

# Logging Keystrokes with Event Tracing for Windows (ETW)

 [cyberpointllc.com/blog-posts/cp-logging-keystrokes-with-event-tracing-for-windows-etw.php](https://cyberpointllc.com/blog-posts/cp-logging-keystrokes-with-event-tracing-for-windows-etw.php)

---

As a follow-up to our talk at [Ruxcon](#), "Make ETW Great Again", we wanted to go into a bit more depth than we could cover in our hour long talk. While our talk consisted of multiple examples of ETW usage, detecting ransomware, USB Keylogging, and sniffing SSL encrypted data from WinINet (our code can be found here:

<https://github.com/CyberPoint/Ruxcon2016ETW>), we wanted to specifically discuss USB Keylogging here. Given the nature and potential impact of our findings, we decided it warranted more explanation especially with regards to mitigation and detection of this technique.

## What is ETW?

---

Event Tracing for Windows is an asynchronous kernel debugging mechanism built into all modern versions of Windows that is typically used to assist administrators and developers troubleshoot and measure system and application performance. It's built-in and enabled by default on Windows 2000, but did not truly become feature-rich until Windows 7.

ETW is thoroughly documented by Microsoft and could easily occupy multiple blog posts. So if you're new to ETW check out MSDN for an introduction<sup>i</sup> or a couple of GitHub repos we used during our research<sup>ii</sup> <sup>iii</sup>. Otherwise, from this point on, we'll assume a basic understanding of ETW and how it works is already known.

## A New Method for Keylogging

---

During our analysis of ETW and its many providers (over 1000 on Windows 10), we encountered a few related to USB. While much of it was tailored towards debugging driver and bus issues, two providers stood out, specifically:

- **Microsoft-Windows-USB-UCX** (36DA592D-E43A-4E28-AF6F-4BC57C5A11E8)
- **Microsoft-Windows-USB-USBPORT** (C88A4EF5-D048-4013-9408-E04B7DB2814A)

ETW and the USB protocol are extremely verbose but searching through the data generated by these providers, we eventually noticed USB transfer data that looked eerily like keyboard and mouse input.

After confirming that keystrokes were indeed present through ETW, we scoured the Internet for malware samples to verify if this method of keylogging had ever been used out in the wild. Of course, simply because we didn't see any existence of samples using this doesn't

mean that samples don't exist We also expected, given how simple it was to retrieve the USB HID transfer data from ETW, that we would see existing tools or at least discussions of this technique in the public (or at least semi-public) domain. With the exception of a few blog posts mentioning ETW for debugging USB problems, we couldn't find any other discussions or tools where ETW is leveraged for the explicit purpose of logging keystrokes. **This led us to believe that we have found a novel method for key logging that's built-in to the Windows operating system.**

An example of our POC leveraging ETW for capturing key strokes can be in the image below. Because this is a POC and not a tool ready for integration with Metasploit or Powershell Empire (but hopefully will be soon), we are verbose in our logging. In our POC, for each key pressed, we're capturing not only its ASCII representation but you will also see the raw bytes captured from the USB device that represent the Human Interface Device (HID) payload data (as well as a timestamp). Part of the implementing a key logger from raw USB data requires mapping data from this 8 byte payload via the [HID specifications](#) (one per line in the next image).

```
starting capture ...
20161017 12:28:43.385 00 00 0B 00 00 00 00 00 h
20161017 12:28:48.609 00 00 08 00 00 00 00 00 e
20161017 12:28:50.161 00 00 0F 00 00 00 00 00 l
20161017 12:28:50.505 00 00 0F 00 00 00 00 00 l
20161017 12:28:51.025 00 00 12 00 00 00 00 00 o
20161017 12:28:53.073 00 00 2C 00 00 00 00 00 [SPACE]
20161017 12:28:56.218 00 00 1A 00 00 00 00 00 w
20161017 12:28:56.769 00 00 12 00 00 00 00 00 o
20161017 12:28:56.993 00 00 15 00 00 00 00 00 r
20161017 12:28:57.209 00 00 0F 00 00 00 00 00 l
20161017 12:28:57.442 00 00 07 00 00 00 00 00 d
```

Figure 1: Running our POC keylogger to capture key strokes "hello world".

Since parsing HID data and mapping it to its ASCII corresponding representation of a keystroke is a fairly well defined process we won't cover here it but we will cover, in our next blog post, the low level details of how to obtain this data (HID payload data) via ETW in our next blog post.

## Why This Matters

For obvious reasons, keylogging is capability that antivirus vendors keep very close watch over. With this type of capability in particular, there is a fairly short list of ways in which keystrokes can be captured on Windows. The existence of a new and unknown method is a

valid concern as it makes detection less likely by antivirus vendors, malware analysts and blue teamers. In fact, at the time of this writing, our POC had a detection rate of 0 on VirusTotal as well as being undetected in run-time tests against a few AV products.



Figure 2: The keylogger POC was not detected by VirusTotal.

## More Than Just a Keylogger

By leveraging ETW for USB data a user can access raw USB HID data. Normally data from a HID compliant USB device (or any USB device) must be first processed by the USB stack before it's of any use to the operating system, an overview of which is provided below. However, by choosing specific USB related ETW providers an attacker can get a copy of the same RAW (unprocessed) USB data. Though the difference may seem subtle, it's important to note that the copy of this RAW USB data is no longer subject to the same policies and mechanisms of the normal, parsed USB data in the operating systems. This means that an attacker now can access USB data such as keystrokes from a keyboard without having to worry about normal security mechanisms or anti-virus solutions.

