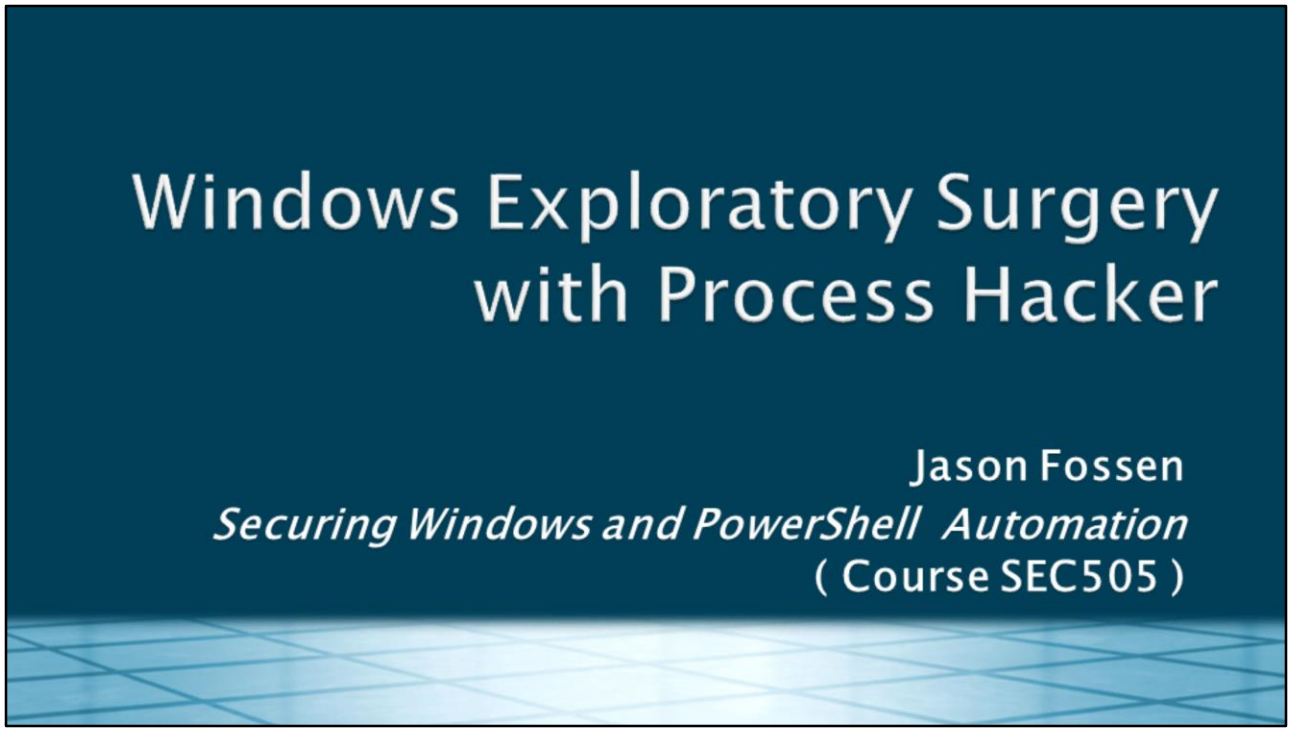


Welcome!

This talk will use the free, open source tool named "Process Hacker" to explore Windows processes, threads, handles and other operating system internals. You can get Process Hacker from **https://github.com/processhacker**. Having at least a basic understanding of these OS internals is important for combating malware, doing forensics, configuration hardening, troubleshooting and many other Windows tasks which require getting your hands dirty inside the OS.

Securing Windows and PowerShell Automation (SEC505) at SANS

The speaker's six-day *Securing Windows and PowerShell Automation* course at SANS (SEC505) is intended for those specializing in Microsoft Windows security (https://sans.org/sec505). You can also download the course author's PowerShell scripts related to the course from **http://SEC505.com**. The scripts are all in the public domain.

Recommended Reading

By far the most useful and authoritative book related to this talk is the latest edition of *Windows Internals* (Microsoft Press) by Mark Russinovich, David Solomon and Alex Ionescu. After that, visit https://docs.microsoft.com/sysinternals/ and download every Sysinternals tool listed there.

About The Speaker

Jason Fossen is an independent consultant in Dallas, Texas (Enclave Consulting LLC), a SANS Institute Fellow, author of the Windows day of *Security Essentials* (SEC401.5), author of the six-day *Securing Windows and PowerShell Automation* course (SEC505) at SANS, plus other past SANS courses.

Download this presentation and the author's PowerShell scripts from GitHub (look in the SANS-Slides section):

http://SEC505.com

For more information about his six-day course at SANS, *Securing Windows and PowerShell Automation* (course number SEC505), please visit:

https://www.sans.org/SEC505

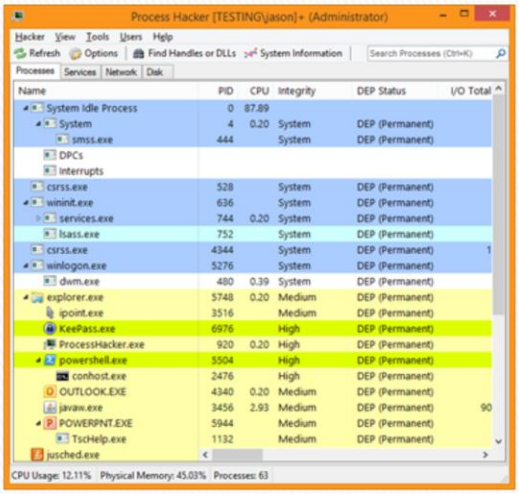
You're invited to connect with the author on social media:

Twitter: @JasonFossen

Linkedin: https://www.linkedin.com/in/JasonFossen

What Is Process Hacker?

- A free graphical tool for managing processes, threads, memory, handles, modules, and tokens on Windows.
- Similar to Microsoft's *Process Explorer*, but open source and a bit more fun.
- Great for forensics, combating malware, troubleshooting, and Windows exploration.

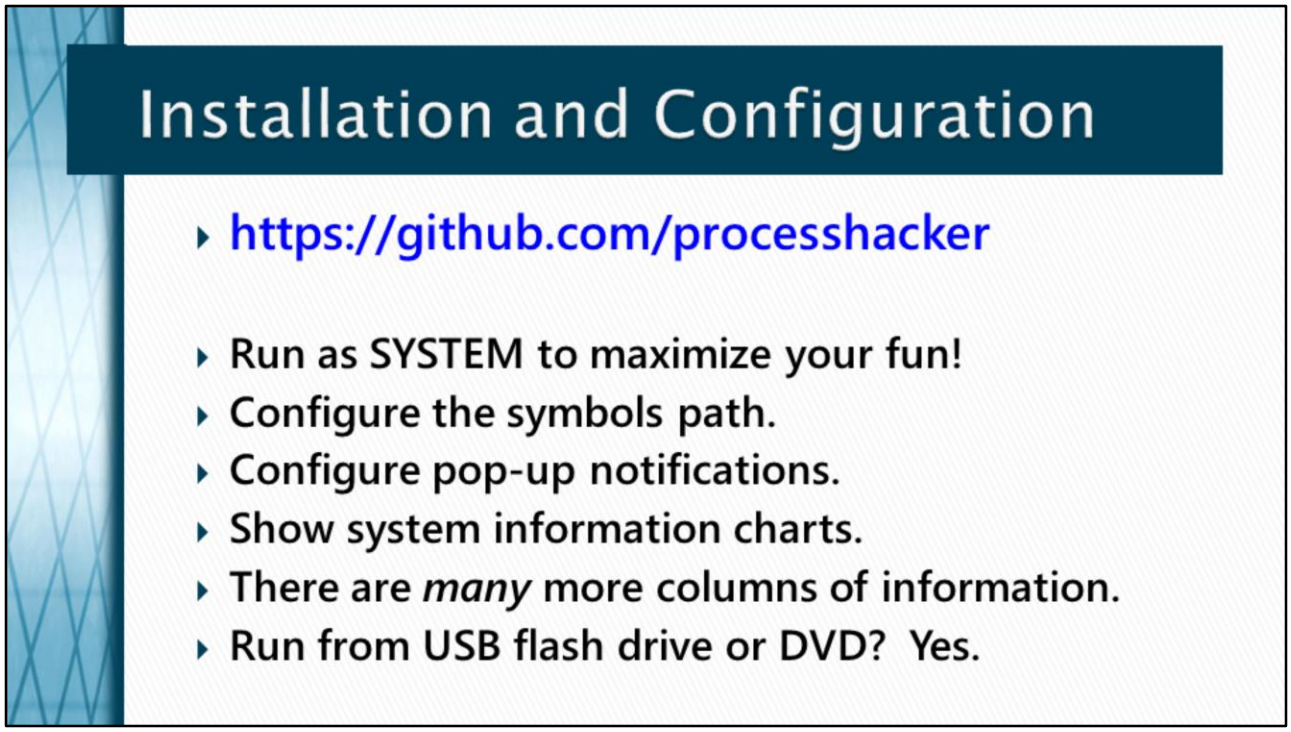


What Is Process Hacker?

Process Hacker is a free, open source, graphical tool for managing 32-bit and 64-bit Microsoft Windows processes, services, threads, memory, handles, modules, Security Access Tokens (SATs) and network connections. It is a wonderful tool for analyzing and combating malware, understanding low-level details of the Windows operating system, troubleshooting, and experimentation.

Process Hacker is similar to the famous Sysinternals Process Explorer tool from Microsoft (http://www.microsoft.com/sysinternals), but it is open source and a bit more fun. There are also no legal hassles when redistributing Process Hacker or its source code (no Microsoft lawyers = good thing). Examining the source code of Process Hacker is an interesting way to learn more about Windows internals. Process Hacker itself is an actively maintained project.

Fortunately, if you prefer Sysinternals Process Explorer, almost all of this presentation applies to that tool as well. So please feel free to use Process Hacker or Process Explorer as you wish. Both tools are great! And if you have questions, don't forget about the discussion forums for Process Hacker (http://processhacker.sourceforge.net/forums/) and Sysinternals Process Explorer (http://forum.sysinternals.com).



Installation and Configuration

Process Hacker installs on 32-bit or 64-bit Windows XP-SP2 and later. Download binaries and source from https://github.com/processhacker. By default, Process Hacker installs into C:\Program Files\Process Hacker\ProcessHacker.exe.

> **Note:** This presentation discusses **Process Hacker 3.0**, which was still in beta at the time of writing, but should be released soon. Get the nightly experimental builds, which may be unstable, from https://wj32.org/processhacker/nightly.php.

If you want to run Process Hacker from a USB drive, just copy the download zip file's contents to the USB drive and launch ProcessHacker.exe. In the same folder as ProcessHacker.exe, create a file named ProcessHacker.exe.settings.xml and your configuration changes will be saved there automatically.

Almost always, you will want to launch Process Hacker by right-clicking its shortcut and running as administrator. To maximize your fun (and potential damage to your computer), launch Process Hacker as SYSTEM. To do this, launch Process Hacker as administrator > pull down the Hacker menu > Run As > enter the path to ProcessHacker.exe > select NT AUTHORITY\SYSTEM as the user name, set the type to Service, select your session ID > click OK. After a few seconds, a new Process Hacker

instance should launch; look for "SYSTEM" in the title bar instead of your own user name. (If this fails, try using Process Hacker 3.0 or later.)

(Another way to run Process Hacker as SYSTEM is to create a shortcut to run the Sysinternals psexec.exe tool like so: "psexec.exe -s -d -i 1 processhacker.exe".)

To have pop-up notifications appear when services and/or processes are created or destroyed, pull down the Hacker menu > Options > Notifications. In this same dialog box, you can also log notifications to a text file, send notifications to Growl (not included), and filter exactly when notifications are displayed. Then, right-click the icon for Process Hacker in the notification area of the taskbar > Notifications > choose which to see.

In the toolbar, click the "System Information" button to display real-time charts of CPU, disk, network and memory utilization.

In order to see the names of functions in stack traces, create a folder named "C:\Symbols" on your hard drive. Then, in Process Hacker, pull down the Hacker menu > Options > General > and enter the following string for the symbols path:

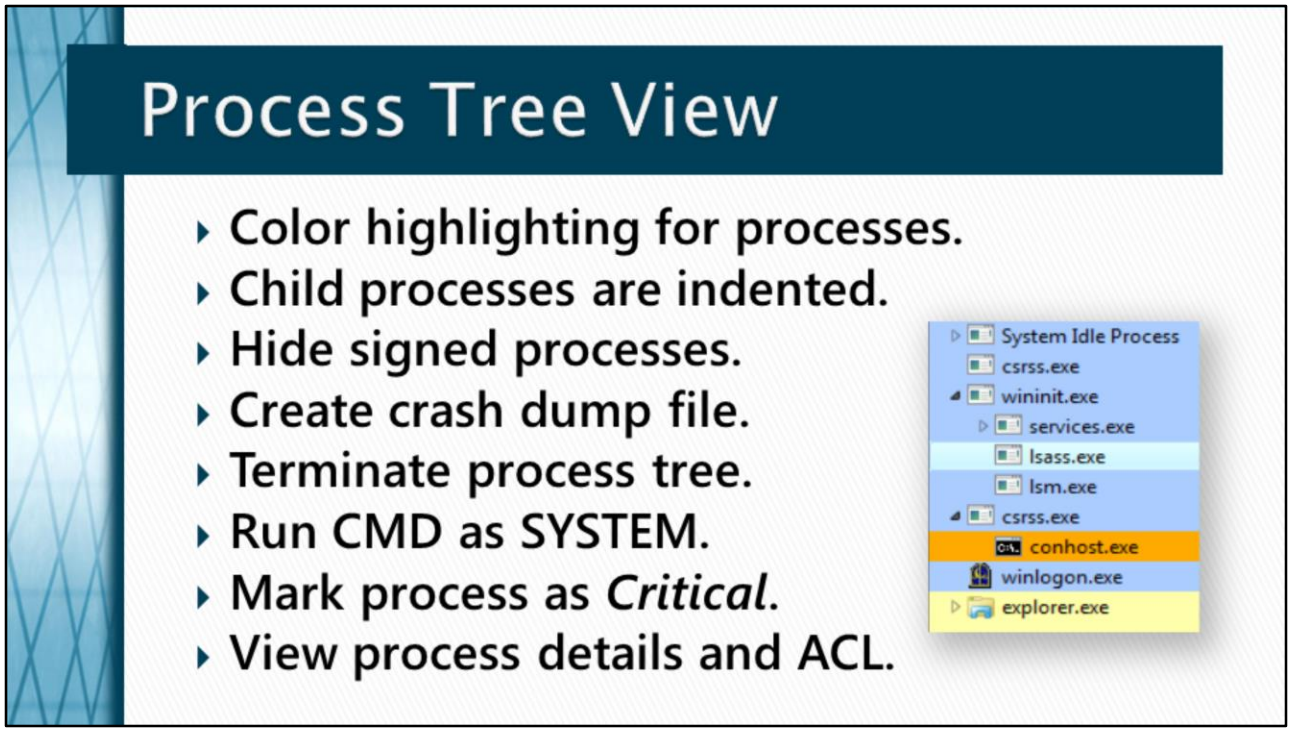
SRV*C:\Symbols*https://msdl.microsoft.com/download/symbols

To manage plugins, pull down the Hacker menu > Plugins > right-click a plugin > enable/disable or access its properties.

