Make ETW Great Again.

Exploring some of the many uses of
Event Tracing for Windows (ETW)

CyberPoint
Security
Research
Team

Ben Lelonek
Nate Rogers

cyberpoint

CyberPoint is a cyber security company.

We're in the business of protecting what's invaluable to you.

Who We Are

CyberPoint Security Research Team

CyberPoint Security Research Team

- www.cyberpointllc.com/srt
- SRT@cyberpointllc.com
- @CyberPoint_SRT

Nate "Million Dollars" Rogers

- CyberPoint International
  - Security Research Team Lead
- Student at NYU
- Previously:
  - eEye Digital Security
- Twitter:
  - @Conjectural_Hex

Ben "Texas Dirt" Lelonek

- CyberPoint International
  - Security Research Team / Developer
- Student at UMBC

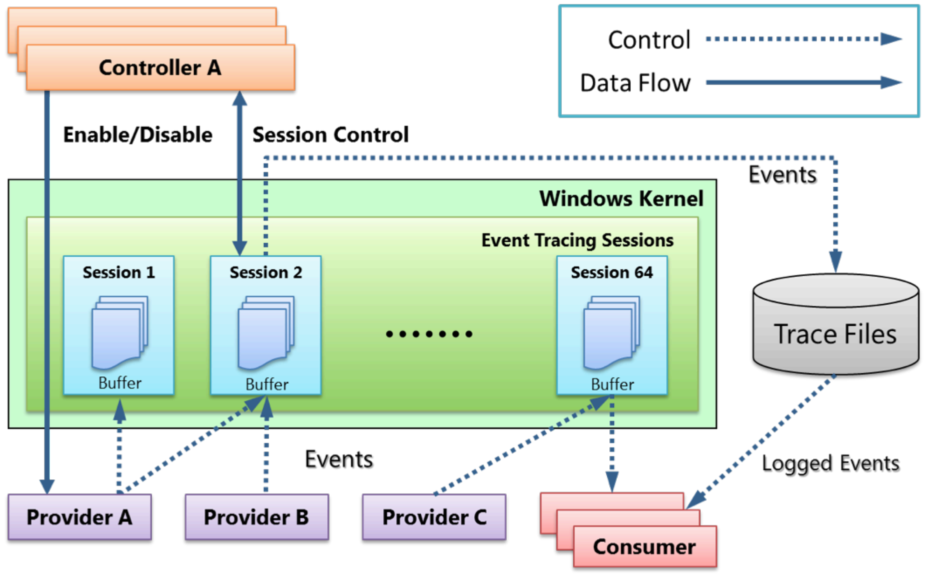
What we're going to be talking about.

- What is ETW
- Quick Overview of ETW
- Usage Examples
- Public Uses and Research
- ETW for Malware Detection
- ETW for Red Team
- Mitigations
- Questions

What is Event Tracing for Windows (ETW)?

- Built-in, general purpose, logging and diagnostic framework
- Efficient: high speed, low overhead
- Dynamically enabled or disabled
- Log to file or consume in real time
- Used for performance analysis and general debugging
- Example usage
  - Google Chrome
    - Performance analysis & profiling
    - UIforETW

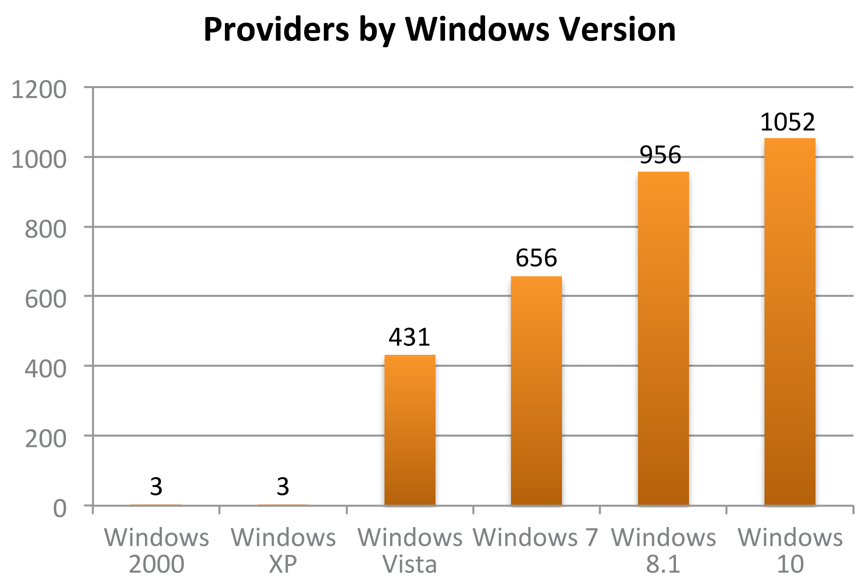
ETW Architecture



Source: https://msdn.microsoft.com/en-us/windows/hardware/commercialize/test/weg/weg-performance

Quick Overview of ETW

- First introduced in Windows 2000
- Greatly expanded in Vista
  - New manifest-based providers and logging in more than just the kernel
  - More in each OS since



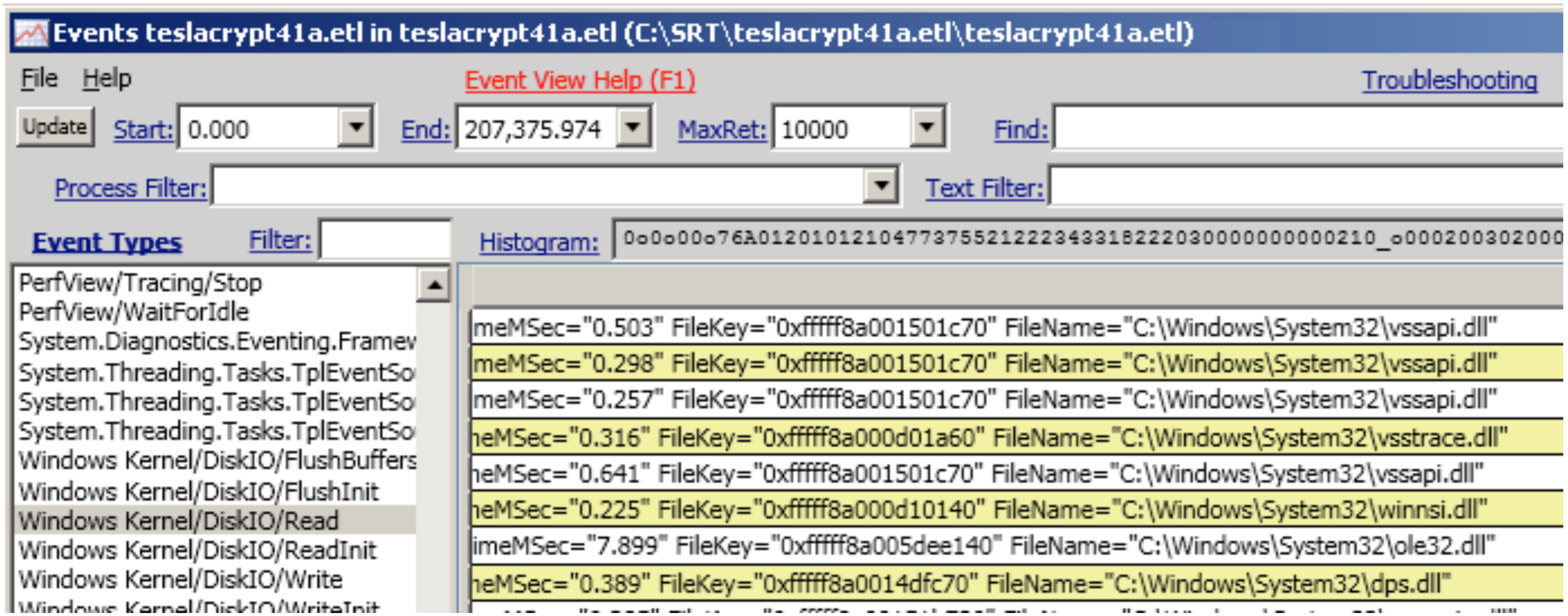
- Ease of use improved with each OS
  - Windows 2000 – MOF classes and WMI
  - Windows Vista – XML Manifests
  - Windows 8/.NET 4.5 – EventSource (C#)
  - Windows 10 – TraceLogging

How to View ETW Events

- API
  - Less commonly used, focus of our work
  - Microsoft.Diagnostics.Tracing.TraceEvent.dll
  - C/C++/C#/etc
- Command Line / Applications
  - More commonly used
  - Built-in: Logman, TraceRpt, Event Viewer, Performance Monitor, wevtutil
  - Installable: Xperf, PerfView, Netmon, Microsoft Message Analyzer, Windows Performance Analyzer
- PerfView example…

Viewing ETW Events – PerfView

Teslacrypt reading files in System32

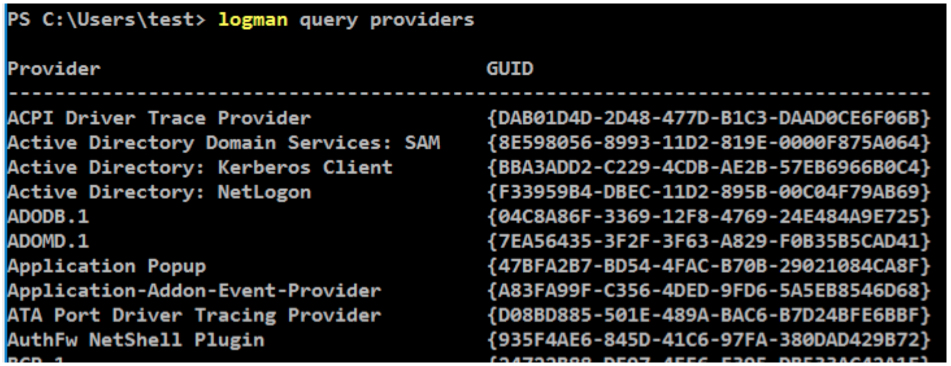

Events teslacrypt41a.etl in teslacrypt41a.etl (C:\SRT\teslacrypt41a.etl\teslacrypt41a.etl)
File
Help
Event View Help (F1)
Troubleshooting
Update
Start: 0.000
End: 207,375.974
MaxRet: 10000
Find:
Process Filter:
Text Filter:
Event Types
Filter:
Histogram: 0o0o00o76A0120101210477375521222343318222030000000000210_o000200302000

```
PerfView/Tracing/Stop
PerfView/WaitForIdle
System.Diagnostics.Eventing.Framew
System.Threading.Tasks.TplEventSo
System.Threading.Tasks.TplEventSo
System.Threading.Tasks.TplEventSo
Windows Kernel/DiskIO/FlushBuffers
Windows Kernel/DiskIO/FlushInit
Windows Kernel/DiskIO/Read
Windows Kernel/DiskIO/ReadInit
Windows Kernel/DiskIO/Write
```

```
meMSec="0.503" FileKey="0xfffff8a001501c70" FileName="C:\Windows\System32\vssapi.dll"
meMSec="0.298" FileKey="0xfffff8a001501c70" FileName="C:\Windows\System32\vssapi.dll"
meMSec="0.257" FileKey="0xfffff8a001501c70" FileName="C:\Windows\System32\vssapi.dll"
neMSec="0.316" FileKey="0xfffff8a000d01a60" FileName="C:\Windows\System32\vsstrace.dll"
neMSec="0.641" FileKey="0xfffff8a001501c70" FileName="C:\Windows\System32\vssapi.dll"
neMSec="0.225" FileKey="0xfffff8a000d10140" FileName="C:\Windows\System32\winnsi.dll"
imeMSec="7.899" FileKey="0xfffff8a005dee140" FileName="C:\Windows\System32\ole32.dll"
neMSec="0.389" FileKey="0xfffff8a0014dfc70" FileName="C:\Windows\System32\dps.dll"
```

