

The COW (Container On Windows) Who Escaped the Silo

Isolations are made to be broken

Eran Segal

Eran Segal

Research team leader

- 7+ years in Cyber Security
- Security Research Team Lead @SafeBreach
- Main focus in vulnerability research



SafeBreach LABS

Agenda

- Background information on process isolation containers
- Investigating the container to gain Admin Privileges
- A technique for finding container vulnerabilities
- Present 2 vulnerabilities in Windows containers
- Demo
- Closure and Q&A

Why this research

1. Containers are everywhere
2. Malicious container image is a real world attack vector
3. Huge attack vector, the entire ntoskrnl
4. Reverse engineering is FUN!
5. Lacking awareness of the vulnerabilities in Windows containers

Intro to CoW

(containers on Windows)



Intro to CoW (containers on Windows)

Containers are similar to virtual machines

Container image contains:

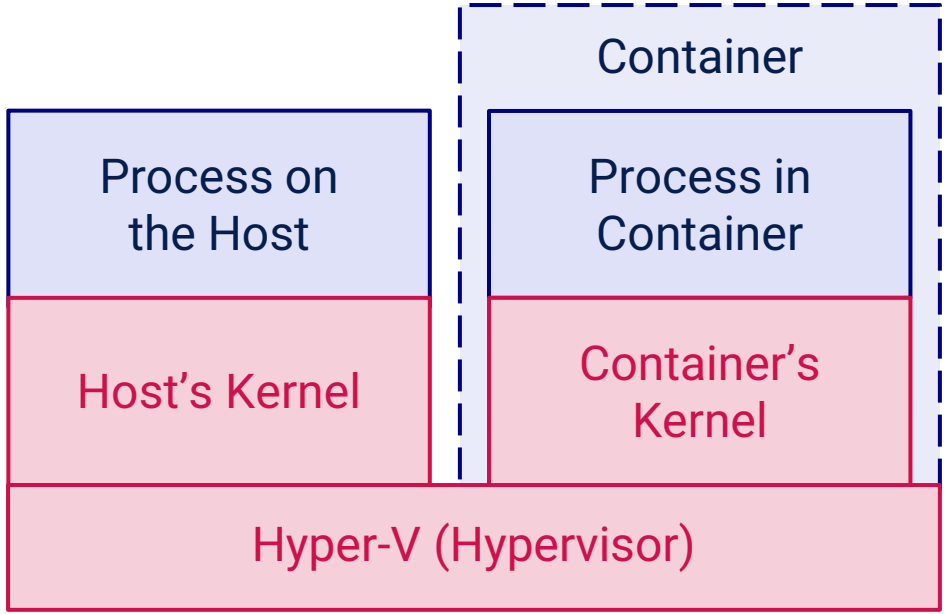
- Filesystem
- Registry
- OS Configurations

Isolation methods of Windows containers

- Process isolated
- Hyper-V isolated

Hyper-V isolated containers

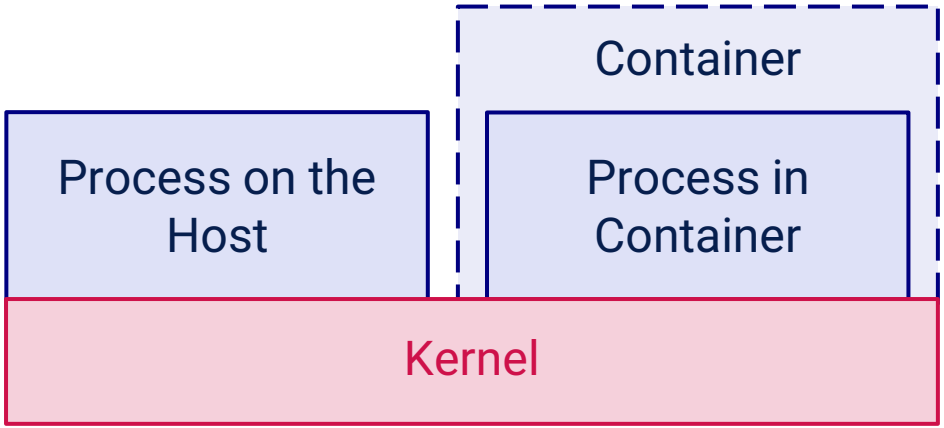
- Similar to a virtual machine over hypervisor
- Kernel is not shared with the Hyper-V container



Process isolated containers

Aspects of isolation inside the container

- File System
- Registry
- Network Ports
- Process and thread ID space
- Object Manager namespace



Windows process isolated container vs Linux container processes

Processes inside Linux container

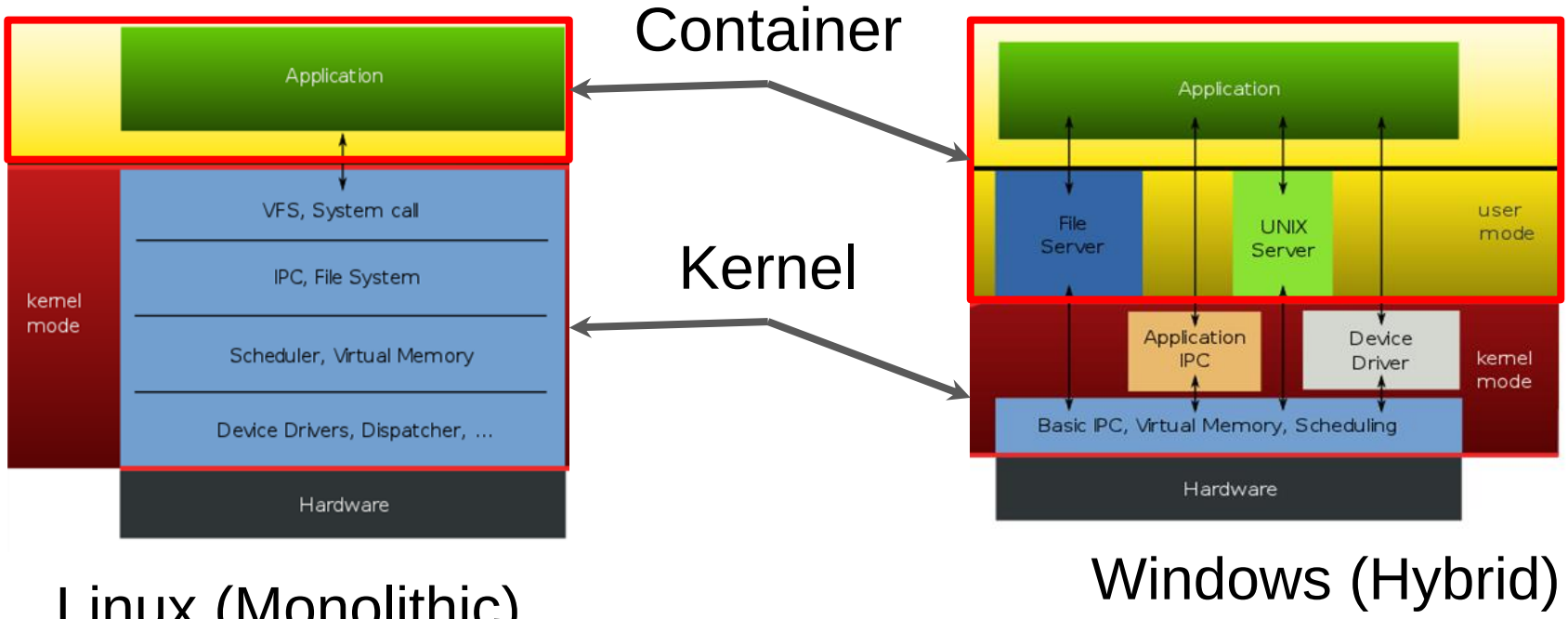
```
root@b63be4c132a9:/# ps -ax
    PID TTY      STAT   TIME COMMAND
      1 pts/0    Ss     0:00 bash
     16 pts/0    R+     0:00 ps -ax
```

Processes inside Windows container

```
Image Name                     PID Session Name        Session#
========================= ======== ================ ===========
System Idle Process              0                            0
System                           4                            0
smss.exe                      6716                            0
csrss.exe                     7560 Services                    2
wininit.exe                   5248 Services                    2
services.exe                  5784 Services                    2
lsass.exe                     6764 Services                    2
fontdrvhost.exe               1588 Services                    2
svchost.exe                   7356 Services                    2
svchost.exe                   5448 Services                    2
svchost.exe                   1148 Services                    2
svchost.exe                   1704 Services                    2
svchost.exe                   7616 Services                    2
svchost.exe                   6588 Services                    2
svchost.exe                   4232 Services                    2
svchost.exe                    196 Services                    2
svchost.exe                   4796 Services                    2
svchost.exe                   7200 Services                    2
svchost.exe                    468 Services                    2
CExecSvc.exe                  5824 Services                    2
conhost.exe                   6848 Services                    2
cmd.exe                       3856 Services                    2
svchost.exe                   7096 Services                    2
MicrosoftEdgeUpdate.exe       7488 Services                    2
svchost.exe                   7724 Services                    2
svchost.exe                    300 Services                    2
taskhostw.exe                 6340 Services                    2
MoUsoCoreWorker.exe           8444 Services                    2
sppsvc.exe                    6920 Services                    2
tasklist.exe                  7640 Services                    2
WmiPrvSE.exe                  8628 Services                    2
```

Why Windows containers are bigger than Linux?

