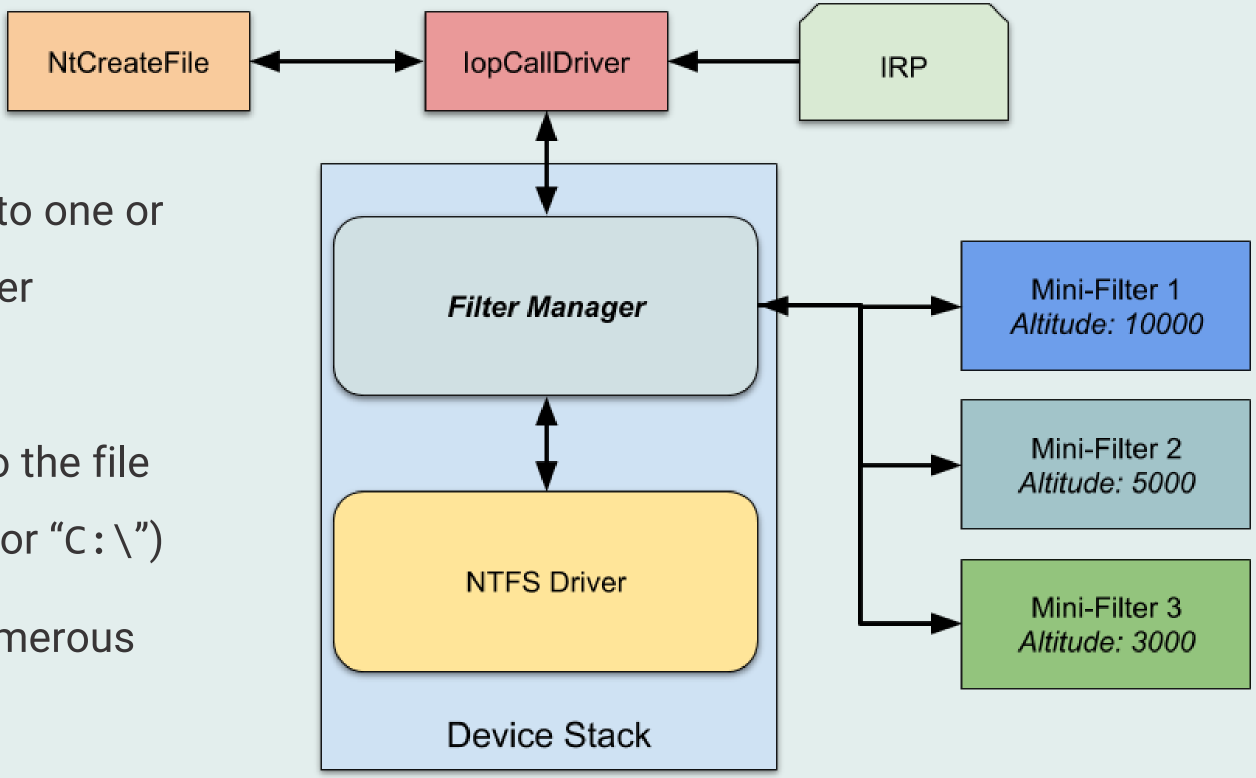


```
c:\>"C:\Users\Daniel\Desktop\SysinternalsSuite\junction64.exe" "C:\Users\All Users"

Junction v1.07 - Creates and lists directory links
Copyright (C) 2005-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Users\All Users: SYMBOLIC LINK
   Print Name     : C:\ProgramData
   Substitute Name: \??\C:\ProgramData
```

# Windows Containers – The Basics

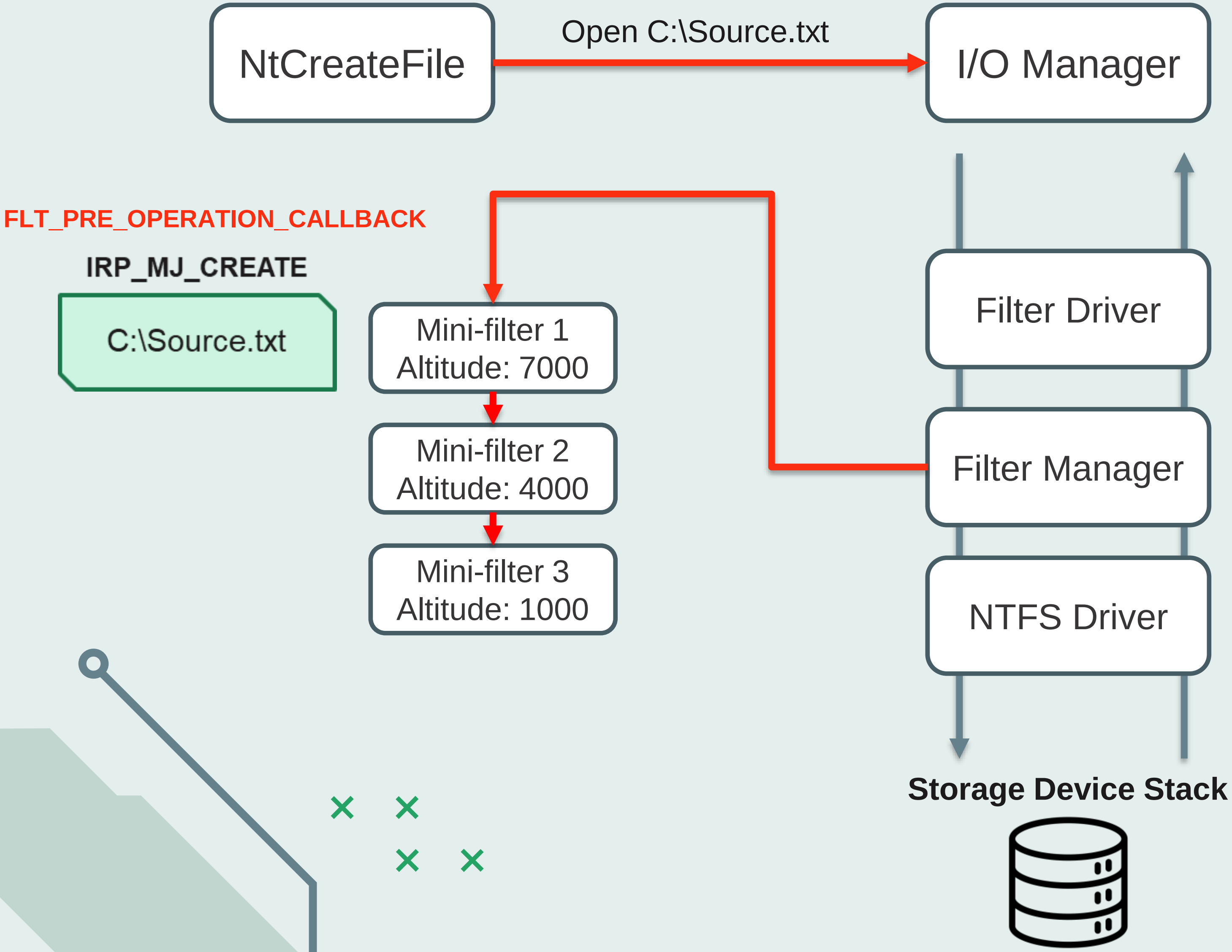
## Mini-filters Background

- Filter manager - a legacy filter that manages other “mini” filter drivers (exposes the `Flt` API)
- Each mini-filter can be attached by the manager to one or more volumes, creating what is called a “mini-filter instance”
- Volume = logical storage unit that is presented to the file system as a disk (“`\Device\HarddiskVolume3`” or “`C:\`”)
- Can intercept the `PRE` and `POST` operations of numerous I/O functions
- Attached and ordered according to an altitude



https://googleprojectzero.blogspot.com/2021/01/hunting-for-bugs-in-windows-mini-filter.html

# Mini-filters and Reparse Points

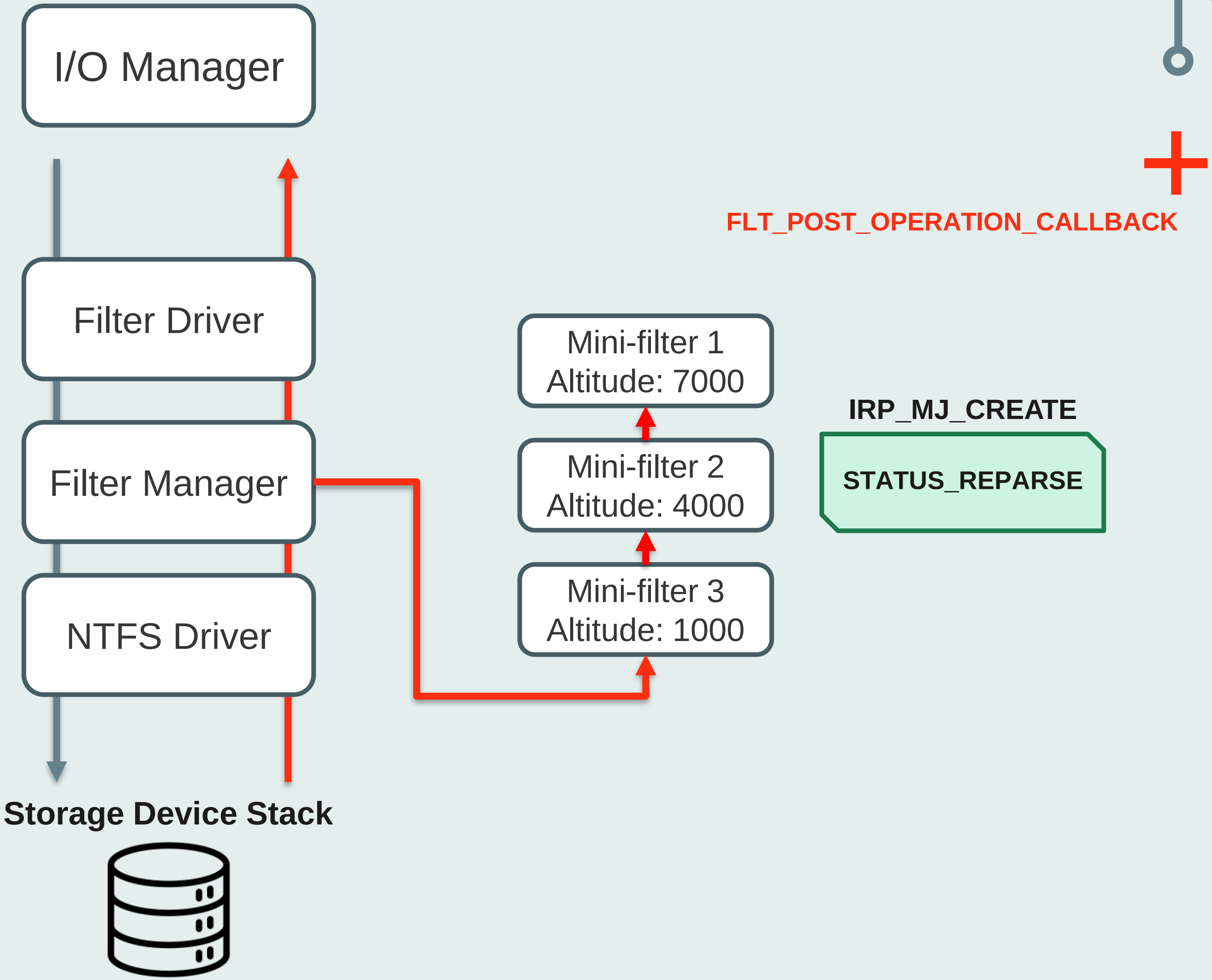


# Mini-filters and Reparse Points

**Option 1:**

```
FltFsControlFile(…,
…,
FSCTL_GET_REPARSE_POINT,
&ReparsePointData);
```

```
ReparsePointData =
{
  MINIFILTER_1_TAG;
  Minifilter1ReparseDataLen;
  Minifilter1ReparseData[];
}
```

