Understanding (y)our evasions

Building an effective evasion methodology

SECFORCE

BEACON

Hodl

Who We Are

Who We Are

Borch

Who We Are

Borch
ElephantSe4l
Glenx

What Is This Talk About?

Learning what questions we need to ask ourselves to start bypassing security products

What This Talk is not About?

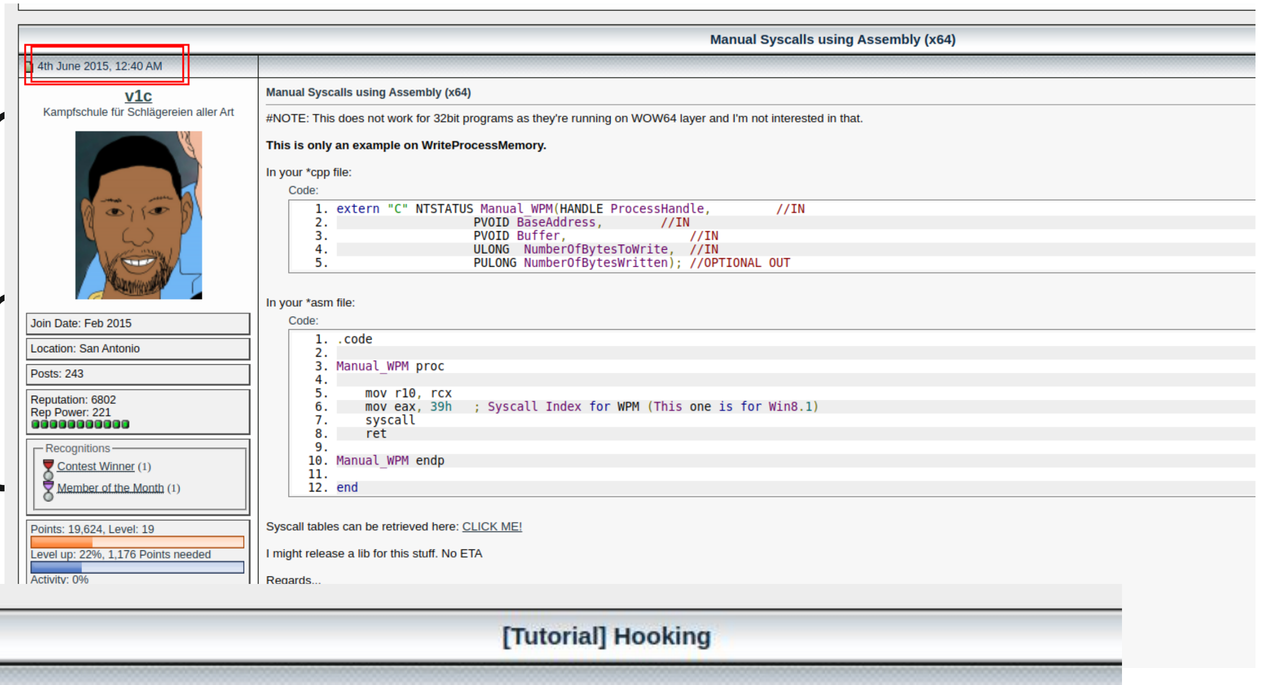
Being hacker stealthier 100% FUD 4K Megaupload

New secret technique

What This Talk is r

Being hacker stealth

New secret techniqu



Manual Syscalls using Assembly (x64)

4th June 2015, 12:40 AM

v1c
Kampfschule für Schlägereien aller Art

Join Date: Feb 2015
Location: San Antonio
Posts: 243
Reputation: 6802
Rep Power: 221

Recognitions
Contest Winner (1)
Member of the Month (1)

Points: 19,624, Level: 19
Level up: 22%, 1,176 Points needed
Activity: 0%

Manual Syscalls using Assembly (x64)

#NOTE: This does not work for 32bit programs as they're running on WOW64 layer and I'm not interested in that.

This is only an example on WriteProcessMemory.

In your *cpp file:

Code:
```
extern "C" NTSTATUS Manual_WPM(HANDLE ProcessHandle,         //IN
                   PVOID BaseAddress,        //IN
                   PVOID Buffer,                //IN
                   ULONG  NumberOfBytesToWrite,  //IN
                   PULONG NumberOfBytesWritten); //OPTIONAL OUT
```

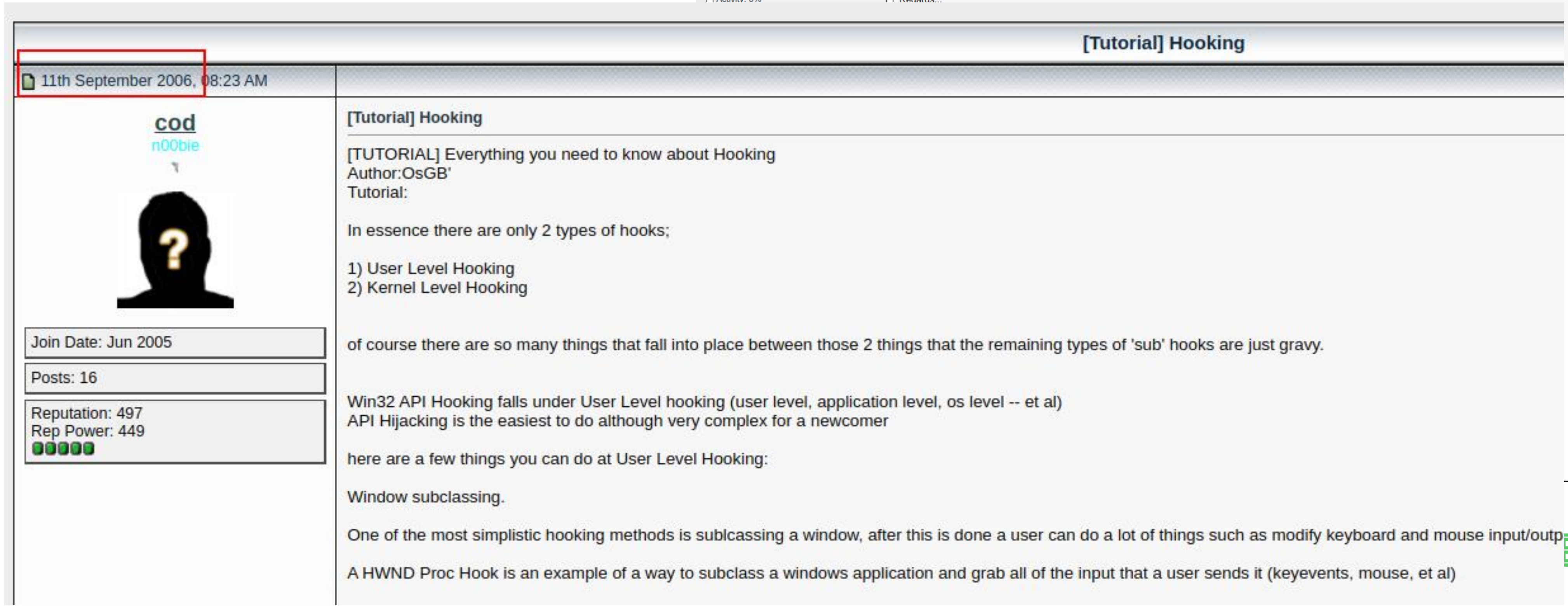
In your *asm file:

Code:
```
.code

Manual_WPM proc

    mov r10, rcx
    mov eax, 39h    ; Syscall Index for WPM (This one is for Win8.1)
    syscall
    ret

Manual_WPM endp

end
```

Syscall tables can be retrieved here: CLICK ME!

I might release a lib for this stuff. No ETA

Regards...



[Tutorial] Hooking

11th September 2006, 08:23 AM

cod
n00bie

Join Date: Jun 2005
Posts: 16
Reputation: 497
Rep Power: 449

[Tutorial] Hooking

[TUTORIAL] Everything you need to know about Hooking
Author:OsGB'
Tutorial:

In essence there are only 2 types of hooks;

1) User Level Hooking
2) Kernel Level Hooking

of course there are so many things that fall into place between those 2 things that the remaining types of 'sub' hooks are just gravy.

Win32 API Hooking falls under User Level hooking (user level, application level, os level -- et al)
API Hijacking is the easiest to do although very complex for a newcomer

here are a few things you can do at User Level Hooking:

Window subclassing.

One of the most simplistic hooking methods is sublcassing a window, after this is done a user can do a lot of things such as modify keyboard and mouse input/outp

A HWND Proc Hook is an example of a way to subclass a windows application and grab all of the input that a user sends it (keyevents, mouse, et al)

BEACON

Motivation

Where do I start...?

Motivation

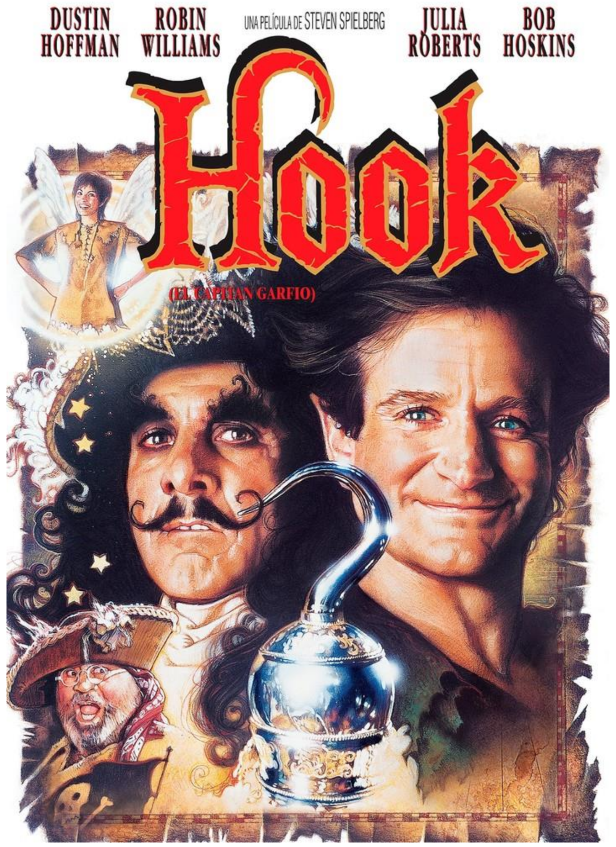
Where do I start...?

Understand the logic after the detection mechanisms and their bypasses so we can build an effective methodology

Motivation II

- Metadata
- Obfuscation
- Encription
- Entropy
- IAT magic
- PPID Spoofing
- Thread/Stack spoofing
- API Hooking
- Syscalling

API Hooking



(Software)Instrumentation

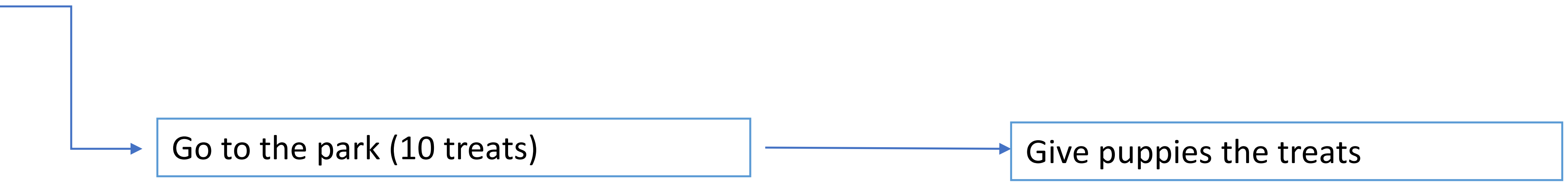
Involves having the tools and capabilities to observe modify or interact during the lifecycle of a program

API Hooking

- We use FRIDA for instrumentation
- EDRs implement their own ways of instrumentation

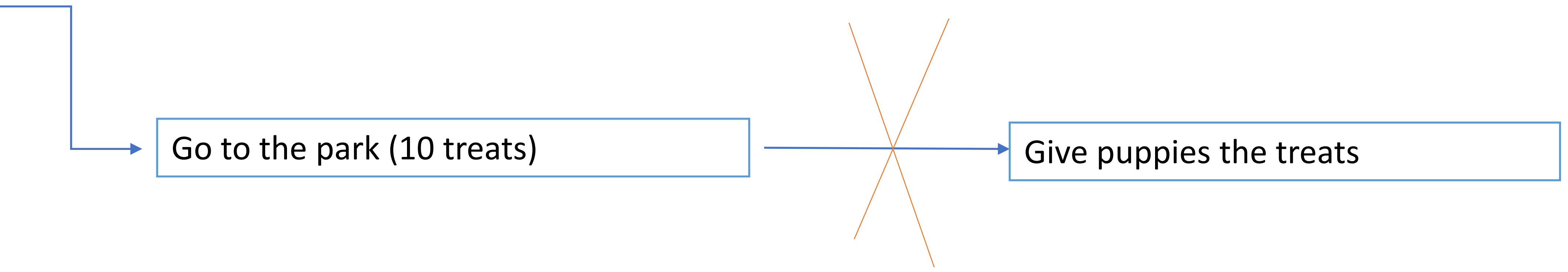
API Hooking

- We use FRIDA for instrumentation
- EDRs implement their own ways of instrumentation