In almost every window with a table, you can right-click on the column header > Choose Columns. There are *many* more columns of information available than is displayed by default.

Process Hacker has a few command-line options (run it with -help to see the list). Using the -debug switch on a debug build will show a special command shell just for Process Hacker (run "help" inside the shell). On a non-debug build, run "procitem" in the shell for grins.

Finally, the people who maintain Process Hacker are volunteers. so please consider clicking on the Help menu > Donate.



Process Tree View

A "process" is an environment in which a thread can run, providing a virtual memory address space, handles to objects, a security context, and other resources. A running process can spawn new processes, and thus there can exist parent-child relationships among processes. These parent-child relationships are shown as indentations in the "tree view" of Process Hacker. Every process contains at least one thread, and may contain many. A "thread" is that which has at least one stack and actually performs computing work within a process. A "job" is a collection of one or more processes, though a given process might not be a member of a job (look for a Job tab). A job allows a group of processes to be managed as a unit, such as placing restrictions on what those processes can do.

In Process Hacker, you can perform many tasks related to processes, including:

- Show different types of processes in different colors (Hacker menu > Options > Highlighting).
- Run commands as SYSTEM on your desktop, such as CMD.EXE (Hacker menu > Run As).
- Create a new driver or interactive service (Tools menu > Create Service).

- Hide processes with verified digital signatures (View menu > Hide Signed Processes).

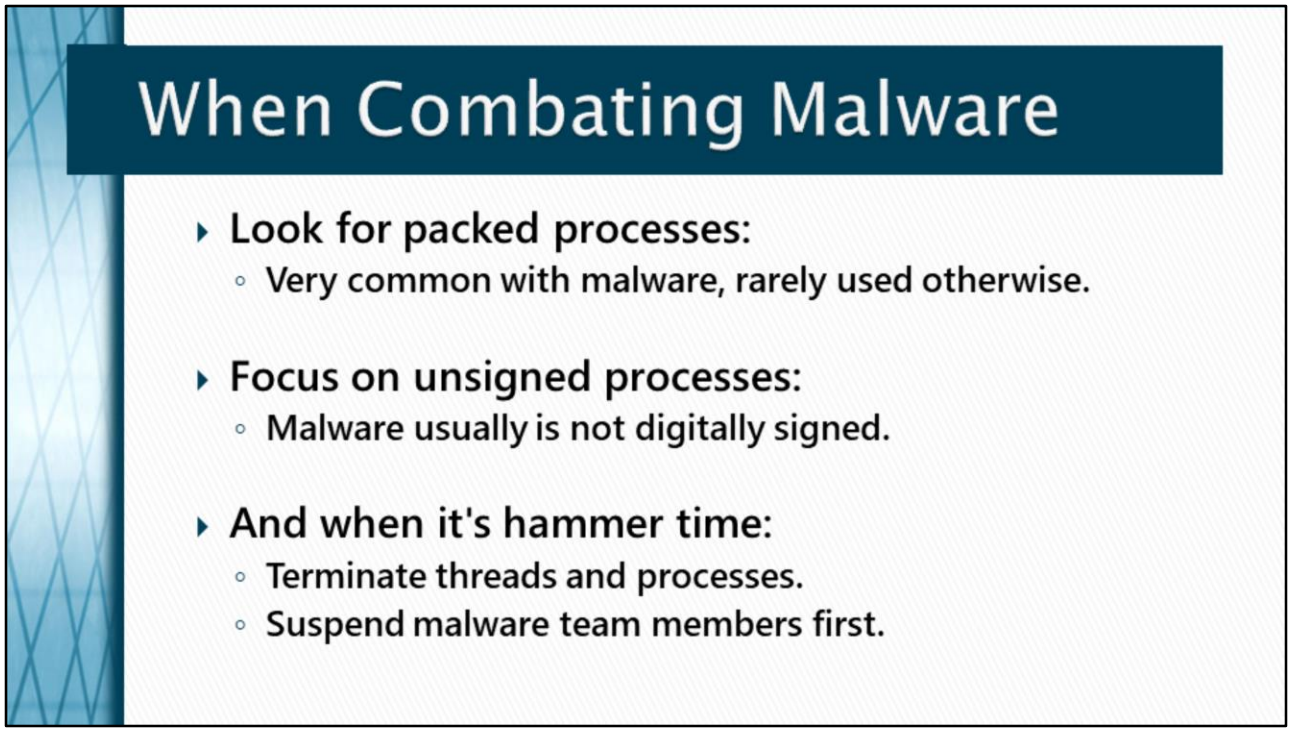
- Create a crash dump file of a process for analysis, but doesn't crash the process (right-click process > Create Dump File).

- Terminate almost any process, even if protected by some rootkits (right-click process > Terminate or Terminate Tree).

- Mark a process as "Critical" so that, if the process terminates, the entire OS hangs and reboots (right-click process > Miscellaneous > Critical).

- Inject a DLL or EXE into the memory of a running process (double-click process > Modules tab > Options button > Load Module).

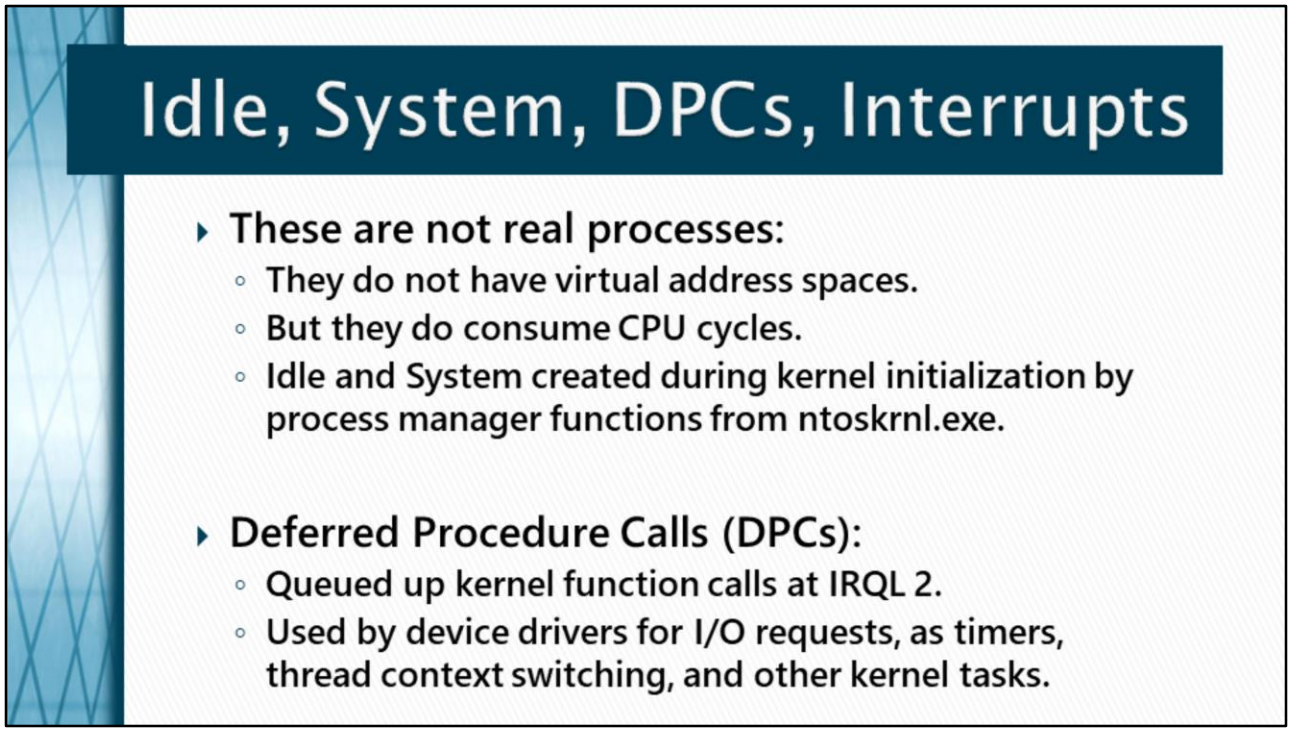
- On the General tab in the properties of a process, view the command-line arguments used to launch the process, the permissions on the process object in the kernel (Permissions button), security mitigation policies (Details button), and numerous details of the EXE image used to create the process, such as its PE header and any NTFS alternate data streams (click the magnifying glass button).



When Combating Malware

It is common for malware executables to be packed. A "packed" process is compressed and/or encrypted, perhaps multiple times, along with a tiny bit of decompression/decryption code which is executed first in order to reconstruct (in memory, not on the hard drive) the working form of the malware. Some legitimate programs use packing to thwart reverse engineering, but this rare. Hence, enable highlighting of packed processes and focus on them first when combating malware. To scan the file system for packed binaries, use free tools like Mandiant's Red Curtain (www.mandiant.com).

Malware binaries are rarely digitally signed, so add the "Verification Status" and "Verified Signer" columns, then hide signed processes (View menu) to quickly identify those processes which lack a verified signature from a trusted code-signing certificate. Keep in mind, however, that it is still very common for legitimate programs to not have signatures. Also, a driver for the Stuxnet worm was digitally signed by a trusted issuer, so the presence of a signature does not automatically rule out that binary as being malicious.



Idle, System, DPCs and Interrupts

System Idle Process, System, Deferred Procedure Calls (DPCs) and Interrupts are not actually full processes in that they do not have virtual address spaces of their own, but they do consume CPU cycles. (If you don't see DPCs on a separate line in Process Hacker > Hacker menu > Options > General > uncheck "Enable cycle-based CPU usage".)

Idle Threads

There is one idle thread per CPU core. Idle threads aren't actually idle, they perform useful functions such as enabling/disabling interrupts, dispatching software interrupts for DPCs, dispatching some of the other threads waiting for a CPU, and granting kernel debuggers access to the OS. Idle threads run in a continuous loop as they check for work to perform. Any other thread can preempt an idle thread.

During kernel initialization, the first quasi-process created is the Idle "process" (it's not a real full-blown process) with PID = 0. Kernel bootstrapping proceeds later to create the System "process" (also not a full real process) and an Idle thread dispatches the first System thread. Idle is shown as the parent of the System process, but this is a fictional artifact of the kernel initialization sequence (winload.exe copies ntoskrnl.exe into memory and hands control to it, ntoskrnl.exe contains the functions collectively called the "Process Manager", and it's the Process Manager which creates

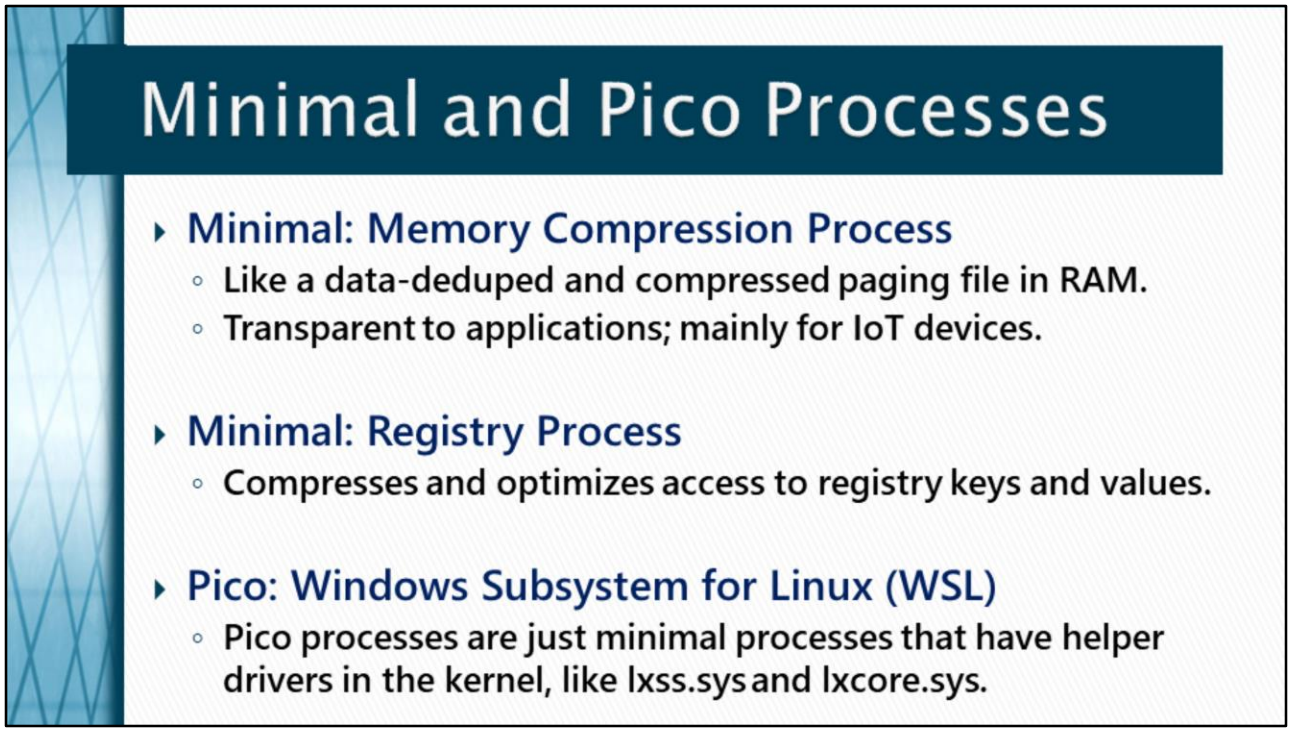
both Idle and System).

