elastic security labs

Kernel ETW
is the best ETW

John Uhlmann
Security Research Engineer

July 2024 BSides Brisbane

whoami

Security Research Engineer at Elastic

- Elastic Endpoint ("EDR") developer
- Elastic Security Labs blogger
- https://www.elastic.co/security-labs/author/john-uhlmann
- https://www.elastic.co/blog/edr-bypass-taxonomy

elastic | The Search AI Company

Elastic Endpoint

awsn

elastic

- Former Technical Director at the Australian Cyber Security Centre.
- Australian Women in Security Network supporter – find me in the Perth chapter on slack if you want to chat.

Quick Glossary

- Kernel Mode – kernel (ntoskrnl.exe) and drivers (.sys)
  - Kernel Patch Protection (KPP aka PatchGuard)
- User Mode – processes (.exe) and libraries (.dll)
  - client programs (user) and system services (SYSTEM)
- Event Log – persistent key-value logs
- Event Tracing for Windows (ETW) – key-value events
  - providers, keywords
- Microsoft-Windows-Threat-Intelligence (ETW-TI)
  - Antimalware-PPL (Protected Process Light)

elastic

Why Kernel ETW?

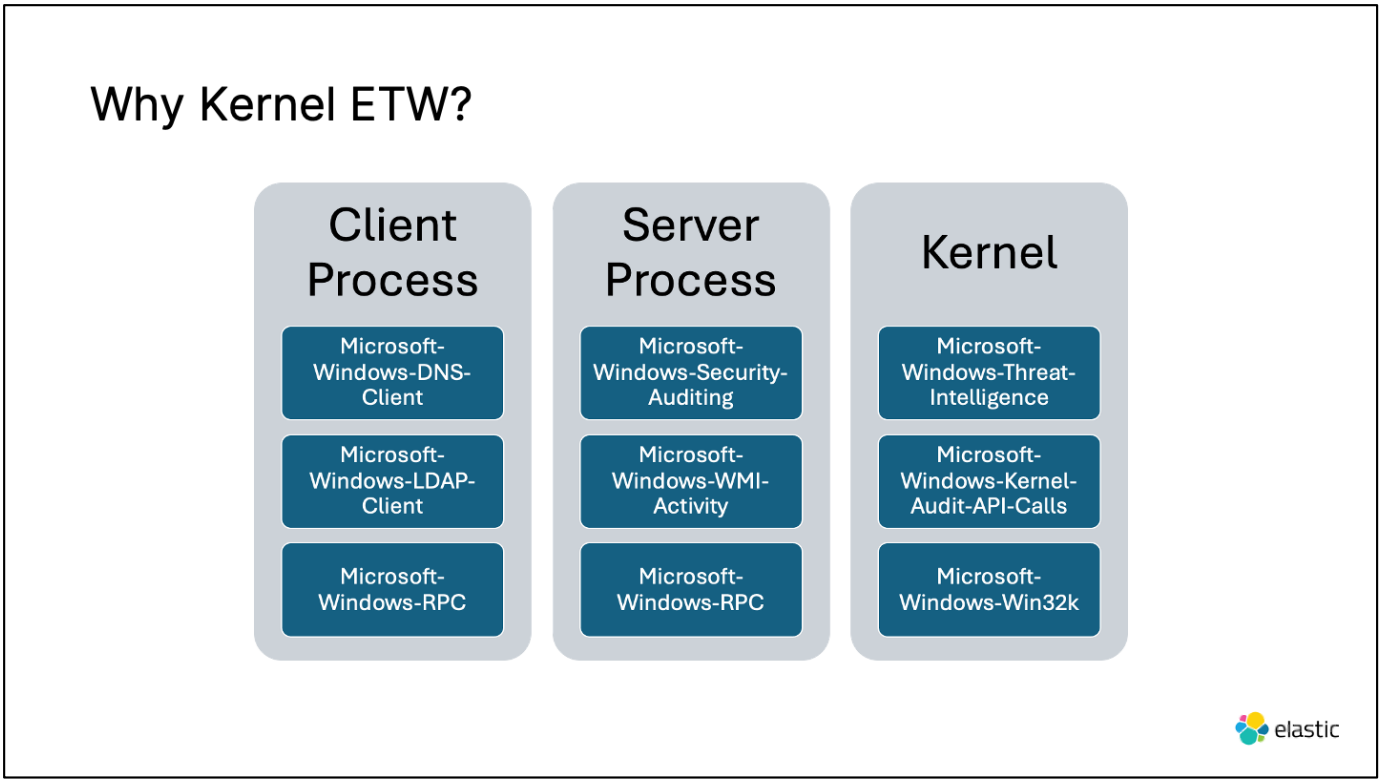
- Why audit logs?
  - Secure-by-Design
- Why Kernel?
  - Security should never be client-side
- Why ETW?
  - Very limited Kernel extension points
  - Kernel Patch Protection (aka PatchGuard)

elastic

- A critical feature of secure-by-design software is generating audit logs when privileged operations are performed.
- These native audit logs can include details of the internal software state which are impractical for third party security vendors to bolt on after the fact.

- Why Kernel? Higher privilege is more trustworthy.
- It's a security boundary for medium integrity, and defence-in-depth otherwise.

- Why ETW? Kernel callbacks and mini-filters are actually strongly preferred.
- Being inline they are not vulnerable to ToCToU and provide prevention opportunities.
- We have good coverage for network, filesystem and registry kernel activity – but they are very rare otherwise.
- On x86, security vendors were able to hook additional APIs such as ZwProtectVirtualMemory.
- On x64, Kernel Patch Protection stops this.
- Microsoft has determined that PatchGuard benefits outweigh the loss of prevention opportunities for security vendors.
- When you are bypassing EDR it’s often by Microsoft design.
- One could even argue that EDR only exists because the native Windows event logging is insufficient.