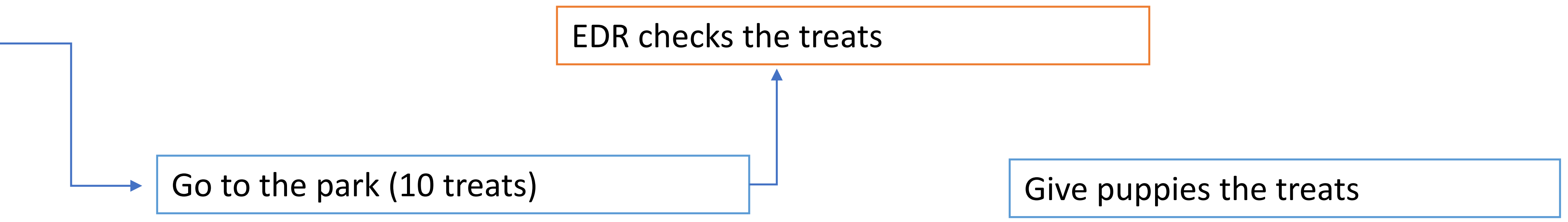
API Hooking

- We use FRIDA for instrumentation
- EDRs implement their own ways of instrumentation



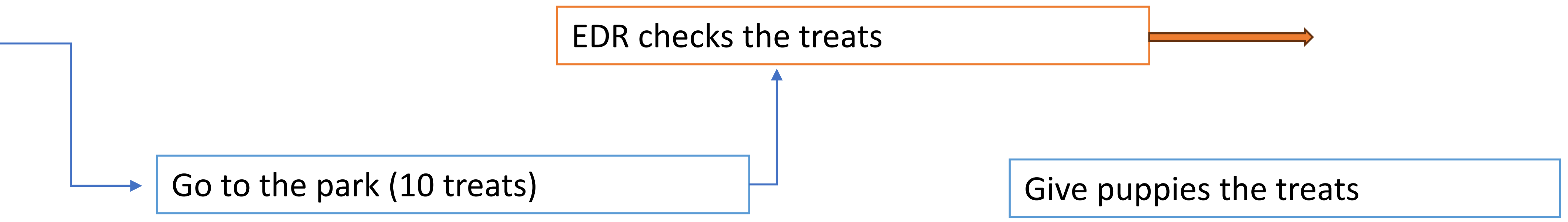
API Hooking

- We use FRIDA for instrumentation
- EDRs implement their own ways of instrumentation



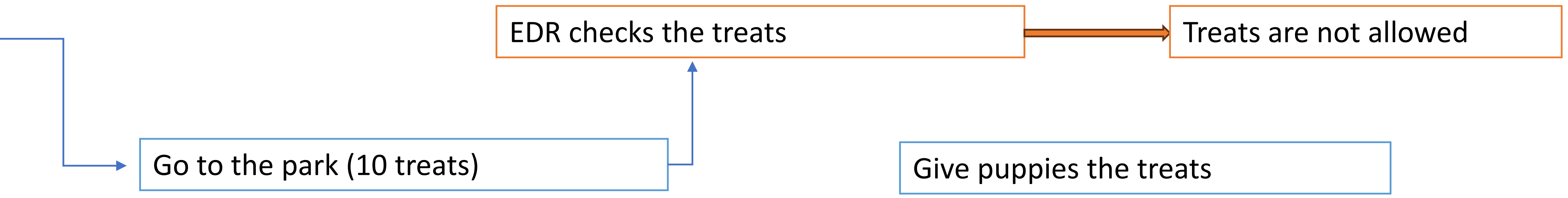
API Hooking

- We use FRIDA for instrumentation
- EDRs implement their own ways of instrumentation



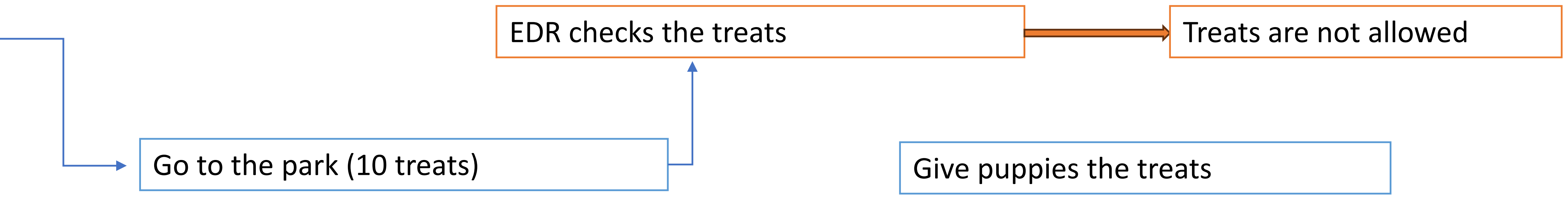
API Hooking

- We use FRIDA for instrumentation
- EDRs implement their own ways of instrumentation



API Hooking

- We use FRIDA for instrumentation
- EDRs implement their own ways of instrumentation



Puppies get 0 treats ☹

API Hooking

- Security products need to be able to «understand» what is going on on a running process
- Where shall the hooks be placed in?

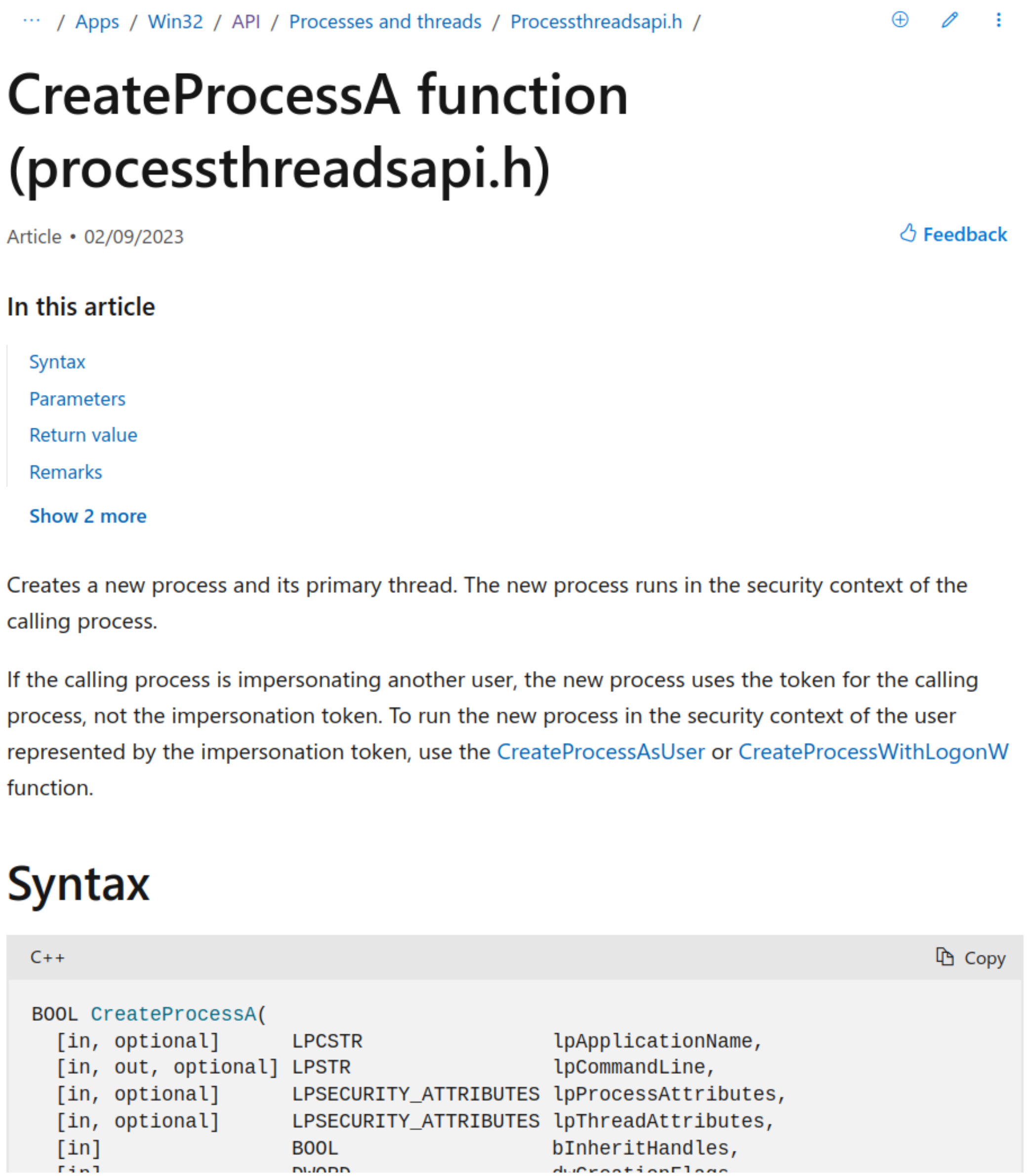
API Hooking: WINAPI

- API like any other
- Created for reusability and to be used by programmers
- Well documented

Create Handle
Create Process
Allocate Memory

API Hooking: WINAPI

- API like any other
- Created for reusability and to be used by pı
- Well documented



… / Apps / Win32 / API / Processes and threads / Processthreadsapi.h /

CreateProcessA function (processthreadsapi.h)

Article • 02/09/2023

Feedback

In this article

Syntax
Parameters
Return value
Remarks
Show 2 more

Creates a new process and its primary thread. The new process runs in the security context of the calling process.

If the calling process is impersonating another user, the new process uses the token for the calling process, not the impersonation token. To run the new process in the security context of the user represented by the impersonation token, use the CreateProcessAsUser or CreateProcessWithLogonW function.

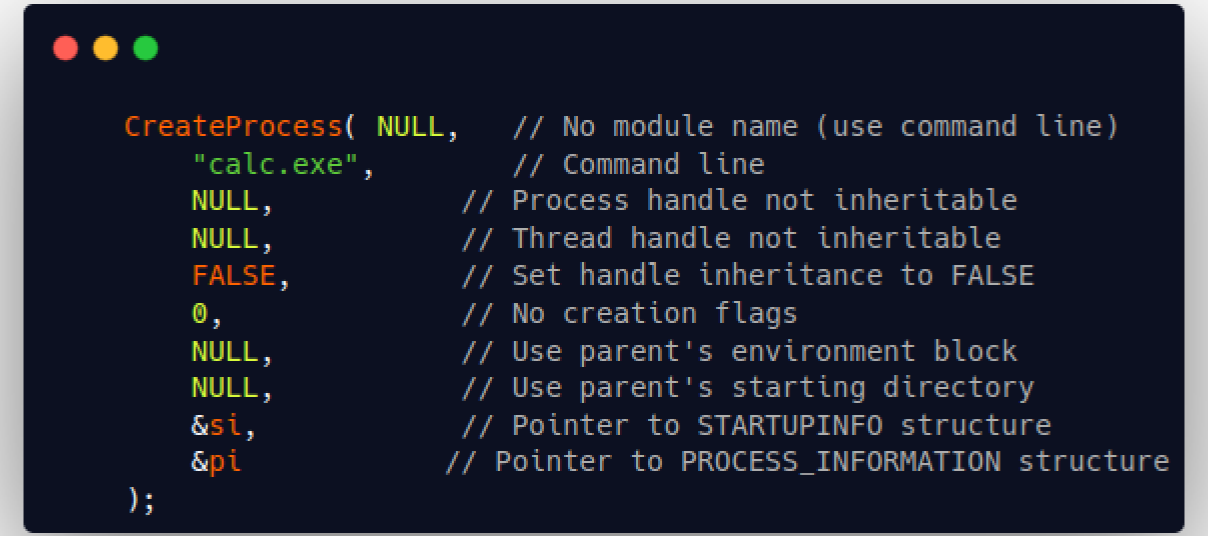
Syntax

C++ Copy

```
BOOL CreateProcessA(
  [in, optional]      LPCSTR                lpApplicationName,
  [in, out, optional] LPSTR                 lpCommandLine,
  [in, optional]      LPSECURITY_ATTRIBUTES lpProcessAttributes,
  [in, optional]      LPSECURITY_ATTRIBUTES lpThreadAttributes,
  [in]                BOOL                  bInheritHandles,
```

API Hooking: WINAPI

- API like any other
- Created for reusability and to be used by programmers
- Well documented



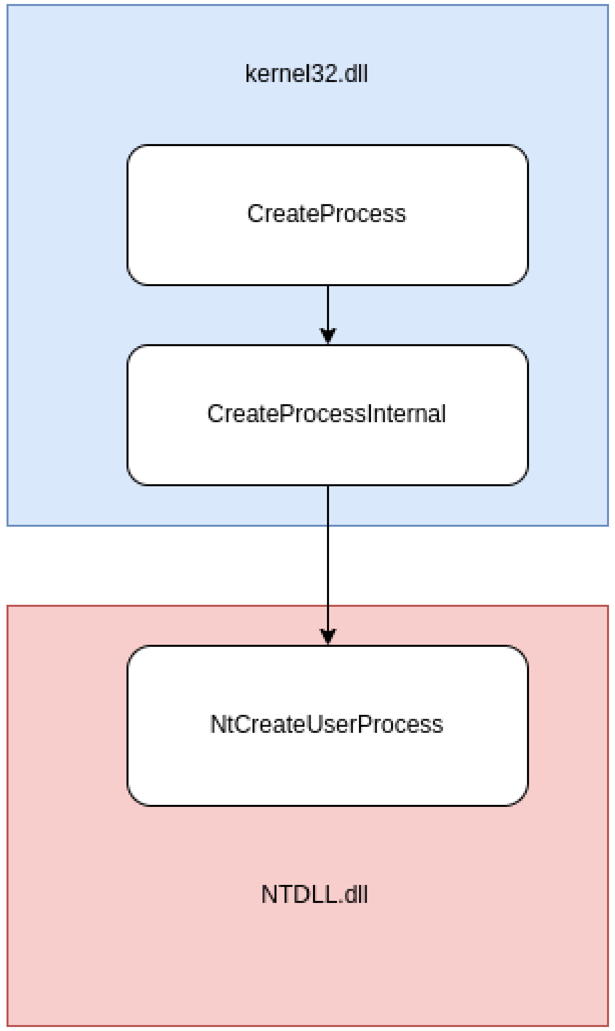
```
CreateProcess( NULL,   // No module name (use command line)
    "calc.exe",        // Command line
    NULL,           // Process handle not inheritable
    NULL,           // Thread handle not inheritable
    FALSE,          // Set handle inheritance to FALSE
    0,              // No creation flags
    NULL,           // Use parent's environment block
    NULL,           // Use parent's starting directory
    &si,            // Pointer to STARTUPINFO structure
    &pi            // Pointer to PROCESS_INFORMATION structure
);
```