ETW Example Providers

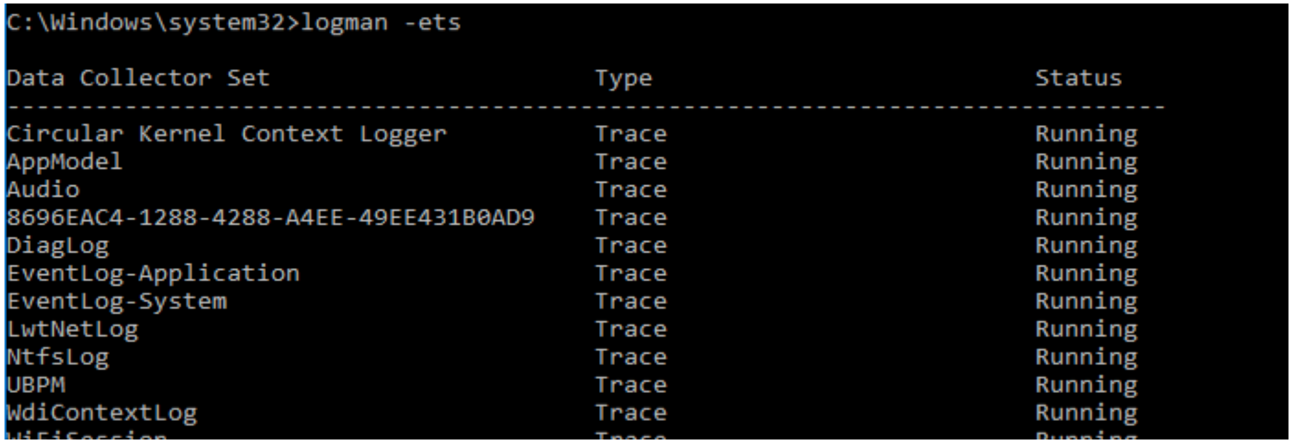
- Listing providers



```
PS C:\Users\test> logman query providers

Provider                                 GUID
-------------------------------------------------------------------------------
ACPI Driver Trace Provider               {DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
Active Directory Domain Services: SAM    {8E598056-8993-11D2-819E-0000F875A064}
Active Directory: Kerberos Client        {BBA3ADD2-C229-4CDB-AE2B-57EB6966B0C4}
Active Directory: NetLogon               {F33959B4-DBEC-11D2-895B-00C04F79AB69}
ADODB.1                                  {04C8A86F-3369-12F8-4769-24E484A9E725}
ADOMD.1                                  {7EA56435-3F2F-3F63-A829-F0B35B5CAD41}
Application Popup                        {47BFA2B7-BD54-4FAC-B70B-29021084CA8F}
Application-Addon-Event-Provider         {A83FA99F-C356-4DED-9FD6-5A5EB8546D68}
ATA Port Driver Tracing Provider         {D08BD885-501E-489A-BAC6-B7D24BFE6BBF}
AuthFw NetShell Plugin                   {935F4AE6-845D-41C6-97FA-380DAD429B72}
```

- Listing running sessions

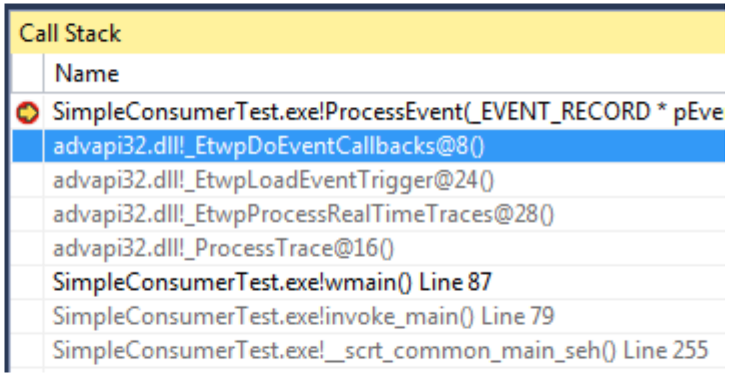


```
C:\Windows\system32>logman -ets

Data Collector Set                      Type                          Status
-------------------------------------------------------------------------------
Circular Kernel Context Logger          Trace                         Running
AppModel                                Trace                         Running
Audio                                   Trace                         Running
8696EAC4-1288-4288-A4EE-49EE431B0AD9    Trace                         Running
DiagLog                                 Trace                         Running
EventLog-Application                    Trace                         Running
EventLog-System                         Trace                         Running
LwtNetLog                               Trace                         Running
NtfsLog                                 Trace                         Running
UBPM                                    Trace                         Running
WdiContextLog                           Trace                         Running
```

Using ETW

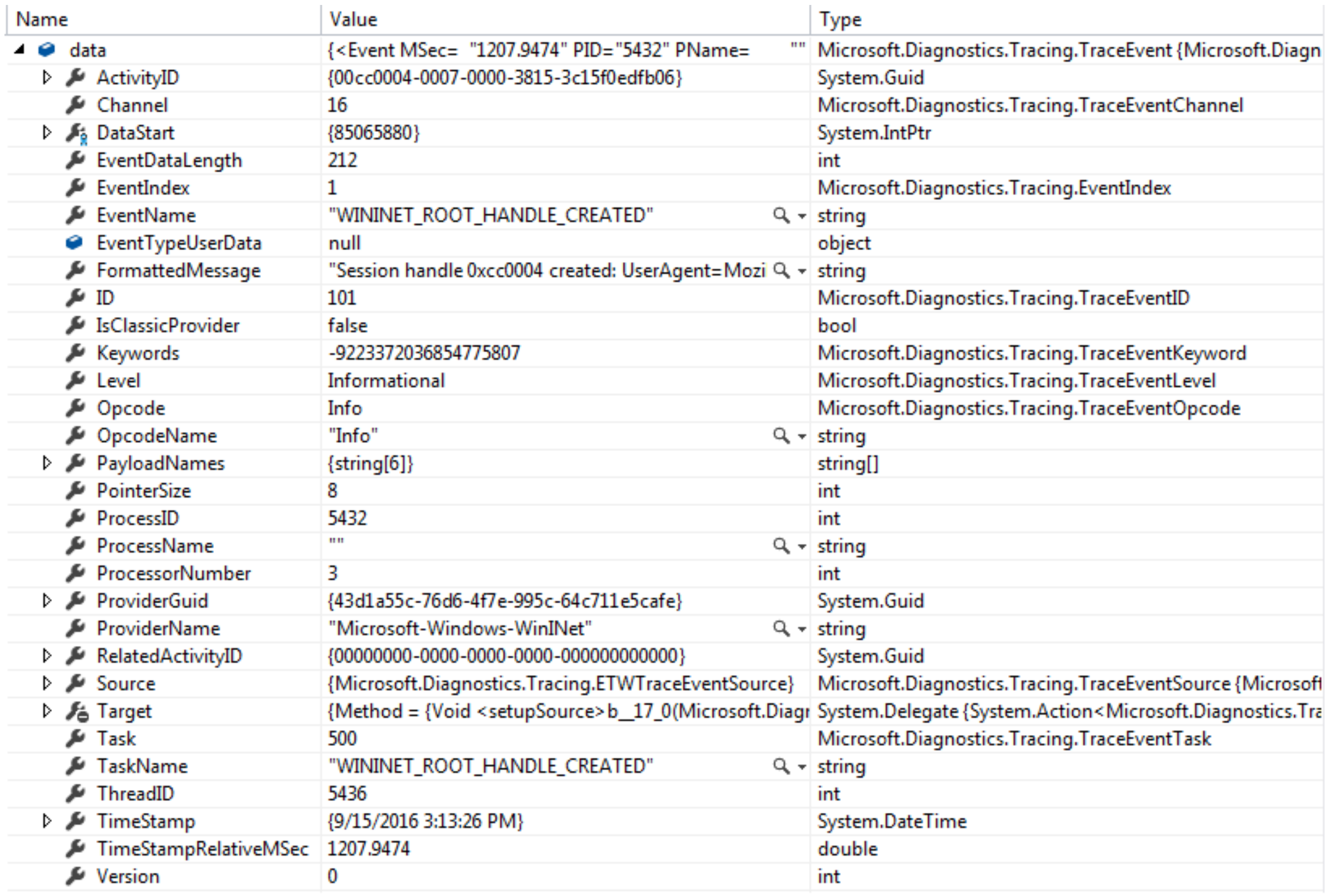
- ETW Events are handled Asynchronously
  - System / Application writes them to the kernel
  - Consumers must establish a session and subscribe to get data
- Typical ETW Structure
  - C/C++: EVENT_HEADER, EVENT_RECORD, EVENT_TRACE structures and trace data helper (TDH) functions
  - C#: TraceEvent object, PayloadStringByName()
- Mechanism
  - OS-side implementation details not publicly available
  - Callbacks from the OS
- Events Can be Collected Remotely
  - Configured via WMI, Powershell
  - Collector machine pulls data from workers



```
Call Stack
Name
SimpleConsumerTest.exe!ProcessEvent(_EVENT_RECORD * pEve
advapi32.dll!_EtwpDoEventCallbacks@8()
advapi32.dll!_EtwpLoadEventTrigger@24()
advapi32.dll!_EtwpProcessRealTimeTraces@28()
advapi32.dll!_ProcessTrace@16()
SimpleConsumerTest.exe!wmain() Line 87
SimpleConsumerTest.exe!invoke_main() Line 79
SimpleConsumerTest.exe!__scrt_common_main_seh() Line 255
```

TraceEvent object

TONS of information!

