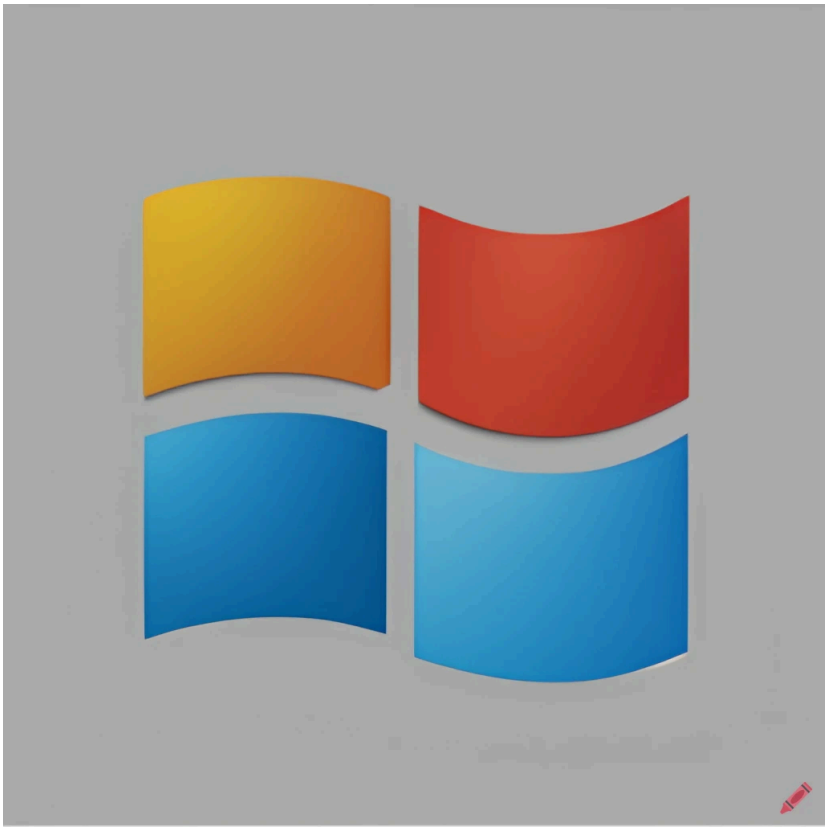
The Windows Process Journey

Version 13
January-2025

By Dr. Shlomi Boutnaru



Created using Craiyon, AI Image Generator

Table of Contents

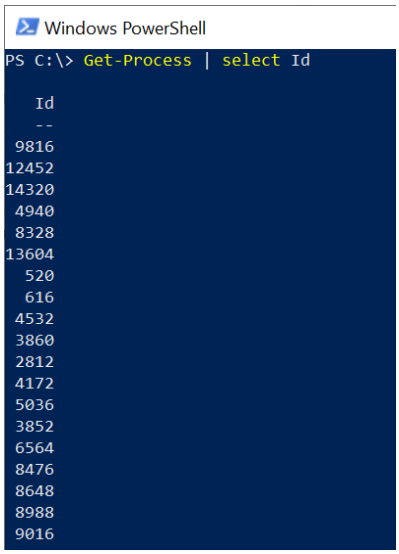
Introduction

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.



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.

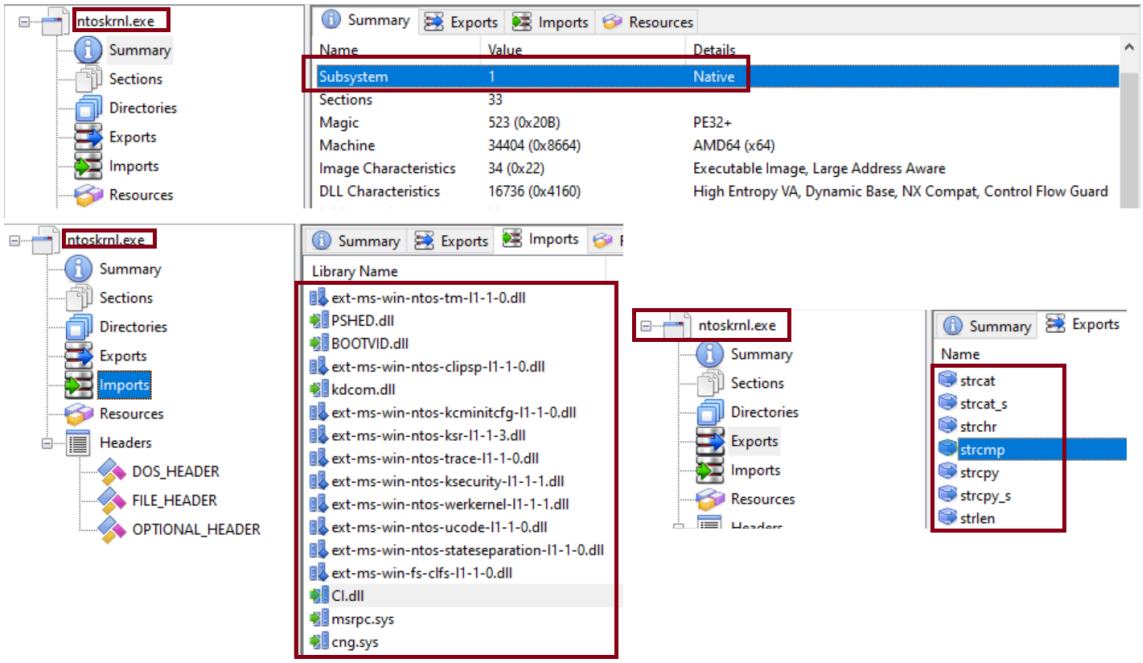
Lastly, you can follow me on twitter - @boutnaru (https://twitter.com/boutnaru). Also, you can read my other writeups on medium - https://medium.com/@boutnaru. Lastly, You can find my free eBooks at https://TheLearningJourneyEbooks.com. Lets GO!!!!!!

ntoskrnl.exe (NT Kernel & System)

In general, “ntoskrnl.exe” is the kernel image of the Windows operating system. It includes both the executive and the kernel layers of Windows NT, which are responsible for memory management, process handling and hardware abstraction. Also, “ntoskrnl.exe” contains the SRM (Security Reference Monitor), cache manager, scheduler and more[1].

Overall, although in the “Subsystem” field of the PE header “ntoskrnl.exe” is defined as “Native”, it is not linked with “ntdll.exe” as other user-mode native applications - as shown in the screenshot below which was taken using “PE Explorer”[2]. Due to that, “ntoskrnl.exe” needs a “static” copy of the C runtime (think about function like “strcmp”, “strcpy”, “strcpy_s”, “strlen” and more) - as shown in the screenshot below. For a reference implementation we can check out the ReactOS source code[3].

Lastly, the functions exported by “ntoskrnl.exe” have specific prefixes which indicate the component in which they are part of, for example: “Io” (I/O manager), Ke (core kernel routines), “Kd” (kernel debugger support functions), “Ldr” (PE image loader support functions), “Mm” (memory management), “Se” (security functions), “Ob” (object manager), “Hal” (hardware abstraction layer), “Ps” (process management functions), “Nls” (native language support) and more[4].



[1] https://en.wikipedia.org/wiki/Ntoskrnl.exe
[2] https://github.com/zodiacon/PEExplorerV2
[3] https://github.com/reactos/reactos/tree/master/ntoskrnl
[4] https://community.osr.com/t/meaning-of-the-function-prefices/21242

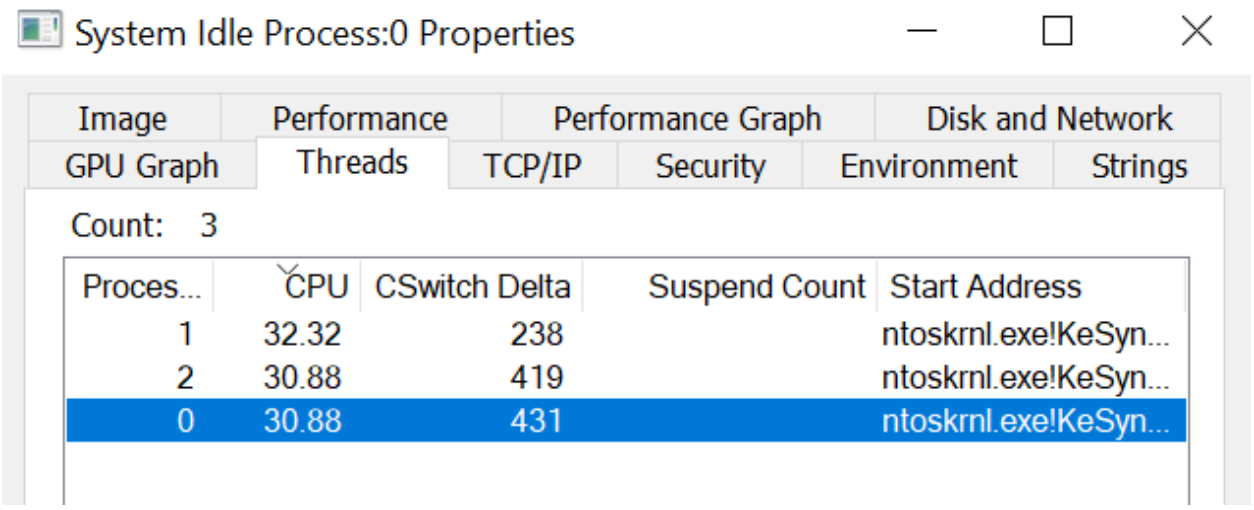
System Idle Process (PID 0)

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.



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.freetechsecrets.com/win32/WIN32DefineDosDevice.htm).

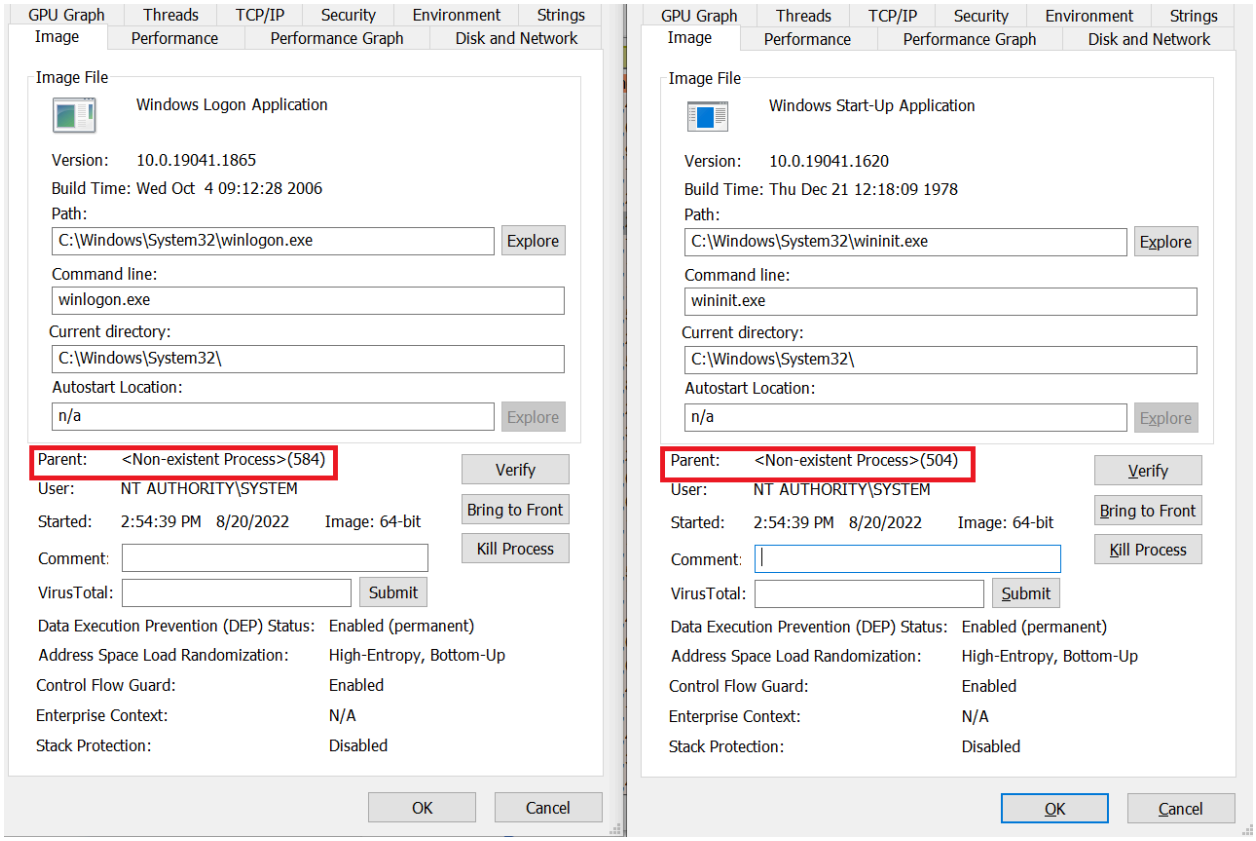
Loading the subsystems which are configured in the Registry - “HKLM\System\CurrentControlSet\Control\Session Manager\SubSystems”. At minimum we have have the kernel part of the Win32 Subsystem (aka win32k.sys) and on session 0, which is the session in which Windows’ services are executed - smss.exe starts “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”). 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).

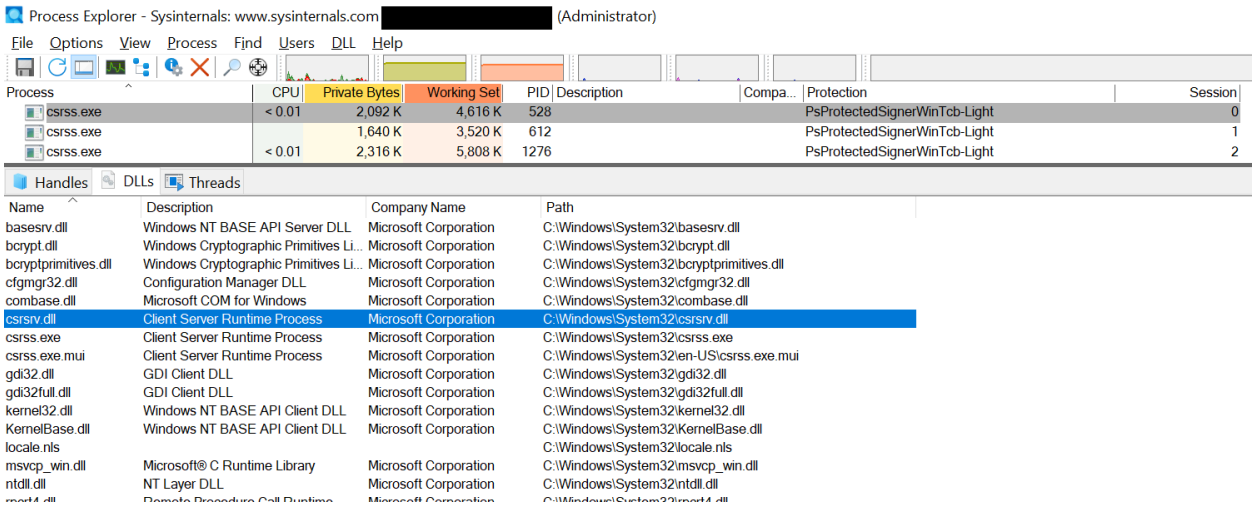
From Windows NT 4.0 most of the Win32 subsystem has been moved to kernel mode - "With this new release, the Window Manager, GDI, and related graphics device drivers have been moved to the Windows NT Executive running in kernel mode"[5]. Thus "csrss.exe" manages today GUI shutdowns and windows console (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 more[6] . 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 on 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 maintained[7], by the way we can leverage this fact for identifying hidden processes (like by using "psxview"[8] 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"[9]. 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 form kernel mode only[10]. A list of all the "csrss.exe" API list can be found here https://j00ru.vexillium.org/csrss_list/api_table.html.

[5] https://learn.microsoft.com/en-us/previous-versions//cc750820(v=technet.10)?redirectedfrom=MSDN#XSLTsection124121120120
[6] https://j00ru.vexillium.org/2010/07/windows-csrss-write-up-the-basics/
[7] https://www.geoffchappell.com/studies/windows/win32/csrsrv/api/process/process.htm
[8] https://github.com/volatilityfoundation/volatility/wiki/Command-Reference-Mal#psxview
[9] https://medium.com/@boutnaru/the-windows-process-journey-smss-exe-session-manager-subsystem-bca2cf748d33
[10] https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-csrss


Process Explorer - Sysinternals: www.sysinternals.com [redacted] (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
csrsrv.dll Client Server Runtime Process Microsoft Corporation C:\Windows\System32\csrsrv.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
msvcp_win.dll Microsoft® C Runtime Library Microsoft Corporation C:\Windows\System32\msvcp_win.dll
ntdll.dll NT Layer DLL Microsoft Corporation C:\Windows\System32\ntdll.dll

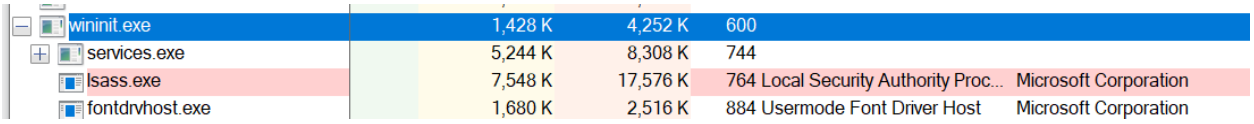
wininit.exe (Windows Start-Up Application)

“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 booting[11].

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 shutdown[12].

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.



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

[12] 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 logons[13].

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”[14], allowing the alteration of the ussr’s password and locking/unlocking the server/workstation[15]. 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”.

In its initialization phase “winlogon.exe” registers the “CTRL+ALT+DEL” secure attention sequence[16] 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”[17].

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”) as minimum and in some case another one which is for the next session that can be created - as seen in the screenshot below.

Lastly, I think it is a good idea to go over the reference implementation in ReactOS for “winlogon.exe”[18].

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

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

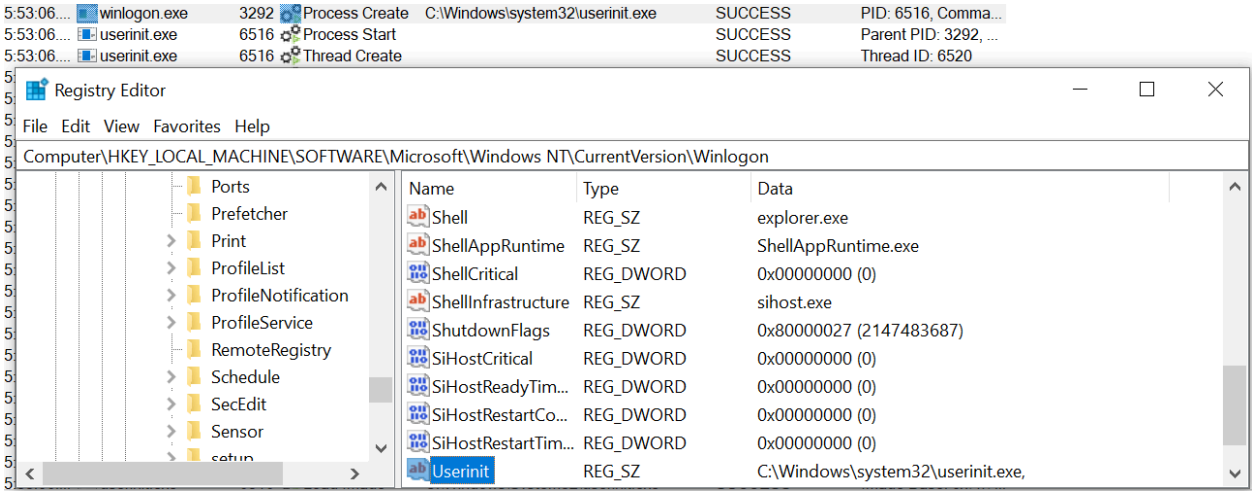
[13] https://learn.microsoft.com/en-us/windows/win32/secgloss/w-gly
[14] https://medium.com/@boutnaru/the-windows-process-journey-logonui-exe-windows-logon-user-interface-host-4b5b8b6417cb
[15] https://www.microsoftpressstore.com/articles/article.aspx?p=2228450&seqNum=8
[16] https://medium.com/@boutnaru/security-sas-secure-attention-sequence-da8766d859b5
[17] https://learn.microsoft.com/en-us/windows/win32/secauthn/initializing-winlogon
[18] https://github.com/reactos/reactos/tree/2752c42f0b472f2db873308787a8b474c4738393/base/system/winlogon

userinit.exe (Userinit Logon Application)

“userinit.exe” is an executable which is located executable is located at “%windir%\System32\userinit.exe“ (On 64 bit systems there is only a 64 bit 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 scripts[19].

“C:\Windows\System32\userinit.exe” is defined by default as the executable for the UserInit phase under the “userinit” key in the registry[20] - 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” key[21] - 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).

[19] https://www.minitool.com/news/userinit-exe.html
[20] HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\UserInit
[21] 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 more[22]. 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).

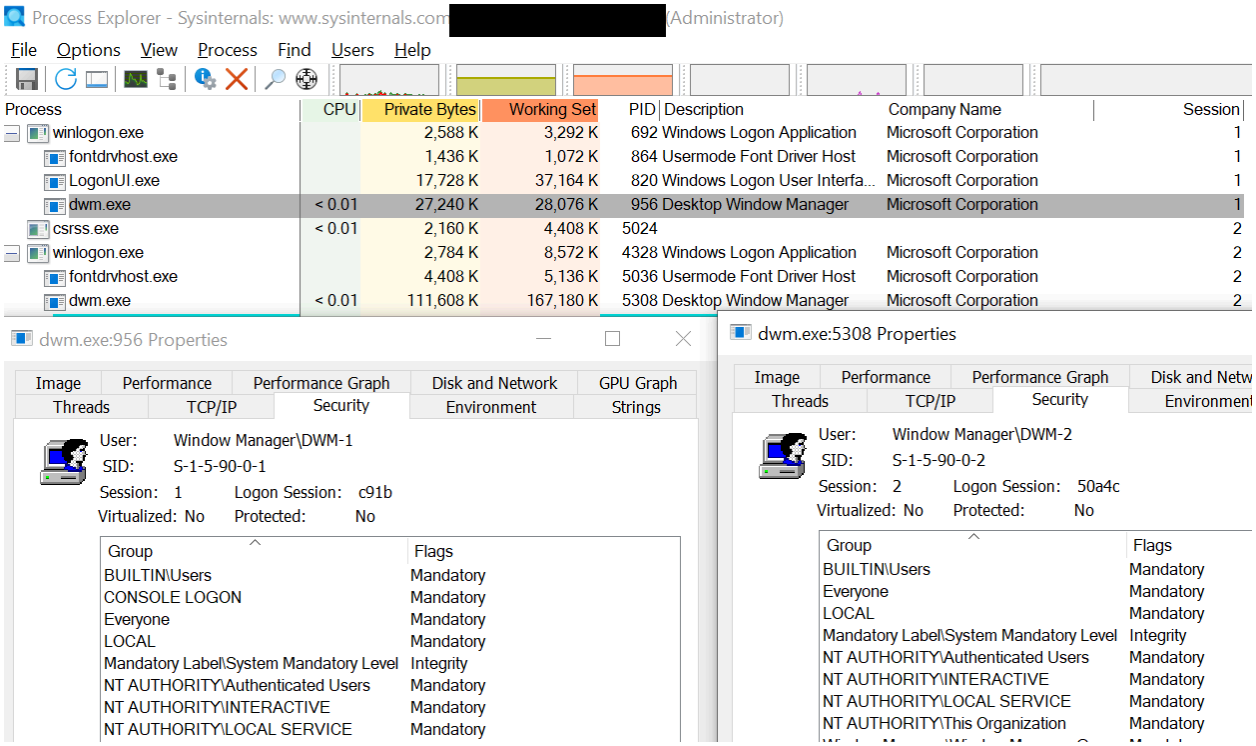
Thus, we can think about “dwm.exe” as a “compositing windows manager”. A “windows manager” is computer software that controls the placement and appearance of a window as part of a “window system” in a GUI environment[23]. So, a “compositing windows manager” is a “window manager” that provides applications 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 memory[24].

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).

[22] https://learn.microsoft.com/en-us/windows/win32/dwm/dwm-overview
[23] https://en.wikipedia.org/wiki/Window_manager
[24] https://en.wikipedia.org/wiki/Compositing_window_manager


Process Explorer - Sysinternals: www.sysinternals.com [redacted] (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 Interfa... 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
Image Performance Performance Graph Disk and Network GPU Graph
Threads TCP/IP Security Environment Strings
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
Image Performance Performance Graph Disk and Netw
Threads TCP/IP Security Environment
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

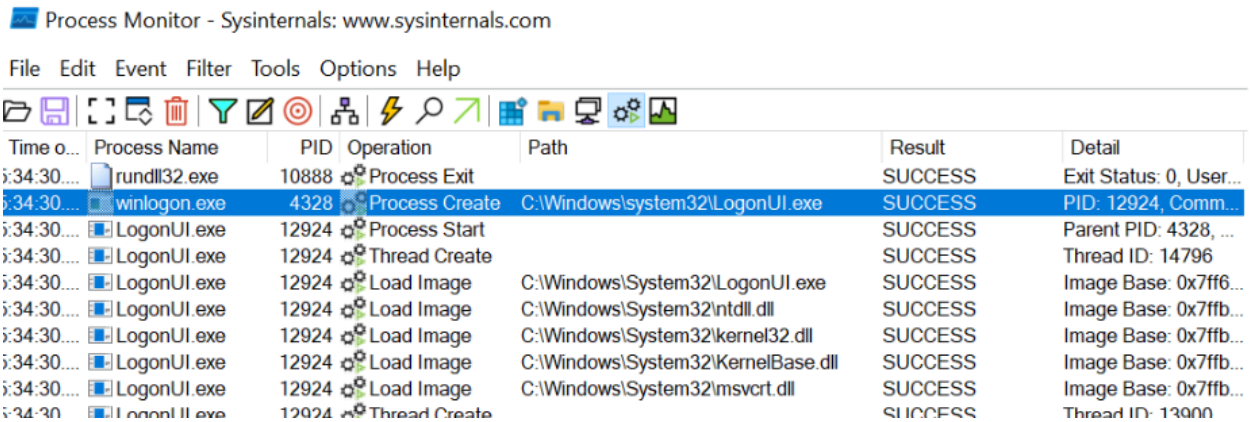
LogonUI.exe (Windows Logon User Interface Host)

"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 Monitor[25]. Also, if you want to see how "LogonUI.exe" GUI looks in different versions of Windows[26].

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 authenticated[27]. 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 ReactOS[28].



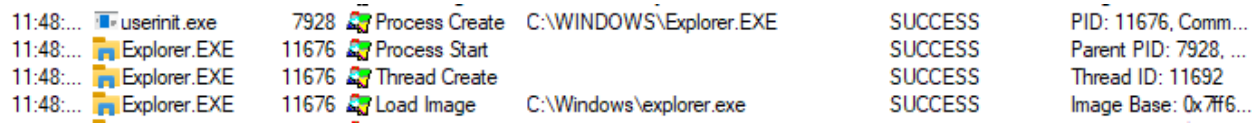
[25] https://learn.microsoft.com/en-us/sysinternals/downloads/procmon
[26] https://media.askvg.com/articles/images3/Windows_Login_Screen.png
[27] https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/credentials-processes-in-windows-authentication
[28] https://github.com/reactos/reactos/tree/3647f6a5eb633b52ef4bf1db6e43fc2b3fc72969/base/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 shell[29].

In case we terminate “explorer.exe” the taskbar will disappear and also the desktop both the shortcuts and the wallpaper itself[30]. For more understanding about “exeplorer.exe” I think it is a good idea to go over the reference implementation in ReactOS[31].

Every time a user logins interactively “explorer.exe” is executed under the user which logged on to the system[32]. 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.



I also suggest going over the following link https://ss64.com/nt/explorer.html to checkout all the arguments that can be passed to “exeplorer.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 faster[33].

[29] https://www.pcmag.com/encyclopedia/term/explorerexe
[30] https://copyprogramming.com/howto/what-happens-if-i-end-the-explorer-exe-process
[31] https://github.com/reactos/reactos/tree/81db5e1da884f76e6cee66b8cb1c7a2f6ff791eb/base/shell/explorer
[32] https://learn.microsoft.com/en-us/windows-server/security/windows-authentication/windows-logon-scenarios
[33] 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).

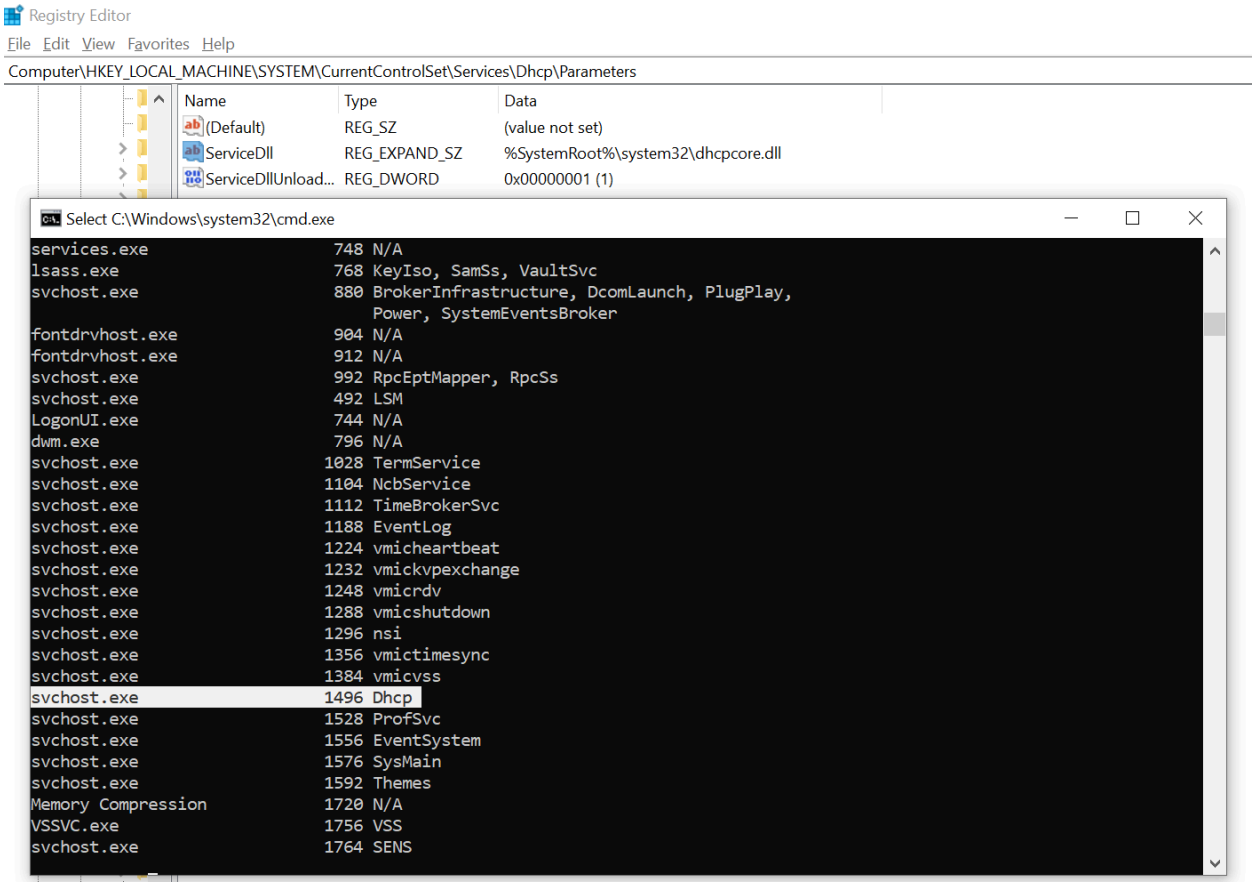
Due to the fact, many of the Windows’ services (you can read on Wndows’ Services on https://medium.com/@boutnaru/windows-services-part-2-7e2bdab5bce4) are implemented as DLLs (Dynamic Link Libraries) there is a need for an executable to host them. Thus, you can think about “svchost.exe” as the implementation 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 that[34].

[34] https://learn.microsoft.com/en-us/windows/application-management/svchost-service-refactoring

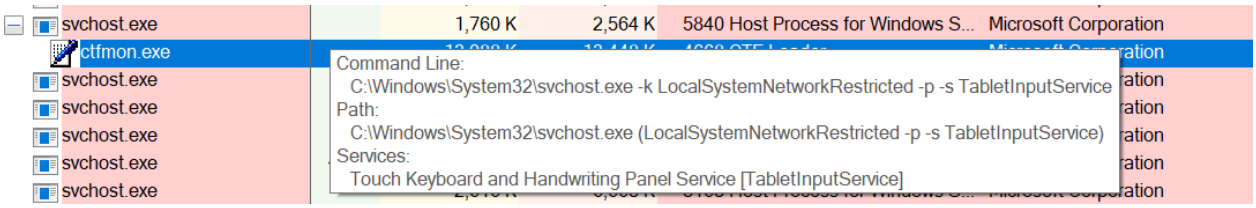

Registry Editor
File Edit View Favorites Help
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Dhcp\Parameters
Name
Type
Data
(Default)
REG_SZ
(value not set)
ServiceDll
REG_EXPAND_SZ
%SystemRoot%\system32\dhcpcore.dll
ServiceDllUnload...
REG_DWORD
0x00000001 (1)
Select C:\Windows\system32\cmd.exe
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.

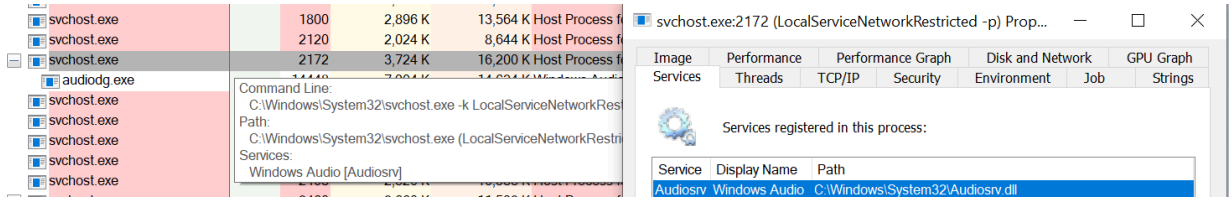


audiodg.exe (Windows Audio Device Graph Isolation)

“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”[35]. 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 modified[36].

Thus, we can say that “audiodg.exe” is being utilized for all audio processing[37]. 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”[38]. There should be one instance only of “audiodg.exe” at a specific time.



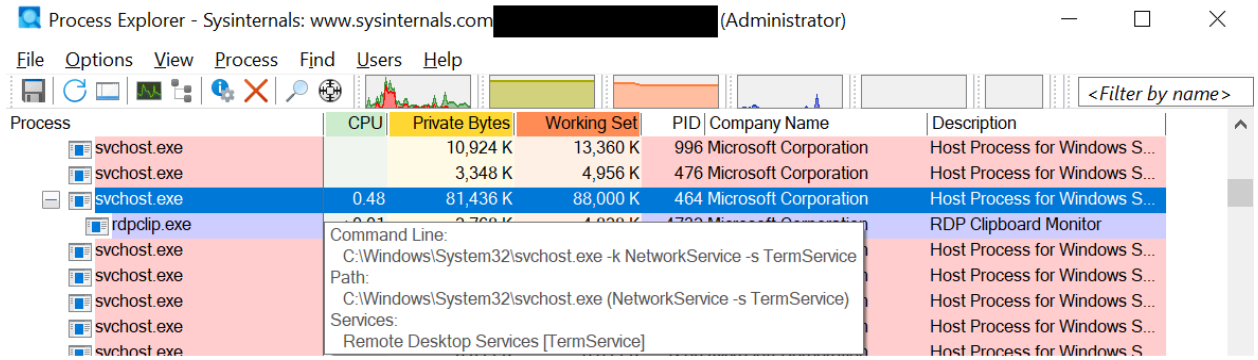
[35] https://learn.microsoft.com/en-us/windows-hardware/drivers/dashboard/audio-measures
[36] https://answers.microsoft.com/en-us/windows/forum/all/audiodgexe/0c86aef4-81a5-480e-9389-d9652fee1d21
[37] https://answers.microsoft.com/en-us/windows/forum/all/windows-10-audiodgexe/af1b70e0-06fe-4952-8205-b6191ccb8882
[38] 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 with[39]. 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” capability[40] 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”.

[39] https://www.winosbite.com/rdpclip-exe/
[40] https://learn.microsoft.com/en-us/windows-server/remote/remote-desktop-services/clients/remote-desktop-allow-access

smartscreen.exe (Windows Defender SmartScreen)

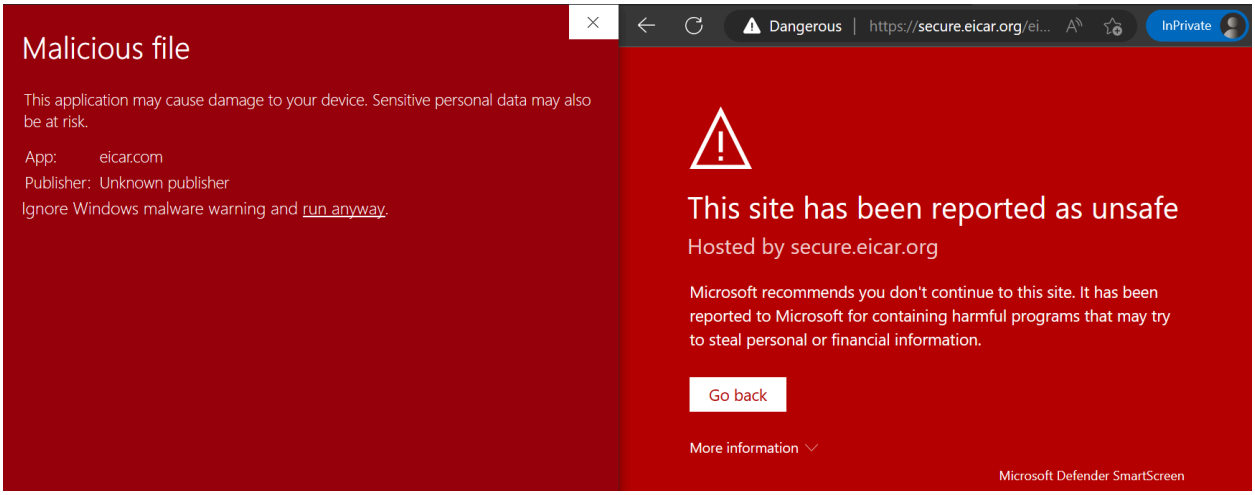
“smartscreen.exe” is an executable which is the “Windows Defender SmartScreen”. The executable is located at “%windir%\System32\smartscreen.exe” (On 64 bit systems there is only a 64 bit version with no 32 bit version—in contrast to other executables such as cmd.exe).

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/).

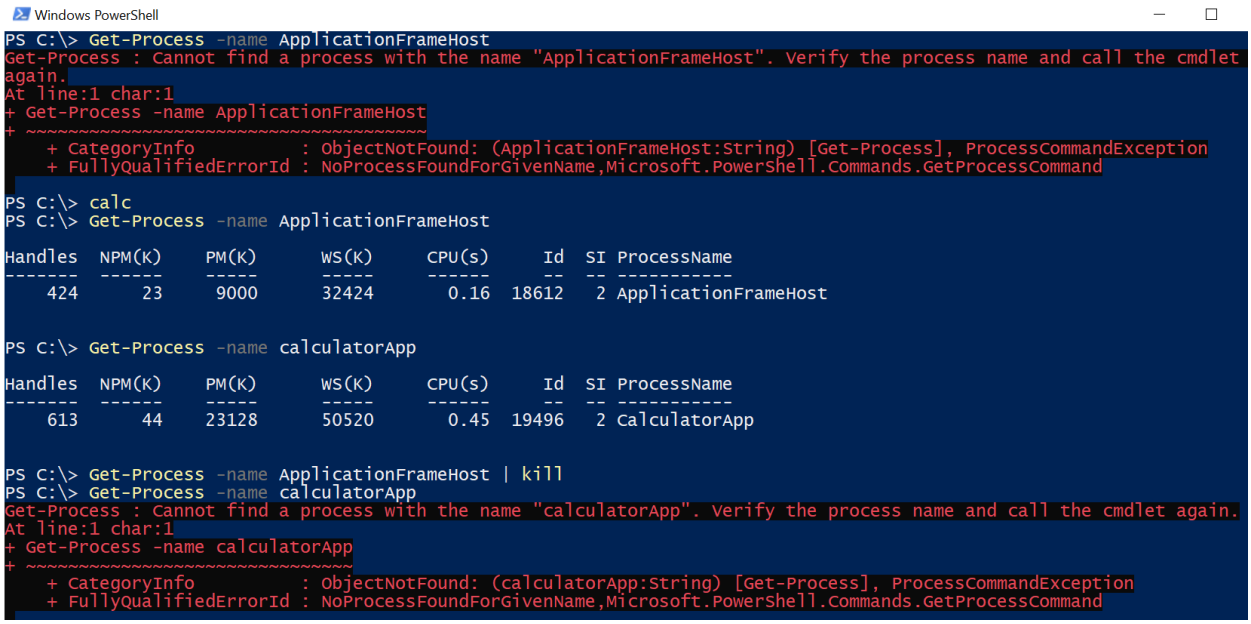


ApplicationFrameHost.exe

The "ApplicationFrameHost.exe" executable is located at the following directory - "%windir%\system32\ApplicationFrameHost.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.

Overall, the goal of "ApplicationFrameHost.exe" is to display the frames (windows) of the applications whether we are in desktop/tablet mode[41]. By the way, if we kill "ApplicationFrameHost.exe" all the UWP applications will be closed also - as we can see in the screenshot below.

There is one instance per session for the "ApplicationFrameHost.exe" in case one or more "Window Store App" which is also known as "Universal Windows Platform App"[42] - I will elaborate about them in a separate writeup. An example for a UWP app is the Calculator ("%windir%\system32\calc.exe"). Also, "ApplicationFrameHost.exe" is running with the permissions of the logged on user (that from whom the session was created).



[41] https://www.howtogeek.com/325127/what-is-application-frame-host-and-why-is-it-running-on-my-pc/
[42] https://www.file.net/process/applicationframehost.exe.html

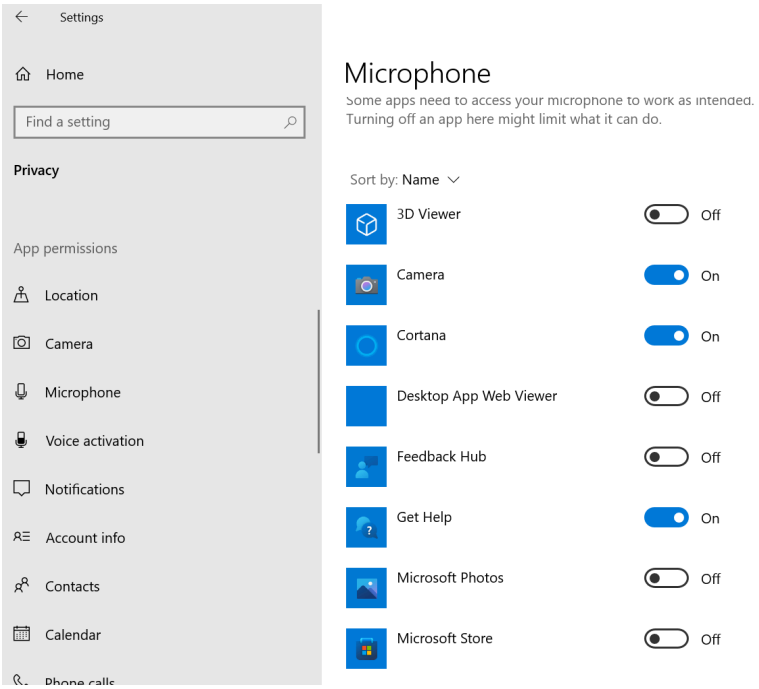
RuntimeBroker.exe

“RuntimeBroker.exe” is an executable which that is located at “%windir%\System32\RuntimeBroker.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).

“RuntimeBroker.exe” is running the permissions of the user (from whom the session was created). “RuntimeBroker.exe” is triggered from execution if the Windows Store is opened or any installed UWP app is started. By the way UWP apps are also known as Windows App/Windows Store App/Metro App[43].

Overall, “RuntimeBroker.exe” is responsible for managing the permissions for “Windows Store App”. We can think about it as a middleman between the application and operating system capabilities[44].

Thus, when an UWP application tries to access a specific OS resource “RuntimeBroker.exe” checks if the application has the appropriate permissions for that. In case it does not, “RuntimeBroker.exe” can ask the user to grant the permissions. We can modify the permissions for different applications using the “Settings” screen (Privacy->App permissions) - as shown in the screenshot below.

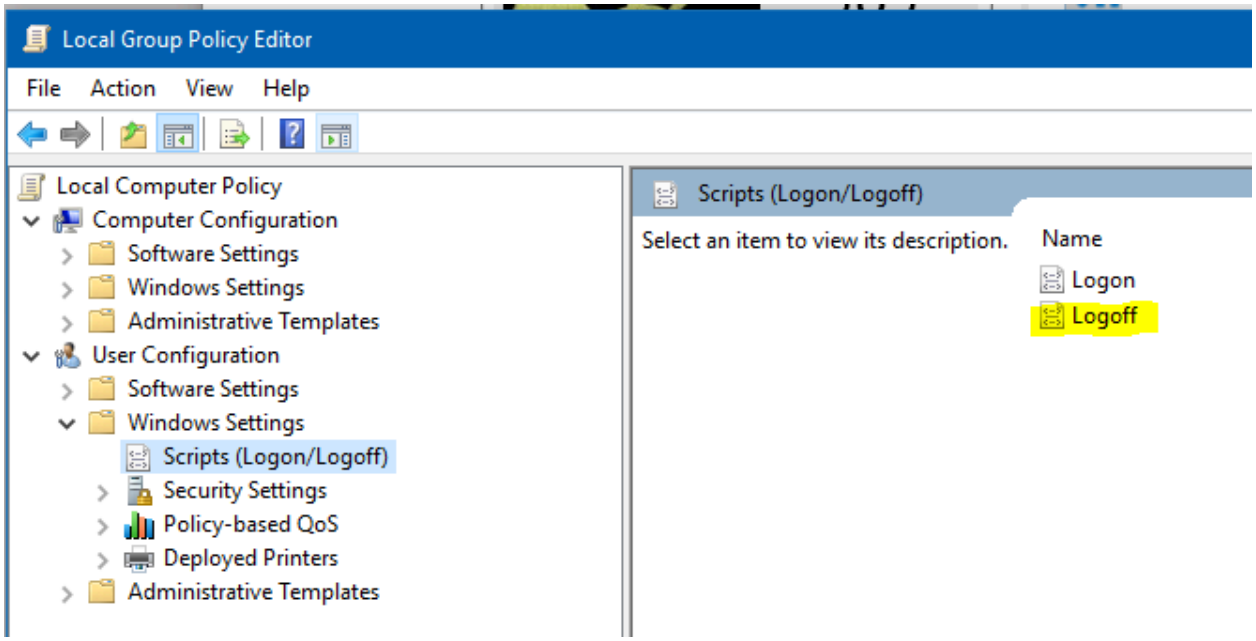


[43] https://www.file.net/process/runtimebroker.exe.html
[44] https://support.microsoft.com/en-us/windows/runtime-broker-is-using-too-much-memory-ca6ed4e3-2a36-964c-4d2e-8c93980d8a98

logoff.exe (Session Logoff Utility)

"logoff.exe" (Session Logoff Utility) is a command line tool that allows logging off a user from a session. The session could be the current session in which the command is executed, a specific session identified by a number or a remote session on a different server[45]. The executable file is located at "%windir%\System32\logoff.exe".

Moreover, an administrator can set a script/executable to be executed when the user is logging off. This setting can be configured using a local policy/group policy and is called "Logoff script). Alos, this configuration is part of the "User Configuration -> Windows Settings -> Scripts" - as shown in the screenshot below[46]. Lastly, we can also go over a reference code for "logoff.exe" from ReactOS[47].



[45] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/logoff
[46] https://social.technet.microsoft.com/Forums/en-US/f9f011e2-59fc-42d3-a1a4-251536ce8287/i-need-to-automatically-run-an-app-at-logoff?forum=win10itprosetup
[47] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/logoff

cscript.exe (Microsoft ® Console Based Script Host)

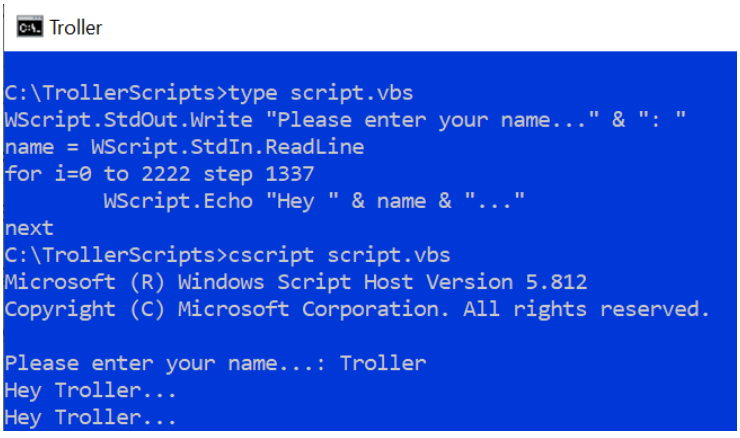
“cscript.exe” is the “Microsoft ® Console Based Script Host” which is a command line version of the “Windows Script Host”. It also allows setting script properties using command line options[48].

Also, “cscript.exe” is a PE binary file located at “%windir%\System32\cscript.exe”. On a 64-bit system (with a 64-bit OS installed) there is also a 32-bit based version located at “%windir%\SysWOW64\cscript.exe”.

Overall, the “Windows Script Host” (WSH) is an automation technology that enables scripting which was first introduced in Windows 95 (after build 950a) and became a standard component since Windows 98 (build 1111). It has support for different language engines, by default it supports JScript (*.js/*.jse) and VBScript (*.vbs/*.vbe) out of the box[49].

Moreover, users can also install other scripting engines for WSH like Perl and Python . By using WSH we can also leverage COM (). In VBScript we can do so by calling CreateObject() and in JSCript we can use an ActivexObject or call WSCript.CreateObject()[50].

When using “cscript.exe” to run a script to run in a command-line environment we don’t have to use administrator permissions. Alos, “cscript.exe” has multiple command line options for different usages like: interactive mode, debugging mode, passing arguments to the script and more[51]. Lastly, in order to demonstrate the usage of “cscript.exe” I have created a simple script and executed it - as shown in the screenshot below. We can also go over a reference implementation of “cscript.exe” for RactOS[52].