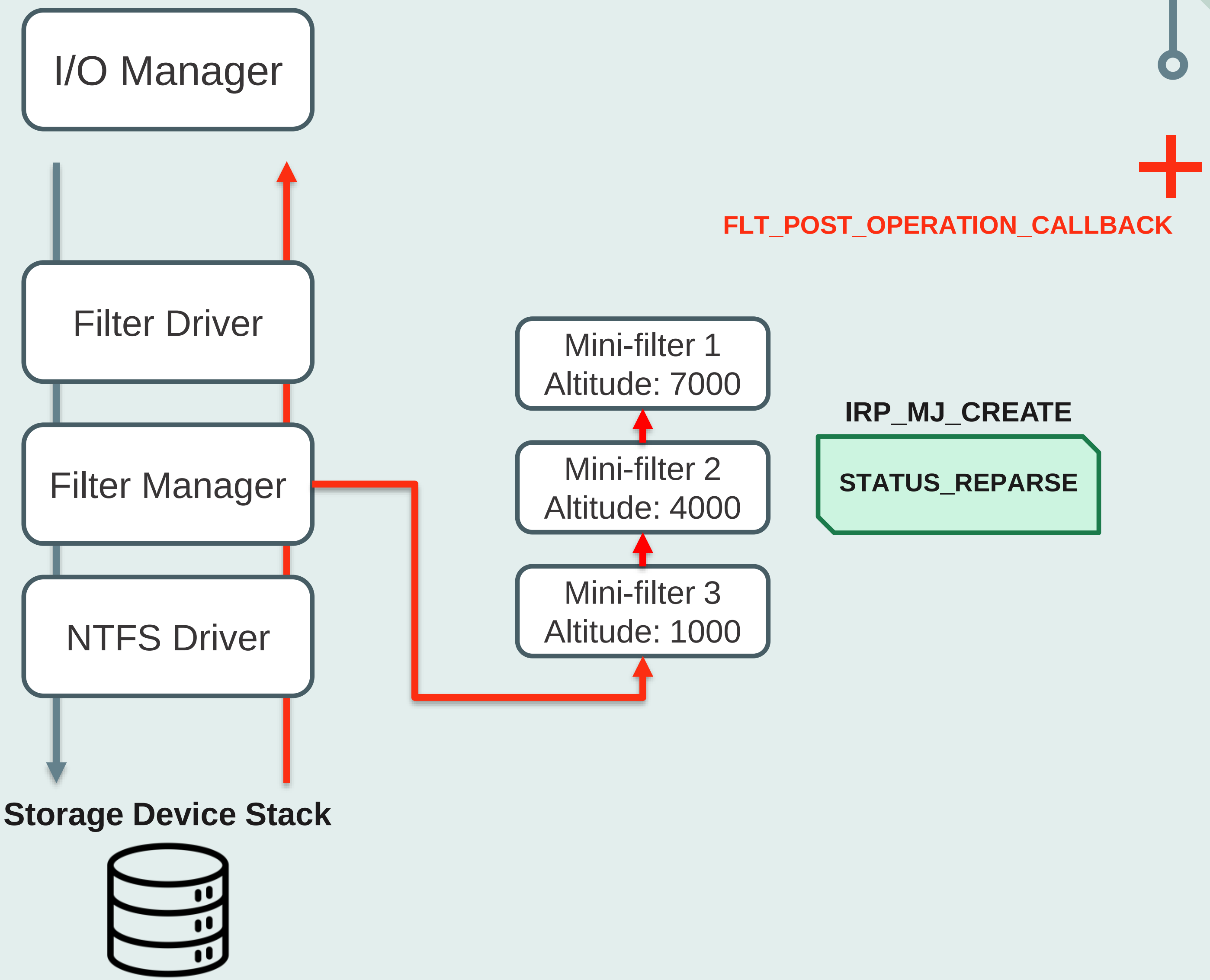


# Mini-filters and Reparse Points

**Option 2:**

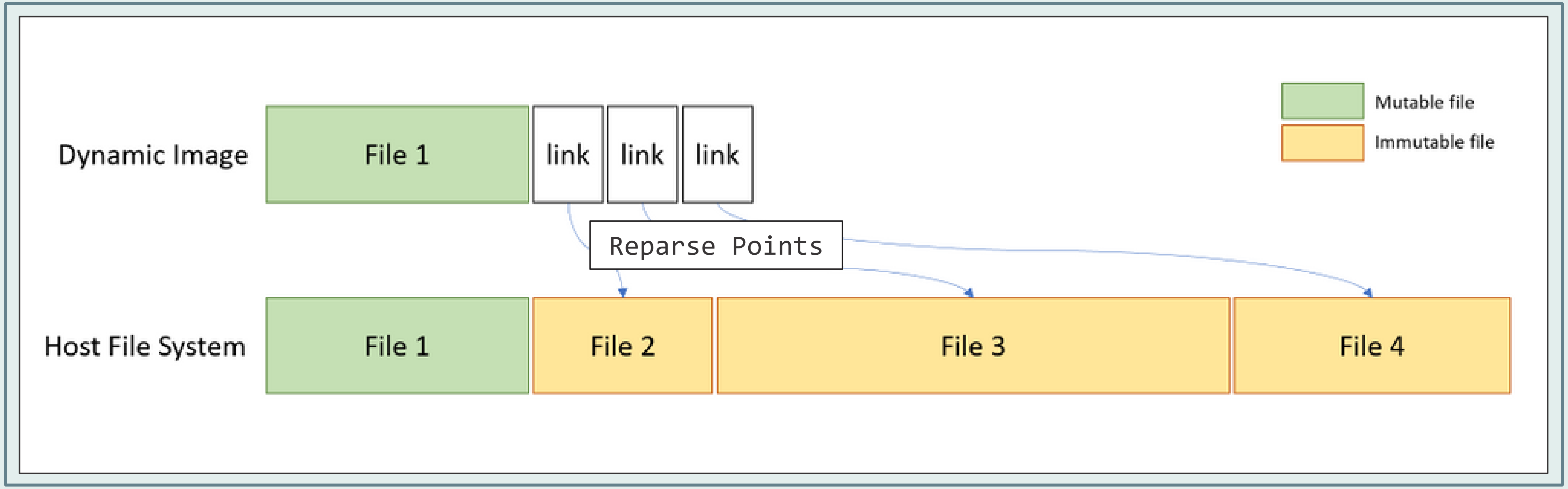
```
FltFsControlFile(…,
…,
FSCTL_GET_REPARSE_POINT,
&ReparsePointData);
```

```
ReparsePointData =
{
  MINIFILTER_2_TAG;
  MInifilter2ReparseDataLen;
  MInifilter2ReparseData;
  {
    TargetPath = “C:\\Target.txt”;
  };
}
```



# Containers File System Separation

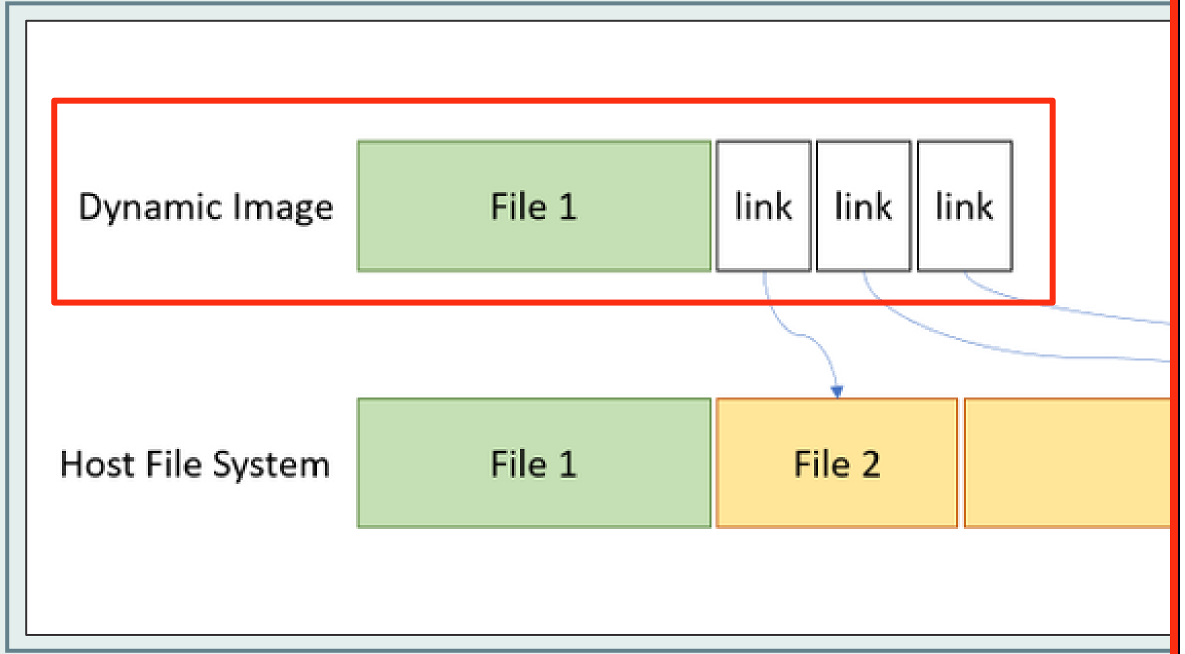
In order to avoid an additional copy of OS files, each container is using a dynamically generated image which points to the original using reparse points

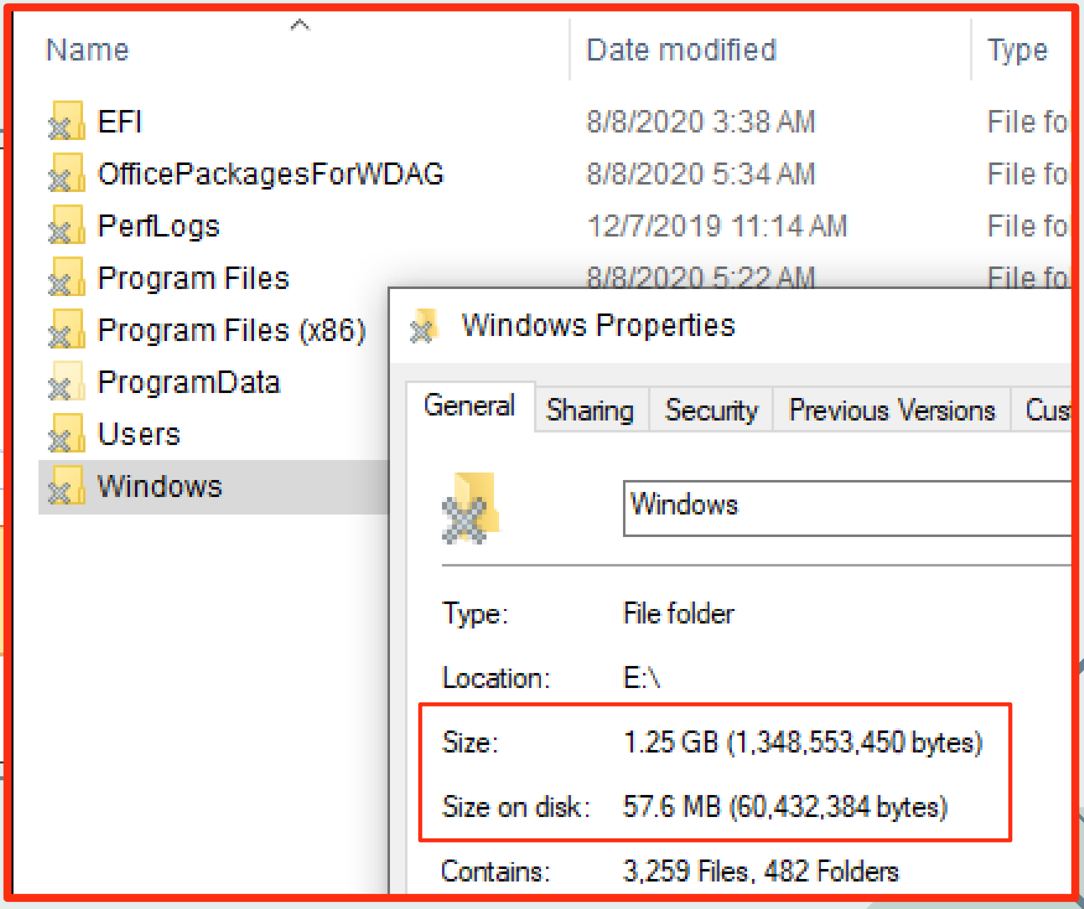


https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/windows-sandbox/windows-sandbox-architecture

# Containers File System Separation

The result is images that contain “ghost files”, which store no actual data but point to a different volume on the system





| Name | Date modified | Type |
|---|---|---|
| EFI | 8/8/2020 3:38 AM | File fo |
| OfficePackagesForWDAG | 8/8/2020 5:34 AM | File fo |
| PerfLogs | 12/7/2019 11:14 AM | File fo |
| Program Files | 8/8/2020 5:22 AM | File fo |
| Program Files (x86) | | |
| ProgramData | | |
| Users | | |
| Windows | | |

Windows

Type: File folder

Location: E:\

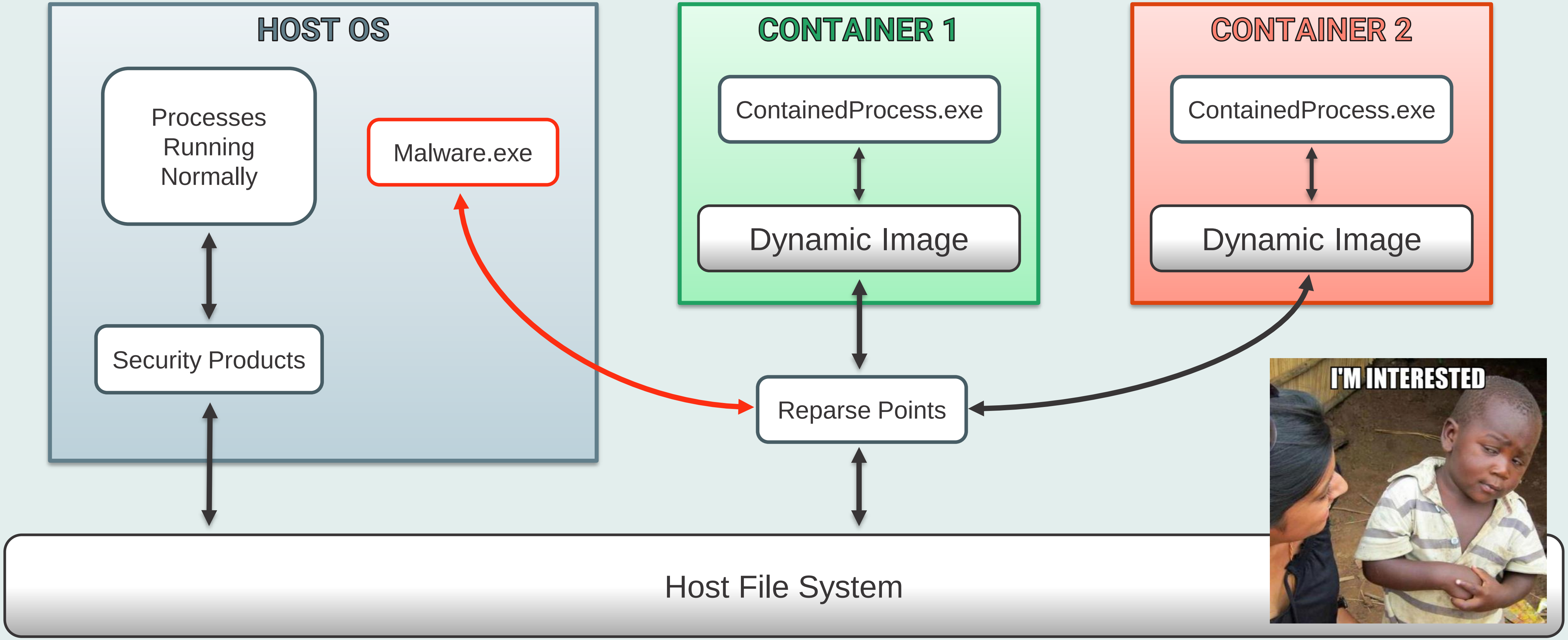
Size: 1.25 GB (1,348,553,450 bytes)

Size on disk: 57.6 MB (60,432,384 bytes)

Contains: 3,259 Files, 482 Folders

https://research.checkpoint.com/2021/playing-in-the-windows-sandbox/

# Containers File System Separation



01

## Overview

Windows containers basics

02

## wcifs.sys

Analyzing the main filter driver responsable for containers FS isolation

03

## Utilization

Bypassing security products using our findings

04

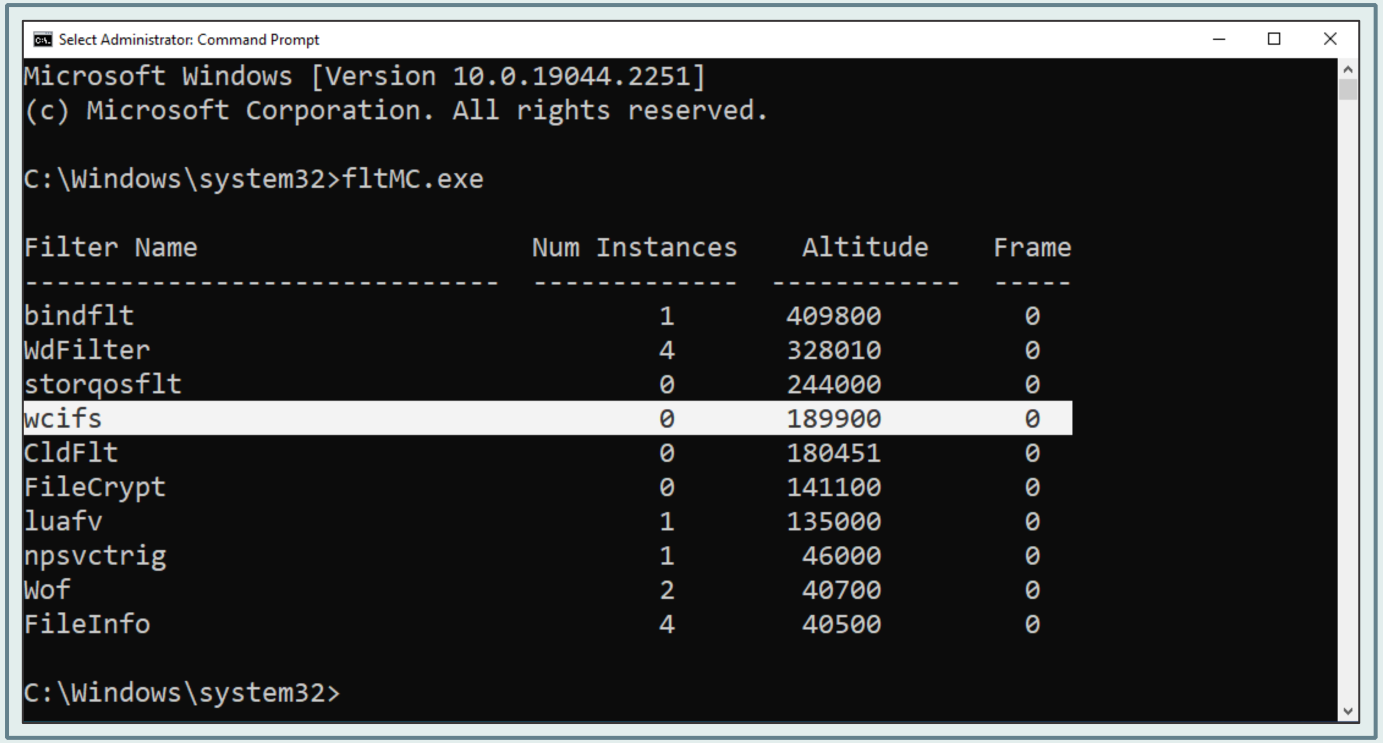
## Summary

Summary, mitigation and future research

# wcifs.sys

The Windows Container Isolation FS (wcifs) mini-filter driver is responsible for the file system separation between windows containers and their host