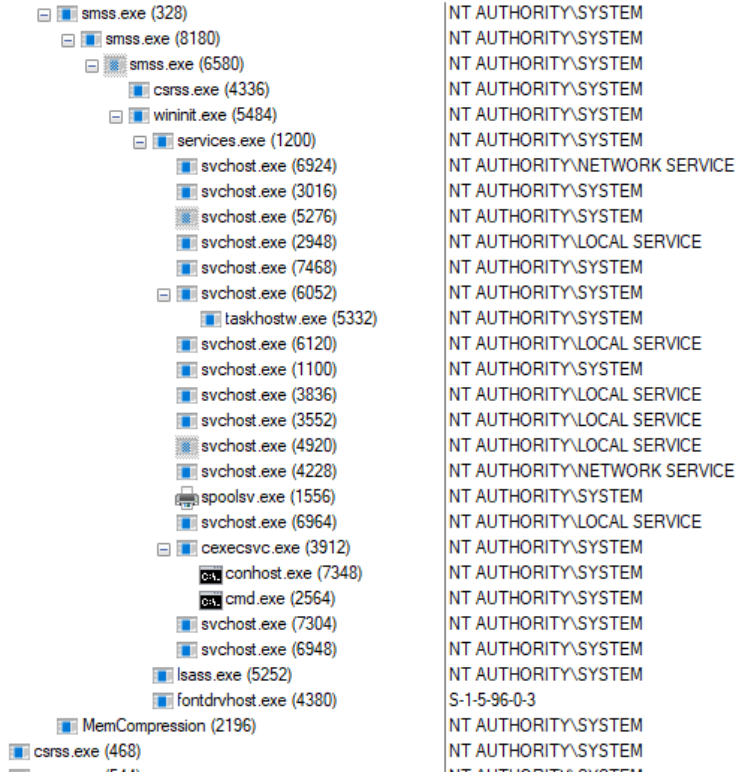
Windows kernel requires more parts to be implemented in the user-mode.



Internals of process isolated Windows container

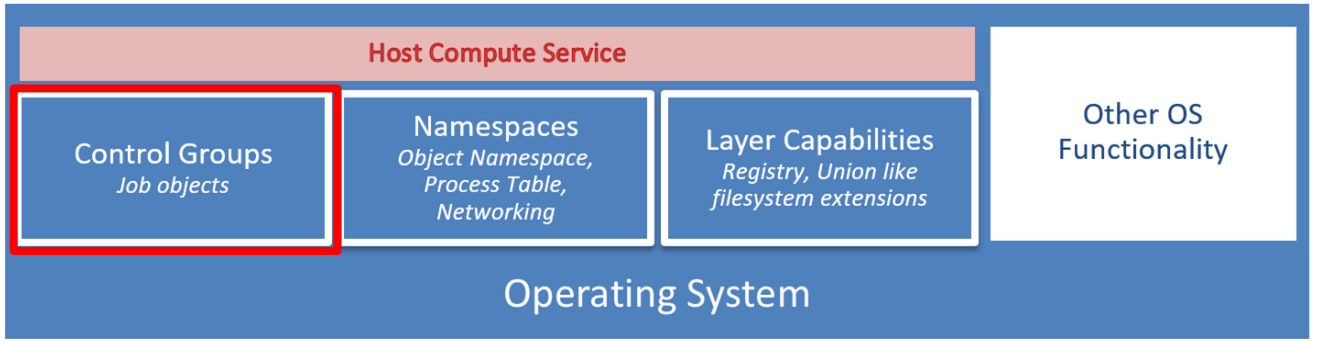
Major container creation events

- Object namespace
- Session for the container
- Virtual registry
- Filesystem
- Server silo object
- And attach a process inside the server silo



Focus of my research - Job objects

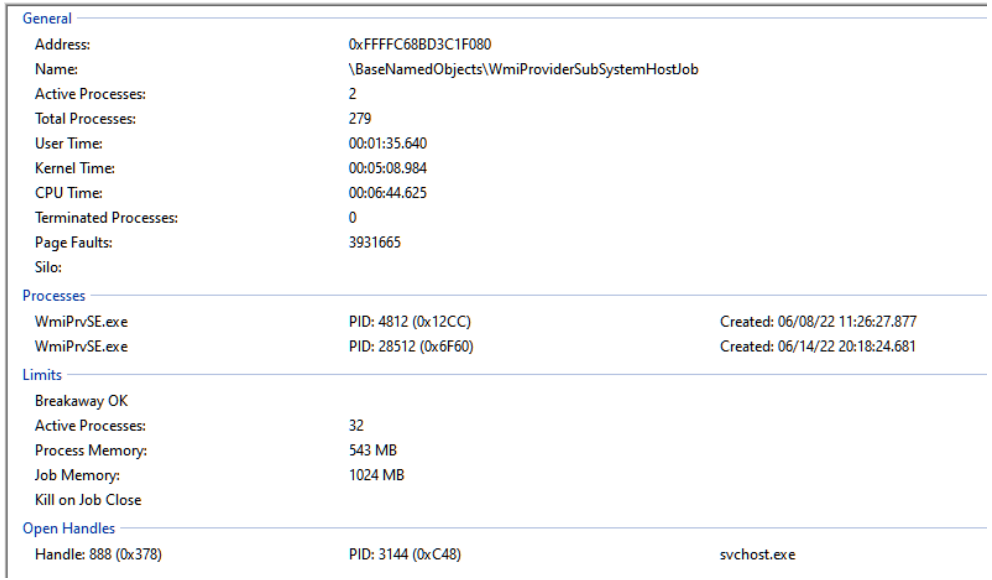
This research focuses on bypassing the job object isolation in the Windows kernel.



Job object (_EJOB)

Jobs are responsible for limiting the container's resources such as:

- CPU
- Memory
- IOPS



Jobexplorer.exe

Upgraded Job - Silo

In order for a job to support isolation, it must be upgraded to a silo.

Name:	\Container_d57732e3(
Active Processes:	24
Total Processes:	33
User Time:	00:00:01.234
Kernel Time:	00:00:03.218
CPU Time:	00:00:04.453
Terminated Processes:	0
Page Faults:	101949
Silo:	Server Silo (948)

Processes

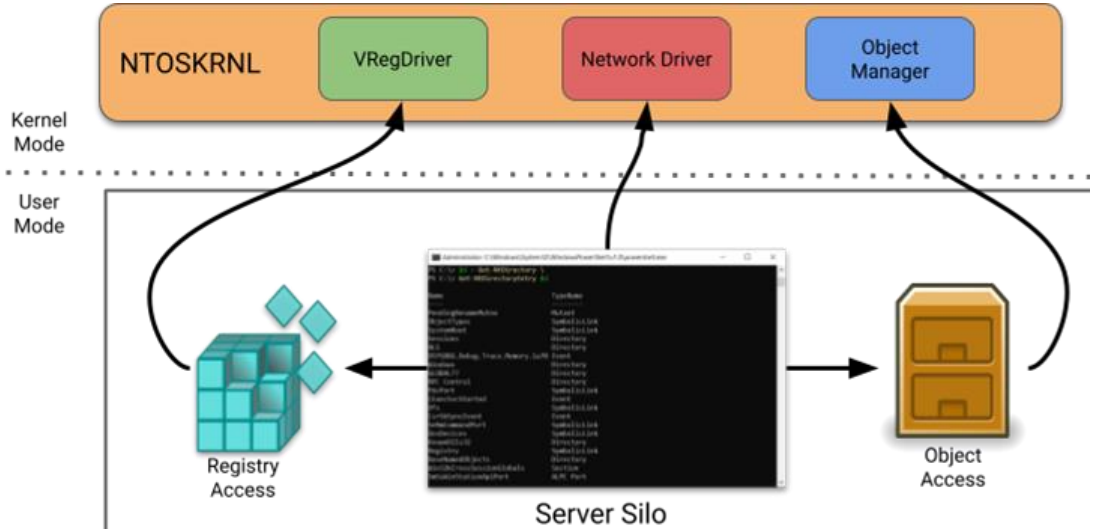
smss.exe	PID: 7032 (0x1B78)
csrss.exe	PID: 6736 (0x1A50)
wininit.exe	PID: 7936 (0x1F00)
services.exe	PID: 5436 (0x153C)
lsass.exe	PID: 7880 (0x1EC8)
fontdrvhost.exe	PID: 3876 (0xF24)
svchost.exe	PID: 3588 (0xE04)
svchost.exe	PID: 7708 (0x1E1C)



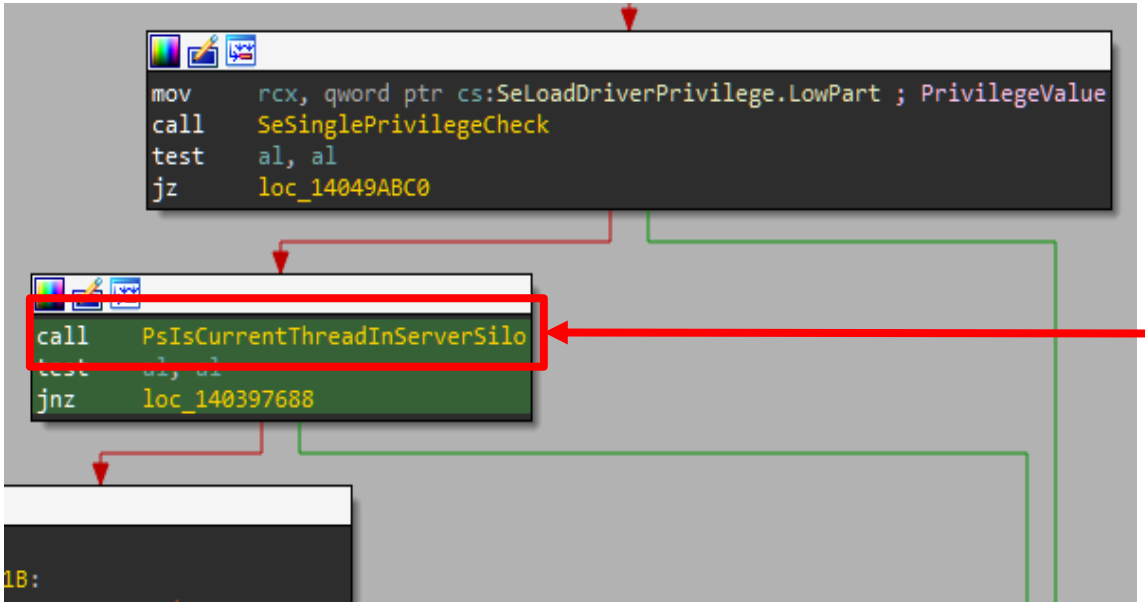
Jobexplorer.exe

Upgraded Silo - SiloServer

SiloServer allows processes inside the container to use resources such as registry that came from the container image and not the host's resources.



