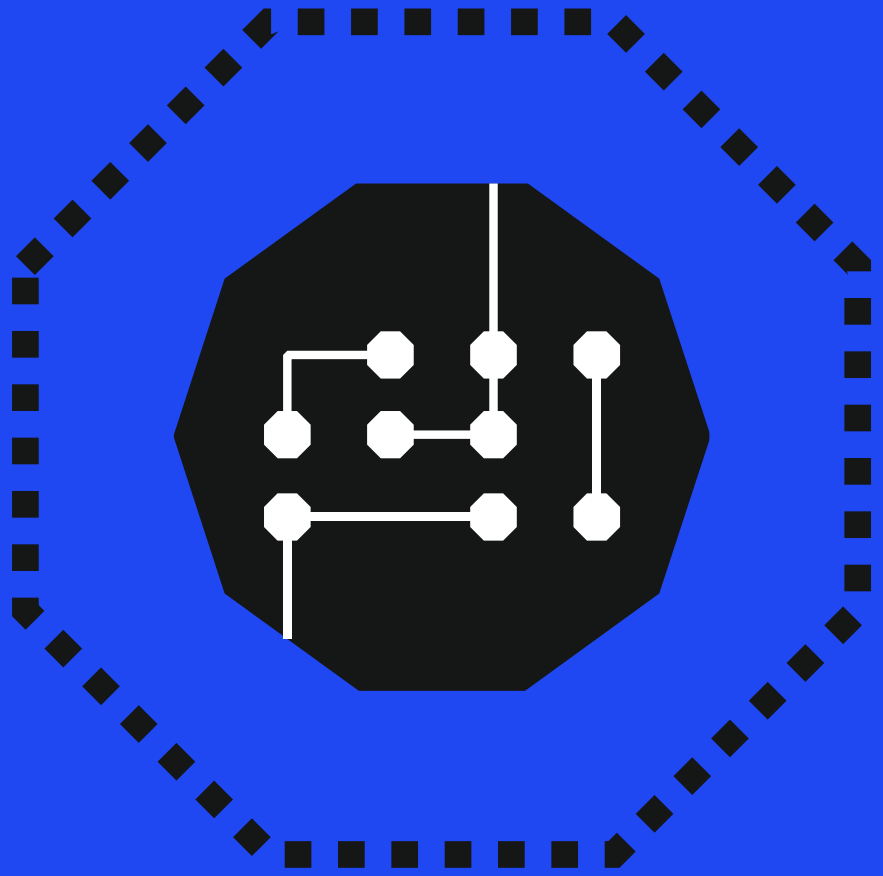
Psychedelic Root(Kit)s: Reshaping Boundaries of Perception to Make the EDR Feel Happy



Who Am I?

Dimitri (GLenX) Di Cristofaro

@d_glenx

Senior Security consultant and researcher @ SECFORCE LTD

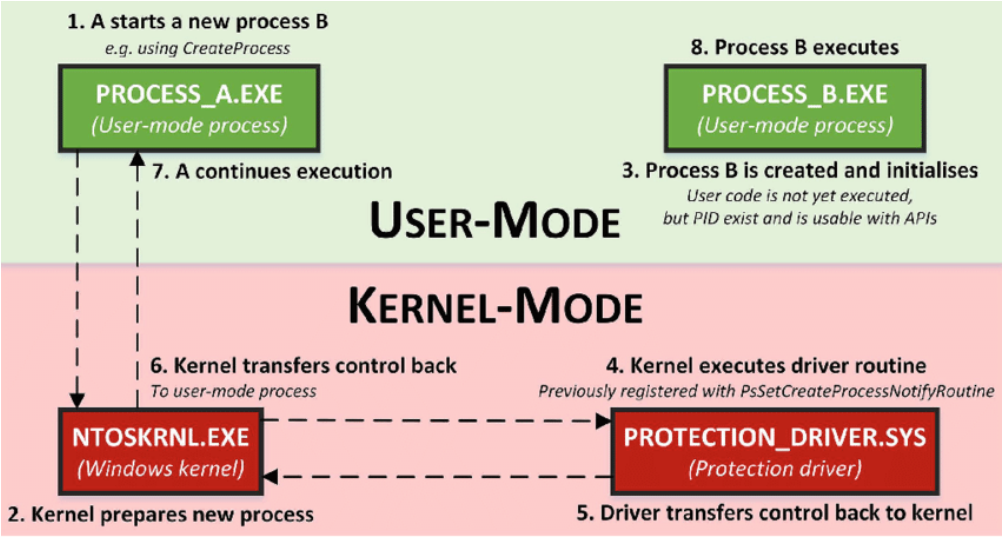
- Red Teaming
- Malware development
- OS Internals