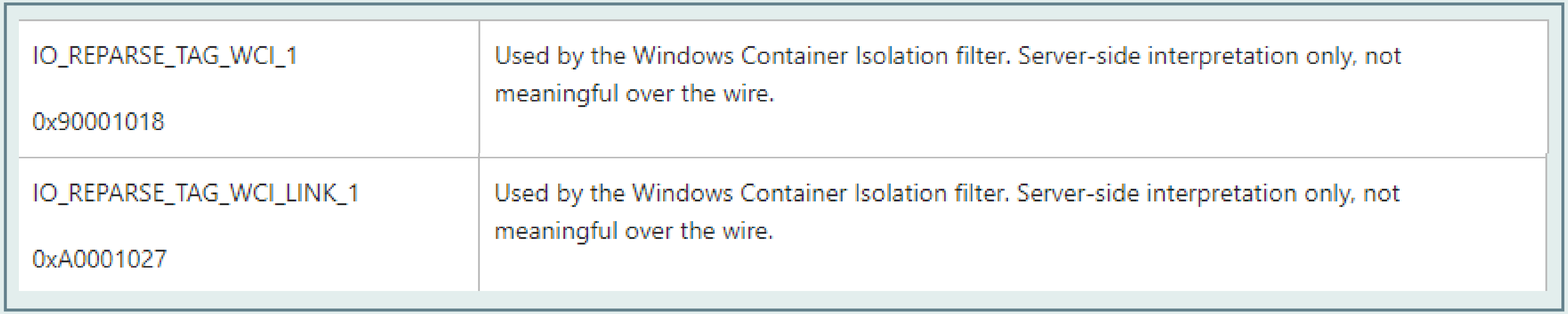
The driver is loaded by default on every Windows system starting from Windows 10



```
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>fltMC.exe

Filter Name                     Num Instances    Altitude    Frame
------------------------------  -------------  ------------  -----
bindflt                                 1       409800         0
WdFilter                                4       328010         0
storqosflt                              0       244000         0
wcifs                                   0       189900         0
CldFlt                                  0       180451         0
FileCrypt                               0       141100         0
luafv                                   1       135000         0
npsvctrig                               1        46000         0
Wof                                     2        40700         0
FileInfo                                4        40500         0

C:\Windows\system32>
```

# wcifs.sys

The main reparse tags associated with this driver are `IO_REPARSE_TAG_WCI_1` and `IO_REPARSE_TAG_WCI_LINK_1`

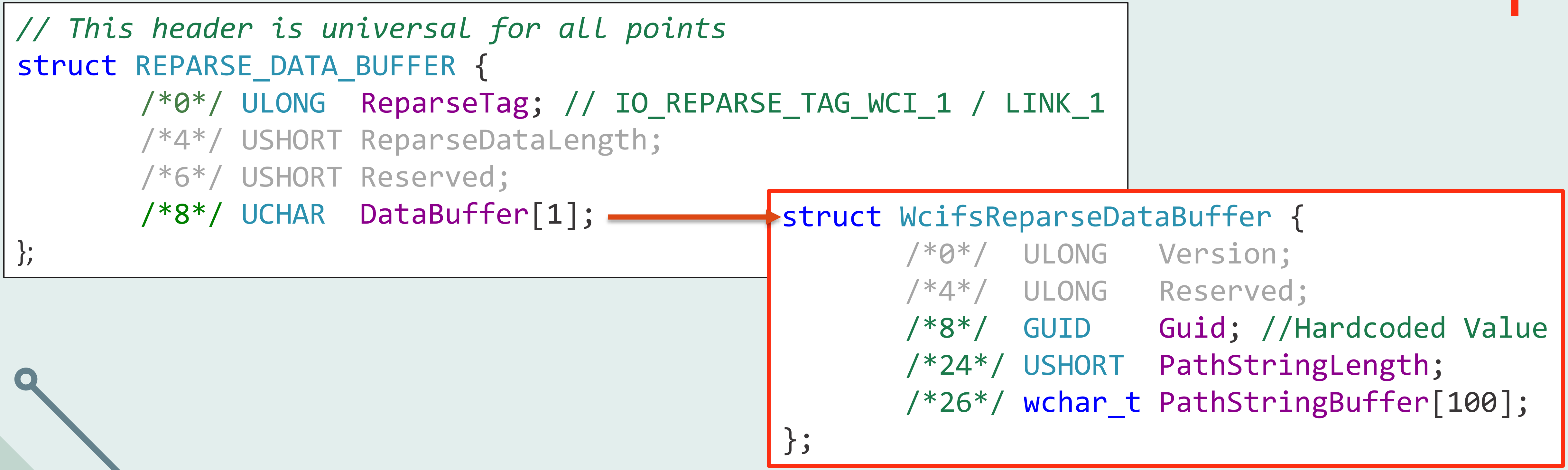


| | |
|---|---|
| IO_REPARSE_TAG_WCI_1<br>0x90001018 | Used by the Windows Container Isolation filter. Server-side interpretation only, not meaningful over the wire. |
| IO_REPARSE_TAG_WCI_LINK_1<br>0xA0001027 | Used by the Windows Container Isolation filter. Server-side interpretation only, not meaningful over the wire. |

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-fscc/c8e77b37-3909-4fe6-a4ea-2b9d423b1ee4

# wcifs.sys

The main reparse tags associated with this driver are **IO_REPARSE_TAG_WCI_1** and **IO_REPARSE_TAG_WCI_LINK_1**

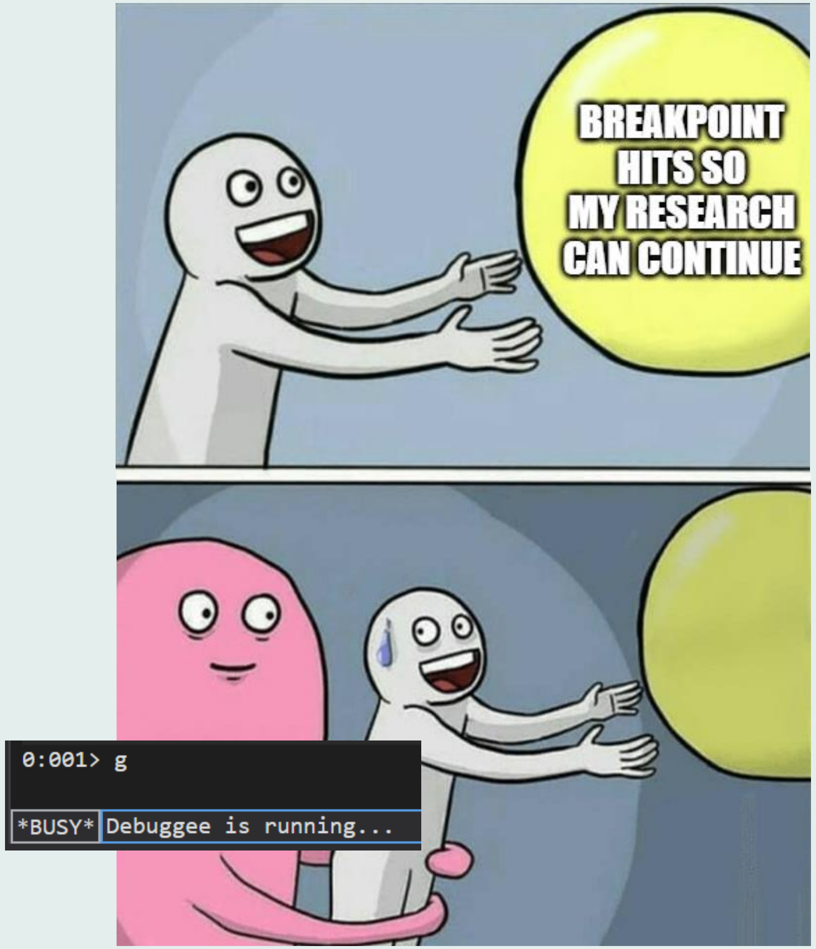


```
// This header is universal for all points
struct REPARSE_DATA_BUFFER {
        /*0*/ ULONG  ReparseTag; // IO_REPARSE_TAG_WCI_1 / LINK_1
        /*4*/ USHORT ReparseDataLength;
        /*6*/ USHORT Reserved;
        /*8*/ UCHAR  DataBuffer[1];
};
```

```
struct WcifsReparseDataBuffer {
        /*0*/   ULONG   Version;
        /*4*/   ULONG   Reserved;
        /*8*/   GUID    Guid; //Hardcoded Value
        /*24*/ USHORT  PathStringLength;
        /*26*/ wchar_t PathStringBuffer[100];
};
```

# wcifs.sys – Reverse Engineering

All there is left to do is to attach wcifs to a volume, place a breakpoint on its POST_OP callback while debugging it and see how its reparse points are being handled



```
0:001> g

*BUSY* Debuggee is running...
```

# wcifs.sys – Reverse Engineering

For the POST_OP callback to invoke, either **FLT_PREOP_SUCCESS_WITH_CALLBACK** or **FLT_PREOP_SYNCRONIZE** must be returned in the PRE_OP!

```
FLT_PREOP_CALLBACK_STATUS WcPreCreate(…)
{

   [snip]

   if (!WcUnionsExistForInstance(FltObjects->Instance, …))
   {
       return FLT_PREOP_SUCCESS_NO_CALLBACK;
   }
   …

   return FLT_PREOP_SYNCRONIZE;
}
```

```
BOOL WcUnionsExistForInstance(…)
{

    [snip]

    Silo = IoGetSilo(FileObject);
    IsHostSilo = PsIsHostSilo(Silo);

    if (IsHostSilo)
        return FALSE;

    if (!NT_SUCCESS(PsGetSiloContext(Silo, …)))
        return FALSE;
    …
}
```

A context is a structure that is defined by the mini-filter driver and that can be associated with a filter manager object, like files, instances and silos.

# wcifs.sys – Creating a Fake Container

1. Create a silo and insert our process into it
2. Inform the driver our silo is representing a container, so it will create a union context and handle it accordingly

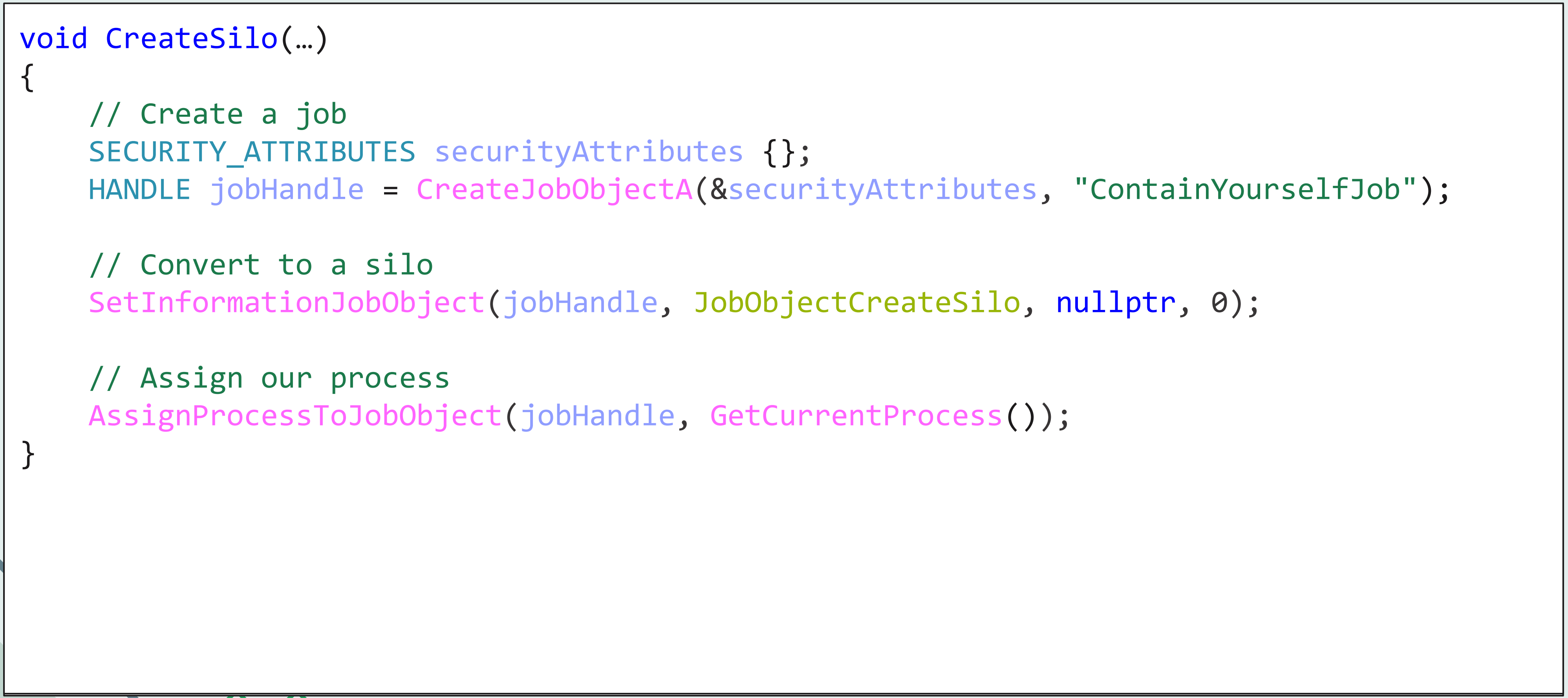
# wcifs.sys – Creating a Fake Container

1. **Create a silo and insert our process into it**
2. Inform the driver our silo is representing a container, so it will create a union context and handle it accordingly

```
void CreateSilo(…)
{
    // Create a job
    SECURITY_ATTRIBUTES securityAttributes {};
    HANDLE jobHandle = CreateJobObjectA(&securityAttributes, "ContainYourselfJob");

    // Convert to a silo
    SetInformationJobObject(jobHandle, JobObjectCreateSilo, nullptr, 0);

    // Assign our process
    AssignProcessToJobObject(jobHandle, GetCurrentProcess());
}
```