[48] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490887(v=technet.10)?redirectedfrom=MSDN
[49] https://en.wikipedia.org/wiki/Windows_Script_Host
[50] https://learn.microsoft.com/vi-vn/windows/win32/com/using-com-objects-in-windows-script-host
[51] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cscript
[52] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/cmdutils/cscript

wscript.exe (Microsoft ® Windows Based Script Host)

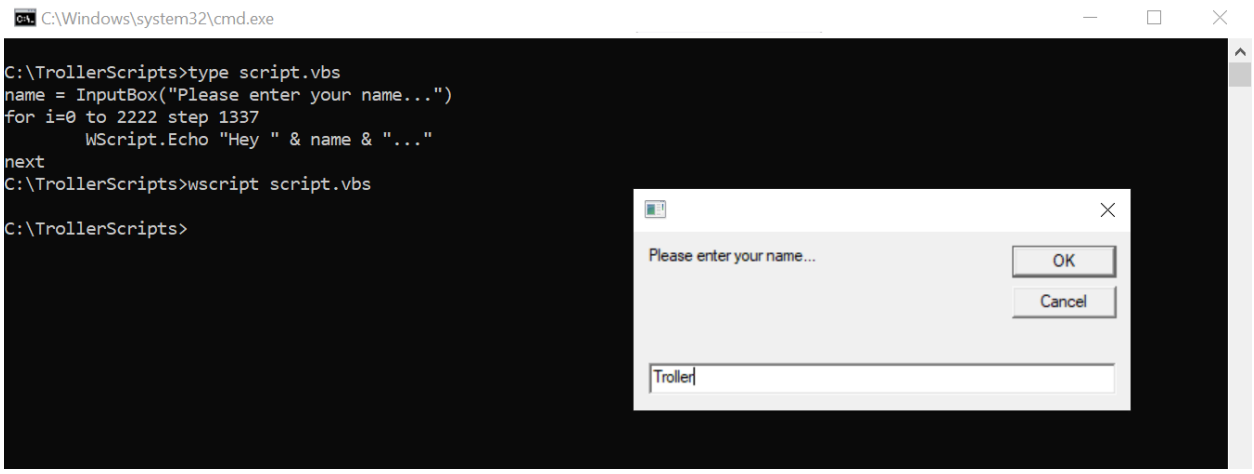
“wscript.exe” is the “Microsoft ® Windows Based Script Host” which provides an environment for executing scripts in a variety of languages[53]. It also allows setting script properties using command line options[54].

Overall, the “Windows Script Host” (WSH) is an automation technology that enables scripting which was first introduced in Windows 95 (after build 950a) and became a standard component since Windows 98 (build 1111). It has support for different language engines, by default it supports JScript (*.js/*.jse) and VBScript (*.vbs/*.vbe) out of the box[55].

Also, “wscript.exe” is a PE binary file located at “%windir%\System32\wscript.exe”. On a 64-bit system (with a 64-bit OS installed) there is also a 32-bit based version located at “%windir%\SysWOW64\wscript.exe”.

“wscript.exe” allows running the scripts in GUI mode in contrast to “cscript” which is CLI mode[56]. Gui mode means that graphical components could be displayed as the script is being executed - as shown in the screenshot below.

Lastly, in case you want to see a reference implementation of “wscript.exe” I suggest going over the implementation which is part of ReactOS[57].



[53] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh875526(v=ws.11)
[54] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/wscript
[55] https://en.wikipedia.org/wiki/Windows_Script_Host
[56] https://medium.com/@boutnaru/the-windows-process-journey-cscript-exe-microsoft-console-based-script-host-5878ba9354a0
[57] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/cmdutils/wscript

utilman.exe (Utility Manager)

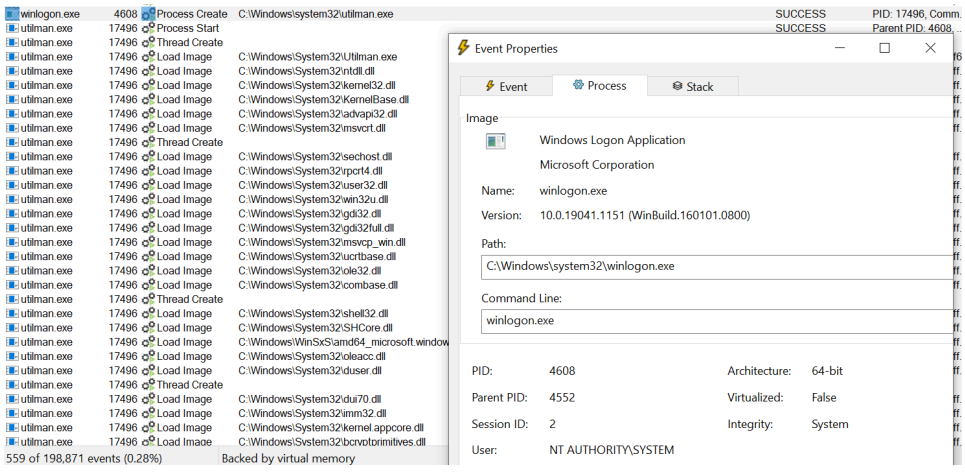
“utilman.exe” is the “Utility Manager” which is a PE binary file located at “%windir%\System32\utilman.exe”. On 64-bit systems there is also a 32-bit version located on “%windir%\SysWOW64\utilman.exe”.

Overall, “utilman.exe” can be started by clicking the icon of “Ease of Access” or by using the keyboard shortcut “WinKey+U”. When using one of those methods while the computer is locked, “utilman.exe” is started by “winlogon.exe” with the permissions of the “LocalSystem” - as shown in the screenshot below. By the way, due to the high level of permissions in use replacing “utilman.exe” is a common trick in order to reset the administrator password in Windows[58].

Moreover, “utilman.exe” allows accessing the following capabilities: narrator, magnifier, onscreen keyboard, high contrast, sticky keys and filter keys. Narrator is the screen reading application made for blind/visually impaired users[59]. Magnifier is an application that allows users to enlarge the screen content[60].

Also, sticky keys allows users to use modifier keys (like Ctrl, Shift, Alt and WinKey) without the need of pressing them constantly[61].. Filter keys is a feature that adjusts the keyboard response and ignores repeated keystrokes caused by inaccurate or slow finger movements[62].

Lastly, in case you want to see a reference implementation of “osk.exe” I suggest going over the implementation which is part of ReactOS[63].



[58] https://learn.microsoft.com/en-us/answers/questions/187973/windows-recovery-cmd
[59] https://support.microsoft.com/en-us/windows/complete-guide-to-narrator-e4397a0d-ef4f-b386-d8ae-c172f109bdb1
[60] https://support.microsoft.com/en-us/windows/use-magnifier-to-make-things-on-the-screen-easier-to-see-414948ba-8b1c-d3bd-8615-0e5e32204198
[61] https://geekflare.com/using-sticky-keys-in-windows/
[62] https://helpdeskgeek.com/how-to/what-are-filter-keys-and-how-to-turn-them-off-in-windows/
[63] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/utilman

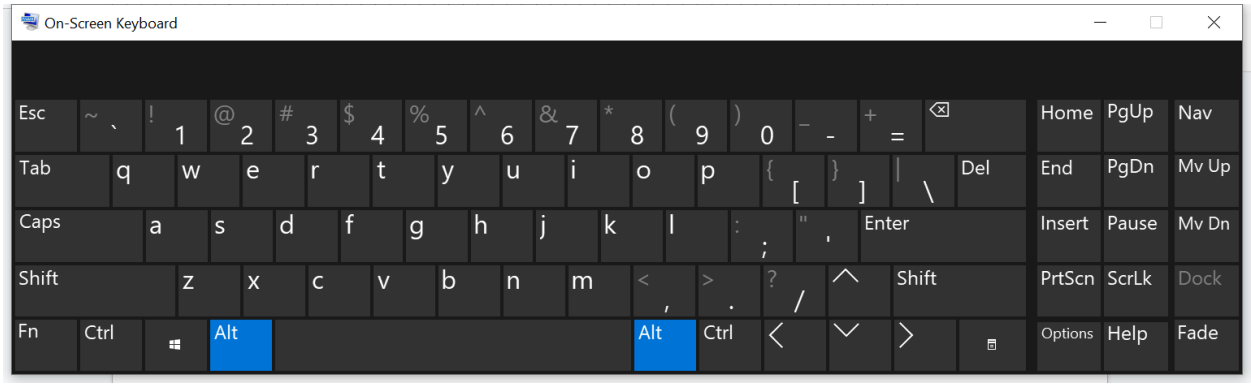
osk.exe (Accessibility On-Screen Keyboard)

“osk.exe” is the “Accessibility On-Screen Keyboard” which presents a virtual keyboard layout inside a resizable window - as shown in the screenshot below. The virtual keyboards enable the user clicking/hovering/scanning using a mouse/joystick in order to select/activate keys[64].

Moreover, “osk.exe” has a 101/102/106 key layout. “osk.exe” is a PE binary located at “%windir%\System32\osk.exe”. It is bundled with Windows and can provide some features for users with limited mobility[65].

Thus, we don’t need a touch screen in order to interact with “osk.exe”[66]. By the way, “osk.exe” is not the only virtual keyboard available as part of Windows, there is also “TabTip.exe” - but more on there is a separate writeup.

Lastly, in case you want to see a reference implementation of “osk.exe” I suggest going over the implementation which is part of ReactOS[67].



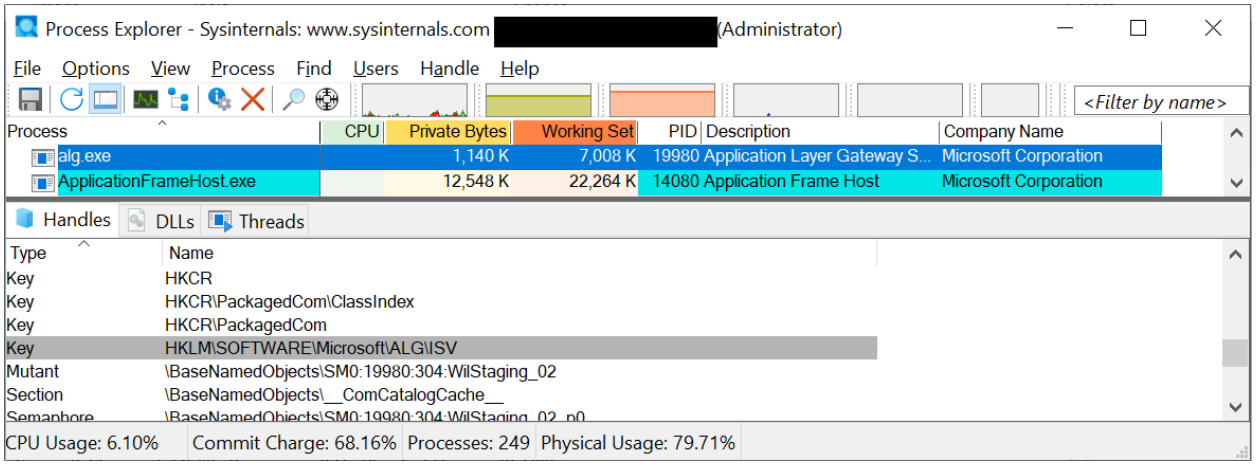
[64] https://www.file.net/process/osk.exe.html
[65] https://www.processlibrary.com/en/directory/files/osk/21965/
[66] https://support.microsoft.com/en-us/windows/use-the-on-screen-keyboard-osk-to-type-ecbb5e08-5b4e-d8c8-f794-81dbf896267a
[67] https://github.com/reactos/reactos/tree/47f3a4e144b897da0e0e8cb08c2909645061dec9/base/applications/osk

alg.exe (Application Layer Gateway Service)

“alg.exe” is the “Application Layer Gateway Service” (ALG) which is configured as a Windows service. Based on the description of the service it provides support for 3rd party protocol plug-ins for Internet Connection Sharing (ICS). The service is executed with the permission of the “LocalService” user. “alg.exe” is a PE binary which is stored in the following location: “%windir%\System32\alg.exe”.

Generally, an “Application Layer Gateway” (ALG) allows a gateway to parse payloads and take actions such as allow/drop/other based on the data contained in the payloads[68]. Thus, ALG’s plugins can modify data in packets, think about things like IP addresses and port numbers[69].

Lastly, “alg.exe” is started by “services.exe” with the permission of “NT AUTHORITY\LOCAL SERVICE” user. There should be at most only one instance of “alg.exe”. “alg.exe” parses information about supported plugins from “HKLM\SOFTWARE\Microsoft\ALG\ISV”[70]. We can see in the screenshot below that there is a handle to that registry location.



[68] https://www.juniper.net/documentation/us/en/software/junos/alg/alg.pdf
[69] https://en.wikipedia.org/wiki/Application-level_gateway
[70] https://www.sigma-uk.net/tech/windows_ftp_alg_iis

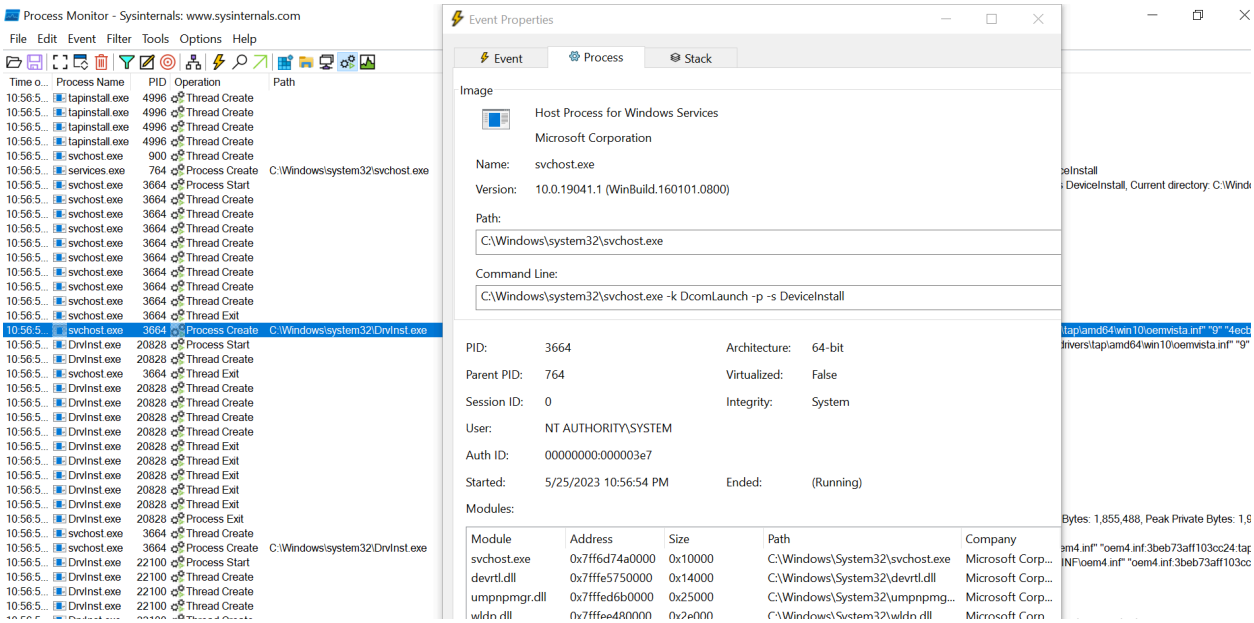
DrvInst.exe (Driver Installation Module)

“DrvInst.exe” is a PE executable located at “%windir%\System32\drvinst.exe”, it is known as “Driver Installation Module”. Since Windows Vista when PnP (Plug and Play) manager detects a new device “DrvInst.exe” is started. It is used for searching and installing the relevant driver for the new device detected[71].

“DrvInst.exe” can also be used for installing drivers while installing a software package. Let us take for example the installation of “OpenVPN Connect”[72].

Thus, as with most VPN (Virtual Private Network) solutions there is a need to install a TAP driver, which is a virtual network device[73]. This causes “services.exe” to launch a new process using the following arguments “C:\Windows\system32\svchost.exe -k DcomLaunch -p -s DeviceInstall”, which is part of the “DCOM Server Process Launcher”. It is executed with the permission of the “LocalSystem” user.

Moreover, by passing as an argument “DeviceInstall” “svchost.exe” loads “%windir%\System32\umpnpmgr.dll”, which is the “User-mode Plug-and-Play Service”. This instance of “svchost.exe” is the one that starts “DrvInst.exe”. It also loads “%windir%\System32\devrtl.dll” (Device Management Run Time Library) - as shown in the screenshot below.



[71] https://learn.microsoft.com/en-us/windows-hardware/drivers/install/debugging-device-installations-with-a-user-mode-debugger
[72] https://openvpn.net/client/
[73] https://www.techradar.com/vpn/what-is-a-tap-adapter

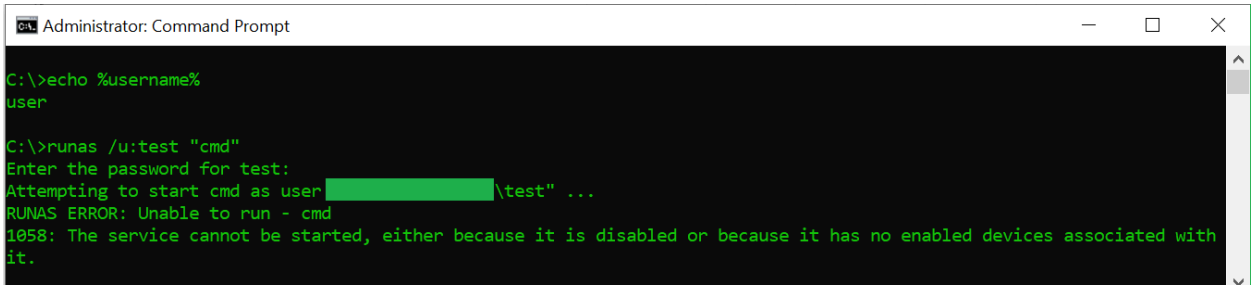
runas.exe (Run As Utility)

“runas.exe” is an executable aka “Run As Utility”, which is located at “%windir%\System32\runsas.exe”. On 64 bit systems there there is also a 32-bit version located at “%windir%\SysWow64\runas.exe”.

Overall, “runas.exe” allows a user to execute specific programs/tools with different permissions than the logged-on user. “runas.exe” also has multiple parameters that can be used like passing credentials from a smartcard instead of a password, loading the user’s profile and more[74].

Moreover, “runas.exe” is dependent on the “Secondary Logon” service. The description of the service states that it “enables starting processes under alternate credentials. If this service is stopped, this type of logon access will be unavailable. If this service is disabled, any services that explicitly depend on it will fail to start”. As described if the service is disabled “runas.exe” will fail - as shown in the screenshot below.

Thus, in case the “Secondary Logon” service can be started it is done with the following command line: “%windir%\system32\svchost.exe -k netsvcs -p -s seclogon” with the permissions of the “Local System” user. Also, in this case “svchost.exe” will load “%windir%\System32\seclogon.dll” (Secondary Logon Service DLL).



[74] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc771525(v=ws.11)

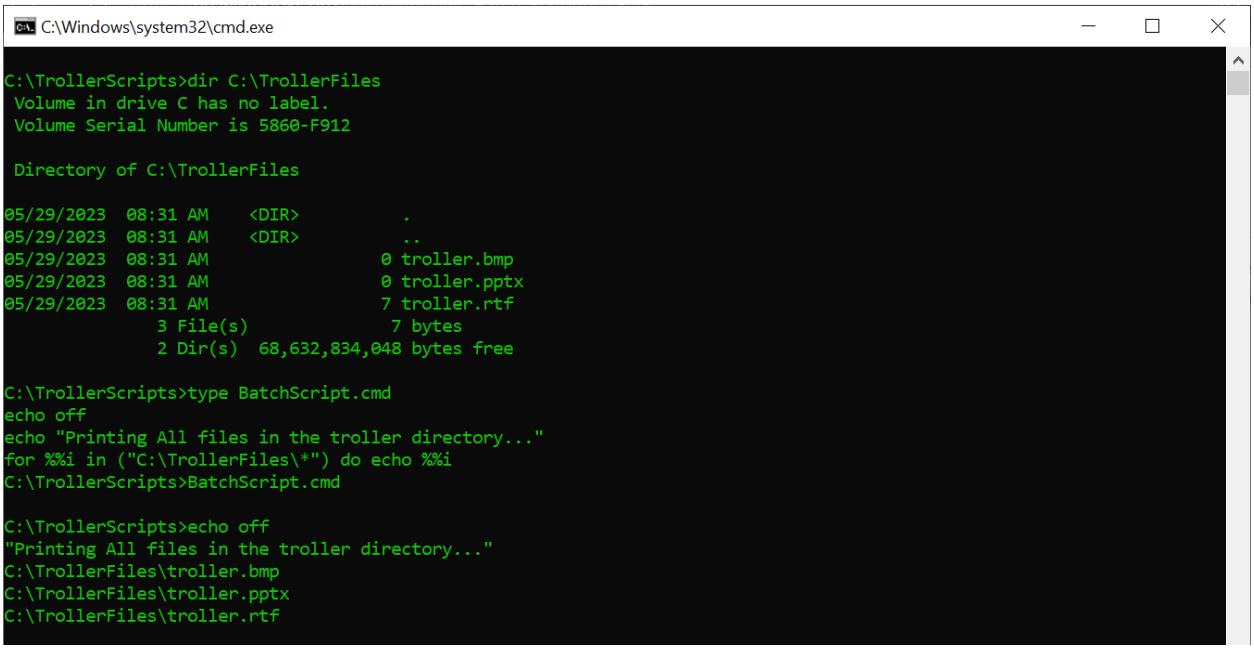
cmd.exe (Windows Command Processor)

“cmd.exe” is the “Windows Command Processor” which is the default CLI (command line interface/interpreter) of Windows (and also reactOS). By the way, it is also known as “Command Prompt”. It is the replacement of “command.com” which was relevant from MS-DOS to Windows XP. In Windows NT/Windows 2000 and Windows XP there was both “cmd.exe” and “command.com”[75].

The executable is located at “%windir%\System32\cmd.exe”. On 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\cmd.exe”. Also, “cmd.exe” allows the execution of any script/executable installed on the system or one of the internal command which included as part of “cmd.exe” like: “cd”, “copy” and “md”[76].

Moreover, “cmd.exe” supports executing batch scripts - as shown in the screenshot below. I suggest going through “Windows Batch Scripting” for more information[77].

Lastly, for a reference of “cmd.exe” I suggest going over the implementation of “cmd.exe” as part of ReacOS[78].



[75] https://www.computerhope.com/cmd.htm
[76] https://wishmesh.com/2014/09/ms-dos-cmd-exe-command-prompt-cd-md-copy/
[77] https://en.wikibooks.org/wiki/Windows_Batch_Scripting
[78] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/shell/cmd

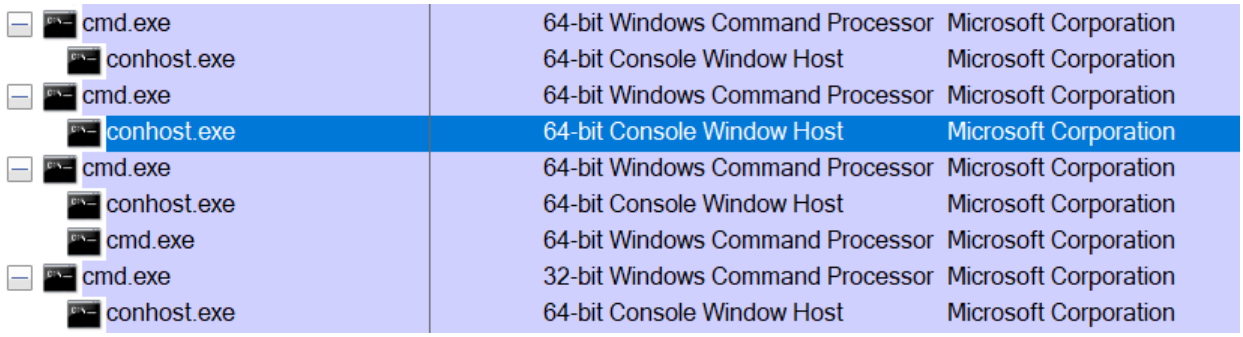
conhost.exe (Console Window Host)

“conhost.exe” is an executable aka the “Console Window Host”, which is located at “%windir%\System32\conhost.exe”. The goal of “conhost.exe” is to provide an interface between “cmd.exe”[79] and “explorer.exe”[80].

Thus, “conhost.exe” is both the server application (for Windows Console API) and also the classic Windows user interface for working with CLI (command line interface) application. Historically, those were the job of “csrss.exe”[81] but they were extracted for isolation and security reasons[82].

Moreover, one of the duties of “conhost.exe” is to provide the ability to “drag and drop” folders/files into “cmd.exe”. By the way, every 3rd party application can use “conhost.exe”[83]. When “conhost.exe” is started with the permissions of the user which “cmd.exe” was started with.

Lastly, we can have multiple instances of “conhost.exe”. For each instance of “cmd.exe” (which is not a descendant of another “cmd.exe”) there will be an instance of “conhost.exe”. Also, in case of a 64-bit system even if a 32-bit “cmd.exe” an instance of a 64-bit “conhost.exe” is going to be started. A demonstration of those points is shown in the screenshot below (taken using “Process Explorer”).



[79] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[80] https://medium.com/@boutnaru/the-windows-process-journey-explorer-exe-windows-explorer-9a96bc79e183
[81] https://medium.com/@boutnaru/the-windows-process-journey-csrss-exe-client-server-runtime-subsystem-cb5fa34c47db
[82] https://learn.microsoft.com/en-us/windows/console/definitions
[83] https://www.lifewire.com/conhost-exe-4158039

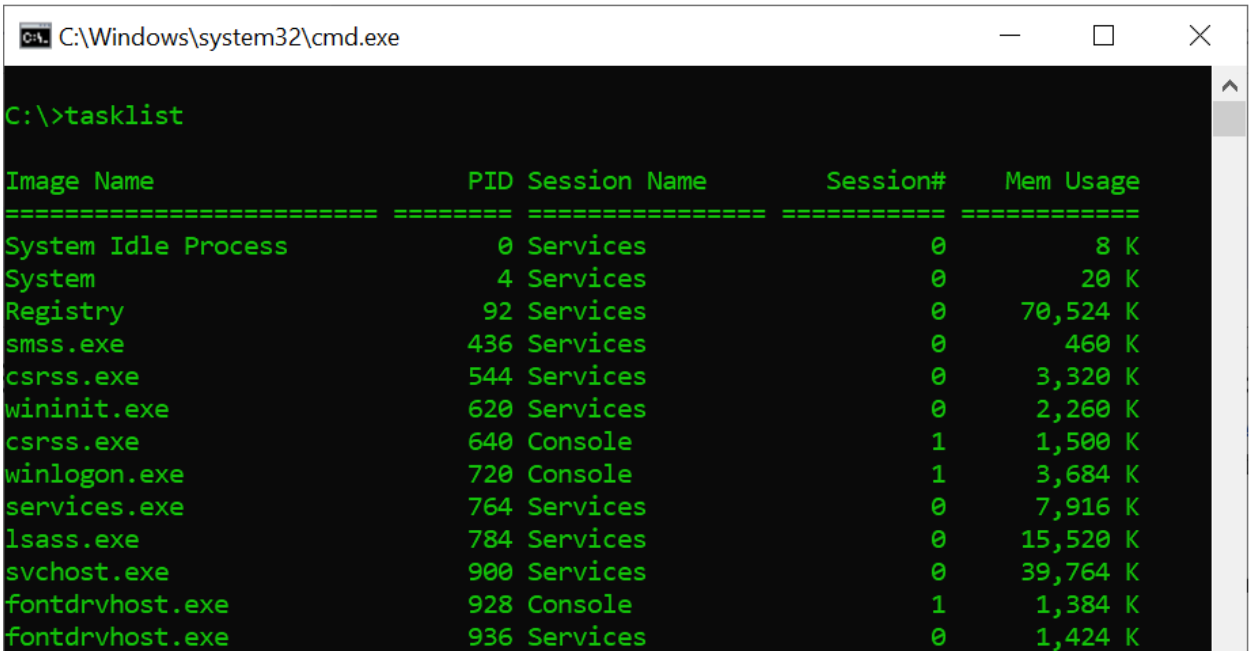
tasklist.exe (Lists the Current Running Tasks)

“tasklist.exe” is an executable which is located at “%windir%\System32\tasklist.exe”. It allows displaying the list of currently running processes on the system[84]. On 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\tasklist.exe”.

Moreover, a user with sufficient permissions can also list the processes of a remote system using “tasklist.exe” by using the “/s” command line switch. For more information about the other switches which are available please refer to https://ss64.com/nt/tasklist.html.

Overall, a user can display the following attributes for each displayed process: image name, pid, session number, session name, cpu time, memory usage, user name, service name (if relevant), window title (if relevant) and more.

Lastly, for a reference of “cmd.exe” I suggest going over the implementation of “cmd.exe” as part of ReacOS[85].



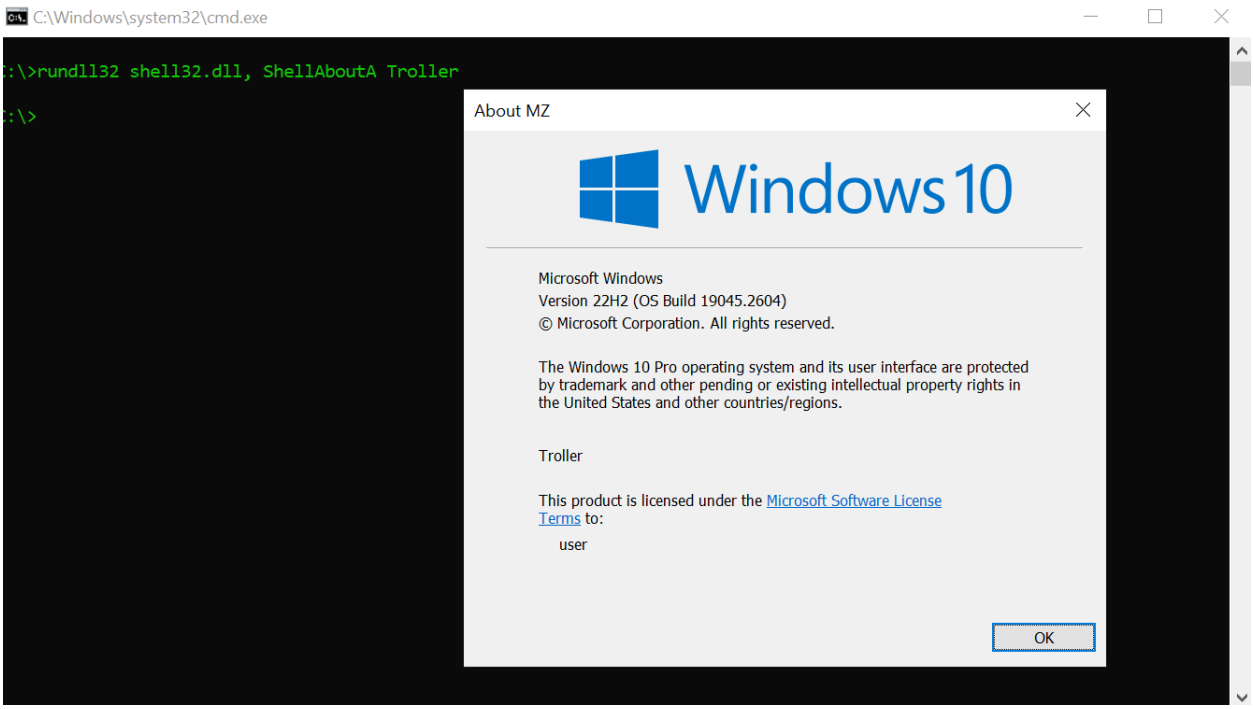
[84] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc730909(v=ws.11)
[85] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/cmdutils/tasklist

rundll32.exe (Windows Host Process)

"rundll32.exe" is an executable aka the "Windows Host Process" (based on the description field of the PE file), which is located at "%windir%\System32\rundll32.exe". On a 64 bit-system the file still has the same name (including the number 32) and a 32-bit version is located at "%windir%\SysWOW64\rundll32.exe".

Overall, the goal of "rundll32.exe" is to load a DLLs (Dynamic Link Libraries) and run a functionality stored in those files[86]. The DLLs are loaded using "LoadLibraryExW"[87]. "rundll32.exe" is digitally signed by Microsoft and shipped by default with the operating system. By the way, there are also places that say "rundll32.exe" means "Run a DLL as an App"[88].

The way is which we can call a function from a "*.dll" file is by passing the name of the file and the name of the function. We can also pass arguments to a function while using "rundll32.exe"[89]. An example of using "rundll32.exe" is shown in the screenshot below. Also, for more examples of using "rundll32.exe" I suggest going over the following link https://www.thewindowsclub.com/rundll32-shortcut-commands-windows. Lastly, for an implementation reference of "rundll32.exe" I suggest going over the one in ReacOS[90].



[86] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/rundll32
[87] https://www.cybereason.com/blog/rundll32-the-infamous-proxy-for-executing-malicious-code
[88] https://www.file.net/process/rundll32.exe.html
[89] https://stmxcsr.com/micro/rundll-parse-args.html
[90] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/system/rundll32

net.exe (Network Command)

“net.exe” is the “Net Command” which is a command line that allows managing different aspects of the operating system such as: users, groups, services and network connections[91]. Also, “net.exe” is a PE binary file located at “%windir%\System32\net.exe” which is signed by Microsoft. On 64-bit based versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\net.exe.

Overall, they are 19 sub commands in net: “accounts”, “computer”, “config”, “continue”, “file”, “group”, “help”, “helpmsg”, localgroup”, “pause”, “session”, “share”, “start”, “statistics”, “stop”, “time”, “use”, “user” and “view”. By using “net help” we can get an explanation about each sub command. In the table below I have gathered a short description for each sub command (excluding “net help”). Lastly, we can also go over a reference implementation of “net.exe” from ReacOS[92].

net command	description
net accounts	updates the user accounts database and modifies password and logon requirements for all accounts
net computer	adds or deletes computers from a domain database
net config	displays configuration information of the Workstation or Server service
net continue	reactivates a Windows service that has been suspended by “net pause”
net file	closes a shared file and removes file locks. When used without options, it lists the open files on a server.
net group	adds, displays, or modifies global groups on servers (used on an AD environment)
net helpmsg	displays information about Windows network messages (such as error, warning, and alert messages)
net localgroup	modifies local groups on computers. When used without options, it displays the local groups on the computer
net pause	suspends a Windows service or resource. Pausing a service puts it on hold
net session	lists or disconnects sessions between the computer and other computers on the network. When used without options, it displays information about all sessions with the computer of current focus
net share	makes a server's resources available to network users. When used without options, it lists information about all resources being shared on the computer
net start	lists running services, also can start a specific service
net statistics	Display s the statistics log for the local Workstation service
net stop	Stopping a service cancels any network connections the service is using
net time	synchronizes the computer's clock with that of another computer or domain, or displays the time for a computer or domain. When used without options displays the current date and time at the computer
net use	connects a computer to a shared resource or disconnects a computer from a shared resource. When used without options, it lists the computer's connections
net user	creates and modifies user accounts on computers. When used without switches, it lists the user accounts for the computer
net view	displays a list of resources being shared on a computer. When used without options, it displays a list of computers in the current domain or Network

[91] https://attack.mitre.org/software/S0039/
[92] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/network/net

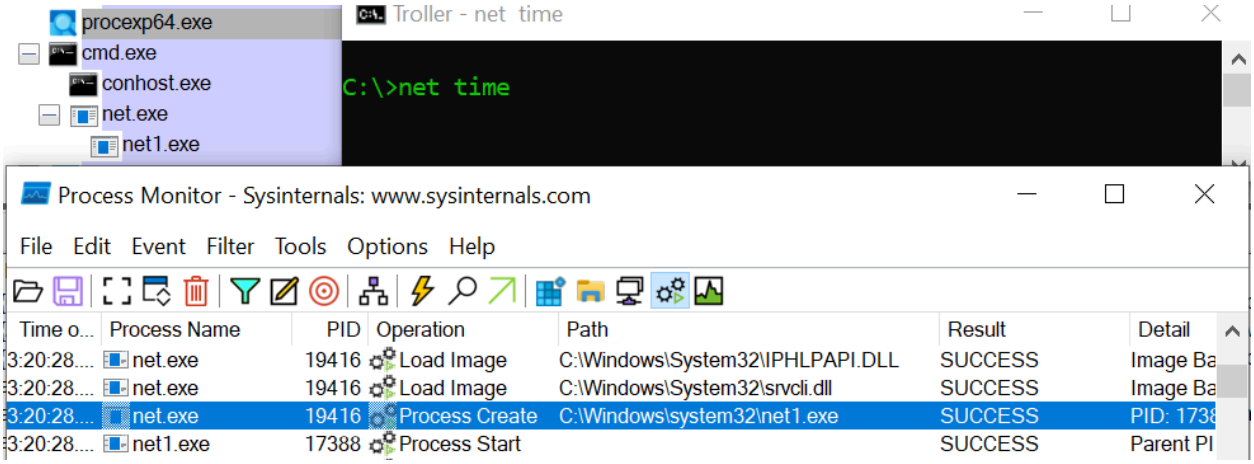
net1.exe (Net Command for the 21st Century)

"net1.exe" is known as the "Net Command for the 21st Century"[93]. It is a PE binary file that is signed by Microsoft, which is located at "%windir%\system32\net1.exe". On 64-bit versions of Windows there is also a 32-bit version of the file located at "%windir%\SysWOW64\net1.exe".

Overall, the "net1.exe" was created as a temporary fix for the Y2K problem that affected "net.exe"[94]. There was an issue while using the command "net user [USERNAME] /times" which is responsible for configuring the logon hours of the user[95].

Thus, "net1.exe" is executed for specific functionality when "net.exe" is run[96]. For example when calling "net time" an instance of "net1.exe" is started by "net.exe" using the command "net1 time" - as seen in the screenshot below.

Lastly, "net1.exe" supports every command the "net.exe" supports. The issue with "net.exe" was corrected in Windows XP, however "net1.exe" is still available today for backward compatibility with old scripts that might use it[97].



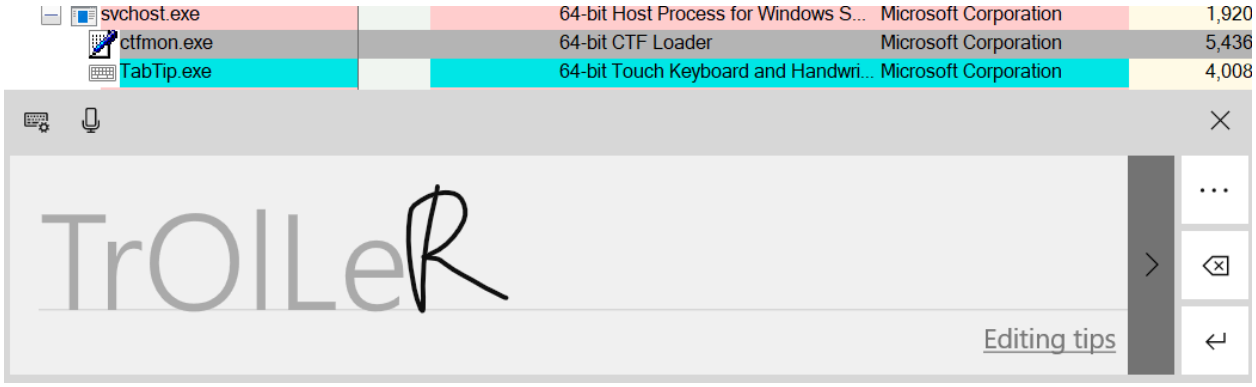
[93] https://www.file.net/process/net1.exe.html
[94] https://www.lifewire.com/net-command-2618094
[95] https://web.archive.org/web/20140830150320/http://support.microsoft.com/kb/240195
[96] https://attack.mitre.org/software/S0039/
[97] https://ss64.com/nt/net.html

TabTip.exe (Touch Keyboard and Handwriting Panel)

"TabTip.exe" (Touch Keyboard and Handwriting Panel) is also known as "Tablet Text Input Panel". It is an interface developed by Microsoft which allows inputting text in different ways: handwriting to text, speech to text and by clicking on the screen like a keyboard[98].

The usage of "TabTip.exe" as a keyboard is very similar to "osk.exe"[99]. The main goal of "TabTip.exe" is to provide handwriting input. This means that even applications that don't have this capability can use "TabTip.exe" to provide users with the ability of writing instead of typing[100] - as shown in the screenshot below.

Overall, "TabTip.exe" is a 64-bit PE binary located at "%ProgramFiles%\Common Files\microsoft shared\ink\TabTip.exe", which is digitally signed by Microsoft. When "TabTip.exe" is launched it is started as a child process of the service TabletInputService (Touch Keyboard and Handwriting Panel Service), similar to "ctfmon.exe"[101] - as shown in the screenshot below. This service is hosted by "svchost.exe"[102], which loads the "%windir%\System32\TabSvc.dll".



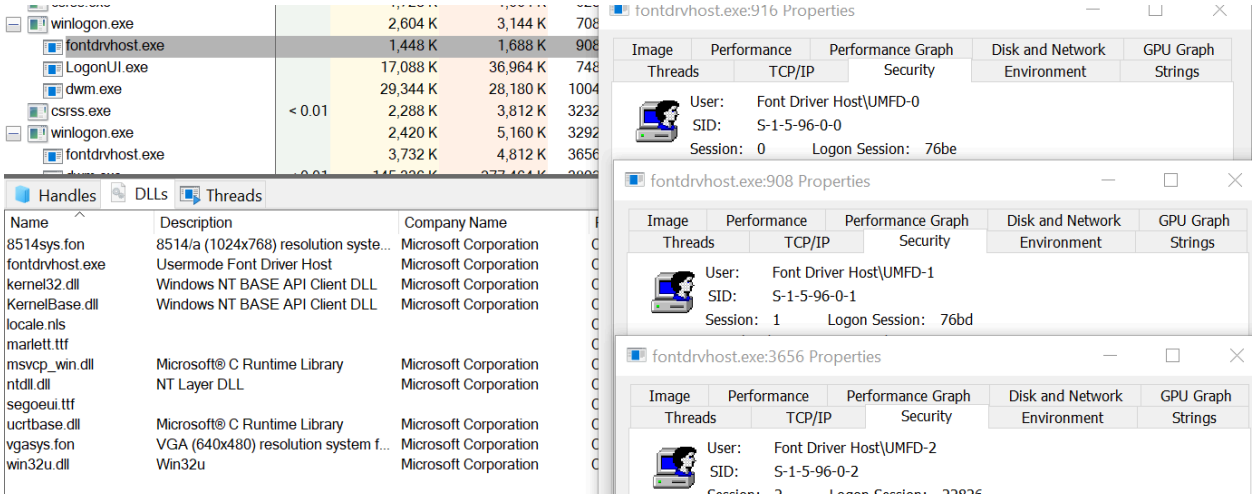
[98] https://www.file.net/process/tabtip.exe.html
[99] https://medium.com/@boutnaru/the-windows-process-journey-osk-exe-accessibility-on-screen-keyboard-72823695321e
[100] https://windowsreport.com/tabtip-exe/
[101] https://medium.com/@boutnaru/the-windows-process-journey-ctfmon-exe-ctf-loader-148f10f5401
[102] https://medium.com/@boutnaru/the-windows-process-journey-svchost-exe-host-process-for-windows-services-b18c65f7073f

fontdrvhost.exe (Usermode Font Driver Host)

On Windows 8.1 (and previous versions) the parsing of fonts takes place in a kernel driver (atmfd.dll, yes they are Dlls which are executed in kernel mode). This was accessible via graphical syscalls exported by win32k.sys, thus it created an attack surface that could lead to privilege escalation. Thus, from Windows 10 the parsing code was moved to the restricted user-mode process "fontdrvhost.exe"[103]

Overall, "fontdrvhost.exe" is an executable which is located at "%windir%\System32\fontdrvhost.exe" (On 64-bit systems there is also a 32-bit located at "%windir%\SysWOW64\fontdrvhost.exe"). It is executed with the permissions of a user in the following pattern: "Font Driver Host\UMFD[SessionID]". Also, the SID of the user is in the pattern of "S-1-5-96-[SessionID]" - as you can see in the screenshot below. Also, "fontdrvhost.exe" is a PE binary that is digitally signed by Microsoft.

Moreover, on session 0 "fontdrvhost.exe" is started by "wininit.exe"[104], in the following sessions (1, 2 , etc) it is started by "winlogon.exe"[105]. Thus, the number of instances of "fontdrvhost.exe" should be as the number of opened sessions on the Windows system. Lastly, UMDF stands for "User Mode Font Driver".



[103] https://googleprojectzero.blogspot.com/2021/01/in-wild-series-windows-exploits.html
[104]https://medium.com/@boutnaru/the-windows-process-journey-wininit-exe-windows-start-up-application-5581bfe6a01e
[105]https://medium.com/@boutnaru/the-windows-process-journey-winlogon-exe-windows-logon-application-88a1d4d3e13c

OpenWith.exe (Pick an App)

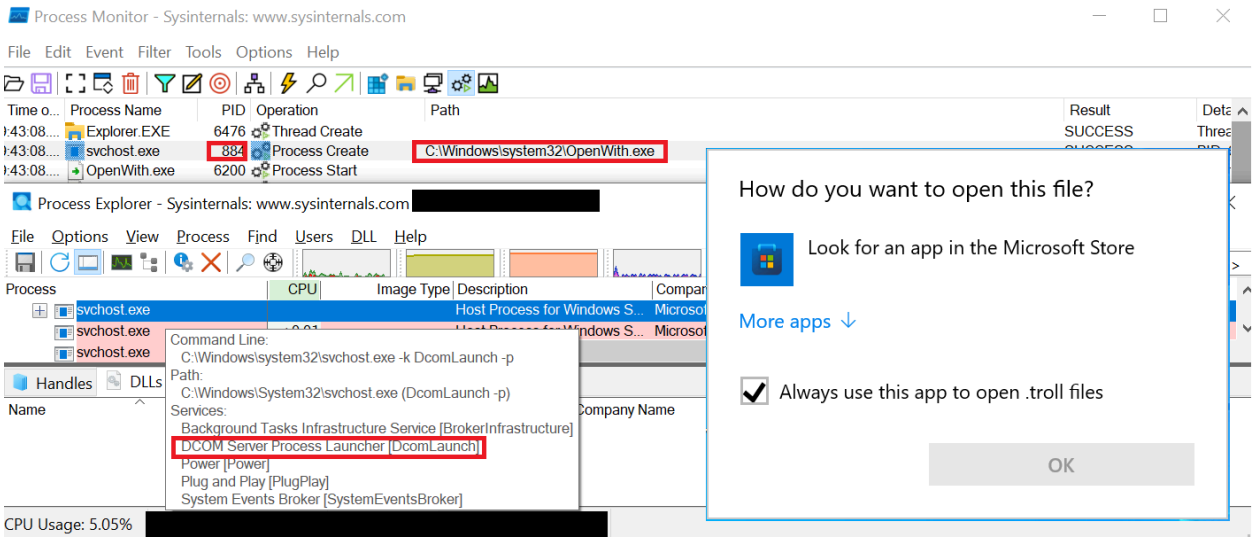
"OpenWith.exe" is also known as the "Pick an App", it is located at "%windir%\System32\OpenWith.exe" and it is digitally signed by Microsoft. On 64-bit systems there is also a 32-bit version located at "%windir%\SysWOW64\OpenWith.exe".

Overall, "OpenWith.exe" is used for selecting the application we want to open a file with a specific extension - as shown in the screenshot below. You might expect that "exlorer.exe" is going to start "OpenWith.exe", however it is done by the "DCOM Server Process Launcher" service which is hosted by "svchost.exe"[106] - as shown in the screenshot below.

Moreover, due to the reason the hosting "svchost.exe" is running with the permissions of the "LocalSystem" the creation of the "OpenWith.exe" process is done using the API "CreateProcessWithUserW"[107]. It allows "svchost.exe" to execute "OpenWith.exe" with the permissions of the logged on user (the same access token as "explorer.exe").

At the end, when we select an app the next time a double click is identified "explorer.exe"[108] is going to start an instance of the application associated with the extension and pass as an argument the full path of the app.

Thus, if we associate "%windir%\system32\notepad.exe" with "*.troll" a double click on "troller.troll" leads to the following command line to be executed: ""C:\Windows\system32\NOTEPAD.EXE" C:\Users\[USERNAME]\Desktop\troller.trl".



[106] https://medium.com/@boutnaru/the-windows-process-journey-svchost-exe-host-process-for-windows-services-b18c65f7073f
[107] https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessasuserw
[108] https://medium.com/@boutnaru/the-windows-process-journey-explorer-exe-windows-explorer-9a96bc79e183

mavinject.exe (Microsoft Application Virtualization Injector)

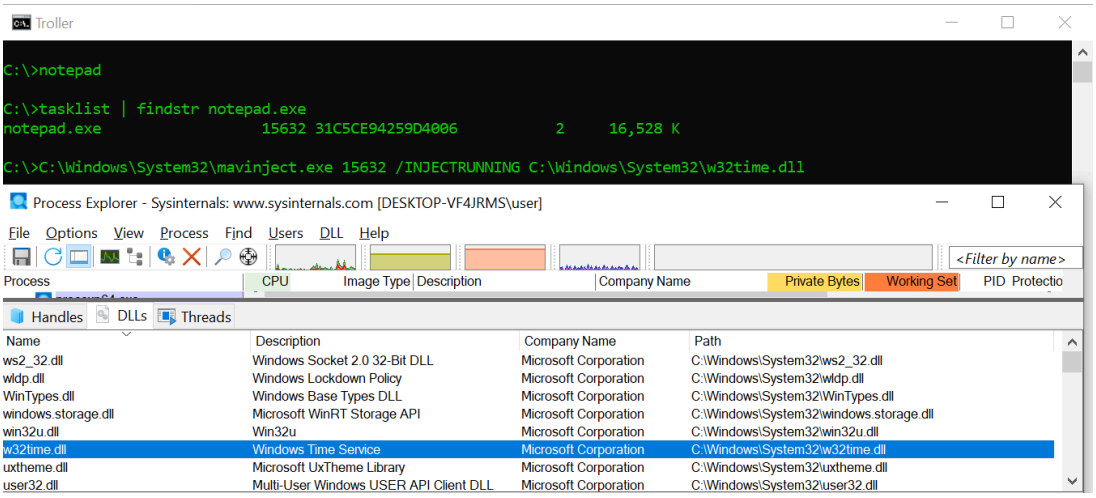
"mavinject.exe" is the "Microsoft Application Virtualization Injector" which is part of App-V (Microsoft Application Virtualization). App-V allows the delivering of applications to users as "virtual applications". This means that "virtual applications" are installed on a central managed server. They are "streamed" to users as a service as they are needed. From the user's perspective it acts as an installed application locally[109].

Overall, "mavinject.exe" is a PE binary located at "%windir%\System32\mavinject.exe", which is digitally signed by Microsoft. In case of a 64-bit system there is also a 32-bit version located at "%windir%\SysWOW64\mavinject.exe".

Moreover, using "mavinject.exe" we can perform DLL injection, meaning loading a DLL in the address space of a different process. In order to do so we need to run "mavinject.exe" with different arguments like: "mavinject.exe [PID] /INJECTRUNNING [PATH_TO_DLL_TO_LOAD]" - as shown in the screenshot below.

Also, there are other arguments that can be used "/HMODULE" which allows import descriptor injection. We can use it in the following manner: "mavinject.exe PID /HMODULE=BASE_ADDRESS PATH_DLL ORDINAL_NUMBER"[110].

Lastly, "mavinject.exe" uses the following Win32 API calls: VirtualProtectEx, CreateRemoteThread, VirtualAllocEx, OpenProcess, LoadLibraryW and WriteProcessMemory[111].