API Hooking: WINAPI

Let's hook every
function on WinAPI

API Hooking: WINAPI

It is not that easy, CreateProcess is a **wrapper**

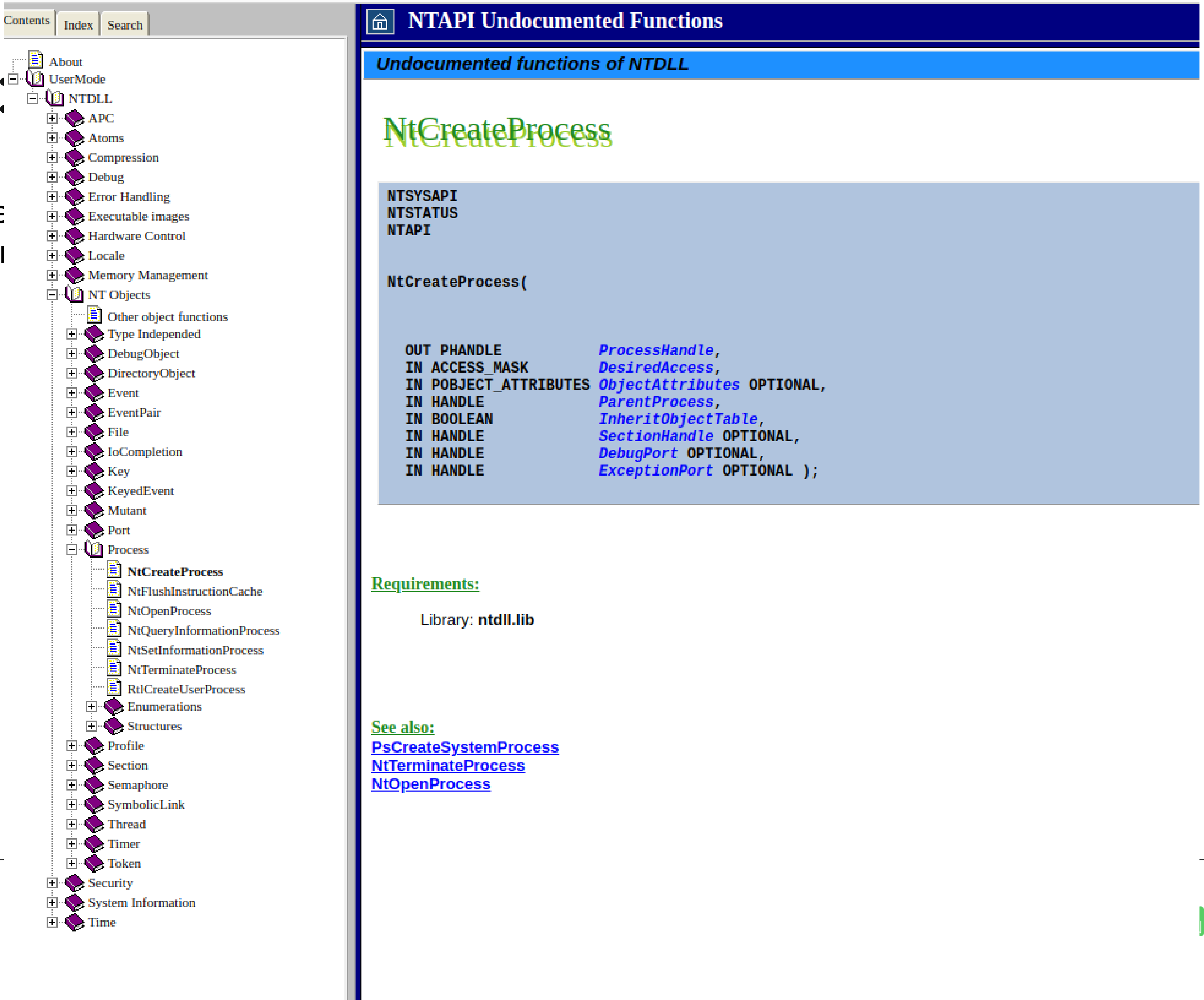


API Hooking: NTDLL.dll

- Was not created to be used by high-level programmers
- Is the last library before calling the SYSCALL
- Is not documented

API Hooking:

- Was not created to be
- Is the last library befo
- Is not documented



NTAPI Undocumented Functions

Undocumented functions of NTDLL

NtCreateProcess

```
NTSYSAPI
NTSTATUS
NTAPI


NtCreateProcess(



  OUT PHANDLE           ProcessHandle,
  IN ACCESS_MASK        DesiredAccess,
  IN POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
  IN HANDLE             ParentProcess,
  IN BOOLEAN            InheritObjectTable,
  IN HANDLE             SectionHandle OPTIONAL,
  IN HANDLE             DebugPort OPTIONAL,
  IN HANDLE             ExceptionPort OPTIONAL );
```

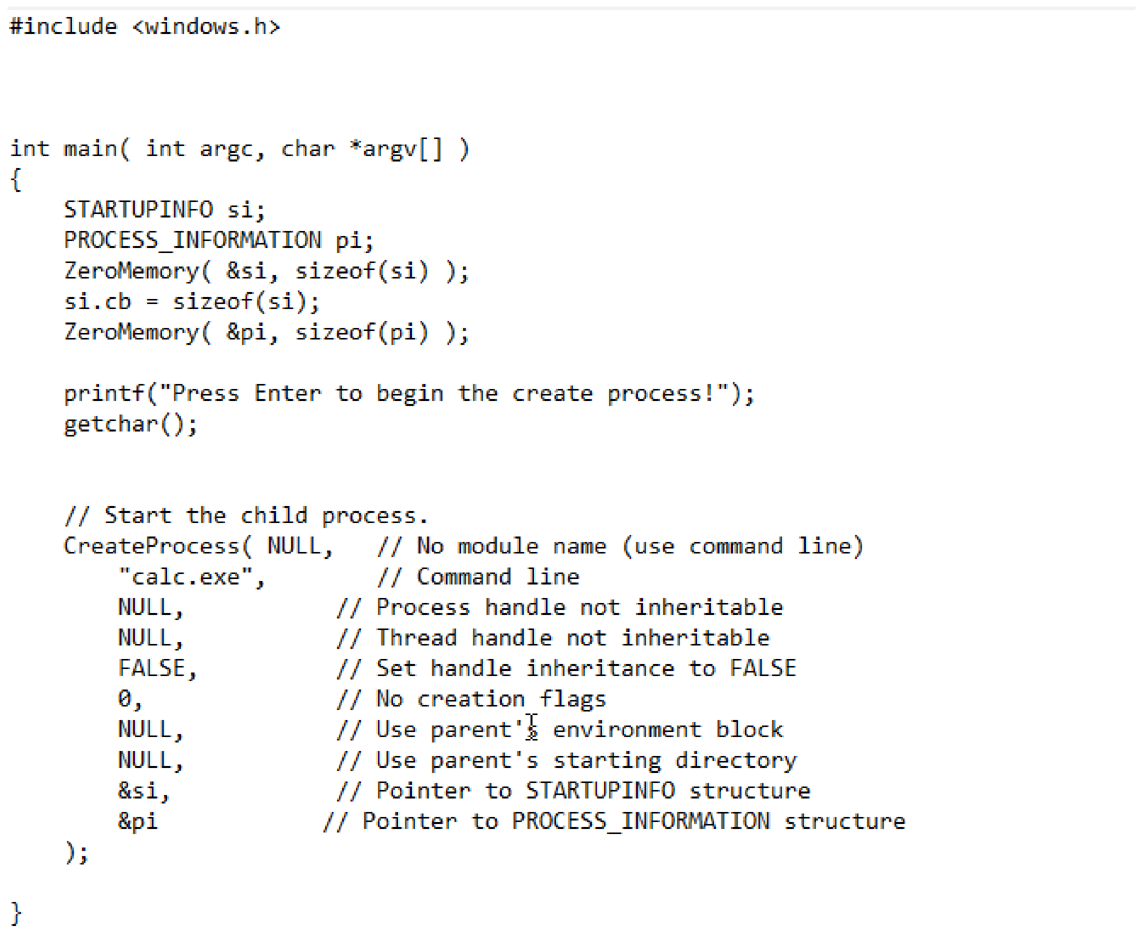
Requirements:

Library: **ntdll.lib**

See also:
PsCreateSystemProcess
NtTerminateProcess
NtOpenProcess

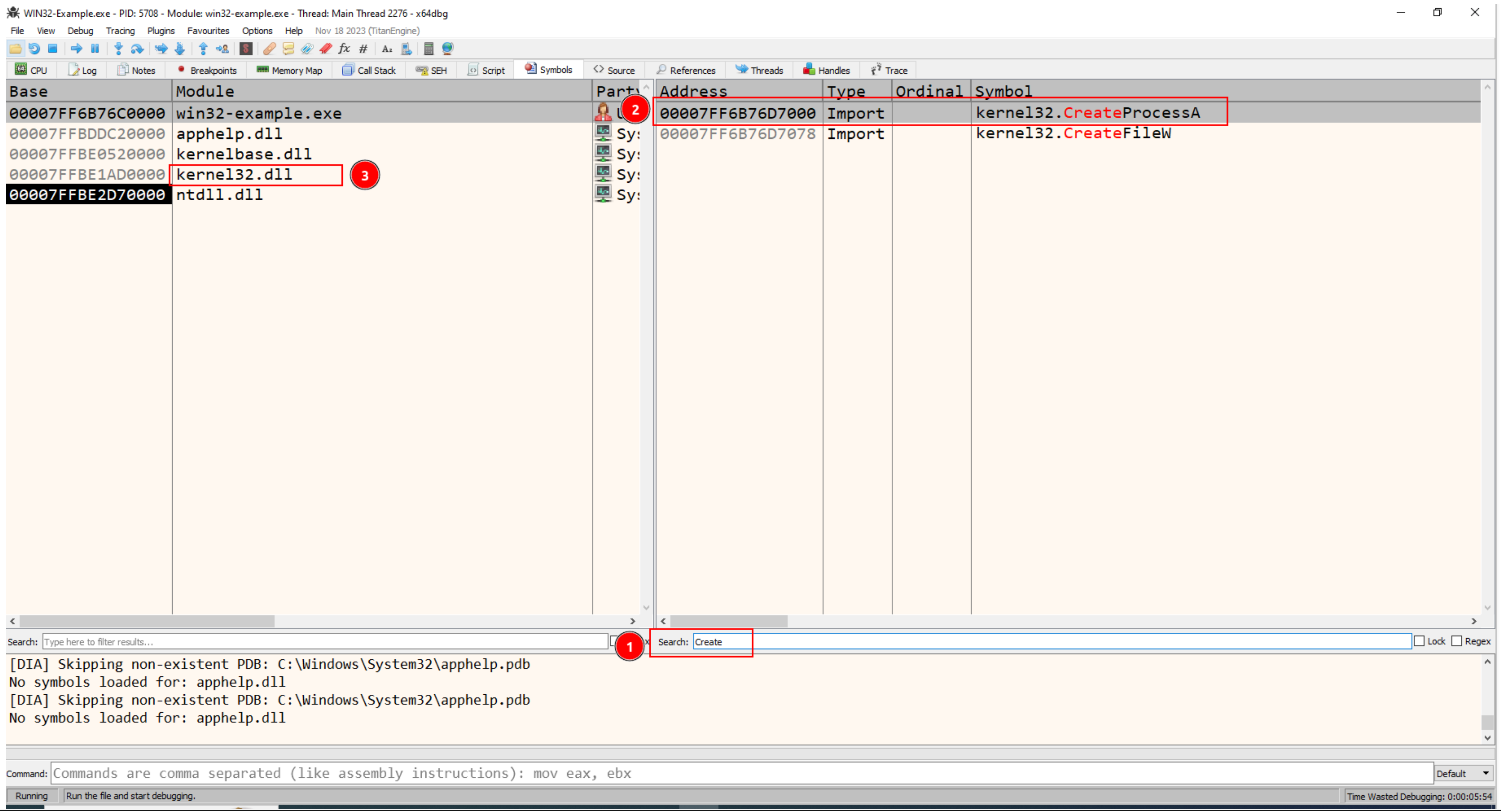
SECFORCE

API Hooking: DEMO – From Win32 API to NTDLL



```
#include <windows.h>



int main( int argc, char *argv[] )
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );

    printf("Press Enter to begin the create process!");
    getchar();


    // Start the child process.
    CreateProcess( NULL,   // No module name (use command line)
        "calc.exe",        // Command line
        NULL,           // Process handle not inheritable
        NULL,           // Thread handle not inheritable
        FALSE,          // Set handle inheritance to FALSE
        0,              // No creation flags
        NULL,           // Use parent's environment block
        NULL,           // Use parent's starting directory
        &si,            // Pointer to STARTUPINFO structure
        &pi            // Pointer to PROCESS_INFORMATION structure
    );

}
```

API Hooking: DEMO – From Win32 API to NTDLL


WIN32-Example.exe - PID: 5708 - Module: win32-example.exe - Thread: Main Thread 2276 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Nov 18 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
1
2
3
Lock
Regex
Default
Running
Time Wasted Debugging: 0:00:05:54

Base	Module	Party
00007FF6B76C0000	win32-example.exe	U
00007FFBDDC20000	apphelp.dll	Sy:
00007FFBE0520000	kernelbase.dll	Sy:
00007FFBE1AD0000	kernel32.dll	Sy:
00007FFBE2D70000	ntdll.dll	Sy:

Address	Type	Ordinal	Symbol
00007FF6B76D7000	Import		kernel32.CreateProcessA
00007FF6B76D7078	Import		kernel32.CreateFileW

Search: Type here to filter results...

