FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Windows Sandbox: Analysis and Verification of Known Vulnerabilities
Student:	Bc. Jakub Štrom
Supervisor:	Ing. Josef Kokeš, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2024/2025

Instructions

1) Describe the features and properties of the Windows Sandbox technology.
2) Research known CVEs related to the Windows Sandbox.
3) Attempt to match the vulnerabilities to specific patches.
4) In cooperation with the supervisor, select several vulnerabilities and verify that the patch is a dependable solution to the vulnerability.
5) If the previous step reveals a potential for a new or extended vulnerability, analyze it further (optimally, reach the "proof-of-concept" stage).
6) Discuss your results.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 15 February 2024 in Prague.

Master's thesis

WINDOWS SANDBOX: ANALYSIS AND VERIFICATION OF KNOWN VULNERABILITIES

Bc. Jakub Štrom

Faculty of Information Technology
Department of Information Security
Supervisor: Ing. Josef Kokeš, Ph.D.
May 9, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Štrom Jakub. *Windows Sandbox: Analysis and Verification of Known Vulnerabilities*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of code listings

I would like to thank my supervisor Ing. Josef Kokeš, Ph.D. for his guidance. I also want to thank my family for their support during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Section 2373(2) of Act No. 89/2012 Coll., the Civil Code, as amended, I hereby grant a non-exclusive authorization (licence) to utilize this thesis, including all computer programs that are part of it or attached to it and all documentation thereof (hereinafter collectively referred to as the "Work"), to any and all persons who wish to use the Work. Such persons are entitled to use the Work in any manner that does not diminish the value of the Work and for any purpose (including use for profit). This authorisation is unlimited in time, territory and quantity.

In Prague on May 9, 2024

Abstract

The subject of this thesis is Windows Sandbox, a feature introduced to the Windows operating system towards the end of 2018. It builds upon the established technology of Windows Containers, sharing its core components used for isolation. Initially, this thesis explores the internals of the feature, especially focusing on the storage implementation and its isolation. Both Windows 11 and Windows 10 variants are discussed, highlighting their differences. Subsequently, it gathers and examines known vulnerabilities associated with the Windows Sandbox and its components, focusing primarily on the effectiveness of the patches in addressing the root causes of these vulnerabilities. This analysis yielded a discovery of a new vulnerability still present in the fully updated version of Windows 11 that is similar to one of the vulnerabilities discussed in the thesis.

Keywords Windows Sandbox, Windows Containers, vulnerability, patch diffing

Abstrakt

Zaměření této práce je Windows Sandbox, funkcionalita představená v operačním systému Windows koncem roku 2018. Tato funkcionalita staví na již zavedené technologii Windows Containers, s kterou sdílí klíčové komponenty využívané pro izolaci svého prostředí. Nejprve jsou prozkoumány implementační detaily funkcionality, zejména se zaměřením na implementaci úložiště a jeho izolace. Zde jsou rozebírány jak Windows 11, tak Windows 10 varianty této funkcionality a jsou porovnány jejich rozdíly. Dále jsou shromážděny a prozkoumány veřejně známé zranitelnosti související s Windows Sandbox a s jednotlivými komponentami funkcionality. Zaměření této analýzy je především na účinnost oprav a jak dobře opravy odstraňují hlavní příčiny zranitelností. Výsledkem této analýzy je nalezení nové zranitelnosti stále přítomné na plně aktualizované verzi Windows 11, která je podobná jedné z rozebíraných zranitelností v této práci.

Klíčová slova Windows Sandbox, Windows Containers, zranitelnost, patch diffing

Acronyms

API Application Programming Interface. 9, 10, 14–16, 25, 36–40

BindFlt Windows Bind minifilter driver. 12–15, 25, 26, 35, 44, 45

CExecSvc Container Execution Service. 16, 17, 48

CLI Command Line Interface. 17, 27, 32

CmService Container Manager Service. 20, 21, 23, 25–27, 35–39, 42–46, 49

CPU Central Processing Unit. 6, 7

CVE Common Vulnerabilities and Exposures. 29, 30, 32, 43, 44, 46, 48, 49

CVSS Common Vulnerability Scoring System. 29, 30, 35

DACL Discretionary Access Control List. 25, 36, 38, 43, 44, 46

EoP Escalation of Privileges. 29, 38, 39, 42, 44, 46, 49

GUI Graphical User Interface. 19

GUID Globally Unique Identifier. 21–23, 25, 40

HCS Host Compute Service. 15–17, 25, 26, 35, 37, 42, 43

IO Input/Output. 10, 13, 14, 25, 45

IRP IO Request Packet. 12

JID Job Identifier. 9, 25

JSON JavaScript Object Notation. 16, 25, 26, 42

LCOW Linux containers on Windows. 4

MSRC Microsoft Security Response Center. 29, 30

NSA National Security Agency. 31

NT New Technology. 8, 9

NTFS NT File System. 13

OCI Open Container Initiative. 4

OMNS Object Manager Namespace. 4, 8, 26, 37, 39, 40, 43

OS Operating System. 1, 3, 4, 6, 7, 9, 10, 16, 19–21, 23, 25, 29, 32, 33

PBL Portable Base Layer. 21

PID Process Identifier. 9

PoC Proof of Concept. 30, 36, 38, 41, 43

RCE Remote Code Execution. 48

RDP Remote Desktop Protocol. 27

RPC Remote Procedure Call. 16, 17, 25, 35, 36, 38, 40

SCSI Small Computer System Interface. 23

SD Security Descriptor. 8, 39, 43, 44

SID Security Identifier. 36, 40

SMB Server Message Block. 25, 37

SRM Security Reference Monitor. 8

syscall system call. 10

TCB Trusted Computing Base. 7–9, 47, 48

TID Thread Identifier. 9

UAC User Account Control. 6

UVM Utility Virtual Machine. 4–7, 19, 22, 29, 35

UWP Universal Windows Platform. 7, 35

VA Virtual Address. 6

VHD Virtual Hard Disk. 16

VHDx Virtual Hard Disk v2. 21–23, 25, 44

VM Virtual Machine. 1, 4, 6, 7, 15, 16, 21, 23, 38

Vmcompute Virtual Machine Compute. 7, 26

VMWP Virtual Machine Worker Process. v, 24, 25, 37, 38, 43

vSMB virtual Server Message Block. 23, 25, 37, 42, 43

Wcifs Windows Container Isolation minifilter driver. 12–14, 23, 25, 26, 31, 35, 46

WCOW Windows containers on Windows. 4

WDAG Windows Defender Application Guard. 20

WIM Windows Imaging Format. 20, 21, 32, 43

XML Extensible Markup Language. 20

Introduction

Given the prevalence of various malicious actors in today's digital space, environments that can isolate execution from host system prove to be valuable, especially in case of execution of untrusted applications. One example of such security mechanism is Windows Sandbox. It is a feature included in Windows Operating System (OS) since the end of 2018 when it was introduced in the preview build 18305.

Windows Sandbox is a temporary and self-contained environment that simulates the entire OS with a state that is not persisted after closing the active session. It is built upon an underlying technology of Windows Containers that predate the newer Sandbox feature. It shares the core components with this earlier technology. The environment itself is contained in a lightweight Virtual Machine (VM), where the goal is to combine security of isolation of VM and the comfort of faster startup and user convenience from being a system built-in feature.

Although itself a security mechanism, it can introduce insecurities, just like any other system feature, by having a new attack surface of its own. Whether it is bypassing the isolation of the Sandbox from the host system or the behavior of the management of the Sandbox on the host side, the feature offers opportunities for abuse.

The primary objective of this thesis is to analyze the publicly disclosed vulnerabilities associated with the Windows Sandbox. A secondary goal is expanding upon the relatively sparse documentation available for the Windows Sandbox internals, particularly the changes introduced with Windows 11.

For the vulnerability analysis, the initial focus is on correlating known vulnerabilities with their corresponding patches. After that, based on the patches, the aim is to assess the effectiveness of each patch in addressing the root causes of the vulnerabilities. If that is not the case, the goal is then to try to identify any variant or further possibilities for an exploitation of the same issue.

The initial chapter provides a description of Windows Containers and introduces the main components that are later discussed in context of the Windows Sandbox. The subsequent chapter explores the implementation of the Windows Sandbox itself. Following this, a general chapter outlines the methods and tools used for collecting and analyzing vulnerabilities. Chapter four is dedicated to the analysis of these vulnerabilities, discussing their specifics, their patches and the effectiveness of the patches.

Chapter 1

Windows Containers

This chapter introduces containers and their history in the context of Windows OS. Then it describes what types of Windows Containers are available, their architecture and how is the technology implemented by the OS.

1.1 Containers

Containers revolutionize application deployment by encapsulating an application and its dependencies into a single portable package known as an image. These images are built using layers, where each layer represents a specific component or configuration of the application. This layered approach enables efficient sharing of common components across multiple containers, leading to reduced storage overhead and faster deployment times. In Docker each layer is created by an execution of a command during build. These layers can then be reused across containers if multiple containers depend on the same layer. [1, Chapter 1]

Furthermore, containers utilize the host operating system's kernel, eliminating the need for a separate guest operating system for each application instance. This shared kernel architecture makes containers a lightweight and resource-efficient environment that can be created or removed quickly, scaling effortlessly to meet fluctuating demand.

By abstracting away the underlying infrastructure, containers provide a consistent runtime environment for applications, irrespective of the host environment. This consistency streamlines the development process, as developers can build and test applications in their local environment before deploying them to production.

Moreover, containers enhance security by enforcing isolation between applications and the host operating system. Each container operates within its own isolated file system and process space, reducing the attack surface and minimizing the impact of potential security vulnerabilities.

Perhaps their most valued feature is that they offer a flexible and efficient approach to application deployment, enabling developers to focus on building applications without being slowed down by the complexities of managing the underlying infrastructure. [1, Chapter 1]

Docker played a significant role in popularizing the container technology. Although

it is not the only platform for containers, it introduced features that improved container adoption. Most important were the standardization of packaging container images, specification of a runtime (simplifying deployment on multiple OSs) and a public repository for sharing container images (container registry). All of these streamlined the usage of containers. It is precisely the ease of deployment across different environments that make containers so attractive. [1, Chapter 1]

In 2015 Docker introduced the Open Container Initiative (OCI). The initiative came forward with 3 specifications: Runtime Specification, Image Specification and Deployment Specification. This effort provides improved compatibility of container ecosystem across different platforms beside Docker. [2]

Given the popularity of containers, it is understandable that in 2016 Microsoft introduced a support for running Windows containers on Windows (WCOW). Prior to this addition, Windows had already offered running Linux containers on Windows (LCOW) for multiple years.[3]

In order to enable execution of containers on an OS, the kernel needs to implement the required functionalities to support the container runtime. This primarily means allowing of isolation of resources which is the basis of the container technology. Specifically, the isolation of network, file system, registry, process space and Object Manager Namespace (OMNS) is required. [4]

Some components used internally to implement Windows Containers and by extension in Windows Sandbox are mostly undocumented. Since the knowledge of them is required for an analysis of their vulnerabilities, some parts of this thesis rely on reverse engineering and on the available research. For the general description in this and the following chapter, 22621.2428 build of Windows 11 was used and 19045.4170 for Windows 10. In the vulnerability analysis in later chapter, the version depends on specific vulnerability.

1.2 Windows Containers Architecture

Windows Containers can be divided into different types based on how they are isolated: [4] [5] [6]

- Process isolation

  In this type of isolation, the container shares the kernel with the host OS. This is the standard type of isolation, which is usually meant by the term container and is approximately what is used by Linux containers.

  This type is sometimes also referred to as Windows Server Containers and is internally code-named Argon Containers. [7, Chapter 8]

- Hyper-V isolation

  For Hyper-V isolation, a process isolated container is additionally placed inside a special lightweight VM. This ensures additional security as the vulnerabilities that allow escaping process isolation are still left inside the isolated VM as it does not share kernel with the host OS.

  These lightweight containers are referred to as Utility Virtual Machines (UVMs) and will be described in section 1.3.

They are internally code-named Xenon or Krypton Containers. It seems Krypton is used in the context of Windows Sandbox and Xenon in the context of Windows Containers. Otherwise, both are referring to the fully isolated Containers using virtualization inside UVM. [7, Chapter 9]

- HostProcess

  This is the most recent supported type of containers. It runs directly on the host and has access similar to regular processes. These won't be further discussed. [8]

Various windows container types are code-named after the noble gases. Along with those mentioned above, application silos are referred to as Helium Containers. When comparing these code names with their position in periodic table of elements, their order corresponds to their level of isolation, with Helium being the least isolated.

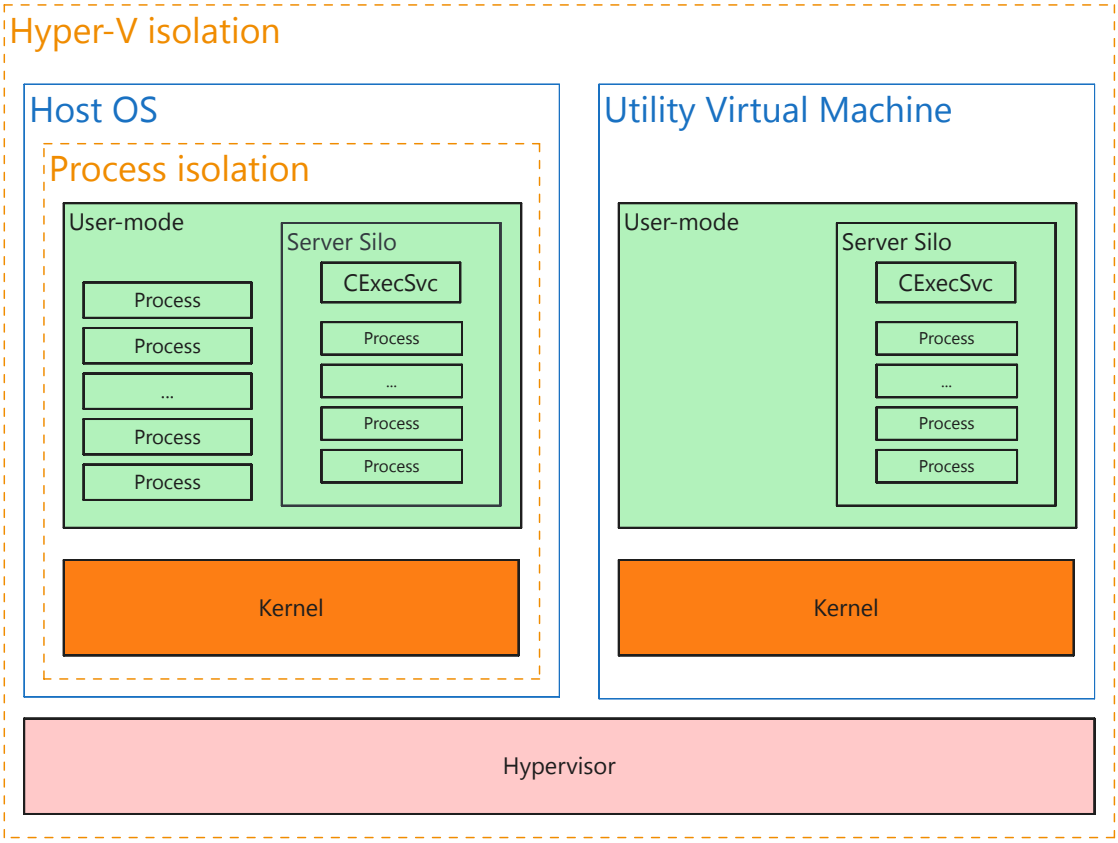

Figure 1.1 The architecture of Windows Containers with process and Hyper-V isolation.

