elastic | The Search AI Company

Windows Keylogger Detection

Targeting Past and Present Keylogging Techniques

Asuka Nakajima | 中島 明日香

Senior Security Research Engineer @ elastic

elastic

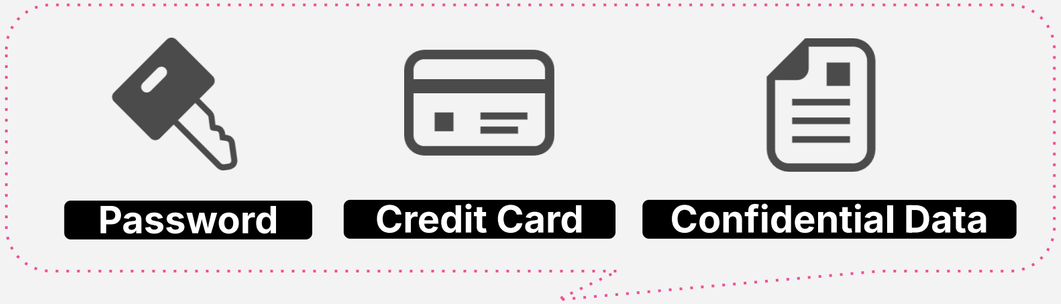
नमस्ते / Hello!😊

Asuka Nakajima

- **Senior Security Research Engineer @ Elastic**
  - ✔ Endpoint Protections Team
  - ✔ Endpoint Security R&D, especially developing new detection features for EDR (Elastic Defend)
  - ✔ 10+ years of experience in cyber security R&D
- **Founder of CTF for GIRLS (est. 2014)**
  - ✔ First female infosec community in Japan
- **Review Board Member**
  - ✔ BlackHat USA / BlackHat Asia / CODE BLUE

Keyloggers: Still Used in Today's Cyber Attacks

❖ What is a keylogger ? 🤔



- ✓ Software that records keystrokes
- ✓ Often misused by malware / malicious actors to steal sensitive data
  - Has been used for a long time and is still being found in malware today (e.g., Agent Tesla)

❖ What are the risks ?🤔

- ✓ e.g., Stolen data may be used for financial theft or further cyber attacks.

Early detection is crucial to prevent subsequent attacks

Types of Keyloggers

Hardware Keyloggers

```
while(true)
{
  for (int key = 1; key <= 255; key++)
  {
    if (GetAsyncKeyState(key) & 0x01)
    {
      SaveTheKey(key, "log.txt");
    }
  }
  Sleep(50);
}
```

Software Keyloggers

Types of Keyloggers

Hardware Keyloggers

```
while(true)
{
  for (i  key = 1; key <= 255; key+
  {
      GetAsyncKeyState(key) & 0x0

      SaveTheKey(key, "log.txt");
    }
  }
  Sleep( );
}
```

Software Keyloggers

Types of Keyloggers

💡 This talk focuses on

Windows API-based user mode keyloggers and their detection

Hardware Keyloggers

Software Keyloggers

About Today's Talk

Part A

Detecting Common Types of Keyloggers Through Windows API Monitoring

💡 Sharing my experience of adding a keylogger behavioral detection feature to an EDR

Part B

Hotkey-based Keylogger Detection

About Today's Talk

Part A

Detecting Common Types of Keyloggers Through Windows API Monitoring

💡 Sharing my experience of adding a keylogger behavioral detection feature to an EDR

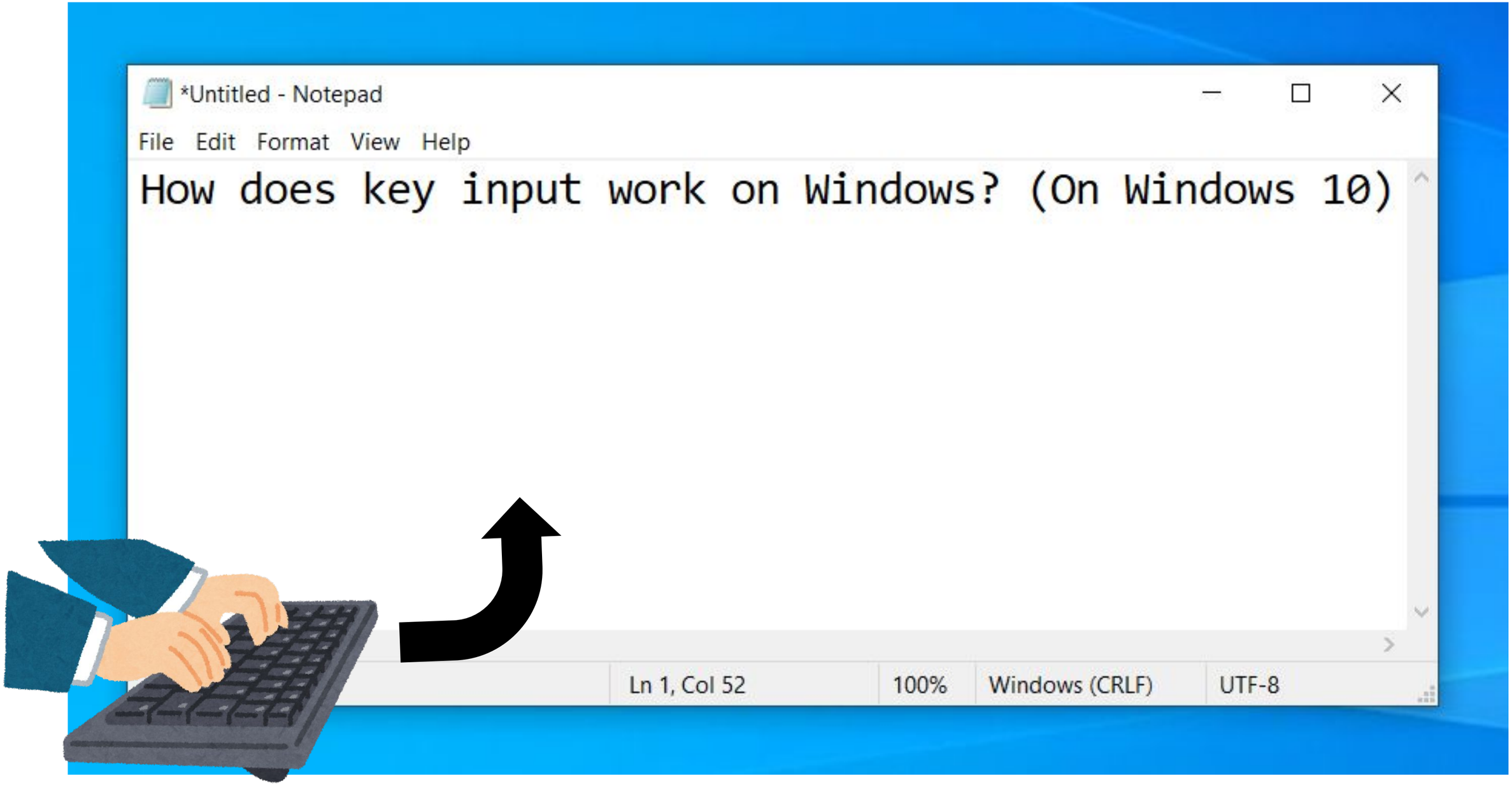
Part B

Hotkey-based Keylogger Detection

Four Common Types of Windows API-based User-mode Keyloggers

- ✅ Polling-based Keyloggers
- ✅ Hooking-based Keyloggers
- ✅ Keyloggers using Raw Input Model
- ✅ Keyloggers using DirectInput

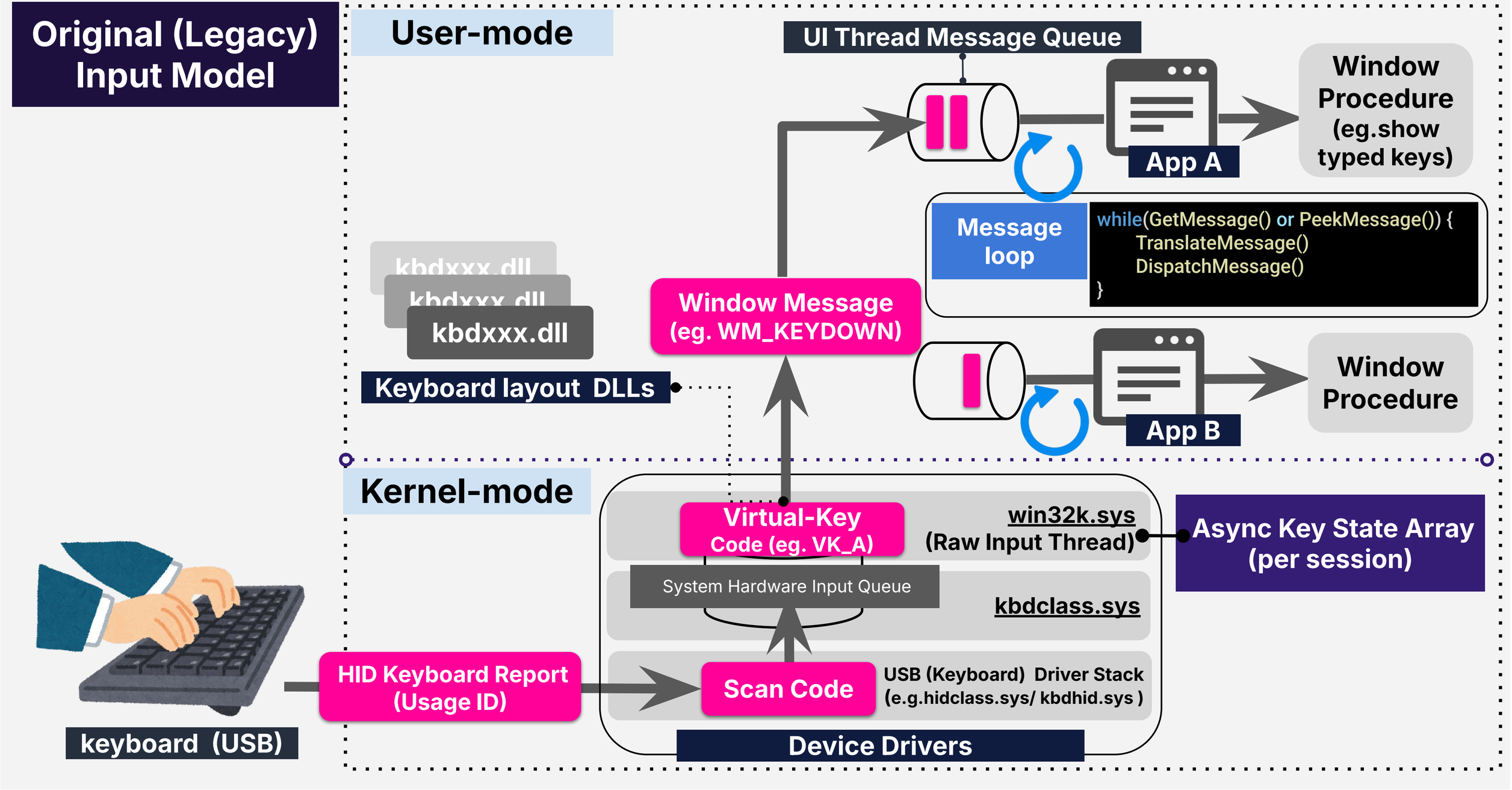
To detect them, we must first understand how they work



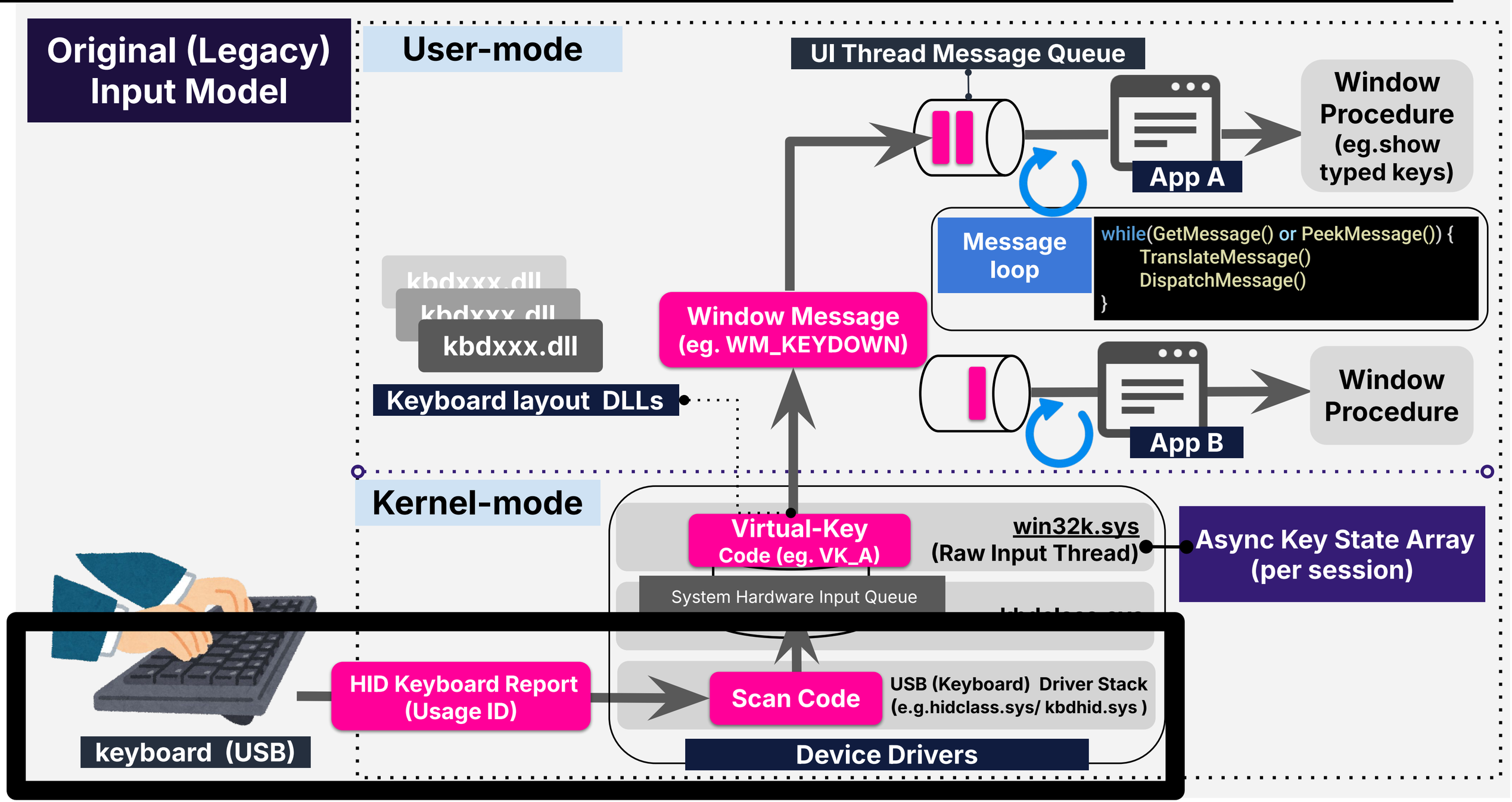
keyboard

*Note: **All information in this talk is based on Windows 10 version 22H2 OS Build 19045.5371 without virtualization-based security**. Please note that internal data structures and behavior may differ in other versions of Windows.*

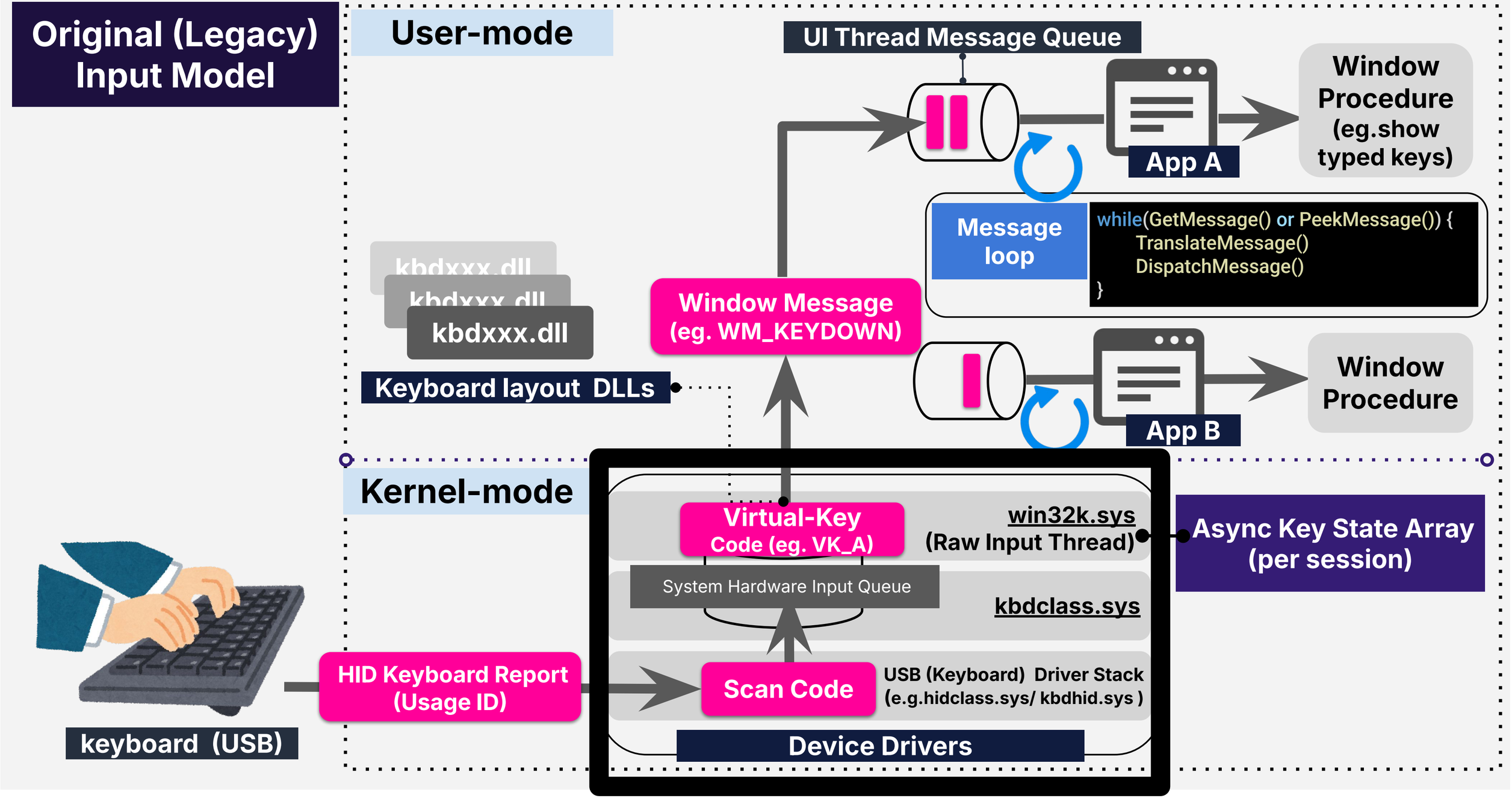
Simplified Diagram of the Key Input Flow from Keyboard to Application (Windows)