Search: Create

```
[DIA] Skipping non-existent PDB: C:\Windows\System32\apphelp.pdb
No symbols loaded for: apphelp.dll
[DIA] Skipping non-existent PDB: C:\Windows\System32\apphelp.pdb
No symbols loaded for: apphelp.dll
```

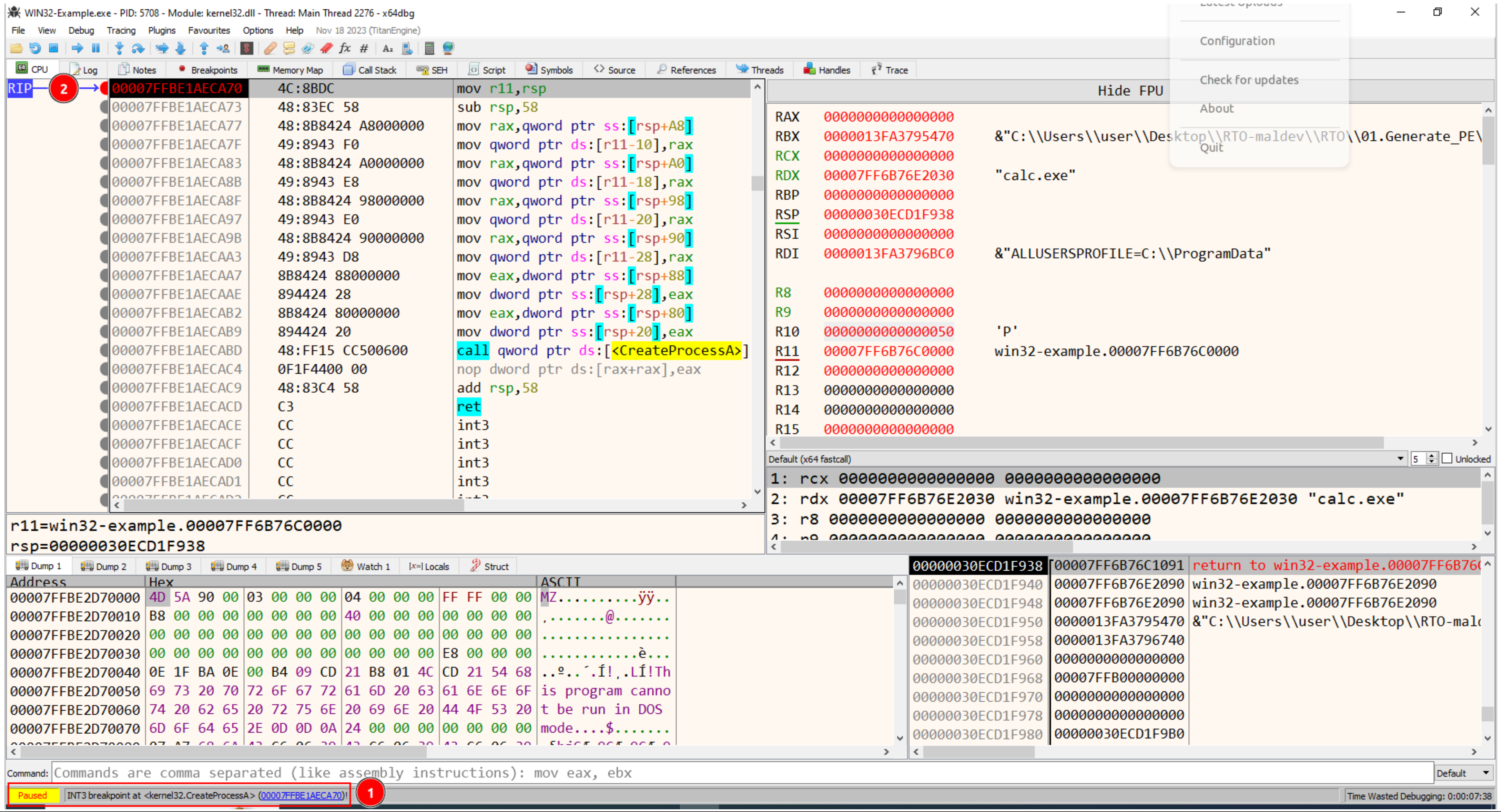
Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Run the file and start debugging.

SECFORCE

BEACON

API Hooking: DEMO – From Win32 API to NTDLL


WIN32-Example.exe - PID: 5708 - Module: kernel32.dll - Thread: Main Thread 2276 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Nov 18 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
Configuration
Check for updates
About
Quit
Hide FPU
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct
Default (x64 fastcall)
Unlocked
Default
Time Wasted Debugging: 0:00:07:38
1
2

```
RIP  00007FFBE1AECA70   4C:8BDC                mov r11,rsp
     00007FFBE1AECA73   48:83EC 58             sub rsp,58
     00007FFBE1AECA77   48:8B8424 A8000000     mov rax,qword ptr ss:[rsp+A8]
     00007FFBE1AECA7F   49:8943 F0             mov qword ptr ds:[r11-10],rax
     00007FFBE1AECA83   48:8B8424 A0000000     mov rax,qword ptr ss:[rsp+A0]
     00007FFBE1AECA8B   49:8943 E8             mov qword ptr ds:[r11-18],rax
     00007FFBE1AECA8F   48:8B8424 98000000     mov rax,qword ptr ss:[rsp+98]
     00007FFBE1AECA97   49:8943 E0             mov qword ptr ds:[r11-20],rax
     00007FFBE1AECA9B   48:8B8424 90000000     mov rax,qword ptr ss:[rsp+90]
     00007FFBE1AECAA3   49:8943 D8             mov qword ptr ds:[r11-28],rax
     00007FFBE1AECAA7   8B8424 88000000        mov eax,dword ptr ss:[rsp+88]
     00007FFBE1AECAAE   894424 28              mov dword ptr ss:[rsp+28],eax
     00007FFBE1AECAB2   8B8424 80000000        mov eax,dword ptr ss:[rsp+80]
     00007FFBE1AECAB9   894424 20              mov dword ptr ss:[rsp+20],eax
     00007FFBE1AECABD   48:FF15 CC500600       call qword ptr ds:[<CreateProcessA>]
     00007FFBE1AECAC4   0F1F4400 00            nop dword ptr ds:[rax+rax],eax
     00007FFBE1AECAC9   48:83C4 58             add rsp,58
     00007FFBE1AECACD   C3                     ret
     00007FFBE1AECACE   CC                     int3
     00007FFBE1AECACF   CC                     int3
     00007FFBE1AECAD0   CC                     int3
     00007FFBE1AECAD1   CC                     int3

r11=win32-example.00007FF6B76C0000
rsp=00000030ECD1F938
```

```
RAX  0000000000000000
RBX  0000013FA3795470    &"C:\\Users\\user\\Desktop\\RTO-maldev\\RTO\\01.Generate_PE\
RCX  0000000000000000
RDX  00007FF6B76E2030    "calc.exe"
RBP  0000000000000000
RSP  00000030ECD1F938
RSI  0000000000000000
RDI  0000013FA3796BC0    &"ALLUSERSPROFILE=C:\\ProgramData"

R8   0000000000000000
R9   0000000000000000
R10  0000000000000050    'P'
R11  00007FF6B76C0000    win32-example.00007FF6B76C0000
R12  0000000000000000
R13  0000000000000000
R14  0000000000000000
R15  0000000000000000
```

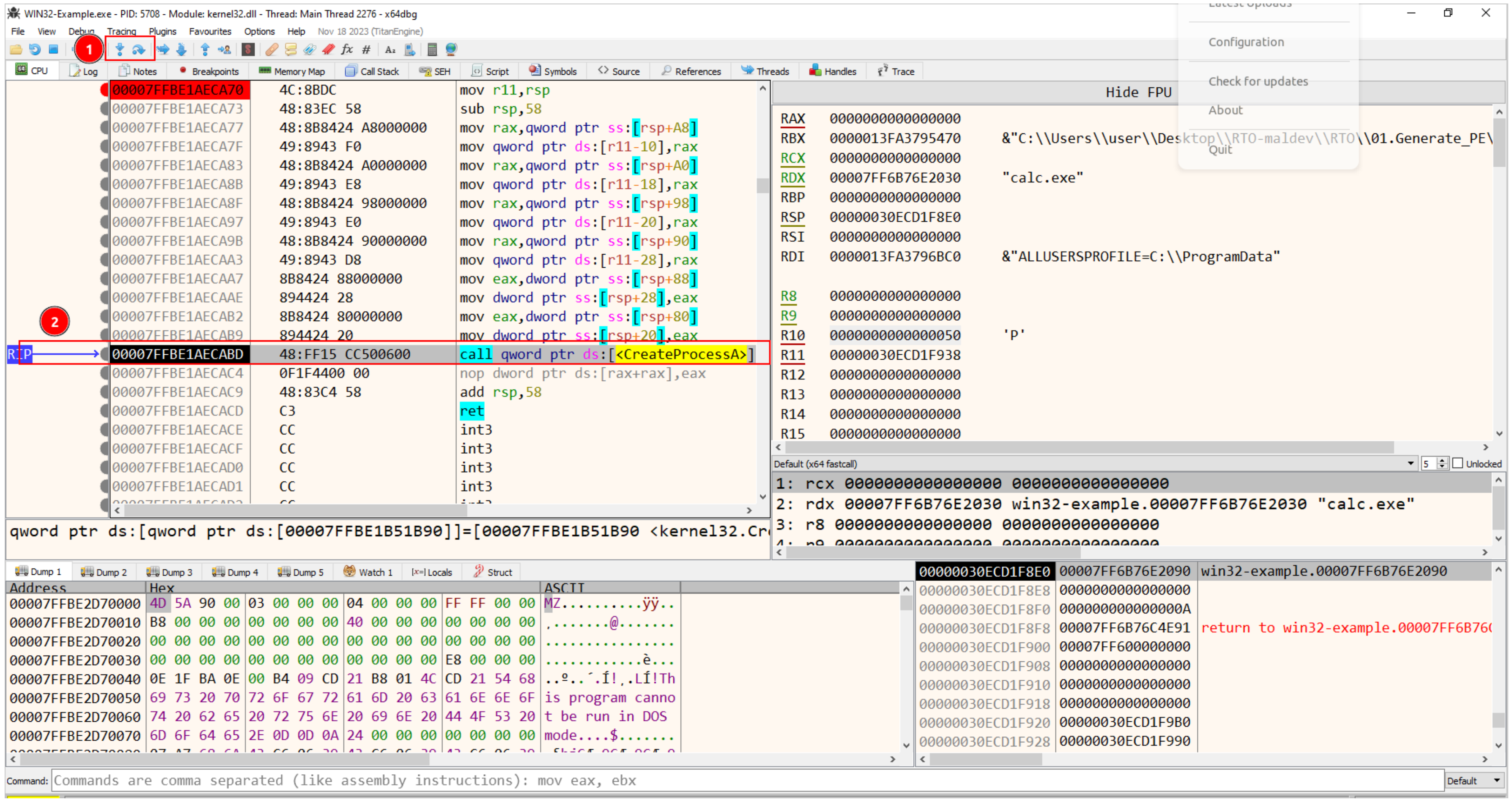
```
1: rcx 0000000000000000 0000000000000000
2: rdx 00007FF6B76E2030 win32-example.00007FF6B76E2030 "calc.exe"
3: r8 0000000000000000 0000000000000000
4: r9 0000000000000000 0000000000000000
```

```
Address           Hex                                              ASCII
00007FFBE2D70000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00007FFBE2D70010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ,.......@.......
00007FFBE2D70020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00007FFBE2D70030  00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00  ............è...
00007FFBE2D70040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00007FFBE2D70050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00007FFBE2D70060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00007FFBE2D70070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
```

```
00000030ECD1F938  00007FF6B76C1091  return to win32-example.00007FF6B76
00000030ECD1F940  00007FF6B76E2090  win32-example.00007FF6B76E2090
00000030ECD1F948  00007FF6B76E2090  win32-example.00007FF6B76E2090
00000030ECD1F950  0000013FA3795470  &"C:\\Users\\user\\Desktop\\RTO-mal
00000030ECD1F958  0000013FA3796740
00000030ECD1F960  0000000000000000
00000030ECD1F968  00007FFB00000000
00000030ECD1F970  0000000000000000
00000030ECD1F978  0000000000000000
00000030ECD1F980  00000030ECD1F9B0
```

```
Command: Commands are comma separated (like assembly instructions): mov eax, ebx
Paused  INT3 breakpoint at <kernel32.CreateProcessA> (00007FFBE1AECA70)!
```

