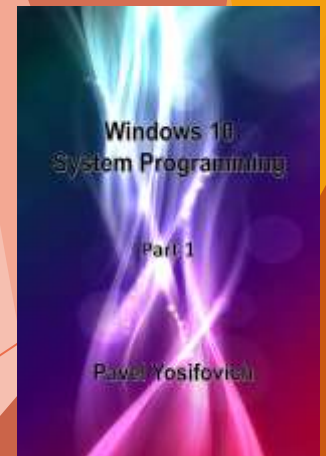
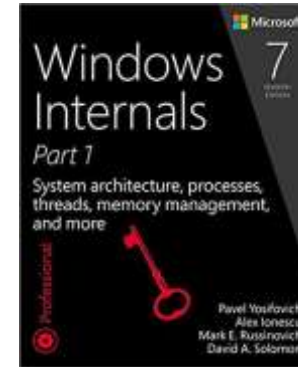


Native Applications: What, why and how

Pavel Yosifovich
@zodiacon

About Me

- ▶ Developer, Trainer, Author and Speaker
- ▶ Book author
 - ▶ “Windows Kernel Programming” (2019)
 - ▶ “Windows Internals 7th edition, Part 1” (co-author, 2017)
 - ▶ “Windows 10 System Programming, Part 1” (2020)
 - ▶ “Windows 10 System Programming, Part 2” (WIP)
- ▶ Video courses on Pluralsight and PentesterAcademy
- ▶ Author of several open-source tools (<http://github.com/zodiacon>)
- ▶ Website: <http://scorpiosoftware.net>