Windows containers offer two default user accounts. An elevated ContainerAdmin and an unprivileged ContainerUser. ContainerUser should be used when possible in order to adhere to the principle of least privilege. This applies especially for multi-tenant environments and is explicitly recommended by Microsoft. A custom user can be additionally created when building a custom image. [9]

Since the attack surface of containers is so vast, process isolation is not considered by Microsoft to be a security boundary. Consequently, this means that possible escapes

are not usually considered vulnerabilities and servicing is not guaranteed. This type of isolation is therefore not recommended by Microsoft for uses where security is an issue. If that is the case, Hyper-V isolation should be used as it adds another layer of protection and does not rely solely on the correct isolation in the kernel. [4]

A security boundary is a logical separation between security domains with different trust levels. Microsoft defines scopes for what they consider to be security boundary, and therefore they intend to service. For example, these include the kernel mode boundary where a non-administrative user mode process cannot tamper with kernel code and data, or the process boundary where unauthorized process cannot access data or code of another process. [10]

On the other hand some boundaries are not included in these definitions. Quite infamously, User Account Control (UAC) bypasses are not considered to be security boundaries since they do not cross the user boundary or any other defined security boundary. Even though it seems as a security feature aimed at restricting user rights, bypassing it is not considered a vulnerability by Microsoft and does not have a priority for servicing. As a consequence there seems to be at least one known UAC bypass present at all times on Windows. Generally the security features that are considered a defense-in-depth security feature (which includes the UAC bypasses) are not serviced by default. [10] [11]

Windows Server Containers are explicitly stated not to be a security boundary. If the processes inside the container are run as a standard user, all other security boundaries still apply. This means that escaping the container can still be considered a vulnerability, but has to be done from the unprivileged ContainerUser to warrant an investigation of the issue from Microsoft. [10] [4]

1.3 Utility Virtual Machine

UVMs are an optimized version of regular VMs and are used for fully isolated Windows Containers. As these containers need to be lightweight, it is required to implement optimizations to minimize memory and Central Processing Unit (CPU) footprint. These optimizations are achieved by leaving host kernel in charge of their allocations as it would be with any other regular process.

To minimize memory footprint, UVMs utilize hosts Virtual Address (VA) space. Such VMs are referred to as VA-backed VMs. VA-backed VMs use hosts memory manager to leverage features as memory deduplication, trimming or direct mapping.

Traditional VMs get allocated physical pages that they manage. For VA-backed VMs, instead of mapping physical pages to the guest, a special VMMEM process is created in the host OS. This process then hosts the virtual address space, representing the memory dedicated to the guest.[1] MicroVM, a kernel component tightly integrated with the memory manager, manages the mapping of guest physical addresses to the VMMEM's virtual addresses.

This enables the VM to use the same memory management features that are available to host processes. VM's physical pages can be simply paged out as the memory

[1] Hence, they are named VA-backed, as VM's memory manager works against the VA space and not the physical address space.

of processes is normally pageable. The underlying physical pages can also be shared between VMs and host processes, reducing memory density.

With ordinary VMs, hypervisor controls the processor scheduling inside the VMs. Same as with the memory, the goal is to allow the host OS to decide when containers get CPU cycles. To address this, an integrated scheduler (also known as the root scheduler) has been introduced. This component allows the host OS to schedule processes belonging to VMs.

Similarly to the memory allocation, a thread is created inside the VMMEM context for each virtual processor assigned to the VM. These represent CPU cycles available to the VM. The host scheduler then schedules these threads as regular threads that can additionally be subject to VM specific policies. [7, Chapter 9] [12]

1.4 Windows Silo

Windows Container process isolation is represented in the kernel by an object called a silo. It is the main object that ensures correct redirection of access to resources and therefore correct isolation of the container. [13, Chapter 3] The silo is used also in the Hyper-V isolation, but there is another layer of isolation in the form of the UVM. [4]

Windows silo is an extension of a job object. Job objects are used to group Windows processes and allow their administration as a single unit. The job object can restrict the usage of resources such as the memory allocation and the CPU usage and provides a basic accounting of processes included in the job. The job object can also restrict how many processes can be created inside it and all contained processes can be terminated together. Compared to the plain job object, the silo has more advanced capabilities that allow better isolation of contained processes.

When the job object is created, it is empty. Processes can be added repeatedly by calling `AssignProcessToJobObject` and also each can be assigned to multiple jobs. By default, all new created processes are also assigned to the parent processes job. Jobs can also be nested inside each other. Each level can be more restrictive than the previous but not less. [13, Chapter 3]

There are two types of silo that can be created: [4] [13, Chapter 3]

- Application silo

  This type can be created by a common user, without requiring any special permissions.

- Server silo

  In contrast to the application silo, to create a server silo, the user needs to possess the Trusted Computing Base (TCB) privilege. This privilege should be held only by administrators.

  According to [13, Chapter 3], TCB privilege is required to prevent malicious use because server silos are used only by Virtual Machine Compute (Vmcompute) service.

The application silo is used by the Windows Desktop Bridge, internally code-named Centennial. Windows Desktop Bridge is a feature that allows to create Universal Windows Platform (UWP) applications from regular Win32 applications by abstracting their access to system resources. [7, Chapter 8]

Server silos are the type used by Windows Containers. They represent the groupings of processes that make up the containers. Furthermore, they are used in many checks inside the Windows kernel to check whether a call is made from a container. This is then used by the kernel to behave accordingly to ensure proper isolation of the container. [13, Chapter 3]

Access to resources in Windows is abstracted by an object model, where every resource is represented as an opaque object. These objects are sometimes called New Technology (NT) objects. Every object has a type that defines what the object represents. These objects are managed by an executive component, the Windows Object Manager. It manages all resources and makes them available for use from the user mode in the form of the objects. Access from the user mode circumventing the Object Manager is not possible.

When a process wants to access an object, it requests access with specified permissions. If the access is granted, handle to the object is created and is written to the process handle table. Handle is just an opaque reference to the object. The Object Manager uses the handle to look up the object and process uses it for any consequent operations.

Thanks to this design, the Object Manager is a centralized point where all access checks can be done. When a process opens a handle to some object, the Object Manager delegates access check to the Security Reference Monitor (SRM). Given the access token and the Security Descriptor (SD) of object, SRM then evaluates whether to grant access. [7, Chapter 8]

Objects are hierarchically structured in the same manner as a file system. This structure is called OMNS. It is created using the directory type objects. Similarly to the file system, these objects may contain other objects of any type, creating a hierarchical structure. Object in OMNS can be specified using a backslash separated path. The root of the namespace is referred to as the root directory object. [14, Chapter 2]

One of the features of the server silo is the ability to change its root directory object. During its creation, it receives an OMNS path, and this path is set as its new namespace root. This effectively means, that it is isolated from the rest of the namespace and processes inside the silo have access only to the branch defined by this path, since the Object Manager resolves all objects relatively to the root directory object.

This way, the accessibility of objects in the silo is controlled either by creating new objects, copying existing ones, or connecting existing ones using symbolic links. Symbolic links are also just a special type of object, that contains target location for redirection. When accessed, the Object Manager handles redirection to the correct location. [13, Chapter 3]

However, redirecting using these links has to respect the changed root, since they are regularly used. Additionally, they do not require any special permissions to create and can be created by low privileged user, all that is required is having access to the target area. [15] These regular symbolic links would not allow linking to the objects located outside the silo's isolated OMNS.

Because of that, a special flag is used instead that makes the links relative to the global root directory, making them global symbolic links. These links bypass the changed root directory object and act as if the access did not come from a silo. Setting this flag requires TCB, so unlike symbolic links, it cannot be set by regular users. [16]

1.4.1 Creating a Server Silo

A silo is created from the job object, so the job object needs to be created first. This is done by calling the `CreateJobObject` Application Programming Interface (API). The job object is assigned a unique ID during its creation called Job Identifier (JID). It is assigned from the same pool of values as the Thread Identifiers (TIDs) and the Process Identifiers (PIDs), so it is unique among all these types of objects. [13, Chapter 3]

After its creation, the job object can be promoted to an application silo by calling the system API `SetInformationJobObject`. [13, Chapter 3] Following the Windows convention of having generic functions for setting and querying object data, there is a `JOBOBJECTINFOCLASS` enumerable defined for the job object. The enumerable defines values for what exactly is to be set or queried from the job. Some values are officially documented, although not all that are used. For example, those used in silo creation are not documented. [17] They can however be found described for example in the NtObjectManager by James Forshaw. [18]

NtObjectManager is a PowerShell module that wraps NTDLL calls and exposes NT objects. Its core functionality is implemented using .NET, which is also available as a NuGet package. The following command from the PowerShell module returns the possible values for the `JOBOBJECTINFOCLASS` enumerable: [18]

```
Get-NtObjectInformationClass Job
```

To create the application silo, the `JobObjectCreateSilo = 0x23` class is used. This sets the silo flag of the job object. Next the `SetInformationJobObject` function is called again with `JobObjectSiloRootDirectory = 0x25` class to assign the custom root directory object. A call containing this class requires the TCB privilege. The last call is made with the `JobObjectServerSiloInitialize = 0x28` class and this call finally creates the server silo. [13, Chapter 3] [18] [19]

1.4.2 Silo Contexts

The support of silos required an overhaul of various kernel components and drivers that now have to be aware of silos (these are so called “enlightened”). Since many functionalities used to be singletons, but with the introduction of silos they no longer have to be, various references cannot be kernel global but have to utilize a silo context.

This is a mechanism for storing contextual data for the silo. Each silo has allocated an array of context slots and these slots contain individual contexts. Each component or driver then has its own index, at which its context is stored. For example the custom root directory object for Object Manager is stored this way.

Even the host OS without any silo is considered a “pseudo” silo. Sometimes it is referred to as the host silo, mostly in kernel routine names. Inside kernel, it is represented by a `NULL` pointer. For example the `PsIsHostSilo` function will return true if passed a `NULL` pointer.

As the kernel components now expect silo context arrays, the host OS has to provide its own context array too, stored in `PspHostSilo-Globals`. When silo APIs are called with `NULL` they will behave correctly using the host silo and its context array. [13, Chapter 3]

1.5 Container File System

When a container is created, it needs to have access to system files for its functionality as it basically abstracts the whole OS. These are contained in a container base image, which is composed of stacked layers. When the container runs, these layers are combined into a view that is transparent to the user. The base image contains system libraries providing system APIs, and so it defines the capabilities of the container. [3]

Microsoft currently provides 4 Windows base images: [3]

- Windows
- Windows Server
- Nano Server
- Server Core

These base images differ mainly in the scope of the API set that they provide. Windows Server has the most complete API set out of these base images. It is also the biggest image. [3]

The base image is read-only and any changes made inside the container are made in the higher "sandbox" layer called the scratch space. This scratch space is an ephemeral storage that captures all writes and file creations but is disposed of when the container stops. Because it is read-only, the base image can be shared between multiple running containers, avoiding redundant copies. [20]

The composition of different layers into one file system, transparent to the user, is implemented by a minifilter. Minifilters will be described in section 1.6.

Because the kernel is shared, `ntdll.dll` from host OS is used. This is the library that handles system calls (syscalls) and is an interface for the kernel from the user mode, so it needs to be shared with the container in case of process isolation. [13, Chapter 3]

1.6 Minifilters

The file system isolation is achieved using minifilters. Minifilters are a special kind of drivers managed by a driver called Filter Manager. They filter Input/Output (IO) operations for file systems or file system volumes. As their name suggests, they can modify, observe or short-circuit these operations, preventing them entirely. This way, minifilters can extend or change functionality of an operation's intended target.

Minifilters get to process the IO operations sequentially. Filter Manager hands them control by invoking their callbacks for the specific operation. These callbacks are specified by minifilters during their registration. [21]

The load order of minifilters is set by a value called altitude. The altitude is an infinite-precision string that is interpreted as a decimal number. This enables setting a fine-tuned load order. The minifilters are loaded beginning from the highest altitude, so the lower the altitude the closer a minifilter is to the device.

When a developer creates a new minifilter, they are required to request an official altitude from Microsoft. Microsoft always allocates integer altitudes. Since altitudes can also be float numbers, there is the infinite portion after the decimal point unused in

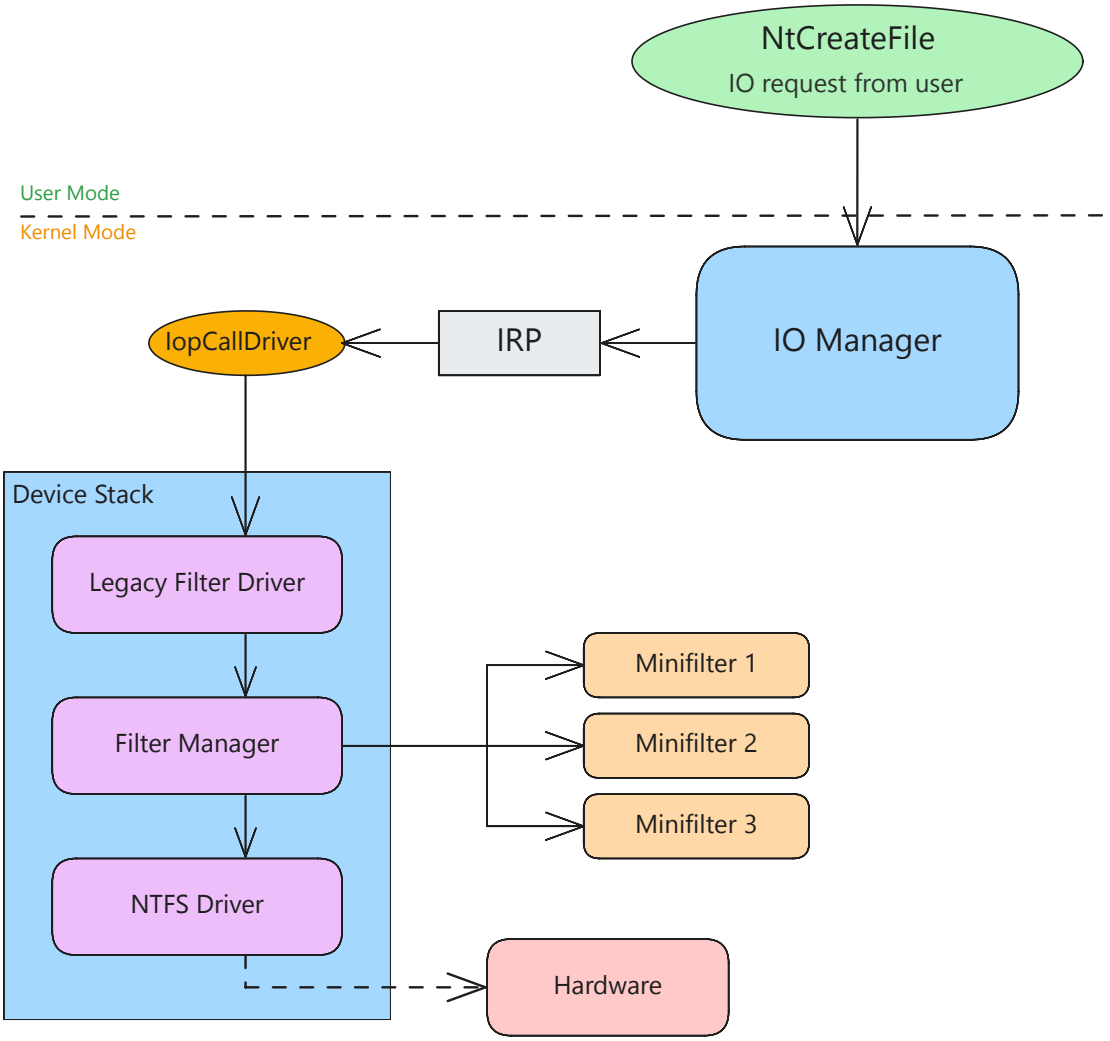

Figure 1.2 File system filter driver architecture

the official allocations for each number. This can be used by the developer who already received an integer altitude without the need for a new registration. [21]

Altitudes are additionally divided into load order groups by their value ranges. These groups are defined by their respective ranges. Each group is specified for different kind of usage and has adequate altitude range for that purpose. For example the FSFilter Activity Monitor group is near the top, lower altitude has the FSFilter Anti-Virus group and even lower the FSFilter Virtualization group. The two bottom groups are reserved for internal use. These are just a few examples with more groups around them.

