Uncovering Hidden Threats:
Intro to Kernel Debugging with WinDbg

lexcentric

plant1330@gmail.com

Goal of the workshop

- To introduce the basics of kernel debugging with WinDbg, exploring kernel memory management, process structures, and demonstrating how to identify and exploit vulnerabilities using real-world examples.

Agenda

- **Introduction to WinDbg**
- **WinDbg Interface Basics**
  - Key commands and GUI overview
- **Understanding Processes**
  - Processes, threads, tokens, and memory
- **Kernel Basics**
  - Explanation of the kernel, its role, and transition from user mode to kernel mode.
- **WinDbg Practice**
  - Viewing SSDT
  - Viewing process list
- **Real-world Exploit Example**
  - rtcore64.sys exploitation and PatchGuard issue
- **Final Demo**
  - Simplified exploit development

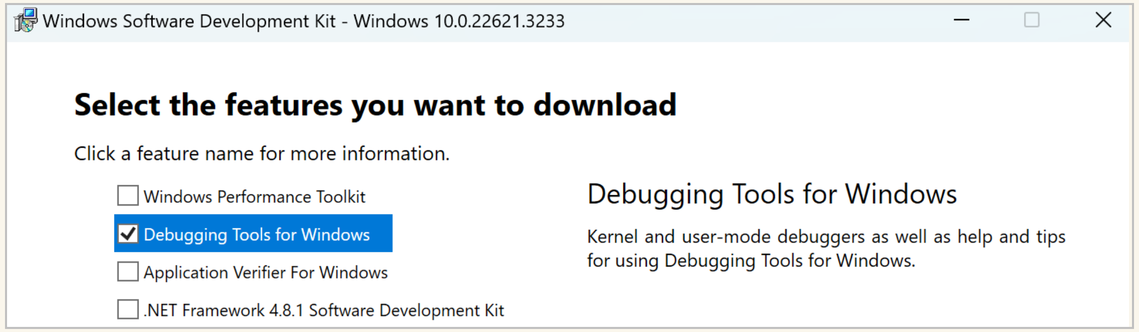
What is WinDbg?

What is WinDbg?

- **Windows debugger**, used by Microsoft itself for user space and kernel debugging.

What is WinDbg?

- Windows debugger, used by Microsoft itself for user space and kernel debugging.
- **WinDbg** from Debugging Tools (part of WinSDK)



Windows Software Development Kit - Windows 10.0.22621.3233

Select the features you want to download

Click a feature name for more information.

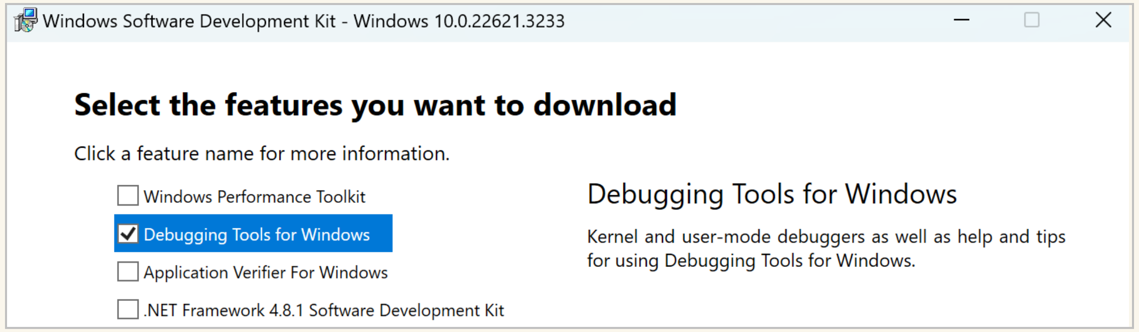
☐ Windows Performance Toolkit
☑ Debugging Tools for Windows
☐ Application Verifier For Windows
☐ .NET Framework 4.8.1 Software Development Kit

Debugging Tools for Windows

Kernel and user-mode debuggers as well as help and tips for using Debugging Tools for Windows.

What is WinDbg?

- Windows debugger, used by Microsoft itself for user space and kernel debugging.
- WinDbg from Debugging Tools (part of WinSDK)



Windows Software Development Kit - Windows 10.0.22621.3233

Select the features you want to download

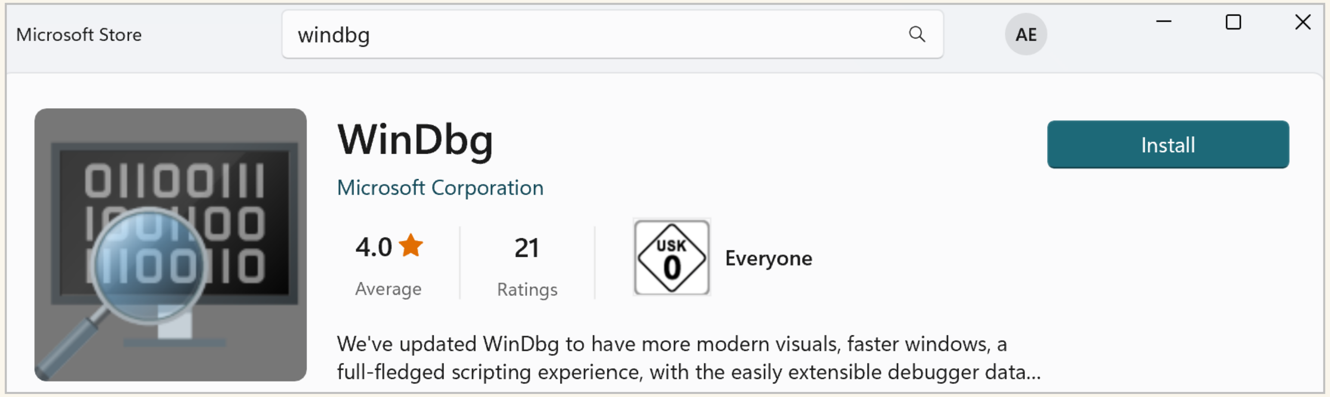
Click a feature name for more information.

- □ Windows Performance Toolkit
- ☑ Debugging Tools for Windows
- □ Application Verifier For Windows
- □ .NET Framework 4.8.1 Software Development Kit

Debugging Tools for Windows

Kernel and user-mode debuggers as well as help and tips for using Debugging Tools for Windows.

- **WinDbg Preview** from Microsoft Store (better UI and more features)



Microsoft Store

windbg

WinDbg

Microsoft Corporation

4.0 ★ Average

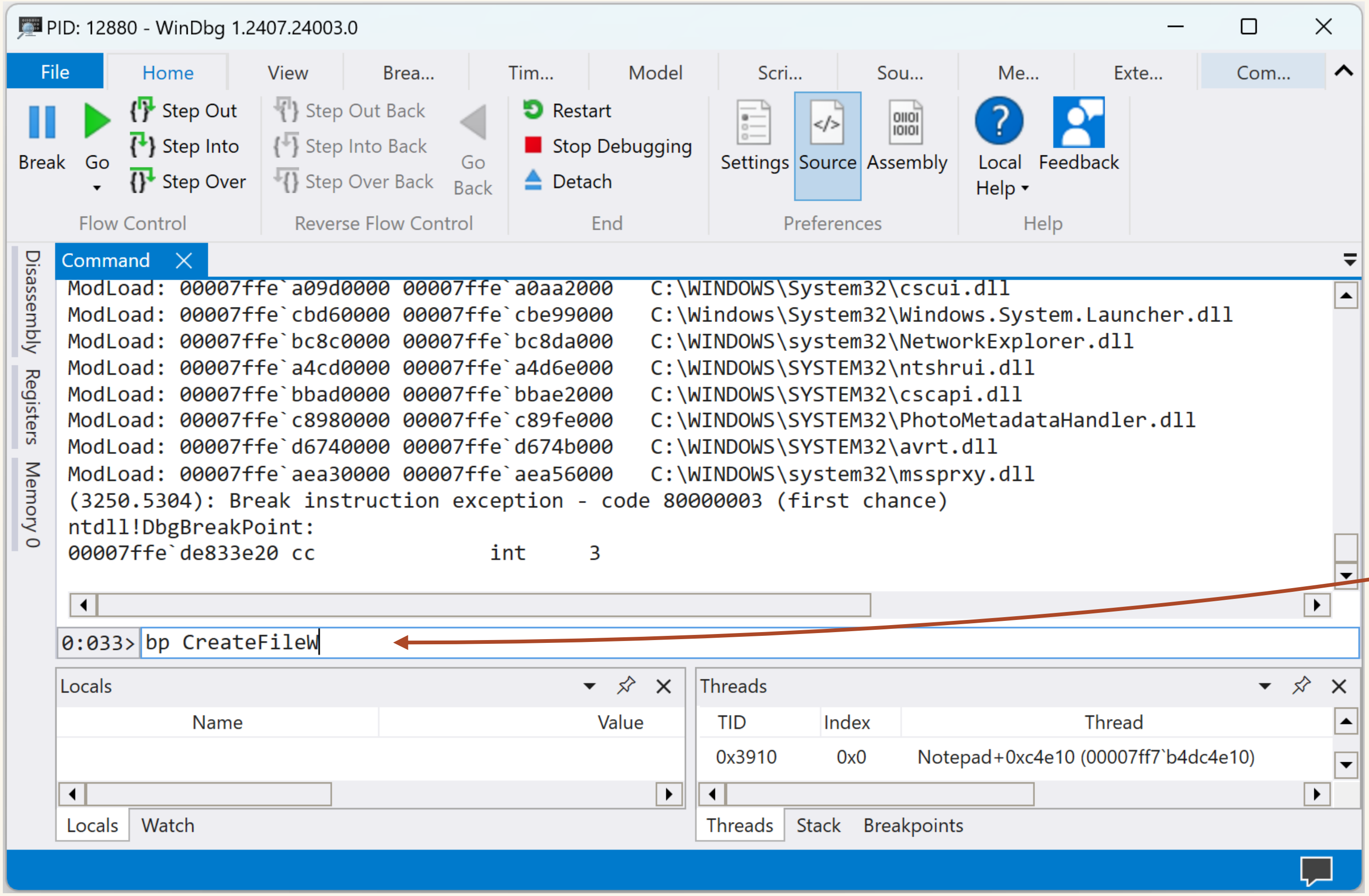
21 Ratings

USK 0 Everyone

Install

We've updated WinDbg to have more modern visuals, faster windows, a full-fledged scripting experience, with the easily extensible debugger data...

GUI? Kind of...



```
ModLoad: 00007ffe`a09d0000 00007ffe`a0aa2000   C:\WINDOWS\System32\cscui.dll
ModLoad: 00007ffe`cbd60000 00007ffe`cbe99000   C:\Windows\System32\Windows.System.Launcher.dll
ModLoad: 00007ffe`bc8c0000 00007ffe`bc8da000   C:\WINDOWS\system32\NetworkExplorer.dll
ModLoad: 00007ffe`a4cd0000 00007ffe`a4d6e000   C:\WINDOWS\SYSTEM32\ntshrui.dll
ModLoad: 00007ffe`bbad0000 00007ffe`bbae2000   C:\WINDOWS\SYSTEM32\cscapi.dll
ModLoad: 00007ffe`c8980000 00007ffe`c89fe000   C:\WINDOWS\SYSTEM32\PhotoMetadataHandler.dll
ModLoad: 00007ffe`d6740000 00007ffe`d674b000   C:\WINDOWS\SYSTEM32\avrt.dll
ModLoad: 00007ffe`aea30000 00007ffe`aea56000   C:\WINDOWS\system32\mssprxy.dll
(3250.5304): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
```

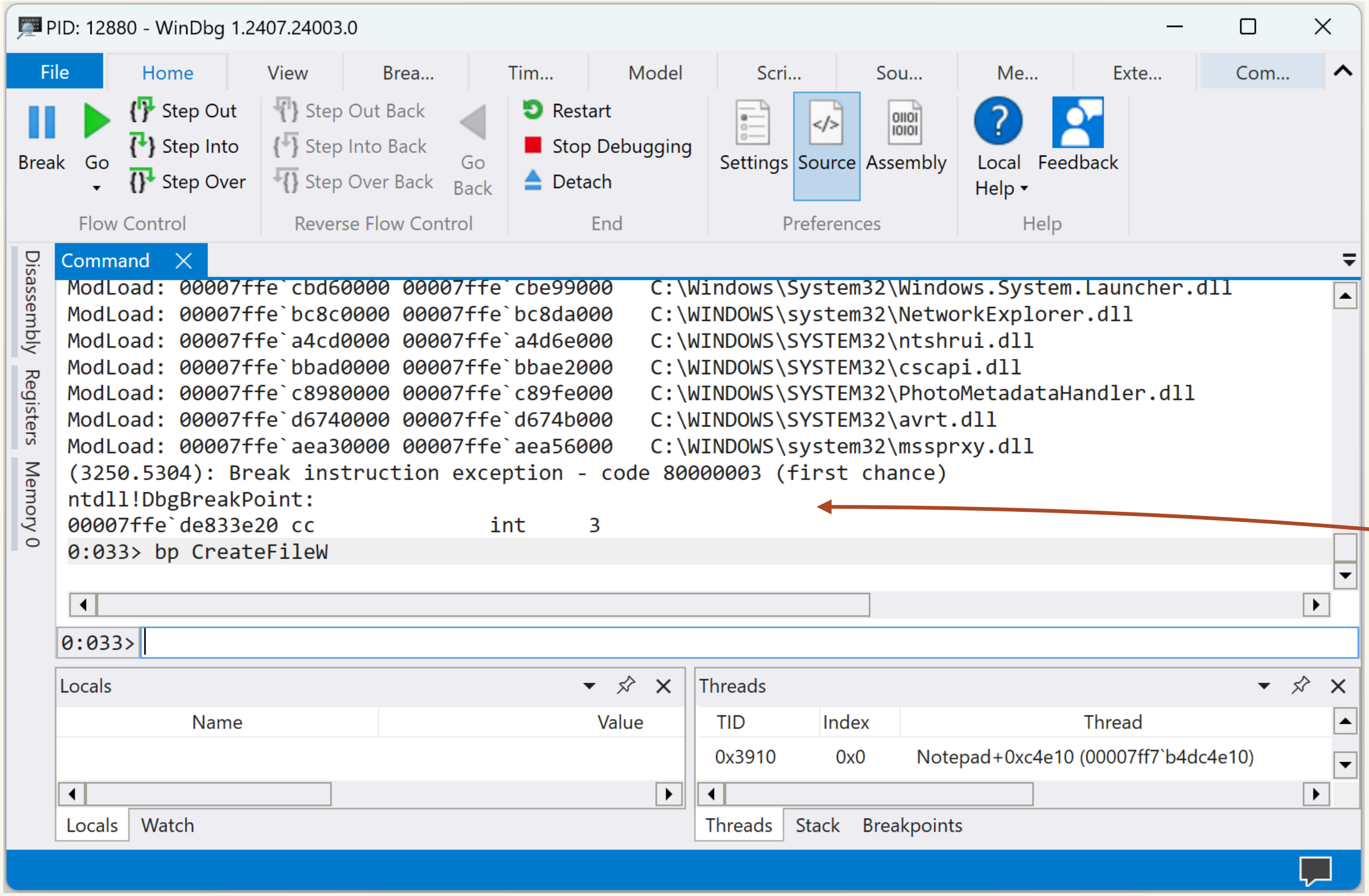
```
0:033> bp CreateFileW
```

Name	Value

TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff7`b4dc4e10)

Enter your commands here...

GUI? Kind of...

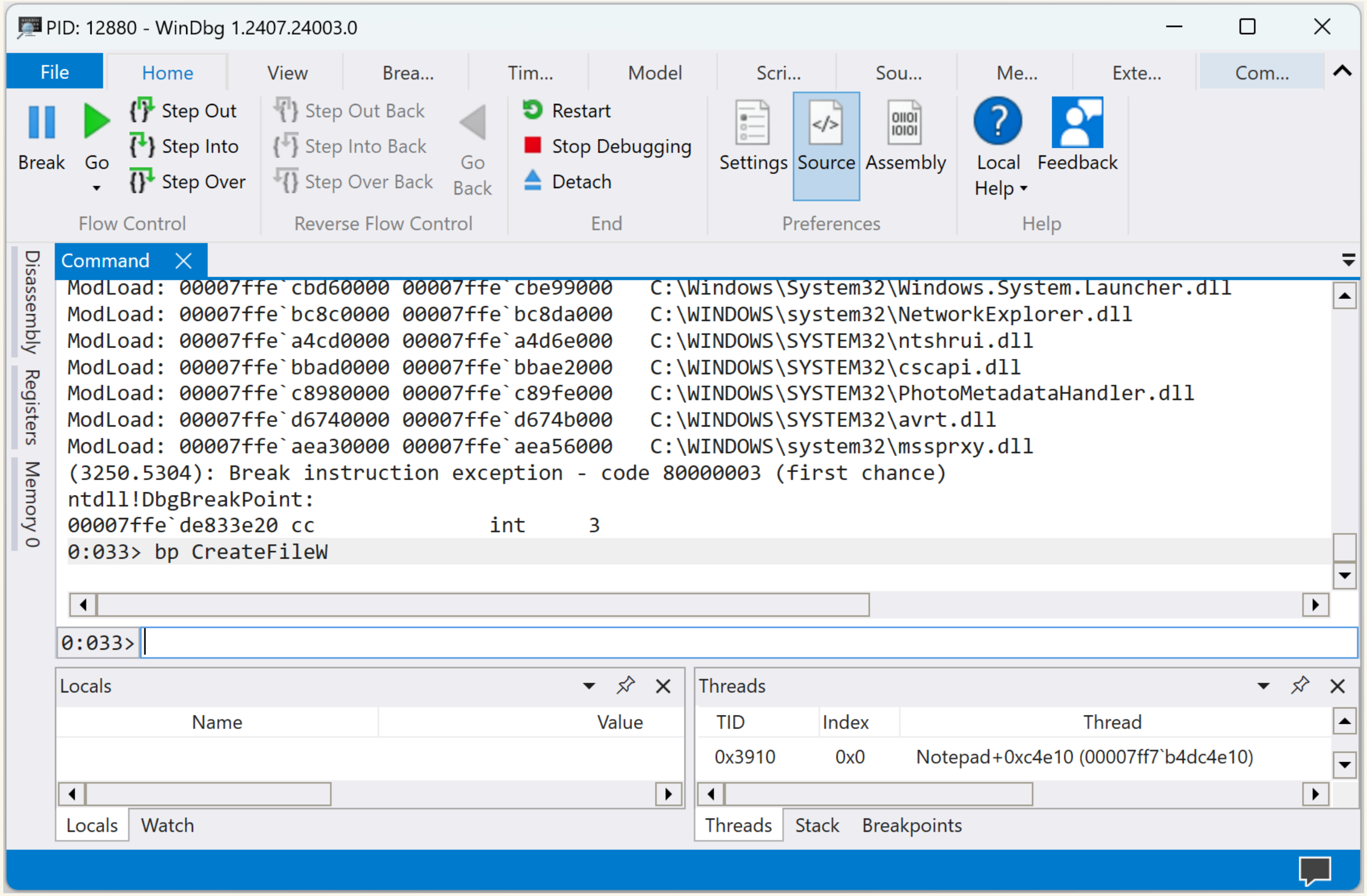


```
ModLoad: 00007ffe`cbd60000 00007ffe`cbe99000   C:\Windows\System32\Windows.System.Launcher.dll
ModLoad: 00007ffe`bc8c0000 00007ffe`bc8da000   C:\WINDOWS\system32\NetworkExplorer.dll
ModLoad: 00007ffe`a4cd0000 00007ffe`a4d6e000   C:\WINDOWS\SYSTEM32\ntshrui.dll
ModLoad: 00007ffe`bbad0000 00007ffe`bbae2000   C:\WINDOWS\SYSTEM32\cscapi.dll
ModLoad: 00007ffe`c8980000 00007ffe`c89fe000   C:\WINDOWS\SYSTEM32\PhotoMetadataHandler.dll
ModLoad: 00007ffe`d6740000 00007ffe`d674b000   C:\WINDOWS\SYSTEM32\avrt.dll
ModLoad: 00007ffe`aea30000 00007ffe`aea56000   C:\WINDOWS\system32\mssprxy.dll
(3250.5304): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
0:033> bp CreateFileW

0:033>
```

TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff7`b4dc4e10)

Get your results here ☺

GUI? Kind of...



```
ModLoad: 00007ffe`cbd60000 00007ffe`cbe99000   C:\Windows\System32\Windows.System.Launcher.dll
ModLoad: 00007ffe`bc8c0000 00007ffe`bc8da000   C:\WINDOWS\system32\NetworkExplorer.dll
ModLoad: 00007ffe`a4cd0000 00007ffe`a4d6e000   C:\WINDOWS\SYSTEM32\ntshrui.dll
ModLoad: 00007ffe`bbad0000 00007ffe`bbae2000   C:\WINDOWS\SYSTEM32\cscapi.dll
ModLoad: 00007ffe`c8980000 00007ffe`c89fe000   C:\WINDOWS\SYSTEM32\PhotoMetadataHandler.dll
ModLoad: 00007ffe`d6740000 00007ffe`d674b000   C:\WINDOWS\SYSTEM32\avrt.dll
ModLoad: 00007ffe`aea30000 00007ffe`aea56000   C:\WINDOWS\system32\mssprxy.dll
(3250.5304): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
0:033> bp CreateFileW
```

```
0:033>
```

Locals

Name	Value

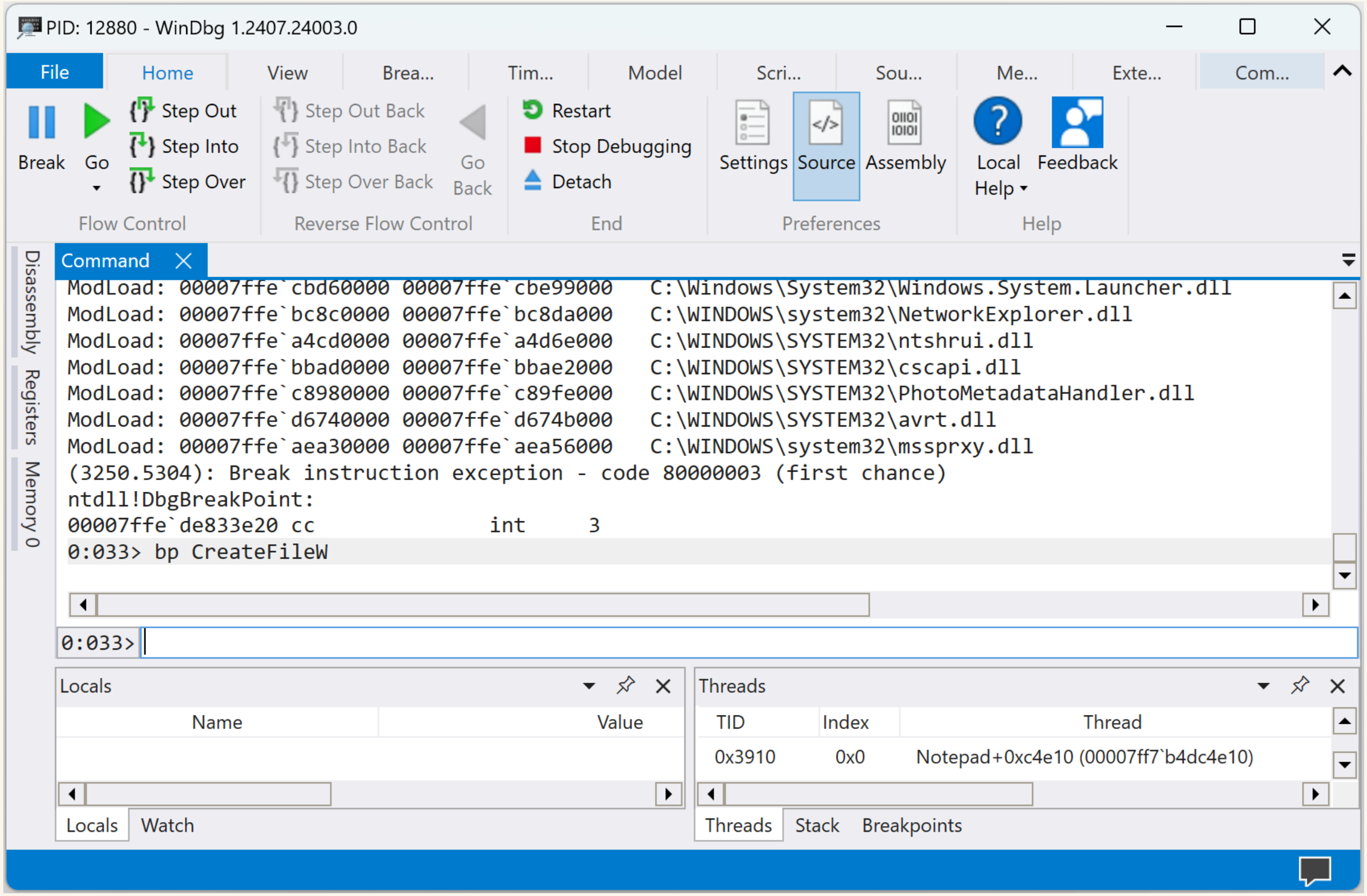
Threads

TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff7`b4dc4e10)

No Panic!

We will need only five base commands ☺

GUI? Kind of...



```
ModLoad: 00007ffe`cbd60000 00007ffe`cbe99000   C:\Windows\System32\Windows.System.Launcher.dll
ModLoad: 00007ffe`bc8c0000 00007ffe`bc8da000   C:\WINDOWS\system32\NetworkExplorer.dll
ModLoad: 00007ffe`a4cd0000 00007ffe`a4d6e000   C:\WINDOWS\SYSTEM32\ntshrui.dll
ModLoad: 00007ffe`bbad0000 00007ffe`bbae2000   C:\WINDOWS\SYSTEM32\cscapi.dll
ModLoad: 00007ffe`c8980000 00007ffe`c89fe000   C:\WINDOWS\SYSTEM32\PhotoMetadataHandler.dll
ModLoad: 00007ffe`d6740000 00007ffe`d674b000   C:\WINDOWS\SYSTEM32\avrt.dll
ModLoad: 00007ffe`aea30000 00007ffe`aea56000   C:\WINDOWS\system32\mssprxy.dll
(3250.5304): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
0:033> bp CreateFileW

0:033>
```

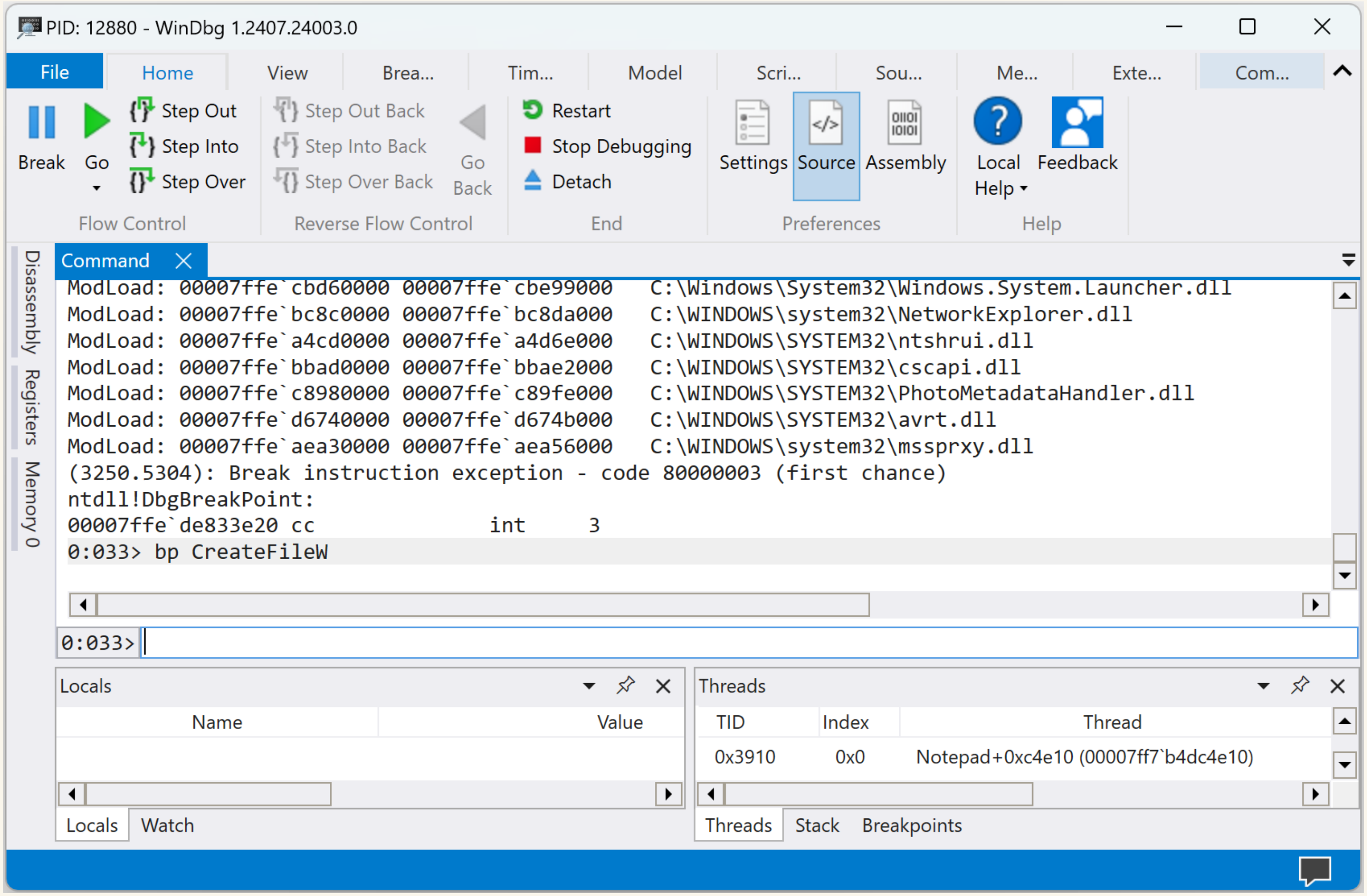
TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff7`b4dc4e10)

No Panic!