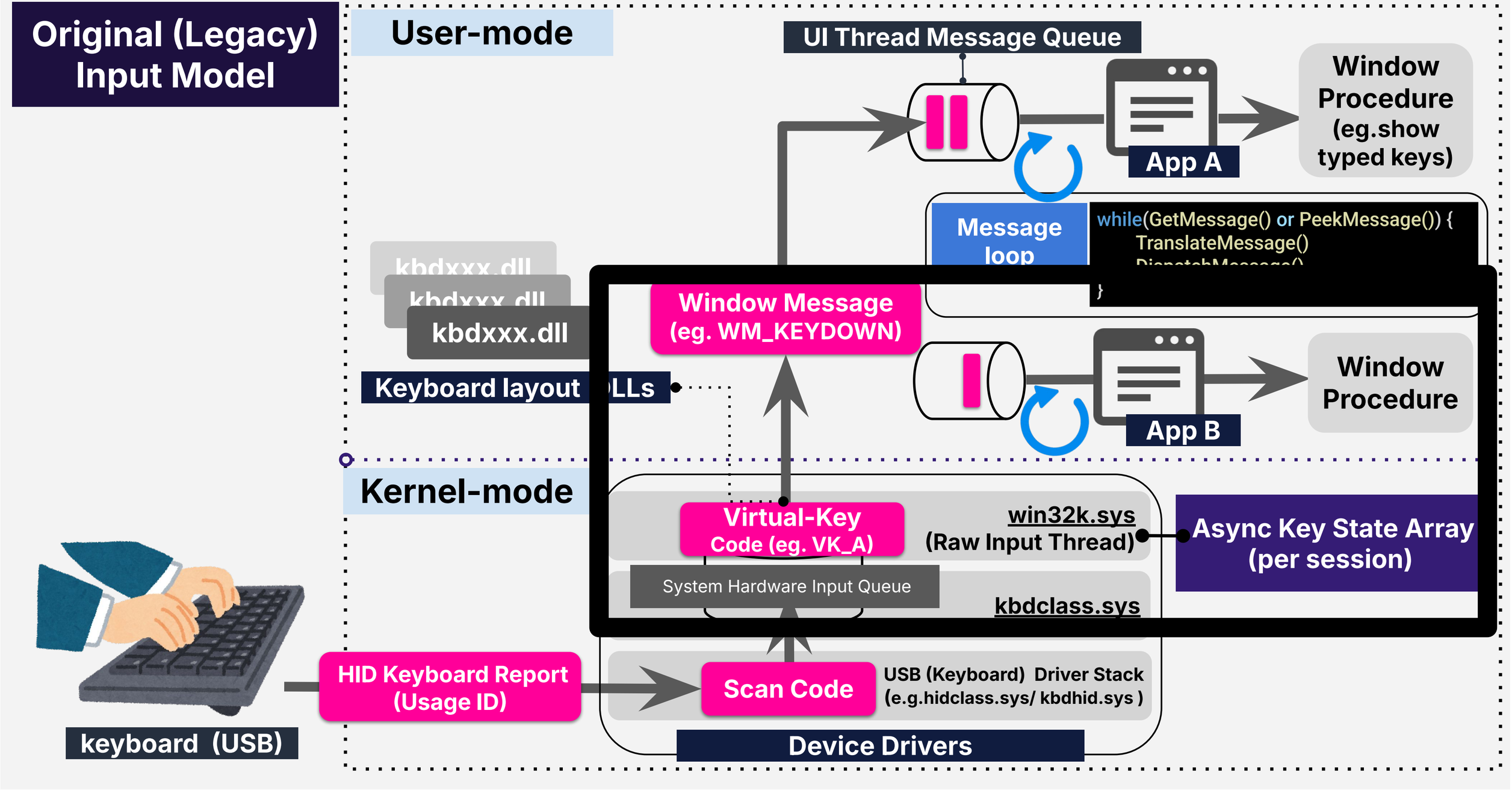
Simplified Diagram of the Key Input Flow from Keyboard to Application (Windows)



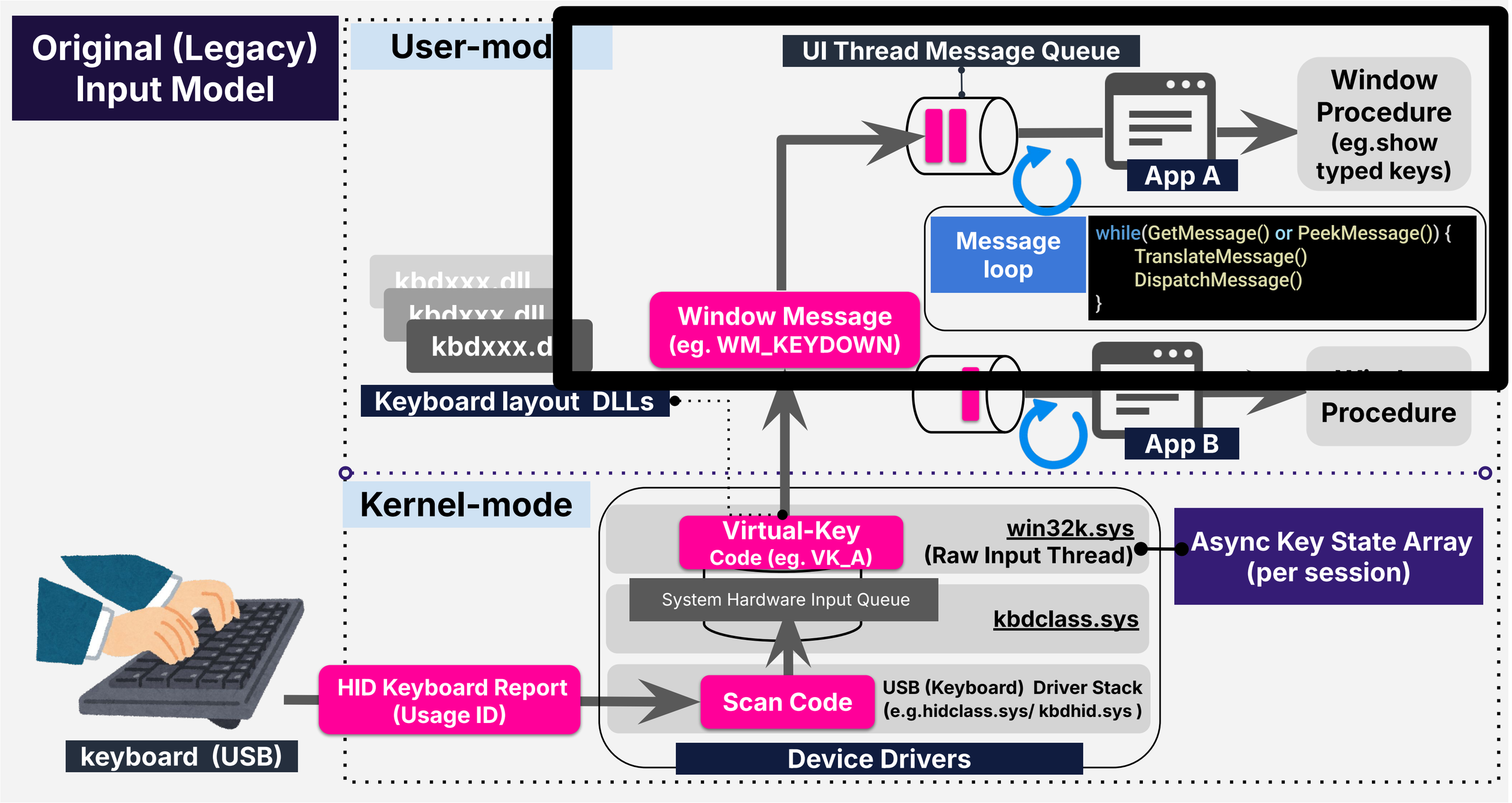
Simplified Diagram of the Key Input Flow from Keyboard to Application (Windows)



Simplified Diagram of the Key Input Flow from Keyboard to Application (Windows)



Simplified Diagram of the Key Input Flow from Keyboard to Application (Windows)



Polling-based Keylogger

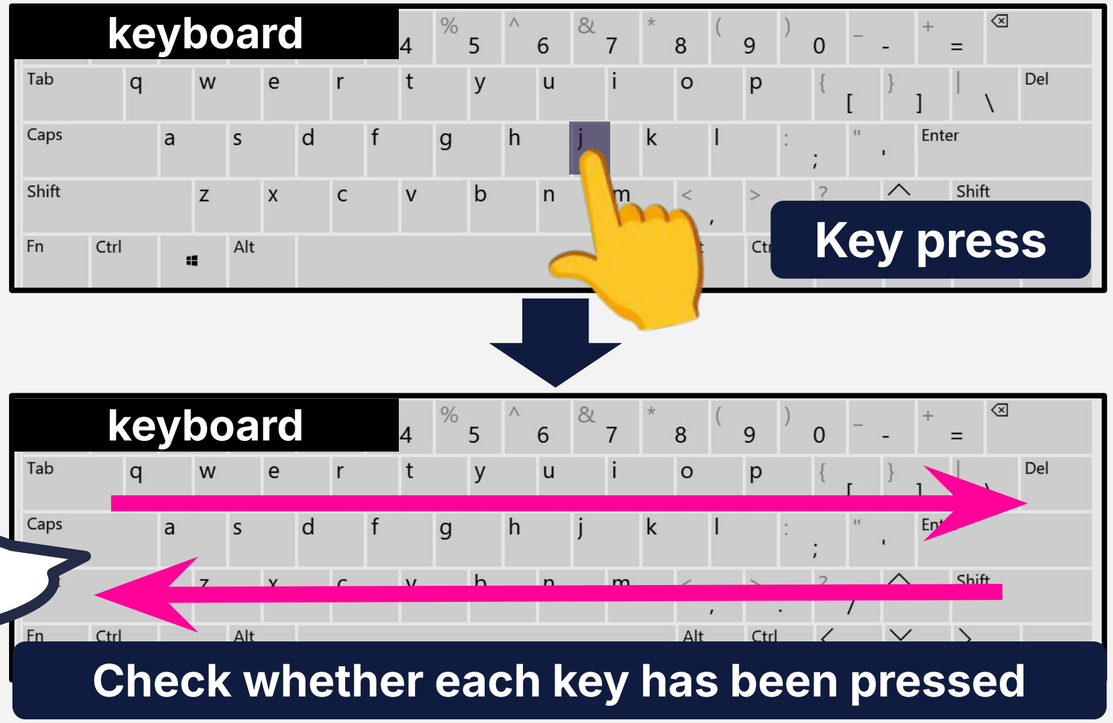
How It Captures Keystrokes

Periodically checks each key state on the keyboard at very short intervals. The *GetAsyncKeyState* API is commonly used for this.

Example Code

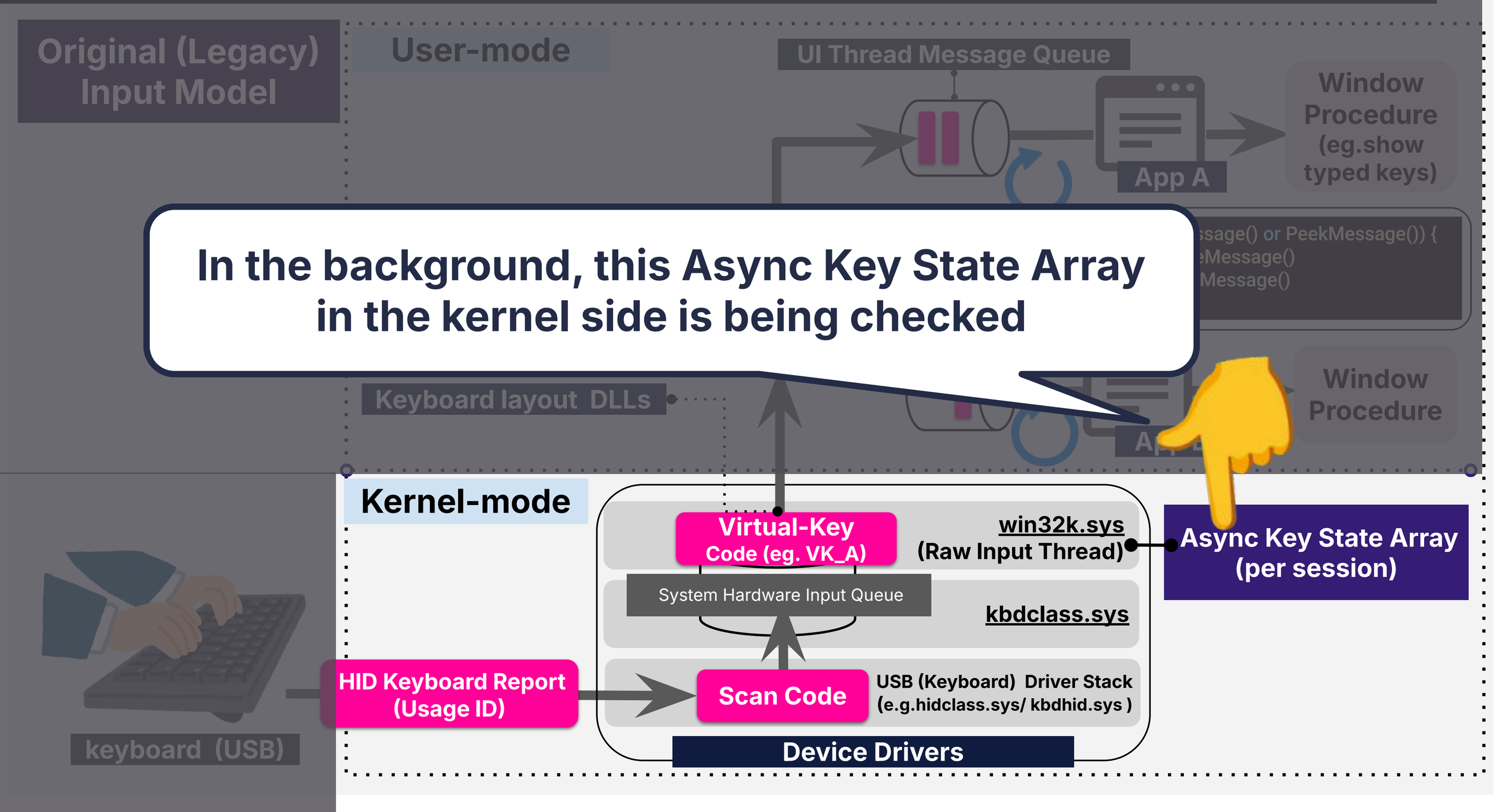
```
while(true)
{
  for (int key = 1; key <= 255; key++)
  {
    if (GetAsyncKeyState(key) & 0x01)
    {
      SaveTheKey(key, "log.txt");
    }
  }
  Sleep(50);
}
```

The J key was pressed!



GetAsyncKeyState API: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getasynckeystate

Simplified Diagram of the Key Input Flow from Keyboard to Application (Windows)



In the background, this Async Key State Array in the kernel side is being checked

Hooking-based Keylogger

How It Captures Keystrokes

Windows provides a hooking mechanism that allows programs to intercept certain window messages before they reach their intended application.

