Digital Forensics and Incident Response

Cheatsheet containing a variety of commands and concepts relating to digital forensics and incident response.

124 minute read

Introduction

This post is inspired by all the hard working DFIR, and more broadly security professionals, who have put in the hard yards over the years to discuss in depth digital forensics and incident response.

Disclaimer

This page contains a variety of commands and concepts which are known through experience, higher education, tutorials, online blogs, YouTube Videos, professional training, reading the manual, and more. All references to original posts or material will aim to be documented in the 'Special Thanks' section.

This is not designed as a manual on how to perform DFIR, and serves only as a quick reference sheet for commands, tools, and common items of interest when performing Incident Response. If you need to undertake Digital Forensics for legal proceedings, seek specialist advice as this requires more rigor around Identification, Preservation, Collection, Examination, Analysis, and Presentation of findings.

Artifact locations

A number of forensic artifacts are known for a number of operating systems.

A large number of these are covered on the Digital Forensics Artifact Repository, and can be ingested both by humans and systems given the standard YAML format.

- ForensicArtifacts (https://github.com/ForensicArtifacts/artifacts/tree/master/data)

One action you can take is to parse this for items of interest and then directly spit out areas for investigation. For example if you have the PowerShell ConvertFrom-Yaml module (https://github.com/cloudbase/powershell-yaml), you can query this directly.

Get an object of forensic artifacts

```
$WindowsArtifacts=$(curl
https://raw.githubusercontent.com/ForensicArtifacts/artifacts/master/data/windows.yaml)
$obj = ConvertFrom-Yaml $WindowsArtifacts.Content -AllDocuments
```

Now that it is stored within a format we can use the below will give us information at a glance.

```
$count=0;
foreach ($Artifact in $obj){
$Artifacts = [pscustomobject][ordered]@{
        Name = $obj.name[$count]
        Description = $obj.doc[$count]
        References = $obj.urls[$count]
        Attributes = $obj.sources.attributes[$count]
}
$count++;
$Artifacts | FL;
}
```

Query object for relevant registry keys:

```
$obj.sources.attributes.keys|Select-String "HKEY"
$obj.sources.attributes.key_value_pairs
```

Query object for relevant file paths:

```
$obj.sources.attributes.paths
```

Windows Cheat Sheet

Order of Volatility

If performing Evidence Collection rather than IR, respect the order of volatility as defined in: rfc3227

- registers, cache
- routing table, arp cache, process table, kernel statistics, memory
- temporary file systems
- disk
- remote logging and monitoring data that is relevant to the system in question
- physical configuration, network topology
- archival media

Memory Files (Locked by OS during use)

Note: To obtain these files while they're in use you can use a low level file extractor such as RawCopy (https://github.com/jschicht/RawCopy)

hiberfil.sys (RAM stored during machine hibernation)

- %SystemRoot%\hiberfil.sys

pagefile.sys (Virtual memory used by Windows)

- %SystemDrive%\pagefile.sys

swapfile.sys (Virtual memory used by Windows Store Apps)

- %SystemDrive%\swapfile.sys

Binalyze IREC Evidence Collector (https://binalyze.com/products/irec) (GUI or CommandLine)

```
IREC.exe --license AAAA-BBBB-CCDD-DDDD --profile memory
```

Note: Can be used as an all in one collector (License required for full collection, free version available).

Latest documentation (https://irec.readthedocs.io/en/latest/commandline.html)

Belkasoft Live RAM Capturer (https://belkasoft.com/get?product=ram)

```
RamCapture64.exe "output.mem"
```

OR for 32 bit OS

```
RamCapture32.exe "output.mem"
```

Redline

Excellent resource:

Infosec Institute - Memory Analysis using Redline (https://resources.infosecinstitute.com/memory-analysis-using-redline/)

Memoryze

```
MemoryDD.bat --output [LOCATION]
```

Comae DumpIT

```
DumpIt.exe /O [LOCATION]

        - Used for getting a memory crash file (Useful for analysis with both windbg and volatility)

DumpIt.exe /O [LOCATION]\mem.raw /T RAW

        - Used for getting a raw memory dump (Considered a legacy format)
```

These can be bundled with PSEXEC to execute on a remote PC; however, this will copy the file to the remote PC for executing. There's limitations if the tool requires other drivers or files to execute (such as RamCapture). An example command may be:

```
psexec \\remotepcname -c DumpIt.exe
```

Powershell

Where the Microsoft Storage namespace is available (known not to be available in Win7), PowerShell can be used to invoke a native live memory dump. Special thanks to Grzegorz Tworek - 0gtweet (https://twitter.com/0gtweet/status/1273264867382788096?s=20).

```
$ss = Get-CimInstance -ClassName MSFT_StorageSubSystem -Namespace Root\Microsoft\Windows\Storage;
Invoke-CimMethod -InputObject $ss -MethodName "GetDiagnosticInfo" -Arguments @{DestinationPath="
[LOCATION]\dmp"; IncludeLiveDump=$true};
```

Magnet Forensics (Mostly GUI)

- Magnet Forensics Tools (https://www.magnetforensics.com/resources/?cat=Free%20Tool)
- Magnet RAM Capture (https://www.magnetforensics.com/free-tool-magnet-ram-capture)
- Magnet Process Capture (https://www.magnetforensics.com/resources/magnet-process-capture/)

Volexity Surge (https://www.volexity.com/blog/2018/06/12/surge-collect-provides-reliable-memory-acquisition-across-windows-linux-and-macos/)

Microsoft LiveKd (https://docs.microsoft.com/en-us/sysinternals/downloads/livekd)

Winpmem (https://github.com/Velocidex/WinPmem/releases)

Winpmem Docs (https://winpmem.velocidex.com/docs/memory/)

```
winpmem.exe -o test.aff4 -dd
winpmem.exe -o test.raw --format raw -dd
```

Imaging Live Machines

FTK Imager (Cmd version, mostly GUI for new versions) (https://accessdata.com/product-download)

```
ftkimager --list-drives
ftkimager \\.\PHYSICALDRIVE0 "[Location]\Case" --e01
ftkimager [source] [destination]
ftkimager \\.\PHYSICALDRIVE0 "[Location]\Case" --e01 --outpass securepasswordinsertedhere
```

DD

```
dd.exe --list
dd.exe if=/dev[drive] of=Image.img bs=1M
dd.exe if=\\.\[OSDrive]: of=[drive]:\[name].img bs=1M --size --progress
(LINUX) sudo dd if=/dev/[OSDrive] of=/mnt/[name].ddimg bs=1M conv=noerror,sync
```

X-Ways Imager (https://www.x-ways.net/order.html)

Encase Forensic (https://www.guidancesoftware.com/encase-forensic)

Tableau Imager (https://www.guidancesoftware.com/tableau/download-center)

Guymager (https://guymager.sourceforge.io/)

Carving Out Files From Image using Scalpel (https://github.com/sleuthkit/scalpel)

```
nano /etc/scalpel/scalpel.conf
mkdir carve
scalpel imagefile.img -o carve
```

Live Windows IR/Triage

CMD and WMIC (Windows Management Instrumentation Command-Line) Note: less information can be gathered by using 'list brief'.

Interact with remote machine

Enable Powershell remoting:

```
wmic /node:[IP] process call create "powershell enable-psremoting -force"
```

Powershell (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/enter-pssession?view=powershell-6):

```
Enter-PSSession -ComputerName [IP]
```

PSExec:

```
PsExec: psexec \\IP -c cmd.exe
```

System information

```
get-computerinfo
echo %DATE% %TIME%
date /t
time /t
reg query "HKLM\System\CurrentControlSet\Control\TimeZoneInformation"
systeminfo
wmic computersystem list full
wmic /node:localhost product list full /format:csv
wmic softwarefeature get name,version /format:csv
wmic softwareelement get name,version /format:csv
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion" /s
echo %PATH%
(gci env:path|Select -exp Value).split(';')
SET
wmic bootconfig get /all /format:List
wmic computersystem get name, domain, manufacturer, model,
numberofprocessors,primaryownername,username,roles,totalphysicalmemory /format:list
wmic timezone get Caption, Bias, DaylightBias, DaylightName, StandardName
wmic recoveros get /all /format:List
wmic os get /all /format:list
wmic partition get /all /format:list
wmic logicaldisk get /all /format:list
wmic diskdrive get /all /format:list
fsutil fsinfo drives
$env
Get-Variable
```

(psinfo requires sysinternals psinfo.exe):

```
psinfo -accepteula -s -h -d
```

Obtain list of all files on a computer

```
tree C:\ /F > output.txt
dir C:\ /A:H /-C /Q /R /S /X
```

User and admin information

```
whoami
whoami /user
net users
net localgroup administrators
net group /domain [groupname]
net user /domain [username]
wmic sysaccount
wmic useraccount get name,SID
wmic useraccount list
```

Logon information

```
wmic netlogin list /format:List
Get-WmiObject Win32_LoggedOnUser
Get-WmiObject win32_logonsession
query user
qwinsta
klist sessions
klist -li
```

NT Domain/Network Client Information

```
wmic ntdomain get /all /format:List
wmic netclient get /all /format:List
nltest /trusted_domains
```

Firewall Information

```
netsh Firewall show state
netsh advfirewall firewall show rule name=all dir=in type=dynamic
netsh advfirewall firewall show rule name=all dir=out type=dynamic
netsh advfirewall firewall show rule name=all dir=in type=static
netsh advfirewall firewall show rule name=all dir=out type=static
```

Firewall Changes

```
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-Windows Firewall With Advanced
Security/Firewall';} | FL TimeCreated, Message
```

