LetsDefend.io - EventID: 114 - [SOC164 - Suspicious Mshta Behavior] Walkthrough

Hey! and welcome to this simple walkthrough of the case “Suspicious Mshta Behavior” on LetsDefend.io!

At first, we should understand what mshta stands for and how can it be manipulated by threat actors.

Adversaries may manipulate mshta.exe to proxy execution of malicious/suspicious .hta files and Javascript or VBScript through a trusted Windows utility. There are several examples of different types of threats leveraging mshta.exe during initial compromise and for execution of code.

As an example, Lazarus, use mshta to execute HTML pages downloaded by initial access documents. APT29, APT32 and FIN7 use mshta for remote code execution on victim systems. We can find many more examples but lets’ get back to our topic for now, I’ll share useful resources at the end of this documentation.

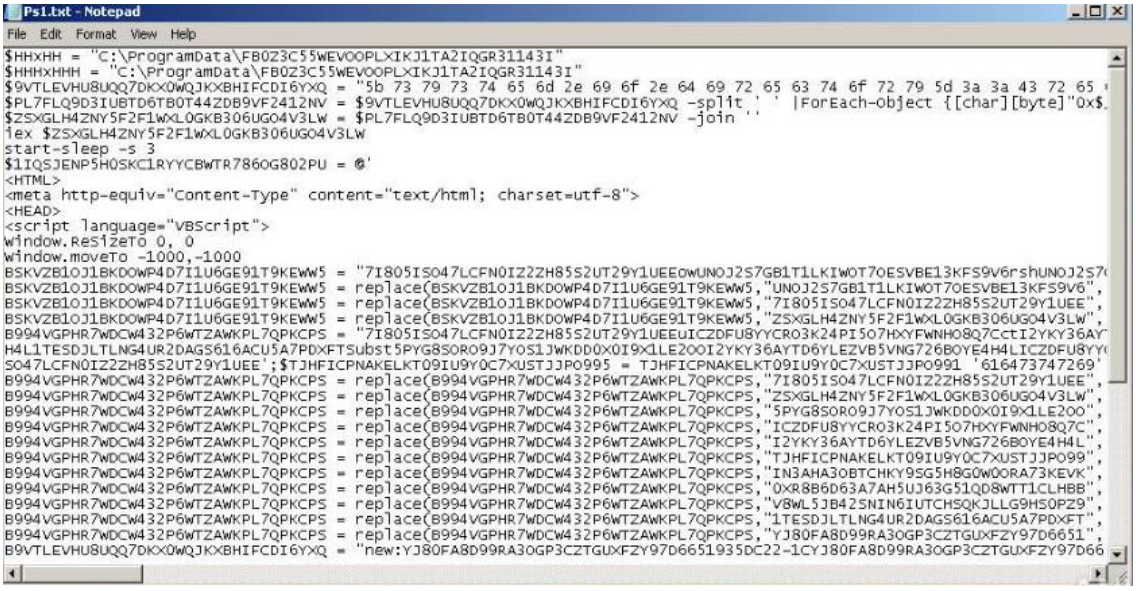
Since this alert triggered from an execution through a file -ps1.hta- with low reputation, we should check the hash value through OSINT sources for understanding the structure and related suspicious behaviors.

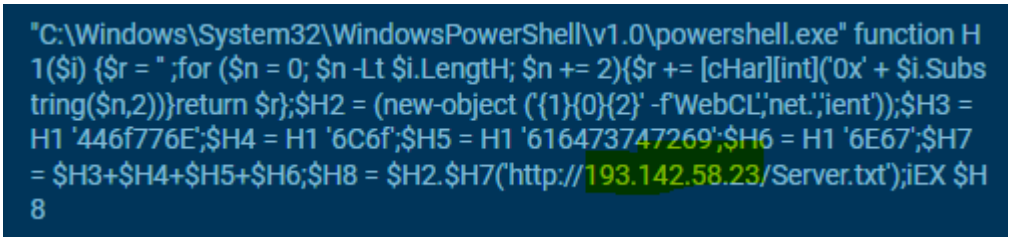
MD5 Hash: 6685c433705f558c5535789234db0e5a

VirusTotal - File - 886095c7861a068d1ee603c71cb161f256941e802e743fe2161f30013947a2f1

Ps1.txt (MD5: 6685C433705F558C5535789234DB0E5A) - Interactive analysis - ANY.RUN

From the OSINT sources Virustotal and Anyrun, we can see this is an obfuscated javascript file changes it’s extention from txt to hta to run it directly through mshta and has a suspicious Powershell script embedded in it, which contains c2 connection details as seen below. And since there is a suspicious child process creation, we have serious indicators that our system could be compromised by this particular .hta file.