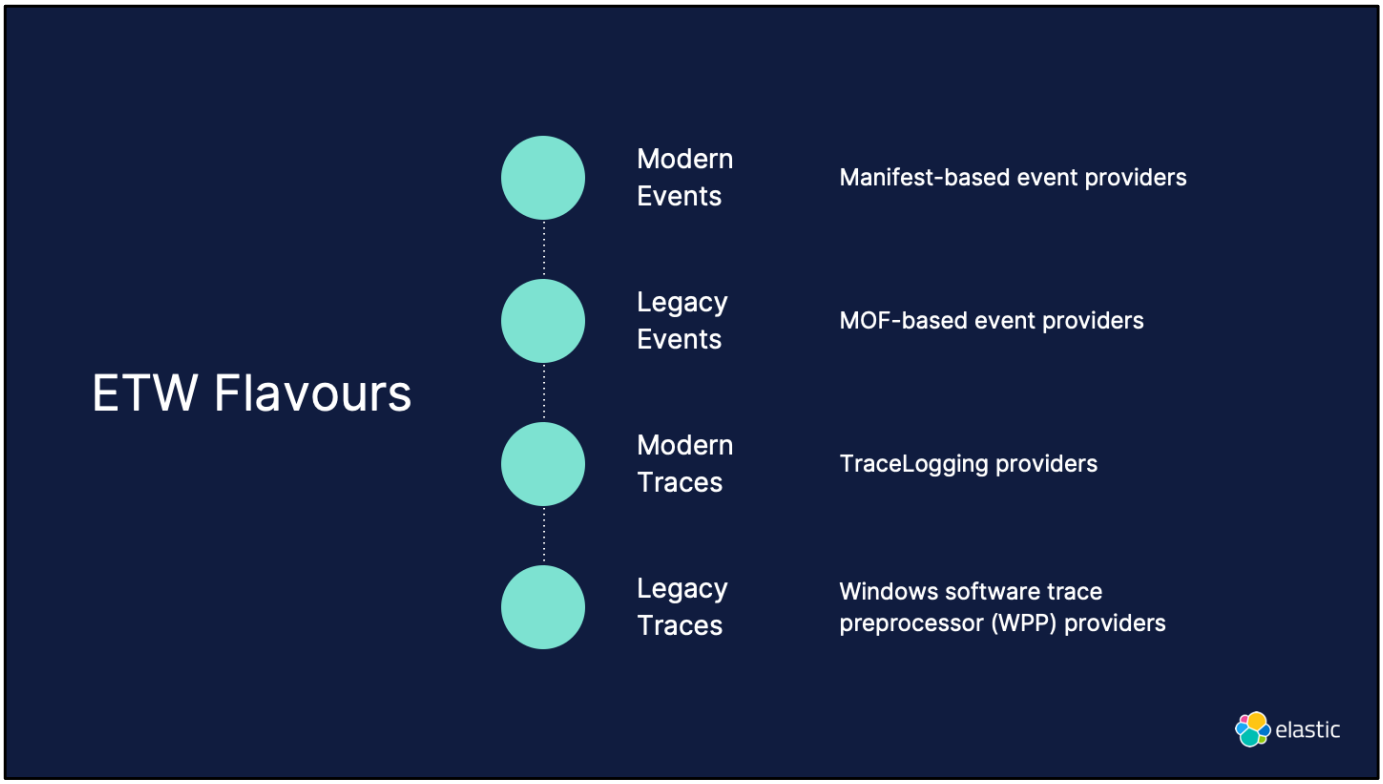


- Why do we need to understand our data sources?
- I highly recommend SpecterOps' Jared Atkinson's multi-part On Detection series.
- Detection Engineers that don't fully understand their data sources can't write robust detections.
- Shifting from detecting tools to detecting techniques.

- Firstly, can they be trivially bypassed?
- Secondly, is the data itself trustworthy?

- Surprisingly, event 4688 for process creation is not a kernel event.
- The kernel dispatches the data to lsass service to log the event. So, I could tamper with it from within the server process.
- Whereas the ProcessStart event in the Microsoft-Windows-Kernel-Process is logged directly by the kernel.

- Microsoft-Windows-WMI-Activity is another interesting case study.
- The WMI Management Service Host seems like a fairly trustworthy log source, right?
- But it blindly logs whatever ClientProcessId value the client provides it!



- The “event” versus “trace” distinction is mostly semantic.
- Event providers are typically registered with the operating system ahead of time and you can inspect the available telemetry metadata.
- These are typically used by system administrators for troubleshooting purposes and are often semi-documented.
- But when something goes really, really wrong there are (hidden) trace providers.
- These are typically used only by the original software authors for advanced troubleshooting and are undocumented.

My Understanding Kernel ETW Approach

1. Modern Events
2. Legacy Events
3. Modern Trace Events
4. Legacy Trace Events

a) Documentation?
b) Native tooling?
c) 3rd party tooling?
d) New tooling || Reversing

elastic

- For each ETW flavour investigate documentation, native tooling, 3rd party tooling and finally new tooling if required.

Step 1 – Modern Kernel ETW Docs?

elastic

- Given this reliance, the public documentation of these kernel telemetry sources is unfortunately somewhat sparse.

- Modern ETW providers are not documented.
- Instead, you can enumerate registered providers at runtime.
- With the caveat that if a feature is not enabled then it's provider will typically not be registered.
- For example, the DNS Server provider is not registered until the DNS Server role is added.