Minifilters replaced the legacy file system filter drivers that were attached directly in the device stack. Those were written as regular drivers and their development was quite complicated. The driver had to handle every request, even those it did not want to process. Additionally, they could not be easily ordered and their position in the stack usually depended on their load order. In the case of using minifilters, Filter Manager is the only driver required to be loaded directly in the device stack. All minifilters are further registered to the Filter Manager and do not require to be placed into the stack. Minifilters then delegate most of the boilerplate to the Filter Manager focusing on their main purpose of processing IO operations. [22]

The minifilter's `DriverEntry` routine contains initialization including the registration to the Filter Manager. This registration includes callbacks for any operations it wants to process. For every operation it can register pre-request or post-request callback. These two types specify in which direction of the request should be minifilter notified. All unwanted requests are just passed to the next minifilter by Filter Manager. [23]

Requests in question are represented as IO Request Packets (IRPs). When a request is sent to a driver device, it is packaged in these IRPs, which contain all information about the request, making them self-contained. IRP can be processed by multiple drivers. Those that are not final recipients but through which the request only passes are called filter drivers.

A high level representation of minifilter architecture is illustrated in figure 1.2.

Minifilters are used for example in antivirus applications, monitoring tools (e.g., Process Monitor from Sysinternals suite) and for redirection of access to file systems. Following sections will introduce two such minifilters that are relevant to this thesis:

- Windows Container Isolation minifilter driver (Wcifs)

  Handles merging of layers (virtualized folders) into a unique view.

- Windows Bind minifilter driver (BindFlt)

  Redirects single files according to its configured mappings.

Code listing 1.1 An example output of the `fltmc` tool.

```
> fltmc

Filter Name                     Num Instances    Altitude    Frame
------------------------------  -------------  ------------  -----
bindflt                                 1       409800         0
UCPD                                   10       385250.5       0
WdFilter                               10       328010         0
```

```
storqosflt                0        244000          0
wcifs                     1        189900          0
CldFlt                    1        180451          0
bfs                      12        150000          0
FileCrypt                 1        141100          0
luafv                     1        135000          0
npsvctrig                 1         46000          0
Wof                       7         40700          0
FileInfo                 10         40500          0
```

While both minifilters virtualize access to files, the main difference between them is that Wcifs operates on folders (layers) and BindFlt on single files. [7, Chapter 8]

Minifilters can be managed using the `fltmc` tool. Using this tool all minifilters that are registered at the moment and their instances can be retrieved. The code listing 1.1 shows an example output of this tool on Windows 11. From this output, it can be seen that the BindFlt is positioned noticeably higher compared to the Wcifs.

1.6.1 Windows Container Isolation Minifilter

The Windows Container Isolation minifilter's purpose is to merge multiple directories into a single view. In the context of containers these directories are the layers. [7, Chapter 8]

The merging is implemented using reparse points. Reparse points are a feature of the NT File System (NTFS). It is a file or directory that has a block of reparse data associated with it, stored using the `$REPARSE_POINT` NTFS attribute. This data structure is dependent on its use and is understood by the system component that uses the reparse point.

Every reparse point also has an associated reparse tag. The reparse tag is used by its owner to recognize whether the stored reparse point belongs to them. The owner can be either the IO Manager itself or a file system filter driver. In case no tag owner is found, meaning nobody handles the reparse tag, the operation fails.

When the NTFS finds a reparse point in the path that it is currently processing, it returns response indicating reparse point had been hit. When the owner of the reparse tag receives this response and recognizes its reparse tag, it can take any needed action and it can also utilize the stored reparse data if present.

There are two options the owner can take. Either the reparse tag owner modifies the path of the IO operation and reissues it again, or they can remove the reparse point and alter the file.

Junctions (also known as volume mount points) are an example of the first option. They store their target path in reparse data. The path of the IO operation is then modified to use this target path. After the modification of the path, the IO operation is reissued. Wcifs also takes the first approach to redirect accesses into different layer locations. [7, Chapter 11]

In doing so, it utilizes the following reparse tags: [24]

- `IO_REPARSE_TAG_WCI` and `IO_REPARSE_TAG_WCI_1`

  These tags are used for the "expansion" of files. Expansion is the term used by the minifilter for copy-on-open protection. When a file is accessed from inside the con-

tainer, Wcifs copies the file to the container's sandbox layer. This enables transparent editing of files without modifying the original, and reduces the need for duplication of all files that may never be opened.

- `IO_REPARSE_TAG_WCI_LINK` and `IO_REPARSE_TAG_WCI_LINK_1`

  This reparse tag acts as a regular link. When received by a minifilter, it retrieves the path from reparse data and reissues a new IO operation with that path relative to the location of the layer that is specified by layer ID in the reparse data. [25]

- `IO_REPARSE_TAG_WCI_TOMBSTONE`

  This reparse tag is used to hide deleted files from container, without deleting them in their real location. [2]

1.6.2 Windows Bind Minifilter

Although itself not documented, it is stated as being the implementation for the documented bindlink API. This API can be used by administrators to bind local virtual paths. These virtual paths are then redirected to a local or remote location called the backing path.

This enables the creation of arbitrary directory structures with potentially different file names than their original. As this is implemented by the BindFlt minifilter in the kernel, the redirection is transparent to all applications and APIs.

No new physical copies of files are created for the virtual path, as all the mappings are held in memory. Furthermore, the access control check is made for the file specified by the backing path as all operations apply to the backing paths, given that the virtual path does not actually physically exist.

Based on whether a physical file exists at the backing path, the resulting link can be either an anchorless link or a shadow link: [26]

- Anchorless links

  When the virtual path does not exist on disk before a link is created, it will be anchorless. The virtual path is created in memory only.

  To create an anchorless virtual path, the path must be unbroken. This means that the parent directory must either exist on disk or be a previously created virtual path for another link.

- Shadow links

  If the virtual path already exists on disk before a link is created, it will be hidden by the content at the backing path. Hidden files on disk are not removed, just made inaccessible. When the shadow link is removed, original content becomes visible again.

  Again, no write is made on disk and the virtual path is stored in memory.

[2] Mentioned in the following comment:
`https://github.com/microsoft/BuildXL/blob/e74619608566da8ebab5079855f226cb5069db84/Public/Src/Utilities/Native/IO/IFileSystem.cs#L357`

The behavior of access through virtual path can be further modified by specifying following flags during creation: [26]

- `CREATE_BIND_LINK_FLAG_NONE`

  No flags specified.

- `CREATE_BIND_LINK_FLAG_READ_ONLY`

  When this flag is specified, a user can not use the virtual path for writing. All write access bits are masked off. This ensures that the user will have a read-only access through the virtual path. If the user has permission to modify the file located on the backing path, they can still modify the file by accessing it through the backing path, but they can not modify it through the virtual path.

- `CREATE_BIND_LINK_FLAG_MERGED`

  Using this flag modifies the behavior of shadow links. Instead of just hiding the directory backing path, it merges the content of directories from both the virtual path and from the backing path.

  This flag affects only directories and has no effect on files. When a file is present in both the virtual and the backing path, the backing file hides the file on the virtual path as it normally would in the case of the shadow link.

BindFlt minifilter can be managed directly using `bindfltapi.dll`, which is not officially documented and has no public header file. An unofficial header file is available from [27], which was created based on reverse engineering and open-source projects.

The bindlink flags listed prior correspond with the first three internal `_BINDFLT_SETUP_FILTER_FLAGS` flags from `bindfltapi.dll`. There are more BindFlt flags that are not exposed by bindlink flags. It is possible that some are used by the bindlink APIs internally.

One example is the `BINDFLT_FLAG_EMPTY_VIRT_ROOT` flag whose attached comment states: *"Tells BindFlt to fail with `STATUS_OBJECT_PATH_NOT_FOUND` when a mapping is being added but its parent paths (ancestors) have not already been added."* This corresponds to the behavior of the bindlink API that requires creating links on complete, unbroken paths with all ancestors. [27] [26]

1.7 Host Compute Service

When a user wants to create a new container, all components mentioned in the previous sections need to be brought together to provide the wanted functionality. This task is handled by the Host Compute Service (HCS). It resides in the `vmcompute.exe` binary and provides an API for managing compute systems. This involves their creation, mapping of resources and management of processes inside the compute system. [28] [25]

A compute system refers to the abstraction that represents either a VM or a server silo container. Compute systems are ephemeral, meaning HCS does not retain any information about the state of compute system after it is shut down. Their state has to be managed by the client side.

HCS also does not set up any resources for the compute system. All resources need to be prepared by the client before calling HCS API and specified as a parameter. The main resources that need to be prepared are Virtual Hard Disk (VHD) files for storage and a configured networking.

The configuration of a compute system is done by passing a JavaScript Object Notation (JSON) schema as a part of the API call to create the compute system. This configuration contains all required properties of the created system. This includes system type (container or VM), the allocation of memory and processor resources, the network configuration and devices, primarily the storage device. [28]

The JSON schema has a public documentation available at [29]. It is also used for other types of API requests such as the creation of processes or the mapping of mount points.

The HCS API is available in the `computecore.dll` library. It contains the client code for Remote Procedure Call (RPC) requests that are made to the RPC service hosted inside `vmcompute.exe`. These RPC procedures are prefixed with `HcsRpc_`. [28] It also contains code for client side resource preparations, such as storage preparation functions, that do not end up calling HCS RPC procedures.

Microsoft provides two official libraries written in C# and in Go for interfacing with HCS:

- hcsshim (Go) [3]
- dotnet-computevirtualization (C#) [4]

  This library is not currently maintained as the last change to code was done in 2018.

Container runtimes such as Moby[5] or containerd[6] do not call the HCS API directly but use the hcsshim as an interface. [24]

1.8 Communication with a Windows Container

When a container is created by the HCS, a Container Execution Service (CExecSvc) is run inside the container. CExecSvc is a special process that provides communication with the host OS. When a user wants to create a new process inside container, this service acts as a gateway and is the one that creates the new process.

To enable communication of containers with the host OS, a named pipe is created during the initialization of a container. Through it the HCS passes on any requests it receives to the container.

Since the CExecSvc process is present inside every container, any process can use it to detect whether it is running in a container. All that is needed is a check whether a process named CExecSvc is present in the running processes list.

If we use Docker as an example, we can use `docker exec` command to run commands in existing containers. When the target container is a Windows Container, the path of the request is: [5]

[3] `https://github.com/microsoft/hcsshim`
[4] `https://github.com/microsoft/dotnet-computevirtualization`
[5] `https://github.com/moby/moby`
[6] `https://github.com/containerd/containerd`

1. Request is passed from the docker Command Line Interface (CLI) to dockerd (docker daemon).
2. Dockerd calls RPC routine `HcsRpc_CreateProcess` of HCS.
3. HCS calls RPC routine `CExecCreateProcess` of CExecSvc over the established named pipe.
4. CExecSvc then impersonates the active user of the container and creates a new process in its security context.

Chapter 2
Windows Sandbox

Windows Sandbox is a feature available since Windows 10 Insiders build 18305. It is based on the same technologies used in Windows Containers and provides a convenient sandboxed environment. All required files are shipped with the system so no additional download is needed. [30]

However, the feature is not enabled by default and needs to be enabled explicitly, either using the Graphical User Interface (GUI) dialog 'Turn Windows Features on or off' or by running a PowerShell command:

```
Enable-WindowsOptionalFeature
    -FeatureName "Containers-DisposableClientVM" -All -Online
```

This will install the feature, preparing the needed components. After restart of the OS, Windows Sandbox can be used.

It is a lightweight, temporary environment allowing a secure execution of applications isolated from the host OS. The environment is ephemeral, so all data created are persisted only during the active session. When the Sandbox is shut down, all changes are lost. When it is started again, it will be back in its default state. However, it is newly possible to retain data after a restart when the restart is initiated from inside the Sandbox since Windows 11 22H2. [12]

2.1 General features

As is the case with containers, Windows Sandbox tries to be as efficient as possible. To achieve that it leverages the same optimizations that were mentioned in Chapter 1. Internally, it is using the Hyper-V isolation of Windows Containers and consequently is running a UVM instance.

It has access to the optimizations listed in section 1.3, such as the root scheduler or direct memory mapping that enables sharing of mapped memory pages with the host. This way, for example the NTDLL pages are shared with host.

The boot time is especially important for Windows Sandbox since it is meant to be a temporary environment that is meant to be frequently created and shut down again. To further minimize the boot time, snapshotting is used. When the Sandbox is set up, a snapshot is taken during the installation after the Sandbox boots up for the first time.

This state is then used as a default, so further start-ups of Sandbox do not need to boot up and simply load the snapshot state into memory. [30]

The default configuration of Sandbox can be changed using `.wsb` files in Extensible Markup Language (XML) format. These allow modification of simple parameters. The possible settings include providing access to devices such as audio input or output and printer redirection, networking settings, clipboard sharing with host or mapping of directories.

When mapping directories, the path of the host directory and of the sandbox directory need to be specified. Additionally, there is an option to make the mapping read-only, which defaults to false. As this provides Sandbox with an access to the host's file system it obviously weakens security of the Sandbox in case it is compromised, as is noted by Microsoft. [12] Besides the `.wsb` file, there is no other official way to customize the Sandbox. [25]

2.2 Windows Sandbox Architecture

The architecture of Windows Sandbox builds mostly upon the Windows Containers, more accurately the Hyper-V isolation described in Chapter 1. It leverages this technology for isolation and brings its own base image.