How the kernel blocks dangerous syscalls?



Is the current thread in a container?
If so - block the syscall

`IopLoadDriverImage`
(`NtLoadDriver` calls to `IopLoadDriverImage`)

Detect process inside container

```
_EJOB *PsGetCurrentServerSilo()
{
  struct _ETHREAD *currentThread; // rax
  _EJOB *v1; // rcx

  currentThread = (struct _ETHREAD *)KeGetCurrentThread();
  v1 = currentThread->Silo;
  if ( v1 == (_EJOB *)-3i64 )
    return *(_EJOB **)&currentThread->Tcb.Process[2].Header.Lock;
  if ( v1 )
  {
    while ( !PsIsServerSilo(v1) )
      v1 = v1->ParentJob;
  }
  return v1;
}
```

1

Iterate over all job objects related to the current thread

2

Check if the job object is a server silo?

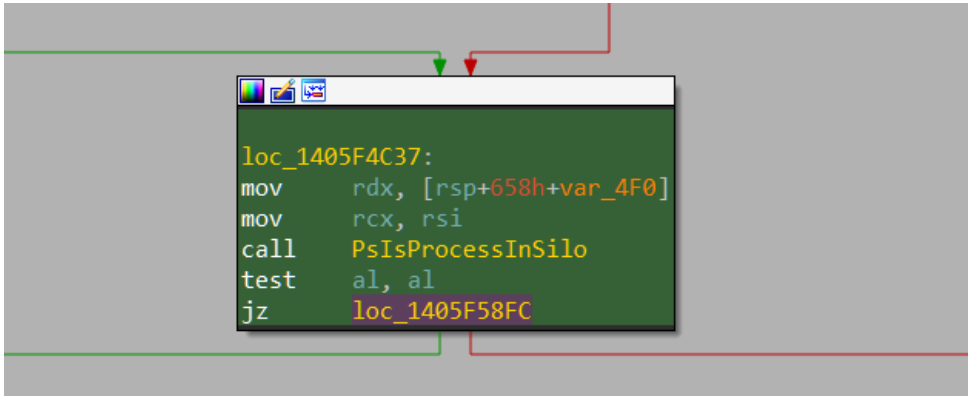
3

```
bool __fastcall PsIsServerSilo(_EJOB *jobObject)
{
  bool result; // al

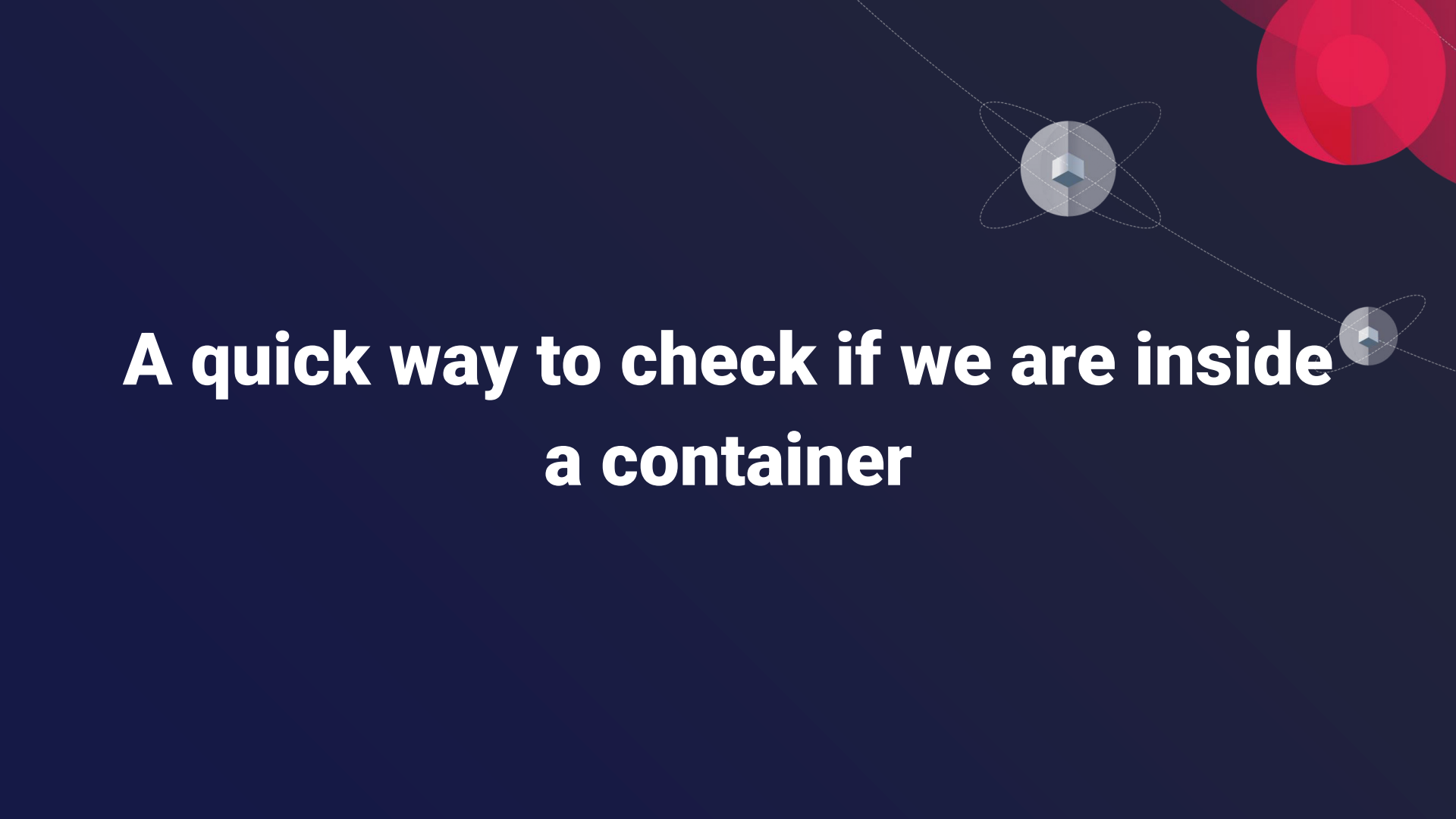
  if ( jobObject )
    result = jobObject->ServerSiloGlobals != 0i64;
  else
    result = 1;
  return result;
}
```

Process isolation

`EnumProcesses -> NtQuerySystemInformation -> ExpGetProcessInformation`



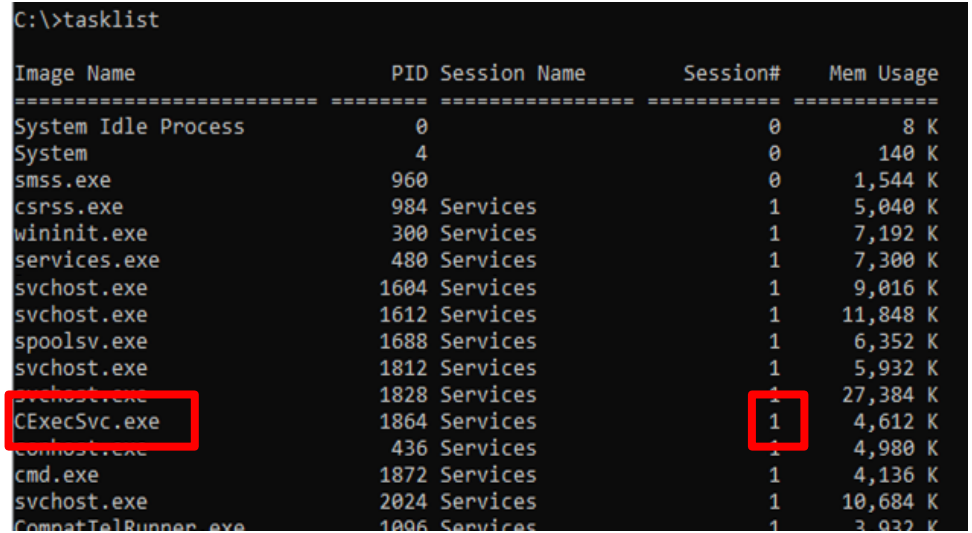
`ExpGetProcessInformation` checks for silo


A quick way to check if we are inside a container

Detect inside Hyper-V container

Indications that we're inside a Hyper-V isolated container

1. *CExecSvc.exe* exists
2. *Dockerd.exe* doesn't exist
3. Session ID is 1



```
C:\>tasklist

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0                            0          8 K
System                           4                            0        140 K
smss.exe                       960                            0      1,544 K
csrss.exe                      984 Services                   1      5,040 K
wininit.exe                    300 Services                   1      7,192 K
services.exe                   480 Services                   1      7,300 K
svchost.exe                   1604 Services                   1      9,016 K
svchost.exe                   1612 Services                   1     11,848 K
spoolsv.exe                   1688 Services                   1      6,352 K
svchost.exe                   1812 Services                   1      5,932 K
                              1828 Services                   1     27,384 K
CExecSvc.exe                  1864 Services                   1      4,612 K
                               436 Services                   1      4,980 K
cmd.exe                       1872 Services                   1      4,136 K
svchost.exe                   2024 Services                   1     10,684 K
CompatTelRunner.exe           1096 Services                   1      3,932 K
```

