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 is a cyber security company.

We’re in the business of protecting what’s invaluable to you.
Who We Are

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

Source:
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
ETW Example Providers

• Listing providers

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

Provider                  GUID
------------------------- -----------------------------------------------
ACPI Driver Trace Provider {DA60D4D-2D48-477D-B1C3-DAAD0CE6F06B}
Active Directory Domain Services: SAM {8E598056-8993-11D2-819E-0000F875A064}
Active Directory: Kerberos Client {BB83ADD2-C229-4C0D-AE2B-578696860C4}
Active Directory: NetLogon {F34599B4-DBEC-11D2-895B-00C04F79AB69}
ADODB.1                   {04C898E6-3369-12F8-4769-24E484A4E725}
ADOMD.1                   {7EA56435-3F2F-3F63-A829-F003585C04D1}
Application Popup         {47BFA2B7-BD5A-4FAC-870B-29B21084CA8F}
Application-AddOn-Event-Provider {A83FA99F-C356-4DDE-9FDE-5A5EBB546D68}
ATA Port Driver Tracing Provider {D0686885-501E-489A-BAC6-7D2B4EBEBF8}
AuthFw NetShell Plugin    {953F4AE6-84D0-41C6-97FA-3800A04A2B72}
```

• Listing running sessions

```powershell
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
IwltNetlog               Trace Running
NtfsLog                  Trace Running
UBPM                     Trace Running
WdiContextLog            Trace Running
WniSession               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
TraceEvent object

TONS of information!

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td><code>&lt;Event MSec=&quot;1207.9474&quot; PID=&quot;5432&quot; PName=&quot;&quot; Name=&quot;&quot;</code></td>
<td>Microsoft.Diagnostics.Tracing.TraceEvent (Microsoft.Diagnostics.Tracing.TraceEvent)</td>
</tr>
<tr>
<td>ActivityID</td>
<td>(0x00c00004-0007-0000-3c150-edf6b06)</td>
<td>System.Guid</td>
</tr>
<tr>
<td>Channel</td>
<td>16</td>
<td>Microsoft.Diagnostics.Tracing.TraceEventChannel</td>
</tr>
<tr>
<td>DataStart</td>
<td>(85065880)</td>
<td>System.IntPtr</td>
</tr>
<tr>
<td>EventDataLength</td>
<td>212</td>
<td>int</td>
</tr>
<tr>
<td>EventIndex</td>
<td>1</td>
<td>Microsoft.Diagnostics.Tracing.TraceEventIndex</td>
</tr>
<tr>
<td>EventName</td>
<td>&quot;WININET_ROOT_HANDLE_CREATED&quot;</td>
<td>string</td>
</tr>
<tr>
<td>EventTimeUserDate</td>
<td>null</td>
<td>object</td>
</tr>
<tr>
<td>FormattedMessage</td>
<td>&quot;Session handle 0xc0004 created. UserAgent=Mozi&quot;</td>
<td>string</td>
</tr>
<tr>
<td>ID</td>
<td>101</td>
<td>Microsoft.Diagnostics.Tracing.TraceEventID</td>
</tr>
<tr>
<td>IsClassicProvider</td>
<td>false</td>
<td>bool</td>
</tr>
<tr>
<td>Keywords</td>
<td>0922372036854775807</td>
<td>Microsoft.Diagnostics.Tracing.TraceEventKeyword</td>
</tr>
<tr>
<td>Level</td>
<td>Information</td>
<td>Microsoft.Diagnostics.Tracing.TraceEventLevel</td>
</tr>
<tr>
<td>Opcode</td>
<td>Info</td>
<td>Microsoft.Diagnostics.Tracing.TraceEventOpcode</td>
</tr>
<tr>
<td>OpCodeName</td>
<td>&quot;Info&quot;</td>
<td>string</td>
</tr>
<tr>
<td>PayloadNames</td>
<td>{string[6]}</td>
<td>string</td>
</tr>
<tr>
<td>PointerSize</td>
<td>8</td>
<td>string</td>
</tr>
<tr>
<td>ProcessID</td>
<td>5432</td>
<td>int</td>
</tr>
<tr>
<td>ProcessName</td>
<td>&quot;&quot;</td>
<td>string</td>
</tr>
<tr>
<td>ProcessorNumber</td>
<td>3</td>
<td>int</td>
</tr>
<tr>
<td>ProviderID</td>
<td>(43d1a55c-76d6-417e-995c-64c711e5cafe)</td>
<td>System.Guid</td>
</tr>
<tr>
<td>ProviderName</td>
<td>&quot;Microsoft-Windows-WinInet&quot;</td>
<td>string</td>
</tr>
<tr>
<td>RelatedActivityID</td>
<td>(00000000-0000-0000-0000-000000000000)</td>
<td>System.Guid</td>
</tr>
<tr>
<td>Target</td>
<td>(Method = (Void &lt;setupSource&gt;b_17_0)(Microsoft.Diagnostics.Trace...)</td>
<td>Microsoft.Diagnostics.Tracing.TraceEventSource</td>
</tr>
<tr>
<td>Task</td>
<td>500</td>
<td>Microsoft.Diagnostics.Tracing.TraceEventTask</td>
</tr>
<tr>
<td>TaskName</td>
<td>&quot;WININET_ROOT_HANDLE_CREATED&quot;</td>
<td>string</td>
</tr>
<tr>
<td>ThreadID</td>
<td>5436</td>
<td>int</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>(9/15/2016 3:13:25 PM)</td>
<td>System.DateTime</td>
</tr>
<tr>
<td>TimeStampRelativeMSec</td>
<td>1207.9474</td>
<td>double</td>
</tr>
<tr>
<td>Version</td>
<td>0</td>
<td>int</td>
</tr>
</tbody>
</table>
Using **ETW API (C#)**

