Tales of Windows detection opportunities for an implant framework

Ollie Whitehouse
Group CTO, NCC Group

Commercial Frameworks are evolving

- in memory patching techniques
- exception handlers for code execution
- sleep routine obfuscation

… and numerous others

Framing the questions

What artefacts do implant frameworks introduce by virtue of being present?

What are the high signal detection opportunities?

Copy on Write patch detection

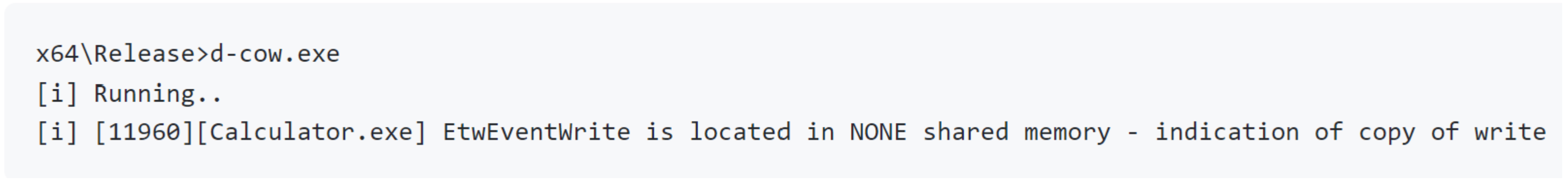
Thesis:

By default Microsoft Windows will back copies of the same DLL against the same physical memory to save space. When a patch occurs a copy on write operation will happen.

Solution:

- Open processes
- Search for the address of `EtwEventWrite`
- Use `QueryWorkingSetEx` to check the page is shared OR not
- If not then it is an indication a patch has occurred

Copy on Write patch detection - result



```
x64\Release>d-cow.exe
[i] Running..
[i] [11960][Calculator.exe] EtwEventWrite is located in NONE shared memory - indication of copy of write
```

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI/tree/master/d-cow
https://github.com/forrest-orr/moneta/blob/master/Source/Subregions.cpp

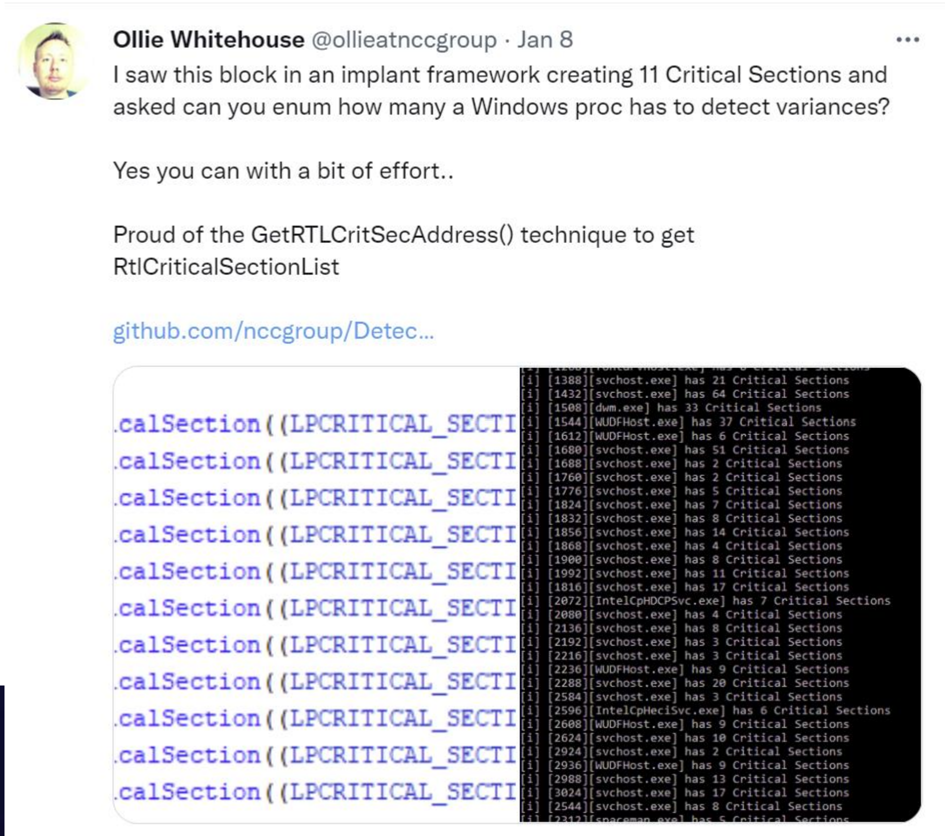
# of Critical Sections

Thesis:

If an implant needs locking it may use critical sections. We can enumerate the number of critical sections and detect variance when we have fleet level visibility for processes we expect.

Solution:

- Open processes
- Enumerate critical sections
- Detect mean/median based on fleet telemetry



Ollie Whitehouse @ollieatnccgroup · Jan 8

I saw this block in an implant framework creating 11 Critical Sections and asked can you enum how many a Windows proc has to detect variances?

Yes you can with a bit of effort..