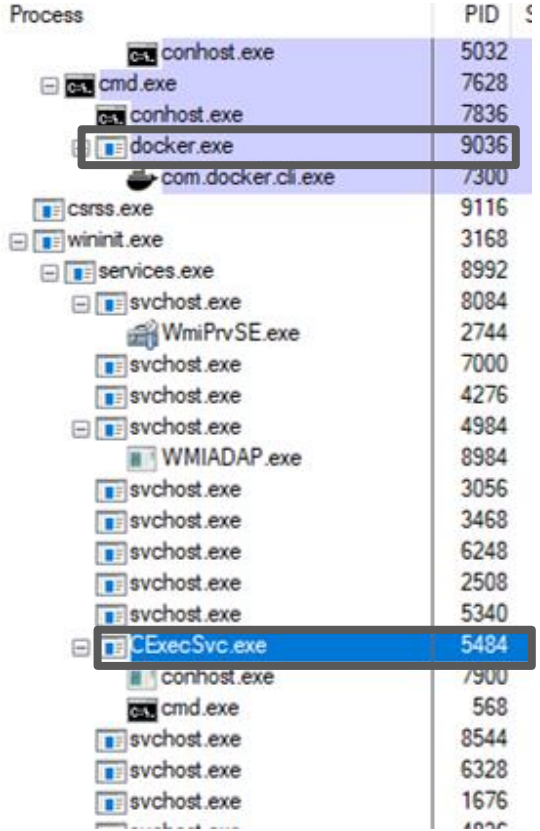
Detect inside process isolated container

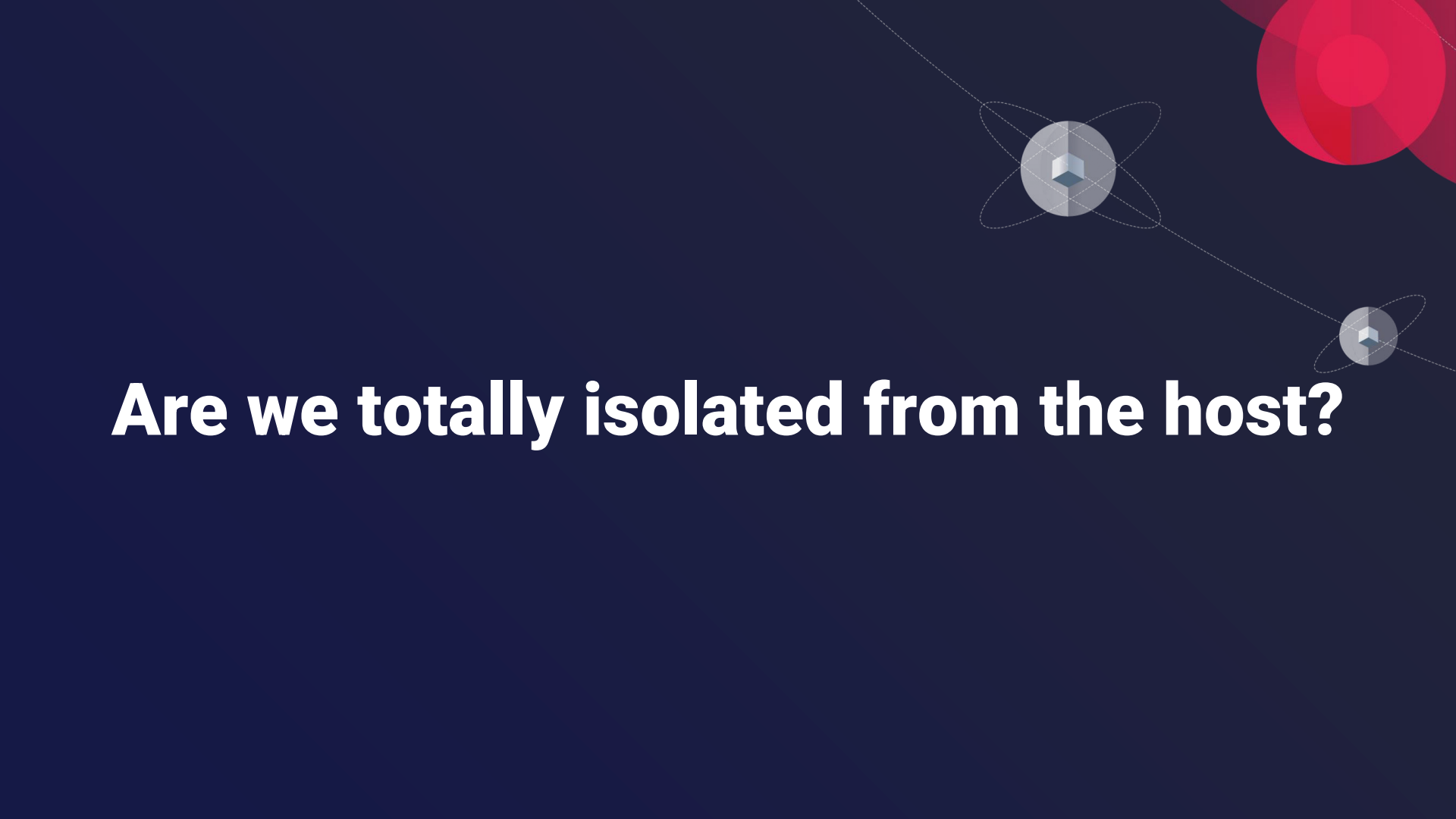
Indications that we're inside a process isolated container

1. `CExecSvc.exe` exists

2. `Dockerd.exe` doesn't exist

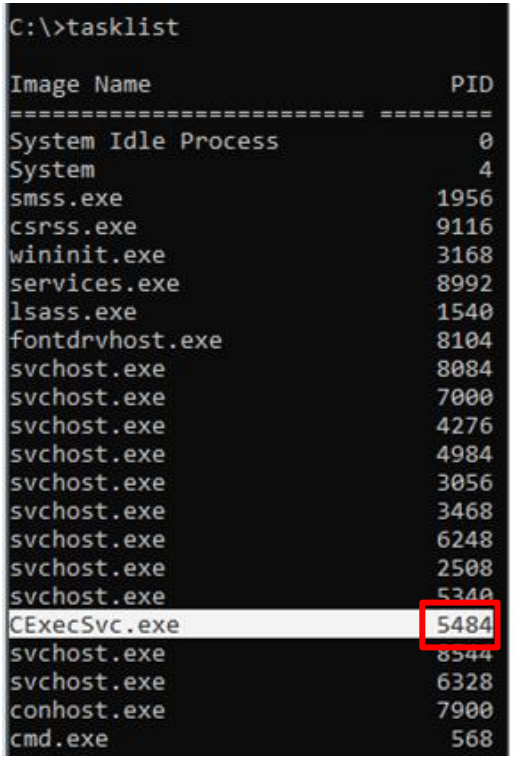
3. Session ID is not 1



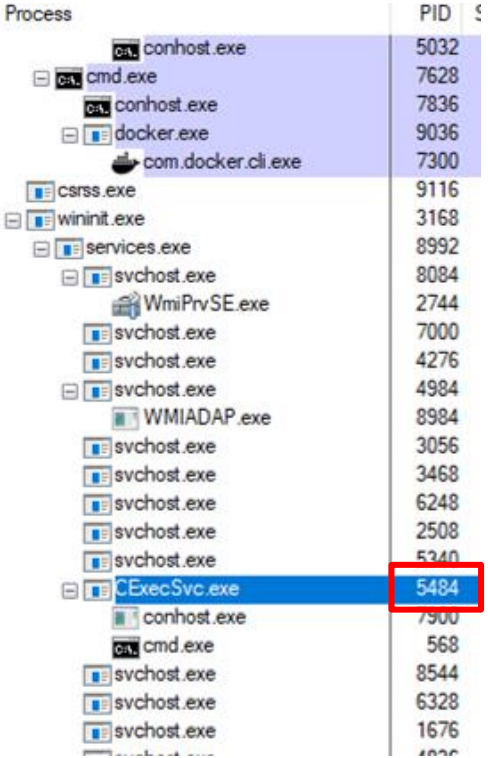

Are we totally isolated from the host?

Process and thread IDS

The PIDs of processes inside the container and outside the container are the same



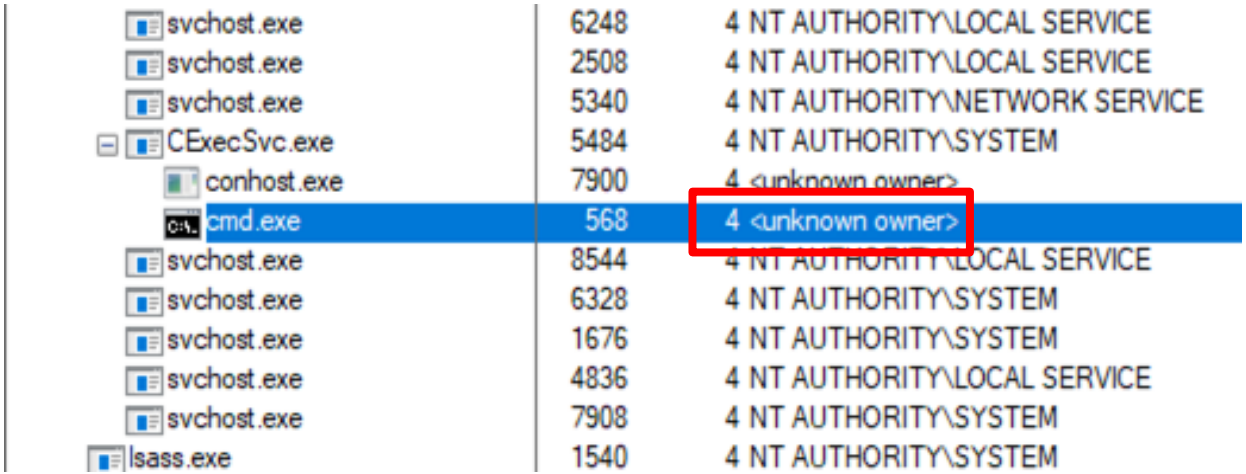
Process list inside container



Process list on the host

User isolation?

```
docker run -it --isolation=process --user="ContainerUser"
mcr.microsoft.com/windows:20H2-amd64 cmd
```



