SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

P O S T E R

digital-forensics.sans.org

$25.00
DFIR-FindEvil-v2-1-16

FOR408 **Windows Forensics** GCFE

FOR518 **Mac Forensics**

FOR526 **Memory Forensics In-Depth**

FOR585 **Advanced Smartphone Forensics** GASF

SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

OPERATING SYSTEM & DEVICE IN-DEPTH

INCIDENT RESPONSE & ADVERSARY HUNTING

FOR508 **Advanced Incident Response** GCFA

FOR572 **Advanced Network Forensics and Analysis** GNFA

FOR578 **Cyber Threat Intelligence**

FOR610 **REM: Malware Analysis** GREM

SEC504 **Hacker Tools, Techniques, Exploits, and Incident Handling** GCIH

MGT535 **Incident Response Team Management**

@sansforensics | sansforensics | dfir.to/DFIRLinkedInCommunity | dfir.to/gplus-sansforensics | dfir.to/MAIL-LIST

Know Abnormal...Find Evil

Memory Artifacts

Rogue Processes

Malware authors generally pick one of two strategies for obscuring their malicious processes: hide in plain sight and attempt to appear legitimate, or use code injection and/or rootkit methods to hide from the view of normal analysis tools. See below for more on code injection and rootkits.

When searching for malware attempting to hide in plain sight, look for process names that appear legitimate but originate from the wrong directory path or with the wrong parent process or SID. Look for misspellings like **scvhost.exe** or **lssass.exe** and check for unusual command-line arguments. See the opposite side of this poster for legitimate Windows process details.

Besides processes, also look for suspicious DLLs executed through **rundll32.exe**, implemented as services with **svchost.exe**, or injected into legitimate processes.

Checking for signed code can help reveal suspicious executables. While there have been and will continue to be signed malware, you can typically only trust using a certificate from a trusted CA. For example, on a default installation of Windows 7 Enterprise, all running processes, device drivers, services, and scheduled tasks are signed by Microsoft. For live response memory analysis, Mandiant's **Redline** will check on-disk signatures for running code. For offline analysis, Didier Stevens' Authenticode Tools or Sysinternal's sigcheck.exe provide a tremendous amount of information about a file's digital signature.

Code Injection and Rootkit Behavior

Code injection and rootkits provide stealth to malware by hiding it from normal analysis techniques. Fortunately, memory analysis provides an effective mechanism for detecting both of these behaviors.

Typical code injection techniques provide an effective way to hide code without relying upon low-level programming knowledge, thus making it very popular among malware authors. Code injection is almost never legitimate, with the one exception of software debugging. Therefore, finding evidence of code injection on a standard system is almost always worth looking into further.

A rootkit is a broad term for describing ways of subverting the operating system with the intent to hide activities and data. There are a number of techniques for doing this, but the end result is stealthy malware that is often undetectable by security tools running on the system. That said, there are a few rootkit detection tools available, such as **GMER** and **Rootkit Revealer**, that can compare the state of the system as determined by the OS versus the state determined by the tool. When there are differences, it is often an indication of rootkit behavior.

The most effective technique for detecting rootkits is via memory forensics, since offline memory analysis does not rely on the compromised OS. For example, memory forensics can identify running processes even if they are unlinked by a rootkit. It can also help locate suspicious function hooks, which are essentially redirects to malicious code. Fortunately, rootkits are relatively rare due to the skill required to create a reliable exploit across the various Windows versions.

Memory analysis tools like Mandiant Redline and Volatility provide robust features for finding code injection and rootkit behaviors.

Suspicious Network Activity

Many core processes in Windows utilize the network, including **svchost.exe**, **lsass.exe**, and even the **System** process. Since you can't rule out the possibility of legitimate network activity from these processes, you need an effective way to identify illegitimate network activity. With memory analysis, one can parse through existing and even residual connections and sockets established by the system. When you are just starting to try to identify unusual network behavior, keep an eye out for the following:

- Any process communicating over port 80, 443, or 8080 that is not a browser
- Any browser not communicating over port 80, 443, or 8080
- Connections to unexplained internal or external IP addresses. For example, why did a process have a TCP connection to a system in Moldova?
- Web requests directly to an IP address rather than a domain name
- RDP connections (port 3389), particularly if originating from odd IP addresses. External RDP connections are typically routed through a VPN concentrator.
- DNS requests for unusual domain names

In an intrusion case, spotting the difference between abnormal and normal is often the difference between success and failure. Your mission is to quickly identify suspicious artifacts in order to verify potential intrusions. Use the information below as a reference for locating anomalies that could reveal the actions of an attacker.