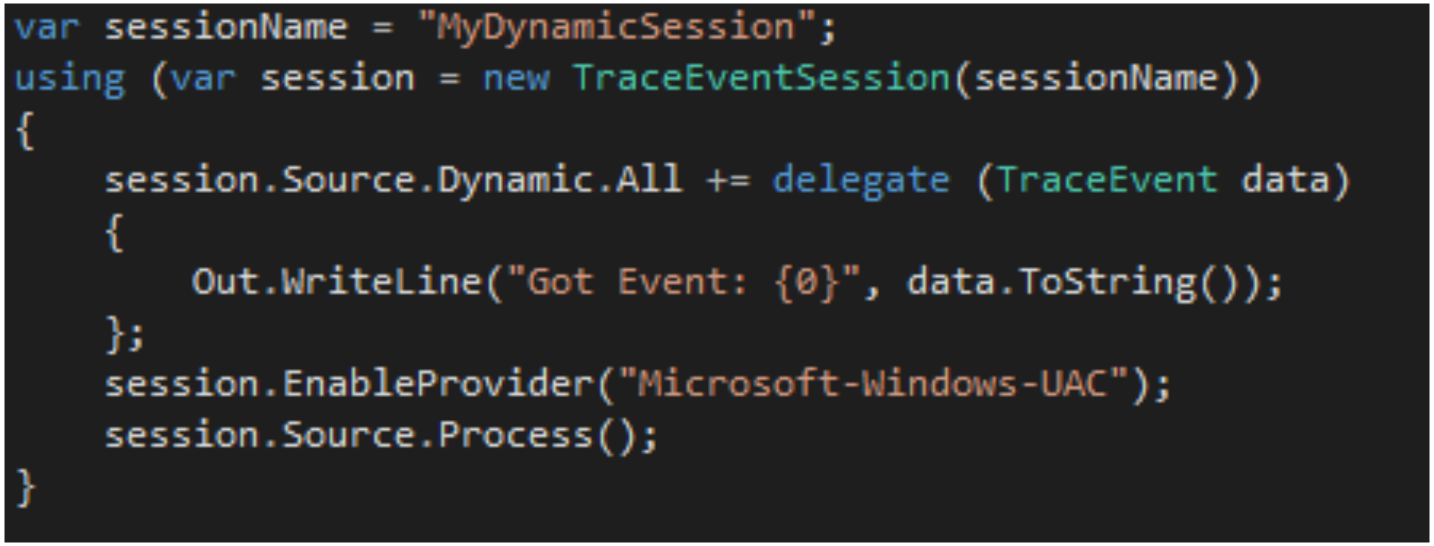


Name	Value	Type
data	{<Event MSec= "1207.9474" PID="5432" PName= ""	Microsoft.Diagnostics.Tracing.TraceEvent {Microsoft.Diagn
ActivityID	{00cc0004-0007-0000-3815-3c15f0edfb06}	System.Guid
Channel	16	Microsoft.Diagnostics.Tracing.TraceEventChannel
DataStart	{85065880}	System.IntPtr
EventDataLength	212	int
EventIndex	1	Microsoft.Diagnostics.Tracing.EventIndex
EventName	"WININET_ROOT_HANDLE_CREATED"	string
EventTypeUserData	null	object
FormattedMessage	"Session handle 0xcc0004 created: UserAgent=Mozi	string
ID	101	Microsoft.Diagnostics.Tracing.TraceEventID
IsClassicProvider	false	bool
Keywords	-9223372036854775807	Microsoft.Diagnostics.Tracing.TraceEventKeyword
Level	Informational	Microsoft.Diagnostics.Tracing.TraceEventLevel
Opcode	Info	Microsoft.Diagnostics.Tracing.TraceEventOpcode
OpcodeName	"Info"	string
PayloadNames	{string[6]}	string[]
PointerSize	8	int
ProcessID	5432	int
ProcessName	""	string
ProcessorNumber	3	int
ProviderGuid	{43d1a55c-76d6-4f7e-995c-64c711e5cafe}	System.Guid
ProviderName	"Microsoft-Windows-WinINet"	string
RelatedActivityID	{00000000-0000-0000-0000-000000000000}	System.Guid
Source	{Microsoft.Diagnostics.Tracing.ETWTraceEventSource}	Microsoft.Diagnostics.Tracing.TraceEventSource {Microsoft
Target	{Method = {Void <setupSource>b__17_0(Microsoft.Diagr	System.Delegate {System.Action<Microsoft.Diagnostics.Tra
Task	500	Microsoft.Diagnostics.Tracing.TraceEventTask
TaskName	"WININET_ROOT_HANDLE_CREATED"	string
ThreadID	5436	int
TimeStamp	{9/15/2016 3:13:26 PM}	System.DateTime
TimeStampRelativeMSec	1207.9474	double
Version	0	int

Using ETW API (C#)

Example Simple UAC Event Listener

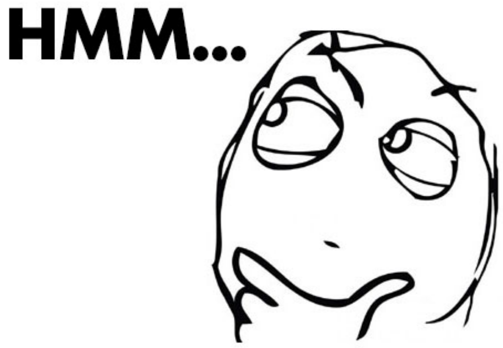
- Extremely easy to implement



```
var sessionName = "MyDynamicSession";
using (var session = new TraceEventSession(sessionName))
{
    session.Source.Dynamic.All += delegate (TraceEvent data)
    {
        Out.WriteLine("Got Event: {0}", data.ToString());
    };
    session.EnableProvider("Microsoft-Windows-UAC");
    session.Source.Process();
}
```

Great, so what does this have to do with security?

- Extensive Integration with Windows
  - Much of the Windows API logs to ETW
  - Vast amount of Windows Subsystems have providers
  - Can be used to collect information for both attackers & defenders/auditors
- Universally Deployed in Windows
  - Exists *(in some form)* in every version since Windows 2000
  - Data provider enabled on demand
  - Huge potential for abuse
    - We'll get back to this later…
  - Great potential for defensive applications/research
    - Lots of potential data points for collection/heuristics
      - process, .NET/CLR, Kernel, IO, Files, Memory, UAC, Logins, Crypto, Firewall, SMB, TCPIP, MANY more…
    - Some examples/tools exist but can be improved



Public Uses and Research

- Defensive
  - Data Mining Heuristics
    - Collecting ETW logs to detect malware
  - Ransomware detection (not ETW)
    - Track file IO / handles
      - Similar to our technique (next slide)
      - Uses driver

- Offensive
  - Persistence
    - ETW triggering service execution
  - Packet capture
    - logman/netsh for capturing network traffic
  - "SSL Sidejacking" / Cookie Stealing
    - ETW listener for WinINet can snoop on traffic *(even SSL/TLS)*

ETW Malware Detection: Room for Improvement

- Few malware ETW tools
  - Existing techniques all use external EXEs
    - Logman.exe, wevtutil.exe, PerfView, etc.
    - Often focus on network traffic (!Ransomware)
  - Can't parse in "real" time
    - Must log to disk then parse
- Ransomware ETW solutions?
  - Virtually none
- Goals:
  - More lightweight *(less overhead)* solutions would be optimal
  - Native ETW API
    - Standalone binary with no dependencies
  - Static AND Dynamic
    - Detect Ransomware in real time
    - Also support captures *(.etl)*

Detecting Ransomware – Our Approach

Classify and Distill Ransomware Behavior

- Iterate files
  - Extension based, location based, etc.
- Read/writing to files
  - access times, creation times, different sizes *(read vs. write)*, location
- Encryption
  - AES, custom, GOST, RSA, Blowfish, TripleDES, XOR, RC4, Salsa20, TEA, zip, rar, etc.
- Move/Rename/Copy/Delete
  - Many different ways to deal with "original" file

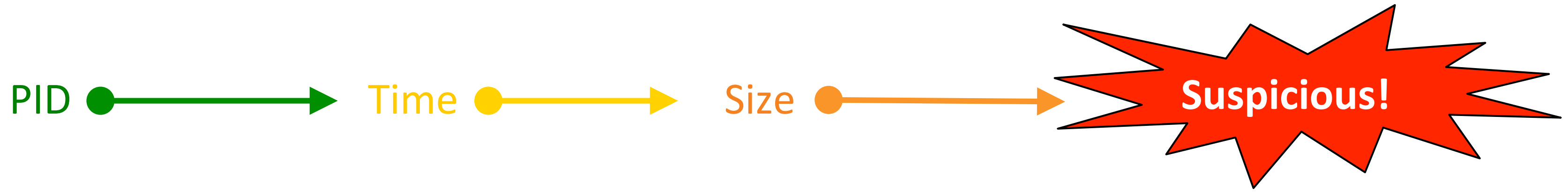
Detecting Ransomware – Our Approach *(cont.)*

Is generalization of behavior possible for all samples?