After collecting related information, we can start our investigation on the host “Roberto” to examine related behavioral activities and determine if this host is compromised or not.

In this case, we should check related logs, endpoint activities and answer how did Roberto get this malicious file step by step.

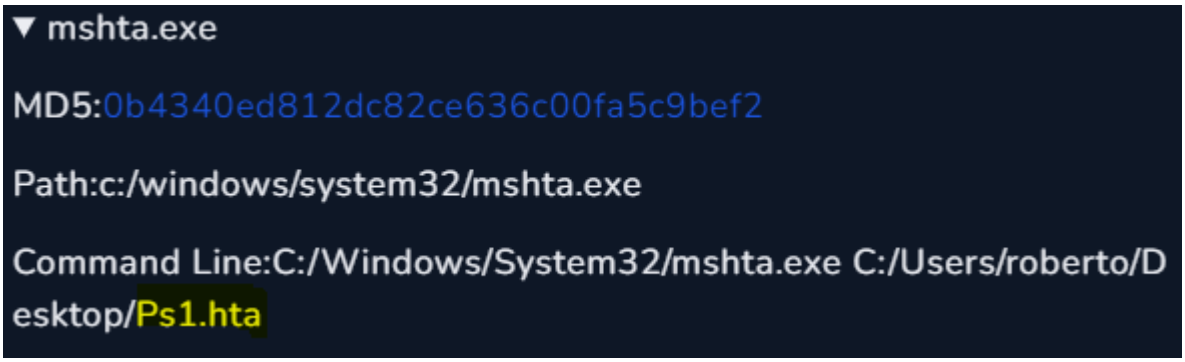
Artifacts:

C2 Address: 193[.]142[.]58[.]23

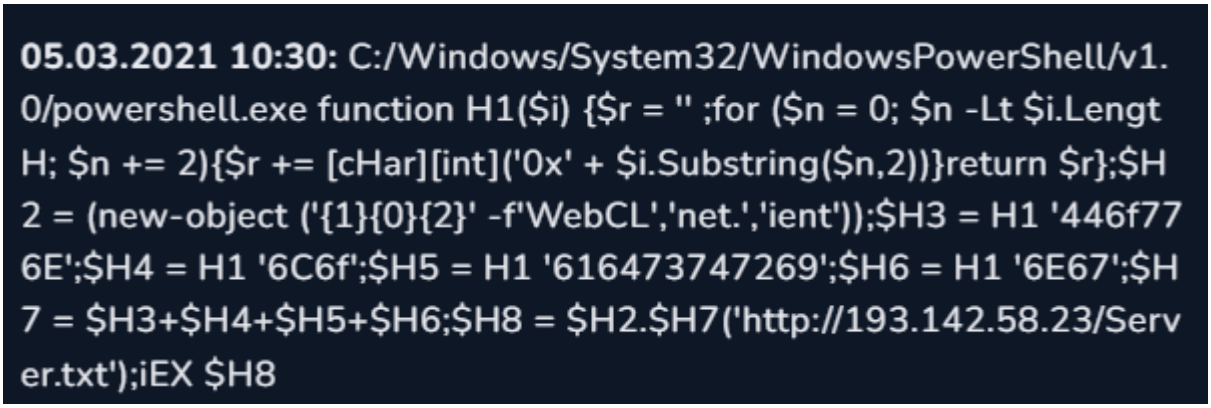
Source IP: 172.16.17.38

Source Host: Roberto

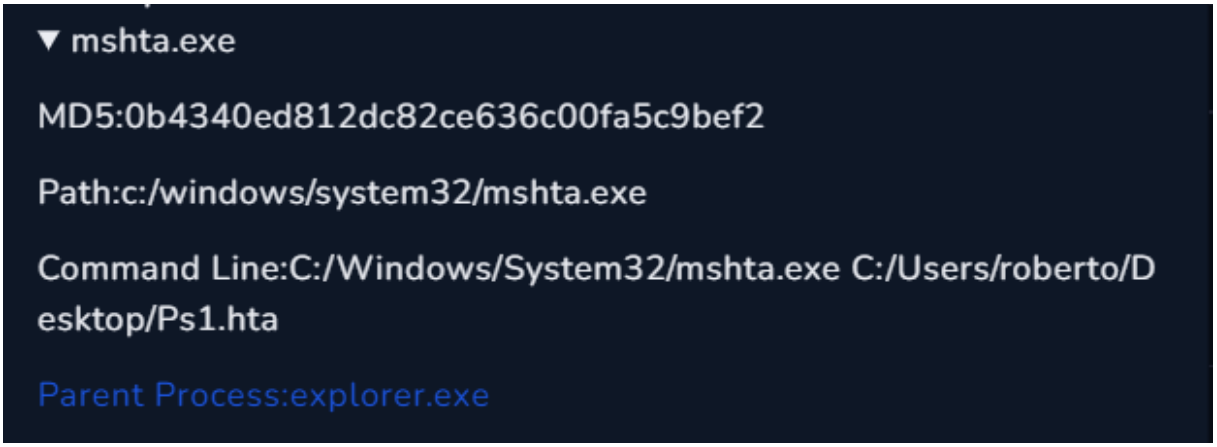
From this particular process entry, we can easily say Roberto run this file on his system.



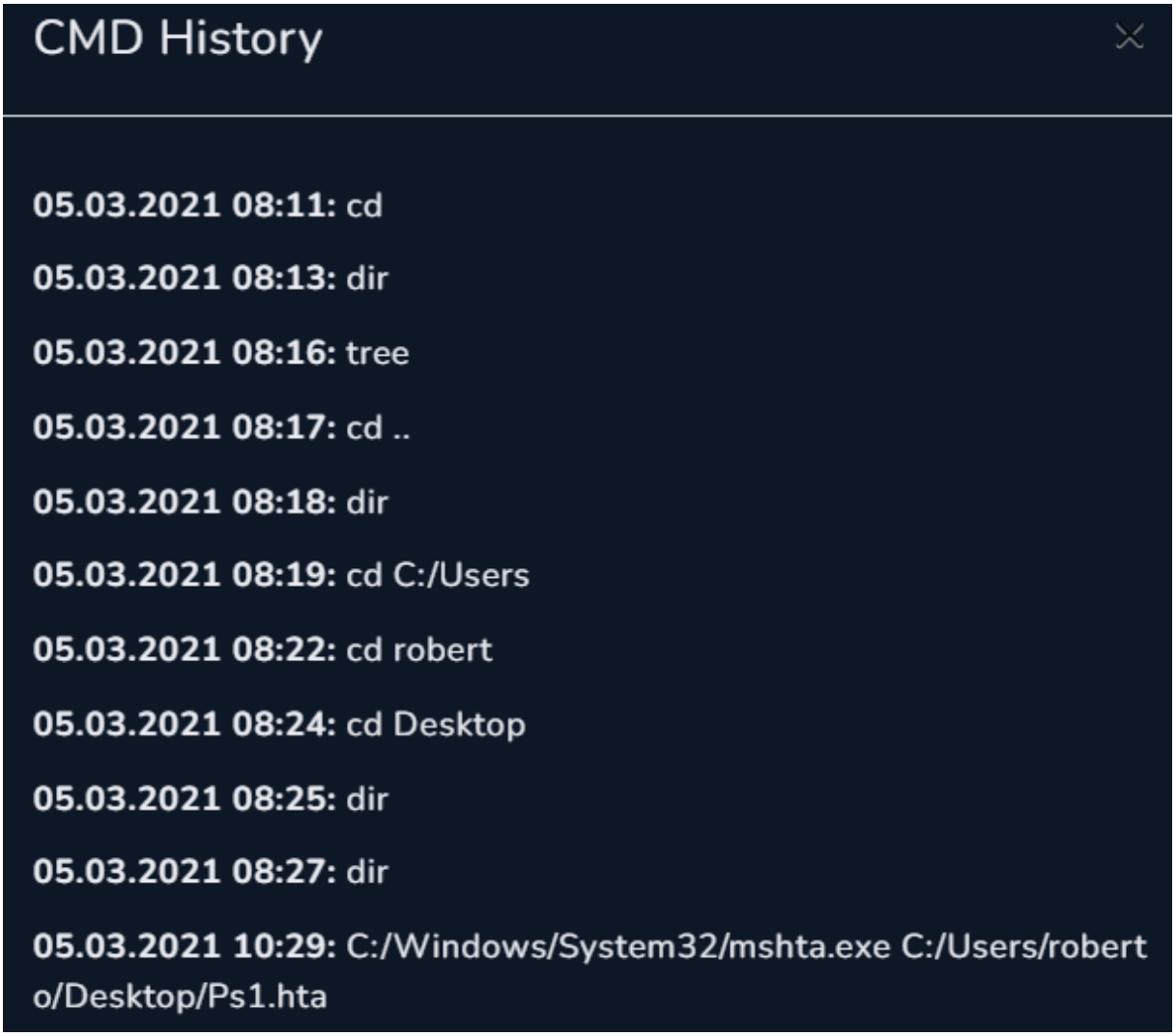
And through the command line history, we can also see the malicious powershell code execution which forwards our victim to the c2 address. 05.03.2021 10:30 is meaningful for us, because this time slot overlaps with our alert time as well.



