



Running Rootkits Like A Nation-State Hacker

Omri Misgav

CTO, Security Research Group @ Fortinet IL

D3FCØN

Agenda

- Introduction and background
- Existing protections
- New techniques (with live demos)
- Mitigation idea
- Conclusions



Introduction

What Is Driver Signing Enforcement (DSE)

- Code Integrity was first introduced over 15 years ago
- Kernel drivers must be digitally signed on x64-based Windows
- Drivers are checked each time they are loaded into memory to be executed
- Allows to improve the security of the OS
 - No drivers from unknown\untrusted origin
 - No drivers that were modified post build, possibly by privileged malicious actor



Introduction

Code Integrity Internals

- ntoskrnl.exe works with an additional kernel library Cl.dll (Code Integrity)
- During boot ntoskrnl.exe first calls Cl.dll
 - nt!Phase1Initialization -> nt!SeInitSystem -> ... -> nt!SepInitializeCodeIntegrity -> Cl!CiInitialize
 - Cl!CiInitialize gets pointer to nt!g_CiCallbacks structure, it sets Cl!g_CiOptions and fills the callbacks
 - 3 callbacks - Cl!CiValidateImageHeader, Cl!CiValidateImageData, Cl!CiQueryInformation
 - ntoskrnl.exe sets g_CiEnabled
- ntos has wrapper functions for the callbacks, like nt!SeValidateImageHeader
 - check nt!g_CiEnabled, check callback pointer is not null and invoke it
 - nt!g_CiEnabled isn't checked in nt!SeCodeIntegrityQueryInformation wrapper though
- When loading a driver, validate image header and data callbacks are used
 - nt!MmLoadSystemImage -> ... -> nt! nt!MmCreateSection -> nt!MiValidateImageHeader



Introduction

Code Integrity Internals

- From Win8 nt!g_CiEnabled variable was removed
- The callbacks structure also changed
 - Cl.dll provides more callbacks
 - Same important callbacks remain but their offsets changed
- Only validate image header is invoked when loading a driver
- From Win8.1 callbacks structure symbol changed to nt!SeCiCallbacks



Introduction

Bypassing Code Integrity

- Use valid digital certificates
 - Either stolen or validly issued by a legitimate CA
 - Starting with Win10 Redstone (August 2016) a 2nd signature from Microsoft is required
- Run shellcode\ROP-chain as outcome of exploitation
 - Not comfortable as just building a driver
 - Can reflectively load a driver as next stage, but leads to other challenges
 - Undocumented APIs: prepare driver object
 - Memory forensic artifacts: no driver module entry, no file mapping, discardable sections remain after initialization
 - Additional heavy lifting: exception handling, protections on callbacks registration
- Disable DSE during runtime instead



Introduction

DSE Tampering in the Wild

- Change the value of `CI!g_CiOptions` or `nt!g_CiEnabled` (“Flag Flipping”)
- Symbols are located by simple pattern matching
 - Minimal changes from between all Windows versions
- Overwrite and quickly proceed to load unsigned driver
- Once finished restore the flags back to their original state to avoid PatchGuard
- Bring Your Own Driver (BYOD) to gain kernel write primitive
 - With vulnerable or poorly written code that breaks security boundaries between user and kernel-mode
 - Pros: portable between OS versions, reusable after patching
 - Cons: Require administrative privileges, artifacts on disk



Protections

Kernel Patch Protection (KPP, PatchGuard)

- Feature of x64 editions of Windows first introduced in 2005
- Prevents modifying the kernel (for long time periods)
 - The code of ntoskrnl.exe and other critical system drivers and data structures
 - Periodically checking these protected targets for changes
 - BSOD is triggered upon detection
- Updated with each new OS release hence bypasses likely won't be universal
 - Callbacks structure in ntoskrnl.exe from Win8
 - Cl!g_CiOptions from Win8.1



Protections

Driver Blocklists

- Reduce the attack vector – make it harder to gain a write primitive
- Self-descriptive
- Enforced via Windows Defender Application Control (WDAC) or Secure Kernel (VTL1)
 - Several other third-party security product vendors adopted this practice as well
 - The latest list can be found [here](#)
- Only previously discovered drivers can be handled
 - Effective, but not a proactive measure



Protections

Kernel Data Protection (KDP)

