

The Windows Process Journey

version 1.0 (beta)

March-2023

By Shlomi Boutnaru



Created using [Craiyon AI Image Generator](#)

Table of Contents

Table of Contents	2
Introduction	3
System Idle Process (PID 0)	5
smss.exe (Session Manager Subsystem)	6
csrss.exe (Client Server Runtime Subsystem)	8
wininit.exe (Windows Start-Up Application)	10
winlogon.exe (Windows Logon Application)	11
userinit.exe (Userinit Logon Application)	12
dwm.exe (Desktop Window Manager)	13
LogonUI.exe (Windows Logon User Interface Host)	15
explorer.exe (Windows Explorer)	17
svchost.exe (Host Process for Windows Services)	18
ctfmon.exe (CTF Loader)	20
audiodg.exe (Windows Audio Device Graph Isolation)	21
rdpclip.exe (RDP Clipboard Monitor)	22
smartscreen.exe (Windows Defender SmartScreen)	23

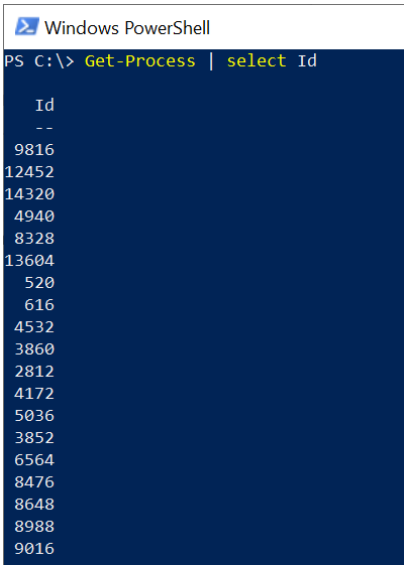
Introduction

Before speaking about a specific process I wanted to talk about an attribute related to all processes on Windows which is not so well known among all administrators/users/programmers etc.

I encourage you before reading the next lines to open any process listing app/program that you like in Windows (tasklist, task manager, process explorer or anything else) and go over PID numbers of all the processes - What can you learn from those numbers?

You probably saw that all of them are even numbers, what is more interesting is that if you divide them by two you will still get an even number - thus all the PIDs are divisible by 4!!!! BTW, the same is true for TIDs (Thread IDs) under Windows. A screenshot from

The reason for that is due to code reuse in the Windows kernel. The PIDs/TIDs are allocated by the same code which allocates kernel handles. Thus, since kernel handles are divisible by 4 so are PIDs/TIDs. We can also use the following powershell command to list only the PIDs: “Get-Process | select ID” - as shown in the screenshot below.



```
Windows PowerShell
PS C:\> Get-Process | select Id

Id
--
9816
12452
14320
4940
8328
13604
520
616
4532
3860
2812
4172
5036
3852
6564
8476
8648
8988
9016
```

But why are the handles divisible by 4? Because the two bottom bits can be ignored by Windows and could be used for tagging. You can verify it by going over the comments in ntdef.h - <https://github.com/tpn/winsdk-10/blob/master/Include/10.0.10240.0/shared/ntdef.h#L846>. Think about the pattern for each PID/TID in binary form to fully understand it.

Lastly, you can follow me on twitter - @boutnaru (<https://twitter.com/boutnaru>). Also, you can read my other writeups on medium - <https://medium.com/@boutnaru>. Lets GO!!!!!!

System Idle Process (PID 0)

The goal of this process is to give the CPU something to execute in case there is nothing else to do (thus it is called idle ;-). Let's think about the next situation, we have a process using 30% of CPU, in that case PID 0 (System Idle) will consume the remaining 70%. Also, Idle is the first process that the kernel starts.

Moreover, there is a kernel thread of System Idle for each vCPU the OS has identified (check out the screenshot below which shows that. The VM which I have used had 3 vCPUs - also see the first field in the table showing the "Processor").

The reason for having an "Idle Process" is to avoid an edge case in which the scheduler (Windows schedule based on threads) does not have any thread in a "Ready" state to execute next. By the way, there are also other schedulers IO and Memory, which we will talk about in one of the next posts/writeups.

When the kernel threads are executed they can also perform different power saving tricks regarding the CPU. One of them could be halting different components which are not in use until the next interrupt arrives. The kernel threads can also call functions in the HAL (hardware abstraction layer, more on that in the future) in order to perform tasks such as reducing the CPU clock speed. Which optimization is performed is based on the version of Windows, hardware and the firmware installed.

System Idle Process:0 Properties

Image		Performance		Performance Graph		Disk and Network	
GPU Graph		Threads		TCP/IP		Security	
				Environment		Strings	

Count: 3

Proces...	CPU	CSwitch Delta	Suspend Count	Start Address
1	32.32	238		ntoskrnl.exe!KeSyn...
2	30.88	419		ntoskrnl.exe!KeSyn...
0	30.88	431		ntoskrnl.exe!KeSyn...

smss.exe (Session Manager Subsystem)

“smss.exe” is the first user-mode process, it is executed from the following location: %SystemRoot%\System32\smss.exe. It's part of Windows since Windows NT 3.1 (1993). Thus, it starts as part of the OS startup phase and performs different tasks such as those we are doing to detail next (The order of writing is not the order of execution).

Performing delayed renaming/file deletion changes based on configuration in the Registry - “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\FileRenameOperations” (for now we should know the Registry central repository for Windows configuration, more on this in the future).

Creation of DOS device mapping based on “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\DOS Devices” such as AUX, CON, PIPE and more (a short explanation could be found here - <http://winapi.freetchsecrets.com/win32/WIN32DefineDosDevice.htm>).

