It's Duck Season, Hunting Bad-USB Attacks

HELLO!

I am George Karantzas

- Working for Bitdefender as a Researcher
- Studying at the University of Piraeus

Bad-USB threat model.

- Every unattended computer is vulnerable to an attempt.
- Endpoint defenses may or may not interfere.
- Daily situations such as going at the building's cafeteria.
- Tailgating increases risk.
- As easy as plugging a USB.
- Scenario studies credential dumping and exfiltration.

Intro & Goals

- Everything started at Defcon's Stores.
  - Bought the manual and noticed the information about speed.
  - Thought how I could weaponize this.
- Focuses on Bad-USB attacks for keystroke injection.
- Focuses on behavior-based detection rather than hardcoded information.

Intro & Goals - Part 2

- Focus on a trajectory-like combinational logic to verify malicious activity.
  - Minimum cognitive ability required.
- Post-mortem approach.
  - At least for the scope of this research.
- Logs from deep system events.
- ETW & Keyboard upper filter-based logs.

Rubber Ducky

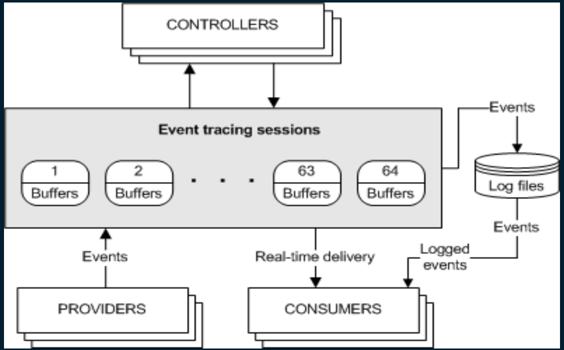
- Representative of the current commercial Bad-USB scene.
- USB2 interface & USBC support.
- Started as a Fast-Typer to automate boring tasks.
- Various features such as:
  - Keystroke Reflection
  - Exfiltration
  - Storage Unit
  - PID & VID spoofing.

Telemetry Source - ETW

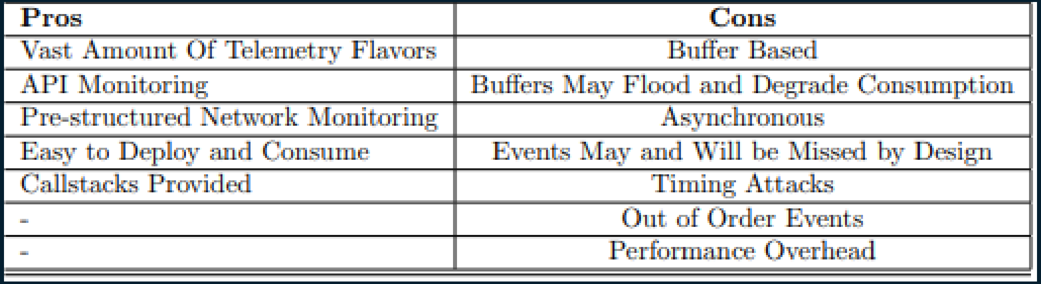
ETW is an asynchronous, buffer-based system allowing event creation , routing and consumption created by Microsoft with its core infrastructure residing in the Windows Kernel allowing for file and in-memory consumption.

ETW Sources:

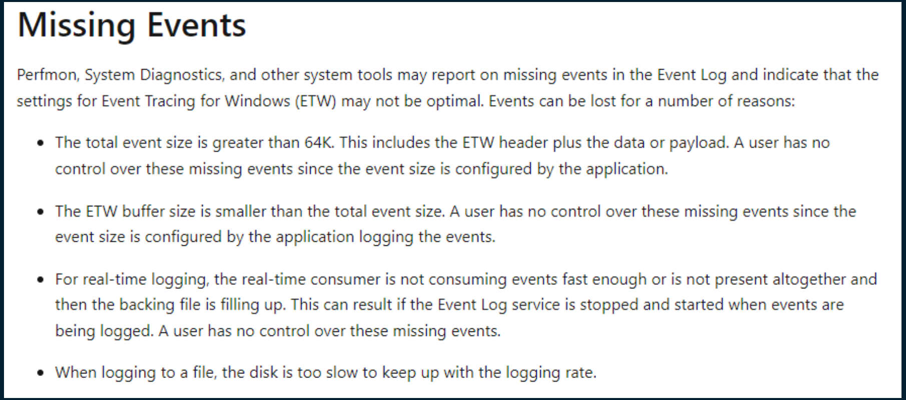
- Drivers
- Kernel Components
- UM Apps and DLLs
- Network Stack