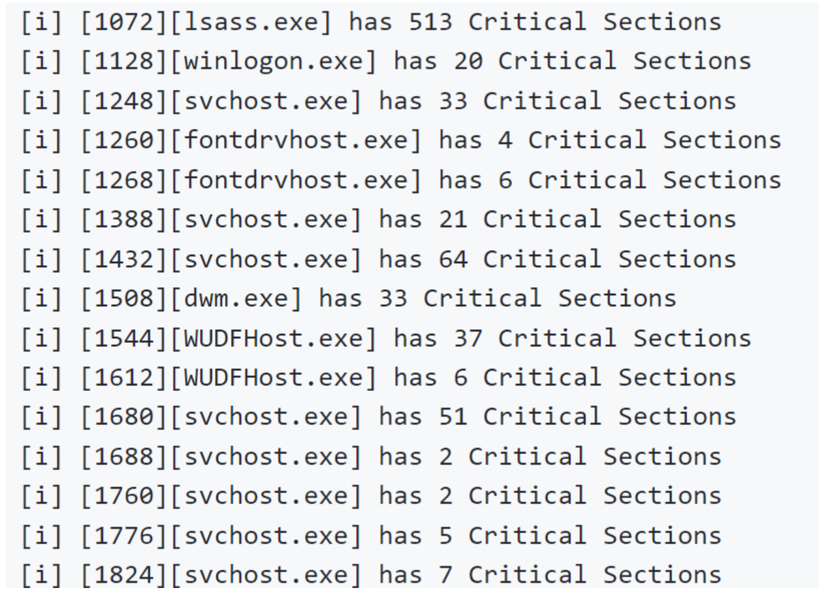
Proud of the GetRTLCritSecAddress() technique to get RtlCriticalSectionList

github.com/nccgroup/Detec...

```
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
.calSection((LPCRITICAL_SECTI
```

```
[i] [1388][svchost.exe] has 21 Critical Sections
[i] [1432][svchost.exe] has 64 Critical Sections
[i] [1508][dwm.exe] has 33 Critical Sections
[i] [1544][WUDFHost.exe] has 37 Critical Sections
[i] [1612][WUDFHost.exe] has 6 Critical Sections
[i] [1680][svchost.exe] has 51 Critical Sections
[i] [1688][svchost.exe] has 2 Critical Sections
[i] [1760][svchost.exe] has 2 Critical Sections
[i] [1776][svchost.exe] has 5 Critical Sections
[i] [1824][svchost.exe] has 7 Critical Sections
[i] [1832][svchost.exe] has 8 Critical Sections
[i] [1856][svchost.exe] has 14 Critical Sections
[i] [1868][svchost.exe] has 4 Critical Sections
[i] [1900][svchost.exe] has 8 Critical Sections
[i] [1992][svchost.exe] has 11 Critical Sections
[i] [1816][svchost.exe] has 17 Critical Sections
[i] [2072][IntelCpHDCPSvc.exe] has 7 Critical Sections
[i] [2080][svchost.exe] has 4 Critical Sections
[i] [2136][svchost.exe] has 8 Critical Sections
[i] [2192][svchost.exe] has 3 Critical Sections
[i] [2216][svchost.exe] has 3 Critical Sections
[i] [2236][WUDFHost.exe] has 9 Critical Sections
[i] [2288][svchost.exe] has 20 Critical Sections
[i] [2584][svchost.exe] has 3 Critical Sections
[i] [2596][IntelCpHeciSvc.exe] has 6 Critical Sections
[i] [2608][WUDFHost.exe] has 9 Critical Sections
[i] [2624][svchost.exe] has 10 Critical Sections
[i] [2924][svchost.exe] has 2 Critical Sections
[i] [2936][WUDFHost.exe] has 9 Critical Sections
[i] [2988][svchost.exe] has 13 Critical Sections
[i] [3024][svchost.exe] has 17 Critical Sections
[i] [2544][svchost.exe] has 8 Critical Sections
```

# of Critical Sections - result



```
[i] [1072][lsass.exe] has 513 Critical Sections
[i] [1128][winlogon.exe] has 20 Critical Sections
[i] [1248][svchost.exe] has 33 Critical Sections
[i] [1260][fontdrvhost.exe] has 4 Critical Sections
[i] [1268][fontdrvhost.exe] has 6 Critical Sections
[i] [1388][svchost.exe] has 21 Critical Sections
[i] [1432][svchost.exe] has 64 Critical Sections
[i] [1508][dwm.exe] has 33 Critical Sections
[i] [1544][WUDFHost.exe] has 37 Critical Sections
[i] [1612][WUDFHost.exe] has 6 Critical Sections
[i] [1680][svchost.exe] has 51 Critical Sections
[i] [1688][svchost.exe] has 2 Critical Sections
[i] [1760][svchost.exe] has 2 Critical Sections
[i] [1776][svchost.exe] has 5 Critical Sections
[i] [1824][svchost.exe] has 7 Critical Sections
```

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI/tree/master/d-criticalsections