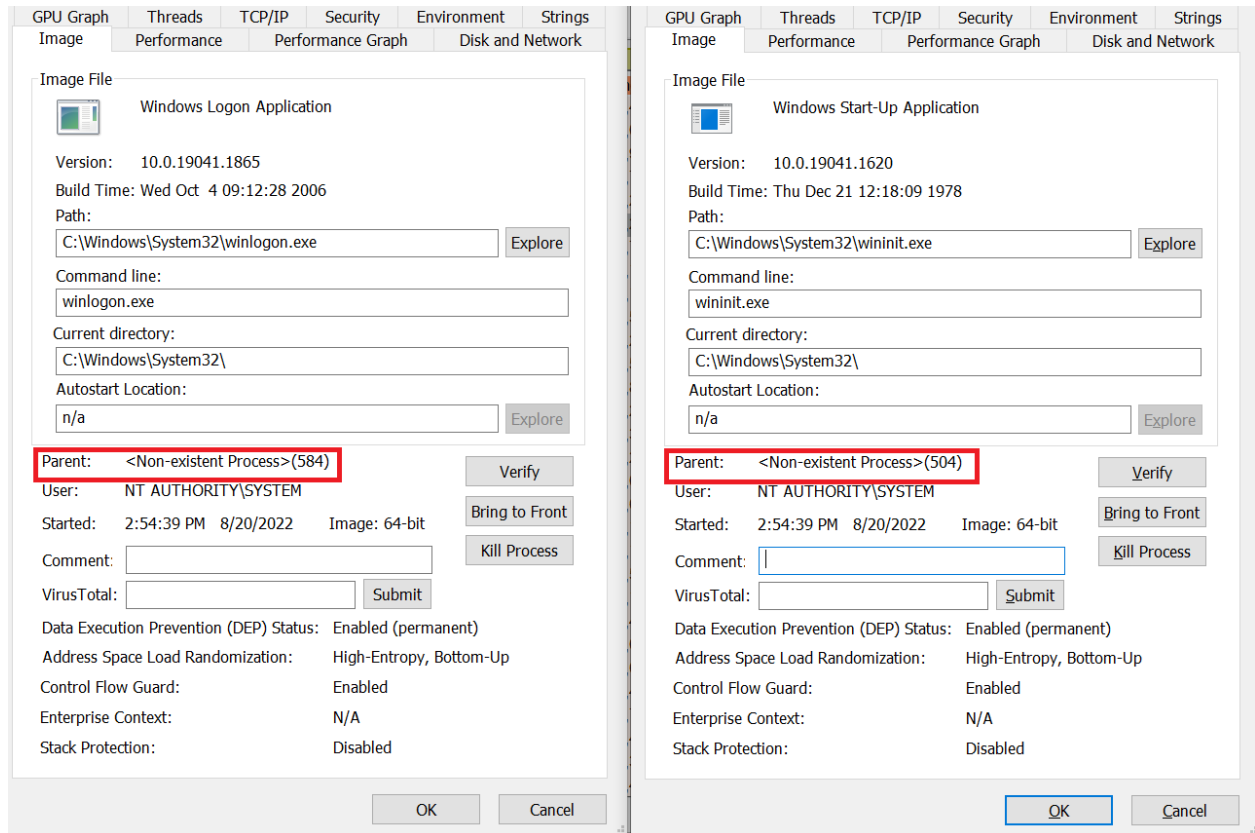
Loading the subsystems which are configured in the Registry - “HKLM\System\CurrentControlSet\Control\Session Manager\SubSystems”. At minimum we have the kernel part of the Win32 Subsystem (aka win32k.sys) and on session 0, which is the session in which Windows' services are executed - smss.exe starts “csrss.exe” and “wininit.exe” (you can also read about them in the following pages).

Also, on session 1, which is the first user session - smss.exe starts “csrss.exe” and “winlogon.exe”. Of course, they could be multiple sessions if more users are logged on (locally or using RDP).

Moreover, both the page files (used for virtual memory) and environment variables (“HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment”) are created by “smss.exe”. There are also more actions regarding memory management, KnownDlls, power management and more that are going to be discussed in the future. “smss.exe” also takes part when creating a new RDP session, we will detail this process after taking more in depth about sessions, desktops and windows stations in a future writeup - so stay tuned.

Anyhow, we should expect only one instance of “smss.exe” running without any children processes on session 0, with PPID 4 (“System Process”, on which we will also have a separate writeup). This “smss.exe” is called the master, it is responsible for creating at minimum 2 instances of itself for session 0 and 1 (in order to do the work we detailed above). The other instances of “smss.exe” (the non-master) will terminate after finishing the session initialization

phase of a new session. On the screenshot below we can see “wininit.exe” from session 0 and “winlogon.exe” from session 1 both of them having a non-existent parent.



csrss.exe (Client Server Runtime Subsystem)

The goal of “csrss.exe” (Client Server Runtime Subsystem) is to be the user-mode part of the Win32 subsystem (which is responsible for providing the Windows API). “csrss.exe” is included in Windows from Windows NT 3.1. It is located at “%windir%\System32\csrss.exe” (which is most of the time C:\Windows\System32\csrss.exe).

From Windows NT 4.0 most of the Win32 subsystem has been moved to kernel mode - “With this new release, the Window Manager, GDI, and related graphics device drivers have been moved to the Windows NT Executive running in kernel mode”¹. Thus “csrss.exe” manages today GUI shutdowns and windows console (today it is “cmd.exe”).

Overall, we can say that today “csrss.exe” handles things like process/threads, VDM (Visual DOS machine emulation), creating of temp files and more². It is executed by “local system” and there is one instance per user session. Thus, at minimum we will have two (one for session 0 and one for session 1) - as shown in the screenshot below. “csrss.exe” has a handle for each process/thread in the specific session it is part of. Also, for each running process a CSR_PROCESS structure is maintained³, by the way we can leverage this fact for identifying hidden processes (like by using “psxview”⁴ from the volatility framework).

“smss.exe” is the process which starts “csrss.exe” together with “winlogon.exe” (more about it in a future writeup), after finishing “smss.exe” exits. In case you want to read more about “smss.exe”⁵. By the way, from Windows 7 (and later) “csrss.exe” executes “conhost.exe” instead of drawing the console windows by itself (I am going to elaborate about that in the next writeup).