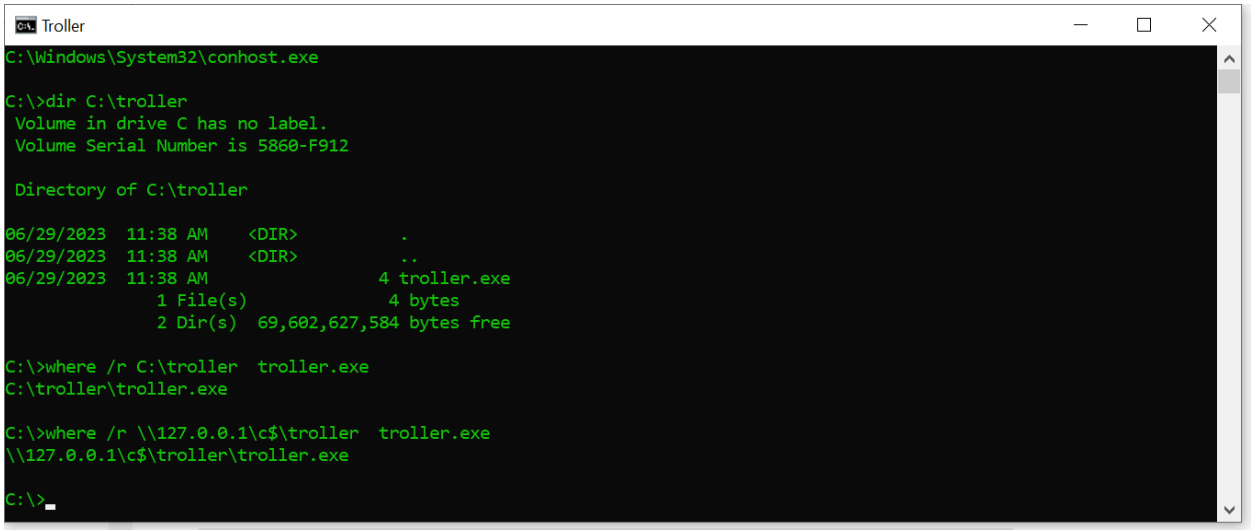
[109] https://learn.microsoft.com/en-us/windows/application-management/app-v/appv-about-appv
[110] https://unprotect.it/technique/system-binary-proxy-execution-mavinject/
[111] https://posts.specterops.io/mavinject-exe-functionality-deconstructed-c29ab2cf5c0e

where.exe (Lists location of Files)

“where.exe” (List Location of Files) is responsible for displaying the location of files which match a specific search pattern. The search is done in the current directory and in the path which are declared as part of the “PATH” environment variable[112]. It is equivalent to the “which” command under Linux[113].

Overall, “where.exe” is a PE binary file located at “%windir%\System32\where.exe”. On 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\where.exe”. Also, the file is digitally signed by Microsoft.

Moreover, we can use “where.exe” to search in subdirectories from a specific location using the “/r” switch. We can also perform the search remotely by specifying a UNC path[114] - as shown in the screenshot below.



[112] https://ss64.com/nt/where.html
[113] https://linux.die.net/man/1/which
[114] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/where

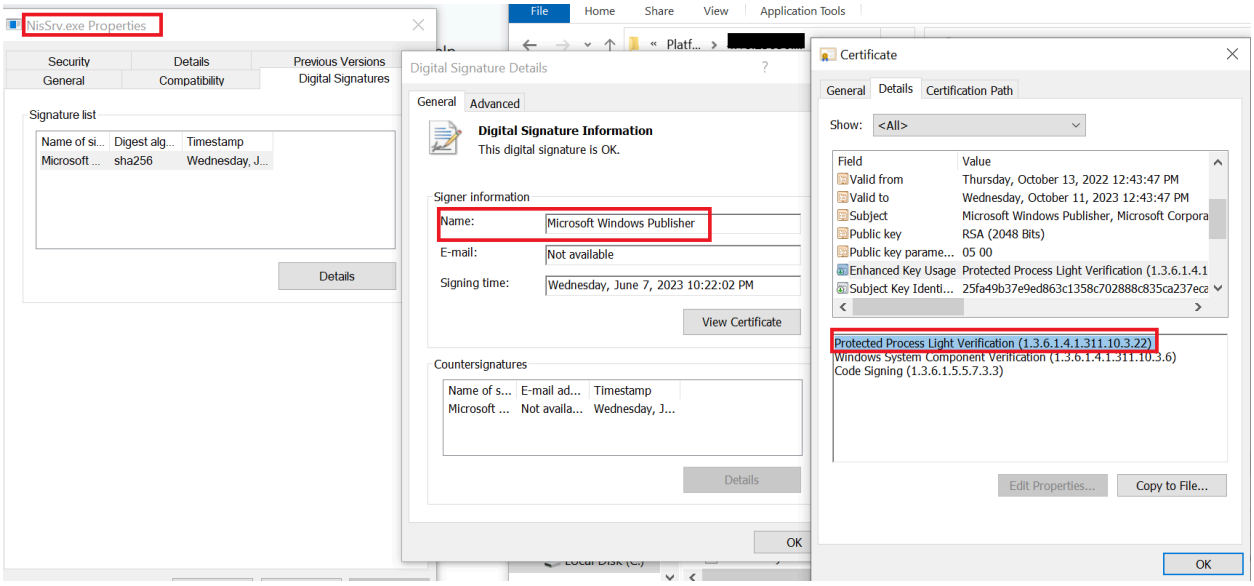
NisSrv.exe (Microsoft Network Realtime Inspection Service)

“NisSrv.exe” is a PE binary which is the main executable that is started by the “WdNisSvc” service aka “Microsoft Network Realtime Inspection”. It is executed by “services.exe” with the permissions of the “NT AUTHORITY\LOCAL SERVICE” user (S-1-5-19). The description of the service states it helps in guarding against intrusion attempts targeting known/newly discovered vulnerabilities in network protocols.

Overall, “NisSrv.exe” monitors and inspects network traffic in real-time. By doing that it searches for suspicious behavior that might suggest an exploit targeting the network protocol is being executed[115].

Moreover, “NisSrv.exe” is part of the “Windows Defender” platform, which is Microsoft’s endpoint security platform. “Windows Defender” provides attack surface reduction and next generation protection for both OS level and network based[116].

Lastly, “NisSrv.exe" is a PE binary file located at "%ProgramData%\Microsoft\Windows Defender\Platform\[VERSION]\NisSrv.exe". It is also signed digitally by Microsoft the same way as the main process of “Windows Defender” (MsMpEng.exe), with a signed level of Antimalware (PsProtectedSignerAntimalware-Light) - as shown in the screenshot below.



[115] https://www.howtogeek.com/357184/what-is-microsoft-network-realtime-inspection-service-nissrv.exe-and-why-is-it-running-on-my-pc/

[116] https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/microsoft-defender-endpoint?view=o365-worldwide

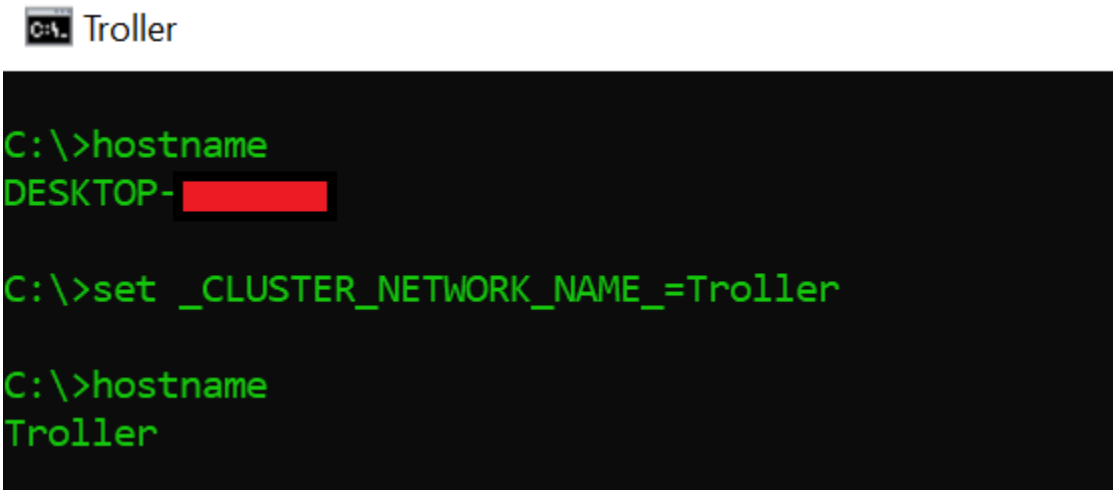
Hostname.exe (Hostname APP)

“hostname.exe” is an executable located at “%windir%\System32\HOSTNAME.EXE”. On a 64-bit system there is also a 32-bit version located at “%windir%\SysWOW64\HOSTNAME.EXE”. The executable is digitally signed by Microsoft.

Overall, “hostname.exe” is responsible for displaying the host name portion of the full computer name. By the way, printing the environment variable %COMPUTERNAME% will output the same result as “hostname.exe”[117]. By the way, “hostname.exe” uses the Win32 API in order to retrieve the information, based on ReactOS[118] the function is “GetComputerNameExW”[119].

Moreover, for cases in which we have a cluster of compute nodes that have a distinct name we can set the environment variable “_CLUSTER_NETWORK_NAME_” which will change the data returned by Win32 API function[120]. Thus, the data returned by “hostname.exe” will also change as shown in the screenshot below.

Lastly, for an implementation reference of “hostname.exe” I suggest going over the one in ReacOS[121].



[117] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/hostname
[118] https://github.com/reactos/reactos/blob/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/cmdutils/hostname/hostname.c#L36
[119] https://learn.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-getcomputernameexw
[120] https://jeffpar.github.io/kbarchive/kb/198/Q198893/
[121] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/cmdutils/hostname

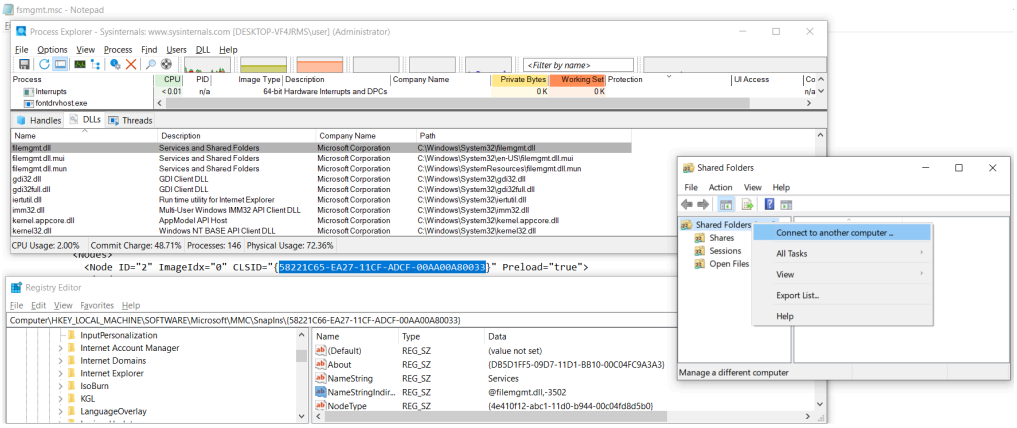
mmc.exe (Microsoft Management Console)

“mmc.exe” is the “Microsoft Management Console” which is responsible for creating/saving/opening consoles (aka administrative tools). They are used in order to manage software/hardware/network components as part of a given system which runs Windows. We can also create our own custom console and distribute it. Those consoles can include different snap-ins, which is a management tool hosted by “mmc.exe”[122].

Moreover, snap-ins/custom console are distributed as part of “*.msc” file, which are as of today are XML files that are parsed “mmc.exe” is order to load the specific snap-ins[123]. Even a clean installation of Windows comes with a couple of builtin “*.msc” file like: “services.msc” (for managing services), “WF.msc” (for managing the “Windows Defender Firewall") and “fsmgmt.msc” (for managing shared folders). You can find them (and more) in the following location: “%windir%\system32\” (of course we can also save them to other locations).

At the end, a snap-in leads to a specific “*.dll” which is loaded by “mmc.exe” (“*.msc” can include a reference for a couple of snap-ins). The relevant configuration is stored in the registry under “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MMC\SnapIns”[124]. The snap-ins are identified using a “CLSID” (as other COM objects) - as seen in the screenshot below. Fun fact about “*.msc” files contain data of the icon we want to be displayed when the file is shown by “explorer.exe”[125] or when “mmc.exe” is executed (as the app icon).

Also, one of the differences between MMC and other management consoles in Windows (like “Control Panel”) is the fact we can also manage remote systems (we have to authenticate for that) - as shown in the screenshot below (on the right side). Lastly, a reference implementation of “mmc.exe” is included as part of ReactOS[126].



[122] https://learn.microsoft.com/en-us/troubleshoot/windows-server/system-management-components/what-is-microsoft-management-console

[123] http://file.fyicenter.com/143_Windows_.MSC_File_Extension_for_Microsoft_Management_Conso.html

[124] https://www.groovypost.com/tips/mmc-exe-windows-process-safe-virus/

[125] https://medium.com/@boutnaru/the-windows-process-journey-explorer-exe-windows-explorer-9a96bc79e183

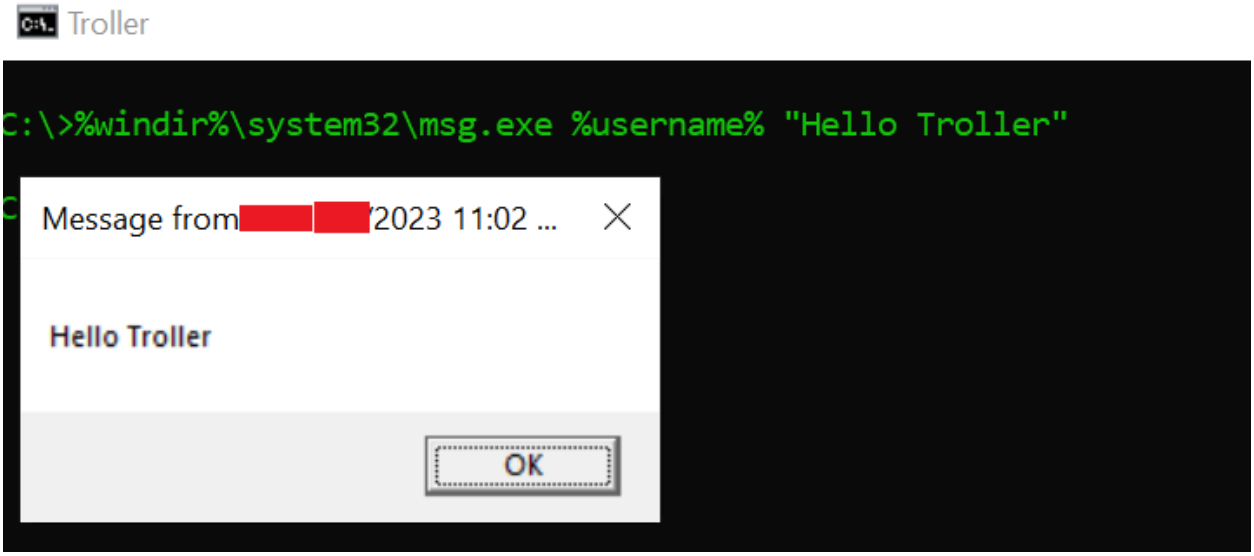
[126] https://github.com/reactos/reactos/tree/master/base/applications/mmc

msg.exe (Message Utility)

“msg.exe” is the “Message Utility” which is a command line which allows sending a message to a user. It is a PE binary located at “%windir%\System32\msg.exe” which is signed by Microsoft. On a 64-bit system there is no 32-bit version of this file (in the SysWOW64 directory).

Overall, we can send a message by specifying a username (using * causes the message to arrive to all users), a session id and even send a message to a remote machine, it is mainly used for sending Terminal Services/Citrix shutdown messages. Also, we can define a delay for waiting for the receiver to acknowledge the message. The executable is not included in ‘Home’ editions of Windows[127].

Moreover, historically this functionality was part of the “Messenger Service” until Windows Vista/2008. It was also operated by using the “net send” command[128]. Lastly, the sending of the message is done using RPC (“msg.exe” loads the RPC runtime DLL) and even MS-RPC over SMB in case of sending the message to a remote[129] . We can see an example of using “msg.exe” in the screenshot shown below.



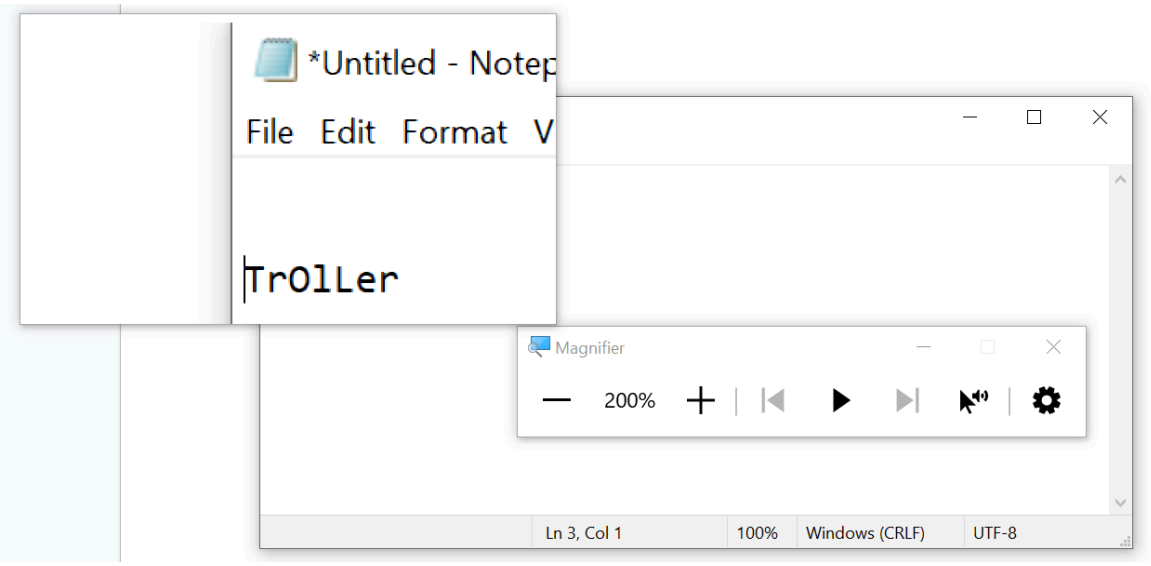
[127] https://ss64.com/nt/msg.html
[128] https://www.lifewire.com/net-send-2618095
[129] https://sid-500.com/2017/10/07/active-directory-send-messages-to-all-currently-logged-on-users-msg-exe/comment-page-1/

Magnify.exe (Microsoft Screen Magnifier)

"Magnify.exe" is the "Microsoft Screen Magnifier" which makes part of the screen bigger in order to see images/text better. "Magnify.exe" has several options like: customizing the zoom level, smoothing the edges of images/text, inverting colors, reading text and more[130]

Overall, "Magnify.exe" is a PE binary located at "%windir%\System32\Magnify.exe" which is signed by Microsoft. Also, on a 64-bit system there is also a 32-bit version located at "%windir%\SysWOW64\Magnify.exe". Also, the file is signed by Microsoft.

Lastly, although there is no help displayed by "Magnify.exe" when running it from the command line it still has a couple of switches that can be used. Examples are "/lens" (as shown in the screenshot below) which defaults to lens view and "/docked" which defaults to "dock view"[131].



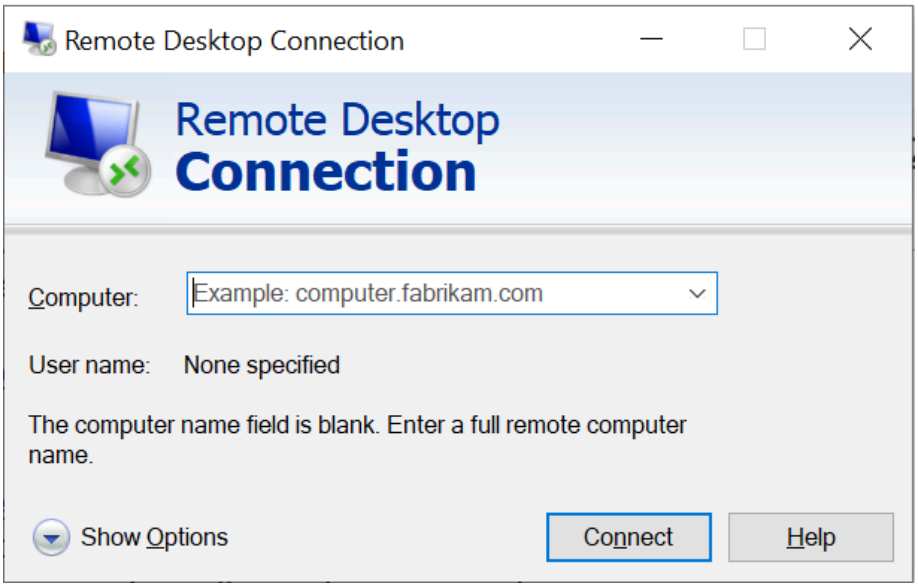
[130] https://support.microsoft.com/en-us/windows/use-magnifier-to-make-things-on-the-screen-easier-to-see-414948ba-8b1c-d3bd-8615-0e5e32204198
[131] https://answers.microsoft.com/en-us/windows/forum/all/magnifyexe-zoom-in-from-cmd-command-prompt/48c7257b-c1f8-483c-a0b8-fff24daf1622

mstsc.exe (Remote Desktop Connection)

“mstsc.exe” is an executable located at “%windir%\System32\mstsc.exe”, it is also known as “Remote Desktop Connection”. On a 64-bit system there is also a 32-bit version located at “%windir%\SysWOW64\mstsc.exe”. It is a PE file which is signed by Microsoft.

Moreover, the name of the executable comes from “Microsoft Terminal Service Client”. “Terminal Service” was the previous name for the protocol used for the remote connection. Today it is called “Remote Desktop Protocol” (RDP). “mstsc.exe” is the default client for RDP that is part of the Windows operating system[132]. I will write a dedicated writeup about the RDP protocol itself.

Overall, “mstsc.exe” allows users to connect to a “Remote Session Host” server or remote computer and to use the GUI interface of the remote system. Also, by using the executable we can edit “*.rdp” file, which is a remote desktop connection configuration file[133]. Using “mstsc.exe” a user can also share its printers/clipboard/audio devices/network drives with the remote system to which the connection is being done. Lastly, for an implementation reference of “mstsc.exe” I suggest going over the one in ReacOS[134].



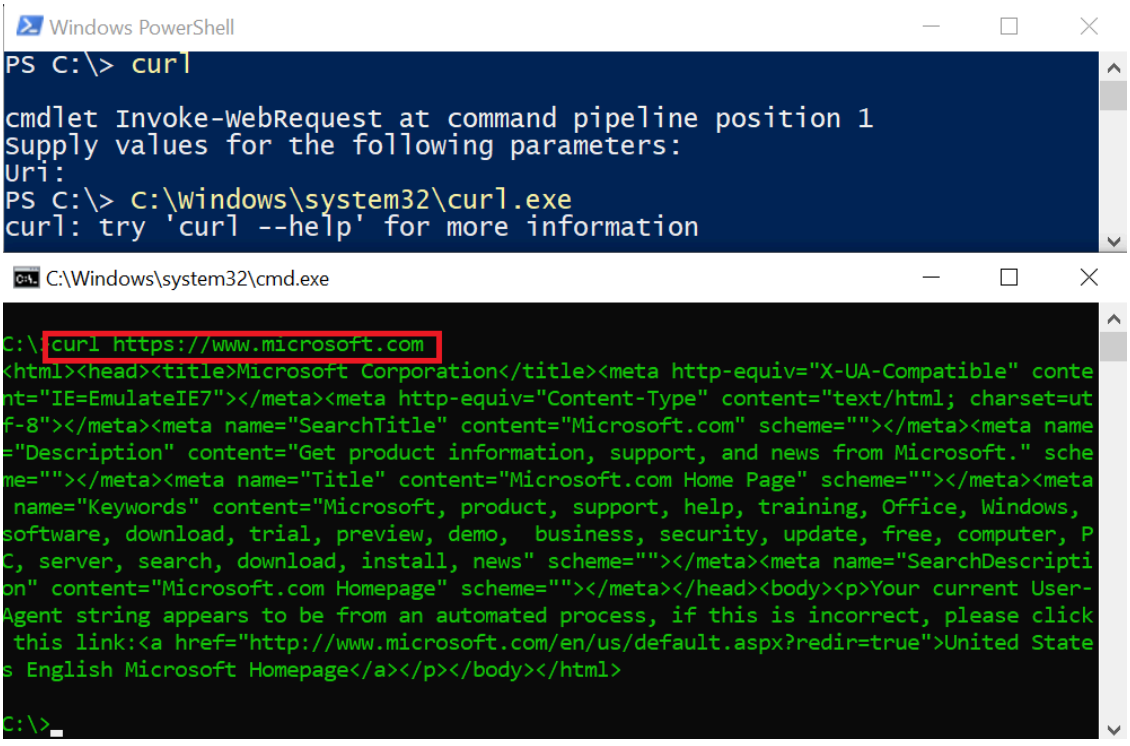
[132] https://en.wikipedia.org/wiki/Remote_Desktop_Protocol
[133] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/mstsc
[134] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/applications/mstsc

curl.exe (cURL executable)

"curl.exe" is a command line tool which allows transferring data with URLs. It supports various protocols like: FTP/S, HTTP/S, IMAP/S, LDAP/S, MQTT, POP3, SMB/S[135]. "curl" is a popular command line tool for Linux[136]. There is also a version of "curl" for Windows. it is statically linked with different libraries like: libssh2, brotli, zlib, zstd, nghttp3, nghttp2, cacert[137].

Moreover, since build 17063 of Windows 10 (December 2017), Microsoft has announced that "curl" is going to be shipped by default as part of Windows[138]. However, "curl.exe" that is shipped with Windows is handled and built by Microsoft. Microsoft's version of "curl" uses the SChannel TLS backend[139].

Lastly, there is also a "curl" command as part of Powershell, but it is just an alias to the "Invoke-WebRequest"cmdlet - as shown in the screenshot below. We can go over the source code of curl in GitHub[140]. Using "curl.exe" we can send HTTP GET requests (as shown below), resuming downloads, specifying max transfer rate and more[141].



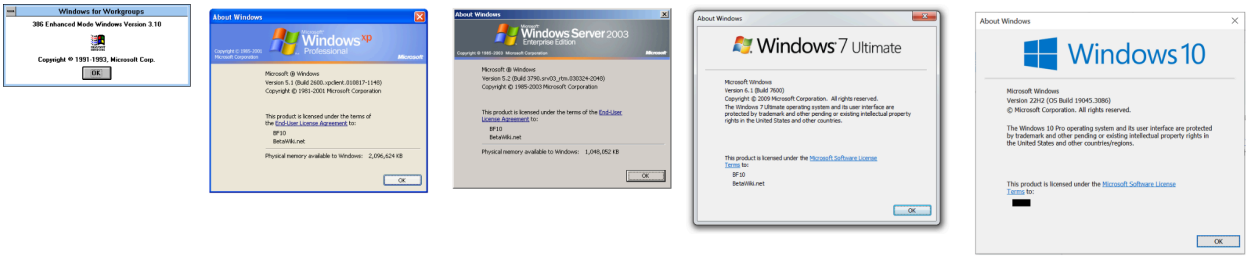
[135] https://curl.se/
[136] https://linux.die.net/man/1/curl
[137] https://curl.se/windows/
[138] https://learn.microsoft.com/en-us/virtualization/community/team-blog/2017/20171219-tar-and-curl-come-to-windows
[139] https://curl.se/windows/microsoft.html
[140] https://github.com/curl/curl
[141] https://www.keycdn.com/support/popular-curl-examples

winver.exe (Version Reporter Applet)

“winver” is the “Version Reporter Applet” which is responsible for displaying information about the version of the running operating system. It is also referred to as the “Windows Version” utility[142]. It is a PE binary file located at “%windir%\System32\winver.exe”, on 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\winver.exe”.

Also, “winver.exe” is signed by Microsoft. It was first include in Windows from “Windows 3.0”, since “Windows 3.5” it calls the “ShellAbout” function from “shell32.dll”[143]. Thus, if we have a version of Windows that does not include “winver.exe”(like Windows PE) we can use “rundll32.exe”[144] to call it with the following command “rundll32 shell32,ShellAbout”.

Moreover, due to the UI changes that have been made in Windows along the way in Windows caused also for changes in “winver.exe” as shown in the screenshots below[145]. The examples are from the following versions of Windows (from left to right): “Windows 3.10”, “Window XP”, “Windows 2003 Server”, “Windows 7” and “Windows 10”. Lastly, we can checkout the implementation of “winver.exe” as part of ReacOS[146].



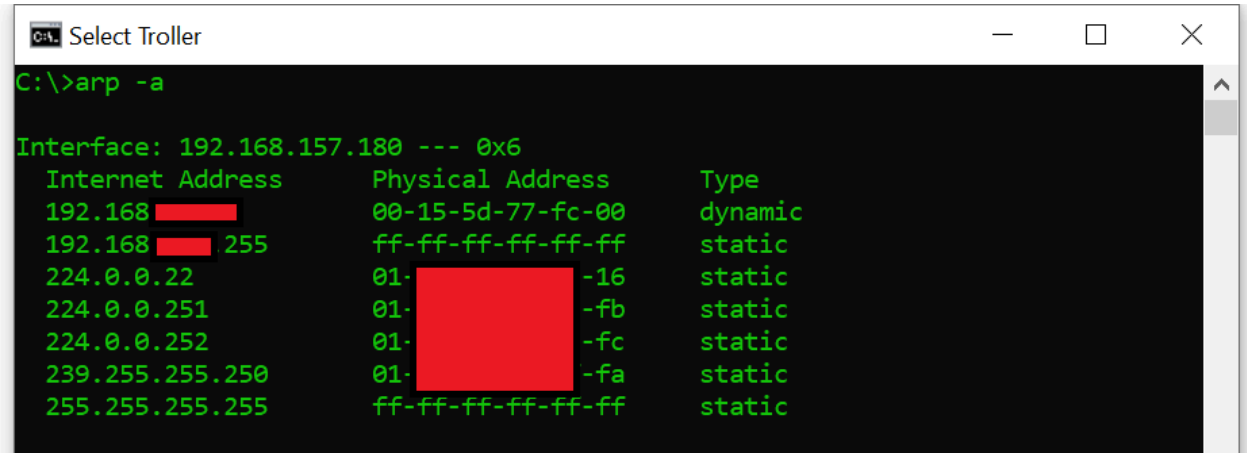
[142] https://betawiki.net/wiki/Winver
[143] https://learn.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellaboutw
[144] https://medium.com/@boutnaru/the-windows-process-journey-rundll32-exe-windows-host-process-415132f1363
[145] https://betawiki.net/wiki/Winver
[146] https://github.com/reactos/reactos/tree/master/base/applications/winver

arp.exe (TCP/IP Arp Command)

“arp.exe” (TCP/IP Arp Command) is a PE binary located at “%windir%\System32\ARP.EXE”. On 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\ARP.EXE”. Also, the binary file is digitally signed by Microsoft.

Overall, “arp.exe” allows displaying (using the “-a” or “/a” switch - as shown in the screenshot below) and modifying (using the “-s” or “/s” switch) entries in the ARP (Address Resolution Protocol) cache. There is a separate table for each network adapter that the system has (which is connected and has IP information). It is relevant for Ethernet/Token Ring network adapters[147].

Basically, ARP is a network protocol used for retrieving the link layer address (like MAC) for a given internet layer address (like IPv4). By the way, in IPv6 the functionality of ARP is implemented by NDP (Neighbor Discovery Protocol). Lastly, ARP is a request/response protocol which is encapsulated by the link layer protocol. Also, it is never routed across inter-networking entities[148].



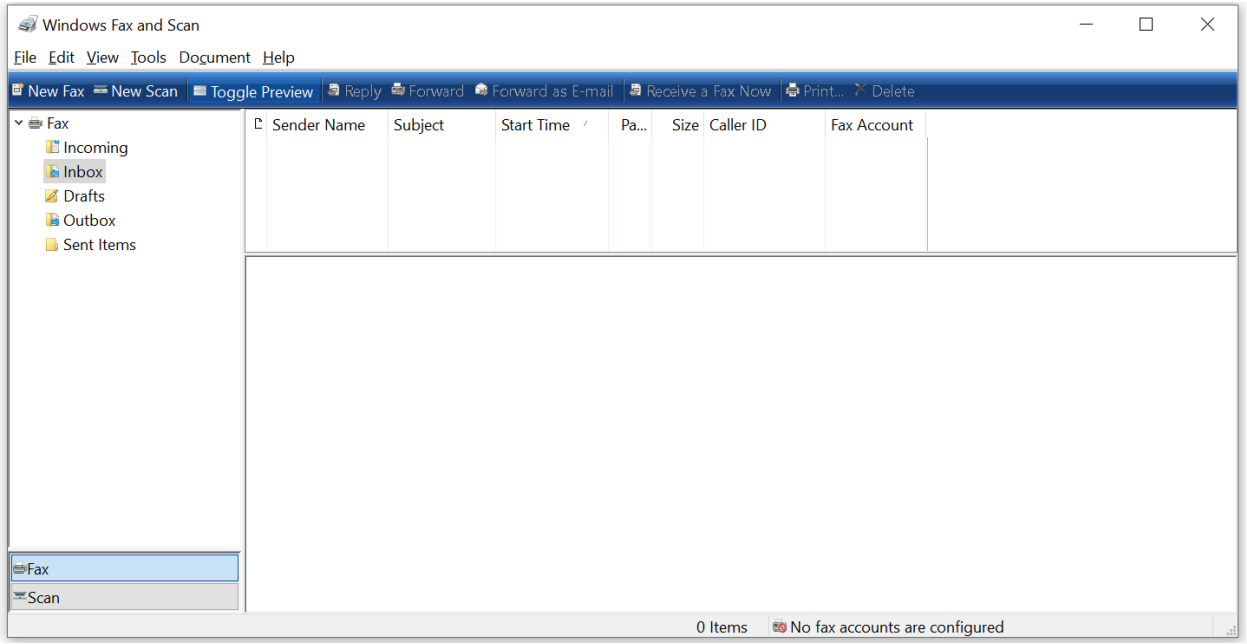
[147] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/arp
[148] https://en.wikipedia.org/wiki/Address_Resolution_Protocol

WFS.exe (Microsoft Windows Fax and Scan)

“WFS.exe” (aka the “Microsoft Windows Fax and Scan”) which is an integrated scanning and faxing app as part of Windows. It is the replacement of the “Fax Console” that was part of Windows XP. Overall, “WFS.exe” provides the ability to send/receive faxes, emailing/faxing scanned documents and forwarding faxes as email attachments[149].

Also, It is a PE binary file located at “%windir%\System32\WFS.exe” which is digitally signed by Microsoft. By the way, on a 64-bit system there is only the 64-bit version, there is not a 32-bit version (in “%windir%\SysWOW64”) like we have with other executables such as “cmd.exe”.

Moreover, in order for “WFS.exe” to operate correctly we need to install it as an “Optional Feature”[150]. It is also dependent on the Fax service, which executable is located at “%windir%\System32\FXSSVC.exe”[151].



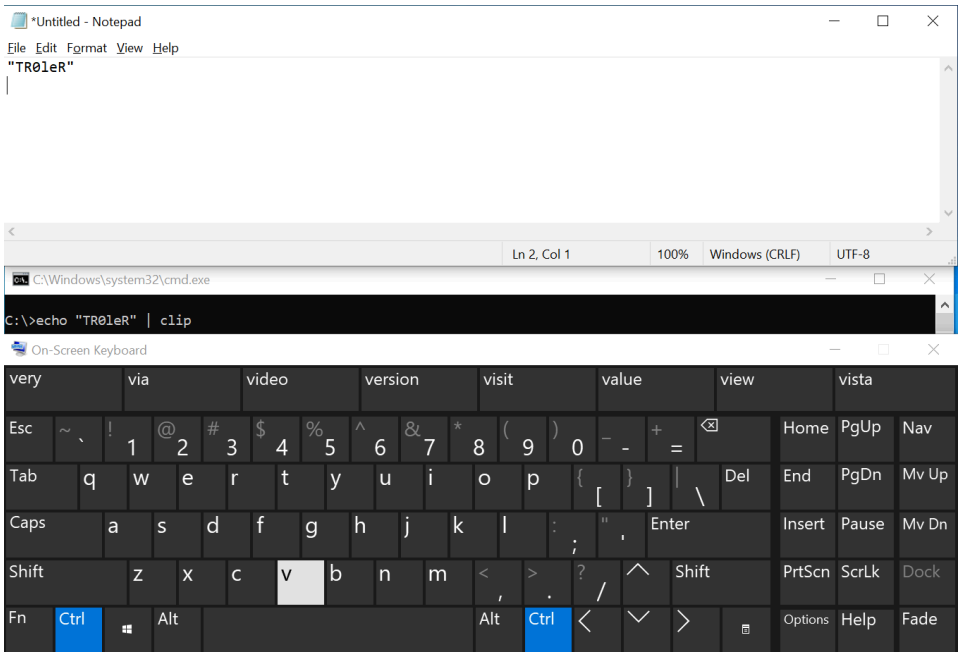
[149] https://en.wikipedia.org/wiki/Windows_Fax_and_Scan
[150] https://www.intowindows.com/how-to-install-windows-fax-and-scan-in-windows-11/
[151] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc725953(v=ws.11)?redirectedfrom=MSDN

clip.exe (Copies the Data into Clipboard)

“clip.exe” (copies the data into clipboard) is a PE binary located at “%windir%\System32\clip.exe”. On 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\clip.exe”. Also, the binary file is a CLI tool which is digitally signed by Microsoft.

Overall, “clip.exe” is used in order to copy the results of commands into the Windows clipboard. We can use it in one of the following ways: “command | clip” or “clip < file.txt”[152]. After using “clip.exe” the text output can be pasted into another program.

Thus, we can see an example of usage in the screenshot below. In the screenshot we use “clip.exe” to store an echoed string into the clipboard. Using “osk.exe” (https://medium.com/@boutnaru/the-windows-process-journey-osk-exe-accessibility-on-screen-keyboard-72823695321e) aka the “On Screen Keyboard” we send “Ctrl+V” to paste the stored text into Notepad. Lastly, In powershell we have a cmdlet (“Set-Clipboard”) which does the same as “clip.exe” (https://ss64.com/ps/set-clipboard.html).



[152] https://ss64.com/nt/clip.html

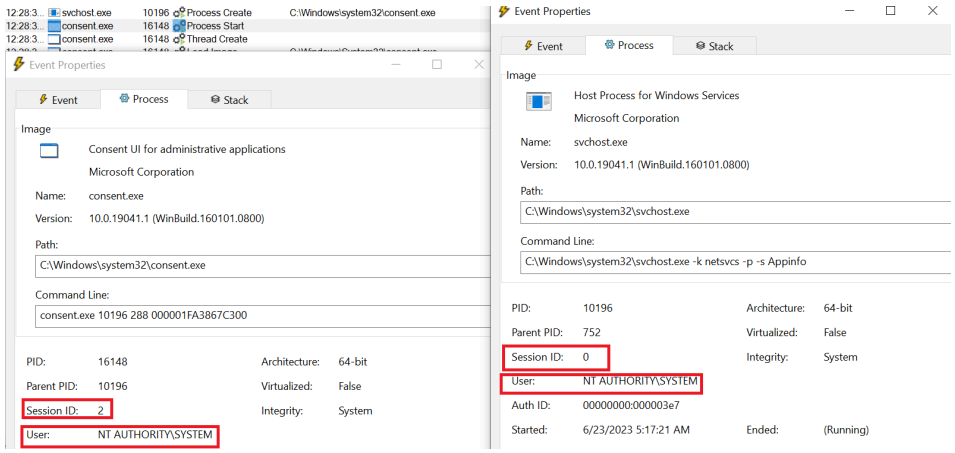
consent.exe (Consent UI for Administrative Applications)

“consent.exe” is the “Consent UI for Administrative Applications” which is called as part of a UAC (User Account Control) flow[153]. It is a PE binary file located at “%windir%\system32\consent.exe”, which is signed digitally by Microsoft. On a 64-bit system there is no 32-bit version, as we have with other binaries such as “cmd.exe”.

Moreover, as shown in the screenshot below, “consent.exe” is started by the service “Application Information” which is hosted by “svchost.exe”[154]. The description of the service states that it “Facilitates the running of interactive applications with additional administrative privileges. If this service is stopped, users will be unable to launch applications with the additional administrative privileges they may require to perform desired user tasks”.

Also, as shown in the screenshot below, although it is running within “session 0” we can see that “consent.exe” is assigned to “session 2” with the permissions of “NT AUTHORITY\SYSTEM”. For further security the consent prompt is displayed on the secure desktop, only Windows processes can access the secure desktop[155].

Lastly, if the logged on user is not an administrative account a credentials prompt will be displayed for getting a username and password for an administrative account - it is also done by “consent.exe” in a secure desktop[156]. We can turn off prompting in secure mode with “reg.exe”: ‘REG ADD “HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System" /V "PromptOnSecureDesktop" /T "REG_DWORD" /D "0x00000000" /F’[157].

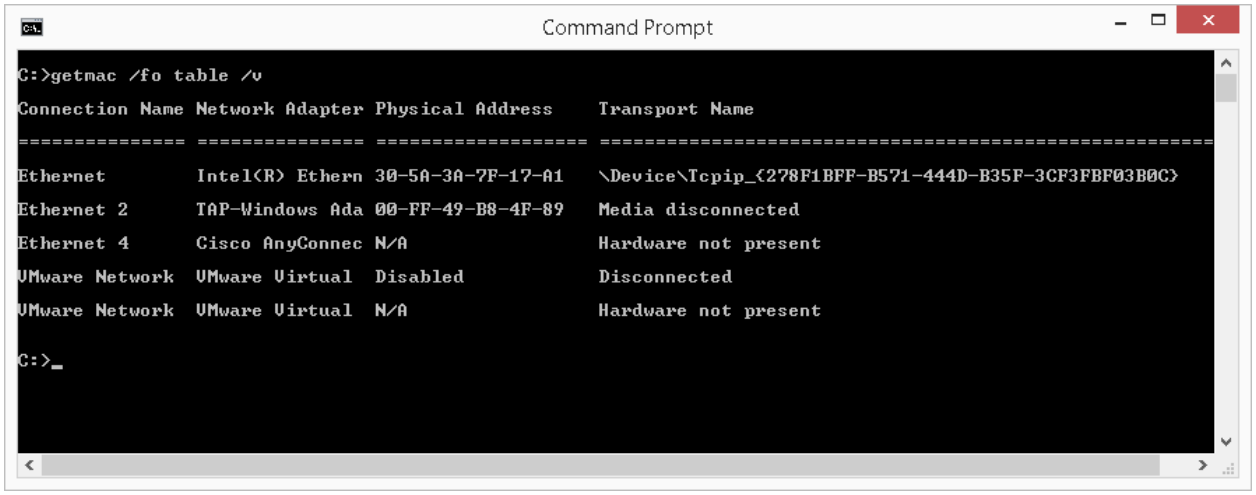


153 https://www.file.net/process/consent.exe.html
154 https://medium.com/@boutnaru/the-windows-process-journey-svchost-exe-host-process-for-windows-services-b18c65f7073f
155 https://learn.microsoft.com/en-us/windows/security/application-security/application-control/user-account-control/how-it-works
156 https://securityinternals.blogspot.com/2014/02/the-user-access-control-uac-prompts.html
157 https://stackoverflow.com/questions/4046940/how-to-screen-shot-a-uac-prompt

getmac.exe (Displays NIC MAC information)

“getmac.exe” is a binary PE file located at “%windir%\System32\getmac.exe”, on 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\getmac.exe”. This is a CLI application which is digitally signed by Microsoft.

Overall, “getmac.exe” is used for retrieving the MAC (Media Access Control) address for all the NIC (Network Interface Cards) on the system (both physical and virtual)[158]. By the way, this is not the only CLI tool we can use to show the MAC address of NICs - we can also use “ipconfig.exe” (on which there is going to be a separate writeup) and even “nbtstat.exe” to show the MAC address of a remote machine (on this there is also going to be a separate writeup). Lastly, an example output of the command is shown below.



[158] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/getmac

defrag.exe (Disk Defragmenter Module)

“defrag.exe” (Disk Defragmenter Module) is used to improve system performance by consolidating fragmented files on local volumes[159].

Overall, defragmentation organizes storage by consolidating files/other data saved on the hard drive. Due to different reasons when files are stored they can be broken down into smaller pieces (aka fragments) that can be spread across the hard drive. The goal of the defragmentation is to take scattered data in a hard drive and organize it for more efficient retrieval - as shown in the diagram below[160]. The above part is before the process and the lower one is after it.

Moreover, we can’t defragment every file system which exists. There is only support for NTFS, ReFS and FAT/FAT32 file system volumes. Thus, CD-ROMs/Network drives/volumes locked by the filesystem are not supported. Also, if the file system is marked as dirty, which might indicate possible corruption - it can be verified using the command “fsutil dirty”[161].

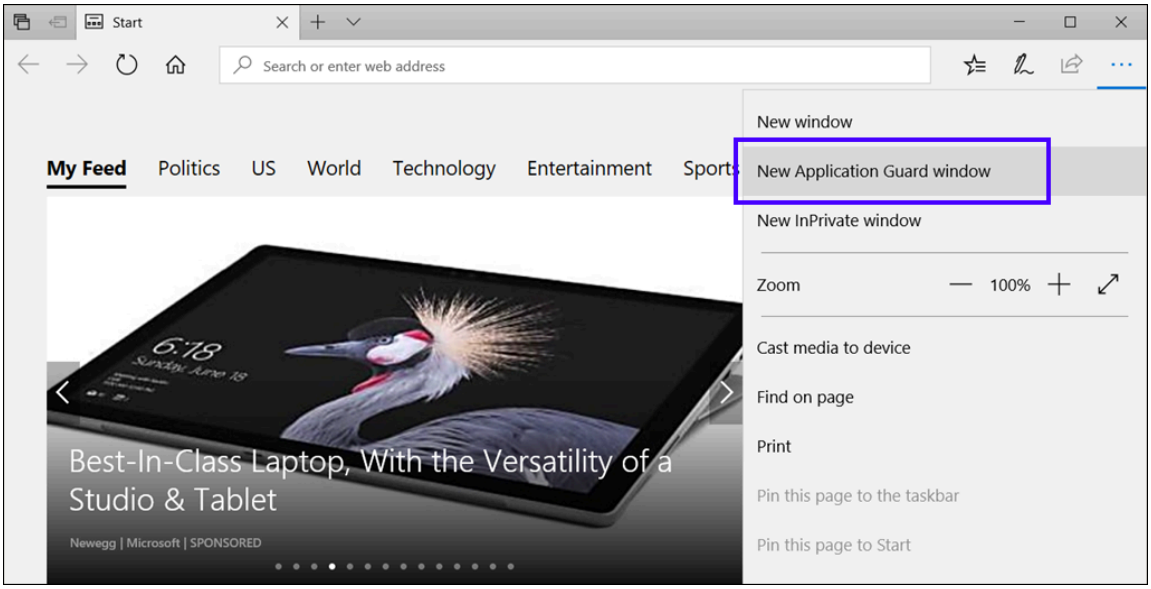
[159] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/defrag
[160] https://www.avast.com/c-how-to-defrag-pc-hard-drive
[161] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/defrag

msedge.exe (Microsoft Edge)

“msedge.exe” is a 64-bit binary which is signed by Microsoft. Although it is a 64-bit binary it is still located by default in the program files directory of 32-bit applications ("C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"). Microsoft Edge (aka Edge) is a web browser that is based on chromium which was released on January 15, 2020. It is supported on Windows, macOS, iOS and Android[162]. By the way, if you want you can also be part of the “Microsoft Edge Insider Channel”. This allows you to be from the first who previews what’s new in Edge[163].

Moreover, from Windows 10 Enterprise/Pro (versions 1803 and later) or Windows 11 Pro users can use the “Application Guard” mode of Edge - as shown in the screenshot below. It disables printing from the application guard window, does not allow copying/pasting between the host PC and the application guard window and does not permit data persistence between application guard windows[164].

Lastly, In order to enable that we need to enable the “"Windows Defender Application Guard" feature (it requires the CPU support for virtualization). It launches Edge in an Hyper-V virtualized isolated environment[165]. A temporary container is created each time, it is destroyed/deleted when the user closes all the related windows[166].



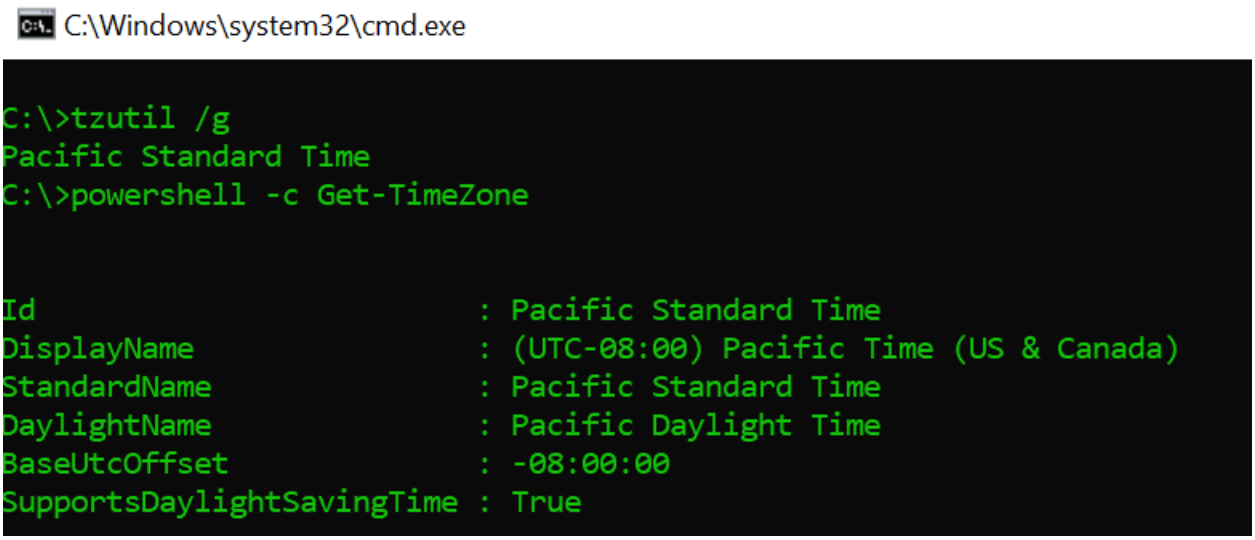
[162] https://support.microsoft.com/en-us/microsoft-edge/download-the-new-microsoft-edge-based-on-chromium-0f4a3dd7-55df-60f5-739f-00010dba52cf
[163] https://www.microsoft.com/en-us/edge/download/insider
[164] https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/microsoft-defender-application-guard/test-scenarios-md-app-guard
[165] ttps://techcommunity.microsoft.com/t5/windows-insider-program/windows-defender-application-guard-standalone-mode/m-p/66903
[166] https://blogs.windows.com/msedgedev/2016/09/27/application-guard-microsoft-edge/

tzutil.exe (Windows Time Zone Utility)

"tzutil.exe" is a binary PE file located at "%windir%\system32\tzutil.exe". It is used in order to display/set the time zone of the current system[167]. On 64-bit systems there is also a 32-bit version of "tzutil.exe" located at "%windir%\SysWOW64\tzutil.exe".

Moreover, "tzutil.exe" is a CLI tool which is digitally signed by Microsoft. For displaying the current time zone ID we use the "/g" switch while for setting the time zone we use the "/s" switch[168]. There are different time zones that can be set using this command[169], we can also list them using the "/l" switch.

Lastly, there are cmdlets which are equal to "tzutil.exe" which is called Get-TimeZone/Set-TimeZone - as shown in the screenshot below.