Example.
WM_KEYDOWN
WM_KEYUP

SetWindowsHookEx **API provides this feature**

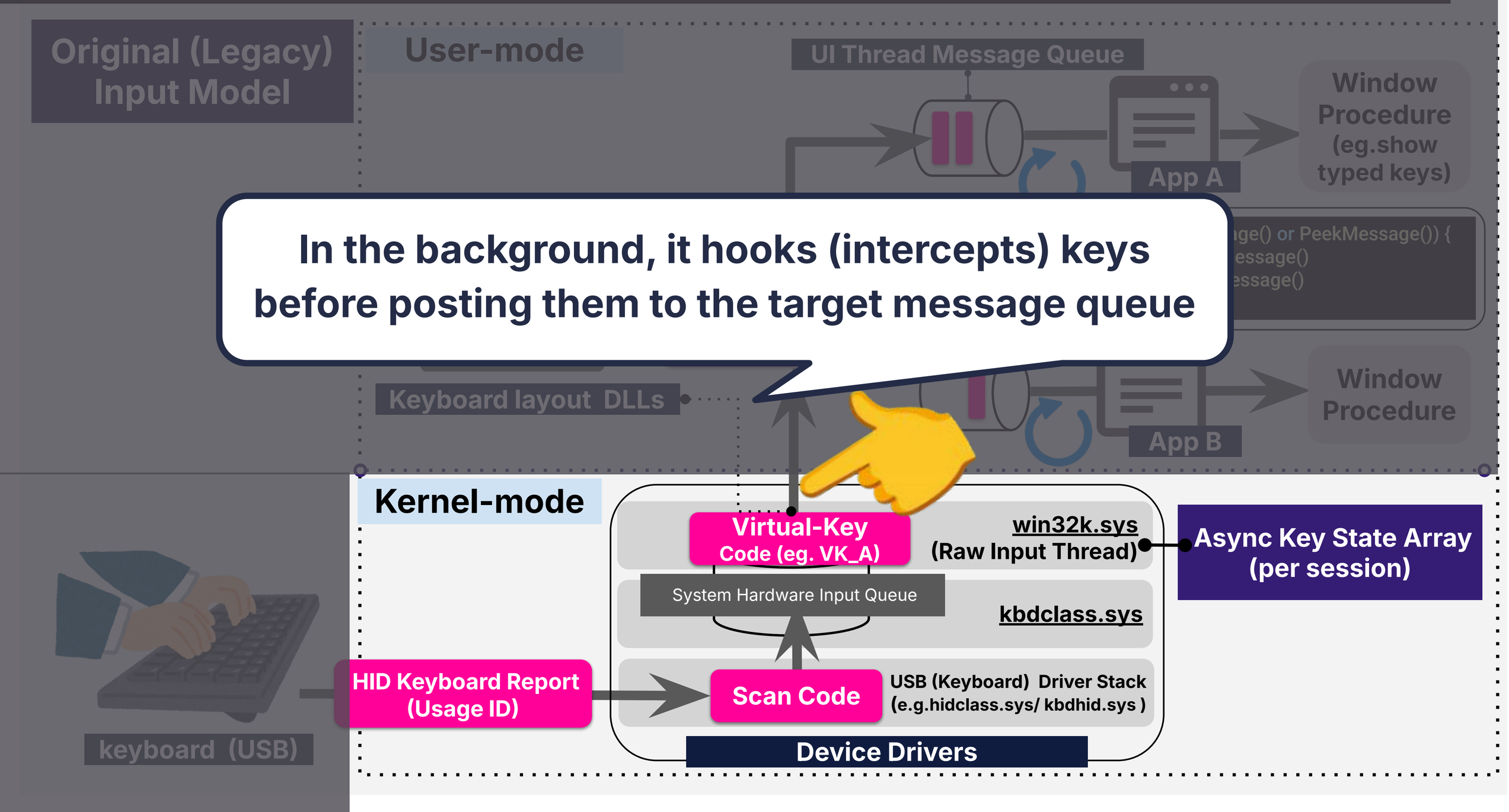
Example Code

```
HMODULE hHookLibrary = LoadLibraryW(L"hook.dll");
FARPROC hookFunc = GetProcAddress(hHookLibrary, "SaveTheKey");

HHOOK keyboardHook = NULL;
keyboardHook = SetWindowsHookEx(WH_KEYBOARD_LL, (HOOKPROC)hookFunc, hHookLibrary, 0);
```

SetWindowsHookEx API: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexw

Simplified Diagram of the Key Input Flow from Keyboard to Application (Windows)



Keylogger using Raw Input Model

Input Model on Windows

❖ Original Input Model

- ✓ The data entered from input devices like keyboards is **processed by the OS** before it is delivered to the target application.

❖ Raw Input Model

- ✓ The data entered from input devices is received directly by the target application **without any intermediate processing by the OS.**

💡 **Raw keyboard input is sent to the application** when the Raw Input Model is used.

About Raw Input: https://learn.microsoft.com/en-us/windows/win32/inputdev/about-raw-input

Keylogger using Raw Input Model

How It Captures Keystrokes

This type of keylogger captures and records raw input data obtained from input devices like keyboards.

Example Code (1/2)

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMessage, WPARAM wParam, LPARAM lParam) {
  UINT dwSize = 0;
  RAWINPUT* buffer = NULL;

  switch (uMessage) {
  case WM_CREATE:
     RAWINPUTDEVICE rid;
     rid.usUsagePage = 0x01;  // HID_USAGE_PAGE_GENERIC
     rid.usUsage = 0x06;        // HID_USAGE_GENERIC_KEYBOARD
     rid.dwFlags = RIDEV_INPUTSINK;
     rid.hwndTarget = hWnd;
     RegisterRawInputDevices(&rid, 1, sizeof(rid));
     break;
-[continues to the next page]-
```

RegisterRawInputDevices API, registers the devices that supply raw input data.

RegisterRawInputDevices API: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-registerrawinputdevices

Keylogger using Raw Input Model

Example Code (2/2)

```
  case WM_INPUT:
    GetRawInputData((HRAWINPUT)lParam, RID_INPUT, NULL, &dwSize, sizeof(RAWINPUTHEADER));
    buffer = (RAWINPUT*)HeapAlloc(GetProcessHeap(), 0, dwSize);

    if (GetRawInputData((HRAWINPUT)lParam, RID_INPUT, buffer, &dwSize, sizeof(RAWINPUTHEADER))){
      if (buffer->header.dwType == RIM_TYPEKEYBOARD){
        SaveTheKey(buffer, "log.txt");
      }
    }
    HeapFree(GetProcessHeap(), 0, buffer);
    break;
  default:
    return DefWindowProc(hWnd, uMessage, wParam, lParam);
  }
  return 0;
}
```

GetRawInputData API retrieves raw input from the registered device

GetRawInputData API: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getrawinputdata

Keylogger using DirectInput

What is DirectInput ?

A collection of APIs used for handling multimedia tasks such as gaming and video

- **One of the components of Microsoft DirectX API**
  - ✓ e.g., DirectInput, DirectShow, DirectAudio, etc.
- **DirectInput can retrieve the keyboard state using APIs such as the following**
  - ✓ DirectInput8Create
  - ✓ IDirectInputDevice8::Acquire
  - ✓ IDirectInputDevice8::GetDeviceState

Learn / Previous Versions / Windows / Desktop /

DirectInput

Article • 09/10/2011

This section provides information about the Microsoft DirectInput component of the Microsoft DirectX application programming interface (API).

The DirectInput API is used to process data from a joystick, or other game controller. The use of DirectInput for keyboard and mouse input is not recommended. You should use Windows messages instead .

Roadmap

The following sections describe how you can use DirectInput. Use this page to guide you through the documentation based on your level of experience as a DirectInput developer.

DirectInput: https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ee416842(v=vs.85)
DirectInput8Create: https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ee416756(v=vs.85)
IDirectInputDevice8::Acquire:https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ee417818(v=vs.85)
IDirectInputDevice8::GetDeviceState:https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ee417897(v=vs.85)

Keylogger using DirectInput

Example Code

```
LPDIRECTINPUT8          lpDI = NULL;
LPDIRECTINPUTDEVICE8    lpKeyboard = NULL;
BYTE key[256];
ZeroMemory(key, sizeof(key));

DirectInput8Create(hInstance, DIRECTINPUT_VERSION, IID_IDirectInput8, (LPVOID*)&lpDI, NULL);
lpDI->CreateDevice(GUID_SysKeyboard, &lpKeyboard, NULL);
lpKeyboard->SetDataFormat(&c_dfDIKeyboard);
lpKeyboard->SetCooperativeLevel(hwndMain, DISCL_FOREGROUND | DISCL_NONEXCLUSIVE | DISCL_NOWINKEY);

while(true) {
  HRESULT ret = lpKeyboard->GetDeviceState(sizeof(key), key);
  if (FAILED(ret)) {
     lpKeyboard->Acquire();
     lpKeyboard->GetDeviceState(sizeof(key), key);
  }
  SaveTheKey(key, "log.txt");
  Sleep(50);
}
```

Keylogger using DirectInput

Confirmed that the RegisterRawInputDevices API is being called internally!

```
\dinput8.dll
na  Options  Windows  Help
No debugger
Data  Unexplored  External symbol  Lumina function
IDA View-A  Pseudocode-A  Hex View-1  Local Types

                                                        ; CODE XREF: CEm_LL_Acquire+C3↑j
                                                edx, edx
                                                        ; CODE XREF: CEm_LL_Acquire+BE↑j
.text:0000000180011DCB                  mov     ecx, edi
.text:0000000180011DCB                  sub     ecx, esi
.text:0000000180011DCD
.text:0000000180011DCD loc_180011DCD:                   ; CODE XREF: CEm_LL_Acquire+CE↑j
.text:0000000180011DCD                  call    CDIRaw_RegisterRawInputDevice
.text:0000000180011DD2                  mov     ebx, eax
.text:0000000180011DD4
.text:0000000180011DD4 loc_180011DD4:                   ; CODE XREF: CEm_LL_Acquire+D5↑j
.text:0000000180011DD4                  test    ebx, ebx
.text:0000000180011DD6                  jns     short loc_180011E27
.text:0000000180011DD8                  and     cs:g_fRawInput, 0
.text:0000000180011DDF                  mov     ebx, edi
.text:0000000180011DE1                  jmp     short loc_180011E27
```

👆 **From dinput8.dll (version: 10.0.19041.1)**

*Note: We haven't fully analyzed **dinput8.dll**, but at least when running the keylogger, we confirmed that the **RegisterRawInputDevices** was being called internally.*

Detecting Keyloggers by Monitoring Windows API calls

Developed a new feature in the EDR that detects keyloggers by monitoring API calls and analyzing their behavior

❖ How can we monitor Windows API calls?

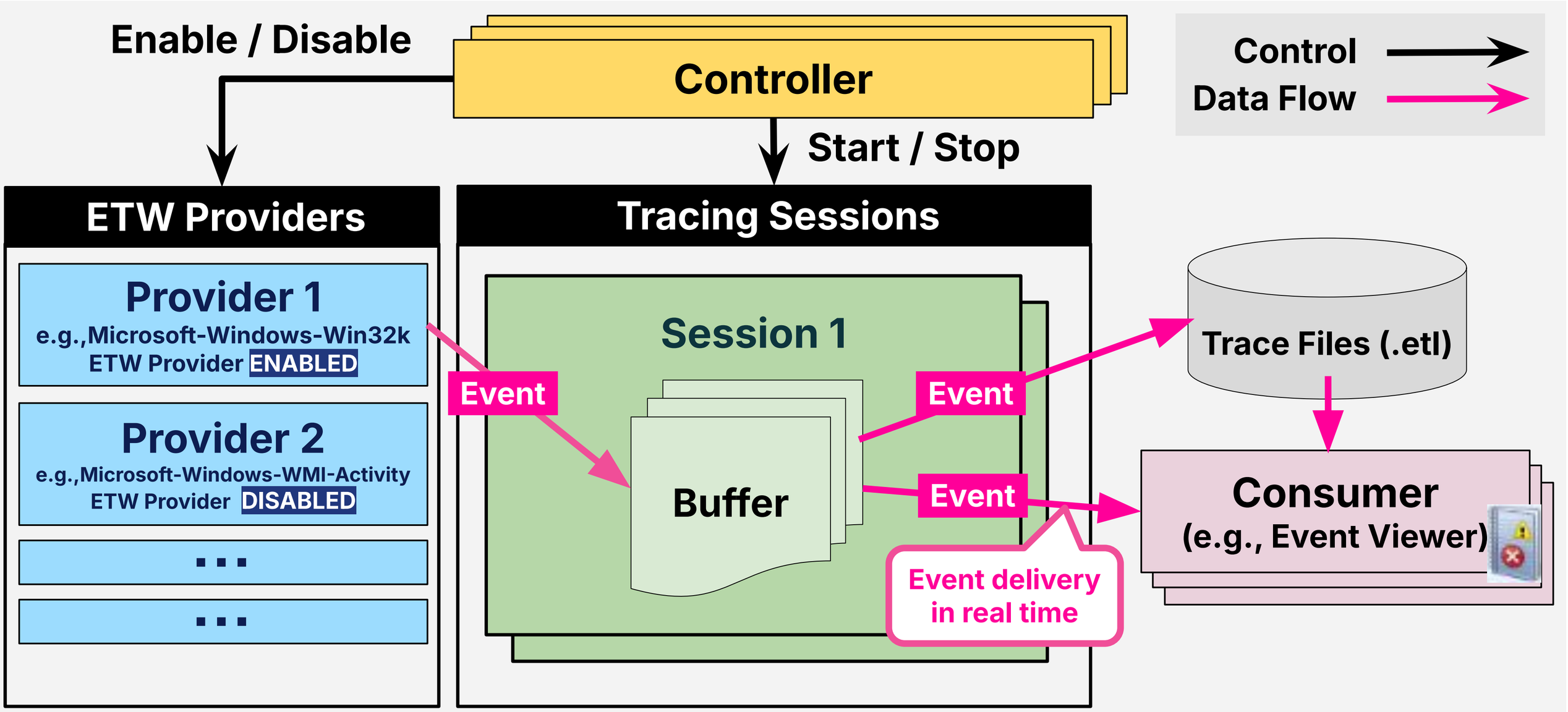
✓ **<u>Event Tracing for Windows (ETW)</u>**

- Framework provided by Microsoft for tracing and logging the execution of applications and system components in Windows
- **Microsoft-Windows-Win32k ETW Provider**
  - **Manifest-based ETW Provider (the modern ETW event provider)**

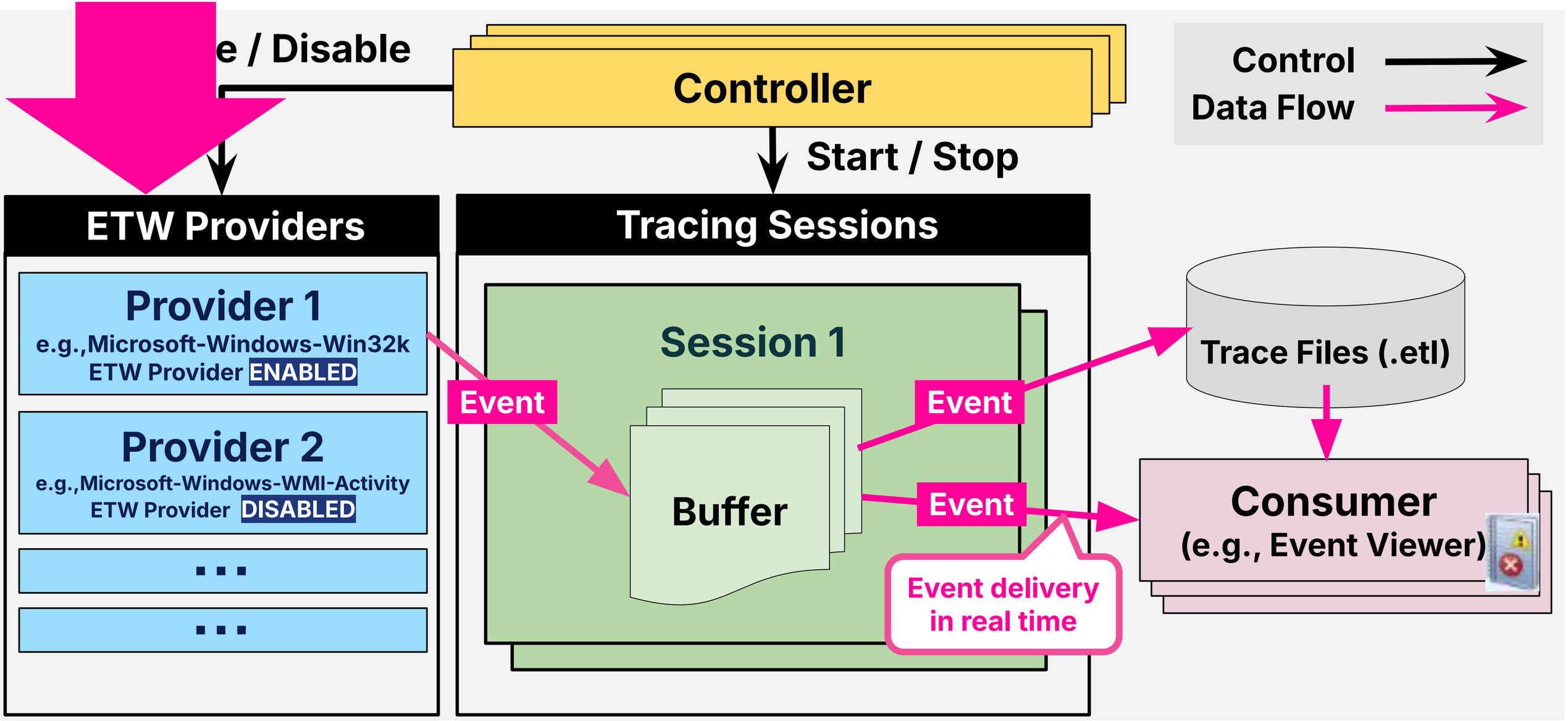
Elastic Security Defend Integration (EDR)

Event Tracing for Windows(ETW): https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw-
Kernel ETW is the best ETW, https://www.elastic.co/security-labs/kernel-etw-best-etw

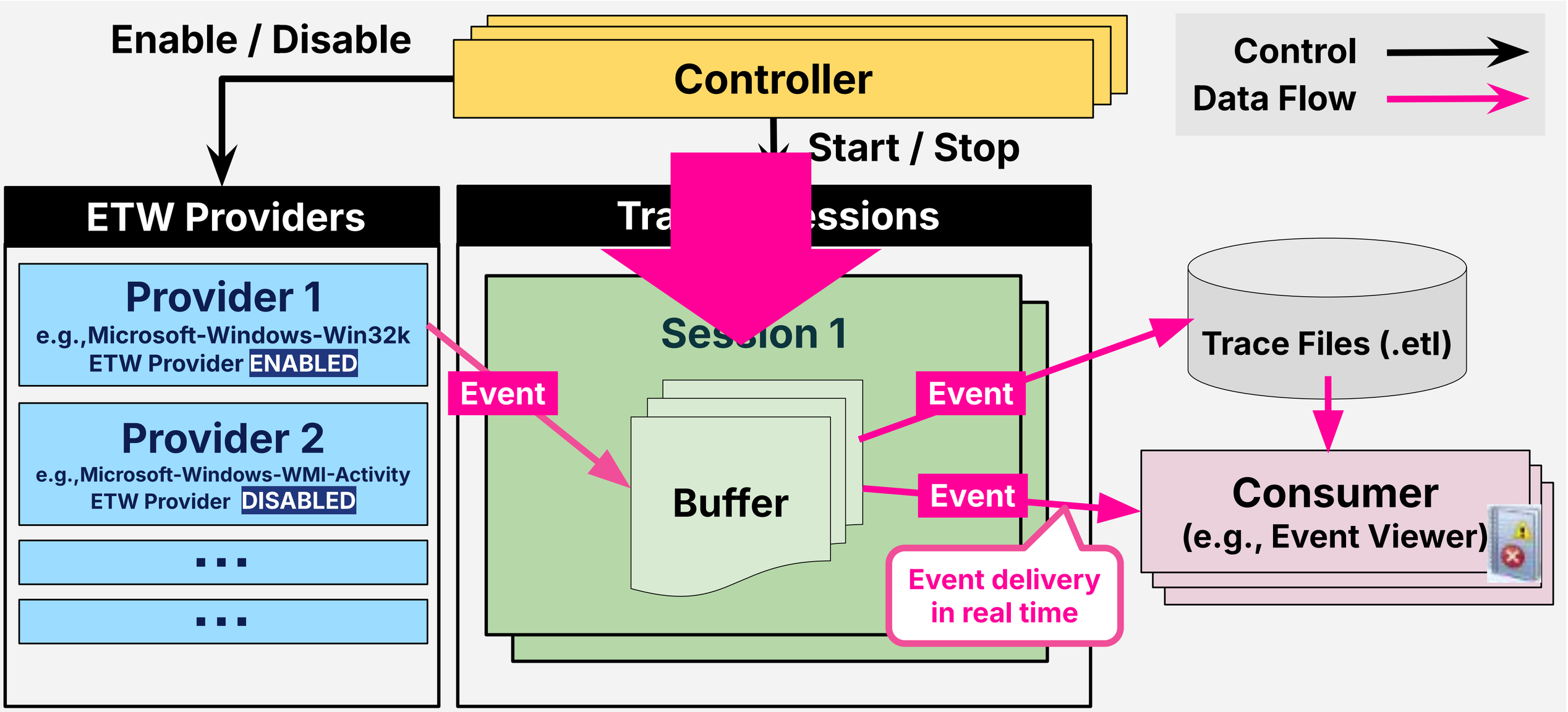
Event Tracing for Windows (ETW)



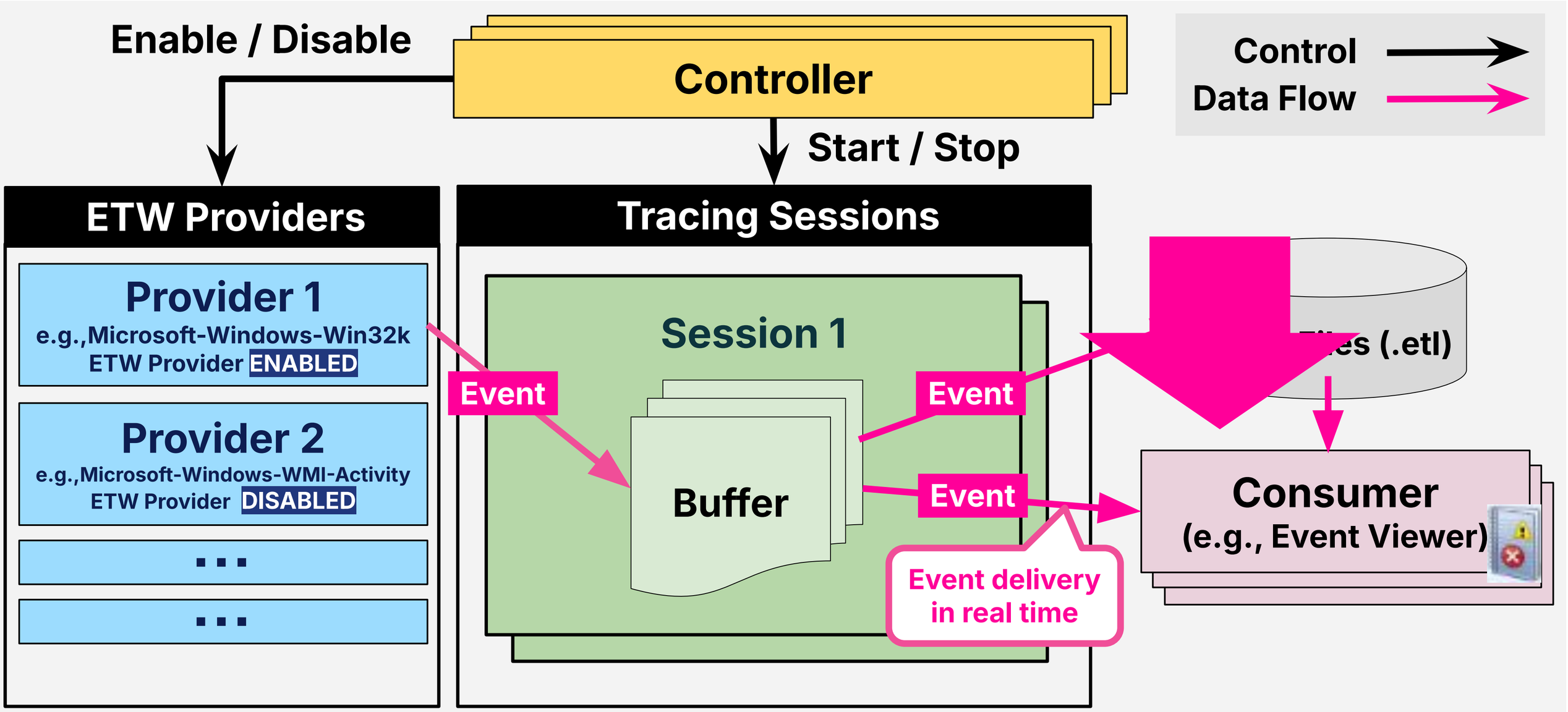
Event Tracing for Windows (ETW)



Event Tracing for Windows (ETW)



Event Tracing for Windows (ETW)



Internals of Providers (e.g., Win32k)

The Microsoft-Windows-Win32k provider is a kernel-level provider that emits ETW events from the kernel level

User-mode

(user32.dll) GetAsyncKeyState

↓

(win32u.dll) NtUserGetAsyncKeyState

Kernel-mode

↓

(win32kbase.sys) NtUserGetAsyncKeyState

IDA View-A | Pseudocode-A | Hex Vie

