Windows Processes: Section 1 Transcript

Introduction

Welcome to the Windows Processes module.

During this module, we'll introduce several terms and definitions related to how processes operate within the Windows environment. In addition, you'll be provided with opportunities to become familiar with various command line tools and the associated output of these tools.

Throughout this module, you'll be presented with opportunities to assess and apply what you've learned.

At the end of the module, you will be able to:

- Enumerate processes using (Windows) CLI tools,
- Enumerate processes using Sysinternal tools,
- Enumerate Dynamic Link Libraries using (Windows) CLI tool, and
- Explain the Portable Executable Format including the data structures and PE file's executable code

Bypass Exam Introduction

If you are already familiar with the subject matter presented in this module, you can choose to take a Bypass Exam to skip this module.

The Bypass Exam option provides a single opportunity to successfully demonstrate your competence with the material presented within the module. If you pass, you'll receive credit for completing the module, unlocking the content within, and you will be free to proceed to the next module. If you do not pass, you will need to successfully complete the module, including all exercises and the Module Exam, to receive credit.

Click the Next Section button to continue.

Windows Processes: Section 2 Transcript

What is a Process? 1/12

In computing, a program is defined as a sequence of instructions written to perform a specific task. This sequence of instructions is a passive entity until executed. Once executed, it causes the computer to behave in a predetermined manner. Although a program can be executed in various ways, once it has been launched, it is represented as a process.

A process represents a program in an active (executed) state. In situations where there are multiple instances of a program on a system, each instance of that executed program will be represented as a distinct process. Therefore, it is possible to have multiple processes associated with one program. Processes act as a container for a set of resources that includes the process code, the parameters of its private memory space, any file handles, and other resources that have been allocated to the process. A file handle is a temporary reference (typically a number) assigned by the operating system to a file that an application has asked it to open. The handle is used throughout the session to access the file.

Process Overview 2/12

A process is defined as a container for a set of thread resources. Within the Windows operating system, each process has its own assigned virtual address space. This address space is independent to the virtual address space assigned to other processes. Within this virtual address space, the processes' code and data is stored and protected from other processes. The processes' threads and allocated resources are also stored within this virtual address space.

A process includes the following components:

- One or more threads,
- One or more code segments,
- One or more data segments,
- Global variables,
- Environmental information,
- A process heap, and
- Other resources such as open handles and other heaps.

Process Binaries 3/12

A binary file is a computer file that a computer reads or interprets. Although a binary file may contain formatted text, it is not a text file, and is not considered human-readable. The file is a sequence of bytes that are intended to be interpreted as something other than text when read by the computer.

Process Memory 4/12

Process memory is the virtual address space allocated to a process at the time the process is

created. This virtual address space is the set of virtual addresses the process can use. This address space is private and cannot be accessed by other processes unless it is shared.

The total amount of physical memory supported by Windows ranges from 2 GB to 2 TB. A process' virtual address space may be greater than or less than the total physical memory available to the system. The physical memory and the virtual address space for each process is organized into units of memory called pages. Because a system's physical memory is shared by all running processes and the functions of the Windows kernel, paging is implemented in order to move page files to disk when they are not in use.

A process' virtual address space consists of the several categories. Click each category to learn about them.

Working set:
The working set is the amount of memory physically mapped to a process at a given time.

Paged pool:
Memory in the paged pool can be transferred to the paging file on the disk, referred to as paged out, when it is not being used.

Non-paged pool:
Memory in the non-paged pool cannot be transferred to the paging file on the disk as long as its corresponding objects are allocated.

Pagefile:
The pagefile monitors memory usage and how much memory is set aside for the process in the system paging file.

Inter-Process Communications (IPCs) 5/12

Inter-Process Communications (IPCs) are the Windows operating system mechanisms that allow applications and processes to communicate and exchange data. From the perspective of the application, IPCs are typically categorized as either clients or servers.

A client is an application or a process that requests a service from another application or process.

A server is an application or process that responds to requests made by clients.