2.2.1 Base Image

The base image is shipped with the OS and is located packaged inside `C:\Windows\Containers\serviced\WindowsDefenderApplicationGuard.wim`. It is stored in a Windows Imaging Format (WIM) file, which is a file-based disk imaging format used by Microsoft. When a user enables the Windows Sandbox feature or during system update when the feature is already enabled, the WIM file is unpacked into the Sandbox data directory.

This data directory is located in `C:\ProgramData\Microsoft\Windows\Containers\` and is used for the data of Container Manager Service (CmService). This is the primary service responsible for the Sandbox management. Its code lives inside `cmservice.dll` and is registered as a Windows Service during Sandbox installation.

This service also manages Windows Defender Application Guard (WDAG), a technology closely related to Windows Sandbox as the name of the packaged base image suggests. WDAG is used in the Edge browser for sandboxed browsing. [25]

There are multiple subdirectories inside the CmService directory for different data used by Windows Sandbox. The base image is expanded into the relative path `BaseImage\<GUID>\BaseLayer`.

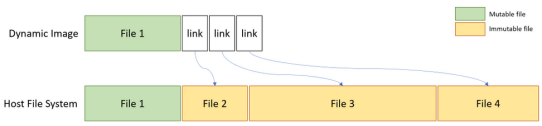

Figure 2.1 Dynamic Image diagram taken from [12].

The base image contains an OS base layer for Windows Sandbox. It is a read-only layer providing system files to boot the Sandbox.

The base image is also optimized using a mechanism called Dynamic Base Image by Microsoft. This technology makes use of existing host OS files to minimize the needed size of the base image. However, not all files can be linked since some OS files can change.

To address this, all mutable files are present as pristine, complete files inside the base image. Immutable files are linked to the host OS file system as illustrated in 2.1. [30]

Although the high level functionality stays the same, some implementation details have changed between Windows 10 and Windows 11. For Windows 10, Windows Sandbox has been researched in [25], but for the Windows 11 changes, there seems to be no comprehensive resource. Most of the information therefore relies on reverse engineering.

Extraction of WIM file is handled by a process `cmimageworker.exe`. It is created by `cmservice.dll` with a provided parameter 'deploy' and the path containing a random Globally Unique Identifier (GUID) assigned to the base image. The GUID is generated random for every new base image (for example after an update).

In case of an update, after the base image shipped with the update is expanded, two Base Images will overlap momentarily. That is, until the old one is deleted, again by `cmimageworker.exe`, this time being called with a parameter `clean`[1] and the path of the older base image. [25]

This is where the first difference occurs between Windows 10 and Windows 11. In case of Windows 10, the WIM file contains a Virtual Hard Disk v2 (VHDx) file of the base layer. When Sandbox starts, this VHDx file is mounted in the same directory. [25]

For Windows 11, the WIM contains a directory structure. Specifically it contains `Bindings` directory, which will be relevant later, and a `Files` directory, containing base layer files, same as the VHDx in Windows 10.

CmService stores its information about the base image into the registry. The base image type for Windows 10 is there referred to as Portable Base Layer (PBL) and for Windows 11 as DynamicImage. The second name is somewhat confusing, since it is used by Microsoft to refer to the redirection of system files inside base image. But this method is not unique to Windows 11 and has been present since the announcement of Windows Sandbox.[30]

2.2.2 File System Layering

Since the base image is just a representation of system files and is itself read-only, there needs to be another mechanism on top of it. As the Sandbox itself is basically a VM, it utilizes VHDx files as its storage.

The VHDx file format can be of three types[31]:

- Fixed

  The virtual drive is created with an allocated size that does not change on disk. When a file is added or removed to the virtual drive it neither expands nor shrinks and stays the same size.

[1] This is the case for Windows 10, parameter `cleanup` is used for Windows 11.

- Dynamic

  In contrast to fixed VHDx, dynamic reflects the size of data actually stored (with the overhead of VHDx specific metadata). When more data is written, the VHDx file expands accordingly and when data is deleted it shrinks. It does, however, have a predetermined maximum size that limits its growth.

- Differencing

  This last type is always used with at least one other VHDx. It stores modified blocks in comparison to the parent virtual hard disk. The parent disk of differencing virtual disk can be of any type and multiple differencing VHDx files can be chained in sequence. The differencing children are functional only with the whole hierarchy of parent virtual hard disks as they alone will include only changes and may be missing crucial data.

  All writes are captured on the latest child. Reads are made by traversing the list of virtual hard disks, starting from the latest child and continuing through parents till the read data is present.

  This type enables easily taking snapshots of the current states of disks.

For Windows Sandbox, dynamic and differencing types are both used, with the latter being especially important for Sandbox functionalities.

With the default installation of sandbox, 4 VHDx files are used as a storage for the Sandbox. There is one dynamic VHDx file with 3 differencing child disks. [25] Although there are some differences in Windows 11 in comparison to Windows 10, they seem to be only in the organization and naming of the VHDx files.

For Windows 11, all VHDx files are stored under a unique GUID in `ContainerStorages\<GUID>`. In Windows 10, they are placed in different directories according to their purpose.

The virtual disks are following, given in order of their dependencies, with the first one being the topmost parent:

1. System Template Base

   This is the baseline dynamic VHDx file on which the rest depends. It is created once per installation of the Sandbox.

   All it contains is a structure mirroring the topmost folders of the base layer. These mirrored directories are all reparse points with reparse tag `IO_REPARSE_TAG_WCI_1`.

   This is the virtual disk responsible for redirection to the base layer from inside the Sandbox.

2. Snapshot

   This VHDx is used to implement the boot optimization mentioned in 2.1. It stores the snapshot of storage taken after the Sandbox UVM first boots. It is used in combination with `Snapshots\<GUID>\SnapshotSavedState.vmrs`, that contains a snapshot of the memory state.

   These are used in all subsequent starts of the Sandbox.

3. Once per host reboot

   This VHDx file is created when CmService starts and gets deleted when it stops. It is described as created once per Sandbox install in [25]. However, on both Windows 11 and Windows 10 tested versions it gets removed when CmService is stopped and is recreated when it starts again.

4. Sandbox State

   This is the final VHDx file and is the last child. As the latest child in the VHDx chain, it captures all writes inside the Sandbox and holds its storage state during the session.

   Since the Sandbox is ephemeral, so is this virtual disk. When the Sandbox is stopped, it gets deleted. It is created anew every time the Sandbox is started again.

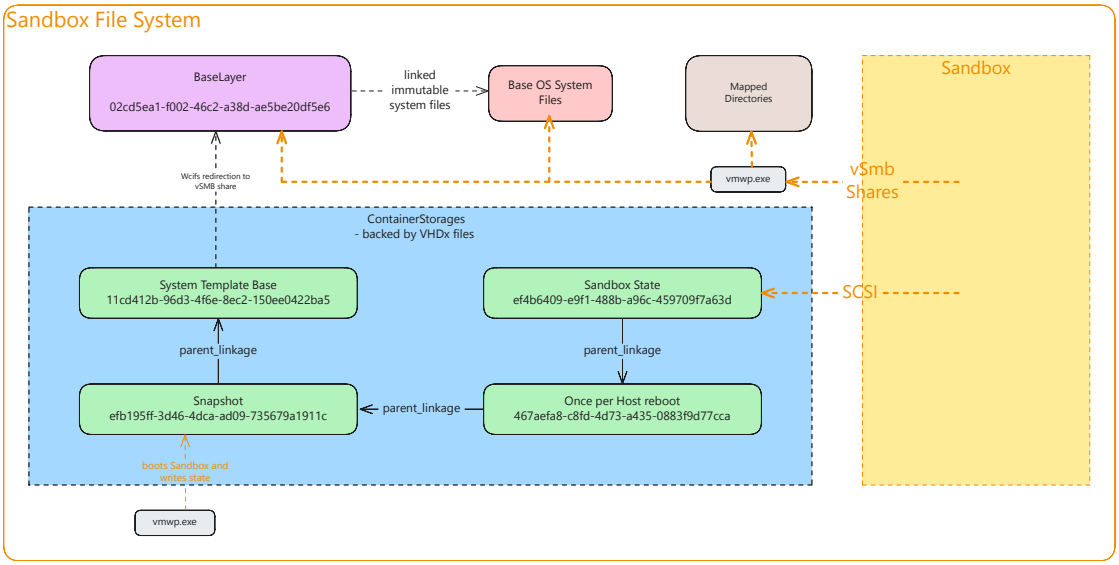

Figure 2.2 Sandbox file system architecture.

This describes the storage from inside the Sandbox. But the base layer is stored on the host file system, aside from this VHDx hierarchy. Depending on what file is accessed, it can be either served from the VHDx files, from the base layer or from the real host OS file system as illustrated in figure 2.2.

The Sandbox State VHDx is used as a backing file for a Small Computer System Interface (SCSI) paravirtualized storage device inside the Sandbox. When this storage device is accessed from within Sandbox, the request is redirected to the host using VMBus. VMBus is a mechanism used for communication between the host OS and the guest VMs. [25]

As was already mentioned, the System Base Template contains Wcifs links to this base layer. Every Wcifs reparse point contains a layer GUID and a relative path to this layer stored in its reparse data. This layer, specified by its ID, has a defined path where it is located. This is used for the redirection of the base layer files. In this case, the path is a virtual Server Message Block (vSMB) share path, since that is the mechanism used for mapping directories from host to the Sandbox. [25]

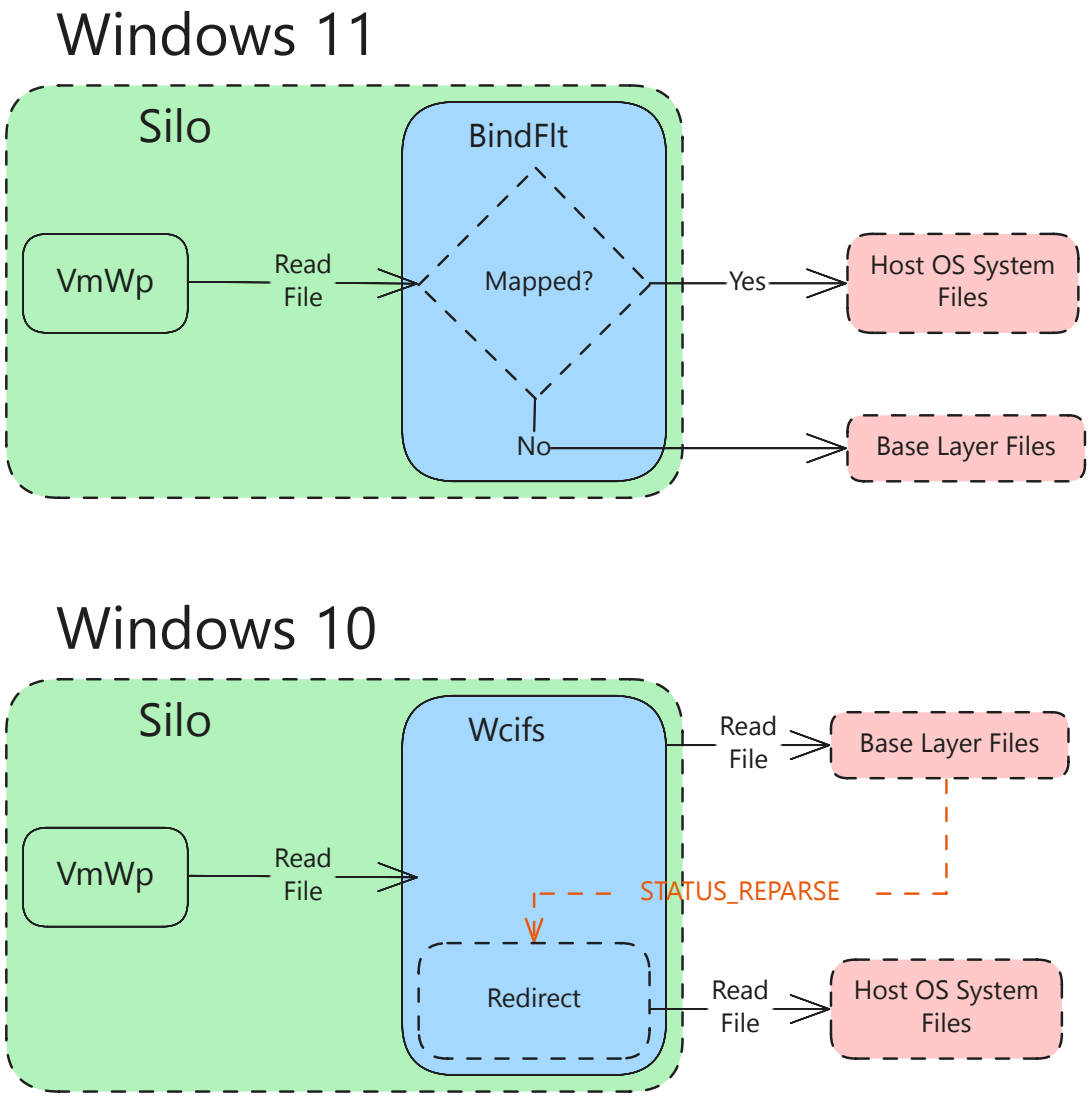

Figure 2.3 Sandbox VMWP attached minifilters.

vSMB is a version of the Server Message Block (SMB) protocol used in Hyper-V. It is used for mapping both the base layer and any other configured mapped directory. The vSMB server runs in a thread created inside the VMWP. When accessed in Sandbox, a vSMB request is made and is handled by the VMWP thread and the target file is accessed from its security context.

In VMWP lays another point of redirection. The so-called dynamic image mechanism used for reducing the size of base image uses minifilters, similarly to how Wcifs is used for base layer redirection from within the container. [25] The specific minifilter used depends on the Windows version, and both options are illustrated in figure 2.3.

Windows 10 uses Wcifs. Those parts of the base layer that are redirected are using reparse tags for redirection, this time the `IO_REPARSE_TAG_WCI_LINK_1`. As described in the section 1.6.1, this type only redirects the IO request to the target location. VMWP accesses the base layer path and is finally redirected to the location of the host OS system file. [25]

Windows 11 uses BindFlt. This minifilter does not use reparse points and instead uses registered mappings in memory. This way, redirection is handled before the operation reaches the volume itself.

All mappings are created by reading stored data from the `Bindings` directory inside the base layer path. This is using the "batched" configuration of BindFlt, where all mappings are read from serialized data. The mappings are attached to a concrete silo, specified by passing a JID value during the call to `BfSetupFilterBatched`.

The mappings can be read from the minifilter using a `bindfltapi` API function `BfGetMappings`. A simple tool that uses this function to read the mappings attached to a silo is included in the attachments of this thesis. When executed with the JID of the silo containing the VMWP, it returns over 23000 mappings.