We will need only five base commands ☺

- **Set breakpoint** **bp**

GUI? Kind of...



```
ModLoad: 00007ffe`cbd60000 00007ffe`cbe99000   C:\Windows\System32\Windows.System.Launcher.dll
ModLoad: 00007ffe`bc8c0000 00007ffe`bc8da000   C:\WINDOWS\system32\NetworkExplorer.dll
ModLoad: 00007ffe`a4cd0000 00007ffe`a4d6e000   C:\WINDOWS\SYSTEM32\ntshrui.dll
ModLoad: 00007ffe`bbad0000 00007ffe`bbae2000   C:\WINDOWS\SYSTEM32\cscapi.dll
ModLoad: 00007ffe`c8980000 00007ffe`c89fe000   C:\WINDOWS\SYSTEM32\PhotoMetadataHandler.dll
ModLoad: 00007ffe`d6740000 00007ffe`d674b000   C:\WINDOWS\SYSTEM32\avrt.dll
ModLoad: 00007ffe`aea30000 00007ffe`aea56000   C:\WINDOWS\system32\mssprxy.dll
(3250.5304): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
0:033> bp CreateFileW
```

```
0:033>
```

Name	Value

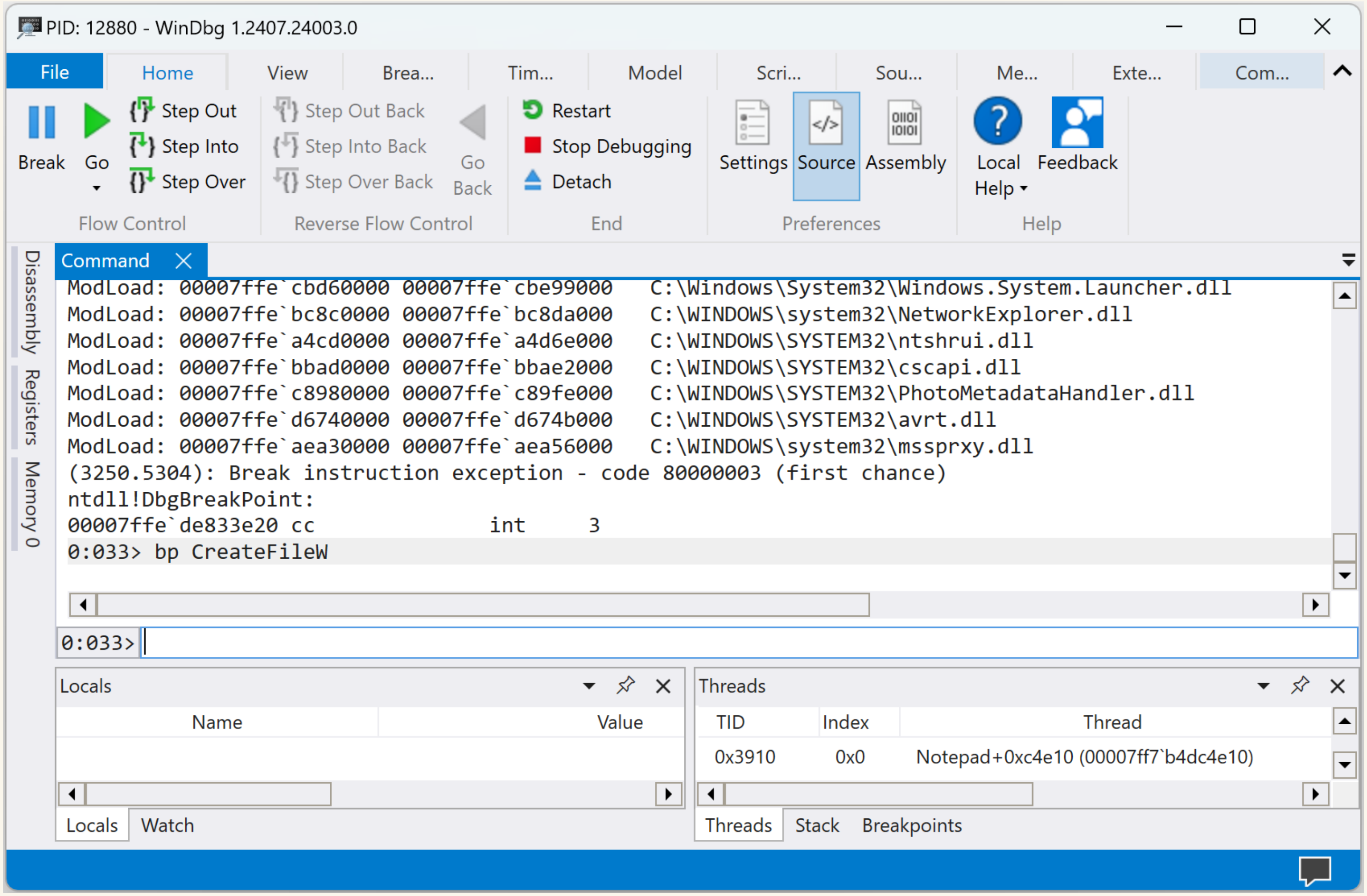
TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff7`b4dc4e10)

No Panic!

We will need only five base commands ☺

• Set breakpoint	bp
• **Show stack trace**	**k**

GUI? Kind of...



```
ModLoad: 00007ffe`cbd60000 00007ffe`cbe99000   C:\Windows\System32\Windows.System.Launcher.dll
ModLoad: 00007ffe`bc8c0000 00007ffe`bc8da000   C:\WINDOWS\system32\NetworkExplorer.dll
ModLoad: 00007ffe`a4cd0000 00007ffe`a4d6e000   C:\WINDOWS\SYSTEM32\ntshrui.dll
ModLoad: 00007ffe`bbad0000 00007ffe`bbae2000   C:\WINDOWS\SYSTEM32\cscapi.dll
ModLoad: 00007ffe`c8980000 00007ffe`c89fe000   C:\WINDOWS\SYSTEM32\PhotoMetadataHandler.dll
ModLoad: 00007ffe`d6740000 00007ffe`d674b000   C:\WINDOWS\SYSTEM32\avrt.dll
ModLoad: 00007ffe`aea30000 00007ffe`aea56000   C:\WINDOWS\system32\mssprxy.dll
(3250.5304): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
0:033> bp CreateFileW
```

```
0:033>
```

Name	Value

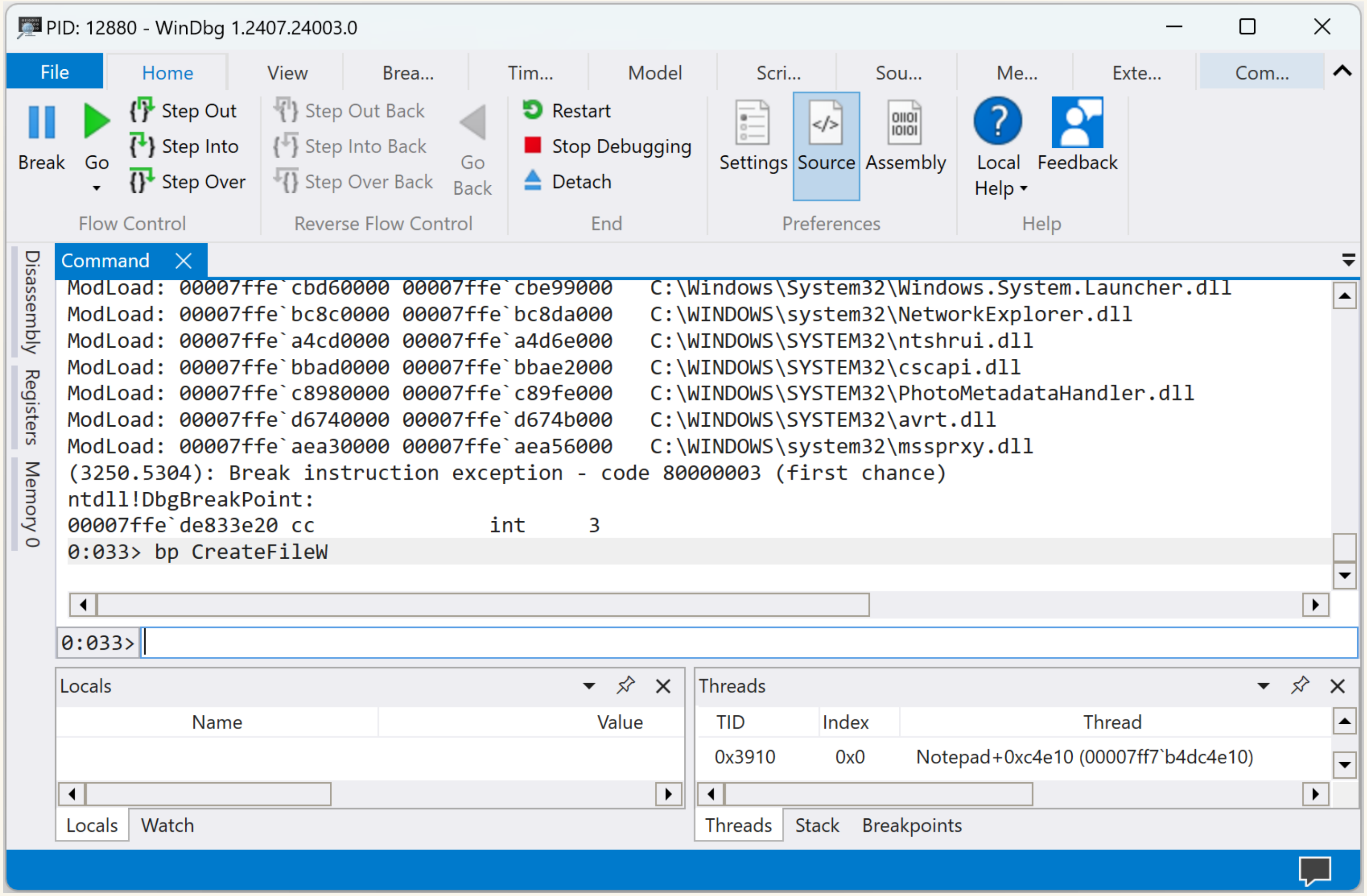
TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff7`b4dc4e10)

No Panic!

We will need only five base commands ☺

• Set breakpoint	bp
• Show stack trace	k
• **Unassemble**	**u**

GUI? Kind of...



```
ModLoad: 00007ffe`cbd60000 00007ffe`cbe99000   C:\Windows\System32\Windows.System.Launcher.dll
ModLoad: 00007ffe`bc8c0000 00007ffe`bc8da000   C:\WINDOWS\system32\NetworkExplorer.dll
ModLoad: 00007ffe`a4cd0000 00007ffe`a4d6e000   C:\WINDOWS\SYSTEM32\ntshrui.dll
ModLoad: 00007ffe`bbad0000 00007ffe`bbae2000   C:\WINDOWS\SYSTEM32\cscapi.dll
ModLoad: 00007ffe`c8980000 00007ffe`c89fe000   C:\WINDOWS\SYSTEM32\PhotoMetadataHandler.dll
ModLoad: 00007ffe`d6740000 00007ffe`d674b000   C:\WINDOWS\SYSTEM32\avrt.dll
ModLoad: 00007ffe`aea30000 00007ffe`aea56000   C:\WINDOWS\system32\mssprxy.dll
(3250.5304): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
0:033> bp CreateFileW
```

```
0:033>
```

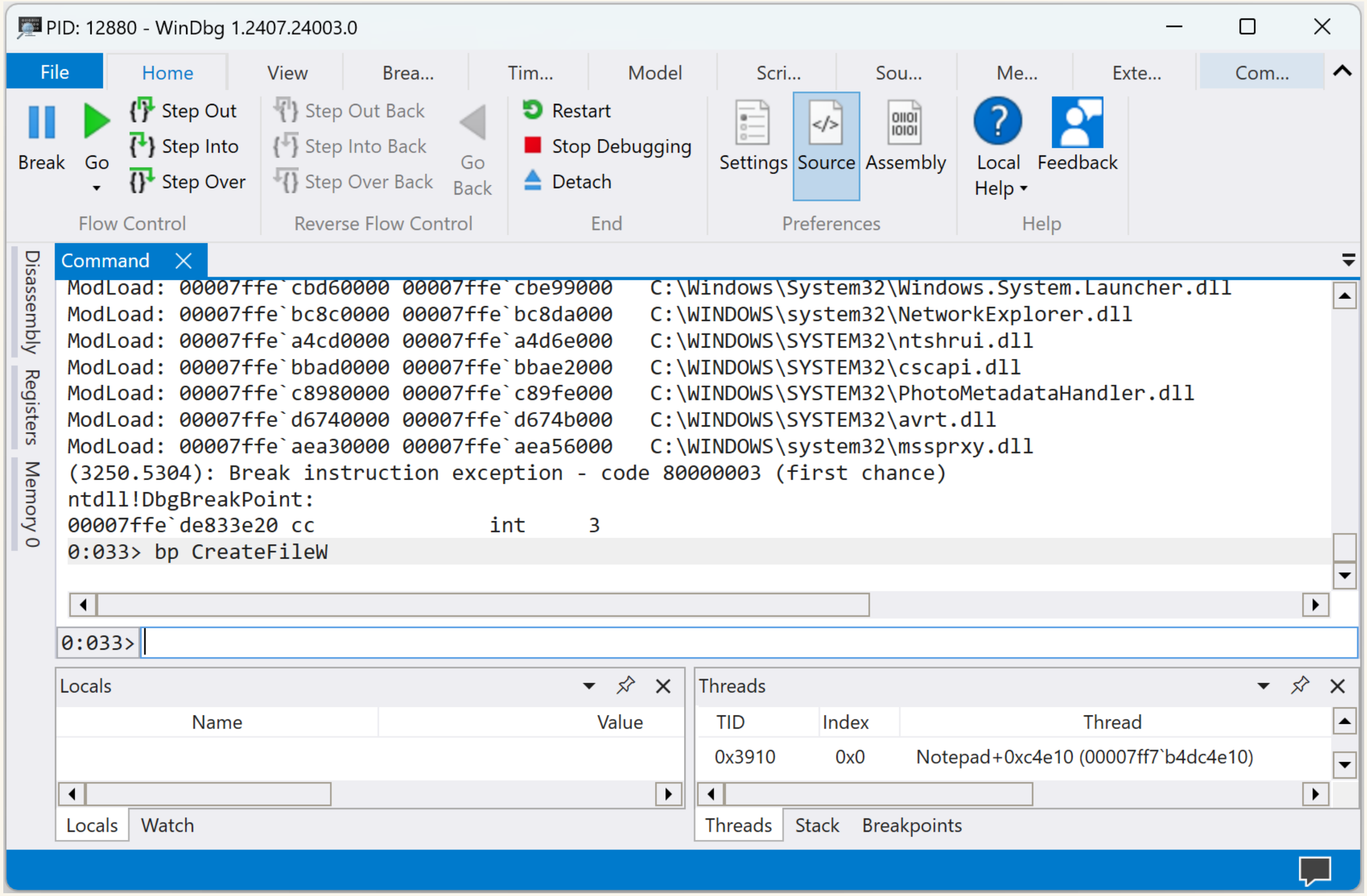
TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff7`b4dc4e10)

No Panic!

We will need only five base commands ☺

- Set breakpoint bp
- Show stack trace k
- Unassemble u
- **Display memory d_**

GUI? Kind of...



```
ModLoad: 00007ffe`cbd60000 00007ffe`cbe99000   C:\Windows\System32\Windows.System.Launcher.dll
ModLoad: 00007ffe`bc8c0000 00007ffe`bc8da000   C:\WINDOWS\system32\NetworkExplorer.dll
ModLoad: 00007ffe`a4cd0000 00007ffe`a4d6e000   C:\WINDOWS\SYSTEM32\ntshrui.dll
ModLoad: 00007ffe`bbad0000 00007ffe`bbae2000   C:\WINDOWS\SYSTEM32\cscapi.dll
ModLoad: 00007ffe`c8980000 00007ffe`c89fe000   C:\WINDOWS\SYSTEM32\PhotoMetadataHandler.dll
ModLoad: 00007ffe`d6740000 00007ffe`d674b000   C:\WINDOWS\SYSTEM32\avrt.dll
ModLoad: 00007ffe`aea30000 00007ffe`aea56000   C:\WINDOWS\system32\mssprxy.dll
(3250.5304): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
0:033> bp CreateFileW
```

```
0:033>
```

Name	Value

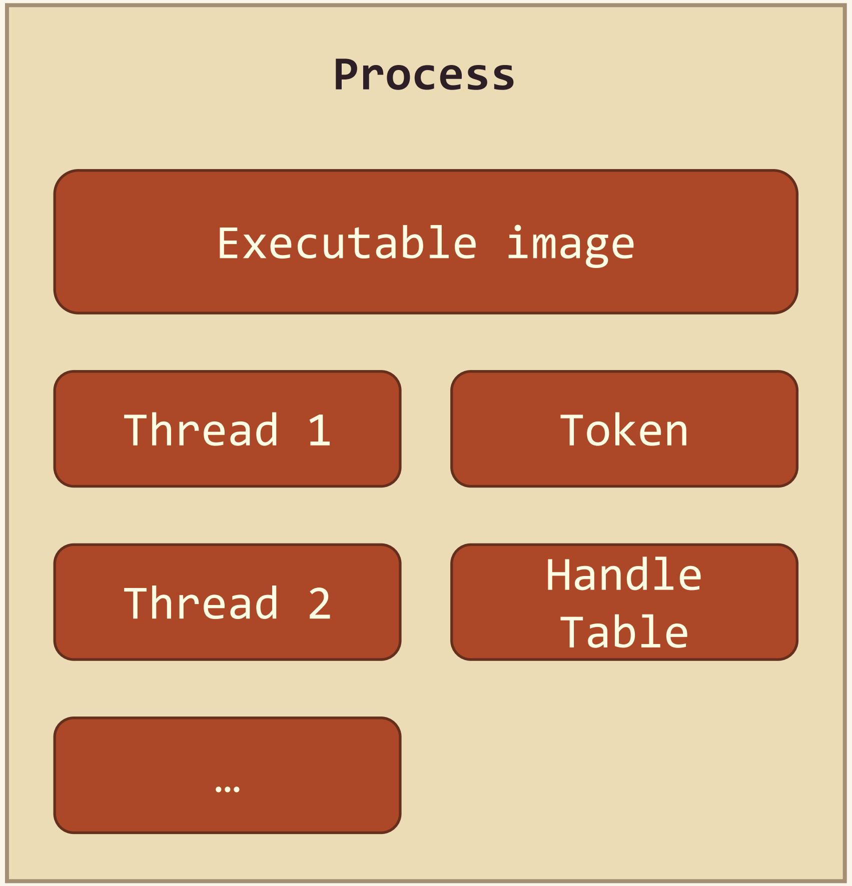
TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff7`b4dc4e10)

No Panic!

We will need only five base commands ☺

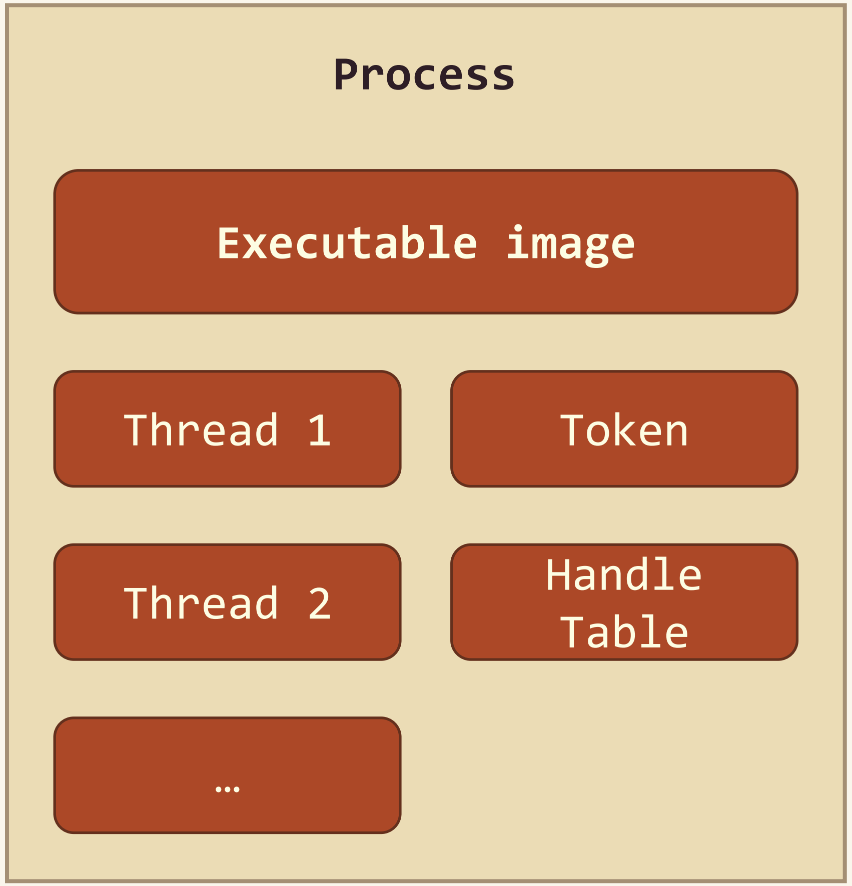
• Set breakpoint	bp
• Show stack trace	k
• Unassemble	u
• Display memory	d_
• **Arithmetic operations**	**?**

What is a process?



A Windows **process** is an instance of a program with its own **memory space**, which contains its code, data, stack, heap, and other necessary **resources** for execution.

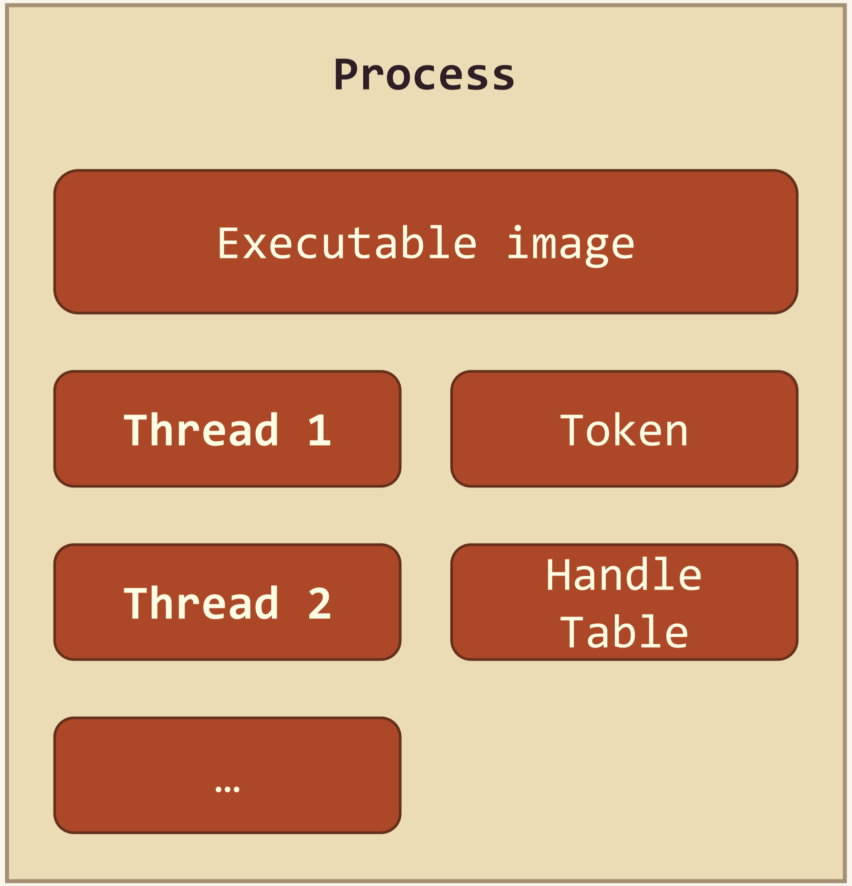
What is a process?



An **executable image** is a file containing initial code, data, and other resources, usually in **.exe** or **.dll** format, that can be loaded into memory for execution by the operating system.

e.g., notepad.exe

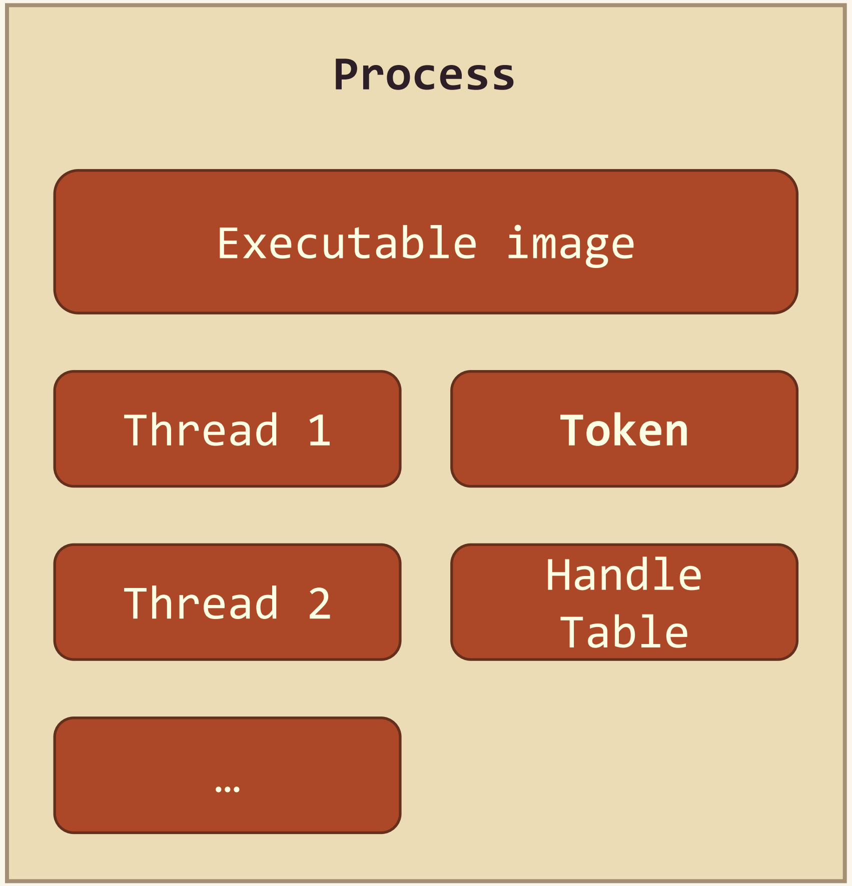
What is a process?