Click the IPC mechanisms to learn about them. Note that this list is not an all-inclusive list of IPC mechanisms.

Pipes:
Windows implements two types of pipes for two-way communications: anonymous pipes and named pipes.

Anonymous pipes are used to transfer information between related processes. An example of this is redirecting the standard input or output of a child process, so that it communicates with its parent process. Anonymous pipes also only support communication in a single direction. To enable the exchange of data in both directions, you must create two anonymous pipes. Anonymous pipes are

only implemented within the local computer.

Named pipes are similar to TCP in that they provide reliable, connection-oriented, duplex communication paths to transfer data between unrelated processes or processes on different computers. A named-pipe server process creates the named pipe with a well-known name. A named-pipe client process that knows the well-known name can connect to the open end of the named pipe. Once this connection is established, data can be exchanged between the client and server processes.

Anonymous pipes provide an efficient way to redirect standard input or output to child processes on the same computer, and named pipes provide a simple programming interface for transferring data between two processes, whether they reside on the same computer or over a network.

Mailslots:
A mailslot is similar to UDP in that they provide an unreliable, connectionless, one way communication path. A mailslot is created by a mailslot server. Mailslot clients send messages to the mailslot server by writing messages to its mailslot. Because a mailslot is actually a pseudo-file, incoming messages are appended to it. Messages are saved in the mailslot until the mailslot server has read them. The handle for the mailslot is required to read the messages in the mailslot. This handle can be delegated to another application.

RPC:
A Remote Procedure Call, or RPC, allows applications to call functions remotely. In this mechanism, the term remotely refers specifically to the application, and not the computer. The remote function being called by the application executes on the local or a remote computer, but external to the calling application.

Windows sockets:
A Windows socket is a protocol-dependent interface that takes advantage of the communication capabilities of other underlying protocols, and is capable of supporting current and emerging networking capabilities. An application that uses sockets can communicate with other socket implementations on other systems.

Clipboard:
When performing copy and paste operations between applications, the clipboard acts as a central repository for the data to be shared. Copied or cut data in an application is placed on the clipboard. Once this data is placed on the clipboard, any other application can retrieve it.

Data Execution Prevention (DEP) 6/12

Data Execution Prevention (DEP) is a security feature that is implemented through a set of hardware and software technologies that prevent malicious code from executing on a computer system. It is implemented by marking memory pages and other areas of memory as either executable or non-executable. These memory locations and pages typically contain data structures such as default heap pages, various stack pages, and memory pool pages. Code does not typically execute from these locations. Code cannot execute from memory that has been marked as non-executable.

The main benefit of DEP is that it helps prevent the execution of code from affecting protected areas

of memory. DEP is most effective against malicious software that takes advantage of buffer overflows or has injected a process with additional code, and attempts to run that code. The execution of this code from protected areas of memory causes an exception to occur. If the exception is unhandled, the process is stopped or blocked.

DEP is only enabled by default for core operating system binaries and applications that are explicitly configured to use it.

Address Space Layout Randomization (ASLR)

In older versions of Windows, core processes were loaded into memory in predictable locations when the system started up. Because of this, it was easy to exploit the system by targeting the specific memory locations used by specific processes.

Address Space Layout Randomization (ASLR) was introduced with Windows Vista. ASLR randomizes the memory locations used by the **system processes** and other applications. This forces the attacker to guess the correct memory locations used by any given process, making it much harder to successfully exploit the operating system. ASLR is very effective against buffer overflow and many zero-day attacks.

Like DEP, ASLR is only enabled by default for core operating system binaries and applications that are explicitly configured to use it.

32-bit Applications on a 64-bit System

Windows 7 is primarily a 64-bit system. At the time of its initial release, a 32-bit version was available, but the proliferation of 64-bit systems made it obsolete. However, not all of the commonly used Windows applications have been recoded to 64-bit versions. This presents the necessity for 64-bit Windows to be able to run 32-bit applications.