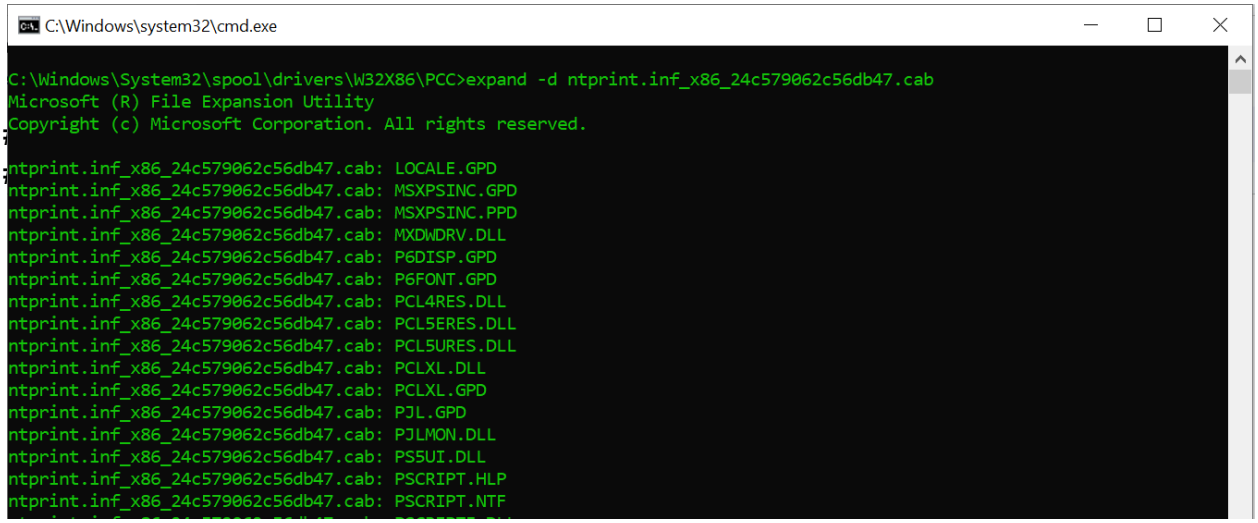
[167] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/tzutil
[168] https://ss64.com/nt/tzutil.html
[169] https://ss64.com/nt/timezones.html

expand.exe (LZ Expansion Utility)

“expand.exe” aka “LZ Expansion Utility” is a PE binary located at “%windir%\System32\expand.exe”. It is used for expanding one or more compressed files. For example we can use it to retrieve compressed files from distribution disks[170]. On 64-bit systems there is also a 32-bit version of “expand.exe” located at “%windir%\SysWOW64\expand.exe”.

Moreover, it is used to uncompress “*.cab” files (cabinet files). “expand.exe” is also called “The Microsoft File Expansion Utility” and it dates back to MS-DOS 5 in 1990[171]. The simplest way to use it could be the following: “expand -d [FILE_NAME].cab” - as shown in the screenshot below.

Lastly, versions of expand before version 6.0 (Windows 7 timeline) included buggy implementation of “*.cab” file which include subfolders[172].



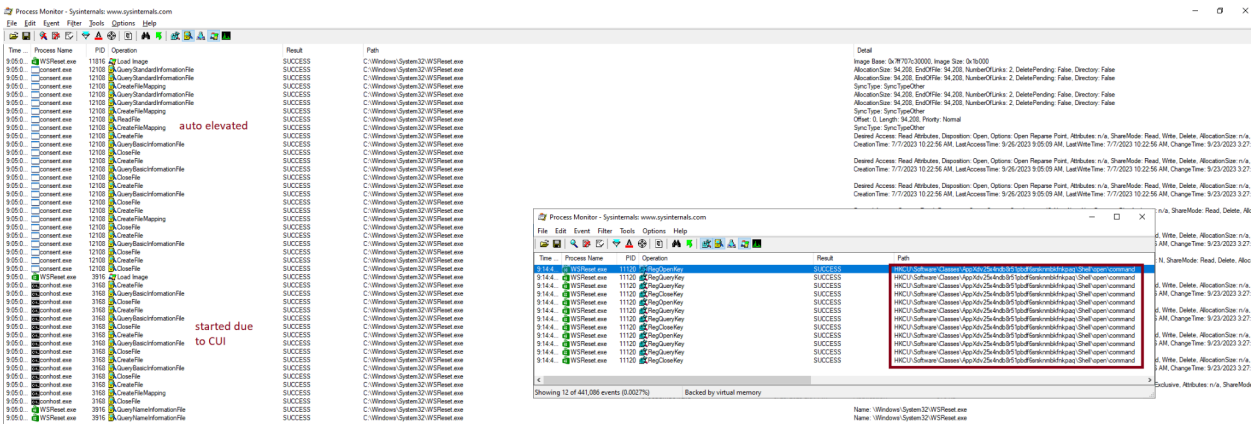
[170] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/expand
[171] https://ss64.com/nt/expand.html
[172] https://ss64.org/viewtopic.php?t=71

WSReset.exe (Windows Store Reset)

In general, “WSReset.exe” is a PE binary file located at “%windir%\System32\WSReset.exe” which is also digitally signed by Microsoft. The description (Part of the PE format) states “This tool resets the Windows Store without changing account settings or deleting installed apps”. By the way, there is no 32-bit version of “WSRest.exe” on 64-bit systems (like we have with “cmd.exe” for example).

Thus, we can say “WSReset.exe” is used for clearing the cache of the “Windows Store”[173]. The “Windows Store” creates temporary/cookies files in the following directories: “%UserProfile%\AppData\Local\Packages\Microsoft.WindowsStore_8wekyb3d8bbwe\AC\INetCache” and “%UserProfile%\AppData\Local\Packages\Microsoft.WindowsStore_8wekyb3d8bbwe\AC\INetCookies”. So in order to clear the cache the executable just needs to delete the files from those folders[174] - as also shown in the screenshot below.

Lastly, “WSReset.exe” is also auto elevated and during its startup it checks the following registry value “HKCU\Software\Classes\AppX82a6gwre4fdg3bt635tn5ctqjf8msdd2\Shell\open\command” for commands to execute[175] - as shown in the screenshot below. This executable is a console tool, due to that “conhost.exe”[176] is also needed as we can see in the screenshot below.



[173] https://helpdeskgeek.com/how-to/how-to-clear-windows-store-cache-with-wsreset-exe/
[174] https://daniels-it-blog.blogspot.com/2020/07/arbitrary-file-delete-via-wsresetexe.html
[175] https://lolbas-project.github.io/lolbas/Binaries/Wsreset/
[176] https://medium.com/@boutnaru/the-windows-process-journey-conhost-exe-console-window-host-f03f8db35574

SlideToShutDown.exe (Windows Slide To Shutdown)

“SlideToShutDown.exe” is a PE binary located at “%windir%\System32\SlideToShutDown.exe”. It can be used in a smart and interactive way for shutting down Windows. Instead of the traditional way, we can just shutdown the system by sliding/dragging the window down - as shown in the screenshot below[177].

Moreover, on 64-bit systems we don’t have a 32-bit version of “SlideToShutdown.exe” (as we have with “cmd.exe” for example). The binary is digitally signed by Microsoft. By default, the “slide to shutdown” should only show if we hold down the power button on a system with a touch screen[178]. Lastly, even if we don’t have a touch screen we can use the mouse for sliding/dragging the window down.

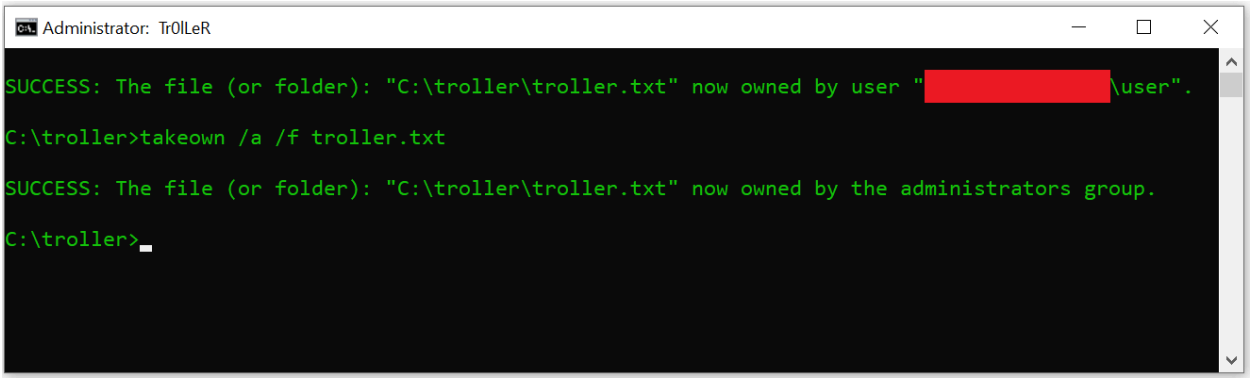
[177] https://www.geeksforgeeks.org/creating-slide-to-shut-down-shortcut-in-windows-10/
[178] https://answers.microsoft.com/en-us/windows/forum/all/slide-to-shut-down/7b7e3f86-ccea-41a4-be8b-74531ea2fcb8

takeown.exe (Takes Ownership of a File)

“takeown.exe” (Takes ownership of a file) is a PE binary located at “%windir%\System32\takeown.exe”. It is a CLI tool which allows an administrator to recover access to a file that was denied, it is done by changing the file-ownership[179]. On 64-bit systems there is also a 32-bit version of “takeown.exe” located at “%windir%\SysWOW64\takeown.exe”.

Thus, after the ownership of the file/folder is taken the logged-on user is provided with the “full control” permissions. This allows the user to change the DACL[180] of the file/folder[181].

Lastly, by default the owner of a securable object[182] is based on the entity described by the access token[183] of the process/thread that has created it. It can be changed by the current owner or by a security context which holds the take ownership (SeTakeOwnershipPrivilege) privilege[184].



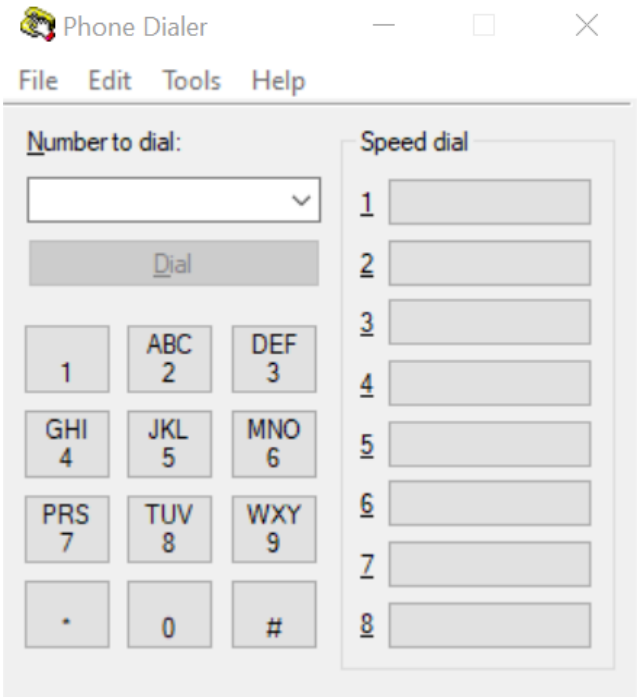
[179] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/takeow
[180] https://medium.com/@boutnaru/the-windows-security-journey-dacl-discretionary-access-control-list-c74545e472ec
[181] https://appuals.com/takeown/
[182] https://medium.com/@boutnaru/windows-securable-objects-311a9d6c83ad
[183] https://medium.com/@boutnaru/windows-security-access-token-81cd00000c64
[184] https://medium.com/@boutnaru/windows-security-privileges-b8fe18cf3d5a

dialer.exe (Microsoft Windows Phone Dialer)

"dialer.exe" (Microsoft Windows Phone Dialer) is a PE binary located at "%windir%\System32\dialer.exe", which can be used to dial outgoing voice calls using the computer. It is done if the system has a modem supporting both voice and data[185] .On 64-bit systems there is also a 32-bit located at %windir%\SysWOW64\dialer.exe.

Thus, "dialer.exe" supports TAPI (Telephony Program Interface) based ActiveVoice[186]. TAPI is an API (Application Programming Interface) allowing Windows systems to use the telephony services[187].

Moreover, TPAPI is a COM[188] based API that merges classic and IP telephony. It allows voice mailing, PBX control, basic voice over PSTN (Public Switched Telephone Network), call center applications, IVR (Interactive Voice Response), multicast multimedia and video conferencing[189]. Lastly, we can think about "dialer.exe" as a software based phone - as also shown in the screenshot below.



[185] https://answers.microsoft.com/en-us/windows/forum/all/how-do-you-set-up-dialer/2aa4ef09-5a6d-4aa1-901b-557ff9ce0ef6
[186] https://answers.microsoft.com/en-us/windows/forum/all/dialerexe/b859ea03-f8f5-4b45-ab3a-19ff032763ff
[187] https://documentation.avaya.com/en-US/bundle/IPOfficeSolutionDescription/page/Telephony_Application_Program_Interface.html
[188] https://medium.com/@boutnaru/windows-com-component-object-model-71a76a97435c
[189] https://learn.microsoft.com/en-us/windows/win32/tapi/tapi-3-1-start-page

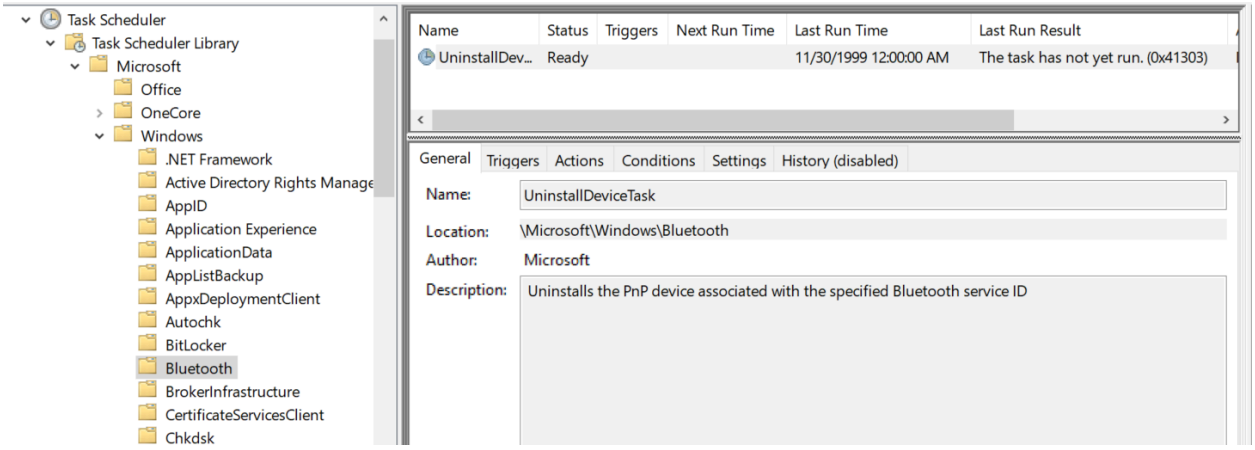
bthudtask.exe (Bluetooth Uninstall Device Task)

"bthudtask.exe" is a PE binary located at "%windir%\System32\bthudtask", which is the Bluetooth uninstall device task. It is used to remove the pairing with a remote Bluetooth device, which is specified by service ID[190].

Moreover, on 64-bit systems there is also a 32-bit version of the executable located at "%windir%\SysWOW64\bthudtask.exe%". Also, the executable is digitally signed by Microsoft and "auto elevated".

Thus, the "Task Scheduler" task[191] that runs "bthudtask.exe" is "UninstallDeviceTask" which is located in the following hierarchy "Microsoft->Windows->Bluetooth" - as shown in the screenshot below. The scheduled task exits after the device is uninstalled[192].

Lastly, from the "Actions" tab we can see that the program is started "BthUdTask.exe $(Arg0)". This means that the Bluetooth service ID is given as the first argument.

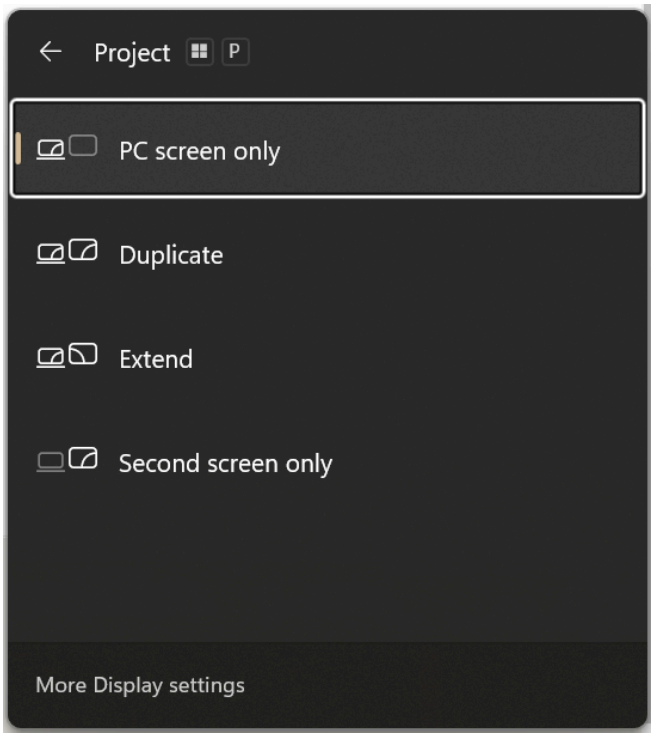


[190] https://www.shouldiblockit.com/bthudtask.exe-91.aspx
[191] https://medium.com/@boutnaru/windows-scheduler-tasks-84d14fe733c0
[192]https://support.microsoft.com/en-gb/topic/description-of-the-scheduled-tasks-in-windows-vista-21f93b44-7260-a612-5ec3-fb2a7be5563c

DisplaySwitch.exe (Windows Display Switch)

"DisplaySwitch.exe" is a PE binary located at "%windir%\System32\DisplaySwitch.exe", it is used for switching the display based on different options like: PC only, duplicate (mirror), extend and second screen only - as shown in the screenshot below[193]. Moreover, "DisplaySwitch.exe" is signed digitally by Microsoft. On a 64-bit system there is no 32-bit version of "DisplaySwitch.exe" (like we have for example with "cmd.exe").

Lastly, on Windows 10 we can pass the following command line arguments:/internal ,/clone, /extend and /external instead of selecting the option in the GUI. On Windows 11 the switches have been replaced with numbers: 1 (=/internal), 2 (=/clone), 3 (=/extend) and 4 (=/external). Keep in mind not to add a space after the number is given as input argument[194].



[193] https://learn.microsoft.com/en-us/answers/questions/1036148/displayswitch-exe-behavior-on-windows-11-22h2
[194] https://learn.microsoft.com/en-us/answers/questions/1036148/displayswitch-exe-behavior-on-windows-11-22h2

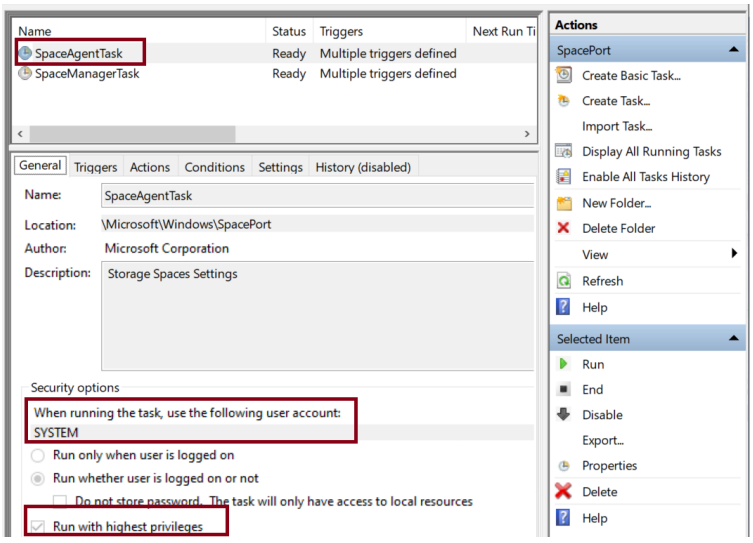
SpaceAgent.exe (Storage Spaces Settings)

“SpaceAgent.exe” is a PE binary located at “%windir%\System32\SpaceAgent.exe”. The description field in the PE format states it is “Storage Spaces Settings”. On 64-bit systems there is no 32-bit version of the binary - as we have with other binaries like “cmd.exe”[195]. It is good to know that the binary itself is also digitally signed by Microsoft.

Overall, “Storage Spaces” allows users to protect data from drive failures. It is a technology similar to RAID (Redundant Array of Independent Disks), which is implemented in software. “Storage Spaces” gives us the ability to combine three or more drives into a single pool of storage. This pool can then be used to create new storage spaces, which typically store multiple copies of your data for redundancy. So, if a drive fails, our data will still be safe[196].

Moreover, “SpaceAgent.exe” is configured to run as a scheduled task using the “Windows Scheduler”[197]. We can see that configuration using the “Computer Management” console (“compmgmt.msc”) - as shown in the screenshot below. The task name is “SpaceAgentTask” and when executed it runs with the permissions of the “Local System” user - also shown in the screenshot. The location of the task configuration is in “%windir%\System32\Tasks\Microsoft\Windows\SpacePort\SpaceAgentTask”.

Lastly, from the manifest’s information as part of the “SpaceAgent.exe” binary, there is a description field which states: “Management agent for the Storage Spaces control panel applet”. Thus, if we click the “Storage Spaces” icon as part of the control panel and after that we click on “"create new pool and storage spaces" an instance of “SpaceAgent.exe” is created.



[195] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[196] https://learn.microsoft.com/en-us/windows-server/storage/storage-spaces/overview
[197] https://medium.com/@boutnaru/windows-scheduler-tasks-84d14fe733c0

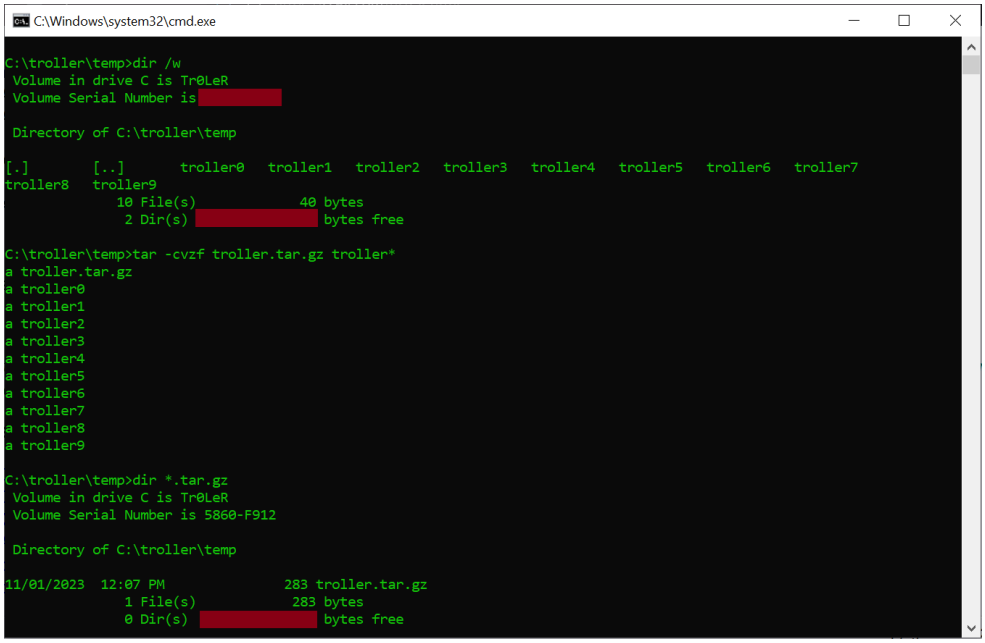
tar.exe (BSD tar Archive Tool)

“tar.exe” is a PE binary located at “%windir%\System32\tar.exe”. It is a command line tool which enables us to create archives and extract files[198]. “tar.exe” is based on the “libarchive”[199], you can check out the code on GitHub[200]. This is referenced by “tar.exe” by using “%windir%\System32\archiveint.dll”.

Moreover, “tar.exe” was added to Windows 10 (1803) from build 17063 or later as a pre-installed binary[201]. There is also a 32-bit version of the binary located at “%windir%\SysWOW64\tar.exe”. Microsoft also digitally signs the “tar.exe” binary.

Overall, by going over the command line options of “tar.exe” we can see that we can perform different operations: create archives, list files inside archives, update archives and extract them. Also, we can compress an archive using gzip/bzip2/xz/lzma and use other formats ustar/pax/cpio/shar[202].

Lastly, when extracting an archive using “tar.exe” we can keep/overwrite existing files, restore (or not) modification times, write data to stdout (and not disk) and restore ACLs[203] and other permission information (ownership and flags).

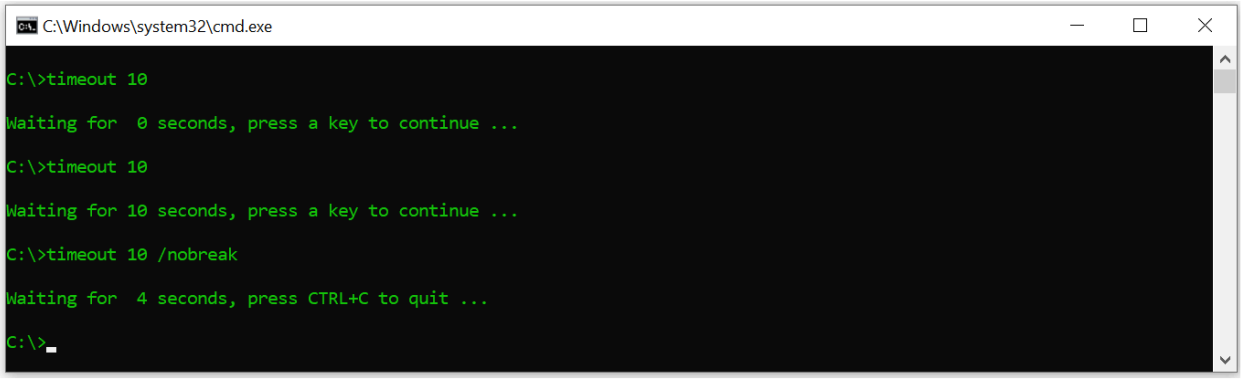


[198] https://learn.microsoft.com/en-us/virtualization/community/team-blog/2017/20171219-tar-and-curl-come-to-windows
[199] https://libarchive.org/
[200] https://github.com/libarchive/libarchive
[201] https://renenyffenegger.ch/notes/Windows/dirs/Windows/System32/tar_exe
[202] https://ss64.com/nt/tar.html
[203] https://medium.com/@boutnaru/the-windows-security-journey-acl-access-control-list-b7d9a6fe428

timeout.exe (Pauses Command Processing)

"timeout.exe" is a PE binary located at "%windir%\System32\timeout.exe". It is a command line tool which enables pausing command processing. By using it we can delay execution for seconds/minutes as part of a batch file[204]. By the way, we don't have "sleep.exe" pre-installed on Windows, it is part of the "Windows Resource Kit"[205].

Moreover, on 64-bit systems of Windows we also have a 32-bit version of "timeout.exe" located at "%windir%\System32\timeout.exe". It is also digitally signed by Microsoft. We can specify using a decimal number the amount of seconds we want to wait. The range is between (-1) to 99999. Using (-1) states to wait indefinitely for a key storkey. There is also an option of ignoring keystores using "/nobreak", which can be canceled using "Ctrl+C"[206]. Lastly, we can see a couple of examples for using "timeout.exe" in the screenshot below.

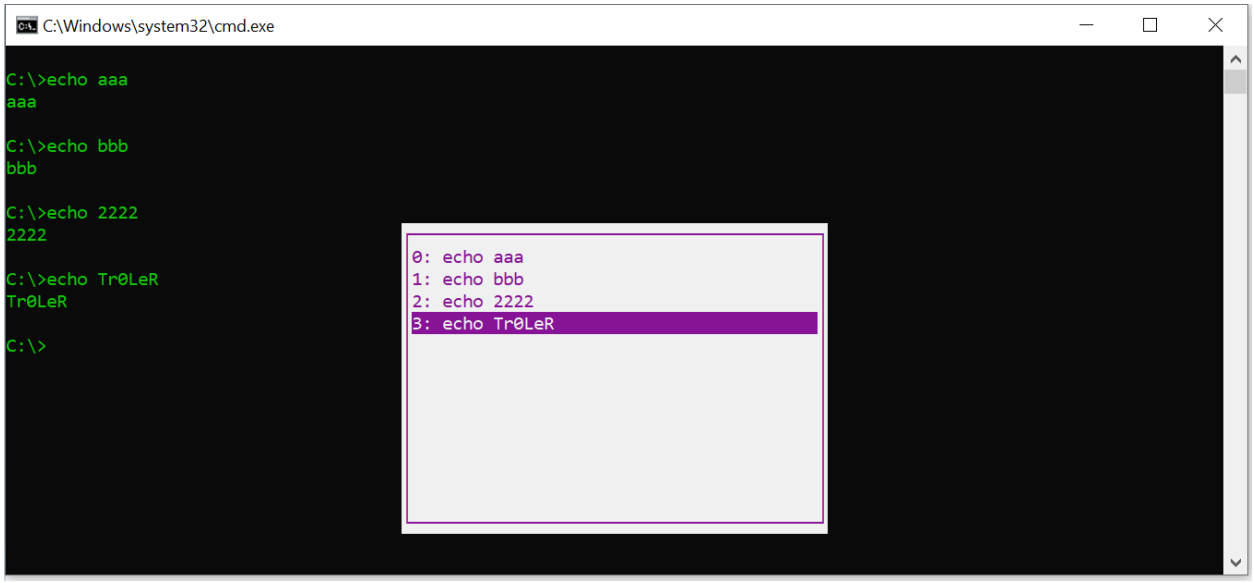


[204] https://ss64.com/nt/timeout.html
[205] https://ss64.com/nt/sleep.html
[206] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/timeout

doskey.exe (Keyboard History Utility)

"doskey.exe" (Keyboard History Utility) is a binary PE file located at "%windir%\system32\doskey.exe". It is a CLI (command line interface) utility which is used for recalling previously entered commands. Also, we can use it for editing commands and creating macros[207].

Moreover, after running "doskey.exe" we can use F7 in order to see the buffer/log/history of commands entered in a menu - as shown in the screenshot below. There are multiple keys/combinations that "doskey.exe" recognizes like "ALT+F7" which clears the history buffer and "End" which moves to the end of the line[208]. Lastly, we can go over a reference implementation of "doskey.exe" from ReactOS[209].



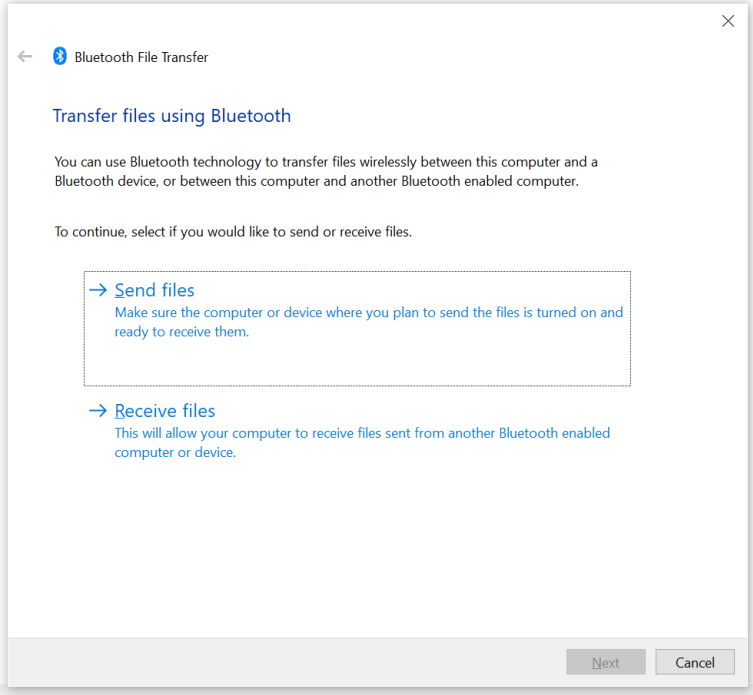
[207] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/doskey
[208] https://kb.iu.edu/d/aers
[209] https://github.com/reactos/reactos/tree/master/base/applications/cmdutils/doskey

fsquirt.exe (Bluetooth File Transfer)

"fsquirt.exe" is a PE binary located at "%windir%\System32\fsquirt.exe" which is used for sending/receiving files using Bluetooth. On 64-bit systems there is a 32-bit version located at "%windir%\SysWOW64\fsquirt.exe". By the way, the binary is also digitally signed by Microsoft.

Thus, "fsquirt.exe" is the default Bluetooth file transfer wizard on Windows systems[210]. The file transfer can be done between two computer that support Bluetooth, mobile phones or any other Bluetooth enabled devices[211].

Lastly, "fsquirt.exe" is also configured in the registry in the following registry location: "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths". The "App Paths" subkey is checked when the ShellExecuteExW[212] API function is called (The same goes for ShellExecuteExA). By registering an application using that subkey we can avoid the need for modifying the PATH environment variable[213].



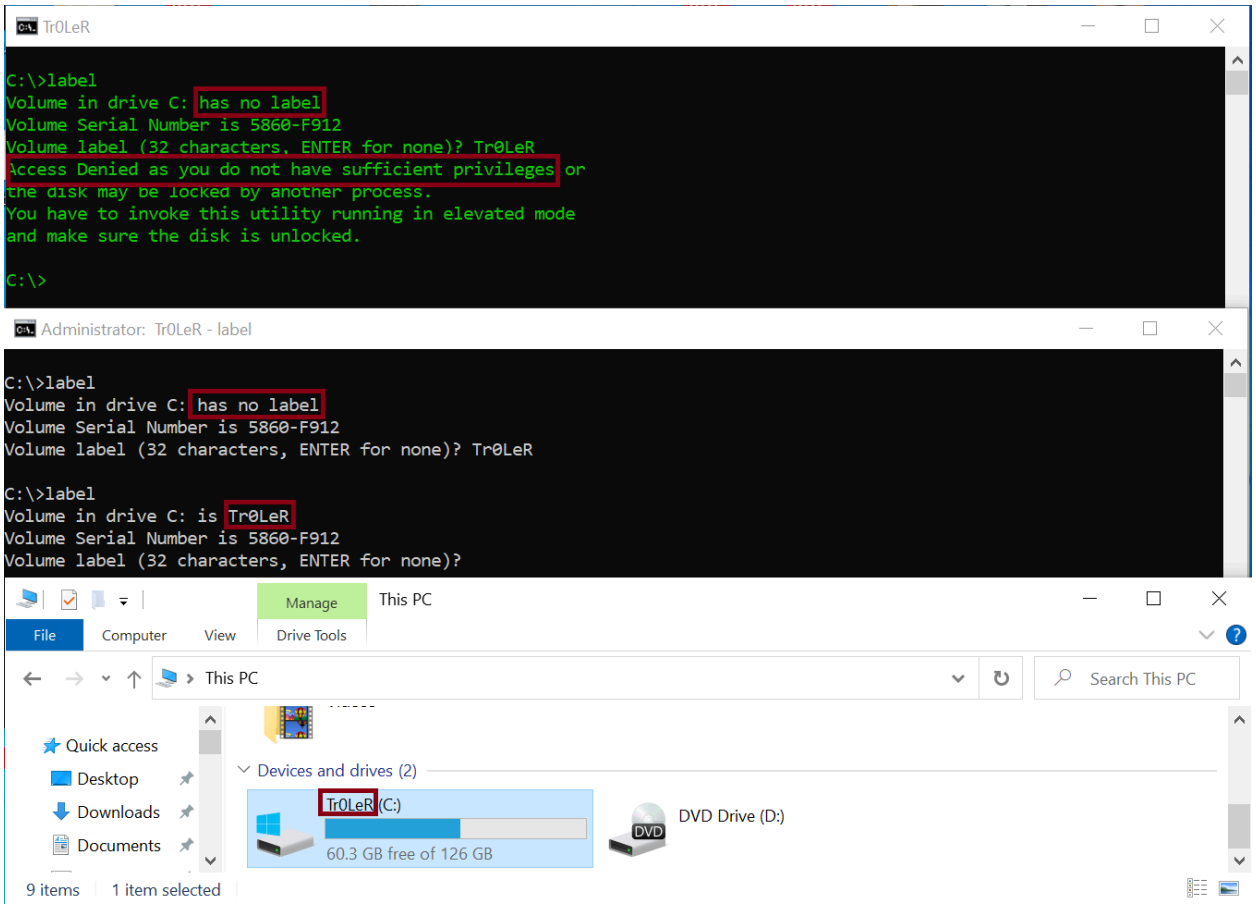
[210] https://renenyffenegger.ch/notes/Windows/dirs/Windows/System32/fsquirt_exe
[211] https://learn.microsoft.com/en-us/windows-hardware/drivers/bluetooth/bluetooth-user-interface
[212] https://learn.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecuteexW
[213] https://learn.microsoft.com/en-us/windows/win32/shell/app-registration

label.exe (Disk Label Utility)

“label.exe” (Disk Label Utility) is a binary PE file located at “%windir%\system32\label.exe”. It is a CLI (command line interface) utility which is used for creating/changing/deleting the volume label of a disk[214].

Moreover, on an NTFS volume we can use a label with up to 32 characters. On 64-bit systems there is also a 32-bit version on “label.exe” located at “%windir%\SysWOW64\label.exe”. Both versions of the PE are signed digitally by Microsoft.

The volume label is displayed in different places like in the “File Explorer” or the output of the “label.exe” - as marked in the screenshot below. In order to change the label there is a need for admin privileges - as shown in the screenshot below. Lastly, we can also go over a reference implementation of “label.exe” as part of ReactOS[215].



[214] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/label
[215] https://github.com/reactos/reactos/tree/master/base/applications/cmdutils/label

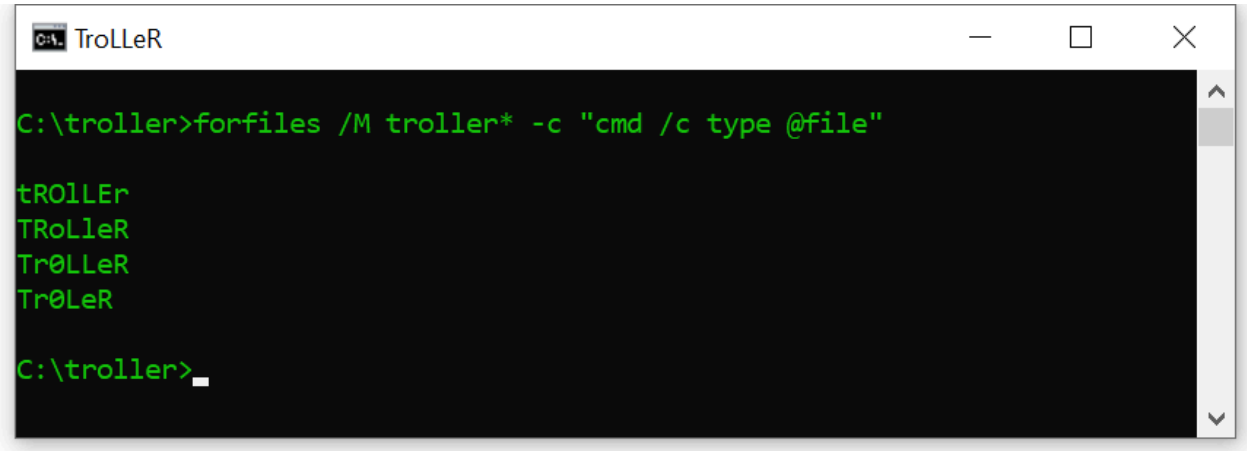
forfiles.exe (Execute a Command on Selected Files)

"forfiles.exe" is a binary PE file located at "%windir%\system32\forfiles.exe". It is a CLI (command line interface) utility which can be used in order to execute a command on selected files. On 64-bit versions of Windows there is also a 32-bit version of the binary located at "%windir%\SysWOW64\forfiles.exe". Also, the file is digitally signed by Microsoft.

Overall, "forfiles.exe" was included as part of Windows 98[216] and Windows 2000[217] resource kits, that means it was not part of the standard OS installation. Since Windows Vista it is part of the executables shipped with the OS[218].

Moreover, "forfiles.exe" has multiple command line parameters including: "/S" (recursive search), "/P" (specifying start directory), "/M" (search pattern mask), "/D" (selecting files by a last modification time frame), "/?" (displaying help text) and "/C" (specifying what command to run on each file). When using "/C" we can also use specific variables as part of the command like "@file" (the file name we are operating on), "@path" (the full path), "@ext" (the file extension) and more[219].

Lastly, we can see an example of using "forfiles.exe" in the screenshot below. In the screenshot we that for every file in the "C:\troller" directory with a "troller*" pattern in the file name we execute the type builtin command of "cmd.exe"[220].



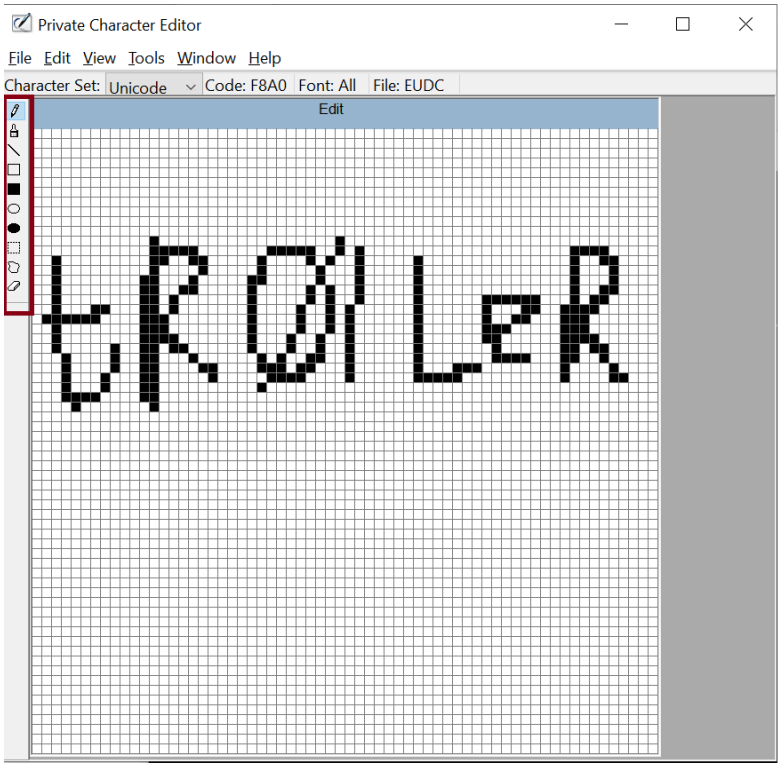
[216] https://web.archive.org/web/20200111203651/https://www.activexperts.com/admin/reskit/reskit98/forfiles/
[217] https://www.activexperts.com/admin/reskit/reskit2000/forfiles/
[218] https://web.archive.org/web/20061109021306/http://computerbits.wordpress.com/2006/07/21/new-command-line-tools-in-vista-beta-2/
[219] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/forfiles
[220] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b

eudcedit.exe (Private Character Editor)

“eudcedit.exe” is a PE binary located at “%windir%\System32\eudcedit.exe” it is known as the “Private Character Editor”. In case we want to use our own character/symbol (like in a document) we can use “eudcedit.exe”. Overall, it provides different tools for creating symbols/characters including: pencil, brush, eraser, hollow/filled eclipse/rectangles, straight line and rectangular/freeform selection[221].

Overall, we can create a character/symbol in one of two ways. First, creating a new custom one or second creating a custom one using a pre-existing character/symbol. By the way, on 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\eudcedit.exe”. The binary itself is also digitally signed by Microsoft.

Lastly, “eudcedit.exe” is configured to be auto elevated by default (based on the manifest information included in the binary itself “<autoElevate>true</autoElevate>”). In the screenshot below we can see an example of using the editor and all the mentioned tools marked in the left side of the UI.



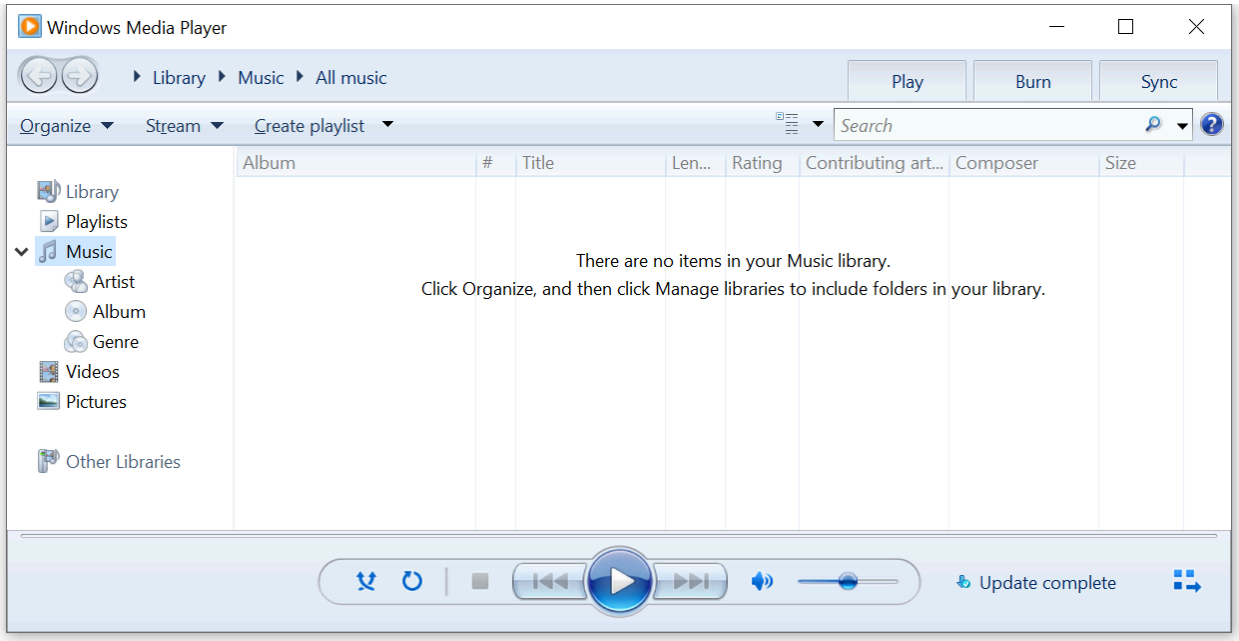
[221] https://www.thewindowsclub.com/charmap-and-eudcedit-windows-10

wmplayer.exe (Windows Media Player)

“wmplayer.exe” is a PE binary located at “%ProgramFiles(x86)%\Windows Media Player\wmplayer.exe”. It is used for lt as a media player, which is an application used for playing multimedia files (video and audio). It can also be used as a media library application - as shown in the screenshot below. By the way, WMP (Windows Media Player) has been included since Windows 3.x[222]. However, since 2022 it is marked as legacy while there is a new UWP based Media Player introduced in Windows 11[223].

Moreover, we can find the new version in the Windows Store. This version is relevant for Windows 10 (19042.0 or higher) on Mobile/PC/HoloLens/Xbox console/Surface Hub targeting x86/x64/Arm64 architectures[224].

Overall, the “wmplayer.exe” which is executed by default is the 32-bit version of WMP. There could also be a 64-bit version in the following location: “%ProgramFiles%\Windows Media Player\wmplayer.exe”. By the way, both versions are digitally signed by Microsoft.



[222] https://www.youtube.com/watch?v=imAUwsksUlY
[223] https://en.wikipedia.org/wiki/Windows_Media_Player
[224] https://apps.microsoft.com/detail/9WZDNCRFJ3PT

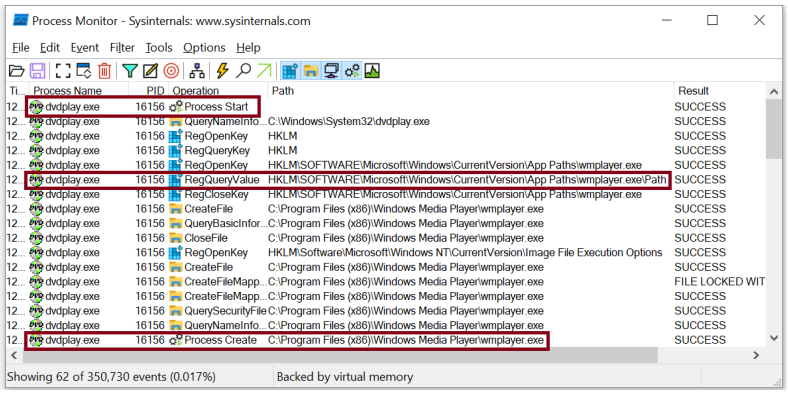
dvdplay.exe (DVD Play Placeholder Application)

“dvdplay.exe” is a PE binary located at “%windir%\System32\dvdplay.exe”. It is used for launching an application which is capable of playing DVD disks. On 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\dvdplay.exe”. The binary is also digitally signed by Microsoft.

On old versions of Windows (like Windows ME), “dvdplay.exe” was its own application - as shown in the screenshot below[225]. However, in new versions (like Windows 10) it is basically launching “wmplayer.exe” which is the “Windows Media Player”[226].

Thus, “dvdplayer.exe” calls the API function “RegGetValueW”[227] in order to read the path of “wmplayer.exe” from the application registration in the registry “HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\wmplayer.exe\Path”. Later, it checks if the file exists using the API call “SearchPathW”[228]. If the file is found it is started using the API call “CreateProcessW”[229].

Lastly, the flow described above aligns with the description found in the PE header which states it is a palace holder application. This flow is also shown in the screenshot below taken from Sysinternals’ “Process Monitor” on Windows 10.



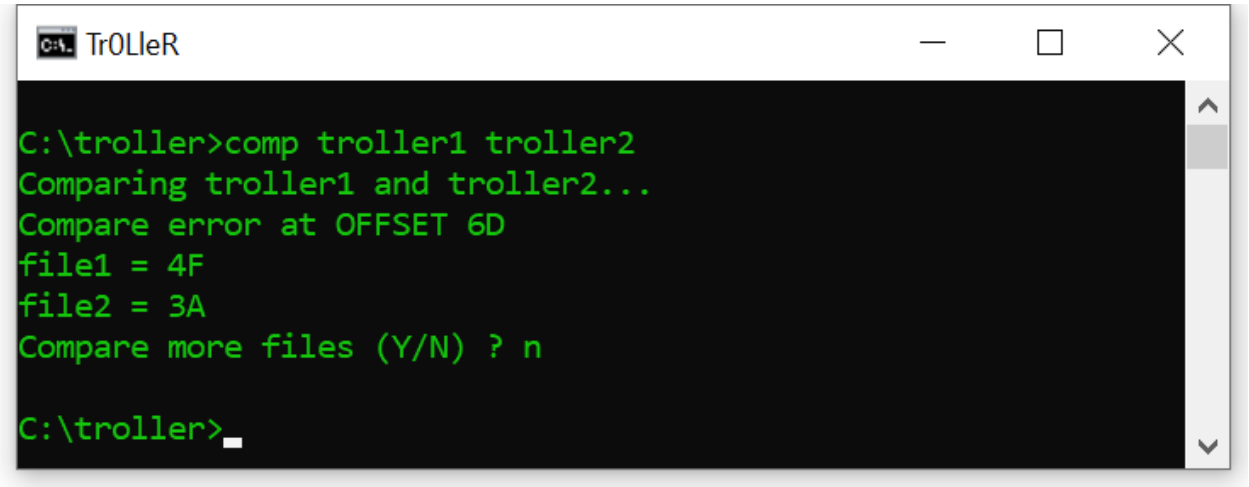
[225] www.activewin.com/tips/tips/microsoft/winme/b3.shtml
[226] https://medium.com/@boutnaru/the-windows-process-journey-wmplayer-exe-windows-media-player-7d25c370c526
[227] https://learn.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-reggetvaluew
[228] https://learn.microsoft.com/en-us/windows/win32/api/processenv/nf-processenv-searchpathw
[229] https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw

comp.exe (File Compare Utility)

"comp.exe" is a PE binary located at "%windir%\System32\comp.exe". It is used for comparing the content of two files/set of files byte-by-byte. The files compared may be located on the same drive/directory or on different drive/directory. On 64-bit systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\comp.exe"[230].