Step 1 – Native Modern Event tooling?

```
C:\>logman query providers | findstr Kernel
Circular Kernel Session Provider         {54DEA73A-ED1F-42A4-AF71-3E63D056F174}
File Kernel Trace; Operation Set 1       {075D8303-6C21-4BDE-9C98-ECC6320F9281}
File Kernel Trace; Operation Set 2       {058DD951-76D4-414D-A5D6-A56035367A46}
File Kernel Trace; Optional Data         {7DA1385C-F8F5-414D-B9D0-02FCA090F1EC}
File Kernel Trace; Volume To Log         {127D46AF-4AD3-489F-9165-F08BA64D5467}
Microsoft-Windows-DirectShow-KernelSupport {3CC2D4AF-DA5E-4ED4-BCBE-3CF995948483}
Microsoft-Windows-DriverFrameworks-KernelMode-Performance {486A5C7C-11CC-46C5-9DE
Microsoft-Windows-Kernel-Acpi            {C514638F-7723-485B-BCFC-96565D735DAA}
Microsoft-Windows-Kernel-AppCompat       {16A1ADC1-989F-4CD9-94B3-D8296AB10130}
Microsoft-Windows-Kernel-Audit-API-Calls {E02A841C-75A3-4FA7-AFC8-AE09CF9B7F23}
Microsoft-Windows-Kernel-Boot            {15CA44FF-4D7A-4BAA-BBA5-09988955E531E}
Microsoft-Windows-Kernel-BootDiagnostics {96AC7637-5950-4A30-B8F7-E07E8E5734C1}
Microsoft-Windows-Kernel-Cache           {A2D34BF1-70AB-5B21-C819-5A0DD42748FD}
Microsoft-Windows-Kernel-CPU-Starvation  {7F54CA8A-6C72-5CBC-B96F-D0EF905B8BCE}
Microsoft-Windows-Kernel-Disk            {C7BDE69A-E1E0-4177-B6EF-283AD1525271}
Microsoft-Windows-Kernel-Dump            {17D2A329-4539-5F4D-3435-F510634CE3B9}
Microsoft-Windows-Kernel-EventTracing    {B675EC37-BDB6-4648-BC92-F3FDC74D3CA2}
Microsoft-Windows-Kernel-File            {EDD08927-9CC4-4E65-B970-C2560FB5C289}
Microsoft-Windows-Kernel-General         {A68CA8B7-004F-D7B6-A698-07E2DE0F1F5D}
Microsoft-Windows-Kernel-Interrupt-Steering {951B41EA-CB30-44DC-A671-E2C9958809BB}
Microsoft-Windows-Kernel-IO              {ABF1F586-2E50-4BA8-928D-49044E6F0DB7}
Microsoft-Windows-Kernel-IoTrace         {A103CABD-8242-4A93-8DF5-1CDF3B3F26A6}
Microsoft-Windows-Kernel-Licensing-StartServiceTrigger {F5528ADA-BE5F-4F14-8AEF-A
Microsoft-Windows-Kernel-LicensingSqm    {A0AF438F-4031-41CB-A675-A265050EE9A7}
Microsoft-Windows-Kernel-LiveDump        {BEF2AA8E-81CD-11E2-A7BB-5EAC6188709B}
Microsoft-Windows-Kernel-Memory          {D1D93EF7-E1F2-4F45-9943-03D245FE6C00}
Microsoft-Windows-Kernel-Network         {7DD42A49-5329-4832-8DFD-43D979153A88}
Microsoft-Windows-Kernel-Pep             {5412704E-B2E1-4624-8FFD-55777B8F7373}
Microsoft-Windows-Kernel-PnP             {9C205A39-1250-487D-ABD7-E831C6290539}
Microsoft-Windows-Kernel-PnP-Rundown     {B3A0C2C8-83BB-4DDF-9F8D-4B22D3C38AD7}
Microsoft-Windows-Kernel-Power           {331C3B3A-2005-44C2-AC5E-77220C37D6B4}
Microsoft-Windows-Kernel-PowerTrigger    {AA1F73E8-15FD-45D2-ABFD-E7F64F78EB11}
Microsoft-Windows-Kernel-Prefetch        {5322D61A-9EFA-4BC3-A3F9-14BE95C144F8}
Microsoft-Windows-Kernel-Prm             {B931ED29-66F4-576E-0579-0B8818A5DC6B}
Microsoft-Windows-Kernel-Process         {22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716}
Microsoft-Windows-Kernel-Processor-Power {0F67E49F-FE51-4E9F-B490-6F2948CC6027}
Microsoft-Windows-Kernel-Registry        {70EB4F03-C1DE-4F73-A051-33D13D5413BD}
Microsoft-Windows-Kernel-ShimEngine      {0BF2FB94-7B60-4B4D-9766-E82F658DF540}
Microsoft-Windows-Kernel-StoreMgr        {A6AD76E3-867A-4635-91B3-490ABA637407}
Microsoft-Windows-Kernel-Tm              {4CEC9C95-A65F-4591-B5C4-30100E51D870}
Microsoft-Windows-Kernel-Tm-Trigger      {CE20D1C3-A247-4C41-BCB8-3C7F52C8B805}
Microsoft-Windows-Kernel-WDI             {2FF3E6B7-CB90-4700-9621-443F389734ED}
Microsoft-Windows-Kernel-WHEA            {7B563579-53C8-44E7-8236-0F87B9FE6594}
Microsoft-Windows-Kernel-WSService-StartServiceTrigger {3635D4B6-77E3-4375-8124-D
Microsoft-Windows-Kernel-XDV             {F029AC39-38F0-4A40-B7DE-404D24404CB}
Microsoft-Windows-KernelStreaming        {648C4817-CE45-41FF-90DD-528F01C6F1E1}
Microsoft-Windows-RemoteDesktopServices-RemoteFX-VM-Kernel-Mode-Transport {7EB5F4
Microsoft-Windows-WerKernel              {87A623F0-8DB5-5C11-7C80-A2EBBCBE5189}
Windows Kernel Trace                     {9E814AAD-3204-11D2-9A82-006008A86939}
```