Dynamic Analysis – Behavioural Detection

Kernel Callbacks

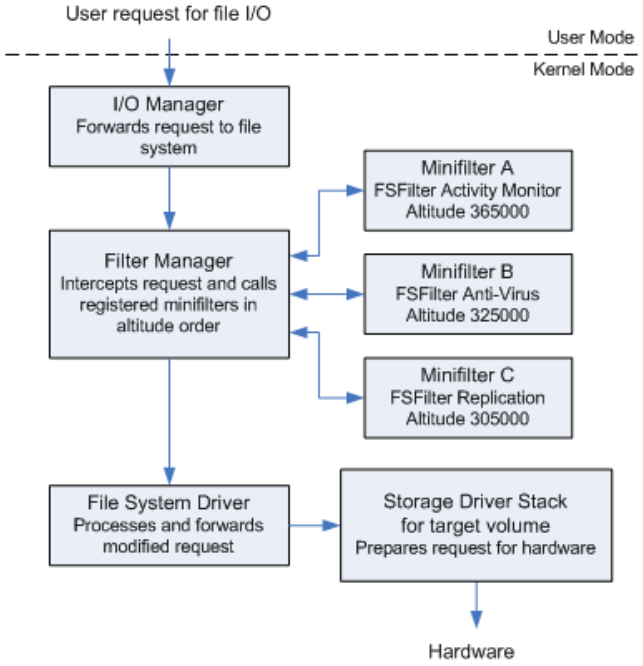
- PsSetCreateProcessNotifyRoutine
- PsSetCreateThreadNotifyRoutine
- PsSetLoadImageNotifyRoutine

Src: Fast and Furious: Outrunning Windows Kernel Notification Routines from User-Mode
http://dx.doi.org/10.1007/978-3-030-52683-2_4



Minifilters

- I/O Activity
- Execution of queued minifilters based on Altitude value



Src: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts

Dynamic Analysis – ETW Ti

`Microsoft-Windows-Threat-Intelligence` ETW provider (or `EtwTi` in short)

Kernel instrumented to log calls to some routines (Memory Management, APC, Threads, etc.)

Clients registered to this provider will get notified when a number of potentially malicious actions are executed (e.g. Memory Allocation, Memory Read, APC Queued, etc.).