- Reduce the attack surface – prevent changes to Cllg_CiOptions
- Protect drivers in the Windows kernel against data-driven attacks
 - Statically: a section of the PE
 - Dynamically: allocated pool memory
- First [introduced](#) in Win10 20H1
- Memory is marked by Secure Kernel (VTL1) as protected using the SLAT PTEs
- Does not enforce how the GVA range mapping of a protected region is translated
 - Verifies only on a periodic basis that it translates to the appropriate GPAs





New Techniques



New Techniques

DSE Tampering Procedure

1. Locate
2. Overwrite
3. Load
4. Revert



New Techniques

Page Swapping

- Principal – KDP doesn't enforce PFN, only permissions
 - Swap GPA (PFN in PTE) to a GPA we control
- Finding Cl!g_CiOptions is just as the same
- For the smallest number of reads use the virtual addresses of the PTEs
- PTEs addresses are randomized (KASLR)
 - Find the base address with a single read [Turning \(Page\) Tables](#), BSidesLV 2019 (slide 18)



New Techniques

Page Swapping

- Read PTE base
- Allocate a new, writable page
- Copy page contents
- Modify page content (CI!CiOptions value)
- Read original PFN from PTE for CI!g_CiOptions page
- Read PFN from PTE for our page
- Write PFN to PTE for CI!g_CiOptions page
- Load driver
- Restore original PFN in PTE

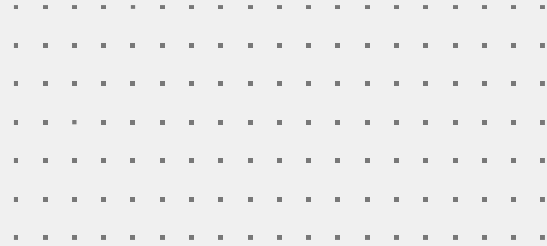


New Techniques

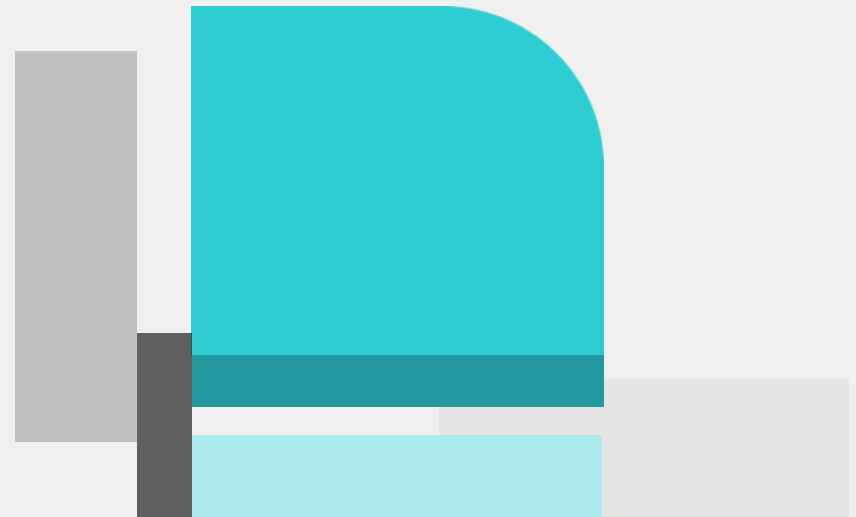
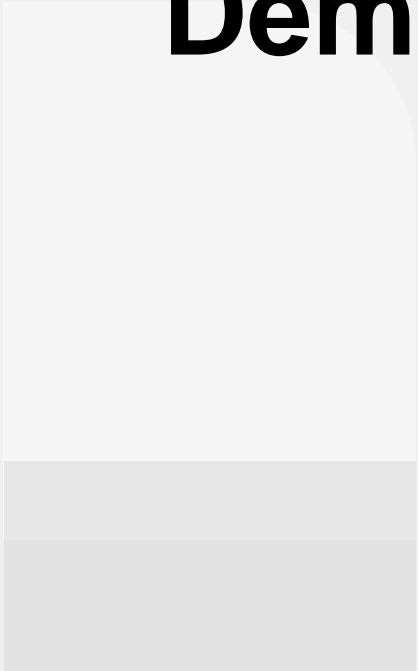
Page Swapping

- Read PTE base
- Allocate a new, writable **user-space** page
- **Initialize** page content with CI!CiOptions disabled
- Read PFN from PTE for CI!g_CiOptions page
- Read PFN from PTE for our page
- Write PFN to PTE for CI!g_CiOptions page
- Load driver
- Restore original PFN in PTE