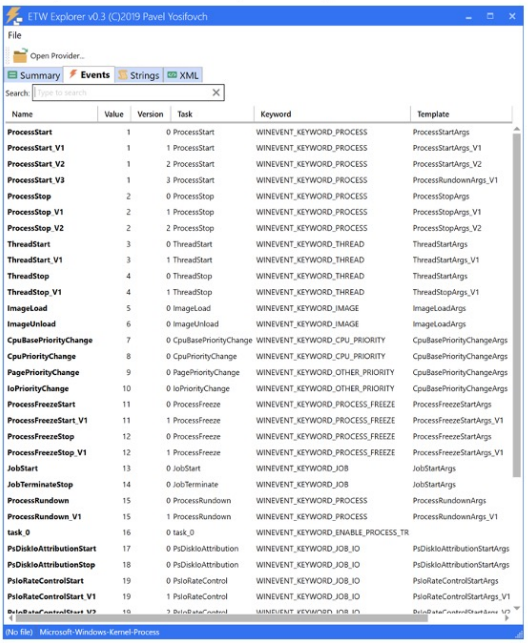
```
C:\>logman query providers Microsoft-Windows-Kernel-Process

Provider                                 GUID
-------------------------------------------------------------------------------
Microsoft-Windows-Kernel-Process         {22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716}

Value               Keyword              Description
-------------------------------------------------------------------------------
0x0000000000000010  WINEVENT_KEYWORD_PROCESS
0x0000000000000020  WINEVENT_KEYWORD_THREAD
0x0000000000000040  WINEVENT_KEYWORD_IMAGE
0x0000000000000080  WINEVENT_KEYWORD_CPU_PRIORITY
0x0000000000000100  WINEVENT_KEYWORD_OTHER_PRIORITY
0x0000000000000200  WINEVENT_KEYWORD_PROCESS_FREEZE
0x0000000000000400  WINEVENT_KEYWORD_JOB
0x0000000000000800  WINEVENT_KEYWORD_ENABLE_PROCESS_TRACING_CALLBACKS
0x0000000000001000  WINEVENT_KEYWORD_JOB_IO
0x0000000000002000  WINEVENT_KEYWORD_WORK_ON_BEHALF
0x0000000000004000  WINEVENT_KEYWORD_JOB_SILO
0x8000000000000000  Microsoft-Windows-Kernel-Process/Analytic

Value               Level                Description
-------------------------------------------------------------------------------
0x04                win:Informational    Information

PID                 Image
-------------------------------------------------------------------------------
0x00000000


The command completed successfully.

C:\>logman query providers Microsoft-Windows-Kernel-Audit-API-Calls

Provider                                 GUID
-------------------------------------------------------------------------------
Microsoft-Windows-Kernel-Audit-API-Calls {E02A841C-75A3-4FA7-AFC8-AE09CF9B7F23}

Value               Level                Description
-------------------------------------------------------------------------------
0x04                win:Informational    Information

PID                 Image
-------------------------------------------------------------------------------
0x00000000


The command completed successfully.
```

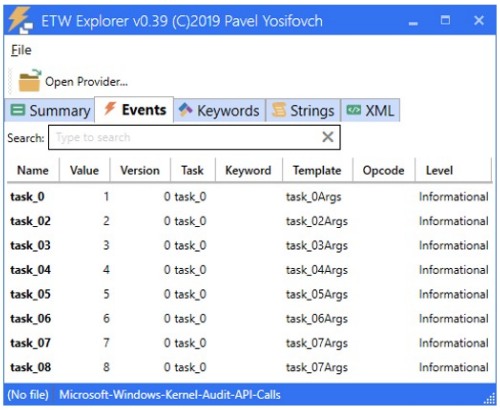
elastic

- Native tooling is quite limited.
- We can enumerate provider names and keywords only. Some providers don't have keywords.
- We could sample each provider with all keywords enabled.
- That's a lot of effort and a huge amount data to wade through.

- Note – equivalent PowerShell cmdlets also exist.

Step 1 – 3rd party Modern Events tooling?





elastic

- There are APIs that retrieve the XML manifest for a registered provider.
- This includes event and field names.

- Pavel Yosifovich, one of the Windows Internals authors, wrapped this in a GUI for us.

- So, we have the data – but is it easily searchable?

Step 1 – Write new Modern Event tooling!

jdu2600 / Windows10EtwEvents

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | Windows10EtwEvents / manifest / Microsoft-Windows-Threat-Intelligence.tsv

jdu2600 Windows 11 21H2 — 8a8a66f · 3 years ago — History

Preview | Code | Blame — 39 lines (39 loc) · 30.8 KB — Raw

Search this file

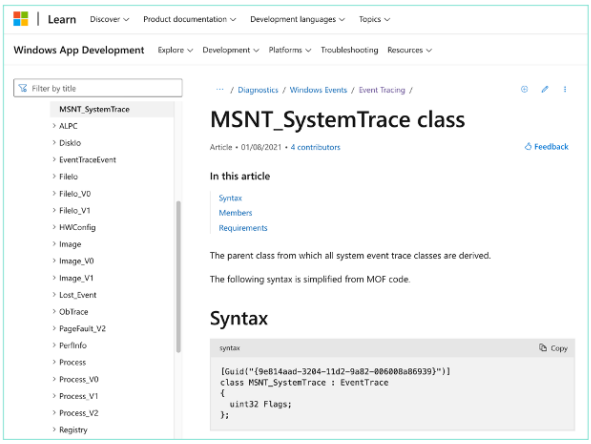
1	provider	event_id	version	event(fields)
2	Microsoft-Windows-Threat-Intelligence	1	1	KERNEL_THREATINT_TASK_ALLOCVM_V1(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProcessSign
3	Microsoft-Windows-Threat-Intelligence	2	1	KERNEL_THREATINT_TASK_PROTECTVM_V1(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProcessS
4	Microsoft-Windows-Threat-Intelligence	2	2	KERNEL_THREATINT_TASK_PROTECTVM_V2(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProcessS
5	Microsoft-Windows-Threat-Intelligence	3	1	KERNEL_THREATINT_TASK_MAPVIEW_V1(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProcessSign
6	Microsoft-Windows-Threat-Intelligence	4	1	KERNEL_THREATINT_TASK_QUEUEUSERAPC_V1(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProce
7	Microsoft-Windows-Threat-Intelligence	5	1	KERNEL_THREATINT_TASK_SETTHREADCONTEXT_V1(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 Calling
8	Microsoft-Windows-Threat-Intelligence	6	1	KERNEL_THREATINT_TASK_ALLOCVM6_V1(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProcessSig
9	Microsoft-Windows-Threat-Intelligence	7	1	KERNEL_THREATINT_TASK_PROTECTVM7_V1(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProcess
10	Microsoft-Windows-Threat-Intelligence	7	2	KERNEL_THREATINT_TASK_PROTECTVM7_V2(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProcess
11	Microsoft-Windows-Threat-Intelligence	8	1	KERNEL_THREATINT_TASK_MAPVIEW8_V1(UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, UInt8 CallingProcessSig
12	Microsoft-Windows-Threat-Intelligence	11	1	KERNEL_THREATINT_TASK_READVM_V1(UInt32 OperationStatus, UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, U
13	Microsoft-Windows-Threat-Intelligence	11	2	KERNEL_THREATINT_TASK_READVM_V2(UInt32 OperationStatus, UInt32 CallingProcessId, FILETIME CallingProcessCreateTime, UInt64 CallingProcessStartKey, U

elastic

- So, I wrote a tool to dump all the things and pushed the results to github for each Windows feature release.
- git blame shows when events are added or updated.