Lastly, “csrss.exe” loads “csrsrv.dll”, “basesrv.dll” and “winsrv.dll” as shown in the screenshot below. If we want to go over some of the source code of “csrss.exe” we can use the ReactOS which is a “A free Windows-compatible Operating System”, which is hosted in github.com. The relevant code of the entire subsystem can be found at <https://github.com/reactos/reactos/tree/master/subsystems/csr>. We can also debug “csrss.exe” using WinDbg, it is important to know that since Windows “csrss.exe” is a protected process so it can be debugged from kernel mode only⁶. A list of all the “csrss.exe” API list can be found here https://j00ru.vexillium.org/csrss_list/api_table.html.

¹[https://learn.microsoft.com/en-us/previous-versions//cc750820\(v=technet.10\)?redirectedfrom=MSDN#XSLTsection124121120120](https://learn.microsoft.com/en-us/previous-versions//cc750820(v=technet.10)?redirectedfrom=MSDN#XSLTsection124121120120)

² <https://j00ru.vexillium.org/2010/07/windows-csrss-write-up-the-basics/>

³ <https://www.geoffchappell.com/studies/windows/win32/csrsrv/api/process/process.htm>

⁴ <https://github.com/volatilityfoundation/volatility/wiki/Command-Reference-Mal#psxview>

⁵ <https://medium.com/@boutnaru/the-windows-process-journey-smss-exe-session-manager-subsystem-bca2cf748d33>

⁶ <https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-csrss>

Process Explorer - Sysinternals: www.sysinternals.com (Administrator)

File Options View Process Find Users DLL Help

Process	CPU	Private Bytes	Working Set	PID	Description	Compa...	Protection	Session
csrss.exe	< 0.01	2,092 K	4,616 K	528			PsProtectedSignerWinTcb-Light	0
csrss.exe		1,640 K	3,520 K	612			PsProtectedSignerWinTcb-Light	1
csrss.exe	< 0.01	2,316 K	5,808 K	1276			PsProtectedSignerWinTcb-Light	2

Handles DLLs Threads

Name	Description	Company Name	Path
basesrv.dll	Windows NT BASE API Server DLL	Microsoft Corporation	C:\Windows\System32\basesrv.dll
bcrypt.dll	Windows Cryptographic Primitives Li...	Microsoft Corporation	C:\Windows\System32\bcrypt.dll
bcryptprimitives.dll	Windows Cryptographic Primitives Li...	Microsoft Corporation	C:\Windows\System32\bcryptprimitives.dll
cfgmgr32.dll	Configuration Manager DLL	Microsoft Corporation	C:\Windows\System32\cfgmgr32.dll
combase.dll	Microsoft COM for Windows	Microsoft Corporation	C:\Windows\System32\combase.dll
csrssv.dll	Client Server Runtime Process	Microsoft Corporation	C:\Windows\System32\csrssv.dll
csrss.exe	Client Server Runtime Process	Microsoft Corporation	C:\Windows\System32\csrss.exe
csrss.exe.mui	Client Server Runtime Process	Microsoft Corporation	C:\Windows\System32\en-US\csrss.exe.mui
gdi32.dll	GDI Client DLL	Microsoft Corporation	C:\Windows\System32\gdi32.dll
gdi32full.dll	GDI Client DLL	Microsoft Corporation	C:\Windows\System32\gdi32full.dll
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\kernel32.dll
kernelBase.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\kernelBase.dll
locale.nls			C:\Windows\System32\locale.nls
msvcrt_win.dll	Microsoft® C Runtime Library	Microsoft Corporation	C:\Windows\System32\msvcrt_win.dll
ntdll.dll	NT Layer DLL	Microsoft Corporation	C:\Windows\System32\ntdll.dll
rpcrt4.dll	Remote Procedure Call Runtime	Microsoft Corporation	C:\Windows\System32\rpcrt4.dll

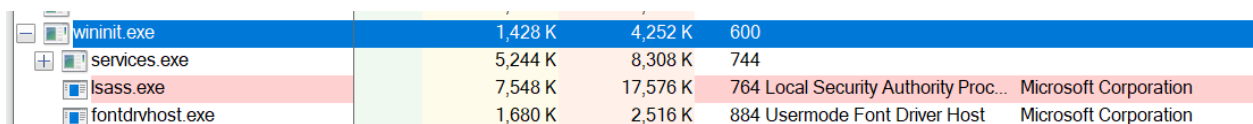
wininit.exe (Windows Start-Up Application)

“wininit.exe” is an executable which is responsible for different initialization steps as described next. The executable is located at “%windir%\System32\wininit.exe” (On 64 bit systems there is only a 64 bit version with no 32 bit version—in contrast to other executables such as cmd.exe). It is started by the first “smss.exe” at session 0 under LocalSystem (S-1-5-18). Overall there should be only one running instance of “wininit.exe”.

Historically, “wininit.exe” was used mainly in order to allow uninstallers to process commands stored in the “WinInit.ini” file. By doing so it allowed programs to take action while the system is booting⁷.

Moreover, “wininit.exe” is responsible for a couple of system initialization steps. Among them are: creating the %windir%\temp folder, initializing the user-mode scheduling infrastructure, creating a window station (Winsta0) and two desktops (Winlogon and Default) for processes to run on in session 0, marking itself critical so that if it exits prematurely and the system is booted in debugging mode (it will break into the debugger) and waiting forever for system shutdown⁸.