Applications which have added a firewall rule

```
$events=Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-Windows Firewall With Advanced
Security/Firewall'; Id='2004'};
$output = @();
foreach ($Event in $events){
$data = New-Object -TypeName PSObject;
$XML = [xml]$Event.ToXml();
$RuleId=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleId'} | Select -exp InnerText;
$RuleName=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleName'} | Select -exp InnerText;
$Origin=$XML.Event.EventData.Data|?{$_.Name -eq 'Origin'} | Select -exp InnerText;
$ApplicationPath=$XML.Event.EventData.Data|?{$_.Name -eq 'ApplicationPath'} | Select -exp InnerText;
$ServiceName=$XML.Event.EventData.Data|?{$_.Name -eq 'ServiceName'} | Select -exp InnerText;
$Direction=$XML.Event.EventData.Data|?{$_.Name -eq 'Direction'} | Select -exp InnerText;
$Protocol=$XML.Event.EventData.Data|?{$_.Name -eq 'Protocol'} | Select -exp InnerText;
$LocalPorts=$XML.Event.EventData.Data|?{$_.Name -eq 'LocalPorts'} | Select -exp InnerText;
$RemotePorts=$XML.Event.EventData.Data|?{$_.Name -eq 'RemotePorts'} | Select -exp InnerText;
$Action=$XML.Event.EventData.Data|?{$_.Name -eq 'Action'} | Select -exp InnerText;
$Profiles=$XML.Event.EventData.Data|?{$_.Name -eq 'Profiles'} | Select -exp InnerText;
$LocalAddresses=$XML.Event.EventData.Data|?{$_.Name -eq 'LocalAddresses'} | Select -exp InnerText;
$RemoteAddresses=$XML.Event.EventData.Data|?{$_.Name -eq 'RemoteAddresses'} | Select -exp InnerText;
$RemoteMachineAuthorizationList=$XML.Event.EventData.Data|?{$_.Name -eq
'RemoteMachineAuthorizationList'} | Select -exp InnerText;
$RemoteUserAuthorizationList=$XML.Event.EventData.Data|?{$_.Name -eq 'RemoteUserAuthorizationList'} |
Select -exp InnerText;
$EmbeddedContext=$XML.Event.EventData.Data|?{$_.Name -eq 'EmbeddedContext'} | Select -exp InnerText;
$Flags=$XML.Event.EventData.Data|?{$_.Name -eq 'Flags'} | Select -exp InnerText;
$EdgeTraversal=$XML.Event.EventData.Data|?{$_.Name -eq 'EdgeTraversal'} | Select -exp InnerText;
$LooseSourceMapped=$XML.Event.EventData.Data|?{$_.Name -eq 'LooseSourceMapped'} | Select -exp
InnerText;
$SecurityOptions=$XML.Event.EventData.Data|?{$_.Name -eq 'SecurityOptions'} | Select -exp InnerText;
$ModifyingUser=$XML.Event.EventData.Data|?{$_.Name -eq 'ModifyingUser'} | Select -exp InnerText;
$ModifyingApplication=$XML.Event.EventData.Data|?{$_.Name -eq 'ModifyingApplication'} | Select -exp
InnerText;
$SchemaVersion=$XML.Event.EventData.Data|?{$_.Name -eq 'SchemaVersion'} | Select -exp InnerText;
$RuleStatus=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleStatus'} | Select -exp InnerText;
$LocalOnlyMapped=$XML.Event.EventData.Data|?{$_.Name -eq 'LocalOnlyMapped'} | Select -exp InnerText;
$data `
| Add-Member NoteProperty RuleId "$RuleId" -PassThru `
| Add-Member NoteProperty RuleName "$RuleName" -PassThru `
| Add-Member NoteProperty Origin "$Origin" -PassThru `
| Add-Member NoteProperty ApplicationPath "$ApplicationPath" -PassThru `
| Add-Member NoteProperty ServiceName "$ServiceName" -PassThru `
| Add-Member NoteProperty Direction "$Direction" -PassThru `
```

```
    | Add-Member NoteProperty Protocol "$Protocol" -PassThru `
    | Add-Member NoteProperty LocalPorts "$LocalPorts" -PassThru `
    | Add-Member NoteProperty RemotePorts "$RemotePorts" -PassThru `
    | Add-Member NoteProperty Action "$Action" -PassThru `
    | Add-Member NoteProperty Profiles "$Profiles" -PassThru `
    | Add-Member NoteProperty LocalAddresses "$LocalAddresses" -PassThru `
    | Add-Member NoteProperty RemoteAddresses "$RemoteAddresses" -PassThru `
    | Add-Member NoteProperty RemoteMachineAuthorizationList "$RemoteMachineAuthorizationList" -PassThru `
    | Add-Member NoteProperty RemoteUserAuthorizationList "$RemoteUserAuthorizationList" -PassThru `
    | Add-Member NoteProperty EmbeddedContext "$EmbeddedContext" -PassThru `
    | Add-Member NoteProperty Flags "$Flags" -PassThru `
    | Add-Member NoteProperty EdgeTraversal "$EdgeTraversal" -PassThru `
    | Add-Member NoteProperty LooseSourceMapped "$LooseSourceMapped" -PassThru `
    | Add-Member NoteProperty SecurityOptions "$SecurityOptions" -PassThru `
    | Add-Member NoteProperty ModifyingUser "$ModifyingUser" -PassThru `
    | Add-Member NoteProperty ModifyingApplication "$ModifyingApplication" -PassThru `
    | Add-Member NoteProperty SchemaVersion "$SchemaVersion" -PassThru `
    | Add-Member NoteProperty RuleStatus "$RuleStatus" -PassThru `
    | Add-Member NoteProperty LocalOnlyMapped "$LocalOnlyMapped" -PassThru | Out-Null
    $output += $data;
    }
    $output = $output
    $output | select -exp ModifyingApplication | sort -u | unique
```

Applications which have modified a firewall rule

```
$events=Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-Windows Firewall With Advanced
Security/Firewall'; Id='2005'};
$output = @();
foreach ($Event in $events){
$data = New-Object -TypeName PSObject;
$XML = [xml]$Event.ToXml();
$RuleId=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleId'} | Select -exp InnerText;
$RuleName=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleName'} | Select -exp InnerText;
$Origin=$XML.Event.EventData.Data|?{$_.Name -eq 'Origin'} | Select -exp InnerText;
$ApplicationPath=$XML.Event.EventData.Data|?{$_.Name -eq 'ApplicationPath'} | Select -exp InnerText;
$ServiceName=$XML.Event.EventData.Data|?{$_.Name -eq 'ServiceName'} | Select -exp InnerText;
$Direction=$XML.Event.EventData.Data|?{$_.Name -eq 'Direction'} | Select -exp InnerText;
$Protocol=$XML.Event.EventData.Data|?{$_.Name -eq 'Protocol'} | Select -exp InnerText;
$LocalPorts=$XML.Event.EventData.Data|?{$_.Name -eq 'LocalPorts'} | Select -exp InnerText;
$RemotePorts=$XML.Event.EventData.Data|?{$_.Name -eq 'RemotePorts'} | Select -exp InnerText;
$Action=$XML.Event.EventData.Data|?{$_.Name -eq 'Action'} | Select -exp InnerText;
$Profiles=$XML.Event.EventData.Data|?{$_.Name -eq 'Profiles'} | Select -exp InnerText;
$LocalAddresses=$XML.Event.EventData.Data|?{$_.Name -eq 'LocalAddresses'} | Select -exp InnerText;
$RemoteAddresses=$XML.Event.EventData.Data|?{$_.Name -eq 'RemoteAddresses'} | Select -exp InnerText;
$RemoteMachineAuthorizationList=$XML.Event.EventData.Data|?{$_.Name -eq
'RemoteMachineAuthorizationList'} | Select -exp InnerText;
$RemoteUserAuthorizationList=$XML.Event.EventData.Data|?{$_.Name -eq 'RemoteUserAuthorizationList'} |
Select -exp InnerText;
$EmbeddedContext=$XML.Event.EventData.Data|?{$_.Name -eq 'EmbeddedContext'} | Select -exp InnerText;
$Flags=$XML.Event.EventData.Data|?{$_.Name -eq 'Flags'} | Select -exp InnerText;
$EdgeTraversal=$XML.Event.EventData.Data|?{$_.Name -eq 'EdgeTraversal'} | Select -exp InnerText;
$LooseSourceMapped=$XML.Event.EventData.Data|?{$_.Name -eq 'LooseSourceMapped'} | Select -exp
InnerText;
$SecurityOptions=$XML.Event.EventData.Data|?{$_.Name -eq 'SecurityOptions'} | Select -exp InnerText;
$ModifyingUser=$XML.Event.EventData.Data|?{$_.Name -eq 'ModifyingUser'} | Select -exp InnerText;
$ModifyingApplication=$XML.Event.EventData.Data|?{$_.Name -eq 'ModifyingApplication'} | Select -exp
InnerText;
$SchemaVersion=$XML.Event.EventData.Data|?{$_.Name -eq 'SchemaVersion'} | Select -exp InnerText;
$RuleStatus=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleStatus'} | Select -exp InnerText;
$LocalOnlyMapped=$XML.Event.EventData.Data|?{$_.Name -eq 'LocalOnlyMapped'} | Select -exp InnerText;
$data `
| Add-Member NoteProperty RuleId "$RuleId" -PassThru `
| Add-Member NoteProperty RuleName "$RuleName" -PassThru `
| Add-Member NoteProperty Origin "$Origin" -PassThru `
| Add-Member NoteProperty ApplicationPath "$ApplicationPath" -PassThru `
| Add-Member NoteProperty ServiceName "$ServiceName" -PassThru `
| Add-Member NoteProperty Direction "$Direction" -PassThru `
```

```
    | Add-Member NoteProperty Protocol "$Protocol" -PassThru `
    | Add-Member NoteProperty LocalPorts "$LocalPorts" -PassThru `
    | Add-Member NoteProperty RemotePorts "$RemotePorts" -PassThru `
    | Add-Member NoteProperty Action "$Action" -PassThru `
    | Add-Member NoteProperty Profiles "$Profiles" -PassThru `
    | Add-Member NoteProperty LocalAddresses "$LocalAddresses" -PassThru `
    | Add-Member NoteProperty RemoteAddresses "$RemoteAddresses" -PassThru `
    | Add-Member NoteProperty RemoteMachineAuthorizationList "$RemoteMachineAuthorizationList" -PassThru `
    | Add-Member NoteProperty RemoteUserAuthorizationList "$RemoteUserAuthorizationList" -PassThru `
    | Add-Member NoteProperty EmbeddedContext "$EmbeddedContext" -PassThru `
    | Add-Member NoteProperty Flags "$Flags" -PassThru `
    | Add-Member NoteProperty EdgeTraversal "$EdgeTraversal" -PassThru `
    | Add-Member NoteProperty LooseSourceMapped "$LooseSourceMapped" -PassThru `
    | Add-Member NoteProperty SecurityOptions "$SecurityOptions" -PassThru `
    | Add-Member NoteProperty ModifyingUser "$ModifyingUser" -PassThru `
    | Add-Member NoteProperty ModifyingApplication "$ModifyingApplication" -PassThru `
    | Add-Member NoteProperty SchemaVersion "$SchemaVersion" -PassThru `
    | Add-Member NoteProperty RuleStatus "$RuleStatus" -PassThru `
    | Add-Member NoteProperty LocalOnlyMapped "$LocalOnlyMapped" -PassThru | Out-Null
    $output += $data;
    }
    $output = $output
    $output | select -exp ModifyingApplication | sort -u | unique
```