# wcifs.sys – Creating a Fake Container

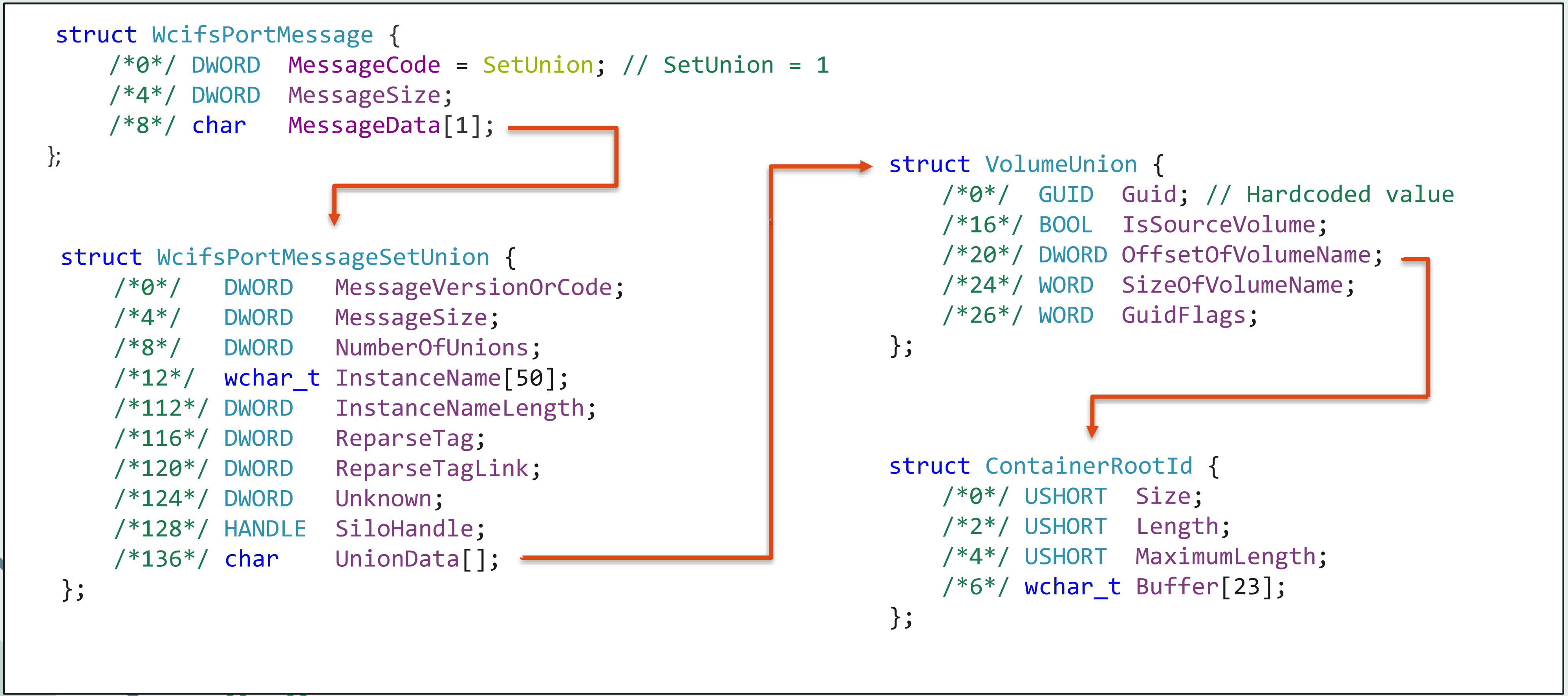
1. Create a silo and insert our process into it
2. **Inform the driver our silo is representing a container, so it will create a union context and handle it accordingly**



```
void CreateSilo(…)
{
    // Create a job
    SECURITY_ATTRIBUTES securityAttributes {};
    HANDLE jobHandle = CreateJobObjectA(&securityAttributes, "ContainYourselfJob");

    // Convert to a silo
    SetInformationJobObject(jobHandle, JobObjectCreateSilo, nullptr, 0);

    // Assign our process
    AssignProcessToJobObject(jobHandle, GetCurrentProcess());
}
```

# wcifs.sys – Creating a Fake Container

1. Create a silo and insert our process into it
2. **Inform the driver our silo is representing a container, so it will create a union context and handle it accordingly**



```
struct WcifsPortMessage {
    /*0*/ DWORD  MessageCode = SetUnion; // SetUnion = 1
    /*4*/ DWORD  MessageSize;
    /*8*/ char   MessageData[1];
};
```

```
struct WcifsPortMessageSetUnion {
    /*0*/    DWORD   MessageVersionOrCode;
    /*4*/    DWORD   MessageSize;
    /*8*/    DWORD   NumberOfUnions;
    /*12*/   wchar_t InstanceName[50];
    /*112*/  DWORD   InstanceNameLength;
    /*116*/  DWORD   ReparseTag;
    /*120*/  DWORD   ReparseTagLink;
    /*124*/  DWORD   Unknown;
    /*128*/  HANDLE  SiloHandle;
    /*136*/  char    UnionData[];
};
```

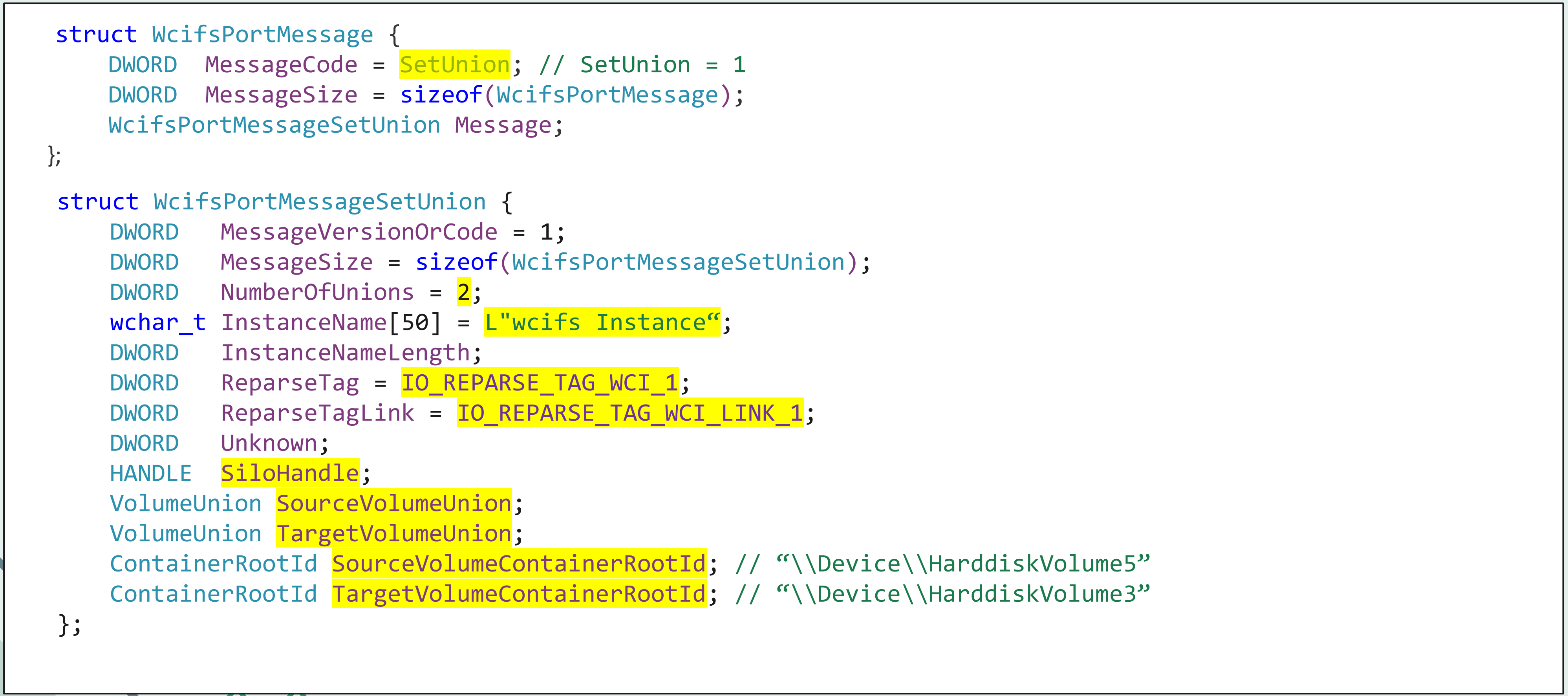
```
struct VolumeUnion {
    /*0*/  GUID  Guid; // Hardcoded value
    /*16*/ BOOL  IsSourceVolume;
    /*20*/ DWORD OffsetOfVolumeName;
    /*24*/ WORD  SizeOfVolumeName;
    /*26*/ WORD  GuidFlags;
};
```

```
struct ContainerRootId {
    /*0*/ USHORT  Size;
    /*2*/ USHORT  Length;
    /*4*/ USHORT  MaximumLength;
    /*6*/ wchar_t Buffer[23];
};
```

# wcifs.sys – Creating a Fake Container

1. Create a silo and insert our process into it

2. **Inform the driver our silo is representing a container, so it will create a union context and handle it accordingly**

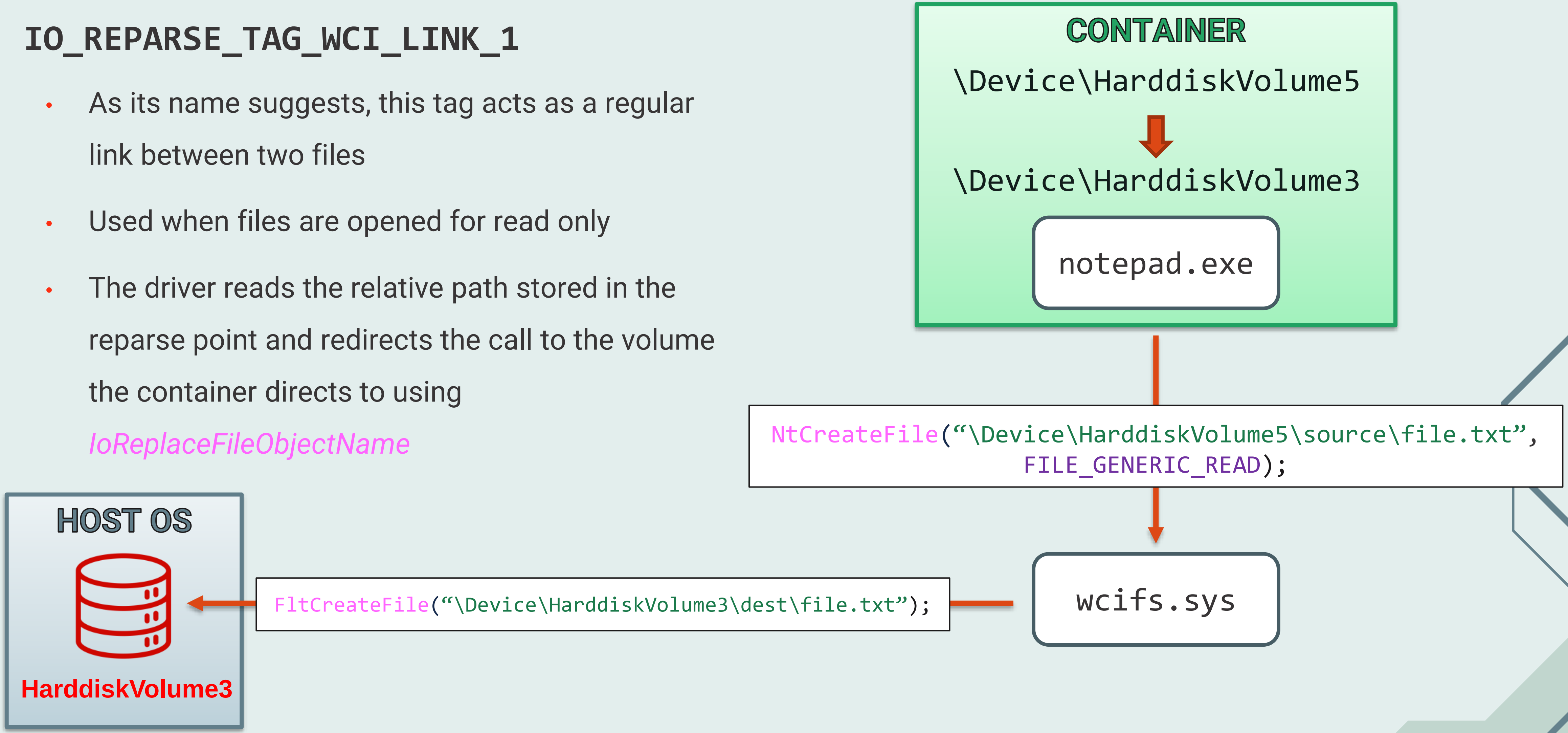


```
struct WcifsPortMessage {
    DWORD  MessageCode = SetUnion; // SetUnion = 1
    DWORD  MessageSize = sizeof(WcifsPortMessage);
    WcifsPortMessageSetUnion Message;
};

struct WcifsPortMessageSetUnion {
    DWORD   MessageVersionOrCode = 1;
    DWORD   MessageSize = sizeof(WcifsPortMessageSetUnion);
    DWORD   NumberOfUnions = 2;
    wchar_t InstanceName[50] = L"wcifs Instance“;
    DWORD   InstanceNameLength;
    DWORD   ReparseTag = IO_REPARSE_TAG_WCI_1;
    DWORD   ReparseTagLink = IO_REPARSE_TAG_WCI_LINK_1;
    DWORD   Unknown;
    HANDLE  SiloHandle;
    VolumeUnion SourceVolumeUnion;
    VolumeUnion TargetVolumeUnion;
    ContainerRootId SourceVolumeContainerRootId; // “\\Device\\HarddiskVolume5”
    ContainerRootId TargetVolumeContainerRootId; // “\\Device\\HarddiskVolume3”
};
```

# wcifs.sys – Processing Reparse Points

## `IO_REPARSE_TAG_WCI_LINK_1`

- As its name suggests, this tag acts as a regular link between two files
- Used when files are opened for read only
- The driver reads the relative path stored in the reparse point and redirects the call to the volume the container directs to using *IoReplaceFileObjectName*



