Malware Analysis

MALWARE COVERT LAUNCHING

COMPILED BY: DHARMESH DAVE | ASST. PROF. | NATIONAL FORENSIC SCIENCES UNIVERSITY

Process Injection

- The most popular covert launching technique is process injection.
- This technique **injects code** into another running process, and that process unwittingly executes the malicious code.
- The Malware authors use process injection for:
  - Malicious behavior of their code
  - Try to bypass host based firewalls
  - Try to bypass process specific security mechanism
- Certain Windows API calls are commonly used for process injection:
- **VirtualAllocEx** and **WriteProcessMemory**

Process Injection

- The ***VirtualAllocEx*** function can be used to allocate space in an external process’s memory.
- ***WriteProcessMemory*** can be used to write data to that allocated space.
- **DLL Injection:**
- A form of process injection where a remote process is forced to load a malicious DLL - is the most commonly used covert loading technique.
- DLL injection works by injecting code into a remote process that calls ***LoadLibrary***, thereby forcing a DLL to be loaded in the context of that process.

Dll Injection

- Once the compromised process loads the malicious DLL, the OS automatically calls the DLL's DllMain function, which is defined by the author of the DLL.
- **This function contains the malicious code** and has as much access to the system as the process in which it is running.
- Everything they do will appear to originate from the compromised process.
- In order to inject the malicious DLL into a host program, the launcher malware must first **obtain a handle to the victim process**.
- The most common way is to use the Windows API calls ***CreateToolhelp32Snapshot, Process32First***, and ***Process32Next*** to search the process list for the injection target.

Dll Injection

- Once the target is found, the launcher retrieves the process identifier (PID) of the target process and then uses it to obtain the **handle** via a call to ***OpenProcess***.
- Using the **handle**, ***VirtualAllocEx*** and ***WriteProcessMemory*** then allocate space and write the name of the malicious DLL into the victim process.
- Next, ***GetProcAddress*** is used to get the address to ***LoadLibrary***.
- Finally, the function ***CreateRemoteThread*** is commonly used for DLL injection to allow the launcher malware to create and execute a new thread in a remote process.
- When ***CreateRemoteThread*** is used, it is passed **three** important parameters: the **process handle** (hProcess) obtained with ***OpenProcess***, along with the **starting point** of the injected thread (lpStartAddress) and an **argument for that thread** (lpParameter).

Direct Injection

- Like DLL injection, direct injection involves allocating and inserting code into the memory space of a remote process.
- Direct injection uses many of the same Windows API calls as DLL injection.
- The difference is that instead of writing a separate DLL and forcing the remote process to load it, direct injection malware injects the malicious code directly into the remote process.
- This technique can be used to **inject compiled code**, but more often, it's used **to inject shellcode**.
- Three functions are commonly found in cases of direct injection: *VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread*.

Process Replacement

- Rather than inject code into a host program, some malware uses a method known as **process replacement** to overwrite the memory space of a running process with a malicious executable.
- Process replacement is used when a malware author wants to disguise malware as a legitimate process, without the risk of crashing a process through the use of process injection.
- This technique provides the malware with the same privileges as the process it is replacing.
- **For example**, if a piece of malware were to perform a process-replacement attack on *svchost.exe*, the user would see a process name svchost.exe running from C:\Windows\System32 and probably think nothing of it.

Process Replacement

- Key to process replacement is **creating a process in a suspended state**. This means that the process will be loaded into memory, but the primary thread of the process is suspended.
- The program will not do anything until an external program resumes the primary thread, causing the program to start running.
- suspended state can be created by passing *CREATE_SUSPENDED (0x4)* as the ***dwCreationFlags*** parameter when performing the call to **CreateProcess**.