Also, “wininit.exe” launches “services.exe” (SCM—Service Control Manager) , “lsass.exe” (Local Security Authority Subsystem) and “fontdrvhost.exe” (Usermode Font Driver Host)—as seen in the screenshot below. If you want more information about service management I suggest reading <https://medium.com/@boutnaru/windows-services-part-1-5d6c2d25b31c> and <https://medium.com/@boutnaru/windows-services-part-2-7e2bdab5bce4>. Regarding the last two (“lsass.exe” and “fontdrvhost.exe”) I am going to write something in the near future.



wininit.exe	1,428 K	4,252 K	600	
services.exe	5,244 K	8,308 K	744	
lsass.exe	7,548 K	17,576 K	764	Local Security Authority Proc... Microsoft Corporation
fontdrvhost.exe	1,680 K	2,516 K	884	Usermode Font Driver Host Microsoft Corporation

⁷<https://social.technet.microsoft.com/Forums/ie/en-US/df6f5eeb-cbb9-404f-9414-320ea02b4a60/wininitexe-what-is-is-and-why-is-it-constantly-running>

⁸ <https://learn.microsoft.com/en-us/answers/questions/405417/explanation-of-windows-processes-and-dlls.html>

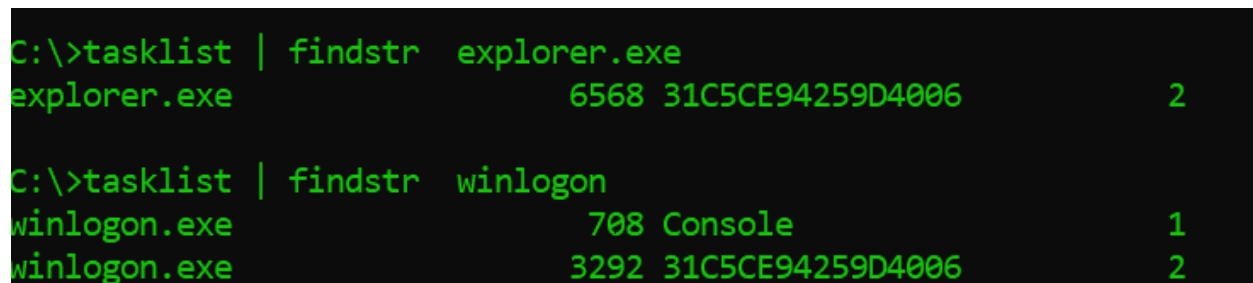
winlogon.exe (Windows Logon Application)

“winlogon.exe” is an executable which is located at “%windir%\System32\winlogon.exe” (On 64 bit systems there is only a 64-bit version with no 32-bit version like with other executables such as cmd.exe). It is executed under the “NT AUTHORITY\SYSTEM” (S-1-5-18) user. “Winlogon.exe” provides interactive support for interactive logons⁹.

Overall, “winlogon.exe” manages user interactions which are related to the security of the system. Among them are: coordination of the logon flow, handling logout (aka logoff), starting “LogonUI.exe”¹⁰, allowing the alteration of the user’s password and locking/unlocking the server/workstation¹¹. In order to obtain user information for logon “winlogon.exe” uses credentials providers which are loaded by “LogonUI.exe” - more on them in a future writeup. For authenticating the user “winlogon.exe” gets help from “lsass.exe” (more on that in a future writeup).

In its initialization phase “winlogon.exe” registers the “CTRL+ALT+DEL” secure attention sequence¹² before any application can do that. Also, “winlogon.exe” creates three desktops within WinSta0: “Winlogon Desktop” (it is the desktop that the user is switched to when SAS is received), “Application Desktop” (this is the desktop created for the logon session of the user) and “ScreenSaver Desktop” (this is the desktop used when a screensaver is running). For more information I suggest reading “Initializing Winlogon”¹³.

Before any logon is performed to the system, the visible desktop is Winlogon’s. Moreover, the number of instances that we expect to have is one for each interactive logon session that is present (as the number of “explorer.exe”) and another one which is for the next session that can be created - as seen in the screenshot below.



```
C:\>tasklist | findstr explorer.exe
explorer.exe                6568 31C5CE94259D4006      2

C:\>tasklist | findstr winlogon
winlogon.exe                708 Console                1
winlogon.exe                3292 31C5CE94259D4006      2
```

Lastly, I think it is a good idea to go over the reference implementation in ReactOS for “winlogon.exe”¹⁴.

⁹ <https://learn.microsoft.com/en-us/windows/win32/secgloss/w-gly>

¹⁰ <https://medium.com/@boutnaru/the-windows-process-journey-logonui-exe-windows-logon-user-interface-host-4b5b8b6417cb>

¹¹ <https://www.microsoftpressstore.com/articles/article.aspx?p=2228450&seqNum=8>

¹² <https://medium.com/@boutnaru/security-sas-secure-attention-sequence-da8766d859b5>

¹³ <https://learn.microsoft.com/en-us/windows/win32/secauthn/initializing-winlogon>

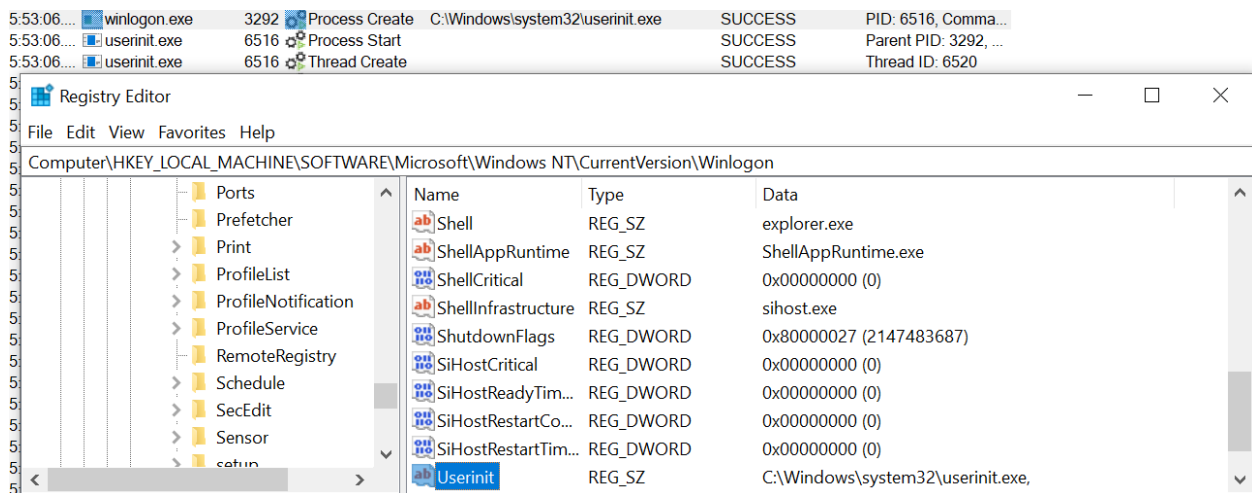
¹⁴ <https://github.com/reactos/reactos/tree/2752c42f0b472f2db873308787a8b474c4738393/base/system/winlogon>

userinit.exe (Userinit Logon Application)