A **thread** is a single sequence of instructions within a process that can run independently, allowing multitasking within the process.

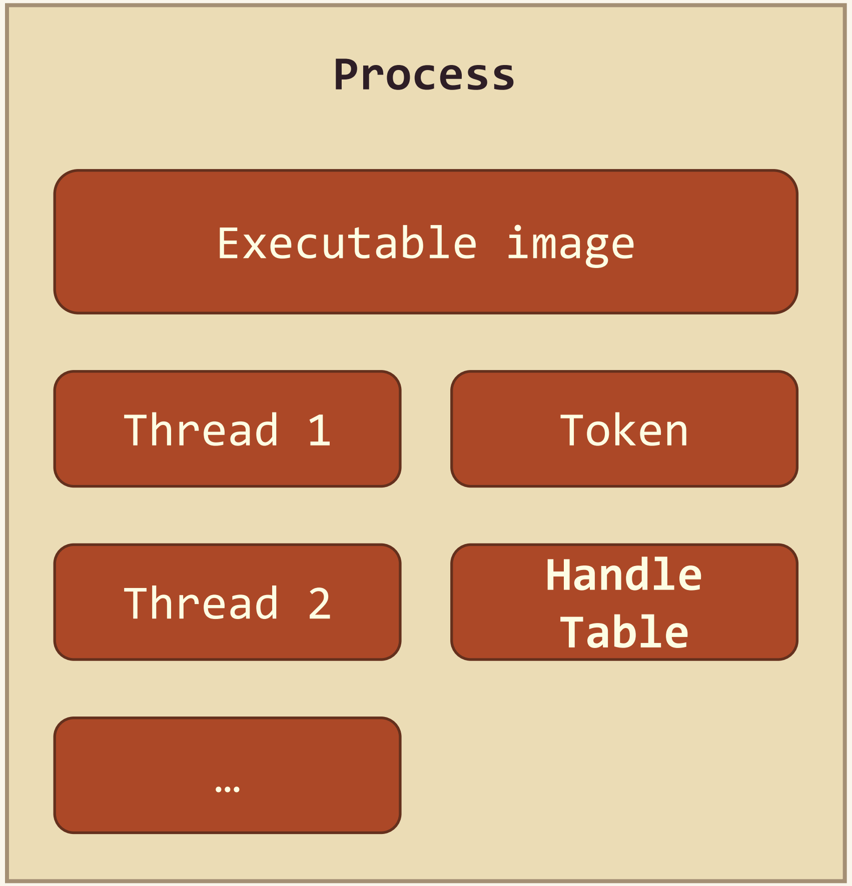
```
…
FILE *file;
int number;
file = fopen("input.txt", "r");
fscanf(file, "%d", &number);
printf("%d\n", number);
…
```

What is a process?



A **process token** contains information like the user’s **SID (Security Identifier)**, **group SIDs**, **privileges**, and **access rights**. It defines the security context of the process, determining what resources it can access.

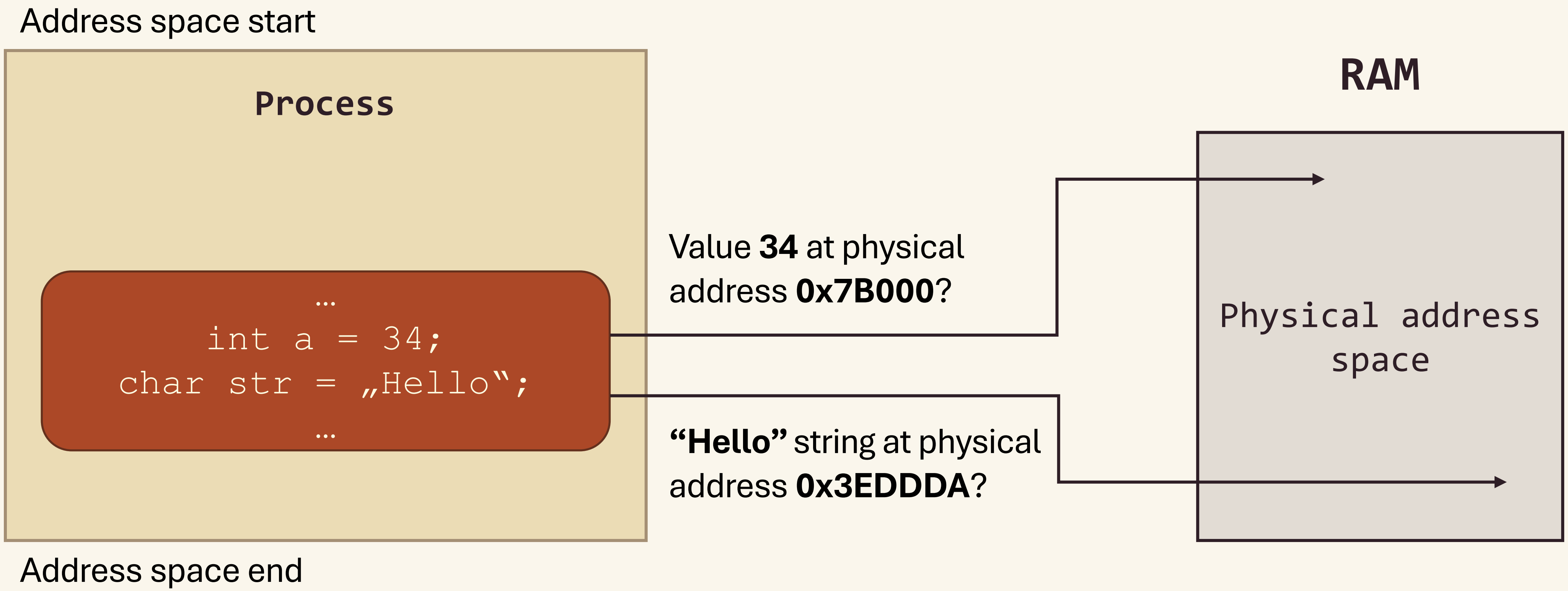
What is a process?



A **handle table** stores references to system resources (e.g., files, threads, registry keys) for a process, managing access to them.

```
+---------+-------------------------+
| Handle  | Resource Type           |
+---------+-------------------------+
| 0x4     | File: "example.txt"     |
| 0x8     | Thread ID: 1234         |
| 0xC     | Mutex: "MyMutex"        |
| 0x10    | Registry Key: HKCU\...  |
| 0x14    | Socket: 192.168.1.1:80  |
+---------+-------------------------+
```

Process memory... like this?



Address space start

Process

```
…
int a = 34;
char str = „Hello“;
…
```

Value **34** at physical address **0x7B000**?

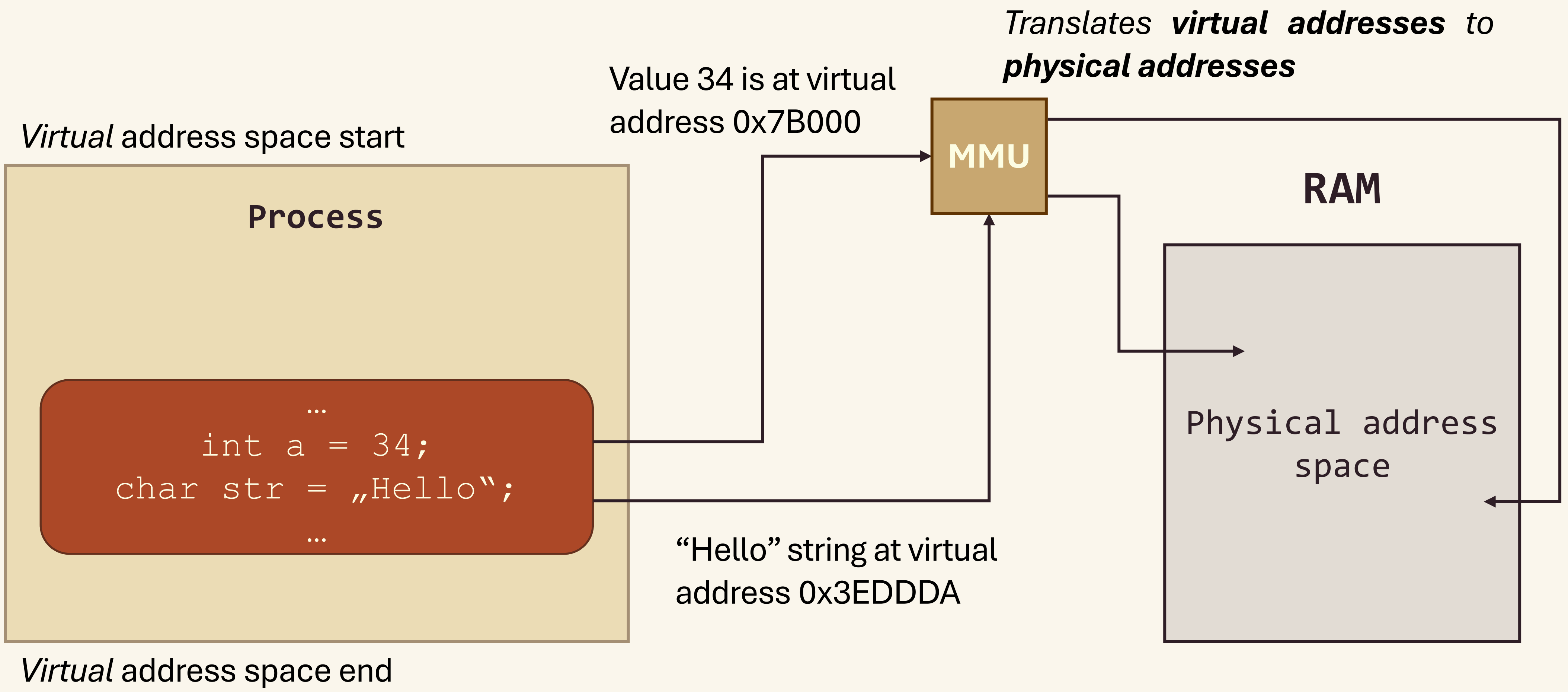
“Hello” string at physical address **0x3EDDDA**?

RAM

Physical address space

Address space end

Virtual Memory



Translates ***virtual addresses*** *to* ***physical addresses***

Value 34 is at virtual address 0x7B000

Virtual address space start

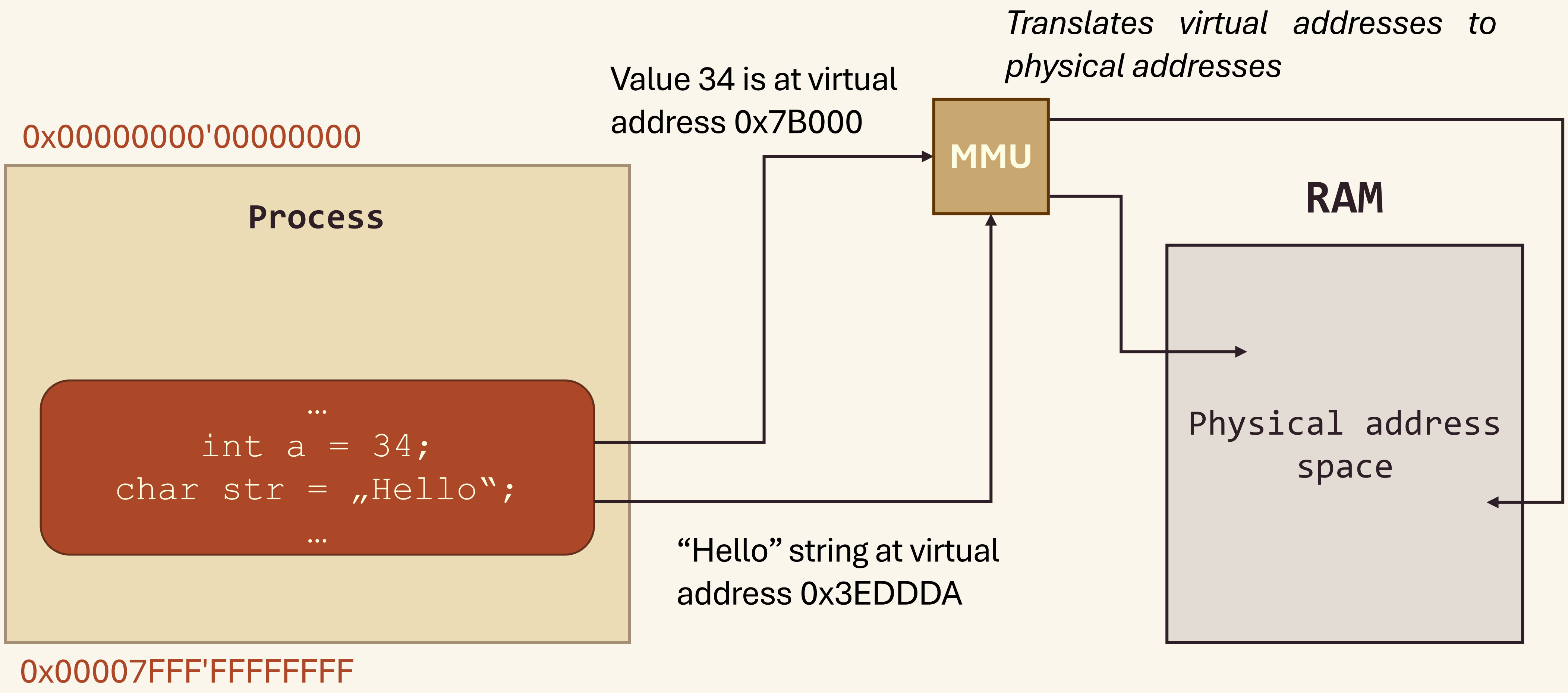
```
…
int a = 34;
char str = „Hello“;
…
```

“Hello” string at virtual address 0x3EDDDA

Virtual address space end

* MMU – Memory Management Unit

Virtual Memory (user mode)



Translates virtual addresses to physical addresses

Value 34 is at virtual address 0x7B000

0x00000000'00000000

Process

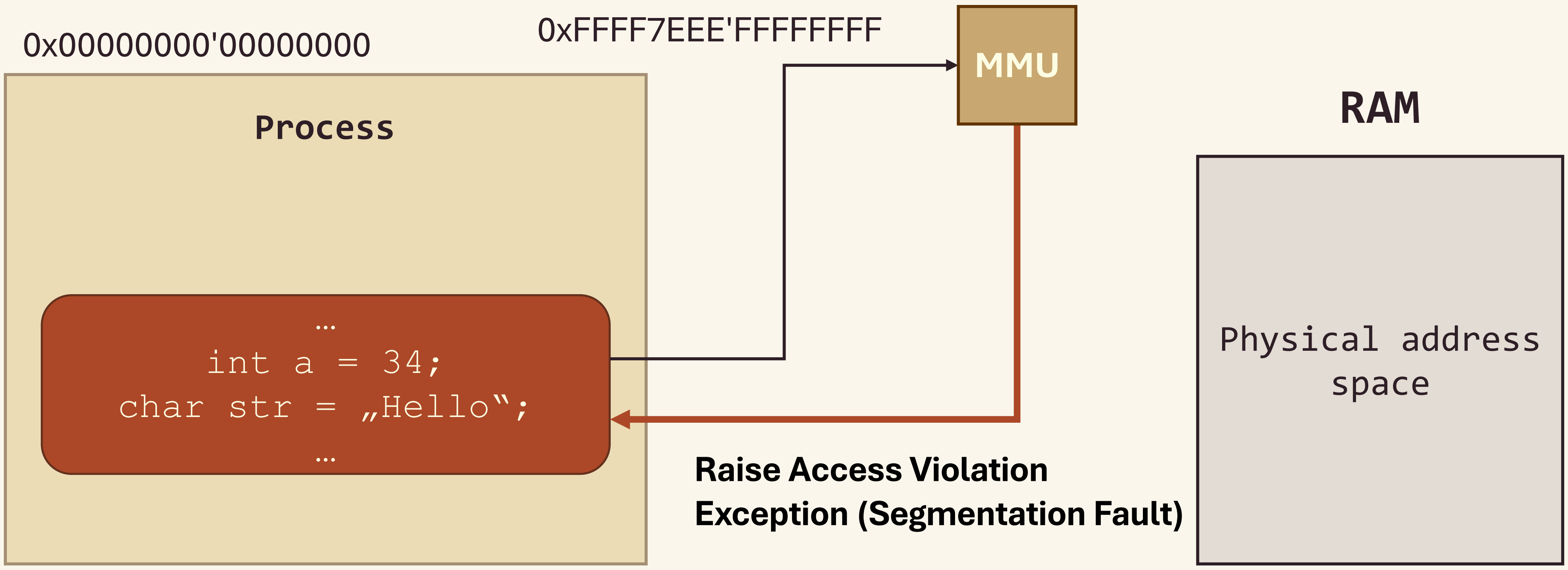
```
…
int a = 34;
char str = „Hello“;
…
```

0x00007FFF'FFFFFFFF

“Hello” string at virtual address 0x3EDDDA

* The usage insights of physical address ranges can be obtained using the *RAMMap* tool from the Sysinternals Suite.

Virtual Memory (user mode)



Detects the invalid memory access since the address does not belong to the process's accessible address space

Read value at virtual address 0xFFFF7EEE'FFFFFFFF

0x00000000'00000000

Process

```
…
int a = 34;
char str = „Hello";
…
```

0x00007FFF'FFFFFFFF

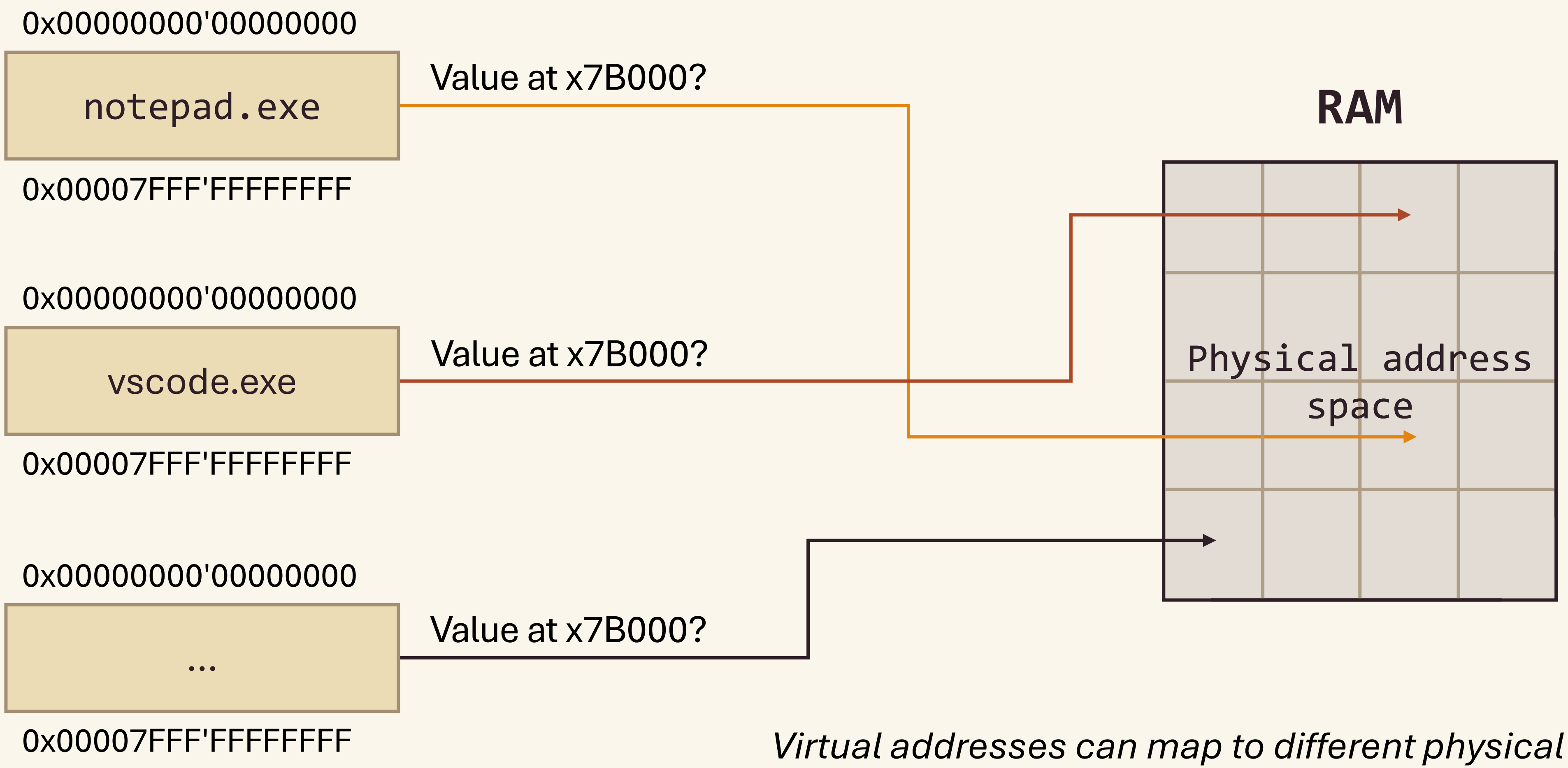
MMU

RAM

Physical address space

Raise Access Violation Exception (Segmentation Fault)

Virtual Memory (multiple processes)



0x00000000'00000000

notepad.exe

0x00007FFF'FFFFFFFF

Value at x7B000?

0x00000000'00000000

vscode.exe

0x00007FFF'FFFFFFFF

Value at x7B000?

0x00000000'00000000

...

0x00007FFF'FFFFFFFF

Value at x7B000?

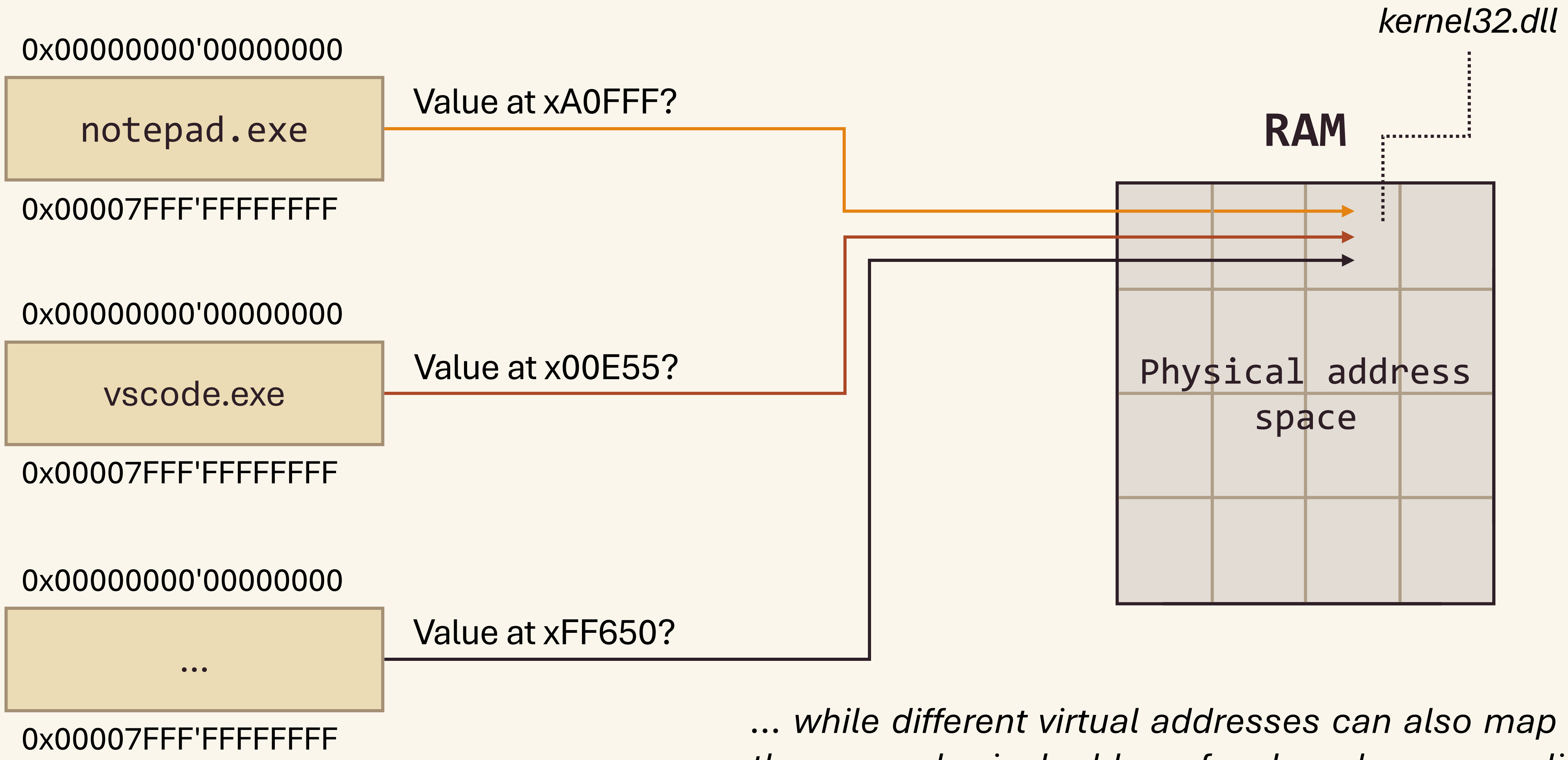
RAM

Physical address space

* in the diagram the MMU module is omitted, but it is implied

Virtual addresses can map to different physical addresses across processes...

Virtual Memory (multiple processes)



0x00000000'00000000

notepad.exe

0x00007FFF'FFFFFFFF

Value at xA0FFF?

0x00000000'00000000

vscode.exe

0x00007FFF'FFFFFFFF

Value at x00E55?

0x00000000'00000000

...

0x00007FFF'FFFFFFFF

Value at xFF650?

kernel32.dll

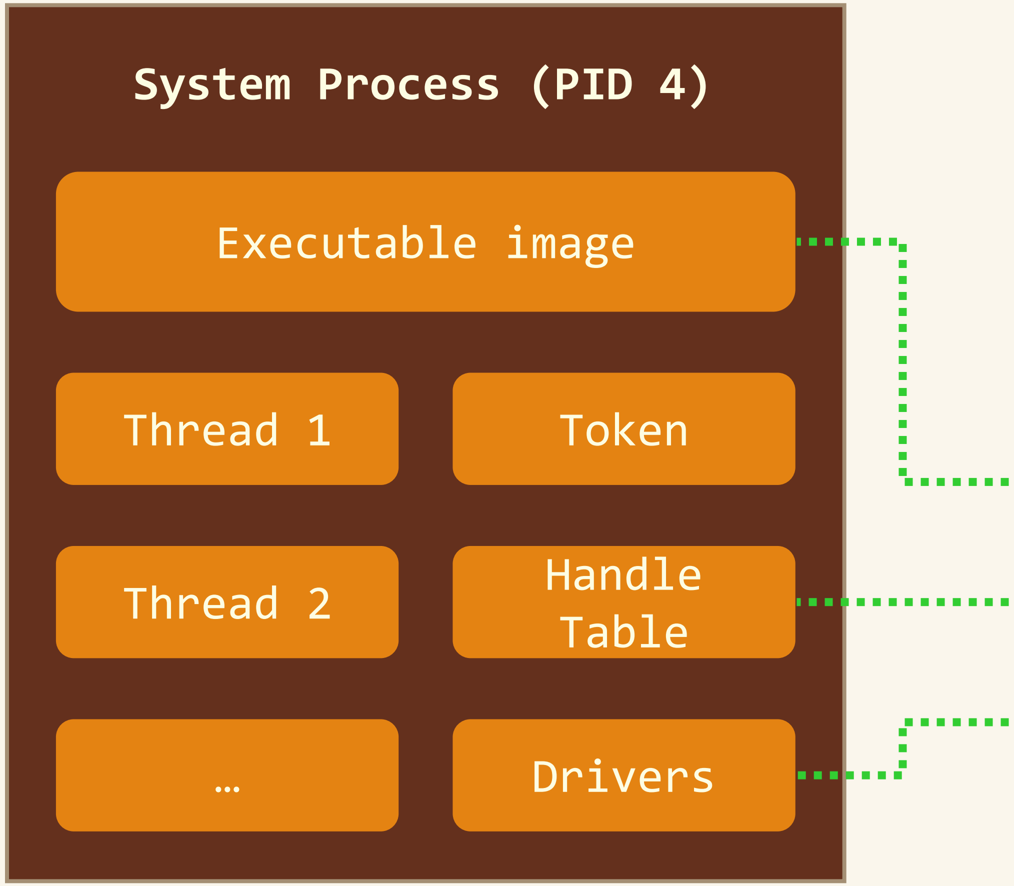
RAM

Physical address space

* in the diagram the MMU module is omitted, but it is implied

... while different virtual addresses can also map to the same physical address for shared resources like system DLLs

Then, what is the kernel?