Process Replacement

- Once the process is created, the next step is to replace the victim process's memory with the malicious executable, typically using ***ZwUnmapViewOfSection*** to release all memory pointed to by a section passed as a parameter.
- After the memory is unmapped, the loader performs ***VirtualAllocEx* to allocate new memory** for the malware, and uses ***WriteProcessMemory* to write each of the malware** sections to the victim process space
- In the final step, malicious code can run by calling ***SetThreadContext*** to set the entry point to point to the malicious code.
- **Finally, *ResumeThread*** is called to initiate the malware, which has now replaced the victim process.

Hooking

- A hook is a mechanism by which an application can **intercept events, such as messages, mouse actions, and keystrokes**. A function that intercepts a particular type of event is known as a **hook procedure**. A hook procedure can act on each event it receives, and then modify or discard the event.
- The following some example uses for hooks:
  - Monitor messages for debugging purposes
  - Provide support for recording and playback of macros
  - Provide support for a help key (F1)
  - Simulate mouse and keyboard input
  - Implement a computer-based training (CBT) application.

Hooking

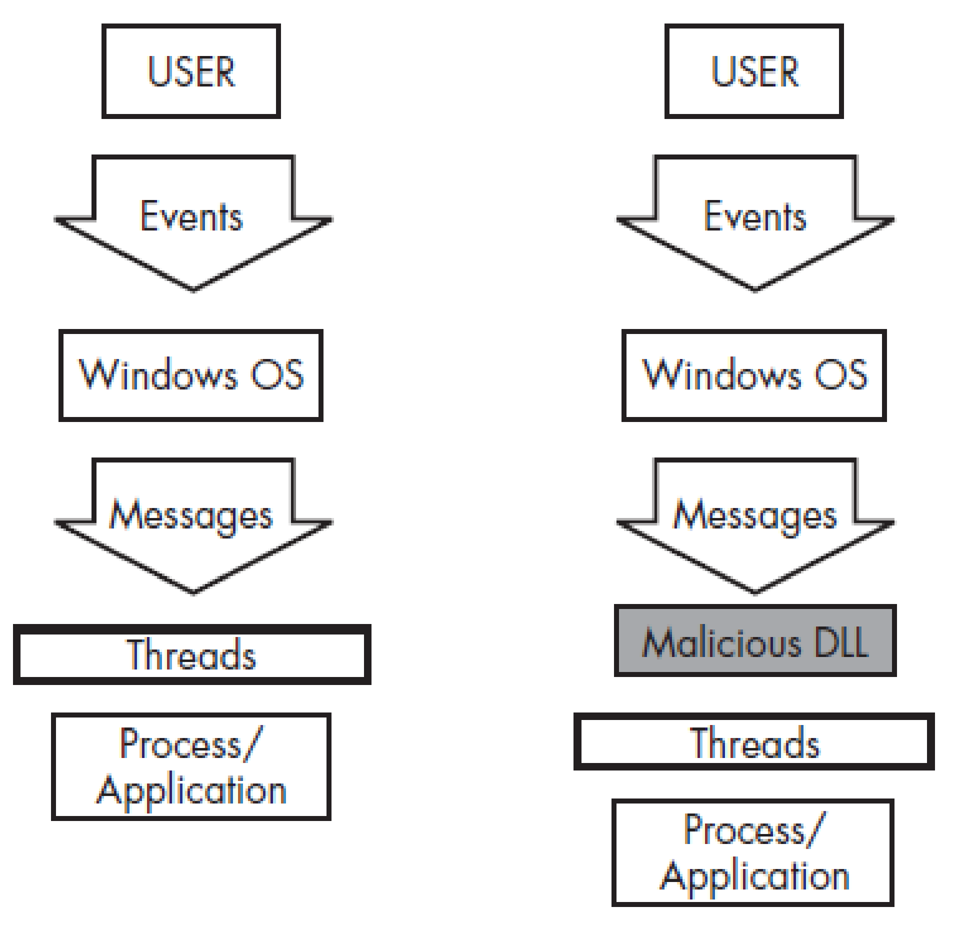


Figure 12-3: Event and message flow in Windows with and without hook injection

Hooking Example (WH_CBT)

- The system calls a **WH_CBT** hook procedure before activating, creating, destroying, minimizing, maximizing, moving, or sizing a window; before completing a system command; before removing a mouse or keyboard event from the system message queue; before setting the input focus; or before synchronizing with the system message queue.
- The value the hook procedure returns determines whether the system **allows or prevents** one of these operations. The WH_CBT hook is intended primarily for computer-based training (CBT) applications.