Applications which have deleted a firewall rule

```
$events=Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-Windows Firewall With Advanced Security/Firewall'; Id='2006'};
$output = @();
foreach ($Event in $events){
$data = New-Object -TypeName PSObject;
$XML = [xml]$Event.ToXml();
$RuleId=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleId'} | Select -exp InnerText;
$RuleName=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleName'} | Select -exp InnerText;
$ModifyingUser=$XML.Event.EventData.Data|?{$_.Name -eq 'ModifyingUser'} | Select -exp InnerText;
$ModifyingApplication=$XML.Event.EventData.Data|?{$_.Name -eq 'ModifyingApplication'} | Select -exp InnerText;
$data `
| Add-Member NoteProperty RuleId "$RuleId" -PassThru `
| Add-Member NoteProperty RuleName "$RuleName" -PassThru `
| Add-Member NoteProperty ModifyingUser "$ModifyingUser" -PassThru `
| Add-Member NoteProperty ModifyingApplication "$ModifyingApplication" -PassThru ` | Out-Null
$output += $data;
}
$output = $output
$output | select -exp ModifyingApplication | sort -u | unique
```

Pagefile information

```
wmic pagefile
```

Group and access information

(Accesschk requires accesschk64.exe or accesschk.exe from sysinternals):

```
net localgroup
accesschk64 -a *
```

Cookies

```
C:\Users\*\AppData\Local\Microsoft\Windows\INetCookies
C:\Users\*\AppData\Roaming\Microsoft\Windows\Cookies
C:\Users\*\AppData\Roaming\Microsoft\Windows\Cookies\Low
```

RecentDocs Information

Special thanks Barnaby Skeggs (https://twitter.com/barnabyskeggs)

*Note: Run with Powershell, get SID and user information with 'wmic useraccount get name,SID'

```
$SID = "S-1-5-21-1111111111-11111111111-1111111-11111"; $output = @(); Get-Item -Path
"Registry::HKEY_USERS\$SID\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs" | Select-
Object -ExpandProperty property | ForEach-Object {$i = [System.Text.Encoding]::Unicode.GetString((gp
"Registry::HKEY_USERS\$SID\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs" -Name
$_).$_); $i = $i -replace '[^a-zA-Z0-9 \.\-_\\/()~ ]', '\^'; $output += $i.split('\^')[0]}; $output |
Sort-Object -Unique
```

More information on recent documents may be found:

```
C:\Users\[username]\AppData\Local\Microsoft\Windows\FileHistory\Data
gci "REGISTRY::HKU\*\Software\Microsoft\Office\*\Word\Reading Locations\*"
```

Get NTFS File Streams from Mounted Windows Drive on Linux OS

```
getfattr -R -n ntfs.streams.list /mnt/[filepath]
```

Startup process information

```
wmic startup list full
wmic startup list brief
Get-CimInstance Win32_StartupCommand | Select-Object Name, command, Location, User | FL
```

Startup process information by path/file name

Note: This will search common persistence areas but not all of them, change the $Malware variable value to a term of your choosing.

```
$Malware = "appdata";
$Processes = gps |?{$_.Path -match $Malware -or $_.Name -match $Malware} | FL Name,Path,Id;
$Tasks = schtasks /query /fo csv /v | ConvertFrom-Csv | ?{"$_.Task To Run" -match $Malware}| FL
"Taskname","Task To Run","Run As User";
$Services = gwmi win32_service | ? {$_.PathName -match $Malware}| FL Name,PathName;
$ServiceDLL = reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "ServiceDLL" | findstr "$Malware";
$RunKey1 = Get-ItemProperty -Path 'HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run*' |
?{$_ -match $Malware};
$RunKey2 = Get-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*' | ?{$_ -match
$Malware};
$UserProfiles = (gwmi Win32_UserProfile | ? { $_.SID -notmatch 'S-1-5-(18|19|20).*' }); $paths =
$UserProfiles.localpath; $sids = $UserProfiles.sid; for ($counter=0; $counter -lt $UserProfiles.length;
$counter++){$path = $UserProfiles[$counter].localpath; $sid = $UserProfiles[$counter].sid; reg load
hku\$sid $path\ntuser.dat};
$RunKey3 = Get-ItemProperty -Path Registry::HKU\*\SOFTWARE\Microsoft\Windows\CurrentVersion\Run* | ?{$_
-match $Malware};
$Startup = Select-String -Path 'C:\Users\*\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\*' -Pattern $Malware | Select Path;
$Startup2 = Select-String -Path 'C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\*' -
Pattern $Malware | Select Path;
if ($Processes) {echo "Process Found!";$Processes} else {echo "No Running Processes Found."};
if ($Tasks) {echo "Tasks Found!";$Tasks} else {echo "No Tasks Found."};
if ($Services) {echo "Services Found!";$Services} else {echo "No Services Found."};
if ($ServiceDLL) {echo "ServiceDLL Found!";$ServiceDll} else {echo "No Service Dlls Found."};
if ($RunKey1) {echo "Wow6432Node Run Key Found!";$RunKey1} else {echo "No Local Machine Wow6432Node Run
Key Found."};
if ($RunKey2) {echo "Local Machine Run Key Found!";$RunKey2} else {echo "No Local Machine Run Key
Found."};
if ($RunKey3) {echo "User Run Key Found!";$RunKey3} else {echo "No User Run Key Found."};
if ($Startup) {echo "AppData Startup Link Found!";$Startup} else {echo "No AppData Startups Found."};
if ($Startup2) {echo "ProgramData Startup Link Found!";$Startup2} else {echo "No ProgramData Startups
Found."};
```

Scheduled task/job information

```
at (For older OS)
schtasks
schtasks /query /fo LIST /v
schtasks /query /fo LIST /v | findstr "Task To Run:"
schtasks /query /fo LIST /v | findstr "appdata"
schtasks /query /fo LIST /v | select-string "Enabled" -CaseSensitive -Context 10,0 | findstr "exe"
schtasks /query /fo LIST /v | select-string "Enabled" -CaseSensitive -Context 10,0 | findstr "Task"
schtasks /query /fo LIST /v | Select-String "exe" -Context 2,27
gci -path C:\windows\system32\tasks -recurse | Select-String Command | ? {$_.Line -match "EXENAME"} |
FL Line, Filename
gci -path C:\windows\system32\tasks -recurse | where {$_.CreationTime -ge (get-
date).addDays(-1)}|Select-String Command|FL Filename,Line
gci -path C:\windows\system32\tasks -recurse | where {$_.CreationTime -ge (get-date).addDays(-1)} |
where {$_.CreationTime.hour -ge (get-date).hour-2}|Select-String Command|FL Line,Filename
schtasks /query /fo csv /v | ConvertFrom-Csv | ?{"$_.Task To Run" -match "MALICIOUS"}| FL
"Taskname","Task To Run"
schtasks /query /fo csv /v | ConvertFrom-Csv | ?{$_.Taskname -ne "TaskName"} | FL "Taskname","Task To
Run"
wmic job get Name, Owner, DaysOfMonth, DaysOfWeek, ElapsedTime, JobStatus, StartTime, Status
```

Powershell:

```
Get-ScheduledTask
gci -path C:\windows\system32\tasks -recurse | Select-String Command | FL Filename, Line
gci -path C:\windows\system32\tasks -recurse | Select-String "<Command>",Argument | FT
Filename,Command,Line
gci -path C:\windows\system32\tasks -recurse | Select-String Command | ? {$_.Line -match
"MALICIOUSNAME"} | FL Filename, Line
(gci -path C:\windows\system32\tasks -recurse | Select-String "<Command>" | select -exp Line).replace("
<Command>","").trim("</Command>").replace("`"","").trim();
```

Finding ‘shadow’ hidden scheduled tasks

```
gci 'REGISTRY::HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree' -rec -force |
?{$_.Property -notcontains 'SD'}
gci 'REGISTRY::HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree' -rec -force |
Get-ItemProperty | ?{$_.SD.length -lt 100}
```

File hash and location of all scheduled tasks

```
$a=((gci C:\windows\system32\tasks -rec | Select-String "<Command>" | select -exp Line).replace("
<Command>","").trim("</Command>").replace("`"","").trim());foreach ($b in $a){filehash
([System.Environment]::ExpandEnvironmentVariables($b))}
```

From System32 Directory:

```
$a=((gci tasks -rec | Select-String "<Command>" | select -exp Line).replace("<Command>","").trim("
</Command>").replace("`"","").trim());foreach ($b in $a){filehash
([System.Environment]::ExpandEnvironmentVariables($b))}
```

Remediate malicious scheduled tasks

```
schtasks /Delete /TN [taskname] /F
```

Powershell:

```
Unregister-ScheduledTask -TaskName [taskname]
Unregister-ScheduledTask -TaskPath [taskname]
```

AV Logs (evidence of malware quarantine etc)

Note: Special thanks to Phill Moore (https://twitter.com/phillmoore) who has put a lot of effort into this project.

The RULER Project - AV (https://ruler-project.github.io/ruler-project/RULER/av/)

Remote Admin / Remote Monitoring and Management (RMM) Tool Logs (evidence of RMM tool usage)

Note: Special thanks to Phill Moore (https://twitter.com/phillmoore) who has put a lot of effort into this project.