Moreover, the files which are compared can also be in a remote location (SMB share). In case there is a difference between the compared files the offsets of change with the different values are displayed - as shown in the screenshot below. By the way, the "comp.exe" binary is also digitally signed by Microsoft.

Lastly, by using command line arguments we can display the difference in decimal (hex is the default), compare only a specific number of lines, display the difference in ascii characters and more[231]. Also, there is a reference implementation of "comp.exe" as part of ReactOS[232].



[230] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/comp
[231] https://ss64.com/nt/comp.html
[232] https://github.com/reactos/reactos/tree/master/base/applications/cmdutils/comp

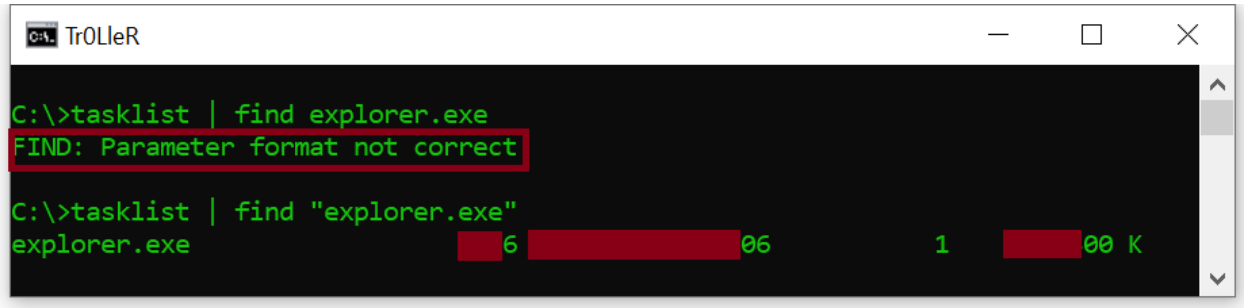
find.exe (Find String (grep) Utility)

“find.exe” is a PE binary located at “%windir%\System32\find.exe”. On 64-bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\find.exe”. Both of the versions are digitally signed by Microsoft. It is used in order to search for patterns of text files and sends them to the standard input device. Thus, we can use it to filter/find a specific string using wildcard characters[233].

Overall, we can compare the functionality of “find.exe” to those of the “grep” utility[234] which is widely used under Unix/Linux systems. On the other hand it is completely different from the “find”[235] utility used in Unix/Linux systems which is similar to the “forfiles.exe”[236] .

Moreover, “find.exe” has different command line switches for: displaying all lines not containing a specific string (“/V”), counting the number of lines containing a string (“/C”), displaying line numbers (“/N”) and ignoring the case of characters while searching (“/I”). Also, we can skip (or not) files that have the offline attribute set[237].

Lastly, we can provide a path/s to file/s (including wildcards) we want to search in their content, pass a standard output of a command as input or just get the input for a prompt by “find.exe”. It is important to understand that the string we want to search for must be in quotes - as shown in the screenshot below.



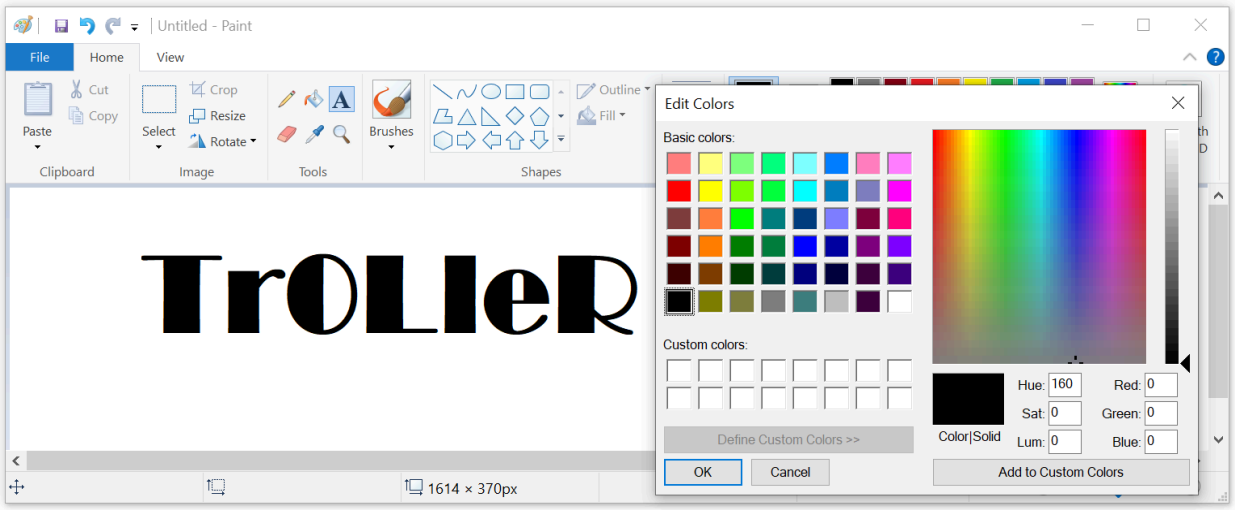
[233] https://en.wikipedia.org/wiki/Find_(Windows)
[234] https://man7.org/linux/man-pages/man1/grep.1.html
[235] https://man7.org/linux/man-pages/man1/find.1.html
[236] https://medium.com/@boutnaru/the-windows-process-journey-forfiles-exe-execute-a-command-on-selected-files-3c10a9b2b5cf
[237] https://ss64.com/nt/find.html

mspaint.exe (Paint)

“mspaint.exe” is a PE binary located at “%windir%\System32\mspaint.exe”. On 64-bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\mspaint.exe”. Both of the versions are digitally signed by Microsoft. It is a simple graphic/drawing editor included as part of the Windows operating system since Windows 1.0. “mspaint.exe” different editing tools like brushes, shape generators, pens, eraser, color selection, bucket (fill with color) and magnifier[238] - as shown in the screenshot below (It is the Windows 10 version).

Overall, “mspaint.exe” supports different image formats like: Windows bitmap (BMP), PNG, GIF, JPG and single-page TIFF. By the way, AI art generators (DALL-E based) are going to be part of Microsoft Paint[239].

Moreover, support for layers (adding/removing/merging/duplicating/etc) and support for opening/saving transparent PNG files had been added to paint[240]. Those features fit together with the ability to remove the background of an image[241]. Lastly, we can check out the reference implementation of “mspaint.exe” as part of ReactOS[242].



[238] https://mspaint.humanhead.com/#local:bd525d07a1f88
[239] https://en.wikipedia.org/wiki/Microsoft_Paint
[240] https://www.theverge.com/2023/9/18/23879221/microsoft-paint-testing-layers-transparency-photoshop-features
[241] https://www.theverge.com/2023/9/7/23863377/microsoft-paint-background-removal-tool
[242] https://github.com/reactos/reactos/tree/master/base/applications/mspaint

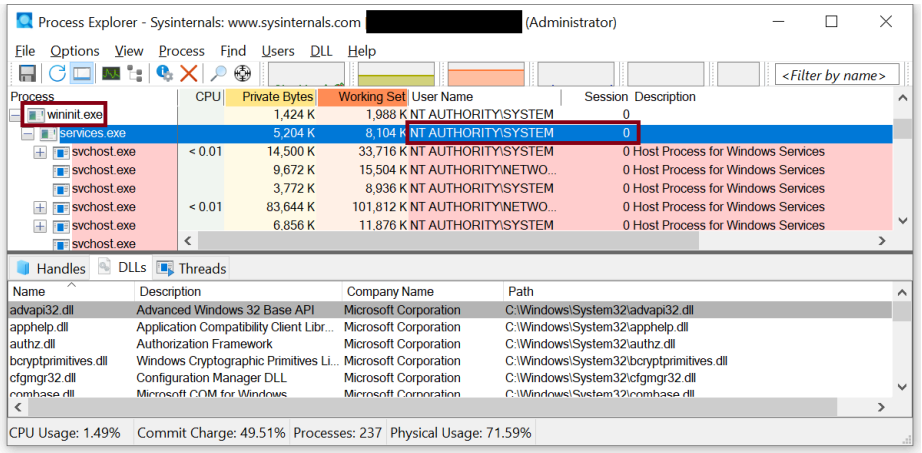
services.exe (Service Control Manager)

"services.exe" is a PE binary located at "%windir%\System32\services.exe". It is part of the "Service Control Manager" (SCM), it provides an RPC (Remote Procedure Call) server ("RPC Control\ntsvcs"). By leveraging it, programs can manipulate and configure Windows services[243] locally or remotely[244]. A reference implementation of "services.exe" can be found as part of ReactOS[245].

Overall, "services.exe" is started when Windows starts. It is launched by "wininit.exe"[246] on session 0 and is executed with the permissions and privileges of the "NT AUTHORITY\SYSTEM" (S-1-5-18) aka "Local System". The binary is digitally signed by Microsoft. There are two built-in major tools for communicating with the SCM: "sc.exe" and the MMC snap-in "services.msc"[247] .

Moreover, the SCM provides an interface for performing various tasks as described next. Starting services/drivers on startup/demand. Maintaining/locking/unlocking the database of installed services (HKLM\SYSTEM\CurrentControlSet\Services). Transmitting control requests for running services. Maintaining the status of running drivers and services[248].

Lastly, it should be executed only once on a Windows system regardless of the number of logged in users. By the way, on 64-bit systems unlike other Windows binaries (like "cmd.exe") we don't have a parallel 32-bit version of "services.exe". We can also use the Win32 API for manipulating services[249]. The client-side API for the SCM is implemented as part of "%windir%\system32\advapi32.dll"[250].



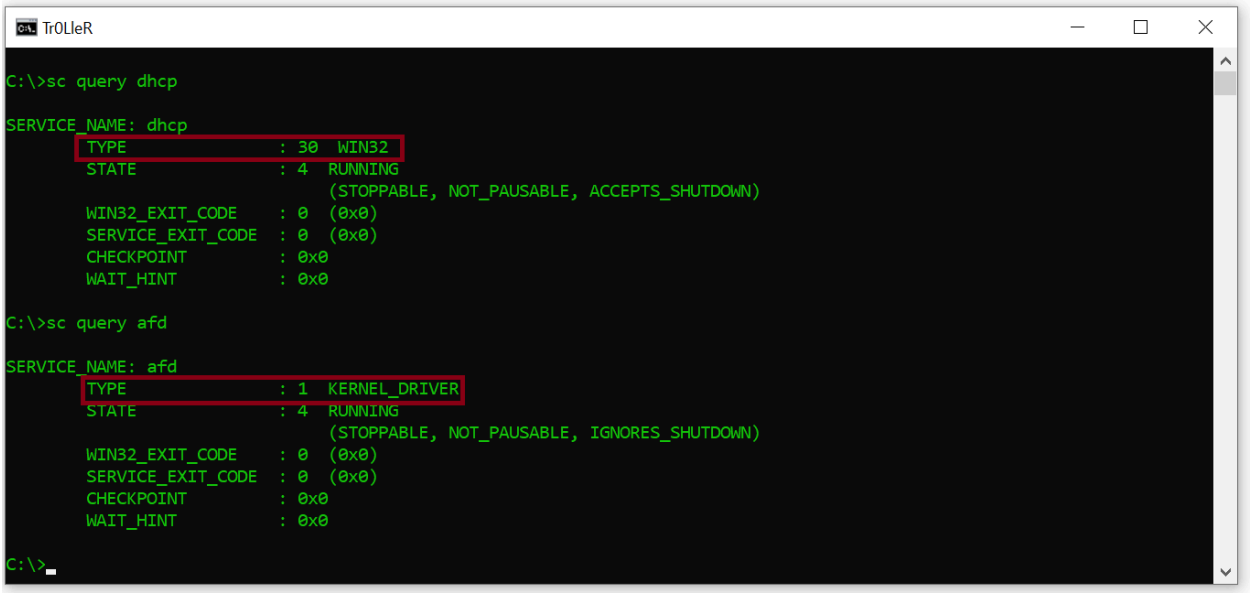
[243] https://medium.com/@boutnaru/windows-services-part-2-7e2bdab5bce4
[244] https://publik.tuwien.ac.at/files/publik_273621.pdf
[245] https://github.com/reactos/reactos/tree/master/base/system/services
[246] https://medium.com/@boutnaru/the-windows-process-journey-wininit-exe-windows-start-up-application-5581bfe6a01e
[247] https://medium.com/@boutnaru/the-windows-process-journey-mmc-exe-microsoft-management-console-a584afe66d86
[248] https://learn.microsoft.com/en-us/windows/win32/services/service-control-manager
[249] https://learn.microsoft.com/en-us/windows/win32/api/winsvc/
[250] https://renenyffenegger.ch/notes/Windows/dirs/Windows/System32/services_exe/index

sc.exe (Service Control Manager Configuration Tool)

"sc.exe" is a PE binary located at "%windir%\System32\sc.exe". By the way, on 64-bit systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\sc.exe". Both files are digitally signed by Microsoft.

Overall, "sc.exe" is used to create/stop/start/query/delete/pause/configure/etc any Windows service[251]. For example, "sc.exe query <servicename>" is done by reading a subkey/entries of the service in the SCM (Service Control Manager) database - as shown in the screenshot below[252]. The SCM database is located in the registry in the following location: "HKLM\SYSTEM\CurrentControlSet\Services".

Moreover, there are other command line options that can be used with "sc.exe" such as (but not limited to) viewing the security descriptor of the service ("sdshow"), showing/changing the description ("qdescription/description"), displaying/modifying the actions that are taken by the service in case of a failure ("qfailure/failure"), showing dependencies ("EnumDepend") and creating/deleting a service ("create/delete"). By the way, "sc.exe" is also used for managing drivers, which are defined as services which execute in kernel mode - as shown in the screenshot below - more on that in future writeups[253]. Lastly, we can go over a reference implementation of "sc.exe" which is part of ReactOS[254].



[251] https://medium.com/@boutnaru/windows-services-part-2-7e2bdab5bce4
[252] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/sc-query
[253] https://ss64.com/nt/sc.html
[254] https://github.com/reactos/reactos/tree/master/base/applications/sc

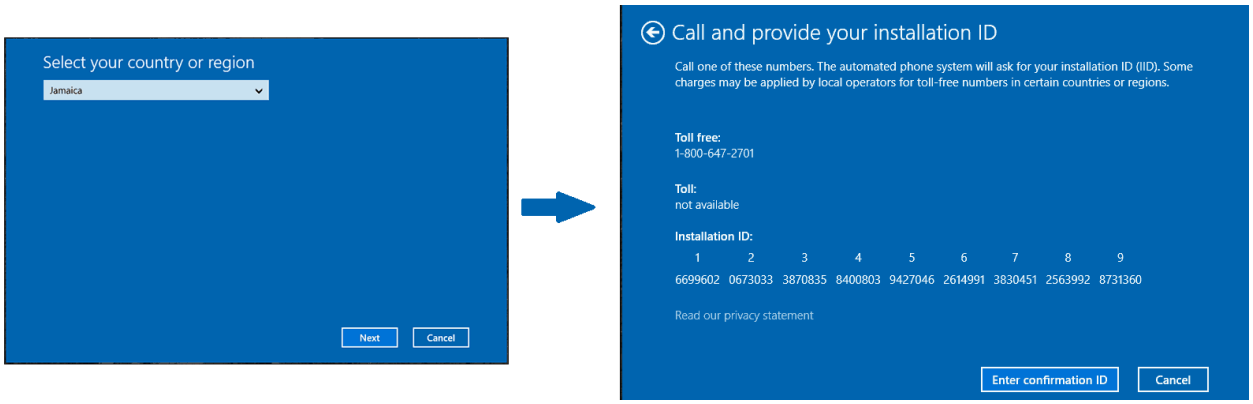
phoneactivate.exe (Phone Activation UI)

"phoneactivate.exe" is a PE binary located at "%windir%\System32\phoneactivate.exe". Unlike other binaries there is no 32-bit version of it in Windows 64-bit systems (as we have with "cmd.exe" for example). The binary is digitally signed by Microsoft.

Overall, we can activate Windows using an internet connection (aka Online activation). Also, we can activate Windows by phone. In this case we try activating our device over the phone, this connects us to Microsoft support for our region and country[255].

Thus, the goal of "phoneactivate.exe" is to provide the phone activation UI (User Interface). One common use case for using it is if the Windows license was used in another computer. After the phone activation is launched we need to choose our country and select next - as shown in the screenshot below. Then, using the phone numbers shown on the screen we can call the support agent and provide the installation ID - also shown in the screenshot below[256].

Lastly, after verifying the product key and using the installation ID the agent will provide a confirmation ID for activating Windows. By the way, we can also launch "Contact Support" and use a chat versus calling.



[255] https://support.microsoft.com/en-us/windows/product-activation-for-windows-online-support-telephone-numbers-35f6a805-1259-88b4-f5e9-b52cccef91a0

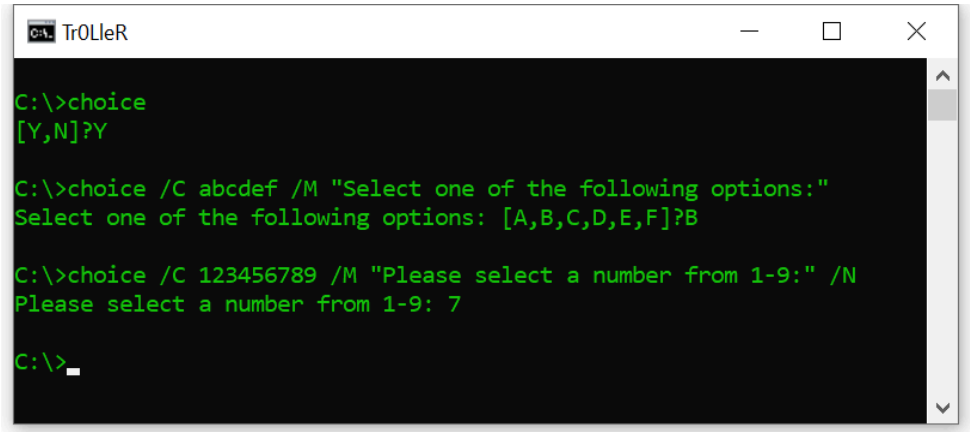
[256] https://www.groovypost.com/howto/save-windows-10-spotlight-lock-screen-pictures/

choice.exe (Offers the User a Choice)

“choice.exe” (Offers the user a choice) is a PE binary located at “%windir%\system32\choice.exe”. It is used for allowing users to select one (single key pressed) item from a list of choices, it returns the index of the selected choice. By default, we can choose between “Y” or “N” - as shown in the screenshot below[257].

Moreover, we can customize the list of options and a text shown to the user using the different switches of “choice.exe” (“/C” and /”M” respectively) - as shown in the screenshot below. There are also other switches that allow us to control behavior of the command like: specify if the choices are case-sensitive (“/CS”), timeout for selecting one of the choices (“/T”) and more[258].

Lastly, on 64-bit systems there is also a 32-bit version of “choice.exe” located at “%windir%\SysWOW64\choice.exe”. Both the 64-bit version and the 32-bit version are digitally signed by Microsoft.



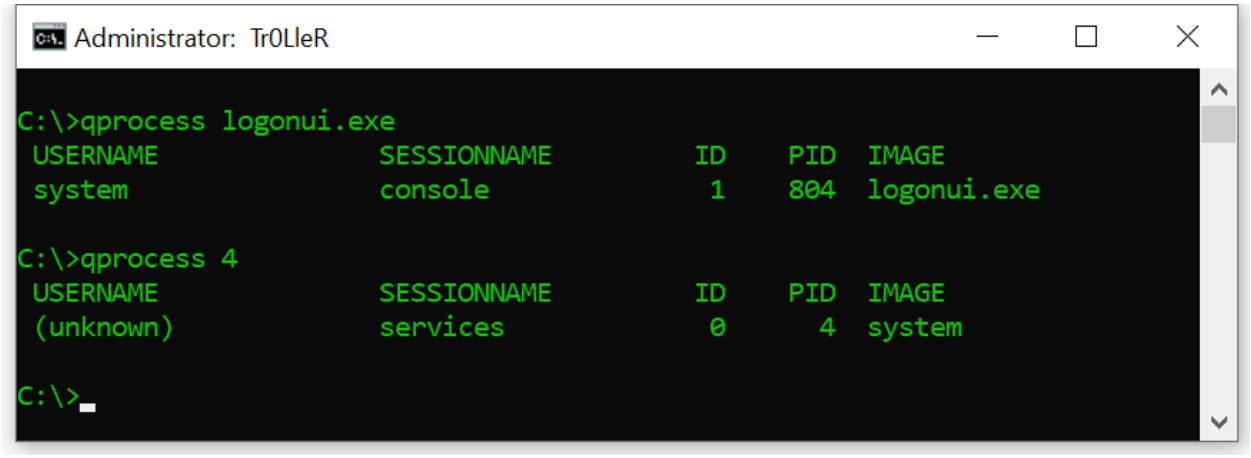
[257] https://ss64.com/nt/choice.html
[258] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/choice

qprocess.exe (Query Process Utility)

"qprocess.exe" is a PE binary located at "%windir%\System32\qprocess.exe". It is used for displaying information about processes. Also, it supports displaying information about processes that have been executed on a Remote Desktop Session Host Server [259].

Moreover, as opposed to other executables like "cmd.exe"[260], on 64-bit versions of Windows there is no 32-bit version of "qprocess.exe". The binary itself is digitally signed by Microsoft.

Lastly, "qprocess.exe" provides different command line switches. Using them we can list all processes for all sessions ("*"), display processes based on/process id/username/session name/session ID/program name[261] - as shown in the screenshot below.



[259] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/qprocess
[260] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[261] https://ss64.com/nt/query-process.html

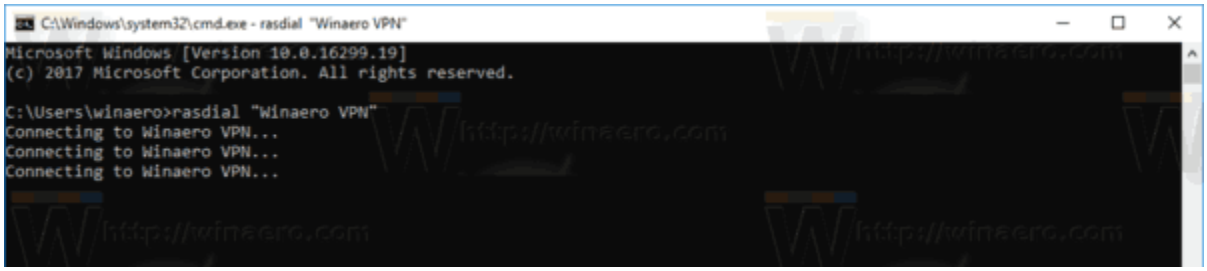
rasdial.exe (Remote Access Command Line Dial UI)

“rasdial.exe” is a PE binary located at “%windir%\System32\rasdial.exe”. It is used for connecting/disconnecting from a VPN (Virtual Private Network)/dial up connection[262].

Overall, on 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\rasdial.exe”. Both the 64-bit version and the 32-bit version are digitally signed by Microsoft.

Moreover, using the command line switches of “rasdial.exe” we can provide different information for a connection. Examples are : a username for connection, a password, a phone number to connect and a callback number. In case we execute “rasdail.exe” without any arguments the status of the current connection is displayed[263].

Lastly, to specify credentials (username and password) we can execute the following command: “ rasdial ‘ConnectionName’ ‘Username’ ‘Password’ ”[264].



[262] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/ff859533(v=ws.11)
[263] https://ss64.com/nt/rasdial.html
[264] https://gist.github.com/stormwild/ec0898fe8bf25f58f4a6bf2576dc5e3f

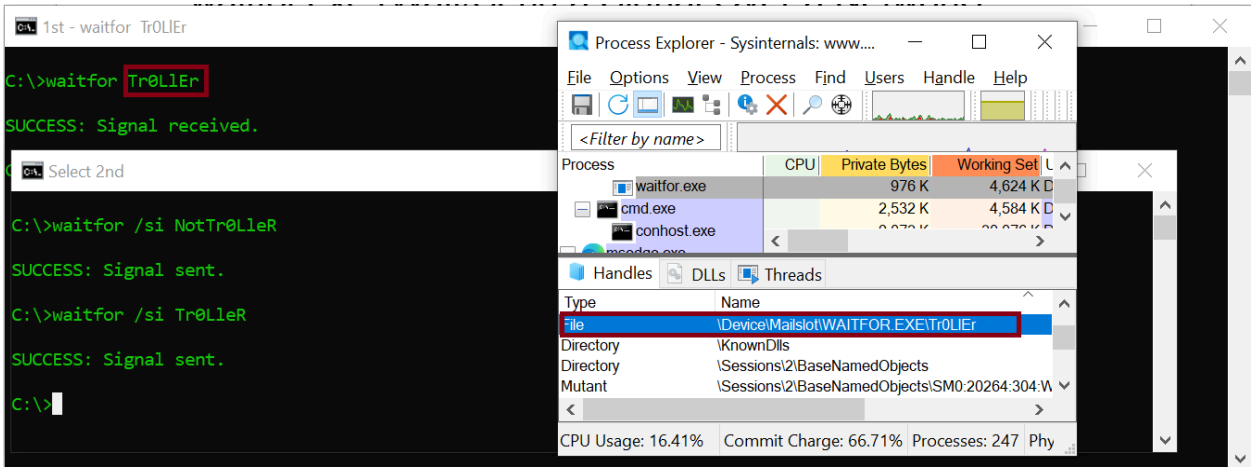
waitfor.exe (Wait/Send a Signal Over a Network)

"waitfor.exe" is a PE binary located at "%windir%\System32\waitfor.exe". It is used for sending/waiting for a signal on a system. We can also use "waitfor.exe" in order to synchronize between computer systems over the network[265].By the way, on 64-bit systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\waitfor.exe". Both the 32-bit version and the 64-bit version are digitally signed by Microsoft.

Overall, "waitfor.exe" is based on the mailslot[266] IPC mechanism. When selecting a name for a signal to wait for, it is used as part of the naming of the mailslot using the following format "\\.\mailslot\WAITFOR.EXE\[SIGNAL NAME]" - as shown in the screenshot below. The signal itself is not case sensitive (the same as files in Windows).

Moreover, when using "waitfor.exe" for remote synchronization we can provide the username/password for authentication using the command line switches ("/u" and "/p" respectively) and "/" for providing the name/IP of the remote system[267].

Lasly, we can think of "waitfor.exe" as a combination of the Linux commands "kill"[268] and the "trap" command[269]. The first can send signals and the second one can wait for signals. Also, "tap" can be implemented in different ways such as a builtin command of a shell.



[265] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/waitfor
[266] https://medium.com/@boutnaru/the-windows-concept-jou-d35f84d8cc02
[267] https://ss64.com/nt/waitfor.html
[268] https://man7.org/linux/man-pages/man1/kill.1.html
[269] https://man7.org/linux/man-pages/man1/trap.1p.html

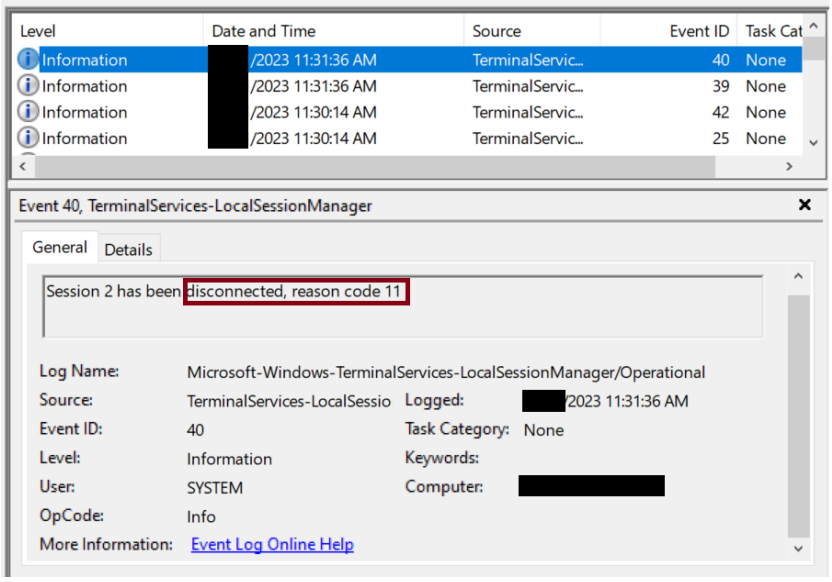
tsdiscon.exe (Session Disconnection Utility)

"tsdiscon.exe" is a PE binary located at "%windir%\System32\tsdiscon.exe". It is used for disconnecting from a remote desktop services session. By the way, on 64-bit systems unlike other binaries like "cmd.exe"[270] there is not 32-bit version of "tsdison.exe" in parallel to the 64-bit version.

Overall, using different switches we can specify the ID of the session or the session name that we want to disconnect. Also, we can provide the name of the terminal server containing the session we want to disconnect ("/server:<SERVER_NAME>). By the way, if we don't provide any session ID/name the current session is going to be disconnected[271].

Moreover, there should not be any data loss when disconnecting from a session. The applications are still running, thus we can reconnect to the session. We must have full control permissions/disconnect permissions in order to disconnect another user from a session[272]. This can also be done for sessions within a virtual machine.

Lastly, when executing "tsdiscon.exe" an event is logged (ID 40) in the event viewer under the following location "Applications and Services Logs -> Microsoft -> Windows -> TerminalServices-LocalSessionManager -> Operational" - as shown in the screenshot below. By the way, "reason code 11" means the user disconnecting from the session initiates the disconnection[273].



[270] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[271] https://ss64.com/nt/tsdiscon.html
[272] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/tsdiscon
[273] https://www.anyviewer.com/how-to/session-has-been-disconnected-reason-code-0-2578.html

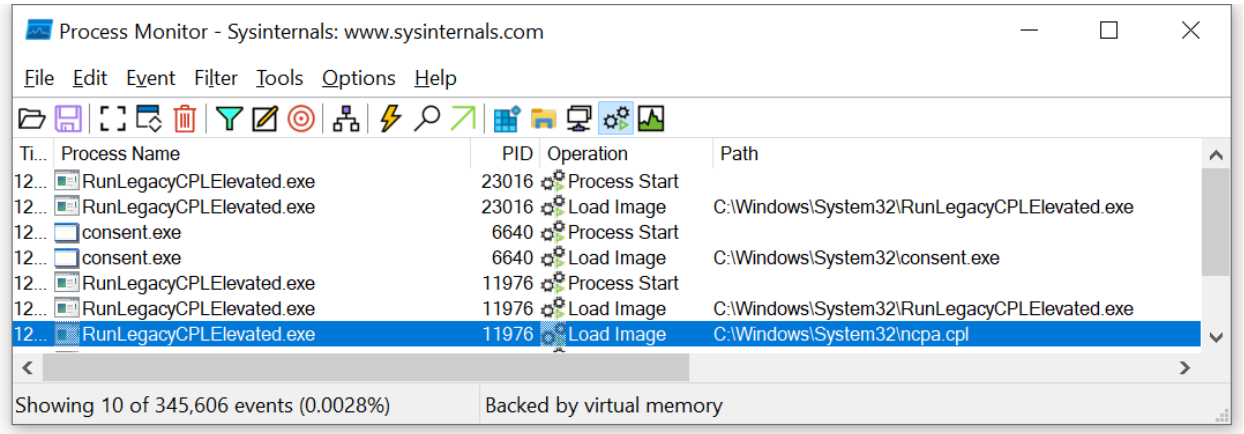
RunLegacyCPLElevated.exe (Running Legacy Control Panel Applet in Elevated Mode)

"RunLegacyCPLElevated.exe" is a PE binary located at "%windir%\System32\RunLegacyCPLElevated.exe". It is used for running a legacy control panel applet in elevated mode. On 64-bit Windows systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\RunLegacyCPLElevated.exe". By the way, both binaries are digitally signed by Microsoft.

Overall, we should execute "RunLegacyCPLElevated.exe" using the following arguments "RunLegacyCPLElevated.exe shell32.dll, Control_RunDLL <CPL_FILE_PATH_TO_LOAD>". An example of execution is "RunLegacyCPLElevated.exe shell32.dll, Control_RunDLL %windir%\system32\ncpa.cpl" - as shown in the screenshot below.

Moreover, when executing the binary the chain of execution is as follows: "RunLegacyCPLElevated.exe" performs an RPC call to execute "consent.exe"[274], which is started by the "Application Information" service (hosted by svchost.exe). After that "RunLegacyCPLElevated.exe" is executed again with the same arguments using the elevated access token, this is the process that loads and executes the function for the "*.cpl" file - as shown in the screenshot below.

Lastly, we can think about "RunLegacyCPLElevated.exe" as a "rundll32.exe"[275] which starts the control panel applet with high permissions. Thus, it is similar to executing (without the elevation part) to "rundll32.exe shell32.dll, Control_RunDLL %windir%\system32\ncpa.cpl".



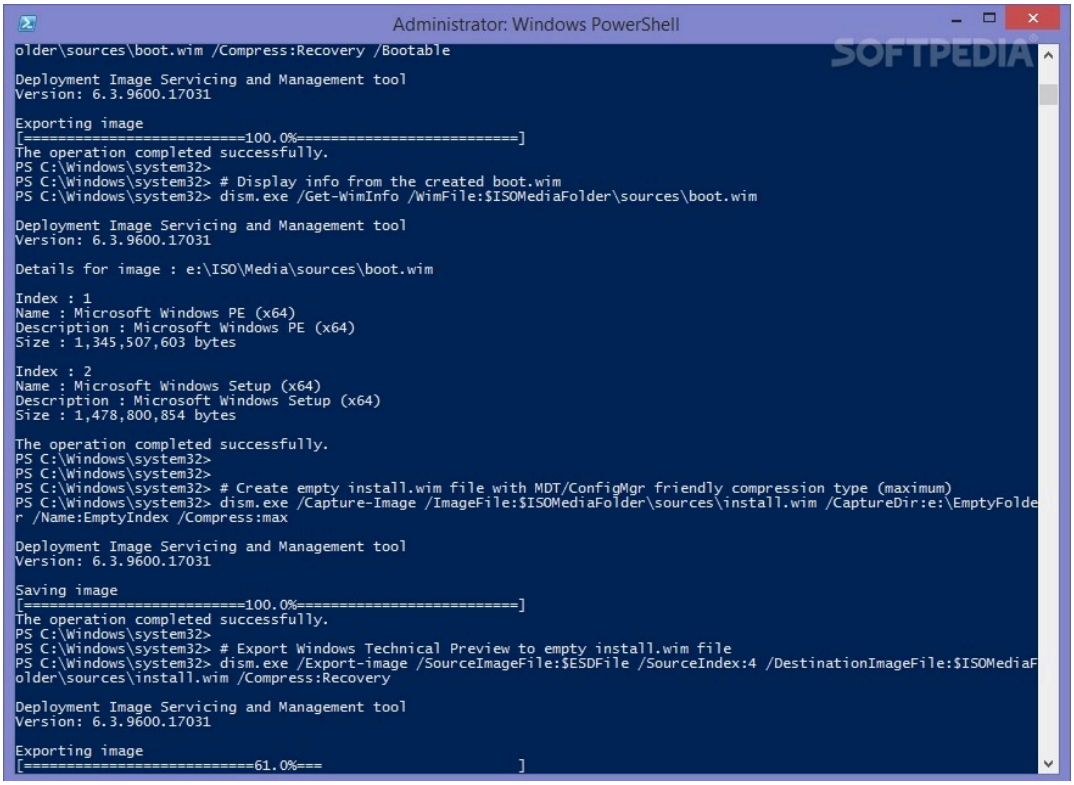
[274] https://medium.com/@boutnaru/the-windows-process-journey-consent-exe-consent-ui-for-administrative-applications-d8e6976e8e40
[275] https://medium.com/@boutnaru/the-windows-process-journey-rundll32-exe-windows-host-process-415132f1363

dism.exe (Deployment Image Servicing and Management Tool)

“dism.exe” is a PE binary located at “%windir%\System32\dism.exe”. We can use it in order to enumerate/install/uninstall/configure/update features and packages as part of the Windows operating system[276]. On 64 bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\Dism.exe”. Both binaries are digitally signed by Microsoft.

Overall, “dism.exe” can be used to prepare/service “Windows Images” that can be used for Windows PE/Windows RE (Recovery Environment)/Windows Setup. It can also service “*.wim” (Windows Image) files or “*.vhd”/”*.vhdx” (virtual hard disks) files[277].

Lastly, “dism.exe” can be executed with elevated permissions which allows parsing of information of image files and saving changes - as shown in the screenshot below[278]. Thus, “dism.exe” can modify offline image files in the different ways such as: ways: add language packs, add package updates, enable/disable OS features, combine images, adding device drivers[279].



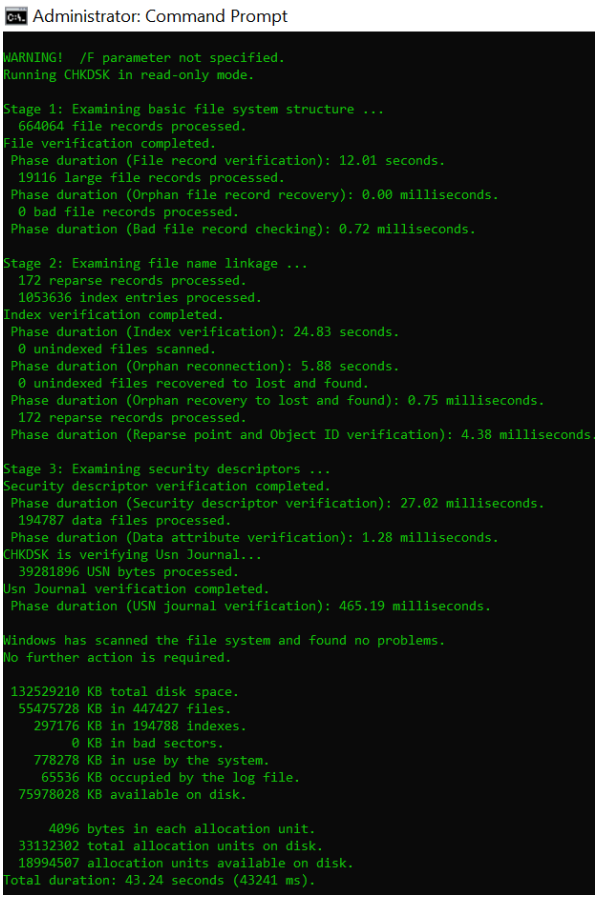
[276] https://ss64.com/nt/dism.html
[277] https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/what-is-dism?view=windows-11
[278] https://shopperlasopa179.weebly.com/dismexe-wim.html
[279] https://www.slideserve.com/akamu/cn1176-computer-support-powerpoint-ppt-presentation

chkdsk.exe (Check Disk Utility)

“chkdsk.exe” (Check Disk Utility) is a PE binary located at “%windir%\System32\chkdsk.exe”. On 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\chkdsk.exe”. It is used to check the file-system/file-system metadata of a volume for logical/physical errors. In order to execute it the user needs to be a member of the local administrator group[280].

Moreover, “chkdsk.exe” can not only scan for errors but also fix some of them based on the different switches given when executing it. If no parameter was given it will run in read-only mode - as shown in the screenshot below. For fixing structural issues we can use “/f” and to try recovering data from corrupted parts of the physical drive we can also add “/r”. To dismount the drive for scanning and fixing we should use “/x”[281].

Lastly, “chkdsk.exe” is a CLI tool which is digitally signed by Microsoft. When running a check “chkdsk.exe” performs 3 main stages: examination of basic filesystem structure, examination of file name linkage and examination of security descriptors - as shown in the screenshot below.



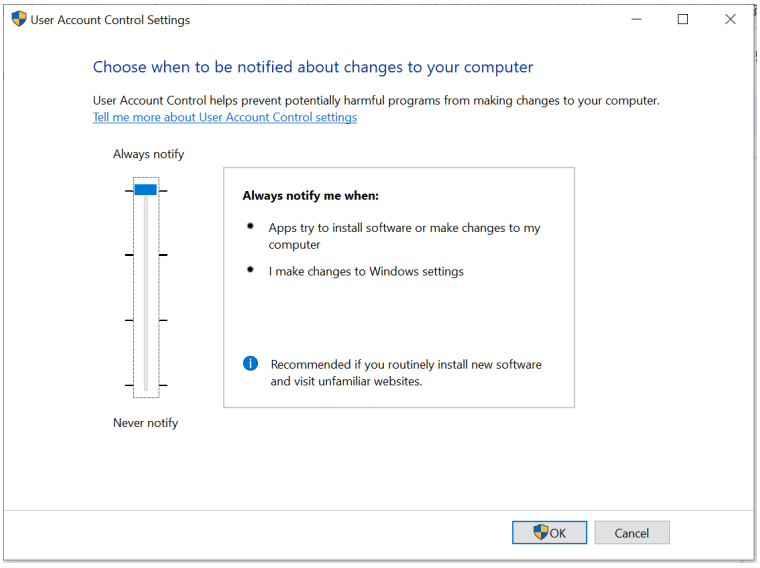
[280] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/chkdsk?tabs=event-viewer
[281] https://www.avg.com/en/signal/how-to-use-chkdsk-windows

UserAccountControlSettings.exe (Configuring UAC Settings)

"UserAccountControlSettings.exe" is a PE binary file located at "%windir%\system32\UserAccountControlSettings.exe". On 64-bit systems there is also a 32-bit version of the file located at "%windir%\SysWOW64\UserAccountControlSettings.exe". It is used in order to change the settings of UAC (User Account Control)[282]. The binary is digitally signed by Microsoft.

Overall, "UserAccountControlSettings.exe" allows a user to select the level of notifications in case apps try to install software/change computer settings or whether the user itself tries to do those things[283]. There are a total of four levels that we can select from (using the slider) - as shown in the screenshot below.

First, the lower one is to never notify (whether app/user is trying to install software making changes to Windows settings). Second, notify only if apps are trying to make changes (not relevant if the user does that), by the way the desktop won't be dimmed. Third, as the previous but dims the desktop (meaning using the secure desktop), it is also the default setting. Fourth, notify if an app/user is trying to install software/make changes to the Windows settings.



[282] https://renenyffenegger.ch/notes/Windows/dirs/Windows/System32/UserAccountControlSettings_exe
[283] https://www.elevenforum.com/t/change-user-account-control-uac-settings-in-windows-11.1523

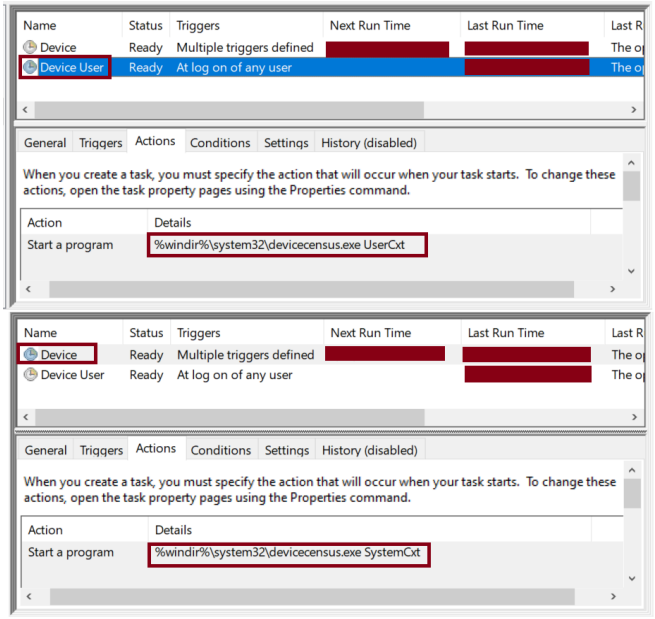
DeviceCensus.exe (Device Information)

"DeviceCensus.exe" is a PE binary located at "%windir%\System32\DeviceCensus.exe". As opposed to other executables such as "cmd.exe"[284] there is only a 64-bit version of "DeviceCensus.exe" as part of a 64-bit version of Windows (no parallel 32-bit version). By the way, the binary is digitally signed by Microsoft.

Overall, "DeviceCensus.exe" is executed by the "Task Scheduler"[285] on Windows. There are two tasks which are configured by default to run "DeviceCensus.exe": "Device" and "Device User". Both of them can be found in the following location in the "Task Scheduler Library": "Microsoft\Windows\Device Information" - as shown in the screenshot below. The second one is executed at log on of every user.

Moreover, "DeviceCensus.exe" accepts as command line arguments the following: "SystemCxt" (used by the "Device" task) and "UserCxt" (used by the "Device User" task). Each flow which is triggered based on them calls exported functions from "%windir%\system32\dcntel.dll". The first one calls the "RunSystemContextCensus" function and the second calls the "RunUserContextCensus" function.

Lastly, based on different documentation "DeviceCensus.exe" helps Microsoft improve user experience by understanding how their products are being used. It is used to collect information like hardware in use, performance data and most used features. Thus, it is part of telemetry data collection in Windows[286].



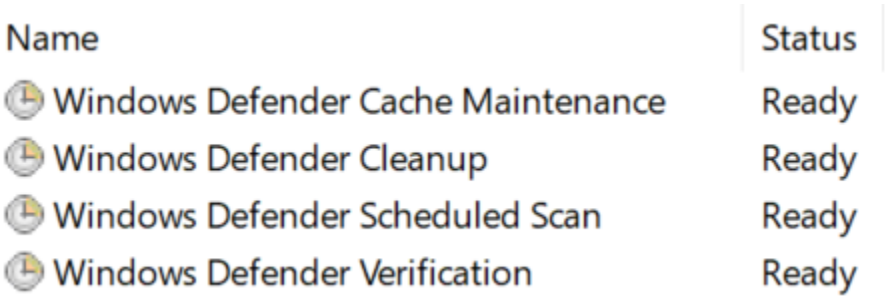
[284] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[285] https://medium.com/@boutnaru/windows-scheduler-tasks-84d14fe733c0
[286] https://www.file.net/process/devicecensus.exe.html

MpCmdRun.exe (Microsoft Malware Protection Command Line Utility)

“MpCmdRun.exe” is a PE binary located at “C:\ProgramData\Microsoft\Windows Defender\Platform\[VERSION]\MpCmdRun.exe”. By the way, [VERSION] matches the file version stored in the PE. Its description states it is the “Microsoft Malware Protection Command Line Utility”. Also, the binary is also digitally signed by Microsoft. By the way, it is also called “Microsoft Defender Antivirus command-line utility” as part of the Microsoft documentation[287]. It is used as a command line frontend for “Microsoft Malware Protection”.

Moreover, by default there are four Windows schedule tasks[288] which are based on “MpCmdRun.exe” as their action: “Windows Defender Cache Maintenance” (periodic maintenance task), “Windows Defender Cleanup” (periodic cleanup task), “Windows Defender Scheduled Scan” (periodic scan task) and “Windows Defender Verification” (periodic verification task) - as shown in the screenshot below. We can find all of them in the following location : “Task Scheduler Library->Microsoft->Windows->Windows Defender”.

Lastly, “MpCmdRun.exe” has multiple command line arguments supported in different categories such as scanning and tracing. We can get information about all the available options using the “-h” switch or the “?”.



[287] https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/microsoft-defender-antivirus-windows?view=o365-worldwide

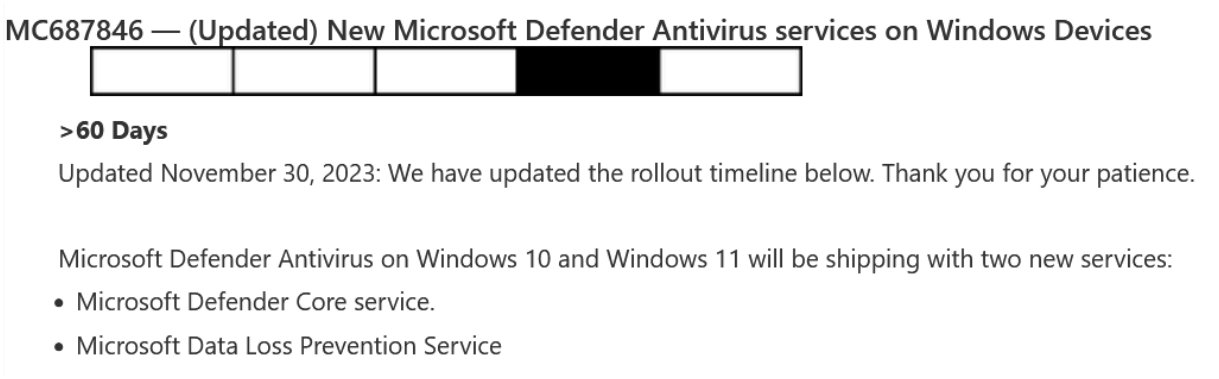
[288] https://medium.com/@boutnaru/windows-scheduler-tasks-84d14fe733c0

MpDefenderCoreService.exe (Antimalware Core Service)

“MpDefenderCoreService.exe” is a PE binary located at “C:\ProgramData\Microsoft\Windows Defender\Platform\[VERSION]\MpDefenderCoreService.exe”. By the way, [VERSION] matches the file version stored in the PE. Its description states it is the “Antimalware Core Service”. Also, the binary is also digitally signed by Microsoft.

Moreover, “MpDefenderCoreService.exe” can be used as the start image of “Microsoft Defender Antivirus Core service” (MdCoreSvc). It goal is to improve the stability and performance of “Windows Defender Antivirus”[289]. The separation to different services is was not since the creation of “Microsoft Defender Antivirus” - as shown in the screenshot below[290].

Lastly, we can think about it as part of the processes of “Microsoft Defender Antivirus”[291] together with processes like: “NisSrv.exe”[292] and “MsMpEng.exe”.



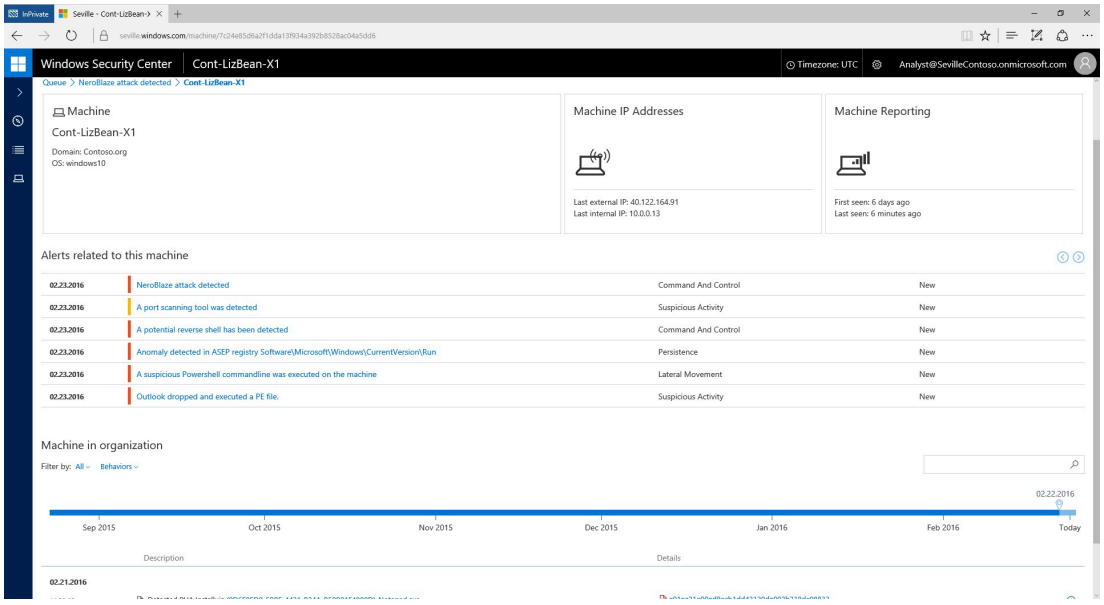
[289] https://github.com/MicrosoftDocs/microsoft-365-docs/blob/public/microsoft-365/security/defender-endpoint/microsoft-defender-antivirus-windows.md
[290] https://techcommunity.microsoft.com/t5/public-sector-blog/december-2023-microsoft-365-us-public-sector-roadmap-newsletter/ba-p/4010161
[291] https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/microsoft-defender-antivirus-windows?view=o365-worldwide
[292] https://medium.com/@boutnaru/the-windows-process-journey-nissrv-exe-microsoft-network-realtime-inspection-service-48b1245f434c

MsSense.exe (Windows Defender Advanced Threat Protection Service Executable)

“MsSense.exe” (Windows Defender Advanced Threat Protection Service Executable) is a PE binary located at “%ProgramFiles%\Windows Defender Advanced Threat Protection\MsSense.exe”. It is used as the main binary of the “Windows Defender Advanced Threat Protection Service” (Sense). The description of the services states “Windows Defender Advanced Threat Protection service helps protect against advanced threats by monitoring and reporting security events that happen on the computer”.