Hook Installation & Relelase

- The installing application must have the **handle** to the DLL module before it can install the hook procedure.
- To retrieve a handle to the DLL module, call the **LoadLibrary** function with the name of the DLL. After you have obtained the handle, you can call the **GetProcAddress** function to retrieve a pointer to the hook procedure.
- Finally, use **SetWindowsHookEx** to install the hook procedure address in the appropriate hook chain.
- You can release a thread-specific hook procedure (remove its address from the hook chain) by calling the **UnhookWindowsHookEx** function, specifying the handle to the hook procedure to release.

Hook Injection

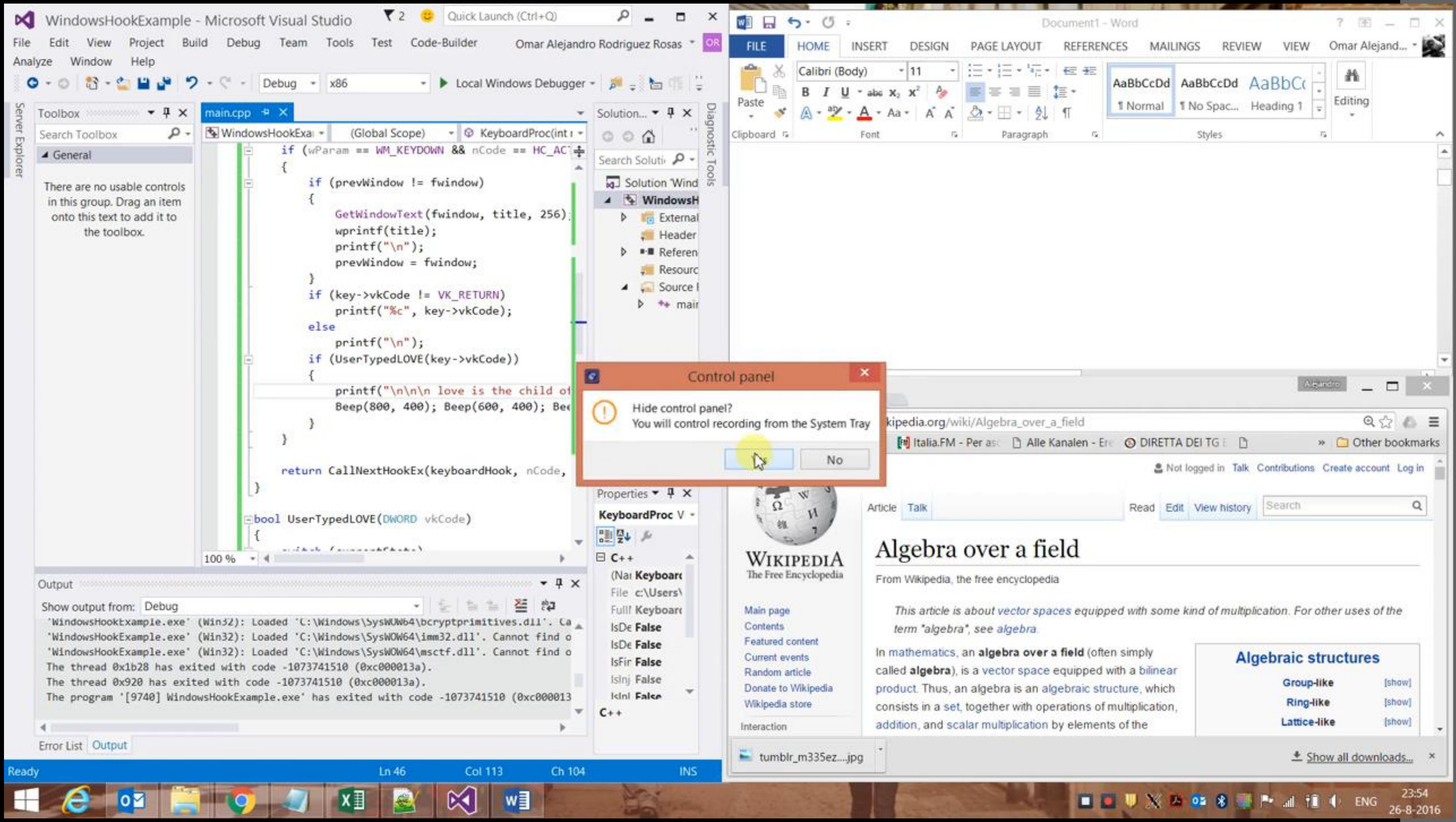
- A hook is a point in the system message-handling mechanism where an application can install a subroutine to monitor the message traffic in the system and process certain types of messages before they reach the target window procedure.
- Using **SetWindowsHookEx**:
- The principal function call used to perform remote Windows hooking is SetWindowsHookEx, which has the following parameters:
- HHOOK SetWindowsHookExA(
- [in] int **idHook**,
- [in] HOOKPROC **lpfn**,
- [in] HINSTANCE **hmod**,
- [in] DWORD **dwThreadId**);

