

# Windows Syscall Quiz

by Mateusz Jurczyk (j00ru)

Do you consider yourself a Windows internals expert? If you do, then try to correctly answer the following questions. If not, feel free to follow along and hopefully learn some interesting facts about the kernel of the most popular desktop operating system in the world. ☺

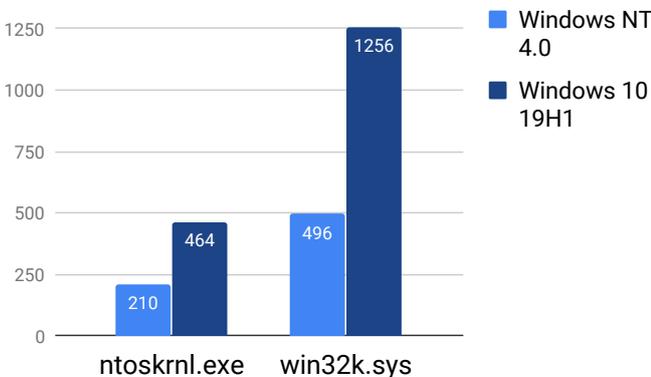
## The quiz:

1. How many syscalls do Windows NT 4.0 and Windows 10 19H1 have, i.e. how much has the system call table grown in the 23 years between 1996–2019?
2. Are there differences in the syscall interfaces between various editions of the same versions of Windows?
3. Have any legitimate driver ever registered their own syscall table(s) beyond the standard `ntoskrnl.exe` and `win32k.sys`?

Ready? Let's see how you did!

## Question 1: syscall table growth

The first release of Windows NT 4.0 Workstation had 210 core system calls and 496 graphical ones, adding up to a total of 706. At the time of this writing, the latest 32-bit build of Windows 10 declares  $464 + 1256 = 1720$  syscalls:



This is a 143% increase in the size of the interface, which is an attack surface available to locally running code. In other words, a new system call has been added on average every week for the past two decades. Of course it is not a fully precise metric as it doesn't account for code hidden behind the `win32k!NtUserCall` family, IOCTLs and many other factors, but it does illustrate the growth of the kernel complexity over time. Fortunately, starting with Windows 8 developers can restrict access to parts of the attack surface for their sandboxed processes, thanks to new features such as the system call disable policy<sup>1</sup>.

<sup>1</sup>[https://docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-process\\_mitigation\\_system\\_call\\_disable\\_policy](https://docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-process_mitigation_system_call_disable_policy)

## Question 2: cross-edition differences

As a general rule, various editions of the same OS (*Home*, *Pro*, *Enterprise* etc.) use the same underlying kernel and thus share the same set of system calls. However, there is one notable exception. In May 2019, I noticed that in the syscall tables served on my blog, there were a few names only present in Windows NT 4.0 SP4, but not in SP3, SP5, or any other system. One such symbol was `NtCreateWinStation`:

System Call Symbol	Windows NT (hide)			
	SP3	SP4	SP5	SP6
<code>NtCreateToken</code>	0x0026	0x0026	0x0026	0x0026
<code>NtCreateWinStation</code>		0x00d3		
<code>NtDelayExecution</code>	0x0027	0x0027	0x0027	0x0027

After a brief evening research with Gynvael Coldwind, we figured out that these syscalls (5 of them in total) were only found in the *Terminal Server Edition* of Windows NT, released two years after *Workstation*. Considering that the data came from the original table created by skape and hosted by Metasploit, the list for SP4 must have been extracted from a TS version of the system, unlike for other service packs, and so it has stayed this way up until recently. And so the riddle was solved.

## Question 3: non-standard syscall tables

In that same evening, we decided to finally establish if there ever had been real syscalls with IDs above `0x2000`, i.e. registered in the SSDT by a non-standard driver. We had heard rumors about IIS doing it at some point in time, but we had never observed it in real life.

Very quickly, we found several online sources confirming that story for IIS4 and IIS5, on Windows NT-2000. Some of them pointed us to a driver called `SPUD.sys`, which stands for *Special Purpose Utility Driver* (if you find that name funny, check the story behind `afd.sys`). We found the driver on an extra Option Pack CD for Windows NT, and on the standard installation disk of Windows 2000. This way, we confirmed that it indeed called `KeAddSystemServiceTable` with 9 entries in IIS4 and 7 entries in IIS5. We also learned that the associated ring-3 library was `isatq.dll`, with “atq” meaning *asynchronous thread queue*. The only missing piece were the names of the syscalls.

After another while of recon, we managed to dig out the symbols for both versions, in a dedicated `.cab` archive (NT) and a complete system symbols package (2000). Our curiosity was finally satisfied, with the mysterious syscalls turning out to be `SPUD{Initialize, Terminate, TransmitFileAndRecv, SendAndRecv, Cancel, GetCounts, CreateFile}` in IIS5, with the addition of `SPUDCheckStatus` and `SPUDOplockAcknowledge` in IIS4. It was a fun archaeological adventure into operating system prehistory.