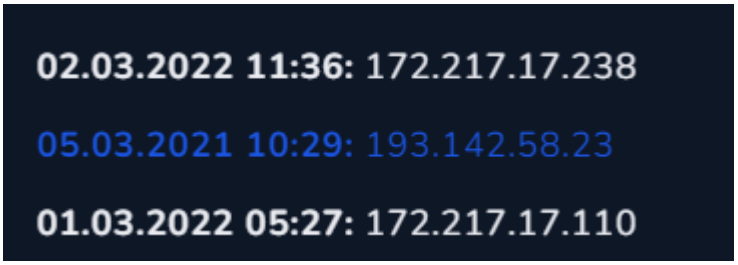
But, there is an important thing to discuss, we can see the parent process of suspicious mshta is Explorer.exe, and this gives us an important hint about a user interacted execution.



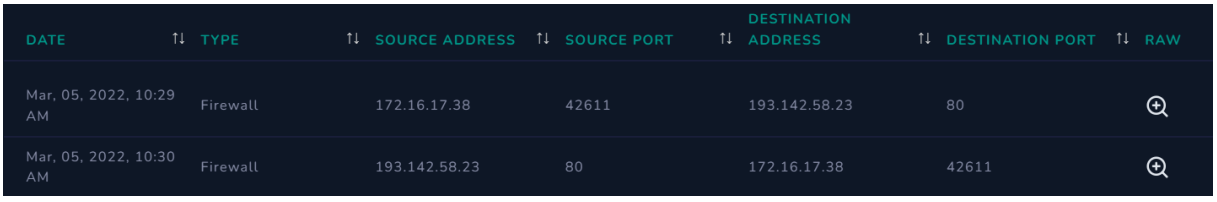
And from the command line history of Roberto's host, we can see the malicious file execution proceed similarly.



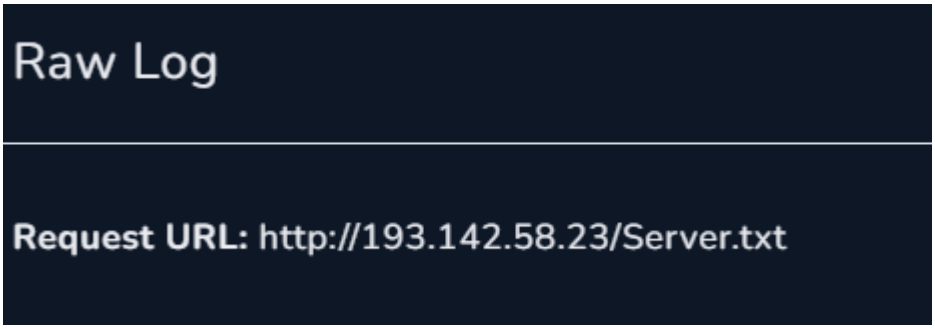
From EDR details, we can see related network connection activity as well:



Now we are totally sure that malicious file executed on our victims' host, so we should take a look at related logs to detect any successful c2 connection activity because with this malicious code block, attacker aims to connect to a c2 address as we seen above.



From the logs, we can see the source 172.16.17.38 sent a request to the c2 server through HTTP port which a 100% match with our suspicious activity as expected.



But in this phase, it's important to check the response from malicious address to determine if it's a successful connection.