Get more Information about the `EtwTi` provider:

```
logman.exe query providers Microsoft-Windows-Threat-Intelligence
```

```
C:\Users\admin>logman.exe query providers Microsoft-Windows-Threat-Intelligence

Provider                                  GUID
-------------------------------------------------------------------------------
Microsoft-Windows-Threat-Intelligence     {F4E1897C-BB5D-5668-F1D8-040F4D8DD344}

Value               Keyword               Description
-------------------------------------------------------------------------------
0x0000000000000001  KERNEL_THREATINT_KEYWORD_ALLOCVM_LOCAL
0x0000000000000002  KERNEL_THREATINT_KEYWORD_ALLOCVM_LOCAL_KERNEL_CALLER
0x0000000000000004  KERNEL_THREATINT_KEYWORD_ALLOCVM_REMOTE
0x0000000000000008  KERNEL_THREATINT_KEYWORD_ALLOCVM_REMOTE_KERNEL_CALLER
0x0000000000000010  KERNEL_THREATINT_KEYWORD_PROTECTVM_LOCAL
0x0000000000000020  KERNEL_THREATINT_KEYWORD_PROTECTVM_LOCAL_KERNEL_CALLER
0x0000000000000040  KERNEL_THREATINT_KEYWORD_PROTECTVM_REMOTE
0x0000000000000080  KERNEL_THREATINT_KEYWORD_PROTECTVM_REMOTE_KERNEL_CALLER
0x0000000000000100  KERNEL_THREATINT_KEYWORD_MAPVIEW_LOCAL
0x0000000000000200  KERNEL_THREATINT_KEYWORD_MAPVIEW_LOCAL_KERNEL_CALLER
0x0000000000000400  KERNEL_THREATINT_KEYWORD_MAPVIEW_REMOTE
0x0000000000000800  KERNEL_THREATINT_KEYWORD_MAPVIEW_REMOTE_KERNEL_CALLER
0x0000000000001000  KERNEL_THREATINT_KEYWORD_QUEUEUSERAPC_REMOTE
0x0000000000002000  KERNEL_THREATINT_KEYWORD_QUEUEUSERAPC_REMOTE_KERNEL_CALLER
0x0000000000004000  KERNEL_THREATINT_KEYWORD_SETTHREADCONTEXT_REMOTE
0x0000000000008000  KERNEL_THREATINT_KEYWORD_SETTHREADCONTEXT_REMOTE_KERNEL_CALLER
0x0000000000010000  KERNEL_THREATINT_KEYWORD_READVM_LOCAL
0x0000000000020000  KERNEL_THREATINT_KEYWORD_READVM_REMOTE
```

Reading ETW Ti Logs

- Sealighter is a tool that allows to parse ETW events - https://github.com/pathtofile/Sealighter

Limitations:

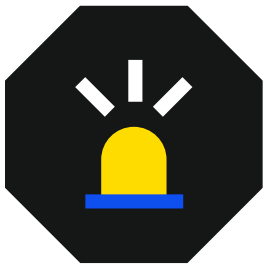
- Only PPL-AntiMalware processes can register to the ETWTi provider
- Running as PPL-AntiMalware is allowed by MS after the binary has been analysed and signed by MS
- PPL-Dump exploit could be used - https://github.com/itm4n/PPLdump. Patched on Windows 10 v21H2 Build 19044.1826 and upwards.
- We need to elevate the process to PPL from the kernel. We can use a vulnerable driver!

Reading ETW Ti Logs

- Sealighter is a tool that allows to parse ETW events - https://github.com/pathtofile/Sealighter

- Only
  regist
- Runn
  MS afte
  signed by MS

- We need t
  the kernel. We can use a vulnerable driver!



Stay tuned till the last part of the talk

User-space what?

We now have full power in user-space

- Depending on the target, we have the same privileges of the user who executed our payload
- If we are not Administrator/SYSTEM?

Normal user -> Administrator/SYSTEM

- Token stealing
- Spawn cmd.exe (or whatever) process from task manager/msconfig
- UAC bypass?
  - https://book.hacktricks.xyz/windows-hardening/authentication-credentials-uac-and-efs/uac-user-account-control
  - https://github.com/hfiref0x/UACME

Kernel Drivers vs DSE

- Loading code inside the Windows Kernel is not an easy task
  - Even with Administrator/SYSTEM privileges
- **D**river **S**ignature **E**nforcement – **DSE**
- **g_CiOptions** is a kernel configuration variable used by PatchGuard, a kernel protection technology introduced by Microsoft to prevent unauthorized changes to kernel structures.
- This variable is stored in **kernel memory**
- This variable controls wether it is possible to load drivers without a valid signature (i.e. co-signed by Microsoft!)
  - By default, it is not!