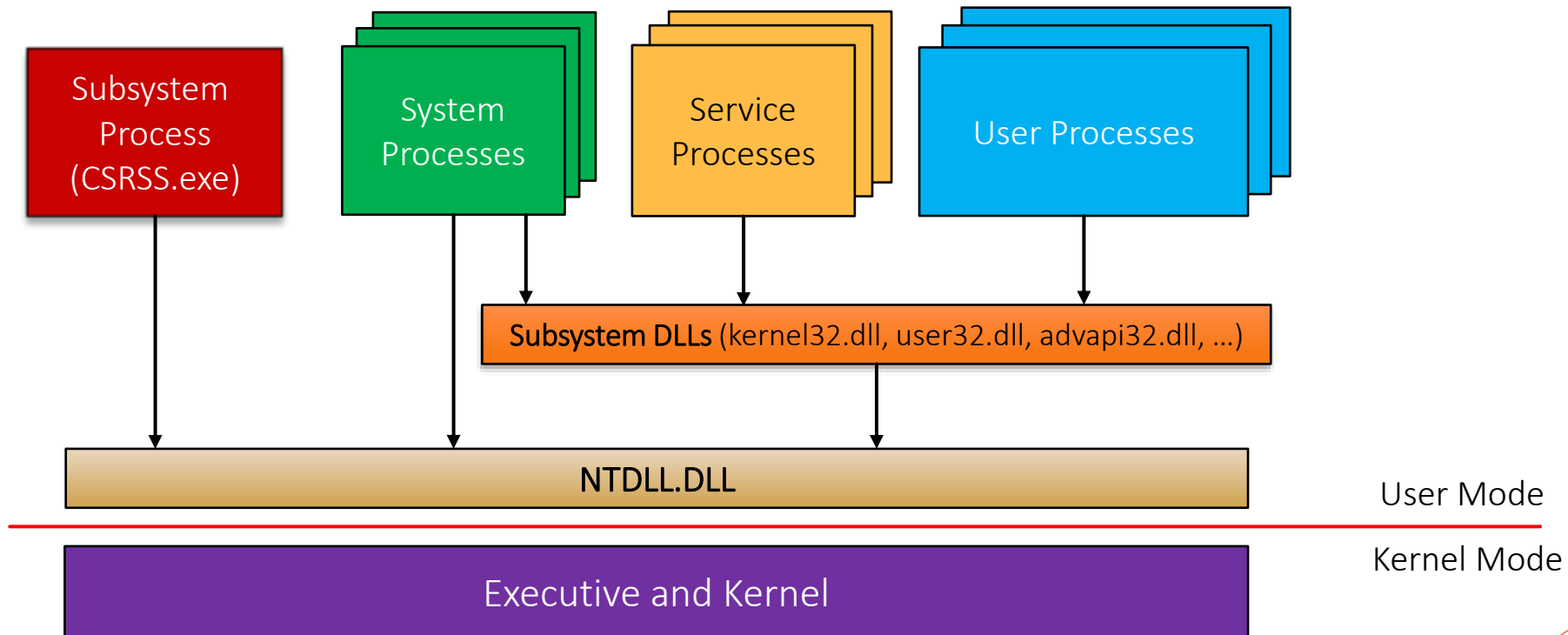
Agenda

- ▶ What is a native application?
- ▶ Why go native?
- ▶ Running native applications
- ▶ Writing native applications

Applications on Windows

- ▶ The first Windows NT version offered three types of applications
 - ▶ Windows, POSIX, OS2
- ▶ Each of these
 - ▶ Runs on top of a subsystem
 - ▶ Links against its own subsystem DLLs (API)
 - ▶ “Managed” by its subsystem process manager

Windows Architecture Overview



Subsystems Today

- ▶ OS 2 support dropped in Windows XP
 - ▶ POSIX support dropped in Windows 8.1
 - ▶ The winner is: The Windows subsystem
-
- ▶ Unrelated to the “Windows Subsystem for Linux” (WSL)

Native Applications

- ▶ Do not use any subsystem
- ▶ Link against NTDLL only
- ▶ What is inside NTDLL?
 - ▶ Image loader, heap manager, (some) thread pool support
 - ▶ Various CRT-like functions (`memset`, `sprintf`, ...)
 - ▶ System calls invokers

NTDLL.DLL Kernel Gate

► 32-bit dispatching code example (Windows 8.1)

```
ntdll!NtReadFile:
77cca930 b88a000000      mov     eax,8Ah
77cca935 e803000000      call   ntdll!NtReadFile+0xd (77cca93d)
77cca93a c22400         ret     24h
77cca93d 8bd4          mov     edx,esp
77cca93f 0f34          sysenter
77cca941 c3            ret
```

► 64-bit dispatching code example (Windows 10)

```
ntdll!NtReadFile:
00007ff9`7efc9fb0 4c8bd1      mov     r10,rcx
00007ff9`7efc9fb3 b806000000      mov     eax,6
00007ff9`7efc9fb8 f604250803fe7f01 test     byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff9`7efc9fc0 7503        jne     ntdll!NtReadFile+0x15 (00007ff9`7efc9fc5)
00007ff9`7efc9fc2 0f05        syscall
00007ff9`7efc9fc4 c3          ret
00007ff9`7efc9fc5 cd2e        int     2Eh
00007ff9`7efc9fc7 c3          ret
```


Why Write Native Applications?

- ▶ Applications running at Windows startup
 - ▶ Canonical example: *autochk.exe*
 - ▶ Native only
- ▶ Ssmss.exe launches native applications listed in the value *BootExecute* under the key *HKLM\System\CurrentControlSet\Control\Session Manager*
 - ▶ Executable must be placed in the *System32* directory



BootExecute Key

Launching Native Applications

- ▶ `CreateProcess` is unable to launch native applications
- ▶ A more elaborate set of steps is required



Launching Native Applications

Writing Native Applications

- ▶ Must use the native API only
 - ▶ Mostly undocumented
 - ▶ Some functions partially documented
 - ▶ Some functions documented indirectly in the Windows Driver Kit (WDK)
- ▶ Most (if not all) function prototypes are available in the *ProcessHacker/phnt* project on Github



Another Native Application

Debugging Native Applications

- ▶ Neither Visual Studio nor WinDbg is able to launch a native application
- ▶ Option 1: Use the native API in a standard Windows application and debug normally
 - ▶ Cannot be used to debug *BootExecute* apps
- ▶ Option 2: Use a kernel debugger



Debugging a Native Application

Summary

- ▶ It's possible to write native applications
- ▶ Stealthy processes, unknown to CSRSS
- ▶ Can execute early when *Smss* is the only user mode process alive
 - ▶ Execute with almost all privileges