“userinit.exe” is an executable which is located at “%windir%\System32\userinit.exe” (On 64 bit systems there is also a 32 bit version located at “%windir%\SysWOW64\userinit.exe”). It is started by the “winlogon.exe” - as seen in the screenshot below (taken from ProcMon). Also, “userinit.exe” is executed with the permissions of the user which is logging in to the system.

Overall, “userinit.exe” is responsible for loading the user’s profile and executing startup applications while the logon process of the user is being performed. Thus, it will execute logon scripts¹⁵.

“C:\Windows\System32\userinit.exe” is defined by default as the executable for the UserInit phase under the “userinit” key in the registry¹⁶ - as shown in the screenshot below (taken from “regedit.exe”). Moreover, “userinit.exe” runs the shell of the logged on user, which is by default “explorer.exe” as configured in the registry under the “shell” key¹⁷ - as shown in the screenshot below (taken from “regedit.exe”).



I think it is a good idea to go over the reference implementation in ReactOS for “userinit.exe” (<https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/system/userinit>).

¹⁵ <https://www.minitool.com/news/userinit-exe.html>

¹⁶ HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\UserInit

¹⁷ HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell

dwm.exe (Desktop Window Manager)

“dwm.exe” (Desktop Window Manager) is the executable which handles different tasks in the display process of the Windows UI like rendering effects. Among those efforts are: live taskbar thumbnails, Flip3D, transparent windows and more¹⁸. The executable is located at “%windir%\System32\dwm.exe” (On 64 bit systems there is only a 64 bit version with no 32 bit version like with other executables such as cmd.exe).

Thus, we can think about “dwm.exe” as a “compositing windows manager”. A “windows manager” is computer software that controls the placement and appearance of a window as part of a “window system” in a GUI environment¹⁹. So, a “compositing windows manager” is a “window manager” that provides applications with an off-screen buffer for each window. The goal of the manager is to composite all the windows’ buffers into an image representing the screen and commit it to the display memory²⁰.

The desktop composition feature was introduced in Windows Vista. It changed the way applications display pixels on the screen (as it was until Windows XP). When desktop composition is enabled, individual windows no longer draw directly to the screen (or primary display device). Their drawings are redirected to off-screen surfaces in video memory, which are then rendered into a desktop image and presented on the display.

For more information I suggest reading the following links <https://learn.microsoft.com/en-us/windows/win32/dwm/dwm-overview> and https://learn.microsoft.com/en-us/archive/blogs/greg_schechter/under-the-hood-of-the-desktop-window-manager.

Under Windows 10, there is one instance of “dwm.exe” for each session (excluding session 0). The parent process for each “dwm.exe” is “winlogon.exe”. The user which is associated with the security token of each “dwm.exe” has a the pattern of “Window Manager\DWM-{SESSION_ID}” and a SID of pattern “S-1-5-90-0-{SESSION_ID}” as shown in the screenshot below (taken from Process Explorer).

¹⁸ <https://learn.microsoft.com/en-us/windows/win32/dwm/dwm-overview>

¹⁹ https://en.wikipedia.org/wiki/Window_manager

²⁰ https://en.wikipedia.org/wiki/Compositing_window_manager

Process Explorer - Sysinternals: www.sysinternals.com (Administrator)

File Options View Process Find Users Help

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Session
winlogon.exe		2,588 K	3,292 K	692	Windows Logon Application	Microsoft Corporation	1
fontdrvhost.exe		1,436 K	1,072 K	864	Usermode Font Driver Host	Microsoft Corporation	1
LogonUI.exe		17,728 K	37,164 K	820	Windows Logon User Interface	Microsoft Corporation	1
dwm.exe	< 0.01	27,240 K	28,076 K	956	Desktop Window Manager	Microsoft Corporation	1
csrss.exe	< 0.01	2,160 K	4,408 K	5024			2
winlogon.exe		2,784 K	8,572 K	4328	Windows Logon Application	Microsoft Corporation	2
fontdrvhost.exe		4,408 K	5,136 K	5036	Usermode Font Driver Host	Microsoft Corporation	2
dwm.exe	< 0.01	111,608 K	167,180 K	5308	Desktop Window Manager	Microsoft Corporation	2

dwm.exe:956 Properties

User: Window Manager\DWM-1
SID: S-1-5-90-0-1
Session: 1 Logon Session: c91b
Virtualized: No Protected: No

Group	Flags
BUILTIN\Users	Mandatory
CONSOLE LOGON	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Mandatory Label\System Mandatory Level	Integrity
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
NT AUTHORITY\LOCAL SERVICE	Mandatory

dwm.exe:5308 Properties

User: Window Manager\DWM-2
SID: S-1-5-90-0-2
Session: 2 Logon Session: 50a4c
Virtualized: No Protected: No

Group	Flags
BUILTIN\Users	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Mandatory Label\System Mandatory Level	Integrity
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
NT AUTHORITY\LOCAL SERVICE	Mandatory
NT AUTHORITY\This Organization	Mandatory

LogonUI.exe (Windows Logon User Interface Host)

“LogonUI.exe” (Windows Logon User Interface Host) is responsible for the graphical user interface which asks the user to logon into the system (aka logon screen/lock screen). The executable file is located at “%SystemRoot%\System32\LogonUI.exe” (On 64 bit systems there is only a 64 bit version with no 32 bit version like with other executables such as cmd.exe).