DSE

- **g_CiOptions** can be found in **CI.dll**, both in user and kernel-space
  - You want to overwrite the user-space one? Good luck ☺
- Good news is: the offset inside the dll is the same
- We find the offset inside CI.dll in user-space
- We find the base address of CI.dll in kernel-space
- We compute the address of **g_CiOptions** in kernel-space
- If we have write access to kernel memory, we can disable DSE

DSE

- How to get the base address of CI.dll in kernel space?
  - We can use **NtQuerySystemInformation** to get information about modules loaded into the kernel
- The hardest task is finding the offset of **g_CiOptions** in user-space
- Fortunatly the Ci kernel module exports **CiInitialize** function and you know what ? This function uses a routine named **CipInitialize** which leaks *gCiOptions* address, making offset calculation possible :-)

https://v1k1ngfr.github.io/loading-windows-unsigned-driver/#step-3---hunting-the-gcioptions

https://news.sophos.com/en-us/2020/02/06/living-off-another-land-ransomware-borrows-vulnerable-driver-to-remove-security-software/

BYOVD

(Ab)use a signed driver to execute malicious actions

- WWW primitives
- Process killer
- Handle Leaks
- Be Creative ☺

LOLDrivers project : https://www.loldrivers.io

Example - Kernel Cactus
https://github.com/SpikySabra/Kernel-Cactus
(dbutil_2_3.sys weaponization)

Microsoft implemented a mitigation using a block list of vulnerable drivers

Install your Unsigned Driver

Once we can write in Kernel memory, we can disable the signature check.

`ci.dll` is responsible for Windows Driver Signing Enforcement (DSE) management.

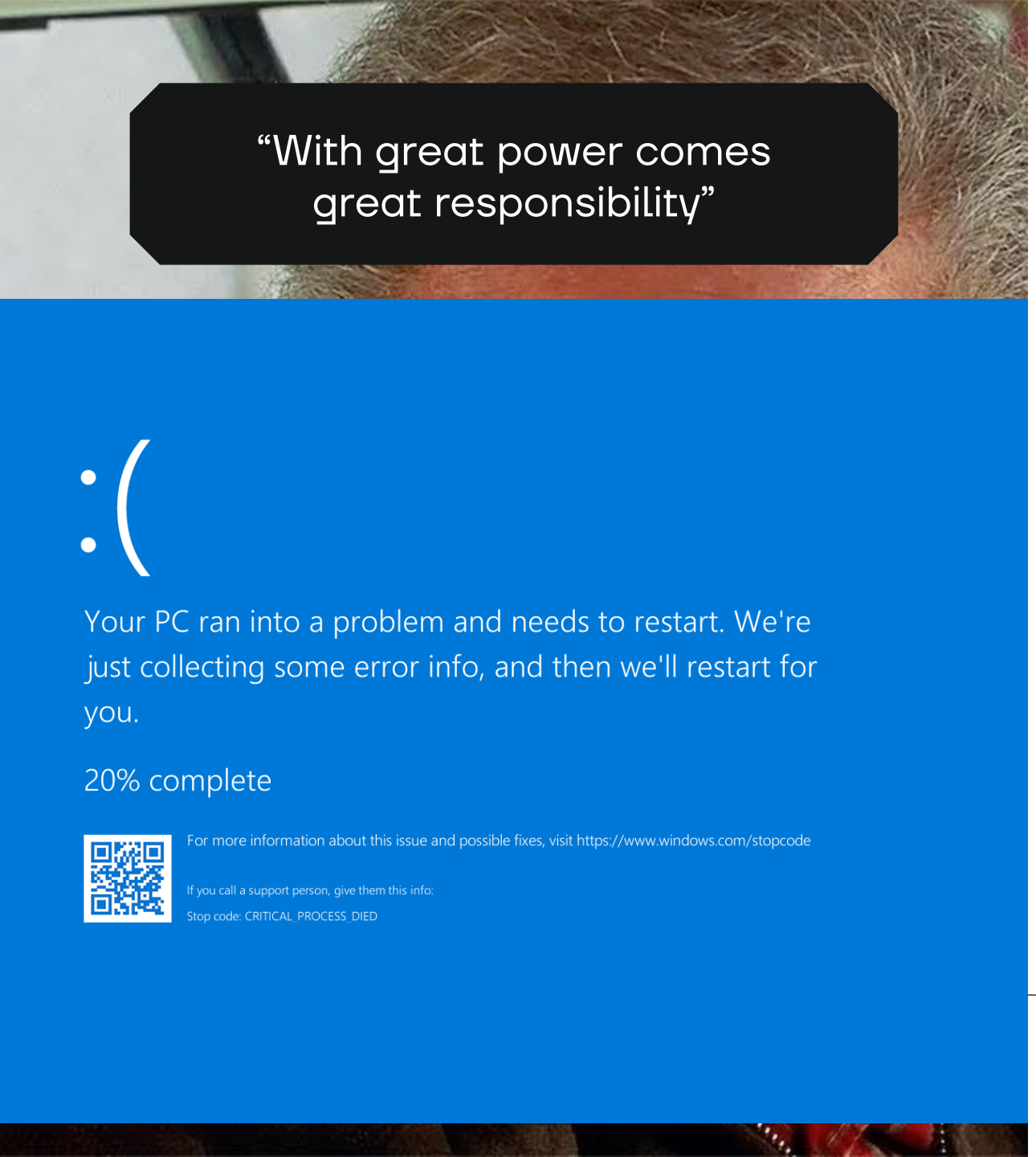
Setting `Ci!g_CiOptions` to `0` will allow us to load our driver