API Hooking: DEMO – From Win32 API to NTDLL



WIN32-Example.exe - PID: 5708 - Module: kernel32.dll - Thread: Main Thread 2276 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Nov 18 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
Configuration
Check for updates
About
Quit
1
2
Hide FPU
Default (x64 fastcall)
5
Unlocked
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct
Default

```
00007FFBE1AECA70   4C:8BDC              mov r11,rsp
00007FFBE1AECA73   48:83EC 58           sub rsp,58
00007FFBE1AECA77   48:8B8424 A8000000   mov rax,qword ptr ss:[rsp+A8]
00007FFBE1AECA7F   49:8943 F0           mov qword ptr ds:[r11-10],rax
00007FFBE1AECA83   48:8B8424 A0000000   mov rax,qword ptr ss:[rsp+A0]
00007FFBE1AECA8B   49:8943 E8           mov qword ptr ds:[r11-18],rax
00007FFBE1AECA8F   48:8B8424 98000000   mov rax,qword ptr ss:[rsp+98]
00007FFBE1AECA97   49:8943 E0           mov qword ptr ds:[r11-20],rax
00007FFBE1AECA9B   48:8B8424 90000000   mov rax,qword ptr ss:[rsp+90]
00007FFBE1AECAA3   49:8943 D8           mov qword ptr ds:[r11-28],rax
00007FFBE1AECAA7   8B8424 88000000      mov eax,dword ptr ss:[rsp+88]
00007FFBE1AECAAE   894424 28            mov dword ptr ss:[rsp+28],eax
00007FFBE1AECAB2   8B8424 80000000      mov eax,dword ptr ss:[rsp+80]
00007FFBE1AECAB9   894424 20            mov dword ptr ss:[rsp+20],eax
RIP  00007FFBE1AECABD   48:FF15 CC500600     call qword ptr ds:[<CreateProcessA>]
00007FFBE1AECAC4   0F1F4400 00          nop dword ptr ds:[rax+rax],eax
00007FFBE1AECAC9   48:83C4 58           add rsp,58
00007FFBE1AECACD   C3                   ret
00007FFBE1AECACE   CC                   int3
00007FFBE1AECACF   CC                   int3
00007FFBE1AECAD0   CC                   int3
00007FFBE1AECAD1   CC                   int3

qword ptr ds:[qword ptr ds:[00007FFBE1B51B90]]=[00007FFBE1B51B90 <kernel32.Cr
```

```
RAX   0000000000000000
RBX   0000013FA3795470     &"C:\\Users\\user\\Desktop\\RTO-maldev\\RTO\\01.Generate_PE\
RCX   0000000000000000
RDX   00007FF6B76E2030     "calc.exe"
RBP   0000000000000000
RSP   00000030ECD1F8E0
RSI   0000000000000000
RDI   0000013FA3796BC0     &"ALLUSERSPROFILE=C:\\ProgramData"

R8    0000000000000000
R9    0000000000000000
R10   0000000000000050     'P'
R11   00000030ECD1F938
R12   0000000000000000
R13   0000000000000000
R14   0000000000000000
R15   0000000000000000
```

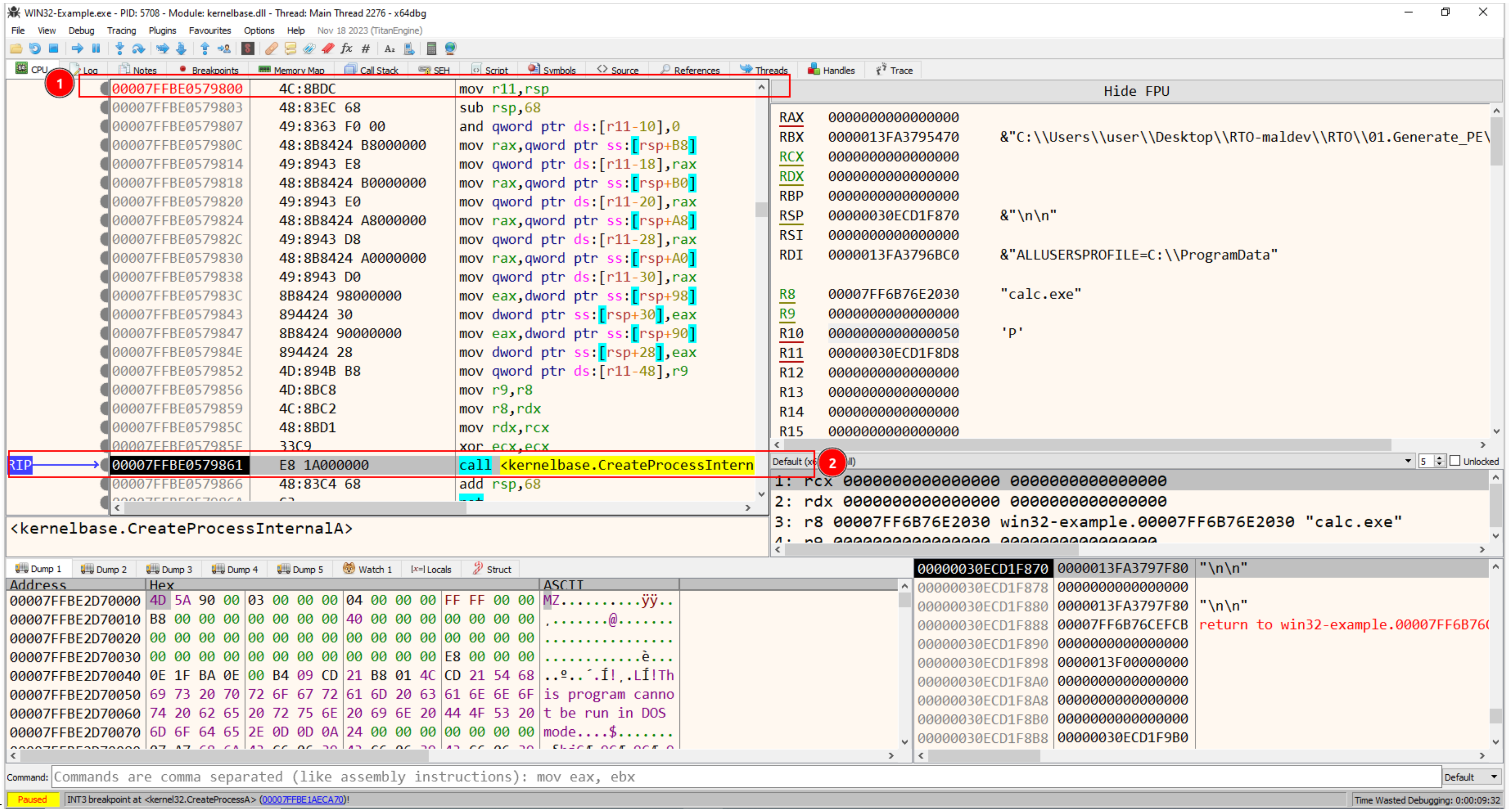
```
1: rcx 0000000000000000 0000000000000000
2: rdx 00007FF6B76E2030 win32-example.00007FF6B76E2030 "calc.exe"
3: r8 0000000000000000 0000000000000000
4: r9 0000000000000000 0000000000000000
```

```
Address            Hex                                               ASCII
00007FFBE2D70000   4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00   MZ..........ÿÿ..
00007FFBE2D70010   B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ¸.......@.......
00007FFBE2D70020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00007FFBE2D70030   00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00   ............è...
00007FFBE2D70040   0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ..º..´.Í!¸.LÍ!Th
00007FFBE2D70050   69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F   is program canno
00007FFBE2D70060   74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20   t be run in DOS 
00007FFBE2D70070   6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00   mode....$.......
```

```
00000030ECD1F8E0   00007FF6B76E2090   win32-example.00007FF6B76E2090
00000030ECD1F8E8   0000000000000000
00000030ECD1F8F0   000000000000000A
00000030ECD1F8F8   00007FF6B76C4E91   return to win32-example.00007FF6B76
00000030ECD1F900   00007FF600000000
00000030ECD1F908   0000000000000000
00000030ECD1F910   0000000000000000
00000030ECD1F918   0000000000000000
00000030ECD1F920   00000030ECD1F9B0
00000030ECD1F928   00000030ECD1F990
```

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

API Hooking: DEMO – From Win32 API to NTDLL


WIN32-Example.exe - PID: 5708 - Module: kernelbase.dll - Thread: Main Thread 2276 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Nov 18 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
1
2
RIP
Hide FPU
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct
Default 5 Unlocked
Command:
Default
Paused

```
00007FFBE0579800   4C:8BDC              mov r11,rsp
00007FFBE0579803   48:83EC 68           sub rsp,68
00007FFBE0579807   49:8363 F0 00        and qword ptr ds:[r11-10],0
00007FFBE057980C   48:8B8424 B8000000   mov rax,qword ptr ss:[rsp+B8]
00007FFBE0579814   49:8943 E8           mov qword ptr ds:[r11-18],rax
00007FFBE0579818   48:8B8424 B0000000   mov rax,qword ptr ss:[rsp+B0]
00007FFBE0579820   49:8943 E0           mov qword ptr ds:[r11-20],rax
00007FFBE0579824   48:8B8424 A8000000   mov rax,qword ptr ss:[rsp+A8]
00007FFBE057982C   49:8943 D8           mov qword ptr ds:[r11-28],rax
00007FFBE0579830   48:8B8424 A0000000   mov rax,qword ptr ss:[rsp+A0]
00007FFBE0579838   49:8943 D0           mov qword ptr ds:[r11-30],rax
00007FFBE057983C   8B8424 98000000      mov eax,dword ptr ss:[rsp+98]
00007FFBE0579843   894424 30            mov dword ptr ss:[rsp+30],eax
00007FFBE0579847   8B8424 90000000      mov eax,dword ptr ss:[rsp+90]
00007FFBE057984E   894424 28            mov dword ptr ss:[rsp+28],eax
00007FFBE0579852   4D:894B B8           mov qword ptr ds:[r11-48],r9
00007FFBE0579856   4D:8BC8              mov r9,r8
00007FFBE0579859   4C:8BC2              mov r8,rdx
00007FFBE057985C   48:8BD1              mov rdx,rcx
00007FFBE057985F   33C9                 xor ecx,ecx
00007FFBE0579861   E8 1A000000          call <kernelbase.CreateProcessIntern
00007FFBE0579866   48:83C4 68           add rsp,68

<kernelbase.CreateProcessInternalA>
```

```
RAX   0000000000000000
RBX   0000013FA3795470   &"C:\\Users\\user\\Desktop\\RTO-maldev\\RTO\\01.Generate_PE\
RCX   0000000000000000
RDX   0000000000000000
RBP   0000000000000000
RSP   00000030ECD1F870   &"\n\n"
RSI   0000000000000000
RDI   0000013FA3796BC0   &"ALLUSERSPROFILE=C:\\ProgramData"

R8    00007FF6B76E2030   "calc.exe"
R9    0000000000000000
R10   0000000000000050   'P'
R11   00000030ECD1F8D8
R12   0000000000000000
R13   0000000000000000
R14   0000000000000000
R15   0000000000000000
```

```
1: rcx 0000000000000000 0000000000000000
2: rdx 0000000000000000 0000000000000000
3: r8 00007FF6B76E2030 win32-example.00007FF6B76E2030 "calc.exe"
4: r9 0000000000000000 0000000000000000
```