To accomplish this, Windows stores 32-bit applications separately from 64-bit applications. Additionally, it stores both a 32-bit and 64-bit version of each DLL separately on the system. 64-bit applications are stored in the Programs Files folder, and 64-bit DLLs are stored in the System32 folder. 32-bit applications are stored in the Programs Files (x86) folder, and 32-bit DLLs are stored in the SysWOW64 folder.

When a 32-bit application is launched and calls for DLLs to be loaded, the file system redirector intercepts the requests to load the DLLs. It recognizes the application as a 32-bit application and redirects the request to load the DLLs from the SysWOW64 folder, instead of loading the DLLs from the System32 folder.

Registry Redirection

The registry redirector performs registry redirection for all registry calls made by a 32-bit application. In much the same manner as the system separately stores all 32-bit and 64-bit applications and DLLs, the system also stores separate 32-bit and 64-bit logical views of the registry. When a 32-bit application makes calls to the registry, the registry redirector intercepts these calls and redirects them to the mapped physical location under the Wow6432Node subkey, i.e., a registry call to

HKLM\Software is redirected to HKLM\Software\Wow6432Node.

Process Enumeration Tools

Command-line tools (CLIs) view and manage processes and process information. Click the various on-screen CLIs to learn about them.

Pslist:
PsList is part of the Sysinternals tool suite that provides a listing of the running processes on a system. Its output includes details about each running process and can be used locally or against a remote computer. This tool can run on either a 32- or 64-bit system.

Tasklist:
Tasklist is a Windows native utility built into the command shell. It provides a listing of the system's currently running processes. This tool can run on either a 32- or 64-bit system.

Pskill:
Also part of the Sysinternals tool suite, pskill allows process termination. This tool can be run on either a 32- or 64-bit system.

Taskkill:
Taskkill, similar to tasklist, is also a Windows native tool built into the command shell. It allows process termination. This tool can run on either a 32- or 64-bit system.

Pssuspend:
Also part of the Sysinternals tool suite, pssuspend provides the capability to suspend a running process or resume a suspended process. This tool can run on either a 32- or 64-bit system.

Handle:
This tool is part of the Sysinternals tool suite. When run, handle provides a list of all the open file references on the system for each running process. This tool can run on either a 32- or 64-bit system.

Listdlls:
Another tool that is part of the Sysinternals tool suite. It provides a listing of all DLLs that are loaded by the running processes. This tool can run on either a 32- or 64-bit system.

Pmon
Process Resource Monitor (Pmon) is part of Windows Resource kit. Pmon is a command-line tool that displays several measures of the CPU and memory use of processes running on the system. Pmon only works on 32-bit systems.

Integrity Levels

The component store uses NTFS hard links between itself and other Windows directories to increase the robustness of the Windows platform.

Click each Features of Integrity Levels listed to learn more about them.

Integrity Levels

- Extends security by restricting access between objects running under same account.
- Enforces security using integrity levels (MAC - Mandatory Access Control).
- A lower-level subject cannot modify a higher-level object.
- Uses Access Control Lists to maintain integrity levels.
  The different Integrity Levels include:
  - Untrusted
  - Low
  - Medium
  - High
  - System

Major parts to the mechanism:

- Predefined integrity levels and their representation.
- Integrity policies that restrict access permissions.
- Integrity level assigned to the security access token.
- Mandatory label access control entry.
- Mandatory labels assigned to objects.
- Integrity restrictions within the AccessCheck and kernel-mode SeAccessCheck APIs.

Mandatory access token policies

- No write up - The default policy that is assigned to all access tokens. The policy restricts write access by this subject to any object at a higher integrity level.
- New process min - Controls the behavior of assigning the integrity level to child processes. Normally, a child process inherits the integrity level of the parent process when the parent process access token is assigned to the child. With the NEW_PROCESS_MIN policy, the integrity level of the child process will be the minimum integrity level of either the parent access token, or the object integrity level of the executable file for the new process. This policy is set by default in all access tokens.

Mandatory label policies

- No write up - the same as the access token policies.