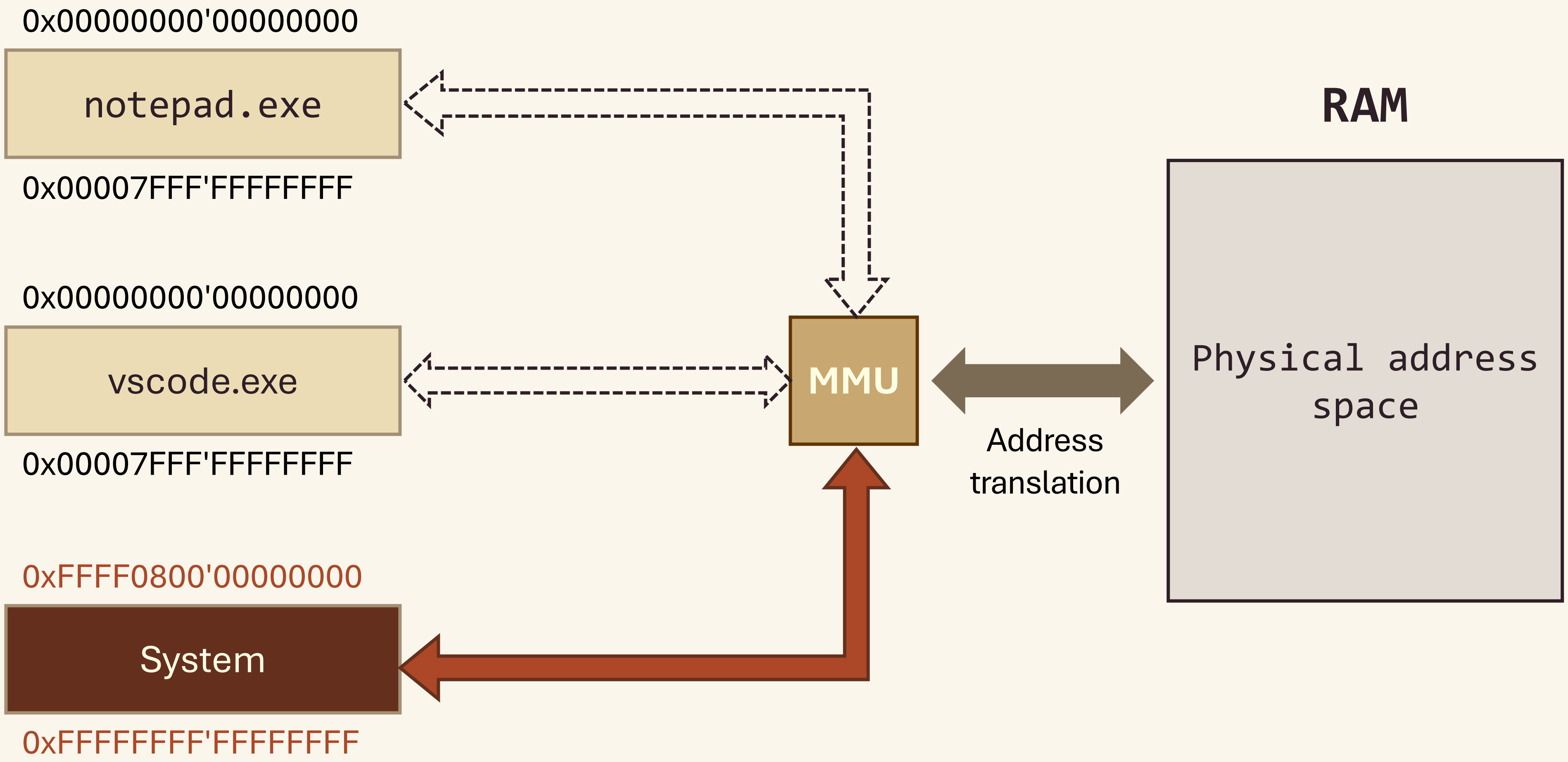
The **kernel** is the core of the operating system that manages memory, hardware through drivers, process scheduling, access management, and provides essential services to applications.

ntoskrnl.exe

kernel objects

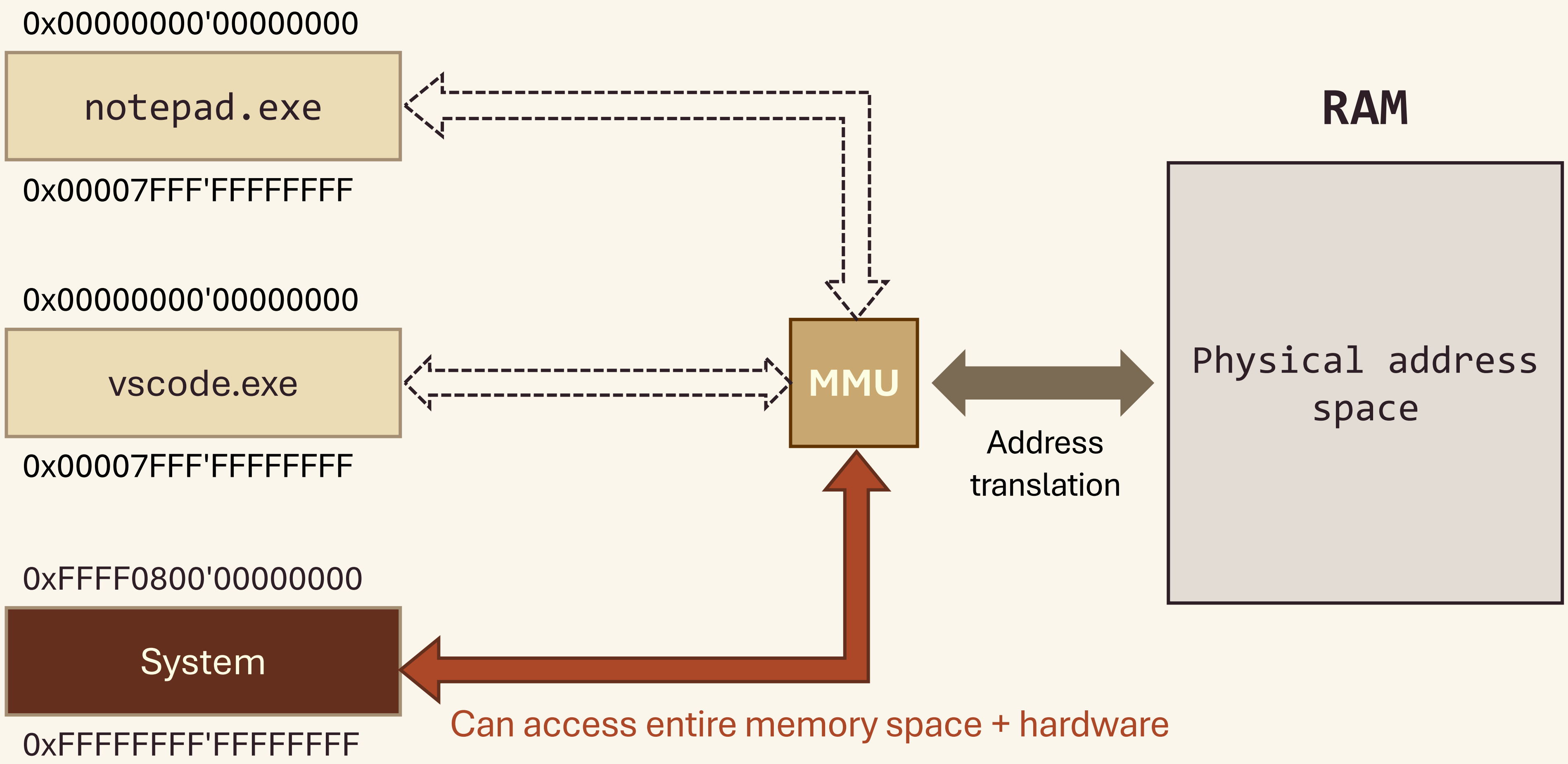
e.g., hal.dll, disk.sys, etc.

Then, what is the kernel?



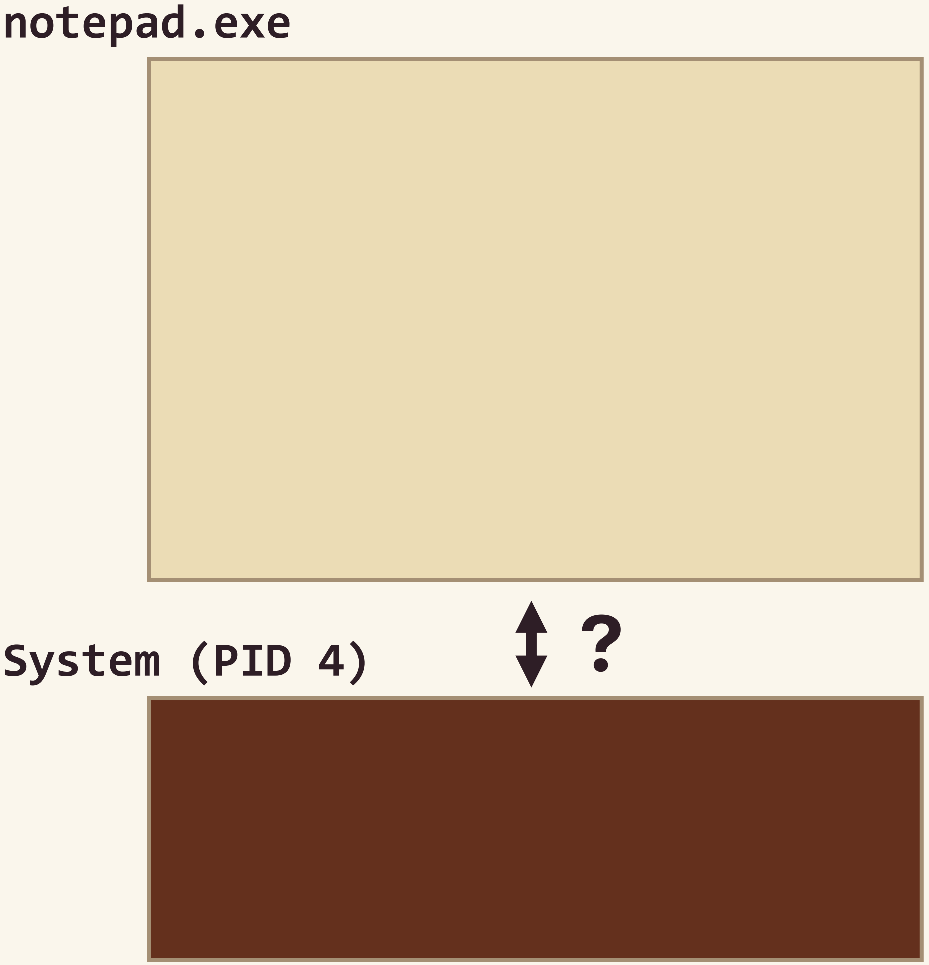
* The image for the kernel in the "System" process (PID 4) is primarily C:\Windows\System32\ntoskrnl.exe

Then, what is the kernel?

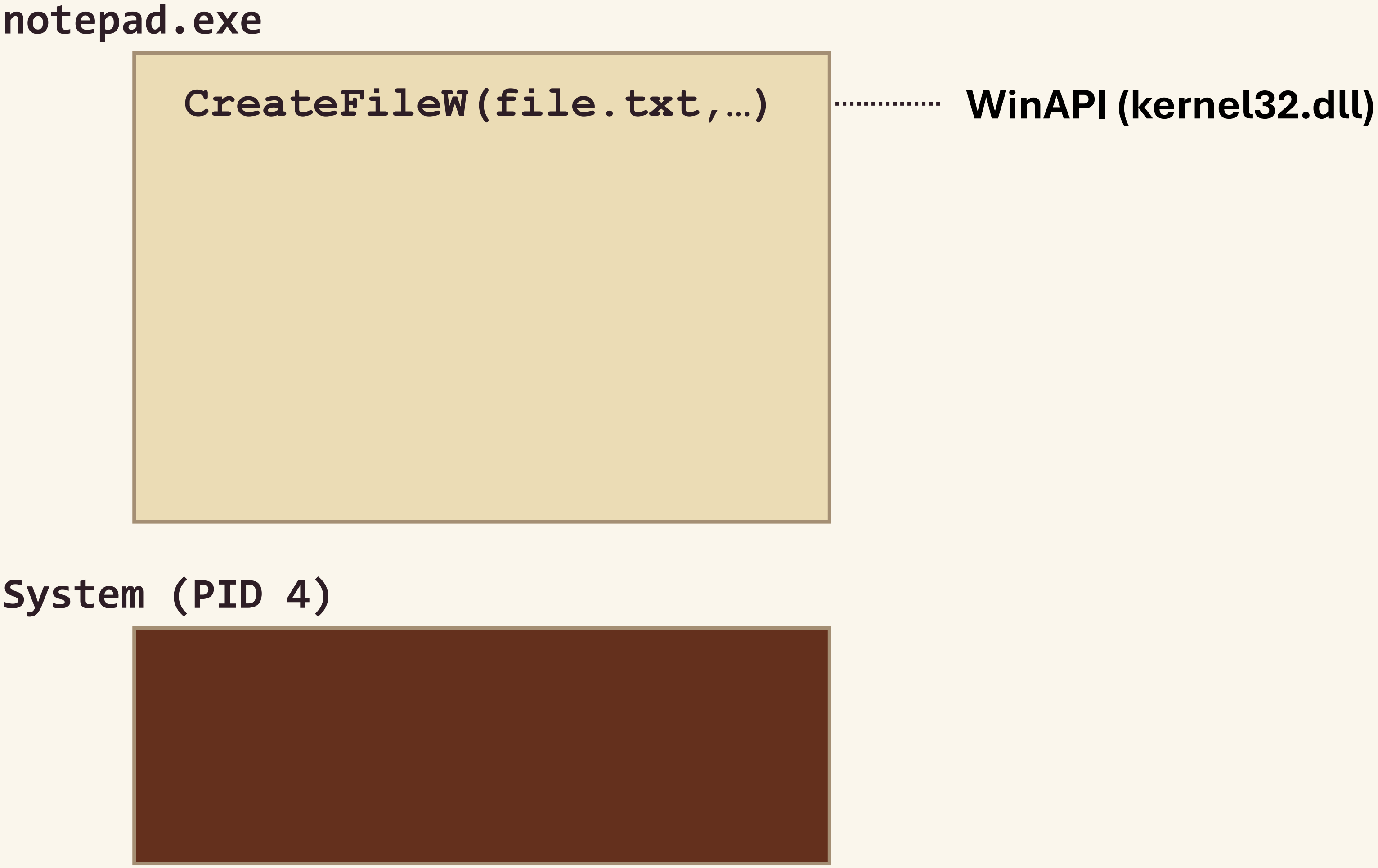


Can access entire memory space + hardware

Transition to kernel mode



Transition to kernel mode



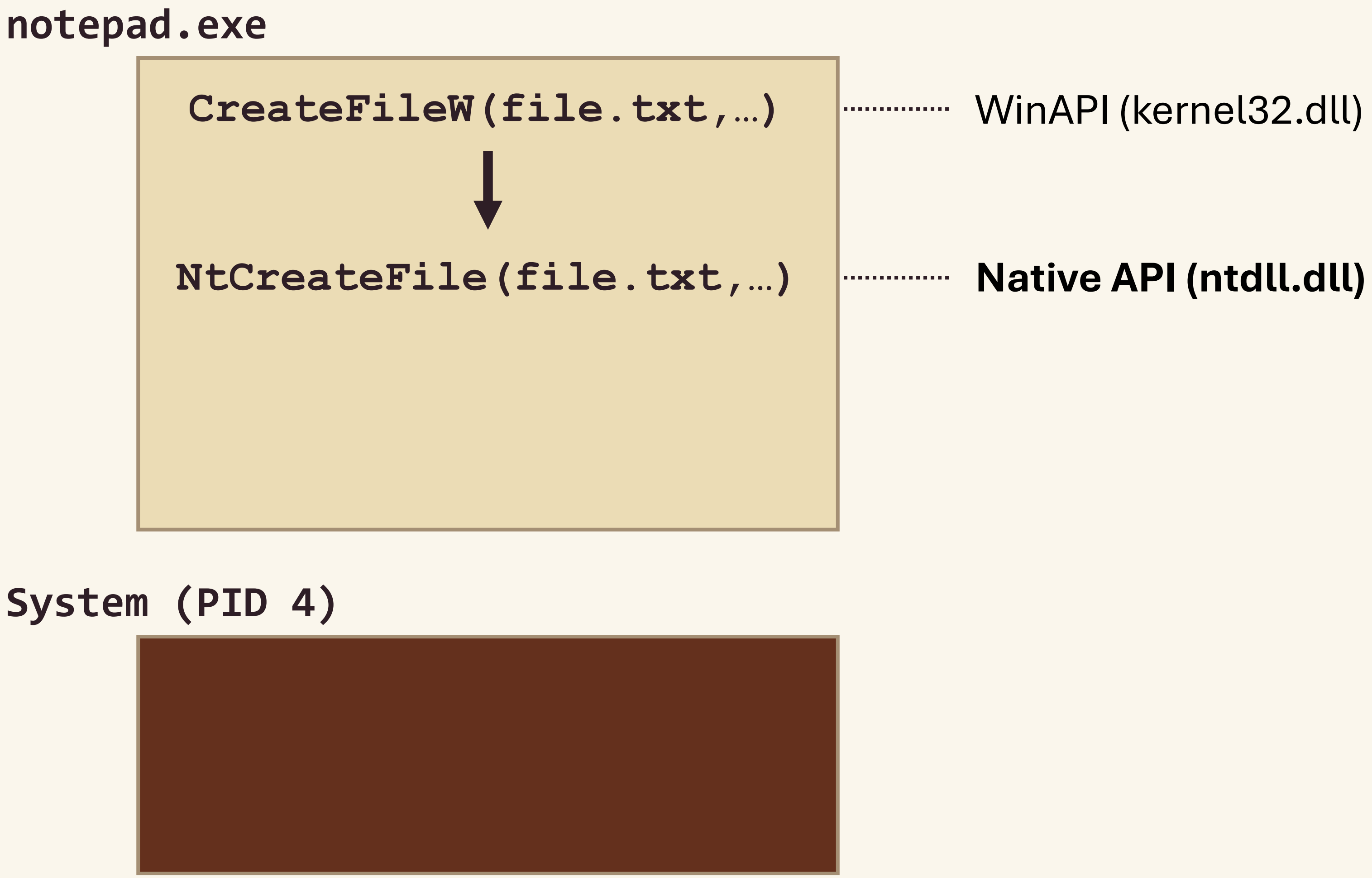
notepad.exe

CreateFileW(file.txt,…)

WinAPI (kernel32.dll)

System (PID 4)

Transition to kernel mode



notepad.exe

`CreateFileW(file.txt,…)`

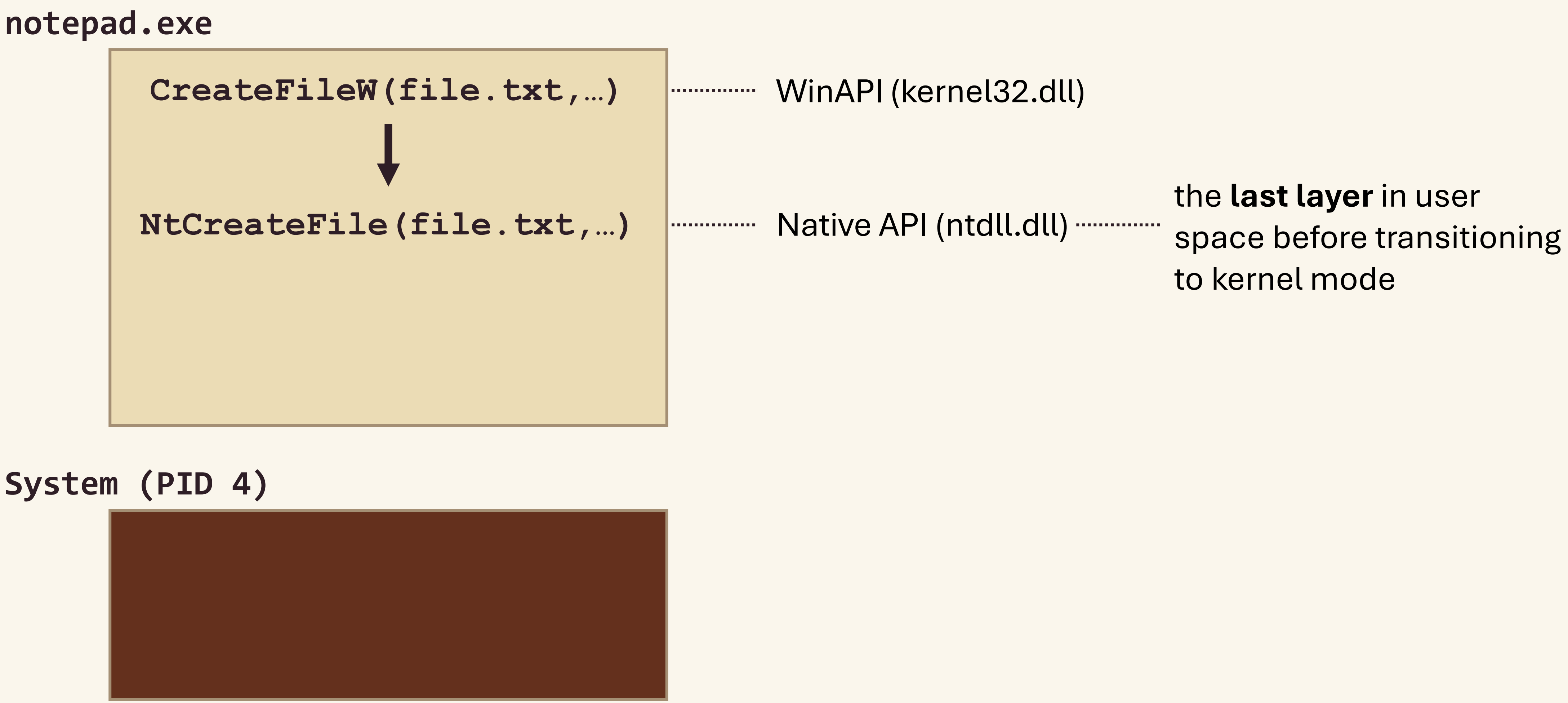
`NtCreateFile(file.txt,…)`

WinAPI (kernel32.dll)

Native API (ntdll.dll)

System (PID 4)

Transition to kernel mode



`notepad.exe`

`CreateFileW(file.txt,…)`

`NtCreateFile(file.txt,…)`

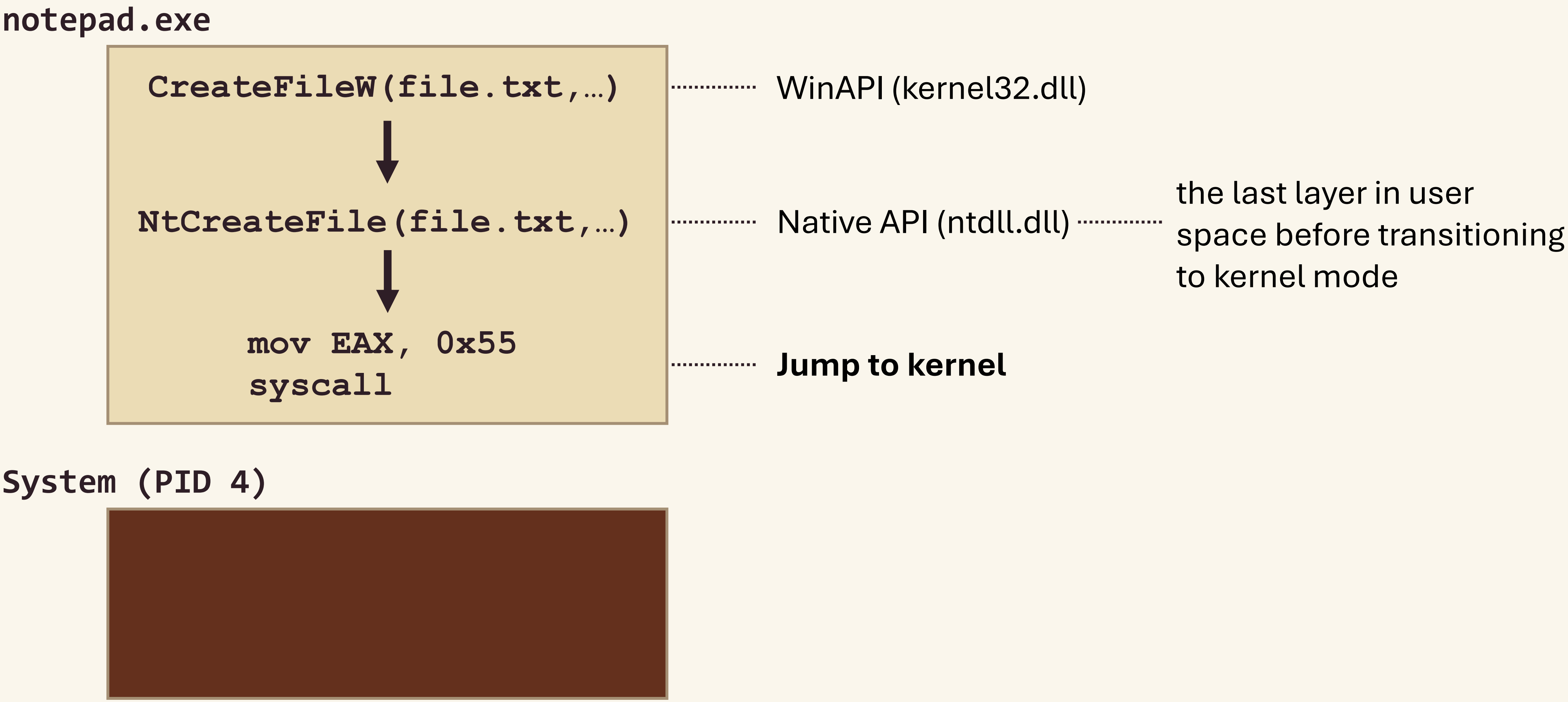
WinAPI (kernel32.dll)

Native API (ntdll.dll)

the **last layer** in user space before transitioning to kernel mode

`System (PID 4)`

Transition to kernel mode



notepad.exe

```
CreateFileW(file.txt,…)
```

WinAPI (kernel32.dll)

```
NtCreateFile(file.txt,…)
```

Native API (ntdll.dll)