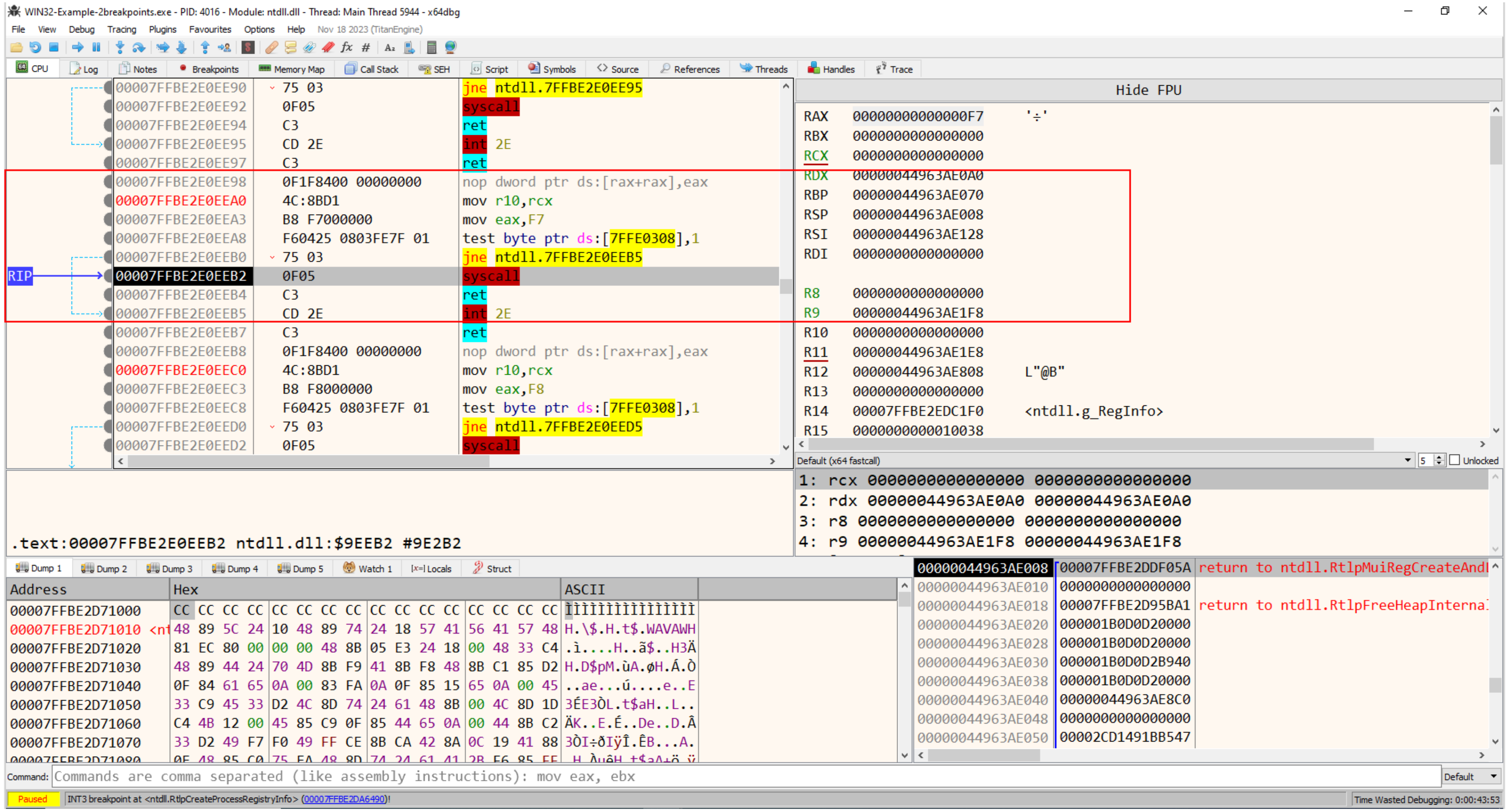
```
Address            Hex                                              ASCII
00007FFBE2D70000   4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00007FFBE2D70010   B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00007FFBE2D70020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00007FFBE2D70030   00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00  ............è...
00007FFBE2D70040   0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00007FFBE2D70050   69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00007FFBE2D70060   74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00007FFBE2D70070   6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
```

```
00000030ECD1F870   0000013FA3797F80   "\n\n"
00000030ECD1F878   0000000000000000
00000030ECD1F880   0000013FA3797F80   "\n\n"
00000030ECD1F888   00007FF6B76CEFCB   return to win32-example.00007FF6B76
00000030ECD1F890   0000000000000000
00000030ECD1F898   0000013F00000000
00000030ECD1F8A0   0000000000000000
00000030ECD1F8A8   0000000000000000
00000030ECD1F8B0   0000000000000000
00000030ECD1F8B8   00000030ECD1F9B0
```

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused   INT3 breakpoint at <kernel32.CreateProcessA> (00007FFBE1AECA70)!   Time Wasted Debugging: 0:00:09:32

API Hooking: DEMO – From Win32 API to NTDLL


File View Debug Tracing Plugins Favourites Options Help Nov 18 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
Hide FPU
Default (x64 fastcall)
5
Unlocked
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct
Default

WIN32-Example-2breakpoints.exe - PID: 4016 - Module: ntdll.dll - Thread: Main Thread 5944 - x64dbg

```
     00007FFBE2E0EE90   75 03              jne ntdll.7FFBE2E0EE95
     00007FFBE2E0EE92   0F05               syscall
     00007FFBE2E0EE94   C3                 ret
     00007FFBE2E0EE95   CD 2E              int 2E
     00007FFBE2E0EE97   C3                 ret
     00007FFBE2E0EE98   0F1F8400 00000000  nop dword ptr ds:[rax+rax],eax
     00007FFBE2E0EEA0   4C:8BD1            mov r10,rcx
     00007FFBE2E0EEA3   B8 F7000000        mov eax,F7
     00007FFBE2E0EEA8   F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
     00007FFBE2E0EEB0   75 03              jne ntdll.7FFBE2E0EEB5
RIP  00007FFBE2E0EEB2   0F05               syscall
     00007FFBE2E0EEB4   C3                 ret
     00007FFBE2E0EEB5   CD 2E              int 2E
     00007FFBE2E0EEB7   C3                 ret
     00007FFBE2E0EEB8   0F1F8400 00000000  nop dword ptr ds:[rax+rax],eax
     00007FFBE2E0EEC0   4C:8BD1            mov r10,rcx
     00007FFBE2E0EEC3   B8 F8000000        mov eax,F8
     00007FFBE2E0EEC8   F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
     00007FFBE2E0EED0   75 03              jne ntdll.7FFBE2E0EED5
     00007FFBE2E0EED2   0F05               syscall
```

```
RAX   00000000000000F7     '÷'
RBX   0000000000000000
RCX   0000000000000000
RDX   00000044963AE0A0
RBP   00000044963AE070
RSP   00000044963AE008
RSI   00000044963AE128
RDI   0000000000000000

R8    0000000000000000
R9    00000044963AE1F8
R10   0000000000000000
R11   00000044963AE1E8
R12   00000044963AE808     L"@B"
R13   0000000000000000
R14   00007FFBE2EDC1F0     <ntdll.g_RegInfo>
R15   0000000000010038
```

```
1: rcx 0000000000000000 0000000000000000
2: rdx 00000044963AE0A0 00000044963AE0A0
3: r8 0000000000000000 0000000000000000
4: r9 00000044963AE1F8 00000044963AE1F8
```

```
.text:00007FFBE2E0EEB2 ntdll.dll:$9EEB2 #9E2B2
```

```
Address               Hex                                              ASCII
00007FFBE2D71000      CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
00007FFBE2D71010 <nt  48 89 5C 24 10 48 89 74 24 18 57 41 56 41 57 48  H.\$.H.t$.WAVAWH
00007FFBE2D71020      81 EC 80 00 00 00 48 8B 05 E3 24 18 00 48 33 C4  .ì....H..ã$..H3Ä
00007FFBE2D71030      48 89 44 24 70 4D 8B F9 41 8B F8 48 8B C1 85 D2  H.D$pM.ùA.øH.Á.Ò
00007FFBE2D71040      0F 84 61 65 0A 00 83 FA 0A 0F 85 15 65 0A 00 45  ..ae...ú....e..E
00007FFBE2D71050      33 C9 45 33 D2 4C 8D 74 24 61 48 8B 00 4C 8D 1D  3ÉE3ÒL.t$aH..L..
00007FFBE2D71060      C4 4B 12 00 45 85 C9 0F 85 44 65 0A 00 44 8B C2  ÄK..E.É..De..D.Â
00007FFBE2D71070      33 D2 49 F7 F0 49 FF CE 8B CA 42 8A 0C 19 41 88  3ÒI÷ðIÿÎ.ÊB...A.
```

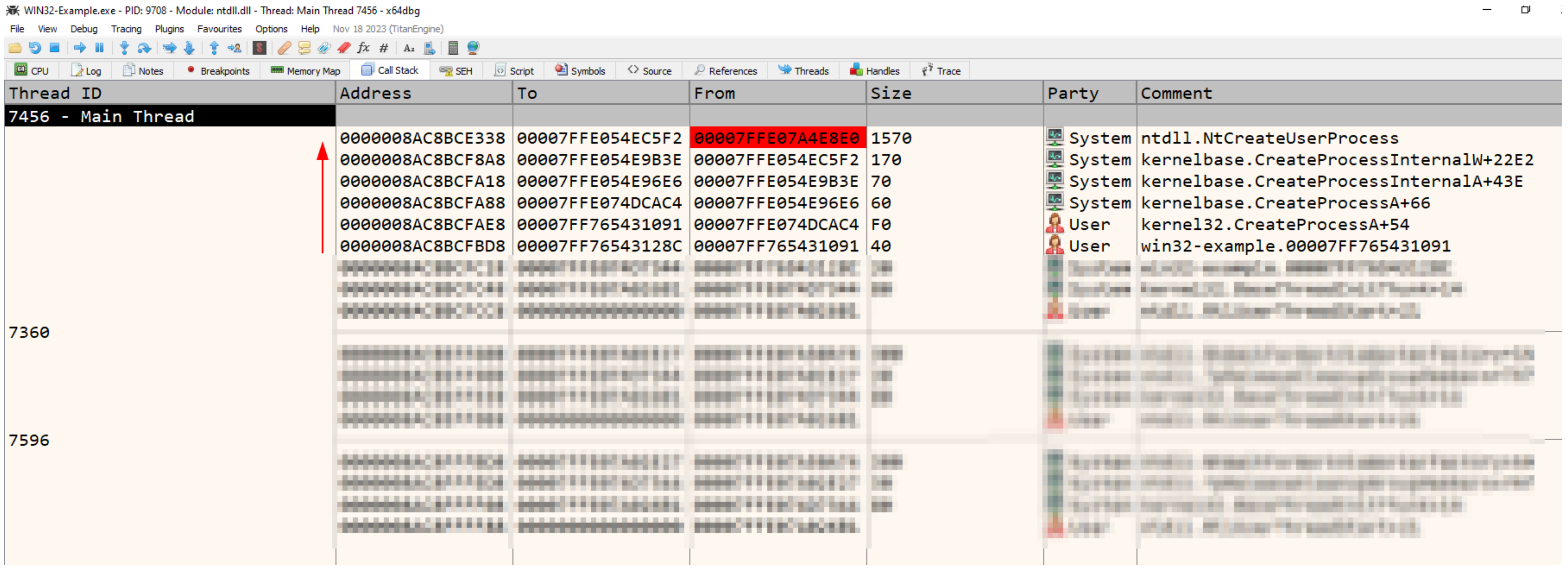
```
00000044963AE008  00007FFBE2DDF05A  return to ntdll.RtlpMuiRegCreateAndI
00000044963AE010  0000000000000000
00000044963AE018  00007FFBE2D95BA1  return to ntdll.RtlpFreeHeapInterna
00000044963AE020  000001B0D0D20000
00000044963AE028  000001B0D0D20000
00000044963AE030  000001B0D0D2B940
00000044963AE038  000001B0D0D20000
00000044963AE040  00000044963AE8C0
00000044963AE048  0000000000000000
00000044963AE050  00002CD1491BB547
```

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused INT3 breakpoint at <ntdll.RtlpCreateProcessRegistryInfo> (00007FFBE2DA6490)!

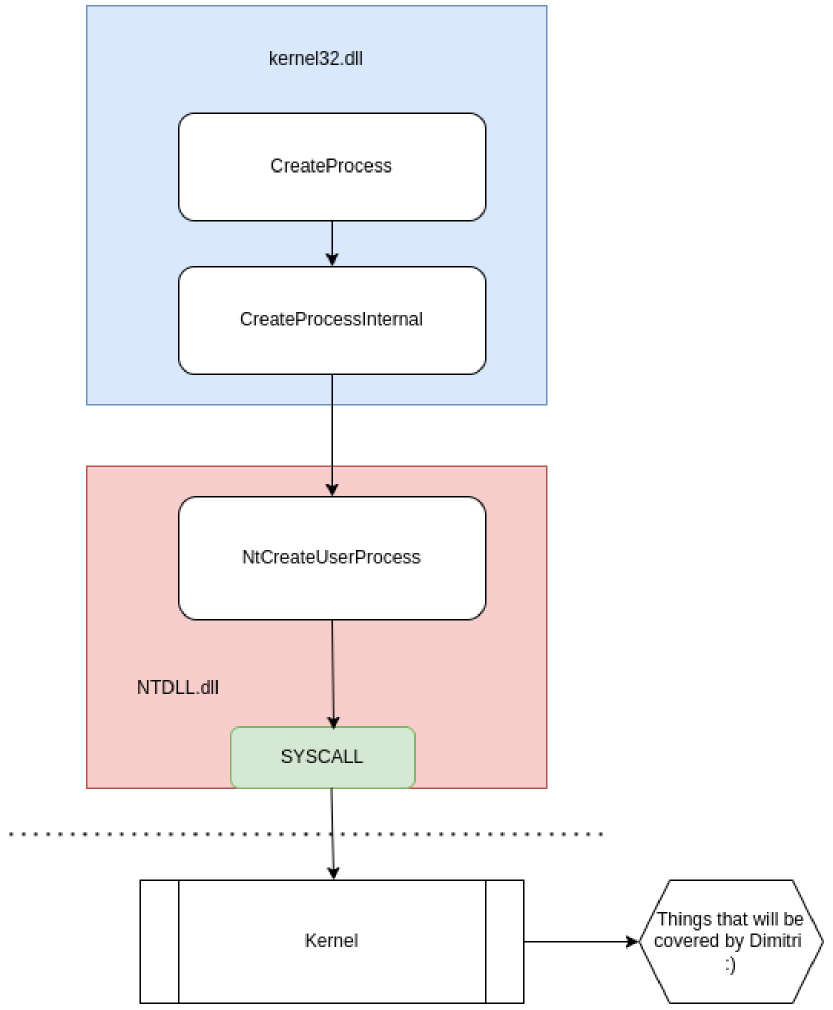
Time Wasted Debugging: 0:00:43:53

API Hooking: DEMO – From Win32 API to NTDLL


WIN32-Example.exe - PID: 9708 - Module: ntdll.dll - Thread: Main Thread 7456 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Nov 18 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace

Thread ID	Address	To	From	Size	Party	Comment
7456 - Main Thread						
	0000008AC8BCE338	00007FFE054EC5F2	00007FFE07A4E8E0	1570	System	ntdll.NtCreateUserProcess
	0000008AC8BCF8A8	00007FFE054E9B3E	00007FFE054EC5F2	170	System	kernelbase.CreateProcessInternalW+22E2
	0000008AC8BCFA18	00007FFE054E96E6	00007FFE054E9B3E	70	System	kernelbase.CreateProcessInternalA+43E
	0000008AC8BCFA88	00007FFE074DCAC4	00007FFE054E96E6	60	System	kernelbase.CreateProcessA+66
	0000008AC8BCFAE8	00007FF765431091	00007FFE074DCAC4	F0	User	kernel32.CreateProcessA+54
	0000008AC8BCFBD8	00007FF76543128C	00007FF765431091	40	User	win32-example.00007FF765431091
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
7360						
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
7596						
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

API Hooking: Where Shall the Hooks be placed?