2.3 Container Manager Service

As was mentioned in the section 1.7, HCS does not keep any information about the compute systems, and they are ephemeral. Windows Sandbox is also ephemeral, but only at the level of the Sandbox State VHDx. It starts from a snapshot state that has to be kept track of. It also has its own base image that needs to be prepared, including setting up the minifilters.

The management of all this state and of all operations needed in preparation and the management of Sandbox is handled by the aforementioned CmService.

CmService exposes a RPC service that provides various management procedures. These procedures can create new containers or manage existing. The RPC service has a permissive Discretionary Access Control List (DACL) that allows everyone to call it. In section 1.7 it is described, that HCS exposes a RPC service too. However, the user needs to be a member of the Hyper-V administrators group to access it and this group should be available only to administrators. [32]

As can be seen in listing 2.1, the JSON schema sent to HCS by CmService to create a compute system has some differences between Windows 10 and Windows 11. Apart from differences in GUIDs that are inherently different since they are generated for new installations, and the different directories, one important change is the removal of the `RunInSilo` section and the addition of the `JobName` property in Windows 11.

This is related to the difference in the type of minifilter used to redirect to host system files from the base layer. Since Windows 11 uses BindFlt, the setup of mappings is done on the CmService side and is bound to the specific silo which is identified by the `JobName` property. This property inside the JSON contains a full name of a job object in the OMNS and this job object is also the mentioned silo (as was previously described, silos are special job objects).

The `RunInSilo` used in case of Windows 10 on the other hand leads to an execution path inside HCS that results in a creation of a new silo on the Vmcompute side. This includes the communication with Wcifs for the redirection setup. [25]

Code listing 2.1 An example of a captured JSON schema sent to HCS to create a compute system by CmService with changes between Windows 10 and Windows 11.

```
{
  "Owner": "Madrid",
  "SchemaVersion": {
    "Major": 2,
-   "Minor": 2
+   "Minor": 4
  },
  "VirtualMachine": {
    "Devices": { "Scsi": { "primary": { "Attachments": {
            "0": {
              "Type": "VirtualDisk",
-             "Path": "C:\\ProgramData\\Microsoft\\Windows\\
    Containers\\Sandboxes\\2a5c024b-d6aa-43a1-b483-18523175c03a\\
    sandbox.vhdx",
+             "Path": "C:\\ProgramData\\Microsoft\\Windows\\
    Containers\\ContainerStorages\\873c2b18-2b6b-48c9-917e-1
    e1480d1452e\\sandbox.vhdx",
              ...
            } },
          ...
        } },
        },
    "GuestState": {
-     "GuestStateFilePath": "C:\\ProgramData\\Microsoft\\Windows\\
    Containers\\Sandboxes\\2a5c024b-d6aa-43a1-b483-18523175c03a\\
    sandbox.vmgs"
+     "GuestStateFilePath": "C:\\ProgramData\\Microsoft\\Windows\\
    Containers\\ContainerStorages\\873c2b18-2b6b-48c9-917e-1
    e1480d1452e\\sandbox.vmgs"
    },
        ...
  },
  "ShouldTerminateOnLastHandleClosed": true,
- "RunInSilo": {
-   "SiloBaseOsPath": "C:\\ProgramData\\Microsoft\\Windows\\
    Containers\\BaseImages\\372e4387-262b-4507-bfb1-21b778902a1a\\
    BaseLayer\\Files",
-   "NotifySiloJobCreated": true,
-   "FileSystemLayers": [
```

```
-       {
-         "Id": "8264f677-40b0-4ca5-bf9a-944ac2da8087",
-         "Path": "C:\\",
-         "PathType": "AbsolutePath"
-       }
-     ]
-   },
+ "JobName": "\\ContainerManager\\Container_e85dfe2a-7714-490d-b349-
    c1352e80367f"
}
```

2.4 User Interface

When a user wants to start Windows Sandbox, they can use the `WindowsSandbox.exe` application. It communicates with the CmService, either setting up default configuration or it parses the `.wsb` file and sets up the user specified configuration for Sandbox. It also establishes a Remote Desktop Protocol (RDP) connection with the Sandbox, conveniently providing the user with a prepared window containing the running Sandbox.

There is also an undocumented CLI tool `CmDiag`. [33] It is a diagnostic tool, that can be used to communicate with CmService with more control than in the case of `WindowsSandbox.exe`. It also has an option to enable the development mode which then allows to enable debugging for the containers created by CmService. Commands to enable these options change settings in the registry and enabling debugging also creates a new Debug Layer. This layer modifies the boot configuration to allow kernel debugging.

`CmDiag` requires to be run as Administrator, but special privileges are not actually needed for most of the operations. Some truly require it, but most of the basic operations with the CmService containers can be done by a normal user. As this is a simple check inside the tool, it can be manually patched out and the tool can be used by regular users.

Chapter 3

Discovering Vulnerabilities and Related Patches

Previous chapters described the implementation of Windows Sandbox and the underlying Windows Containers technology. This and the following chapter's focus is on the discovery of existing vulnerabilities and the analysis of patches regarding these features.

As could already be seen in the previous chapters, there are many components involved and consequently, there is a relatively vast attack surface. It is no surprise, then, that over the years of the features presence in Windows OS, many vulnerabilities were discovered.

It should, however, be noted, that the vulnerabilities are often exploitable in these components from the host point of view, enabling Escalation of Privileges (EoP) on the host OS. They can also have mitigated impact on Windows Sandbox, for example in the case of silo, since the isolation is strengthened by the use of UVM. These can be considered weakening the security of the Sandbox. However, they do not allow a complete exploitation that would radically reduce the security by allowing for example an escape from the Sandbox.

Here we focus on collecting the disclosed vulnerabilities and on the methods and tools used. The next chapter continues with narrowing down where exactly the issue was located, i.e. in which component of Windows Sandbox and which part of the component. Applied patches are then further analyzed.

3.1 Microsoft Update Guides

When a vulnerability is fixed it is publicly disclosed in the Microsoft Security Response Center (MSRC) Security Update Guide[1]. It contains the name of the vulnerability, a summary and updates that fix this issue. The Security Update Guide also contains a Common Vulnerabilities and Exposures (CVE) number and a Common Vulnerability Scoring System (CVSS) score.

The CVE program's purpose is to catalog publicly disclosed vulnerabilities. When a vulnerability is discovered by a partnered organization, it is assigned a CVE number that

[1] `https://msrc.microsoft.com/update-guide/vulnerability`

identifies a CVE record. This record then holds publicly disclosed information about the vulnerability. [34]

CVSS is an open standard for assessing the severity of vulnerabilities. It uses several metrics. The metrics from the base group describe exploitability and impact and are used to create a general score of the severity.

There have been several versions of CVSS published. The current version is v4.0, released in November 2023.[35] However, all described vulnerabilities use the version v3.1.

A majority of discussed vulnerabilities is older than the new specification and MSRC as of the time of writing seems to still be using the v3.1 anyway.

3.2 Locating the Patch

The search of vulnerability itself can be quite unreliable. Although the vulnerabilities are publicly disclosed, their description released by Microsoft is often quite sparse. Sometimes the most specific detail is the name of the component stated in the CVE title, and it is of course not guaranteed that the fix does not span other related components. This can complicate the connection of a vulnerability with the location where it manifests.

Fortunately, there are several vulnerabilities related to the Sandbox that are described in detail or have at least a Proof of Concept (PoC) available. However, it is possible that the vulnerabilities listed in the following chapter are non-exhaustive.

One of the important information contained in the Security Update Guides is exactly what updates are fixing the issue. This is helpful during the first step for identifying the specific patch of the vulnerability. Combining the description of vulnerability and the contents of updates, we can narrow down the search and deduce which specific file has been fixed in all the updates and therefore where the vulnerability should be located.

After identifying the potential patched file, the specific patch inside the file has to be isolated. If the fix is contained in a binary, we can use a binary code similarity detection method (usually referred to as binary diffing or patch diffing in the context of patches). [36] There are multiple tools for binary diffing available, such as BinDiff or Diaphora. Binary diffing allows taking two different versions of a binary and using various approaches to determine what changes are between them.

As a majority of the discussed vulnerabilities are in binaries, this method is quite helpful. The potential patched files are compared with their version before the patch. This provides specific function changes that help with identification of the patched issue and deciding whether it does fit the CVE in question.

3.3 Used tools

This section includes a short description of tools and projects used in this thesis. These were partially utilized also in the previous chapters for the reverse engineering of the Windows Sandbox internals.

3.3.1 WinDbg

WinDbg is a user mode and kernel mode Windows debugger included in Debugging Tools for Windows. It was chosen mainly because of the kernel mode debugging support that is quite useful given that Windows Sandbox is not implemented by a single component, but multiple including the kernel and minifilter support. [37]

3.3.2 Process Monitor

Process monitor from the Sysinternals suite is used for real-time monitoring of file system, registry and process activity. [38]

For the file system activity capturing it uses a minifilter driver. The default altitude is higher than the altitude of many other system minifilters, but Process Monitor can be launched on a lower altitude. This is useful especially because the Wcifs minifilter resides at a relatively low altitude. Minifilters below the process monitor minifilter will not be detectable in stack trace since they handle the request after it had been captured by the Process Monitor minifilter. [39]

3.3.3 Ghidra

Ghidra is an open source disassembler with a built-in decompilation support developed by the National Security Agency (NSA). [40] It was chosen mainly because of the support for its extension BinExport that allows to export files that are used by BinDiff.

Another option could be IDA, which is an alternative reverse engineering tool for static analysis. The main downside is its lack of support for BinDiff integration in the free version. It offers support for BinDiff, but only for the IDA Pro version.

3.3.4 BinDiff

BinDiff is a tool designed for the comparison of executable files to find differences in the disassembled code (binary diffing). It is developed by Google and has been recently open sourced. As BinDiff depends on a separate disassembler, the disassembly has to be done in another application, in this case using Ghidra. [41]

In this thesis, its primary use involves the initial detection of methods that had been patched.

3.3.5 Winbindex

Winbindex is a project indexing Windows file changes across Windows updates.

It significantly simplifies the retrieval of Windows system binaries of specific versions. Without this index, there is no official source tracking the changes of files across updates. Furthermore, the only reliable means of detecting a change is to download the suspected updates, that are usually quite sizable.

The index data is stored in a GitHub repository and is also available through a web application[2]. [42]

[2] `https://winbindex.m417z.com/`

3.3.6 WinbinDiff

WinbinDiff is a simple CLI tool created as a part of this thesis to aid in matching of vulnerabilities with specific patched files. It leverages the data available in the Winbindex project.

This tool offers a functionality that allows users to effectively search for files that have been modified in updates or those that are associated with particular CVEs by accessing information about updates listed as part of the vulnerability's update guide. The Winbindex web application in its current state does not support querying changed files based on the updates and it only allows the reverse type of search, listing changes for specific files. [42]

The patched file can often be deduced by the name of corresponding CVE, but this tool allows us to confirm that and possibly identify other files if the patch spans multiple files. After the identification of the potential files of interest, they can be downloaded for binary diffing using the tool. Files can be downloaded for all updates related to vulnerability and the tool downloads the patched version and the version from before the update for binary diffing.

The downloads of files are by default attempted from the Microsoft public symbol server which aside from public symbols hosts also the binary files. This method of downloading of the binaries is offered by the Winbindex web app too. [42]

This is not always reliable, since some files may be missing from the symbol server, or they are overwritten due to collisions of identifiers used by the symbol server. [42] Consequently, the downloaded file may be a different version. It may also be missing from the server entirely. Since the symbol server supports binaries only, other types of files distributed in updates cannot be downloaded this way (for example, WIM files).

A more reliable and more resource expensive method is to extract the files directly from update files. The tool supports this method too, although it has some caveats.

In newer versions of Windows, updates are distributed using the so-called differential files to minimize size. The process of retrieving a full file from the differentials is called hydration. There can be 3 types of differential files that can be present in updates: [43]

- Null differentials (n)

  Used when the shipped file is new and does not have previous versions. It is only compressed and does not need any other file for hydration.

- Forward differentials (f)

  These differentials are used with base files to create the updated version.

- Reverse differentials (r)

  These are stored with the new version and are used in future updates to create base file from it.

If the file is not of the null type, the new version has to be hydrated using forward differential. It requires the base version of the file to be available, as it is applied to it during the hydration process. [43]

The base file can be created by applying reverse differentials to the binaries present on current system or by having the base version of the OS where all binaries are base

version. In case the downloaded update is a different major release of Windows than the currently used OS, the latter may be the only option as the binaries are not compatible otherwise.

Chapter 4

Known Vulnerabilities Analysis

The vulnerabilities are described in following sections, grouped by the affected components. These components are CmService, BindFlt, Wcifs and the server silo. All the following vulnerabilities are assigned high severity according to CVSS metrics.

4.1 Container Manager Service

This section contains 7 vulnerabilities related to the CmService. The first 5 listed are all fixed in the same update KB5003173 and 4 of them were reported by the security researcher James Forshaw.

CVE-2021-31165

When a Windows Sandbox instance is created, one of the functions called on CmService is the `CmsRpcSrv_CreateContainer`. It prepares the required resources and initializes the compute system (the UVM for the Sandbox) in HCS. This function also takes a handle to a token which is later used for creation of a support process `cmproxyd.exe`. As this RPC service is accessible to everyone, any user can call this function and supply a handle.

The issue is, that the supplied handle is taken by the CmService but it does not check whether the user can pose as the referenced token. So when a user gains a handle to some privileged token, they can leverage CmService to create a process using this token for them, even when they lack the required rights to impersonate the token. [32]

The impact of this is somewhat lowered by the fact that `cmproxyd.exe` is created as an AppContainer process. AppContainers (also referred to as LowBox, which was their original name) are a sandbox[1] and they are primarily used to host UWP applications. Their usage, however, is not restricted to only hosting UWP applications and can host any other application. Although they are a container by name too, and also used for isolation of applications, their implementation is different from the Windows Containers described in Chapter 1. [13, Chapter 7]

[1] In the broader sense of word, not the Windows Sandbox.

The main mechanism through which AppContainers achieve isolation is integrity level. Integrity is a mandatory access control in Windows. During access check, integrity is evaluated first. Only after it succeeds, the standard DACL check is made. There are 3 integrity levels: Low, Medium and High. [13, Chapter 7]

AppContainers are assigned a low integrity. Most objects in Windows are assigned medium integrity (or higher), and by default objects can be accessed only by those with the same or higher integrity. This alone significantly reduces the access of AppContainer process. [13, Chapter 7]

The token of AppContainer has a special AppContainer Security Identifier (SID) also and can contain a set of capabilities. AppContainer capabilities are represented by SIDs and each may give access to some operations. The access is decided by kernel based the presence of the specific capabilities. [13, Chapter 7]

Given that the user can get hold of a SYSTEM token and supply it to the RPC `CmsRpcSrv_CreateContainer` call, the process creation is still exploitable even though it is an AppContainer token. There are still ways to abuse AppContainer tokens when they have SYSTEM identity. [32]

User can abuse the process creation by setting a permissive default DACL on the token. This ensures that the newly created process can be fully controlled by the user. This way, they can effectively pose as the AppContainer SYSTEM token even without impersonating it, by controlling the created process.