Not completely…
The container's process list from the host shows users exist outside of it

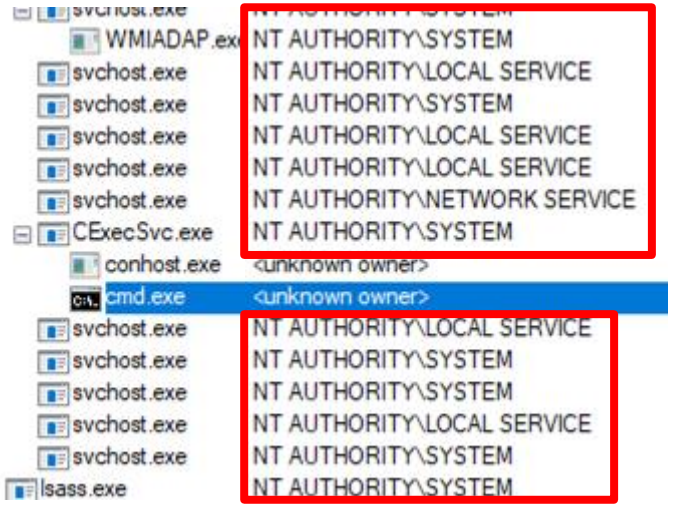
How to gain NT/System inside the container

NT/System is all around us

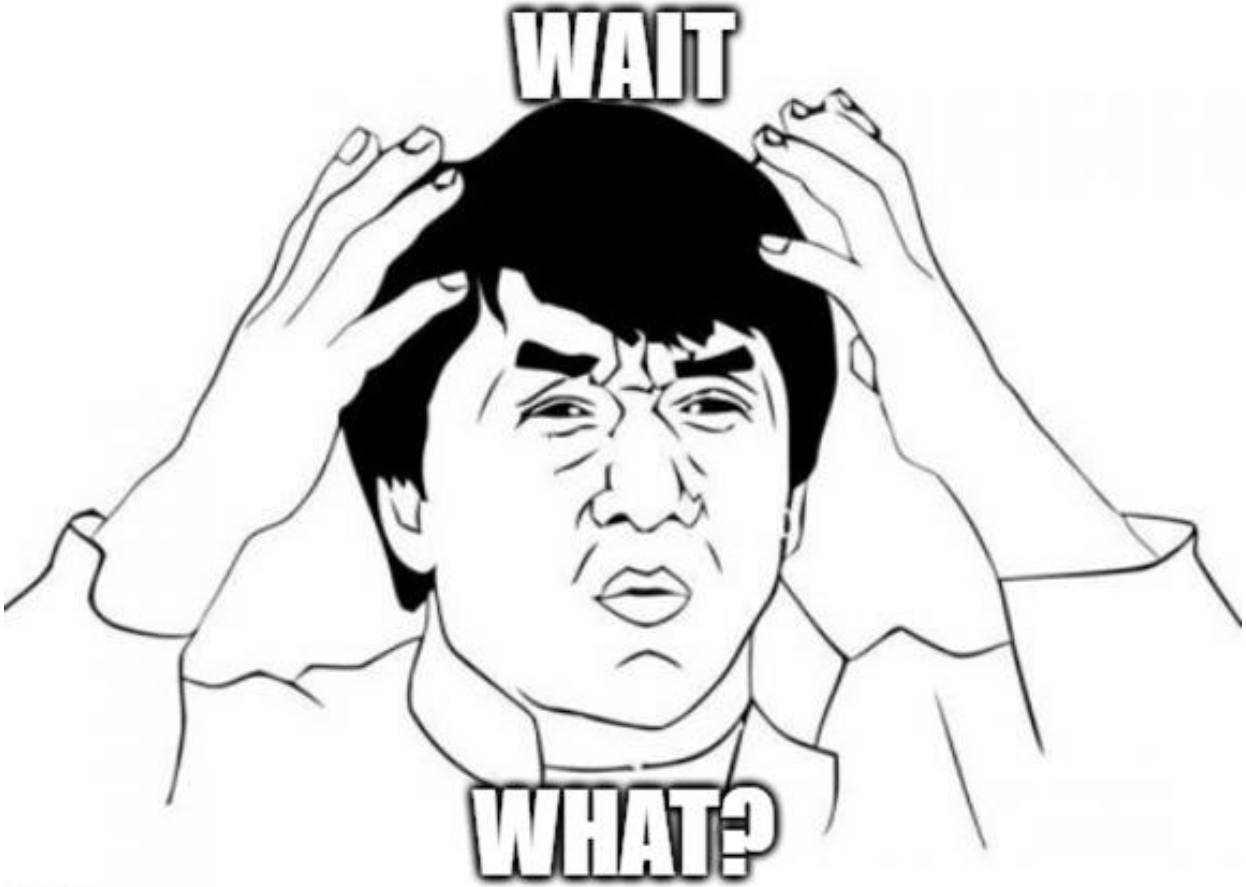
When running

```
docker run --isolation=process
--user="ContainerUser" {IMAGE}
cmd.exe
```

We see NT/System users in the container processes even though we executed the container with a weak user!



Container's process list from the host with NT/System users


WAIT
WHAT?

Gain system permissions using malicious image

1. Run container as system
2. Register service that will run as NT/System
3. Start the service
4. Store the container as a new image

Privilege escalation

- Modifications of filesystem permissions
- Scheduled task
- Modifications of the permissions of the weak user.
- 1-day vulnerability in the image
- And more!

A technique for finding container vulnerabilities

Past container escape vulnerabilities

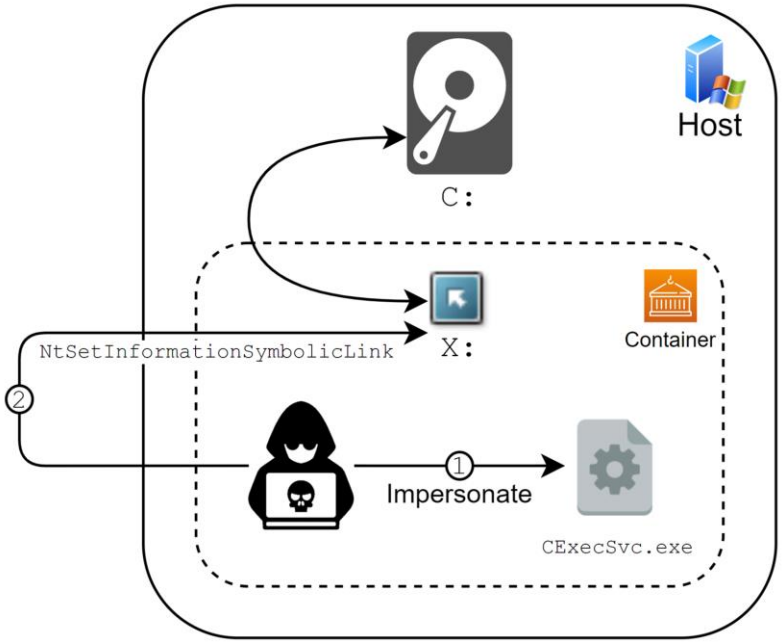
James Forshaw, Project Zero

Bypass existing validations

```
PS> $root = Get-NtDirectory "\"
PS> $root.FullPath
\
PS> $silo = New-NtJob -CreateSilo
-NoSiloRootDirectory
PS> Set-NtProcessJob $silo -
Current
PS> $root.FullPath
\Silos\748
```