Action1 RMM Usage Evidence

The RULER Project - Action1 (https://ruler-project.github.io/ruler-project/RULER/remote/Action1/)

AmmyAdmin RMM Usage Evidence

The RULER Project - AmmyAdmin (https://ruler-project.github.io/ruler-project/RULER/remote/AmmyAdmin/)

AnyDesk RMM Usage Evidence

The RULER Project - AnyDesk (https://ruler-project.github.io/ruler-project/RULER/remote/AnyDesk/)

Atera RMM Usage Evidence

The RULER Project - Atera (https://ruler-project.github.io/ruler-project/RULER/remote/Atera/)

Citrix GoToMyPC RMM Usage Evidence

The RULER Project - Citrix GoToMyPC (https://ruler-project.github.io/ruler-project/RULER/remote/Citrix%20GoToMyPC/)

Kaseya RMM Usage Evidence

The RULER Project - Kaseya (https://ruler-project.github.io/ruler-project/RULER/remote/Kaseya/)

Level RMM Usage Evidence

The RULER Project - Level (https://ruler-project.github.io/ruler-project/RULER/remote/Level/)

LogMeIn RMM Usage Evidence

The RULER Project - LogMeIn (https://ruler-project.github.io/ruler-project/RULER/remote/LogMeIn/)

RealVNC RMM Usage Evidence

The RULER Project - RealVNC (https://ruler-project.github.io/ruler-project/RULER/remote/RealVNC/)

Splashtop RMM Usage Evidence

The RULER Project - Splashtop (https://ruler-project.github.io/ruler-project/RULER/remote/Splashtop/)

SupRemo RMM Usage Evidence

The RULER Project - SupRemo (https://ruler-project.github.io/ruler-project/RULER/remote/SupRemo/)

SupRemo RMM Usage Evidence

The RULER Project - SupRemo (https://ruler-project.github.io/ruler-project/RULER/remote/SupRemo/)

Connectwise/Screenconnect RMM Usage Evidence

The RULER Project - Connectwise/Screenconnect (https://ruler-project.github.io/ruler-project/RULER/remote/Connectwise_Screenconnect/)

TeamViewer RMM Usage Evidence

The RULER Project - TeamViewer (https://ruler-project.github.io/ruler-project/RULER/remote/TeamViewer/)

TightVNC RMM Usage Evidence

The RULER Project - TightVNC (https://ruler-project.github.io/ruler-project/RULER/remote/TightVNC/)

UltraVNC RMM Usage Evidence

The RULER Project - UltraVNC (https://ruler-project.github.io/ruler-project/RULER/remote/UltraVNC/)

Ultraviewer RMM Usage Evidence

The RULER Project - Ultraviewer (https://ruler-project.github.io/ruler-project/RULER/remote/Ultraviewer/)

Xeox RMM Usage Evidence

The RULER Project - Xeox (https://ruler-project.github.io/ruler-project/RULER/remote/Xeox/)

ZohoAssist RMM Usage Evidence

The RULER Project - ZohoAssist (https://ruler-project.github.io/ruler-project/RULER/remote/ZohoAssist/)

ISO Phishing Execution Evidence

Special Thanks, created from intel by The DFIR Report (https://thedfirreport.com/2022/04/25/quantum-ransomware/)

```
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-VHDMP-Operational';} | FL TimeCreated,
Message
```

MSI Execution Evidence

Special Thanks (https://stackoverflow.com/questions/29937568/how-can-i-find-the-product-guid-of-an-installed-msi-setup/29937569#29937569)

```
Get-WinEvent -FilterHashtable @{LogName='Application';ProviderName='MsiInstaller'} | FL
gci
REGISTRY::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\*\Products\*\
InstallProperties -force
gci
REGISTRY::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\*\Products\*\
InstallProperties -force | Get-ItemProperty | FL PSPath, LocalPackage, InstallDate, InstallSource,
ModifyPath, Publisher, DisplayName
get-wmiobject Win32_Product | Sort-Object -Property Name |Format-Table IdentifyingNumber, Name,
LocalPackage -AutoSize
reg query HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\ /s /f .msi
```

Azure Run Command Evidence

Special Thanks (https://github.com/AtomicGaryBusey/AzureForensics/blob/master/FORENSIC%20ARTIFACTS%20-%20Azure%20Run%20Command%20Extension%20Use.md)

```
gci C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows\*\Status\
gci C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows\*\RuntimeSettings\
gci C:\Packages\Plugins\Microsoft.CPlat.Core.RunCommandWindows\*\
```

Examine:

```
%SystemRoot%\System32\Winevt\Logs\Microsoft-WindowsAzure-Status%4Plugins.evtx
```

UAC Bypass Fodhelper

```
reg query HKCU\Software\Classes\ms-settings\shell\open\command
reg query HKU\{SID}\Software\Classes\ms-settings\shell\open\command
```

Quick overview of persistent locations (AutoRuns) (https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns)

```
autorunsc.exe -accepteula -a * -c -h -v -m > autoruns.csv
autorunsc.exe -accepteula -a * -c -h -v -m -z 'E:\Windows' > autoruns.csv
```

Persistence and Automatic Load/Run Reg Keys

Replace: "reg query" with "Get-ItemProperty -Path HK:" in Powershell*

e.g.: Get-Item -Path HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

User Registry (NTUSER.DAT HIVE) - Commonly located at:

```
C:\Users\[username]
```

*Note: These are setup for querying the current users registry only (HKCU), to query others you will need to load them from the relevant NTUSER.DAT file and then query them.

```
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx"
reg query "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run"
reg query "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run32"
reg query "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\StartupFolder"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run"
reg query "HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows" /f run
reg query "HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows" /f load
reg query "HKCU\Environment" /v UserInitMprLogonScript
reg query "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v RESTART_STICKY_NOTES
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Windows\Scripts"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\RecentDocs"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\RunMRU"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU"
reg query "HKCU\SOFTWARE\AcroDC"
reg query "HKCU\SOFTWARE\Itime"
reg query "HKCU\SOFTWARE\info"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\User Shell Folders"
reg query "HKCU\SOFTWARE\Microsoft\Command Processor"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Applets\RegEdit" /v LastKey
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\OpenSavePidlMRU" /s
reg query "HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Shell
reg query "HKCU\SOFTWARE\Microsoft\Windows\currentversion\run"
reg query "HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Terminal
Server\Install\Software\Microsoft\Microsoft\Windows\CurrentVersion\Run"
reg query "HKCU\Software\Microsoft\Windows NT\CurrentVersion\Terminal
Server\Install\Software\Microsoft\Microsoft\Windows\CurrentVersion\RunOnce"
reg query "HKCU\SOFTWARE\Microsoft\Active Setup\Installed Components\[Random]\StubPath" /s
reg query "HKCU\SOFTWARE\Wow6432Node\Microsoft\Active Setup\Installed Components\[Random]\StubPath" /s
reg query "HKCU\SOFTWARE\Microsoft\Office\[officeversion]\[word/excel/access etc]\Security\AccessVBOM"
reg query "HKCU\SOFTWARE\Microsoft\IEAK\GroupPolicy\PendingGPOs" /s
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Control Panel\CPLs"
reg query "HKCU\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Control Panel\CPLs"
        reg query "HKCU\SOFTWARE\Microsoft\Office\15.0\Excel\Security\AccessVBOM
        reg query "HKCU\SOFTWARE\Microsoft\Office\15.0\Word\Security\AccessVBOM
        reg query "HKCU\SOFTWARE\Microsoft\Office\15.0\Powerpoint\Security\AccessVBOM
        reg query "HKCU\SOFTWARE\Microsoft\Office\15.0\Access\Security\AccessVBOM
```

Local Machine (SOFTWARE HIVE)

```
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices"
reg query "HKLM\SOFTWARE\Policies\Microsoft\Windows\System\Scripts"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows" /f AppInit_DLLs
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Userinit
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options" /s
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit" /s
reg query "HKLM\SOFTWARE\wow6432node\Microsoft\Windows\CurrentVersion\policies\explorer\run"
reg query "HKLM\SOFTWARE\wow6432node\Microsoft\Windows\CurrentVersion\run"
reg query "HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows"
reg query "HKLM\SOFTWARE\Microsoft\Office\[officeversion]\[word/excel/access etc]\Security\AccessVBOM"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run32"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\StartupFolder"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug"
reg query "HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components\[Random]\StubPath" /s
reg query "HKLM\SOFTWARE\Wow6432Node\Microsoft\Active Setup\Installed Components\[Random]\StubPath" /s
reg query "HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Control Panel\CPLs"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Control Panel\CPLs"
        reg query "HKLM\SOFTWARE\Microsoft\Office\15.0\Excel\Security\AccessVBOM
        reg query "HKLM\SOFTWARE\Microsoft\Office\15.0\Word\Security\AccessVBOM
        reg query "HKLM\SOFTWARE\Microsoft\Office\15.0\Powerpoint\Security\AccessVBOM
        reg query "HKLM\SOFTWARE\Microsoft\Office\15.0\Access\Security\AccessVBOM
```

Don’t be afraid to use “findstr” or ‘/f’ to find entries of interest, for example file extensions which may also invoke malicious executables when run, or otherwise.

```
reg query "HKLM\SOFTWARE\Classes" | findstr "file"
reg query "HKLM\SOFTWARE\Classes" /f "file"
reg query HKCR\CLSID\{AB8902B4-09CA-4bb6-B78D-A8F59079A8D5} /s
reg query HKCR\AppID\ /s | findstr "exe"
```

Local Machine (SYSTEM HIVE)

Note: This not only contains services, but also malicious drivers which may run at startup (these are in the form of “.sys” files and are generally loaded from here: \SystemRoot\System32\drivers)

```
reg query "HKLM\SYSTEM\CurrentControlSet\Services\[Random_name]\imagePath"
reg query "HKLM\SYSTEM\CurrentControlSet\Services\ /s /f "*.exe"
reg query "HKLM\SYSTEM\CurrentControlSet\Services" /s /v ImagePath /f "*.exe"
reg query "HKLM\SYSTEM\CurrentControlSet\Services" /s /v ImagePath /f "*.sys"
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager" /v BootExecute
Get-Service -Name "*MALICIOUSSERVICE*"
gwmi win32_service | ? {$_.PathName -match "MALICIOUSSERVICE"}|FL Name,PathName
Get-ItemProperty "HKLM:\SYSTEM\CurrentControlSet\Services\*" | FL DisplayName,ImagePath,ObjectName
gci -Path C:\Windows\system32\drivers -include *.sys -recurse -ea 0 -force | Get-AuthenticodeSignature
gci -Path C:\Windows\system32\drivers -include *.sys -recurse -ea 0 -force | Get-FileHash
```

Note: Some useful commands to show relevant service information

```
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "ImagePath"
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "ServiceDLL"
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "FailureCommand"
```

T1015 Accessibility Features

```
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\sethc.exe" /v
"Debugger"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\utilman.exe"
/v "Debugger"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\AtBroker.exe"
/v "Debugger"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Narrator.exe"
/v "Debugger"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Magnify.exe"
/v "Debugger"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution
Options\DisplaySwitch.exe" /v "Debugger"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\osk.exe" /v
"Debugger"
sfc /VERIFYFILE=C:\Windows\System32\sethc.exe
sfc /VERIFYFILE=C:\Windows\System32\utilman.exe
sfc /VERIFYFILE=C:\Windows\System32\AtBroker.exe
sfc /VERIFYFILE=C:\Windows\System32\Narrator.exe
sfc /VERIFYFILE=C:\Windows\System32\Magnify.exe
sfc /VERIFYFILE=C:\Windows\System32\DisplaySwitch.exe
sfc /VERIFYFILE=C:\Windows\System32\osk.exe
```

T1098 Account Manipulation

```
N/A
```

T1182 AppCert DLLs

```
reg query "HKLM\System\CurrentControlSet\Control\Session Manager" /v AppCertDlls
```

T1103 AppInit DLLs

```
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v Appinit_Dlls
reg query "HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /v Appinit_Dlls
reg query "HKU\{SID}\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v Appinit_Dlls
reg query "HKU\{SID}\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /v Appinit_Dlls
Get-WinEvent -FilterHashtable @{ LogName='System'; Id='11'} | FL TimeCreated,Message
```

T1138 Application Shimming

```
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB"
dir %WINDIR%\AppPatch\custom
dir %WINDIR%\AppPatch\AppPatch64\Custom
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-Kernel-ShimEngine/Operational';}|FL
```

Note: Some other similar methods exist such as abusing the 'Command' value of Windows Telemetry Controller - Special Thanks to TrustedSec (https://www.trustedsec.com/blog/abusing-windows-telemetry-for-persistence/).