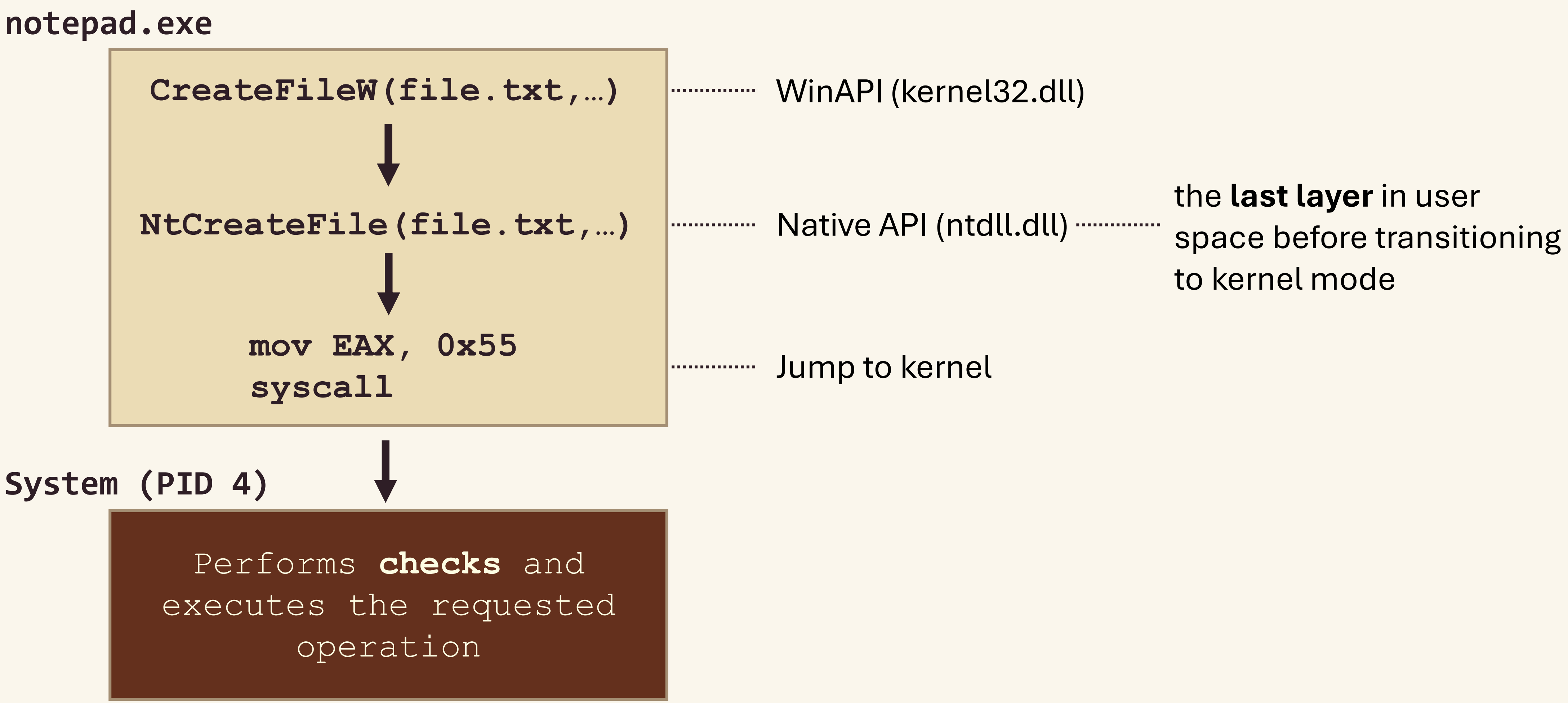
the last layer in user space before transitioning to kernel mode

```
mov EAX, 0x55
syscall
```

Jump to kernel

System (PID 4)

Transition to kernel mode



notepad.exe

```
CreateFileW(file.txt,…)
NtCreateFile(file.txt,…)
mov EAX, 0x55
syscall
```

WinAPI (kernel32.dll)

Native API (ntdll.dll)

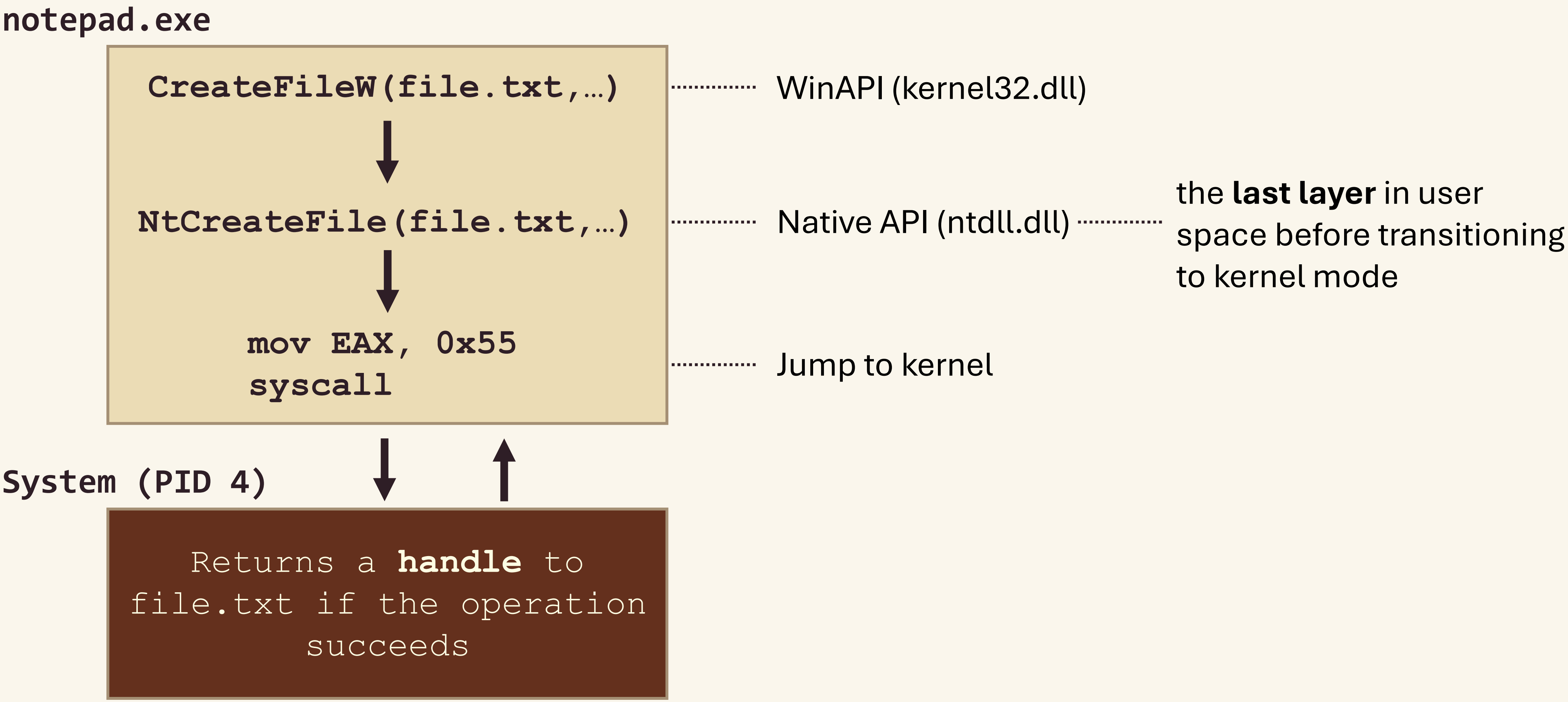
the **last layer** in user space before transitioning to kernel mode

Jump to kernel

System (PID 4)

Performs **checks** and executes the requested operation

Transition to kernel mode



notepad.exe

```
CreateFileW(file.txt,…)
```

WinAPI (kernel32.dll)

```
NtCreateFile(file.txt,…)
```

Native API (ntdll.dll)

the **last layer** in user space before transitioning to kernel mode

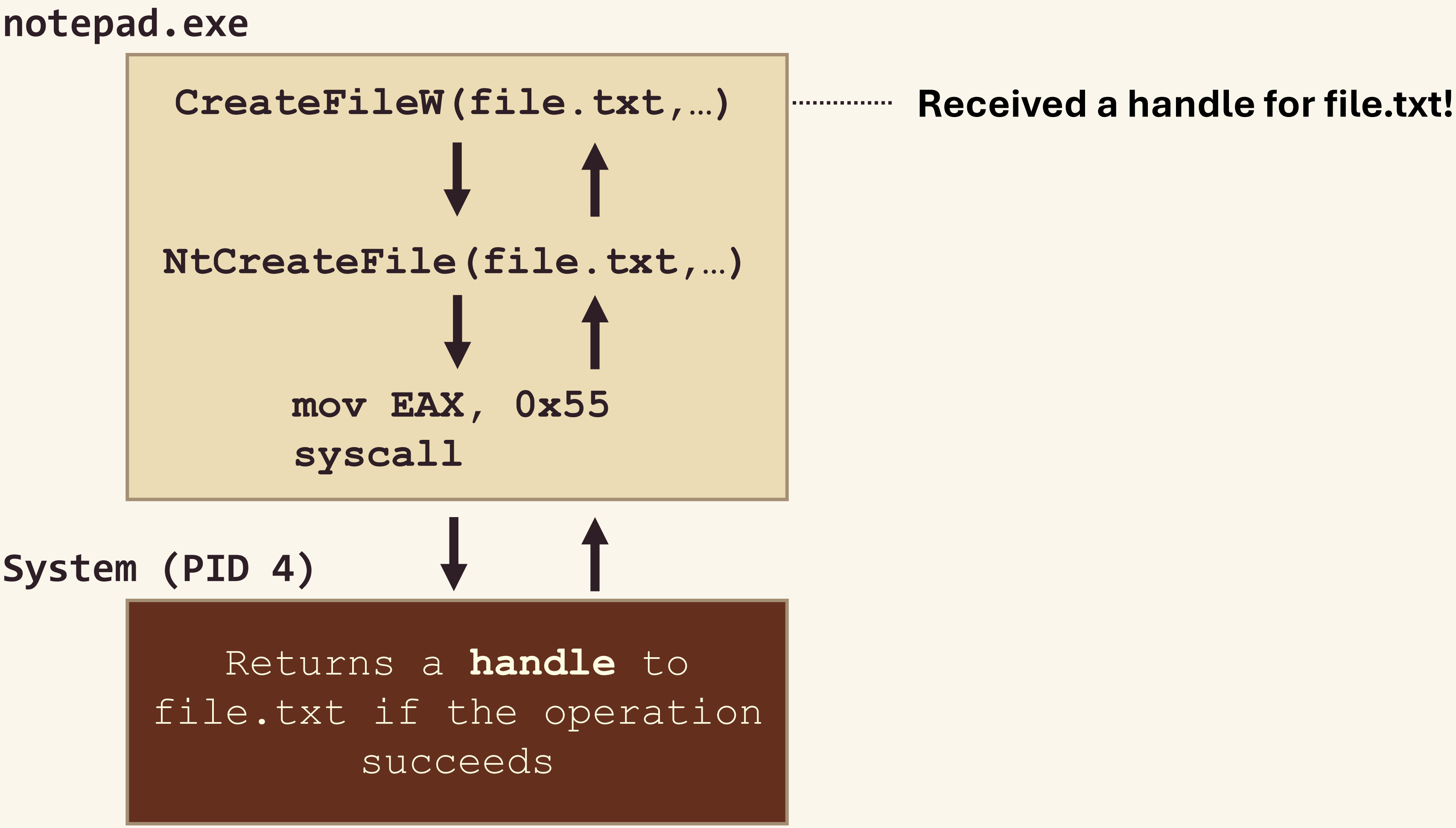
```
mov EAX, 0x55
syscall
```

Jump to kernel

System (PID 4)

Returns a **handle** to file.txt if the operation succeeds

Transition to kernel mode



notepad.exe

```
CreateFileW(file.txt,…)

NtCreateFile(file.txt,…)

mov EAX, 0x55
syscall
```

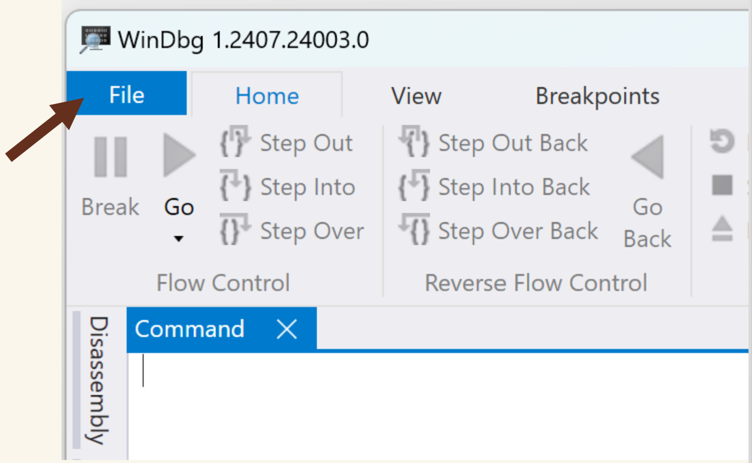
Received a handle for file.txt!

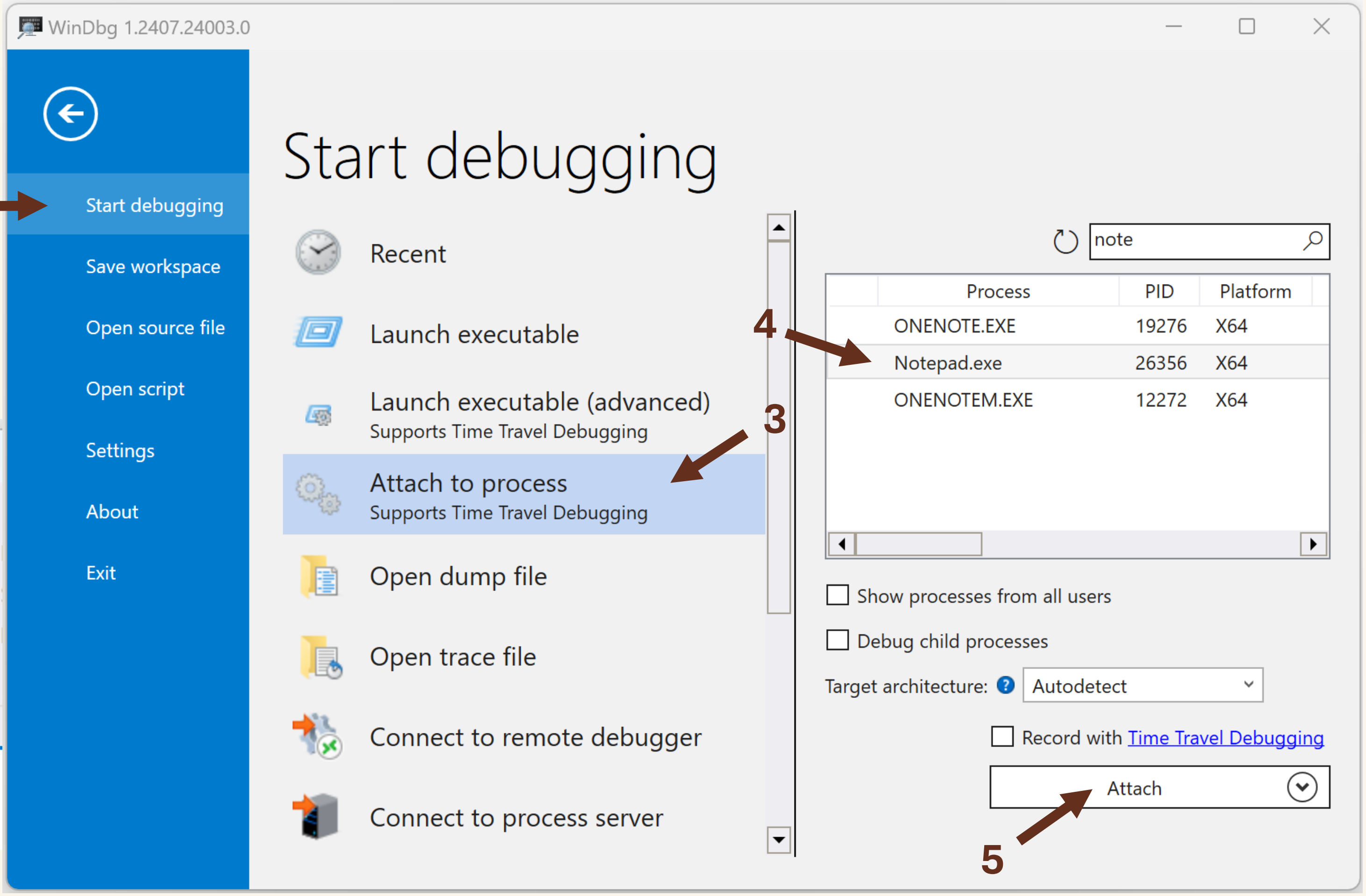
System (PID 4)

Returns a **handle** to file.txt if the operation succeeds

WinDbg practice: Attach to process

- Launch notepad.exe
- Launch WinDbg Preview
    1. File
    2. Start debugging
    3. Attach to process
    4. Select process
    5. Attach



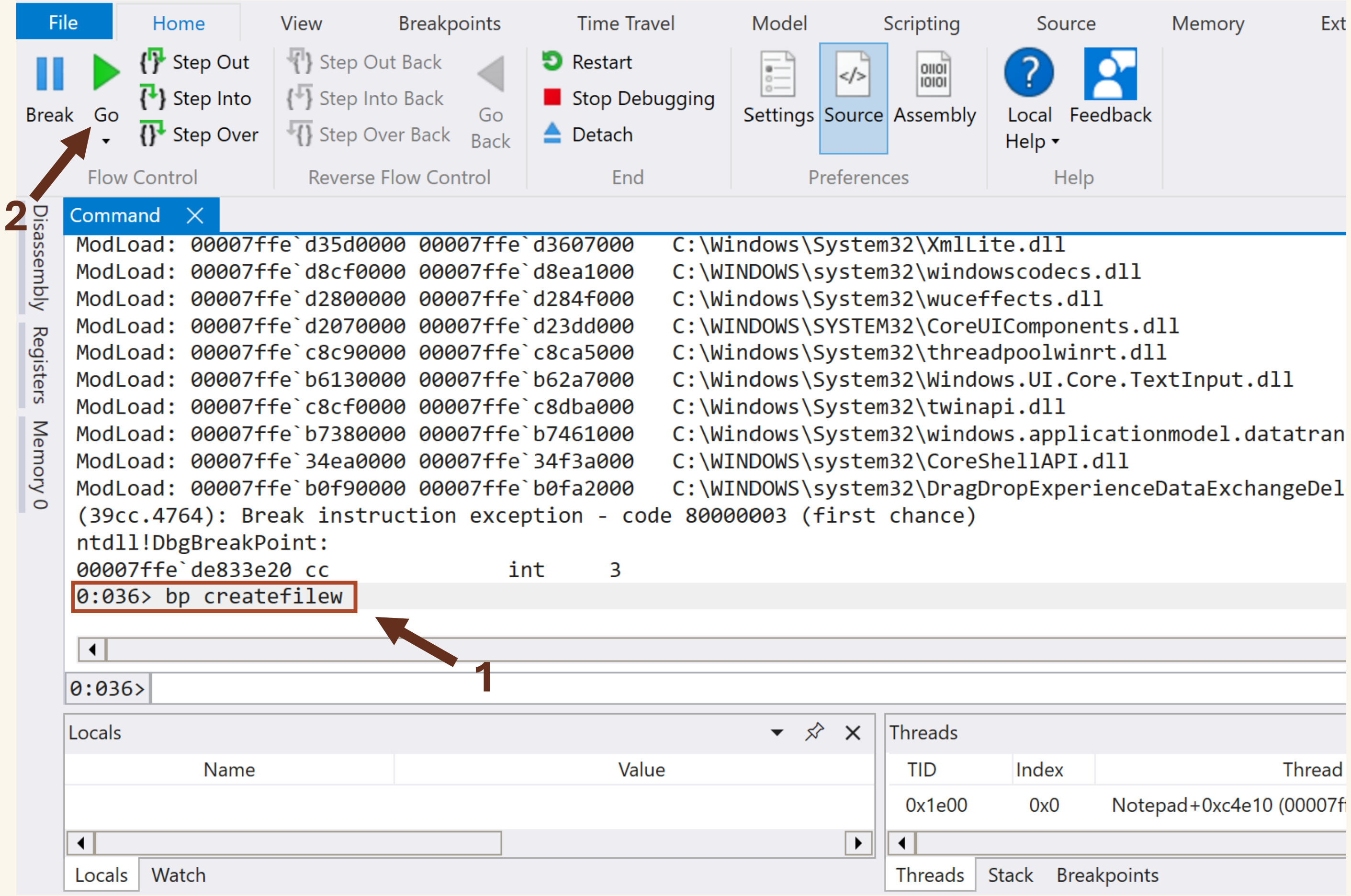


note

Process	PID	Platform
ONENOTE.EXE	19276	X64
Notepad.exe	26356	X64
ONENOTEM.EXE	12272	X64

Target architecture: Autodetect

WinDbg practice: Set a breakpoint



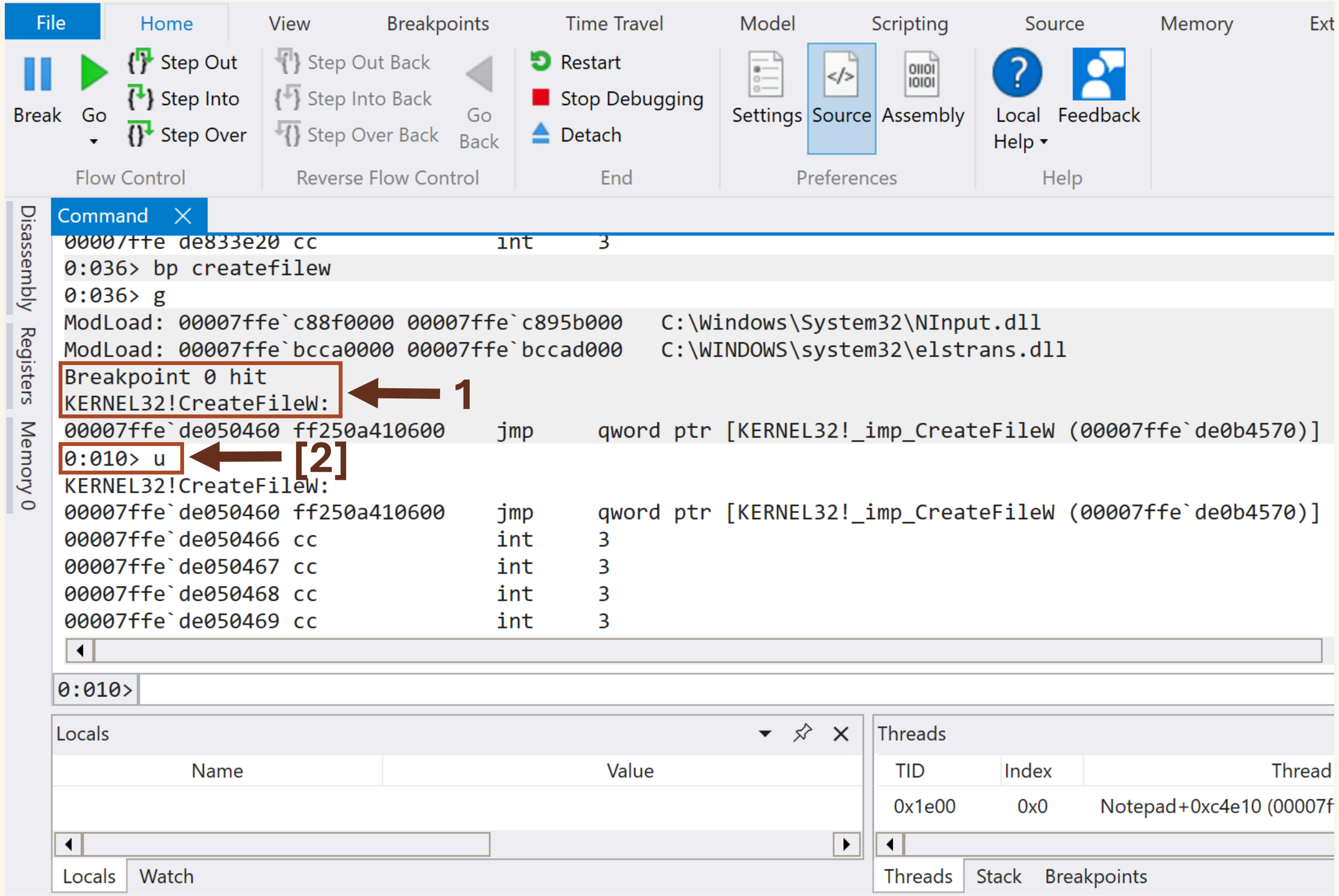
```
ModLoad: 00007ffe`d35d0000 00007ffe`d3607000   C:\Windows\System32\XmlLite.dll
ModLoad: 00007ffe`d8cf0000 00007ffe`d8ea1000   C:\WINDOWS\system32\windowscodecs.dll
ModLoad: 00007ffe`d2800000 00007ffe`d284f000   C:\Windows\System32\wuceffects.dll
ModLoad: 00007ffe`d2070000 00007ffe`d23dd000   C:\WINDOWS\SYSTEM32\CoreUIComponents.dll
ModLoad: 00007ffe`c8c90000 00007ffe`c8ca5000   C:\Windows\System32\threadpoolwinrt.dll
ModLoad: 00007ffe`b6130000 00007ffe`b62a7000   C:\Windows\System32\Windows.UI.Core.TextInput.dll
ModLoad: 00007ffe`c8cf0000 00007ffe`c8dba000   C:\Windows\System32\twinapi.dll
ModLoad: 00007ffe`b7380000 00007ffe`b7461000   C:\Windows\System32\windows.applicationmodel.datatran
ModLoad: 00007ffe`34ea0000 00007ffe`34f3a000   C:\WINDOWS\system32\CoreShellAPI.dll
ModLoad: 00007ffe`b0f90000 00007ffe`b0fa2000   C:\WINDOWS\system32\DragDropExperienceDataExchangeDel
(39cc.4764): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffe`de833e20 cc              int     3
0:036> bp createfilew

0:036>
```

Locals	
Name	Value

Threads		
TID	Index	Thread
0x1e00	0x0	Notepad+0xc4e10 (00007f

- Notepad pauses when WinDbg attaches
  1. Set a breakpoint
  2. Continue execution

WinDbg practice: Hit a breakpoint

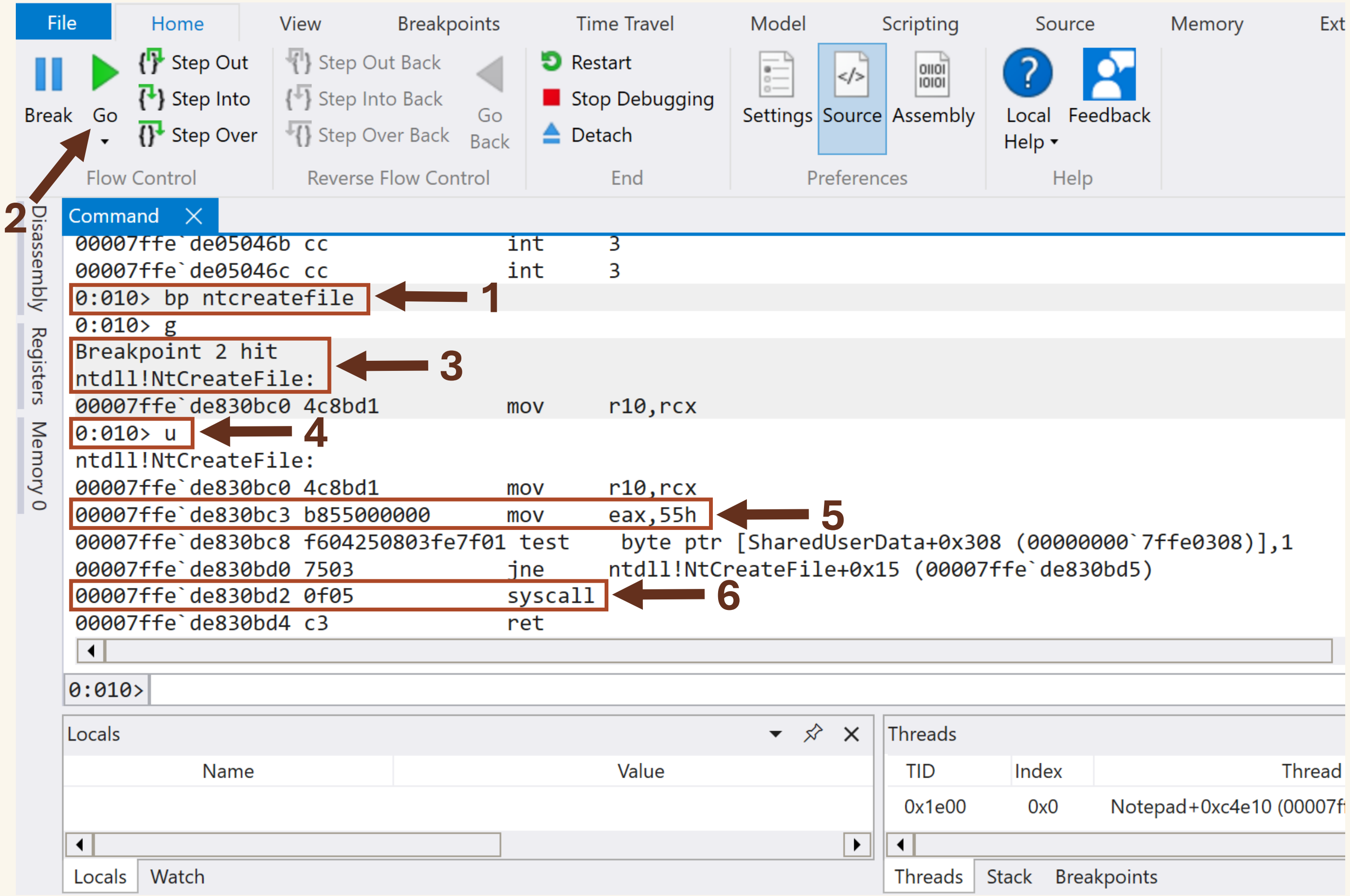


```
00007ffe`de833e20 cc              int     3
0:036> bp createfilew
0:036> g
ModLoad: 00007ffe`c88f0000 00007ffe`c895b000   C:\Windows\System32\NInput.dll
ModLoad: 00007ffe`bcca0000 00007ffe`bccad000   C:\WINDOWS\system32\elstrans.dll
Breakpoint 0 hit
KERNEL32!CreateFileW:
00007ffe`de050460 ff250a410600    jmp     qword ptr [KERNEL32!_imp_CreateFileW (00007ffe`de0b4570)]
0:010> u
KERNEL32!CreateFileW:
00007ffe`de050460 ff250a410600    jmp     qword ptr [KERNEL32!_imp_CreateFileW (00007ffe`de0b4570)]
00007ffe`de050466 cc              int     3
00007ffe`de050467 cc              int     3
00007ffe`de050468 cc              int     3
00007ffe`de050469 cc              int     3

0:010>
```