- Read then Write
  - Yes, but varies…
  - Lots of false positives
  - Timing Threshold?
    - account for OS delays, iterations, etc.
- File Size Delta?
  - Encrypted file vs. original
  - Different encryption, IVs, etc., add size!
  - Sizes deltas vary
    - Lots of false positives in benign processes
- File Name Changes
  - Original file name vs. Encrypted
  - Original is in encrypted name *(in some form)*
    - Almost always
- Encryption
  - Too much variance for generic rule

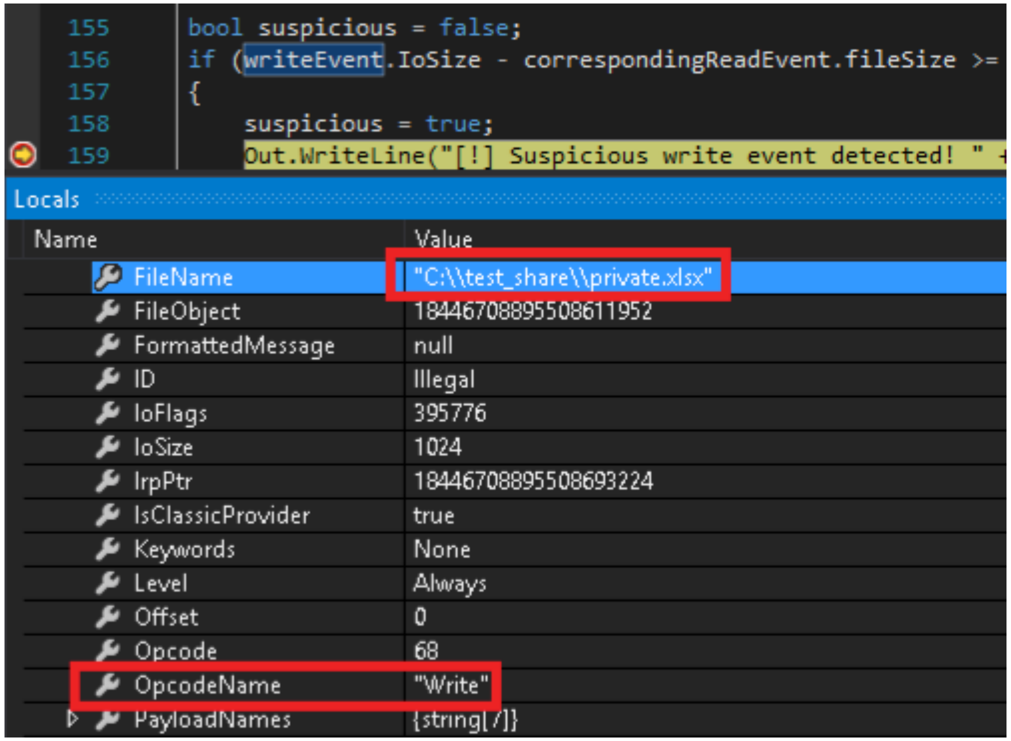
Detecting Ransomware – Our Approach *(cont.)*

- Generic Detection Algorithm
  - Track writes to files that were previously read
    - Must be the same PID
    - Must be within time threshold 80ms
      - Highest average ~49ms *(Nanolocker)*
    - Must be within size delta threshold 1024 bytes
      - Higher than needed for malware
      - Browser caches and temp files
  - If above criteria is met increment SuspiciousEvent counter
- Suspicious Event Counter = 3
  - Filter false positives
    - temp files, caching, windows search, etc.



Detecting Ransomware – Our Approach *(cont.)*

- Which provider is needed?
  - "Windows Kernel"
  - Can use others but not necessary
- What data is needed from provider?
  - "Type Field"
    - "FileIOReadWriteTraceData"
    - Multiple Event Types
  - EventName
    - "FileIO/Write"
    - "FileIO/Read"
  - "OpCode"
    - Sub-types know as OpCodes
    - represented with INT and ASCII name
      - OpcodeNames: "Read", "Write"
      - Opcode Values: 0x67, 0x68



```
bool suspicious = false;
if (writeEvent.IoSize - correspondingReadEvent.fileSize >=
{
    suspicious = true;
    Out.WriteLine("[!] Suspicious write event detected! " +
```

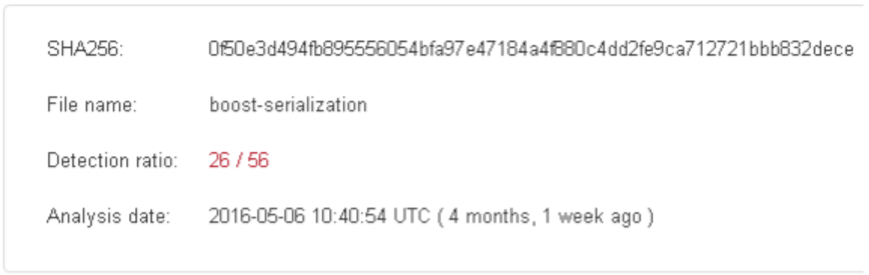
Locals

Name	Value
FileName	"C:\\test_share\\private.xlsx"
FileObject	18446708895508611952
FormattedMessage	null
ID	Illegal
IoFlags	395776
IoSize	1024
IrpPtr	18446708895508693224
IsClassicProvider	true
Keywords	None
Level	Always
Offset	0
Opcode	68
OpcodeName	"Write"
PayloadNames	{string[7]}

What can we detect?

- **EVERYTHING!** *(That we tested.)*
- Specifically, cerber, chimera, ctb-locker, locky, hydracrypt, jigsaw, lockscreen, mobef, radamant, samsam, shade, teslascrypt, torrentlocker, trucrypter, 7ev3n, coverton, kimcilware, petya
- Generically detected all samples
- Even those with *(relatively)* low detections on VirusTotal
- TorrentLocker:

virustotal



SHA256:	0f50e3d494fb895556054bfa97e47184a4f880c4dd2fe9ca712721bbb832dece
File name:	boost-serialization
Detection ratio:	26 / 56
Analysis date:	2016-05-06 10:40:54 UTC (4 months, 1 week ago)

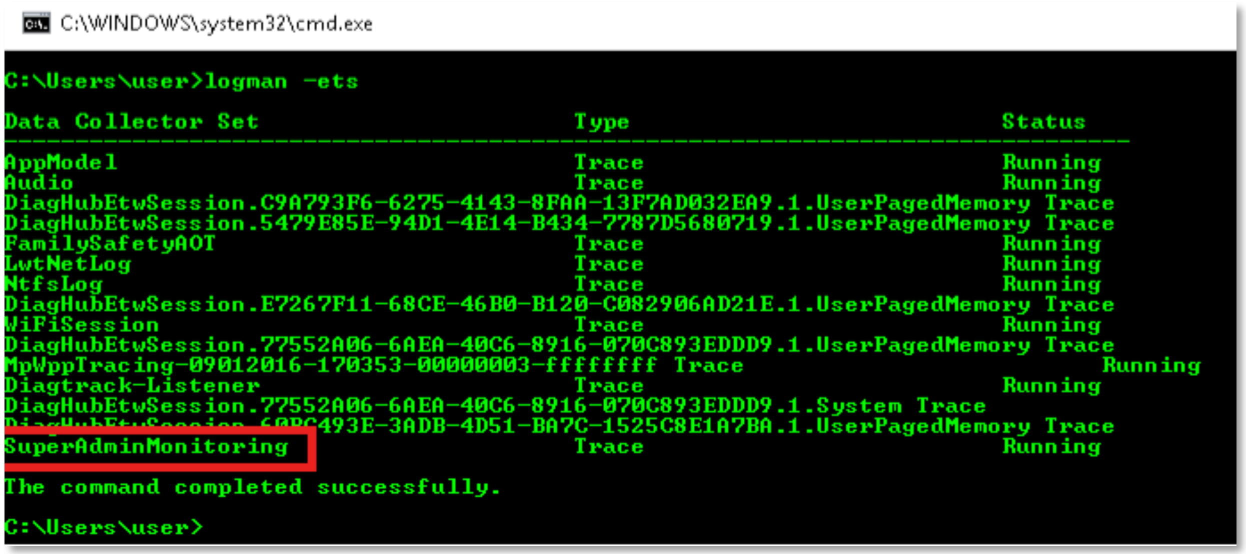
ETW & Ransomware Detection Limitations

- Not Perfect
  - Needs at least 3 files to be encrypted to be effective
- Dynamic Captures can be delayed
  - Varies greatly
  - Depends on number of consumed events, system activity, etc.
  - Usually small delay
- Hard to Hide Sessions from Malware and Attacker
  - Easy for malware to see who’s “listening”
    - Trivial to access...

Malware Detection of ETW

How easily can attackers "see" ETW?

- Anti-Analysis?
- Easy to see sessions – logman.exe, C# API
- No Baseline of sessions or providers
  - Which are good? Which are bad?


C:\WINDOWS\system32\cmd.exe

```
C:\Users\user>logman -ets

Data Collector Set                                    Type                                    Status
-------------------------------------------------------------------------------------------------------
AppModel                                              Trace                                   Running
Audio                                                 Trace                                   Running
DiagHubEtwSession.C9A793F6-6275-4143-8FAA-13F7AD032EA9.1.UserPagedMemory Trace
DiagHubEtwSession.5479E85E-94D1-4E14-B434-7787D5680719.1.UserPagedMemory Trace
FamilySafetyAOT                                       Trace                                   Running
LwtNetLog                                             Trace                                   Running
NtfsLog                                               Trace                                   Running
DiagHubEtwSession.E7267F11-68CE-46B0-B120-C082906AD21E.1.UserPagedMemory Trace
WiFiSession                                           Trace                                   Running
DiagHubEtwSession.77552A06-6AEA-40C6-8916-070C893EDDD9.1.UserPagedMemory Trace
MpWppTracing-09012016-170353-00000003-ffffffff Trace                                                   Running
Diagtrack-Listener                                    Trace                                   Running
DiagHubEtwSession.77552A06-6AEA-40C6-8916-070C893EDDD9.1.System Trace
DiagHubEtwSession.60BC493E-3ADB-4D51-BA7C-1525C8E1A7BA.1.UserPagedMemory Trace
SuperAdminMonitoring                                  Trace                                   Running

The command completed successfully.

C:\Users\user>
```

ETW Providers for Red Team

Tons of potential ETW providers!

- Some uses are obvious
  - Winlogin, SCM, WLAN, WMI, Firewall, UAC, TCPIP, Task Scheduling, SMB, SmartCards, Terminal Services, Powershell, Location, Kernel Resources/ Events,IPSEC, FileHistory/FileManage, DNS/DHCP Client, BlueTooth, Bits, BitLocker, Cryptography, Antimalware, LsaSrv, SAM, ActiveDirectory
- Some are a little less...
  - Microsoft-Windows-Bluetooth-HidBthLE
  - Microsoft-Windows-USB-UCX
  - Microsoft-Windows-WinINet
  - Etc....