Virtualization Based Security

- Introduced in Windows 10 and Server 2016, it is a system protection mechanism that uses a "Virtual Secure Mode", a process by which the Windows hypervisor is used to create and isolate a secure region of memory.
- Provides an added layer of system security by enforcing restrictions that protect critical system resources and other local assets.
- Because of its ability to create and isolate sections of memory, helps mitigate and contain malware even if it has gained access to the OS kernel.

Exercise Introduction

It is time for an Exercise. In order to successfully complete the Exercise, you are expected to use your notes and any available resources presented throughout the course, in addition to conducting your own Internet research.

Windows Processes: Section 3 Transcript

Threads, Heaps, and Stacks

Threads, heaps, and stacks are three important concepts.

A thread is the basic unit of CPU utilization; the smallest sequence of programmed instructions that can be managed independently. It is the component of a process that is actually scheduled for execution on a processor. It normally consists of a program counter, a stack, a set of registers, and a thread ID.

A heap is best described as an area of pre-reserved memory that a program or process uses to store data in varying amounts. These amounts will not be known until runtime, when the program or process is actually running.

For example, a program may accept data as input from other sources, and then process this data all at once. The heap storage is used to store this data. When the process is created, it is allocated a certain amount of heap storage, from which the process manages its allocated heap by requesting a chunk, called a heap block, of storage every time it's needed. The block is returned to the heap for reallocation when it is no longer needed.

In computing, a stack is a data structure used to store a collection of objects. This data structure is a LIFO (Last-In, First-Out) abstract data type. There are two principle operations that can be performed on the objects in the stack: the push operation adds an object to the top of the stack, and the pop operation removes an object from the top of the stack.

Security Context and Impersonation

The context of a process describes the resources and environment of a process. The security context of a process describes the privileges or permissions associated with a process. The threads of a process execute in the security context of its parent process.

Impersonation is the ability of a thread to execute in a security context that is different from that of its parent process. Impersonation is generally implemented to meet the security requirements of client/server applications.

In client/server applications, a service running in the security context of the client application represents the client application to the server application. The service accomplishes this by creating a worker thread and associating it with the client's security token. This worker thread is now impersonating the client application when requesting a service from the server application.

Thread Scheduling

Windows implements a priority-driven, preemptive scheduling system. This means that at least one of the highest priority threads, that are ready to be dispatched to a processor, will be executing

(running) on a processor.

Within Windows, there is no single scheduler module, mechanism, or routine. The code for performing this task exists throughout the kernel where scheduling-related events occur. These routines are collectively known as the kernel dispatcher.

When a thread has been scheduled to run, it executes for a predetermined amount of time, called a quantum. A quantum is the length of time a thread is allowed to run on a processor before being preempted, to allow another thread of the same priority to run.

Because Windows implemented a preemptive scheduling system, a thread may not be allowed to run to the end of its quantum because another thread of higher priority becomes ready to execute. In this case, the currently running thread is preempted, removed from the processor, and the higher priority thread is dispatched to the available processor to run.

Process and Thread Priorities

Windows uses priorities to determine the order the threads will run. Threads with higher priorities will be dispatched to a processor and allowed to run before other lower priority threads.

Windows uses 32 priority levels, ranging from the lowest level of 0, to the highest level of 31. There are 16 real-time levels (16 through 31) and 16 variable levels (0 through 15). Level 0 is reserved for the zero page thread.

Thread scheduling priorities are assigned from two different perspectives: those of the Windows Application Programming Interface (API) and those of the Windows kernel.

Each process belongs to a priority class. The Windows API first organizes the threads according to the priority class to which its parent process is assigned at the time the process is created:

- REALTIME_PRIORITY_CLASS
- HIGH_PRIORITY_CLASS
- ABOVE_NORMAL_PRIORITY_CLASS
- NORMAL_PRIORITY_CLASS
- BELOW_NORMAL_PRIORITY_CLASS
- IDLE_PRIORITY_CLASS

Within those processes, the relative priority of the individual thread is assigned:

- THREAD_PRIORITY_TIME_CRITICAL
- THREAD_PRIORITY_HIGHEST
- THREAD_PRIORITY_ABOVE_NORMAL
- THREAD_PRIORITY_NORMAL
- THREAD_PRIORITY_BELOW_NORMAL
- THREAD_PRIORITY_LOWEST
- THREAD_PRIORITY_IDLE

The process priority class and the thread priority level are combined to form the base priority of each thread.