```
1  __int64 __fastcall NtUserGetAsyncKeyState(unsigned int a1)
2  {
3    __int64 Thre
4    __int16 Asyn
5    char v5; //
6
7    EnterSharedCrit(0LL, 1LL);
8    ThreadWin32Thread = 32GetThreadWin32Thread(KeGetCurrentThread()
9    AsyncKeyState = 0
10   if ( gptiForeground && PsGetCurrentProcessWin32Process() != *((_
11     EtwTraceGetAsyncKeyState(ThreadWin32Thread);
12   if ( (unsigned int)ApiSetEditionIsGetAsyncKeyStateBlocked() )
13     goto LABEL_10;
14   if ( !(unsigned int)ApiSetEditionIsGpqForegroundAccessibleCurren
15   {
16     EtwTraceUIPIInputError((struct tagTHREADINFO *)ThreadWin32Thre
```

EtwTraceGetAsyncKeyState function which is associated to ETW event writing

👆 From win32kbase.sys (version: 10.0.19041.5247)

ETW Providers

We can see all the providers registered in Windows using the *logman query providers* command

Select Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.19045.5487]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vagrant>logman query providers

Provider                                        GUID
-------------------------------------------------------------------------------
ACPI Driver Trace Provider                      {DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
Active Directory Domain Services: Core          {1C83B2FC-C04F-11D1-8AFC-00C04FC219
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}
BCP.1                                           {24722B88-DF97-4FF6-E395-DB533AC42A1E}
BFE Trace Provider                              {106B464A-8043-46B1-8CB8-E92A0CD7A560}
BITS Service Trace                              {4A8AAA94-CFC4-46A7-8E4E-17BC45608F0A}
Certificate Services Client CredentialRoaming Trace {FF4109DC-68FC-45AF-B329-CA2825437209}
```

More than <u>1,000 providers</u> are registered by default!😮

Manifest Files (for Manifest Based ETW Providers)

A document that specifies event structures such as event categories, fields, and levels for the tracing

https://github.com/microsoft/perfview

```
> PerfView.exe /noGUI userCommand DumpRegisteredManifest Microsoft-Windows-Win32k
```

```
<instrumentationManifest xmlns="http://schemas.microsoft.com/win/2004/08/events">
 <instrumentation xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:win="http://man
  <events>
   <provider name="Microsoft-Windows-Win32k" guid="{8c416c79-d49b-4f01-a467-e56d3aa8234c}" resourceFileName="Microsoft-Windows-Win32k" me
    <keywords>
     <keyword name="AuditApiCalls" message="$(string.keyword_AuditApiCalls)" mask="0x400"/>
     <keyword name="CompatImpact" message="$(string.keyword_CompatImpact)" mask="0x800"/>
     <keyword name="Updates" message="$(string.keyword_Updates)" mask="0x1000"/>
     <keyword name="Focus" message="$(string.keyword_Focus)" mask="0x2000"/>
     <keyword name="UIPI" message="$(string.keyword_UIPI)" mask="0x4000"/>
     <keyword name="win32Power" message="$(string.keyword_win32Power)" mask="0x8000"/>
     <keyword name="Concurrency" message="$(string.keyword_Concurrency)" mask="0x10000"/>
     <keyword name="UserActivity" message="$(string.keyword_UserActivity)" mask="0x20000"/>
     <keyword name="UIUnresponsiveness" message="$(string.keyword_UIUnresponsiveness)" mask="0x40000"/>
     <keyword name="ThreadRundown" message="$(string.keyword_ThreadRundown)" mask="0x80000"/>
     <keyword name="Rendering" message="$(string.keyword_Rendering)" mask="0x100000"/>
     <keyword name="ThreadInfo" message="$(string.keyword_ThreadInfo)" mask="0x200000"/>
     <keyword name="MessagePump" message="$(string.keyword_MessagePump)" mask="0x400000"/>
     <keyword name="MessagePumpInternalAndInput" message="$(string.keyword_MessagePumpInternalAndInput)" mask="0x800000"/>
     <keyword name="TouchInput" message="$(
     <keyword name="TimerSurvey" message="$(
     <keyword name="PointerInput" message="$
```

Microsoft-Windows-Win32k.manifest.xml

Event Fields Definition (Manifest File)

GetAsyncKeyState (Event ID: 1003)

```
<template tid="task_01003Args">
 <data name="PID" inType="win:UInt32"/>
 <data name="MsSinceLastKeyEvent" inType="win:UInt32"/>
 <data name="BackgroundCallCount" inType="win:UInt32"/>
```

Microsoft-Windows-Win32k.manifest.xml

ETW events also include the process ID, thread ID, and other metadata of the triggered event.

```
- <Event xmlns="http://schemas.microsoft.com/win/2004/08/events
  - <System>
      <Provider Name="Microsoft-Windows-Win32k" Guid="{8c416c79-d
      <EventID>1003</EventID>
      <Version>0</Version>
      <Level>4</Level>
      <Task>0</Task>
      <Opcode>0</Opcode>
      <Keywords>0x400</Keywords>
      <TimeCreated SystemTime="2021-11-18T10:05:13.5657616Z" />
      <EventRecordID>21</EventRecordID>
      <Correlation ActivityID="{8c416c79-d49b-4f01-a467-e56d3aa823
      <Execution ProcessID="864" ThreadID="1356" />
      <Channel />
      <Computer>LAPT[                E</Computer>
      <Security />
    </System>
  - <EventData>
      <Data Name="PID">18716</Data>
      <Data Name="MsSinceLastKeyEvent">141</Data>
      <Data Name="BackgroundCallCount">449</Data>
    </EventData>
  </Event>
```

Challenges in Understanding Manifest Files

Challenges

- The event name may not be provided.
- The field name may not clearly describe the collected data.
- Events and fields may change based on the Windows version.
- The manifest file does not specify event trigger conditions.

Sometimes, it is necessary to perform reverse engineering, call relevant APIs to check which events are generated, and search for other researchers' findings.

Target ETW Events and Useful Fields for Detection

Event Name (Event ID)	Field Name	Reason
GetAsyncKeyState (Event ID: 1003)	MsSinceLastKeyEvent	For detecting polling-based keyloggers
	BackgroundCallCount	
SetWindowsHookEx (Event ID: 1002)	FilterType	For detecting hooking-based keyloggers
	pstrLib	
	pfnFilterProc	
RegisterRawInputDevices (Event ID: 1001)	ReturnValue	For detecting keyloggers using Raw Input and DirectInput
	UsagePage	
	Usage	
	Flags	
	ThreadStartAddress	
	cWindows	
	cVisWindows	
	ThreadInfoFlags	
	ThreadStartAddressMappedModuleName	
	ThreadStartAddressVadAllocationProtect	

Tool Release: ETW_Win32kAPIMonitor

https://github.com/AsuNa-jp/ETW_Win32kAPIMonitor

A standalone tool which monitors API calls related to keyloggers (GetAsyncKeyState / SetWindowsHookEx / RegisterRawInputDevices) using the Win32k ETW provider

- **ETW APIs for starting, configuring, opening, and processing trace sessions**
  - ✓ StartTraceW
  - ✓ EnableTraceEx2
  - ✓ OpenTraceW
  - ✓ ProcessTrace

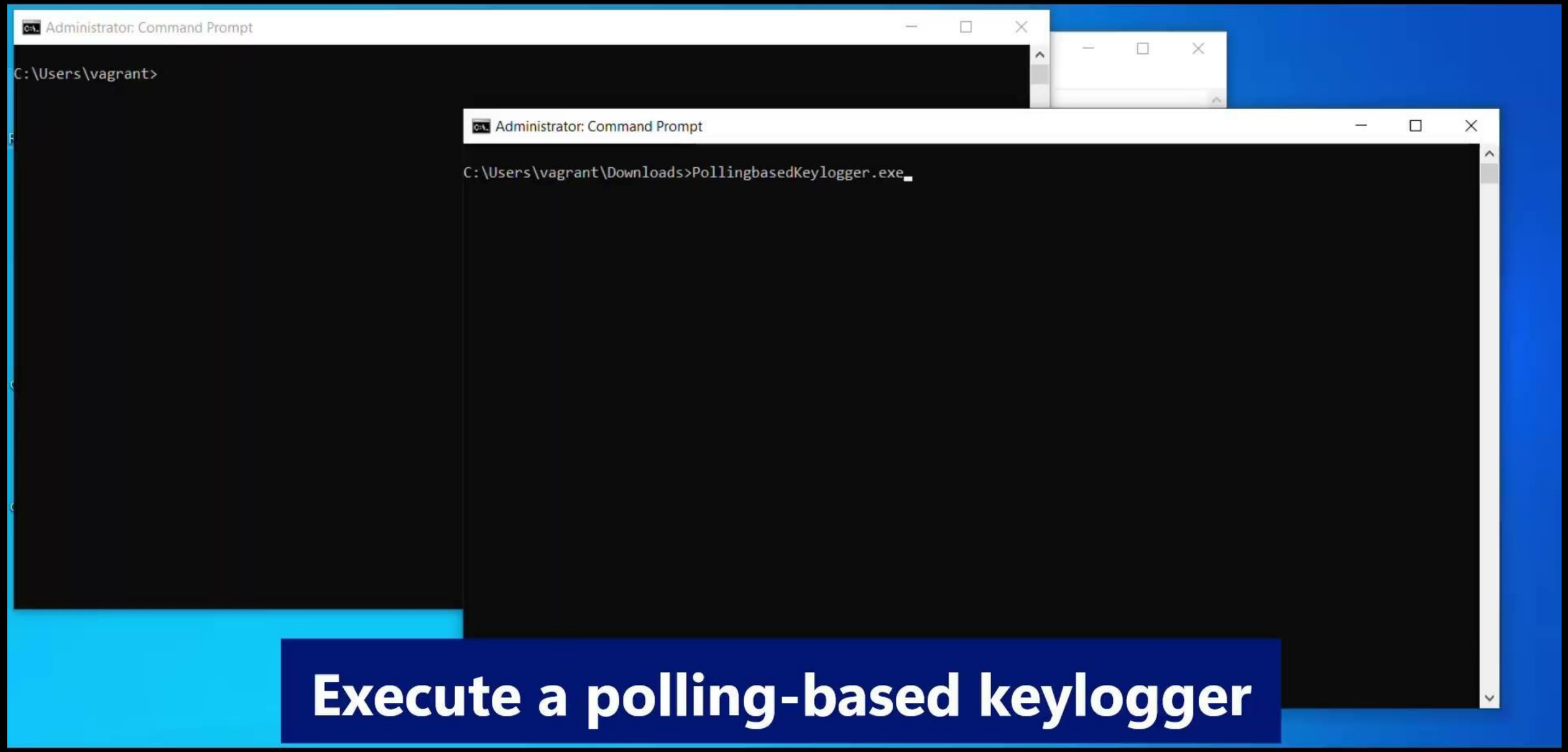
README MIT license

ETW_Win32kAPIMonitor

Introduction

ETW_Win32kAPIMonitor is a Windows-based C++ application that leverages Event Tracing for Windo
(ETW) to monitor and capture specific Win32k API events in real time. The tool focuses on tracking the
following Win32k events which are related to keylogging:

- `RegisterRawInputDevices (Event Id: 1001)`
- `SetWindowsHookEx  (Event Id: 1002)`
- `GetAsyncKeyState (Event Id: 1003)`



Execute a polling-based keylogger

Developing Behavioral Detection Rules

elastic

GetAsyncKeyState (Event ID: 1003)

Useful Fields for Detection

Field Name	Description	Example
MsSinceLastKeyEvent	The elapsed time in milliseconds since the last GetAsyncKeyState event.	**141**
BackgroundCallCount	The total number of GetAsyncKeyState API calls, including unsuccessful calls, since the last successful GetAsyncKeyState event.	**449**

Behavioral Detection Rules for Polling-based Keyloggers

Behavior Detection Rule

GetAsyncKeyState API Call from Suspicious Process (Excerpt of key points)

```
[api where process.Ext.api.name == "GetAsyncKeyState" and
 process.Ext.api.metadata.background_callcount >= 400 and process.Ext.api.metadata.ms_since_last_keyevent >= 100 and
 not (process.executable : "?:\\Windows\\System32\\rundll32.exe" and
      process.command_line : "\"?:\\Windows\\System32\\rundll32.exe\" C:\\Windows\\System32\\LogiLDA.dll,LogiFetch" and
      process.thread.Ext.call_stack_summary : "win32u.dll|winsrvext.dll|ntdll.dll")]
```

https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_getasynckeystate_api_call_from_suspicious_process.toml

💡 **Polling-based keylogger might be present**

Checks whether **BackgroundCallCount >= 400**, which indicates that the GetAsyncKeyState API is being called frequently

SetWindowsHookEx (Event ID: 1002)

Useful Fields for Detection

Field Name	Description	Example
FilterType	Type of hook procedure that will be installed.	13 (*WH_KEYBOARD_LL*)
pstrLib	The DLL that contains the hook procedure.	"C:\Windows\System32\Taskbar.dll"
pfnFilterProc	The memory address of the hooked procedure or function.	2431737462784

Behavioral Detection Rules for Hooking-based Keyloggers

Behavior Detection Rule

Keystrokes Input Capture via SetWindowsHookEx (Excerpt of key points)

```
[api where
  process.Ext.api.name == "SetWindowsHookEx" and process.Ext.api.parameters.hook_type == "WH_KEYBOARD_LL" and
  process.thread.Ext.call_stack_final_user_module.name == "Unknown" and
  not process.Ext.api.parameters.hook_module :
                    ("?:\\Program Files\\*", "?:\\Program Files (x86)\\*", "?:\\Windows\\*")]
```

https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystrokes_input_capture_via_setwindowshookex.toml

💡 **Hooking-based keylogger might be present**

Checks whether a hook procedure for monitoring low-level keyboard input events is installed when SetWindowsHookEx is called

FilterType (hook type) == *"WH_KEYBOARD_LL"*

RegisterRawInputDevices (Event ID: 1001)

Useful Fields for Detection

Field Name	Description	Example
ReturnValue	Return value of the RegisterRawInputDevices API call.	**1**
UsagePage	This parameter indicates the top-level collection (Usage Page) of the device. It is the first member of the RAWINPUTDEVICE structure.	**1 (*HID_USAGE_PAGE_GENERIC*)**
Usage	This parameter indicates the specific device (Usage) within the Usage Page. It is the second member of the RAWINPUTDEVICE structure.	**6 (*HID_USAGE_GENERIC_KEYBOARD*)**
Flags	A mode flag that specifies how to interpret the information provided by UsagePage and Usage. It is the third member of the RAWINPUTDEVICE structure.	**256 (*RIDEV_INPUTSINK*)**
ThreadStartAddress	The thread start address of the thread.	**0x95b7de**

RegisterRawInputDevices (Event ID: 1001)

Useful Fields for Detection

Field Name	Description	Example
cWindows	Number of windows owned by the calling thread.	2
cVisWindows	Number of visible windows owned by the calling thread.	0
ThreadInfoFlags	Thread info flags.	16
ThreadStartAddressMappedModuleName	Name of the module associated with the starting address of a thread.	\Device\HarddiskVolume3\Users\vagrant\keylogger.exe
ThreadStartAddressVadAllocationProtect	The memory protection attributes associated with the starting address of a thread.	128

Behavioral Detection Rules (for Raw Input Keyloggers)

Behavior Detection Rule

Keystroke Input Capture via RegisterRawInputDevices (Excerpt of key points)

```
api where
 process.Ext.api.name == "RegisterRawInputDevices" and not process.code_signature.status : "trusted" and
 process.Ext.api.parameters.usage : ("HID_USAGE_GENERIC_KEYBOARD", "KEYBOARD") and
 process.Ext.api.parameters.flags : "*INPUTSINK*" and process.thread.Ext.call_stack_summary : "?*" and
 process.thread.Ext.call_stack_final_user_module.hash.sha256 != null and process.executable != null and
```

https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystroke_input_capture_via_registerrawinputdevices.toml

💡 **Keyloggers using Raw Input Model might be present**

Checks the arguments of the RegisterRawInputDevices API call to see if the registered device is a keyboard and if the ***RIDEV_INPUTSINK*** flag is set.

commonly used by keyloggers

Behavioral Detection Rule Name	URL
GetAsyncKeyState API Call from Suspicious Process	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_getasynckeystate_api_call_from_suspicious_process.toml
GetAsyncKeyState API Call from Unusual Process	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_getasynckeystate_api_call_from_unusual_process.toml
Keystroke Input Capture via DirectInput	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystroke_input_capture_via_directinput.toml
Keystroke Input Capture via RegisterRawInputDevices	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystroke_input_capture_via_registerrawinputdevices.toml
Keystroke Messages Hooking via SetWindowsHookEx	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystroke_messages_hooking_via_setwindowshookex.toml
Keystrokes Input Capture from a Managed Application	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystrokes_input_capture_from_a_managed_application.toml
Keystrokes Input Capture from a Suspicious Module	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystrokes_input_capture_from_a_suspicious_module.toml
Keystrokes Input Capture from Suspicious CallStack	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystrokes_input_capture_from_suspicious_callstack.toml
Keystrokes Input Capture from Unsigned DLL	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystrokes_input_capture_from_unsigned_dll.toml
Keystrokes Input Capture via SetWindowsHookEx	https://github.com/elastic/protections-artifacts/blob/main/behavior/rules/windows/collection_keystrokes_input_capture_via_setwindowshookex.toml

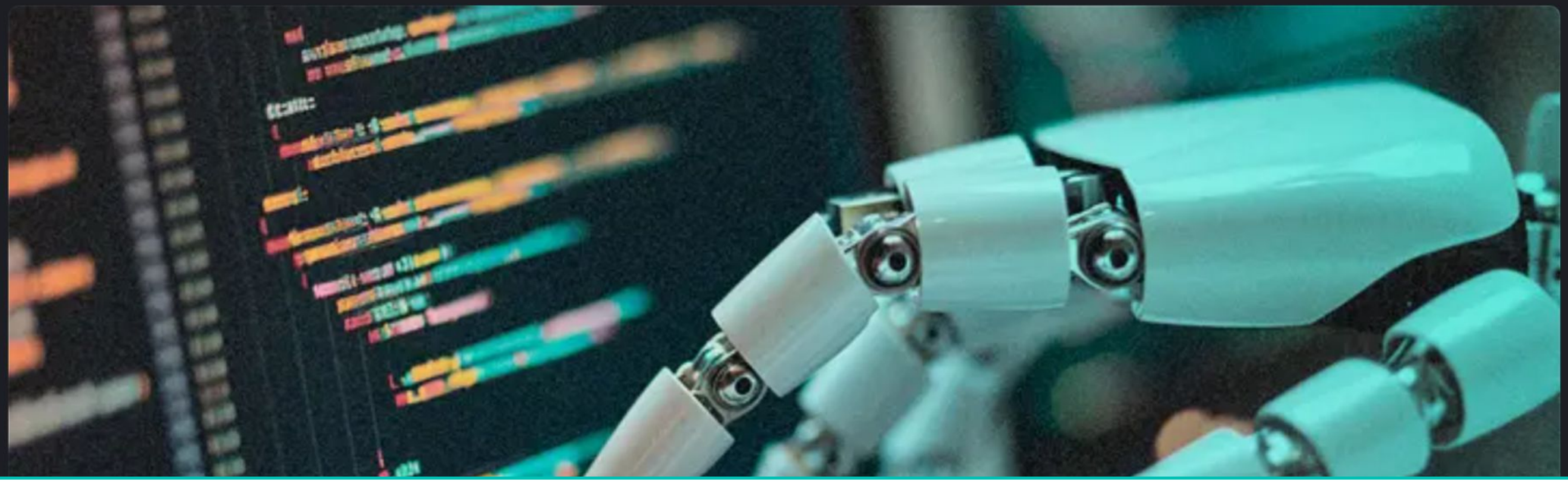
https://www.elastic.co/security-labs/protecting-your-devices-from-information-theft-keylogger-protection

30 MAY 2024 • ASUKA NAJAKIMA

Protecting your devices from information theft

Keylogger detection using Windows API behaviors

14 min read Security operations, Security research, Detection science



About Today's Talk

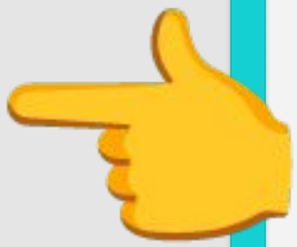
Part A

Detecting Common Types of Keyloggers Through Windows API Monitoring

💡 Sharing my experience of adding a keylogger behavioral detection feature to an EDR

Part B

Hotkey-based Keylogger Detection



Encountering Hotkey-based Keylogging Method

One day, I received a message introducing me to a new keylogging method

jonathanbaror.com

You reposted

Jonathan Bar Or (JBO) 🇮🇱🇺🇸🇺🇦🎗️

@yo_yo_yo_jbo

@AsuNa_jpThank you for the article. Today I also wrote an article about other keylogger methods, let me know your thoughts Asuka!

※ https://x.com/yo_yo_yo_jbo/status/1797778371939893504 (This message was originally written in Japanese, but translated in English)

Encountering Hotkey-based Keylogging Method

NULLCON

https://nullcon.net/review-panel/jonathan-bar-or

Jonathan Bar Or

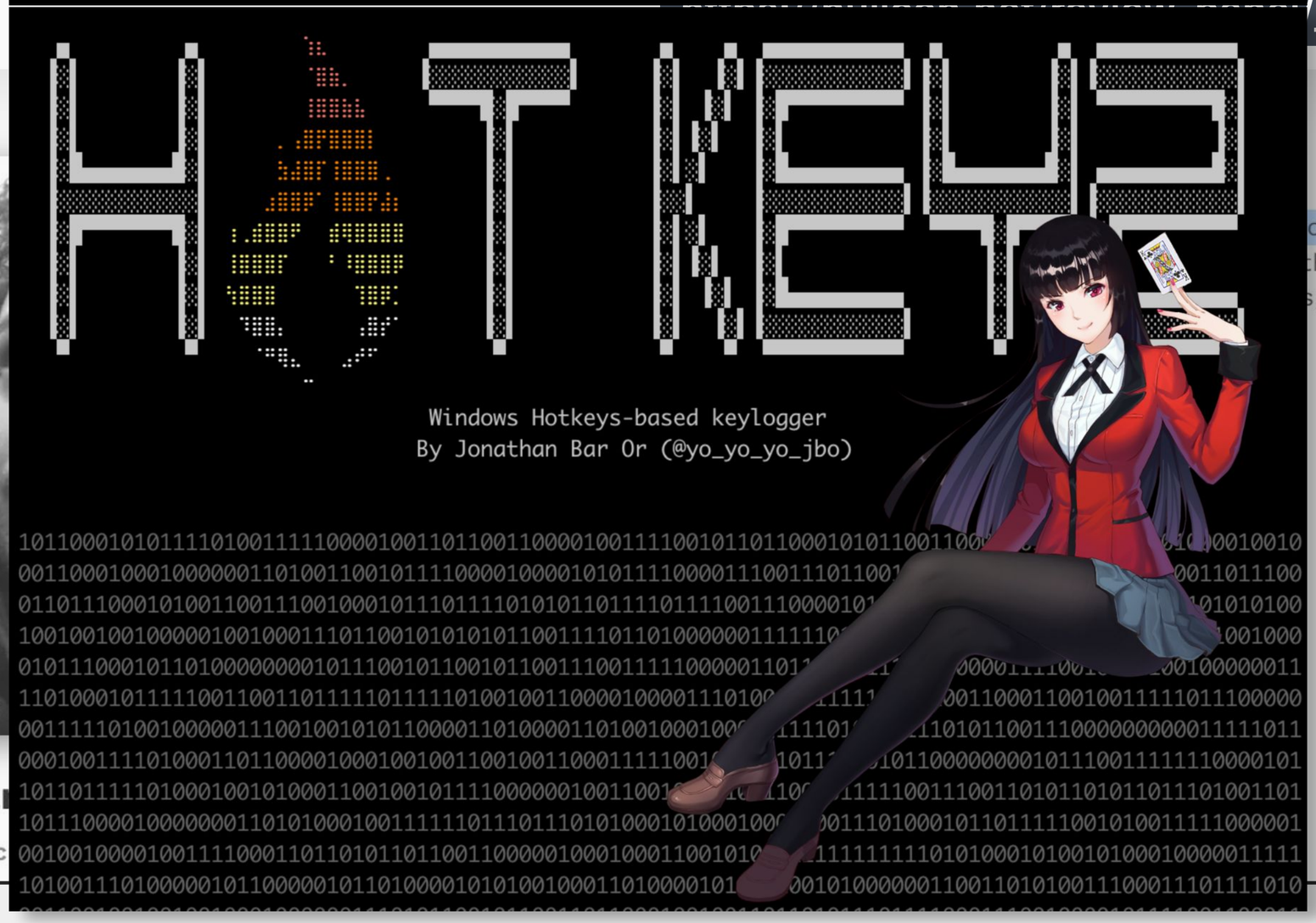
Principal Security Researcher at Microsoft

< REVIEWER BIO />

Jonathan Bar Or ("JBO") is a Principal Security Researcher at Microsoft, working as the Microsoft Defender research architect for cross-platform. Jonathan has rich experience in vulnerability research, exploitation, cryptoanalysis, and offensive security in general.

Encountering Hotkey-based Keylogging Method

https://github.com/yo-yo-yo-jbo/hotkeyz

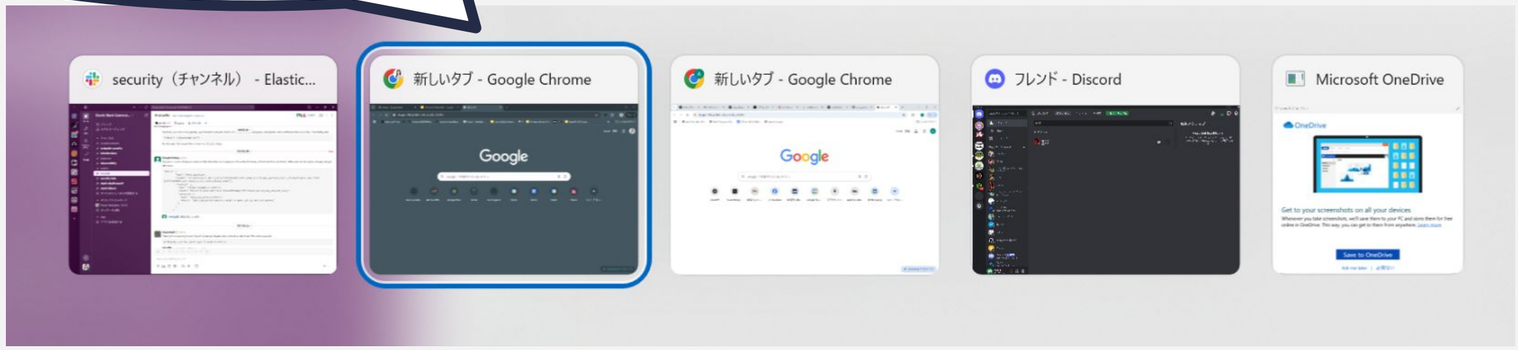


What is a Hotkey?

A type of keyboard shortcut that directly invokes a specific function on a computer by pressing a single key or a combination of keys

ALT + TAB

Task Switching



With the **RegisterHotKey** API, we can set custom hotkeys.

💡 Hotkey-based keyloggers abuse this capability to capture the keystrokes entered by the user.

How Hotkey-based Keyloggers Capture Keystrokes Stealthily [Step 1]

Basically, a keylogger registers each virtual keycode as a system-wide hotkey using the *RegisterHotKey* API (※)

※Modifier keys such as Alt (VK_MENU), Ctrl (VK_CONTROL), Shift (VK_SHIFT), and Win (VK_LWIN/VK_RWIN) cannot be registered as hotkeys on their own. However, combinations of these modifier keys with other keys, such as SHIFT+A, can be registered as hotkeys.