TID	Index	Thread
0x1e00	0x0	Notepad+0xc4e10 (00007f

- Open a file or create a new tab in notepad
    1. Breakpoint is hit!
    2. [Disassemble instructions at the current address to show the assembly code of **CreateFileW**]
- **CreateFileW** doesn't directly interact with the kernel, so we set a new breakpoint at ntdll's **NtCreateFile**

WinDbg practice: Get syscall number

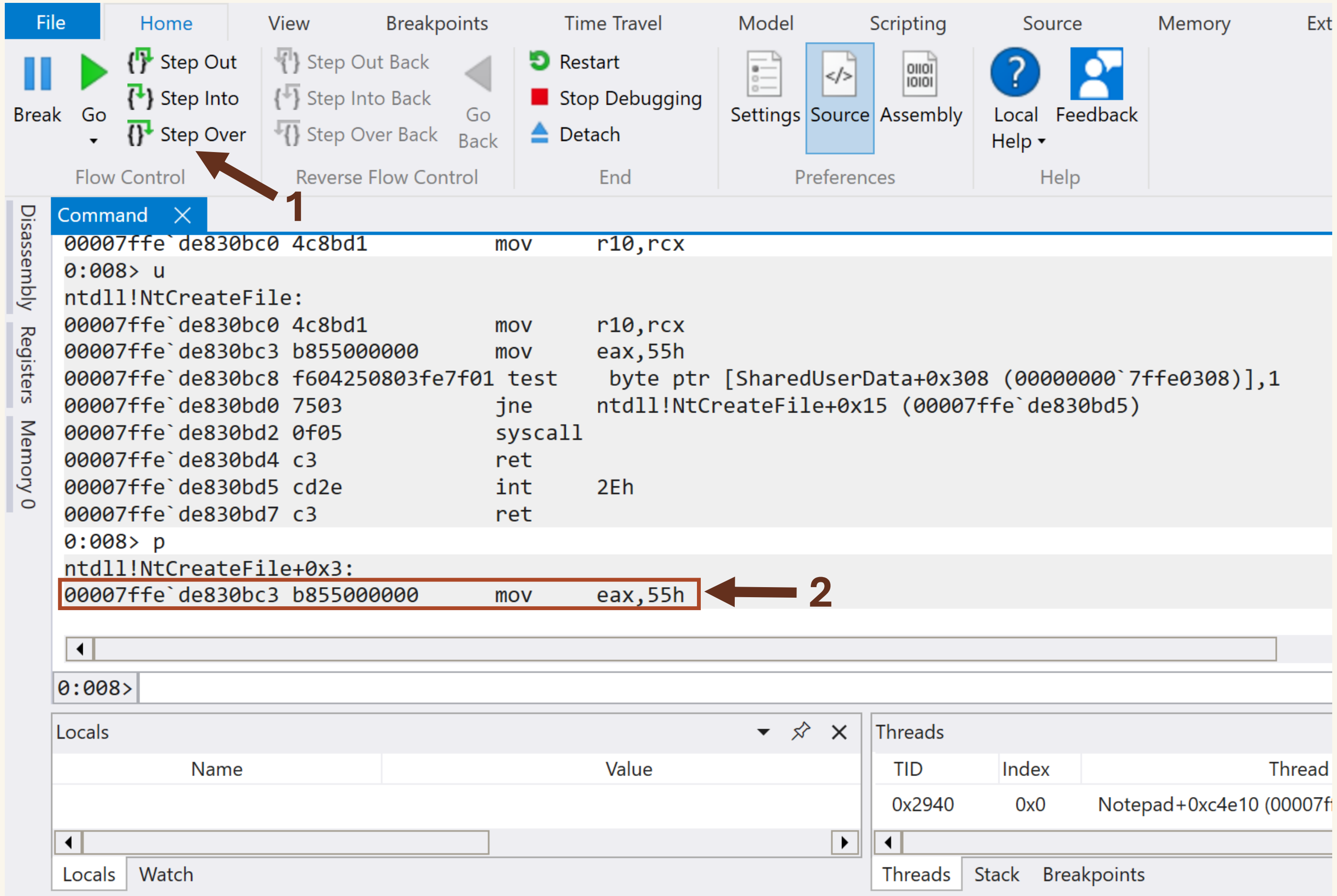


```
00007ffe`de05046b cc              int     3
00007ffe`de05046c cc              int     3
0:010> bp ntcreatefile
0:010> g
Breakpoint 2 hit
ntdll!NtCreateFile:
00007ffe`de830bc0 4c8bd1          mov     r10,rcx
0:010> u
ntdll!NtCreateFile:
00007ffe`de830bc0 4c8bd1          mov     r10,rcx
00007ffe`de830bc3 b855000000      mov     eax,55h
00007ffe`de830bc8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`de830bd0 7503            jne     ntdll!NtCreateFile+0x15 (00007ffe`de830bd5)
00007ffe`de830bd2 0f05            syscall
00007ffe`de830bd4 c3              ret
0:010>
```

TID	Index	Thread
0x1e00	0x0	Notepad+0xc4e10 (00007ff

1. Sets a breakpoint at the **NtCreateFile** function in **ntdll.dll**
2. Continue execution
3. Breakpoint at **NtCreateFile** is hit!
4. Disassemble instructions at the current address to show the assembly code of **NtCreateFile**
5. Get syscall number: look for the **mov eax, 55h** instruction, which loads the **syscall number (0x55)** into the **EAX** register
6. The **syscall** instruction triggers the transition to **kernel mode**

WinDbg practice: Step over



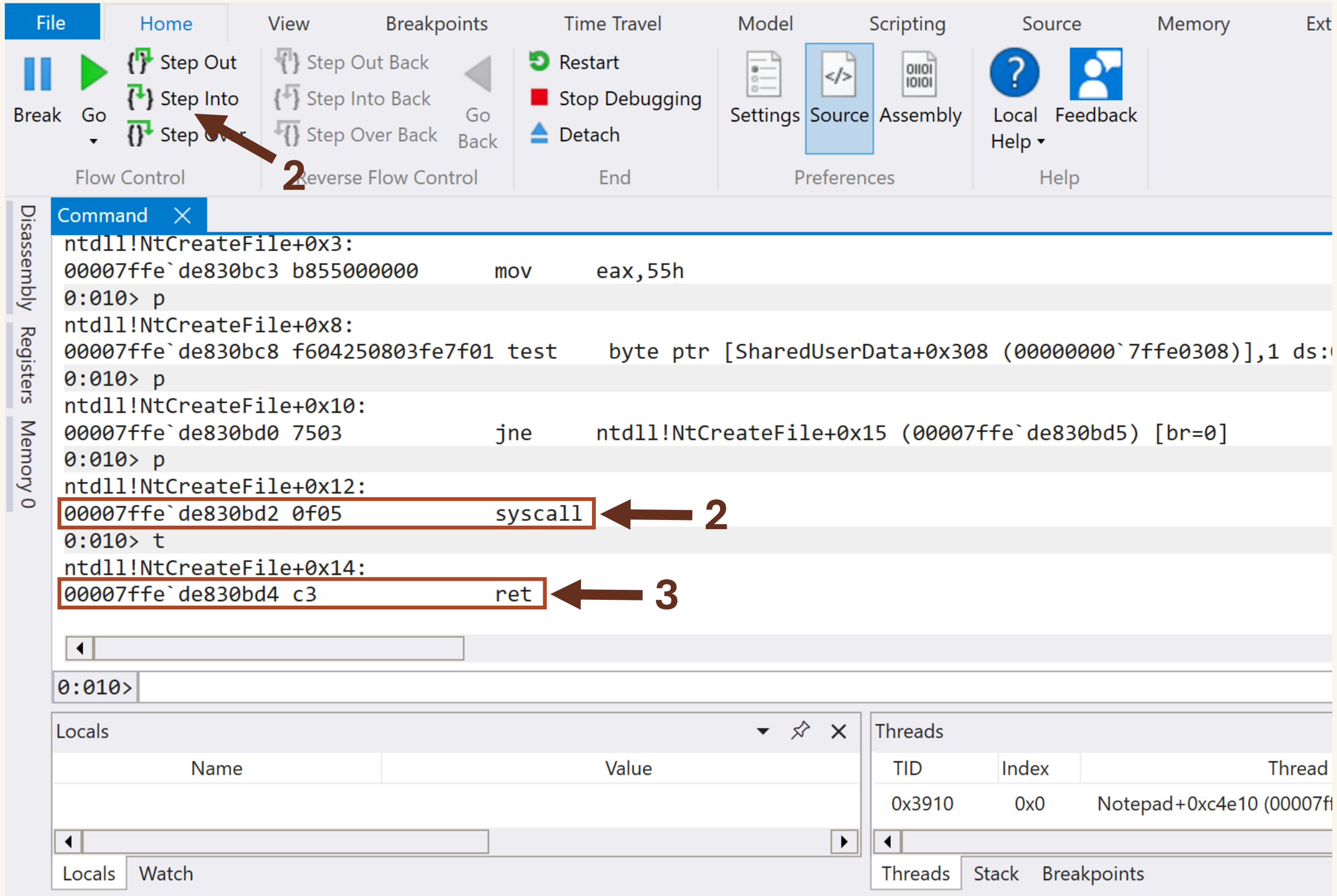
```
00007ffe`de830bc0 4c8bd1          mov     r10,rcx
0:008> u
ntdll!NtCreateFile:
00007ffe`de830bc0 4c8bd1          mov     r10,rcx
00007ffe`de830bc3 b855000000      mov     eax,55h
00007ffe`de830bc8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`de830bd0 7503            jne     ntdll!NtCreateFile+0x15 (00007ffe`de830bd5)
00007ffe`de830bd2 0f05            syscall
00007ffe`de830bd4 c3              ret
00007ffe`de830bd5 cd2e            int     2Eh
00007ffe`de830bd7 c3              ret
0:008> p
ntdll!NtCreateFile+0x3:
00007ffe`de830bc3 b855000000      mov     eax,55h
```

```
0:008>
```

1. The **"Step Over"** action proceeds to the next instruction without entering functions

2. After stepping over, the debugger shows the next instruction to be executed: **mov eax, 55h**. This sets the syscall number **(0x55)** in the **EAX** register, which the kernel uses to **identify the NtCreateFile request**

WinDbg practice: Step into



```
ntdll!NtCreateFile+0x3:
00007ffe`de830bc3 b855000000       mov     eax,55h
0:010> p
ntdll!NtCreateFile+0x8:
00007ffe`de830bc8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1 ds:
0:010> p
ntdll!NtCreateFile+0x10:
00007ffe`de830bd0 7503             jne     ntdll!NtCreateFile+0x15 (00007ffe`de830bd5) [br=0]
0:010> p
ntdll!NtCreateFile+0x12:
00007ffe`de830bd2 0f05             syscall
0:010> t
ntdll!NtCreateFile+0x14:
00007ffe`de830bd4 c3               ret
```

```
0:010>
```

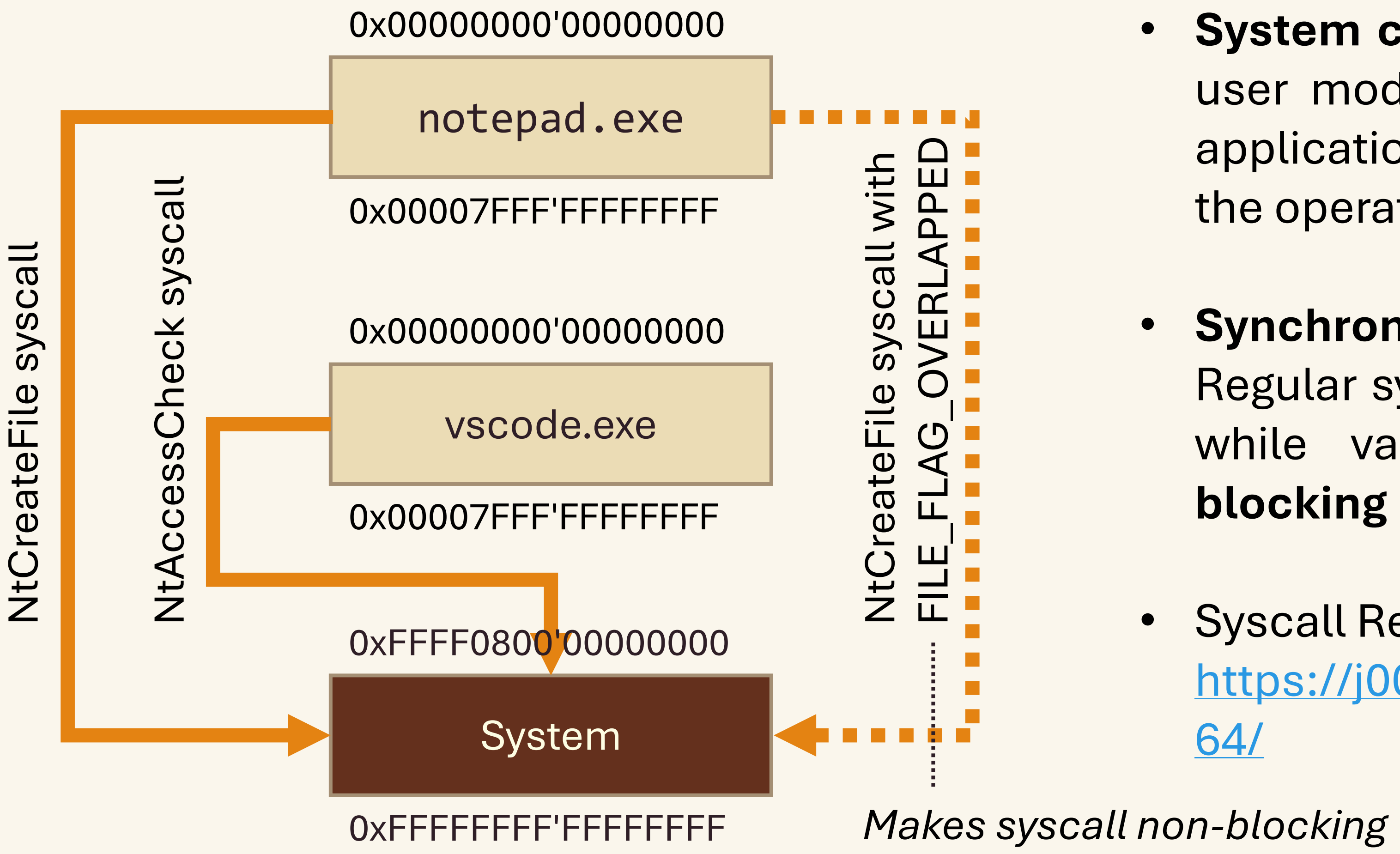
TID	Index	Thread
0x3910	0x0	Notepad+0xc4e10 (00007ff

1. **"Step Over"** until the syscall instruction

2. When the **syscall** is about to execute, try to **"Step Into"** to attempt to observe what happens inside the kernel

3. The kernel code cannot be stepped into in user-mode debugging; **"Step Into"** behaves like **"Step Over"** in WinDbg because it **does not transition into kernel-mode code during user-mode debugging**

System Calls: User to Kernel Mode



- **System calls** act as **"gateways"** from user mode to kernel mode, enabling applications to request services from the operating system
- **Synchronous vs. Asynchronous:** Regular syscalls **wait for completion**, while various options allow **non-blocking behavior**
- Syscall Reference Guide: https://j00ru.vexillium.org/syscalls/nt/64/

Intro to kernel debugging

WinDbg can operate in either user mode or kernel mode, but **not in both simultaneously**.

Intro to kernel debugging

WinDbg can operate in either user mode or kernel mode, but not in both simultaneously.

Local kernel debugging

Remote kernel debugging

Intro to kernel debugging

WinDbg can operate in either user mode or kernel mode, but not in both simultaneously.

Local kernel debugging

Remote kernel debugging *(over network, USB, 1394, and serial connections)*

Intro to kernel debugging

WinDbg can operate in either user mode or kernel mode, but not in both simultaneously.

Local kernel debugging

- 🤔 View kernel objects *(reliable with restrictions)*

Remote kernel debugging *(over network, USB, 1394, and serial connections)*

- 😊 View kernel objects

Intro to kernel debugging

WinDbg can operate in either user mode or kernel mode, but not in both simultaneously.

Local kernel debugging

- 🤔 View kernel objects *(reliable with restrictions)*
- 😟 Cannot use breakpoints

Remote kernel debugging *(over network, USB, 1394, and serial connections)*

- 😊 View kernel objects
- 😊 Set breakpoints

Intro to kernel debugging

WinDbg can operate in either user mode or kernel mode, but not in both simultaneously.

Local kernel debugging

- 🤔 View kernel objects *(reliable with restrictions)*
- 😟 Cannot use breakpoints
- 😊 Only one host is needed

Remote kernel debugging *(over network, USB, 1394, and serial connections)*

- 😊 View kernel objects
- 😊 Set breakpoints
- 😟 Requires two hosts

Intro to kernel debugging

WinDbg can operate in either user mode or kernel mode, but not in both simultaneously.

Local kernel debugging ← **Let’s try this for now**

- 🤔 View kernel objects *(reliable with restrictions)*
- 😟 Cannot use breakpoints
- 😊 Only one host is needed

Remote kernel debugging *(over network, USB, 1394, and serial connections)*

- 😊 View kernel objects
- 😊 Set breakpoints
- 😟 Requires two hosts

WinDbg practice: VM preparations

If using own Windows 11 VM:

- **Disable secure boot** in VM settings
  VMWare: Settings → Options → Advanced → UEFI → Uncheck “Enable secure boot”

- Start VM, run cmd.exe as an Administrator and **enable debugging** by entering:
  bcdedit /set debug on
  You should get “The operation completed successfully.”

- Install **WinDbg Preview** from Microsoft Store

- Enjoy!

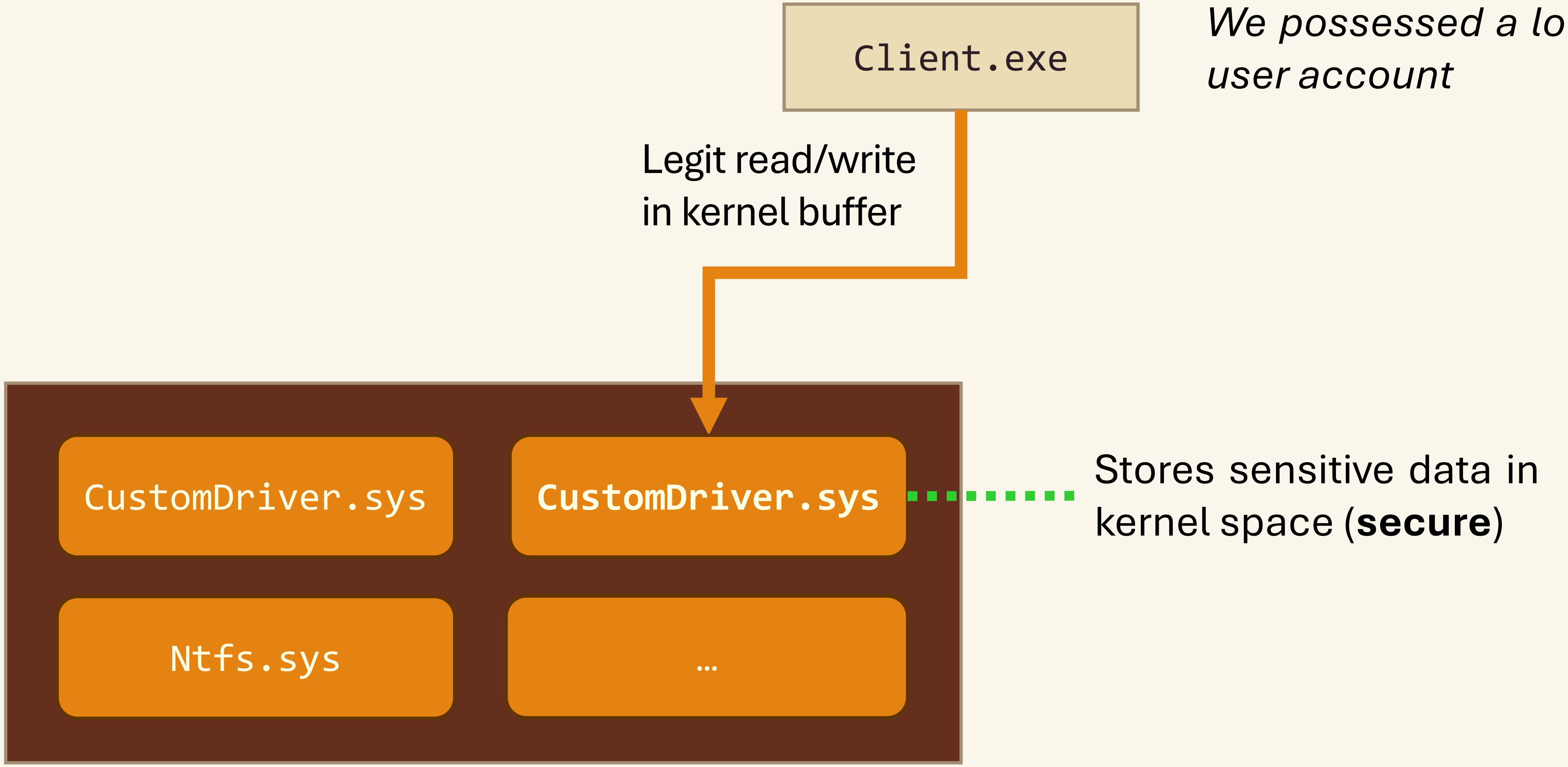
Alternatively download preconfigured VM from https://tinyurl.com/axkc9txy

WinDbg practice: Attach to kernel

WinDbg practice: View SSDT

WinDbg practice: View Process List

Real-world example



Client.exe

We possessed a low-privileged user account

Legit read/write in kernel buffer

CustomDriver.sys

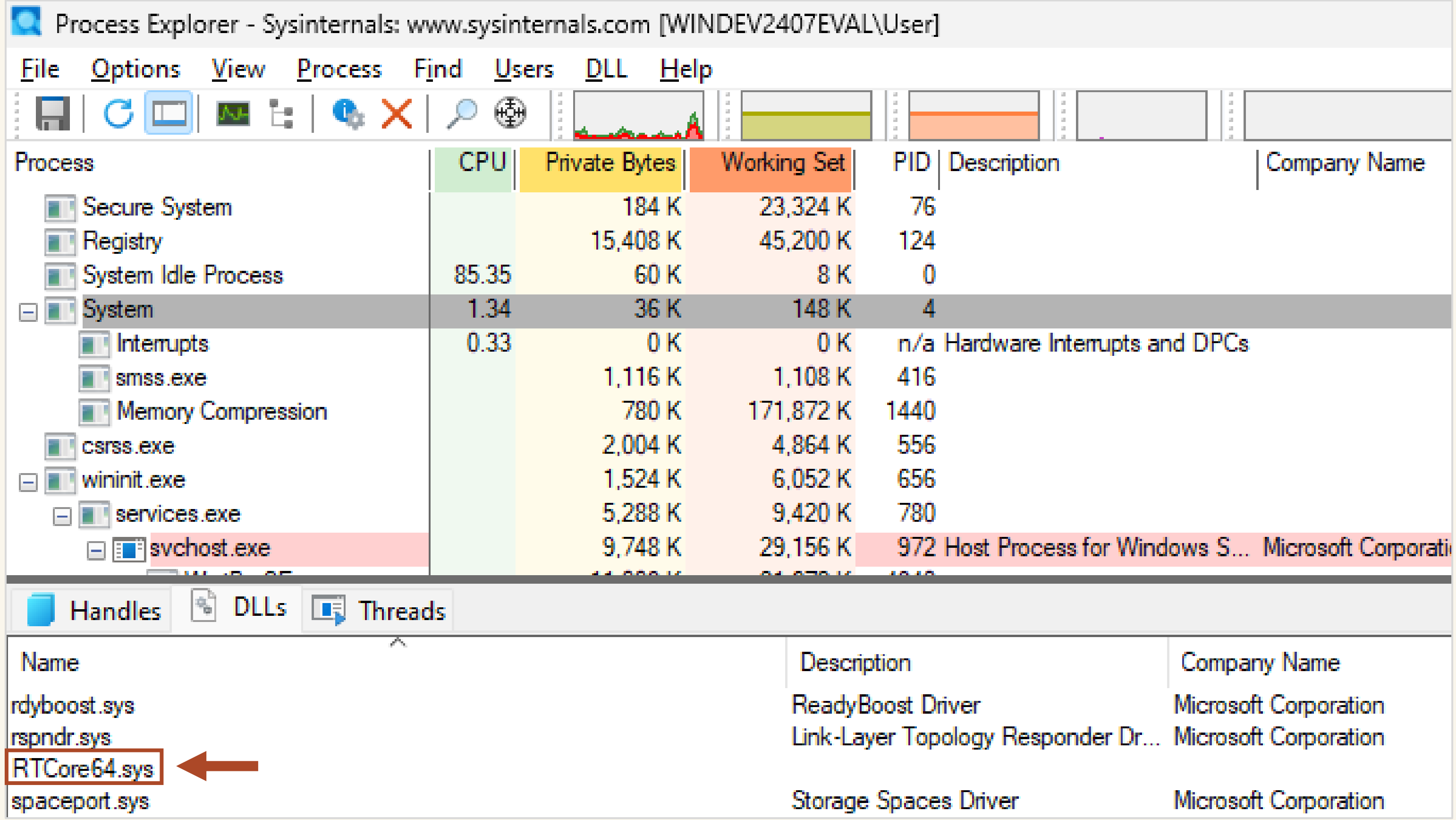
CustomDriver.sys

Ntfs.sys

…

Stores sensitive data in kernel space (**secure**)