SetWindowsHookEx

Params	Description
idHook	Specifies the **type of hook** procedure to install. (i.e. WH_CBT)
lpfn	**Pointer** to the hook procedure.
hMod	For high-level hooks, identifies the **handle to the DLL** containing the hook procedure defined by lpfn. For low-level hooks, this identifies the **local module** in which the lpfn procedure is defined.
dwThreadId	**Specifies the identifier of the thread** with which the hook procedure is to be associated. If this **parameter is zero**, the hook procedure is associated with **all existing threads running** in the same desktop as the calling thread. This must be set to zero for low-level hooks.

Type	Val	Type	Val	Type	Val	Type	Val
WH_CALLWNDPROC	4	WH_CALLWNDPROCRET	12	WH_CBT	5	WH_KEYBOARD	2

Keyboard Hooking



```
    if (wParam == WM_KEYDOWN && nCode == HC_AC
    {
        if (prevWindow != fwindow)
        {
            GetWindowText(fwindow, title, 256);
            wprintf(title);
            printf("\n");
            prevWindow = fwindow;
        }
        if (key->vkCode != VK_RETURN)
            printf("%c", key->vkCode);
        else
            printf("\n");
        if (UserTypedLOVE(key->vkCode))
        {
            printf("\n\n\n love is the child of
            Beep(800, 400); Beep(600, 400); Bee
        }
    }

    return CallNextHookEx(keyboardHook, nCode,
}

bool UserTypedLOVE(DWORD vkCode)
{
```

Show output from: Debug

```
'WindowsHookExample.exe' (Win32): Loaded 'C:\Windows\SysWOW64\bcryptprimitives.dll'. Ca
'WindowsHookExample.exe' (Win32): Loaded 'C:\Windows\SysWOW64\imm32.dll'. Cannot find o
'WindowsHookExample.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msctf.dll'. Cannot find o
The thread 0x1b28 has exited with code -1073741510 (0xc000013a).
The thread 0x920 has exited with code -1073741510 (0xc000013a).
The program '[9740] WindowsHookExample.exe' has exited with code -1073741510 (0xc000013
```

Control panel

Hide control panel?
You will control recording from the System Tray

No

Algebra over a field

From Wikipedia, the free encyclopedia

This article is about vector spaces equipped with some kind of multiplication. For other uses of the term "algebra", see algebra.