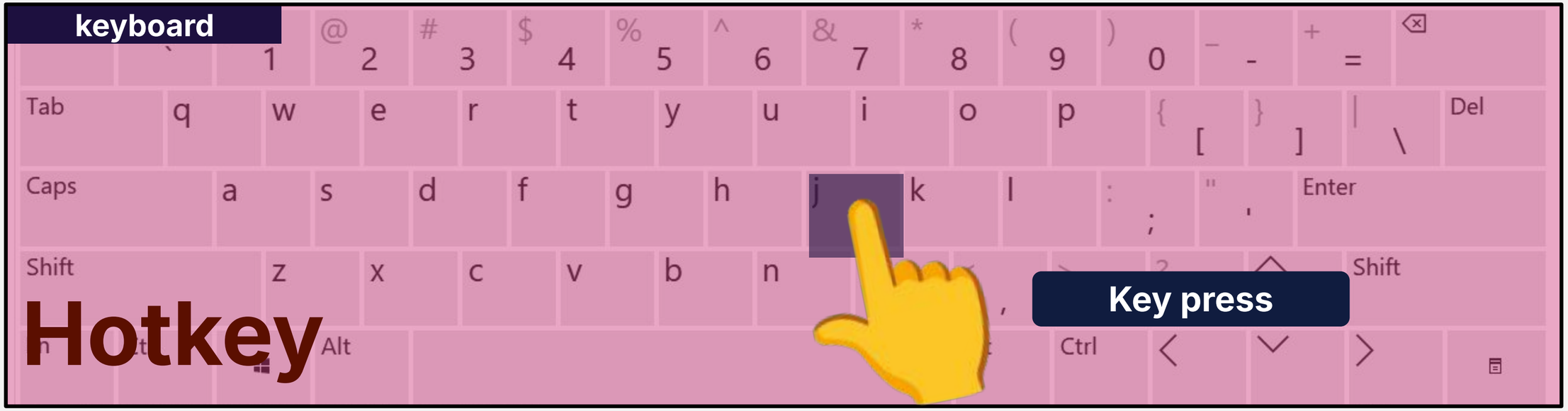
How Hotkey-based Keyloggers Capture Keystrokes Stealthily [Step 1]

Basically, a keylogger registers each virtual keycode as a system-wide hotkey using the *RegisterHotKey* API (※)

※Modifier keys such as Alt (VK_MENU), Ctrl (VK_CONTROL), Shift (VK_SHIFT), and Win (VK_LWIN/VK_RWIN) cannot be registered as hotkeys on their own. However, combinations of these modifier keys with other keys, such as SHIFT+A, can be registered as hotkeys.

How Hotkey-based Keyloggers Capture Keystrokes Stealthily [Step 2]

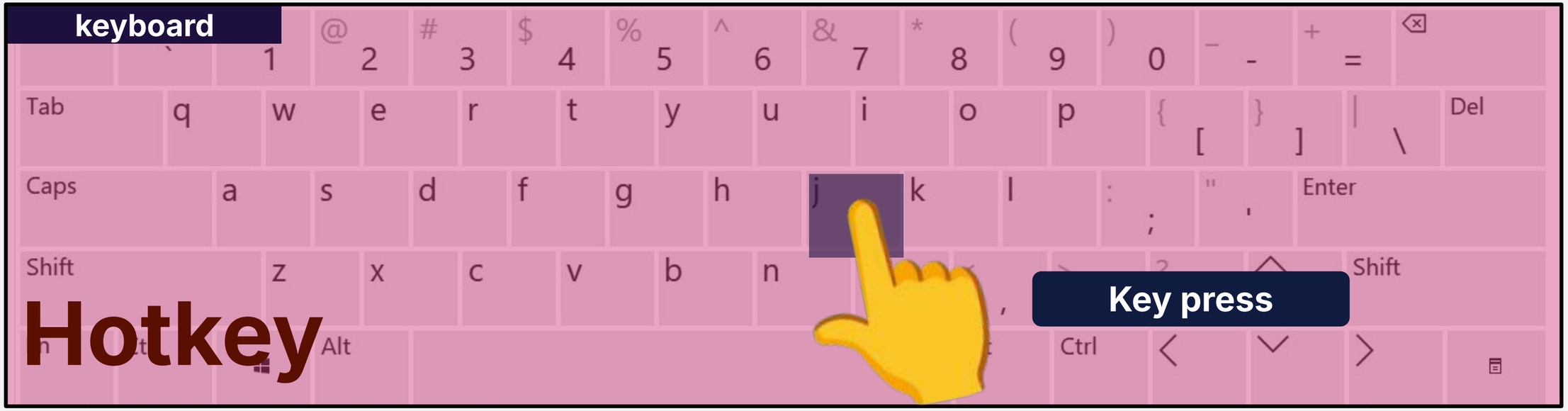
Basically, a keylogger registers each virtual keycode as a system-wide hotkey using the *RegisterHotKey* API (※)



※Modifier keys such as Alt (VK_MENU), Ctrl (VK_CONTROL), Shift (VK_SHIFT), and Win (VK_LWIN/VK_RWIN) cannot be registered as hotkeys on their own. However, combinations of these modifier keys with other keys, such as SHIFT+A, can be registered as hotkeys.

How Hotkey-based Keyloggers Capture Keystrokes Stealthily [Step 2]

Basically, a keylogger registers each virtual keycode as a system-wide hotkey using the *RegisterHotKey* API (※)



Sends a ***WM_HOTKEY*** message, which includes virtual keycode (*VK_J*) to the keylogger's thread message queue

How Hotkey-based Keyloggers Capture Keystrokes Stealthily [Step 3]

Retrieves the *WM_HOTKEY* message using the *PeekMessage* API and extracts the virtual keycode from it

Inside the keylogger's message loop

```
// Get the message in a no
if (!PeekMessageW(&tMsg, NULL, WM_HOTKEY, WM_HOTKEY, PM_REMOVE))
{
        Sleep(POLL_TIME_MILLIS);
        continue;
}
```

Get virtual key code (& log it)

```
// Get the key from the message
cCurrVk = (BYTE)((((DWORD)tMsg.lParam) & 0xFFFF0000) >> 16);
```

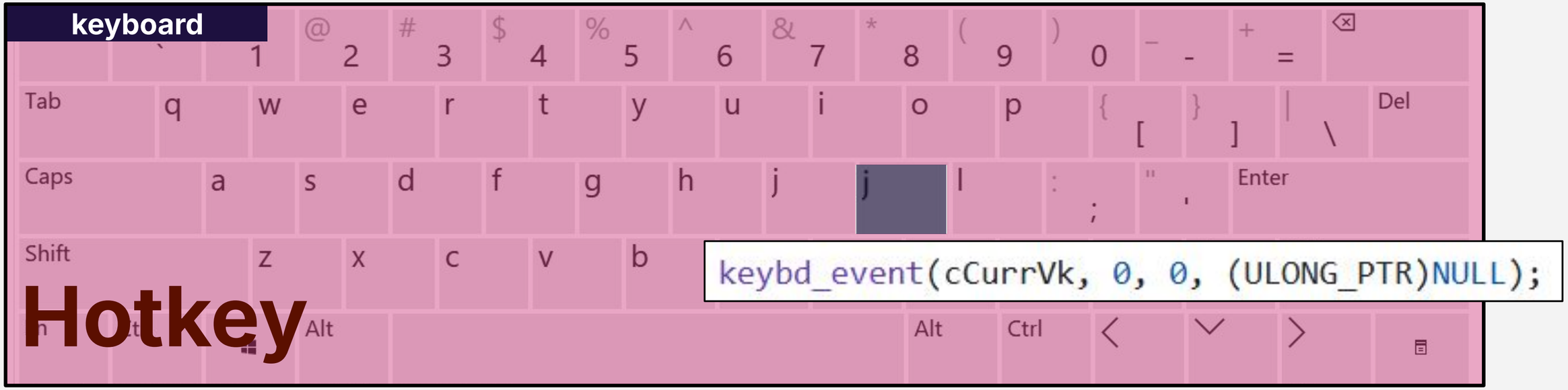
How Hotkey-based Keyloggers Capture Keystrokes Stealthily [Step 4]

Unregister the hotkey using the *UnRegisterHotKey* API

How Hotkey-based Keyloggers Capture Keystrokes Stealthily [Step 5]

Simulate a key press using the *keybd_event* API

💡 **To the user, it appears as if the key was pressed normally**




```
keybd_event(cCurrVk, 0, 0, (ULONG_PTR)NULL);
```

Sends a ***WM_KEYDOWN*** message to an application

How Hotkey-based Keyloggers Capture Keystrokes Stealthily [Step 6]

Re-register the key as a hotkey using the *RegisterHotKey* API, and wait for the further user input (Back to Step 2)

Can ETW Monitor the RegisterHotKey API Calls?

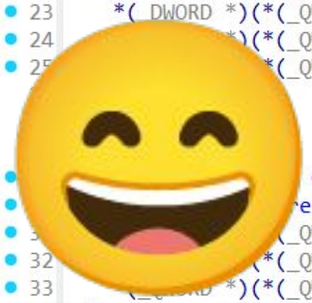
NtUserGetAsyncKeyState (win32k)

IDA View-A | Pseudocode-A | Hex View-1

```
__int64 __fastcall NtUserGetAsyncKeyState(unsigned int a1)
{
  __int64 ThreadWin32Thread; // rdi
  __int16 AsyncKeyState; // bx
  char v5; // [rsp+78h] [rbp+10h] BYREF

  EnterSharedCrit(0LL, 1LL);
  ThreadWin32Thread = W32GetT[illegible]in32Thread(KeGetCurrentThread());
  AsyncKeyState = 0;
  if ( gptiForeground && PsGetCurrentProcessWin32Process() != *((_QWORD *)gptiForegr[illegible]
    EtwTraceGetAsyncKeyState(ThreadWin32Thread);
  if ( (unsigned int)ApiSetEditionIsGetAsyncKeyStateBlocked() )
    goto LABEL_10;
  if ( !(unsigned [illegible]tionIsGpqForegroundAccessibleCurrent(1LL) )
[illegible]
  if ( (unsigned __int8)IsKeyboardDelegationEnabledForThread(ThreadWin32Thread) )
  {
    *(_DWORD *)(*(_QWORD *)(ThreadWin32Thread + 480) + 124LL) = 0;
    [illegible]*(_QWORD *)(ThreadWin32Thread + 480) + 128LL) = 0LL;
    [illegible]*(_QWORD *)(ThreadWin32Thread + 480) + 136LL) = 0LL;
[illegible]
[illegible] GetAsyncKeyState(a1);
[illegible]edLeaf<DLT_ASYNCKEYSTATE>::CLockDomainSharedLeaf<DLT_ASYNCKEYSTAT[illegible]
[illegible]_QWORD *)(ThreadWin32Thread + 480) + 124LL) = *((_DWORD *)gpsi + 1[illegible]
[illegible]*(_QWORD *)(ThreadWin32Thread + 480) + 128LL) = gafAsyncKeyState;
[illegible]*)(*(_QWORD *)(ThreadWin32Thread + 480) + 136LL) = gafAsyncKeyStateRece[illegible]
```

EtwTraceGetAsyncKeyState function which is associated to the ETW event writing



👆 From win32kbase.sys (version: 10.0.19041.5247)

NtUserRegisterHotKey (win32k)

IDA View-A | Pseudocode-A | Hex View-1

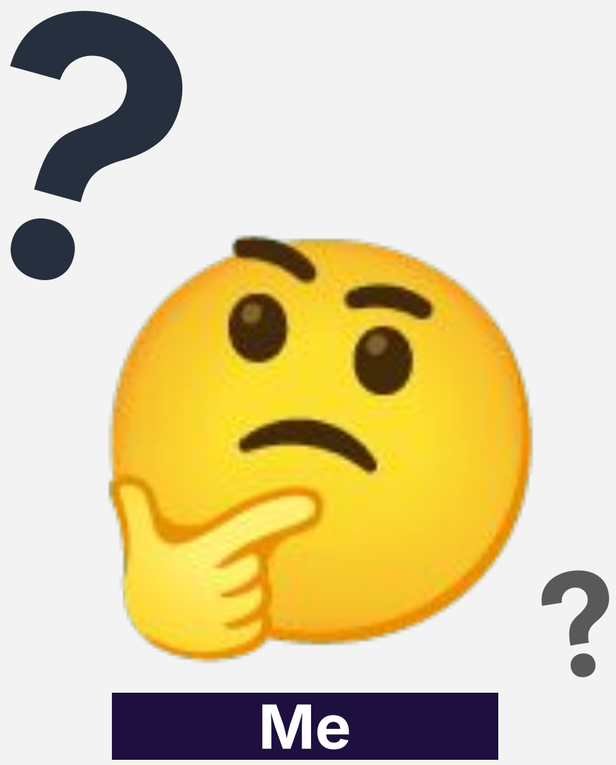
```
__int64 __fastcall NtUserRegisterHotKey(__int64 a1, __int64 a2, int a3, int a4)
{
  int v7; // ebx
  struct tagWND *v8; // rax
  __int64 v10; // rcx
  ULONG_PTR BugCheckParameter2; // [rsp+20h] [rbp-28h]
  char v12[24]; // [rsp+30h] [rbp-18h] BYREF

  EnterCrit(0LL, 1LL);
  UserAtomicCheck::UserAtomicCheck((UserAtomicCheck *)v12);
  v7 = 0;
  if ( (unsigned int)IsImmersiveAppRestricted(*(_QWORD *)(gptiCurrent + 424LL)) )
    goto LABEL_12;
  if ( (a3 & 0xFFFF95F0) != 0 )
  {
    v10 = 1004LL;
[illegible]
[illegible]
[illegible]
  {
LABEL_12:
    v10 = 5LL;
    goto LABEL_13;
  }
  if ( !a4 )
  {
    v10 = 87LL;
LABEL_13:
    UserSetLastError(v10);
```

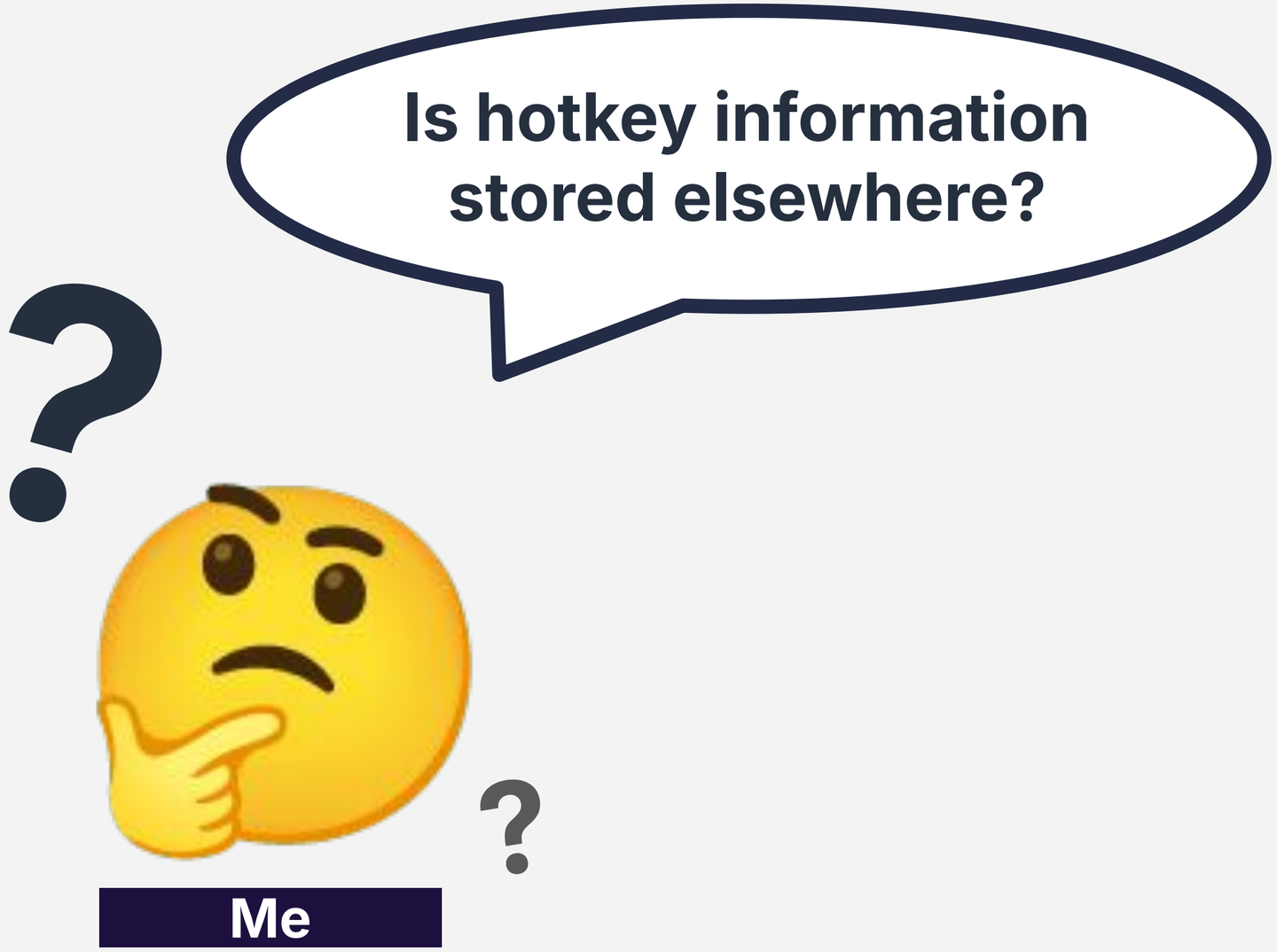
ETW is not monitoring this API..

👆 From win32kfull.sys (version: 10.0.19041.5247)

Are there any detection methods other than ETW?



Are there any detection methods other than ETW?



Are there any detection methods other than ETW?

Is hotkey information stored elsewhere?

Me

keyboard

keyboard

If all main virtual key codes were registered as hotkeys, then 😈 ?

Undocumented Hotkey-table (*gphkHashTable*)

Found a global hash table gphkHashTable, which contains registered hotkey data!

_RegisterHotKey (called by NtUserRegisterHotkey)

```
      v16 = 0;
    }
    ((_WORD *)v15 + 13) = v11 | v16;
    ((_WORD *)v15 + 12) = v10;
    ((_DWORD *)v15 + 7) = BugCheckParameter2;
    v15[1] = a2;
    v15[7] = v15 + 6;
    v15[6] = v15 + 6;
    v17 = *((_BYTE *)v15 + 28) & 0x7F;
    v15[5] = (&gphkHashTable)[v17];
    (&gphkHashTable)[v17] = (struct tagHOTKEY * near *)v15;
    NotifyHotKeyRegistrationChanged((struct tagHOTKEY *const)v15, 0LL, 1);
LABEL_21:
    qword_1C0339AC8 = 0LL;
    return 1LL;
  }
  return 0LL;
```

Undocumented Hotkey-table (*gphkHashTable*)

Found a global hash table gphkHashTable, which contains registered hotkey data!