Moreover, the service is executed using the permissions/privileges of the “Local System” user[293]. By the way, “MsSense.exe” is digitally signed by Microsoft. It is dependent on “MsSense.dll” (Windows Defender Advanced Threat ProtectionSense Library), which by default is located in the same directory as “MsSense.exe”.

Lastly, the goal of “Windows Defender Advanced Threat Protection” is to help detect, investigate and respond to advanced attacks (focused on enterprises). This is done by providing key information about who/what/why the attack happened - as shown in the screenshot below. Also, it provides response recommendations and time-travel like capabilities (6-months historical data on state of the machine) - as shown in the screenshot below[294].



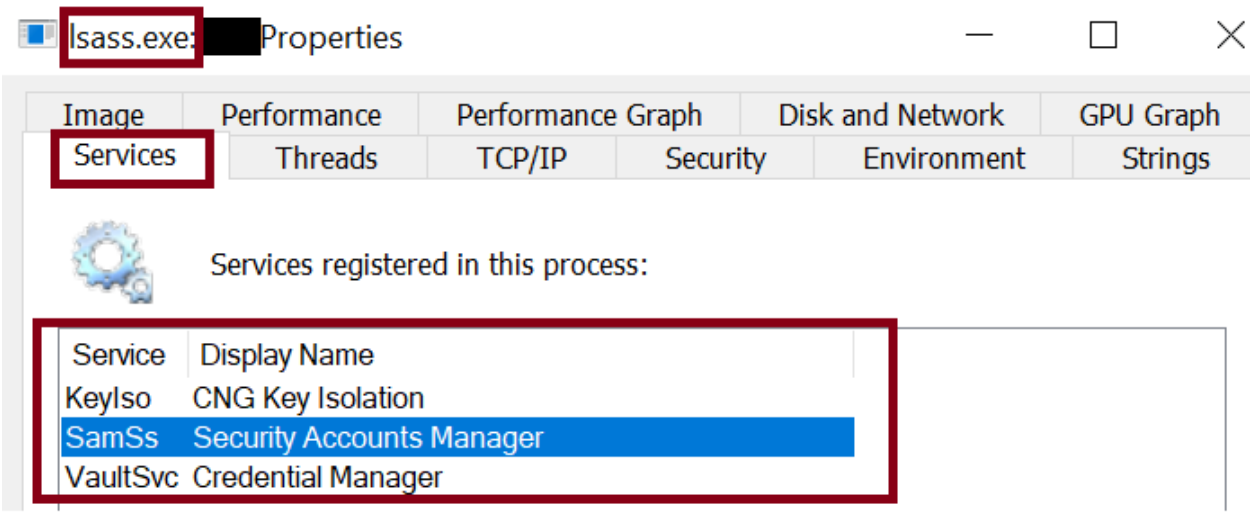
[293] https://medium.com/@boutnaru/the-windows-security-journey-local-system-nt-authority-system-f087dc530588
[294] https://blogs.windows.com/windowsexperience/2016/03/01/announcing-windows-defender-advanced-threat-protection/

lsass.exe (Local Security Authority Process)

"lsass.exe" (Local Security Authority Subsystem Service) is a PE binary located in "%windir%\System32\lsass.exe". It is used for enforcing security policy, creating access tokens for logging on users, writing the security event log and more[295].

Moreover, "lsass.exe" can hold valuable authentication data like: kerberos tickets (TGT/TGS), LM/NT hashes, encrypted password and more[296]. Thus,. Because "lsass.exe" stores the current user OS credentials (and can even store domain admin credentials in some cases). Due to that, it is an appealing target for attacks which can allow them to perform lateral movement. For hardening "lsass.exe" administrators can: enable it as PPL, enable credential guard, enable restricted admin mode for RDP and disable WDigest logon[297].

Lastly, the "lsass.exe" process is hosting different services inside its own process memory address space. We have "KeyIso" (CNG Key Isolation) which provides key process isolation to private keys and associated cryptographic operations as required by Common Criteria. "SamSs" (Security Account Manager), the startup of this service signals other services that the SAM is ready to accept requests. "VaultSvc" (Credential Manager), which is used to provide secure storage and retrieval of credentials to users/applications/security service packages - as shown in the screenshot below (taken from Process Explorer). By the way, if the computer is joined into a domain there will also be a service for network logon.



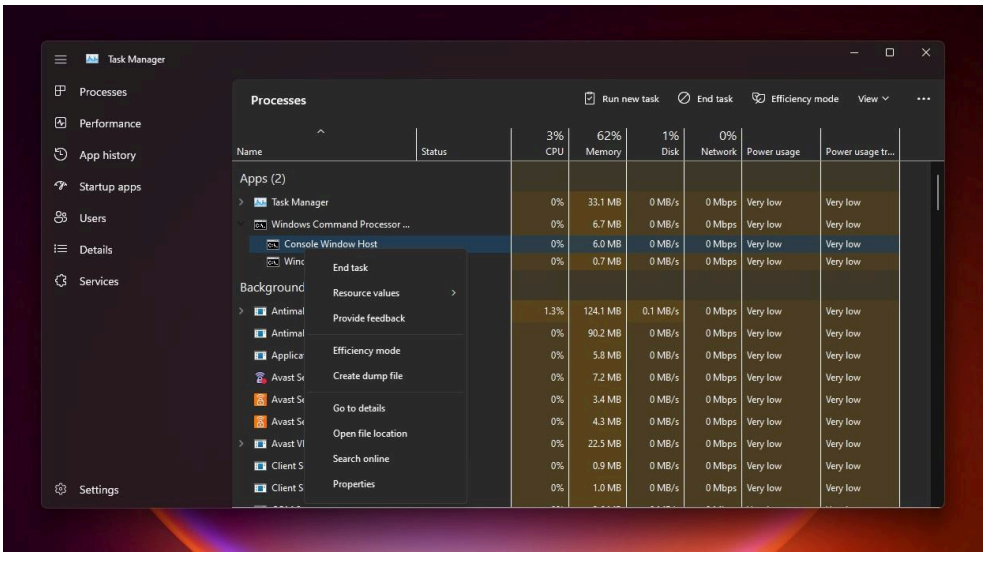
[295] https://en.wikipedia.org/wiki/Local_Security_Authority_Subsystem_Service
[296] https://redcanary.com/threat-detection-report/techniques/lsass-memory/
[297] https://www.microsoft.com/en-us/security/blog/2022/10/05/detecting-and-preventing-lsass-credential-dumping-attacks/

Taskmgr.exe (Task Manager)

"Tasgmgr.exe" (Task Manager) is a PE binary located in "%windir%\system32\Taskmgr.exe". It can be used in order to view/manage current running processes, view system resources usage, analyze performance and close unresponsive applications by leveraging its user interface[298]. The binary is digitally signed by Microsoft. By the way, on 64-bit Windows systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\Taskmgr.exe".

Overall, since Windows 11 22H2 "Task Manager" has a new design based on Fluent UI and WinUI. Thus, the classic interface was changed to a hamburger menu layout - as shown in the screenshot below. We can find the different viewing options: "Processes" (limited information about each running process) , "Performance" (CPU/memory/IO/networking usage and performance), "App History" (usage history for UWP applications), "Startup Apps", "Users", "Details" and "Services" on the hamburger menu in the left side of the UI. This has been done to improve the accessibility in case of touchscreen based devices[299].

Lastly, we can go over a reference implementation of "takmgr.exe" as part of ReactOS[300]. Also, there are different ways to open "Task Manager" such as (but not limited to): "CTRL+Shift+ESC", "CTRL+ALT+DELETE"-> "Task Manager" and "WinKey+X"->"Task Manager"[301]. By the way, based on the command line arguments passed to "taskmgr.exe" we can identify the way in which it was launched[302].



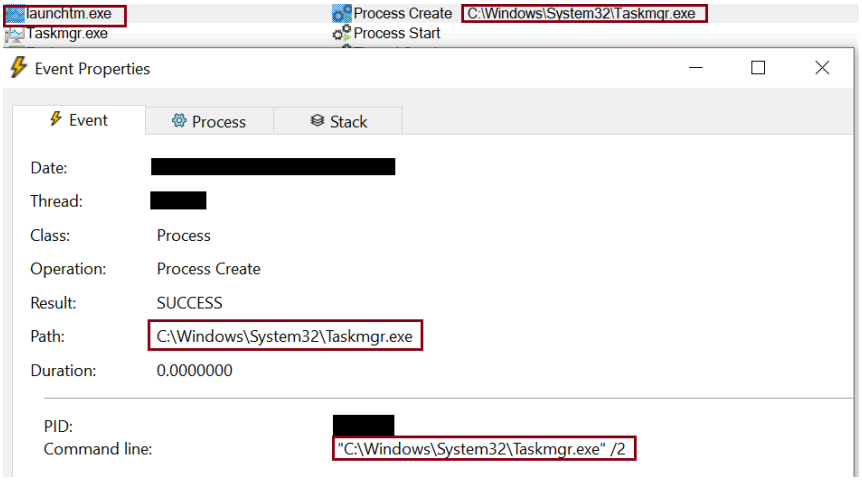
[298] https://www.spyshelter.com/exe/microsoft-windows-taskmgr-exe/
[299] https://www.bleepingcomputer.com/news/microsoft/hands-on-with-windows-11s-new-task-manager/
[300] https://github.com/reactos/reactos/tree/master/base/applications/taskmgr
[301] https://www.howtogeek.com/66622/stupid-geek-tricks-6-ways-to-open-windows-task-manager/
[302] https://www.hexacorn.com/blog/2018/07/22/taskmgr-exe-slashing-numbers/

LaunchTM.exe (Task Manager Launcher)

"LaunchTM.exe" (Task Manager Launcher) is a PE binary located in "%windir%\System32\LaunchTM.exe". It is used for launching "taskmgr.exe"[303]. The "LaunchTM.exe" binary is digitally signed by Microsoft.

Overall, when pressing "CTRL+SHIFT+ESC" the executable "takmgr.exe" is not launched directly. However, the "LaunchTM.exe" is the binary that is executed and it launches "taskmgr.exe" - as shown in the screenshot below (taken using ProcMon from the Sysinternals Suite). The "LaunchTM.exe" binary uses the following variable based path "%WINDIR%\System32\Taskmgr.exe". Thus, by overriding the WINDIR variable we can cause a different binary to execute[304].

Lastly, when pressing "CTRL+SHIFT+ESC" "LaunchTM.exe" is started by the "winlogon.exe"[305] process which is in the session as the users pressing the key combination[306]. On 64-bit Windows systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\LaunchTM.exe".



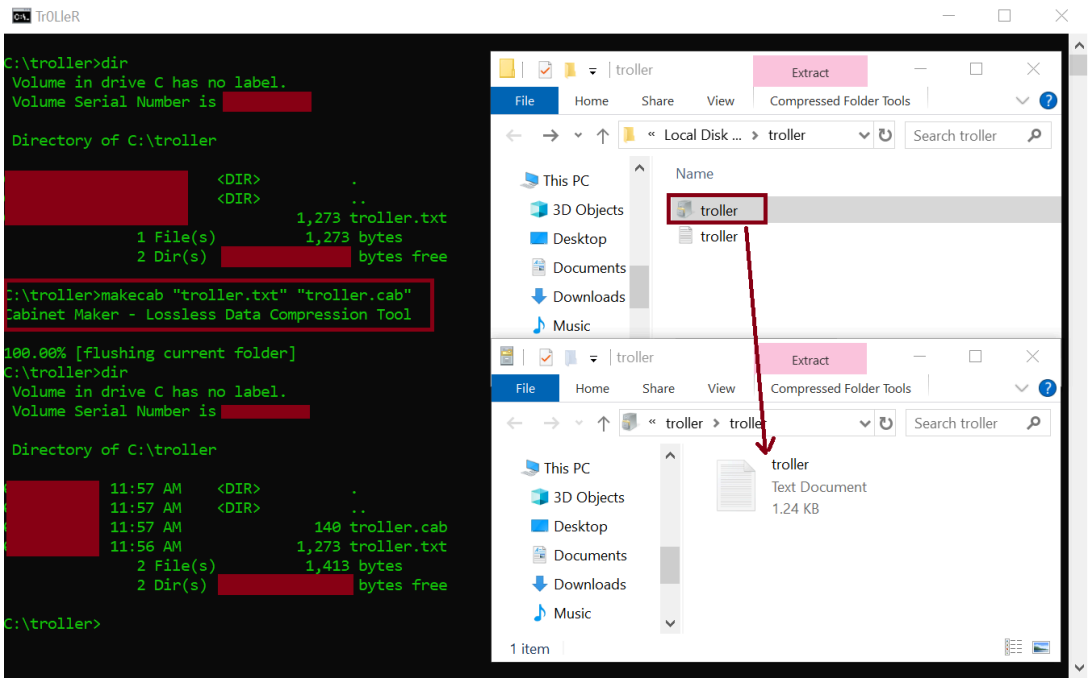
[303] https://medium.com/@boutnaru/the-windows-process-journey-taskmgr-exe-task-manager-005753dbcf3a
[304] https://www.hexacorn.com/blog/2020/05/23/lolbin-ltd/
[305] https://medium.com/@boutnaru/the-windows-process-journey-winlogon-exe-windows-logon-application-88a1d4d3e13c
[306] https://twitter.com/uberAgentApp/status/1007766836677668864

makecab.exe (Cabinet Maker)

“makecab.exe” (Cabinet Maker) is a PE binary located in “%windir%\System32\makecab.exe”. The binary is a lossless data compression tool built-in as part of the Windows operating system. This is done by packaging files into a cabinet (*.cab) file[307]. The “makecab.exe” binary is signed by Microsoft.

Moreover, a single “*.cab” file can contain up to 65,536 files with a limit of 1.99 GiB in size. “makecab.exe” is a replacement for an old utility called “cabarc.exe”. Also, the “makecab.exe” defaults are configured for optimizing to a floppy disk layout[308]. By the way, in 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\makecab.exe”.

Lastly, we can create a cabinet file by specifying the file we want to compress and the name of the destination file - as shown in the screenshot below. Also, we can use a directives file (using the “/F” switch). Directives begin with a period (“.”), followed by a command name, and possibly by (blank delimited) arguments[309]. By the way, “diantz.exe” is the same as the “makecab.exe” command are can be found in Windows servers[310].



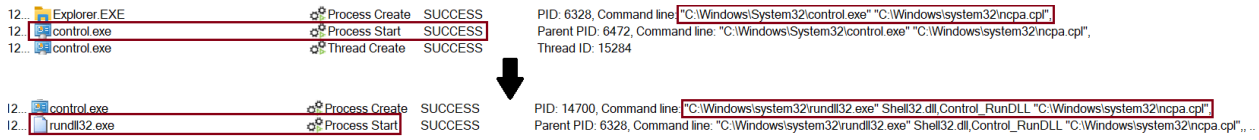
[307] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/makecab
[308] https://ss64.com/nt/makecab.html
[309] https://ss64.com/nt/makecab-directives.html
[310] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/diantz

control.exe (Windows Control Panel)

“control.exe” is a PE binary located in “%windir%\System32\control.exe”. It is the “Windows Control Panel”[311], which is used for configuring system-level features of the operating system. The “control.exe” binary file is signed by Microsoft. Also, on 64-bit Windows systems, there is also a 32-bit version of the binary located at “%windir%\SysWOW64\control.exe”.

Overall, when we execute a “*.CPL” file “control.exe” is launched with the path to the file given as an argument - as shown in the screenshot below (taken using Sysinternals’ ProcMon). After it is started “control.exe” launches “rundll32.exe”[312] and calls the “Control_RunDLL” function form “shell32.dll” with the path to “*.CPL” (which is basically a DLL file) file as an argument - as shown also in the screenshot below. We can go over a reference implementation of “control.exe” from ReactOS[313].

Lastly, since “Windows Vista” some options that were accessed using a “*.CPL” files are implemented as a separate “.exe” file. Also, using “control.exe” we can open specific “Control Panel” windows or even pages such as: “control.exe /name Microsoft.ProgramsAndFeatures” or “control.exe /name Microsoft.RegionalAndLanguageOptions /page /p:"administrative"”[314]. By the way, such commands can also cause “SystemSettings.exe” to be launched[315] .



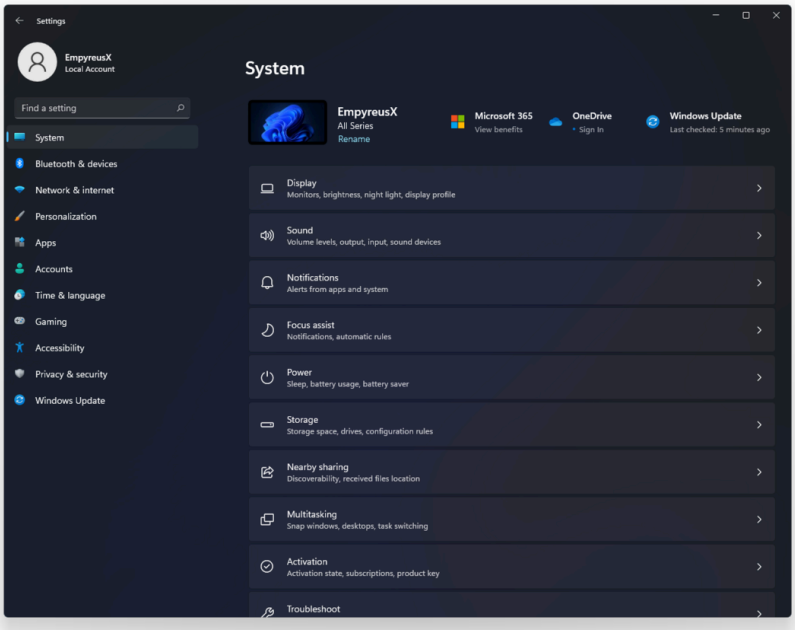
[311] https://medium.com/@boutnaru/the-windows-concept-journey-control-panel-34bf84ca7ff0
[312] https://medium.com/@boutnaru/the-windows-process-journey-rundll32-exe-windows-host-process-415132f1363
[313] https://github.com/reactos/reactos/tree/master/base/applications/control
[314] https://learn.microsoft.com/en-us/windows/win32/shell/executing-control-panel-items
[315] https://medium.com/@boutnaru/the-windows-process-journey-systemsettings-exe-immersive-control-panel-system-settings-app-930969b84b40

SystemSettings.exe (Immersive Control Panel System Settings App)

“SystemSettings.exe” (Immersive Control Panel System Settings App) is a PE binary located in “%windir%\ImmersiveControlPanel\SystemSettings.exe”. It is used for viewing\making system configuration changes in Windows[316]. Also, the “SystemSettings.exe” binary is signed by Microsoft. The goal of the “System Settings” was to replace “Control Panel” which for more than a decade has not happened yet[317].

Overall, “SystemSettings.exe” is a UWP (Universal Windows Platform) application. “SystemSettings.exe” provides a modern settings panel user interface - as shown in the screenshot below. It has been included as part of Windows since “Windows 8”/”Windows Server 2012” while still having the old "Control Panel". We are still in a transformation period in which Microsoft is migrating functions from “Control Panel” to the “System Settings”. However, some functions are in both and some open even the old “Control Panel” dialog boxes[318].

Lastly, the settings are clustered into different categories: “Home”, “System”, “Devices”, “Phone”, “Network & Internet”, “Personalization”, “Apps”, “Accounts”, “Time & Language”, “Gaming”, “Accessibility”, “Privacy & Security” and “Windows Update &” - as shown in the screenshot below taken from Windows 11[319].



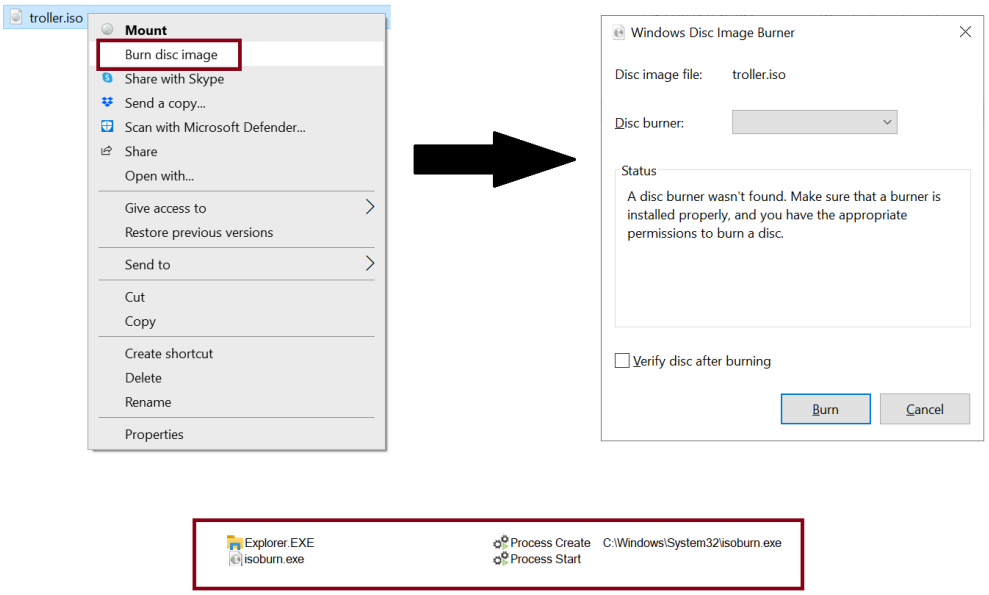
[316] https://support.lenovo.com/us/en/solutions/ht515504-overview-of-system-settings-in-windows-11
[317] https://en.wikipedia.org/wiki/Settings_(Windows)
[318] https://www.file.net/process/systemsettings.exe.html
[319] https://betawiki.net/wiki/File:Windows11-22000.51-SettingsDark.png

isoburn.exe (Windows Disc Image Burning Tool)

“isoburn.exe” (Windows Disc Image Burning Tool) is a PE binary located in “%windir%\System32\isoburn.exe”. It is used for burning ISO files without the need for third party software. It was added as part of Windows 7[320]. Also, the binary is digitally signed by Microsoft.

Moreover, on 64-bit versions of Windows there is also a 32-bit version of “isoburn.exe” located at “%windir%\SysWOW64\isoburn.exe”. By the way, both versions are digitally signed by Microsoft. Also, we can pass command line arguments to “isoburn.exe”[321].

Lastly, we can you right click on an “*.iso” file and select “Burn disc image” that we open the GUI of “isoburn.exe” - as shown in the screenshot below. It is important to know that “isoburn.exe” also supports burning an “*.iso” files to USB devices[322].



[320] https://winaero.com/how-to-burn-an-iso-file-from-the-command-prompt-in-windows-10/
[321] https://www.windows-faq.de/2018/01/27/isoburn-windows-iso-brennprogramm-als-kommandozeilen-befehl/
[322] https://www.passcue.com/burn-iso-image-to-usb-on-windows.html

MoUsoCoreWorker.exe (MoUSO Core Worker Process)

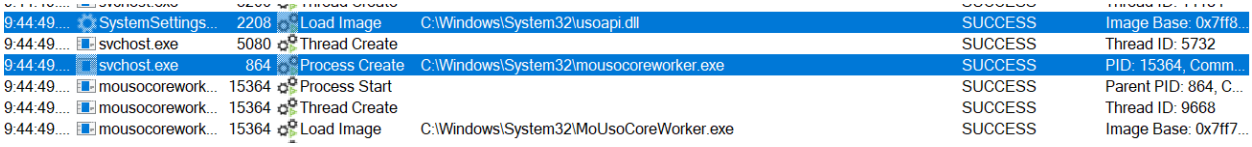
“MoUsoCoreWorker.exe” is an executable which is responsible for performing Windows updates. It is a replacement for some of the operations performed by “wuauclt.exe”, which does support updating Windows 10/11 environments as part of the move to “Unified Update Platform” aka UUP[323].

Thus, with the release of Windows 10 Microsoft has moved to UUP which allows a single publishing, hosting, scanning and downloading for all types of OS updates (monthly and new features updates) targeting any client devices which are running a Windows based OS[324].

Moreover, The executable is located at “%windir%\System32\MoUsoCoreWorker.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). “MoUsoCoreWorker.exe" is started by “svchost.exe” and executed with the permissions of “Local System”[325] which has an SID[326] of “S-1-5-18”.

By the way, USO stands for “Update Session Orchestrator”. “MoUsoCoreWorker.exe” it is a crucial component which helps in controlling the order in which updates are download and installed in Windows[327].

Lastly, each time Windows is looking for updates “MoUsoCoreWorker.exe” is started[328]. A demonstration for that is shown in the screenshot below (taken using Sysinternals’ ProcMon). The screenshot was taken after pressing on “Check for Updates” (from the “System Settings”). We can see that “usoapi.dll” (Update Session Orchestrator API) is loaded by “SystemSettings.exe”[329] and then “MoUsoCoreWorker.exe” is started by “svchost.exe”.



[323] https://helpdeskgeek.com/help-desk/what-is-mousocoreworker-exe-and-is-it-safe/
[324] https://learn.microsoft.com/en-us/windows/deployment/update/windows-update-overview
[325] https://medium.com/@boutnaru/the-windows-security-journey-local-system-nt-authority-system-f087dc530588
[326] https://medium.com/@boutnaru/windows-security-sid-security-identifier-d5a27567d4e5
[327] https://ugetfix.com/ask/how-to-fix-mouso-core-worker-process-high-cpu-and-memory-usage-in-windows/
[328] https://www.groovypost.com/explainer/what-is-mousocoreworker-exe-and-why-is-it-running/
[329] https://medium.com/@boutnaru/the-windows-process-journey-systemsettings-exe-immersive-control-panel-system-settings-app-930969b84b40

sppsvc.exe (Microsoft Software Protection Platform Service)

“sppsvc.exe” (Microsoft Software Protection Platform Service) is a PE binary located at “%windir%\System32\sppsvc.exe”. On 64-bit versions of Windows there is no 32-bit version of the executable as we have with other binaries such as “cmd.exe”[330]. Also, the “sppsvc.exe” binary is digitally signed by Microsoft.

Overall, “sppsvc.exe” is the main image of the “Software Protection” service (aka sppsvc). The description of the service states it: “Enables the download, installation and enforcement of digital licenses for Windows and Windows applications. If the service is disabled, the operating system and licensed applications may run in a notification mode. It is strongly recommended that you not disable the Software Protection service”. The service is executed with the permissions/privileges of the “Network Service”[331] user - as shown in the screenshot below.

Thus, we can say that “sppsvc.exe” performs the following functions. Ensuring the Windows operating system is genuine and properly activated. Performing periodic checks to ensure that your Windows license is still valid (and not revoked). Also, handles the activation process when you install a new copy of Windows or make significant hardware changes to your computer. It is important to know that it can also collect and send anonymous data to Microsoft about your system’s activation status[332].

Lastly, the directory ““%windir%\System32\spp\” which holds activation tokens[333]. We can backup files from that directory in order to reactivate different software offerings such as Office[334].

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\sppsvc

Name	Type	Data
(Default)	REG_SZ	(value not set)
DelayedAutoStart	REG_DWORD	0x00000001 (1)
DependOnService	REG_MULTI_SZ	RpcSs
Description	REG_SZ	@%SystemRoot%\system32\sppsvc.exe,-100
DisplayName	REG_SZ	@%SystemRoot%\system32\sppsvc.exe,-101
ErrorControl	REG_DWORD	0x00000001 (1)
FailureActions	REG_BINARY	80 51 01 00 00 00 00 00 00 00 00 00 03 00 00 00 14 00 00 00 01 00 00 00 c0 d4 01 00 01 00 0...
ImagePath	REG_EXPAND_SZ	%SystemRoot%\system32\sppsvc.exe
LaunchProtected	REG_DWORD	0x00000001 (1)
ObjectName	REG_SZ	NT AUTHORITY\NetworkService
RequiredPrivileges	REG_MULTI_SZ	SeAuditPrivilege SeChangeNotifyPrivilege SeCreateGlobalPrivilege SeImpersonatePrivilege
ServiceSidType	REG_DWORD	0x00000001 (1)
Start	REG_DWORD	0x00000002 (2)
Type	REG_DWORD	0x00000010 (16)

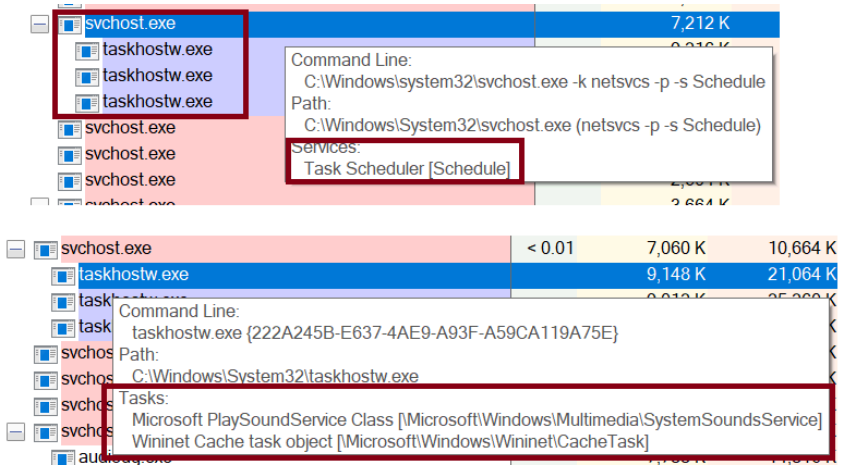
[330] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[331] https://medium.com/@boutnaru/the-windows-security-joruney-network-service-nt-authority-network-service-e8706688e383
[332] https://malwaretips.com/blogs/microsoft-software-protection-platform-service/
[333] https://community.spiceworks.com/t/windows-10-repeatedly-deactivates/681310
[334] https://community.citrix.com/forums/topic/230472-layered-image-office-2016-will-not-activate-on-first-boot/

taskhostw.exe (Host Process for Windows Tasks)

“taskhostw.exe” (Host Process for Windows Tasks) is a PE binary located at “%windir%\system32\taskhostw.exe”. It is responsible for hosting DLLs executed from tasks. It is similar to “svchost.exe”[335], which hosts DLLs implementing a specific service (and also “dllhost.exe”). The “taskhostw.exe” is digitally signed by Microsoft. By the way, on 64-bit versions of Windows there is no parallel 32-bit version of the binary as with “rundll32.exe”[336].

Overall, we can find in “%windir%\Tasks” some tasks have their action configured with a CLSID (Class ID), which is a COM object reference to a DLL[337]. “taskhostw.exe” should be a child process of the “Task Scheduler”, which is hosted by “svchost.exe” -
as shown in the screenshot below, which was taken using Sysinternals’ “Process Explorer”.

Lastly, as with “svchost.exe” in which “Process Explorer” can display which services are hosted also with “taskhostw.exe” we know that tasks are executed - as shown in the screenshot below.



[335] https://medium.com/@boutnaru/the-windows-process-journey-svchost-exe-host-process-for-windows-services-b18c65f7073f
[336] https://medium.com/@boutnaru/the-windows-process-journey-rundll32-exe-windows-host-process-415132f13634
[337] https://chentiangemalc.wordpress.com/2011/05/08/windows-7-default-scheduled-taskscomplete-overview/

wuauclt.exe (Windows Update Auto Update Client)

"wuauclt.exe" (Windows Update Auto Update Client) is a PE binary located at "%windir%\system32\wuauclt.exe". It is used as the "Windows Update Agent", which basically downloads new "Windows Update" files[338]. Also, "wuauclt.exe" is digitally signed by Microsoft.

Overall, "wuauclt.exe" is started by "MoUsoCoreWorker.exe"[339] - as shown in the screenshot below, which was taken using Sysinternals' "Process Explorer" while performing a "Windows Update". The arguments which are passed to the executable are: "/UpdateDeploymentProvider UpdateDeploymentProvider.dll /ClassId [CLSID] /RunHandlerComServer". Using a similar command "wuauclt.exe /UpdateDeploymentProvider <Full_Path_To_DLL> /RunHandlerComServer" we can load an arbitrary DLL to the memory address space of a newly created "wuauclt.exe"[340].

Lastly, we can say that the "wuauclt.exe" command line utility allows us some control over the functioning of the Windows Update Agent. Also, it is updated as part of "Windows Update"[341].

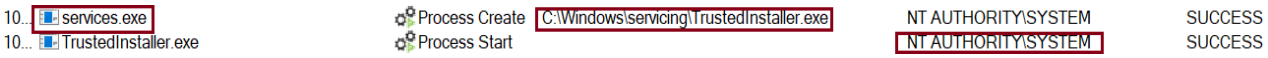
[338] https://ss64.com/nt/wuauclt.html
[339] https://medium.com/@boutnaru/the-windows-process-journey-mousocoreworker-exe-mouso-core-worker-process-c39934971fbc
[340] https://dtm.uk/wuauclt/
[341] https://learn.microsoft.com/pt-br/security-updates/windowsupdateservices/18139070?ref=dtm.uk

TrustedInstaller.exe (Windows Modules Installer)

“TrustedInstaller.exe” (Local Security Authority Subsystem Service) is a PE binary located in “%windir%\servicing\TrustedInstaller.exe”. It is the main image of the “TrustedInstaller” service which is responsible for enabling installation/modification/removal of Windows updates and optional components[342].

Moreover, by default the “TrustedInstaller” service is set at “Manual”[343] and is executed under the “Local System” account[344] - as shown in the screenshot below. By the way, the description of the service states it: “Enables installation, modification, and removal of Windows updates and optional components. If this service is disabled, install or uninstall of Windows updates might fail for this computer”.

Lastly, when performing an update the “TrustedInstaller.exe” binary is executed by “services.exe” - as shown in the screenshot below (taken using Sysinternals' ProcMon). By the way, “TrustedInstaller.exe” is documented as standing for “Windows Trusted OS Component Installer”. Also, it causes “TiWorker.exe” to be executed[345].



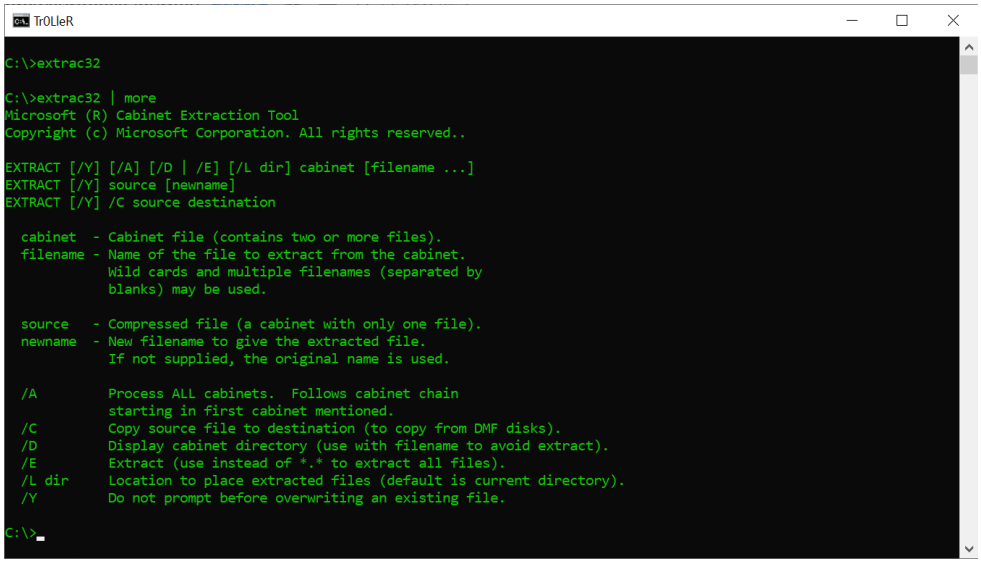
[342] https://www.minitool.com/news/trustedinstaller-exe.html
[343] https://learn.microsoft.com/en-us/answers/questions/597773/trustedinstaller-file-location
[344] https://medium.com/@boutnaru/the-windows-security-journey-local-system-nt-authority-system-f087dc530588
[345] https://www.file.net/process/trustedinstaller.exe.html

extrac32.exe (CAB File Extract Utility)

“extrac32.exe” (Microsoft® CAB File Extract Utility) is a PE binary located in “%windir%\System32\extrac32.exe”. The binary is used to extract files from a cabinet or source. By default this utility does not display any output to the console. We can redirect the help output using the more command[346] - as shown in the screenshot below.

Moreover, the “extrac32.exe” binary is digitally signed by Microsoft. On 64-bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\extrac32.exe”. We can use it to extract one or more compressed “*.CAB” (cabinet) files and even extract specific file\s from a cabinet file[347].

Lastly, we can use “extrac32.exe” to read a file (download) for a UNC path, write (upload) to a UNC path. copy a file and extract data to an ADS[348]. Also, while extracting specific file/s from a “*.CAB” file/s we can specify a pattern like “*.*” for all files or a list of multiple files (separated by blanks).



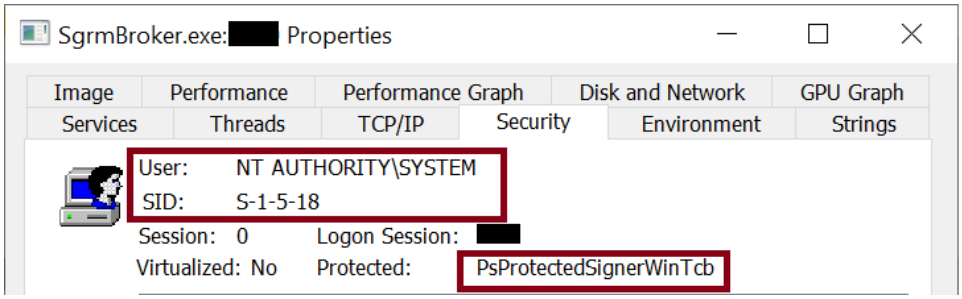
[346] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/extract
[347] https://ss64.com/nt/extract.html
[348] https://lolbas-project.github.io/lolbas/Binaries/Extrac32/

SgrmBroker.exe (System Guard Runtime Monitor Broker Service)

"SgrmBroker.exe" is the main image of the "System Guard Runtime Monitor Broker" service. The description of the service states it "Monitors and attests to the integrity of the Windows platform". The service is responsible for monitoring/proving the integrity of the operating system[349].

Overall, SGRM (System Guard Runtime Monitor) is used for remote attestation for verifying the integrity of the operating system[350]. The "SgrmBroker.exe" process is executed using context of the "Local System" user[351]. It is also configured as a protected process, specifically "PsProtectedSignerWinTcb" - as shown in the screenshot below.

Lastly, "SgrmBroker.exe" is only one component of SGRM in conjunction with SgrmAgent.sys (the agent driver that exposes functionality used by "SgrmBroker.exe"), "SgrmEnclave.dll" (the enclave controller shim, contains the Lua runtime) and "SgrmLpac.exe" (A local RPC service, which exposes a method to send an HTTP POST request to a specified endpoint). By the way, there is also "SgrmEnclave_secure.dll" that is loaded/run on VTL-1 mode[352].



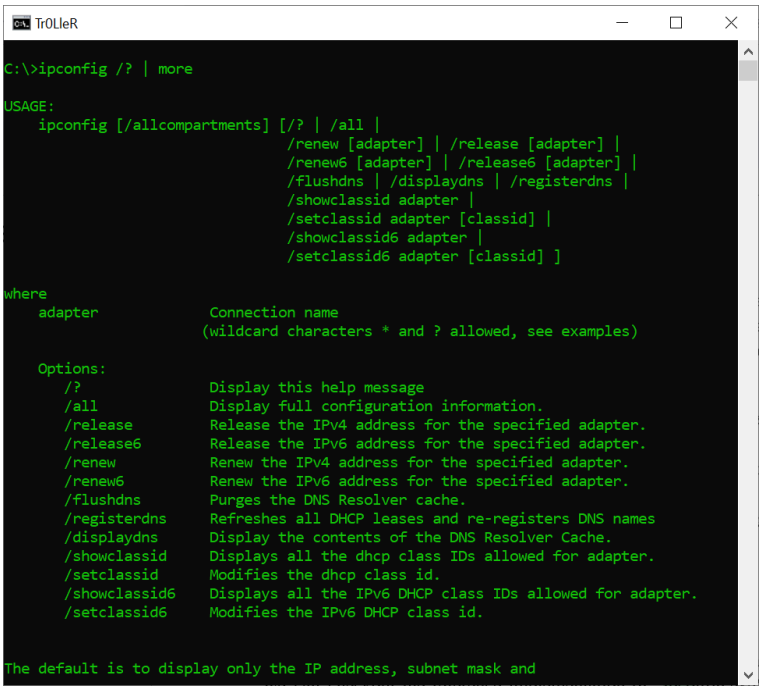
[349] https://www.minitool.com/news/system-guard-runtime-monitor.html
[350] https://medium.com/@boutnaru/the-windows-security-journey-sgrm-system-guard-runtime-monitor-04b0971f2492
[351] https://medium.com/@boutnaru/the-windows-security-journey-local-system-nt-authority-system-f087dc530588
[352] https://blog.syscall.party/2022/08/02/inside-windows-defender-system-guard-runtime-monitor.html

ipconfig.exe (IP Configuration Utility)

“ipconfig.exe” (IP Configuration Utility) is a binary PE file located at “%windir%\System32\ipconfig.exe”, on 64-bit systems there is also a 32-bit version located at “%windir%\SysWOW64\ipconfig.exe”. This is a CLI application which is digitally signed by Microsoft.

Overall, “ipconfig.exe” allows performing different tasks such as: retrieving information (using “ipconfig.exe /all” which include data like IP address, NIC physical address, DNS server DHCP info and more), releasing DHCP configuration (“ipconfig.exe /renew”) and reinitiating a DHCP request (“ipconfig.exe /renew”). We can also display the content of the the DNS resolver cache using “ipconfig.exe /displaydns” and purge it using “ipconfig.exe /flushdns”[353] - details about each argument of the utility is shown in the screenshot below.

Lastly, we can think about “ipconfig.exe” as an equivalent to “ifconfig”[354] or “ip”[355] in the sense of managing network interfaces. On macOS we also have “ipconfig”[356], used for viewing/controlling IP configuration state. Also, we can checkout the reference implementation of “ipconfig.exe” as part of ReactOS for more information[357].



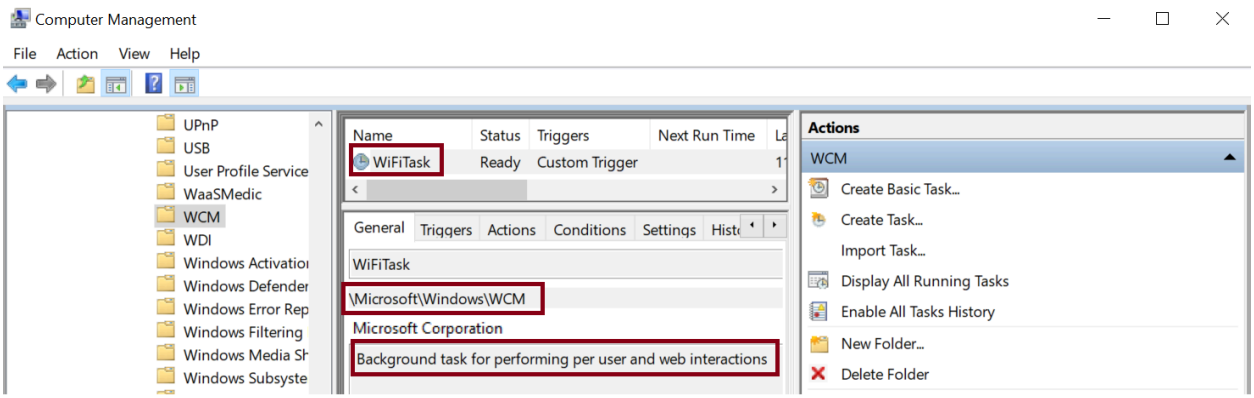
[353] https://www.ninjaone.com/blog/ipconfig-commands/
[354] https://man7.org/linux/man-pages/man8/ifconfig.8.html
[355] https://linux.die.net/man/8/ip
[356] https://ss64.com/mac/ipconfig.html
[357] https://github.com/reactos/reactos/tree/master/base/applications/network/ipconfig

wifitask.exe (Wireless Background Task)

“wifitask.exe” (Wireless Background Task) is a PE binary located in “%windir%\System32\wifitask.exe”. On 64-bit versions of Windows there is no parallel 32-bit version of the binary like we have with other executables like “cmd.exe”[358]. Also, the binary is digitally signed by Microsoft.

Overall, “wifitask” is configured as a scheduled task with the name of “WiFiTask”[359]. It is used as a background task for performing per user and web interactions - as shown in the screenshot below. It is part of the “Windows Connection Manager” (WCM), its service description states it can perform automatic connect/disconnect decisions based on the network connectivity options currently available to the device[360].

Lastly, “wifitask.exe” is part of the “Wifi Network Manager” (as part of WCM). “wifitask.exe” is dependent on “%windir%\System32\wlanapi.dll” which is the Windows WLAN AutoConfig Client Side API DLL. It is used for controlling\managing wireless connections on a Windows device. “wifitask.exe” helps in scanning for available wireless networks and connecting to a chosen network[361].



[358] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[359] https://medium.com/@boutnaru/windows-scheduler-tasks-84d14fe733c0
[360] https://learn.microsoft.com/en-us/windows/win32/wcm/windows-connection-manager-portal
[361] https://www.spyshelter.com/exe/microsoft-windows-wifitask-exe/

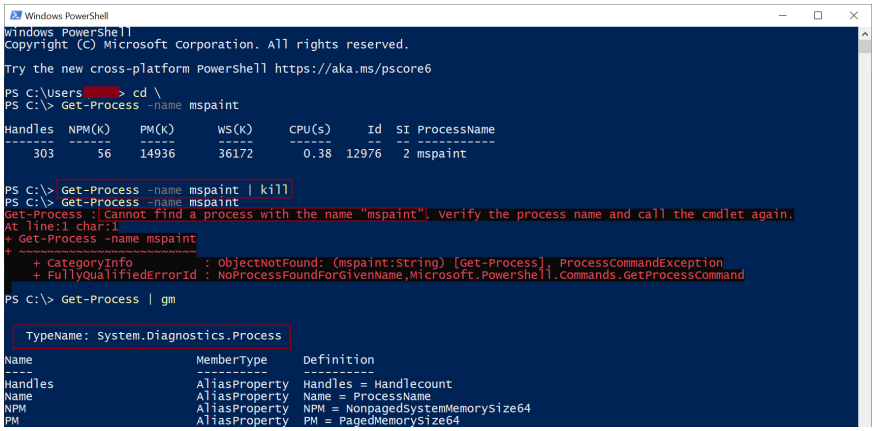
powershell.exe (Windows PowerShell)

“powershell.exe” (Windows PowerShell) is a PE binary located in “%windir%\system32\WindowsPowerShell\v1.0\powershell.exe”. On 64-bit versions we also have a 32-bit version of the binary located at “%windir%\SysWOW64\WindowsPowerShell\v1.0\powershell.exe”. By the way, the binary is digitally signed by Microsoft. It is important to know that we can check out the PowerShell source code in its Github repository[362].

Overall, “powershell.exe” is a cross-platform task automation solution which includes: a scripting language, a configuration management framework and a command-line shell. It is important to know that PowerShell can run on Windows, Linux, and macOS[363]. For more information about announcements, features regarding “PowerShell” we can check out Microsoft’s on demand video content[364].

Moreover, when developing code we can use the “PowerShell Module Browser” from Microsoft in order to search for modules and cmdlets[365]. A cmdlet is a lightweight command that is used in the PowerShell environment[366]. There is also the “PowerShell Gallery” which is a central repository of PowerShell modules/scripts/DCS resources[367].

Lastly, we can think about “powershell.exe” as a more mature replacement for “cmd.exe”[368]. This is due to the fact we can do anything supported in “cmd.exe” with “powershell.exe” and much more than that. One of the biggest benefits of PowerShell is the fact cmdlets can return as a return value an object and not just a string - as shown in the screenshot below (we call the kill method of the return object).



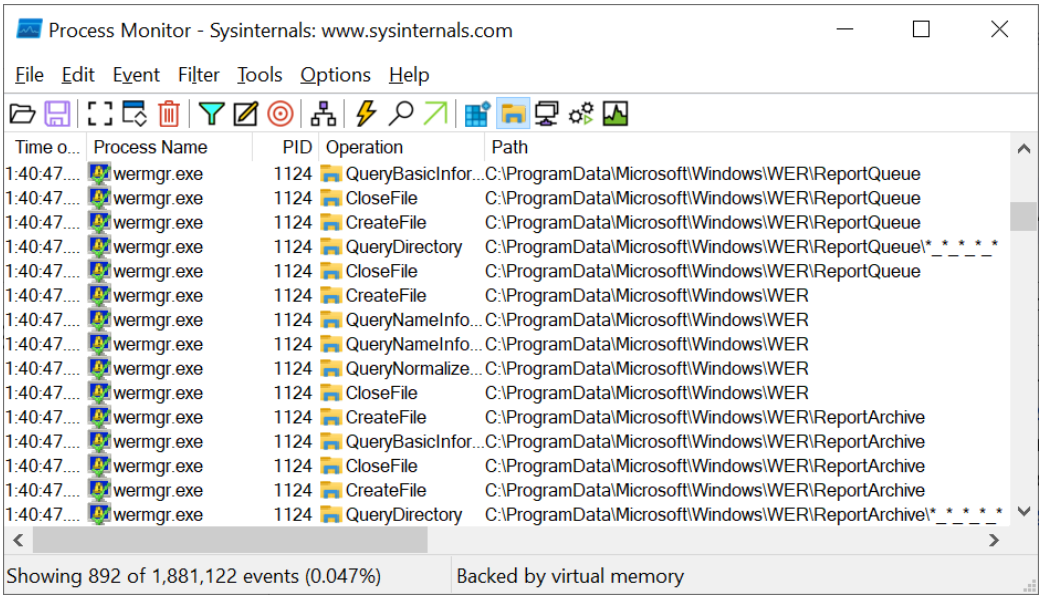
[362] https://github.com/PowerShell/PowerShell
[363] https://learn.microsoft.com/en-us/powershell/scripting/overview
[364] https://learn.microsoft.com/en-us/shows/browse?terms=powershell
[365] https://learn.microsoft.com/en-us/powershell/module/
[366] https://learn.microsoft.com/en-us/powershell/scripting/developer/cmdlet/cmdlet-overview
[367] https://www.powershellgallery.com/
[368] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b

wermgr.exe (Windows Problem Reporting)

“wermgr.exe” is a PE binary located at “%windir%\system32\wermgr.exe”. On 64-bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\wermgr.exe”. This binary is one of the components of the “Windows Error Reporting” feature[369] of the operating system which interacts with the “Windows Error Reporting Service” (WerSvc). “wermgr.exe” is used to read/parse/copy/move/delete report files files[370].