Use of Vector Exception Handlers

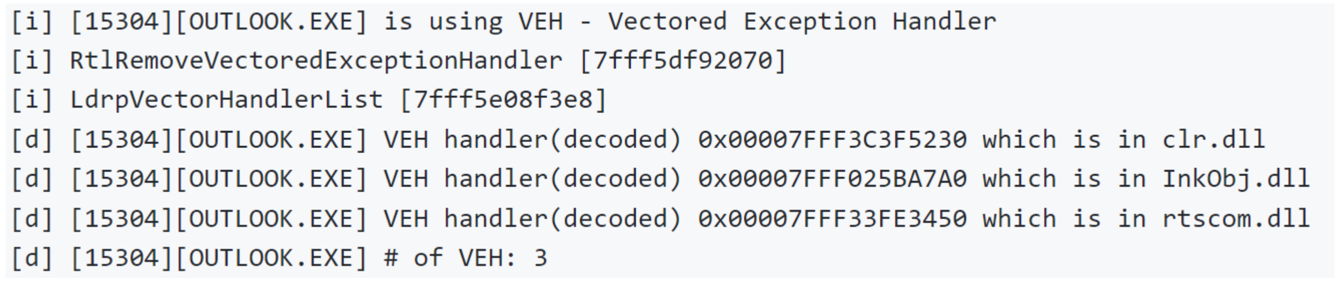
Thesis:

VEH is used to do function hooking but avoid copy on write detection when combined with hardware breakpoints. We can detect the use of VEH and enumerate where they point to in order to detect.

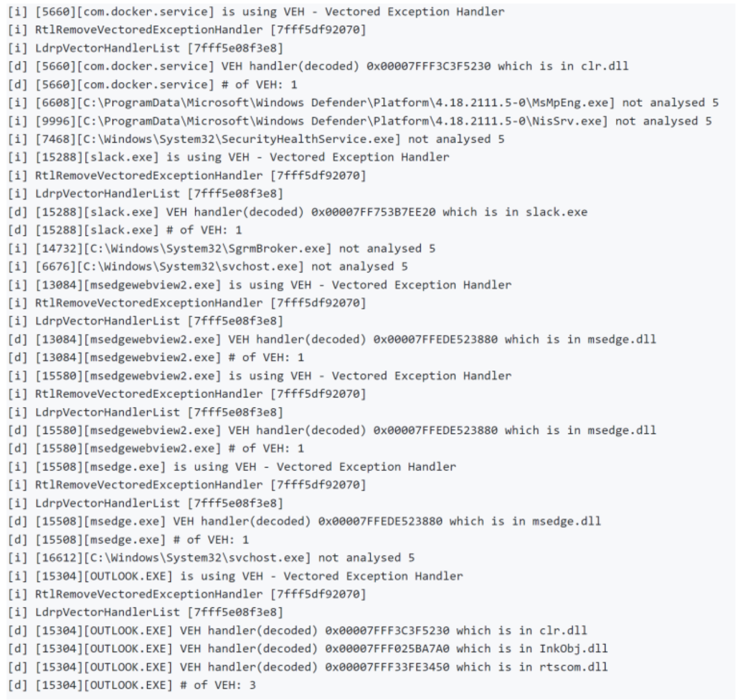
Solution:

- Open processes
- Query the PEB for VEH usage
- Load the VEH linked list and decode

Use of Vector Exception Handlers - result



```
[i] [15304][OUTLOOK.EXE] is using VEH - Vectored Exception Handler
[i] RtlRemoveVectoredExceptionHandler [7fff5df92070]
[i] LdrpVectorHandlerList [7fff5e08f3e8]
[d] [15304][OUTLOOK.EXE] VEH handler(decoded) 0x00007FFF3C3F5230 which is in clr.dll
[d] [15304][OUTLOOK.EXE] VEH handler(decoded) 0x00007FFF025BA7A0 which is in InkObj.dll
[d] [15304][OUTLOOK.EXE] VEH handler(decoded) 0x00007FFF33FE3450 which is in rtscom.dll
[d] [15304][OUTLOOK.EXE] # of VEH: 3
```



```
[i] [5660][com.docker.service] is using VEH - Vectored Exception Handler
[i] RtlRemoveVectoredExceptionHandler [7fff5df92070]
[i] LdrpVectorHandlerList [7fff5e08f3e8]
[d] [5660][com.docker.service] VEH handler(decoded) 0x00007FFF3C3F5230 which is in clr.dll
[d] [5660][com.docker.service] # of VEH: 1
[i] [6608][C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2111.5-0\MsMpEng.exe] not analysed 5
[i] [9996][C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2111.5-0\NisSrv.exe] not analysed 5
[i] [7468][C:\Windows\System32\SecurityHealthService.exe] not analysed 5
[i] [15288][slack.exe] is using VEH - Vectored Exception Handler
[i] RtlRemoveVectoredExceptionHandler [7fff5df92070]
[i] LdrpVectorHandlerList [7fff5e08f3e8]
[d] [15288][slack.exe] VEH handler(decoded) 0x00007FF753B7EE20 which is in slack.exe
[d] [15288][slack.exe] # of VEH: 1
[i] [14732][C:\Windows\System32\SgrmBroker.exe] not analysed 5
[i] [6676][C:\Windows\System32\svchost.exe] not analysed 5
[i] [13084][msedgewebview2.exe] is using VEH - Vectored Exception Handler
[i] RtlRemoveVectoredExceptionHandler [7fff5df92070]
[i] LdrpVectorHandlerList [7fff5e08f3e8]
[d] [13084][msedgewebview2.exe] VEH handler(decoded) 0x00007FFEDE523880 which is in msedge.dll
[d] [13084][msedgewebview2.exe] # of VEH: 1
[i] [15580][msedgewebview2.exe] is using VEH - Vectored Exception Handler
[i] RtlRemoveVectoredExceptionHandler [7fff5df92070]
[i] LdrpVectorHandlerList [7fff5e08f3e8]
[d] [15580][msedgewebview2.exe] VEH handler(decoded) 0x00007FFEDE523880 which is in msedge.dll
[d] [15580][msedgewebview2.exe] # of VEH: 1
[i] [15508][msedge.exe] is using VEH - Vectored Exception Handler
[i] RtlRemoveVectoredExceptionHandler [7fff5df92070]
[i] LdrpVectorHandlerList [7fff5e08f3e8]
[d] [15508][msedge.exe] VEH handler(decoded) 0x00007FFEDE523880 which is in msedge.dll
[d] [15508][msedge.exe] # of VEH: 1
[i] [16612][C:\Windows\System32\svchost.exe] not analysed 5
[i] [15304][OUTLOOK.EXE] is using VEH - Vectored Exception Handler
[i] RtlRemoveVectoredExceptionHandler [7fff5df92070]
[i] LdrpVectorHandlerList [7fff5e08f3e8]
[d] [15304][OUTLOOK.EXE] VEH handler(decoded) 0x00007FFF3C3F5230 which is in clr.dll
[d] [15304][OUTLOOK.EXE] VEH handler(decoded) 0x00007FFF025BA7A0 which is in InkObj.dll
[d] [15304][OUTLOOK.EXE] VEH handler(decoded) 0x00007FFF33FE3450 which is in rtscom.dll
[d] [15304][OUTLOOK.EXE] # of VEH: 3
```