Windows Side By Side (SxS)

The Windows component store (C:\Windows\winsxs) directory is used during servicing operations within Windows installations. Servicing operations include, but are not limited to, Windows Update, service pack, and hotfix installations. The component store contains all the files that are required for a Windows installation. And, any updates to those files are also held within the component store as the updates are installed. This causes the component store to grow over time as more updates, features, or roles are added to the installation.

Dynamic-Link Library (DLL)

A Dynamic-Link Library, or DLL, is a library of functions that can be used by an application. Much of the functionality in the Windows operating system is provided through the implementation of DLLs. Each DLL normally represents a specific function; an example of this would be the DLL to execute the Remote Registry function. DLLs are unique in that the code contained within a DLL can be used by more than one program at the same time. An additional benefit of using DLLs is that they allow a program to be modularized, making it easier to apply updates.

The Windows API is implemented as a set of DLLs. Any program that uses the Windows API, uses DLLs.

DLLs come in a variety of extensions. Three of the more common extensions are: .dll, .ocx, and .drv. Click each extension to learn more about them.

DLL:
DLL stands for Dynamic-Link Library. DLLs are binary files that are shared among running processes, allowing programs to interact with the operating system. Each DLL is written to perform a specific function, such as the LoadResource function used to load a resource into main memory.

OCX:
OCX stands for Object Linking and Embedding (OLE) Control Extension, but is now commonly referred to as Active X Controls. This type of DLL is used for interface behaviors that are triggered by users of programs; such as scroll bar movements and windows resizing.

DRV:
This extension is associated with legacy device driver files. A device driver is the software that interacts with a specific device or other specific software. Device driver files have specific knowledge about its associated device or the device's software interface that enable other programs to communicate with the device.

DLLs

Some advantages of DLLs are that they use fewer resources and reduce code duplication on the disk and within physical memory by sharing its code between more than one process.

A major disadvantage with using DLLs is that it is vulnerable to DLL hijacking, also known as a DLL preloading attack or a binary planting attack.

DLL hijacking is an attack that exploits the way some Windows applications locate and load DLLs. This type of attack is used against programs that do not use the fully qualified or absolute path when loading the DLL.

DLL hijacking occurs when programs load and execute a malicious DLL that has been preloaded onto the system.

In the example shown, an attacker injects a malicious DLL on a remote system. The victim opens an application that loads the malicious DLL. The malicious DLL enables the attacker to gain local admin rights on the target system.

Windows Defender Application Control

Windows Defender Application Control (WDAC), formally known as Device Guard, is a set of features that work together to protect the system by preventing untrusted and malicious code from executing.

Device Guard consists of three primary components: The Configurable Code Integrity (CCI), the VSM Protected Code Integrity, and the Platform and UEFI Secure Boot.

Windows Defender Application Control

Discussing the Device Guard components in a bit more detail:

The CCI enforces policies that require software to be signed and trusted before executing.

The VSM Protected Code Integrity moves the Kernel Mode Code Integrity (KCMI) and Hypervisor Code Integrity (HVCI) security solutions into a protected area to guard them against attacks associated with malware and malicious code.

The Platform and UEFI Secure Boot, as discussed earlier in this course, helps defend the system against known vulnerabilities that are associated with the boot up process.

Portable Executable (PE) Files

Portable Executable (PE) is a file format for executables, object codes, DLLs, and font files that are used in variants of the Windows operating system. The PE format is a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code.

It is important to know that for a PE file, the executable file on disk is very similar to what the module will look like after Windows has loaded it into system memory. Essentially, it is brought into place in one piece, or to use an analogy, it is pre-fabricated.

This reduces the amount of work the loader program needs to perform, in order to create a process from the file on disk.