Hint: Look for a Command not pointing to "CompatTelRunner.exe" or which has '-cv', '-oobe', or '-fullsync' in the command line.

```
reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\AppCompatFlags\TelemetryController" /s
```

T1197 BITS Jobs

```
bitsadmin /list /allusers /verbose
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-Bits-Client/Operational'; Id='59'} | FL
TimeCreated,Message
ls 'C:\ProgramData\Microsoft\Network\Downloader\qmgr.db'
```

T1067 Bootkit

Note: This exists below the OS in the Master Boot Record or Volume Boot Record. The system must be booted through Advanced Startup Options with a Command Prompt, or through a recovery cd.

```
bootrec /FIXMBR
bootrec /FIXBOOT
```

Extra: If your boot configuration data is missing or contains errors the below can fix this.

```
bootrec /REBUILDBCD
```

If you're thinking of a bootkit more as a rootkit (malicious system drivers) you can go with the below.

General Driver Enumeration

```
gci C:\Windows\*\DriverStore\FileRepository\ -recurse -include *.inf | FL
FullName,LastWriteTime,LastWriteTimeUtc
gci -path C:\Windows\System32\drivers -include *.sys -recurse -ea SilentlyContinue
reg query HKEY_LOCAL_MACHINE\DRIVERS\DriverDatabase\DriverPackages\ /s /f Provider
sc.exe query type=driver state=all
```

Unsigned Drivers

```
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-CodeIntegrity/Operational'; Id='3001'} | FL
TimeCreated,Message
gci -path C:\Windows\System32\drivers -include *.sys -recurse -ea SilentlyContinue | Get-
AuthenticodeSignature | where {$_.status -ne 'Valid'}
```

Previous Unusual Loaded Filter Drivers (Often used by rootkits)

Note: This will likely have false positives, particularly relating to filter drivers which are used by AV products, EDR solutions, or otherwise.

```
$FilterEvents = Get-WinEvent -FilterHashtable @{LogName='System'; ProviderName="Microsoft-Windows-
FilterManager"} | ForEach-Object {
        [PSCustomObject] @{
                TimeCreated = $_.TimeCreated
                MachineName = $_.MachineName
                UserId = $_.UserId
                FilterDriver = $_.Properties[4].Value
        Message = $_.Message
        }
}
echo "Scanning for suspicious filter drivers. Any found will be displayed below:"
$FilterEvents | sort TimeCreated | where-object {$_.FilterDriver -ine "FileInfo" -AND $_.FilterDriver -
ine "WdFilter" -AND $_.FilterDriver -ine "storqosflt" -AND $_.FilterDriver -ine "wcifs" -AND
$_.FilterDriver -ine "CldFlt" -AND $_.FilterDriver -ine "FileCrypt" -AND $_.FilterDriver -ine "luafv" -
AND $_.FilterDriver -ine "npsvctrig" -AND $_.FilterDriver -ine "Wof" -AND $_.FilterDriver -ine
"FileInfo" -AND $_.FilterDriver -ine "bindflt" -AND $_.FilterDriver -ine "PROCMON24" -AND
$_.FilterDriver -ine "FsDepends" -AND $_.FilterDriver -ine "SysmonDrv"}
```

Or, filter by unique drivers:

```
$FilterEvents = Get-WinEvent -FilterHashtable @{LogName='System'; ProviderName="Microsoft-Windows-
FilterManager"} | ForEach-Object {
        [PSCustomObject] @{
                TimeCreated = $_.TimeCreated
                MachineName = $_.MachineName
                UserId = $_.UserId
                FilterDriver = $_.Properties[4].Value
                Message = $_.Message
        }
}
echo "Scanning for suspicious filter drivers. Any found will be displayed below:"
$FilterEvents | sort FilterDriver -Unique | where-object {$_.FilterDriver -ine "FileInfo" -AND
$_.FilterDriver -ine "WdFilter" -AND $_.FilterDriver -ine "storqosflt" -AND $_.FilterDriver -ine
"wcifs" -AND $_.FilterDriver -ine "CldFlt" -AND $_.FilterDriver -ine "FileCrypt" -AND $_.FilterDriver -
ine "luafv" -AND $_.FilterDriver -ine "npsvctrig" -AND $_.FilterDriver -ine "Wof" -AND $_.FilterDriver
-ine "FileInfo" -AND $_.FilterDriver -ine "bindflt" -AND $_.FilterDriver -ine "PROCMON24" -AND
$_.FilterDriver -ine "FsDepends" -AND $_.FilterDriver -ine "SysmonDrv"}
```

Unusual Loaded Filter Drivers (No longer present or filtering registry keys)

```
$FilterEvents = Get-WinEvent -FilterHashtable @{LogName='System'; ProviderName="Microsoft-Windows-
FilterManager"} | ForEach-Object {
        [PSCustomObject] @{
                TimeCreated = $_.TimeCreated
                MachineName = $_.MachineName
                UserId = $_.UserId
                FilterDriver = $_.Properties[4].Value
                Message = $_.Message
        }
}
echo "Scanning for suspicious filter drivers. Any found will be compared against existing services."
echo "Suspicious filter drivers found:"
echo ""
$SuspectDrivers = $($FilterEvents | sort FilterDriver -Unique | where-object {$_.FilterDriver -ine
"FileInfo" -AND $_.FilterDriver -ine "WdFilter" -AND $_.FilterDriver -ine "storqosflt" -AND
$_.FilterDriver -ine "wcifs" -AND $_.FilterDriver -ine "CldFlt" -AND $_.FilterDriver -ine "FileCrypt" -
AND $_.FilterDriver -ine "luafv" -AND $_.FilterDriver -ine "npsvctrig" -AND $_.FilterDriver -ine "Wof"
-AND $_.FilterDriver -ine "FileInfo" -AND $_.FilterDriver -ine "bindflt" -AND $_.FilterDriver -ine
"PROCMON24" -AND $_.FilterDriver -ine "FsDepends" -AND $_.FilterDriver -ine "SysmonDrv"} | select -exp
FilterDriver)
foreach ($driver in $SuspectDrivers){
Write-Warning "Unknown Driver Found - $driver"
}
echo ""
foreach ($driver in $SuspectDrivers){
echo "Checking $driver for relevant service. Any which aren't present may indicate a filter driver
which has since been removed, or an active rootkit filtering service registry keys."
try{gci REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\$driver -force -ErrorAction Stop}Catch{Write-
Warning "$driver NOT FOUND"}
}
```

Safe Boot registry keys

Special Thanks - Didier Stevens (https://blog.didierstevens.com/2007/02/19/restoring-safe-mode-with-a-reg-file/), multiple times (https://blog.didierstevens.com/2006/06/22/save-safeboot/)

Note: These keys specify what services are run in Safe Mode. Sometimes they'll be modified by malware to ensure rootkits can still function in Safe Mode.

```
reg query HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot
reg query HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal /s
reg query HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Network /s
```

Unload malicious filter driver

```
fltmc filters
fltmc volumes
fltmc instances
fltmc unload [filtername]
fltmc detach [filtername] [volumeName] [instanceName]
```

Note: Common legitimate filter drivers include

- WdFilter – Windows Defender Filter
- storqosflt - Storage QoS Filter
- wcifs - Windows Container Isolation File System Filter
- CldFlt - Windows Cloud Files Filter
- FileCrypt - Windows Sandboxing and Encryption Filter
- luafv – LUA File Virtualization Filter (UAC)
- npsvctrig – Named Pipe Service Trigger Provider Filter
- Wof – Windows Overlay Filter
- FileInfo – FileInfo Filter (SuperFetch)
- bindflt - Windows Bind Filter system driver
- FsDepends - File System Dependency Minifilter
- PROCMON24 - Procmon Process Monitor Driver

T1176 Browser Extensions

Chrome

```
Get-ChildItem -path "C:\Users\*\AppData\Local\Google\Chrome\User Data\Default\Extensions" -recurse -
erroraction SilentlyContinue
Get-ChildItem -path 'C:\Users\*\AppData\Local\Google\Chrome\User Data\Default\Extensions' -recurse -
erroraction SilentlyContinue -include manifest.json | cat
reg query "HKLM\Software\Google\Chrome\Extensions" /s
reg query "HKLM\Software\Wow6432Node\Google\Chrome\Extensions" /s
```

Firefox

```
Get-ChildItem -path "C:\Users\*\AppData\Roaming\Mozilla\Firefox\Profiles\*\extensions" -recurse -
erroraction SilentlyContinue
Get-ChildItem -path "C:\Program Files\Mozilla Firefox\plugins\" -recurse -erroraction SilentlyContinue
Get-ChildItem -path registry::HKLM\SOFTWARE\Mozilla\*\extensions
```

Edge

```
Get-ChildItem -Path C:\Users\*\AppData\Local\Packages\ -recurse -erroraction SilentlyContinue
```

Internet Explorer

```
Get-ChildItem -path "C:\Program Files\Internet Explorer\Plugins\" -recurse -erroraction
SilentlyContinue
reg query 'HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects'
reg query 'HKLM\Software\Microsoft\Internet Explorer\Toolbar'
reg query 'HKLM\Software\Microsoft\Internet Explorer\URLSearchHooks'
reg query 'HKLM\Software\Microsoft\Internet Explorer\Explorer Bars'
reg query 'HKU\{SID}\Software\Microsoft\Internet Explorer\Explorer Bars'
reg query 'HKLM\SOFTWARE\Microsoft\Internet Explorer\Extensions'
```

T1109 Component Firmware

Note: This is incredibly rare, and doesn't have an easy detection/remediation mechanism. Using the Windows CheckDisk utility, System File Checker, or Deployment Image Servicing and Management may assist but isn't guaranteed.

```
chkdsk /F
sfc /scannow
dism /Online /Cleanup-Image /ScanHealth
dism /Online /Cleanup-Image /RestoreHealth
dism /Online /Cleanup-Image /StartComponentCleanup /ResetBase
```