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI/tree/master/d-vehimplant
https://research.nccgroup.com/2022/01/03/detecting-anomalous-vectored-exception-handlers-on-windows/

Use of Debug Registers (Hardware Breakpoints)

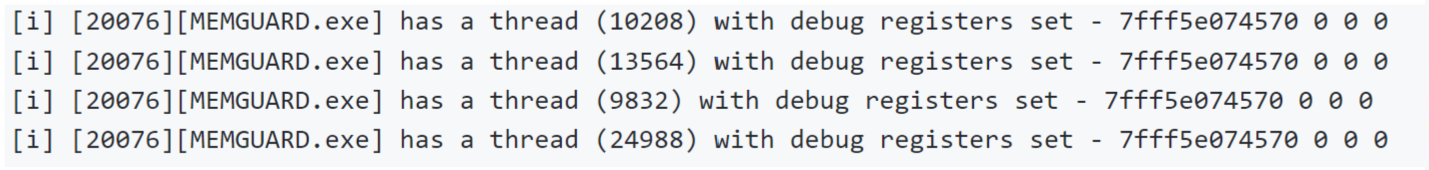
Thesis:

Using exception handlers requires either software or hardware breakpoints. Hardware breakpoints can be enumerated on a per process via the presence of debug registers. We don’t expect any to be set on a typical system.

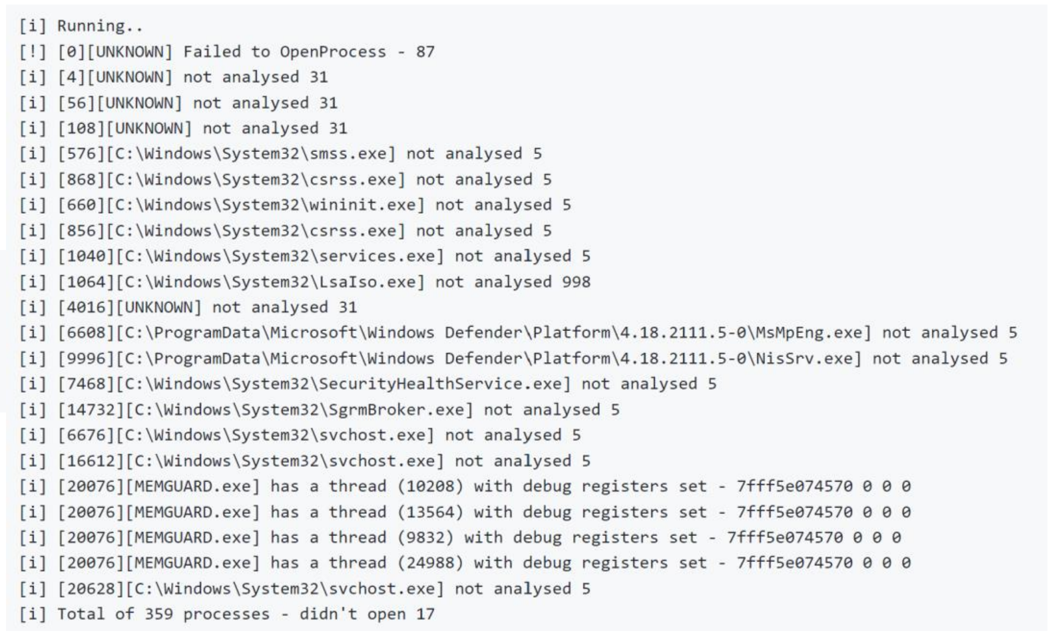
Solution:

- Open processes
- Get thread context
- Inspect the Dr0, Dr1, Dr2 and Dr3 registers

Use of Debug Registers - result



```
[i] [20076][MEMGUARD.exe] has a thread (10208) with debug registers set - 7fff5e074570 0 0 0
[i] [20076][MEMGUARD.exe] has a thread (13564) with debug registers set - 7fff5e074570 0 0 0
[i] [20076][MEMGUARD.exe] has a thread (9832) with debug registers set - 7fff5e074570 0 0 0
[i] [20076][MEMGUARD.exe] has a thread (24988) with debug registers set - 7fff5e074570 0 0 0
```



```
[i] Running..
[!] [0][UNKNOWN] Failed to OpenProcess - 87
[i] [4][UNKNOWN] not analysed 31
[i] [56][UNKNOWN] not analysed 31
[i] [108][UNKNOWN] not analysed 31
[i] [576][C:\Windows\System32\smss.exe] not analysed 5
[i] [868][C:\Windows\System32\csrss.exe] not analysed 5
[i] [660][C:\Windows\System32\wininit.exe] not analysed 5
[i] [856][C:\Windows\System32\csrss.exe] not analysed 5
[i] [1040][C:\Windows\System32\services.exe] not analysed 5
[i] [1064][C:\Windows\System32\LsaIso.exe] not analysed 998
[i] [4016][UNKNOWN] not analysed 31
[i] [6608][C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2111.5-0\MsMpEng.exe] not analysed 5
[i] [9996][C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2111.5-0\NisSrv.exe] not analysed 5
[i] [7468][C:\Windows\System32\SecurityHealthService.exe] not analysed 5
[i] [14732][C:\Windows\System32\SgrmBroker.exe] not analysed 5
[i] [6676][C:\Windows\System32\svchost.exe] not analysed 5
[i] [16612][C:\Windows\System32\svchost.exe] not analysed 5
[i] [20076][MEMGUARD.exe] has a thread (10208) with debug registers set - 7fff5e074570 0 0 0
[i] [20076][MEMGUARD.exe] has a thread (13564) with debug registers set - 7fff5e074570 0 0 0
[i] [20076][MEMGUARD.exe] has a thread (9832) with debug registers set - 7fff5e074570 0 0 0
[i] [20076][MEMGUARD.exe] has a thread (24988) with debug registers set - 7fff5e074570 0 0 0
[i] [20628][C:\Windows\System32\svchost.exe] not analysed 5
[i] Total of 359 processes - didn't open 17
```

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI/tree/master/d-dr-registers

DLL loading

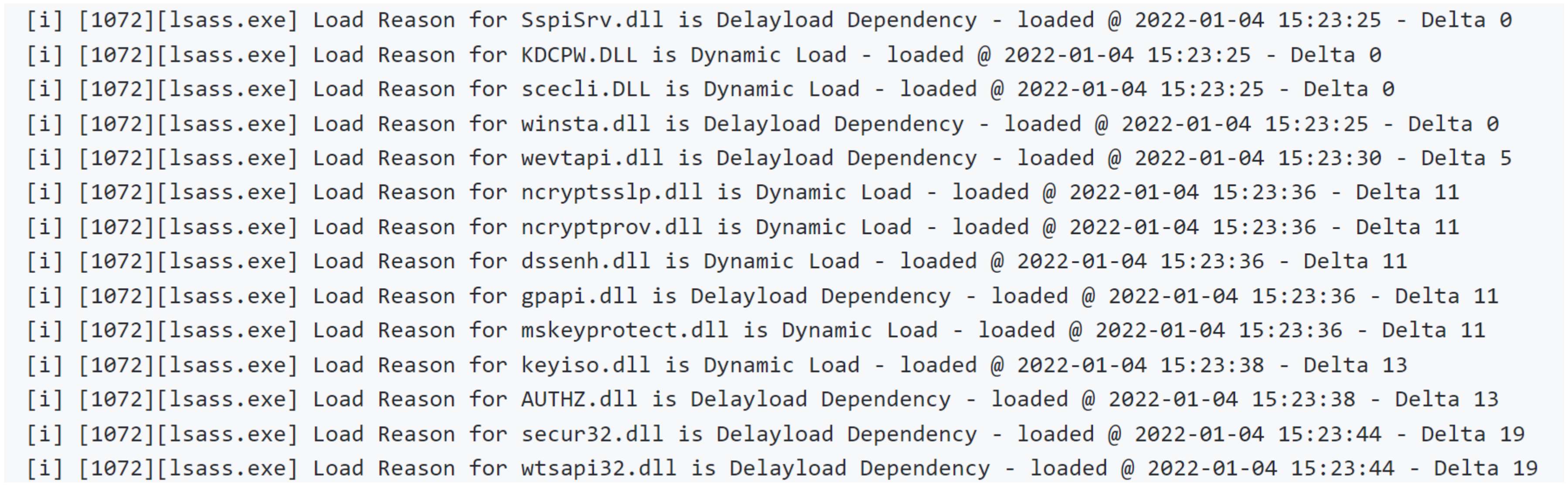
Thesis:

Implants will need libraries above and beyond what the host process would typically need. We can enumerate the libraries loaded, the date/time they occurred and the reason as a source of signal