This is especially concerning as Microsoft has recently made changes to Windows specifically to minimize the attack surface for well-known credential and hash stealing techniques that are completely circumvented with ETW keylogging. Two changes in particular, Virtual Secure Mode and Credential Guard were introduced in Windows 10 with the explicit purpose of mitigating credential stealing via Pass-the-Hash and memory dumping attacks for domain credentials:

<https://www.blackhat.com/docs/us-15/materials/us-15-Moore-Defeating%20Pass-the-Hash-Separation-Of-Powers-wp.pdf>

These mitigations greatly increase the difficulty of stealing credentials or NTLM hashes by virtualizing and adding extra protections to Windows credential handing and caching mechanisms. These new mechanisms, while effectively against current credential stealing tools like Mimikatz are, unfortunately, completely ineffective against USB keylogging via ETW.

Lastly, it should also be noted this method for logging keystrokes is a viable attack vector on multiuser environments. Using ETW would allow an attacker, which has gained elevated access to the system, to capture clear-text credentials for any other user that physically logs into the system.

## Detecting ETW USB Keyloggers

---

The ETW interface requires that a controller (in this case, our keylogger application) creates a session with event providers, from which it can consume data. This means that any process that wishes to ingest ETW event data must first create a session. These are easy to detect (as shown above) but it is not sufficient to simply look for strange session names, as there really isn't a trusted list of known ETW sessions. Also, there is no way to predict what legitimate software may have ETW sessions established for either debugging or logging purposes.

The ideal method for detection would involve searching for suspicious providers and their corresponding GUIDS:

- **Microsoft-Windows-USB-UCX** (36DA592D-E43A-4E28-AF6F-4BC57C5A11E8)
- **Microsoft\_Windows-USB-USBPORT** (C88A4EF5-D048-4013-9408-E04B7DB2814A)

One shouldn't really expect these providers to be running in a production system; however, that is not to say that some applications may have legitimate uses for debugging purposes so detection strategies based on provider names/GUIDs alone are prone to a possible false positive.

Searching for malicious providers can be accomplished a number of ways. The UI method, through the Computer Management Console, is a decent way for investigating or troubleshooting a single machine, but can also be accomplished with the command line utility, logman.exe. With the "ets" switch and the session name it also allows the user to view the providers for an existing session. For example, we can detect our POC keylogger in the above example with the following:

```

C:\Windows\system32>logman -ets ntfslog

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

Name:                NtfsLog\NtfsLog
Type:                Trace
Append:              Off
Circular:            Off
Overwrite:           Off
Buffer Size:         64
Buffers Lost:        0
Buffers Written:     4
Buffer Flush Timer:  1
Clock Type:          Performance
File Mode:           Real-time

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

```

Figure 3: Detecting keylogging in a benign ETW process. The keylogging provider is outlined in red.

Additionally, detecting keylogging via ETW could also be accomplished without continually iterating sessions through programmatic introspection or function hooking. As can be seen from our example code any application that gains access to ETW via the .NET TraceEvent interface, provided by Microsoft, must do so through the **EnableProvider()** function:

```

var session = new TraceEventSession("keylogger");
session.EnableProvider("36DA592D-E43A-4E28-AF6F-4BC57C5A11E8");
session.EnableProvider("C88A4EF5-D048-4013-9408-E04B7DB2814A");

```

This code uses the EnableProvider function to enable the required providers (**Microsoft-Windows-USB-UCX** and **Microsoft-Windows-USB-USBPORT**) by GUID. In C#, providers can be enabled by either a GUID or ASCII name so any detection strategy must support both formats for the provider. After calling these functions with either the GUID or the provider name, an attacker would then be able to obtain all the USB data for the given machine. It should also be noted that equivalent functions exist in the Win32 API (**EnableTraceEx** and **EnableTraceEx2**) which could also be hooked or monitored to detect key logging from a native Windows application.

## Mitigation

Though there doesn't seem to be a way to permanently disable this technique from an attacker as ETW is relied upon by the operating system for things like Event Logs, the steps below should lower the likelihood of an attacker successfully leveraging this technique in a real-work scenario.

- Restrict administrator account usage
- Always keep UAC enabled
- Avoid granting administrative privileges to trusted users/applications if possible.

## Impact Summary

---

Below is a summary of important information for administrators and blue team members to aid in understanding and properly defending against this technique.

- Cons
  - This technique is currently not detected. At the time of this writing there was no known detection(s) on VirusTotal or in limited run-time testing we conducted.
  - ETW is intended behavior, included and documented by Microsoft and enabled by default. This will make generic detection of this technique more difficult for vendors.
  - Unlike many other credential stealing techniques this method does not need to inject into well-known processes such as winlogon.exe or lsass.exe to capture credentials/hashes.
- Pros
  - This technique only works with USB devices. This may seem obvious but it is important to note that laptop keyboards can't be logged using this method, as they typically use PS/2.
  - Requires administrator privileges. A UAC bypass will typically be required for ETW access.
  - Will not work on versions older than Windows 7. The USB 3.0 ETW provider, Microsoft-Windows-USB-UCX, is not included in Windows 7. This means that only keyboards in USB 2.0 ports will work on Windows 7. For Windows 8 and later both providers, Microsoft-Windows-USB-UCX (USB3.0) and USB-USBPORT (USB2.0) are included, allowing an attacker to capture keystrokes on any USB port regardless of version.

— <sup>i</sup>About Event Tracing – [https://msdn.microsoft.com/en-us/library/windows/desktop/aa363668\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa363668(v=vs.85).aspx)

— <sup>ii</sup>Perfview – <https://github.com/Microsoft/perfview>

— <sup>iii</sup>ETW with .NET – <https://github.com/Microsoft/dotnetsamples/tree/master/Microsoft.Diagnostics.Tracing>