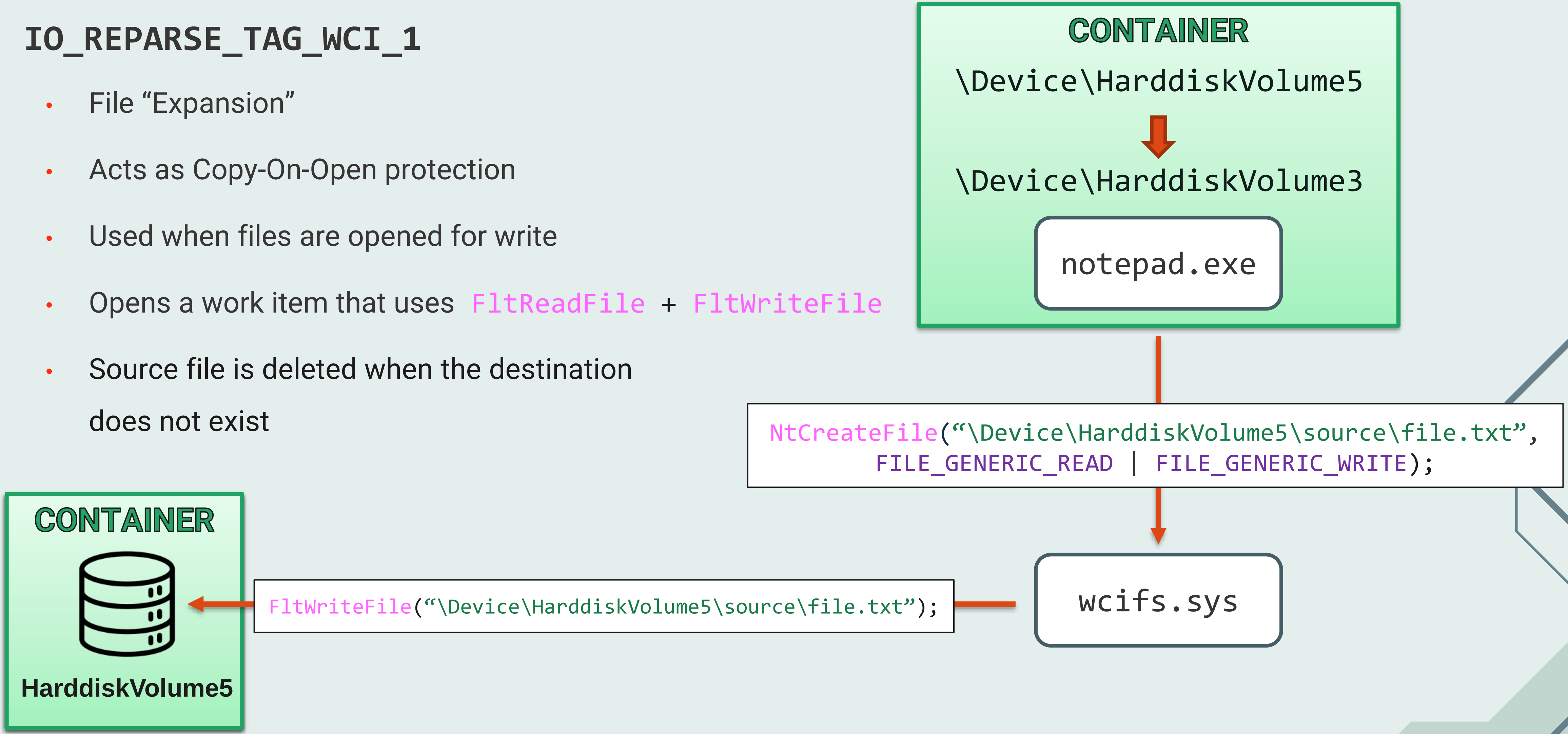
```
NtCreateFile("\Device\HarddiskVolume5\source\file.txt",
FILE_GENERIC_READ);
```

```
FltCreateFile("\Device\HarddiskVolume3\dest\file.txt");
```

# wcifs.sys – Processing Reparse Points

## `IO_REPARSE_TAG_WCI_1`

- File “Expansion”
- Acts as Copy-On-Open protection
- Used when files are opened for write
- Opens a work item that uses `FltReadFile` + `FltWriteFile`
- Source file is deleted when the destination does not exist



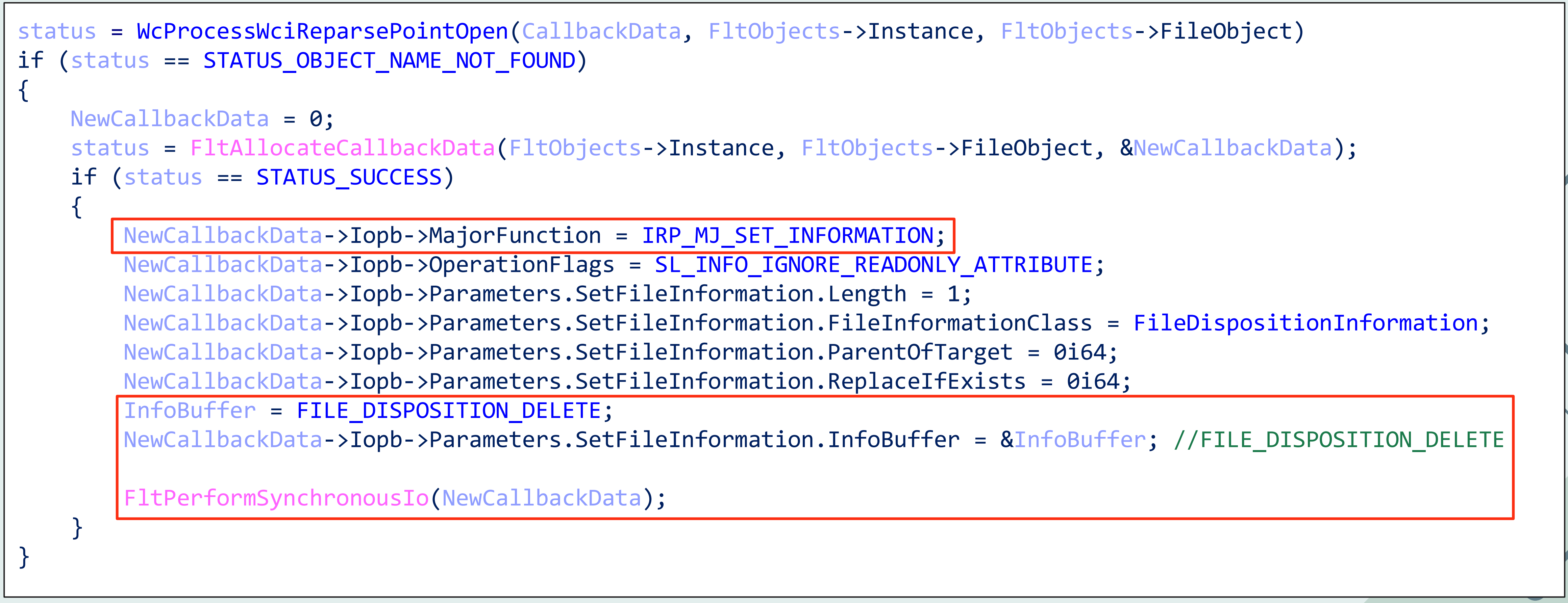
```
NtCreateFile(“\Device\HarddiskVolume5\source\file.txt”,
      FILE_GENERIC_READ | FILE_GENERIC_WRITE);
```

```
FltWriteFile(“\Device\HarddiskVolume5\source\file.txt”);
```

# wcifs.sys – Processing Reparse Points

## IO_REPARSE_TAG_WCI_1

- Source file is deleted when the destination does not exist

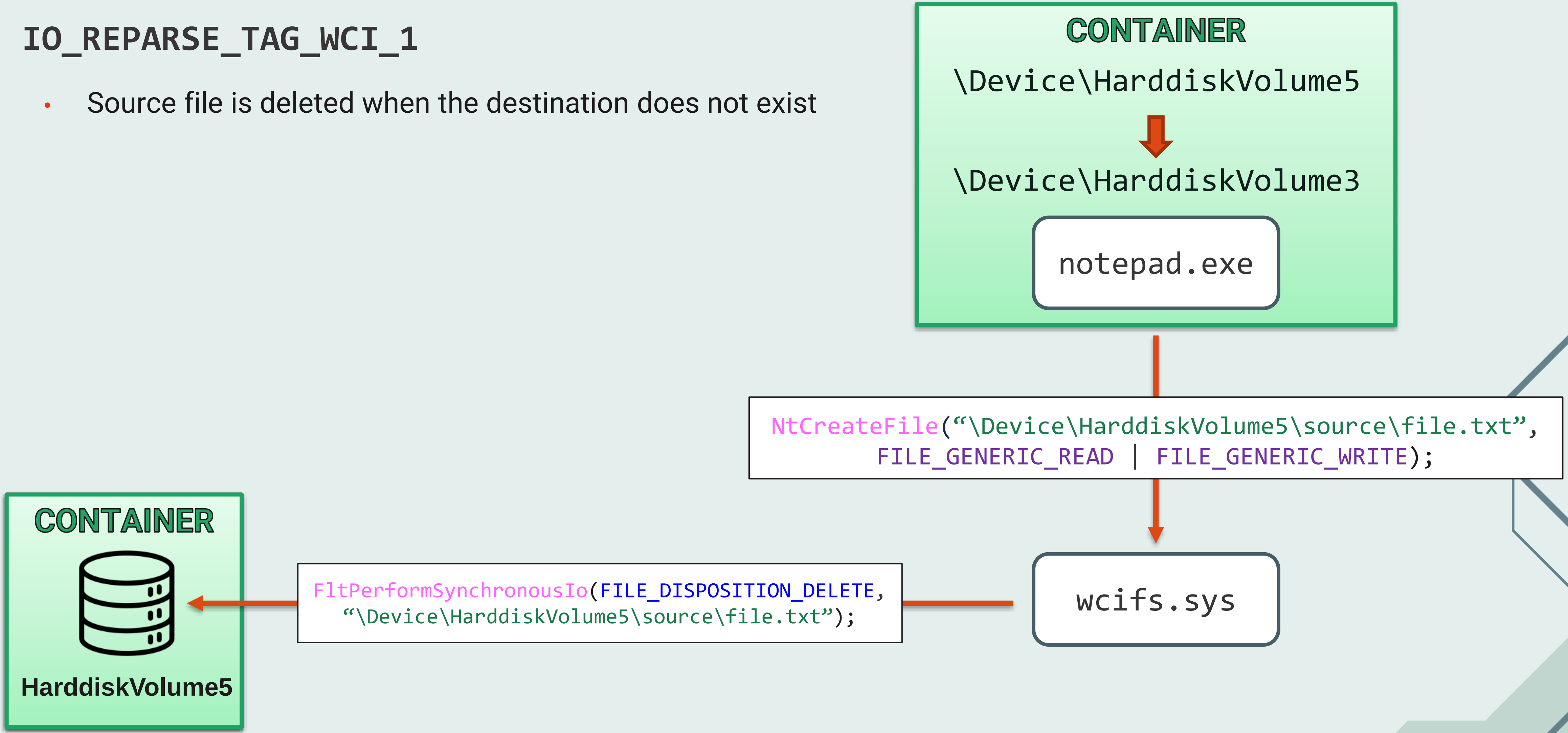


```
status = WcProcessWciReparsePointOpen(CallbackData, FltObjects->Instance, FltObjects->FileObject)
if (status == STATUS_OBJECT_NAME_NOT_FOUND)
{
    NewCallbackData = 0;
    status = FltAllocateCallbackData(FltObjects->Instance, FltObjects->FileObject, &NewCallbackData);
    if (status == STATUS_SUCCESS)
    {
        NewCallbackData->Iopb->MajorFunction = IRP_MJ_SET_INFORMATION;
        NewCallbackData->Iopb->OperationFlags = SL_INFO_IGNORE_READONLY_ATTRIBUTE;
        NewCallbackData->Iopb->Parameters.SetFileInformation.Length = 1;
        NewCallbackData->Iopb->Parameters.SetFileInformation.FileInformationClass = FileDispositionInformation;
        NewCallbackData->Iopb->Parameters.SetFileInformation.ParentOfTarget = 0i64;
        NewCallbackData->Iopb->Parameters.SetFileInformation.ReplaceIfExists = 0i64;
        InfoBuffer = FILE_DISPOSITION_DELETE;
        NewCallbackData->Iopb->Parameters.SetFileInformation.InfoBuffer = &InfoBuffer; //FILE_DISPOSITION_DELETE

        FltPerformSynchronousIo(NewCallbackData);
    }
}
```

# wcifs.sys – Processing Reparse Points

## IO_REPARSE_TAG_WCI_1

- Source file is deleted when the destination does not exist



CONTAINER

\Device\HarddiskVolume5

\Device\HarddiskVolume3

notepad.exe

```
NtCreateFile("\Device\HarddiskVolume5\source\file.txt",
    FILE_GENERIC_READ | FILE_GENERIC_WRITE);
```

wcifs.sys

```
FltPerformSynchronousIo(FILE_DISPOSITION_DELETE,
  "\Device\HarddiskVolume5\source\file.txt");
```

