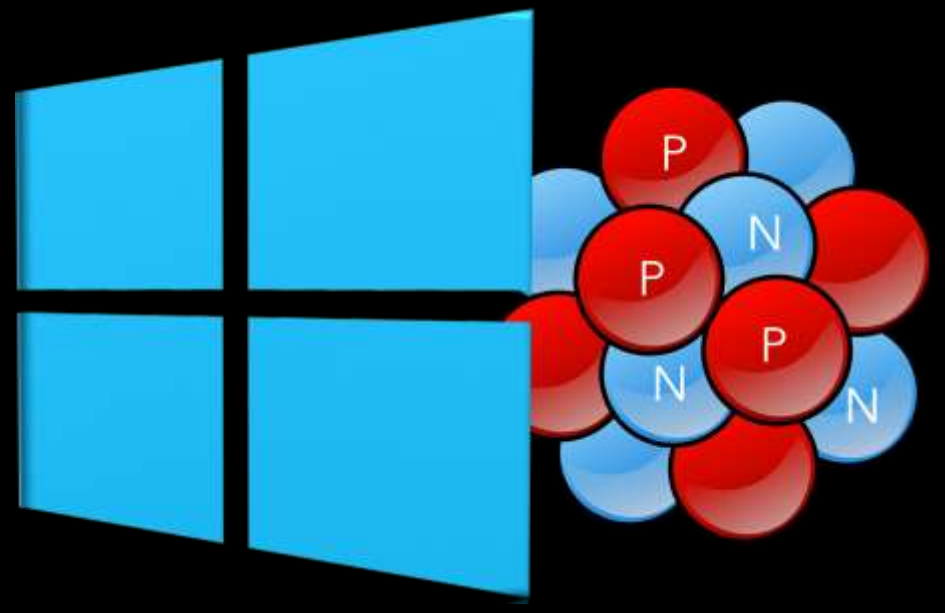
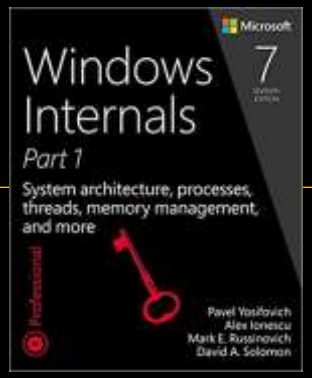
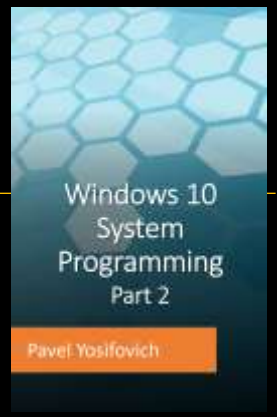
Native Powers: Using the Native API for Power and Flexibility

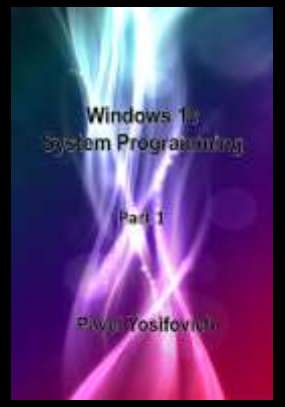
Pavel Yosifovich
@zodiacon



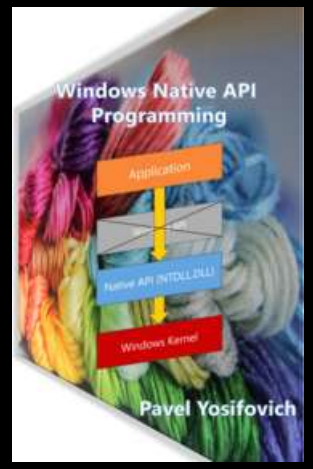
About Me







- Developer, Trainer, Author, Speaker
- Book author
  - "Windows Internals 7th edition, Part 1" (co-author, 2017)
  - "Windows 10 System Programming, Part 1+2" (2020/1)
  - "Windows Kernel Programming, 2nd ed." (2023)
  - "Windows Native API programming" (WIP)
- *Pluralsight* and *Pentester Academy* course author
- Founder at https://training.trainsec.net
- Author of several open-source tools (http://github.com/zodiacon)
- Website: http://scorpiosoftware.net