Daniel Prizmant, Unit42

Missing validations



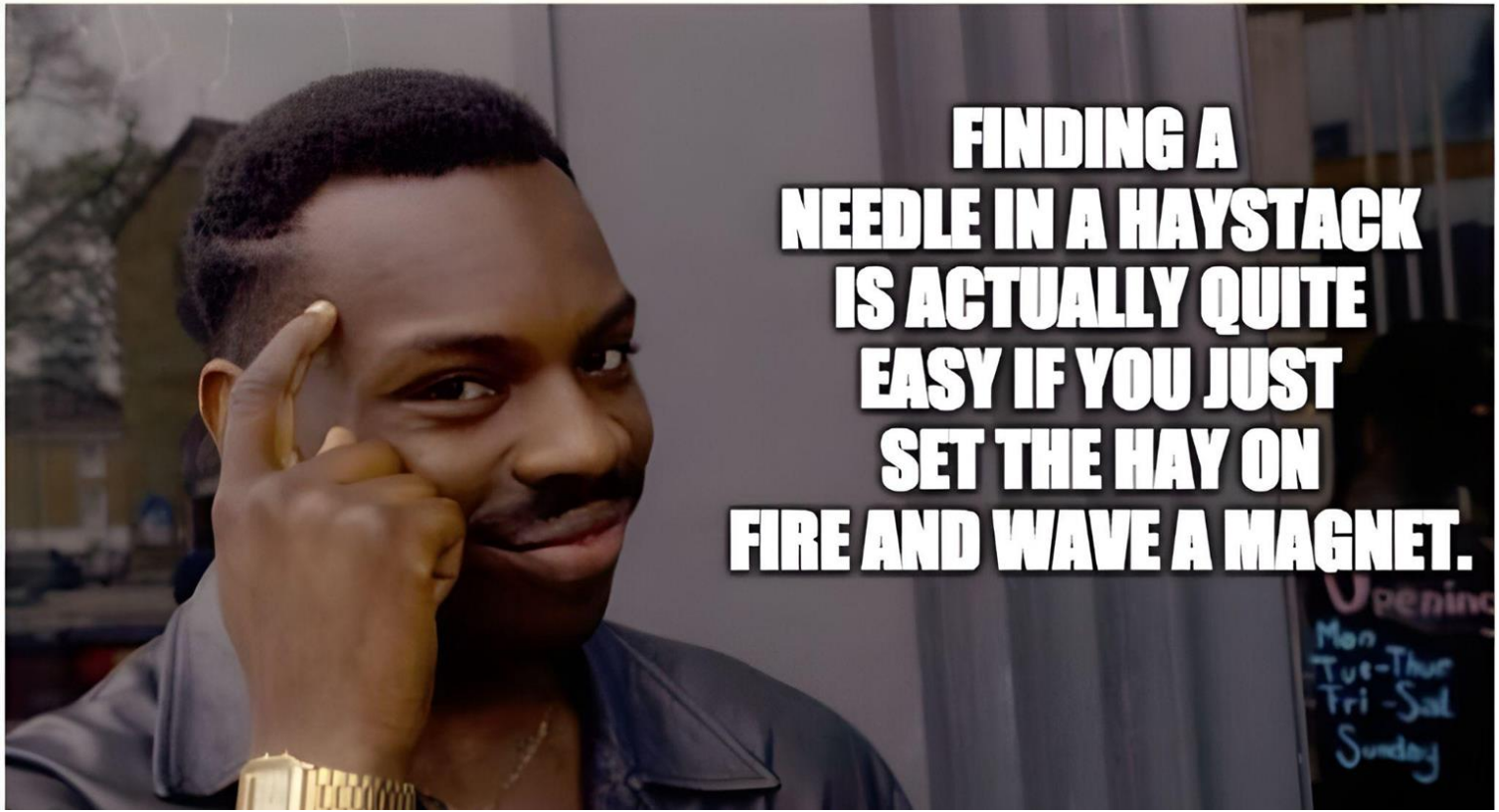
Interesting, high-odds functions

1. Syscall Functions (start with NT)

2. No isolation checks (doesn't check silo or silo server)
   The isolation functions are not called from the syscall function

3. Requires admin privileges
   The syscall function calls the function `SeSinglePrivilegeCheck`

Looking for vulnerable syscalls



NtQuerySystemInformation

```
__kernel_entry NTSTATUS NtQuerySystemInformation(
  [in]       SYSTEM_INFORMATION_CLASS SystemInformationClass, // Enum
  [in, out]  PVOID                    SystemInformation,
  [in]       ULONG                    SystemInformationLength,
  [out]      PULONG                   ReturnLength
);
```

NtQuerySystemInformation(SystemHandleInformation)

PID	Handle	Address	Granted Access	Flags
4	0000000000000004	FFFFC68BB16D6080	00000000001FFFFF	0000000000000000
4	0000000000000008	FFFFC68BB16E2140	00000000001FFFFF	0000000000000000
4	000000000000000C	FFFF8400F63A34F0	0000000000020019	0000000000000000
4	0000000000000010	FFFFC68BB16B7C80	00000000001F0001	0000000000000000
4	0000000000000014	FFFF8400F5C21E50	00000000000F000F	0000000000000000
4	0000000000000018	FFFFC68BD0248040	000000000000102A	0000000000000000
4	000000000000001C	FFFFC68BB16BEF80	00000000001F0003	0000000000000000
4	0000000000000020	FFFF8400F5C81D40	00000000000F000F	0000000000000000
4	0000000000000024	FFFF8400F5C99BB0	00000000000F000F	0000000000000000
4	0000000000000028	FFFFC68BB16AEE20	00000000001F0003	0000000000000000
4	000000000000002C	FFFFC68BB16AE420	00000000001F0003	0000000000000000

……

Parsed output of
`NtQuerySystemInformation(SystemHandleInformation,...)`
List of all the handles, PIDs and kernel addresses

First vulnerable syscall

`NtSystemDebugControl`

First vulnerable syscall - NtSystemDebugControl

```
NTSTATUS NtSystemDebugControl(
 SYSDBG_COMMAND    command,
 PVOID                        InputBuffer,...)

// Command can be 37 or 29
if (DebuggerDiabled && command != 29 && command != 37)
    return STATUS_DEBUGGER_INACTIVE;

Switch(command)
{
        …
    case 29:
        DbgkCaptureLiveDump(...);
        Case 37:
         DbgkCaptureLiveKernelDump(...);
        …
}
```

Kernel dump settings - `NtSystemDebugControl`

```
struct SYSDBG_LIVEDUMP_CONTROL
{
…
     PVOID DumpFileHandle;
     PVOID CancelEventHandle;
    SYSDBG_LIVEDUMP_CONTROL_FLAGS Flags;
     SYSDBG_LIVEDUMP_CONTROL_ADDPAGES
AddPagesControl;
…
}
```

`DbgkCaptureLiveKernelDump` gets the struct `SYSDBG_LIVEDUMP_CONTROL` in order to do kernel dump

Kernel dump flags - `NtSystemDebugControl`

- Use dump storage
- Compressed memory pages data
- Include Hypervisor pages
- Include user space memory pages - possible only if kernel debugger is enabled :/

How to extract passwords from kernel dump?

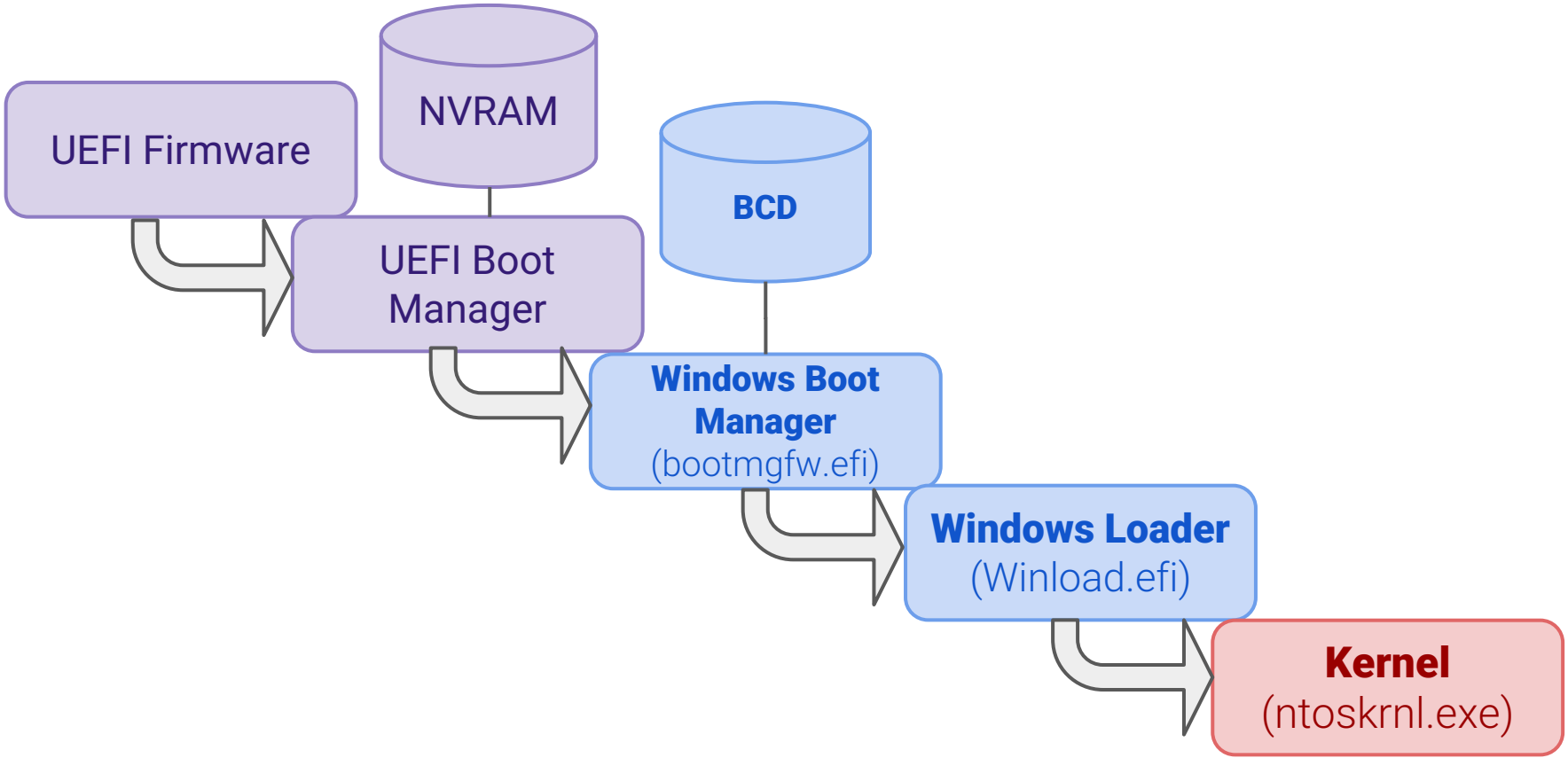
Process list

Registry hives

Lsass memory - only if kernel debugger is enabled

Background for the second vulnerability on UEFI

Windows UEFI boot sequence



Boot configuration (NVRAM)

NVRAM memory is used in UEFI to store variables between boots.

The configurations are stored on the motherboard itself.

Format of NVRAM variable:

{GUID} VARIABLE_NAME

Example:

```
8BE4DF61-93CA-11D2-AA0D-00E098032B8C BootOrder
```

Boot variables from NVRAM

Boot%d

Defines a method to boot from, such as bootmgfw.efi requires to be in FAT32 partition.