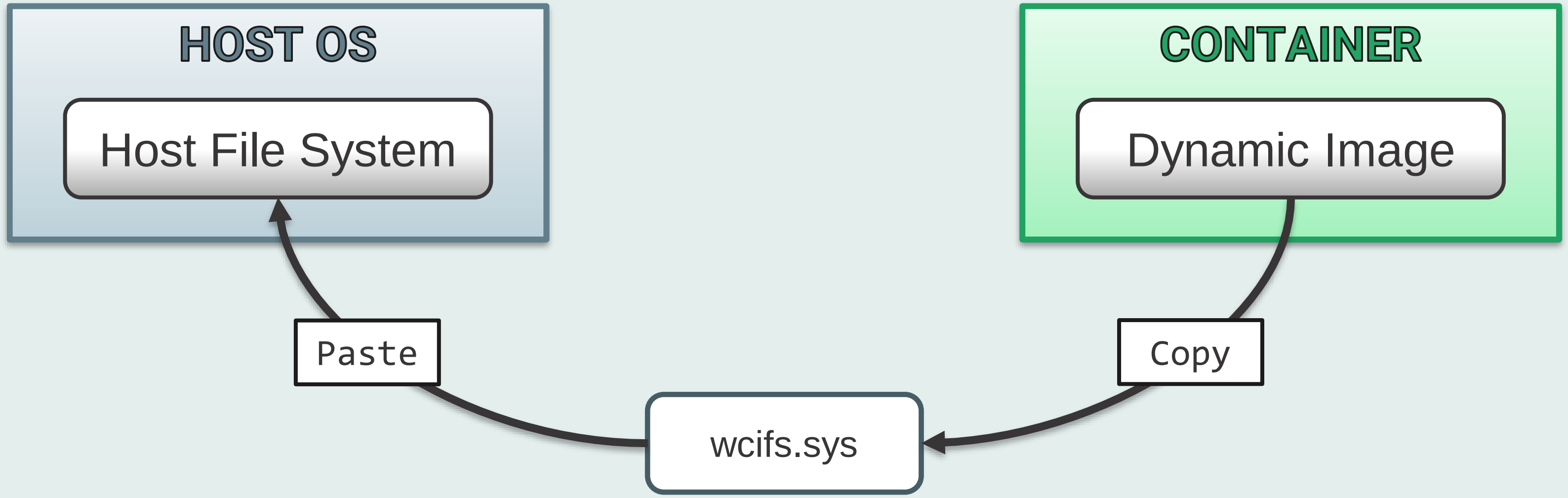
CONTAINER

HarddiskVolume5

# wcifs.sys – Other Features

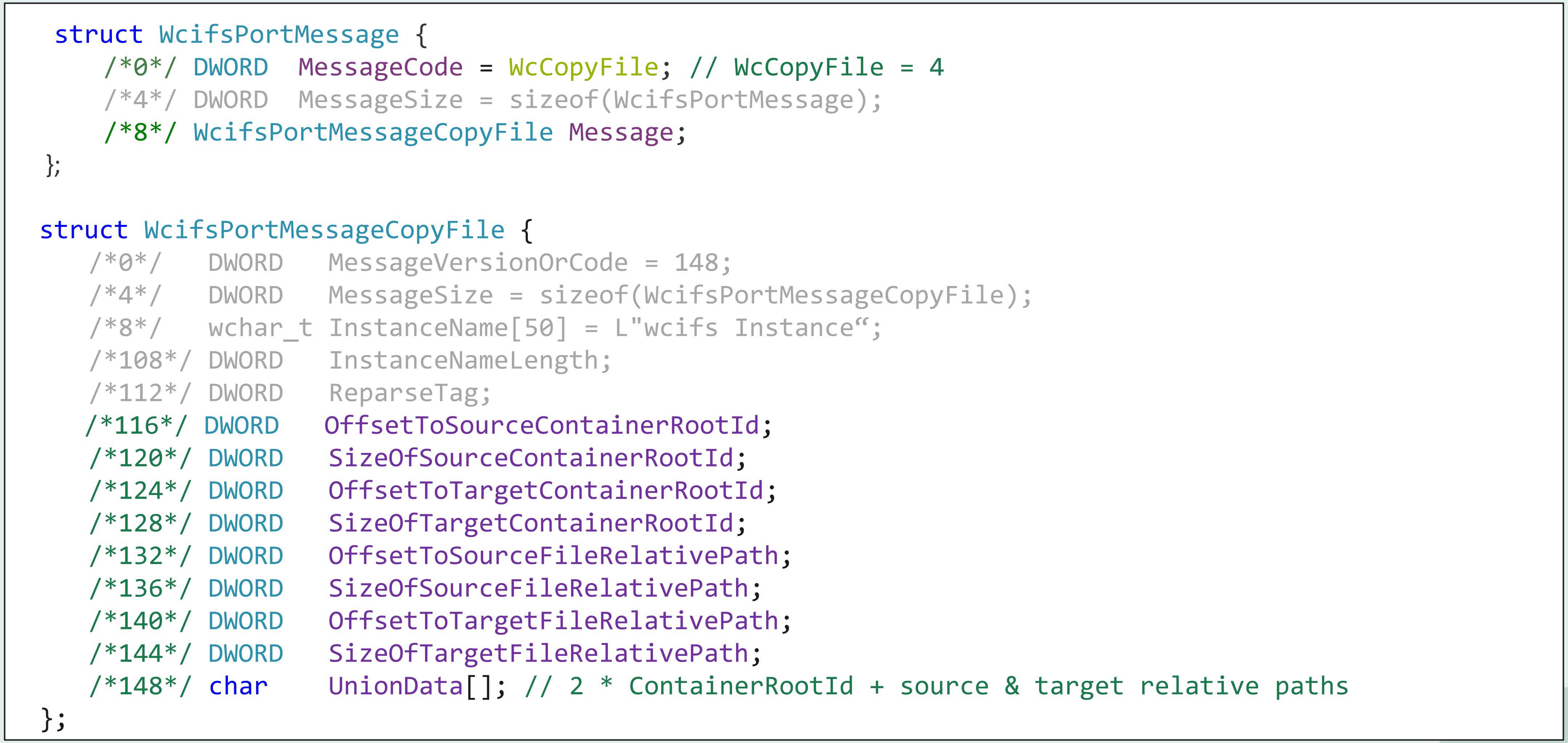
## Copy & Paste Files

- Another functionally this driver offers is to copy & paste files, **without the need to enter a container / use reparse points**
- Used when files from needs to be transferred to/from a container's volume
- `FltReadFile` + `FltWriteFile`



# wcifs.sys – Other Features

## Copy & Paste Files



```
 struct WcifsPortMessage {
    /*0*/ DWORD  MessageCode = WcCopyFile; // WcCopyFile = 4
    /*4*/ DWORD  MessageSize = sizeof(WcifsPortMessage);
    /*8*/ WcifsPortMessageCopyFile Message;
};

struct WcifsPortMessageCopyFile {
   /*0*/   DWORD   MessageVersionOrCode = 148;
   /*4*/   DWORD   MessageSize = sizeof(WcifsPortMessageCopyFile);
   /*8*/   wchar_t InstanceName[50] = L"wcifs Instance“;
   /*108*/ DWORD   InstanceNameLength;
   /*112*/ DWORD   ReparseTag;
   /*116*/ DWORD   OffsetToSourceContainerRootId;
   /*120*/ DWORD   SizeOfSourceContainerRootId;
   /*124*/ DWORD   OffsetToTargetContainerRootId;
   /*128*/ DWORD   SizeOfTargetContainerRootId;
   /*132*/ DWORD   OffsetToSourceFileRelativePath;
   /*136*/ DWORD   SizeOfSourceFileRelativePath;
   /*140*/ DWORD   OffsetToTargetFileRelativePath;
   /*144*/ DWORD   SizeOfTargetFileRelativePath;
   /*148*/ char    UnionData[]; // 2 * ContainerRootId + source & target relative paths
};
```

# wcifs.sys – Summary

## So, what have we accomplished so far?

- Create a silo, insert our process into it and register it as a fake container by communicating with the wcifs driver
    1. `IO_REPARSE_TAG_WCI_LINK_1`
        - Open one file and receive the handle of another
    2. `IO_REPARSE_TAG_WCI_1`
        - Override a file with the content of another (`FltReadFile` + `FltWriteFile`)
        - Delete a file (`FltPerformSynchronousIo`)
- Copy & paste a file (`FltReadFile` + `FltWriteFile`)

01

## Overview

Windows containers basics

02

## wcifs.sys

Analyzing the main filter driver responsable for containers FS isolation

03

## Utilization

Bypassing security products using our findings

04

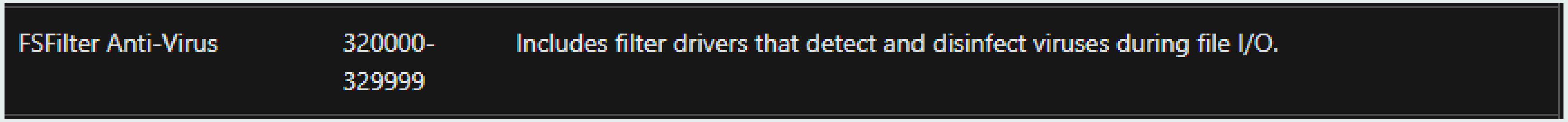
## Summary

Summary, mitigation and future research

# Security Products Mini-filter Bypass

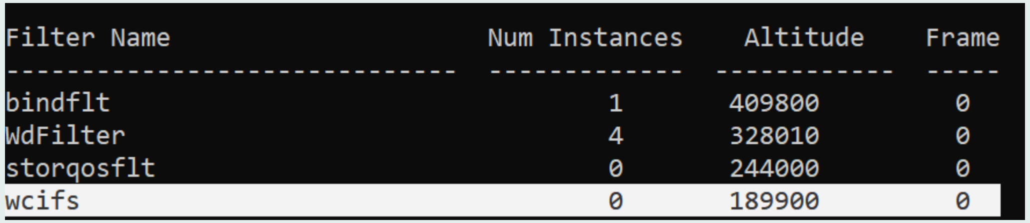
- Security products use mini-filters to monitor I/O activity
- `FltReadFile`, `FltWriteFile`, `FltPerformSynchronousIo`
- > “ **[function]** causes the request to be sent to the mini-filter driver instances attached below the initiating instance and to the file system. The specified instance and the Instances attached above it do not receive the request.” - MSDN

  https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/fltkernel/nf-fltkernel-fltreadfile

- 

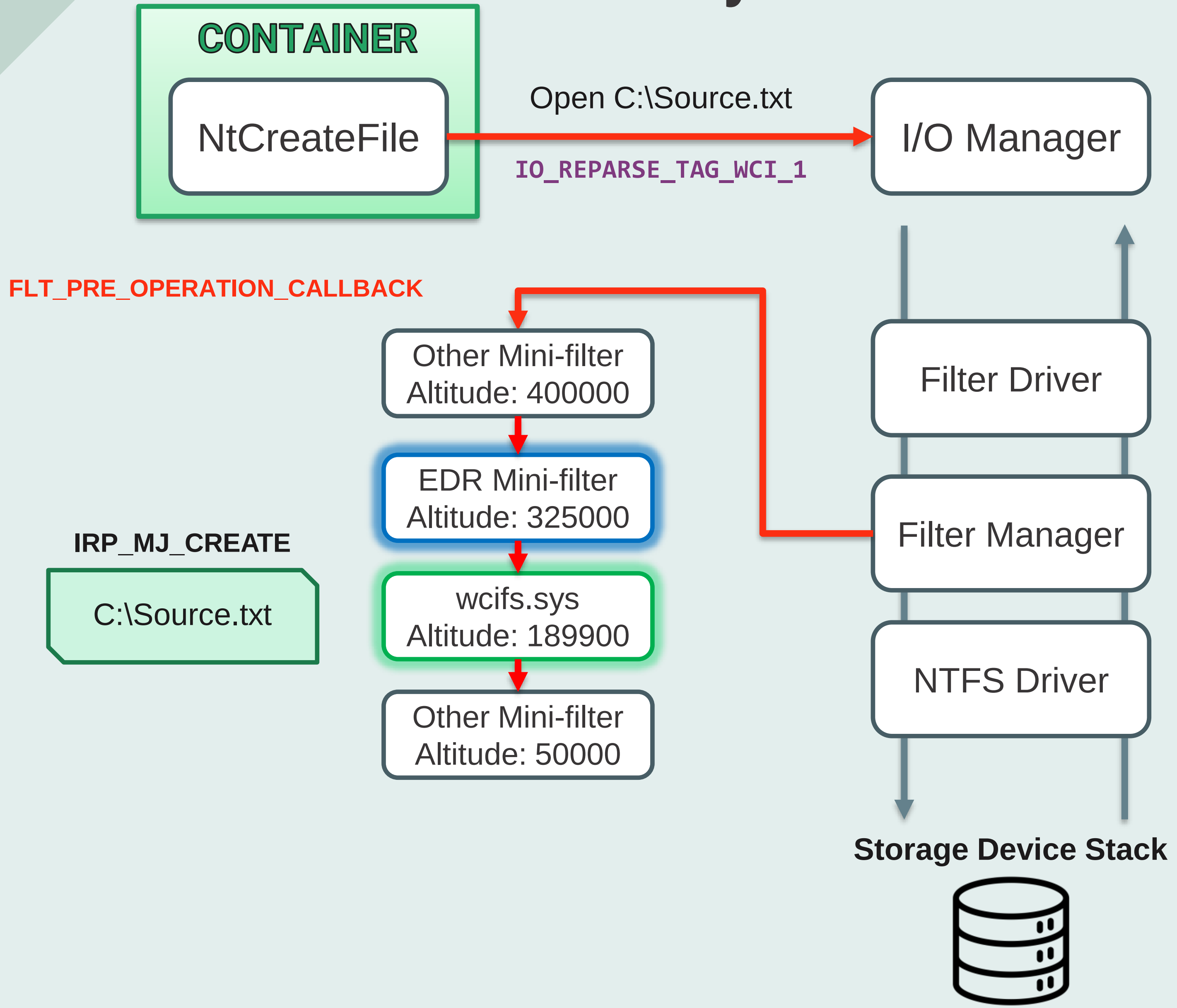
  |  |  |  |
  |---|---|---|
  | FSFilter Anti-Virus | 320000-<br>329999 | Includes filter drivers that detect and disinfect viruses during file I/O. |

  https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/load-order-groups-and-altitudes-for-minifilter-drivers



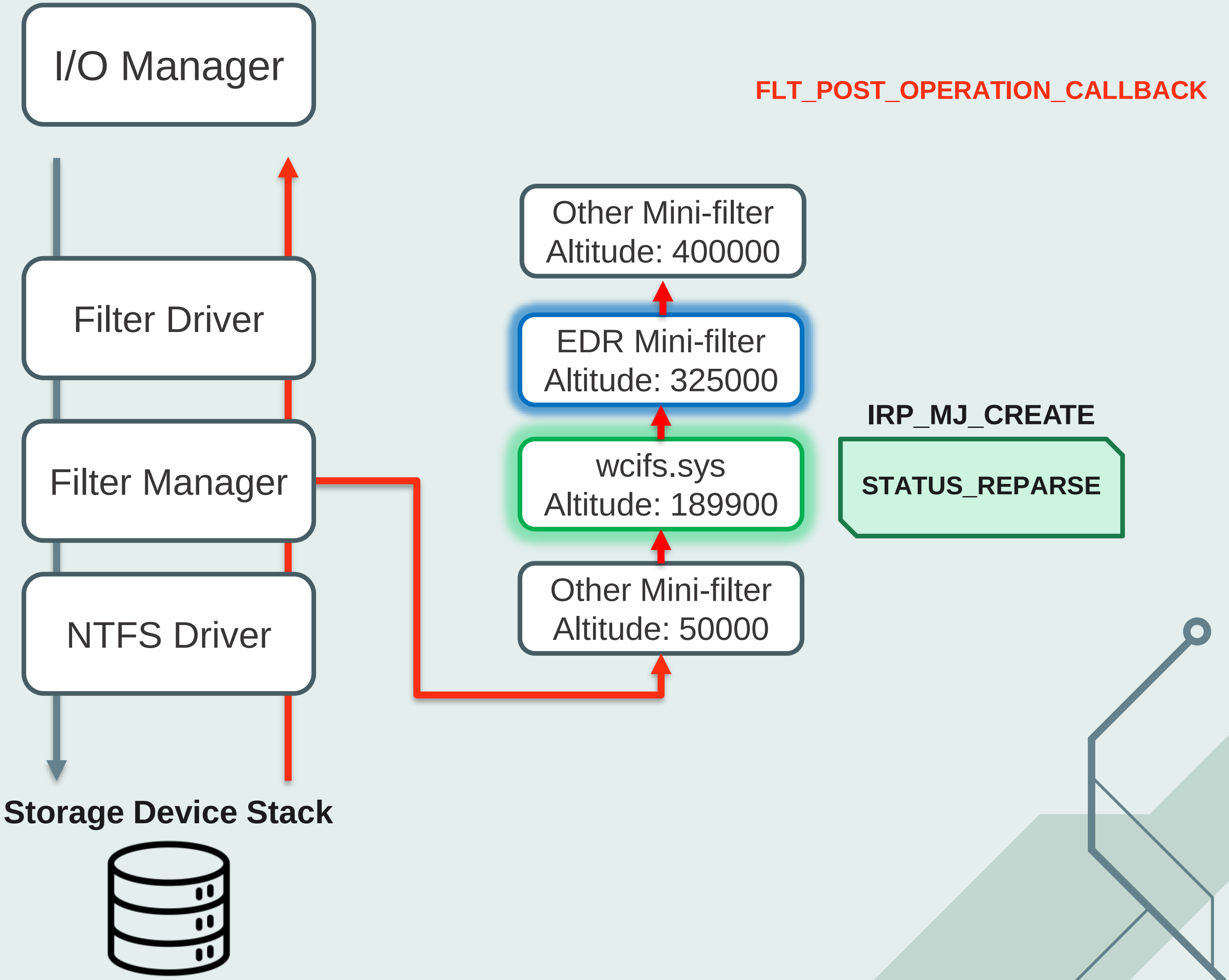
```
Filter Name                     Num Instances    Altitude    Frame
------------------------------  -------------  ------------  -----
bindflt                                 1       409800         0
WdFilter                                4       328010         0
storqosflt                              0       244000         0
wcifs                                   0       189900         0
```