Deferred Procedure Calls (DPCs)
Deferred Procedure Calls (DPCs) are requests for kernel functions to be executed on behalf of device drivers and other kernel components. Each CPU core has a queue of DPCs which are not executed immediately (they are "deferred") but only after the CPU has finished its current higher-priority task. Device drivers use their Interrupt Service Routines (ISRs) to create DPCs to manage I/O requests. Kernel components use DPCs to handle things like timer expirations and performing context switches from one thread to another. DPCs execute at IRQ Level 2, so a DPC can interrupt any normal thread (which run at IRQL 0), but another hardware interrupt (IRQL 3-15) can interrupt a DPC.

IRQ Levels, Thread Priorities, and Interrupts
Hal.dll imposes an IRQ Level scheme: IRQL 0 for user-mode threads ("passive", "low" or "normal" priority), IRQL 1 for Asynchronous Procedure Calls (APCs), IRQL 2 for DPCs ("dispatch" priority), IRQL 3+ for hardware interrupts. Only non-paged memory can be accessed at IRQL 2 or higher because page faults themselves are handled at IRQL 2. Threads have their own priorities (0-31), but a higher IRQL thread can always preempt a lower-IRQL thread, no matter what their relative thread priorities are. Thread priorities can be viewed and modified in Process Hacker (properties of a process > Threads tab > right-click a thread).

In Process Hacker, the Interrupts includes the CPU time spent processing hardware and software Interrupt Service Routines (ISRs). Often, an ISR will simply create a DPC to do the real work, but at a lower IRQL. CPU utilization for Interrupts should be low on average.

Registry and Memory Compression
The "Registry" and "Memory Compression" processes were added in Windows 10 to optimize memory usage. Microsoft calls these "minimal" processes, which are akin to the "pico" processes used by the Windows Subsystem for Linux (WSL). A minimal process is not much more than a virtual address space with thread support, with most of the Windows subsystem components having been stripped out. A minimal process, for example, does not have a Process Environment Block (PEB) to describe it. The Registry minimal process loads registry hives as modules, as well as the Boot Configuration Database (BCD) and some other data files too. The Memory Compression process handles data deduplication of memory pages, which are then also data-deduplicated and compressed. In effect, it is a compressed paging file, but in RAM. And, just like an on-disk paging file, this compression and decompression of memory pages is completely invisible to applications.



Minimal and Pico Processes
A "minimal" process is not much more than just a virtual address space. A minimal process can have threads, but there is no default or mandatory thread, and it does not have a Process Environment Block (PEB) to describe the process to the kernel. Most of the Windows subsystem components are stripped out too, such as user32.dll and kernel32.dll, so minimal processes cannot run standard Windows programs. A minimal process is like a blank slate in user mode that the OS may fill with other stuff as Microsoft sees fit.

A "pico" process is just a minimal process that has one or more helper drivers in the kernel. These drivers support or interact with the pico processes to enhance them in some way. What ways? Any ways that Microsoft dreams up. There can be different types of pico processes associated with different helper drivers, hence, it's a general framework or capability that is very similar to a subsystem (like the Windows or POSIX subsystem), but more generalizable.

Minimal: Memory Compression and Registry Processes
The "Registry" and "Memory Compression" processes were added in Windows 10 to optimize memory usage and performance. The Memory Compression minimal process handles data deduplication and compression of memory pages. In effect, it is a compressed paging file *in RAM*. And, just like an on-disk paging file, this compression and decompression of memory pages is completely invisible to

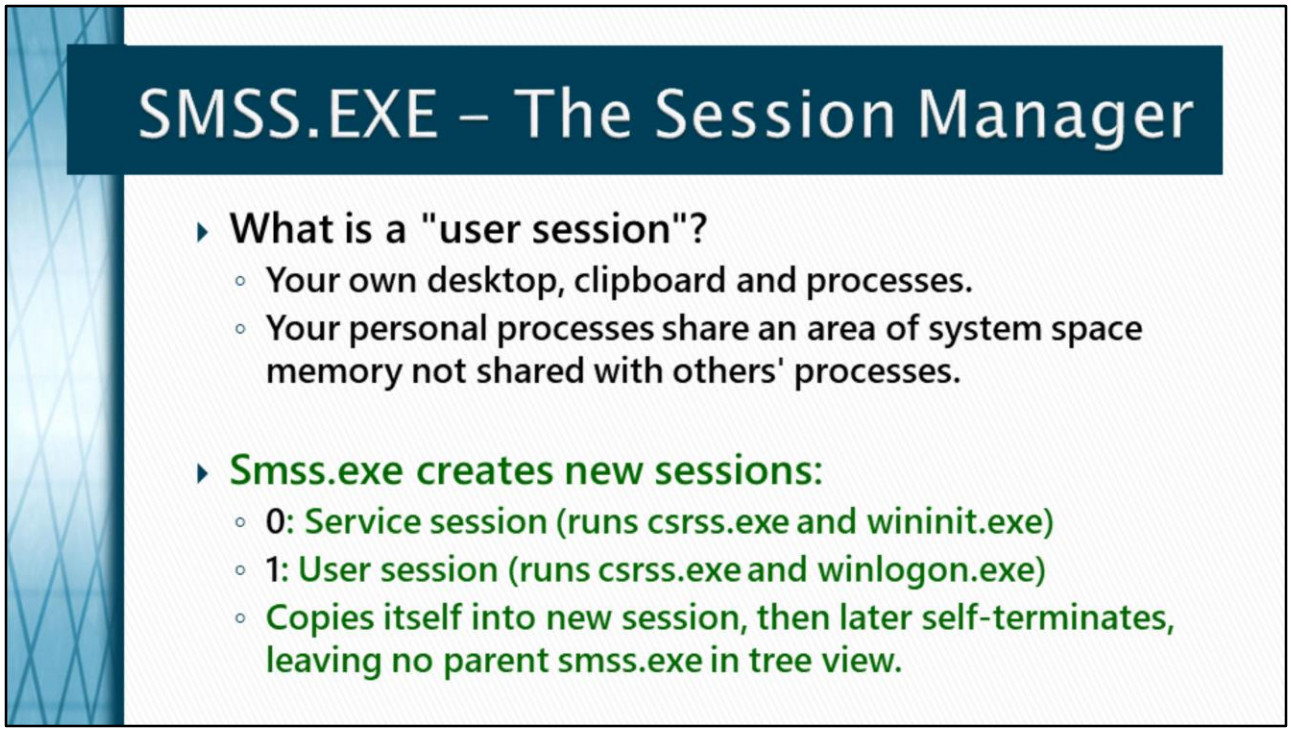
applications. SSDs are slow; RAM is fast. On machines with plenty of RAM, little or no compression is done, but RAM is scarce on IoT or other low-end devices.

The Registry minimal process loads registry hives as modules, as well as the Boot Configuration Database (BCD) and some other data files too. Registry access occurs very frequently, so a special in-memory cache optimizes performance and avoids unnecessary disk I/O.

Pico: Windows Subsystem for Linux (WSL)

Since Microsoft can cram anything inside a minimal process, why not Linux binaries? When Ubuntu bash runs in a minimal process, for example, it expects to be able to make hundreds of Linux-flavored system calls, hence, Microsoft has added drivers in the kernel (lxss.sys and lxcore.sys) to catch these syscalls and translate them into native Windows kernel function calls. This is translation is invisible to the Linux binaries even though there is not a real Linux kernel underneath (there are limitations though). When you run bash.exe, this contacts the LXSS Manager service, which runs a Linux-ish init as a pico process, which then runs the real compiled-for-Linux ELF64 bash as another pico process. When the last WSL pico process terminates, the LXSS Manager service kills init and cleans up. For disk I/O, WSL implements the VOLFS and DRIVEFS file systems, which cannot be discussed here.

What else could use minimal or pico processes in the future? Who knows, maybe Google Play store Android apps…(hint hint).



SMSS.EXE - The Session Manager

The Session Manager (smss.exe) is the first user-mode process created after kernel initialization. It is responsible for loading the rest of the registry, building the list of environment variables to be inherited by other processes, performing post-boot delayed file delete/rename changes, opening additional swap files, creating symbolic links for DOS device names like LPT1 and COM1, loading known DLLs into shared memory sections, and other tasks. More importantly, it also launches csrss.exe, wininit.exe and winlogon.exe.

Session 0

The first session created by smss.exe is for operating system services (session 0), not for human users. When session 0 is created, smss.exe copies itself into that new session, starts csrss.exe and wininit.exe (plus other tasks), then smss.exe exits (the original instance under System still runs). This is why in tree view you see csrss.exe and wininit.exe in session 0 without any parent processes, that is to say, why the session 0 csrss.exe and wininit.exe processes are left-aligned without any indentation. New sessions are created by smss.exe by its calling certain functions in the kernel's memory manager.

Session 1

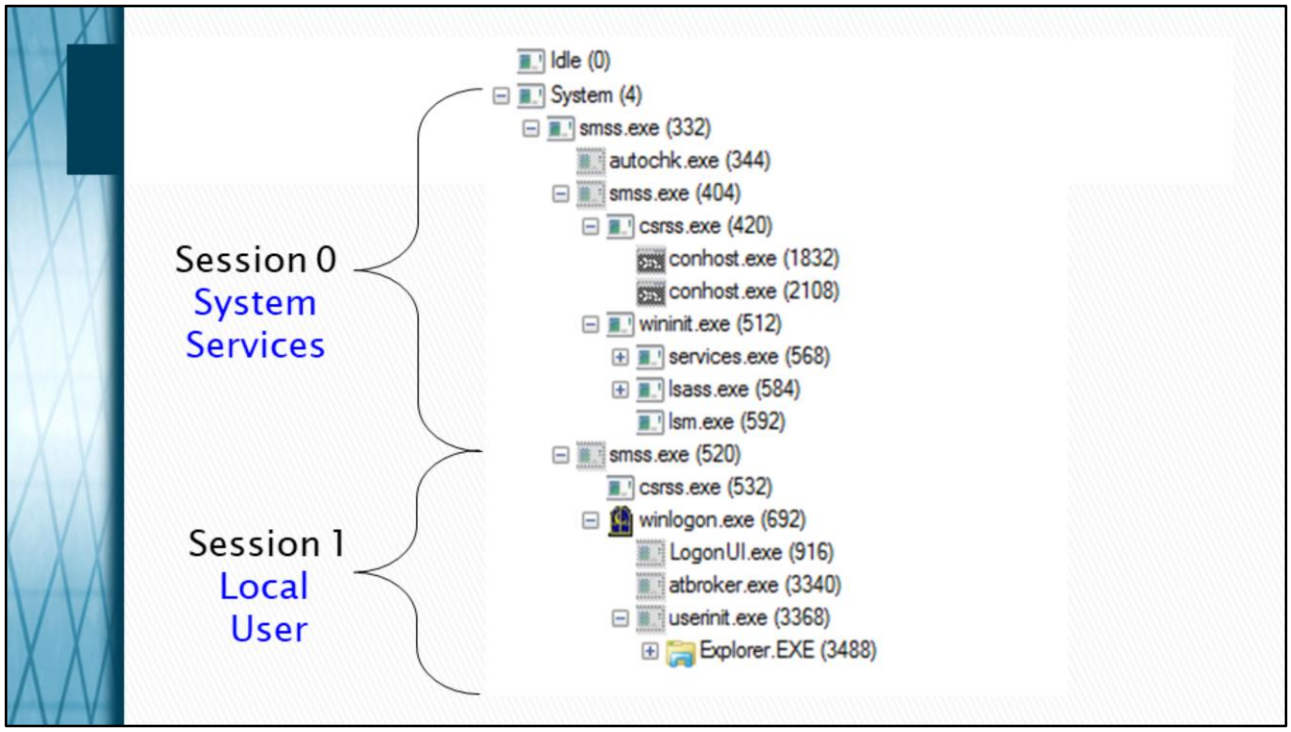
When a new user session is created (session 1 and greater), smss.exe copies itself

into that new session, starts csrss.exe and winlogon.exe in that session (plus other tasks), then the smss.exe in the new session exits. This is why csrss.exe and winlogon.exe in session 1 are shown in the process tree without any parents. Remember, wininit.exe for session 0, winlogon.exe for user sessions greater than 0.

What Is A "Session" Anyway?
Loosely speaking, your session is all of the processes you own, plus your desktop.

Strictly speaking, a session is a small area of kernel address space which is not mapped into the virtual address spaces of all processes, but only into some, namely, into the processes originally created for the OS (session 0) or a user (session 1 or greater) by smss.exe and the child processes of these parents. This area of kernel memory is called "session space" and is shared by all the processes in the same session. Each session contains its own copy of win32k.sys and associated video drivers, a per-session directory in the object manager's namespace (\Sessions*), and a per-session paged pool of memory. The win32k.sys driver is that part of the Windows subsystem which manages the output of GUI windows on the desktop (USER, GDI and DirectX), gathers input from the keyboard and mouse, and sends input messages from the mouse or keyboard to the correct process.

The processes in a session share the same station and desktop(s).