Unusual Windows Behavior:

- Rogue Processes
- Unknown Services
- Code Injection and Rootkit Behavior
- Unusual OS Artifacts
- Suspicious Network Activity
- Evidence of Persistence

Poster References

- *Windows Internals*, 6th Edition, Parts 1 & 2
- *Rootkit Arsenal*, 2nd Edition
- *Windows Sysinternals Administrator's Reference*
- And the following SANS courses:
  - Securing Windows (SEC505)
  - Advanced Incident Response (FOR508)
  - Memory Forensics (FOR526)
  - REM: Malware Analysis (FOR610)

sansforensics | digital-forensics.sans.org/blog | @sansforensics | http://gplus.to/sansforensics

OS Artifacts

Unknown Services

Windows services are designed to run applications in the background without user interaction. Many services are required at system boot, including the DHCP Client, Windows Event Log, Server, and Workstation services. These services provide critical functionality for the OS and must be started immediately without requiring user input.

Services can be implemented as standalone executables or loaded as DLLs. In order to conserve resources, many service DLLs are grouped together and run under a smaller set of **svchost.exe** instances. **svchost.exe** is a Windows generic service host process, and it is typical to see several running instances of **svchost.exe** (5 or more is common).

Service configurations, as well as device driver configurations, are stored in the registry under **HKLM\SYSTEM\CurrentControlSet\Services**. The keys here provide the parameters for each service, including the service name, display name, path to the service's executable image file, the start type, required privileges, dependencies, and more. Each service has a start type configured to start at boot, by manual intervention, or on trigger events such as obtaining an IP address or hardware device connections. Windows services provide great flexibility to developers, and similarly malware authors, for automatically running code on a Windows host.

For offline analysis, investigate service configurations within the registry. On live or remote systems, use the built-in "**sc**" command to query installed services. Try parameters such as "**queryex**", "**qc**", "**qprivs**", and "**qtriggerinfo**" to get detailed information on service configurations.

Unusual OS Artifacts

Malware does not need to be present on a system for it to be compromised. We need to also look for unusual OS-based artifacts that would not exist on a typical workstation in the organization. When looking for program execution, focus on prefetch, shimcache, userassist registry keys, and even jump lists. Many of these artifacts can result from an adversary using your system but not implanting malware. Look for evidence showing odd behavior such as tools being run outside the scope of non-technical or normal user activity:

- **cmd.exe** execution – Provides command-line access
- **rar.exe** execution or presence of .rar files – Difficult to crack archiving tool for data exfiltration
- **at.exe** or **schtasks.exe** execution – Used for privilege escalation and persistence
- Existence of Sysinternals tools such as **PsExec**, **PsLoggedOn**, and **ProcDump** – Provide remote execution, interactive logon enumeration, and dumping of credentials within **lsass.exe** address space respectively
- **wmic.exe**, **powershell.exe**, or **winrm.vbs** execution – Used for remote execution
- **net.exe** execution – Used for mapping drives for lateral movement and enumerating groups like "Domain Admins"
- **reg.exe** or **sc.exe** execution – Add persistence such as Run keys or services
- **MountPoints2** registry key – Records shares on remote systems such C$, Temp$, etc.
- **.job** files in **C:\Windows\Tasks** – Related to odd application executions

Evidence of Persistence

Malware commonly accomplishes persistence using a variety of techniques. The most often used capability to achieve persistence with elevated rights is through scheduled tasks using the "**at**" command. With elevated rights, an adversary can create a service to automatically load malware or replace an existing service with a new malicious executable. The next most common malware persistence mechanism is using the registry auto-start mechanisms to load malware at boot or during user logon. Some of the latest techniques include DLL Search Order Hijacking and using local group policy to run scripts at logon/logoff. Finally, malware can also be installed as a Microsoft Office Add-in. When MS Word starts, the malware is executed.

- Scheduled Tasks
- Service Replacement
- Service Creation
- Auto-Start Registry Keys
- DLL Search Order Hijacking
- Trojaned Legitimate System Libraries
- More Advanced – PowerShell background job, Local Group Policy, MS Office Add-In, or BIOS Flashing

This poster was created by SANS instructors Mike Pilkington and Rob Lee.

Know Normal...Find Evil

SANS DFIR — DIGITAL FORENSICS & INCIDENT RESPONSE

sansforensics | digital-forensics.sans.org/blog | @sansforensics | http://gplus.to/sansforensics

Knowing what's normal on a Windows host helps cut through the noise to quickly locate potential malware.
Use the information below as a reference to know what's normal in Windows and to focus your attention on the outliers.