Moreover, “LogonUI.exe” is executed under the Local System user (S-1-5-18) for every session (excluding session 0). “winlogon.exe” is the process which is responsible for running “LogonUI.exe” as we can see in the screenshot below, which was taken from Process Monitor²¹. Also, if you want to see how “LogonUI.exe” GUI looks in different versions of Windows²².

In the perspective of the data flow between “LogonUI.exe” and “winlogon.exe” the basic phases are as follows (after “LogonUI.exe” was launched by “winlogon.exe”). “LogonUI.exe” gets credentials from the user (like username and password) and sends them to “winlogon.exe”. “winlogon.exe” performs the authentication (since Windows Vista it is done using a credential provider, before that it was done by msgina.dll). If the authentication process succeeds, it sends a message back to “LogonUI.exe” to indicate that the user has been authenticated²³. We will get deeper into this flow after talking about “winlogon.exe”, sessions, ALPC (which is the communication line between the processes) and more.

In addition, settings for LogonUI.exe are stored in the registry in the following branch: “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI”. Among those settings we can find the user list that should be shown, the last user that logged-on and the background image. Lastly, if you want to see a reference code for “LogonUI.exe” you can check out ReactOS²⁴.

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help



Time o...	Process Name	PID	Operation	Path	Result	Detail
i:34:30...	rundll32.exe	10888	Process Exit		SUCCESS	Exit Status: 0, User...
i:34:30...	winlogon.exe	4328	Process Create	C:\Windows\system32\LogonUI.exe	SUCCESS	PID: 12924, Comm...
i:34:30...	LogonUI.exe	12924	Process Start		SUCCESS	Parent PID: 4328, ...
i:34:30...	LogonUI.exe	12924	Thread Create		SUCCESS	Thread ID: 14796
i:34:30...	LogonUI.exe	12924	Load Image	C:\Windows\System32\LogonUI.exe	SUCCESS	Image Base: 0x7ff6...
i:34:30...	LogonUI.exe	12924	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7ffb...
i:34:30...	LogonUI.exe	12924	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x7ffb...
i:34:30...	LogonUI.exe	12924	Load Image	C:\Windows\System32\KernelBase.dll	SUCCESS	Image Base: 0x7ffb...
i:34:30...	LogonUI.exe	12924	Load Image	C:\Windows\System32\msvcrt.dll	SUCCESS	Image Base: 0x7ffb...
i:34:30...	LogonUI.exe	12924	Thread Create		SUCCESS	Thread ID: 13000

²¹ <https://learn.microsoft.com/en-us/sysinternals/downloads/procmon>

²² https://media.askvg.com/articles/images3/Windows_Login_Screen.png

²³ <https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/credentials-processes-in-window-s-authentication>

²⁴ <https://github.com/reactos/reactos/tree/3647f6a5eb633b52ef4bfl6b6e43fc2b3fc72969/base/system/logonui>

explorer.exe (Windows Explorer)

“explorer.exe” is an executable which is the “Windows Explorer”. The executable is located at “%windir%\explorer.exe (On 64 bit systems there is also a 32 bit version located in %windir%\SysWOW64\explorer.exe). It is responsible for handling elements of the graphical user interface in Windows (including the taskbar, start menu, and desktop), the “File Explorer” and more. Thus, we can think about it as a graphical shell²⁵.

In case we terminate “explorer.exe” the taskbar will disappear and also the desktop both the shortcuts and the wallpaper itself²⁶. For more understanding about “explorer.exe” I think it is a good idea to go over the reference implementation in ReactOS²⁷.

Every time a user logs in interactively “explorer.exe” is executed under the user which logged on to the system²⁸. The process which starts “explorer.exe” is “userinit.exe” (I will post on it in the near future) - as can be seen in the screenshot below.

11:48:...	userinit.exe	7928	Process Create	C:\WINDOWS\Explorer.EXE	SUCCESS	PID: 11676, Comm...
11:48:...	Explorer.EXE	11676	Process Start		SUCCESS	Parent PID: 7928, ...
11:48:...	Explorer.EXE	11676	Thread Create		SUCCESS	Thread ID: 11692
11:48:...	Explorer.EXE	11676	Load Image	C:\Windows\explorer.exe	SUCCESS	Image Base: 0x7f6...

I also suggest going over the following link <https://ss64.com/nt/explorer.html> to checkout all the arguments that can be passed to “explorer.exe” while launching it. There are also several examples of usage there. By the way, it seems that Microsoft wants to decouple features from “explorer.exe” in order to make Windows 11 faster²⁹.

²⁵ <https://www.pcmag.com/encyclopedia/term/explorerexe>

²⁶ <https://copyprogramming.com/howto/what-happens-if-i-end-the-explorer-exe-process>

²⁷ <https://github.com/reactos/reactos/tree/81db5e1da884f76e6cee66b8cb1c7a2f6ff791eb/base/shell/explorer>

²⁸ <https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/windows-logon-scenarios>

²⁹ <https://www.windowslatest.com/2022/12/22/microsoft-wants-to-make-windows-11-faster-by-decoupling-features-from-explorer-exe/>

svchost.exe (Host Process for Windows Services)

“svchost.exe” is probably the builtin executable which has the most instances (for example 78 on the my testing VM) among all the running processes in Windows. We can split its name to “Svc” and “Host”, that is service host which hits its responsibility (more on that later).

The executable “svchost.exe” is located in %windir%\System32\svchost.exe. In case we are talking about the 64 bit version of Windows, there is also %windir%\SysWOW64\svchost.exe (which is a 32 bit version). Both of the files are signed digitally by Microsoft. It was introduced during Windows 2000, even though there was support for “shared service processes” already in Windows NT 3.1 (more on this in the following paragraphs).

Due to the fact, many of the Windows’ services (you can read on Windows’ Services on <https://medium.com/@boutnaru/windows-services-part-2-7e2bdab5bce4>) are implemented as DLLs (Dynamic Link Libraries) there is a need for an executable to host them. Thus, you can think about “svchost.exe” as the implementation of “shared service process” - A process which hosts/executes/runs multiple services in a single memory address space.