Windows KPP (Parch Guard) <u>periodically checks for tampering</u> of Kernel memory

Install your Unsigned Driver

- Find `Ci!g_CiOptions`
- Set `Ci!g_CiOptions` to `0`
- Install Driver
- Set `Ci!g_CiOptions` value back

References:
[Loading unsigned Windows drivers without reboot]



Install your Unsigned Driver

- Find `Ci!g_CiOptions`
- Set `Ci!g_CiOptions` to `0`
- Install Driver
- Set `Ci!g_CiOptions` value back

- **If KPP check happens at the wrong time, GAME OVER**

References:
[Loading unsigned Windows drivers without reboot]

Install your Unsigned Driver

- Find `Ci!g_CiOptions`
- Set `Ci!g_CiOptions` to `0`
- Install Driver
- Set `Ci!g_CiOptions` value back

The Symbol `Ci!g_CiOptions` is not exported :(

We need to find the address using an offset from an exported symbol (Windows version dependent)

References:
[Loading unsigned Windows drivers without reboot]

Agent Killer

1. Get handle to the target process from killer process using `NtOpenProcess()`
2. Find in kernel `EPROCESS` structure associated to the killer process
3. Get the Handle Table `HANDLE_TABLE *ObjectTable`
4. Search the Handle associated to the target process
5. Make the handle privileged by editing `HANDLE_TABLE_ENTRY->GrantedAccess`
6. Call `TerminateProcess()`

References: [Kernel Cactus]

Agent Killer-PPL

- `EPROCESS` struct holds the protection level associated to the process
- We can manipulate the protection level by editing the `Protection` (`struct _PS_PROTECTION`)
  - <u>WinTCB:</u> `Protection.Type` = 2 , `Protection.Signer` = 6
- If we set to all 0s, we have "downgraded" the process protection and we can kill it