System

Image Path: N/A – Not generated from an executable image

Parent Process: None

Number of Instances: One

User Account: Local System

Start Time: At boot time

Description: The **System** process is responsible for most kernel-mode threads. Modules run under **System** are primarily drivers (.sys files), but also several important DLLs as well as the kernel executable, **ntoskrnl.exe**.

smss.exe

Image Path: `%SystemRoot%\System32\smss.exe`

Parent Process: `System`

Number of Instances: One master instance and another child instance per session. Children exit after creating their session.

User Account: Local System

Start Time: Within seconds of boot time for the master instance

Description: The Session Manager process is responsible for creating new sessions. The first instance creates a child instance for each new session. Once the child instance initializes the new session by starting the Windows subsystem (**csrss.exe**) and **wininit.exe** for Session 0 or **winlogon.exe** for Session 1 and higher, the child instance exits.

wininit.exe

Image Path: `%SystemRoot%\System32\wininit.exe`

Parent Process: Created by an instance of **smss.exe** that exits, so tools usually do not provide the parent process name.

Number of Instances: One

User Account: Local System

Start Time: Within seconds of boot time

Description: Wininit starts key background processes within Session 0. It starts the Service Control Manager (**services.exe**), the Local Security Authority process (**lsass.exe**), and the Local Session Manager (**lsm.exe**).

taskhost.exe

Image Path: `%SystemRoot%\System32\taskhost.exe`

Parent Process: `services.exe`

Number of Instances: One or more

User Account: Multiple taskhost.exe processes are normal. One or more may be owned by logged-on users and/or by local service accounts.

Start Time: Start times vary greatly

Description: The generic host process for Windows Tasks. This process was renamed to taskhostex.exe for Windows 8 and then to taskhostw.exe for Windows 10. Upon initialization, **taskhost.exe** runs a continuous loop listening for trigger events. Example trigger events that can initiate a task include a defined schedule, user logon, system startup, idle CPU time, a Windows log event, workstation lock, or workstation unlock.

There are more than 70 tasks preconfigured on a default installation of Windows 7 Enterprise (though many are disabled). For example, defrag.exe is scheduled to run against all volumes every Wednesday at 1:00 am. Another default task backs up the core registry hive files every 10 days. All executable files (DLLs & EXEs) used by the default Windows 7/8 scheduled tasks are signed by Microsoft.

lsass.exe

Image Path: `%SystemRoot%\System32\lsass.exe`

Parent Process: `wininit.exe`

Number of Instances: One

User Account: Local System

Start Time: Within seconds of boot time

Description: The Local Security Authentication Subsystem Server process is responsible for authenticating users by calling an appropriate Security Service Provider (SSP) authentication package specified in **HKLM\SYSTEM\CurrentControlSet\Control\Lsa**. Typically this will be the Kerberos SSP for domain accounts or the MSV1_0 SSP for local accounts. Once a user is authenticated, **lsass.exe** generates an access token for the user that specifies security rights and constraints for the user and the user's processes. Only one instance of this process should occur and it should never have child processes.

winlogon.exe

Image Path: `%SystemRoot%\System32\winlogon.exe`

Parent Process: Created by an instance of **smss.exe** that exits, so analysis tools usually do not provide the parent process name.