The fix for this vulnerability is in essence quite simple since the cause of the vulnerability is basically a missing check. The patched version adds a check that the user has the `SeImpersonatePrivilege` by calling `PrivilegeCheck` on his token. It ensures that this API cannot be abused to gain higher privileges than the user already has. Additionally, it checks whether the supplied token has a sufficient impersonation level by querying its `ImpersonationLevel` value. If it is not at least `SecurityImpersonation`, the token cannot be used for impersonating its security context. [13, Chapter 7] The added checks are located in the `CmsRpcSrv_CreateContainerWithPersistedStorage` function since `CmsRpcSrv_CreateContainer` basically only forwards to this function.

If we try to run the provided PoC after the fix without having the required privilege, the RPC call will fail. When no token is supplied, CmService will use the token of the RPC client (i.e. the identity of the user who is creating the Sandbox).

Aside from its usage during the creation of container, it is also stored as a part of the structure representing the container for possible further usage by the CmService. Since in the subsequent communication with the CmService RPC service the container is identified by an ID and the structure is not available to the user, this token cannot be further tampered with.

CVE-2021-31167

The vulnerability CVE-2021-31167 is fixed by the same update as the previous vulnerability, but a patch for it is listed also for 2 other Windows 10 versions. The analysis was done on the update shared with the other vulnerabilities. The vulnerable function is again a RPC procedure, this time the `CmsRpcSrv_MapNamedPipeToContainer` function that can be used for mapping a named pipe from the host to the Sandbox. [44]

When a named pipe is mapped, the user inside the Sandbox can open it as if it

was located inside the Sandbox. This is achieved by using the existing vSMB, used for mapping directories. It is similar to how regular remote named pipes are mapped over SMB. Given that named pipes act as a normal file object, this seems like a logical solution.

After the named pipe in Sandbox is accessed, there needs to be a middle man to open the corresponding named pipe on the host. Since the vSMB is used, this is done inside the `storvsp` driver. The root cause of this vulnerability is that the creation of host named pipe is done using the VMWP security context. [44]

The reason for why the identity of the Sandbox creator is not used is that the CmService does not pass the user's access token to the HCS when mapping the named pipe. [44] Consequently it does not propagate to the VMWP and the named pipe creation. This is the same token which was abused in `CVE-2021-31165` through its usage for the creation of the `cmproxyd` process.

There are two relevant functions in the `storvsp` driver:

- `VspVsmbFileCreate`
- `VspVsmbCommonRelativeCreate`

The first one is called when a new vSMB share is created. As can be seen using WinDbg, creating a default instance of Sandbox results in two calls of this function, one for the Base Layer and one for the mapping of the Edge browser directory. The latter will be relevant for `CVE-2021-31208`.

In this function, the handle for the target directory on host is opened and the security context is captured by calling `SeCaptureSubjectContext` which takes a snapshot of the calling threads context. All this is stored into the `_VVSMB_SHARE_ROOT_OBJECT` struct. This struct is then used in `VspVsmbCommonRelativeCreate` in combination with the accessed relative path.

During the `VspVsmbCommonRelativeCreate` call, the captured context is impersonated. The previously opened target path handle is used as the root directory for the API call accessing the file, so the path is processed relative to it.

The first part of the patch for this vulnerability is present in CmService. The service now correctly passes the user's token to the HCS when mapping a new named pipe. Thanks to the fix of CVE-2021-31165, this token can no longer be abused for escalation of privileges and has to be accessible to the user to be accepted.

The second part is in the `storvsp` driver. This is mainly to handle the second issue inherent to the behavior of the named pipe impersonation, where the security context of the caller is captured for each write and not just when opening the named pipe. [44] Named pipes are now configured to use `SECURITY_STATIC_TRACKING` and it is done so in the `VspVsmbFileCreate` function.

This patched function contains a hard-coded check whether the target object is `\??\pipe\` (directory in OMNS used for named pipes). If so, the `SECURITY_STATIC_TRACKING` is set for the `PSECURITY_QUALITY_OF_SERVICE` of the `_VVSMB_SHARE_ROOT_OBJECT` struct. Consequently, it will be used for the subsequent accesses to relative files in the function `VspVsmbCommonRelativeCreate`. With this option set, the writes to the pipe will not result in capturing writer's security context, instead only the context captured on open will be relevant (e.g. in case the VMWP would write to this pipe without impersonation of the user's token). [44]

The patched `storvsp` driver is probably the reason why this vulnerability is listed as fixed even for older versions of Windows 10 since those versions do not patch the `cmservice.dll`. In reality, those versions do not contain `cmservice` at all. `Storvsp` on the other hand can be used in other Hyper-V use cases and this could also be exploitable elsewhere.

CVE-2021-31168

Another vulnerability addressed by update KB5003173 is CVE-2021-31168. The problematic RPC function this time is the `CmsRpcSrv_MapVirtualDiskToContainer`. This function can be used to map a virtual disk into the container, which is an operation that is not officially exposed by the `.wsb` configuration file of Windows Sandbox. The virtual disk can be mapped using this function without requiring any additional privileges. [45]

Because the mapping is handled by the VMWP process which runs as the VM service user, `HcsGrantVmAccess` API from `computecore.dll` is called to grant it access to the target file. Since CmService does not impersonate the user during the call of the function, it is done in the security context of SYSTEM. This means that any file that can be opened by SYSTEM for DACL write can be modified using this RPC function. [45]

Given that the user isn't running as a VM service, an immediate exploitation of CVE-2021-31168 may not be apparent. However, security researcher James Forshaw who discovered this vulnerability has suggested multiple potential ways of exploitation. Notably, the provided PoC demonstrates only the capability to set the DACL and not a full EoP.

The first possibility involves the unauthorized modification of virtual disks owned by other users using the API. Once mounted, these virtual disks' content becomes accessible and modifiable from within the Sandbox environment to which they were attached. [45]

While James Forshaw's report does not confirm this particular exploit scenario, users are indeed able to mount virtual disks, including those accessible only to SYSTEM, through the API.

In the patched version, the RPC function begins with the impersonation of the RPC client upon entry.

It proceeds to open the target file with the `ReadControl` access while impersonating. It uses the `CreateFileW` function to open the file, consequently it will also fail if supplied a directory, ruling them out. This function allows opening directories only when `FILE_FLAG_BACKUP_SEMANTICS` is set, which is not the case.

Next, the function validates the client's access token against the open file handle using the `AccessCheck` function, checking read and write permissions, including the `WriteDac` access right.

Following the patch, users are restricted to supply files to which they have the required access (notably, can modify file's DACL), with directories being excluded altogether. Despite the patch, users can still call the function on files that are not virtual disks, although this action has no security implications as they can adjust DACL of these files regardless.

CVE-2021-31169

This vulnerability is again related to the creation of `cmproxyd` process, same as CVE-2021-31165. In addition to the issue of creation of the process with arbitrary token supplied by user, the process is created without impersonation, meaning it is created by the SYSTEM user. The issue here is, that the `cmproxyd` process is created as an AppContainer. [46]

In contrast to ordinary processes, AppContainers have a special directory in OMNS where their objects are isolated. When an AppContainer process is created, this structure has to be prepared. As this is done during the call to `CreateProcessAsUser`, it is carried out in the context of the caller who creates the process. Since the call in the case of `cmproxyd` creation is not impersonated, the SYSTEM user of CmService is used. [46]

The `AppContainerNamedObjects` directory that stores the AppContainer objects is accessible to the user. As they can write in this directory, they can prepare a symbolic link to redirect the creation of the new object. Since the newly created object is created with a SD granting the user full access, this results in the creation of directory object in an arbitrary location that is fully accessible to the user. This effectively leads to a full EoP to the SYSTEM user. [47]

The arbitrary directory object creation itself does not allow running code with SYSTEM user privileges, but a technique leveraging `DefineDosDevice` can be used to allow DLL hijacking which can be used to hijack SYSTEM process.

This exploit creates a `KnownDlls` subdirectory in the `\Global??` object directory. Normally, regular users cannot create directories here. The user creates a temporary symbolic link in this directory named as the target DLL. The destination does not matter, since it will be deleted in the next step and just needs to be a symbolic link.

Next stage uses the behavior of an API `DefineDosDevice`. This API is used to enable users to define new drive letters (called MS-DOS devices, hence the name of API).

Drive letters are in reality just symbolic links to the actual device objects. They are located in a directory, usually referenced using the DOS device prefix `\??`. This prefix refers to a virtual directory object, that does not actually exist in OMNS but is handled by the Object Manager. It is translated to DOS device directories in OMNS and the specific directory that is used depends on who is the calling user. Each user has a subdirectory created in `\Sessions\0\DosDevices\`, while for SYSTEM, it always translates to the aforementioned `\GLOBAL??` directory which is the global DOS device directory.

The user-specific directories also "shadow" the `\GLOBAL??` meaning that if the wanted object is not present in the user specific directory, the search falls back to the global directory. Users can create arbitrary objects in their specific directories, but can only read from the `\GLOBAL??` directory. The function itself prepends received drive letters with `\??` [47]

`DefineDosDevice` takes a set of flags, the drive letter and the target device to map to. An unexpected behavior of `DefineDosDevice` is that it allows any string to be passed as the drive letter and does not actually check whether it is indeed a valid drive letter. As a consequence, this API allows creating symbolic links with any name, possibly outside the user's DosDevices directory. Additionally, the created symbolic links are permanent, which normally requires the `SeCreatePermanentPrivilege` privilege that is not held by

regular users. By default, named kernel objects are temporary and are removed when the last handle to them is closed, permanent objects on the other hand are not deleted even without any open handle. [47]

The functionality itself is implemented in `BaseSrvDefineDosDevice` in `BASESRV.DLL`. This method is called through a RPC procedure of CSRSS process. [47]

The method does have further interesting behavior. First, it tries to open the handle of the source object with `DELETE` access while impersonating the user. This is because the method also supports redefining existing symbolic links. If the handle is successfully opened, the symbolic link is deleted. But first, it checks whether it is located in the `\GLOBAL??` directory. In case it is, it turns off the impersonation for the next part of the function. [47]

The next part is the main functionality, that is the creation of the symbolic link itself. If the impersonation is turned off, it is carried out as the user running CSRSS, which is the SYSTEM user. This specific case is used in the exploit of CVE-2021-31169.

The behavior of `DefineDosDevice` function is not that big of an issue in common scenarios as the `\GLOBAL??` directory does not allow unprivileged users to create objects. This is, however, achieved through this vulnerability. When combined, the result is creation of a DLL record inside of `\KnownDlls\` pointing to a user-controlled section object.

Section objects are kernel objects that represent block of memory that can be shared by multiple processes. It is used to implement memory-mapped files and used to map executable images into memory. [7] In this case, it can be leveraged for DLL hijacking through the OMNS.

The exploitation approach is illustrated in figure 4.1. Directories that allow users the creation of child objects and user accessible objects are depicted green with the rest being red. Objects outlined blue are symbolic links with dashed blue arrows pointing to their targets.

As the `cmproxyd` AppContainer directory is named after the AppContainer SID of the process, it can be prepared by the user. The SID is derived using known method from the AppContainer name which is in the form of `cmproxyd-<Container-GUID>`. The container GUID is known to the user, therefore they can derive the SID and create a symbolic link in `AppContainerNamedObjects` located in the user's session directory. This directory is writable to users.

The target of the symbolic link is `\GLOBAL??\KnownDlls`. It does not exist by default and is created by the redirection through the symbolic link during the process creation. The directory is created modifiable by the user, which is important for the next phase that leverages the `DefineDosDevice` API.

Before calling the API, 2 more symbolic links and the section object need to be prepared.

1. `\Global??\TAPI32.DLL` symbolic link

   The target of this does not matter as it is deleted by `DefineDosDevice` when it redefines the symbolic link. It needs to be present as a symbolic link due to the expectations of the function.

2. `\??\GLOBALROOT` symbolic link

This shadows the `\GLOBAL??\GLOBALROOT` object in the user's DosDevices directory. The original points to the root directory `\`. This newly created symbolic link has the `\GLOBAL??` directory as its target.

3. `\??\MaliciousSection` section object

   It does not matter where the section object is created. `\??` is a suitable choice as it is writable by the user (it translates to his DOS device directory), but there are other possibilities. The provided PoC for example uses `\RPC Control` as it is another location where common users can create objects.

`DefineDosDevice` is called with `GLOBALROOT\KnownDlls\TAPI32.DL` as the "drive letter" and the path of the section `\??\MaliciousSection` as its target. When the method tries to delete the existing symbolic link, it does so while impersonating the user, so the prefix `\??` is interpreted as the user's DOS devices directory (as shown in part 3a. in Figure 4.1) and the prepared `GLOBALROOT` redirects the deletion to the fabricated symbolic link `\GLOBAL??\KnownDlls\TAPI32.DLL` (as shown in part 3b. in Figure 4.1).

As the link is located in `\GLOBAL??`, impersonation is disabled. This means that during the creation of the new symbolic link, the SYSTEM user is used and `\GLOBAL??\GLOBALROOT` is accessed (as shown in part 4a. in Figure 4.1). As this symbolic link points to the root directory object, it actually creates a new symbolic link located at `\KnownDlls\TAPI32.DLL` with its target object located at `\??\MaliciousSection` (as shown in part 4b. in Figure 4.1).

The fix implemented closely aligns with the recommendations outlined in the vulnerability report. Specifically, it ensures proper impersonation of the received user token is used with a correct check whether the impersonation succeeded. While under impersonation, the call to `CreateProcessAsUserW` is made to create the supporting process and then the service reverts back from the impersonation.

Interestingly, the vulnerable version of the function creating the process also receives user's token as a parameter but does not utilize it for impersonation. It is only passed as a parameter to the `CreateProcessAsUserW` function as was described in the CVE-2021-31165 vulnerability.

Although this vulnerability has been fixed, the `BaseSrvDefineDosDevice` does not seem to be changed in more recent versions of Windows (this has been tested for Windows 11 22621.2428). As this is not considered a vulnerability and could introduce compatibility issues (as is pointed out in [47]), it is probable that it will not be changed in the near future.

CVE-2021-31208

A fix for this vulnerability is present within the same update as the previously identified vulnerabilities, yet its discovery is attributed to a different individual. While additional details may not be easily discoverable, a short description does indeed exist.