As conclusion, we can say Roberto executed this malicious file on his host, and we are lucky that this c2 address respond with 404 this time! Usually those kind of malicious files drop another malicious content or serves for the purpose of data leakage, we should be aware of legal binaries can be used for malicious activities (known as LOLBins) as well and in similar cases, we should analyze behavioral activities and develop related scenarios to detect and prevent those kind of manipulations. As a baseline, we can refer to those SANS posters as well.

So, in this case, we must isolate the compromised host through EDR (containment) for prevent any further lateral movement or malicious activity and we can close this alert with an explaination defined in the resolution summary.

Resolution Summary:

- Alert Type: True Positive
- Containment Required? : Yes
- Who Performed the Activity? : User
- What Is Suspicious Activity? / Determine Suspicious Activity : Mshta.exe, is normally used for executing HTLM application files. In this case, it's malformed by executing ps1.hta file which is detected as a trojan. From the EDR, we can see an IP communication through cmd and downloaded a file using web client. So it works as a downloader at initial state. Although there are communication (from logs) with c2 IP, there is not any successful c2 communication(404).
- Identify the Binary : Mshta.exe as LOLBin

Useful Links:

System Binary Proxy Execution: Mshta, Sub-technique T1218.005 - Enterprise | MITRE ATT&CK®

Hunt Evil | SANS Poster

FIN7 Evolution and the Phishing LNK | Mandiant

eset_threat_report_t32021.pdf (welivesecurity.com)

Cybereason Labs Analysis Operation Cobalt Kitty.pdf (hubspot.net)

LolZarus: Lazarus Group Incorporating Lolbins into Campaigns | Qualys Security Blog

North Korea's Lazarus APT leverages Windows Update client, GitHub in latest campaign | Malwarebytes Labs

lazyscripter.pdf (malwarebytes.com)

Security 101: What are LOLBins and How Can They be Used Maliciously? - SecurityHQ

SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

Know Normal...Find Evil

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. Tasks are similar in nature to services, and in fact beginning with Windows 7, are handled through the same Universal Background Process Manager (UBPM) facility. 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

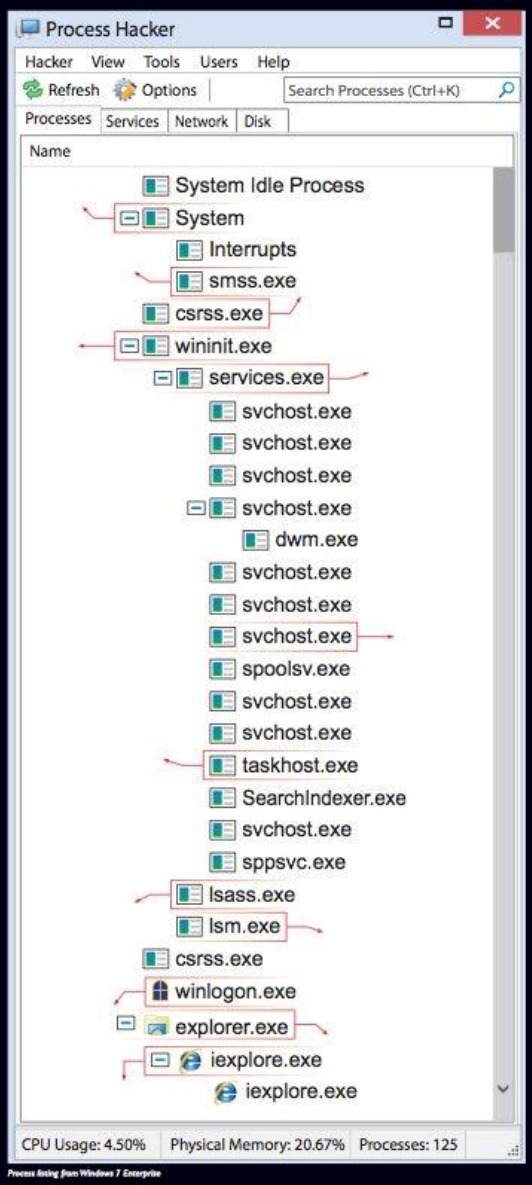
Image Path: `%SystemRoot%\System32\lsass.exe`
Parent Process: `wininit.exe`
Number of Instances: One
User Account: Local System
Start Time: Within seconds of boot time
Description: The Local Security Authentication Subsystem Server process is responsible for authenticating users by calling an appropriate 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. Instances running under any other account should be investigated.
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, all service executables and all service 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 and it should never have child processes.

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`.