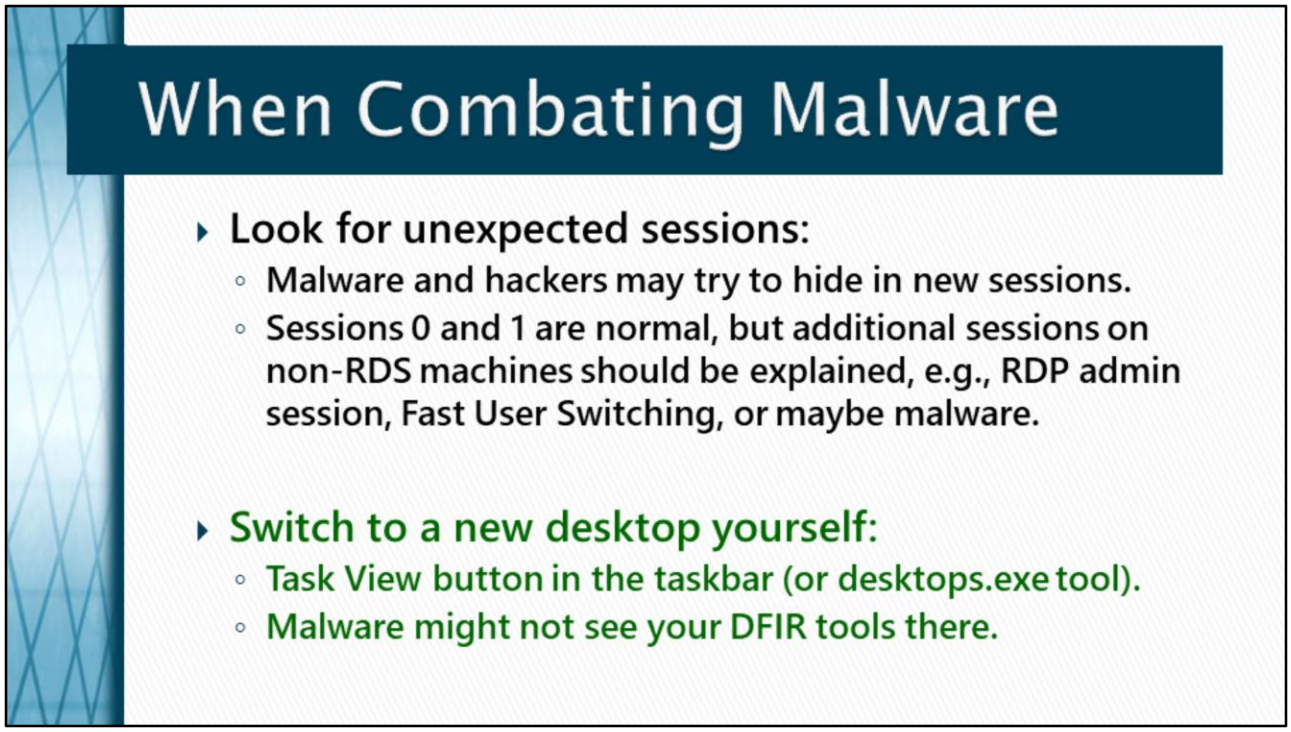


Sysinternals Process Monitor Tree

The screenshot above was made with Sysinternals Process Monitor, not Process Hacker, after configuring that tool for boot logging and then rebooting the computer (http://www.microsoft.com/sysinternals). The screenshot shows parent processes even if those processes are no longer running at the time the screenshot was made. It is an historical snapshot of parent-child process relationships. The processes no longer running when the screenshot was made are shown with greyed-out icons. The numbers in parentheses are their Process ID numbers (PIDs). The screenshot is from a computer running 64-bit Windows 7 Ultimate.

To see this yourself, in Process Monitor pull down the Options menu > Enable Boot Logging. Reboot the computer. Open Process Monitor again, and when prompted, save the boot log to a file. Then pull down the File menu > Open > load the boot log > Tools menu > Process Tree.

The screenshot shows how smss.exe creates session 0 for csrss.exe and wininit.exe, then smss.exe exits. In session 0, wininit.exe then runs services.exe, lsass.exe and lsm.exe. For session 1, smss.exe runs csrss.exe and winlogon.exe, then winlogon.exe spawns logonui.exe and userinit.exe



When Combating Malware

Sessions 0 and 1 are normal, but additional sessions should be explained. Additional sessions can come from RDP connections, remote Windows Media Center sessions with an extender (like an Xbox), and Fast User Switching on shared computers, but beyond these examples any unexpected sessions should be investigated. Because window stations and desktop objects have Access Control Lists (ACLs), it's possible that a rootkit might try to harden these ACLs and then hide the entire station.

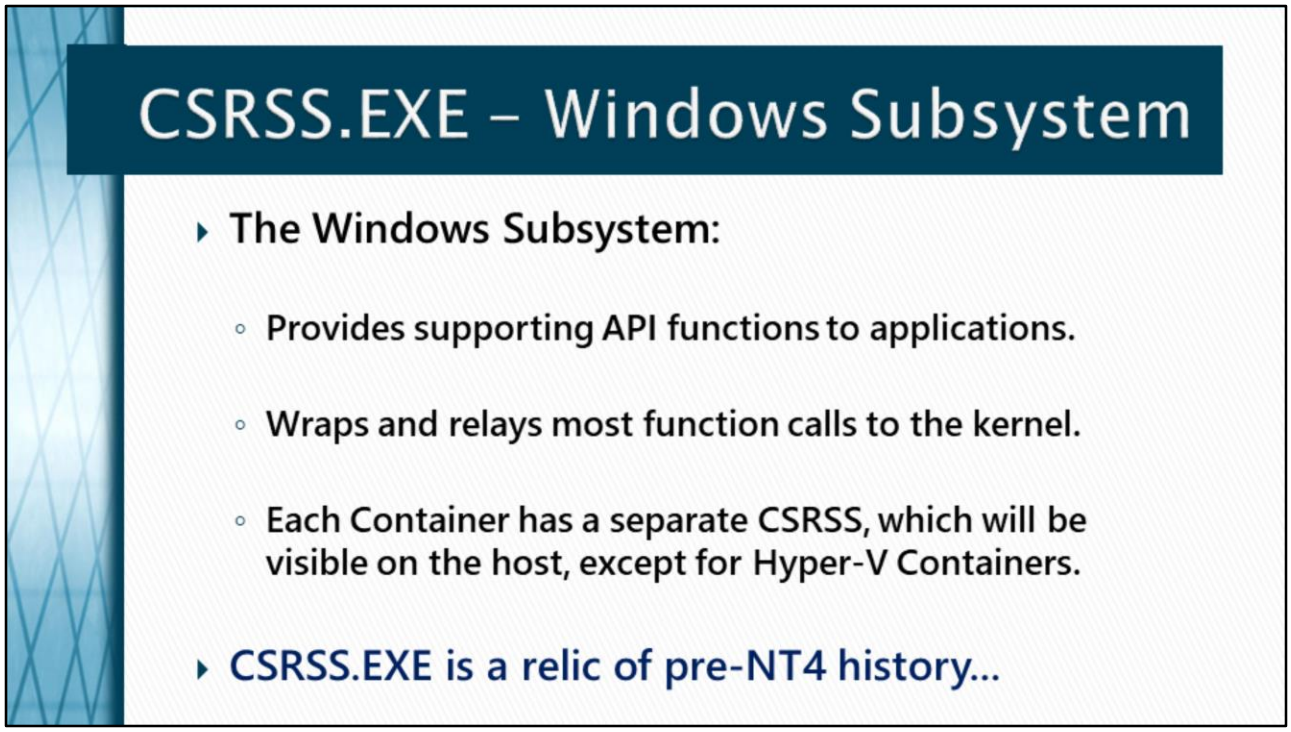
A window station (or "winstation") contains a clipboard and one or more desktops. Each window station is a securable object, hence, each has an access control list. When a window station is created, it is associated with the calling process and assigned to the session of that process. The interactive window station, named "Winsta0", is the only window station which can display a graphical user interface and receive user input through one of its desktops. Each RDP remote desktop session has its own Winsta0.

A window station can have one or more desktop objects associated with it. The desktops associated with the interactive window station, Winsta0, can be made to display a user interface and receive user input, but only one of these desktops can be seen by the user at a time (this is called the "active desktop"). Windows messages can be sent only between processes that are on the same desktop. Each process

which depends on the Windows subsystem is associated with a desktop, station and session (smss.exe and csrss.exe help implement the Windows subsystem, so they do not depend on it of course, so wininit.exe is the first).

Use the Windows 10 "virtual desktops" feature to play around with multiple desktops in your interactive window station (on Windows 7, get the Sysinternals desktops.exe tool). Right-click your taskbar and ensure that "Show Task View Button" is enabled. Click this button in your taskbar and then click "New Desktop +" or choose an existing desktop. Each desktop can have its own taskbar, program windows and "desktop" as we normally use that word in everyday conversation. Only one desktop can be active at any given time, but because they all share the same window station, there is one shared clipboard. If you initiate logoff in one of the desktops, you'll log off from all of them because you are really exiting your entire session.

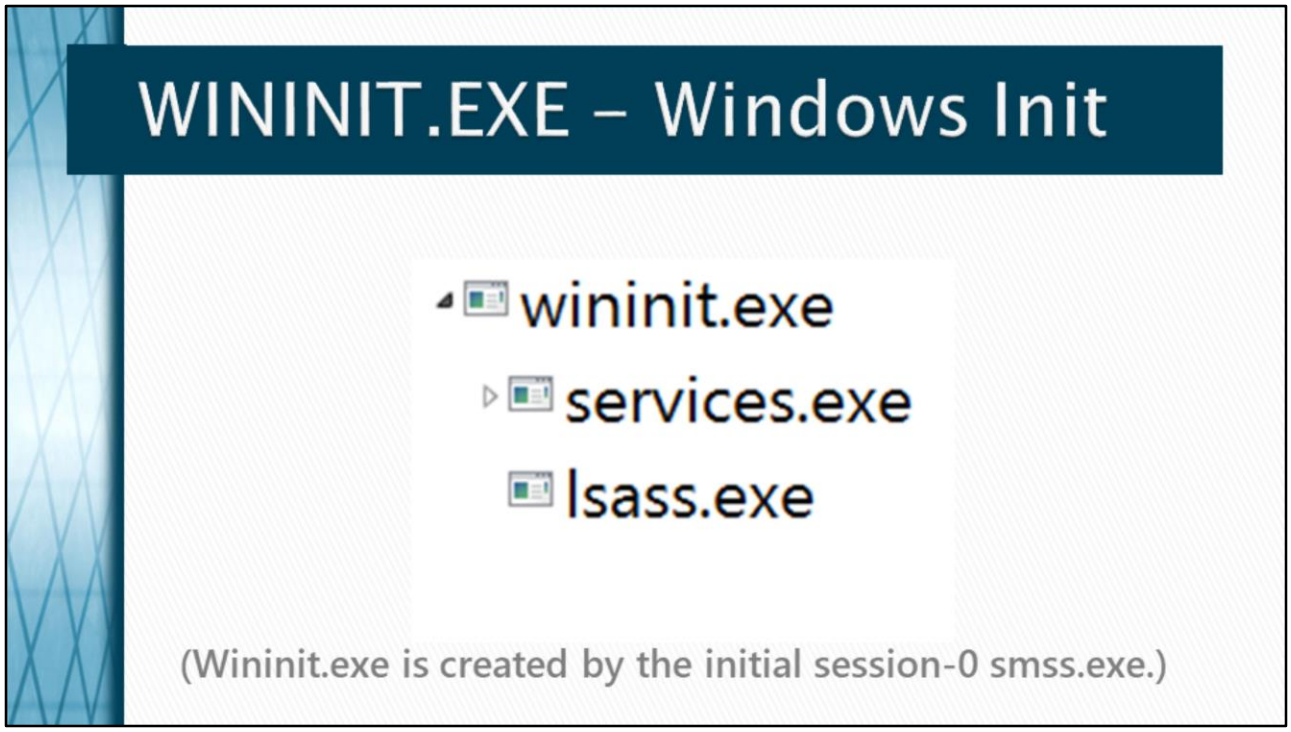
If you are combating malware that is blocking your forensics tools, you might try switching to another session or desktop in order to try to hide *from it*.



CSRSS.EXE - The Windows Subsystem Process

A "subsystem" provides part of the environment in which processes can run. A subsystem makes various operating system functions available for processes to call as needed, such as for accessing the file system, using the network, or drawing graphical windows. The Windows subsystem loads DLLs into the address space of a process which can provide these functions themselves or relay these function calls down to the kernel or to other supporting processes. There are many DLLs that implement the Windows subsystem, including kernel32.dll, advapi32.dll, user32.dll and gdi32.dll.

The Windows subsystem also includes csrss.exe as a helper process. Prior to Windows NT 4.0, csrss.exe was very important because it handled drawing graphical windows, but since then this functionality has been moved into the win32k.sys kernel-mode driver for performance reasons. Drawing text-based consoles used to be handled by csrss.exe too, but with Windows 7 and Server 2008-R2, even this functionality has been moved into a different process (conhost.exe) for security reasons. Spawning new processes and threads, mapping drive letters, handling much of the shutdown procedure, loading win32k.sys into the kernel at boot-up, and creating temp files is still managed by csrss.exe, but not much else of importance anymore, so csrss.exe is mainly an historical relic maintained for backwards compatibility. A future Windows version may eliminate it entirely.



WININIT.EXE - The Windows Initialization Process

After the kernel is initialized and smss.exe is running, smss.exe creates session 0, copies itself into session 0, launches csrss.exe, launches wininit.exe, then that session-0 copy of smss.exe exits. At this point, csrss.exe and wininit.exe are still running in session 0.