_RegisterHotKey (called by NtUserRegisterHotkey)

```
      v16 = 0;
    }
    ((_WORD *)v15 + 13) = v11 | v16;
    ((_WORD *)v15 + 12) = v10;
    ((_DWORD *)v15 + 7) = BugCheckParameter2;
    v15[1] = a2;
    v15[7] = v15 + 6;
    v15[6] = v15 + 6;
    v17 = *((_BYTE *)v15 + 28) & 0x7F;
    v15[5] = (&gphkHashTable)[v17];
    (&gphkHashTable)[v17] = (struct tagHOTKEY * near *)v15;
    NotifyHotKeyRegistrationChanged((struct tagHOTKEY *const)v15, 0LL, 1);
LABEL_21:
    qword_1C0339AC8 = 0LL;
    return 1LL;
  }
  return 0LL;
```

The gphkHashTable stores HOT_KEY objects (i.e., Registered Hotkey Info) in a hash table, with their index calculated simply as *virtual keycode % 0x80*

win32kfull.sys (version: 10.0.19041.5247)

Structure of *HOT_KEY* Object

windbg

```
3: kd> dd win32kfull!gphkHashTable
ffffff63`63c06310  44b0e3e0 ffffff1d 40
ffffff63`63c06320  4069c6a0 ffffff1d 4069c1a0 ffffff1d
ffffff63`63c06330  44b0ec00 ffffff1d 44b0dda0 ffffff1d
ffffff63`63c06340  44b0f2e0 ffffff1d 44bc3300 ffffff1d
ffffff63`63c06350  44bc59c0 ffffff1d 44bc61e0 ffffff1d
ffffff63`63c06360  00000000 00000000 00000000 00000000
ffffff63`63c06370  00000000 00000000 44bc5290 ffffff1d
ffffff63`63c06380  00000000 00000000 00000000 00000000
```

```
3: kd> dd FFFFFF1D44BC5290
ffffff1d`44bc5290  40799010 ffffff1d 00000000 00000000
ffffff1d`44bc52a0  00000000 00000000 00000000 0000000d
ffffff1d`44bc52b0  00000003 00000008 44b0eca0 ffffff1d
ffffff1d`44bc52c0  44bc52c0 ffffff1d 44bc52c0 ffffff1d
```

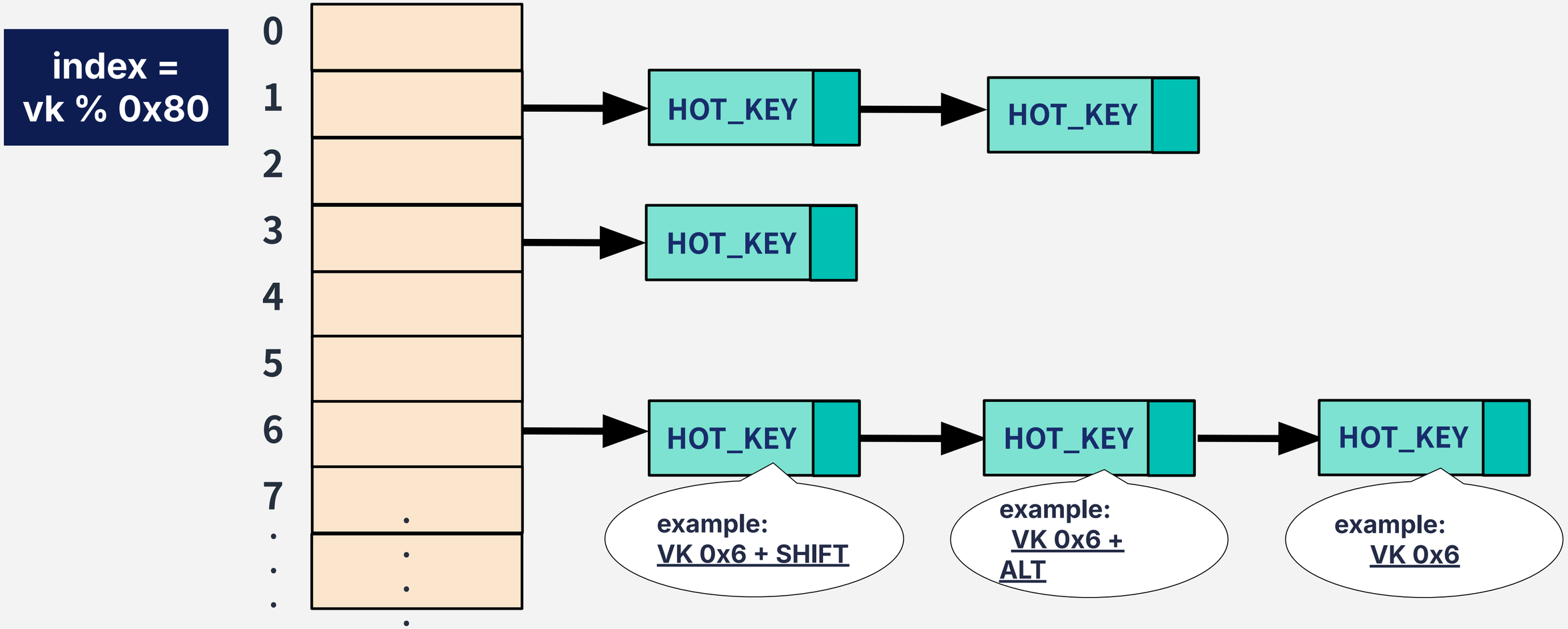
HOT_KEY object for the Enter key with no modifiers (Virtual Key Code: 0xd, ID: 3)

Structure

```
typedef struct _HOT_KEY {
    PTHREADINFO pti,
    PVOID callback,
    PWND pWnd,
    UINT16 fsModifiers1,  // eg. MOD_CONTROL(0x0002)
    UINT16 fsModifiers2,  // eg. MOD_NOREPEAT(0x4000)
    UINT32 vk,            // virtual keycode
    UINT32 id,            // identifier
#ifdef _AMD64_
    PADDING32 pad;
#endif
    struct _HOT_KEY *pNext;  // pointer to the next object
...[skip]...
} HOT_KEY, * PHOT_KEY;
```

Each *HOT_KEY* object contains a <u>virtual key code</u> and <u>modifiers</u>

Structure of gphkHashTable



By scanning all *HOT_KEY* objects in *gphkHashTable*, we can identify all registered hotkeys.

If all of the main keys are registered as hotkeys, it's suspicious!

Challenges in Developing a Detection Tool

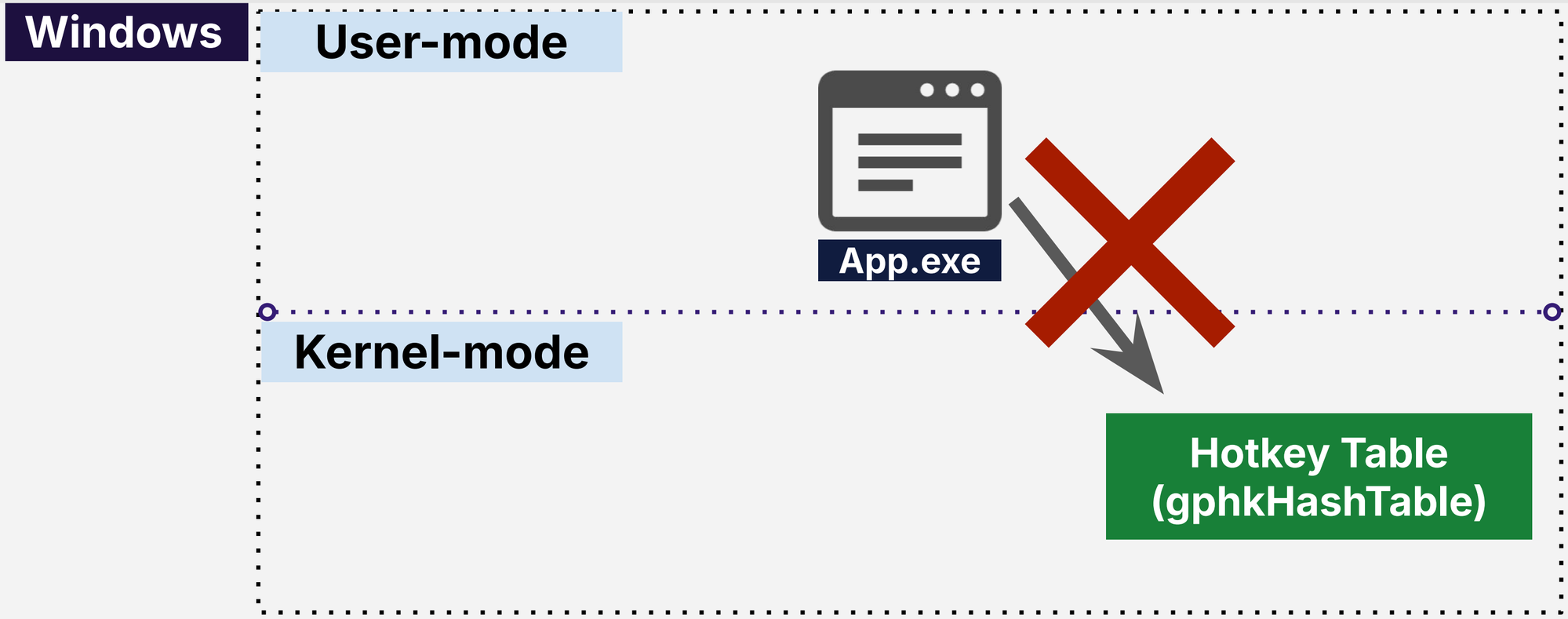
Challenge #1 How to Access Kernel Space?

Challenge #2 How to Find the Address of *gphkHashTable*?

Challenge #3 win32kfull.sys is a *Session Driver*

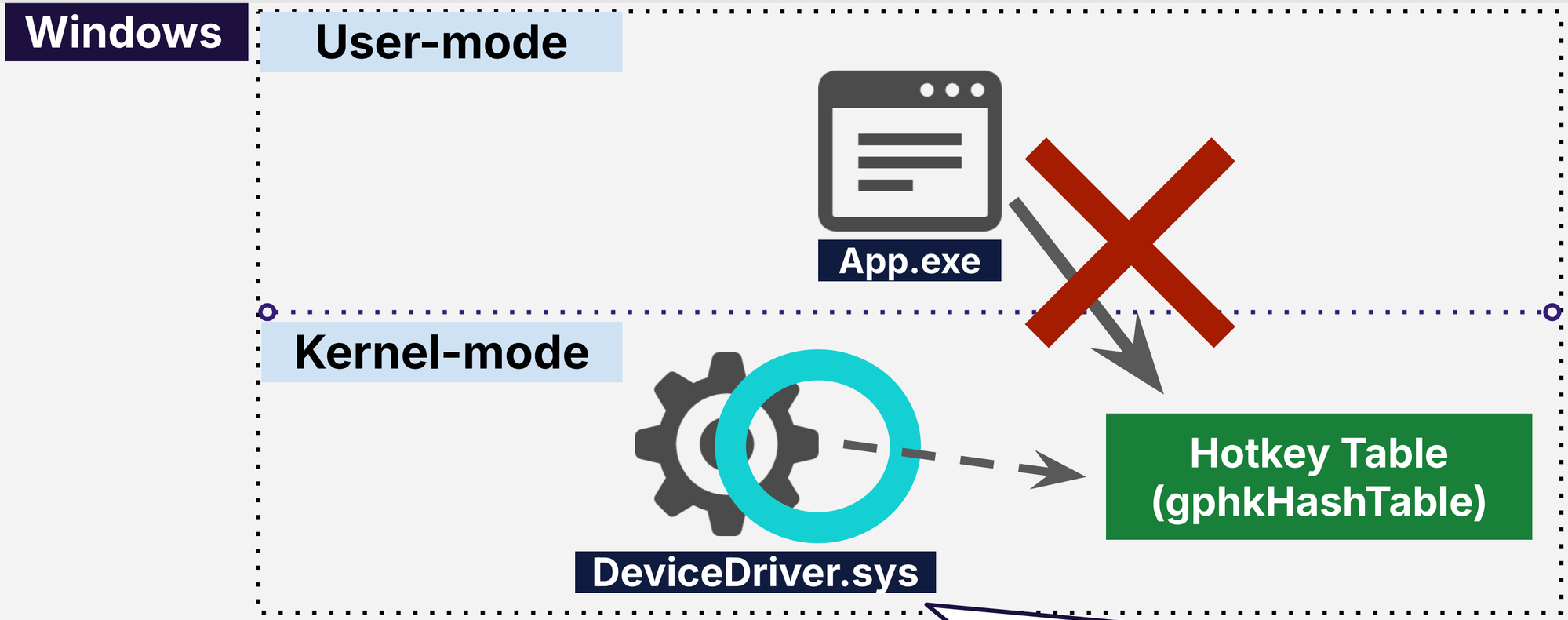
Challenge #1: How to Access Kernel Space?

The hotkey table cannot be directly accessed by a user-mode application since the table resides in kernel space.



Challenge #1: How to Access Kernel Space?

The hotkey table cannot be directly accessed by a user-mode application since the table resides in kernel space.



We need to develop a device driver to access gphkHashTable !

Challenge #2: How to Find the Address of gphkHashTable?

Inside the *IsHotKey* function (win32kfull.sys) which is called from an exported function named EditionIsHotKey

Opcode bytes	Assembly Code
48 89 5C 24 08	mov [rsp+arg_0], rbx
48 89 74 24 10	mov [rsp+arg_8], rsi
57	push rdi
48 83 EC 50	sub rsp, 50h
0F B6 C2	movzx eax, dl
48 8D 1D 1F 8D 26 00	lea rbx, ?gphkHashTable@@3PAPEAUtagHOTKEY@@A
83 E0 7F	and eax, 7Fh
8B FA	mov edi, edx
8B F1	mov esi, ecx
48 8B 1C C3	mov rbx, [rbx+rax*8]

https://www.felixcloutier.com/x86/lea or, Intel® 64 and IA-32 Architectures Software Developer's Manual, Vol. 2A 3-605 for the details on the LEA instruction

Note - win32kfull.sys (version: 10.0.19041.5247)

Challenge #2: How to Find the Address of gphkHashTable?

Inside the *IsHotKey* function (win32kfull.sys) which is called from an exported function named EditionIsHotKey

Opcode bytes	Assembly Code
48 89 5C 24 08	mov [rsp+arg_0], rbx
48 89 74 24 10	mov [rsp+arg_8], rsi
57	push rdi
48 83 EC 50	sub rsp, 50h
0F B6 C2	movzx eax, dl
48 8D 1D 1F 8D 26 00	lea rbx, ?gphkHashTable@@3PAPEAUtagHOTKEY@@A
83 E0 7F	and eax, 7Fh
8B FA	mov edi, edx
8B F1	
48 8B 1C C3	

0x48 0x8D 0x1D ["32 bit offset"];

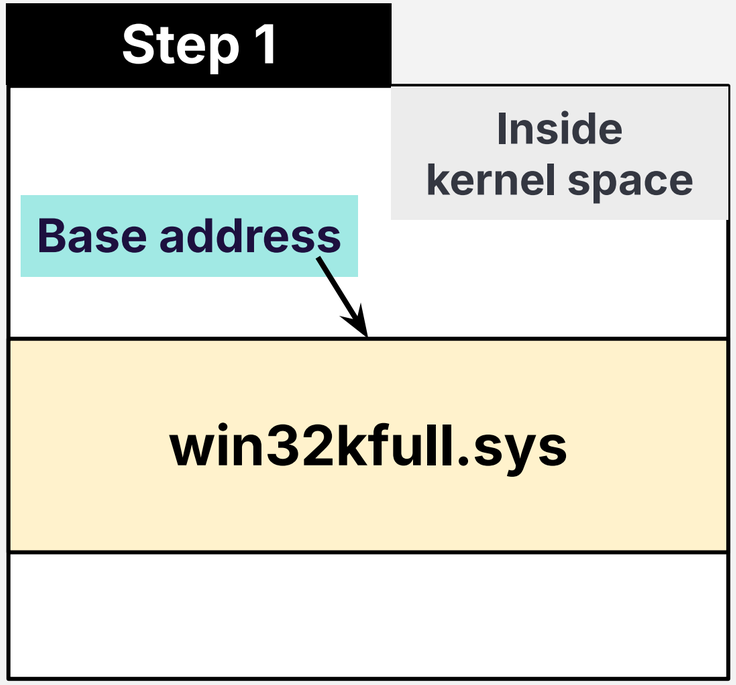
Find this pattern and determine the gphkHashTable address

https://www.felixcloutier.com/x86/lea or, Intel® 64 and IA-32 Architectures Software Developer's Manual, Vol. 2A 3-605 for the details on the LEA instruction

Note - win32kfull.sys (version: 10.0.19041.5247)

Challenge #2: How to Find the Address of gphkHashTable?