T1122 Component Object Model (COM) Hijacking

Note: This involves replacing legitimate components with malicious ones, and as such the legitimate components will likely no longer function. If you have a detection based on DLLHost.exe with /Processid:{xyz}, you can match xyz with the CLSID (COM Class Object) or AppID mentioned below to check for any malicious EXE or DLL.

```
HKLM\SOFTWARE\Classes\WOW6432Node\CLSID\{abc} /v AppID /t REG_SZ /d {xyz}
HKLM\SOFTWARE\Classes\CLSID\{abc} /v AppID /t REG_SZ /d {xyz}
HKLM\SOFTWARE\Classes\AppID\{xyz}
HKLM\SOFTWARE\Classes\WOW6432Node\AppID\{xyz}
HKLM\SOFTWARE\WOW6432Node\Classes\AppID\{xyz}
```

Example analysis:

```
reg query "HKLM\SOFTWARE\Classes\WOW6432Node\CLSID" /s /f "{973D20D7-562D-44B9-B70B-5A0F49CCDF3F}"
reg query "HKLM\SOFTWARE\Classes\WOW6432Node\CLSID\{178167bc-4ee3-403e-8430-a6434162db17}" /s
reg query "HKLM\SOFTWARE\Classes\AppID\{973D20D7-562D-44B9-B70B-5A0F49CCDF3F}"
```

Queries:

```
reg query HKLM\SOFTWARE\Classes\CLSID\ /s /f ".dll"
reg query HKLM\SOFTWARE\Classes\CLSID\ /s /f ".exe"
reg query HKLM\SOFTWARE\Classes\AppID\ /s /f DllSurrogate
gci -path REGISTRY::HKLM\SOFTWARE\Classes\*\shell\open\command
reg query HKU\{SID}\SOFTWARE\Classes\CLSID\ /s /f ".dll"
reg query HKU\{SID}\SOFTWARE\Classes\CLSID\ /s /f ".exe"
gci 'REGISTRY::HKU\*\Software\Classes\CLSID\*\TreatAs'
gci 'REGISTRY::HKU\*\Software\Classes\Scripting.Dictionary'
gci "REGISTRY::HKU\*\SOFTWARE\Classes\CLSID\*\LocalServer32" -ea 0
gci "REGISTRY::HKU\*\SOFTWARE\Classes\CLSID\*\InprocServer32" -ea 0
gci "REGISTRY::HKU\*\SOFTWARE\Classes\CLSID\*\InprocHandler*" -ea 0
gci "REGISTRY::HKU\*\SOFTWARE\Classes\CLSID\*\*Server32" -ea 0
gci "REGISTRY::HKU\*\SOFTWARE\Classes\CLSID\*\ScriptletURL" -ea 0
reg query HKU\{SID}\SOFTWARE\Classes\CLSID\ /s /f "ScriptletURL"
```

Get list of all COM Objects

Original by Jeff Atwood (https://stackoverflow.com/questions/660319/where-can-i-find-all-of-the-com-objects-that-can-be-created-in-powershell)

```
gci HKLM:\Software\Classes -ea 0| ? {$_.PSChildName -match '^\w+\.\w+$' -and(gp "$($_.PSPath)\CLSID" -
ea 0)} | select -ExpandProperty PSChildName
```

T1136 Create Account

```
net user
net user /domain
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\SpecialAccounts" /s
```

T1038 DLL Search Order Hijacking

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs"
gci -path C:\Windows\* -include *.dll | Get-AuthenticodeSignature | Where-Object Status -NE "Valid"
gci -path C:\Windows\System32\* -include *.dll | Get-AuthenticodeSignature | Where-Object Status -NE
"Valid"
gps | FL ProcessName, @{l="Modules";e={$_.Modules|Out-String}}
gps | ? {$_.Modules -like '*{DLLNAME}*'} | FL ProcessName, @{l="Modules";e={$_.Modules|Out-String}}
$dll = gps | Where {$_.Modules -like '*{DLLNAME}*' } | Select Modules;$dll.Modules;
(gps).Modules.FileName
(gps).Modules | FL FileName,FileVersionInfo
(gps).Modules.FileName | get-authenticodesignature | ? Status -NE "Valid"
```

T1133 External Remote Services

```
N/A
```

T1044 File System Permissions Weakness

```
Get-WmiObject win32_service | FL name,PathName
get-acl "C:\Program Files (x86)\Google\Update\GoogleUpdate.exe" | FL | findstr "FullControl"
```

T1158 Hidden Files and Directories

```
dir /S /A:H
```

T1179 Hooking

Finding EasyHook Injection

```
tasklist /m EasyHook32.dll;tasklist /m EasyHook64.dll;tasklist /m EasyLoad32.dll;tasklist /m
EasyLoad64.dll;
```

More Material:

- GetHooks (https://github.com/jay/gethooks/releases/tag/1.01)

T1062 Hypervisor

```
N/A
```

T1183 Image File Execution Options Injection

```
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit" /s /f "MonitorProcess"
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options" /s /f
"Debugger"
```

T1037 Logon Scripts

```
reg query "HKU\{SID}\Environment" /v UserInitMprLogonScript
```

T1177 LSASS Driver

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4614';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='3033';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='3063';} | FL TimeCreated,Message
```

T1031 Modify Existing Service

```
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "ImagePath"
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "ServiceDLL"
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "FailureCommand"
Get-ItemProperty REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\*\* -ea 0 | where {($_.ServiceDll -ne
$null)} | foreach {filehash $_.ServiceDll}
Get-ItemProperty REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\*\* -ea 0 | where {($_.ServiceDll -ne
$null)} | select -uniq ServiceDll -ea 0 | foreach {filehash $_.ServiceDll} | select -uniq -exp hash
```

T1128 Netsh Helper DLL

```
reg query HKLM\SOFTWARE\Microsoft\Netsh
```

T1050 New Service

```
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "ImagePath"
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "ServiceDLL"
reg query HKLM\SYSTEM\CurrentControlSet\Services /s /v "FailureCommand"
Get-WmiObject win32_service | FL Name, DisplayName, PathName, State
Get-WinEvent -FilterHashtable @{ LogName='System'; Id='7045';} | FL TimeCreated,Message
```

Note: If not examining the registry directly and looking at services in a 'live' capacity you may encounter 'hidden services' which aren't shown due to a SDDL applied to them. You can find solely these services using the following (Special thanks - Josh Wright (https://gist.github.com/joswr1ght/c5d9773a90a22478309e9e427073fd30))

```
Compare-Object -ReferenceObject (Get-Service | Select-Object -ExpandProperty Name | % { $_ -replace
"_[0-9a-f]{2,8}$" } ) -DifferenceObject (gci -path hklm:\system\currentcontrolset\services | % {
$_.Name -Replace "HKEY_LOCAL_MACHINE\\","HKLM:\" } | ? { Get-ItemProperty -Path "$_" -name objectname -
erroraction 'ignore' } | % { $_.substring(40) }) -PassThru | ?{$_.sideIndicator -eq "=>"}
```

Some common legitimate hidden services are:

```
WUDFRd
WUDFWpdFs
WUDFWpdMtp
```

T1137 Office Application Startup

```
Get-ChildItem -path C:\Users\*\Microsoft\*\STARTUP\*.dotm -force
Get-ChildItem -path C:\Users\*\AppData\Roaming\Microsoft\*\STARTUP\* -force
reg query "HKU\{SID}\Software\Microsoft\Office test\Special\Perf" /s
reg query "HKLM\Software\Microsoft\Office test\Special\Perf" /s
Get-ChildItem -path registry::HKLM\SOFTWARE\Microsoft\Office\*\Addins\*
Get-ChildItem -path registry::HKLM\SOFTWARE\Wow6432node\Microsoft\Office\*\Addins\*
Get-ChildItem -path registry::HKLM\SOFTWARE\Wow6432node\Microsoft\Office\*\Addins\*
Get-ChildItem -path "C:\Users\*\AppData\Roaming\Microsoft\Templates\*" -erroraction SilentlyContinue
Get-ChildItem -path "C:\Users\*\AppData\Roaming\Microsoft\Excel\XLSTART\*" -erroraction
SilentlyContinue
Get-ChildItem -path C:\ -recurse -include Startup -ea 0
ls 'C:\Program Files\Microsoft Office\root\*\XLSTART\*'
ls 'C:\Program Files\Microsoft Office\root\*\STARTUP\*'
reg query HKCU\Software\Microsoft\Office\[Outlook Version]\Outlook\WebView\Inbox
reg query HKCU\Software\Microsoft\Office\[Outlook Version]\Outlook\Security
reg query HKCU\Software\Microsoft\Office\[Outlook Version]\Outlook\Today\UserDefinedUrl
reg query HKCU\Software\Microsoft\Office\[Outlook Version]\Outlook\WebView\Calendar\URL
Get-WinEvent -FilterHashtable @{ LogName='Microsoft Office Alerts'; Id='300';} | FL TimeCreated,Message
```

T1034 Path Interception

```
N/A
```

T1013 Port Monitors

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Print\Monitors" /s /v "Driver"
```

T1504 PowerShell Profile

```
ls C:\Windows\System32\WindowsPowerShell\v1.0\Profile.ps1
ls C:\Windows\System32\WindowsPowerShell\v1.0\Microsoft.*Profile.ps1
ls C:\Windows\System32\WindowsPowerShell\v1.0\Microsoft.*Profile.ps1
gci -path "C:\Users\*\Documents\PowerShell\Profile.ps1"
gci -path "C:\Users\*\Documents\PowerShell\Microsoft.*Profile.ps1"
```

T1108 Redundant Access

```
N/A
```

T1060 Registry Run Keys / Startup Folder

```
reg query "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\Run"
reg query "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\RunOnce"
reg query "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\RunOnceEx"
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\Run"
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce"
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnceEx"
reg query "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders"
reg query "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders"
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders"
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce"
reg query "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce"
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices"
reg query "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\RunServices"
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run"
reg query "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run"
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Userinit
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Shell
reg query "HKU\{SID}\Software\Microsoft\Windows NT\CurrentVersion\Windows"
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager" /v BootExecute
gci -path "C:\Users\*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*" -include
*.lnk,*.url
gci -path "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\*" -include *.lnk,*.url
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-Shell-Core/Operational'; Id='9707'} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-Shell-Core/Operational'; Id='9708'} | FL
TimeCreated,Message
```

T1053 Scheduled Task