ETW Pros & Cons



Pros	Cons
Vast Amount Of Telemetry Flavors	Buffer Based
API Monitoring	Buffers May Flood and Degrade Consumption
Pre-structured Network Monitoring	Asynchronous
Easy to Deploy and Consume	Events May and Will be Missed by Design
Callstacks Provided	Timing Attacks
-	Out of Order Events
-	Performance Overhead



Missing Events

Perfmon, System Diagnostics, and other system tools may report on missing events in the Event Log and indicate that the settings for Event Tracing for Windows (ETW) may not be optimal. Events can be lost for a number of reasons:

- The total event size is greater than 64K. This includes the ETW header plus the data or payload. A user has no control over these missing events since the event size is configured by the application.
- The ETW buffer size is smaller than the total event size. A user has no control over these missing events since the event size is configured by the application logging the events.
- For real-time logging, the real-time consumer is not consuming events fast enough or is not present altogether and then the backing file is filling up. This can result if the Event Log service is stopped and started when events are being logged. A user has no control over these missing events.
- When logging to a file, the disk is too slow to keep up with the logging rate.

ETW Horror Stories #1

Scenario: *Using ETW-Threat-Intelligence Feeds to detect in-memory patches and certain cases of PIC.*

Memory Processing Issues:

- Logical Issues: e.g. Reading memory from a process that exited fast and its PID (Process Identified) was re-issued as the event came ”late”.
- Memory Processing Issues: e.g. Attempting to find a module in-memory that is not already loaded or was removed during a short timeframe, excluding NTDLL which is a standardized case.

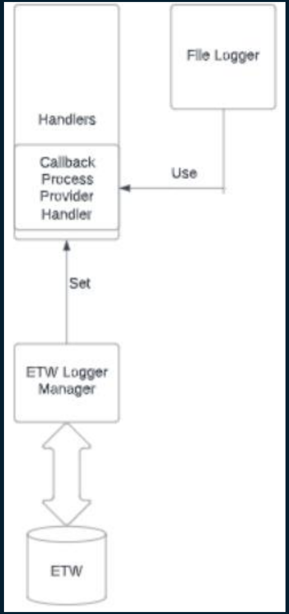
ETW Horror Stories #2

Performance, Events Missing and Buffer Degradation Issues:

- Performance impact and overwhelming load of data to be analyzed given APIs monitored may be used by the Windows loader and cause massive overhead and difficulties
- False Positives: e.g. Applications that reside inside the main application such as additional plugins and anti-exploit agents may increase overhead even more.

How we use ETW in the experiment?

- ETW is suitable for our scenario.
  - ETW is not for everything.
  - ETW detections may need additional effort.
- ETW has a lot of settings.
- ETW has a lot of APIs.
- ETW is complex.
- Krabs-ETW by Microsoft provides an OOP wrapper around ETW and makes using it easier.



What providers do we use ?

- Microsoft-Windows-Kernel-PnP
  - PnP Device Events
- Microsoft-Windows-Kernel-Process
  - Process Events.
  - Image Load Events.

Usual Keypress IRP Flow

- **Keypress**: A user presses a key on a keyboard.
- **Hardware Interrupt**: The physical keypress triggers a hardware interrupt sent to the processor.
- **Keyboard Controller**: The keyboard controller reads keypress data and puts it in the hardware buffer.
- **Keyboard Class Driver**: The keyboard class driver handles the hardware interrupt and reads the keypress data from the hardware buffer, creating an IRP for the keypress.
- **I/O Manager**: The IRP is sent to the I/O Manager.
- **Device Stack**: The IRP travels "down" the device stack, passing through filter drivers and other relevant drivers in turn. Each driver has a chance to inspect or modify the IRP.