Most have Good Potential

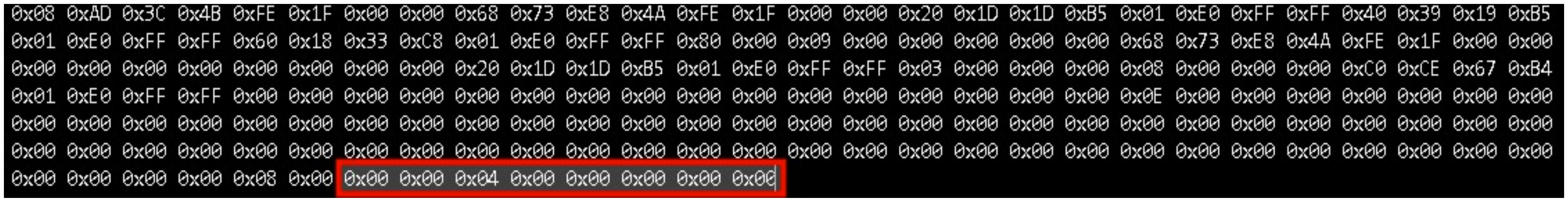
- All require closer inspection before use
  - Some more than others *(USB)*
- Lots of Metadata
  - Must be filtered out

USB Key Logging with ETW

- Motivation
  - USB key logging discussed but no tools exist
  - API based, no dependencies
    - No need to log to disk first
    - More “tactical” solution
- ETW is VERBOSE, especially with USB-UCX Data
  - ETW provides RAW USB data
  - Requires we parse it ourselves
  - USB Keyboards poll
    - Send data regardless of key press
    - Poll rate: 125 Hz = 8ms
- Providers
  - Microsoft-Windows-USB-UCX - {36DA592D-E43A-4E28-AF6F-4BC57C5A11E8}
  - Microsoft-Windows-USB-USBPORT - {C88A4EF5-D048-4013-9408-E04B7DB2814A}
- Pros
  - ETW is INTENDED functionality (debugging)
  - New Technique. No AV coverage… yet
  - Can capture keystrokes when computer is locked!
- Cons
  - Real time ETW captures can have delays
  - Requires admin

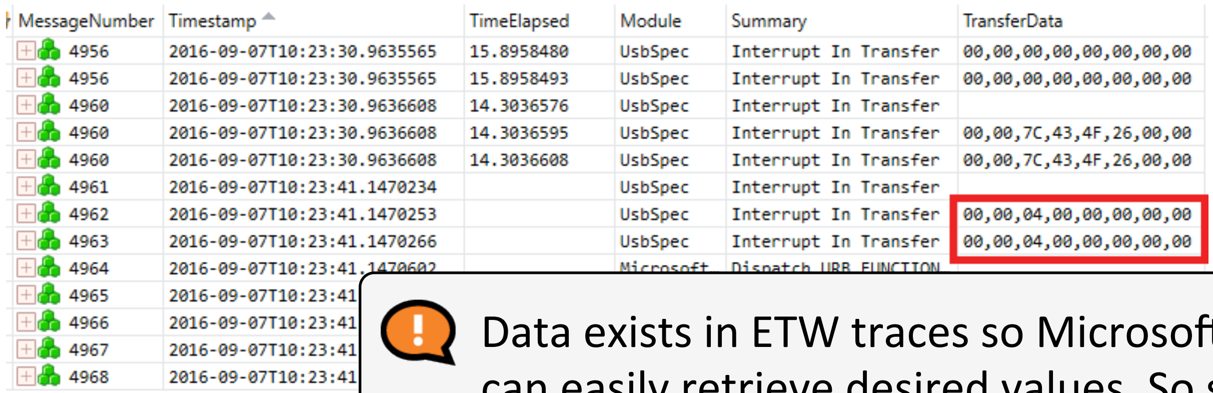
Microsoft Message Analyzer FTW!

- Microsoft Message Analyzer (MMA) **GREATLY** reduced the “noise” on the wire
- Excellent tool for USB, general ETW troubleshooting
- Does most USB/ETW parsing for you
  - From this...

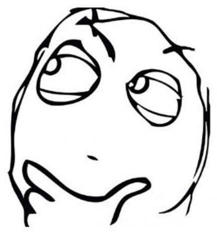


```
0x08 0xAD 0x3C 0x4B 0xFE 0x1F 0x00 0x00 0x68 0x73 0xE8 0x4A 0xFE 0x1F 0x00 0x00 0x20 0x1D 0x1D 0xB5 0x01 0xE0 0xFF 0xFF 0x40 0x39 0x19 0xB5
0x01 0xE0 0xFF 0xFF 0x60 0x18 0x33 0xC8 0x01 0xE0 0xFF 0xFF 0x80 0x00 0x09 0x00 0x00 0x00 0x00 0x00 0x68 0x73 0xE8 0x4A 0xFE 0x1F 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x20 0x1D 0x1D 0xB5 0x01 0xE0 0xFF 0xFF 0x03 0x00 0x00 0x00 0x08 0x00 0x00 0x00 0xC0 0xCE 0x67 0xB4
0x01 0xE0 0xFF 0xFF 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0E 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x08 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x00 0x00
```

  - To this!



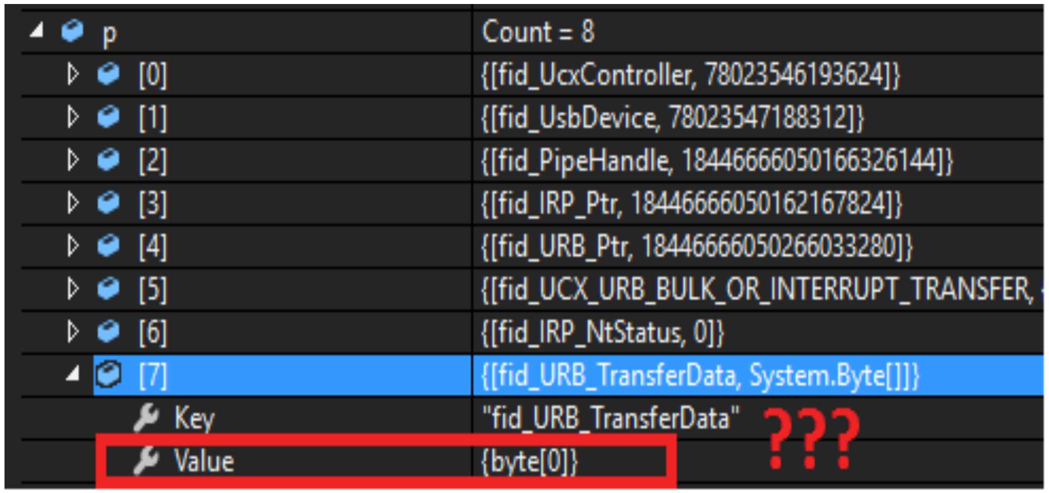
MessageNumber	Timestamp	TimeElapsed	Module	Summary	TransferData
4956	2016-09-07T10:23:30.9635565	15.8958480	UsbSpec	Interrupt In Transfer	00,00,00,00,00,00,00,00
4956	2016-09-07T10:23:30.9635565	15.8958493	UsbSpec	Interrupt In Transfer	00,00,00,00,00,00,00,00
4960	2016-09-07T10:23:30.9636608	14.3036576	UsbSpec	Interrupt In Transfer	
4960	2016-09-07T10:23:30.9636608	14.3036595	UsbSpec	Interrupt In Transfer	00,00,7C,43,4F,26,00,00
4960	2016-09-07T10:23:30.9636608	14.3036608	UsbSpec	Interrupt In Transfer	00,00,7C,43,4F,26,00,00
4961	2016-09-07T10:23:41.1470234		UsbSpec	Interrupt In Transfer	
4962	2016-09-07T10:23:41.1470253		UsbSpec	Interrupt In Transfer	00,00,04,00,00,00,00,00
4963	2016-09-07T10:23:41.1470266		UsbSpec	Interrupt In Transfer	00,00,04,00,00,00,00,00
4964	2016-09-07T10:23:41.1470602		Microsoft	Dispatch URB_FUNCTION	
4965	2016-09-07T10:23:41				
4966	2016-09-07T10:23:41				
4967	2016-09-07T10:23:41				
4968	2016-09-07T10:23:41				



Data exists in ETW traces so Microsoft’s TraceEvent library can easily retrieve desired values. So simple, right?!

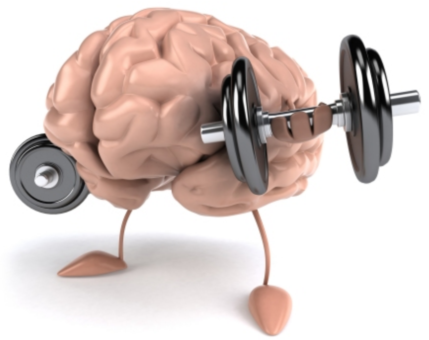
Actually Parsing Events

- Unfortunately TraceEvent isn't perfect
  - TraceEvent returns an empty byte[] with the xferData
- We know data is there
  - MMA & Xperf, etc (previous slide)
- Had to dump the whole ETW payload and parse ourselves
  - Just takes a little extra work...



```
◢ p          Count = 8
  ▷ [0]      {[fid_UcxController, 78023546193624]}
  ▷ [1]      {[fid_UsbDevice, 78023547188312]}
  ▷ [2]      {[fid_PipeHandle, 18446666050166326144]}
  ▷ [3]      {[fid_IRP_Ptr, 18446666050162167824]}
  ▷ [4]      {[fid_URB_Ptr, 18446666050266033280]}
  ▷ [5]      {[fid_UCX_URB_BULK_OR_INTERRUPT_TRANSFER,
  ▷ [6]      {[fid_IRP_NtStatus, 0]}
  ◢ [7]      {[fid_URB_TransferData, System.Byte[]]}
      Key    "fid_URB_TransferData"
      Value  {byte[0]}
```

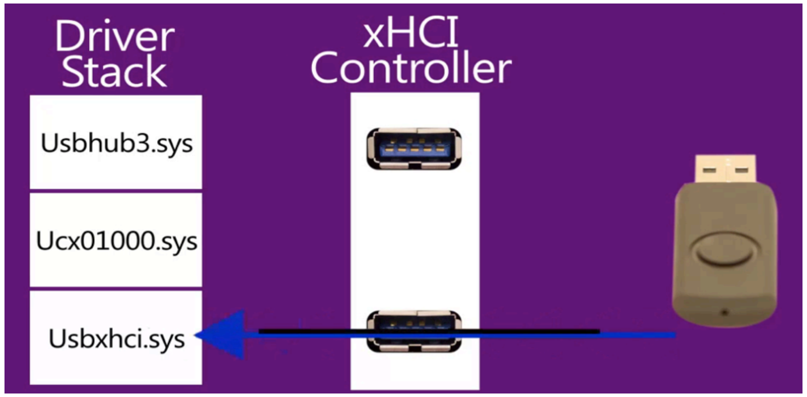
???