<u>Offsets change with Windows versions</u>

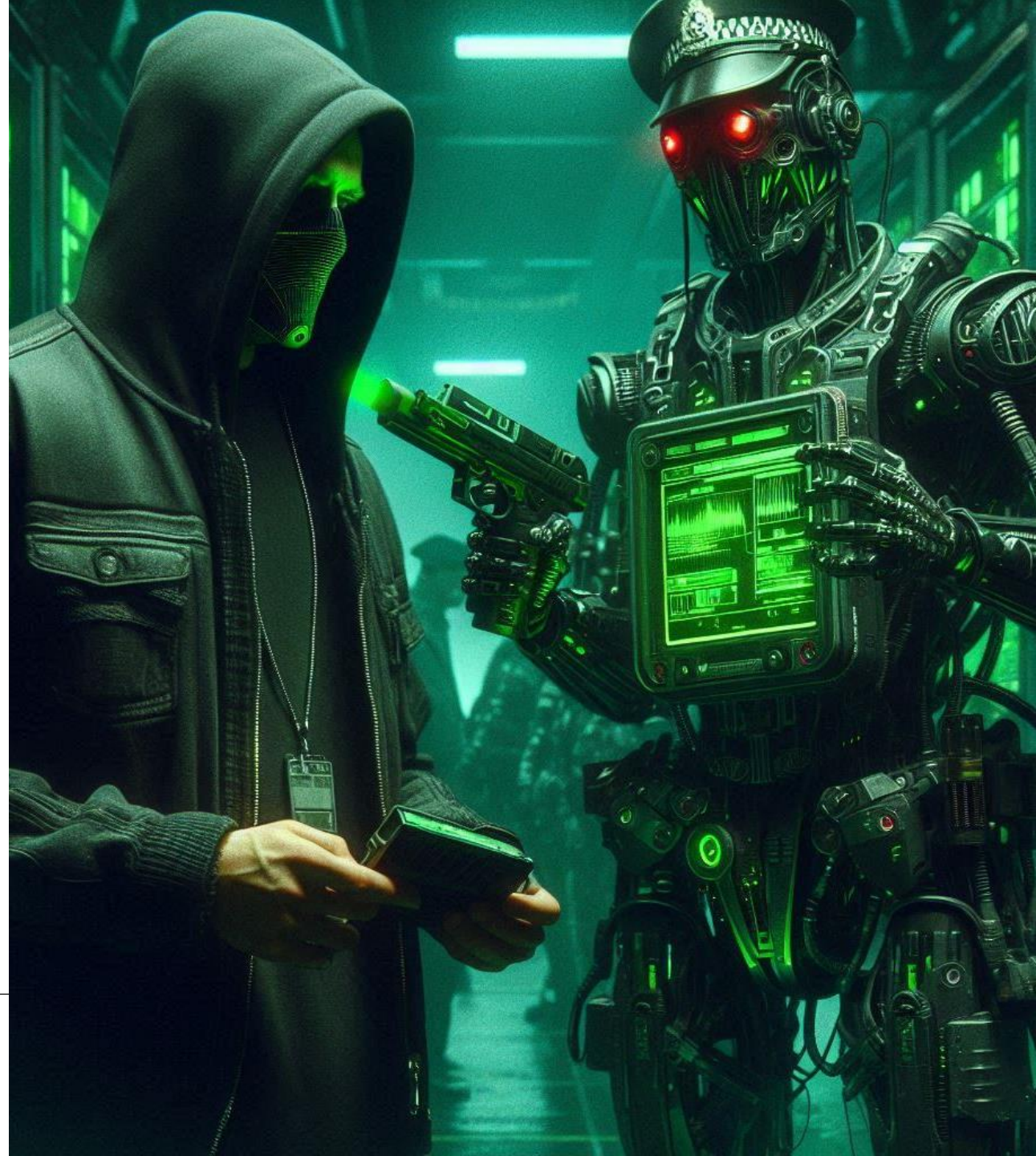
- Use Vergilius project as a reference – https://www.vergiliusproject.com

Code Integrity – Signature Level

- `EPROCESS` struct holds the protection level associated to the process
  - `SignatureLevel`
  - `SectionSignatureLevel`
- `SignatureLevel` and `SectionSignatureLevel` used by Code Integrity for signature validation

Offsets change with Windows versions

- Use Vergilius project as a reference – https://www.vergiliusproject.com



Protect Our Processes

- `EPROCESS` struct holds the protection level associated to the process
  - `Protection`
  - `SignatureLevel`
  - `SectionSignatureLevel`

- We can manipulate the protection level by editing the `Protection` (`struct _PS_PROTECTION`)

- `SignatureLevel` and `SectionSignatureLevel` used by Code Integrity for signature validation

Offsets change with Windows versions

- Use Vergilius project as a reference – https://www.vergiliusproject.com

Tamper ~~Disable~~ Kernel Callbacks

- Identify the callback array (e.g. `PspCreateProcessNotifyRoutine`)
  - Symbol NOT EXPORTED - We need to find an offset from an exported symbol (Windows version dependent) by analysing the routine that sets the values in the array (e.g. `nt!PsSetCreateProcessNotifyRoutine`)
- Identify which entry in the corresponding callback array is pointing to the EDR driver.
  - The callback handler is likely in the address space of the EDR driver
- ~~Zero the entry~~ ☺ **Overwrite pointer with our function**

Reading ETW Ti Logs

- Sealighter is a tool that allows to parse ETW events - https://github.com/pathtofile/Sealighter

- Only
  regist
- Runn
  MS afte
  signed by MS

- We need t
  the kernel. We can use a vulnerable driver!