BootOrder

Defines the boot order

The container can’t control FAT32 from the container

Type of NVRAM variables

- Non-volatile
- Bootservice access
- Runtime access
- Authenticated access
- And more

Second group of vulnerable syscalls

`Nt.*SystemEnvironmentValue(Ex)`

`NtSetSystemEnvironmentValue`, `NtQuerySystemEnvironmentValue`,
`NtSetSystemEnvironmentValueEx`, `NtQuerySystemEnvironmentValueEx`
`NtEnumerateSystemEnvironmentValuesEx`

Step I - `NtEnumerateSystemEnvironmentValuesEx`

Enumerate all the variables accessible in the NVRAM memory.

Permission required: `SE_SYSTEM_ENVIRONMENT_NAME` (Admin)

Step II - `NtQuerySystemEnvironmentValue(Ex)`

Reads the value of the NVRAM variable

Permission required: `SE_SYSTEM_ENVIRONMENT_NAME` (Admin)

Step III - `NtSetSystemEnvironmentValue`

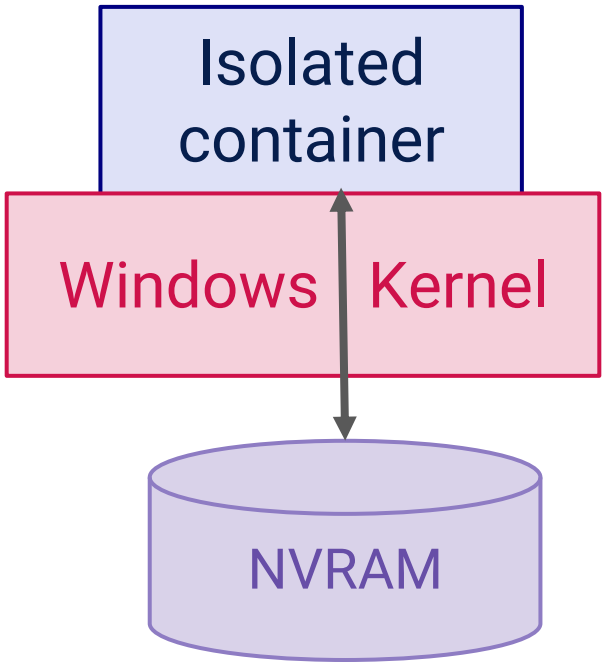
Write the value of the NVRAM variable.

Permission required: `SE_SYSTEM_ENVIRONMENT_NAME` (Admin)

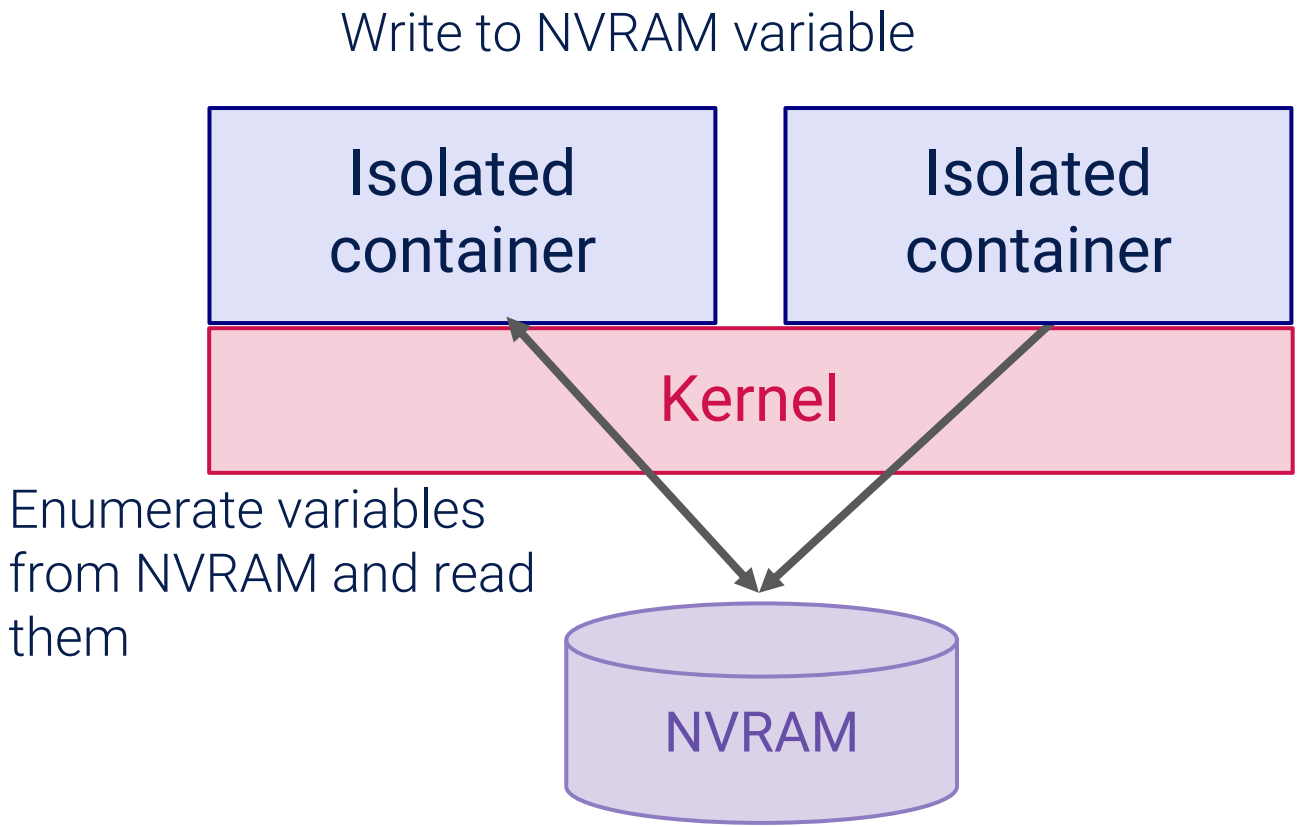
Store persistent information

It is possible to read and write from NVRAM variables.

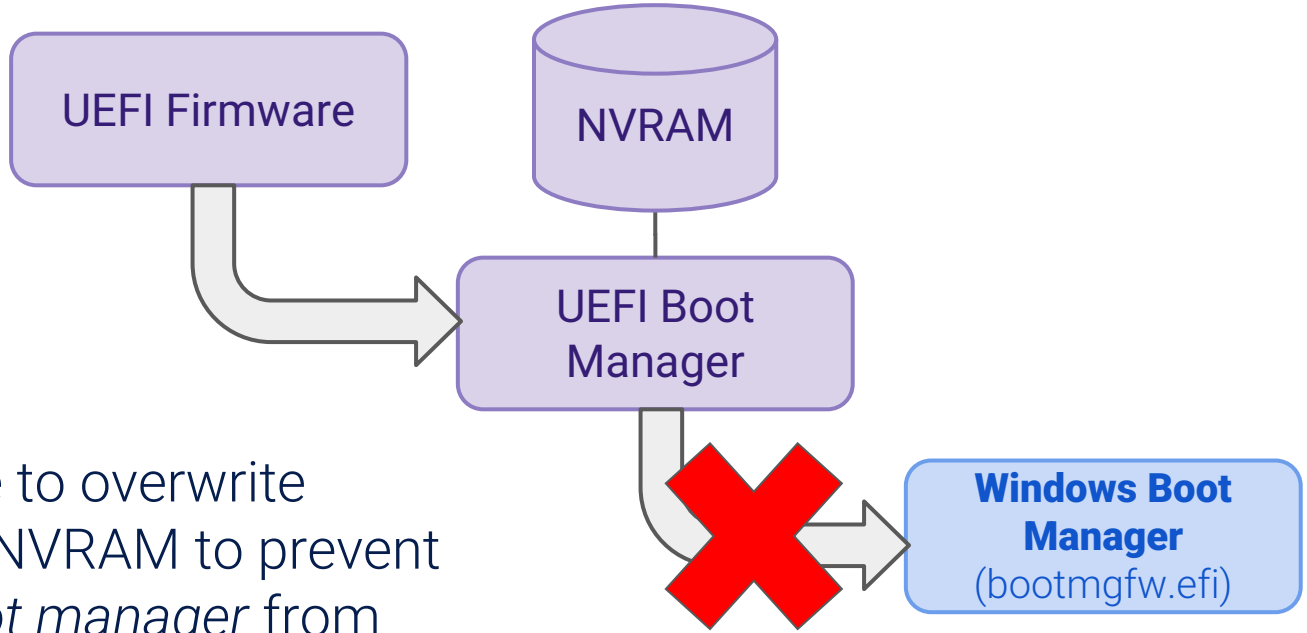
The NVRAM will keep the variables forever.