API Hooking: When are the Hooks introduced and how can we evade that?

We need to understand

- What are DLLs
- User mode / Kernel mode
- What are the steps involved in process creation

API Hooking: DLLs

Almost the same as exe

- But, they need to be added into a process
- They contain functions!
- Made for reusability
- Functions can be set as **export so they can be used externally. The EAT table contains these functions**
- Logic is in the .text section

API Hooking: User mode / Kernel mode

Applications run in User Mode

- They interact at some point with the Kernel
- We can interact with our running applications (almost everything running under our context can be modified/checked)

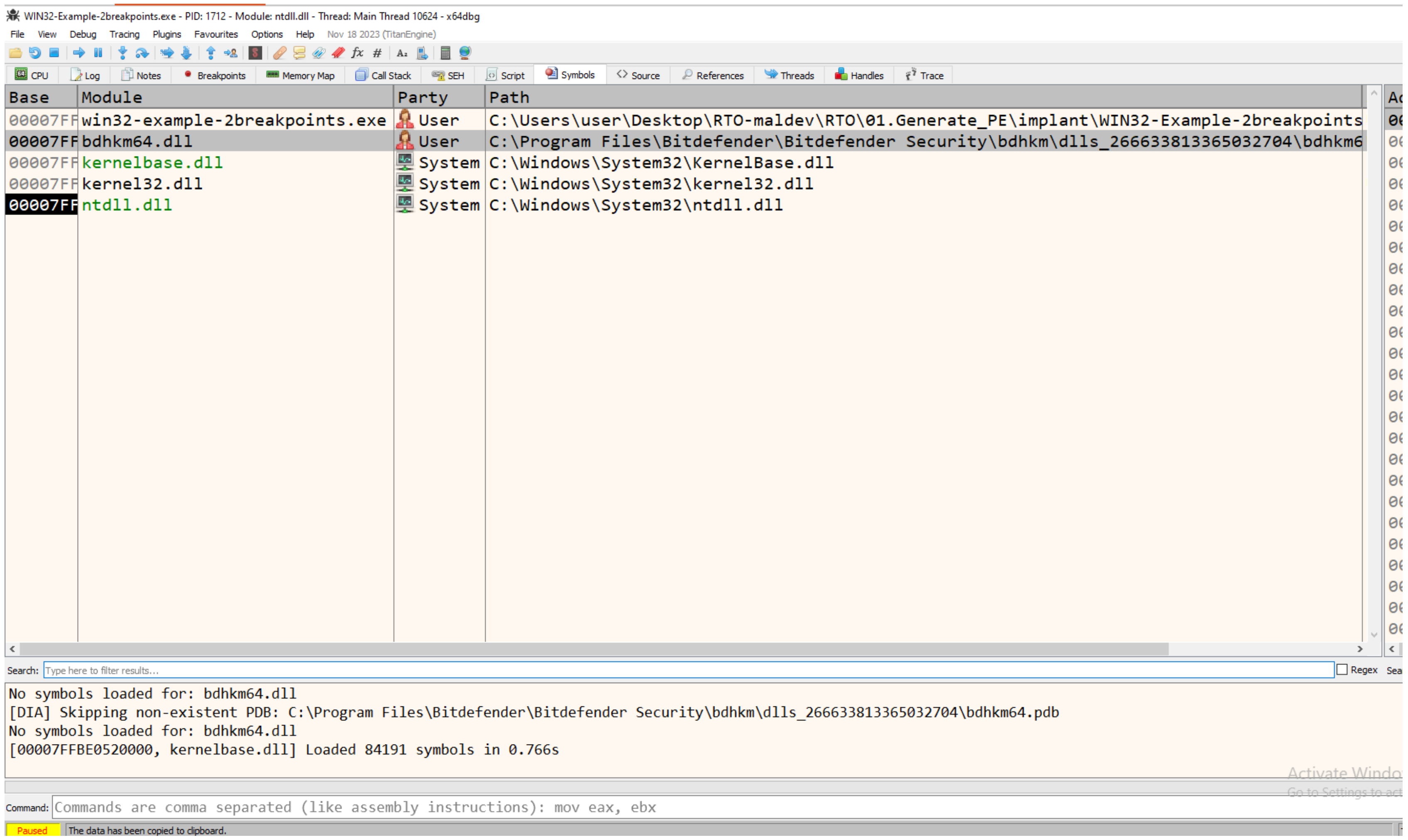
API Hooking: Process creation

- Parameters validation
- The .exe is opened
- …
- Process adress space is set up
- …
- PEB is set up
- …
- Set up the first thread and its TEB
- Load Modules (LDR)
- Start execution of the first thread

API Hooking: Process creation

- Parameters validation
- The .exe is opened
- …
- Process adress space is set up
- …
- PEB is set up
- …
- Set up the first thread and its TEB
- Load Modules (LDR)
- **Here!**
- Start execution of the first thread

API Hooking: When are the Hooks introduced?



WIN32-Example-2breakpoints.exe - PID: 1712 - Module: ntdll.dll - Thread: Main Thread 10624 - x64dbg

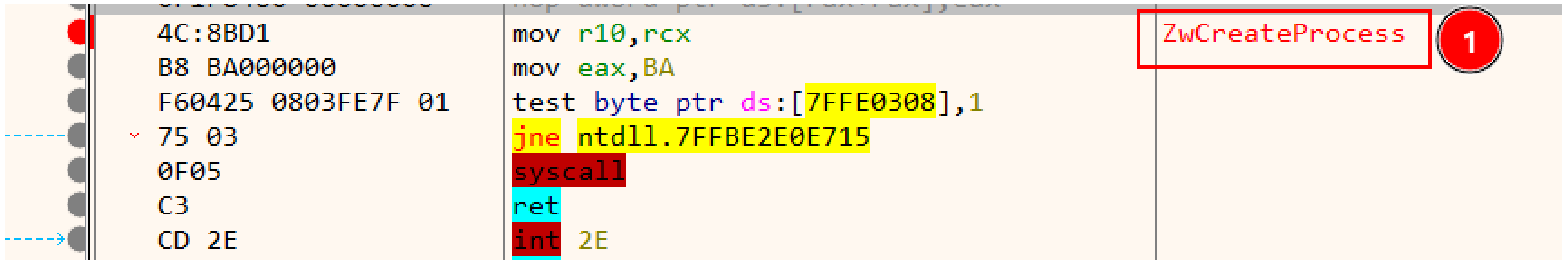
Base	Module	Party	Path
00007FF	win32-example-2breakpoints.exe	User	C:\Users\user\Desktop\RTO-maldev\RTO\01.Generate_PE\implant\WIN32-Example-2breakpoints
00007FF	bdhkm64.dll	User	C:\Program Files\Bitdefender\Bitdefender Security\bdhkm\dlls_266633813365032704\bdhkm6
00007FF	kernelbase.dll	System	C:\Windows\System32\KernelBase.dll
00007FF	kernel32.dll	System	C:\Windows\System32\kernel32.dll
00007FF	ntdll.dll	System	C:\Windows\System32\ntdll.dll

```
No symbols loaded for: bdhkm64.dll
[DIA] Skipping non-existent PDB: C:\Program Files\Bitdefender\Bitdefender Security\bdhkm\dlls_266633813365032704\bdhkm64.pdb
No symbols loaded for: bdhkm64.dll
[00007FFBE0520000, kernelbase.dll] Loaded 84191 symbols in 0.766s
```

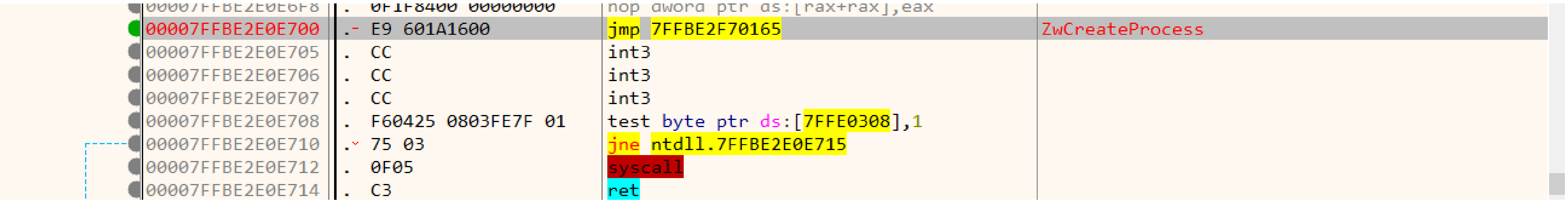
Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused The data has been copied to clipboard.

API Hooking: When are the Hooks introduced?



```
4C:8BD1                   mov r10,rcx                          ZwCreateProcess
B8 BA000000               mov eax,BA
F60425 0803FE7F 01        test byte ptr ds:[7FFE0308],1
75 03                     jne ntdll.7FFBE2E0E715
0F05                      syscall
C3                        ret
CD 2E                     int 2E
```



```
00007FFBE2E0E700  .- E9 601A1600             jmp 7FFBE2F70165                ZwCreateProcess
00007FFBE2E0E705  .  CC                      int3
00007FFBE2E0E706  .  CC                      int3
00007FFBE2E0E707  .  CC                      int3
00007FFBE2E0E708  .  F60425 0803FE7F 01      test byte ptr ds:[7FFE0308],1
00007FFBE2E0E710  .  75 03                   jne ntdll.7FFBE2E0E715
00007FFBE2E0E712  .  0F05                    syscall
00007FFBE2E0E714  .  C3                      ret
```

API Hooking – (Logical) Solutions out there

- Interfere with the code (Patching)
  - Process is running with in our User-Land, we can modify it if we want
  - How do we want to perform this operation?
    - Copy and Paste original NTDLL.DLL?
      - IOC: mapping of the fresh NTDLL (why would a process map NTDLL?)
      - IOC: mapping of the fresh NTDLL into our process (why would a process do this operation?)
    - Create a new process in a state where the hooks have not been applied yet and get the NTDLL from there

API Hooking – (Logical) Solutions out there

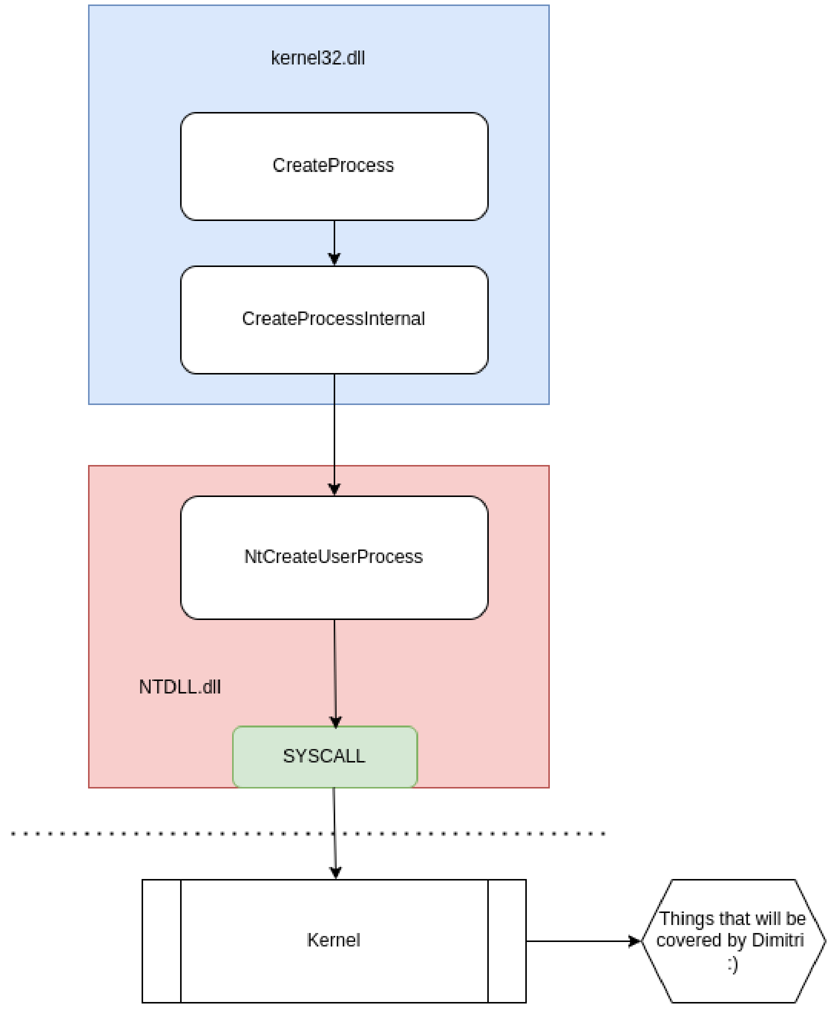
- Interfere with the code (Patching)
  - Process is running with in our User-Land, we can modify it if we want
  - How do we want to perform this operation?
    - Copy and Paste original NTDLL.DLL?
      - IOC: mapping of the fresh NTDLL (why would a process map NTDLL?)
      - IOC: mapping of the fresh NTDLL into our process (why would a process do this operation?)
    - Create a new process in a state where the hooks have not been applied yet and get the NTDLL from there