Quick Note Sniffing USB

What to do with the data?

- Data blobs represent raw bytes on the wire + ETW headers
  - Strip off ETW and parse reaming data
  - Remaining data is USB Request Block (URB)
- Data from devices must be processed by drivers
  - Usbxhci.sys -> Ucx01000.sys -> USBhub3.sys (USB3)
  - We can cheat using ETW headers!
- Human Interface Device *(HID)* data in URB_FUNCTION: _URB_BULK_OR_INTERRUPT_TRANSFER

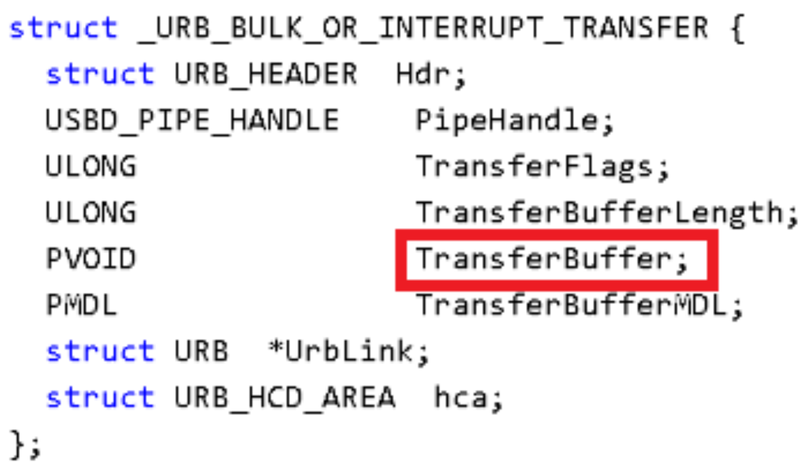


Source: https://msdn.microsoft.com/en-us/library/windows/hardware/dn741264(v=vs.85).aspx

Filtering and Parsing Events

Turn Raw Data in HID data

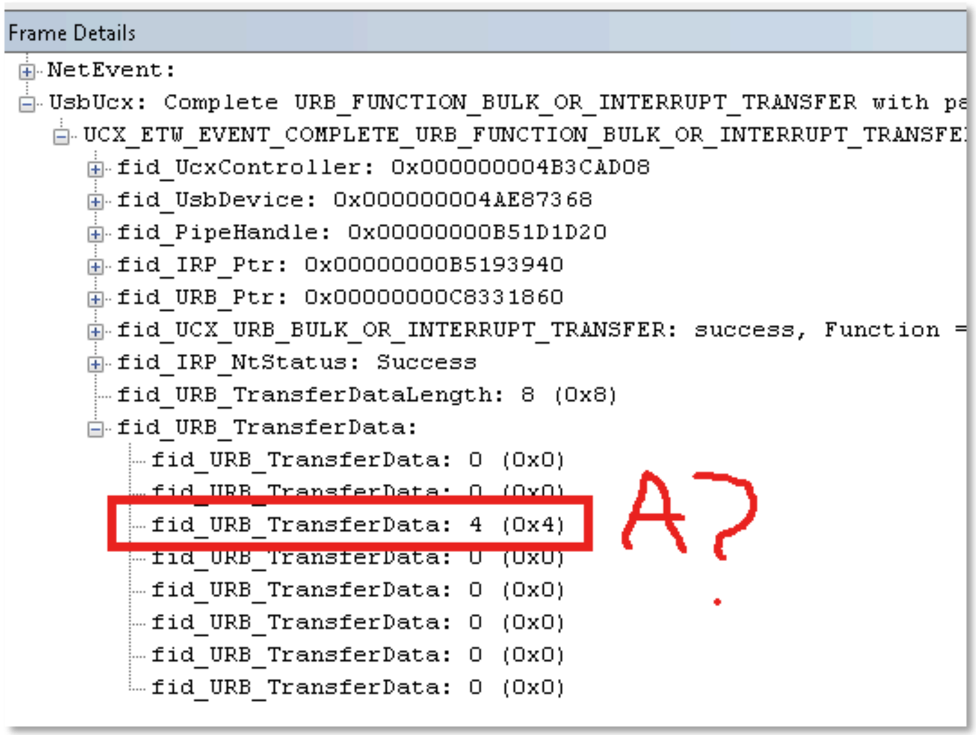
- Find USB Request Blocks (URBs) of interest
  - UCX_URB_BULK_OR_INTERRUPT_TRANSFER
  - "payload": TransferBuffer
- Find Correct payload size
  - fid_URB_TransferDataLength
    - Keyboard HID packets = 8 bytes
    - Mouse HID payload = 4 bytes
- Get Data!
  - fid_URB_TransferData



```
struct _URB_BULK_OR_INTERRUPT_TRANSFER {
  struct URB_HEADER  Hdr;
  USBD_PIPE_HANDLE    PipeHandle;
  ULONG               TransferFlags;
  ULONG               TransferBufferLength;
  PVOID               TransferBuffer;
  PMDL                TransferBufferMDL;
  struct URB  *UrbLink;
  struct URB_HCD_AREA  hca;
};
```

USB HID Usage Tables

- fid_URB_TransferData
  - "Payload" from HID data = keystroke
- Payload is then mapped to HID spec



```
Frame Details
 NetEvent:
 UsbUcx: Complete URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER with pe
   UCX_ETW_EVENT_COMPLETE_URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFE
     fid_UcxController: 0x000000004B3CAD08
     fid_UsbDevice: 0x000000004AE87368
     fid_PipeHandle: 0x00000000B51D1D20
     fid_IRP_Ptr: 0x00000000B5193940
     fid_URB_Ptr: 0x00000000C8331860
     fid_UCX_URB_BULK_OR_INTERRUPT_TRANSFER: success, Function =
     fid_IRP_NtStatus: Success
     fid_URB_TransferDataLength: 8 (0x8)
     fid_URB_TransferData:
       fid_URB_TransferData: 0 (0x0)
       fid_URB_TransferData: 0 (0x0)
       fid_URB_TransferData: 4 (0x4)
       fid_URB_TransferData: 0 (0x0)
       fid_URB_TransferData: 0 (0x0)
       fid_URB_TransferData: 0 (0x0)
       fid_URB_TransferData: 0 (0x0)
       fid_URB_TransferData: 0 (0x0)
```

A?

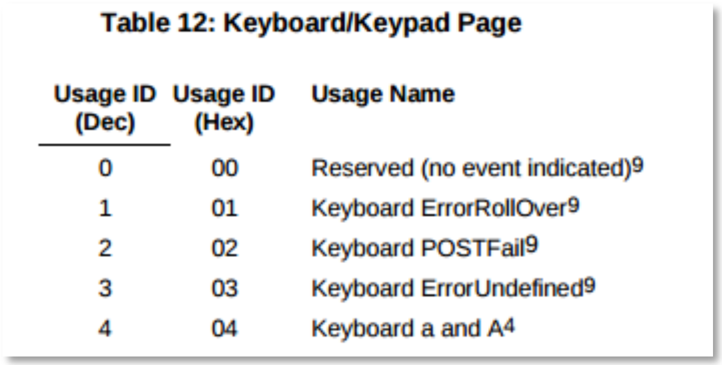
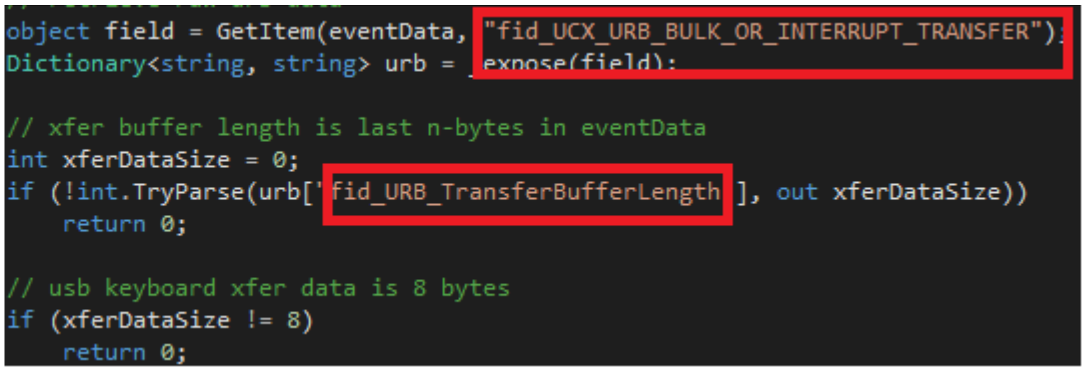


Table 12: Keyboard/Keypad Page

Usage ID (Dec)	Usage ID (Hex)	Usage Name
0	00	Reserved (no event indicated)9
1	01	Keyboard ErrorRollOver9
2	02	Keyboard POSTFail9
3	03	Keyboard ErrorUndefined9
4	04	Keyboard a and A4

Actually Parsing ETW USB Events in C#

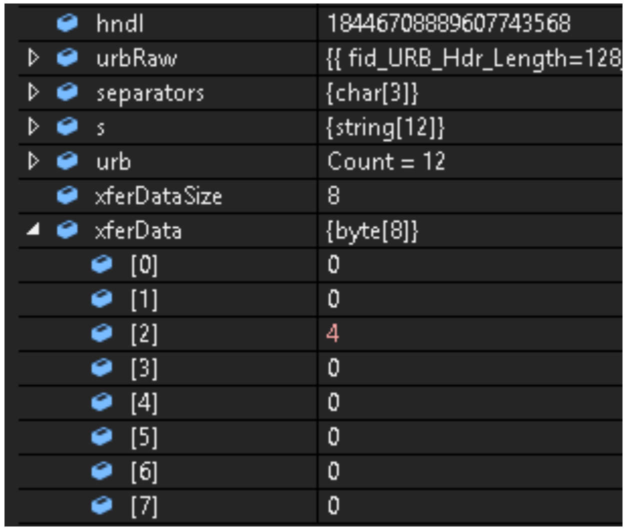
- Use ETW to find correct URB
  - UCX_URB_BULK_OR_INTERRUPT_TRANSFER
- Use ETW to select payload size for keyboards
  - TransferBufferLength
- Manually populate xferData with URB payload



```
object field = GetItem(eventData, "fid_UCX_URB_BULK_OR_INTERRUPT_TRANSFER");
Dictionary<string, string> urb = _expose(field);

// xfer buffer length is last n-bytes in eventData
int xferDataSize = 0;
if (!int.TryParse(urb["fid_URB_TransferBufferLength"], out xferDataSize))
    return 0;

// usb keyboard xfer data is 8 bytes
if (xferDataSize != 8)
    return 0;
```