Usual Keypress IRP Flow – Contd.

- **Hardware/Device Driver**: The IRP reaches the lowest-level driver, which interacts with the actual hardware device.
- **IRP Processing**: The IRP is processed, and the hardware operation corresponding to the original request (i.e., the keypress) is performed.
- **IRP Travels Back Up**: The IRP then travels "up" the device stack, triggering any relevant completion routines that were registered by the drivers.
- **I/O Manager**: The IRP reaches the I/O Manager again, which then cleans up the IRP.
- **Application Layer**: The result of the operation (i.e., the translated keypress) is returned to the application layer. This could be, for example, a character appearing in a text editor.

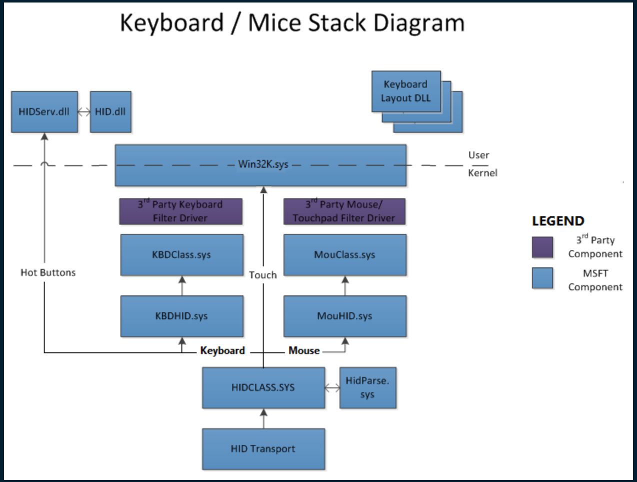
Keyboard Filtering Approaches

WDM Completion Routine Hook Approach. (Vanilla)

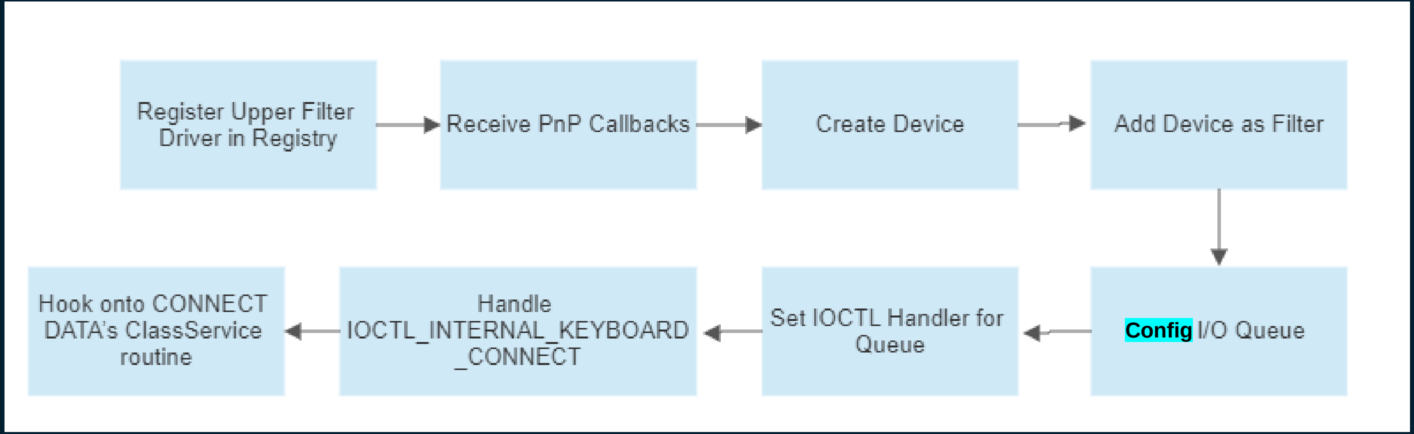
- Non-PnP-aware
- Synchronous

KMDF CONNECT_DATA Hook Approach.