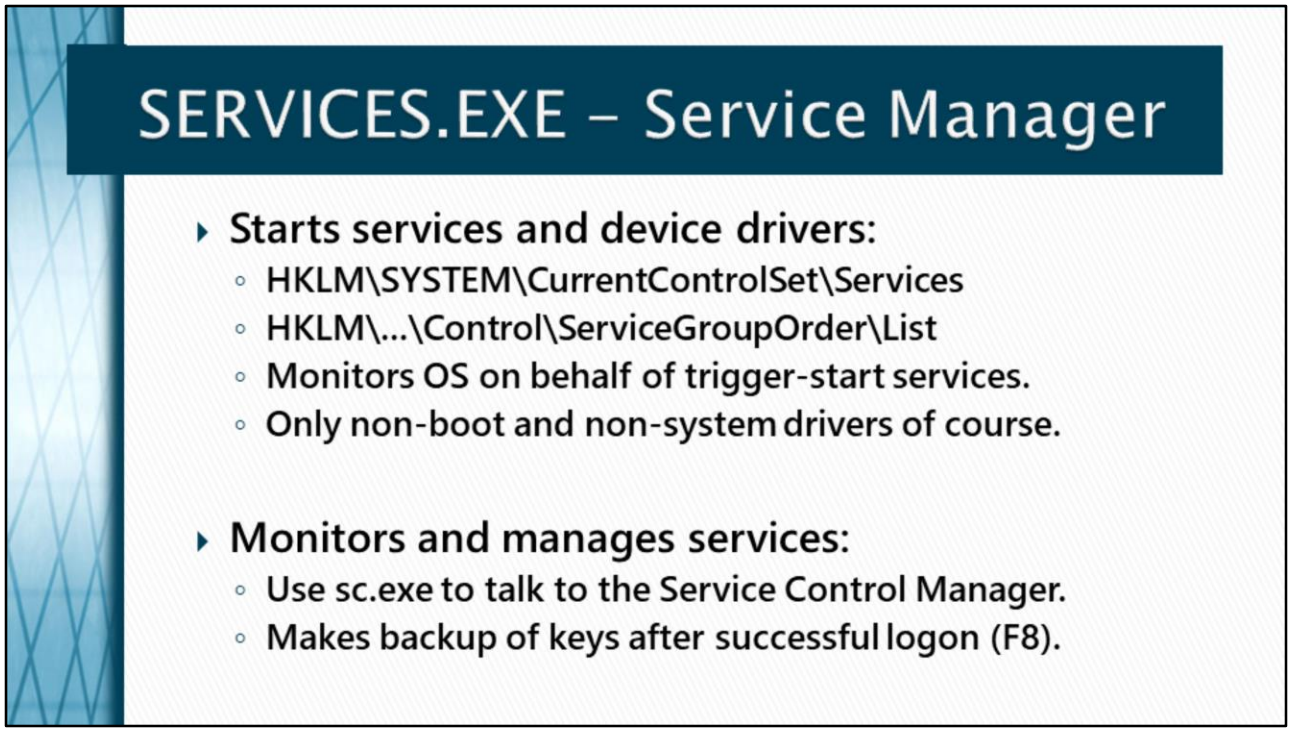
The main job of wininit.exe is to 1) launch services.exe to start services and more device drivers, 2) launch lsass.exe to handle security, and 3) launch lsm.exe or load lsm.dll as a service to do some of the session management. (Note: In Windows XP and Server 2003, lsass.exe was launched by winlogon.exe instead.)

LSM.EXE or LSM.DLL - The Local Session Manager

On Windows Vista and 7, the Local Session Manager (lsm.exe) creates, destroys and manipulates sessions, mainly by directing smss.exe to do so. On Windows 10 and later, this process was converted into a regular shared-svchost.exe DLL service (lsm.dll). There is session 0 for services and session 1 is typically for the interactive user logon, but additional sessions may exist for the sake of Fast User Switching, Windows Media Center Extenders, and Remote Desktop Protocol (RDP) connections, such as RDP connections to a Server 2008-R2 machine running Remote Desktop Services.

When an RDP user connects, the LSM directs smss.exe to create a new session (including csrss.exe, winlogon.exe and logonui.exe). If there is already a desktop for that user, then the RDP connection is linked to that desktop and its currently-running processes in its current session. If a new desktop is required, then userinit.exe and explorer.exe run like normal, but in another new session. An exception to this is when an RDP connection is made for Remote Assistance, in which case the RDP connection goes to the Remote Assistance Server (raserver.exe) and a new session is not created by smss.exe; instead, the RDP stream is simply linked to the desktop of the user being helped, either for passive viewing or for shared control of the mouse and keyboard.

LSM.EXE never has any child processes of its own, and neither does the hosting process for LSM.DLL. Because multiple smss.exe processes can be spawned concurrently, it improves the performance of servers running Remote Desktop Services when responding to many simultaneous logon requests.



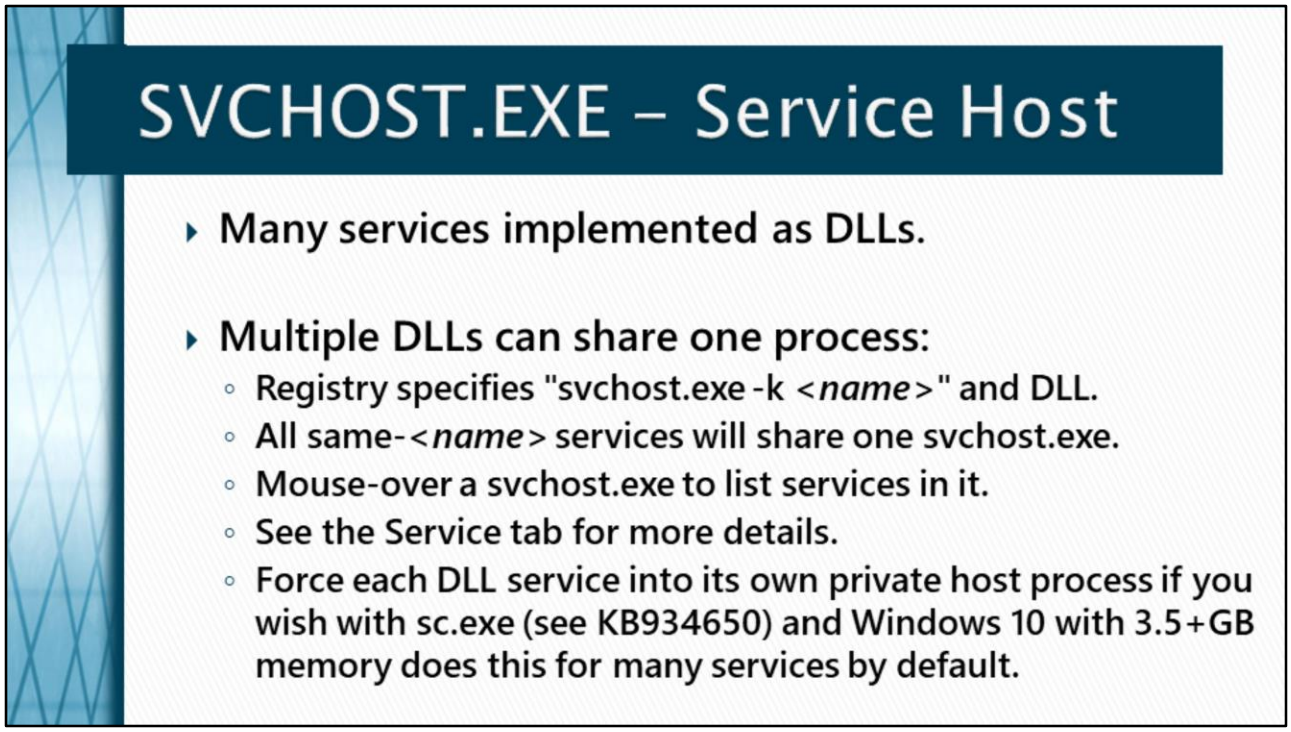
SERVICES.EXE - The Service Control Manager

The job of services.exe is to read keys in the registry representing services and device drivers (under HKLM\SYSTEM\CurrentControlSet\Services), then run these processes and load these drivers according to their dependency groups (as defined in HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder\List). Boot and system drivers are not loaded at this time, they are already loaded. When a service does not have a "Group" value to identify its dependency group, it is run last.

By default, non-interactive services running as SYSTEM are attached to an invisible window station named "Service-0x0-3e7$" in session 0; when these services are run interactively, they still run in session 0, but are attached to the Winsta0 station instead.

Windows services are not just fire-and-forget processes: services.exe maintains an in-memory database of service information, such as error state and recovery actions, similar to systemd on Linux. Use the built-in sc.exe tool to interact with services.exe.

After a successful interactive user logon, services.exe will also make a backup copy of the keys under HKLM\SYSTEM\CurrentControlSet\Services known as the "last known good configuration", which can be restored at the next reboot with the F8 key.

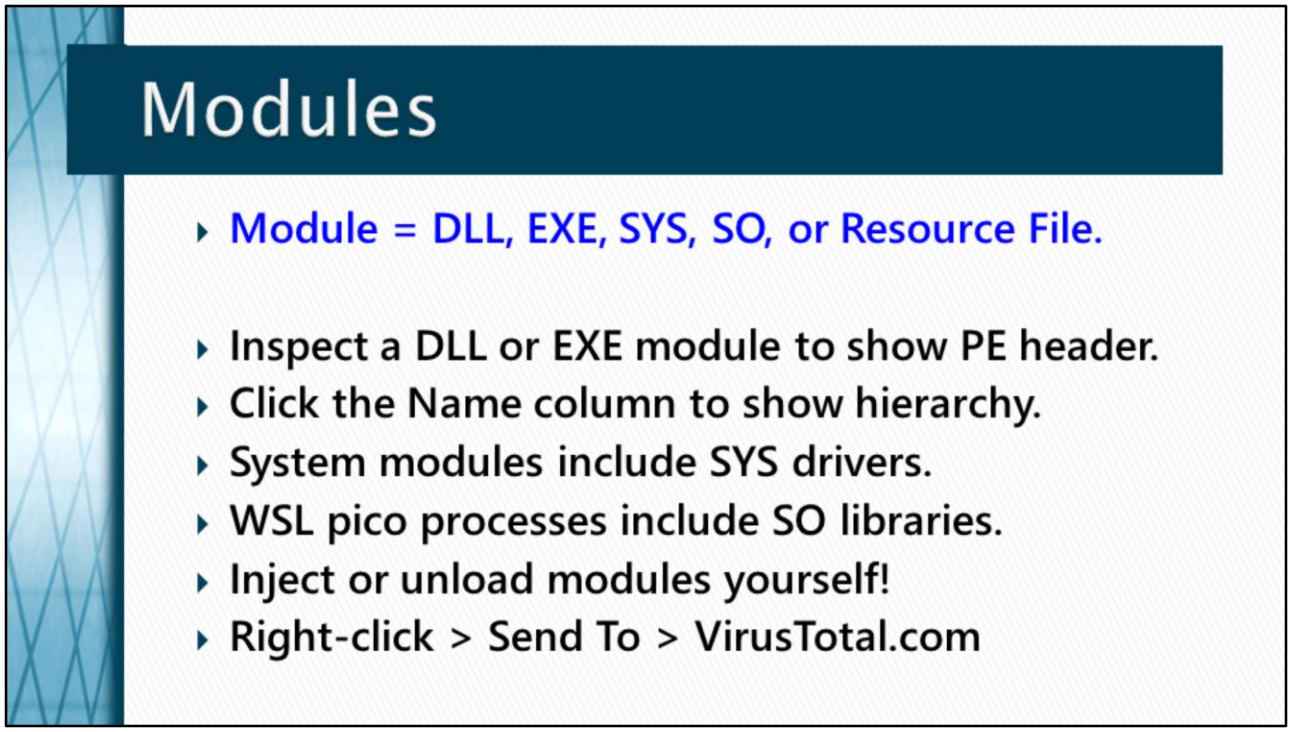


SVCHOST.EXE - The Generic Service Host Process

Many services are implemented as DLLs which can be loaded together into a generic shared svchost.exe process launched by services.exe. The registry entry for the service will include a ServiceDll value to identify the DLL to be used and a path to "svchost.exe -k *<name>*" to load it. All DLL-based services with the same "*<name>*" will share the same svchost.exe process; for example, both the Base Filtering Engine (bfe.dll) and Windows Defender Firewall (mpssvc.dll) services are DLL-based, and they share a single svchost.exe because both are launched with the "svchost.exe -k LocalServiceNoNetwork" command.

Hold your mouse pointer over a svchost.exe process to see a pop-up list of the services contained within it.

Sharing a single host process is nice for conserving OS resources, but not great for troubleshooting or analyzing infected services. You can force a DLL-based service into its own svchost.exe process with the sc.exe command (see KB251192 and KB934650). On Windows 10 systems with 3.5GB or more of memory, many services are broken out into their own svchost.exe processes by default.



Modules

The modules of a process are the Dynamic Linked Libraries (DLLs), resource files and the original executable image (EXE) loaded into the memory address space of that process. The modules loaded into the System "process" at the top includes drivers (*.sys) and ntoskrnl.exe. Remember that System is not really a process. And for Windows Subsystem for Linux (WSL) pico processes, like Ubuntu bash, the modules tab will show Shared Object (SO) library files.

In the Modules tab of a process, right-click the column headers, select Choose Columns, and add every possible column. Make sure to add the "Verification Status" and "Verified Signer" columns in particular, perhaps moving them near the top of the list. When combating malware, modules which fail their digital signature tests, or which have no signature, deserve additional analysis. You can add the same columns in the process tree view and then hide processes with verified signatures (View menu > Hide Signed Processes).