```
gci -path C:\windows\system32\tasks | Select-String Command | FT Line, Filename
gci -path C:\windows\system32\tasks -recurse | where {$_.CreationTime -ge (get-date).addDays(-1)} |
Select-String Command | FL Filename,Line
gci -path C:\windows\system32\tasks -recurse | where {$_.CreationTime -ge (get-date).addDays(-1)} |
where {$_.CreationTime.hour -ge (get-date).hour-2}| Select-String Command | FL Line,Filename
gci -path 'registry::HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks\'
gci -path 'registry::HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\'
ls 'C:\Windows\System32\WptsExtensions.dll'
```

Note: thanks to Markus Piéton (https://twitter.com/markus_pieton/status/1189559716453801991) for the WptsExtensions.dll one.

T1180 Screensaver

```
reg query "HKU\{SID}\Control Panel\Desktop" /s /v "ScreenSaveActive"
reg query "HKU\{SID}\Control Panel\Desktop" /s /v "SCRNSAVE.exe"
reg query "HKU\{SID}\Control Panel\Desktop" /s /v "ScreenSaverIsSecure"
```

T1101 Security Support Provider

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Lsa\OSConfig" /v "Security Packages"
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Lsa" /v "Security Packages"
```

T1505 Server Software Component

```
N/A
```

T1058 Service Registry Permissions Weakness

```
get-acl REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\* |FL
get-acl REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\servicename |FL
```

T1023 Shortcut Modification

```
Select-String -Path "C:\Users\*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.lnk" -
Pattern "exe"
Select-String -Path "C:\Users\*\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\*.lnk" -
Pattern "dll"
Select-String -Path "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\*" -Pattern "dll"
Select-String -Path "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\*" -Pattern "exe"
gci -path "C:\Users\" -recurse -include *.lnk -ea SilentlyContinue | Select-String -Pattern "exe" | FL
gci -path "C:\Users\" -recurse -include *.lnk -ea SilentlyContinue | Select-String -Pattern "dll" | FL
```

T1198 SIP and Trust Provider Hijacking

```
reg query "HKLM\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllGetSignedDataMsg" /s /v "Dll"
reg query "HKLM\SOFTWARE\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData" /s /v "Dll"
reg query "HKLM\SOFTWARE\Microsoft\Cryptography\Providers\Trust\FinalPolicy" /s /v "`$DLL"
reg query "HKLM\SOFTWARE\WOW6432Node\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllGetSignedDataMsg" /s /v "Dll"
reg query "HKLM\SOFTWARE\WOW6432Node\Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData" /s /v "Dll"
reg query "HKLM\SOFTWARE\WOW6432Node\Microsoft\Cryptography\Providers\Trust\FinalPolicy" /s /v "`$DLL"
```

T1019 System Firmware

```
reg query HKLM\HARDWARE\DESCRIPTION\System\BIOS
Confirm-SecureBootUEFI
Get-WmiObject win32_bios
```

T1209 Time Providers

```
reg query "HKLM\System\CurrentControlSet\Services\W32Time\TimeProviders" /s /f "Dll"
```

T1078 Valid Accounts

```
net users
net group /domain "Domain Admins"
net users /domain [name]
```

T1100 Web Shell

Note: The presence of files with these values isn't necessarily indicative of a webshell, review output.

```
gci -path "C:\inetpub\wwwroot" -recurse -File -ea SilentlyContinue | Select-String -Pattern "runat" | FL
gci -path "C:\inetpub\wwwroot" -recurse -File -ea SilentlyContinue | Select-String -Pattern "eval" | FL
```

ProxyShell (https://www.thezdi.com/blog/2021/8/17/from-pwn2own-2021-a-new-attack-surface-on-microsoft-exchange-proxyshell) - May reveal evidence of mailbox exfil or Web Shell being dropped:

```
Get-WinEvent -FilterHashtable @{LogName='MSExchange Management';} | ? {$_.Message -match
'MailboxExportRequest'} | FL TimeCreated, Message
Get-WinEvent -FilterHashtable @{LogName='MSExchange Management';} | ? {$_.Message -match 'aspx'} | FL
TimeCreated, Message
```

T1084 Windows Management Instrumentation Event Subscription

Get WMI Namespaces

```
Function Get-WmiNamespace ($Path = 'root')
{
        foreach ($Namespace in (Get-WmiObject -Namespace $Path -Class __Namespace))
        {
                $FullPath = $Path + "/" + $Namespace.Name
                Write-Output $FullPath
                Get-WmiNamespace -Path $FullPath
        }
}
Get-WMINamespace -Recurse
```

Query WMI Persistence

```
Get-WmiObject -Class __FilterToConsumerBinding -Namespace root\subscription
Get-WmiObject -Class __EventFilter -Namespace root\subscription
Get-WmiObject -Class __EventConsumer -Namespace root\subscription
```

T1004 Winlogon Helper DLL

```
reg query "HKU\{SID}\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify" /s
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify" /s
reg query "HKU\{SID}\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v "Userinit"
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v "Userinit"
reg query "HKU\{SID}\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v "Shell"
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v "Shell"
reg query "HKLM\Software\WOW6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon" /s
```

Other - Winsock Helper DLL Persistence

Special Thanks - odzhan (https://modexp.wordpress.com/2019/07/27/process-injection-winsock/)

Special Thanks - Hexacorn (https://www.hexacorn.com/blog/2015/01/13/beyond-good-ol-run-key-part-24/)

```
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-Winsock-WS2HELP/Operational'; Id='1'} | FL
TimeCreated,Message
reg query HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Winsock\Parameters /v Transports
get-itemproperty 'REGISTRY::HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Winsock\Parameters' -
ea 0 | select -exp Transports
get-item 'REGISTRY::HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\*\Parameters\Winsock' -ea 0
get-itemproperty 'REGISTRY::HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\*\Parameters\Winsock'
-ea 0 | select -exp HelperDllName
get-item 'REGISTRY::HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Winsock2\Parameters' -ea 0
```

Check disabled task manager (often from malware)

```
reg query HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\System /v DisableTaskMgr
```

Review Hivelist

```
gp REGISTRY::HKLM\SYSTEM\CurrentControlSet\Control\hivelist | Select *USER*
```

Locate all user registry keys

```
$UserProfiles = Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\*" |
Where {$_.PSChildName -match "S-1-5-21-(\d+-?){4}$" } | Select-Object @{Name="SID"; Expression=
{$_.PSChildName}}, @{Name="UserHive";Expression={"$($_.ProfileImagePath)\ntuser.dat"}}
```

Load all users registry keys from their ntuser.dat file (perform above first)

```
Foreach ($UserProfile in $UserProfiles) {If (($ProfileWasLoaded = Test-Path
Registry::HKEY_USERS\$($UserProfile.SID)) -eq $false) {reg load HKU\$($UserProfile.SID)
$($UserProfile.UserHive) | echo "Successfully loaded: $($UserProfile.UserHive)"}}
```

Query all users run key

```
Foreach ($UserProfile in $UserProfiles) {reg query
HKU\$($UserProfile.SID)\SOFTWARE\Microsoft\Windows\CurrentVersion\Run};
```

Unload all users registry keys

```
Foreach ($UserProfile in $UserProfiles) {reg unload HKU\$($UserProfile.SID)};
```

Remediate Automatic Load/Run Reg Keys

```
reg delete [keyname] /v [ValueName] /f
reg delete [keyname] /f
Foreach ($UserProfile in $UserProfiles) {reg delete
HKU\$($UserProfile.SID)\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce /f}
Foreach ($UserProfile in $UserProfiles) {reg delete
HKU\$($UserProfile.SID)\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /f}
```

Powershell:

```
Remove-ItemProperty -Path "[Path]" -Name "[name]"
```

Check Registry for IE Enhanced Security Modification

```
gci 'REGISTRY::HKU\*\Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap'
gci 'REGISTRY::HKLM\Software\Policies\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap'
```

Check Registry for disabling of UAC (1=UAC Disabled)

```
gci REGISTRY::HKU\*\Software\Microsoft\Windows\CurrentVersion\Policies\System\EnableLUA
gci REGISTRY::HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System\EnableLUA
```

Review Software Keys for malicious entries

```
gci registry::HKLM\Software\*
gci registry::HKU\*\Software\*
```

Scan Registry keys for specified text

```
Get-ChildItem -path HKLM:\ -Recurse -ea SilentlyContinue | where {$_.Name -match 'notepad' -or $_.Name -match 'sql'}
Get-ChildItem -path HKLM:\ -Recurse -ea SilentlyContinue | get-itemproperty | where {$_ -match 'notepad' -or $_ -match 'sql'}
reg query HKLM\SOFTWARE /s /f ".exe"
reg query HKLM\SYSTEM /s /f ".exe"
reg query HKLM\SECURITY /s /f ".exe"
reg query HKLM /s /f ".exe"
```

Persistent file locations of interest

```
%localappdata%\[random]\[random].[4-9 file ext]
%localappdata%\[random]\[random].lnk
%localappdata%\[random]\[random].bat
%appdata%\[random]\[random].[4-9 file ext]
%appdata%\[random]\[random].lnk
%appdata%\[random]\[random].bat
%appdata%\[random]\[random].bat
%SystemRoot%\[random 4 chars starting with digit]
%appdata%\Microsoft\Windows\Start Menu\Programs\Startup\*
"C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\*"
%SystemRoot%\System32\[randomnumber]\
%SystemRoot%\System32\tasks\[randomname]
%SystemRoot%\[randomname]
C:\Users\[username]\appdata\roaming\[random]
C:\Users\[username]\appdata\roaming\[random]
C:\Users\Public\*
```

You can scan these directories for items of interest e.g. unusual exe, dll, bat, lnk etc files with:

```
dir /s /b %localappdata%\*.exe | findstr /e .exe
dir /s /b %appdata%\*.exe | findstr /e .exe
dir /s /b %localappdata%\*.dll | findstr /e .dll
dir /s /b %appdata%\*.dll | findstr /e .dll
dir /s /b %localappdata%\*.bat | findstr /e .bat
dir /s /b "%appdata%\Microsoft\Windows\Start Menu\Programs\Startup\" | findstr /e .lnk
dir /s /b "C:\Users\Public\" | findstr /e .exe
dir /s /b "C:\Users\Public\" | findstr /e .lnk
dir /s /b "C:\Users\Public\" | findstr /e .dll
dir /s /b "C:\Users\Public\" | findstr /e .bat
ls "C:\Users\[User]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup" | findstr /e .lnk
```

Locate any file of interest from cmd.exe using 'where.exe'

```
where.exe /r C:\ *password*
where.exe /r D:\  version.dll /t
```

Locate LNK Files with a particular string (Special thanks to the notorious)

```
Select-String -Path 'C:\Users\[User]\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\*.lnk' -Pattern "powershell" | Select Path
```

Master File Table

The Master File Table is an incredibly important artifact; however, this can only be read or obtained using low level disk reading. This contains an entry for every file or directory on the filesystem including metadata about these files, and may provide evidence on files which have been removed (MFT entries marked as 'free'). More information can be found on Microsoft Docs (https://docs.microsoft.com/en-us/windows/win32/fileio/master-file-table)

Determine Timestomping

Within the Master File Table (Located at the Win root) there are 2 elements, $STANDARD_INFORMATION and $FILE_NAME, both of which have values for a file being created, modified, accessed and written.