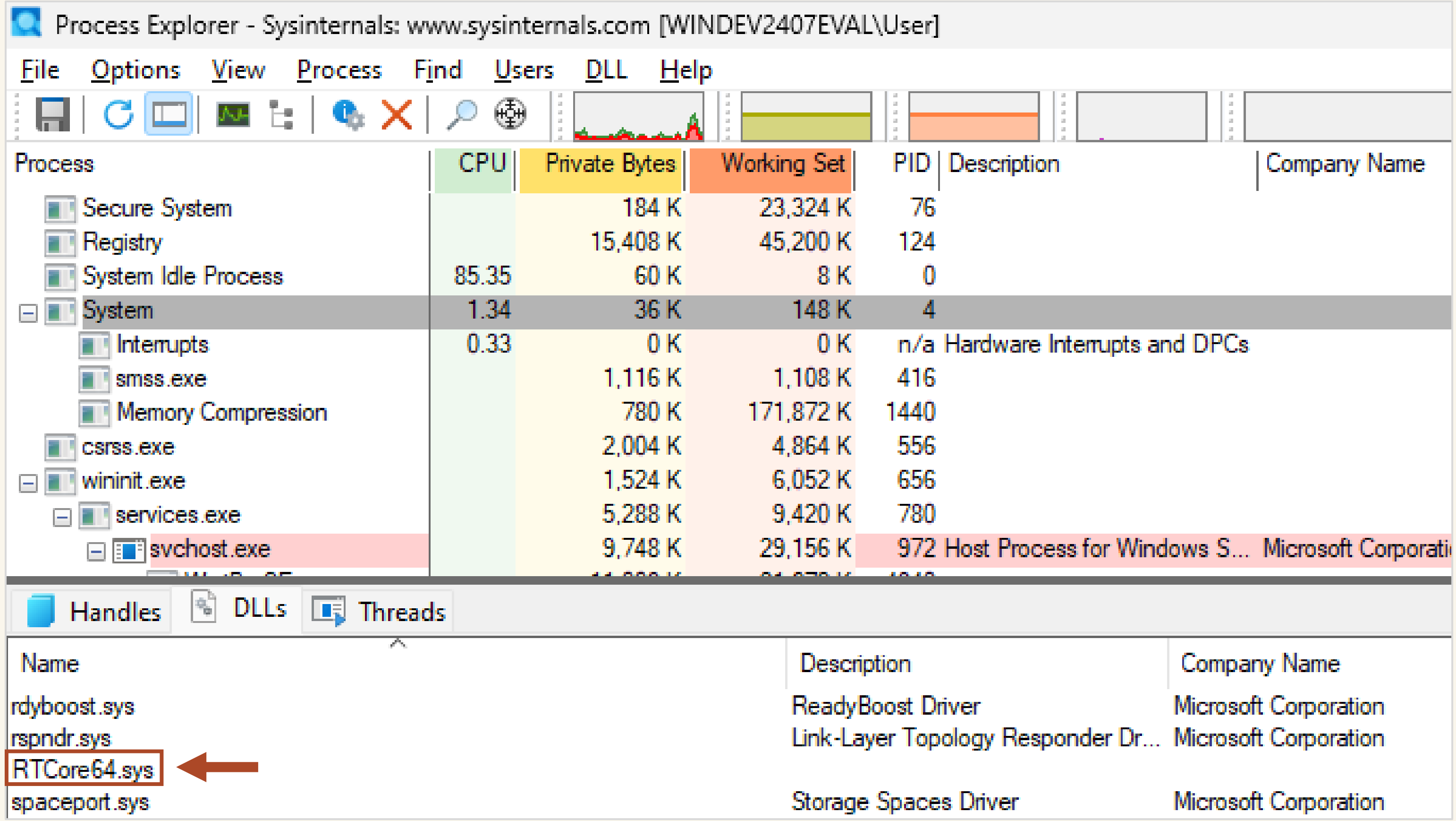
Real-world example


Process Explorer - Sysinternals: www.sysinternals.com [WINDEV2407EVAL\User]
File
Options
View
Process
Find
Users
DLL
Help
Handles
DLLs
Threads

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
Secure System		184 K	23,324 K	76		
Registry		15,408 K	45,200 K	124		
System Idle Process	85.35	60 K	8 K	0		
System	1.34	36 K	148 K	4		
Interrupts	0.33	0 K	0 K	n/a	Hardware Interrupts and DPCs	
smss.exe		1,116 K	1,108 K	416		
Memory Compression		780 K	171,872 K	1440		
csrss.exe		2,004 K	4,864 K	556		
wininit.exe		1,524 K	6,052 K	656		
services.exe		5,288 K	9,420 K	780		
svchost.exe		9,748 K	29,156 K	972	Host Process for Windows S...	Microsoft Corporati

Name	Description	Company Name
rdyboost.sys	ReadyBoost Driver	Microsoft Corporation
rspndr.sys	Link-Layer Topology Responder Dr...	Microsoft Corporation
RTCore64.sys		
spaceport.sys	Storage Spaces Driver	Microsoft Corporation

Real-world example

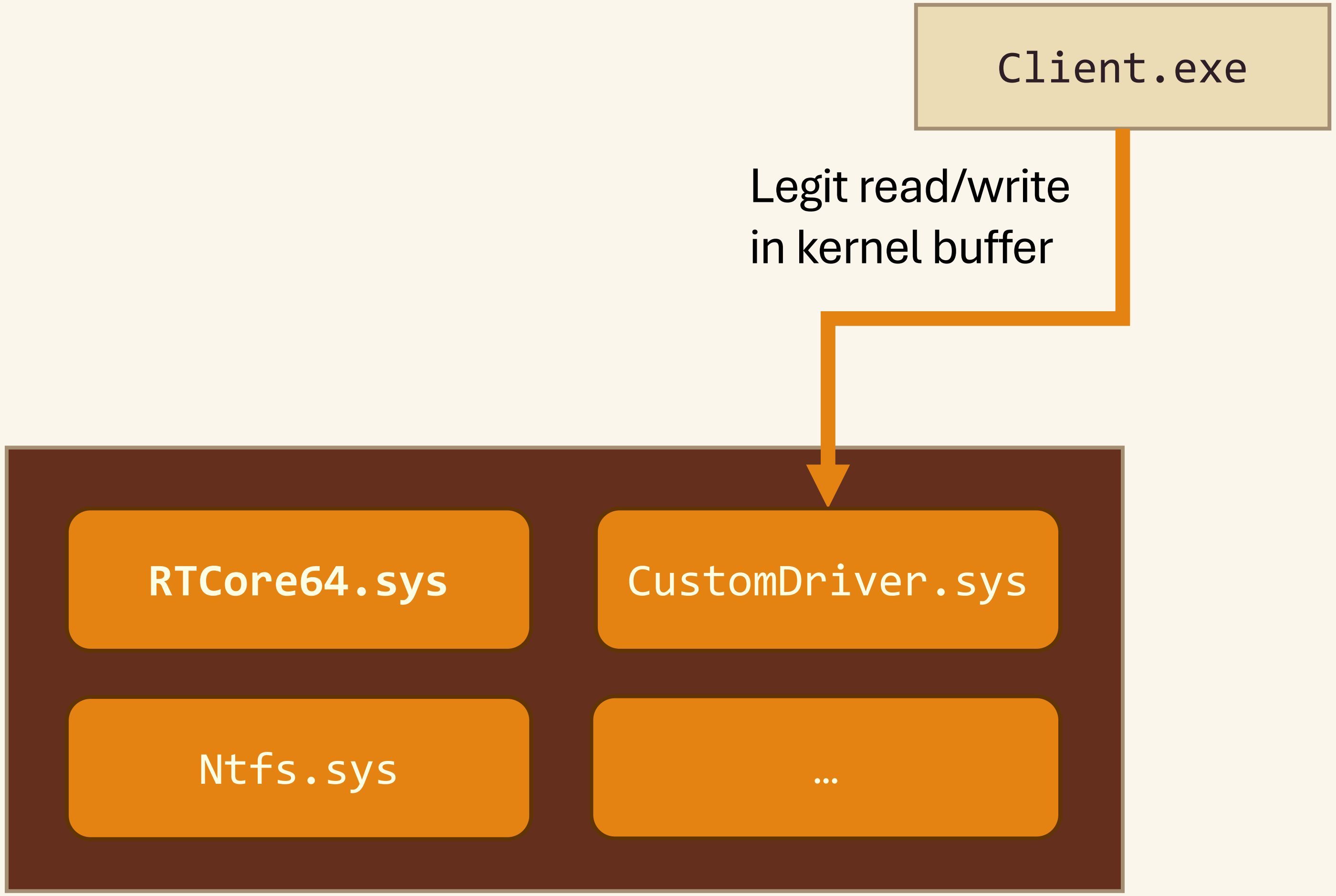

Process Explorer - Sysinternals: www.sysinternals.com [WINDEV2407EVAL\User]
File
Options
View
Process
Find
Users
DLL
Help
Handles
DLLs
Threads

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
Secure System		184 K	23,324 K	76		
Registry		15,408 K	45,200 K	124		
System Idle Process	85.35	60 K	8 K	0		
System	1.34	36 K	148 K	4		
Interrupts	0.33	0 K	0 K	n/a	Hardware Interrupts and DPCs	
smss.exe		1,116 K	1,108 K	416		
Memory Compression		780 K	171,872 K	1440		
csrss.exe		2,004 K	4,864 K	556		
wininit.exe		1,524 K	6,052 K	656		
services.exe		5,288 K	9,420 K	780		
svchost.exe		9,748 K	29,156 K	972	Host Process for Windows S...	Microsoft Corporati

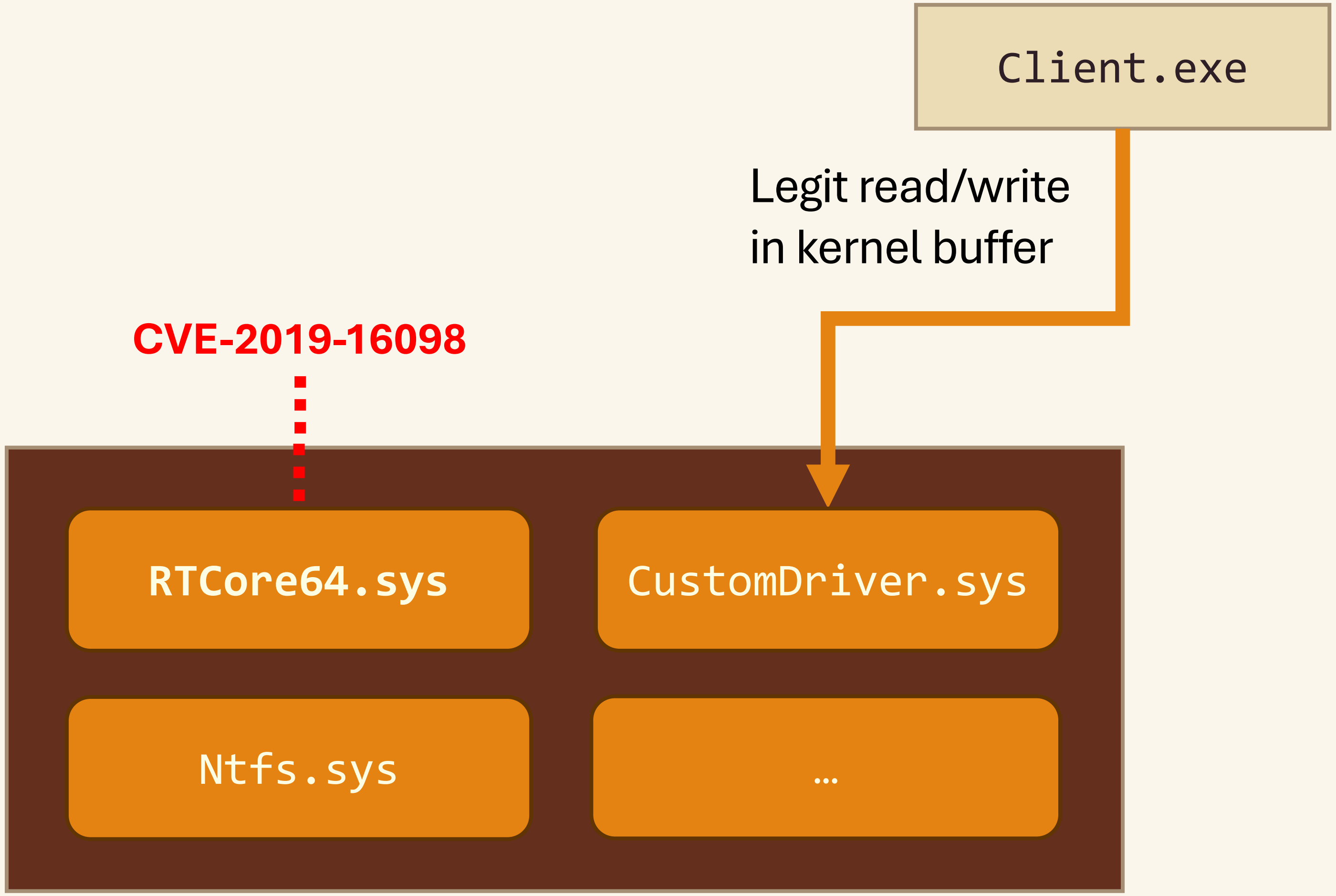
Name	Description	Company Name
rdyboost.sys	ReadyBoost Driver	Microsoft Corporation
rspndr.sys	Link-Layer Topology Responder Dr...	Microsoft Corporation
RTCore64.sys		
spaceport.sys	Storage Spaces Driver	Microsoft Corporation

The **RTCore64.sys** driver is part of the MSI Afterburner and RivaTuner software packages. This driver provides **low-level hardware access for monitoring and overclocking features** on a Windows system, specifically for graphics cards.

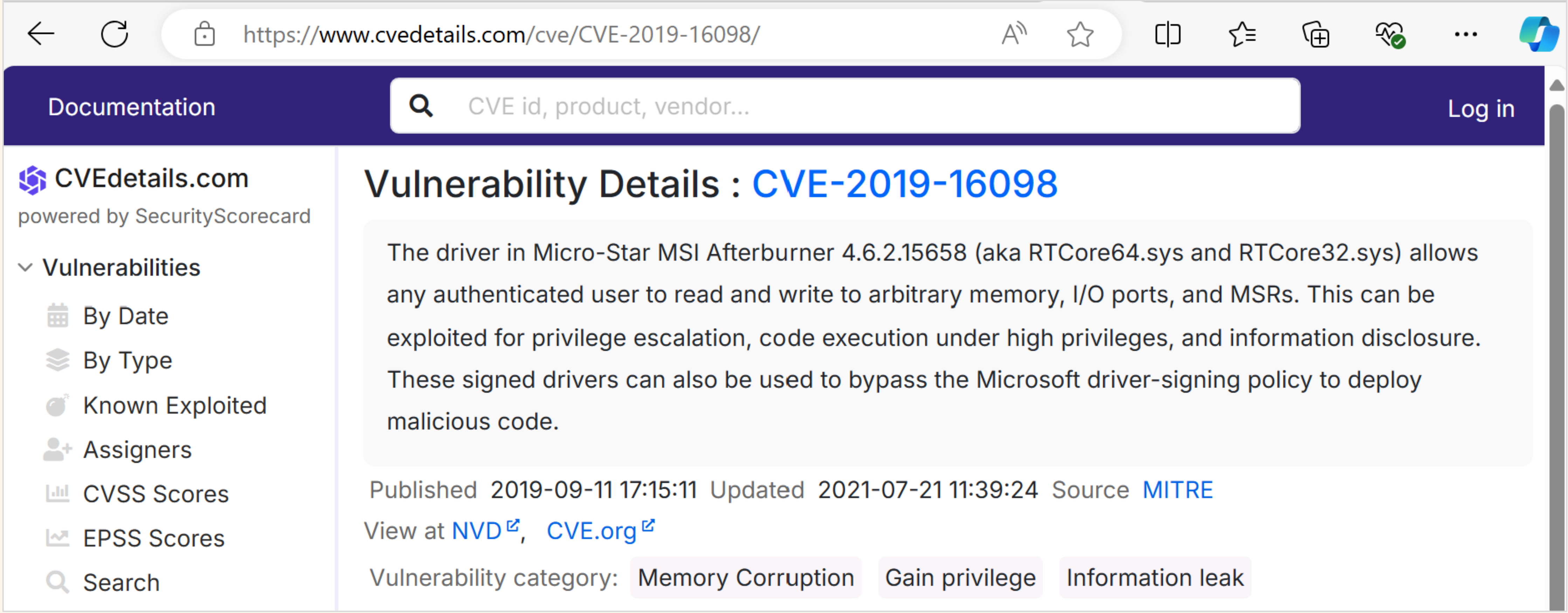
Real-world example



Real-world example



CVE-2019-16098



Vulnerability Details : CVE-2019-16098

The driver in Micro-Star MSI Afterburner 4.6.2.15658 (aka RTCore64.sys and RTCore32.sys) allows any authenticated user to read and write to arbitrary memory, I/O ports, and MSRs. This can be exploited for privilege escalation, code execution under high privileges, and information disclosure. These signed drivers can also be used to bypass the Microsoft driver-signing policy to deploy malicious code.

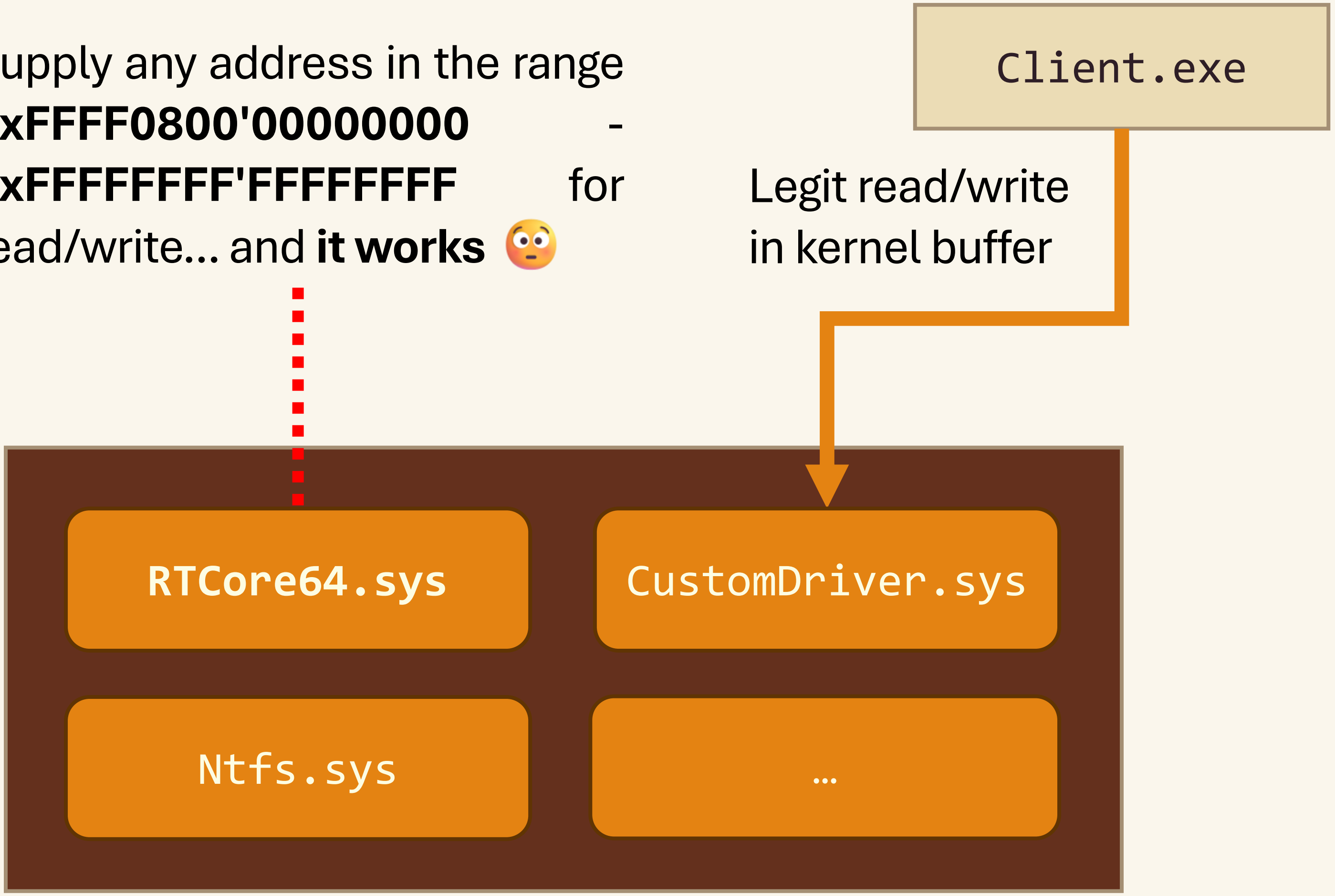
Published 2019-09-11 17:15:11 Updated 2021-07-21 11:39:24 Source MITRE

View at NVD, CVE.org

Vulnerability category: Memory Corruption Gain privilege Information leak

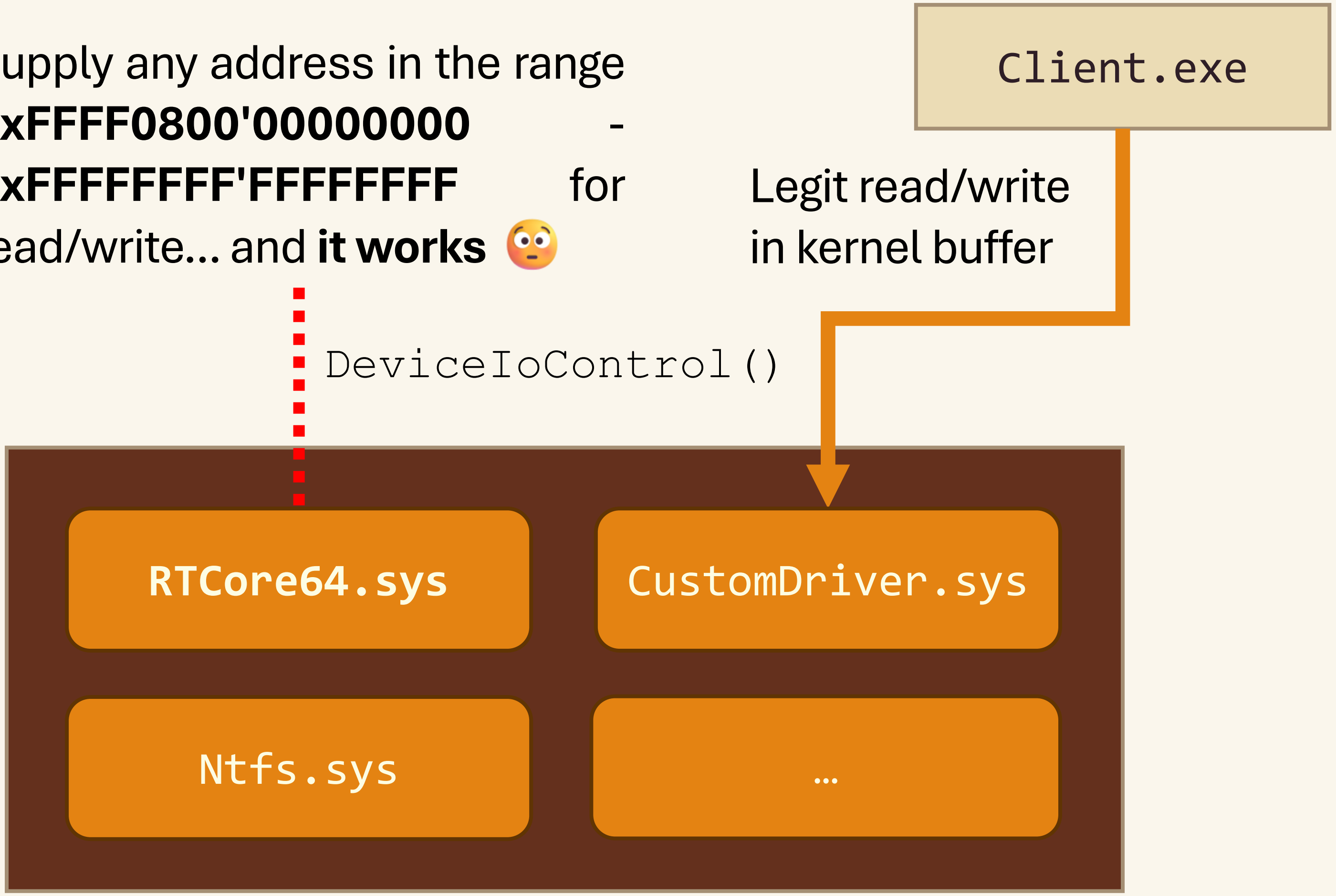
Real-world example

Supply any address in the range **0xFFFF0800'00000000 - 0xFFFFFFFF'FFFFFFFF** for read/write... and **it works** 😳



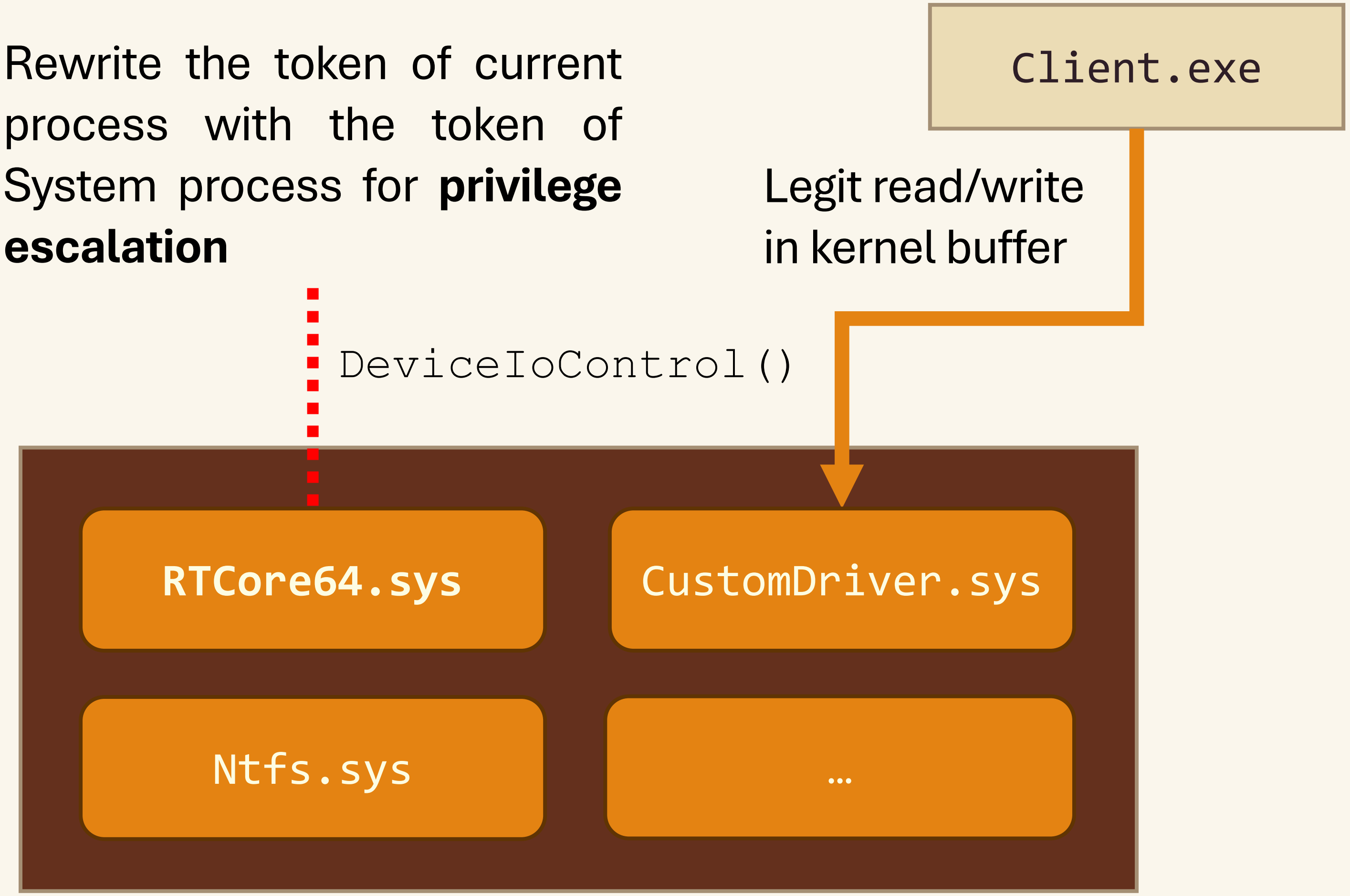
Real-world example

Supply any address in the range **0xFFFF0800'00000000 - 0xFFFFFFFF'FFFFFFFF** for read/write… and **it works** 😳