Overall, when “wermgr.exe” is executed with the “-upload” argument the function “wermgr!DoCoreUpload” is called. This function lists all the subdirectories under the ReportQueue directory (“C:\ProgramData\Microsoft\Windows\WER\ReportQueue”) - as shown in the printscreen below. Its goal is to read the error reports and submit them to Microsoft[371].

Lastly, “wermgr.exe” is depended on “%windir%\system32\wer.dll” (Windows Error Reporting DLL) and in the case of the 32-bit version it is dependent on “%windir%\SysWOW64\wer.dll”. Also, the binary is digitally signed by Microsoft. When it is executed “wermgr.exe” can also access other subdirectories of “C:\ProgramData\Microsoft\Windows\WER” like “ReportArchive” and “Temp”.



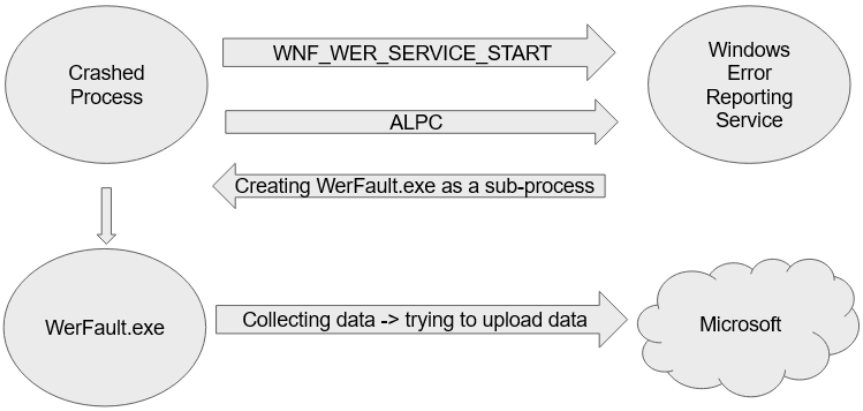
[369] https://medium.com/@boutnaru/the-windows-concept-journey-wer-windows-error-reporting-812316b8eb0a
[370] https://unit42.paloaltonetworks.com/tale-of-a-windows-error-reporting-zero-day-cve-2019-0863/
[371] https://unit42.paloaltonetworks.com/tale-of-a-windows-error-reporting-zero-day-cve-2019-0863/

WerFault.exe (Windows Problem Reporting)

"WerFault.exe" (Windows Problem Reporting) is a PE binary located at "%windir%\system32\WerFault.exe". On 64-bit systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\WerFault.exe". This binary is one of the components of the "Windows Error Reporting" feature[372] of the operating system which interacts with the "Windows Error Reporting Service" (WerSvc). By the way, the binary is digitally signed by Microsoft.

Moreover, "WerFault.exe" is created when a process crashes[373]. The goal of the binary is to collect data, exception info and even memory dumps. Later "WerFault.exe" is used for uploading the data to Microsoft's cloud. In case there is no Internet connection "WerFault.exe" saves the reports locally which can be later uploaded by "wermgr.exe"[374].

Thus, in case of an unhandled exception a signal (WNF_WER_SERVICE_START) is sent to ensure the "Windows Error Reporting Service" (WerSvc) is started. Afterwards the crashed process talks with "WerSvc" using ALPC which leads to the creation of "WerFault.exe" as a sub-process of the crashed process (with the same level of user-permissions and not with the token of local system as "WerSvc") - as shown in the diagram below. The reports created by "WerFault.exe" are saved at "C:\ProgramData\Microsoft\Windows\WER\ReportQueue\" and moved to "C:\ProgramData\Microsoft\Windows\WER\ReportArchive" in case the report was not uploaded due to network problems/issues[375].



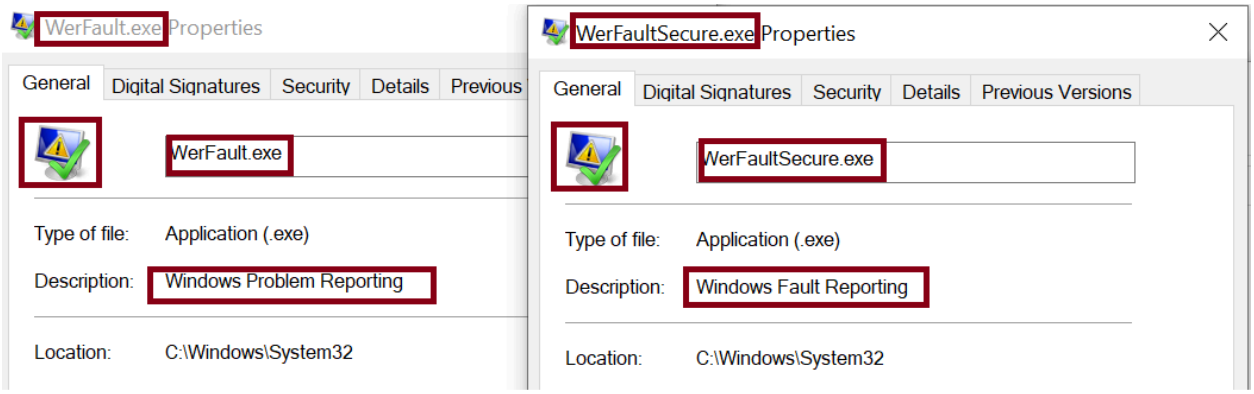
[372] https://medium.com/@boutnaru/the-windows-concept-journey-wer-windows-error-reporting-812316b8eb0a
[373] https://www.microsoft.com/en-us/security/blog/2021/03/02/hafnium-targeting-exchange-servers/
[374] https://medium.com/@boutnaru/the-windows-process-journey-wermgr-exe-windows-problem-reporting-a9055d0a6b96
[375] https://msrndcdn360.blob.core.windows.net/bluehat/bluehatil/2022/assets/doc/Exploiting%20Errors%20in%20Windows%20Error%20Reporting.pdf

WerFaultSecure.exe (Windows Fault Reporting)

"WerFaultSecure.exe" (Windows Fault Reporting) is a PE binary located at "%windir%\system32\WerFaultSecure.exe". On 64-bit systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\WerFaultSecure.exe". This binary is one of the components of the "Windows Error Reporting" feature[376] of the operating system which interacts with the "Windows Error Reporting Service" (WerSvc).

Moreover, as with the binary "WerFault.exe"[377] also "WerFaultSecure.exe" is used for collecting data, exception info and even memory dumps. The difference is that "WerFaultSecure.exe" is used by the "Windows Error Reporting" service to create crash dumps from protected processes. Due to that it is executed at elevated PP levels. Another difference is that "WerFaultSecure.exe" encrypts the content of crash dumps before it is written to disk. The encryption is done by leveraging asymmetric encryption to allow only Microsoft to decrypt the data[378].

Lastly, although "WerFault.exe" and "WerFaultSecure.exe" are quite similar they have a different string stored in the description field as part of the PE[379] file format: "Windows Problem Reporting" and "Windows Fault Reporting" respectively, at least they have the same icon - as shown in the screenshot below. By the way, the "WerFaultSecure.exe" binary is digitally signed by Microsoft.



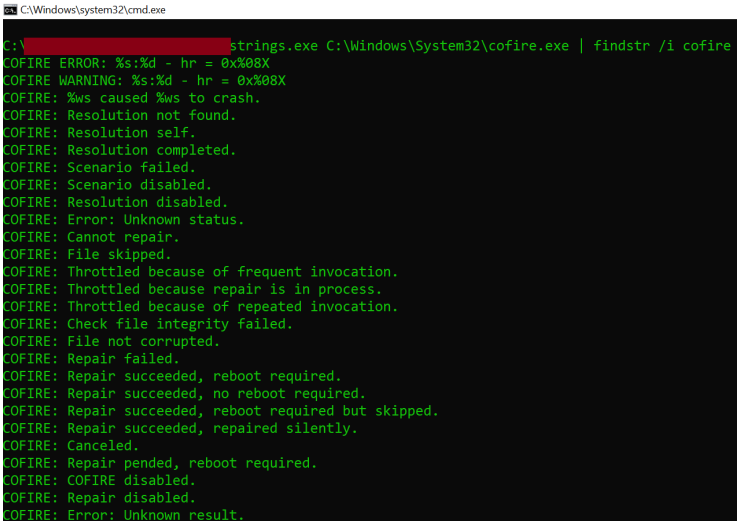
[376] https://medium.com/@boutnaru/the-windows-concept-journey-wer-windows-error-reporting-812316b8eb0a
[377] https://medium.com/@boutnaru/the-windows-process-journey-werfault-exe-windows-problem-reporting-77fe9b9fae34
[378] https://googleprojectzero.blogspot.com/2018/11/injecting-code-into-windows-protected.html
[379] https://wiki.osdev.org/PE

cofire.exe (Corrupted File Recovery Client)

"cofire.exe" (Corrupted File Recovery Client) is a PE binary located in "%windir%\System32\cofire.exe". On 64-bit versions of Windows there is no parallel 32-bit version of the binary like we have with other executables like "cmd.exe"[380]. The binary is dependent on "wdi.dll" which is part of the "Windows Diagnostic Infrastructure" (WDI). By extracting strings from the binary we can see the different messages that can be printed and get a better understanding about the capabilities of the binary like: repairing files and failing to repair a file - as shown in the screen below.

Overall, there are multiple cases in which files can be corrupted on a Windows device. Examples of such cases are: system crash, sudden power outage, update errors and hard disk problems[381]. The corrupted file client ("cofire.exe") checks the value "EnabledScenarioExecutionLevel" which is located in the following registry location "HKLM\Software\Policies\Microsoft\Windows\WDI\{8519d925-541e-4a2b-8b1e-8059d16082f2}" which is responsible for configuring corrupted file recovery behavior[382]. By the way, "corefire.exe" is digitally signed by Microsoft.

Lastly, there are three different states for recovery behavior for corrupted files: "Regular", "Silent" and "Troubleshooting Only". The first and the second perform detection, troubleshooting, and recovery. The difference between them is that "Regular" does that with minimal UI while "Silent" does it with no UI. The last one does only detection and troubleshooting without automatic recovery. The state configuration is only relevant if the "Diagnostic Policy Service" (DPS) is running[383].



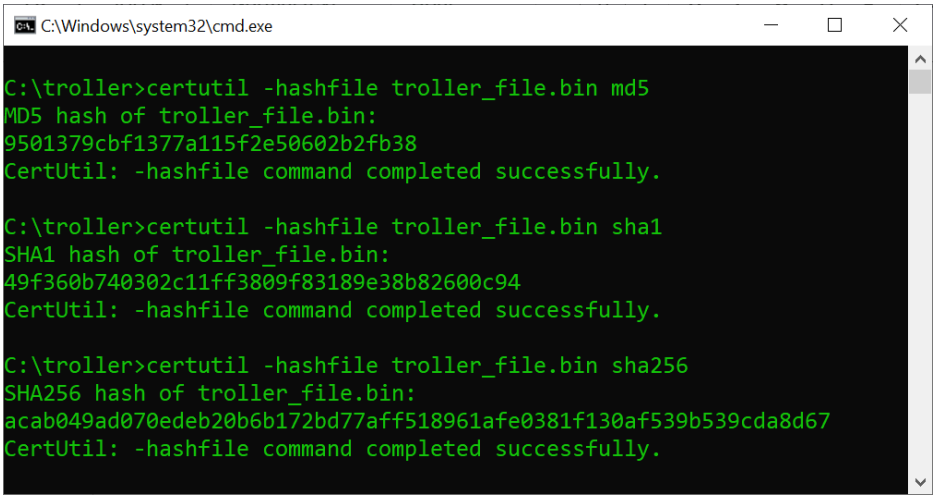
[380] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[381] https://recoverit.wondershare.com/computer-problems/restoring-corrupted-files.html
[382] https://csrc.nist.gov/CSRC/media/Projects/national-vulnerability-database/documents/CCE/cce-win2k8r2-5.20120314.xls
[383] https://admx.help/?Category=Windows_10_2016&Policy=Microsoft.Policies.FileRecovery::WdiScenarioExecutionPolicy

certutil.exe (Digital Certificate Utility)

"certutil.exe" (Digital Certificate Utility) is a binary PE file located at "%windir%\system32\certutil.exe". On 64-bit versions of Windows there is also a 32-bit version of the binary located at "%windir%\SysWOW64\certutil.exe". It is used to display certification authority (CA) configuration information, configure Certificate Services, and back up and restore CA components[384]. The binary is also digitally signed by Microsoft.

Overall, "certutil.exe" has multiple command line arguments available which can be used to: dump configuration, dump PFX structure, parse and display the contents of a file using Abstract Syntax Notation (ASN.1), decode base 64 files, submit/deny pending certificate request, attempt to connect the Active Directory "Certificate Services Request" interface, shut down the "Active Directory Certificate Services", hashing files and more[385] - as shown in the screenshot below.

Lastly, due to extensive command line switches that "certutil.exe" has it can be used as a "living of the land" binary under Windows[386]. It can be leveraged for downloading files over http/s, we even save the downloaded file as an alternate data stream[387]. By the way, there is also a reference implementation of "certutil.exe" as part of ReactOS[388].



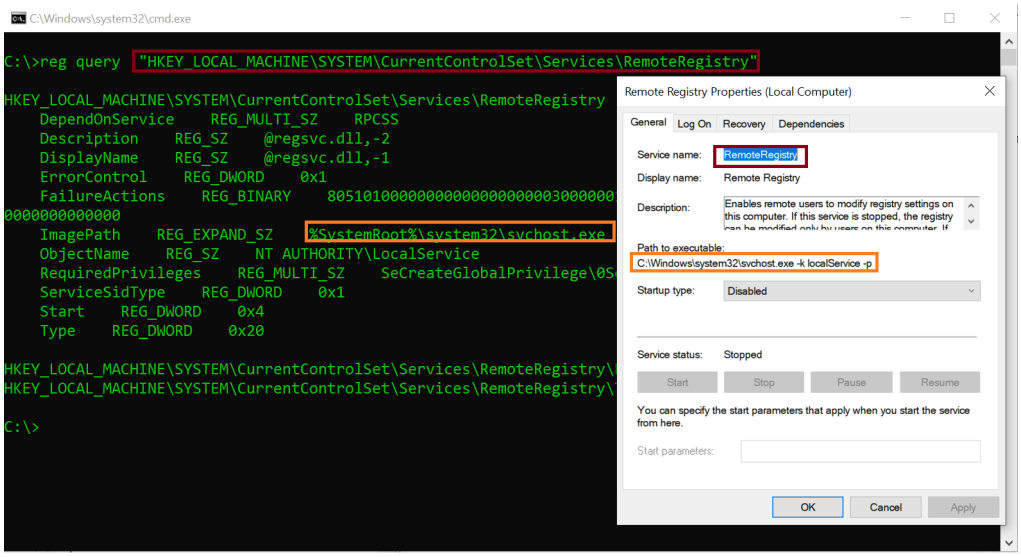
[384] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/certutil
[385] https://ss64.com/nt/certutil.html
[386] https://lolbas-project.github.io/lolbas/Binaries/Certutil/
[387] https://medium.com/@boutnaru/the-windows-concept-journey-ads-alternate-data-stream-4cfafba9088c
[388] https://github.com/reactos/reactos/tree/master/base/applications/cmdutils/certutil

reg.exe (Registry Console Tool)

“reg.exe” (Registry Console Tool) is a binary PE file located at “%windir%\system32\reg.exe”. It is a command line utility which is used for performing registry operations[389]. operations. On 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\reg.exe”.

Overall, using “reg.exe” we can add a new subkey/entry (“reg add”), compare registry subkeys/entries (“reg compare), copy entries (“reg copy”), delete subkeys/entries (“reg delete”), export/import subkeys/entries/values (“reg export” or “reg import”), write/delete subkeys/entries into a different subkey (“reg load” or “reg unload”), get a list of the next tier of subkeys/entries from a specific key (“reg query”), save a copy of subkeys/entries/values (“reg save”) and write from a backup subkeys/entries/values (“reg restore”)[390] - an example is shown in the screenshot below.

Lastly, we can export data from the registry to “*.reg file” and we can also import data from “*.reg file”. We can perform part of the command not only locally but also on a remote machine by leveraging the “Remote Registry” service[391]. For a reference implementation of “reg.exe” I suggest going over the ReactOS implementation[392].



[389] https://medium.com/@boutnaru/the-windows-concept-journey-registry-0767e79387a9
[390] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/reg
[391] https://ss64.com/nt/reg.html
[392] https://github.com/reactos/reactos/tree/master/base/applications/cmdutils/reg

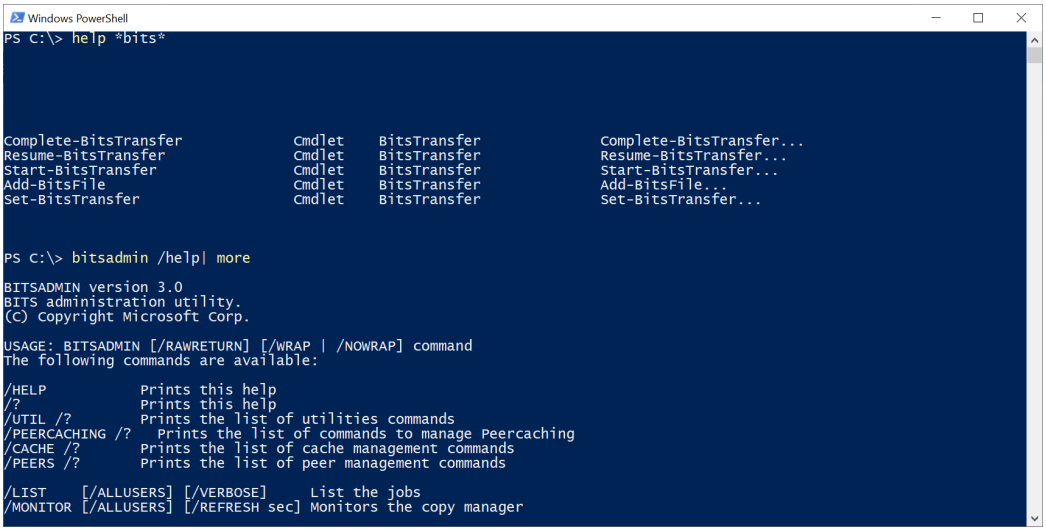
bitsadmin.exe (BITS administration utility)

“bitadmin.exe” is a binary PE file located at “%windir%\system32\bitsadmin.exe”. It is used in order to create download/upload jobs while also monitoring them[393]. On 64-bit systems there is also a 32-bit version of it located at “%windir%\SysWOW64\bitsadmin.exe”. Alos, the binary is digitally signed by Microsoft.

Overall, we can use “bitsadmin.exe” command line utility (which is built in to Windows) in order to manage the BITS (Background Intelligent Transfer) service[394]. Thus, “bitsadmin.exe” allows us to transfer large files from remote hosts, while throttling and asynchronously transferring files between Windows devices using idle network bandwidth. By the way, BITS is used by Windows Update, SUS, SMS and many third party packages[395].

Lastly, today besides the “bitsadmin.exe” utility we can also use “powershell.exe”[396] in order to perform BITS operations. There are a couple of cmdlets for that[397] - as shown in the screenshot below. Due to the fact “bitsadmin.exe” has multiple command line switches I suggest going over the documentation[398] - as shown in the screenshot below.

.



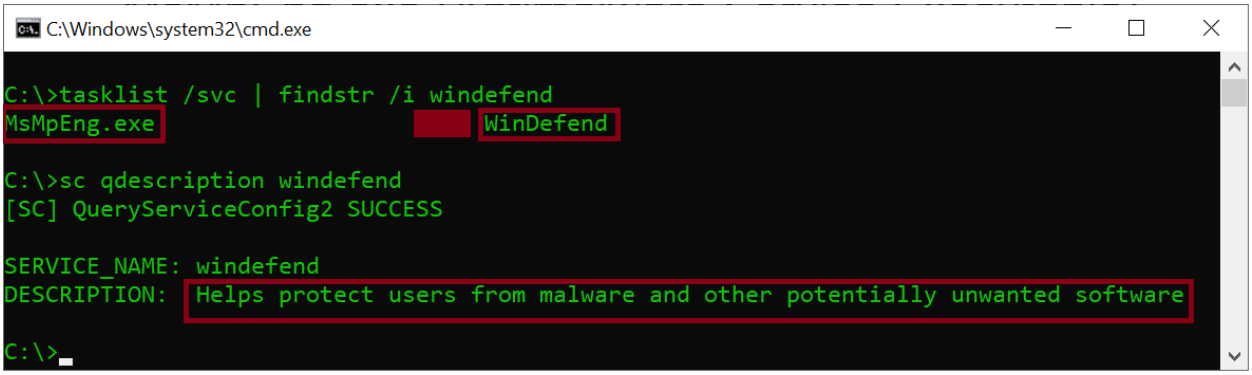
[393] https://learn.microsoft.com/en-us/windows/win32/bits/bitsadmin-tool
[394] https://medium.com/@boutnaru/the-windows-concept-journey-bits-background-intelligent-transfer-service-40532cd09cca
[395] https://ss64.com/nt/bitsadmin.html
[396] https://medium.com/@boutnaru/the-windows-process-journey-powershell-exe-windows-powershell-36daabaa74c4
[397] https://ss64.com/ps/bits.html
[398] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/bitsadmin

MsMpEng.exe (Antimalware Service Executable)

"MsMpEng.exe" is the main binary launched by the "WinDefend" (as shown below) Windows service[399]. The binary is digitally signed by Microsoft and has the following text as part of the PE description field: "Antimalware Service Executable".

Moreover, the description of the services states it helps in protecting users from malware and other potentially unwanted software - as shown in the screenshot below. "MsMpEng.exe" is a core process of "Windows Defender" which is Microsoft's anti malware solution[400].

Lastly, the binary is started from the following location: "C:\ProgramData\Microsoft\Windows Defender\Platform\[VERSION]\MsMpEng.exe". By the way, [VERSION] matches the file version stored in the PE. By the way, in case you see high CPU usage by "MsMpEng.exe" it is best to use the "New-MpPerformanceRecording" PowerShell cmdlet which collects performance recording of "Microsoft Defender Antivirus" scans[401]. This is due to the fact that high CPU usage can be just a symptom of something else[402].



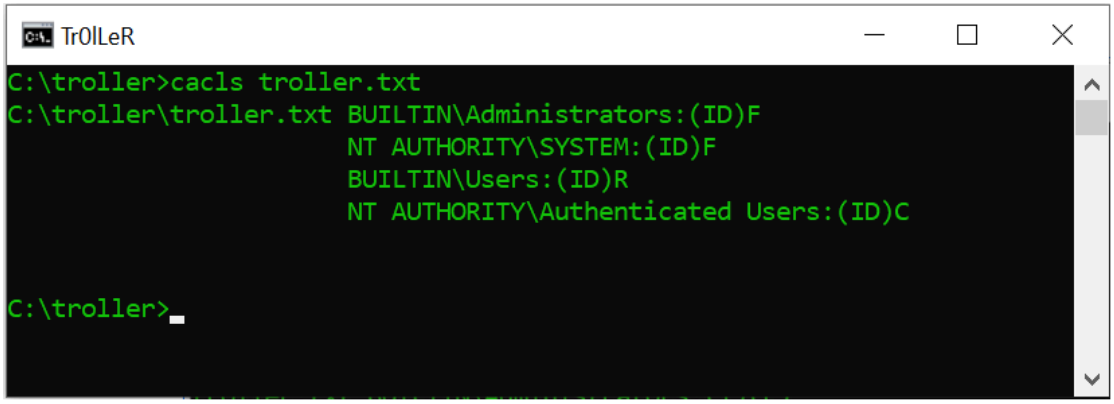
[399] https://medium.com/@boutnaru/windows-services-part-2-7e2bdab5bce4
[400] https://www.neuber.com/taskmanager/process/msmpeng.exe.html
[401] https://learn.microsoft.com/en-us/powershell/module/defenderperformance/new-mpperformancerecording?view=windowsserver2022-ps
[402] https://x.com/SwiftOnSecurity/status/1575625955766194176

cacls.exe (Control ACLs Program)

“cacls.exe” is a PE binary located at “%windir%\System32\cacls.exe”. It is a CLI tool used for displaying/modifying DACLs[403] of files[404] - an example is shown in the screenshot below. Also, the “cacls.exe” file is digitally signed by Microsoft.

Moreover, on 64-bit systems there is also a 32-bit version of “cacls.exe” located at “%windir%\SysWOW64\cacls.exe”. Also, this binary is digitally signed by Microsoft. We can think about it similar to “chmod” on Unix/Linux systems[405].

Thus, we can use the different command line arguments of “cacls.exe” in order to edit an ACL (“/E”), perform the operation on the symbolic link and not on the target (“/L”), grant access rights to a user (“/G”), revoke user’s rights (“/R”), replace access rights (“/P”), deny access for a specific user (“/R”), replace the ACLs using an SDDL string (/S:SSDL) and more[406].



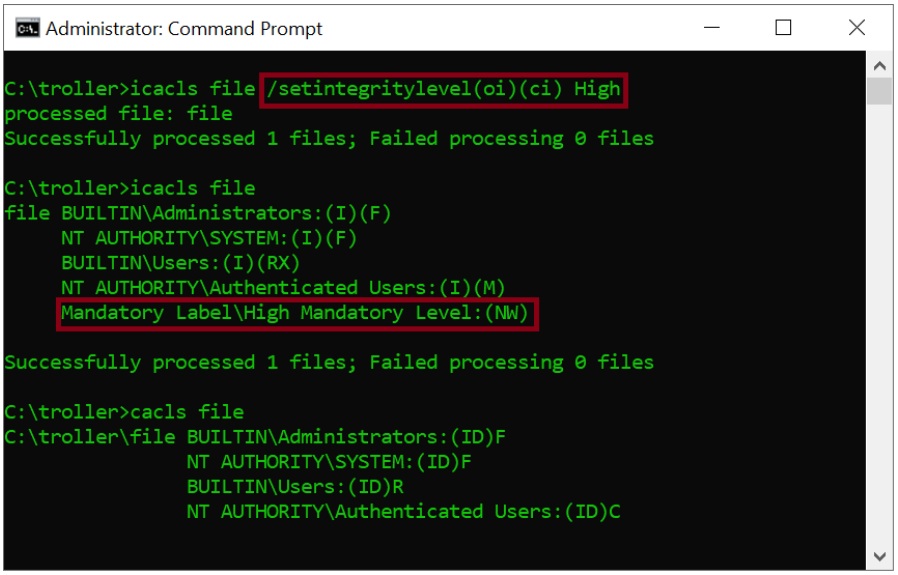
[403] https://medium.com/@boutnaru/the-windows-security-journey-dacl-discretionary-access-control-list-c74545e472ec
[404] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cacls
[405] https://linux.die.net/man/1/chmod
[406] https://www.computerhope.com/cacls.htm

icacls.exe (Integrity Control ACLs Program)

“icacls.exe” is a PE binary located at “%windir%\System32\icacls.exe”. The binary is digitally signed by Microsoft. On 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\icacls.exe”. Like “cacls.exe”[407] the “icacls.exe” utility is also used for displaying/modifying DACLs[408] of files and directories.

Moreover, “icacls.exe” is a replacement of “cacls.exe”[409] due to the fact it supports both viewing/modifying the integrity levels[410] of files - as shown in the screenshot below.

Lastly, we have equivalent PowerShell cmdlets that we use: “Get-Acl”/”Set-Acl”[411]. Alos, there is no reference implementation of “icacls.exe” as part of ReactOS, there is only for “cacls.exe”[412].



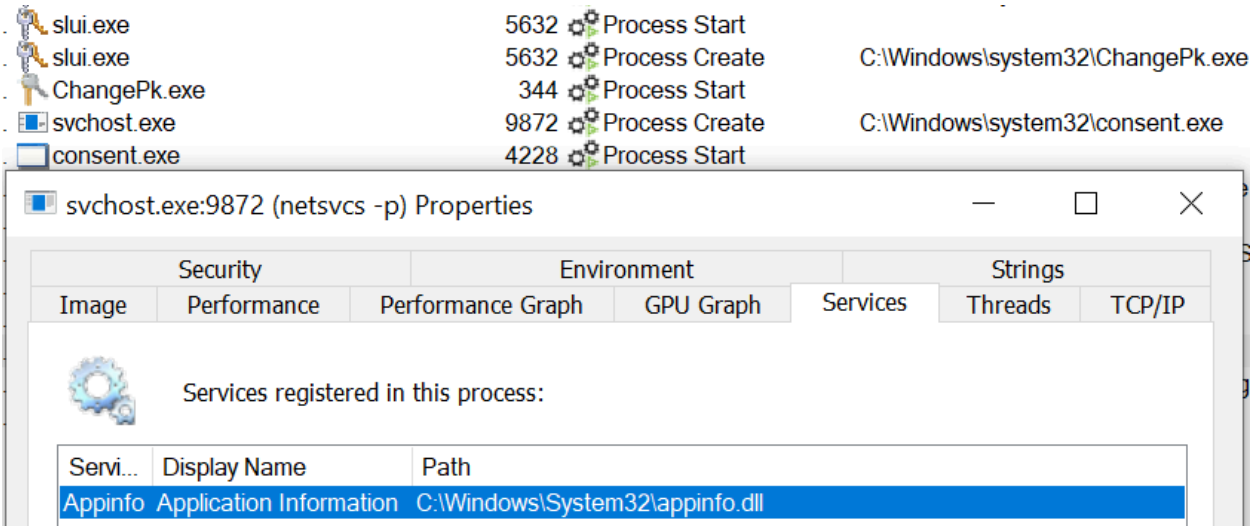
[407] https://medium.com/@boutnaru/the-windows-process-journey-cacls-exe-control-acls-program-296ba9e7761c
[408] https://medium.com/@boutnaru/the-windows-security-journey-dacl-discretionary-access-control-list-c74545e472ec
[409] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/icacls
[410] https://medium.com/@boutnaru/the-windows-security-journey-mandatory-integrity-control-mic-f7963550c0e7
[411] https://petri.com/icacls-command/
[412] https://github.com/reactos/reactos/tree/master/base/applications/cacls

slui.exe (Windows Activation Client)

"slui.exe" (Windows Activation Client) is a PE binary located at "%windir%\System32\slui.exe". On 64-bit systems there is one 32-bit version of "slui.exe". The binary file is digitally signed by Microsoft. Also,"slui.exe" is an auto-elevated binary[413].

Overall, SLUI stands for "Windows Software Licensing User Interface". It runs in the background in order to keep track of the system's activation process. This means that every time we try to change a product key/activate Windows this process manages tha[414]t.

When starting "slui.exe" it starts "changepk.exe", which causes the service "Application Information" to launch "consent.exe" - as shown in the screenshot below. By the way, the description of the "Application Information" services states that it facilitates the running of interactive applications with additional administrative privileges. If this service is stopped, users will be unable to launch applications with the additional administrative privileges they may require to perform desired user tasks.



[413] https://atomicredteam.io/defense-evasion/T1548.002/
[414] https://candid.technology/slui-exe/

xcopy.exe (Extended Copy Utility)

“xcopy.exe” is the “Extended Copy Utility” which is a command line is responsible for copying files/directories including subdirectories[415]. “xcopy.exe” is a PE binary file located at “%windir%\System32\net.exe”, in case of a 64-bit system there is also a 32-bit version located at “%windir%\SysWOW64\xcopy.exe”.

Moreover, “xcopy.exe” is a separate executable in contrast to “copy” which is a builtin command of the “cmd.exe” executable[416]. This means that when we perform a copy operation with “copy” it is done by “cmd.exe”. By the way, the “xcopy.exe” binary is digitally signed by Microsoft.

Lastly, we can say that “xcopy” is similar to “copy” except that it has additional switches (there is also “robocopy” - more on that in a future writeup) - as shown in the table below[417]. Examples of such switches are “/COMPRESS” which can be used for requesting SMB network compression while performing a file transfer and “/J” which supports unbuffered I/O that is recommended for very large files[418]. Also, we can go over a reference implementation of “xcopy” as part of ReactOS[419].

COPY	DISK COPY	XCOPY
Copy and combined file one or more file from one location to another.	Copy the contents of one flopy disk to another flopy of the same size and type.	The xcopy command copies all the file and sub directory into another directory or drive.
C:\>copy <source file> <destination>	C:\> disk copy <source drive> <target drive>	C:\> xcopy <source file> <target file> <switch>
It is a internal command.	It is a external command	It is a powerful external command .

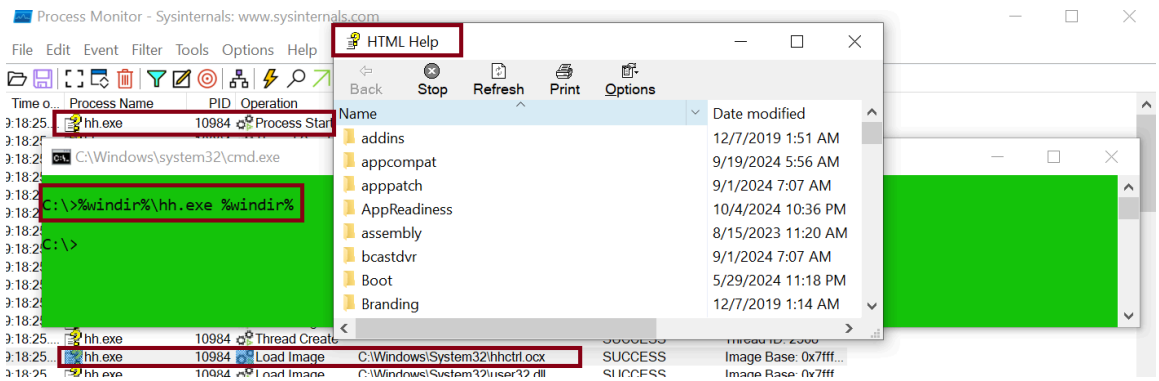
[415] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc771254(v=ws.11)
[416] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[417] https://www.youtube.com/watch?v=cW-j_1qbAf8
[418] https://ss64.com/nt/xcopy.html
[419] https://github.com/reactos/reactos/tree/master/base/applications/cmdutils/xcopy

hh.exe (Microsoft® HTML Help Executable)

“hh.exe” (Microsoft® HTML Help Executable) is a binary PE file located at “%windir%\hh.exe”, on 64-bit system there is also a 32-bit version located at “%windir%\SysWOW64\hh.exe”. It is used to open “*.chm” files (compiled help)[420].

Overall, “*.chm” files are distributed as part of the “Microsoft HTML Help” system. “*.chm” files contain different content like: HTML documents, JScript, VBA and ActiveX This content is parsed/displayed by leveraging component which are part of the browser “Internet Explorer”[421]. By the way, we can also explore file system locations using “hh.exe”- as shown in the screenshot below.

Lastly, we can check out a reference implementation of “hh.exe” as part of ReactOS[422]. Also, “hh.exe” leverages (loads) “hhctrl.ocx” (Microsoft® HTML Help Control) - as shown in the screenshot below. It provides a rich feature set that includes: expanding table of contents, keyword search, shortcuts, and pop-up help topics. Moreover, “hhctrl.ocx supports both compiled html files (*.chm) and uncompiled html files[423].



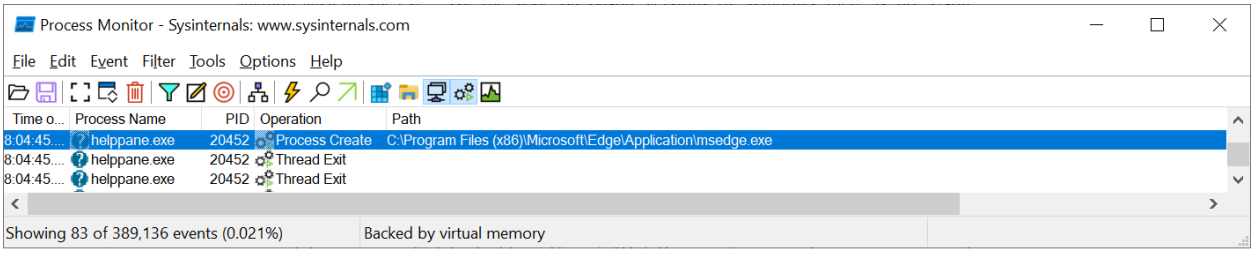
[420] https://learn.microsoft.com/en-us/previous-versions/windows/desktop/htmlhelp/about-the-html-help-executable-program
[421] https://attack.mitre.org/techniques/T1218/001/
[422] https://github.com/reactos/reactos/blob/master/base/applications/hh/main.c
[423] https://learn.microsoft.com/en-us/previous-versions/windows/desktop/htmlhelp/html-help-activex-control-overview

HelpPane.exe (Microsoft Help and Support)

“HelpPane.exe” (Microsoft Help and Support) is a 64-bit PE binary file located at “%windir%\HelpPane.exe”. By the way, on 64-bit versions of Windows there is no 32-bit version of binary. As part of Windows 8/8.1 help components were shipped as part of the operating system. This included the “Help and Support Windows” desktop application aka “HelpPane.exe”. Since Windows 10 “HelpAndSupport” settings are deprecated because the help component they are relevant for has been retired[424].

However, “HelpPane.exe” is still part of the Windows operating system builtin executables. Executing the binary without any parameters does not open any new window. In case we provide command line arguments (such as -Embedding/-Home) “HelpPane.exe” can create a new process of “Microsoft Edge” which is “msedge.exe”[425] - as shown in the screenshot below (taken using “Process Monitor”).

Lastly, running “HelpPane.exe” from Windows 10 can result in launching the “Getting Started” application or opening a browser instance and redirecting to an online topic[426]. Also, the “HelpPane.exe” binary is digitally signed by Microsoft.



[424] https://learn.microsoft.com/en-us/windows-hardware/customize/desktop/unattend/microsoft-windows-helpandsupport-helpandsupport

[425] https://medium.com/@boutnaru/the-windows-process-journey-msedge-exe-microsoft-edge-747e00211a65

[426] https://learn.microsoft.com/en-us/windows-hardware/customize/desktop/unattend/microsoft-windows-helpandsupport

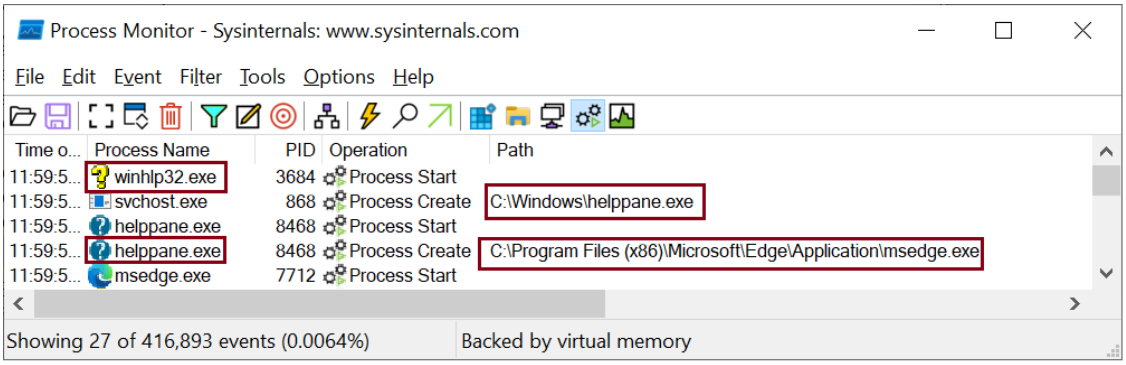
winhlp32.exe (Windows Winhlp32 Stub)

"winhlp32.exe" (Windows Winhlp32 Stub) is a 32-bit PE binary file located at "%windir%\winhlp32.exe". By the way, even on 64-bit versions of the Windows "winhlp32.exe" is a 32-bit binary (there is no 64-bit version). By the way, the "Windows Help" application is not supported since Windows 10/Windows Server 2012. Thus, "winhlp32.exe" is supported for Windows Vista, Windows 7 Windows 8 or Windows 8.1[427]. For a reference implementation of "winhelp32.exe" we can check out the source code of ReactOS[428].

Overall, it is used for viewing 32-bit "*.hlp" files. If we want to read them on newer versions of Windows we can download "winhlp32.exe" from the "Microsoft Download Center"[429]. For security reasons some of the macros supported by "winhlp32.exe" are disabled such as: "ExecFile", "ShortCut", "Test", "ShellExecute", "Generate", "ExecProgram" and "RegisterRotine"[430].

Moreover, from Windows 10 when running the "winhelp32.exe" binary it starts "HelpPane.exe"[431], this is done using the "DCOM Server Process Launcher" service which is hosted by the "svchost.exe" process[432] - as shown in the screenshot below taken using "Process Monitor".

Lastly, "HelpPane.exe" launches a web browser application - as shown in the screenshot below. The browser opens a website which states "Error opening Help in Windows-based programs: "Feature not included" or "Help not supported""[433]. Also, we can't access "*.hlp" files stored on intranet sites.



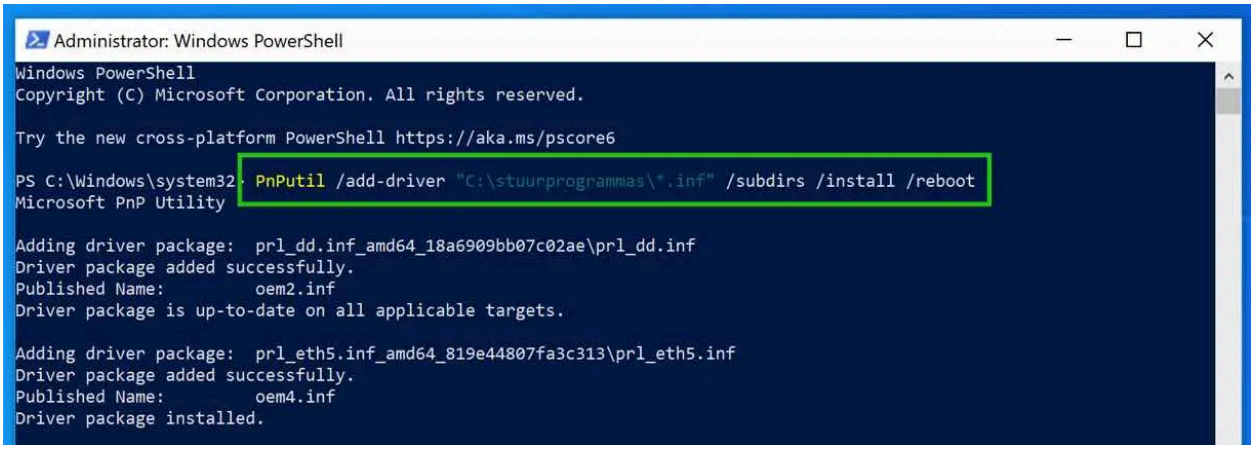
[427] https://support.microsoft.com/en-us/topic/error-opening-help-in-windows-based-programs-feature-not-included-or-help-not-supported-3c841463-d67c-6062-0ee7-1a149da3973b
[428] https://github.com/reactos/reactos/blob/master/base/applications/winhlp32/winhelp.c
[429] https://www.microsoft.com/en-us/download/details.aspx?id=35449
[430] https://www.sevenforums.com/tutorials/141117-help-hlp-files-cannot-open-windows-fix.html
[431] https://medium.com/@boutnaru/the-windows-process-journey-helppane-exe-microsoft-help-and-support-0174ea107681
[432] https://medium.com/@boutnaru/the-windows-process-journey-svchost-exe-host-process-for-windows-services-b18c65f7073f
[433] https://support.microsoft.com/en-us/topic/error-opening-help-in-windows-based-programs-feature-not-included-or-help-not-supported-3c841463-d67c-6062-0ee7-1a149da3973b

pnputil.exe (Plug and Play Utility)

“pnputil.exe” (Plug and Play Utility) is a PE binary file located at “%windir%\System32\pnputil.exe”. On 64-bit versions there is no parallel 32-bit version of the binary. It is used for managing device drivers. By the way, the binary is digitally signed by Microsoft and included in every version of Windows starting with Windows Vista[434].

Overall, based on the description stored in the PE header, “pnputil.exe” is a command line utility which can be used for adding\installing or deleting or exporting or enumerating driver packages - as shown in the screenshot below[435]. We can checkout Microsoft’s documentation GitHub repository for PnPUtil usage examples[436]. For a reference implementation of “pnputil.exe” we can checkout the source code or ReactOS[437].

Lastly, among the return values we can get form PnPUtil we can find: “ERROR_SUCCESS” (the requested operation completed successfully), “ERROR_NO_MORE_ITEMS” (no devices match the supplied driver or the target device is already using a better\newer driver), “ERROR_SUCCESS_REBOOT_REQUIRED” (the operation completed successfully and a system reboot is required) and “ERROR_SUCCESS_REBOOT_INITIATED” (The operation was successful and a system reboot is underway) as described in the documentation[438].



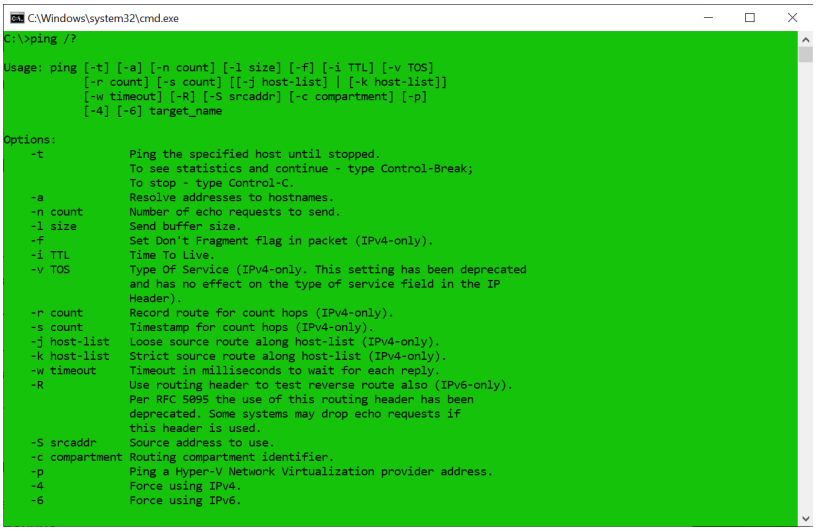
[434] https://ss64.com/nt/pnputil.html
[435] https://www.pc-tips.info/tips/backup-maken-van-windows-10-drivers-en-drivers-herstellen/
[436] https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/devtest/pnputil.md5
[437] https://github.com/reactos/reactos/blob/master/ntoskrnl/io/pnpmgr/pnputil.c
[438] https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windows-driver-docs-pr/devtest/pnputil-return-values.md

ping.exe (TCP/IP Ping Command)

“ping.exe” (TCP/IP Ping Command) is a PE binary located at “%windir%\System32\PING.exe”. On 64-bit systems there is a 32-bit version of the binary located at “%windir%\SysWOW64\PING.EXE”. It is used for verifying IP connectivity with another node in a TCP/IP based network. This is done by sending an ICMP (Internet Control Management Protocol) “echo request” message. The receipt of replies with an “echo reply” message which is displayed, along with round-trip times[439].

Overall, “ping.exe” supports both IPv4 and IPv6 based communication. The binary is also digitally signed by Microsoft. “ping.exe” is similar to the ping command in other operating systems like Linux[440] or macOS[441]. The backronym “Packet InterNet Groper” has been used for ping more than 30[442].

Lastly, we have different command line arguments supported by “ping.exe” that can modify its behavior - as shown in the screenshot below. Examples of such arguments are: “-t” which sends echo request messages until we stop it using “Ctrl+C”, “-a” that tries to resolve the IP address and “-i” which allows setting the value of the TTL (Time To Live) field as part of the IP header[443]. We can checkout a reference implementation of “ping.exe” as part of ReactOS[444].



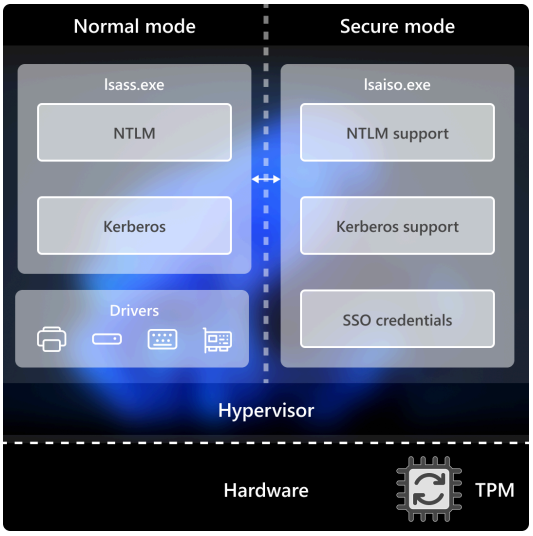
[439] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/ping
[440] https://linux.die.net/man/8/ping
[441] https://ss64.com/mac/ping.html
[442] https://en.wikipedia.org/wiki/Ping_(networking_utility)
[443] https://www.lifewire.com/ping-command-2618099
[444] https://github.com/reactos/reactos/blob/master/base/applications/network/ping/ping.c

LsaIso.exe (Credential Guard & Key Guard)

"LsaIso.exe" is a PE binary located at "%windir%\system32\LsaIso.exe". On 64-bit versions of Windows we don't have a 32-bit version of the binary as with other binaries like "cmd.exe"[445]. It is used as part of "Credentials Guard" in order to isolate secrets (NTLM\Kerberos crypto materials) by leveraging VBS (Virtualization Based Security). This is due to the fact Windows stores secrets in the address space of "lsass.exe"[446]. Thus, they can be dumped/extracted by different tools like Mimikaz[447].

Overall, in case "Crentails Guard" is enabled "lsass.exe" (LSA process) talks with "LsaIso.exe" (Isolated LSA) which stores the secrets by leveraging VBS (more on that in a future writeup) - as shown in the diagram below. Because of that the secrets are not accessible by other components/processes running on the system even with Administrator/Local System access[448].

Lastly, It is important to understand that there are different cases in which "Credentials Guard" won't be able to protect Windows secrets like: keyloggers and non-microsoft security packages. LSA isolated ("LsaIso.exe") runs as an IUM (Isolated User Mode) process[449]. By the way, the "LsaIso.exe" is digitally signed by Microsoft.



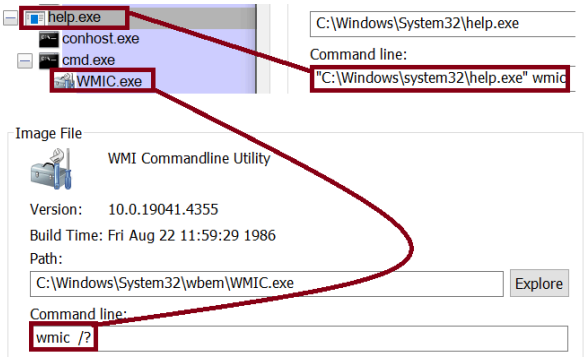
[445] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[446] https://medium.com/@boutnaru/the-windows-process-journey-lsass-exe-local-security-authority-process-24166cb0358f
[447] https://github.com/ParrotSec/mimikatz
[448] https://learn.microsoft.com/en-us/windows/security/identity-protection/credential-guard/how-it-works
[449] https://learn.microsoft.com/en-us/troubleshoot/windows-client/performance/lsaiso-process-high-cpu-usage

help.exe (Command Line Help Utility)

"help.exe" (Command Line Help Utility) is a PE binary file located at "%windir%\System32\help.exe". On 64-bit versions of Windows there is also a 32-bit version of the binary located at "%windir%\SysWOW64\help.exe". It is used for providing help information on a few command line utilities (for example "cmd.exe" and "powershell.exe") and some shell built in commands like "assoc" and "call"[450].