- Some folks didn't like my grep-friendly single line record approach so have since created similar repositories hosting the XML manifests or JSON representations instead.

Step 2 – Legacy Kernel ETW Docs?



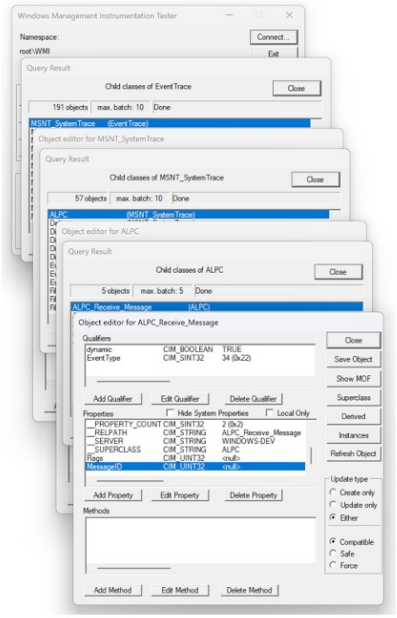
elastic

- Newer isn't always better.

- There is some overlap between legacy and modern kernel providers – eg both have a Process start event.
- But the modern provider doesn't log the command line.
- And there are some legacy events with no modern equivalent.

- We have docs this time.
- But I wanted my docs in a single location – my github repo.

Step 2 – Native Legacy Events tooling?

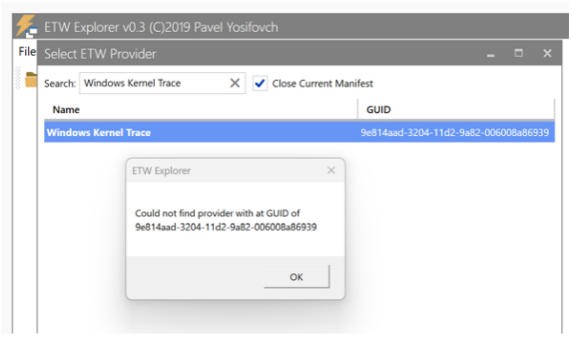


elastic

- It turns out that Legacy Provider metadata is stored in the WMI repository as instances of the EventTrace class.
- Providers are the root classes, groups are the children and events are the grandchildren.

- I needed a better GUI than wbemtest for ad-hoc browsing. Seven windows to view a single event!
- And I needed something to scrape all of the EventTrace classes registered in WMI.

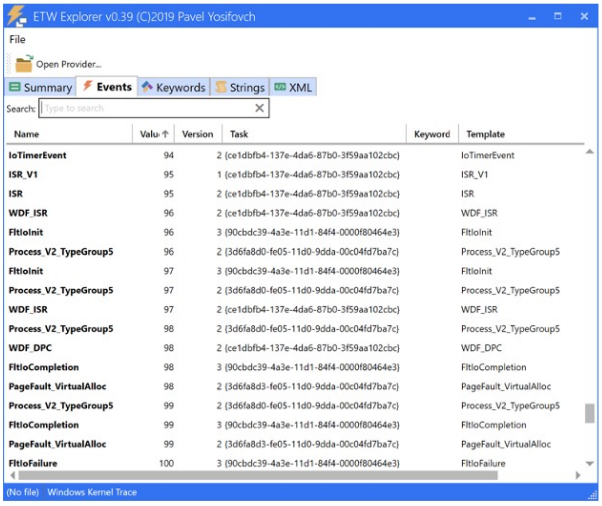
Step 2 – 3rd party Legacy Events tooling?

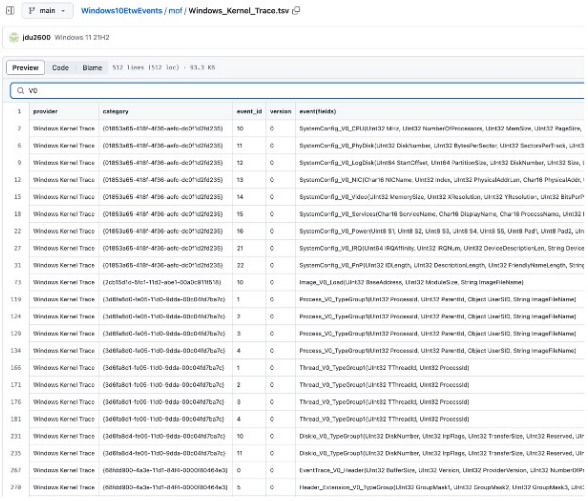


elastic

- Existing third-party tooling at the time didn’t support legacy providers.

Step 2 – Write new Legacy Event tooling!





elastic

- So, I updated it.
- And used the same code to dump all of events for my github repo.

- I discovered that there there were 340 registered legacy events, but only 116 were documented.

- Typically, each legacy event needs to be enabled via a specific flag, but these weren't documented either.
- I found a clue in the Object Trace documentation. It mentioned PERF_OB_HANDLE - a constant not defined in the headers in the latest SDK.
- Luckily, Geoff Chappell and the Windows 10 1511 WDK came to the rescue. I used this information to add support for PERFINFO_GROUPMASK kernel trace flags to Microsoft's krabsetw library.
- It also turned out that the Object Trace documentation was wrong. That non-public constant can only be used with undocumented APIs.
- Luckily, public Microsoft projects such as perfview often provide examples of how to use undocumented APIs.

Finding Security Relevant Kernel Events Methodology

- Kernel
- Security
- Task_
- ProcessStartKey
- API Parameter names – eg DesiredAccess
- Unique terms – eg Named Pipe

elastic

- We now have all of the ETW event metadata – how do we find the security-relevant kernel events?
- Interestingly, Microsoft often obfuscates the names of security relevant events so searching for events with a generic name prefix such as task_ yields some interesting results.
- Sometimes the keyword hints to the event's purpose. For example, task_014 in Microsoft-Windows-Kernel-General is enabled with the keyword KERNEL_GENERAL_SECURITY_ACCESSCHECK.
- Another useful query is to search for events with an explicit ProcessStartKey (LUID) field. Any event that includes this information for another process is often security relevant.
- If you had a specific API in mind, you might query for its name or its parameters.
- Thankfully, the parameters are almost always well named. We might guess that task_05 in Microsoft-Windows-Kernel-Audit-API-Calls is related to OpenProcess since it logs fields named TargetProcessId and DesiredAccess.
- Searching for Named Pipe leads us to Microsoft-Windows-SEC which (unfortunately) belongs to the built-in Microsoft Security drivers that Microsoft Defender for Endpoint utilises.