First idea

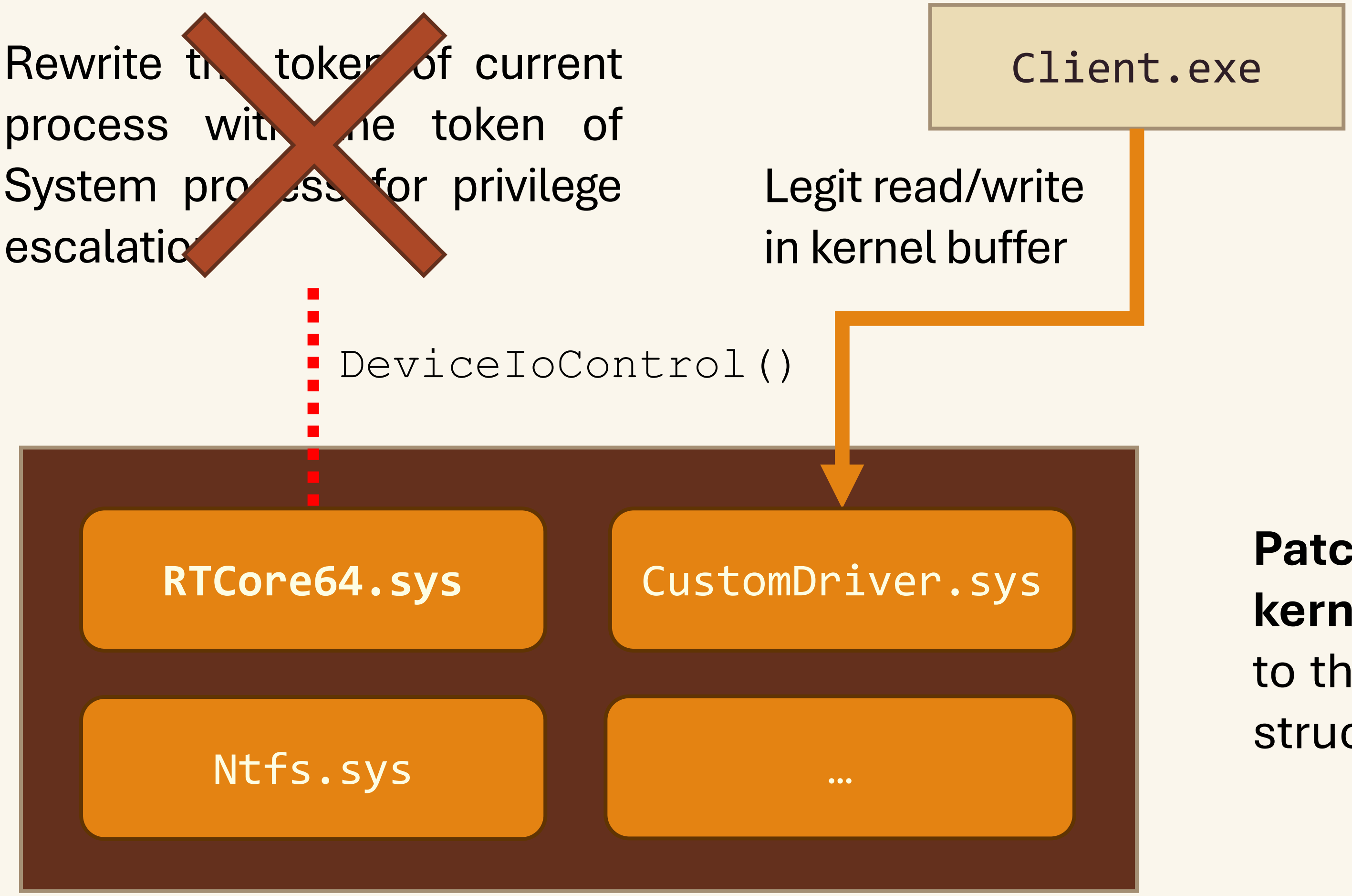
Rewrite the token of current process with the token of System process for **privilege escalation**



It is possible to get the pointer to System process out of user space, more information here:

[Exploring the Windows kernel using vulnerable driver - Part 2 - Ring 0x00 (idafchev.github.io)]()

PatchGuard



~~Rewrite the token of current process with the token of System process for privilege escalation~~

Client.exe

Legit read/write in kernel buffer

`DeviceIoControl()`

RTCore64.sys

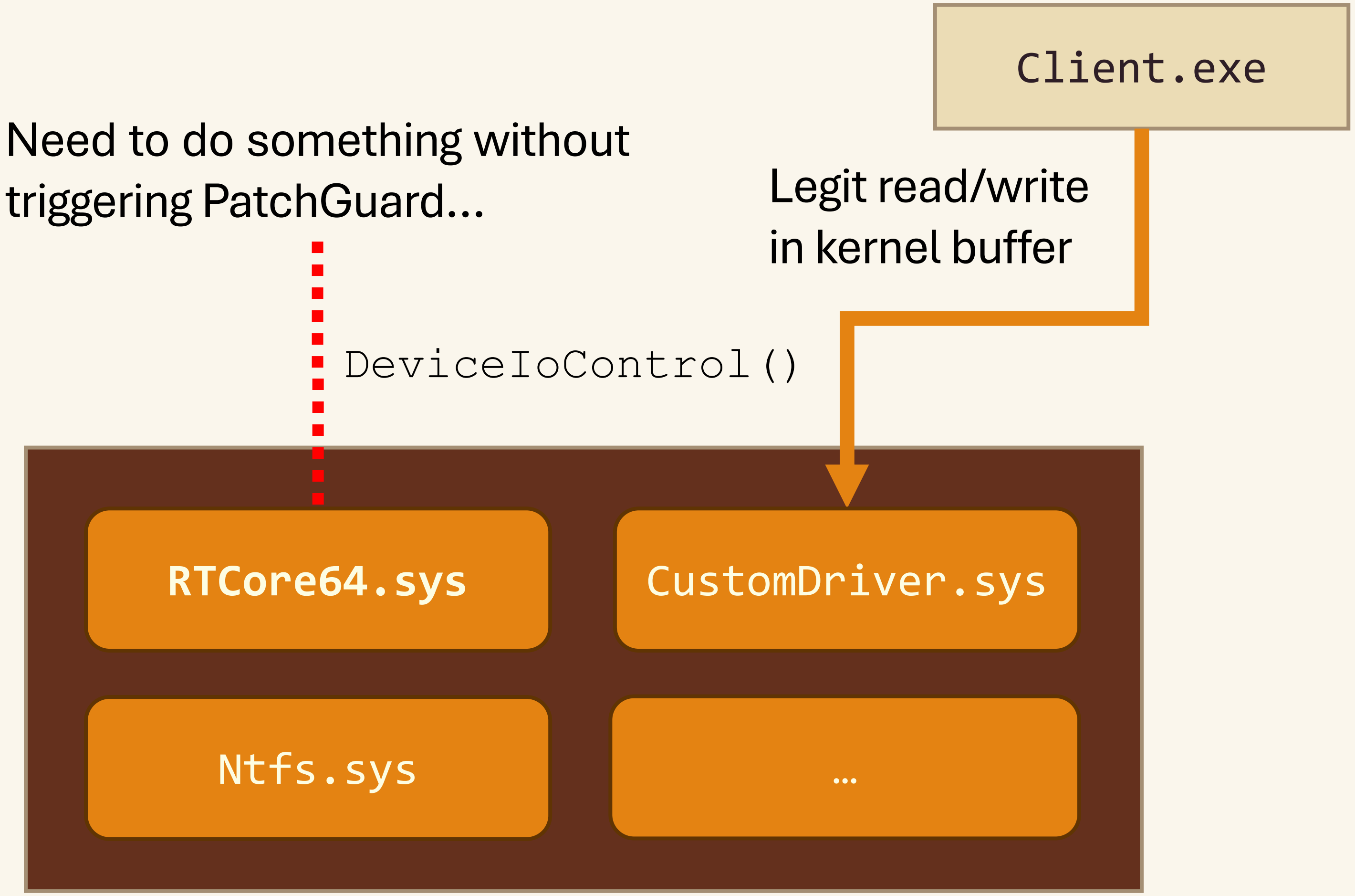
CustomDriver.sys

Ntfs.sys

…

PatchGuard prevents **unauthorized kernel modifications**, including changes to the SSDT, IDT, GDT, and process token structures, in 64-bit Windows systems.

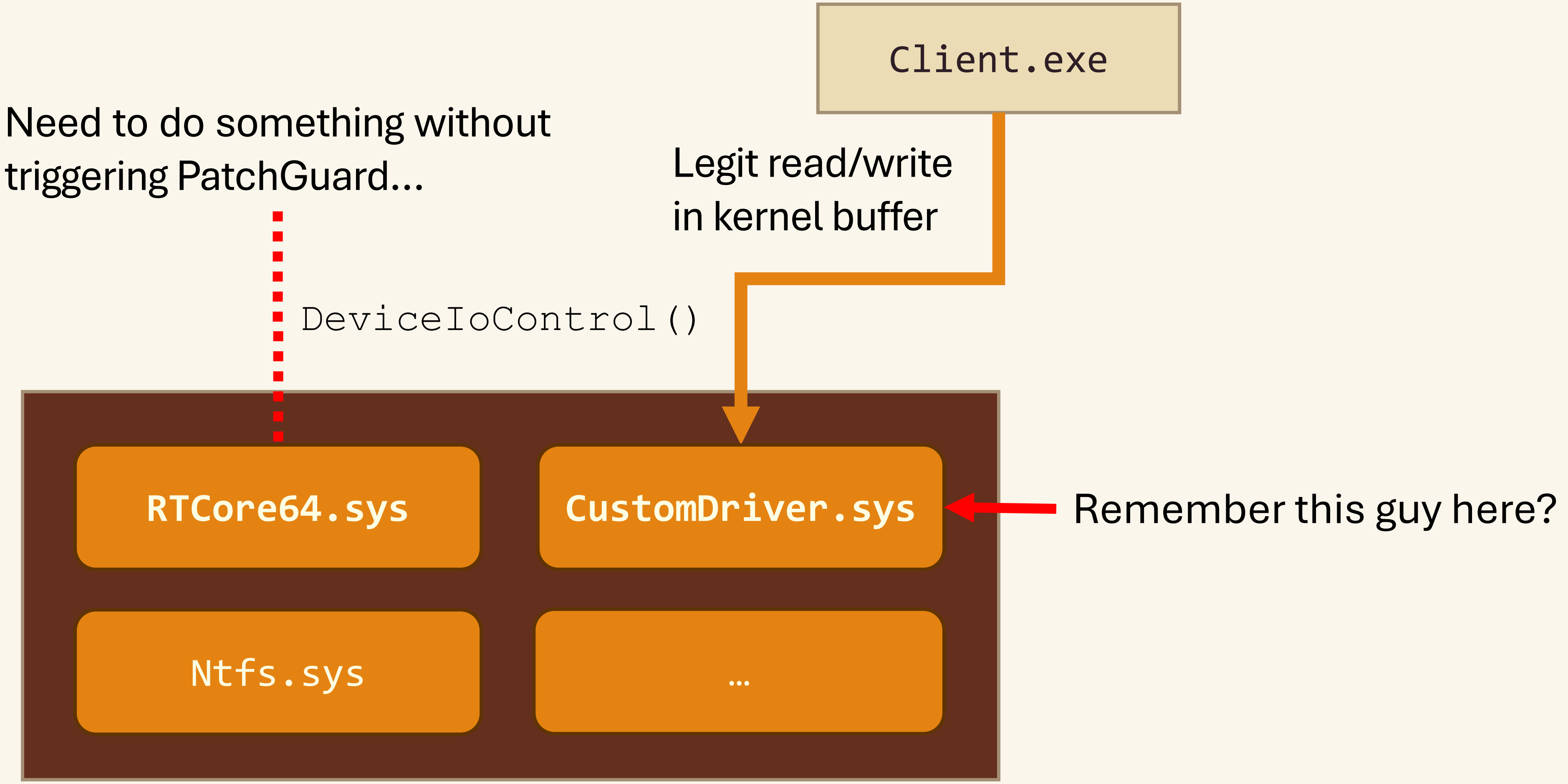
Hmm…



Need to do something without triggering PatchGuard…

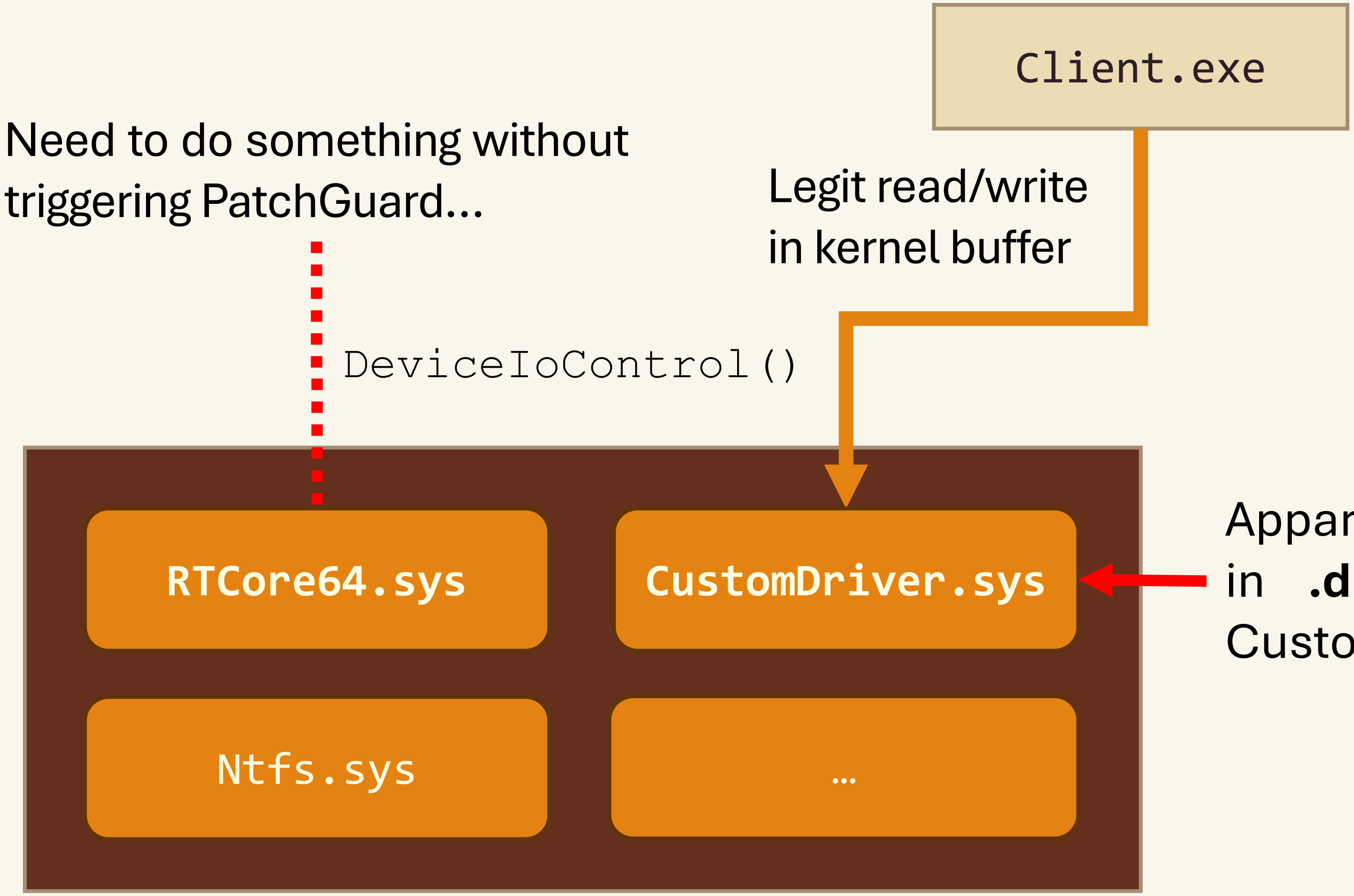
Legit read/write in kernel buffer

Hmm...???


Client.exe
Need to do something without triggering PatchGuard...
Legit read/write in kernel buffer
DeviceIoControl()
RTCore64.sys
CustomDriver.sys
Remember this guy here?
Ntfs.sys
...

Hmm…???

Need to do something without triggering PatchGuard…

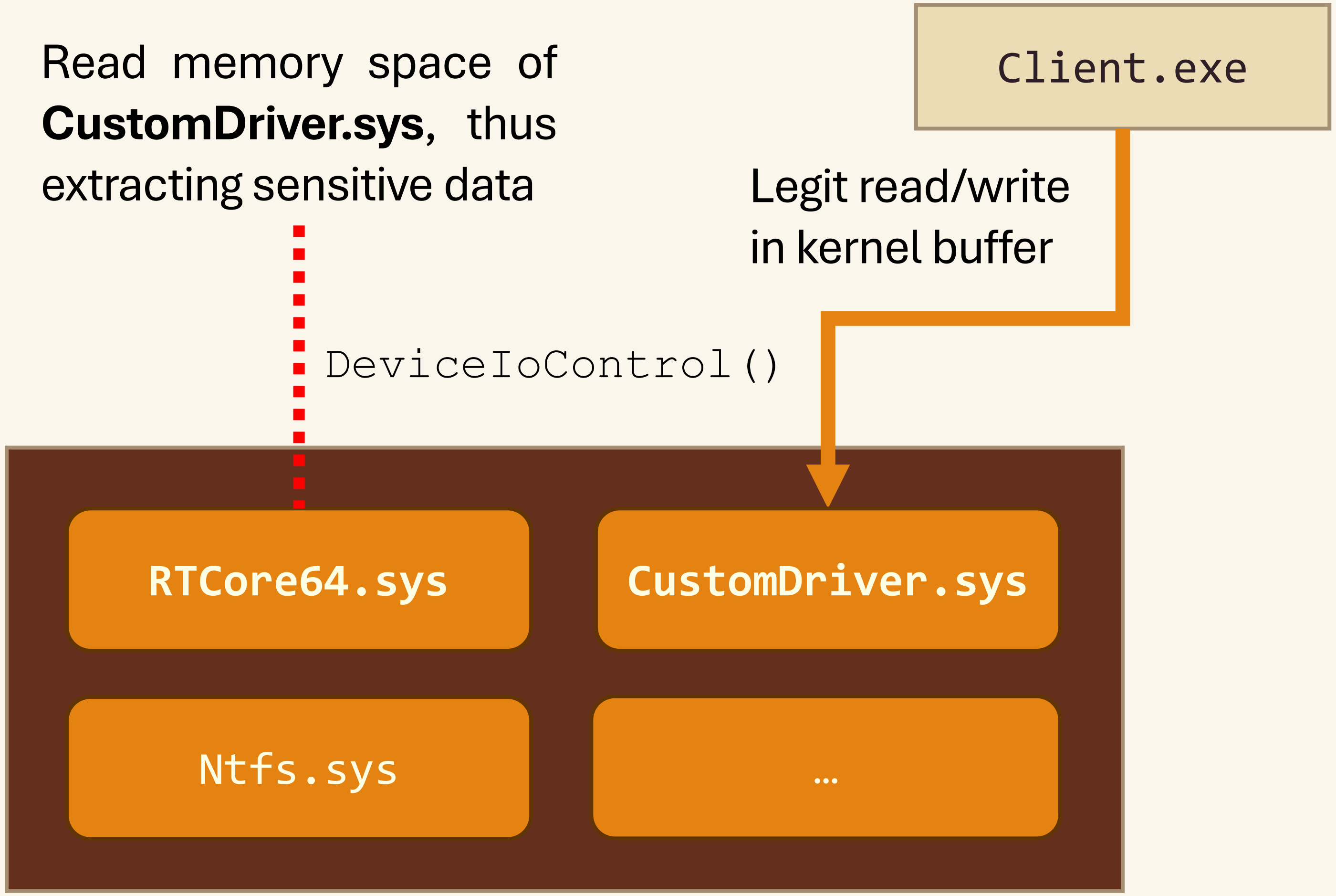


Apparently data was stored in **.data** section of the CustomeDriver.sys module

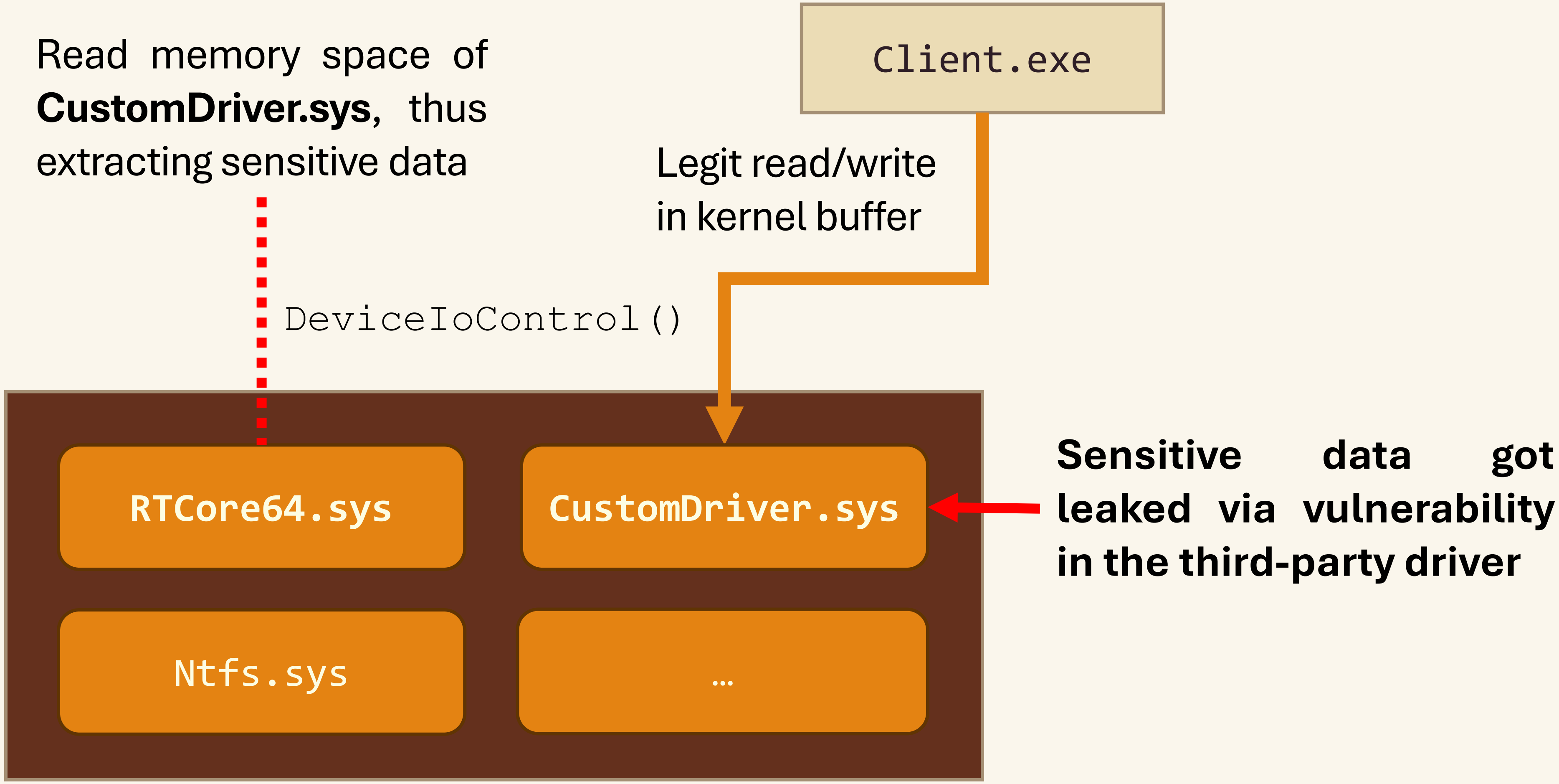
PatchGuard

Read memory space of **CustomDriver.sys**, thus extracting sensitive data

Legit read/write in kernel buffer



PatchGuard



Read memory space of **CustomDriver.sys**, thus extracting sensitive data

DeviceIoControl()

Client.exe

Legit read/write in kernel buffer

RTCore64.sys

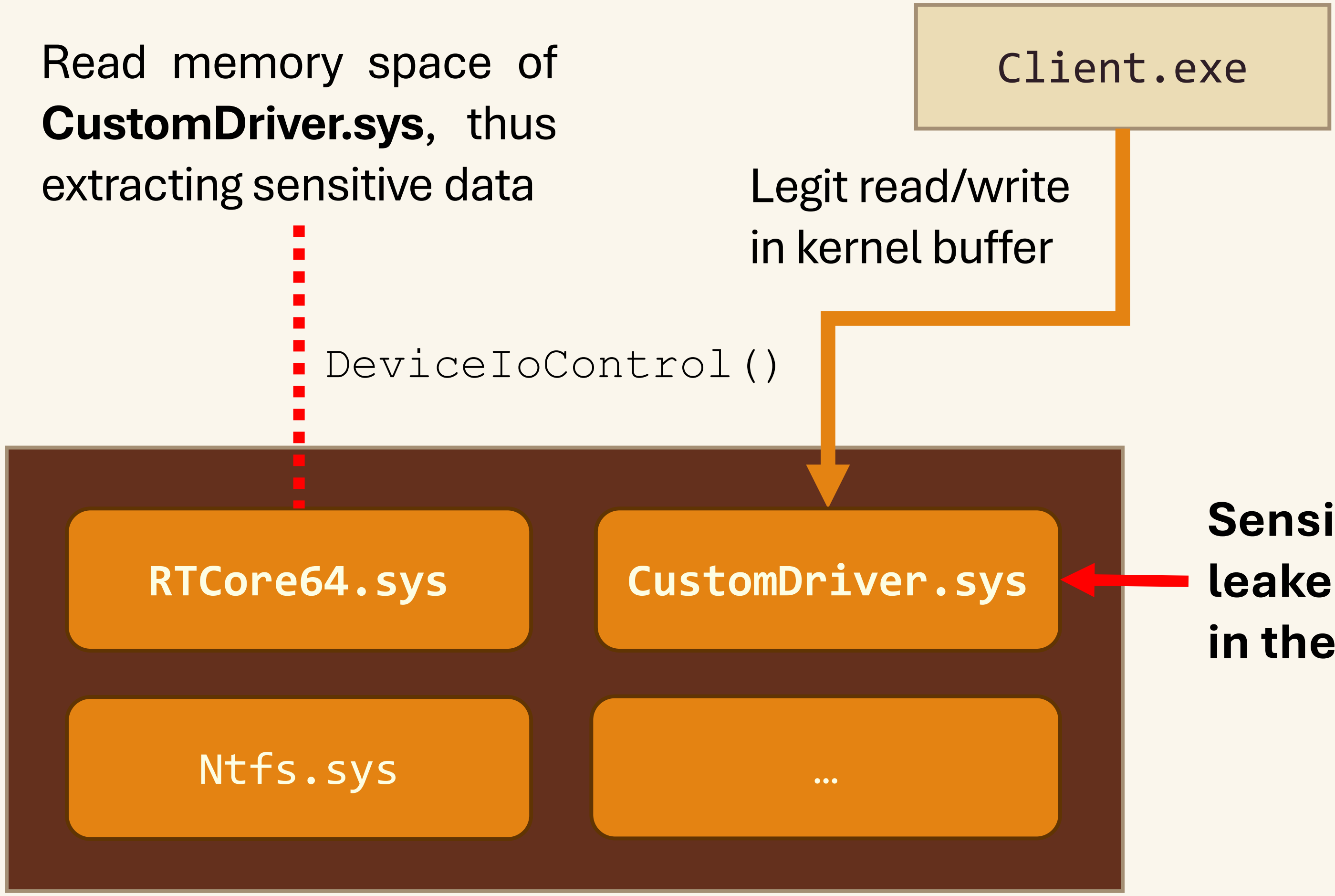
CustomDriver.sys

Ntfs.sys

…

Sensitive data got leaked via vulnerability in the third-party driver

PatchGuard


Read memory space of **CustomDriver.sys**, thus extracting sensitive data
DeviceIoControl()
Client.exe
Legit read/write in kernel buffer
RTCore64.sys
CustomDriver.sys
Ntfs.sys
…

Full code of the exploit is available here: [ExploitRTCore64]

Sensitive data got leaked via vulnerability in the third-party driver

Final demo

Developing simplified exploit on reading System process’s token

Summary

- In this session, we covered the fundamentals of kernel debugging with WinDbg, explored the Windows process and memory model, and dived into real-world kernel exploitation scenarios.
- By understanding the internal workings of the kernel and utilizing tools like WinDbg, we can effectively identify and explore potential security vulnerabilities.
- Remember, responsible handling of kernel-level access and knowledge of protections like PatchGuard are crucial in maintaining system integrity.