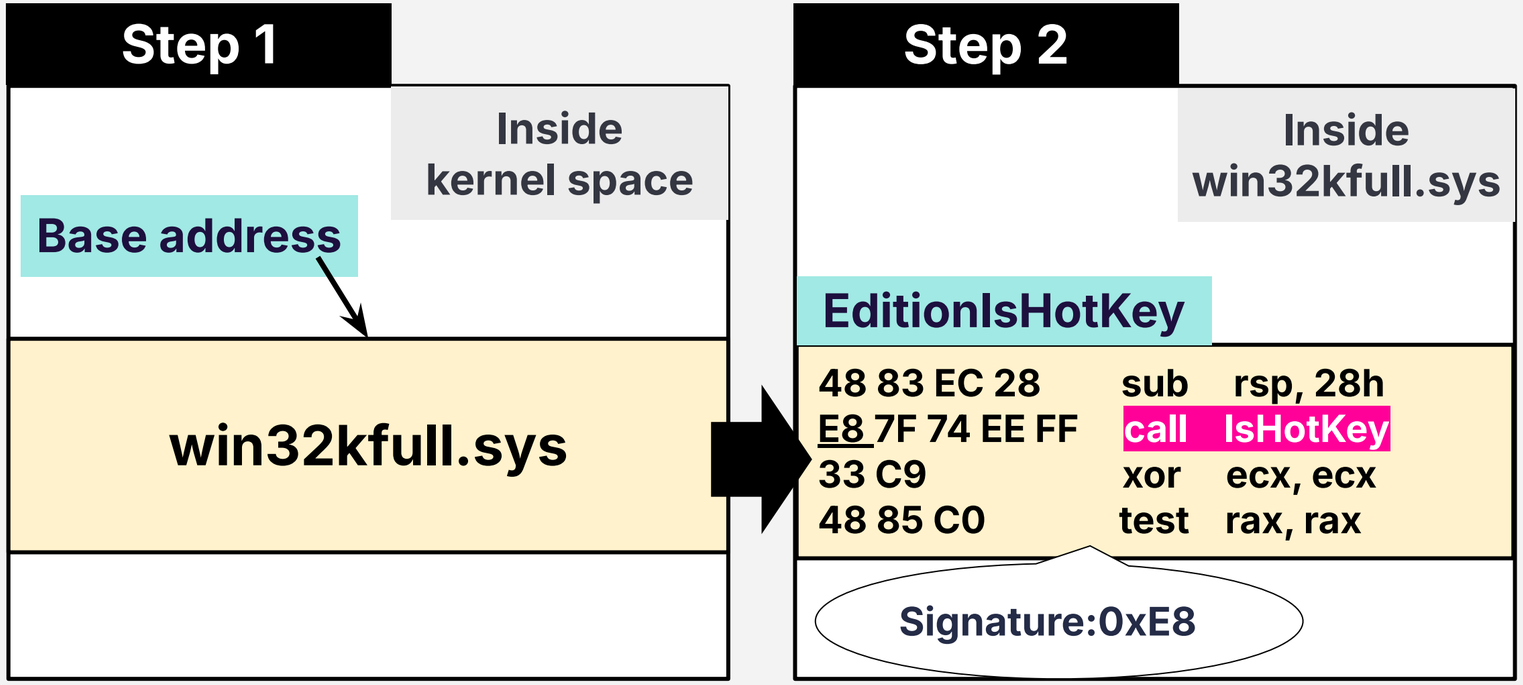
Overview of How to Find the gphkHashTable Address



Resolve the base address of win32kfull.sys using PsLoadedModuleList API

Challenge #2: How to Find the Address of gphkHashTable?

Overview of How to Find the gphkHashTable Address



Resolve the base address of win32kfull.sys using PsLoadedModuleList API

Resolve the address of the EditionIsHotKey function using RtlFindExportedRoutineByName, then find the address of IsHotKey.

Challenge #2: How to Find the Address of gphkHashTable?

Overview of How to Find the gphkHashTable Address

Step 1

Inside kernel space

Base address

win32kfull.sys

Resolve the base address of win32kfull.sys using PsLoadedModuleList API

Step 2

Inside win32kfull.sys

EditionIsHotKey

```
48 83 EC 28      sub   rsp, 28h
E8 7F 74 EE FF   call  IsHotKey
33 C9            xor   ecx, ecx
48 85 C0         test  rax, rax
```

Signature:0xE8

Resolve the address of the EditionIsHotKey function using RtlFindExportedRoutineByName, then find the address of IsHotKey.

Step 3

Inside win32kfull.sys

IsHotKey

```
48 83 EC 50                sub   rsp, 50h
0F B6 C2                   movzx eax, dl
48 8D 1D 1F 8D 26 00       lea   rbx, gphkHashTable
83 E0 7F                   and   eax, 7Fh
```

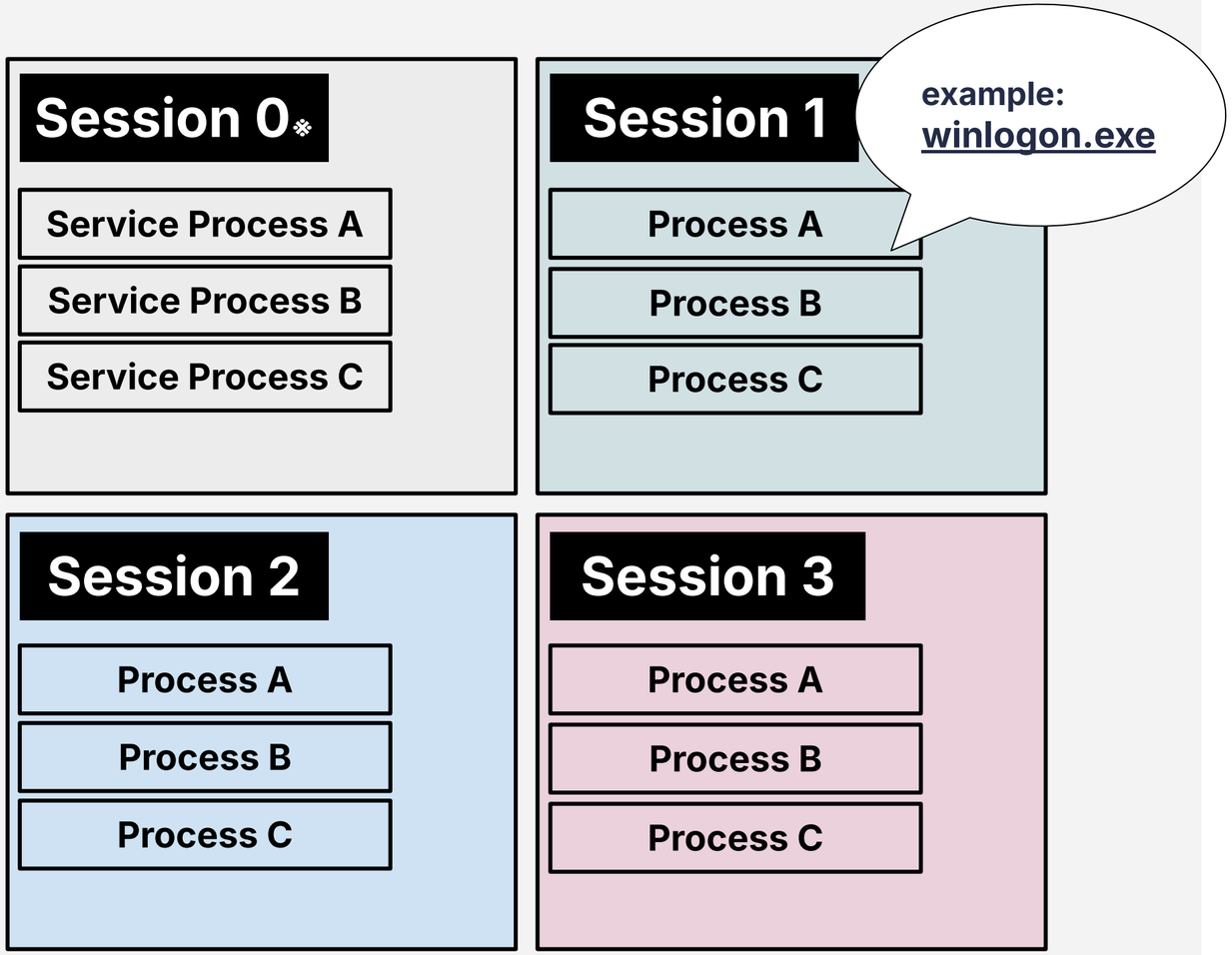
Signature:
0x48, 0x8d, 0x1d

Find the address of gphkHashTable

Challenge #3: win32kfull.sys is a *Session Driver*

What is a Session? (Quick Summary)

❖ In Windows, each logged-in user is assigned a separate session (starting from session 1), with a dedicated desktop environment.



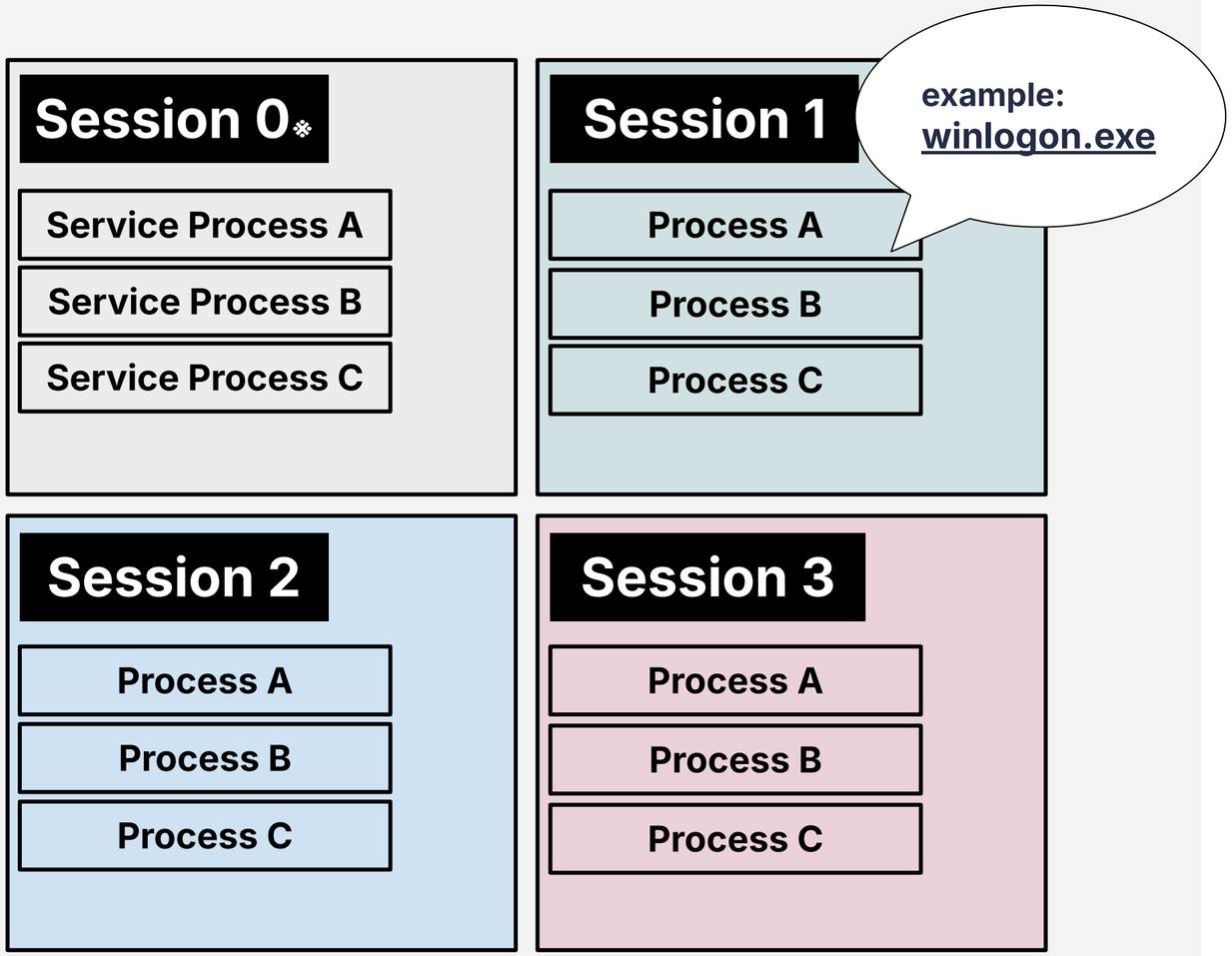
※ Session 0 is dedicated exclusively to service processes.

Challenge #3: win32kfull.sys is a *Session Driver*

What is a Session? (Quick Summary)

- ❖ **In Windows, each logged-in user is assigned a separate session (starting from session 1), with a dedicated desktop environment.**

- ❖ **Kernel data that must be managed separately for each session, including win32k drivers data (such as keyboard input), is stored in an isolated kernel memory area called *session space*.**
  - ✓ This ensures that each user's screen and input remain separate and isolated.



※ Session 0 is dedicated exclusively to service processes.

Hotkey info registered in Session 1 can only be accessed from within that session.

Challenge #3: win32kfull.sys is a *Session Driver*

The *KeStackAttachProcess* API allows the current thread to temporarily attach to the address space of a specified process

Example Code

```
KAPC_STATE apc;
PEPROCESS winlogon;
UNICODE_STRING processName;

RtlInitUnicodeString(&processName, L"winlogon.exe");
HANDLE procId = GetPidFromProcessName(processName);
NTSTATUS status = PsLookupProcessByProcessId(procId, &winlogon);
KeStackAttachProcess(winlogon, &apc);

KeUnstackDetachProcess(&apc);
ObDereferenceObject(winlogon);
```

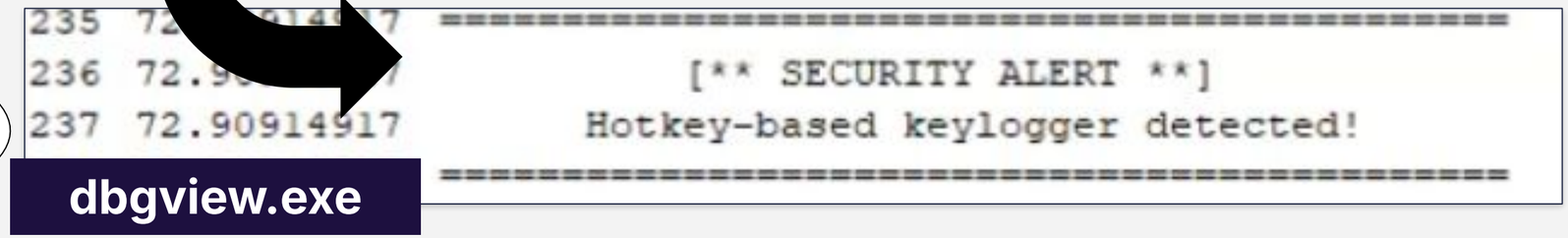
Session 1
winlogon.exe
(if only one user is logged in)

Thread is attached to the process

~ Can access gphkHashTable as the attached process context (session 1 context) ~

Detection Logic

If all alphanumeric keys are registered as hotkeys, an alert will be raised, as it is likely that a hotkey-based keylogger is present.



It is also easy to check all VK + modifiers hotkeys

Tool Release: Hotkey-based Keylogger Detector

https://github.com/AsuNa-jp/HotkeybasedKeyloggerDetector

AsuNa-jp fixed the index — eca1e62 · yesterday — 11 Commits

HotkeybasedKeyloggerDetector	fixed the index	yesterday
HotkeybasedKeyloggerDetector.sln	Initial commit	last week
LICENSE	Initial commit	last week
README.md	Update README.md	last week

README MIT license

Hotkey-based Keylogger Detector

Introduction

Hotkey-based Keylogger Detector is a Windows kernel-mode driver designed to detect hotkey-based keyloggers that hijack and record keystrokes using system hotkeys (RegisterHotKey API). To detect such keyloggers, the driver scans the win32kfull.sys module, resolves the address of the global hotkey table

DESKTOP-RNM57KT (local)

Options Computer Help

```
Debug Print
[+] hk->id: 24 hk->vk: 48
[+] hk->id: 25 hk->vk: 49
[+] hk->id: 26 hk->vk: 4a
[+] hk->id: 27 hk->vk: 4b
[+] hk->id: 28 hk->vk: 4c
[+] hk->id: 29 hk->vk: 4d
[+] hk->id: 30 hk->vk: 4e
[+] hk->id: 31 hk->vk: 4f
[+] hk->id: 32 hk->vk: 50
[+] hk->id: 33 hk->vk: 51
[+] hk->id: 34 hk->vk: 52
[+] hk->id: 35 hk->vk: 53
[+] hk->id: 36 hk->vk: 54
[+] hk->id: 37 hk->vk: 55
[+] hk->id: 38 hk->vk: 56
[+] hk->id: 39 hk->vk: 57
[+] hk->id: 40 hk->vk: 58
[+] hk->id: 41 hk->vk: 59
[+] hk->id: 42 hk->vk: 5a
==========================================
          [** SECURITY ALERT **]
      Hotkey-based keylogger detected!
==========================================
238  72.90914917
```

DEMO TIME!

elastic

Administrator: Command Prompt

```
C:\Users\vagrant\hotkeyz\x64\Release>dir
 Volume in drive C has no label.
 Volume Serial Number is FC34-A2E4

 Directory of C:\Users\vagrant\hotkeyz\x64\Release

02/12/2025  08:45 AM    <DIR>          .
02/12/2025  08:45 AM    <DIR>          ..
02/12/2025  08:36 AM            12,288 Hotkeyz.exe
02/12/2025  08:36 AM           577,536 Hotkeyz.pdb
02/12/2025  08:47 AM                 7 test.txt
               3 File(s)        589,831 bytes
               2 Dir(s)  12,151,107,584 bytes free

C:\Users\vagrant\hotkeyz\x64\Release>
```

Administrator: Command Prompt

```
C:\Users\vagrant\HotkeybasedKeyloggerDetector\Debug>sc create HotkeybasedKeyloggerDetector type=kernel start=demand binP
ath="C:\Users\vagrant\HotkeybasedKeyloggerDetector\Debug\HotkeybasedKeyloggerDetector.sys"
```

Test Mode
Windows 10 Pro
Build 19041.vb_release.191206-1406

धन्यवाद

Thank You !

elastic | The Search AI Company

E-mail	asuka.nakajima@elastic.co
X (Twitter)	@AsuNa_jp
Website	https://www.kun0ichi.net
Github	https://github.com/AsuNa-jp