Next, let's look at the headers and file sections of the PE format.

Within the header of a PE file, the following data structures are found: DOS header, PE header, optional header, data directories, and section table. Click each data structure to learn about them.

DOS Header:
The DOS header is the first component in the PE file format and occupies the first 64 bytes of the file. It is there in case the program is run from DOS, to allow DOS to identify the file as a valid executable. When this happens, the DOS stub, stored immediately after the header, is executed. The DOS stub normally prints out the error message, "This program cannot be run in MS-DOS mode."

The DOS Header contains a signature and the offset to the PE header.

PE Header:
The PE header defines what the rest of the file looks like. It contains essential information used by the loader. You will notice that this main header isn't at the very beginning of the file; instead, it is located a few hundred bytes into the file following the DOS Header.

The PE header contains a signature field, processor information and type, the number of sections, the relative offset of the section table, and the characteristics of the file, such as the file type or extension.

Optional Header:
The optional header contains the most meaningful information about the executable image, such as the initial stack size, program entry point location, preferred base address, operating system version, section assignment information, etc.

Data Directories:
The data directory is the final 128 bytes of the Optional Header. It is basically a data array of 16 IMAGE_DATA_DIRECTORY structures. Each data directory specifies the size and relative virtual memory address relating to an important data structure in the PE file. The following is a listing of the data directory structures:

- IMAGE_DIRECTORY_ENTRY_EXPORT
- IMAGE_DIRECTORY_ENTRY_IMPORT
- IMAGE_DIRECTORY_ENTRY_RESOURCE
- IMAGE_DIRECTORY_ENTRY_EXCEPTION
- IMAGE_DIRECTORY_ENTRY_EXPORT
- IMAGE_DIRECTORY_ENTRY_SECURITY
- IMAGE_DIRECTORY_ENTRY_BASERELOC
- IMAGE_DIRECTORY_ENTRY_DEBUG
- IMAGE_DIRECTORY_ENTRY_COPYRIGHT
- IMAGE_DIRECTORY_ENTRY_SECURITY
- IMAGE_DIRECTORY_ENTRY_GLOBALPTR
- IMAGE_DIRECTORY_ENTRY_TLS
- IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG
- IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT
- IMAGE_DIRECTORY_ENTRY_IAT

- IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT
- IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR
- IMAGE_DIRECTORY_ENTRY_ENTRIES

Section Table:
The section table is an array of data structures. Each data structure contains information about one section in the PE file. This includes the section name, virtual size, virtual address size of raw data, pointer to raw data, and characteristics (whether the section contains executable code, initialized or uninitialized data, or whether it can be written to or read from). The sections follow the ordering presented in the section table.

PE File Section

Following the header section of a PE file is the PE file section. Several of the pre-defined data sections typical to an application or DLL are: .text, .bss, .rdata, .data, .rsrc, .edata, .idata, and imports/exports.

Click each data section to learn what they do.

.text:
.text is the file's executable code. The code resides in a single section. Having one large code section makes it easier for the operating system and the application developers to manage.

.bss:
.bss represents uninitialized data used by the applications. It includes the variables declared as static within a function or the source module.

.rdata:
.rdata represents read-only data, such as the constants, literal strings, and debug directory information.

.data:
.data is used to store all other variables, to include the application and module global variables.

.rsrc:
.rsrc is used to store resource information for a module.

.edata:
.edata contains the application or DLL's export directory, which contains information about the names and addresses of exported functions.

.idata:
.idata contains the information about various imported functions including the Import Directory and Import Address Table.

Imports/Exports:
Imports/exports contain information on the functions that have been imported and exported by the executable from DLLs.

Exercise Introduction

It is time for an Exercise. In order to successfully complete the Exercise, you are expected to use your notes and any available resources presented throughout the course, in addition to conducting your own Internet research.

Windows Processes: Section 4 Transcript

Common System Processes

System processes are common in variants of the OS. The processes are present across multiple instances of the OS regardless of the computer's role or set of services it offers to the network. The system processes common to Windows are listed. Click each system process to learn about them.