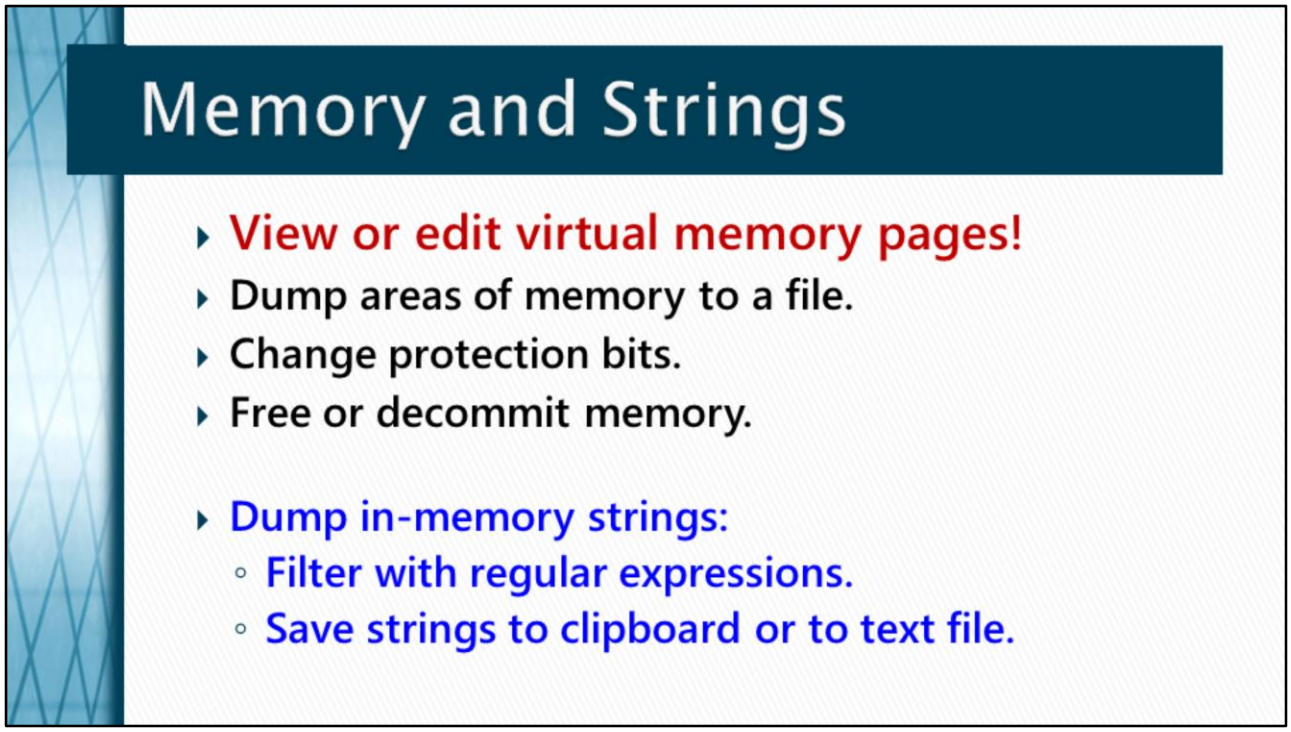
If you click the Name column repeatedly, it will show dependency relationships among the modules in a hierarchy. Add the Load Reason column header to help explain the hierarchy.

If you right-click a DLL or EXE module, you can select Inspect to show its Portable

Executable (PE) sections, ASLR compatibility (Dynamic Base), DEP compatibility (NX Support), the functions it calls from other modules (Imports tab), and the functions it makes available to other modules (Exports tab). You can also add ASLR and DEP as columns.

You can also unload a module when you right-click it (which often kills the process) or load a DLL or EXE into the process (Modules tab > Options button > Load Module). When you inject a DLL, the operating system runs the DllMain() function within it automatically.

Another great feature for fighting malware is the ability to right-click a binary on the Modules tab (or in the main process list) and Send To web sites which perform free AV scanning with many different scanners, e.g., VirusTotal.com. This is a really fast and convenient way to at least do basic checks on suspicious binaries, especially when they are not digitally signed and not found on other similar computers in your organization.



Memory and Strings

The 4GB virtual address space of an x86 32-bit process is typically divided into two regions: user space (0x00000000-0x7FFFFFFF) at the bottom and system space at the top (0x80000000-0xFFFFFFFF). The 16EB virtual address space of an x64 64-bit process is also divided into user space (0x0000000000000000-0x7FFFFFFFFFFFFFFF) and system space (0x8000000000000000-0xFFFFFFFFFFFFFFFF), though not all of this is addressable today due to software and hardware limitations. System space also includes a small area of session space for each SMSS-created session. The CPU must be in kernel mode in order to read/write system space pages.

Virtual-to-Physical Address Mapping

The user space is private to that process, except for shared memory sections; and the system space is common to all processes, except for a small area of system space called "session space", which is only common across processes in the same smss.exe session. An address in virtual memory can be common across processes because each committed page of virtual memory is backed by or "mapped to" a page of real physical memory, and two virtual memory addresses (like in two processes) can be mapped to the same physical address. Hence, a DLL can be loaded into physical memory once, then mapped into the system space of all processes using virtual addresses, saving a great deal of physical memory.

Kernel Mode vs. User Mode
The CPU can be in "kernel mode" or "user mode". The CPU must be in kernel mode in order to read/write memory in system space. The CPU can be in either kernel mode or user mode to read/write memory in user space. When in kernel mode, the CPU can execute any instruction it supports and can read/write any memory anywhere. Malware running in kernel mode can do anything. The CPU and OS tightly control the switching of the CPU's mode in order to try to maintain security.

Reserved vs. Committed Memory
A process can "reserve" an area of virtual memory, which means those addresses have been set aside for possible future use, or the process can "commit" an area of memory, which means those virtual addresses have actually been mapped to physical memory. Reserved virtual memory does not consume any physical memory, but committed virtual memory does consume physical memory.

Free, Mapped, Image or Private Memory
"Free" virtual memory is neither reserved nor committed, but can be so as needed. "Mapped" memory is either part of a heap, a section of memory shareable with other processes, or all/part of the contents of a file. "Image" memory contains the contents of file, usually a binary, cached in memory. "Private" memory is for the private use of the process, such as for a thread stack. (See the Sysinternals vmmap.exe and rammap.exe tools for a better view of memory use.)

Protection Bits
The page tables in system space, which do all the virtual-to-physical address mappings, also contain per-page protection bits, similar to permissions. An area of memory can be marked as read-only (R), write (W), execute (X), some combination of these, plus a few other markings used internally for housekeeping. Finding mapped, committed memory marked as RWX could be interesting, especially if a dump of the region began with "MZ" and "This program cannot be run in DOS mode" (Stuxnet did this, for example).

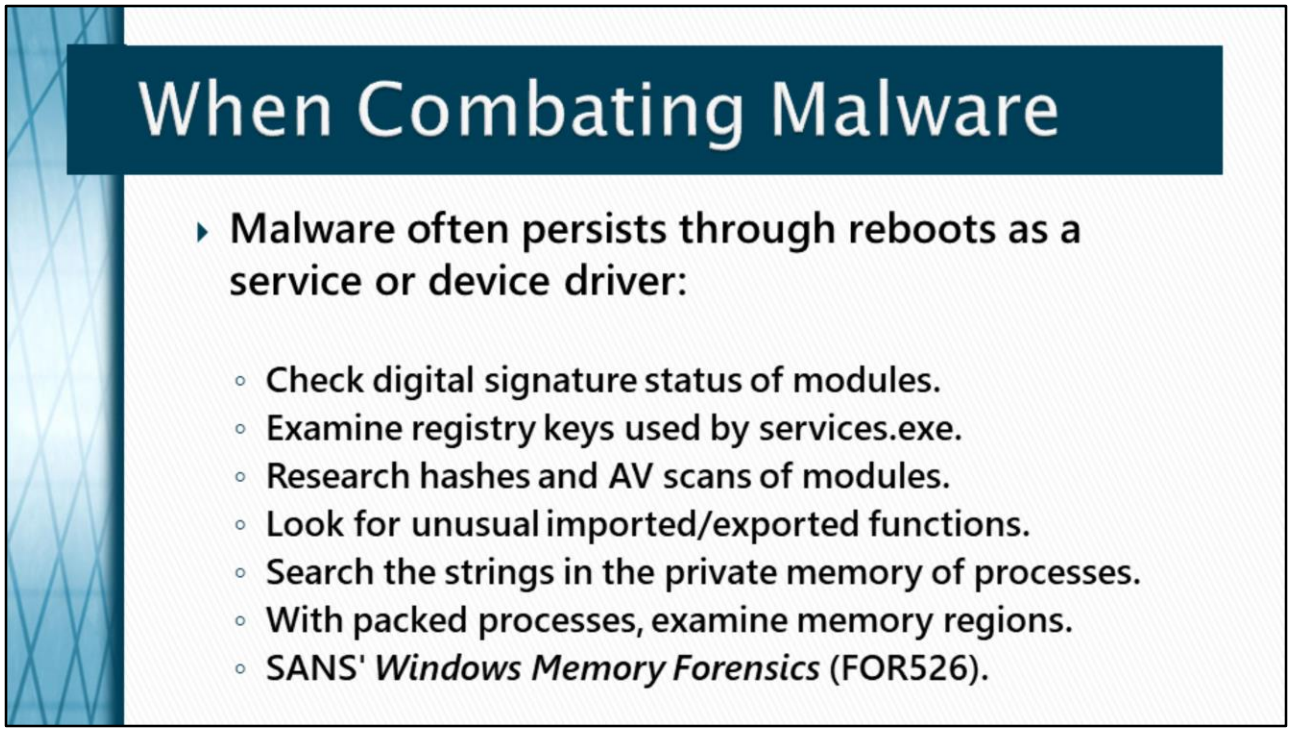
Memory Tab in Process Hacker
The Memory tab in Process Hacker permits several useful actions:

- View or edit areas of virtual memory (right-click > Read/Write Memory).
- Save an area of memory to a binary dump file (right-click > Save).
- Change the protection bits, with limitations (right-click > Change Protection).
- Free or decommit memory (right-click > Free or Decommit). Decommitted

memory is still reserved.

- Dump the ASCII and/or Unicode strings found in the process virtual address space, then filter these strings using a regular expression pattern (Strings button > OK > Filter button). Strings can be copied to the clipboard or saved as a TXT file.

For combating malware, the ability to dump and search strings from in-memory data is very important. The strings of a process are helpful in identifying malware types and analyzing their purposes and capabilities. If you wish to script the analysis of strings from the command line, get the Sysinternals strings.exe tool.

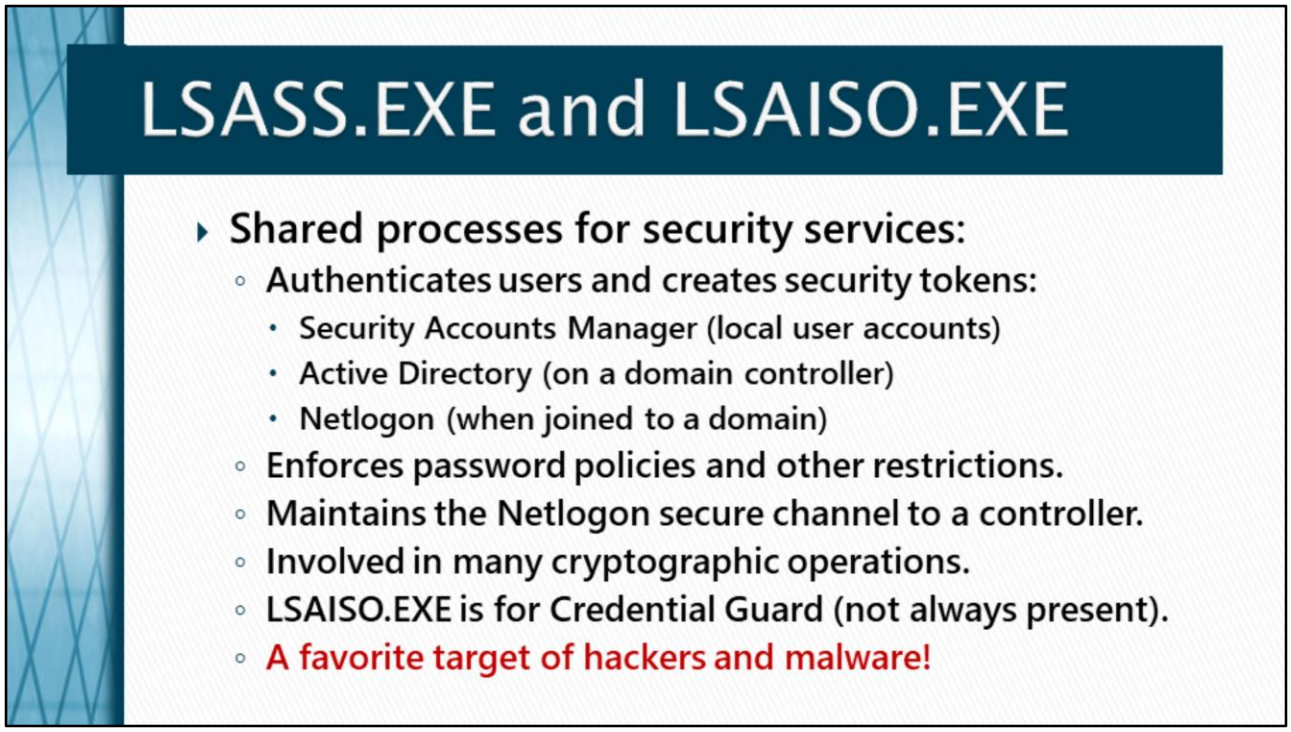


When Combating Malware

How can malware survive reboots? A common way to survive reboots is for the malware to get loaded by or as a service or device driver. By modifying an existing binary which is already loaded by services.exe, the malware can survive reboots without changing the registry. Or by adding a new binary and editing the registry, the malware can be run by services.exe automatically.

So, when combating malware, pay special attention to the child processes of services.exe and the associated registry keys. In particular, it's common for malware to load a DLL into a svchost.exe because there are so many of these processes. An infected machine might also have a malicious device driver visible in the Modules tab of the System process.

Make sure to check the digital signature status of modules and device drivers, confirm that the hashes of these binaries are known to be good, examine the imported and exported functions of suspicious modules, examine the strings of a process (look for strange URLs, file system paths, embedded passwords, or other "hinky" data), and with packed processes in particular you may need to examine raw memory regions. If these tasks seem difficult, because they are, you might consider taking *Reverse Engineering Malware* (FOR610), *Windows Memory Forensics* (FOR526) or *Advanced Forensics* (FOR508).