DO YOU REMEMBER?

Read ETW Ti Logs

- Elevate the process to PPL
- Use Sealighter to register to the provider and parse the data

https://github.com/pathtofile/Sealighter

https://github.com/pathtofile/SealighterTI

https://blog.tofile.dev/2022/11/30/kdu_sealighter.html

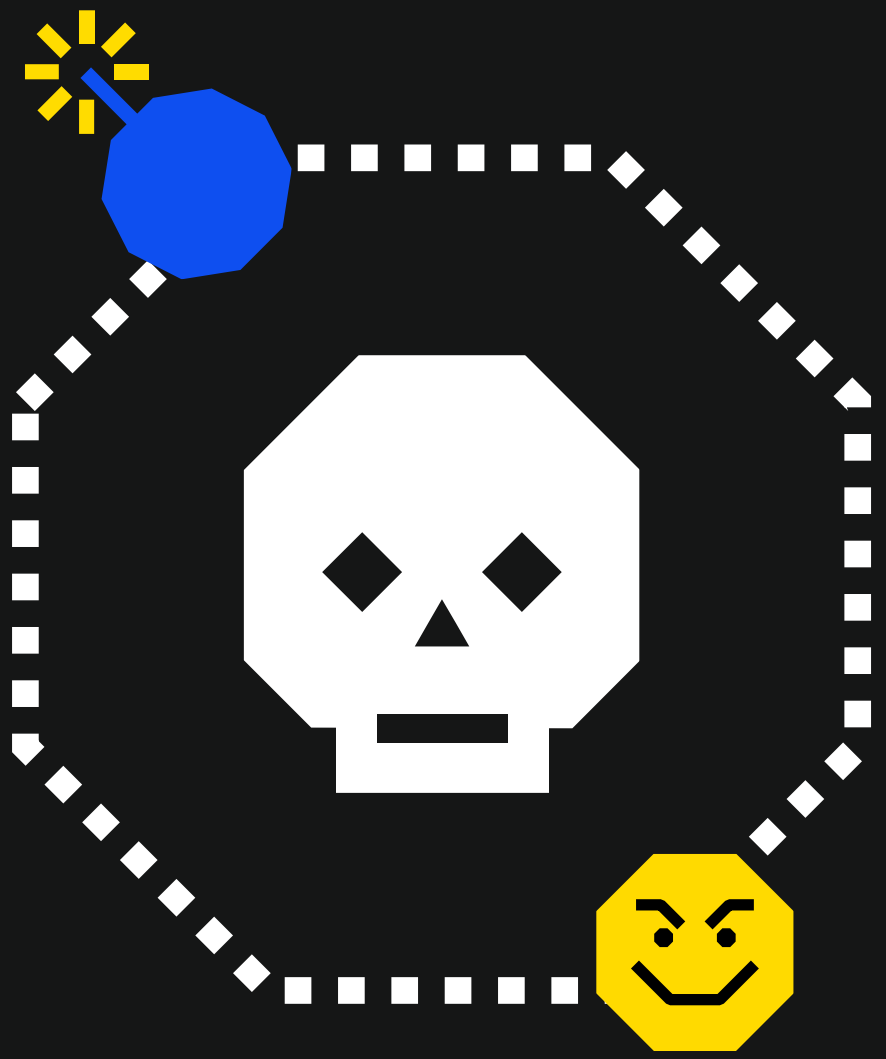
Read ETW Ti Logs


```
ask_name":"KERNEL_THREATINT_TASK_ALLOCVM","thread_id":5860,"timestamp":"2024-08-09 00:58:41Z","trace_name":"etwt
i_trace"},"properties":{"AllocationType":12288,"BaseAddress":"0x1D78D9F0000","CallingProcessCreateTime":"2024-08
-08 20:18:55Z","CallingProcessId":3032,"CallingProcessProtection":49,"CallingProcessSectionSignatureLevel":7,"Ca
llingProcessSignatureLevel":55,"CallingProcessStartKey":5348024557502537,"CallingThreadCreateTime":"2024-08-08
20:19:12Z","CallingThreadId":5860,"OriginalProcessCreateTime":"2024-08-08 20:18:55Z","OriginalProcessId":3032,"O
riginalProcessProtection":49,"OriginalProcessSectionSignatureLevel":7,"OriginalProcessSignatureLevel":55,"Origin
alProcessStartKey":5348024557502537,"ProtectionMask":64,"RegionSize":"0x4D000","TargetProcessCreateTime":
"2024-08-08 20:18:55Z","TargetProcessId":3032,"TargetProcessProtection":49,"TargetProcessSectionSignatureLevel":
7,"TargetProcessSignatureLevel":55,"TargetProcessStartKey":5348024557502537},"property_types":{"AllocationType":
```

```
,"event_opcode":0,"event_version":1,"process_id":3032,"provider_name":"Microsoft-Windows-Threat-Intelligence","t
ask_name":"KERNEL_THREATINT_TASK_PROTECTVM","thread_id":5860,"timestamp":"2024-08-09 00:58:41Z","trace_name":"et
wti_trace"},"properties":{"BaseAddress":"0x1D78D9F0000","CallingProcessCreateTime":"2024-08-08 20:18:55Z","Calli
ngProcessId":3032,"CallingProcessProtection":49,"CallingProcessSectionSignatureLevel":7,"CallingProcessSignature
Level":55,"CallingProcessStartKey":5348024557502537,"CallingThreadCreateTime":"2024-08-08
20:19:12Z","CallingThreadId":5860,"LastProtectionMask":64,"OriginalProcessCreateTime":"2024-08-08 20:18:55Z","Or
iginalProcessId":3032,"OriginalProcessProtection":49,"OriginalProcessSectionSignatureLevel":7,"OriginalProcessSi
gnatureLevel":55,"OriginalProcessStartKey":5348024557502537,"ProtectionMask":4,"RegionSize":"0x1000","TargetProc
essCreateTime":"2024-08-08 20:18:55Z","TargetProcessId":3032,"TargetProcessProtection":49,"TargetProcessSectionS
```