The configuration of services is stored in the registry (“HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services”), for each service which is hosted the name of the DLL is stored under the “Parameter” subkey in a value named “ServiceDll”. For example, in the case of the DHCP client is “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Dhcp\Parameters\ServiceDll” - as shown in the screenshot below. The ImagePath (which stores the path to the executable to run when starting the service) will be “svchost.exe” with a command line parameter of “-k” and the name of the service groups (like netsvcs, Dcomlaunch, utcsvc, and LocalServiceNoNetwork, LocalSystemNetworkRestricted).

At the end services are splitted into different groups, every group is hosted by one host process which is a single instance of “svchost.exe”. If we want to see which services are hosted on which “svchost.exe” you can use tools like “Process Explorer” and “tasklist” - as you can see in the screenshot below. The configuration of which services are part of what group we can see at “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost” (on my test VM a total of 49 groups are defined).

It is important to know that from Windows 10 (version 1903) on systems with more than 3.5GB or RAM by default there is no grouping. That is, every service will be executed in a single instance of “svchost.exe” for better security and reliability. Of course there are exceptions for that³⁰.

³⁰ <https://learn.microsoft.com/en-us/windows/application-management/svchost-service-refactoring>

Registry Editor

File Edit View Favorites Help

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Dhcp\Parameters

Name	Type	Data
(Default)	REG_SZ	(value not set)
ServiceDll	REG_EXPAND_SZ	%SystemRoot%\system32\dhcpcore.dll
ServiceDllUnload...	REG_DWORD	0x00000001 (1)

Select C:\Windows\system32\cmd.exe

```

services.exe          748 N/A
lsass.exe             768 KeyIso, SamSs, VaultSvc
svchost.exe           880 BrokerInfrastructure, DcomLaunch, PlugPlay,
                        Power, SystemEventsBroker
fontdrvhost.exe       904 N/A
fontdrvhost.exe       912 N/A
svchost.exe           992 RpcEptMapper, RpcSs
svchost.exe           492 LSM
LogonUI.exe           744 N/A
dwm.exe               796 N/A
svchost.exe           1028 TermService
svchost.exe           1104 NcbService
svchost.exe           1112 TimeBrokerSvc
svchost.exe           1188 EventLog
svchost.exe           1224 vmicheartbeat
svchost.exe           1232 vmickvpexchange
svchost.exe           1248 vmicrdv
svchost.exe           1288 vmicshutdown
svchost.exe           1296 nsi
svchost.exe           1356 vmictimesync
svchost.exe           1384 vmicvss
svchost.exe           1496 Dhcp
svchost.exe           1528 ProfSvc
svchost.exe           1556 EventSystem
svchost.exe           1576 SysMain
svchost.exe           1592 Themes
Memory Compression    1720 N/A
VSSVC.exe             1756 VSS
svchost.exe           1764 SENS

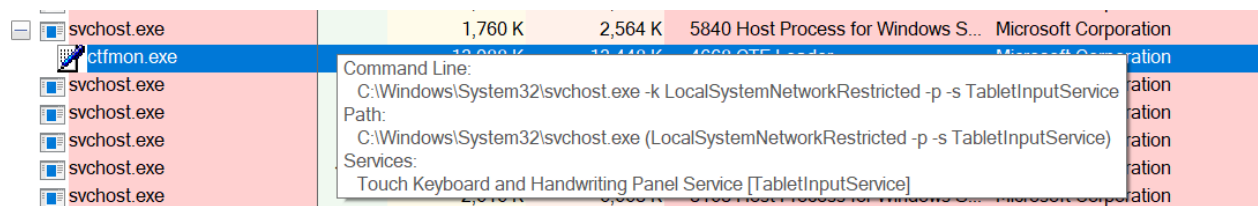
```

ctfmon.exe (CTF Loader)

“ctfmon.exe” is a user-mode process which is executed from the following location %SystemRoot%\System32\ctfmon.exe. If you are using a 64 bit version of Windows, there is also a 32 bit version of “ctfmon.exe” located at C:\Windows\SysWOW64\ctfmon.exe. By parsing the file information we can see that it is described as a “CTF Loader”. CTF stands for “Collaboration Translation Framework”, it is used by Microsoft Office.

The goal of “ctfmon.exe” is to provide different input capabilities for users such as speech and handwriting recognition. By the way, it will run even if you are not using Microsoft Office.

“Ctfmon.exe” is launched as a child process of the service TabletInputService ("Touch Keyboard and Handwriting Panel Service"), which is hosted by “svchost.exe” - as shown in the screenshot below. Thus, if we want to stop “ctfmon.exe” we can just disable/stop that service. For more information about what is “svchost.exe” you can read the following link <https://medium.com/@boutnaru/the-windows-process-journey-svchost-exe-host-process-for-windows-services-b18c65f7073f>.

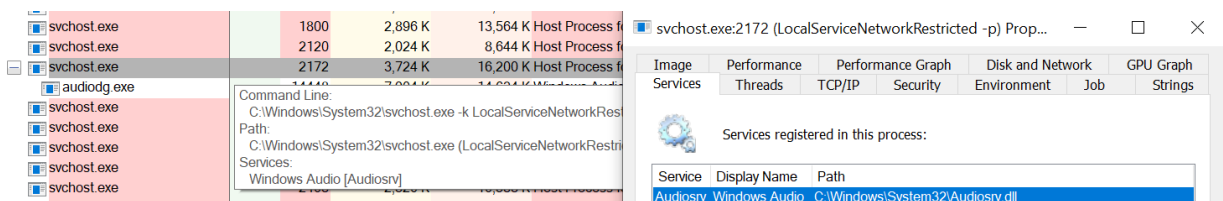


audiodg.exe (Windows Audio Device Graph Isolation)

“audiodg.exe” is an executable which is part of the Windows shared-mode audio engine as described next. The executable is located at “%windir%\System32\audiodg.exe” (On 64 bit systems there is only a 64 bit version with no 32 bit version—in contrast to other executables such as cmd.exe). The process is running under the user “NT AUTHORITY\LOCAL SERVICE”.