- More PnP-aware
- Higher Success Rate Across Machines
- Synchronous

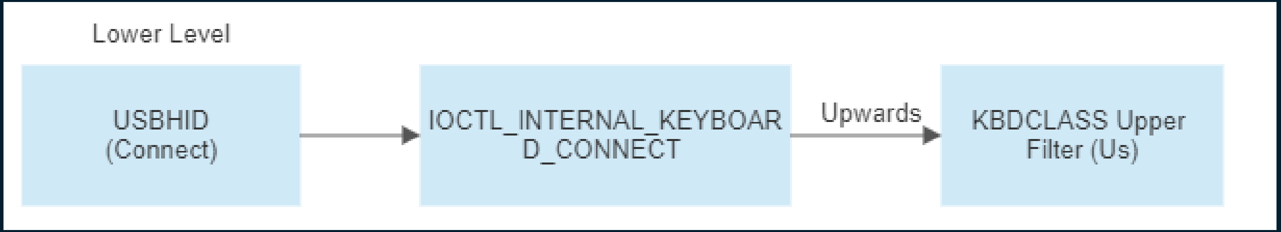


KMDF Driver Approach

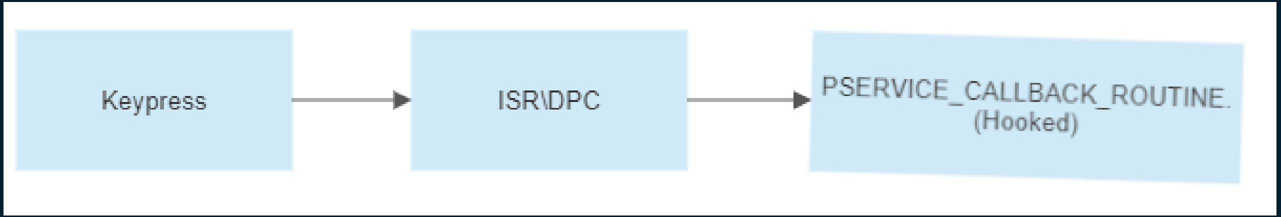

Register Upper Filter Driver in Registry
Receive PnP Callbacks
Create Device
Add Device as Filter
Config I/O Queue
Set IOCTL Handler for Queue
Handle IOCTL_INTERNAL_KEYBOARD_CONNECT
Hook onto CONNECT DATA's ClassService routine

Hooking Flows Visualized

Initial Routine Hooking – On Connect



Keypress Interception – On Press



Avoiding Mistakes of the Past

Privacy of the keystrokes must be taken into consideration.

The means of transport of the logs should be secured.

https://www.cyberpointllc.com/blog-posts/cp-logging-keystrokes-with-event-tracing-for-windows-etw.php
https://www.bitdefender.com/blog/hotforsecurity/hp-laptops-found-carrying-keylogger-in-synaptics-touchpad-driver/

Data to consider

Processes

Suspicious processes.

Images

Suspicious images indicating an action of interest.

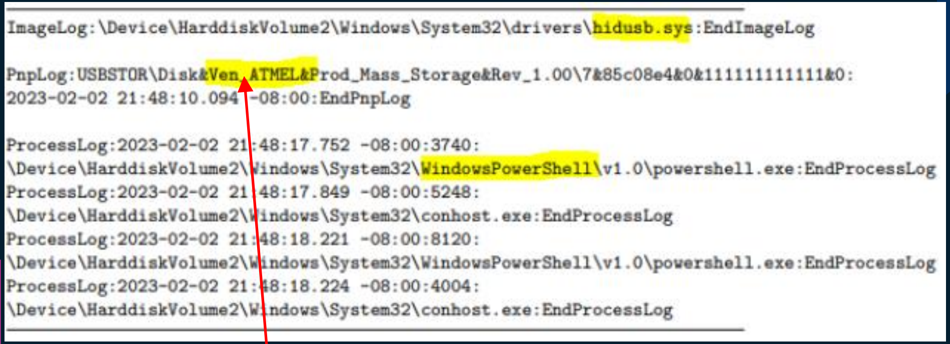
PnP

Suspicious device info commonly used by hacking gadgets.

Keystrokes

Speed of typing exceeding human capabilities.