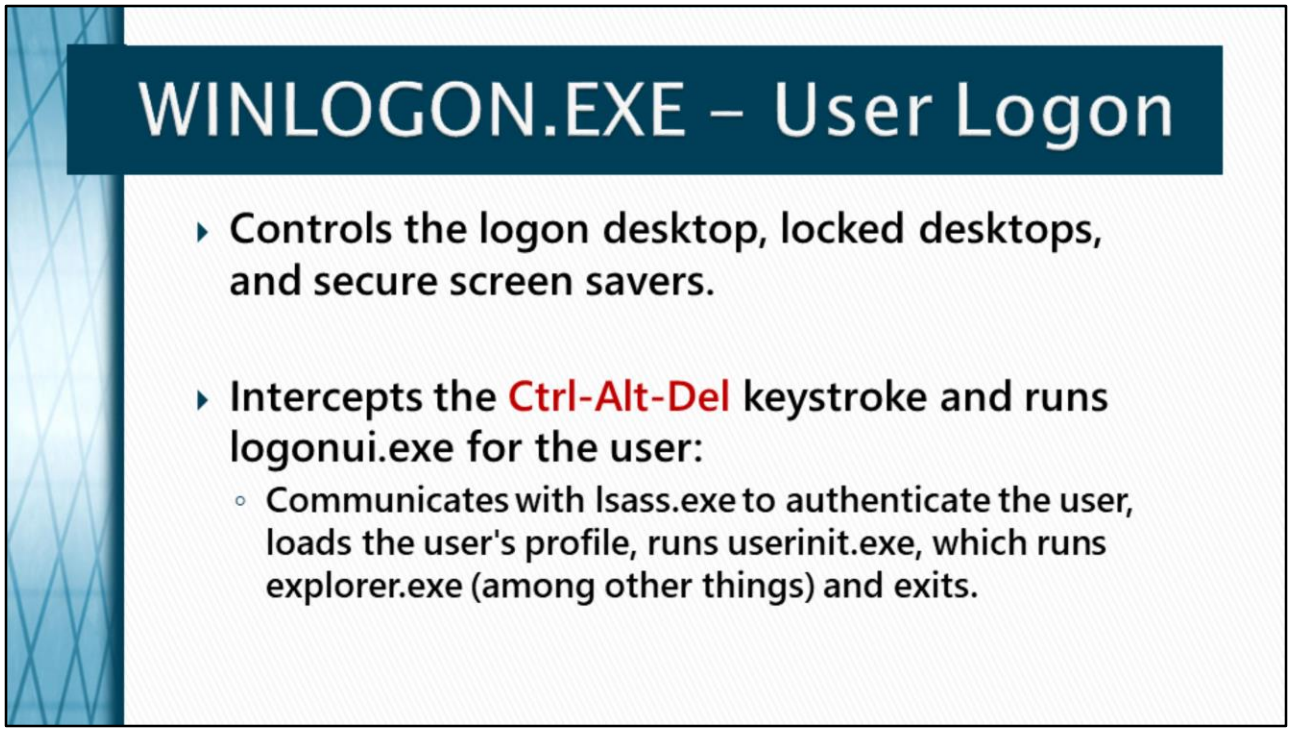


LSASS.EXE and LSAISO.EXE - The Local Security Authority Subsystem

Another process launched by wininit.exe is lsass.exe, which is a special shared-host process for security-related services like Active Directory on a domain controller (ntdsa.dll, kdcsvc.dll), Netlogon (netlogon.dll), Security Accounts Manager for local accounts (samsrv.dll), and Crypto Next Generation Key Isolation (keyiso.dll).

Many critical operations are handled by lsass.exe, such as user authentication, enforcing password policies, creating handles to new Security Access Tokens (SATs), writing to the Security event log, and maintaining the Netlogon RPC secure channel with a domain controller. Not surprisingly, this process is the target for many attacks, such as DLL-injections to dump password hashes or to perform pass-the-hash impersonation attacks.

However, if Credential Guard is enabled, lsass.exe gets a companion process: lsaiso.exe. With Credential Guard enabled, password hashes, encryption keys and other secrets are moved out of lsass.exe and into a strange netherworld of memory controlled by the Hyper-V hypervisor and the IOMMU in the motherboard. These memory pages are inaccessible to the kernel! For the OS to use these password hashes and encryption keys, lsass.exe must route its requests through lsaiso.exe, which acts like a proxy server to make calls into this virtualization-based netherworld. (Unfortunately, we can't cover Credential Guard here, but it's in SEC505.)



WINLOGON.EXE - The Windows User Logon Process

Right after boot-up, the original smss.exe creates session 1, copies itself into session 1 as a new process, launches csrss.exe and winlogon.exe, then that smss.exe exits, leaving csrss.exe and winlogon.exe still running. Next, winlogon.exe creates a window station (Winsta0) and displays the interactive desktop, which means winlogon.exe is now in control of the local keyboard, mouse and monitor (and running as Local System).

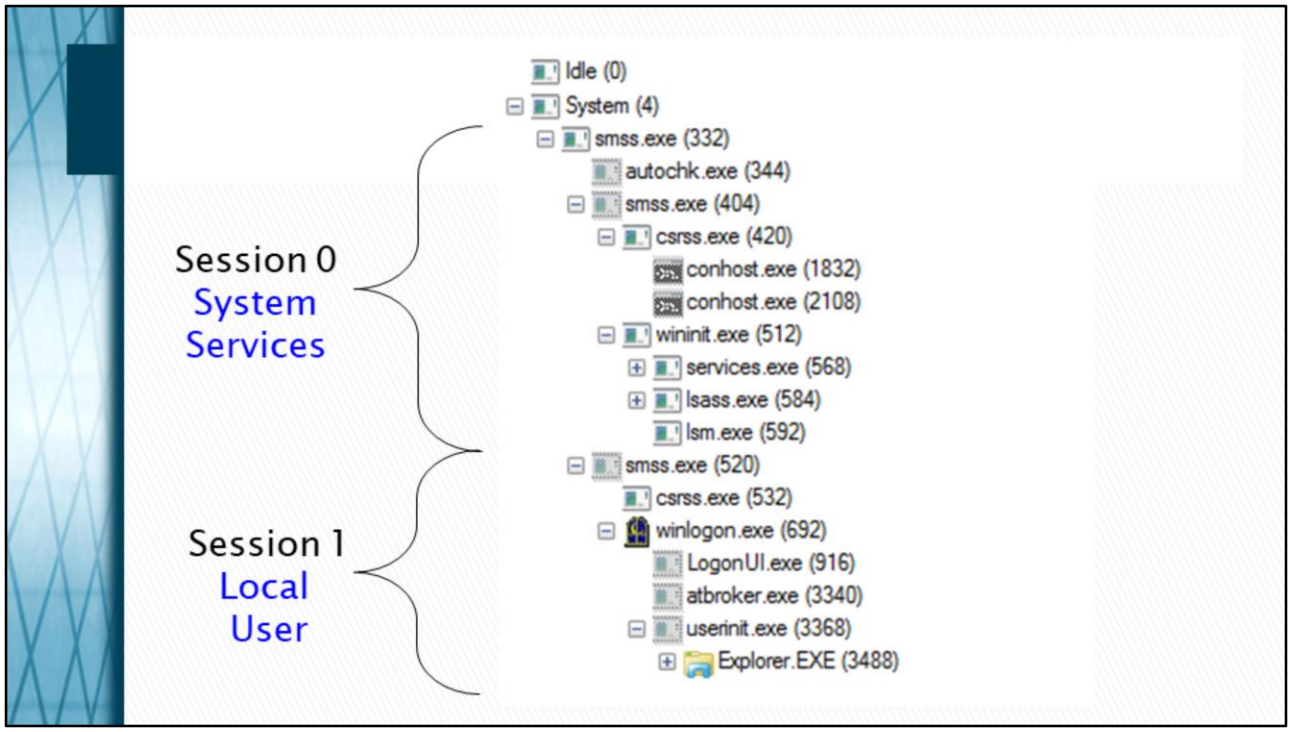
Whenever a user hits Ctrl-Alt-Del, winlogon.exe switches to another desktop and launches a special program, logonui.exe, to interact with the user. The user may be logging on initially, (un)locking the desktop, changing her password or some other task, but the user is interacting with logonui.exe on a special desktop, not winlogon.exe on the default desktop. This special desktop in which logonui.exe runs is called the "secure desktop" because its permissions only allow winlogon.exe to manage it, thus hopefully preventing trojans from capturing keystrokes. When a user locks her desktop, it's actually winlogon.exe doing the locking.

When authenticating, logonui.exe loads DLLs called "credential providers" which can handle the password, smart token or biometric information, to authenticate against the local SAM database, Active Directory, or some other third-party authentication service. The resulting credentials are given to winlogon.exe, which calls functions

within lsass.exe to do the actual authentication checking. If the authentication succeeds, lsass.exe gives winlogon.exe a handle to a Security Access Token (SAT) for the user, then winlogon.exe uses that token to set permissions on the interactive desktop (not the secure desktop) for the user's benefit, loads the user's profile, launches userinit.exe with that user's token, and then userinit.exe typically runs explorer.exe. Once userinit.exe finishes its other tasks, such as running logon scripts, it exits, leaving explorer.exe still running (which is why explorer.exe doesn't have a running parent in tree view).

Incidentally, when user environment debug logging is enabled (KB221833), it is mainly winlogon.exe writing to the userenv.log file. And while userinit.exe and explorer.exe are the typical initial processes, registry edits permit different or additional processes to be launched for the user instead (for example, on Windows Server Core, userinit.exe runs CMD.EXE instead).

Finally, atbroker.exe helps users with disabilities navigate winlogon.exe's desktops and logonui.exe's prompts.

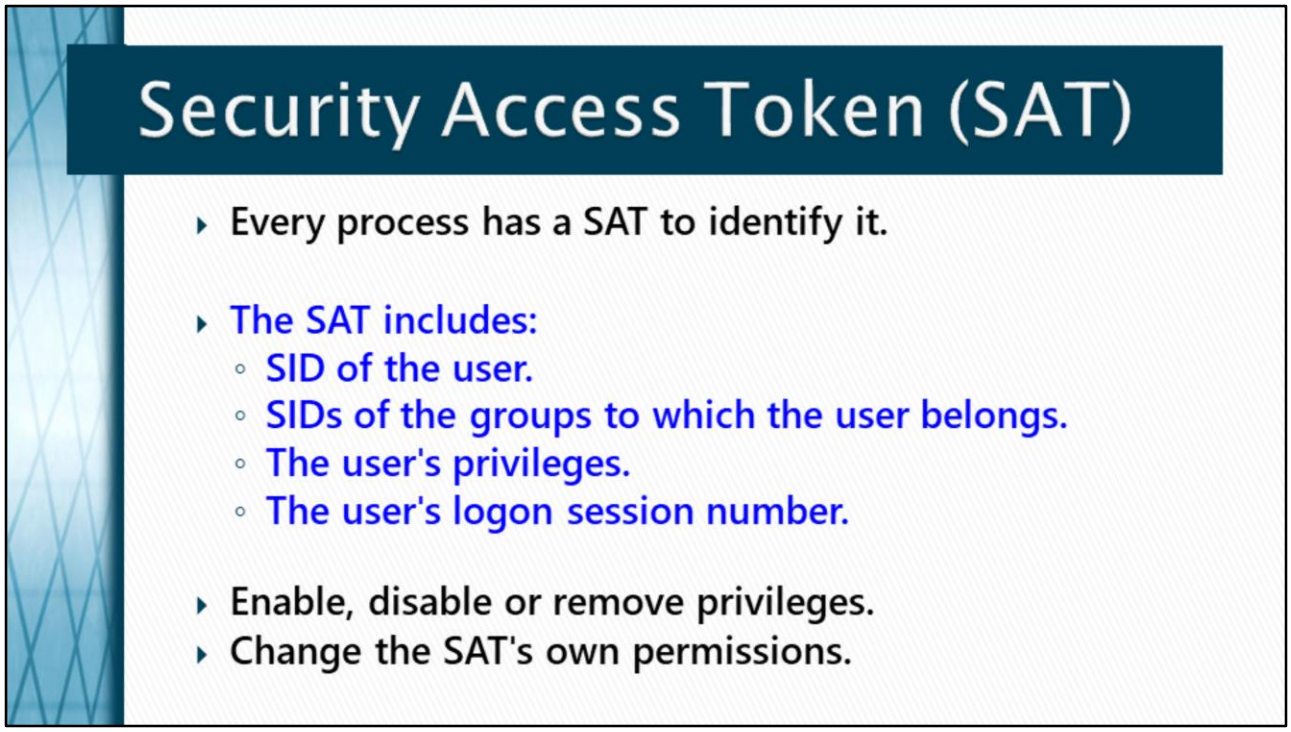


Sysinternals Process Monitor Tree

The above screenshot was made with Sysinternals Process Monitor, not Process Hacker, after configuring that tool for boot logging and then rebooting the computer (http://www.microsoft.com/sysinternals).

To see this yourself, in Process Monitor pull down the Options menu > Enable Boot Logging. Reboot the computer. Open Process Monitor again, and when prompted, save the boot log to a file. Then pull down the File menu > Open > load the boot log > Tools menu > Process Tree.

The screenshot shows how smss.exe creates session 1 for csrss.exe and winlogon.exe, smss.exe exits, then winlogon.exe spawns logonui.exe and userinit.exe. Credentials are collected by logonui.exe, given to winlogon.exe, which checks them with lsass.exe, then winlogon.exe runs userinit.exe to create the user's desktop environment for explorer.exe to create the Start menu, taskbar and desktop icons. Explorer.exe is run by default as the user's initial process, but actually any process could be launched first instead.