Step 3 – native Modern Trace tooling?

```
C:\>logman query providers -pid 3260

Provider                                  GUID
-------------------------------------------------------------------------------
Microsoft-Windows-AppModel-Runtime        {F1EF270A-0D32-4352-BA52-DBAB41E1D859}
Microsoft-Windows-AsynchronousCausality   {19AAC69A-28EB-4D4B-BD94-5F19055A1B5C}
Microsoft-Windows-COM-Perf                {B8D6861B-D20F-4EEC-BBAE-87E0DD80602B}
Microsoft-Windows-COM-RundownInstrumentation {2957313D-FCAA-5D4A-2F69-32CE5F0AC44E}
Microsoft-Windows-Crypto-BCrypt           {C7E089AC-BA2A-11E0-9AF7-68384824019B}
Microsoft-Windows-FeatureConfiguration    {C2F36562-A1E4-4BC3-A6F6-01A7ADB643E8}
Microsoft-Windows-Heap-Snapshot           {901D2AFA-4FF6-46D7-8D0E-53645E1A47F5}
Microsoft-Windows-Kernel-AppCompat        {16A1ADC1-9B7F-4CD9-94B3-D8296AB1B130}
Microsoft-Windows-Networking-Correlation {83ED54F0-4D48-4E45-B16E-726FFD1FA4AF}
Microsoft-Windows-Shell-Core              {30336ED4-E327-447C-9DE0-51B652C86108}
Microsoft-Windows-User-Diagnostic         {305FC87B-002A-5E26-D297-60223012CA9C}
Microsoft-Windows-WinRT-Error             {A86F8471-C31D-4FBC-A035-665D06047B03}
{05F95EFE-7F75-49C7-A994-60A55CC09571}    {05F95EFE-7F75-49C7-A994-60A55CC09571}
{0C4C1871-8B0C-4577-9471-0B45454ADF50}    {0C4C1871-8B0C-4577-9471-0B45454ADF50}
{1AFF6089-E863-4D36-BDFD-3581F07440BE}    {1AFF6089-E863-4D36-BDFD-3581F07440BE}
{3C74AFB9-8D82-44E3-B52C-365DBF48382A}    {3C74AFB9-8D82-44E3-B52C-365DBF48382A}
{57A6ED1A-79F7-5011-B242-4784E5620CF7}    {57A6ED1A-79F7-5011-B242-4784E5620CF7}
{6AF9E939-1D95-430A-AFA3-7526FADEE37D}    {6AF9E939-1D95-430A-AFA3-7526FADEE37D}
{703FCC13-B66F-5868-DDD9-E2DB7F381FFB}    {703FCC13-B66F-5868-DDD9-E2DB7F381FFB}
{A74EFE00-14BE-4EF9-9DA9-1484D5473302}    {A74EFE00-14BE-4EF9-9DA9-1484D5473302}
{B6FD710B-F783-4B1C-AB9C-C68099DCC0C7}    {B6FD710B-F783-4B1C-AB9C-C68099DCC0C7}
{B749553B-D950-5E03-6282-3145A61B1002}    {B749553B-D950-5E03-6282-3145A61B1002}
{BDA92AE8-9F11-4D49-BA1D-A4C2ABCA692E}    {BDA92AE8-9F11-4D49-BA1D-A4C2ABCA692E}
{C7E09E2A-C663-5399-AF79-2FCCD321D19A}    {C7E09E2A-C663-5399-AF79-2FCCD321D19A}
{CA967C75-04BF-40B5-9A16-98B5F9332A92}    {CA967C75-04BF-40B5-9A16-98B5F9332A92}
{DAF2C08D-5AC9-5066-FF0B-2D3D95275779}    {DAF2C08D-5AC9-5066-FF0B-2D3D95275779}
{F0558438-F56A-5987-47DA-040CA75AEF05}    {F0558438-F56A-5987-47DA-040CA75AEF05}
{F3A71A4B-6118-4257-8CCB-39A33BA059D4}    {F3A71A4B-6118-4257-8CCB-39A33BA059D4}

The command completed successfully.
```

```
C:\>logman query providers -pid 4


Error:
Element not found.

C:\>logman query providers -pid 0

Error:
The GUID passed was not recognized as valid by a WMI data provider.
```

elastic

- That's events. Now what about traces.
- No public docs by design. What about tooling?
- For user-mode providers we can enumerate GUIDs at most.
- For kernel providers we get nothing.

Step 3 – 3rd party Modern Trace tooling?

- Data Source Analysis and Dynamic Windows RE using WPP and TraceLogging | SpecterOps

ntoskrnl.exe TraceLogging Metadata - Windows 11 23H2 (Build 22621.2861)

ntoskrnl_TraceLoggingMetadata.json


```
1  {
2    "Providers": [
3      {
4        "ProviderGUID": "0f51c5a7-0e76-47a5-bede-7cf62c5822f6",
5        "ProviderName": "Microsoft.Windows.Kernel.HAL",
6        "ProviderGroupGUID": "4f50731a-89cf-47B2-b3e0-dce8c90476ba"
7      },
8      {
9        "ProviderGUID": "74093e1d-dbe3-4019-b97d-54edcb02cfed",
10       "ProviderName": "Microsoft.Windows.FileSystem.Cache",
11       "ProviderGroupGUID": "4f50731a-89cf-47B2-b3e0-dce8c90476ba"
12     },
13     {
14       "ProviderGUID": "107cf38f-b01b-4ffc-bf6b-d0ed96d8ecc6",
15       "ProviderName": "Microsoft.Windows.Kernel.Oplocks",
16       "ProviderGroupGUID": "4f50731a-89cf-47B2-b3e0-dce8c90476ba"
17     },
18     {
19       "ProviderGUID": "73a33ab2-1966-4999-8add-868c41415269",
20       "ProviderName": "Microsoft.Windows.IsolatedUserMode",
21       "ProviderGroupGUID": "4f50731a-89cf-47B2-b3e0-dce8c90476ba"
22     },
23     {
24       "ProviderGUID": "a51ee86b-8ea5-454c-9a7d-37b6655a535d",
25       "ProviderName": "Microsoft.Windows.Kernel.Dump",
```