Overall, the help information is not stored as part of "help.exe". Every time we want help information about a command line utility "help.exe" runs the following command "cmd /c {COMMAND_LINE_UTILITY} /?" using the "_wsystem" API function[451]. If the command line utility is not a built in command of "cmd.exe"[452] the utility is executed under "cmd.exe" - as shown in the screenshot below

Lastly, we can check out a reference implementation of "help.exe" as part of ReactOS[453]. Also, the help text displayed by "help.exe" is also not included directly inside the binary. This is due to the fact the "Multilingual User Interface"[454] is leveraged (for example "%windir%\en-US\helppane.exe.mui"). By the way, the binary is digitally signed by Microsoft.



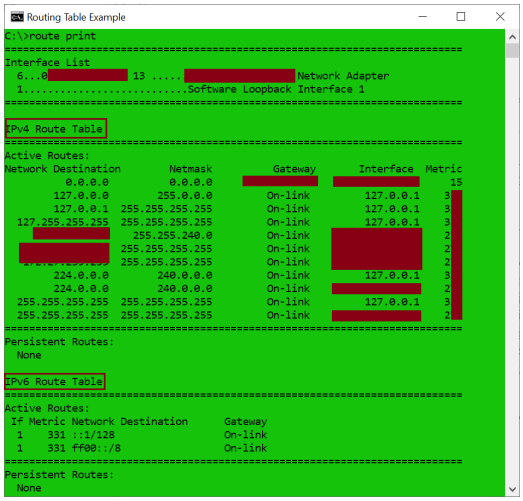
[450] https://renenyffenegger.ch/notes/Windows/dirs/Windows/System32/help_exe
[451] https://learn.microsoft.com/en-us/cpp/c-runtime-library/reference/system-wsystem?view=msvc-170
[452] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[453] https://github.com/reactos/reactos/blob/master/base/applications/cmdutils/help/help.c
[454] https://medium.com/@boutnaru/the-windows-concept-journey-multilingual-user-interface-mui-c225998d9262

route.exe (TCP/IP Route Command)

“route.exe” (TCP/IP Route Command) is a PE binary located at “%windir%\System32\ROUTE.EXE”. On 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\ROUTE.EXE”. It is used for displaying/modifying entries of the local IP routing table. It is important to know that this command line utility is available only if the Internet Protocol (TCP/IP) protocol is installed as a component in the properties of a network adapter[455].

Overall, by manipulating the routing table using “route.exe” we can route packets of network traffic from one subnet to another. By default routes added (based on destination IP and subnet mask) are not persistent unless the” -p” switch is used. Also, each route has its own metric (cost for destination) which allows us to have multiple paths for redundancy while prioritizing between them. The utility also supports patterns in different fields[456]. We can think about “route.exe” as similar to “ip route” on Linux systems[457]. In case both IPv4 and IPv6 are enabled “route.exe” will show us both routing tables - as shown in the screenshot below.

Lastly, “route.exe” leverages different API calls for reading/altering the routing table. Example of such APIs are: “GetIpForwardTable” for retrieving the IPv4 routing table[458] and “GetAdaptersAddresses” for retrieving addresses associated with adapters on the device[459]. For a reference implementation of “route.exe” we can check out the source code of ReactOS[460].



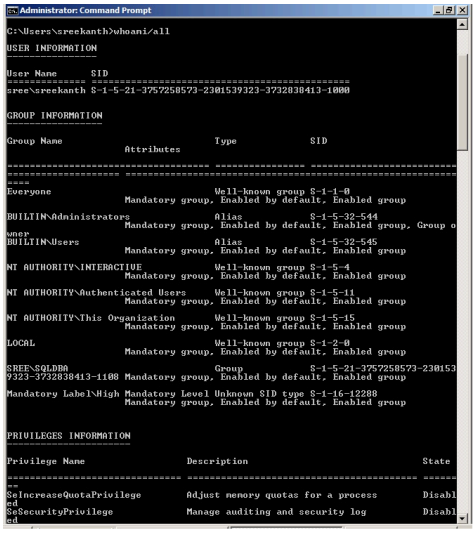
[455] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/route_ws2008
[456] https://ss64.com/nt/route.html
[457] https://ss64.com/bash/ip-route.html
[458] https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getipforwardtable
[459] https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getadaptersaddresses
[460] https://github.com/reactos/reactos/blob/master/base/applications/network/route/route.c

whoami.exe (Displays Logged On User Information)

“whoami.exe” (Displays Logged On User Information) is a PE binary located at “%windir%\System32\whoami.exe”. On 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\whoami.exe”. It is used for displaying user\group and privileges information for the currently logged on user. In case we execute “whoami.exe” without any parameters the current domain and user name are shown[461].

Overall, we can use “whomai.exe” in order to show: the UPN (user principal name) of the current user (/upn), logon ID of the current user (/logonid), privileges of the current user (/priv), showing the groups to which the current user belongs to (/groups) and more. The output of the command can also be in: table (default), list or csv (/fo FORMAT). For showing all information from the current access token we can use the “/all” argument[462] - as shown in the screenshot below[463].

Lastly, we can think about “whoami.exe” as an equivalent to the “whoami” command on Linux[464]. Also, for a reference implementation of “whomai.exe” we can check out the source code of ReactOS[465].



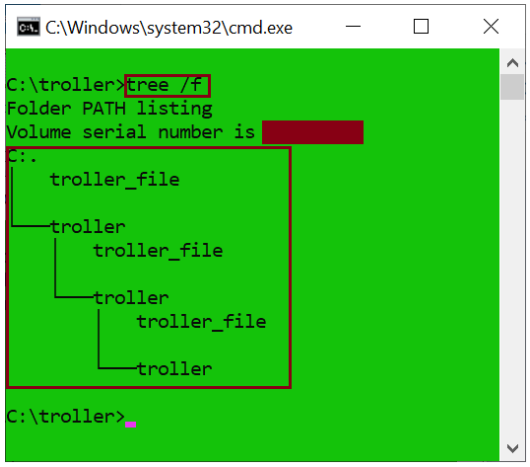
[461] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/whoami
[462] https://ss64.com/nt/whoami.html
[463] https://mssqltrek.com/2012/03/12/whoami-and-echo-in-windows/
[464] https://man7.org/linux/man-pages/man1/whoami.1.html
[465] https://github.com/reactos/reactos/blob/master/base/applications/cmdutils/whoami/whoami.c

tree.com (Tree Walk Utility)

“tree.com” (Tree Walk Utility) is a PE binary (although it has a “.com” extension) located at “%windir%\System32\tree.com”. On 64-bit versions of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\tree.com”. It is used for displaying the directory structure of a path or a disk in a drive graphically (CLI based). If we don't specify a drive\path the tree structure beginning with the current directory is printed[466] - as shown in the screenshot below.

Overall, it is an equivalent to the “tree” command in Linux[467]. The “tree.com” utility is relevant for MS-DOS, Windows 200, Windows XP, Windows Vista, Windows 8, Windows 10 and Windows 11. In order to include also the files inside the directories we can use the “/F” switch[468] - as shown in the screenshot below.

Lastly, the “tree.com” binary is digitally signed by Microsoft. By the way, for a reference implementation of “tree.com” we can check out the source code of ReactOS[469].



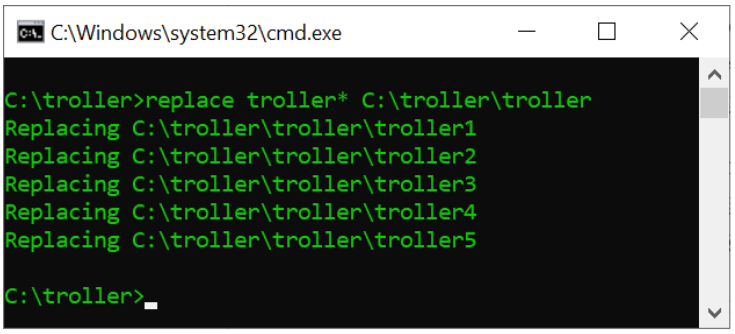
[466] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/tree
[467] https://linux.die.net/man/1/tree
[468] https://www.computerhope.com/treehlp.htm
[469] https://github.com/reactos/reactos/blob/master/base/applications/cmdutils/tree/tree.c

replace.exe (Replace File Utility)

“replace.exe” (Replace File Utility) is a PE binary located at “%windir%\System32\replace.exe”. On 64-bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\replace.exe”. It is used for replacing existing files in a directory. If we use the “/A” switch the utility adds new files to a directory as opposed to replacing existing files[470].

Overall, “replace.exe” is mostly relevant for cases where we want to replace files having the same filename. There are multiple arguments that we can use for setting the behavior of the utility such as: prompting for confirmation for each file (“/P”), replace read-only file (“/R”), include sub-directories of the destination (“/S”), update only the files which are older then the source (“/U”) and more[471].

Lastly, for a reference implementation of “replace.exe” we can check out the code of ReactOS[472]. Also, “replace.exe” is digitally signed by Microsoft. For each file which is replaced a message is displayed - as shown in the screenshot below.



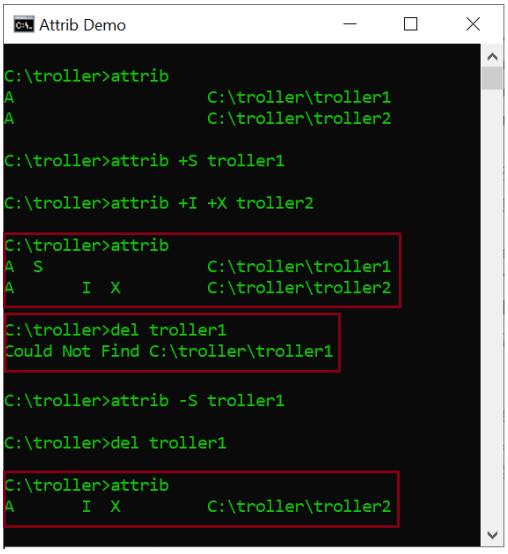
[470] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/replace
[471] https://ss64.com/nt/replace.html
[472] https://github.com/reactos/reactos/blob/master/base/applications/cmdutils/replace/replace.c

attrib.exe (Attribute Utility)

“attrib.exe” (Attribute Utility) is a PE binary located at “%windir%\System32\attrib.exe”. On 64-bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\attrib.exe”. The utility is used for displaying, setting and\or removing attributes assigned to files\directories. In case executing “attrib.exe” without any parameters it displays the attributes for all files in the current directory[473]. By the way, the binary is digitally signed by Microsoft.

Overall, if we want to set and attribute we use the “+” sign before the letter describing the attribute. For removing an attribute we use the “-” sign. We can cluster the attributes to two different groups: “attributes” and “extended attributes”. In the first group we have: read-only (R), archive (A), system (S) and hidden (H). In the second group we have SMB blob (B), encrypted (E), compressed (C), non indexed content (I), normal (N), offline (O), temporary (T), integrity (I), no scrub (X), pinned which means "Always available on this device" setting for OneDrive files and unpinned (U)[474] - as shown in the screenshot below.

Lastly, we can also use “attrib.exe” for processing folders and not only files (“/D”) and/or apply also for sub-directories (“/S”). Moreover, we can also perform tasks on the attributes of the symbolic link (“/L”) versus the target of the Symbolic link[475]. For a reference implementation of “attrib.exe” we can check out the code of ReactOS[476].



[473] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/attrib
[474] https://ss64.com/nt/attrib.html
[475] https://monovm.com/blog/attrib-command/
[476] https://github.com/reactos/reactos/blob/master/base/applications/cmdutils/attrib/attrib.c

tabcal.exe (Digitizer Calibration Tool)

“tabcal.exe” (Digitizer Calibration Tool) is a PE binary located at “%windir%\System32\tabcal.exe”. On 64-bit systems there is no parallel 32-bit version of the binary as opposed to other utilities like “cmd.exe”[477].

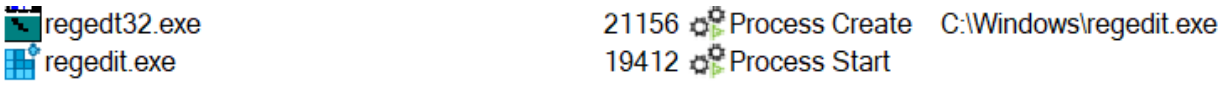
Overall, “tabcal.exe” is used for calibrating touch screens as part of initial setup and/or because of input issues - as shown in the screenshot below. By the way, in case we want to clear saved calibration we just need to provide the “ClearCal” and “DisplayID” arguments to the “tabcal.exe” executable for example: “tabcal.exe ClearCal DisplayID=\\.\DISPLAY1”[478].

Lastly, the “tabcal.exe” is also described as “Tablet PC Calibration”. Also, the “tabcal.exe” binary is digitally signed by Microsoft.

[477] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[478] https://myelo.elotouch.com/support/s/article/How-to-Calibrate-HID-Monitor-Using-Windows-Tabcal

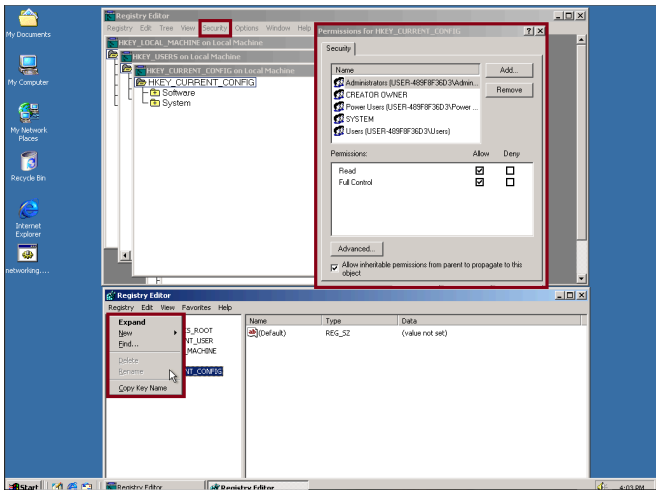
regedt32.exe (Registry Editor Utility)

"regedt32.exe" (Registry Editor Utility) is a PE binary file located at "%windir%\system32\regedt32.exe". On 64-bit systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\regedt32.exe". Since Windows XP/Windows 2003 Server "regedt32.exe" executes "regedit.exe"[479] - as shown in the screenshot next.



Overall, "regedt32.exe" is based on the MDI (Multiple Document Interface) which means a single program can display one or more windows\documents[480]. - as shown in the screenshot below (taken using https://copy.sh/v86/?profile=windows2000).

Lastly, one of the big differences between "regedit.exe" and "regedt32.exe" in old Windows versions was that "regedt32.exe" supported reading/modifying permissions to registry keys - as shown also in the screenshot below. For a reference implementation of "regedt32.exe" we can check out the source code of ReactOS[481].



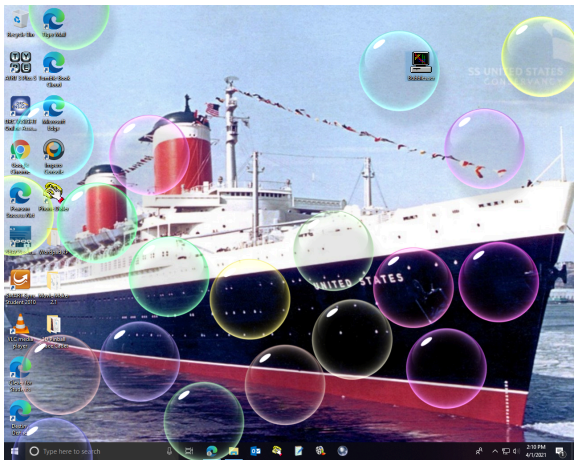
[479] https://superuser.com/questions/295605/what-are-main-differences-between-two-windows-registry-editors-regedit-and-reged
[480] https://www2.isye.gatech.edu/~mgoetsch/cali/Windows%20Configuration/Windows%20Configuration%20Html/WindowsNT4_02.htm
[481] https://github.com/reactos/reactos/blob/master/base/applications/regedt32/regedt32.c

Bubbles.scr (Bubbles ScreenSaver)

“Bubbles.scr” (Bubbles ScreenSaver) is a PE binary (with a “.scr” extension) located at “%windir%\system32\Bubbles.scr”. On 64-bit systems there is no parallel 32-bit version of the binary as opposed to other utilities like “cmd.exe”[482]. By the way, the binary is digitally signed by Microsoft.

Overall, this screensaver basically draws bubbles over the user’s desktop using different colors - as shown in the screenshot below[483]. By the way, a screensaver is a computer program that can fill the screen with images/patterns when the computer is idle[484].

Lastly, the bubbles screensaver does not have any options\ settings that are modifiable using the “Screen Saver Settings” menu. However, we can still customize it by creating/altering registry values in the following location: “HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Screensavers\Bubbles”. Among those values are: “ShowShadows”, “MaterialGlass”, "SphereDensity” and “SpanMultiMon”[485].



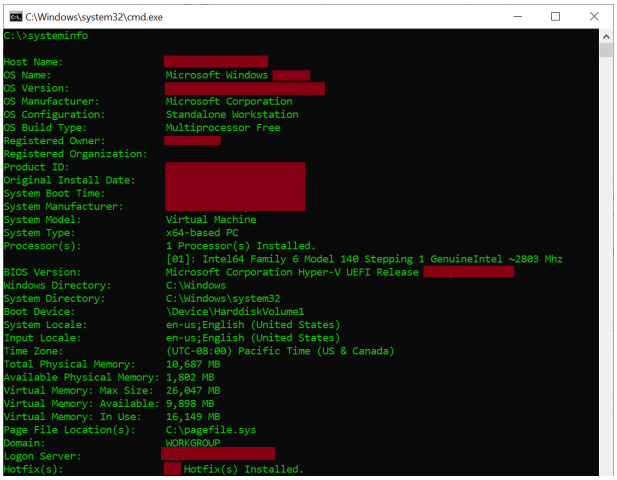
[482] https://medium.com/@boutnaru/the-windows-process-journey-cmd-exe-windows-command-processor-501be17ba81b
[483] https://archive.org/details/bubbles_screensaver_20210401
[484] https://en.wikipedia.org/wiki/Screensaver
[485] https://winaero.com/customize-screen-savers-in-windows-10-using-secret-hidden-options/

systeminfo.exe (Displays system information)

“systeminfo.exe” (Displays system information) is a PE located at “%windir%\system32\systeminfo.exe”. On 64-bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\systeminfo.exe”. It is used for displaying configuration information regarding a specific system (local/remote) and its operating system. Among the information displayed we can find: operating system configuration, security information, product ID and hardware properties like RAM\disk space\network cards[486] - as shown in the screenshot below (the output is not full).

Overall, we can collect and display the information of a remote system (using the “/S” switch) while providing a username (“/U”) and a password (“/P”) for a user context to execute. Also, there are options for specifying the output format (“/FO”) which include: “Table”, “List” or “CSV”[487].

Lastly, the utility is relevant from Windows XP (Professional only) until (and including) Windows 11[488]. For a reference implementation of “systeminfo.exe” we can check the code of ReactOS[489]. By the way, we can think about “systeminfo.exe” as similar (but not identical) to PowerShell’s “Get-ComputerInfo” cmdlet[490] and “msinfo32.exe”.



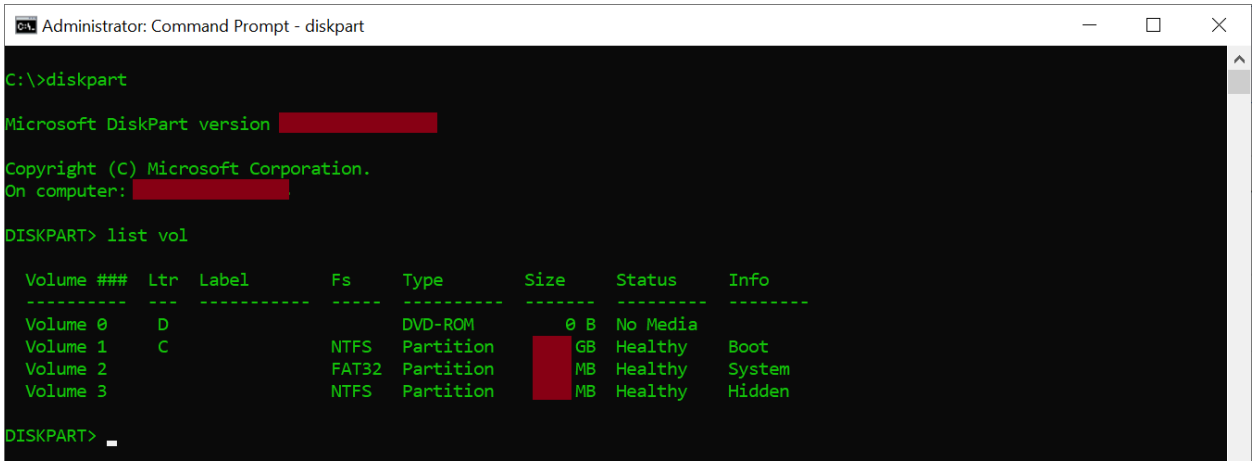
[486] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/systeminfo
[487] https://ss64.com/nt/systeminfo.html
[488] https://www.computerhope.com/systemin.htm
[489] https://github.com/reactos/reactos/blob/master/modules/rosapps/applications/sysutils/systeminfo/systeminfo.c
[490] https://blog.idera.com/database-tools/get-computerinfo-vs-systeminfo-exe-part-1/

diskpart.exe (Microsoft DiskPart Utility)

"diskpart.exe" (Microsoft DiskPart Utility) is a PE binary located at "%windir%\system32\diskpart.exe". On 64-bit systems there is a 32-bit version of the binary located at "%windir%\SysWOW64\diskpart.exe". It is a command line interpreter which is used to manage the systems' drivers (disks/partitions/volumes/virtual disks) - as shown in the screenshot below. It is important to understand that admin permissions are needed (for example being part of the local Administrators group) in order to use "diskpart.exe"[491].

Overall, "diskpart.exe" can be used for converting a FAT/FAT32 volume to NTFS without affecting the files/directories (using the "convert" command). Also, the utility supports MBR (Master Boot Record) and GPT (GUID Partition Table) partition layout schemes. "diskpart.exe" may also be used with VHD (Virtual Hard Disk) files for attaching/detaching and more[492]. The binary is digitally signed by Microsoft and in case we don't have admin permissions "consent.exe"[493] is executed for starting a new process with the relevant permissions.

Lastly, the "diskpart.exe" utility has been available since Windows 2000. By the way, we can also create a text file with diskpart commands and leverage it as a script (by using the "/s" switch). For a reference implementation of "diskpart.exe" we can check out the source code of ReactOS[494].



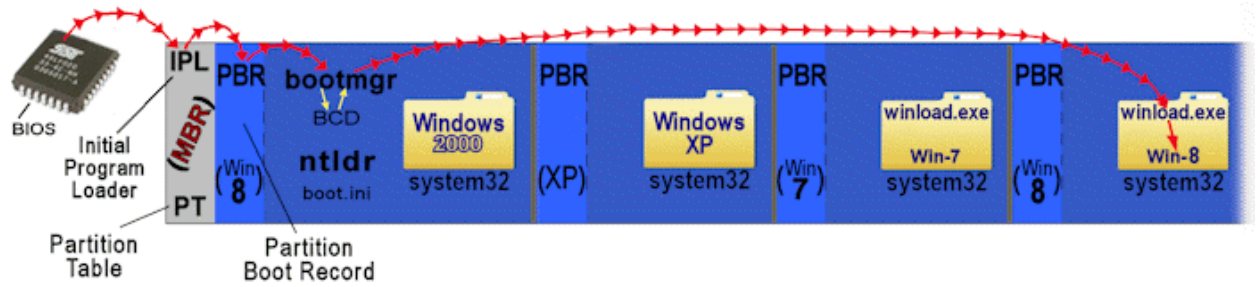
[491] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/diskpart
[492] https://ss64.com/nt/diskpart.html
[493] https://medium.com/@boutnaru/the-windows-process-journey-consent-exe-consent-ui-for-administrative-applications-d8e6976e8e40
[494] https://github.com/reactos/reactos/tree/master/base/system/diskpart

bootmgr.exe (Windows Boot Manager)

“bootmgr.exe” (Windows Boot Manager) is a PE binary that can be located in one of two locations. The binary could be located on the primary drive (for example “C:\). Also, “bootmgr.exe” could be located as part of the “System Reserved Partition”. “bootmgr.exe” (together with “winload.exe”) replaces NTLDR which was used in older versions of Windows such as “Windows XP”[495].

Overall, in case our system leverages BIOS firmware (instead of UEFI firmware, more on that in a future writeup) it calls the MBR (Master Boot Record) which jumps to the VBR (Volume Boot Loader) Windows’ specific code. The VBR (aka PBR which stands for “Partition Boot Record”) code loads “bootmgr.exe” which reads BCD (Boot Configuration Data) for determining which OSes are present and if to display a menu for selecting between boot options. By the way, until Vista the data was stored in “boot.ini”[496] - as shown in the diagram below[497].

Lastly, it is important to know that without BOOTMGR the operating system won’t load and the following error message would display “‘BOOTMGR is missing press Crl+Alt+Del to restart”[498]. For a reference implementation of “bootmgr.exe” we can check out the source of ReactOS[499].



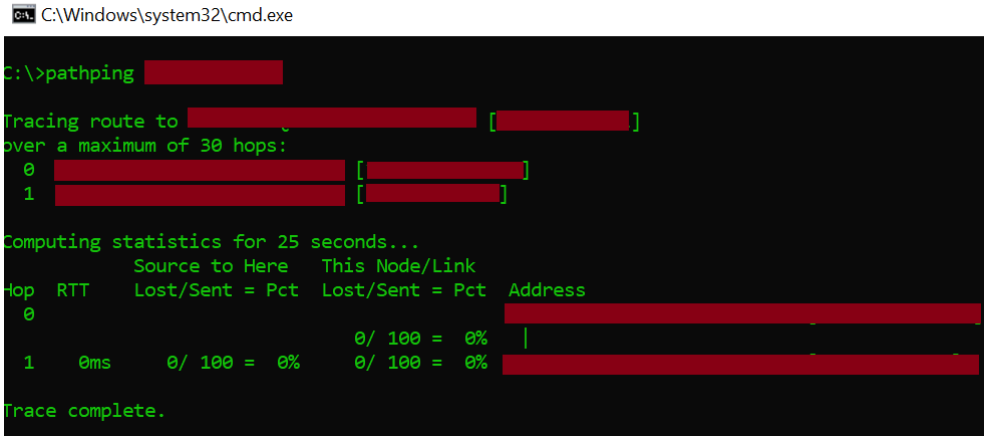
[495] https://www.lifewire.com/windows-boot-manager-bootmgr-2625813
[496] https://en.wikipedia.org/wiki/Windows_Boot_Manager
[497] https://web.archive.org/web/20220312143249/http://www.multibooters.com/guides/boot-sequence-of-mixed-windows-multiboot.html
[498] https://www.diskpart.com/articles/bootmgr-is-missing-5740i.html
[499] https://github.com/reactos/reactos/blob/master/boot/environ/app/bootmgr/bootmgr.c

PathPing.exe (TCP/IP PathPing Command)

"PathPing.exe" (TCP/IP PathPing Command) is a PE binary located at "%windir%\system32\PATHPING.EXE". On 64-bit systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\PATHPING.EXE". We can use it for collecting information about network latency\network loss at intermediate hops for network transmissions[500] - as shown in the example below. By the way, the binary is also digitally signed by Microsoft.

Overall, we can think about "pathping.exe" as a combination between "ping.exe"[501] and "tracert.exe". This command line utility has been included as part of the operating system since "Windows 2000"[502]. The way in which "PathPing.exe" works is by sending packets (using ICMP) to each router on the way to a final destination over a period of time. Based on those packets it computes results from each hop[503].

Lastly, by providing different switches we can control the behavior of "PathPing.exe" like forcing IPv4 ("-4") or IPv6 ("-6), declare a wait period between ICMP echo requests ("-p"), setting the max number of hops to search for a target ("-h"), disabling the resolving of addresses to hostnames ("-n"), checking for RSVP (Resource Reservation Protocol) awareness ("-R") and more[504].



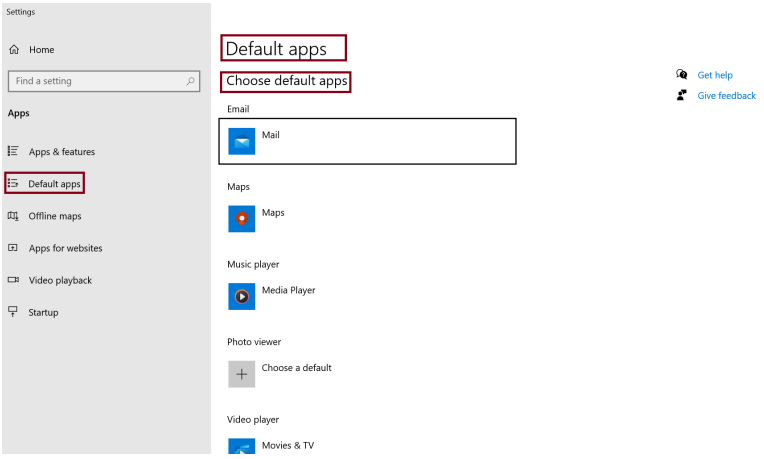
[500] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/pathping
[501] https://medium.com/@boutnaru/the-windows-process-journey-ping-exe-tcp-ip-ping-command-80d958f515d8
[502] https://en.wikipedia.org/wiki/PathPing
[503] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc958876(v=technet.10)
[504] https://ss64.com/nt/pathping.html

ComputerDefaults.exe (Set Program Access and Computer Defaults Control Panel)

“ComputerDefaults.exe” (Set Program Access and Computer Defaults Control Panel) is a PE binary located at “%windir%\system32\ComputerDefaults.exe”. On 64-bit systems there is also a 32-bit version of the binary located at “%windir%\SysWOW64\ComputerDefaults.exe”. It is used for managing/configuring default applications for different tasks such as emailing, web browsing, video playing and mode[505] - as shown in the screenshot below (it is the “Windows 10” layout, under “Windows 11” the layout is different). By the way, the binary is also digitally signed by Microsoft.

Overall, “ComputerDefaults.exe” needs administrator permissions because of that “consent.exe”[506] is launched (by the “AppInfo” service). In case the user approves (if he needs to based on the UAC settings) it causes the “Defaults App” menu of “SystemSettings.exe” to appear[507]. This is done using different services and processes related to the “Universal Windows Platform” (UWP).

Lastly, when “ComputerDefaults.exe” is started it checks for the specific values in the following registry location “HKCU\Software\Classes\ms-settings\Shell\Open\command”. By setting them we can instruct “ComputerDefaults.exe” to execute commands. Due to the fact the binary is auto elevated we can use it as a UAC[508] bypass[509]. It is important to know that “Windows Defender” flags that as “VirTool:Win32/UACBypassExp.gen!B”.



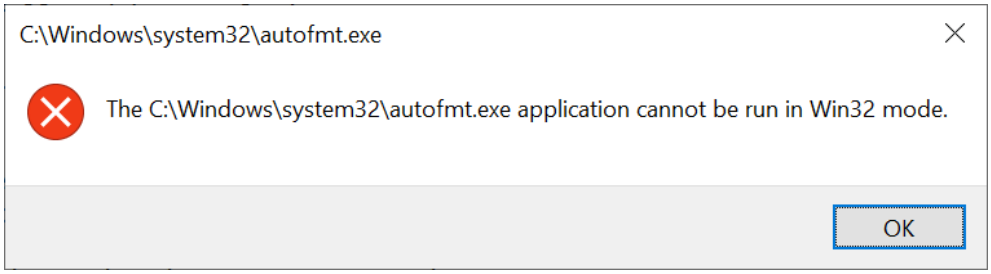
[505] https://lolbas-project.github.io/lolbas/Binaries/ComputerDefaults/
[506] https://medium.com/@boutnaru/the-windows-process-journey-consent-exe-consent-ui-for-administrative-applications-d8e6976e8e40
[507] https://medium.com/@boutnaru/the-windows-process-journey-systemsettings-exe-immersive-control-panel-system-settings-app-930969b84b40
[508] https://medium.com/@boutnaru/the-windows-security-journey-uac-user-account-control-ce395df5c784
[509] https://github.com/blue0x1/uac-bypass-oneliners

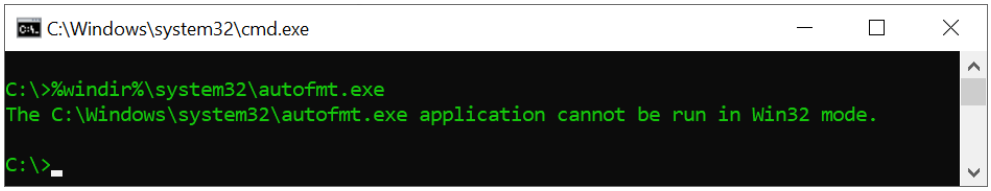
autofmt.exe (Auto File System Format Utility)

"autofmt.exe" (Auto File System Format Utility) is a PE binary located at "%windir%\system32\autofmt.exe". On 64-bit systems there is also a 32-bit version of the binary located at "%windir%\SysWOW64\autofmt.exe". This binary is used for formatting a drive/partition when it is called from the "Windows Recovery Console". By the way, we can't start "autofmt.exe" directly from the command line or the run dialog[510] - as shown in the screenshots below.

Thus, overall "autofmt.exe" automates the file format during reboots[511]. Also, the binary is digitally signed by Microsoft. Based on the extracted strings for the binary we can learn about the capabilities of the utility. Among those are: the ability to support formatting for different filesystems (FAT/FAT32/extFAT/NTFS/UDF). By the way, we can also override the default allocation unit size and specify if we want to support short filenames or not.

Lastly, there is also the ability to specify the size for the NTFS log and for disabling the NTFS repair log (in this case the command "chkdsk /spotfix" won't work) - more on that in future writeups. Basically, we can think about "autofmt.exe" as a replacement of the old fdisk command[512].





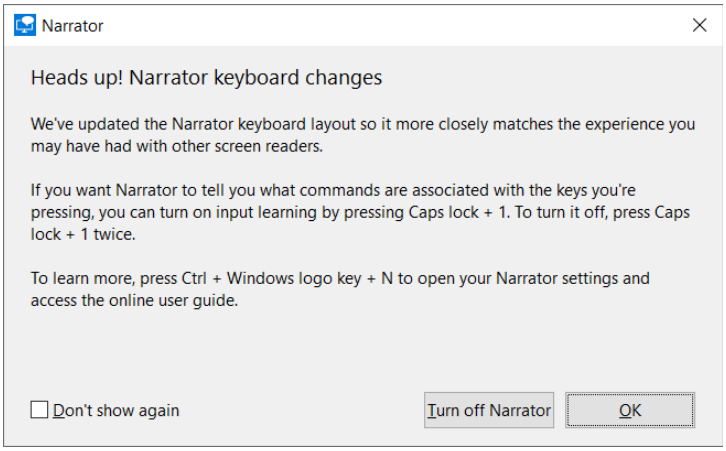
[510] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/autofmt
[511] https://www.cs.toronto.edu/~simon/howto/win2kcommands.html
[512] https://www.quora.com/How-do-you-partition-a-drive-in-Windows-10-using-a-command-prompt

Narrator.exe (Screen Reader)

“Narrator.exe” is a PE binary located at “%windir%\system32\Narrator.exe”. As opposed to other Windows utilities, on 64-bit versions of Windows there is no parallel 32-bit version of the binary. It is used as a screen-reader utility which is built in as part of the operating system[513].

Overall, with the narrator utility we can use our PC without a mouse in order to complete common tasks (very helpful for blind or low vision users). This is done by reading\interacting with on the screen components (such as text and buttons). Examples for use cases are reading\writing email, browsing the internet and working with documents[514].

Lastly, we can configure the narrator's pitch, volume, speaking rate and even install text-to-speech voices. Also, we can use the keyboard’s arrows or a braille display to navigate through the different UI components\text on screen[515] - as also described in the screenshot below.



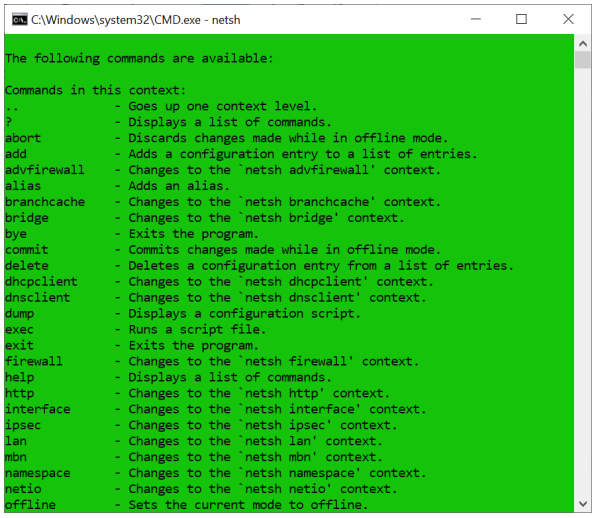
[513] https://social.cyware.com/news/attackers-abuse-narrator-utility-to-access-windows-systems-63b580fd
[514] https://support.microsoft.com/en-us/windows/chapter-1-introducing-narrator-7fe8fd72-541f-4536-7658-bfc37ddaf9c6
[515] https://www.tenforums.com/tutorials/88188-turn-off-narrator-windows-10-a.html

netsh.exe (Network Command Shell)

“netstat.exe” (Network Command Shell) is a PE binary located at “%windir%\system32\netsh.exe”. On 64-bit systems of Windows there is also a 32-bit version of the binary located at “%windir%\SysWOW64\netsh.exe”. It is a command line utility which is used for showing/modifying network configuration of the local/remote system. We can type the commands directly at the “netsh shell” or leverage it as part of a batch file/script[516].

Overall, among the different configuration realms that “netsh.exe” supports are: WLAN, firewall, IPSec, DNS client, DHCP client, RPC and more - as shown in the screenshot below. The “netsh” command provides similar functionality as the “Microsoft Management Console” snap-ins[517]. The way in which “netsh.exe” interacts with other operating systems components is by leveraging DLL files called “helpers”[518].

Lastly, we can check out a reference implementation of “netsh.exe” as part of ReactOS[519]. For a complete “netsh” command reference I suggest going over the Microsoft documentation[520].



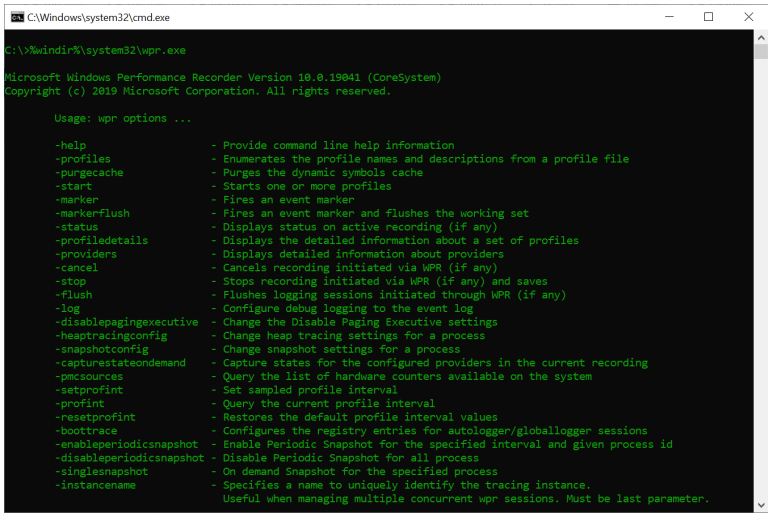
[516] https://learn.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts
[517] https://learn.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh
[518] https://lolbas-project.github.io/lolbas/Binaries/Netsh/
[519] https://github.com/reactos/reactos/tree/master/base/applications/network/netsh
[520] https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc754516(v=ws.10)

wpr.exe (Microsoft Windows Performance Recorder)

wpr.exe (Microsoft Windows Performance Recorder) is a PE binary located at "%windir%\system32\wpr.exe". As opposed to other Windows built-in utilities on 64-bit versions of the operating systems there is no parallel (32-bit) version of the binary[521]. It is used in order to record system events (extending ETW) which can be analyzed using the "Windows Performance Analyzer"[522] .

Overall, "wpr.exe" is a CLI (Command Line Interface) utility - as shown in the screenshot below. It has been shipped since "Windows 8.1". We can use it for recording events to a file or to a memory buffer and also control the detail level of logging (light vs verbose). By using WPR IT professionals can proactively identify and resolve performance issues[523].

Lastly, there is also "wprui.exe" (more on it in a future writeup) which is similar to "wpr.exe" (they are dependent on the same DLLs). However, the second has less features. WPR has recording profiles which are lists of providers that are used for recording performance data. Examples of built-in recording profiles are: "Heap Usage", "GPU Activity", "Handle Usage", "File I/O Activity", "Registry Activity", "Disk I/O Activity" and "Networking Activity"[524].



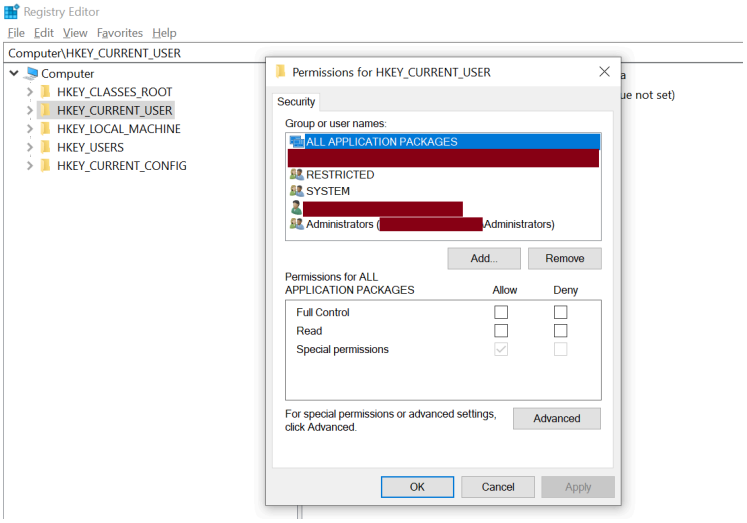
[521] https://learn.microsoft.com/en-us/windows-hardware/test/wpt/wpr-command-line-options
[522] https://ss64.com/nt/wpr.html
[523] https://learn.microsoft.com/en-us/windows-hardware/test/wpt/introduction-to-wpr
[524] https://learn.microsoft.com/en-us/windows-hardware/test/wpt/built-in-recording-profiles

regedit.exe (Registry Editor)

regedit.exe (Registry Editor) is a PE binary located at "%windir%\regedit.exe". On 64-bit versions of Windows there is also a 32-bit version located at "%windir%\SysWOW64\regedit.exe". Since Windows XP/Windows Server 2003 it is the replacement of "regedt32.exe". Thus, "regedit.exe" is called in case "regedt32.exe" is executed[525].

Overall, it is used the manage the "Registry" which is a hierarchical database used by the Windows operating system to store configuration, settings and even data in some cases[526]. Also, it can export\import\delete registry settings based on a "*.reg" file . Also, as opposed to "reg.exe" the "regedit.exe" binary normally requests permission elevation[527].

Lastly, it is important to know that although "regedit.exe" shows information such as the permissions of a hive\key\subkey it does not expose all the metadata such as last modification times[528]. For a reference implementation of "regedit.exe" we can checkout the source code of ReactOS[529].



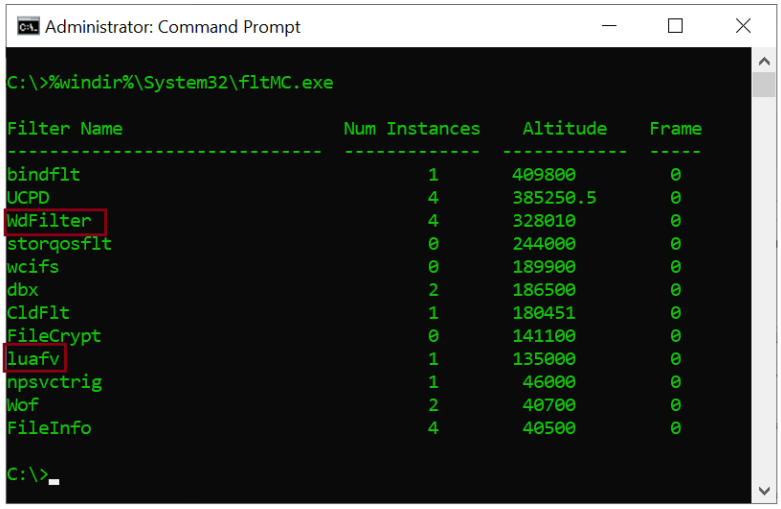
[525] https://medium.com/@boutnaru/the-windows-process-journey-regedt32-exe-registry-editor-utility-21a372f65615
[526] https://medium.com/@boutnaru/the-windows-concept-journey-registry-0767e79387a9
[527] https://ss64.com/nt/regedit.html
[528] https://en.wikipedia.org/wiki/Windows_Registry
[529] https://github.com/reactos/reactos/tree/master/base/applications/regedit

fltMC.exe (Filter Manager Control Program)

fltMC.exe (Filter Manager Control Program) is a PE binary located at "%windir%\system32\fltMC.exe". On 64-bit versions of Windows there is also a 32-bit version of the binary located at "%windir%\SysWOW64\fltMC.exe". It is used for managing "MiniFilter" drivers (which can add value to or modify the behavior of a file system) - more on that in future writeups. We can use it for: loading\unloading filter drivers, listing filter information, listing all the instances\associated instances with a filter\volume (including network ones) and attach\detach a filter from a volume[530].

Overall, "fltMC.exe" requires administrator privileges due to the fact it can unload drivers. This can also be leveraged for unloading drivers of security agents\products and thus bypassing them[531]. By the way, the binary is digitally signed by Microsoft. Also, the binary is dependent on "%windir%\system32\fltLib.dll" (Filter Library) which is itself signed by Microsoft.

Lastly, examples of such drivers are: "%windir%\system32\drivers\WdFilter.sys" (Microsoft Antimalware File System Filter Driver) which is part of "Windows Defender"[532] and "%windir%\system32\drivers\luafv.sys" (LUA File Virtualization Filter Driver) which is responsible for the "UAC File Virtualization"[533]- as shown in the screenshot below. For a reference implementation of "fltMC.exe" we can check out the source code of ReactOS[534].



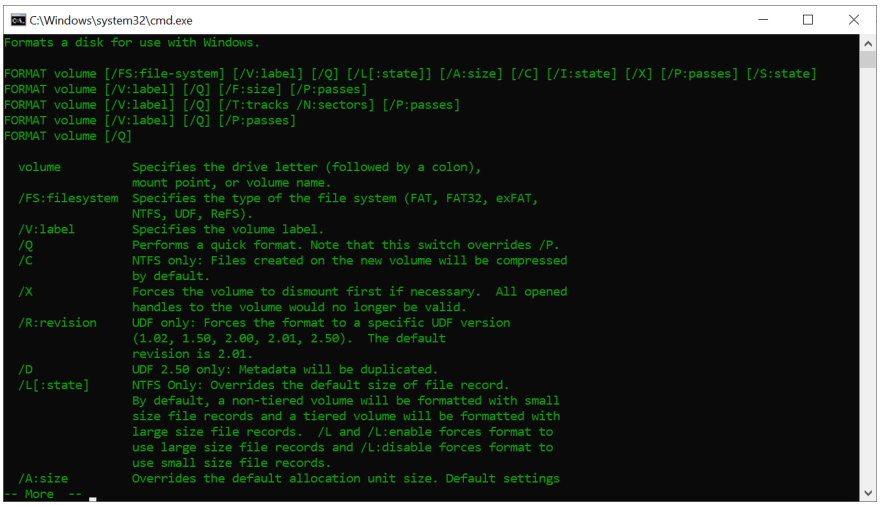
[530] https://ss64.com/nt/fltmc.html
[531] https://www.elastic.co/guide/en/security/current/potential-evasion-via-filter-manager.html
[532] https://medium.com/@boutnaru/the-windows-security-journey-windows-defender-antivirus-cf0c76a6802e
[533] https://medium.com/@boutnaru/the-windows-security-journey-file-virtualization-fdeb68e7e174
[534] https://github.com/reactos/reactos/tree/master/base/applications/fltmc

format.com (Disk Format Utility)

format.com (Disk Format Utility) is a PE binary (although it has a ".com" extension) located at "%windir%\system32\format.com". On 64-bit versions of Windows there is also a 32-bit version of the binary located at "%windir%\SysWOW64\format.com". It is used for preparing a volume (hard disk\usb stick\etc) for use by the operating system[535].

Overall, when formatting a volume (for usage under Windows) we can select the type of filesystem we want to use (FAT\FAT32\exFAT\NTFS\UDF\ReFS)[536]. By using "format.com" we create a new root directory and a file system for a drive and is supported only for local one (not over the network)[537].

Lastly, "format.com" is available on all versions of "MS-DOS" and since and since Windows 95. supports multiple options which can customize the format flow. Among those options are: setting the cluster size, zeroing the sectors, specifying the default allocation size and more[538] - as shown in the screenshot below. For a reference implementation of "format.com" we can checkout the source code of ReactOS[539].



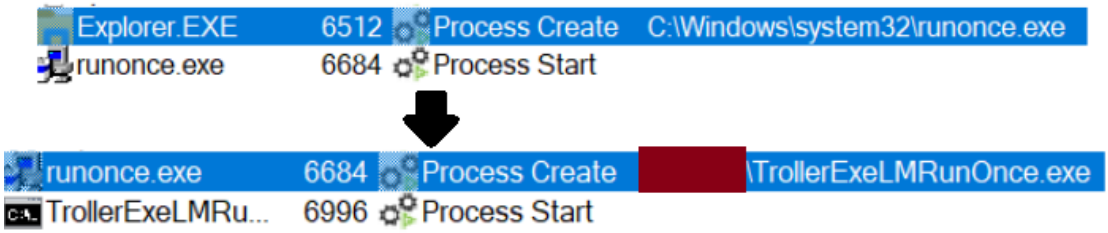
[535] https://docs.oracle.com/cd/E19683-01/817-2874/6migoia5c/index.html
[536] https://ss64.com/nt/format.html
[537] https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/format
[538] https://www.computerhope.com/formathl.htm
[539] https://github.com/reactos/reactos/tree/master/base/system/format

runonce.exe (Run Once Wrapper)

“runonce.exe” is an executable aka the “Run Once Wrapper” (based on the description field of the PE file), which is located at “%windir%\System32\runonce.exe”. On a 64 bit-system there is also a 32-bit version located at “%windir%\SysWOW64\runonce.exe”. Also, the file is digitally signed by Microsoft. It is used by applications as part of their installation process in order to ensure that post installation some additional programs (like for configuration) will execute only once[540]. By the way, the binary is digitally signed by Microsoft.

Overall, the “RunOnce” registry key allows code to execute once (due to the fact the configuration is removed) when a user signs in to the system[541]. It can be when the first user logs on to the system (“HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce”) or the first time a specific user logs on after setting the configuration (“HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce”). The HLKM one is relevant only when members of the administrator group log on after reboot[542].

Lastly, the way it works is that “explorer.exe”[543] starts “runonce.exe” which then executes the relevant applications/programs that are configured as part of the “RunOnce” registry key - as shown in the screenshot below (taken using Sysinternals' “Process Monitor”). For an implementation reference of “runonce.exe” I suggest going over the one in ReacOS[544].



[540] https://community.spiceworks.com/topic/1236047-runonce-exe
[541] https://learn.microsoft.com/en-us/windows/security/threat-protection/use-windows-event-forwarding-to-assist-in-intrusion-detection
[542] https://medium.com/@boutnaru/the-windows-concept-journey-runonce-registry-key-06eedd56f218
[543] https://medium.com/@boutnaru/the-windows-process-journey-explorer-exe-windows-explorer-9a96bc79e183
[544] https://github.com/reactos/reactos/tree/3fa57b8ff7fcee47b8e2ed869aecaf4515603f3f/base/system/runonce