In Windows the audio engine runs in user mode. We have the "Windows Audio" service which is implemented in AudioSrv.dll, it is hosted using the “svchost.exe” process. The service launches a helper process “audiodg.exe”³¹. All of that is demonstrated in the screenshot below. It runs in a different login session from the logged on user (isolated) in order to that content and plug-ins cannot be modified³².

Thus, we can say that “audiodg.exe” is being utilized for all audio processing³³. It hosts the audio engine for Windows so all the digital signal processing (DSP) is performed by “audiodg.exe”. Vendors can install their own audio effects which will be processed by “audiodg.exe”³⁴. There should be one instance only of “audiodg.exe” at a specific time.



³¹ <https://learn.microsoft.com/en-us/windows-hardware/drivers/dashboard/audio-measures>

³² <https://answers.microsoft.com/en-us/windows/forum/all/audiodgexe/0c86aef4-81a5-480e-9389-d9652fee1d21>

³³ <https://answers.microsoft.com/en-us/windows/forum/all/windows-10-audiodgexe/af1b70e0-06fe-4952-8205-b6191ccb8882>

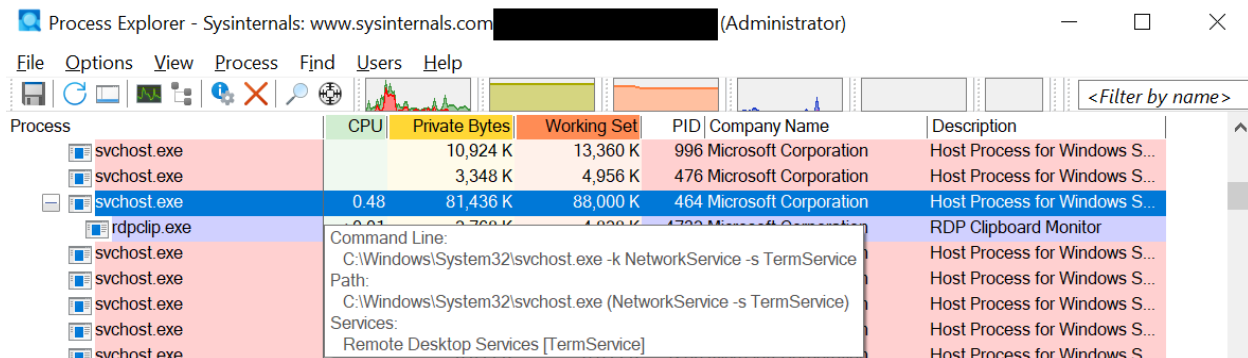
³⁴ <https://answers.microsoft.com/en-us/windows/forum/all/audiodgexe-high-cpu-and-memory/42b3f122-87bf-45cd-8ea7-08abafa9442c>

rdpclip.exe (RDP Clipboard Monitor)

“rdpclip.exe” (RDP Clipboard Monitor) is responsible for managing the shared clipboard between the local computer and the remote desktop which the user is interacting with³⁵. The executable file is located at “%windir%\System32\rdpclip.exe” (On 64 bit systems there is only a 64 bit version with no 32 bit version like with other executables such as cmd.exe).

By enabling the “Remote Desktop” capability³⁶ on Windows it allows remote management of a system using a GUI (graphical user interface) by leveraging the Remote Desktop Protocol (RDP). The default port of the protocol is TCP/3389. For more information about the protocol I suggest reading the following link <https://www.cyberark.com/resources/threat-research-blog/explain-like-i-m-5-remote-desktop-protocol-rdp>.

“rdpclip” is started when a new remote desktop session is created by the service which is called “Remote Desktop Services” - as shown in the screenshot below. Fun fact, the old display name of the service was “Terminal Services” which was changed while the service name is still “TermService”.



Lastly, the description of the service states “it allows users to connect interactively to a remote computer. Remote Desktop and Remote Desktop Session Host Server depend on this service. To prevent remote use of this computer, clear the checkboxes on the Remote tab of the System properties control panel item”.

³⁵ <https://www.winosbite.com/rdpclip-exe/>

³⁶ <https://learn.microsoft.com/en-us/windows-server/remote/remote-desktop-services/clients/remote-desktop-allow-access>

smartscreen.exe (Windows Defender SmartScreen)

“smartscreen.exe” is an executable which is the “Windows Defender SmartScreen”. The executable is located at “%windir%\System32\smartscreen.exe” (On 64 bit systems there is only a 64 bit version with no 32 bit version—in contrast to other executables such as cmd.exe).

SmartScreen is a cloud-based anti-phishing/anti-malware component which is included in different Microsoft products such as: Windows, Internet Explorer and Microsoft Edge (https://en.wikipedia.org/wiki/Microsoft_SmartScreen).

Microsoft Defender SmartScreen helps with determining whether a site is potentially malicious and by determining if a downloaded application/installer is potentially malicious. We can sum up the benefits of SmartScreen as follows: anti-phishing/anti-malware support, reputation-based URL/application protection, operating system integration, ease of management using group policy/Microsoft Intune and blocking URLs associated with potentially unwanted applications. (<https://learn.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-smartscreen/microsoft-defender-smartscreen-overview>).

In order to demonstrate the working of SmartScreen I have tried to download (using Edge) - you can see the warning in the left side of the screenshot below. Moreover, after downloading it using a different browser I have executed the EICAR test file - you can see the result in the left side of the screenshot below. By the way, the EICAR (European Institute from Computer Antivirus Research) test file was created to test the response of AV software (https://en.wikipedia.org/wiki/EICAR_test_file).

Lastly, we can enable/disable SmartScreen using the settings window, bot for the OS/browser (<https://www.digitalcitizen.life/how-disable-or-enable-smartscreen-filter-internet-explorer-or-windows-8/>).