Solution:

- Open processes
- Enumerate the PEB
- Walk the LDR_DATA_TABLE

DLL loading - result



```
[i] [1072][lsass.exe] Load Reason for SspiSrv.dll is Delayload Dependency - loaded @ 2022-01-04 15:23:25 - Delta 0
[i] [1072][lsass.exe] Load Reason for KDCPW.DLL is Dynamic Load - loaded @ 2022-01-04 15:23:25 - Delta 0
[i] [1072][lsass.exe] Load Reason for scecli.DLL is Dynamic Load - loaded @ 2022-01-04 15:23:25 - Delta 0
[i] [1072][lsass.exe] Load Reason for winsta.dll is Delayload Dependency - loaded @ 2022-01-04 15:23:25 - Delta 0
[i] [1072][lsass.exe] Load Reason for wevtapi.dll is Delayload Dependency - loaded @ 2022-01-04 15:23:30 - Delta 5
[i] [1072][lsass.exe] Load Reason for ncryptsslp.dll is Dynamic Load - loaded @ 2022-01-04 15:23:36 - Delta 11
[i] [1072][lsass.exe] Load Reason for ncryptprov.dll is Dynamic Load - loaded @ 2022-01-04 15:23:36 - Delta 11
[i] [1072][lsass.exe] Load Reason for dssenh.dll is Dynamic Load - loaded @ 2022-01-04 15:23:36 - Delta 11
[i] [1072][lsass.exe] Load Reason for gpapi.dll is Delayload Dependency - loaded @ 2022-01-04 15:23:36 - Delta 11
[i] [1072][lsass.exe] Load Reason for mskeyprotect.dll is Dynamic Load - loaded @ 2022-01-04 15:23:36 - Delta 11
[i] [1072][lsass.exe] Load Reason for keyiso.dll is Dynamic Load - loaded @ 2022-01-04 15:23:38 - Delta 13
[i] [1072][lsass.exe] Load Reason for AUTHZ.dll is Delayload Dependency - loaded @ 2022-01-04 15:23:38 - Delta 13
[i] [1072][lsass.exe] Load Reason for secur32.dll is Delayload Dependency - loaded @ 2022-01-04 15:23:44 - Delta 19
[i] [1072][lsass.exe] Load Reason for wtsapi32.dll is Delayload Dependency - loaded @ 2022-01-04 15:23:44 - Delta 19
```

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI/tree/master/d-peb-dll-loadreason

Impersonating threads

Thesis:

An implant will need to impersonate other users on the system to conduct some operations. Such impersonations would be an anomaly when observed in the context of certain host processes.

Solution:

- Open processes and each sub thread
- Enumerate the TEB
- Enumerate IsImpersonating in the TEB

Impersonating threads - result

```
[i] [2544][svchost.exe] is impersonating
[i] [4016][UNKNOWN] not analysed 31
[i] [5500][svchost.exe] is impersonating
[i] [5500][svchost.exe] is impersonating
```

```
[i] Running..
[!] [0][UNKNOWN] Failed to OpenProcess - 87
[i] [4][UNKNOWN] not analysed 31
[i] [56][UNKNOWN] not analysed 31
[i] [108][UNKNOWN] not analysed 31
[i] [576][C:\Windows\System32\smss.exe] not analysed 5
[i] [868][C:\Windows\System32\csrss.exe] not analysed 5
[i] [660][C:\Windows\System32\wininit.exe] not analysed 5
[i] [856][C:\Windows\System32\csrss.exe] not analysed 5
[i] [1040][C:\Windows\System32\services.exe] not analysed 5
[i] [1064][C:\Windows\System32\LsaIso.exe] not analysed 998
[i] [2544][svchost.exe] is impersonating
[i] [4016][UNKNOWN] not analysed 31
[i] [5500][svchost.exe] is impersonating
[i] [5500][svchost.exe] is impersonating
[i] [6608][C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2111.5-0\MsMpEng.exe] not analysed 5
[i] [9996][C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2111.5-0\NisSrv.exe] not analysed 5
[i] [7468][C:\Windows\System32\SecurityHealthService.exe] not analysed 5
[i] [14732][C:\Windows\System32\SgrmBroker.exe] not analysed 5
[i] [6676][C:\Windows\System32\svchost.exe] not analysed 5
[i] [16612][C:\Windows\System32\svchost.exe] not analysed 5
[i] [20628][C:\Windows\System32\svchost.exe] not analysed 5
[!] [18480][UNKNOWN] Failed to OpenProcess - 87
[!] [25680][UNKNOWN] Failed to OpenProcess - 87
[!] [25304][UNKNOWN] Failed to OpenProcess - 87
[i] Total of 360 processes - didn't open 17
```