Idle:
Though not a real process, its purpose is to count the number of cycles where the processor(s) are not doing anything.

System:
This process represents the kernel process (ntoskrnl.exe).

smss:
The smss process creates and manages the various sessions on the system as well as finalizing system initialization and starting wininit.exe and winlogon.exe.

csrss:
This is the user-mode portion of the Windows subsystem.

wininit:
This is the session 0 initialization process. This process runs in session 0 (OS's session) only. It starts any auto-start devices and drivers applicable to session 0 and launches lsass.exe.

winlogon:
This is the Interactive Logon Manager. An instance of it runs in all sessions above 0. Its purpose is to coordinate all logon activities and security-related functions within its session, and to launch services.exe.

services:
The SCM is responsible for starting all auto-start devices and drivers applicable to its session.

lsass:
The lsass runs in session 0. It performs all logon functions to include user authentication, verifying the user's access and creating the user's access token.

lsm:
The lsm also runs in session 0 only. Its purpose is to manage all connections related to the terminal server and monitor all other sessions. If it detects an unstable state in any of the system's sessions above session 0, it signals smss to terminate the session.

Svchost.exe

Svchost.exe is a process on your computer that hosts or contains other individual services that

Windows uses to perform various functions. For example, Windows Defender uses a service that is hosted by an svchost.exe process.

There can be multiple instances of svchost.exe running on your computer, with each instance containing different services. One instance of svchost.exe might host a single service for a program, and another instance might host several services related to Windows.

Each can contain a group of services Groups (in this registry location = HKLM\Software\Microsoft \Windows NT\CurrentVersion\Svchost). You can use Task Manager to view which services are running under each instance of svchost.exe.

Server-Mode Process

The term server-mode process refers to a process that provides a service to the local computer or to the network. Server-mode processes can be used to determine:

- The role of the computer,
- If the computer is a server or a workstation, and
- What services it is providing.

These processes can be all sorts of services: file replication services (ntfrs.exe) on Domain Controllers, domain name server (DNS) services (dns.exe), IIS services (inetinfo.exe), print spooler services (spoolsv.exe), etc. These processes run regardless of whether a user is logged on. Examples of server-mode processes are: DNS, MSDTC (Microsoft Distributed Transaction Coordinator), IDS (Intrusion Detection Software), and Endpoint Security Products.

User-Mode Processes

User-mode processes are those that are directly related to the user logged on to the computer. They represent applications and services that are launched either by the user or as a result of the user logging on. Most of these processes are applications that normally only run when the user is logged on. If the user logs out, these processes are closed as part of logging the user off the system. A couple examples of user-mode processes are:

- Explorer, which is a process that represents the user shell. It is used in determining whether or not a user is logged on when performing a system characterization, which is discussed later in this module. And,
- Application processes, which represent the programs that are launched either by the user or by the OS, as part of the user logging onto the system.

Protected Mode Internet Explorer (PMIE)

Protected Mode Internet Explorer (PMIE), which applies to Internet Explorer 7+ in Windows Vista and later, is a feature that makes it much more difficult to install malicious software or destroy data by significantly restricting user privileges. Protected mode is turned on by default via the Internet, intranet, and restricted sites, and warns users when the webpage is trying to install, or run software on their system. PMIE:

- Works with IE7+ in Vista+,
- Runs in Low integrity level,
- If compromised, should limit damage, and
- Cannot affect/create any object at higher level.

Section Completed

Windows Processes: Section 5 Transcript

Summary

You have completed the Windows Processes module.

During this module, we discussed several terms and definitions related to how processes operate within the Windows environment. Then we focused on applying what was learned.

You should now be able to:

- Enumerate processes using (Windows) CLI tools,
- Enumerate processes using Sysinternal tools,
- Enumerate Dynamic Link Libraries using (Windows) CLI tool, and
- Explain the Portable Executable Format including the data structures and PE file's executable code

To receive credit and advance to the next module, you must achieve a passing score on the Module Exam. Click the Next Section button to begin the Module Exam.