Number of Instances: One or more

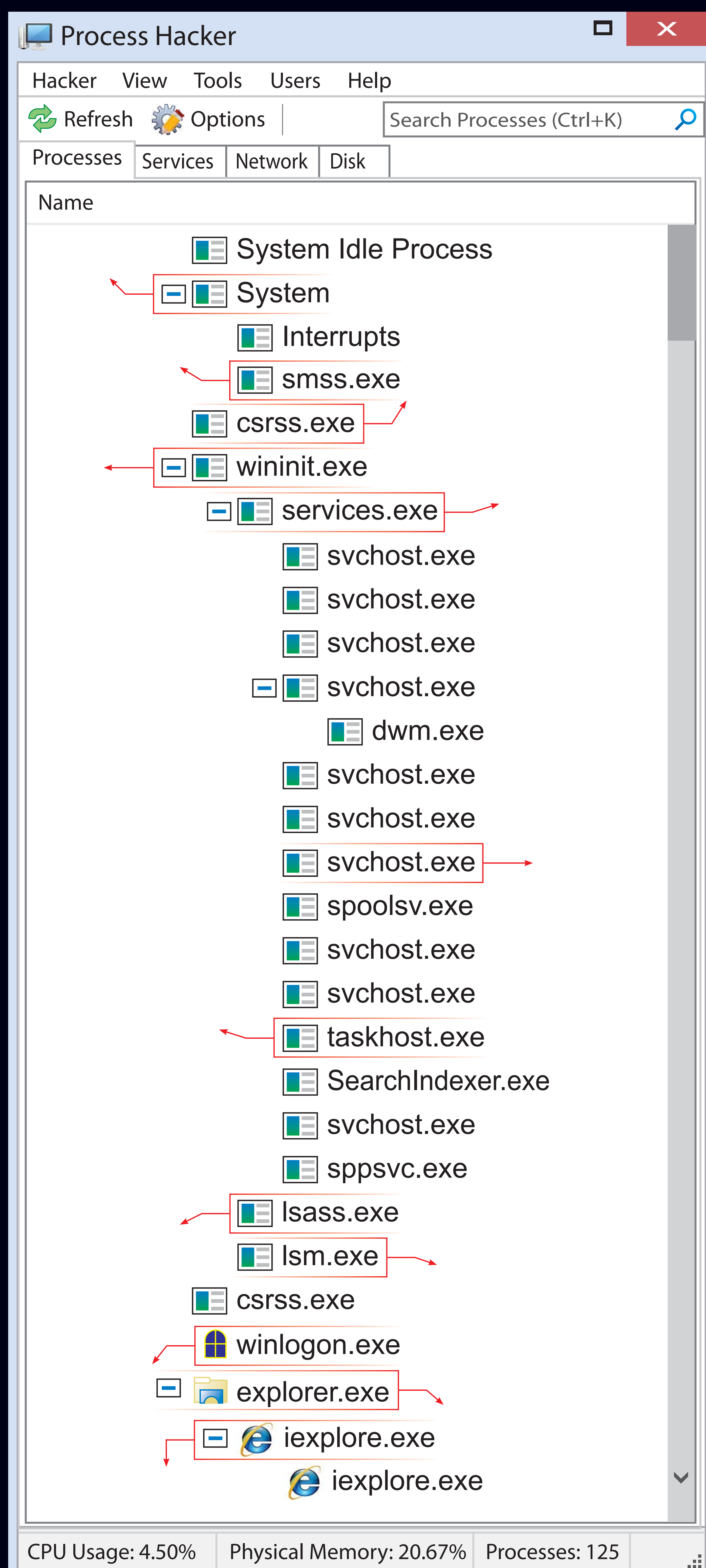
User Account: Local System

Start Time: Within seconds of boot time for the first instance (for Session 1). Start times for additional instances occur as new sessions are created, typically through Remote Desktop or Fast User Switching logons.

Description: Winlogon handles interactive user logons and logoffs. It launches **LogonUI.exe**, which accepts the username and password at the logon screen and passes the credentials to **lsass.exe** to validate the credentials. Once the user is authenticated, Winlogon loads the user's **NTUSER.DAT** into **HKCU** and starts the user's shell (**explorer.exe**) via **Userinit.exe**.

When searching for malicious processes, look for any of these anomalous characteristics:

- **Started with the wrong parent process**
- **Image executable is located in the wrong path**
- **Misspelled processes**
- **Processes that are running under the wrong account (incorrect SID)**
- **Processes with unusual start times (i.e., starts minutes or hours after boot when it should be within seconds of boot)**
- **Unusual command-line arguments**
- **Packed executables**



Process listing from Windows 7 Enterprise

iexplore.exe

Image Path: `\Program Files\Internet Explorer\iexplore.exe` [or `\Program Files (x86)\Internet Explorer\iexplore.exe`]

Parent Process: `explorer.exe`

Number of Instances: 0 to many

User Account: <logged-on user(s)>

Start Time: Typically when user starts Internet Explorer. However, it can be started without explicit user interaction via the "-embedding" switch (in which case, parent may not be **explorer.exe**).

Description: Internet Explorer (IE) is a typical desktop application launched by a user. Such applications will almost always be a child of **explorer.exe**. Modern versions of IE will have a sub-process for each open tab. It does this for several reasons, including enhanced security. When accessing an Internet site, IE will run the tab process with low integrity, which sandboxes the process, making it more difficult for attackers to modify sensitive areas of the registry or file system if they are able to compromise the IE child process. Attackers often name their malware **iexplore.exe** and place it in an alternate directory or misspell **iexplore.exe** as **iexplorer.exe**.

csrss.exe

Image Path: `%SystemRoot%\System32\csrss.exe`

Parent Process: Created by an instance of **smss.exe** that exits, so analysis tools usually do not provide the parent process name.

Number of Instances: Two or more

User Account: Local System

Start Time: Within seconds of boot time for the first 2 instances (for Session 0 and 1). Start times for additional instances occur as new sessions are created, although often only Sessions 0 and 1 are created.