In mathematics, an **algebra over a field** (often simply called **algebra**), is a vector space equipped with a bilinear product. Thus, an algebra is an algebraic structure, which consists in a set, together with operations of multiplication, addition, and scalar multiplication by elements of the

Thread Targeting with Hook

```
00401100 push esi
00401101 push edi
00401102 push offset LibFileName ; "hook.dll"
00401107 call LoadLibraryA
0040110D mov esi, eax
0040110F push offset ProcName ; "MalwareProc"
00401114 push esi ; hModule
00401115 call GetProcAddress
0040111B mov edi, eax
0040111D call GetNotepadThreadId
00401122 push eax ; dwThreadId
00401123 push esi ; hmod
00401124 push edi ; lpfn
00401125 push WH_CBT ; idHook
00401127 call SetWindowsHookExA
```

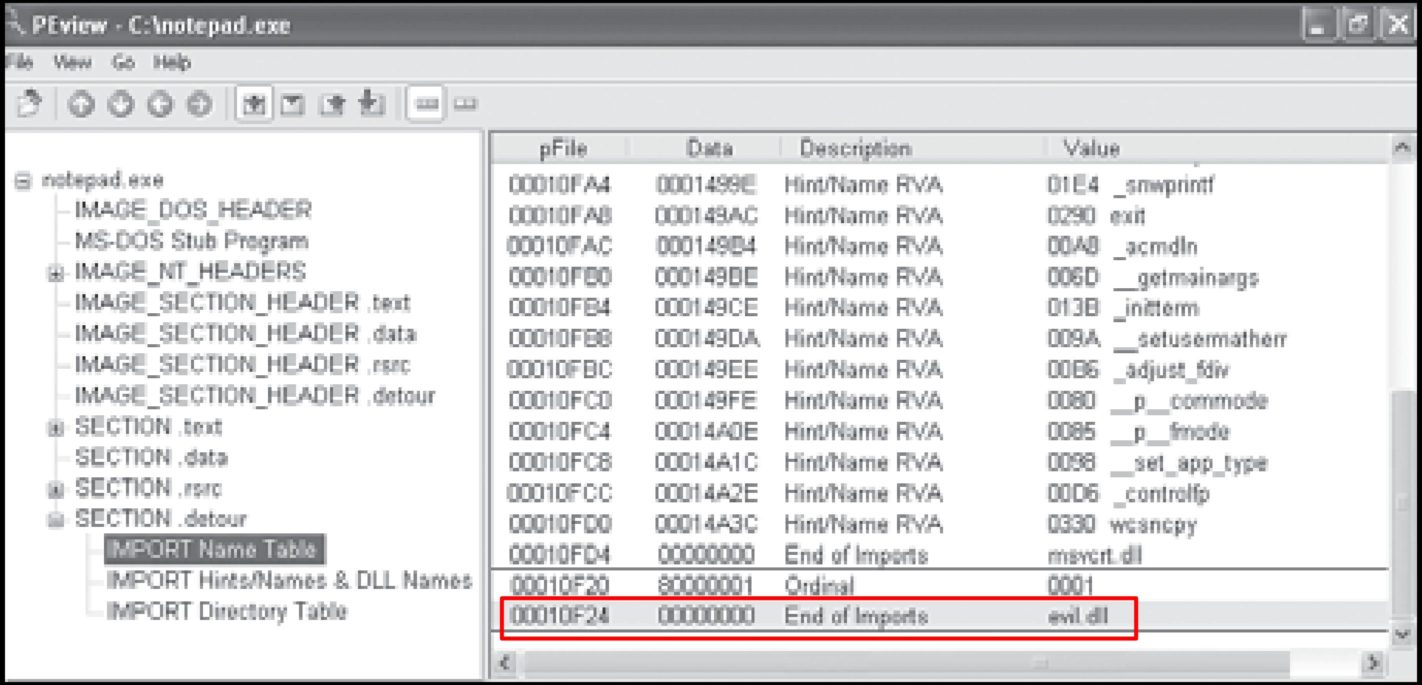
- Targeting a single thread requires a **search of the process** listing for the target process and can require that the malware run a program if the **target process is not already running**.
- If a malicious application **hooks a Windows message** that is used frequently, it’s more likely to trigger an IPS, so malware will often set a hook with a **message that is not often used**, such as WH_CBT
- Here, **malicious DLL** (*hook.dll*) is loaded by the malware, and the malicious hook procedure address is obtained. The **hook procedure, MalwareProc**, calls only CallNextHookEx.
- SetWindowsHookEx is then called for a **thread in *notepad.exe*** (assuming that *notepad.exe* is running). GetNotepadThreadId is a locally defined function that obtains a dwThreadId for *notepad.exe*.
- Finally, a **WH_CBT message** is sent to the injected *notepad.exe* in order to force *hook.dll* to be loaded by *notepad.exe*. This allows *hook.dll* to run in the *notepad.exe* process space.