The discovery of this vulnerability is attributed to an individual named Mayx with a link to a GitHub profile attached. Through examination of this profile, a Chinese-language blog repository can be discovered where further details regarding the vulnerability are present. [48] This blog post, dated May 2021, coincides with the public

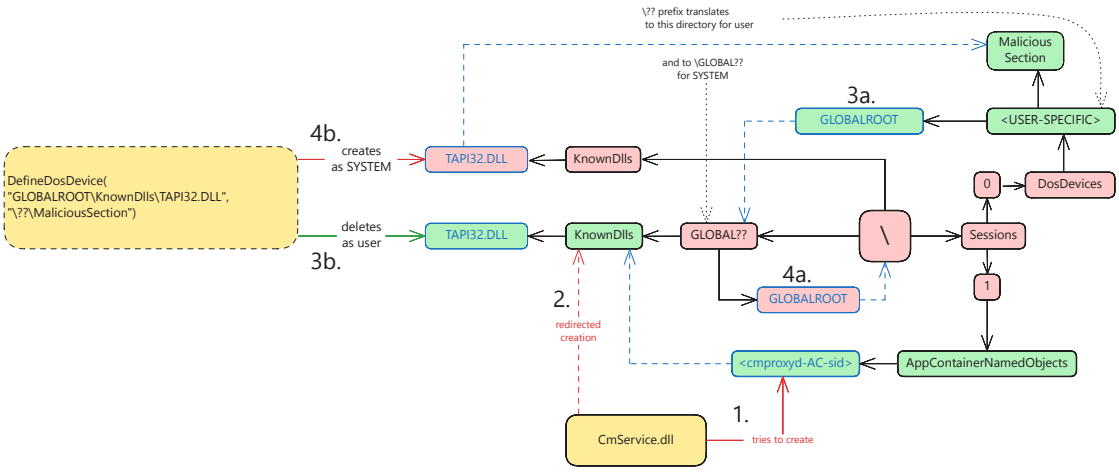

Figure 4.1 CVE-2021-31169 exploitation.

disclosure of the vulnerability by Microsoft and offers a short description of the identified issue.

Specifically the blog post mentions an accidental discovery made while trying to uninstall the Edge browser within the Sandbox environment. It was found that uninstalling Edge in the Sandbox also removed it from the host system, pointing to a vulnerability. The Edge browser directory at `C:\ProgramFiles(x86)\Microsoft\Edge\Application` could have been accessed with write permissions from within the Sandbox, posing a potential security risk to the host system. [48]

However, the blog post does not offer much detail beyond this observation. It does not explain how exactly the Sandbox interacts with the Edge browser directory. Therefore, while it highlights the location of the issue, further investigation is needed to fully understand the problem.

From the description of how the layering of the Sandbox file system works in section 2.2.2, it should not be possible to have a writable mapping of the Base Layer. The directory for Edge is, however, not mapped as a part of the Base Layer files. It is mapped as a separate vSMB share, separate from the “os” share used for the Base Layer.

This mapping configuration is hard-coded within the CmService code and is established during the Windows Sandbox initialization. The process involves the CmService preparing the mapping and subsequently delegating it to the HCS via a call to the `HcsModifyComputeSystem` library function, along with a prepared JSON schema. Optionally, this call can also include a handle to an access token. This token serves the purpose of applying the specified settings, thus ensuring that the appropriate access permissions are enforced within the Sandbox environment.

An issue arises from the default setting of the flag that determines whether the share should be read-only. By default, this flag is set to false. Since the CmService fails to explicitly set this flag to true, the mapping is established with write permissions enabled.

Consequently, any user within the Sandbox environment can manipulate files within this directory, including actions that typically require administrative privileges. This oversight renders the system vulnerable to an EoP exploit as well as a potential escape

from the sandbox environment. Given that the creation of the Sandbox is accessible to regular users, this vulnerability allows for unauthorized access elevating user's privileges. Any malicious actor that is able to act within the Sandbox can also tamper with the Edge binaries, possibly escaping the Sandbox.

In essence, the writable mapping of the Edge browser directory within the Sandbox environment is caused by a misconfiguration within the CmService. Consequently, the fix of this issue is straightforward. In the patched version, the read-only flag is changed to true and the mapping cannot be abused anymore. This fix is located inside the `OnComputeSystemBootedWakeReferenceAcquired` function that handles the mapping of the Edge directory.

When a user maps a directory, for example using the `.wsb` file, it is parsed by `WindowsSandbox.exe` and ultimately set through `CmsRpcSrv_MapFolderToContainer`. This is the standard call route for mapping additional files, and it captures user's token and passes it to the HCS. This way, writes made from within the Sandbox will not be allowed if the user does not have access rights to the target location, which is ensured by the vSMB server running in VMWP.

CVE-2023-36723

This vulnerability is accompanied by a publicly available PoC, including a short description [49]. The issue originates from the base layer files that are extracted from the WIM file, as described in section 2.2.1. Before the patch, the Authenticated Users group had permissions to modify child objects of the `Base Layer` directory, which is an inheritable DACL entry.

Specifically, issues arise with the `Bindings` directory and its subdirectories. If the directory or its contents are missing during the Sandbox startup, `cmimageworker.exe` process that is run by the CmService attempts to recreate them. Since the executable runs as SYSTEM and fails to verify whether it is opening any reparse points, it allows exploitation by regular users who can set up a junction in these locations.

The PoC removes the `BaseLayer\Bindings` directory and recreates it as a junction point with a target path pointing to a directory object in OMNS that is writable for common users. Specifically, it uses the `\RPC Control` directory. Since `Bindings` contains an `Entries` subdirectory, `cmimageworker.exe` tries to recreate it too. Due to the junction, creation is redirected to `\RPC Control\Entries` where the PoC prepares a symbolic link pointing to the arbitrary target location. The creation is made with SYSTEM privileges and the resulting directory is assigned a permissive DACL, allowing the user to create child objects in the new location. This results in an Arbitrary Directory Create vulnerability. Although the PoC does not continue further, it is possible to run code as SYSTEM by abusing the SxS sideloading feature. [49]

Although the CVE name suggests that the vulnerability is located in CmService, which is indeed the source of exploitable behavior, the fixed file is the WIM file distributing the base image as it enables the exploit. The WIM file is shipped with Windows updates.

The root issue seems to be that the permissive SD set on the directory has been captured inside the WIM file during its creation with incorrect access permissions. When the base image is unpacked, a call to `WimSetSecurityDescriptor` inside wimgapi.dll is

made. This will set descriptors for every unpacked file. It will also set the problematic DACL for the `BaseLayer` directory that causes this vulnerability. After the fix, the DACL in question is modified and no longer allows unprivileged users to tamper with the `Bindings` directory.

It appears to be exploitable solely on Windows 11, but patches have also been issued for Windows 10 versions. Since the Windows 10 Base Layer is located in a VHDx file that is mounted when starting the Sandbox, this should not be possible to exploit, given that regular users cannot modify the VHDx contents. One possible explanation for the issued fixes is that the directory structure, including assigned SDs, is identical for Windows 10, so they are also changed. Consequently, the changes are shipped for Windows 10 too, even though there is no immediate danger of abuse.

Although this vulnerability has been successfully fixed, another similar issue has been discovered during the analysis. This newly discovered issue, leads to an Arbitrary Directory Delete vulnerability which also enables an EoP from a regular user to SYSTEM. Although the cause is similar to the cause of CVE-2023-36723, the exploitation is slightly different. In comparison to the CVE-2023-36723, this issue requires more strict conditions and should be exploitable only during the update or initial installation of the Windows Sandbox base layer.

The issue has been reported to Microsoft and they confirmed the reported behavior, acknowledging it as an EoP vulnerability. Since the issue has not been fully assessed and fixed yet, the issue will not be described in further detail.

Summary

The vulnerabilities reported by James Forshaw were mostly about incorrect impersonation or user token management and were fixed mostly according to the suggestions included in the reports.

Altogether, the listed vulnerabilities are in majority EoPs. The CVE-2021-31208 is the only exception that can be considered a sandbox escape.

After the analysis of CVE-2023-36723, a similar issue has been discovered and reported to Microsoft.

4.2 Bind Filter

This section contains the only CVE related to the BindFlt minifilter.

CVE-2022-30132

Although this CVE is listed as an issue present in CmService, CmService does not seem to be changed in the updates listed by the associated update guide.

Instead, the BindFlt is more likely the target of this CVE as the `bindflt.sys` file is changed in all of the listed updates. Furthermore, none other CVEs disclosed in the same month seem to be related to the BindFlt by their description.

There is also a similar vulnerability CVE-2022-30131. Aside from both being attributed to k0shl and fixed in the same month (even with a sequentially assigned CVE number), they also seem to be caused by the same issue, although in two different

minifilters. CVE-2022-30131 seems to be located in the Windows Container Name Virtualization minifilter (which has not been described in the thesis). This minifilter contains the same changes as the BindFlt, which will be described further, in its `WcnFsctlSetLayerRoot` function.

Given the previous, it is assumed that this vulnerability CVE-2022-30132 is assigned to the issue located in the BindFlt even though its name suggests CmService.

The issue in BindFlt is located in the `BfFsctlSetLayerRoot` function. This function is called when the minifilter processes an IO operation with the undocumented `FSCTL_SET_LAYER_ROOT = 0x90394`.

A file name is copied into an output buffer of the request. Since other data is stored before the copied string, the length of the output buffer is checked against the file name length with an added value 8 (for 4 short integers also stored in the buffer).

In the vulnerable version, there is no check whether this addition overflows. This can consequently lead to a buffer overflow by bypassing the length check using the integer overflow. Since a `memcpy` function is used with the original file name length, in case the condition is passed, one can write outside the output buffer. Additionally, name length with a value 6 added is also stored at the beginning of the output buffer. When the integer overflow is not handled, incorrect value can be stored here. This could possibly lead to an unexpected behavior in further processing of the buffer. The simplified code with highlighed fix can be seen in code listing 4.1.

Code listing 4.1 Difference between the original and patched function BfFsctlSetLayerRoot for CVE-2022-30132.

```
short fileNameLen = fileName.Length;
+ if (fileNameLen + 8 < 8){
+    // error (overflow)
+ }
if (OutputBufferLength < fileNameLen + 8) {
    // error
}
else {
    outBuffer[0] = fileNameLen + 6;
    ...
    outBuffer[3] = fileNameLen;

    memcpy(outBuffer + 4, fileName, fileNameLen);
    ...
}
```

The fix simply checks for the integer overflow by comparing the result of the addition. If the result is less than the added value, an overflow occurred and an error is returned.

Although the fix eliminates the possibility of abuse, correctly checking for integer overflow, it is not clear whether this could have been exploited through the usage of Windows Sandbox or CmService in general. From the Windows Sandbox point of view, this function is run once during the Sandbox setup. However, the input of the processed request during setup does not seem to be controlled by the user.

This last fact undermines the assumption that the BindFlt issue is truly the CVE-2022-30132.

One possible explanation is the exploitation through the Argon containers, that can also be created by CmService. Their creation requires administrator privileges in contrast to the Krypton container used by Windows Sandbox that can be created by unprivileged users. This could possibly lead to an escape from the Argon container. If the user inside the container would be unprivileged, this could be considered an EoP vulnerability by Microsoft. This has not been confirmed, however.

4.3 Windows Container Isolation filter

This section contains 2 CVEs related to the Wcifs minifilter.

CVE-2021-34461

This vulnerability is located in the `WcPreGetUnions` function. The function contained an incorrect input buffer length check. In the vulnerable version, the condition required the length to be greater than `0xC` and the fixed version requires the length to be greater than `0x18`, double the previous value.

The exploitable part, however, is in the further called `WcProcessGetUnions`. This function contains code that accesses data beyond the `0xC` outside the buffer bounds.

`WcProcessGetUnions` is reachable in another way. The other option, aside from the one already mentioned, is through the minifilter port messages.

This route does not include the change in condition and still compares against the value `0xC` at the beginning of the function. Since the minifilter port object has strict DACL and is accessible only to privileged users, this cannot be used to escalate privileges and does not allow the possibility of exploitation.

CVE-2021-31190

The main issue in this vulnerability was that a new control code for setting reparse points had been added. The added control code `FSCTL_SET_REPARSE_POINT_EX` allows setting reparse points similarly to the original `FSCTL_SET_REPARSE_POINT` code that can also still be used. The minifilter has special handling of the setting of reparse points and using the new reparse tag would bypass this.

The fix of this vulnerability is simple, as mostly what the patch added was another condition checking for `FSCTL_SET_REPARSE_POINT_EX` in `WcPreFsControl`. It ensures that it will also call `WcFsctlSetReparsePoint` for the new tag. It is then further handled in the `WcFsctlSetReparsePoint` function, where the operation fails with the `STATUS_NOT_SUPPORTED` NtStatus code.

This type of issue is not unique. As is stated in [23], this mismatching of IO operation handling can be often source of vulnerabilities for minifilter drivers. Specifically, the addition of `FSCTL_SET_REPARSE_POINT_EX` is stated as the cause for another vulnerability CVE-2020-17139 in the WOF filter driver.

4.4 Silo

This section contains the issues related to the server silo. Although the issues are focused more on the Windows Containers, silos are utilized by the Windows Sandbox too.

The Siloscape subsection showcases an issue that was exploited in Kubernetes in wild.

CVE-2021-26865

This silo vulnerability is caused by incorrectly retrieving the root directory object from the server silo. The first issue is that it uses `PsGetCurrentSilo` that returns the latest silo in the job object chain (job objects can be nested). This is, however, not guaranteed to be the server silo. The second issue occurs when querying a root directory from the silo. If the silo does not have the root directory set, the host root directory will be returned.

If combined, the behavior can be exploited by leveraging application silos. These silos can be created by unprivileged users, unlike the server silo. They do not have the root directory set, but due to the second issue it does not matter since returning the real root directory is the wanted behavior.

All that is required to exploit the issue is to create a new application silo. Any process assigned to it will use the real root directory, since the application silo is the latest job object and does not have a context with a root directory set. This way any user can access host objects outside the isolation of the server silo, which effectively provides access to host's drive. [4]

To patch this vulnerability, `PspCreateSilo` in ntoskrnl.exe has been changed. It now contains a call to the `PsIsCurrentThreadInServerSilo` function, allowing only calls that are not made from a server silo. This change forbids the creation of any silo from within server silos. Since the creation of an application silo is the main step of exploitation, this change prevents it and also prevents any other vulnerabilities that could be caused by the creation of new silos inside server silo.

Siloscape

Siloscape is not a vulnerability per se but a malware that is labeled as the first known malware targeting Windows Containers. It was first discovered in March 2021 and targets Kubernetes clusters through the Windows Containers.

It leverages an issue of how global symbolic links are handled inside server silos. This was discovered by Daniel Prizmant (who also later discovered the Siloscape) in July 2020. However, it was not considered to be a vulnerability by Microsoft. [50] [16]

This was mainly because this issue requires a TCB privilege that is available only to administrators. Consequently, this does not cross any security boundary which were discussed in Chapter 1. This stands even after Microsoft changed its position after the report of CVE-2021-26865 and CVE-2021-26864 by James Forshaw and now considers escapes from a process isolated container a vulnerability if they are possible for an unprivileged user. [4]

The issue itself lied in the `NtSetInformationSymbolicLink` function. As was mentioned in section 1.4, objects from the root directory of the host can be linked from inside of server silo using an undocumented global flag of symbolic links. The mentioned function is responsible for setting this flag on the symbolic link object. To set the global flag, the user has to possess the TCB privilege.