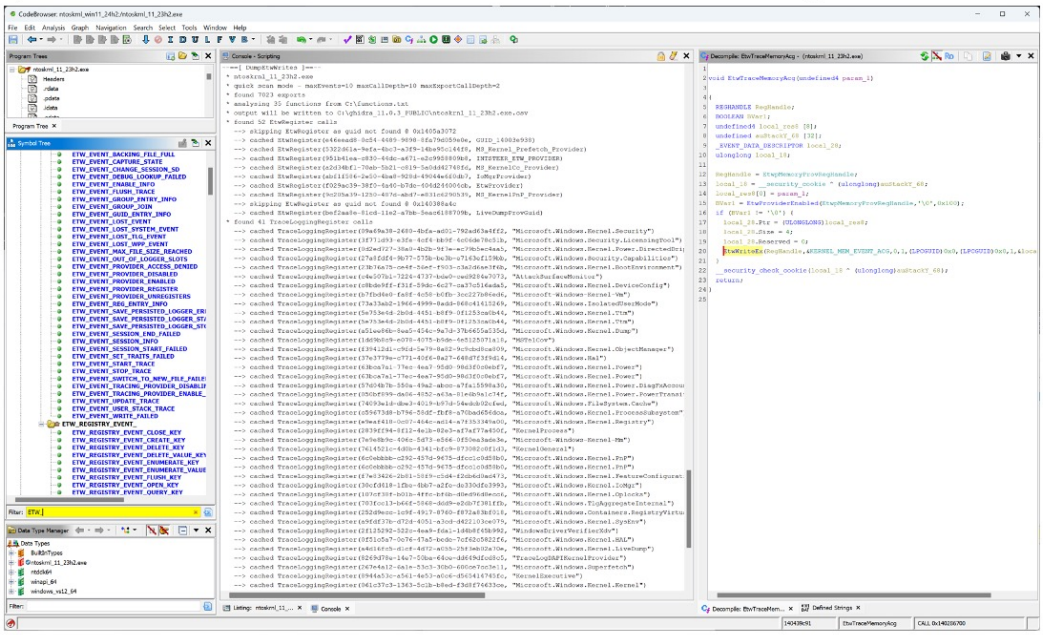
```
14819        "Keyword": "0x0000000000000002",
14820        "Extension": [
14821          128
14822        ],
14823        "EventName": "SecurityDescriptorChanging",
14824        "FieldInfo": [
14825          {
14826            "FieldName": "KeyPath",
14827            "InType": "COUNTEDSTRING"
14828          },
14829          {
14830            "FieldName": "OriginalSD",
14831            "InType": "BINARY"
14832          },
14833          {
14834            "FieldName": "InformationToChange",
14835            "InType": "UINT32"
14836          },
14837          {
14838            "FieldName": "ChangeSD",
14839            "InType": "BINARY"
14840          },
14841          {
14842            "FieldName": "ResultingSD",
14843            "InType": "BINARY"
14844          }
14845        ]
14846      },
```


elastic

- TraceLogging metadata is stored as an opaque blob within the logging binary.
- Thankfully, this format has been reversed by Matt Graeber at SpecterOps.
- We can use Matt's script to dump all TraceLogging metadata for `ntoskrnl.exe`.

- Unfortunately, the metadata structure alone doesn't retain the correlation between providers and events.
- There are interesting provider names such as `Microsoft.Windows.Kernel.Security` and `AttackSurfaceMonitor`, but it's not yet clear which events belong to these providers.

Step 3 – Write new Modern Trace tooling!



- Ghidra is free. Ghidra is scriptable. And, at the time, I'd just read a blog from Pat Hogan about scripting Ghidra to hunt malware.
- Various projects out there to recover a call graph from a binary – including a great blog from Adam Chester on "Analysing RPC With Ghidra and Neo4j".
- But I wanted more. I wanted as many parameters at the final call site as possible.
- Two relevant calls to correlate – EtwRegister(ProviderId, hProvider) and EtwWrite(hProvider, pDescriptor, pData)

Step 3 – Write new Modern Trace tooling!