Hooking with Detours

- Detours is a library developed by Microsoft Research in 1999. It was originally intended as a way to easily instrument and extend existing OS and application functionality.
- The Detours library makes it possible for a **developer to make application modifications** simply.
- Malware Authors use the Detours library to perform **import table modification**, **attach DLLs** to existing program files, and **add function hooks** to running processes.
- The malware modifies the PE structure and creates a section named **.detour**, which is typically placed between the export table and any debug symbols

Detours

- The .detour section contains the original PE header with a new import address table. The malware author then uses Detours to modify the PE header **to point to the new import table**, by using the **setdll tool** provided with the Detours library.



- notepad.exe
  - IMAGE_DOS_HEADER
  - MS-DOS Stub Program
  - IMAGE_NT_HEADERS
  - IMAGE_SECTION_HEADER .text
  - IMAGE_SECTION_HEADER .data
  - IMAGE_SECTION_HEADER .rsrc
  - IMAGE_SECTION_HEADER .detour
  - SECTION .text
  - SECTION .data
  - SECTION .rsrc
  - SECTION .detour
    - IMPORT Name Table
    - IMPORT Hints/Names & DLL Names
    - IMPORT Directory Table

pFile	Data	Description	Value
00010FA4	0001499E	Hint/Name RVA	01E4 _snwprintf
00010FA8	000149AC	Hint/Name RVA	0290 exit
00010FAC	000149B4	Hint/Name RVA	00A8 _acmdln
00010FB0	000149BE	Hint/Name RVA	006D __getmainargs
00010FB4	000149CE	Hint/Name RVA	013B _initterm
00010FB8	000149DA	Hint/Name RVA	009A __setusermatherr
00010FBC	000149EE	Hint/Name RVA	00B6 _adjust_fdiv
00010FC0	000149FE	Hint/Name RVA	0080 __p__commode
00010FC4	00014A0E	Hint/Name RVA	0085 __p__fmode
00010FC8	00014A1C	Hint/Name RVA	0098 __set_app_type
00010FCC	00014A2E	Hint/Name RVA	00D6 _controlfp
00010FD0	00014A3C	Hint/Name RVA	0330 wcsncpy
00010FD4	00000000	End of Imports	msvcrt.dll
00010F20	80000001	Ordinal	0001
00010F24	00000000	End of Imports	evil.dll

APC Injection

- Asynchronous procedure call (APC). APCs can direct a thread to execute some **other code prior to executing its regular execution path**.
- Every thread has a queue of APCs attached to it, and these are processed when the **thread is in an alertable state**, such as when they call functions like WaitForSingleObjectEx, WaitForMultipleObjectsEx, and Sleep.
- Malware authors use APCs to preempt threads in an alertable state in order to get immediate execution for their code.

References

- Practical Malware Analysis by Michael Sikorski
- Microsoft: https://learn.microsoft.com/en-us/windows/win32/winmsg/about-hooks
- Microsoft: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa
- Microsoft: https://learn.microsoft.com/en-us/windows/win32/winmsg/using-hooks
- Youtube: https://www.youtube.com/watch?v=xWHnhEZYTA0