Demo



New Techniques

Comparison

	Flag Flipping	Page Swapping		
		Naïve	User Page	DSE Specific
Read	0	Page + 3	Page + 3	3*
Memory Allocation	0	Page	0	0
Write	1	Page + 1	1	1

* Assuming default values



New Techniques


Callback Swapping

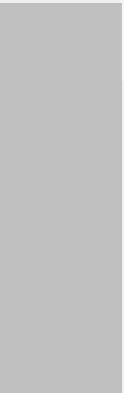
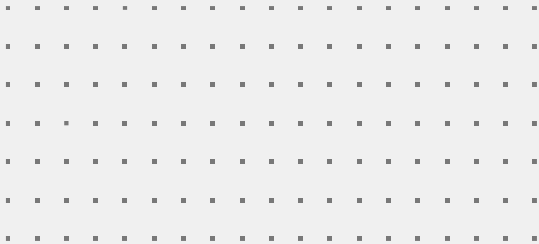
- Principal – Replace Cl.dll
 - Change addresses in the callbacks structure in ntos
 - No need to find Cl!g_CiOptions
- Cl validation routines returns 0 (usually STATUS_SUCCESS)



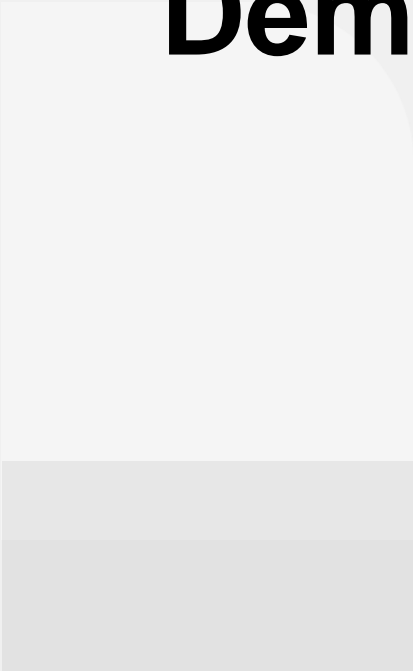
New Techniques

Callback Swapping

- Find the callbacks structure in ntoskrnl.exe
 - Find the call to C!CilInitialize by reference to the Import Lookup Table (ILT) entry
 - Walk back to register assignment of parameter pointing to uninitialized memory .data section (lea r8\r9, offset ...)
- Find new callback target
 - In ntos, exported functions like nt!FsRtlSyncVolumes\ZwFlushInstructionCache
 - In Cl.dll - exported functions like C!GetDR7Value or “xor eax,eax ret” gadgets
- Finding the original callbacks
 - Search the pattern of instructions that sets the callbacks structures in C!CipInitialize
 - Verify the addresses by traversing unwind information
- Swap the callback\s to the new target
- Load driver
-  Restore the callback\s back to the originals



Demo



New Techniques

Updated Comparison

	Flag Flipping	DSE Specific Page Swapping	Callback Swapping	
			Vista-Win7	Win8+
Read	0	3*	0	0
Memory Allocation	0	0	0	0
Write	1	1	2	1

* Assuming default values



Mitigation

- Confirm the state of DSE during driver loading
- Why not finding the internal variables the same way done for tampering?
- Copy the callbacks structure
- Copy flags
 - Cl!g_CiOption
 - nt!g_CiEnabled on Win7
- Driver loading can be intercepted by
 - Hook ntdll!NtLoadDriver
 - Registry callbacks when accessing the driver's key
 - File system minifilter callback IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION
- Prevention either by blocking the \IO request in callback or restoring the variables



Summary

Conclusions

- DSE tampering is still feasible
- Data-oriented mitigations are intricate to implement
 - “Robustly preventing this is non-trivial...” – Matt Miller, MSRC [[BlueHatIL 2019](#)]
- Defenders should adopt suggested attestation as defense in-depth
- HVCI is the real solution for this case, enable it!
 - Introduced 7 years ago, with VBS, as Device Guard
 - Runs independently from KDP
 - Eliminate the attack surface – prevent any code from running in the kernel without being validated first in the Secure World (VTL1) by SKCI.DLL (Secure Kernel Code Integrity)
- Won't be completely obsolete due to misconfigured and legacy systems





Questions?

 [in/omri-misgav](https://www.linkedin.com/in/omri-misgav)