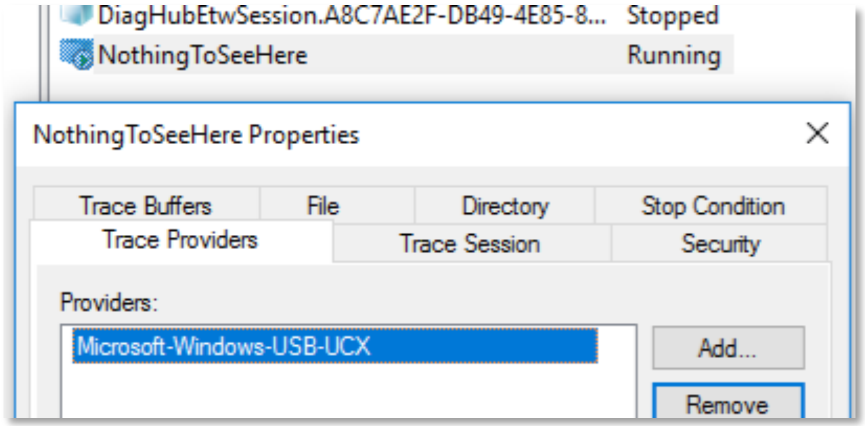
hndl	18446708889607743568
urbRaw	{{ fid_URB_Hdr_Length=128
separators	{char[3]}
s	{string[12]}
urb	Count = 12
xferDataSize	8
xferData	{byte[8]}
[0]	0
[1]	0
[2]	4
[3]	0
[4]	0
[5]	0
[6]	0
[7]	0

SHOW ME THE KEYS!

(A DEMO)

Detecting ETW USB Attacks

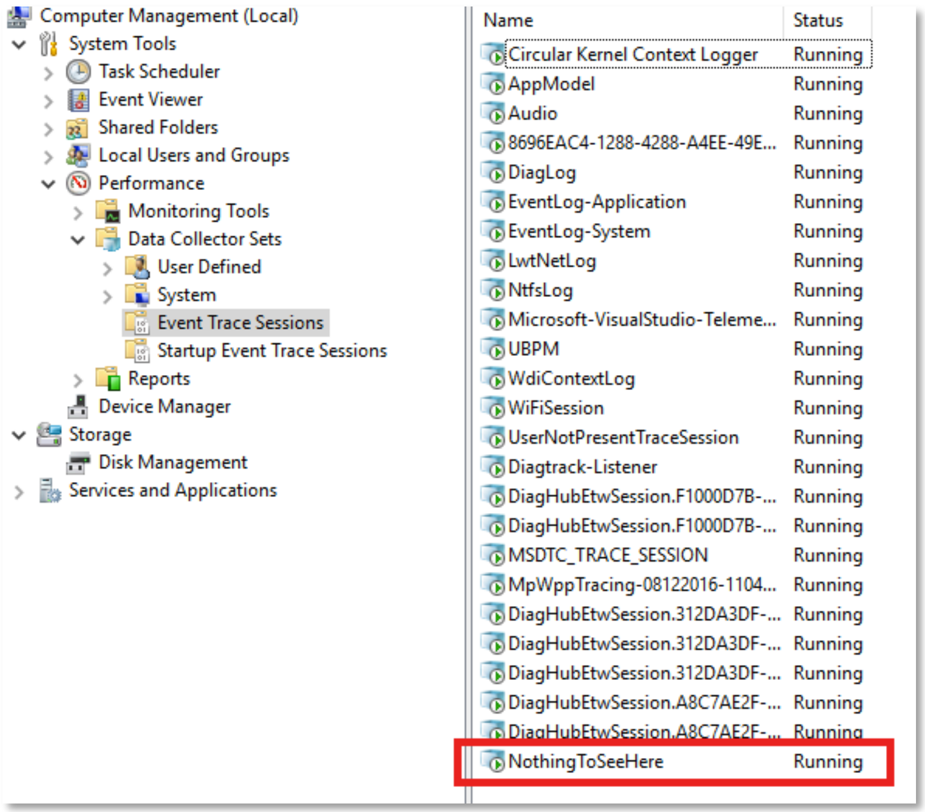
- Monitor for use
  - Microsoft-Windows-USB-UCX (USB 3)
  - Microsoft-Windows-USB-USBPORT (USB 2)
  - Potential False Positives?
- Suspicious ETW sessions
  - No baseline of "trusted sessions"
- Sessions can be overwritten!
  - Everything but Real-time sessions
  - Stops previous session. Not restarted



DiagHubEtwSession.A8C7AE2F-DB49-4E85-8...	Stopped
NothingToSeeHere	Running

NothingToSeeHere Properties

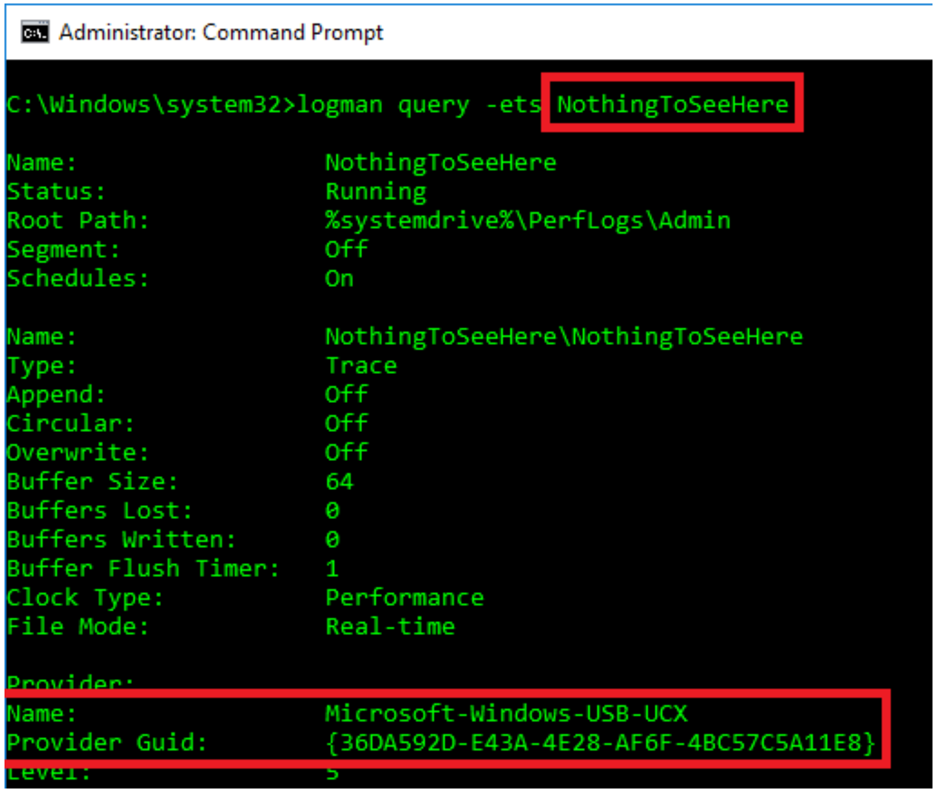
Providers:
Microsoft-Windows-USB-UCX



Name	Status
Circular Kernel Context Logger	Running
AppModel	Running
Audio	Running
8696EAC4-1288-4288-A4EE-49E...	Running
DiagLog	Running
EventLog-Application	Running
EventLog-System	Running
LwtNetLog	Running
NtfsLog	Running
Microsoft-VisualStudio-Teleme...	Running
UBPM	Running
WdiContextLog	Running
WiFiSession	Running
UserNotPresentTraceSession	Running
Diagtrack-Listener	Running
DiagHubEtwSession.F1000D7B-...	Running
DiagHubEtwSession.F1000D7B-...	Running
MSDTC_TRACE_SESSION	Running
MpWppTracing-08122016-1104...	Running
DiagHubEtwSession.312DA3DF-...	Running
DiagHubEtwSession.312DA3DF-...	Running
DiagHubEtwSession.312DA3DF-...	Running
DiagHubEtwSession.A8C7AE2F-...	Running
DiagHubEtwSession.A8C7AE2F-...	Running
NothingToSeeHere	Running

Detecting ETW USB Attacks *(cont.)*

- Logman is your friend!
  - List all details for a session



```
C:\Windows\system32>logman query -ets NothingToSeeHere

Name:                     NothingToSeeHere
Status:                   Running
Root Path:                %systemdrive%\PerfLogs\Admin
Segment:                  Off
Schedules:                On

Name:                     NothingToSeeHere\NothingToSeeHere
Type:                     Trace
Append:                   Off
Circular:                 Off
Overwrite:                Off
Buffer Size:              64
Buffers Lost:             0
Buffers Written:          0
Buffer Flush Timer:       1
Clock Type:               Performance
File Mode:                Real-time

Provider:
Name:                     Microsoft-Windows-USB-UCX
Provider Guid:            {36DA592D-E43A-4E28-AF6F-4BC57C5A11E8}
Level:                    5
```

ETW USB Keylogger Limitations

- USB…
  - No laptop support (PS/2)
  - Windows 11?!
  - Kidding, but who knows?
- Windows 7+
  - Windows 7: USB 2 only
  - USB 3 Provider (UCX) not introduced until Windows 8
- Requires admin (UAC)
- Performance Issues?
  - “Real-time” filtering and capturing can drop events
  - Haven’t seen this occur in our *(limited)* testing

IE Info Leak

- Microsoft-Windows-WinINet
  - All data that passes through the WinINet library
    - HTTP and HTTPS
- No need to inject into browser process
- Works even when site uses HTTPS
- Most private information exposed
  - URLs visited *(recon)*
  - Cookies *(session hijacking)*
  - POST parameters *(credential stealing)*
- Works on IE, Edge, many Windows 10 Apps, and any program using WinINet for HTTP requests
- Similar technique using logman/wevtutil
  - http://securityweekly.com/2012/07/18/post-exploitation-recon-with-e/
  - Requires writing to disk and parsing in separate steps