SECFORCE

Let’s be evil again now



ETW Providers

- ntoskrnl exports `EtwRegister` function to register a ETW provider
- During the execution of `nt!EtwRegister` a _ETW_REG_ENTRY structure is created
- _ETW_REG_ENTRY contains the information associated to an ETW provider
- Every ETW provider is identified by a GUID
- ETW Ti GUID : **f4e1897c-bb5d-5668-f1d8-040f4d8dd344**

Reference: https://securityintelligence.com/x-force/direct-kernel-object-manipulation-attacks-etw-providers/

Disable ETW Ti

- NULL the _ETW_REG_ENTRY pointer. Any functions referencing the registration handle would then assume that the provider had not been initialized.
- NULL the `_ETW_REG_ENTRY->GuidEntry.ProviderEnableInfo` pointer. This should effectively disable the provider's collection capabilities as ProviderEnableInfo is a pointer to a _TRACE_ENABLE_INFO structure which outlines how the provider is supposed to operate.
- `_ETW_REG_ENTRY->GuidEntry.ProviderEnableInfo.IsEnabled = 0`

Reference: https://securityintelligence.com/x-force/direct-kernel-object-manipulation-attacks-etw-providers/

Disable ETW Ti

https://securityintelligence.com/x-force/direct-kernel-object-manipulation-attacks-etw-providers/

https://web.archive.org/web/20220703151911/https://public.cnotools.studio/bring-your-own-vulnerable-kernel-driver-byovkd/exploits/data-only-attack-neutralizing-etwti-provider#neutralizing-etwti-provider

https://github.com/wavestone-cdt/EDRSandblast/blob/master/EDRSandblast/KernellandBypass/ETWThreatIntel.c

https://github.com/SpikySabra/Kernel-Cactus/blob/main/KernelCactus/KernelOps.cpp

Thank you!