Description: The Client/Server Run-Time Subsystem is the user-mode process for the Windows subsystem. Its duties include managing processes and threads, importing most of the DLLs that provide the Windows API, and facilitating shutdown of the GUI during system shutdown. An instance of **csrss.exe** will run for each session. Session 0 is for services and Session 1 for the local console session. Additional sessions are created through the use of Remote Desktop and/or Fast User Switching. Each new session results in a new instance of **csrss.exe**. Depending on the OS version, **csrss.exe** (prior to Win7/2008 R2) or its child process **conhost.exe** (Win7/2008 R2 and later) contain command history for instances of **cmd.exe**. Searching the address space for these processes is particularly useful when analyzing the memory of compromised hosts.

services.exe

Image Path: `%SystemRoot%\System32\services.exe`

Parent Process: `wininit.exe`

Number of Instances: One

User Account: Local System

Start Time: Within seconds of boot time

Description: Implements the Unified Background Process Manager (UBPM), which is responsible for background activities such as services and scheduled tasks. **Services.exe** also implements the Service Control Manager (SCM), which specifically handles the loading of services and device drivers marked for auto-start. In addition, once a user has successfully logged on interactively, the SCM (**services.exe**) considers the boot successful and sets the Last Known Good control set (**HKLM\SYSTEM\Select\LastKnownGood**) to the value of the CurrentControlSet.

svchost.exe

Image Path: `%SystemRoot%\System32\svchost.exe`

Parent Process: `services.exe`

Number of Instances: Five or more

User Account: Varies depending on svchost instance, though it typically will be Local System, Network Service, or Local Service accounts. Beginning with Windows 10, an instance will start under the user context at logon (with command line "-k UnistackSvcGroup").

Start Time: Typically within seconds of boot time. However, services can be started after boot, which might result in new instances of **svchost.exe** well after boot time.

Description: The generic host process for Windows Services. It is used for running service DLLs. Windows will run multiple instances of **svchost.exe**, each using a unique "**-k**" parameter for grouping similar services. Typical "**-k**" parameters include BTsvcs, DcomLaunch, RPCSS, LocalServiceNetworkRestricted, netsvcs, LocalService, NetworkService, LocalServiceNoNetwork, secsvcs, and LocalServiceAndNoImpersonation. Malware authors often take advantage of the ubiquitous nature of **svchost.exe** and use it either directly or indirectly to hide their malware. They use it directly by installing the malware as a service in a legitimate instance of **svchost.exe**. Alternatively, they use it indirectly by trying to blend in with legitimate instances of **svchost.exe**, either by slightly misspelling the name (e.g., **scvhost.exe**) or spelling it correctly but placing it in a directory other than System32. Keep in mind that a legitimate **svchost.exe** should always run from **%SystemRoot%\System32**, should have **services.exe** as its parent, and should host at least one service. Also, on default installations of Windows 7 and later, all service executables and DLLs are signed by Microsoft.

lsm.exe

Image Path: `%SystemRoot%\System32\lsm.exe`

Parent Process: `wininit.exe`

Number of Instances: One

User Account: Local System

Start Time: Within seconds of boot time

Description: Local Session Manager handles terminal services, including Remote Desktop sessions as well as additional local sessions via Fast User Switching. It communicates with **smss.exe** to start new sessions. Smss in turn creates an additional **csrss.exe** and **winlogon.exe** to support the new session. Only one instance of this process should occur prior to Windows 8 and it should never have child processes. Beginning with Windows 8, this functionality was moved to a service, utilizing a service DLL named lsm.dll (lsm.exe no longer exists).

explorer.exe

Image Path: `%SystemRoot%\explorer.exe`

Parent Process: Created by an instance of **userinit.exe** that exits, so analysis tools usually do not provide the parent process name.

Number of Instances: One per interactively logged-on user

User Account: <logged-on user(s)>

Start Time: Starts when the owner's interactive logon begins

Description: At its core, Explorer provides users access to files. Functionally though, it is both a file browser via Windows Explorer (though still **explorer.exe**) and a user interface providing features such as the user's Desktop, the Start Menu, the Taskbar, the Control Panel, application launching via file extension association, and shortcut files. Note that there should be just one running instance of **explorer.exe** per interactive logon, regardless of multiple Windows Explorer windows opened by the user. Also notice that the legitimate **explorer.exe** resides in the **%SystemRoot%** directory rather than **%SystemRoot%\System32**. Attackers often name their malware **explorer.exe** and place it in System32 or misspell **explorer.exe** as **explore.exe**.