# Security Products Mini-filter Bypass

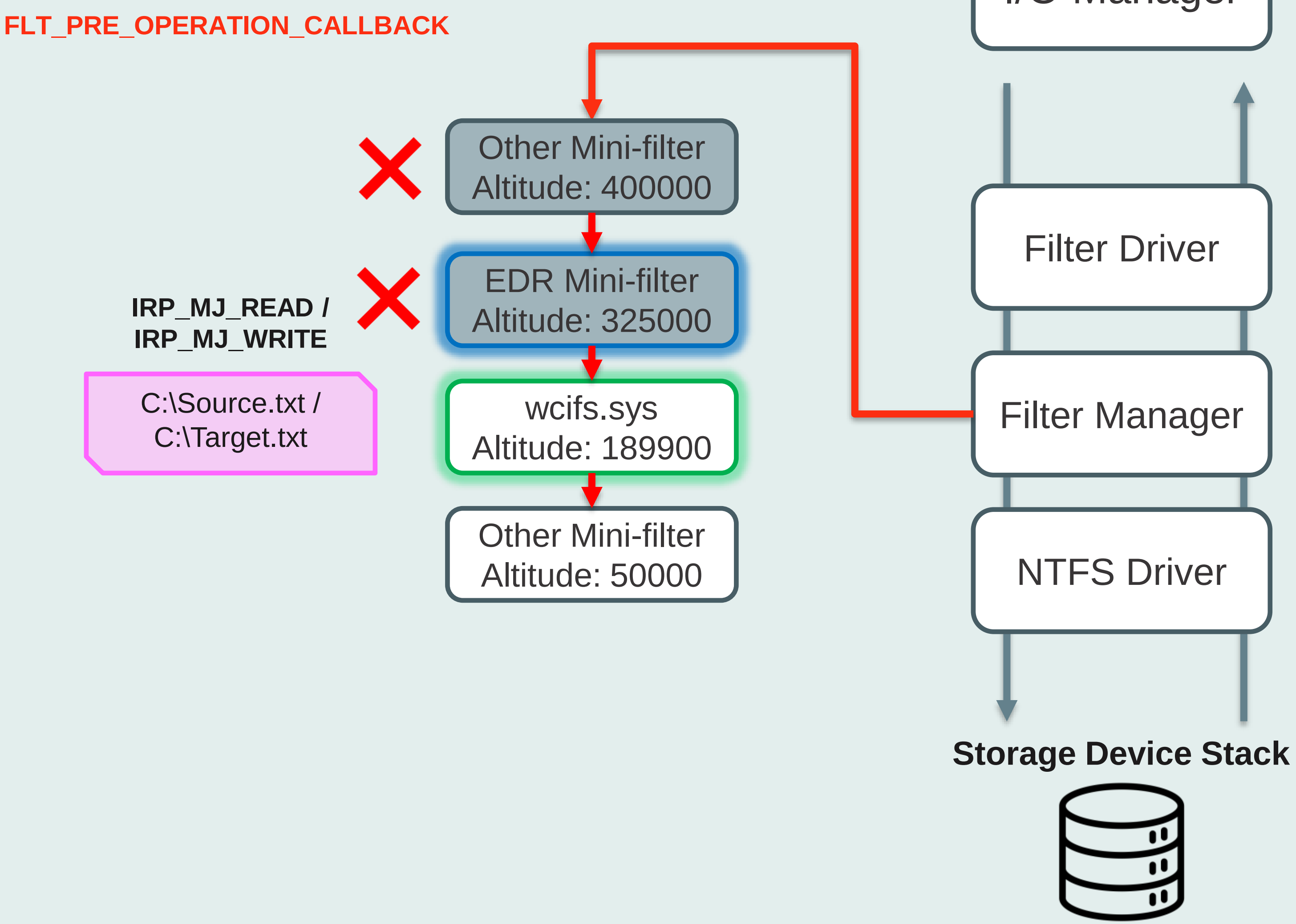


# Security Products Mini-filter Bypass

```
ReparsePointData =
{
  IO_REPARSE_TAG_WCI_1;
  MInifilter2ReparseDataLen;
  MInifilter2ReparseData
  {
    TargetPath = “C:\\Target.txt”;
  };
}

FltReadFile(“C:\\Source.txt”);
FltWriteFile(“C:\\Target.txt”);
OR
FltPerformSynchronousIo(
FILE_DISPOSITION_DELETE,
“C:\\Source.txt”);

return FLT_POSTOP_FINISHED_PROCESSING;
```



# Security Products Mini-filter Bypass



FLT_PRE_OPERATION_CALLBACK

IRP_MJ_READ /
IRP_MJ_WRITE

C:\Source.txt /
C:\Target.txt

Other Mini-filter
Altitude: 400000

EDR Mini-filter
Altitude: 325000

wcifs.sys
Altitude: 189900

Other Mini-filter
Altitude: 50000

I/O Manager

Filter Driver

Filter Manager

NTFS Driver

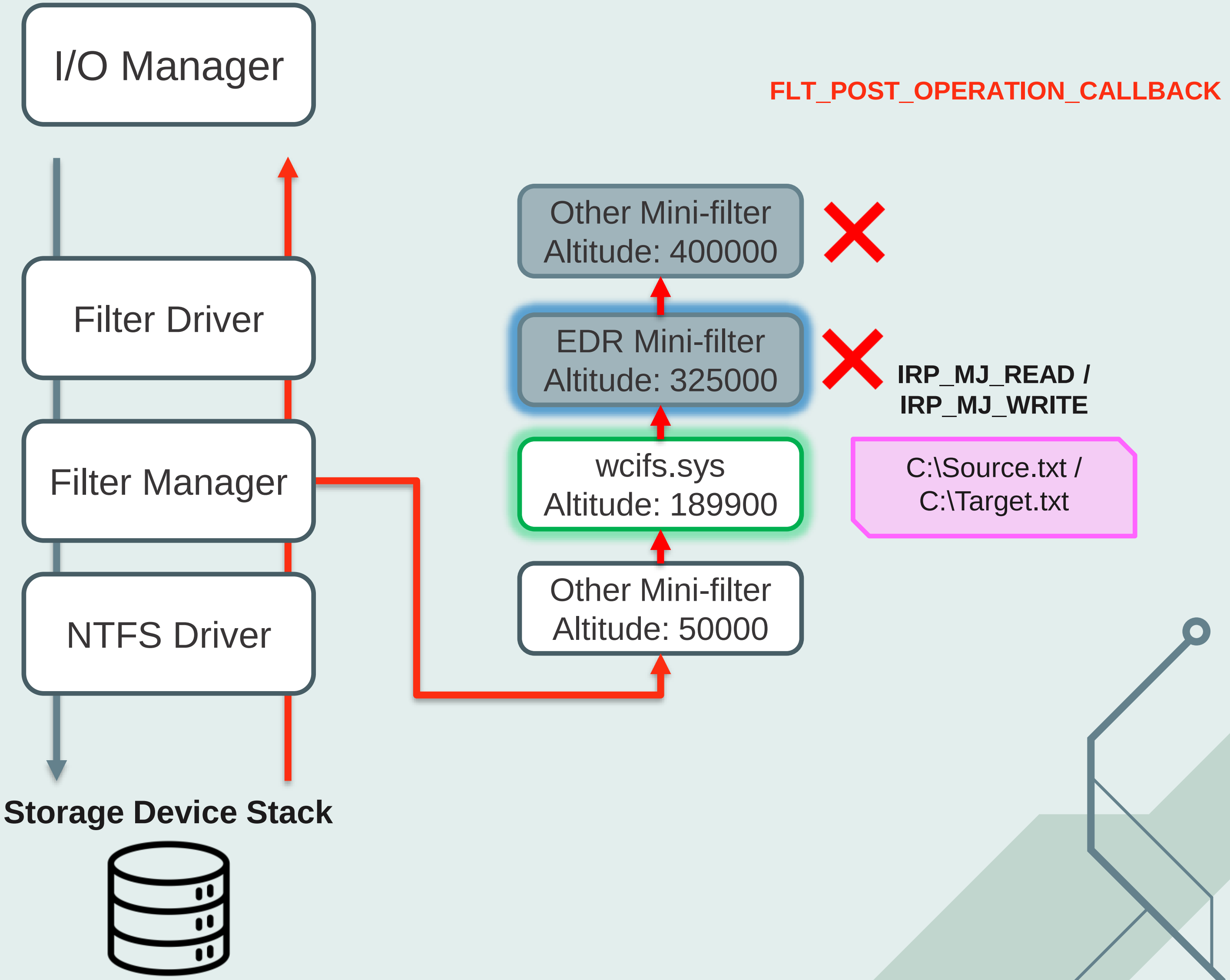
Storage Device Stack

```
FltReadFile(“C:\\Source.txt”);
FltWriteFile(“C:\\Target.txt”);
OR
FltPerformSynchronousIo(
FILE_DISPOSITION_DELETE,
“C:\\Source.txt”);
```

“The specified instance and the Instances attached above it **do not receive** the request.”

# Security Products Mini-filter Bypass

```
ReparsePointData =
{
  IO_REPARSE_TAG_WCI_1;
  MInifilter2ReparseDataLen;
  MInifilter2ReparseData
  {
    TargetPath = “C:\\Target.txt”;
  };
}

FltReadFile(“C:\\Source.txt”);
FltWriteFile(“C:\\Target.txt”);
OR
FltPerformSynchronousIo(
FILE_DISPOSITION_DELETE,
“C:\\Source.txt”);

return FLT_POSTOP_FINISHED_PROCESSING;
```



I/O Manager
Filter Driver
Filter Manager
NTFS Driver
Storage Device Stack
FLT_POST_OPERATION_CALLBACK
Other Mini-filter Altitude: 400000
EDR Mini-filter Altitude: 325000
IRP_MJ_READ / IRP_MJ_WRITE
wcifs.sys Altitude: 189900
C:\Source.txt / C:\Target.txt
Other Mini-filter Altitude: 50000

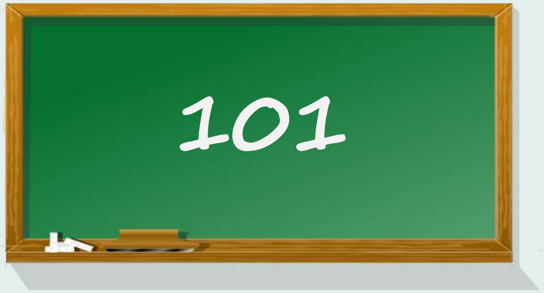
# Security Products Mini-filter Bypass

## EDR 101: Ransomware/Wiper Protection

- Security products employ algorithms that analyze mini-filter I/O logs, searching for specific patterns to detect file system-based malware and prevent them before any irreversible damage is done
- Most EDRs rely on a set of standard principles to categorize a process as ransomware/wiper:

  I. Process opens handles to a vast number of files

  II. Process reads data from a file and then writes **to the same file**, making the file's data inaccessible (using pre read/write callbacks)



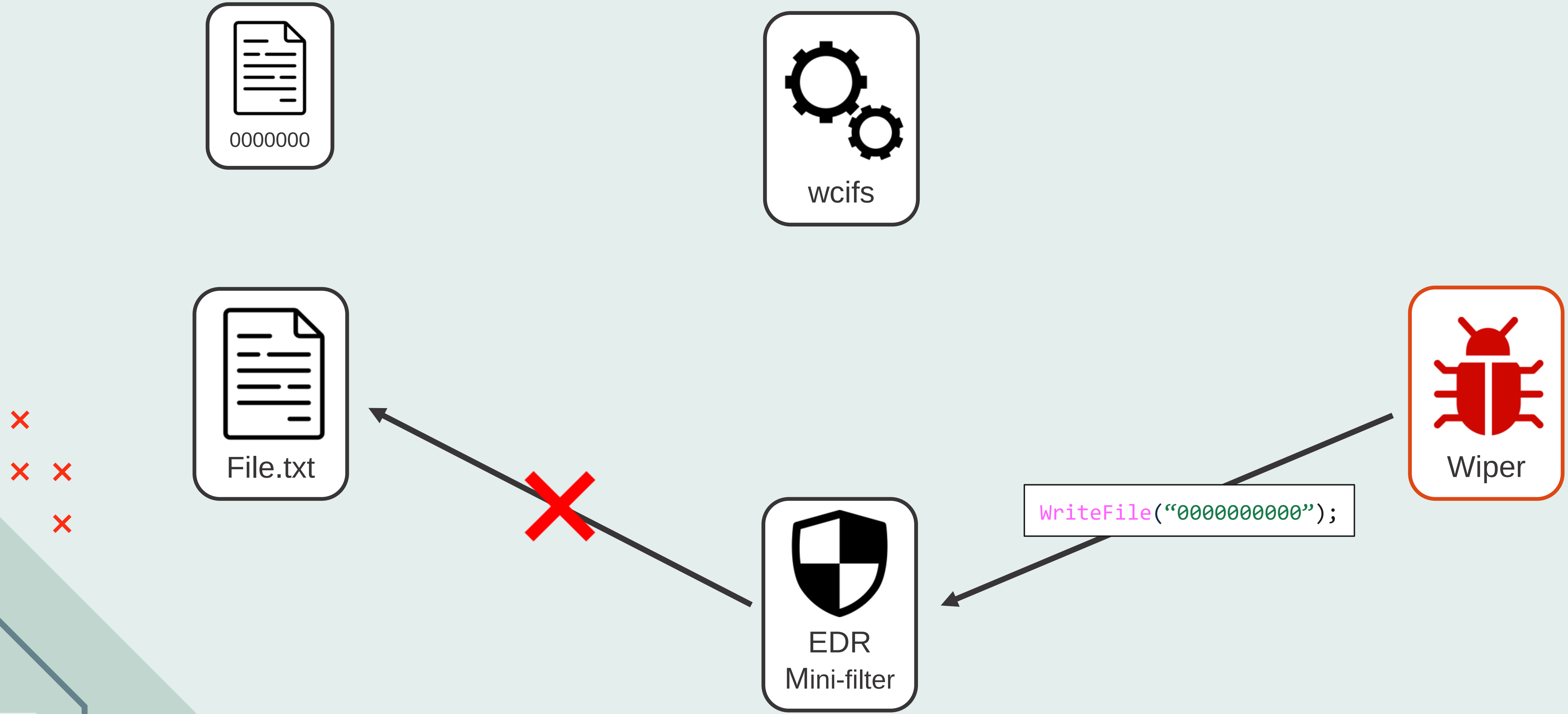
# Security Products Mini-filter Bypass

## Creating an Undetectable Wiper

1. Create an empty file that will be our target file. Write a buffer of zeros/random data to it.
2. Traverse each file on the system and for each:
    1. Set an `IO_REPARSE_TAG_WCI_1` reparse point on the source file that will point to the target file
3. Create a silo, assign the current process to it and register it as a fabricated container to wcifs where both source and target volumes are the main one (`\Device\HarddiskVolume3`).
4. Traverse each file on the system and for each:
    1. Open the file using `CreateFile` – the files will be overridden with the target file data by the wcifs driver, **the call will not trigger security mini-filter drivers callback function**