Agenda

- Windows Architecture Basics
- What is the Native API?
- Native Applications
- Exploring the Native API

Applications on Windows

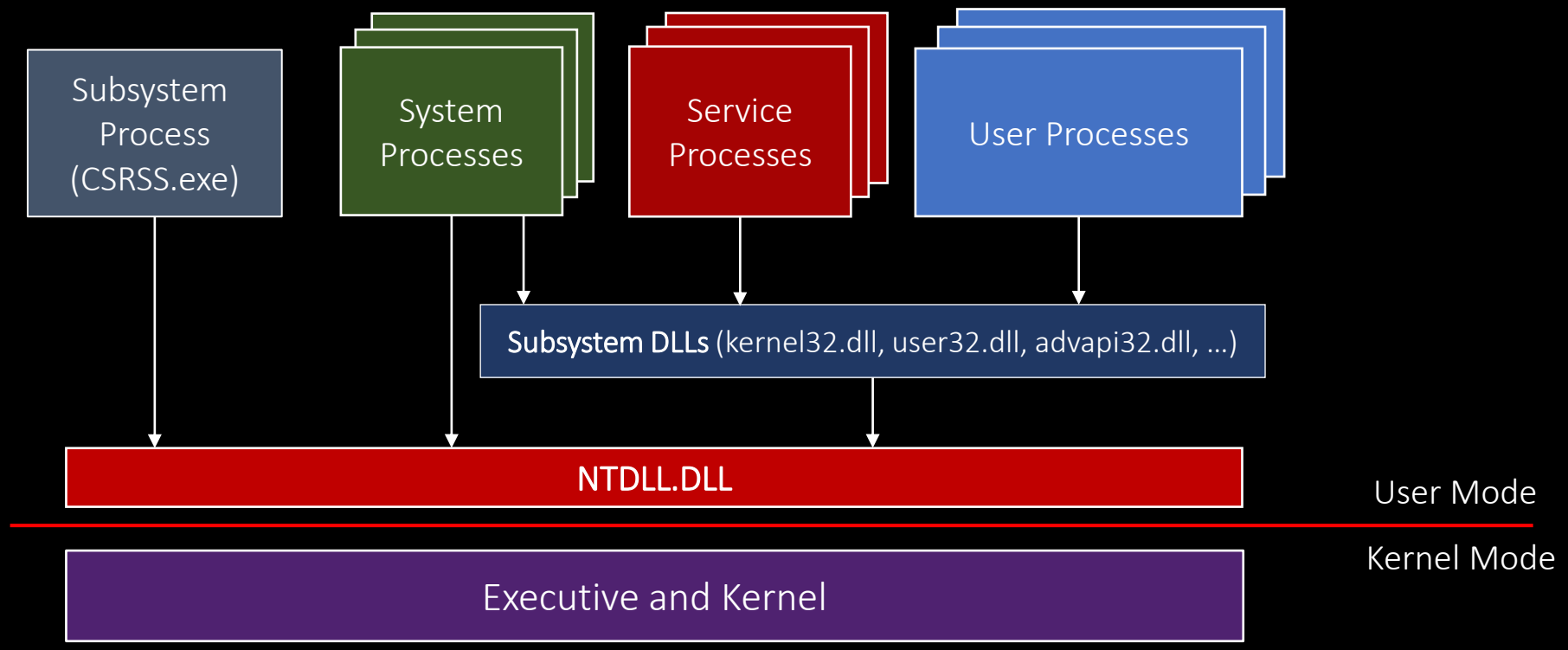
- The first Windows NT version offered three types of applications
  - Windows, POSIX, OS2
- Each of these
  - Runs on top of a subsystem
  - Links against its own subsystem DLLs (API)
  - “Managed” by its subsystem process manager

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)

Windows Architecture Overview



Native Applications

- Applications running at Windows startup
  - Canonical example: *autochk.exe*
  - Native only
- *Smss.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

What is Inside NTDLL?