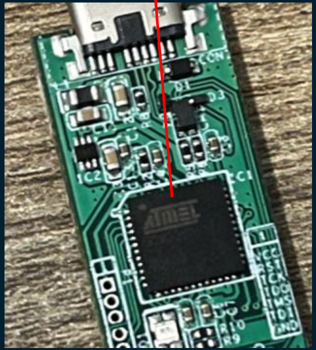
SAMPLE LOGS



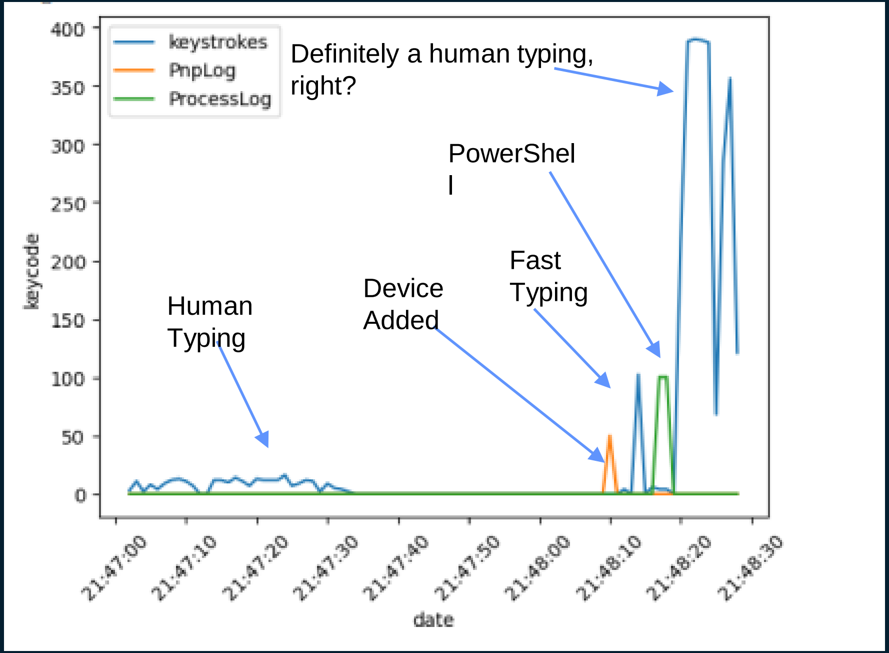
```
ImageLog:\Device\HarddiskVolume2\Windows\System32\drivers\hidusb.sys:EndImageLog

PnpLog:USBSTOR\Disk&Ven_ATMEL&Prod_Mass_Storage&Rev_1.00\7&85c08e4&0&111111111111&0:
2023-02-02 21:48:10.094 -08:00:EndPnpLog

ProcessLog:2023-02-02 21:48:17.752 -08:00:3740:
\Device\HarddiskVolume2\Windows\System32\WindowsPowerShell\v1.0\powershell.exe:EndProcessLog
ProcessLog:2023-02-02 21:48:17.849 -08:00:5248:
\Device\HarddiskVolume2\Windows\System32\conhost.exe:EndProcessLog
ProcessLog:2023-02-02 21:48:18.221 -08:00:8120:
\Device\HarddiskVolume2\Windows\System32\WindowsPowerShell\v1.0\powershell.exe:EndProcessLog
ProcessLog:2023-02-02 21:48:18.224 -08:00:4004:
\Device\HarddiskVolume2\Windows\System32\conhost.exe:EndProcessLog
```



Result Graph



*Hidusb.sys loading is not present in the graph

Before you ask.

Can't I spread the keystrokes with sleeps?

- The fast peak is not the only suspicious pattern someone with basic cognitive abilities can deem to be malicious.
- The gadget has to be fast: plug-in & plug-out.
  - If things go slow suspicion is raised and the attack may fail.

Can your detection model be automated and made real-time?

Possibly but it would require much effort to:

- Have proper timing.
- Develop a cognitive-like model to identify various kinds of speed peaks, correlate data from various sources and produce adequate verdicts and mitigations.
- Such an approach was out of scope, hence the "forensic" mindset.

THANKS!

Any questions?

You can find me at:

@GeKarantzas · gck.kara@gmail.com