Function	ProviderSymbol	ReghandleSymbol	WriteFunction	EventDescriptorSymbol
NtAllocateVirtualMemory	MemoryProvGuid	EtwpMemoryProvRegHandle	EtwWriteEx	KERNEL_MEM_EVENT_ACG
NtAllocateVirtualMemory	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_AUDIT_PROHIBIT_DYNAMIC_CODE
NtAllocateVirtualMemory	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_ENFORCE_PROHIBIT_DYNAMIC_CODE
NtAllocateVirtualMemory	ThreatIntProviderGuid	EtwThreatIntProvRegHandle	EtwWrite	
NtAllocateVirtualMemory	KernelGeneral		_tlgWriteTransfer	GenericMitigationForProcess
NtAllocateVirtualMemory			_tlgWriteEx	ProcessReserveMemFailed
NtCreateSection	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_AUDIT_PROHIBIT_NON_MICROSOFT_BINARIES
NtCreateSection	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_ENFORCE_PROHIBIT_NON_MICROSOFT_BINARIES
NtCreateSection	KernelGeneral		_tlgWriteTransfer	ProhibitNonMicrosoftBinaries
NtDeviceIoControlFile	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_AUDIT_PROHIBIT_FSCTL_SYSTEM_CALLS
NtDeviceIoControlFile	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_ENFORCE_PROHIBIT_FSCTL_SYSTEM_CALLS
NtDeviceIoControlFile	AttackSurfaceMonitor		_tlgWriteTransfer	Ast.IoctlCalled
NtDeviceIoControlFile	KernelGeneral		_tlgWriteTransfer	GenericMitigationForProcess
NtFsControlFile	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_AUDIT_PROHIBIT_FSCTL_SYSTEM_CALLS
NtFsControlFile	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_ENFORCE_PROHIBIT_FSCTL_SYSTEM_CALLS
NtFsControlFile	AttackSurfaceMonitor		_tlgWriteTransfer	Ast.IoctlCalled
NtFsControlFile	KernelGeneral		_tlgWriteTransfer	GenericMitigationForProcess
NtGetEnvironmentVariableEx	Microsoft.Windows.Kernel.SysEnv		_tlgWriteTransfer	GetVariable
NtMapViewOfSection	MemoryProvGuid	EtwpMemoryProvRegHandle	EtwWriteEx	KERNEL_MEM_EVENT_ACG
NtMapViewOfSection	PsProvGuid	EtwpPsProvRegHandle	EtwWriteEx	ImageLoad
NtMapViewOfSection	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_AUDIT_PROHIBIT_DYNAMIC_CODE
NtMapViewOfSection	SecurityMitigationsProviderGuid	EtwSecurityMitigationsRegHandle	EtwWrite	MITIGATION_AUDIT_PROHIBIT_LOWIL_IMAGE_MAP
NtMapViewOfSection	ThreatIntProviderGuid	EtwThreatIntProvRegHandle	EtwWrite	
NtMapViewOfSection	ThreatIntProviderGuid	EtwThreatIntProvRegHandle	EtwWrite	
NtMapViewOfSection	KernelGeneral		_tlgWriteTransfer	GenericMitigationForProcess
NtOpenProcess	KernelAuditApiCallsGuid	EtwApiCallsProvRegHandle	EtwWrite	KERNEL_AUDIT_API_OPENPROCESS
NtOpenProcess	Microsoft.Windows.Kernel.ProcessSubsystem		_tlgWriteTransfer	ProcessOpenFailedForForcedAccessCheck
NtOpenThread	KernelAuditApiCallsGuid	EtwApiCallsProvRegHandle	EtwWrite	KERNEL_AUDIT_API_OPENTHREAD
NtOpenThread	Microsoft.Windows.Kernel.ProcessSubsystem		_tlgWriteTransfer	ThreadOpenFailedForForcedAccessCheck
NtQueryEnvironmentVariableInfoEx	Microsoft.Windows.Kernel.SysEnv		_tlgWriteTransfer	QueryVariables

elastic

- The Microsoft.Windows.Kernel.SysEnv TraceLogging provider has events for the NtGetEnvironmentVaribaleEx syscall
- Probably lots more interesting kernel events to discover.
- Note – because ETW was originally designed for debugging rather than security auditing there is lots of logging for failure cases only.

Step 4 – Write new Legacy Trace tooling!

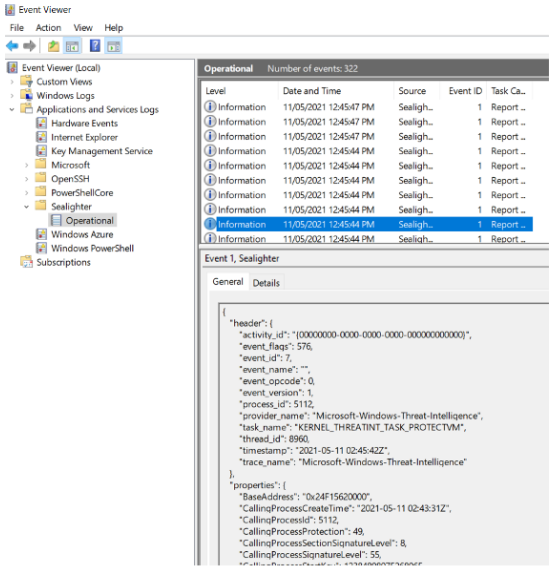
- Public symbols for some drivers
- No `ntoskrnl.exe` WPP trace events

elastic

- WPP metadata is stored within symbols files (PDBs).
- Microsoft includes this information in the public symbols for some, but not all, drivers.
- The kernel itself, however, does not produce any WPP events.

- My Ghidra script includes support for WPP events anyway.
- It should work on any binaries with similar coding styles to the kernel – and preferably with symbols available.

Step 5 – 3rd party Threat-Intelligence PPL?



elastic

- Antimalware-PPL is the Kernel ETW elephant.
- The best events are only available if you have a security driver co-signed by Microsoft.

- Pat Hogan provided the first publicly available ETW-TI tooling to the community.
- https://github.com/pathtofile/SealighterTI

- It didn't quite meet my needs - it logs a json blob to the event log.
- Because of the massive event volumes, I wanted fine-grained filtering.

Step 5 – Write new BYOVD PPL Kernel ETW tooling!

- https://github.com/microsoft/krabsetw

  +

- https://github.com/Mattiwatti/PPLKiller

elastic

- So, I rolled my own.

My Kernel ETW Journey

- Step 1 – Write new Modern Event tooling!
- Step 2 – Write new Legacy Event tooling!
- Step 3 – Write new Modern Trace tooling!
- Step 4 – Write new Legacy Trace tooling!
- Step 5 – Write new BYOVD PPL Kernel ETW tooling!

elastic

- If you haven't noticed, I'm a big fan of writing tooling as a way of consolidating knowledge on a topic.
- Other good approaches include explaining the topic to others including blogging and presenting at SecTalks, CSides or BSides.

My Kernel ETW Journey

- https://github.com/jdu2600/Windows10EtwEvents
- https://github.com/zodiacon/EtwExplorer
- https://github.com/microsoft/krabsetw
- https://github.com/jdu2600/API-To-ETW *new
- https://github.com/jdu2600/ETW-PPL-Tester *new

elastic

- There will also be an Elastic Security Labs blog post for those (like me) that prefer to learn by reading.
- This is also why I try to include speaker notes in my slides.

Other free ETW Tooling

- https://github.com/mandiant/SilkETW
- https://docs.velociraptor.app/blog/2021/2021-08-18-velociraptor-and-etw/
- https://www.elastic.co/docs/current/integrations/windows_etw
- https://www.elastic.co/security/endpoint-security
  - `advanced.events.api_verbose: true`

elastic

- afaik Elastic Endpoint is the only freely available option to receive de-duplicated, enriched ETW-TI events.

https://github.com/jdu2600

https://twitter.com/jdu2600

elastic