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI/tree/master/d-teb

Non Module Call Stack

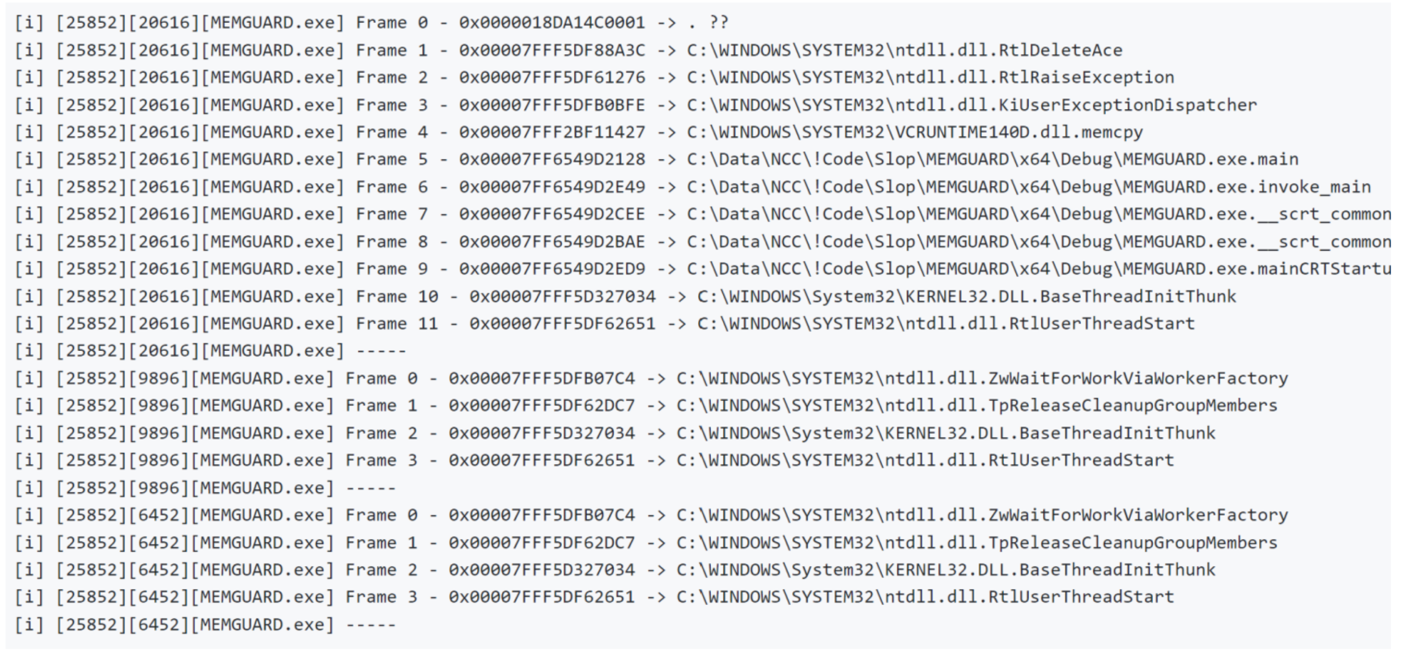
Thesis:

JIT code exists in very few processes. Implants will want to execute their own code and won’t typically use ROP. We can enumerate the call stacks of all threads and identify those frames which don’t point to a module.

Solution:

- Open processes and each sub thread
- Enumerate the TEB
- Enumerate IsImpersonating in the TEB

Non Module Call Stack



```
[i] [25852][20616][MEMGUARD.exe] Frame 0 - 0x0000018DA14C0001 -> . ??
[i] [25852][20616][MEMGUARD.exe] Frame 1 - 0x00007FFF5DF88A3C -> C:\WINDOWS\SYSTEM32\ntdll.dll.RtlDeleteAce
[i] [25852][20616][MEMGUARD.exe] Frame 2 - 0x00007FFF5DF61276 -> C:\WINDOWS\SYSTEM32\ntdll.dll.RtlRaiseException
[i] [25852][20616][MEMGUARD.exe] Frame 3 - 0x00007FFF5DFB0BFE -> C:\WINDOWS\SYSTEM32\ntdll.dll.KiUserExceptionDispatcher
[i] [25852][20616][MEMGUARD.exe] Frame 4 - 0x00007FFF2BF11427 -> C:\WINDOWS\SYSTEM32\VCRUNTIME140D.dll.memcpy
[i] [25852][20616][MEMGUARD.exe] Frame 5 - 0x00007FF6549D2128 -> C:\Data\NCC\!Code\Slop\MEMGUARD\x64\Debug\MEMGUARD.exe.main
[i] [25852][20616][MEMGUARD.exe] Frame 6 - 0x00007FF6549D2E49 -> C:\Data\NCC\!Code\Slop\MEMGUARD\x64\Debug\MEMGUARD.exe.invoke_main
[i] [25852][20616][MEMGUARD.exe] Frame 7 - 0x00007FF6549D2CEE -> C:\Data\NCC\!Code\Slop\MEMGUARD\x64\Debug\MEMGUARD.exe.__scrt_common
[i] [25852][20616][MEMGUARD.exe] Frame 8 - 0x00007FF6549D2BAE -> C:\Data\NCC\!Code\Slop\MEMGUARD\x64\Debug\MEMGUARD.exe.__scrt_common
[i] [25852][20616][MEMGUARD.exe] Frame 9 - 0x00007FF6549D2ED9 -> C:\Data\NCC\!Code\Slop\MEMGUARD\x64\Debug\MEMGUARD.exe.mainCRTStartu
[i] [25852][20616][MEMGUARD.exe] Frame 10 - 0x00007FFF5D327034 -> C:\WINDOWS\System32\KERNEL32.DLL.BaseThreadInitThunk
[i] [25852][20616][MEMGUARD.exe] Frame 11 - 0x00007FFF5DF62651 -> C:\WINDOWS\SYSTEM32\ntdll.dll.RtlUserThreadStart
[i] [25852][20616][MEMGUARD.exe] -----
[i] [25852][9896][MEMGUARD.exe] Frame 0 - 0x00007FFF5DFB07C4 -> C:\WINDOWS\SYSTEM32\ntdll.dll.ZwWaitForWorkViaWorkerFactory
[i] [25852][9896][MEMGUARD.exe] Frame 1 - 0x00007FFF5DF62DC7 -> C:\WINDOWS\SYSTEM32\ntdll.dll.TpReleaseCleanupGroupMembers
[i] [25852][9896][MEMGUARD.exe] Frame 2 - 0x00007FFF5D327034 -> C:\WINDOWS\System32\KERNEL32.DLL.BaseThreadInitThunk
[i] [25852][9896][MEMGUARD.exe] Frame 3 - 0x00007FFF5DF62651 -> C:\WINDOWS\SYSTEM32\ntdll.dll.RtlUserThreadStart
[i] [25852][9896][MEMGUARD.exe] -----
[i] [25852][6452][MEMGUARD.exe] Frame 0 - 0x00007FFF5DFB07C4 -> C:\WINDOWS\SYSTEM32\ntdll.dll.ZwWaitForWorkViaWorkerFactory
[i] [25852][6452][MEMGUARD.exe] Frame 1 - 0x00007FFF5DF62DC7 -> C:\WINDOWS\SYSTEM32\ntdll.dll.TpReleaseCleanupGroupMembers
[i] [25852][6452][MEMGUARD.exe] Frame 2 - 0x00007FFF5D327034 -> C:\WINDOWS\System32\KERNEL32.DLL.BaseThreadInitThunk
[i] [25852][6452][MEMGUARD.exe] Frame 3 - 0x00007FFF5DF62651 -> C:\WINDOWS\SYSTEM32\ntdll.dll.RtlUserThreadStart
[i] [25852][6452][MEMGUARD.exe] -----
```