- Image loader, heap manager, (some) thread pool support
- Various CRT-like functions
  - **`memset`**, **`sprintf`**, …
- System calls invokers (Nt…)
- Higher level functions, most of them eventually invoking system calls (Rtl…)

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
```

Windows API vs. Native API

	Windows API	Native API
Errors	**`GetLastError`** returns last error (usually) 0=no error, positive value=error	**`NTSTATUS`** return type 0=no error, negative value=error
Strings	C-style Unicode or ANSI strings	**`UNICODE_STRING`**
Common Options	Specific to the API	**`OBJECT_ATTRIBUTES`**
APIs names	No general convention	Most APIs start with **`Nt`** or **`Rtl`**
API style	Specific	Generic
Documentation	Full	Very Limited (mostly in the WDK)
Headers	Provided by Microsoft (Windows SDK)	Best source: *phnt* project on Github

System Information

```
NTSTATUS NTAPI NtQuerySystemInformation(
    _In_ SYSTEM_INFORMATION_CLASS SystemInformationClass,
    _Out_ PVOID SystemInformation,
    _In_  ULONG Length,
    _Out_opt_ PULONG ReturnLength);
```

```
typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemBasicInformation, // q: SYSTEM_BASIC_INFORMATION
    SystemProcessorInformation, // q: SYSTEM_PROCESSOR_INFORMATION
    SystemPerformanceInformation, // q: SYSTEM_PERFORMANCE_INFORMATION
    SystemTimeOfDayInformation, // q: SYSTEM_TIMEOFDAY_INFORMATION
    SystemPathInformation, // not implemented
    SystemProcessInformation, // q: SYSTEM_PROCESS_INFORMATION
    SystemCallCountInformation, // q: SYSTEM_CALL_COUNT_INFORMATION
    SystemDeviceInformation, // q: SYSTEM_DEVICE_INFORMATION
    SystemProcessorPerformanceInformation, // q: SYSTEM_PROCESSOR_PERFORMANCE_INFORMATION
    SystemFlagsInformation, // q: SYSTEM_FLAGS_INFORMATION
    //...
    SystemFeatureUsageSubscriptionInformation,
    SystemSecureSpeculationControlInformation,
    SystemSpacesBootInformation = 214,
    SystemFwRamdiskInformation = 215,
    SystemWheaIpmiHardwareInformation = 216,
    SystemDifSetRuleClassInformation = 217,
    SystemDifClearRuleClassInformation = 218,
    SystemDifApplyPluginVerificationOnDriver = 219,
    SystemDifRemovePluginVerificationOnDriver = 220,
    SystemShadowStackInformation = 221, // SYSTEM_SHADOW_STACK_INFORMATION
    SystemBuildVersionInformation = 222, // SYSTEM_BUILD_VERSION_INFORMATION
} SYSTEM_INFORMATION_CLASS;
```

Process Enumeration

- Documented Windows API functions like `CreateToolhelp32Snapshot`, `WTLEnumerateProcesses`, `EnumProcesses` provide certain details
  - Pale in comparison to `NtQuerySystemInformation` with `System(Extended)ProcessInformation`
  - Includes thread information
- More process information is available with `NtQueryInformationProcess`

SYSTEM_PROCESS_INFORMATION

```
typedef struct _SYSTEM_PROCESS_INFORMATION {
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    LARGE_INTEGER WorkingSetPrivateSize;
    ULONG HardFaultCount;
    ULONG NumberOfThreadsHighWatermark;
    ULONGLONG CycleTime;
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ImageName;
    KPRIORITY BasePriority;
    HANDLE UniqueProcessId;
    HANDLE InheritedFromUniqueProcessId;
    ULONG HandleCount;
    ULONG SessionId;
    ULONG_PTR UniqueProcessKey;
    SIZE_T PeakVirtualSize;
    SIZE_T VirtualSize;
    ULONG PageFaultCount;
    SIZE_T PeakWorkingSetSize;
    SIZE_T WorkingSetSize;
    SIZE_T QuotaPeakPagedPoolUsage;
    SIZE_T QuotaPagedPoolUsage;
    SIZE_T QuotaPeakNonPagedPoolUsage;
    SIZE_T QuotaNonPagedPoolUsage;
    SIZE_T PagefileUsage;
    SIZE_T PeakPagefileUsage;
    SIZE_T PrivatePageCount;
```

Admin