Windows 10 Store Application Leaks

- Full leaks
  - Plain-text password logged to ETW
- Partial leaks
  - OAuth 2.0 or hashing/encrypting password
  - Allows for hijack session cookies/headers
- Affected Applications
  - Most ☹
  - Categories
    - Entertainment
    - Financial institutions
    - Windows Store and other built-in apps
    - Social media
    - Email Providers
    - E-Retailers
    - More….
- No leaks

PayPal

! Out of 15 tested Applications:
4 Full Leaks
9 Partial Leaks
2 No Leaks

Microsoft-Windows-WinINet

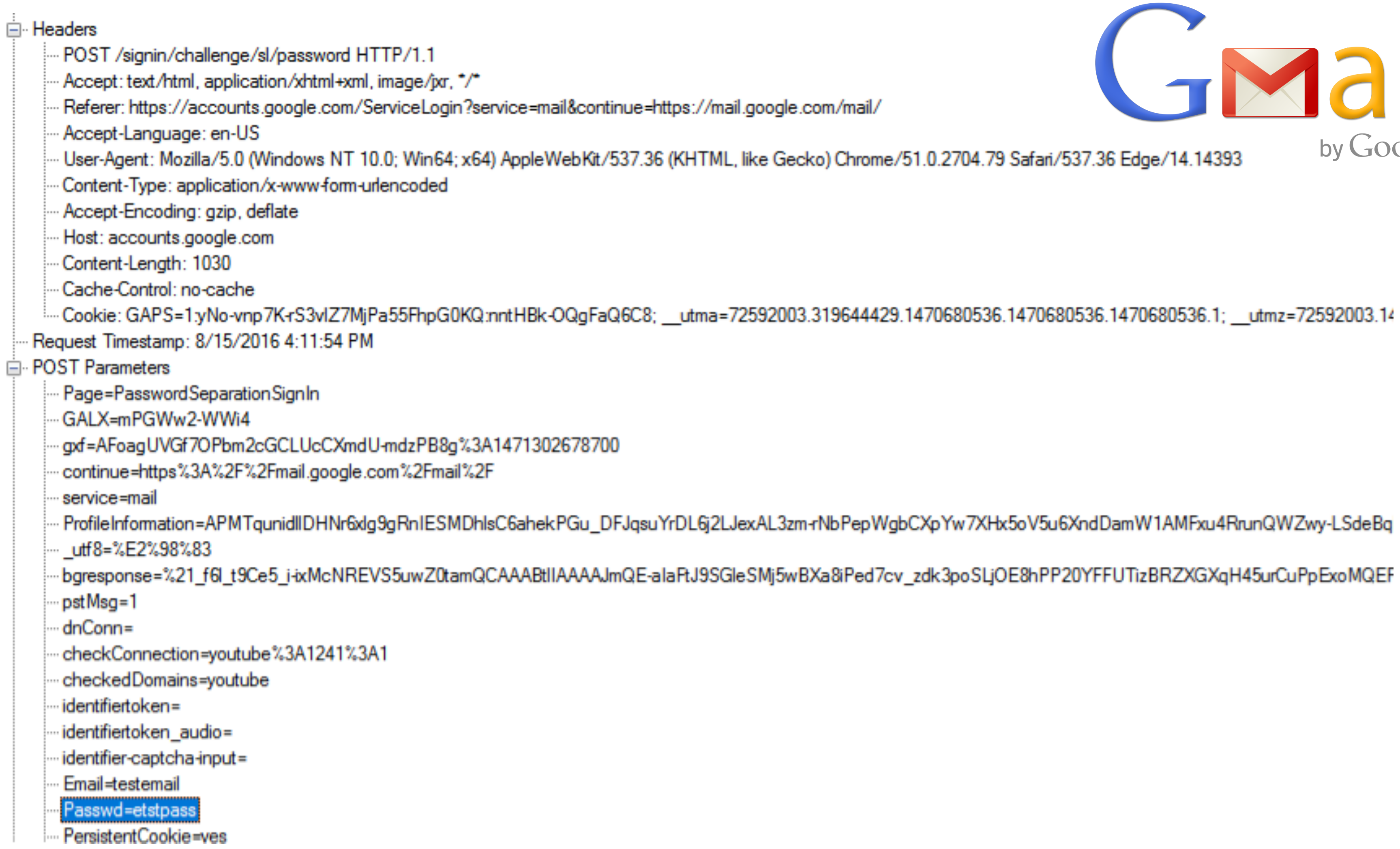
Event types *(available as keywords for filtering, i.e. WININET_KEYWORD_HANDLES)*

- Handle Events – creation and destruction of HINTERNET handles
- HTTP Events – processing of HTTP requests and responses
- Connection Events – underlying network operations *(TCP, DNS)*
- Authentication Events
- HTTPS Events
- Autoproxy Events
- Cookie Events
- WININET_KEYWORD_PII_PRESENT – keyword for events of multiple types potentially containing personally identifiable information

Useful event names

- WININET_COOKIE_STORED, Wininet_UsageLogRequest, WININET_HTTP_REQUEST_HANDLE_CREATED, WININET_REQUEST_HEADER, WININET_REQUEST_HEADER_OPTIONAL, WININET_RESPONSE_HEADER

Logging in to Gmail



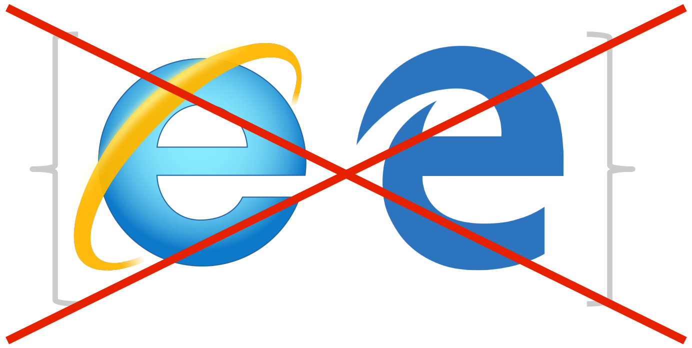
```
Headers
    POST /signin/challenge/sl/password HTTP/1.1
    Accept: text/html, application/xhtml+xml, image/jxr, */*
    Referer: https://accounts.google.com/ServiceLogin?service=mail&continue=https://mail.google.com/mail/
    Accept-Language: en-US
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.79 Safari/537.36 Edge/14.14393
    Content-Type: application/x-www-form-urlencoded
    Accept-Encoding: gzip, deflate
    Host: accounts.google.com
    Content-Length: 1030
    Cache-Control: no-cache
    Cookie: GAPS=1:yNo-vnp7K-rS3vlZ7MjPa55FhpG0KQ:nntHBk-OQgFaQ6C8; __utma=72592003.319644429.1470680536.1470680536.1470680536.1; __utmz=72592003.14
Request Timestamp: 8/15/2016 4:11:54 PM
POST Parameters
    Page=PasswordSeparationSignIn
    GALX=mPGWw2-WWi4
    gxf=AFoagUVGf7OPbm2cGCLUcCXmdU-mdzPB8g%3A1471302678700
    continue=https%3A%2F%2Fmail.google.com%2Fmail%2F
    service=mail
    ProfileInformation=APMTqunidlIDHNr6xlg9gRnIESMDhlsC6ahekPGu_DFJqsuYrDL6j2LJexAL3zm-rNbPepWgbCXpYw7XHx5oV5u6XndDamW1AMFxu4RrunQWZwy-LSdeBq
    _utf8=%E2%98%83
    bgresponse=%21_f6l_t9Ce5_i-ixMcNREVS5uwZ0tamQCAAABtlIAAAAJmQE-alaFtJ9SGleSMj5wBXa8iPed7cv_zdk3poSLjOE8hPP20YFFUTizBRZXGXqH45urCuPpExoMQEF
    pstMsg=1
    dnConn=
    checkConnection=youtube%3A1241%3A1
    checkedDomains=youtube
    identifiertoken=
    identifiertoken_audio=
    identifier-captcha-input=
    Email=testemail
    Passwd=etstpass
    PersistentCookie=yes
```

SET COURSE FOR THE DEMO.

ENGAGE.

Mitigation (a.k.a. good advice)

- Don’t use IE or Edge
  - Use Chrome, Tor, etc.
- Use a standard *(non-admin)* user account
  - Leave UAC Enabled
  - ETW requires admin
- Only run trusted applications as admin
- Monitor for sessions with WinINet provider enabled



When using message tracing feature, messages carrying sensitive information such as credentials, personal information, etc. may be persisted to the disk or be viewed by anyone who has access to the system event viewer. As a mitigation to this issue, tracing can be enabled by System or Administrator users on Windows 2003 and later. ~ *MSDN*

Thanks for coming!

Special thanks to

- Ruxcon
- Chris Spencer
- Stan Chua
- John Eiben
- Mark McLarnon
- Andre Protas

Questions?

CyberPoint
Security
Research
Team

Blog
www.cyberpointllc.com/srt

Email
SRT@cyberpointllc.com

Twitter
@CyberPoint_SRT

Code From our Demos/Research
github.com/CyberPoint/Ruxcon2016ETW

Thanks for coming!

References

MSDN Event Tracing

- https://msdn.microsoft.com/en-us/library/windows/desktop/bb968803(v=vs.85).aspx

USB Device Class Definition for Human Interface Devices (HID)

- http://www.usb.org/developers/hidpage/Hut1_12v2.pd

USB traces with Microsoft Message Analyzer

- https://msdn.microsoft.com/en-us/library/windows/hardware/dn741264(v=vs.85).aspx

Viewing/capturing USB data

- http://www.usblyzer.com/
- https://www.microsoft.com/en-us/download/details.aspx?id=44226

USB/URB

- http://www.beyondlogic.org/usbnutshell/usb5.shtml
- https://msdn.microsoft.com/en-us/library/windows/hardware/ff538930(v=vs.85).aspx

Ransomware samples

- https://www.virustotal.com/
- https://cyberpointllc.com/products/darkpoint/index.html

Xperf Basics: Recording a Trace *(the easy way)*

- https://randomascii.wordpress.com/2013/04/20/xperf-basics-recording-a-trace-the-easy-way/

SSL Side Jacking

- http://wiki.securityweekly.com/wiki/index.php/Episode300