In an unscientific sample set of one host searching for the output `. ??` in the result we only saw the following - one of which was the test case:

```
[i] [5516][7280][cb.exe] Frame 0 - 0x00007FF74E50EF77 -> C:\WINDOWS\CarbonBlack\cb.exe. ??
[i] [5516][7280][cb.exe] Frame 1 - 0x00007FF74E50FC51 -> C:\WINDOWS\CarbonBlack\cb.exe. ??
[i] [5516][7280][cb.exe] Frame 2 - 0x00007FF74E4A1AA2 -> C:\WINDOWS\CarbonBlack\cb.exe. ??
[i] [24212][21104][MEMGUARD.exe] Frame 0 - 0x000002CF87690000 -> . ??
```

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI/tree/master/d-teb

CreateRemoteThread

Thesis:

We can enumerates the address and which module the starting address of each thread points to. This will help detect when threat actors allocate memory for their payload and use that address as the start address to CreateThread or CreateRemoteThread etc.

Solution:

- Open processes and each sub thread
- Enumerate the TEB start address
- Enumerate which module the start address points to

CreateRemoteThread- result

```
[i] [22516][9764][MEMGUARD.exe] Start Address of Thread 7ff734f41023 in C:\Data\NCC\!Code\Slop\MEMGUARD\x64\Debug\MEMGUARD.exe->ILT+3
[i] [22516][18584][MEMGUARD.exe] Start Address of Thread 7ffb72dc2ad0 in C:\WINDOWS\SYSTEM32\ntdll.dll->TpReleaseCleanupGroupMembers
[i] [22516][4240][MEMGUARD.exe] Start Address of Thread 7ffb72dc2ad0 in C:\WINDOWS\SYSTEM32\ntdll.dll->TpReleaseCleanupGroupMembers
[i] [22516][9668][MEMGUARD.exe] Start Address of Thread 1e8c7330000 in UnknownModule->UnknownFunction    <---- result of CreateRemote
[i] [22516][7772][MEMGUARD.exe] Start Address of Thread 7ffb72dc2ad0 in C:\WINDOWS\SYSTEM32\ntdll.dll->TpReleaseCleanupGroupMembers
...
```

```
[i] Total of 355 processes - didn't open 20 - total of 2983 threads - 2 start in unknown modules
```

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI/tree/master/d-thread-start

Best of the Rest

Things I didn’t get to cover where we see there is opportunity

Variances in the # executable pages on a per process basis

Hooking condrv and catching tool output via `stdout` and Yara rules

Summary & Conclusions

We can extract a number of signals at low cost

A number of these signals can be stand alone and high signal

Others can be combined when we have fleet level visibility to detect anomalies in the population

ROP detection strategies will likely be more involved, but not impossible including CFG variances, shadow stack comparison, stack analysis and similar

Code

various techniques discussed

https://github.com/nccgroup/DetectWindowsCopyOnWriteForAPI

variance in executable memory pages

https://github.com/nccgroup/WindowsMemPageDelta

stdout condrv output observer via hooking

https://github.com/nccgroup/mimikatz-detector-condrv thank you!

twitter: ollieatnccgroup