Example Simple UAC Event Listener
- Extremely easy to implement

```csharp
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, TCP/IP, MANY more...
    - Some examples/tools exist but can be improved

HMM...
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
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 \textit{(in some form)}
    - Almost always

- Encryption
  - Too much variance for generic rule
Detected 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.

<table>
<thead>
<tr>
<th>PID</th>
<th>Time</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Suspicious!**
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
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](https://www.virustotal.com/vt api/v2/file/0f60e3d494fb696556b554b63a57f7e47184a6f805c4d2f9c27e8f2721b832de8eb/desc)

<table>
<thead>
<tr>
<th>SHA256:</th>
<th>0f60e3d494fb696556b554b63a57f7e47184a6f805c4d2f9c27e8f2721b832de8eb</th>
</tr>
</thead>
<tbody>
<tr>
<td>File name:</td>
<td>boost-serialization</td>
</tr>
<tr>
<td>Detection ratio:</td>
<td>26 / 56</td>
</tr>
<tr>
<td>Analysis date:</td>
<td>2016-05-06 10:40:54 UTC (4 months, 1 week ago)</td>
</tr>
</tbody>
</table>
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?

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

<table>
<thead>
<tr>
<th>Data Collector Set</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppModel</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>Audio</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>DiagHubEtwSession.C9A793F6-6275-4143-8F0F-1B70A9293E1.1.UserpagedMemory Trace</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>DiagHubEtwSession.54779E8E-94D1-4E14-8434-7787D56067A.1.UserpagedMemory Trace</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>FamilySafety98T</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>LwtNetLog</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>NtisLog</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>NtFisSession</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>DiagHubEtwSession.77552A06-6AE6-4C66-8916-070C893EDDD9.1.UserpagedMemory Trace</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>MsvppTracing-09012016-170353-00000003-ffffff00 Trace</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>Diagtrack-Listener</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>DiagHubEtwSession.77552A06-6AE6-4C66-8916-070C893EDDD9.1.System Trace</td>
<td>Trace</td>
<td>Running</td>
</tr>
<tr>
<td>DiagHubEtwSession.6C9E93E-3ADB-1D51-D07C-15C58E1A7BA.1.UserpagedMemory Trace</td>
<td>Trace</td>
<td>Running</td>
</tr>
</tbody>
</table>

The command completed successfully.
**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...

![Image](image1.png)

- To this!

![Image](image2.png)

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...

![Image](image.png)
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

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