API Hooking – (Logical) Solutions out there

- Avoid the code. We need to «reconstruct» the method that we want to use
    - Dinvoke (C#)
        - Avoid having imports in the IAT
        - Runtime resolution of the NtCreateProcess function
            - DInvoke creates function pointers to the resolved addresses of the target functions.
            - Instead of directly calling the function by name, DInvoke invokes the function indirectly through these function pointers.
                - IOC: we are loading DLLs (modload)
                - IOC: CLR

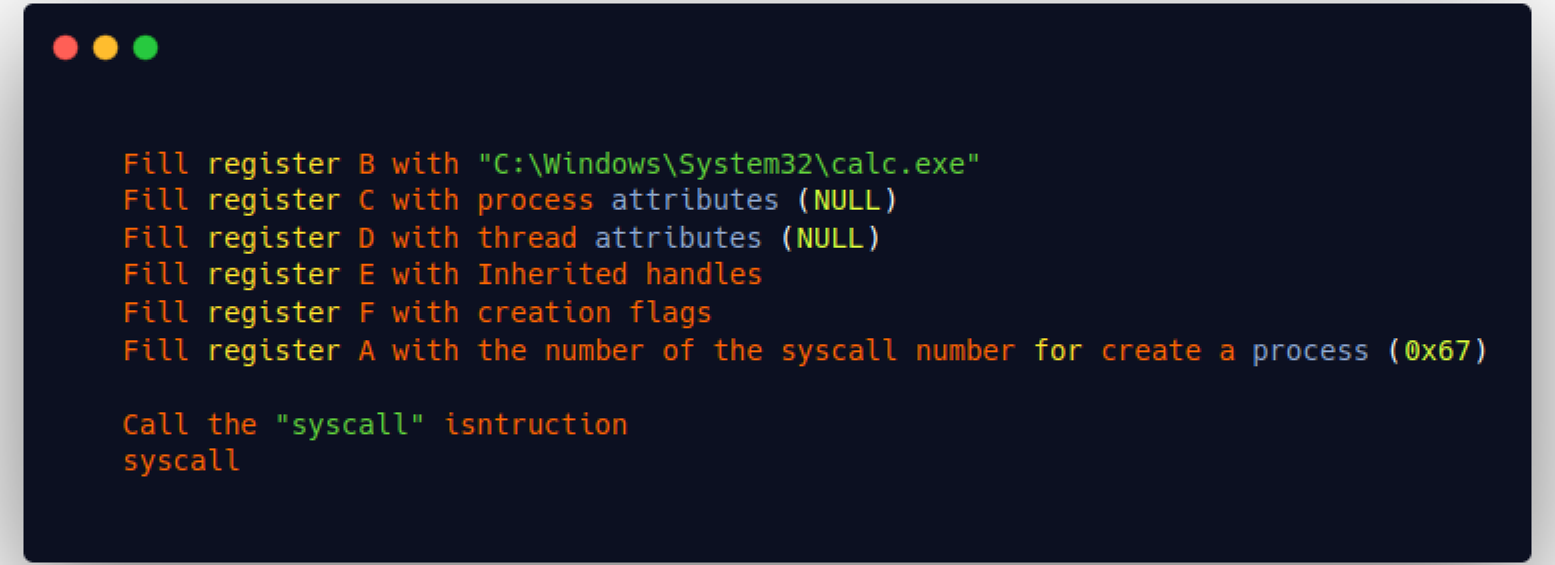
    - Use of direct **Syscalls**

Syscalls



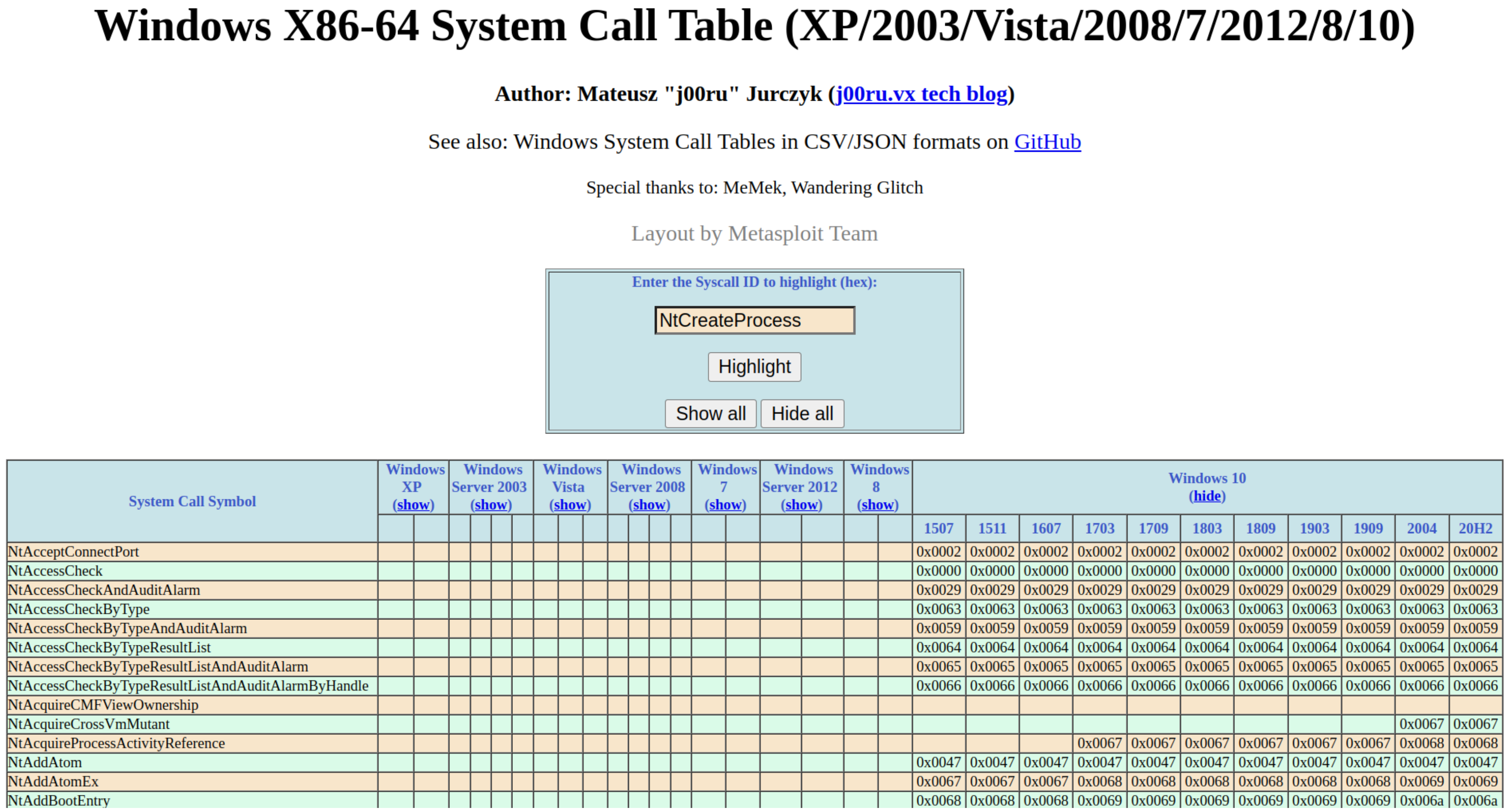
- The Real OG
- Created for reusability
- Lowest form of execution before jumping into Kernel mode
- Syscalls are defined with a number, each OS changes the number
- Syscalls are not really magic, things happen in the Kernel, but you will need to wait for Dimitri's talk!

Syscalls – How are they called



```
Fill register B with "C:\Windows\System32\calc.exe"
Fill register C with process attributes (NULL)
Fill register D with thread attributes (NULL)
Fill register E with Inherited handles
Fill register F with creation flags
Fill register A with the number of the syscall number for create a process (0x67)

Call the "syscall" isntruction
syscall
```

Syscalls – How to choose the syscall number



Windows X86-64 System Call Table (XP/2003/Vista/2008/7/2012/8/10)

Author: Mateusz "j00ru" Jurczyk (j00ru.vx tech blog)

See also: Windows System Call Tables in CSV/JSON formats on GitHub

Special thanks to: MeMek, Wandering Glitch

Layout by Metasploit Team

Enter the Syscall ID to highlight (hex):

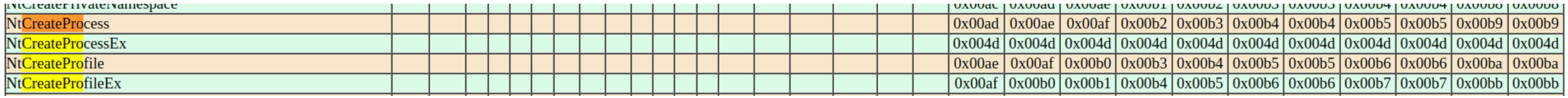
NtCreateProcess

Highlight

Show all Hide all

System Call Symbol	Windows XP (show)	Windows Server 2003 (show)	Windows Vista (show)	Windows Server 2008 (show)	Windows 7 (show)	Windows Server 2012 (show)	Windows 8 (show)	Windows 10 (hide)										
								1507	1511	1607	1703	1709	1803	1809	1903	1909	2004	20H2
NtAcceptConnectPort								0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
NtAccessCheck								0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
NtAccessCheckAndAuditAlarm								0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029
NtAccessCheckByType								0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063
NtAccessCheckByTypeAndAuditAlarm								0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059
NtAccessCheckByTypeResultList								0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064
NtAccessCheckByTypeResultListAndAuditAlarm								0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065
NtAccessCheckByTypeResultListAndAuditAlarmByHandle								0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066
NtAcquireCMFViewOwnership																		
NtAcquireCrossVmMutant																	0x0067	0x0067
NtAcquireProcessActivityReference											0x0067	0x0067	0x0067	0x0067	0x0067	0x0067	0x0068	0x0068
NtAddAtom								0x0047	0x0047	0x0047	0x0047	0x0047	0x0047	0x0047	0x0047	0x0047	0x0047	0x0047
NtAddAtomEx								0x0067	0x0067	0x0067	0x0068	0x0068	0x0068	0x0068	0x0068	0x0068	0x0069	0x0069
NtAddBootEntry								0x0068	0x0068	0x0068	0x0069	0x0069	0x0069	0x0069	0x0069	0x0069	0x006a	0x006a

Syscalls – How to choose the syscall number



NtCreatePrivateNamespace																				0x00ac	0x00ad	0x00ae	0x00b1	0x00b2	0x00b3	0x00b3	0x00b4	0x00b4	0x00b8	0x00b8
NtCreateProcess																				0x00ad	0x00ae	0x00af	0x00b2	0x00b3	0x00b4	0x00b4	0x00b5	0x00b5	0x00b9	0x00b9
NtCreateProcessEx																				0x004d	0x004d	0x004d	0x004d	0x004d	0x004d	0x004d	0x004d	0x004d	0x004d	0x004d
NtCreateProfile																				0x00ae	0x00af	0x00b0	0x00b3	0x00b4	0x00b5	0x00b5	0x00b6	0x00b6	0x00ba	0x00ba
NtCreateProfileEx																				0x00af	0x00b0	0x00b1	0x00b4	0x00b5	0x00b6	0x00b6	0x00b7	0x00b7	0x00bb	0x00bb

Syscalls – Main problem

`Inter-operability`

Syscalls are different on every OS (even in different versions of the same OS)

Syscalls were not created to be directly used

However, using them avoids any kind of hooking

Src: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts

Syscalls – Solving the problem, approaches

We could hardcode the syscalls in the code however that brings us new problems (Syswhispers)

This is all about situational awareness. We need use the **meta** Information that we have available

The **PEB structure** is a meta structure whose address lives in the register GS of a process.

From there, we can walk the PEB , to search for the LDR , which has the addresses of the different DLLs loaded.

PEB -> LDR -> Get base address of NTDLL -> EAT

Syscalls – Solutions bring new problems

Walking a DLL that has already been hooked is tricky. We can’t rely on use the same offsets (Hellsgate)

Same applies if important structures such as the EAT is hooked.

We need to be able to understand if we are hooked (inline vs EAT hooking):

- Are addressess successive?

Src: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts

THANKS!

SECFORCE

proud sponsor of

BEACON '23