Communication between isolated containers



Permanent DoS in boot sequence



It is possible to overwrite variables in NVRAM to prevent the *UEFI boot manager* from loading *Windows boot manager*

Exploitation - permanent DoS in boot sequence

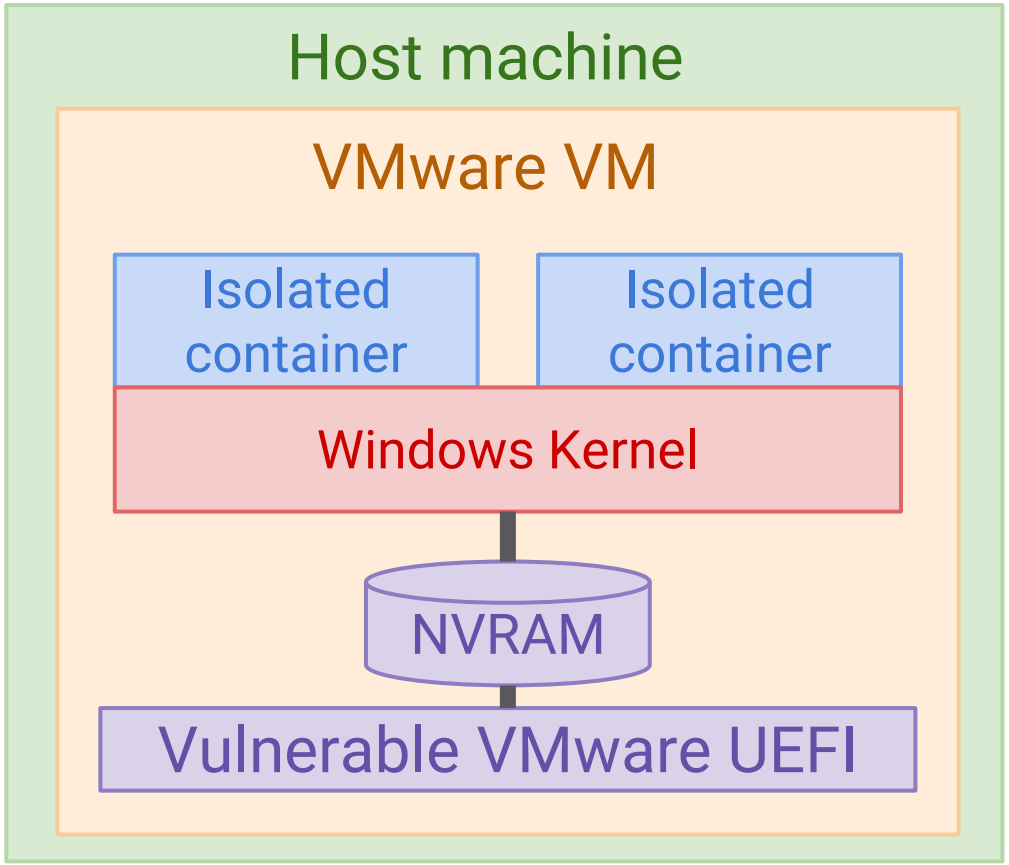
Just writing to the NVRAM variable;
`{FAB7E9E1-39DD-4F2B-8408-E20E906CB6DE} HDDP`
sequence of bytes: ‘`aaaaaa`’

`HDDP` is not referenced in all of the UEFIs

DoS root cause VMware UEFI

The root cause lies in VMware UEFI which reads the **`HDDP`** variable and stops the boot sequence.

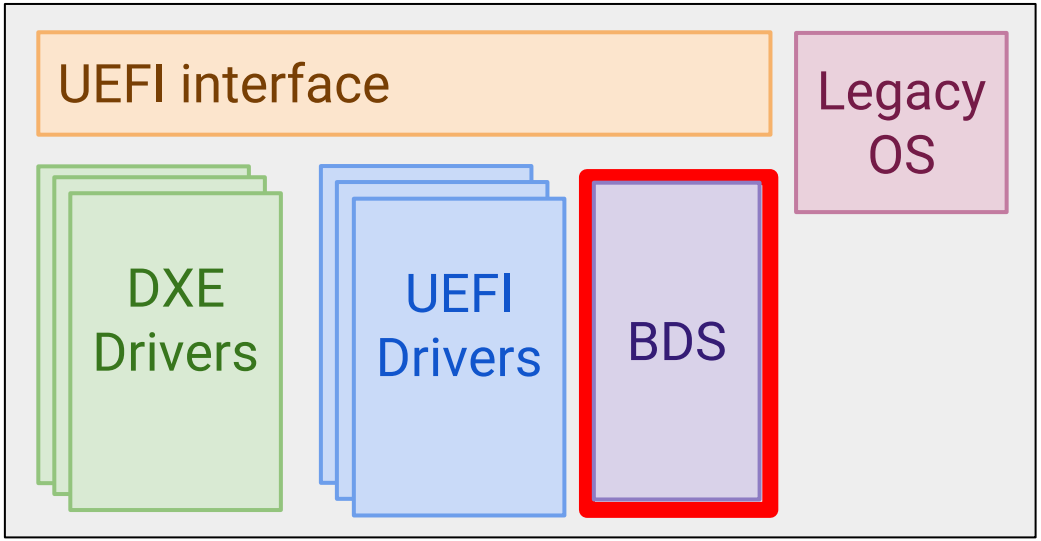
VMware UEFI is stored in the host-machine but it runs from the VM's context.



DoS root cause VMware UEFI

The root cause is found in the UEFI driver:
`BdsDxe`

Which is responsible for *Boot Device Selection* (BDS)



UEFI Firmware architecture

Root cause in `BdsDxe`

```
GetVariable2(L"HDDP",...);

if ((CachedDevicePath != NULL) && !IsDevicePathValid(..))
{
    CachedDevicePath = NULL;
    Status           = gRT->SetVariable(L"HDDP"...);
    ASSERT_EFI_ERROR (Status);
}
```

Demo


COMPUTER WORKS. REBOOT.
COMPUTER DOESN'T WORK
MAGIC
HISTORY.COM
imgflip.com

Demo explanation

1. Before the Demo I created a malicious container which contains a service that run as system

2. The service is read the command from `input.txt`
   And write the output of the command to `output.txt`

3. When it execute "`NVRAM.exe w {FAB7E9E1-39DD-4F2B-8408-E20E906CB6DE} HDDP aaaaaa`"
   It overwrote the NVRAM variable HDDP which caused the DoS

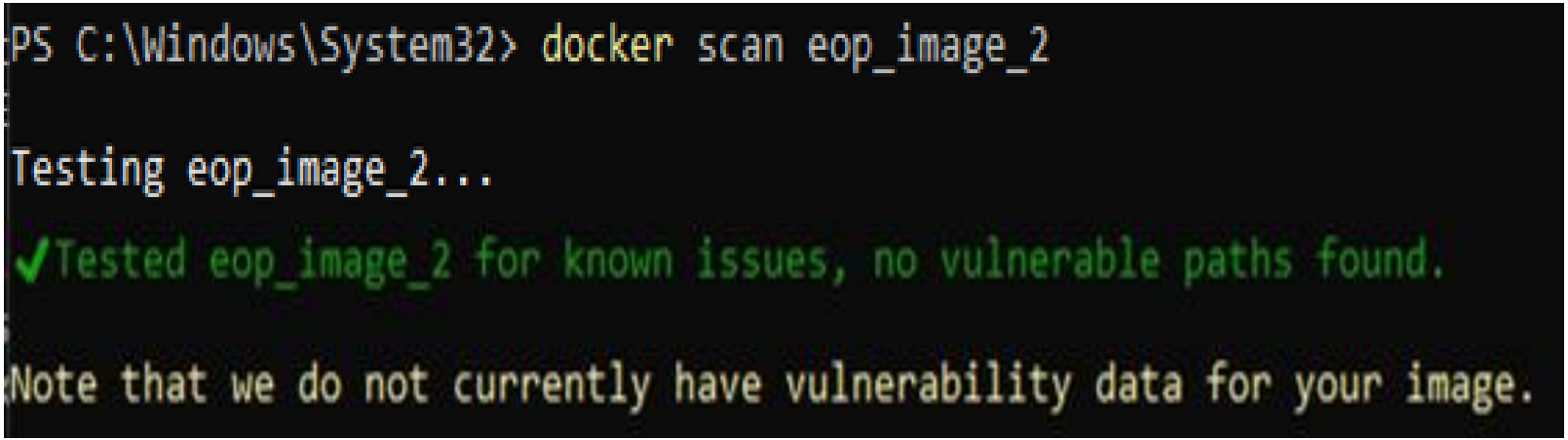
Mitigation of the vulnerabilities

- Execute Windows container with Hyper-V isolation
- Do not execute unknown container images
- Use single-tenant architecture

  Do not assume containers will provide security isolation

Saved by container image scanning?

Image scanning detects malicious images or security issues in the configurations of the image.



Microsoft responses

- **Privilege escalation using infected container image**

  “Malicious image was designed to run as System, and approved by the admin when installed to run as SYSTEM, therefore it is expected that the user would have the (malicious) container code running as SYSTEM”

- **Kernel dump from inside the container**

  “At this time, we do not know if this vulnerability will be addressed through defense-in-depth measures, or with a fix in some future release.”

Vendors’ responses (2)

Microsoft

- **List/Read/Write NVRAM variables from inside the container**

  "It was rated as a Moderate severity DoS.

  Unfortunately, that means that it is not eligible for servicing in a Windows Security Update. Engineering did recommend that a fix be considered in a future full release”

VMware

- **Prevent boot by overwriting NVRAM variable**

  “We see this to be outside our threat boundary, and it requires elevated privileges to cause DoS condition. Hence we consider this as a functional issue. We plan to address this functional issue in the future releases.”

GitHub

1. Privilege escalation container image
2. Kernel dump from inside a container
3. Permanent DoS host from inside container



https://github.com/SafeBreach-Labs/CoWTools

Acknowledgement

Thanks to Mickey Shkatov for his help with reverse engineering the VMware UEFI

Credits

1. https://qiita.com/kikuchi_kentaro/items/2fb0171e18821d402761
2. https://wikileaks.org/ciav7p1/cms/page_26968084.html
3. https://googleprojectzero.blogspot.com/2021/04/who-contains-containers.html
4. https://unit42.paloaltonetworks.com/windows-server-containers-vulnerabilities/
5. https://unit42.paloaltonetworks.com/what-i-learned-from-reverse-engineering-windows-containers/
6. https://thomasvanlaere.com/posts/2021/06/exploring-windows-containers/
7. https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/hyperv-container

SafeBreach LABS

Thank You!

Eran Segal
eran.segal@safebreach.com





Q&A