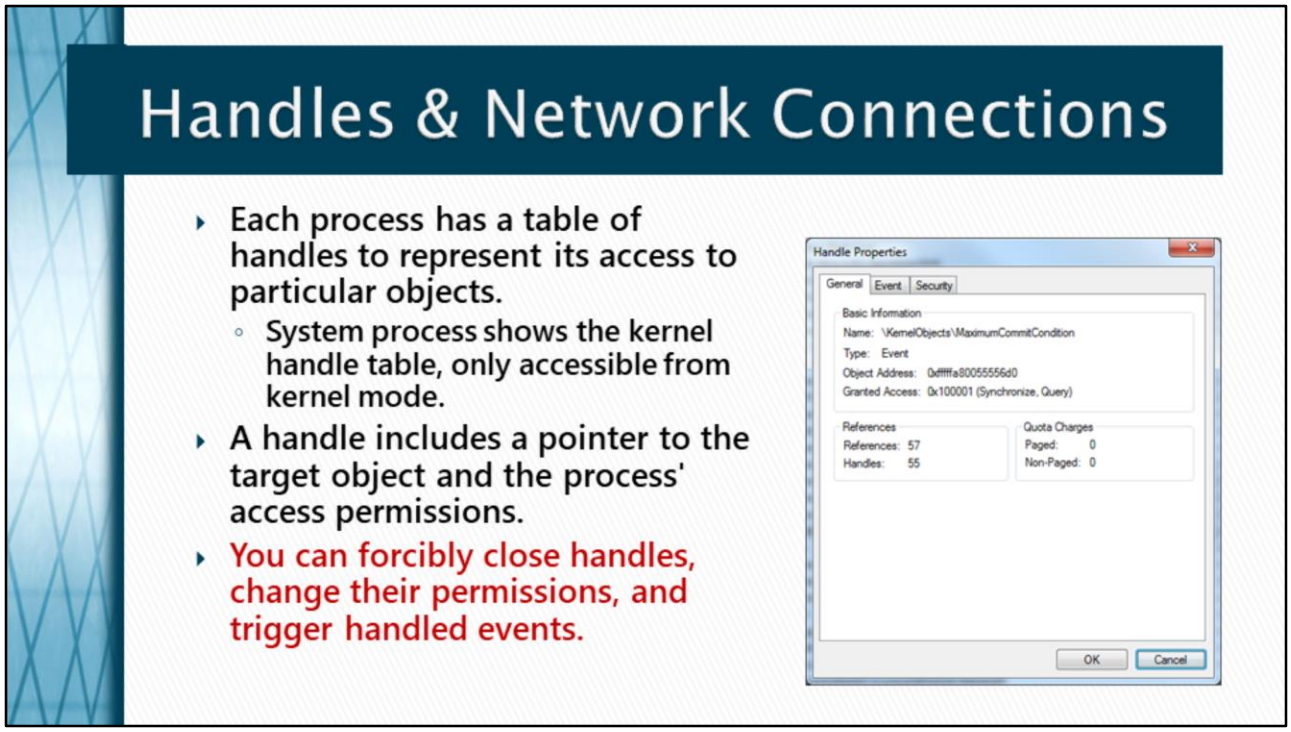


Security Access Token (SAT)

Every process and thread has an associated Security Access Token (SAT) to represent the identity under which that process or thread is running. The SAT includes the user's Security ID (SID) number, the SIDs of the groups to which the user belongs, the user's privileges, the session number, and other information. The SAT is created by lsass.exe (and the kernel's security reference monitor) when a user logs on, then it is attached to every process the user launches.

Not only can you view SAT data on the Token tab of a process, you can also enable/disable/remove privileges (but not add them) and modify the permission on the token object itself.

When User Account Control (UAC) modifies the SAT in an un-elevated process, you can also see the original elevated SAT (the Linked Token).

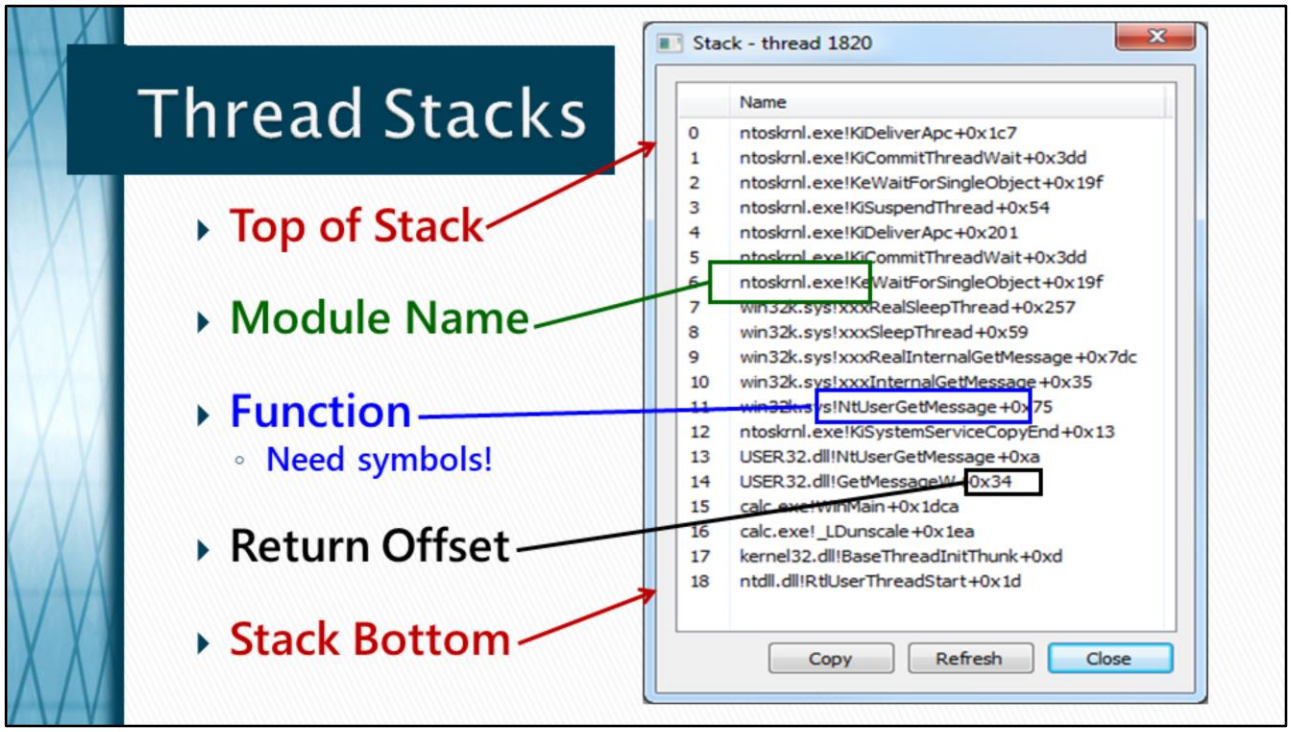


Handles

Every process has an associated table of handles. A "handle" represents the access of a process to another object. Some of these objects exist only in the kernel, such as threads, tokens, sessions, ports and processes; while other objects are created by the kernel to represent "real" resources to which the OS controls access, such as files and registry keys. A handle includes a pointer to the target object, as well as a list of access permissions. Processes can acquire handles by creating new objects, requesting the object by name, inheriting the handle from parent processes, and other methods. Before a thread in a process can interact with an object, the process must obtain a handle. In the kernel, only the object manager can create handles, and it does so only after checking permissions with the security reference monitor.

On the Handles tab of a process, not only can you view handles, you can forcibly close them, change their permissions, and change their "Inheritable" and "Protect from Close" flags. If the handle is to an event, you can also trigger that event by hand and see what the process does -- it's fun!

When you examine the handles of the System process, you are seeing a different handle table: the kernel handle table. Kernel handles are only accessible in kernel mode, but from within *any* process in kernel mode.



Thread Stacks

Each process must have one or more threads. A thread has a stack (the process does not) and the thread is what actually performs work (the process is just a supporting environment for threads). The Threads tab of a process shows the current threads in that process. Each thread has a unique ID number (TID), a multitasking priority for scheduling by the dispatcher, and a starting address for a function in a module.

If you have symbols configured, the start address of a thread will be formatted like "*module* **!** *function*", where *module* is the DLL or EXE which asked the OS to create the thread and *function* is what was given to the OS as the first thing the thread should do. If you don't have symbols configured, the start address will simply show "*module* **+***number*" where *number* is a hex offset into the module of a function.

Manipulate Threads

If you right-click a thread, you can terminate, suspend or resume it. You can also cancel any synchronous input/output procedures it is carrying out, change its CPU affinity, multitasking priority, input/output priority, and permissions. If the thread manipulates a graphical desktop window, it will usually be shown with yellow highlighting, and you can right-click and select Windows to see which ones.

Inspect Stack

If you right-click a thread and select Inspect (or just double-click the thread), you will see the thread's stack. The bottom of the stack is historically the first function call, then later function calls appear as you move up the stack in the dialog box, each function calling the one above it, from bottom to top. When you get to the top of the stack, you'll see the most recent function call.

Strictly speaking, most threads begin life as a call to "ntdll.dll **!** RtlUserThreadStart", so that is what's shown at the bottom of the stack, but that wouldn't be very informative if shown on the Threads tab; the thread's interesting starting function (as shown on the Threads tab) is typically the second or third function call up from the bottom of the stack.

Line 14 in the screenshot looks like "USER32.DLL **!** GetMessageW**+**0x34". The module is "USER32.DLL", the called function is "GetMessageW", and the return address offset is "0x34". What is the return address offset used for? Notice that line 14 shows that GetMessageW called another function above it named "NtUserGetMessage" in line 13. Later, when NtUserGetMessage finishes (line 13), control should return to the address of GetMessageW (line 14) and add 0x34 bytes to the stack pointer to clean up the stack so that the stack is the same as before the other function was called.

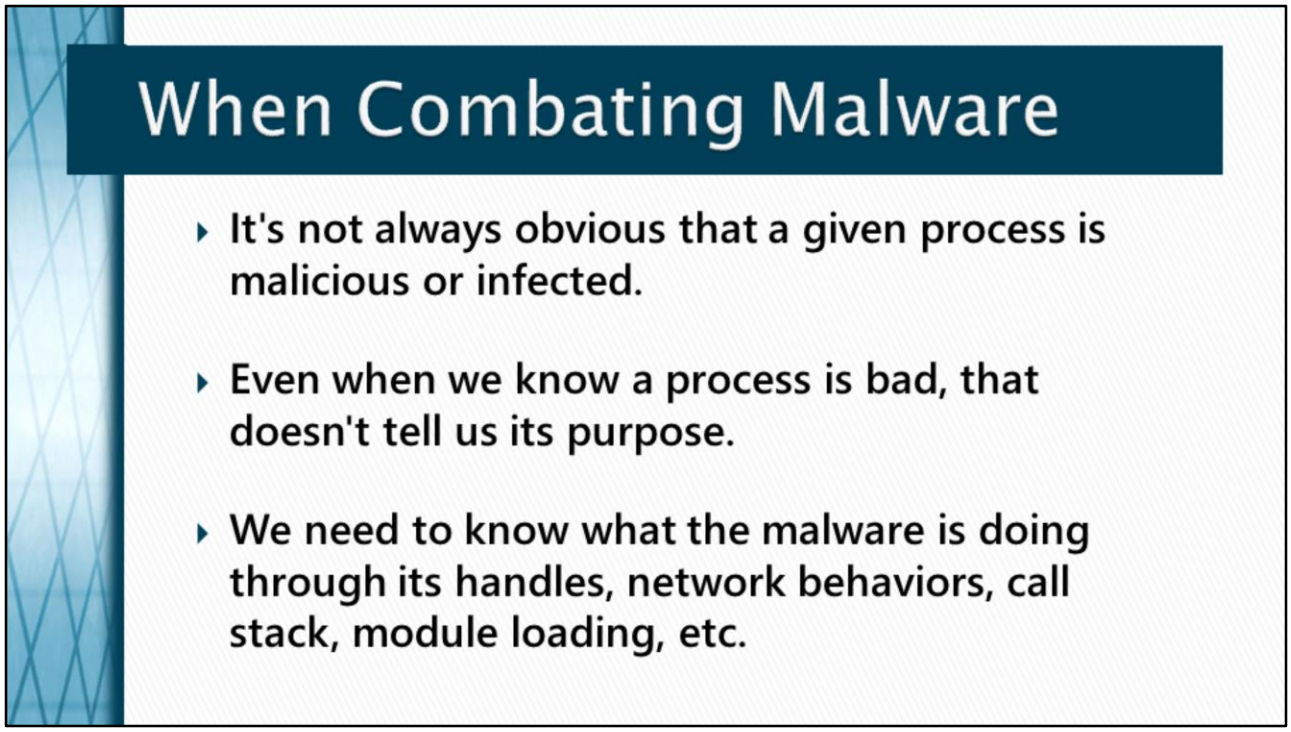
Misc Notes
On both the Threads tab and in the stack window, you can add many more column headers.

Sometimes in a stack trace you'll see strange "_", "@" and "?" characters. These come from the "decorations" of the symbols in the symbol files. The decorations indicate the function's calling convention, the function's number of argument bytes, whether it is a symbol from a precompiled header (pch), an imported symbol from another module, linker-generated strings, or other low-level details too numerous to describe here.

You will often see function calls in the stack with a name like "*Wait*" since most threads spend most of their time just sitting and waiting for other events to occur.

Each user-mode process has two stacks per thread: one user-mode, one kernel-mode. Additionally, each CPU core has its own Deferred Procedure Call (DPC) stack too, but this is a different kind of stack, it's more like a work queue or TODO list.

When you see a thread referred to as "2c5.4ba", the "2c5" is the process ID number in hex and the "4ba" is the thread ID number in hex, hence, "2c5.4ba" refers to PID 709 with TID 1210.



When Combating Malware

When you are not certain whether a process is malicious or not, examining the SAT, handles, listening ports, network connections, and stacks of that process can help.

Often, malware requires elevated privileges in order for the initial infection to succeed (such as having the Debug Programs privilege) or will seek to raise the privileges of a victim process; in either case, viewing the Security Access Token (SAT) of the process helps to understand what it can and cannot do.

The handles and networking characteristics of a process reveals what that process is interested in and what it may be doing. If a process appears to be one thing (a game, for example), but has handles, listening ports and live network connections which are very unusual for that type of program, it's a sign that the process may be malicious.

The imported and exported functions of a process can be very revealing, and so can the stack traces of the threads in a process for the same reasons. When viewing stack traces, we are getting a glimpse into what a process is doing at a moment in time. When a process appears to be one thing, again, but we see strange function calls for that supposed type of process, it's another sign that the process may be infected or malicious.

Thank You for Attending!

Please fill out the evaluation form. All written comments are read and appreciated!

To get the PDF of these slides and notes, please visit:

http://SEC505.com

This will redirect to GitHub, then look in the "SANS-Slides" section for the PDF.

For the speaker's six-day SANS Institute's *Securing Windows and PowerShell Automation* course (course number SEC505), please visit:

https://sans.org/SEC505

Get updates and other Windows security news by following **@JasonFossen** on Twitter.

Thank You! Have A Wonderful Conference!