```
typedef struct _SYSTEM_PROCESS_INFORMATION_EXTENSION {
    PROCESS_DISK_COUNTERS DiskCounters;
    ULONGLONG ContextSwitches;
    union {
        ULONG Flags;
        struct {
            ULONG HasStrongId : 1;
            ULONG Classification : 4;
            ULONG BackgroundActivityModerated : 1;
            ULONG Spare : 26;
        };
    };
    ULONG UserSidOffset;
    ULONG PackageFullNameOffset;        // since THRESHOLD
    PROCESS_ENERGY_VALUES EnergyValues; // since THRESHOLD
    ULONG AppIdOffset;                  // since THRESHOLD
    SIZE_T SharedCommitCharge;          // since THRESHOLD2
    ULONG JobObjectId;                  // since REDSTONE
    ULONG SpareUlong;                   // since REDSTONE
    ULONGLONG ProcessSequenceNumber;
} SYSTEM_PROCESS_INFORMATION_EXTENSION;
```

```
    LARGE_INTEGER ReadOperationCount;
    LARGE_INTEGER WriteOperationCount;
    LARGE_INTEGER OtherOperationCount;
    LARGE_INTEGER ReadTransferCount;
    LARGE_INTEGER WriteTransferCount;
    LARGE_INTEGER OtherTransferCount;
    SYSTEM_THREAD_INFORMATION Threads[1];
} SYSTEM_PROCESS_INFORMATION;
```

Objects and Handles

- Kernel Objects are data structures in kernel space
  - Processes, threads, mutexes, jobs, files, events, desktops, sections, …
- User-mode code can only access objects through opaque handles
  - Private to a process
- Kernel objects can be shared between processes

Enumerating Handles

- The Windows API does not provide a way to enumerate handles in a process or the system
- The native API supports this with `NtQuerySystemInformation` with `SystemHandleInformation`
  - Enumerates handles in all processes, including protected and PPL processes

Opening a Named Section

```
HANDLE hSection = OpenFileMapping(FILE_MAP_READ, FALSE,
     L"\\\\.\\GLOBALROOT\\KnownDlls\\kernelbase.dll");
```

- hSection returns NULL
- GetLastError() returns 0xe1 (ERROR_BAD_PATHNAME)

```
UNICODE_STRING name;
RtlInitUnicodeString(&name, L"\\KnownDlls\\kernelbase.dll");
OBJECT_ATTRIBUTES oa = RTL_CONSTANT_OBJECT_ATTRIBUTES(&name, 0);
status = NtOpenSection(&hSection, SECTION_MAP_READ, &oa);
```

- status returns STATUS_SUCCESS (0)

Example: Process Operations

- Suspend a process

```
NTSTATUS NtSuspendProcess(_In_ HANDLE ProcessHandle);
```

- Resume a process

```
NTSTATUS NtResumedProcess(_In_ HANDLE ProcessHandle);
```

- Getting process PEB
  - NtQueryInformationProcess with ProcessBasicInformation

Memory

- The “Virtual” API
  - NtAllocateVirtualMemory
  - NtReadVirtualMemory, NtWriteVirtualMemory
  - NtQueryVirtualMemory
- Heaps
  - NtAllocateHeap
  - NtCreateHeap
  - RtlQueryHeapInformation
    - Allows looking into other processes’ heaps

More Native APIs

- System Information
- ALPC
- I/O
- Security
- Threading
- Registry

Summary

- The Native API is the most direct channel to the kernel
- Mostly undocumented by powerful
- https://github.com/zodiacon/NativePowers