# Security Products Mini-filter Bypass

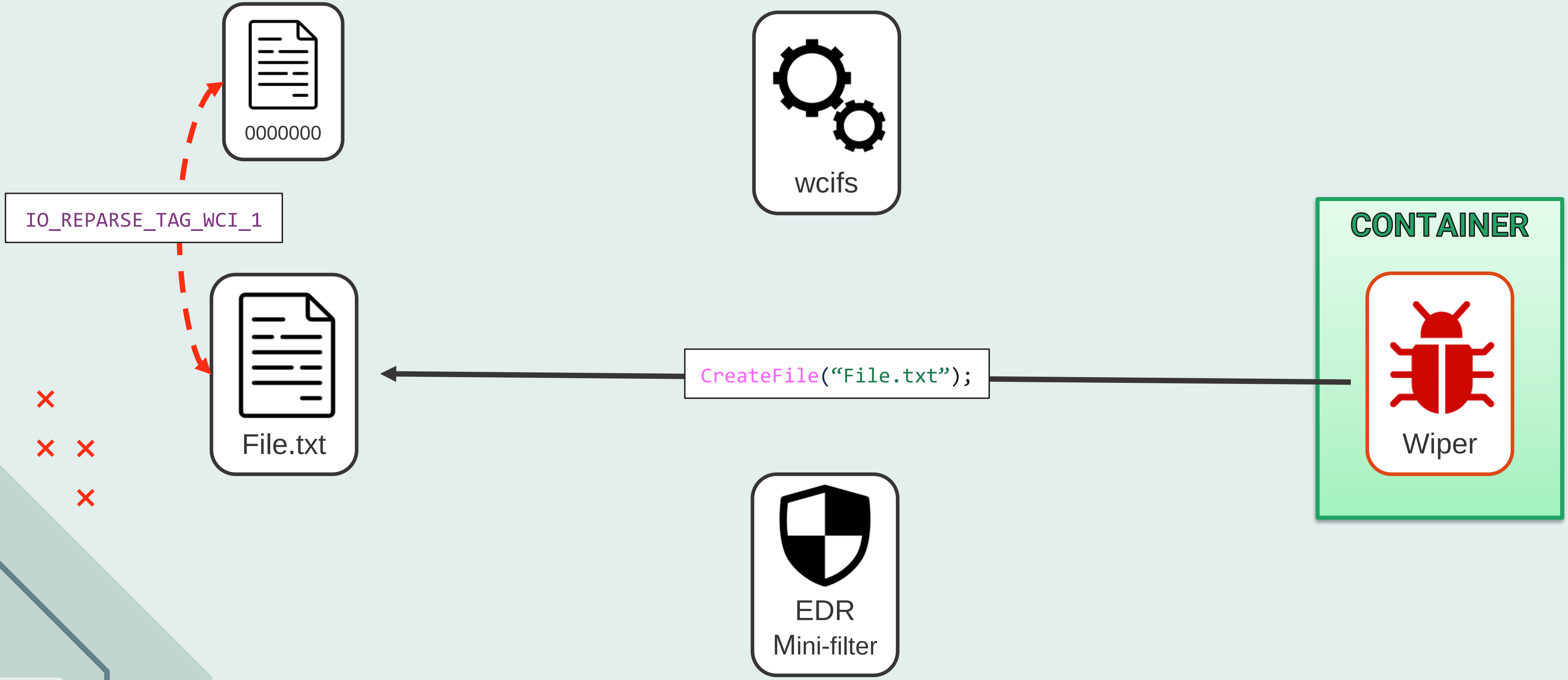
## Creating an Undetectable Wiper



```
WriteFile(“0000000000”);
```

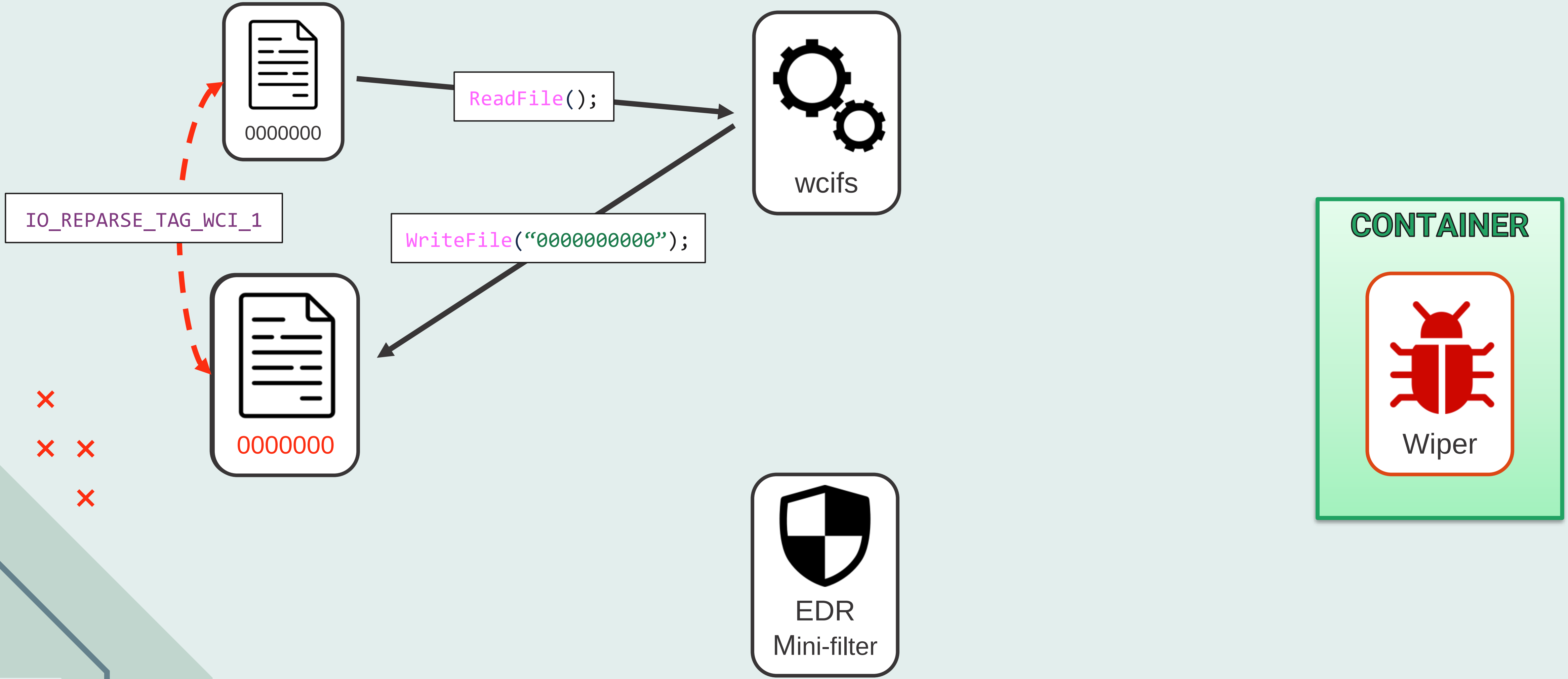
# Security Products Mini-filter Bypass

## Creating an Undetectable Wiper



# Security Products Mini-filter Bypass

## Creating an Undetectable Wiper



# Security Products Mini-filter Bypass

## Creating an Undetectable Ransomware

1. Traverse each file on the system and for each:
   1. Read its content and encrypt in-memory
   2. Create a target file and write the encrypted data to it – **will be ignored by security mini-filter because the data is written to a new file and not overriding existing content**
   3. Set an `IO_REPARSE_TAG_WCI_1` reparse point on the source file that will point to the target file
2. Create a silo, assign the current process to it and register it as a fabricated container to wcifs where both source and target volumes are the main one (`\Device\HarddiskVolume3`).
3. Traverse each file on the system and for each:
   1. Open the file using `CreateFile` – the wcifs driver will write the encrypted content to the original file, **the call will not trigger security mini-filter drivers callback function**

# Security Products Mini-filter Bypass

## EDR 102: DLP Protection – Set Read-Only Devices & Directories

- Security vendor’s products can block write operations on certain directories/volumes, which can be utilized in several ways:
  1. Organizations often determine a read-only policy for removable devices to avoid data exfiltration
  2. Block file writes to folders containing sensitive data
- This write protection is implemented by a mini-filter driver

# Security Products ETW Bypass

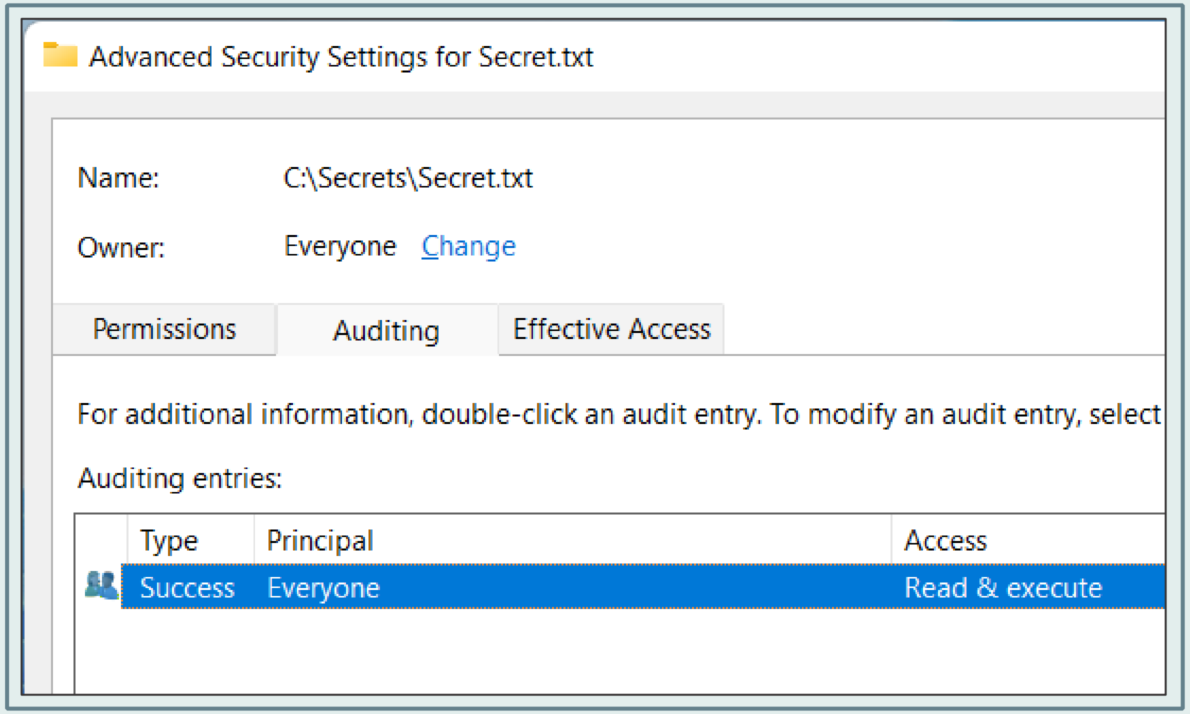
## EDR 103: Correlating ETW Logs

- The Windows kernel acts as a log provider of a vast number of operations occurring on the system, including those on the file system
- Vendors often consume and analyze these events for any suspicious activity (usually by cross-referencing and creating an attack flow)
- When overriding a file using the `IO_REPARSE_TAG_WCI_1` tag, the read and write operations take place within a kernel work item. This will cause the ETW log to mention the system process (PID 4) as responsible for them instead of the actual process.
- Any vendor who consumes events number 15 (`Read`) and 16 (`Write`) from the `Microsoft-Windows-Kernel-File` provider will receive false information

# Security Products ETW Bypass

## SACL Bypass

- Windows provides the option to set an auditing policy (SACL) to a file system object, which can yield a log of any I/O operation done on it
- By design, ETW-based Windows tools ignore logs that originate from the system because they should be irrelevant to a user monitoring the system (and to avoid unnecessary overhead)
- This will cause our I/O requests to be absent from the auditing logs



Advanced Security Settings for Secret.txt

Name: C:\Secrets\Secret.txt

Owner: Everyone Change

For additional information, double-click an audit entry. To modify an audit entry, select

Auditing entries:

| Type | Principal | Access |
|---|---|---|
| Success | Everyone | Read & execute |

01

## Overview

Windows containers basics

02

## wcifs.sys

Analyzing the main filter driver responsable for containers FS isolation

03

## Utilization

Bypassing security products using our findings

04

## Summary

Summary, mitigation and future research

# Summary

- The Windows containers framework provides a file system isolation solution that is implemented by reparse points and mini-filter drivers
- By reverse engineering the framework's main driver – wcifs.sys, we managed to create a fake container, insert our process into it and utilize the framework's I/O redirection mechanism to our advantage
  - I. Override files
  - II. Delete files
  - III. Copy & paste files
- **This allows us to perform file system calls that will not be detected by security vendors' mini-filters, and ETW-based products**
  - **Ransomware/Wiper protection bypass**
  - **DLP/Secured folders write bypass**
  - **ETW-based correlations bypass**

# Summary

## Mitigation

- `DeviceIoControl` + `FSCTL_SET_REPARSE_POINT` + `IO_REPARSE_TAG_WCI` – files with these tags should already exist on container's ghost volumes
- Check whether wcifs communication port was opened / a silo is created by a non-system process
- Check if the wcifs driver is attached to a volume while the containers feature is turned off

# Summary

## Further Research

- The wcifs driver is only one mini-filter among many, there might be more that can be utilized to perform system operations from the kernel itself
- It is possible to set reparse points on directories. Wcifs‘s symbols reference directory expansion and redirection handling
- **There are more mini-filter-based and ETW-based protections implemented by security products!**

# THANK YOU!

**Do you have any questions?**

@daniel_avinoam

https://github.com/deepinstinct/ContainYourself

**References:**

- Playing in the (Windows) Sandbox - Check Point Research
- Who Contains the Containers?
- https://unit42.paloaltonetworks.com/what-i-learned-from-reverse-engineering-windows-containers/
- NTFS Reparse Points
- About Windows containers
- Windows Kernel Programming, by Pavel Yosifovich