Linking objects from the global namespace outside the server silo isolation is a legitimate and intended usage of the global flag. However, it can also be abused to escape the container. To prevent the escape, this function should not allow setting the global flag if called from inside a silo. But no such check was present. [16]

Siloscape malware used this for escaping from Windows Containers inside Kubernetes clusters. First, it achieved Remote Code Execution (RCE) using some other known vulnerability. Next, to obtain the TCB privilege needed for the escape, it impersonated the CExecSvc process which is present in every Windows Container. Since it runs as a SYSTEM user, it possesses the TCB privilege. Then it used the global symbolic link to link its container drive to the host `C:` drive. [50]

Later in 2021 Microsoft patched the issue. The patch is described in [51], again by Daniel Prizmant. Although the article describes the fix, it does not mention if the issues has been assigned a CVE or the specific update which contains the fix.

The patch of the vulnerability is simply adding a single check. In the patched function a call to `PsIsCurrentThreadInServerSilo` is made to check if the call is made from a server silo. In case it is indeed made from a server silo, setting the global flag is not allowed. [50]

Chapter 5
Conclusions

This thesis comprehensively described the Windows Sandbox along with the Windows Containers that are the underlying technology. This includes the differences in Windows 11 that were yet not documented in deeper detail.

Following this, an analysis of publicly disclosed vulnerabilities was conducted. A majority of these vulnerabilities was related to the CmService. An analysis of public vulnerabilities revealed that most do not compromise the security of the Sandbox directly but rather exposed a potential EoP on the host system.

There was one exception that could be classified as a sandbox escape. It is possible, though, that some vulnerabilities may have gone undetected due to the insufficient detail in the public CVE descriptions, complicating connection of vulnerabilities with Windows Sandbox.

The evaluation of patches for these known vulnerabilities showed that nearly all effectively addressed the root issues or eliminated potential exploitation pathways.

However, during the analysis of CVE-2023-36723, one issue similar to those analyzed was discovered, although the original issue was fixed correctly. This issue, caused by allowing regular users to create new subdirectories, was still present on fully updated Windows 11 systems. As the previous vulnerability, this one also enables EoP from a regular user to SYSTEM, although the exploitation itself is less straightforward than in the original case. The issue was reported to Microsoft and they confirmed the problematic behavior, acknowledging it as an EoP vulnerability. Since it was not fully assessed and remedied yet, this issue was not described in further detail.

There remains room for further investigation into the altered attack surface of Windows 11 which has not been covered by this thesis. Especially the changes regarding the CmService RPC service, which now includes new methods that were absent in Windows 10. These methods were not analyzed in detail as the focus in this thesis remained primarily on analyzing the patches of existing vulnerabilities, and no vulnerabilities were detected targeting the RPC service in Windows 11. It is possible, that this attack surface has not yet been thoroughly investigated.

Bibliography

1. RAMOS APOLINARIO, Vinicius. *Windows Containers for IT Pros: Transitioning Existing Applications to Containers for On-premises, Cloud, or Hybrid* [online]. Berkeley, CA: Apress, 2021 [visited on 2024-04-05]. ISBN 978-1-4842-6685-4 978-1-4842-6686-1. Available from DOI: 10.1007/978-1-4842-6686-1.
2. THE LINUX FOUNDATION. *Open Container Initiative - Open Container Initiative* [online]. [visited on 2024-03-22]. Available from: https://opencontainers.org/.
3. MICROSOFT. *About Windows Containers* [online]. 2023-03-20. [visited on 2024-04-05]. Available from: https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/.
4. FORSHAW, James. *Who Contains the Containers?* [online]. Project Zero, 2021-04-01. [visited on 2024-04-05]. Available from: https://googleprojectzero.blogspot.com/2021/04/who-contains-containers.html.
5. GERZI, Eviatar. *Understanding Windows Containers Communication* [online]. [visited on 2024-04-06]. Available from: https://www.cyberark.com/resources/threat-research-blog/understanding-windows-containers-communication.
6. MICROSOFT. *Isolation Modes* [online]. 2023-03-17. [visited on 2024-04-13]. Available from: https://learn.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/hyperv-container.
7. ALLIEVI, Andrea; IONESCU, Alex; RUSSINOVICH, Mark E.; SOLOMON, David A. *Windows Internals, Part 2*. Microsoft Press, 2021. ISBN 978-0-13-546244-7.
8. MICROSOFT. *Use Windows HostProcess Containers* [online]. 2023-05-09. [visited on 2024-04-08]. Available from: https://learn.microsoft.com/en-us/azure/aks/use-windows-hpc.
9. MICROSOFT. *Secure Windows Containers* [online]. 2023-03-17. [visited on 2024-04-13]. Available from: https://learn.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/container-security.

10. MICROSOFT. *Microsoft Security Servicing Criteria for Windows* [online]. [visited on 2024-04-08]. Available from: https://www.microsoft.com/en-us/msrc/windows-security-servicing-criteria.

11. HFIREF0X. *UACME: Defeating Windows User Account Control* [online]. [visited on 2024-04-13]. Available from: https://github.com/hfiref0x/UACME.

12. MICROSOFT. *Windows Sandbox - Windows Security* [online]. 2024-03-26. [visited on 2024-03-30]. Available from: https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/windows-sandbox/windows-sandbox-overview.

13. YOSIFOVICH, Pavel; RUSSINOVICH, Mark E.; SOLOMON, David A.; IONESCU, Alex. *Windows Internals: System Architecture, Processes, Threads, Memory Management, and More, Part 1*. Microsoft Press, 2017. ISBN 978-0-13-398646-4.

14. FORSHAW, James. *Windows Security Internals*. San Francisco: No Starch Press, 2024. ISBN 978-1-71850-199-7.

15. FORSHAW, James. *Windows 10^H^H Symbolic Link Mitigations* [online]. Project Zero, 2015-08-25. [visited on 2024-04-06]. Available from: https://googleprojectzero.blogspot.com/2015/08/windows-10hh-symbolic-link-mitigations.html.

16. PRIZMANT, Daniel. *Windows Server Containers Are Open, and Here's How You Can Break Out* [online]. Unit 42, 2020-07-15. [visited on 2024-04-06]. Available from: https://unit42.paloaltonetworks.com/windows-server-containers-vulnerabilities/.

17. MICROSOFT. *SetInformationJobObject Function (Jobapi2.h)* [online]. 2021-11-23. [visited on 2024-04-06]. Available from: https://learn.microsoft.com/en-us/windows/win32/api/jobapi2/nf-jobapi2-setinformationjobobject.

18. FORSHAW, James. *Sandbox-Attacksurface-Analysis-Tools* [online]. Github. [visited on 2024-04-06]. Available from: https://github.com/googleprojectzero/sandbox-attacksurface-analysis-tools/tree/53b658707b3f8fe82cc421dd98a4ea0a8e728fc8.

19. DI MARTINO, Lucas. *Reversing Windows Container, Episode II: Silo to Server Silo* [online]. Quarkslab's blog, 2024-03-26. [visited on 2024-04-05]. Available from: https://blog.quarkslab.com/./reversing-windows-container-part-ii-silo-to-server-silo.html.

20. MICROSOFT. *Container Storage Overview* [online]. 2023-03-17. [visited on 2024-04-13]. Available from: https://learn.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/container-storage.

21. MICROSOFT. *Filter Manager Concepts - Windows Drivers* [online]. 2023-12-07. [visited on 2024-04-08]. Available from: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts.

22. MICROSOFT. *Load Order Groups and Altitudes for Minifilter Drivers* [online]. 2021-12-15. [visited on 2024-04-08]. Available from: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/load-order-groups-and-altitudes-for-minifilter-drivers.

23. FORSHAW, James. *Hunting for Bugs in Windows Mini-Filter Drivers* [online]. Project Zero, 2021-01-14. [visited on 2024-04-06]. Available from: https://googleprojectzero.blogspot.com/2021/01/hunting-for-bugs-in-windows-mini-filter.html.

24. VAN LAERE, Thomas. *Exploring Windows Containers* [online]. 2021-06-30. [visited on 2024-04-06]. Available from: https://thomasvanlaere.com/posts/2021/06/exploring-windows-containers/.

25. ILGAYEV, Alex. *Playing in the (Windows) Sandbox* [online]. Check Point Research, 2021-03-11. [visited on 2024-03-30]. Available from: https://research.checkpoint.com/2021/playing-in-the-windows-sandbox/.

26. MICROSOFT. *Overview of the Bindlink API* [online]. 2024-02-27. [visited on 2024-04-09]. Available from: https://learn.microsoft.com/en-us/windows/win32/bindlink/bindlink-overview.

27. NUKEM9. *Bindfltapi* [online]. 2024-01-28. [visited on 2024-04-09]. Available from: https://github.com/Nukem9/bindfltapi/tree/1398b9b95944384d43d38189f23799044684c522.

28. MICROSOFT. *Host Compute System Overview* [online]. 2022-04-26. [visited on 2024-03-30]. Available from: https://learn.microsoft.com/en-us/virtualization/api/hcs/overview.

29. MICROSOFT. *JSON Schema Reference* [online]. 2022-04-26. [visited on 2024-04-08]. Available from: https://learn.microsoft.com/en-us/virtualization/api/hcs/schemareference.

30. PULAPAKA, Hari. *Windows Sandbox* [online]. Microsoft Tech Community, 2018-12-18. [visited on 2024-04-11]. Available from: https://techcommunity.microsoft.com/t5/windows-os-platform-blog/windows-sandbox/ba-p/301849.

31. MICROSOFT. *[MS-VHDX]: Overview* [online]. 2022-09-30. [visited on 2024-04-13]. Available from: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-vhdx/83f6b700-6216-40f0-aa99-9fcb421206e2.

32. FORSHAW, James. *Container Manager Service CmsRpcSrv_CreateContainer EoP* [online]. Project Zero. [visited on 2024-04-17]. Available from: https://bugs.chromium.org/p/project-zero/issues/detail?id=2149.

33. OGILVIE, Duncan; GOODS, Dylan. *Bootkitting Windows Sandbox* [online]. secret club, 2022-08-29. [visited on 2024-04-14]. Available from: https://secret.club/2022/08/29/bootkitting-windows-sandbox.html.

34. MITRE. *Overview | CVE* [online]. [visited on 2024-04-17]. Available from: https://www.cve.org/About/Overview.

35. FORUM OF INCIDENT RESPONSE AND SECURITY TEAMS. *CVSS v4.0 Specification Document* [online]. FIRST —Forum of Incident Response and Security Teams. [visited on 2024-04-17]. Available from: https://www.first.org/cvss/v4.0/specification-document.

36. HAQ, Irfan Ul; CABALLERO, Juan. A Survey of Binary Code Similarity. *ACM Computing Surveys* [online]. 2021, vol. 54, no. 3, 51:1–51:38 [visited on 2024-05-01]. ISSN 0360-0300. Available from DOI: 10.1145/3446371.

37. MICROSOFT. *Get Started with WinDbg (Kernel-Mode)* [online]. 2023-03-08. [visited on 2024-04-27]. Available from: https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/getting-started-with-windbg--kernel-mode-.

38. RUSSINOVICH, Mark. *Process Monitor - Sysinternals* [online]. 2023-03-09. [visited on 2024-04-27]. Available from: https://learn.microsoft.com/en-us/sysinternals/downloads/procmon.

39. NIEMANN, Teeda. *Change Altitude of Process Monitor (ProcMon)* [online]. Microsoft Tech Community. [visited on 2024-05-08]. Available from: https://techcommunity.microsoft.com/t5/ask-the-performance-team/change-altitude-of-process-monitor-procmon/ba-p/2118159.

40. NSA. *Ghidra: Ghidra Is a Software Reverse Engineering (SRE) Framework* [online]. [visited on 2024-05-01]. Available from: https://github.com/NationalSecurityAgency/ghidra.

41. GOOGLE. *BinDiff* [online]. Google, 2024. [visited on 2024-05-01]. Available from: https://github.com/google/bindiff.

42. MALTSEV, Michael. *Introducing Winbindex - the Windows Binaries Index* [online]. [visited on 2024-04-27]. Available from: https://m417z.com/Introducing-Winbindex-the-Windows-Binaries-Index/.

43. MICROSOFT. *PSFx Whitepaper* [online]. GitHub. [visited on 2024-04-27]. Available from: https://github.com/MicrosoftDocs/windows-itpro-docs/blob/public/windows/deployment/update/PSFxWhitepaper.md.

44. FORSHAW, James. *Container Manager Service CmsRpcSrv_MapNamedPipeToContainer EoP* [online]. Project Zero, 2021-02-06. [visited on 2024-04-17]. Available from: https://bugs.chromium.org/p/project-zero/issues/detail?id=2153&q=CVE-2021-31167&can=1.

45. FORSHAW, James. *Container Manager Service CmsRpcSrv_MapVirtualDiskToContainer EoP* [online]. Project Zero, 2021-02-04. [visited on 2024-04-14]. Available from: https://bugs.chromium.org/p/project-zero/issues/detail?id=2150.

46. FORSHAW, James. *Container Manager Service Arbitrary Object Directory Creation EoP* [online]. Project Zero, 2021-02-04. [visited on 2024-04-17]. Available from: https://bugs.chromium.org/p/project-zero/issues/detail?id=2151&q=CVE-2021-31169&can=1.

47. FORSHAW, James. *Windows Exploitation Tricks: Exploiting Arbitrary Object Directory Creation for Local Elevation of Privilege* [online]. Project Zero, 2018-08-14. [visited on 2024-04-26]. Available from: https://googleprojectzero.blogspot.com/2018/08/windows-exploitation-tricks-exploiting.html.

48. MAYX. 论如何发现一个计算机漏洞 [online]. Mayx 的博客, 2021-05-15. [visited on 2024-04-18]. Available from: https://mabbs.github.io/2021/05/15/vulnerability.html.

49. DRAGOVIĆ, Filip. *Wh04m1001/CVE-2023-36723* [online]. 2024. [visited on 2024-04-30]. Available from: https://github.com/Wh04m1001/CVE-2023-36723.

50. PRIZMANT, Daniel. *Siloscape: First Known Malware Targeting Windows Containers to Compromise Cloud Environments* [online]. Unit 42, 2021-06-07. [visited on 2024-04-06]. Available from: https://unit42.paloaltonetworks.com/siloscape/.

51. PRIZMANT, Daniel. *Microsoft Patched the Issue With Windows Containers That Enabled Siloscape* [online]. Unit 42, 2021-08-05. [visited on 2024-04-06]. Available from: https://origin-unit42.paloaltonetworks.com/windows-container-escape-patch/.

Contents of the attached archive

- `README.md` ..... description of contents
- `sandbox_analysis` ..... files related to the analysis of Sandbox
  - `hcs_json_schemas` ..... captured JSON files sent by CmService to HCS
  - `patch_diffing` ..... screenshots and code snippets from patch diffing
  - `procmon_captures` ..... Process Monitor event captures
  - `sandbox_configs` ..... examples of used .wsb configurations
- `src` ..... source codes of created applications
  - `bindflt_mappings` ..... tool for dumping bindflt mappings
  - `poc.7z` ..... encrypted archive with the PoC of the discovered vulnerability
  - `WinbinDiff` ..... tool aiding with matching files with CVEs for binary diffing
- `text` ..... text of the thesis
  - `thesis.pdf` ..... thesis in PDF format
  - `src` ..... thesis source code