```c
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 perror`
- `UCX_ETW_EVENT_COMPLETE_URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER`
- `fid_UcxController: 0x0000000d43CAD08`
- `fid_UsbDevice: 0x00000004A287368`
- `fid_PipeHandle: 0x00000000B51D1D20`
- `fid_IRP_Ptr: 0x00000005193940`
- `fid_IRB_Ptr: 0x00000000C8331860`
- `fid_UCX_URB_BULK_OR_INTERRUPT_TRANSFER: success, Function = Success`
- `fid_IRP_NtStatus: Success`
- `fid_URB_TransferDataLength: 8 (0x8)`
- `fid_IRB_TransferData: 0 (0x0)`
- `fid_IRB_TransferData: 0 (0x0)`
- `fid_IRB_TransferData: 4 (0x4)`
- `fid_IRB_TransferData: 0 (0x0)`
- `fid_IRB_TransferData: 0 (0x0)`
- `fid_IRB_TransferData: 0 (0x0)`
- `fid_IRB_TransferData: 0 (0x0)`

Table 12: Keyboard/Keypad Page

<table>
<thead>
<tr>
<th>Usage ID (Dec)</th>
<th>Usage ID (Hex)</th>
<th>Usage Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Reserved (no event indicated)9</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Keyboard ErrorRollOver9</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Keyboard POSTFail9</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>Keyboard ErrorUndefined9</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>Keyboard a and A4</td>
</tr>
</tbody>
</table>
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

```csharp
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;
```
SHOW ME THE KEYS!

(A DEMO)
Detecting ETW USB Attacks

• Monitor for use
  – Microsoft-Windows-USB-UCX (USB 3)
  – Microsoft-Windows-USB-USPSPORT (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
Detecting ETW USB Attacks (cont.)

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

```
\[\text{C:}\text{\textbackslash Windows}\text{\textbackslash system32}\text{\textgreater logman query -ets \texttt{NothingToSeeHere}}\]

\begin{verbatim}
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
\end{verbatim}
```
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
  - 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**

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

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: GAPv=1:yNo-vnp7K6i9z7MvMa55FhpG0KVc:nmtHBk-oOgFaQ6C8; __utma=72592003.319644429.1470680536.1470680536.1.72592003.14; __utmz=72592003.1470680536.1.1.utmcsr=direct|utmccn=(referral)|utmcmd=(unknown)
Request Timestamp: 8/15/2016 4:11:54 PM

POST Parameters
Page=PasswordSeparationSignin
GALX=mgPWw2-WWw4
gd=AFoagUVGF70Pbm2cGCLUCXmdU-mdzPB8g%3A1471302678700
continue=https%3A%2F%2Fmail.google.com%2Fmail%2F
service=mail
ProfileInformation=APMTqunidI7DHn6g9gRnI6SMDhIsC6ahekPGu_DFJqasuYrDL6j2LJexAL3zm+nBpPepWgbCXpY7XHx5oV5u5xNdxnDmmW1AMFxtu4RrunQWZwy-LSdeBq
_utf8=%E2%80%83
bgresponse=%21_f6I_t9Ce5_i+xMcNREVSSuwZDtmQCAAABtIAAAAJmQE-alRtJ9SGLcSMj5wMXB&Ped7cv_zdk3poSLjOE8hPP20YFFUItzBRZXGxqH45urCuPpExoMQEF
pstMsg=1
checkConnection=youtube%3A1241%3A1
checkedDomains=youtube
identifiedtoken=
identifiedtoken_audio=
identifier-captcha-input=
Email=testemail
Passwd=etspass
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 \emph{(non-admin)} user account
  - Leave UAC Enabled
  - ETW requires admin
- Only run trusted applications as admin
- Monitor for sessions with WinInNet provider enabled

\begin{itemize}
  \item 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. \textit{~MSDN}
\end{itemize}
Thanks for coming!

Special thanks to

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

Make ETW Great Again – Ruxcon 2016
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

USB Device Class Definition for Human Interface Devices (HID)

USB traces with Microsoft Message Analyzer

Viewing/capturing USB data
- http://www.usblyzer.com/

USB/URB
- http://www.beyondlogic.org/usbnutshell/usb5.shtml

Ransomware samples
- https://www.virustotal.com/

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