These are known as MACB times (Modified, Accessed, Changed, Birth). The $STANDARD_INFORMATION element can be modified from a malicious process, but the $FILE_NAME element is left intact and cannot without some extra trickery.

These discrepancies generally indicate Timestomping with the $FILE_NAME entry being the source of truth. This can be determined by obtaining the MFT (e.g. using a tool such as Rawcopy), and comparing timestamps on the file (e.g. using a tool such as MFTExplorer).

Rawcopy (https://github.com/jschicht/RawCopy)

```
RawCopy.exe /FileNamePath:C:0 /OutputPath:C:\Audit /OutputName:MFT_C.bin
```

MFTExplorer (https://ericzimmerman.github.io/#!index.md)

Enable Date Accessed Timestamps

```
reg add "HKLM\SYSTEM\CurrentControlSet\Control\FileSystem" /v NtfsDisableLastAccessUpdate /d 0 /t
REG_DWORD /f
```

Remove BITSAdmin Persistence

```
bitsadmin /reset /allusers
import-module bitstransfer
Get-BitsTransfer -AllUsers | Remove-BitsTransfer
```

Check system directories for executables not signed as part of an operating system release

```
gci C:\windows\*\*.exe -File -force |get-authenticodesignature|?{$_.IsOSBinary -notmatch 'True'}
```

Locate Possible Trickbot

```
gci -path C:\Users\*\AppData\Roaming\*\Data -recurse -force -ea SilentlyContinue
gci -path C:\Users\*\AppData\Roaming\*\Modules -recurse -force -ea SilentlyContinue
gci -path C:\Users\*\AppData\Local\*\Data -recurse -force -ea SilentlyContinue
gci -path C:\Users\*\AppData\Local\*\Modules -recurse -force -ea SilentlyContinue
gci -path C:\Users\*\AppData\Roaming\*\*\Data -recurse -force -ea SilentlyContinue
gci -path C:\Users\*\AppData\Roaming\*\*\Modules -recurse -force -ea SilentlyContinue
gci -path C:\Users\*\AppData\Local\*\*\Data -recurse -force -ea SilentlyContinue
gci -path C:\Users\*\AppData\Local\*\*\Modules -recurse -force -ea SilentlyContinue
gci -path C:\Windows\System32\config\systemprofile\appdata\roaming -recurse -force -include *.exe
schtasks /query /fo LIST /v | findstr "appdata"
schtasks /query /fo LIST /v | findstr "programdata"
schtasks /query /fo LIST /v | findstr "public"
tasklist /svc | findstr "svchost"
```

Determine if user Trusted a doc/spreadsheet etc and ran a macro

Note: Don’t forget to load in user hives.

```
reg query 'HKU\[SID]\Software\Microsoft\Office\[versionnumber]\Word\Security\Trusted
Documents\TrustRecords';
gci 'REGISTRY::HKU\*\Software\Microsoft\Office\*\*\Security\Trusted Documents\TrustRecords' -ea 0 |
foreach {reg query $_.Name}
```

Note: This will show the file name/location and metadata in Hex. If the last lot of hex is FFFFFF7F then the user enabled the macro.

Check Office Security Settings

```
gci REGISTRY::HKU\*\Software\Microsoft\Office\*\*\Security -rec
gci REGISTRY::HKCU\Software\Microsoft\Office\*\*\Security -rec
```

Check Outlook Temporary Files

```
gci ((gp REGISTRY::HKU\*\Software\Microsoft\Office\[VerNumber]\Outlook\Security\ -ea
0).OutlookSecureTempFolder)
gci (((gp REGISTRY::HKU\*\Software\Microsoft\Office\*\Outlook\Security\ -ea 0)|select -exp
OutlookSecureTempFolder -ea 0))
```

Check MS Office Logs for high risk file names

```
Get-WinEvent -FilterHashtable @{ LogName='OAlerts';} |Where { $_.Message -Match 'invoice' }| FL
TimeCreated, Message
```

Prevent CVE-2017-11882, CVE-2018-0802, CVE-2018-0804, CVE-2018-0805, CVE-2018-0806, CVE-2018-0807 (EQNEDT32.EXE) Exploitation

Note: This is the “Equation Editor” exploit, either patch or mitigate. More information on the below process (https://support.microsoft.com/en-us/help/4055535/how-to-disable-equation-editor-3-0).

64-Bit Windows:

```
reg add "HKLM\SOFTWARE\Microsoft\Office\Common\COM Compatibility\{0002CE02-0000-0000-C000-
000000000046}" /v "Compatibility Flags" /t REG_DWORD /d 0x400 /f
reg delete "HKEY_CLASSES_ROOT\CLSID\{0002CE02-0000-0000-C000-000000000046}"
reg delete "HKLM\SOFTWARE\Classes\Equation.3"
```

32-Bit Windows:

```
reg add "HKLM\SOFTWARE\Wow6432Node\Microsoft\Office\Common\COM Compatibility\{0002CE02-0000-0000-C000-
000000000046}" /v "Compatibility Flags" /t REG_DWORD /d 0x400 /f
reg delete "HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{0002CE02-0000-0000-C000-000000000046}"
reg delete "HKLM\SOFTWARE\Classes\Equation.3" /f
```

Determine if user opened a document

```
gci "REGISTRY::HKU\*\Software\Microsoft\Office\*\Word\Reading Locations\*"
```

Number of Sub-Directories

```
(gci -Path C:\Users\User\ -Recurse -Directory).length
```

Prevent Executable from Running.

Note: Load in hives and add particular SID to prevent users running named files, helps prevent for example your IIS service account from running cmd.exe or powershell.exe

```
reg add "HKU\{SID}\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer" /v DisallowRun /t
REG_DWORD /d "00000001" /f
reg add "HKU\{SID}\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\DisallowRun" /v
malware.exe /t REG_SZ /d "malware.exe" /f
```

Show known file extensions and hidden files (excluding OS hidden files)

```
reg add "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced" /v Hidden /t REG_DWORD
/d "1" /f
reg add "HKU\{SID}\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced" /v HideFileExt /t
REG_DWORD /d "0" /f
Stop-Process -processname explorer
```

Open File Extension (e.g. scripts) with certain application (elevated cmd)

```
FTYPE Custom=Notepad.exe "%1"
ASSOC .wsf=Custom
```

Disable Command Prompt

```
reg add "HKCU\SOFTWARE\Microsoft\Windows\System" /v DisableCMD /t REG_DWORD /d 0 /f
```

Locate Possible DLL Search Order Hijacking (https://attack.mitre.org/techniques/T1038/)

Note: A legitimate clean executable can be used to run malicious DLLs based on how the software searches for them.

More information on Microsoft Docs (https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order)

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs"
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SafeDllSearchMode"
```

Search order for desktop applications:

If SafeDllSearchMode is enabled (is by default), the search order is as follows:

- The same directory from which the executable is run.
- The System Directory (Usually C:\Windows\System32).
- The 16-bit System Directory.
- The Windows Directory (Usually C:\Windows).
- The Current Directory (From the process which executed the executable).
- The directories that are listed in the PATH environment variable.

If SafeDllSearchMode is disabled (SafeDllSearchMode has a reg value of 0), the search order is as follows:

- The same directory from which the executable is run.
- The Current Directory (From the process which executed the executable).
- The System Directory (Usually C:\Windows\System32).
- The 16-bit System Directory.
- The Windows Directory (Usually C:\Windows).
- The directories that are listed in the PATH environment variable.

Locate Possible Dll Side Loading (https://attack.mitre.org/techniques/T1073/)

Note: A legitimate clean executable can be used to run malicious DLLs based on issues with a manifest file used by the application to load DLLs.

```
reg query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide\Winners"
```

By placing a malicious DLL in the below locations legitimate binaries may have been used to sideload these malicious DLLs.

- C:\Windows\WinSxS
- C:\Windows\SXS

Unique Sideload DLL hashes (may take some time)

```
(gci -path C:\Windows\WinSxS -recurse -include *.dll|gi -ea SilentlyContinue|filehash).hash|sort -u
```

Unsigned or Invalid Sideload DLLs (there will be a lot)

```
gci -path C:\Windows\WinSxS -recurse -include *.dll | Get-AuthenticodeSignature | Where-Object Status -NE "Valid"
```

Unsigned Sideload DLLs (Less noise)

```
gci -path C:\Windows\WinSxS -recurse -include *.dll | Get-AuthenticodeSignature | Where-Object Status -E "NotSigned"
gci -path C:\Windows\WinSxS -recurse -include *.ocx | Get-AuthenticodeSignature | Where-Object Status -NE "Valid"
```

Hash of Unsigned Sideload DLLs

```
gci -path C:\Windows\WinSxS -recurse -include *.dll | Get-AuthenticodeSignature | Where-Object Status -E "NotSigned" | Select Path | gi -ea SilentlyContinue | filehash | sort -u
gci -path C:\Windows\WinSxS -recurse -include *.ocx | Get-AuthenticodeSignature | Where-Object Status -NE "Valid" | Select Path | gi -ea SilentlyContinue | filehash | sort -u
```

Find files without extensions

```
Get-ChildItem -Path C:\Users\[user]\AppData -Recurse -Exclude *.* -File -Force -ea SilentlyContinue
```

Remediate malicious files

```
rmdir %localappdata%\maliciousdirectory\ /s
del /F %localappdata%\maliciousdirectory\malware.exe
```

Powershell:

```
Remove-Item [C:\Users\Public\*.exe]
Remove-Item -Path [C:\Users\Public\malware.exe] -Force
Get-ChildItem * -Include *.exe -Recurse | Remove-Item
```

Detect Persistent WMI Subscriptions

These will appear as children spawning from wmiprvse.

```
Get-WmiObject -Class __FilterToConsumerBinding -Namespace root\subscription
Get-WmiObject -Class __EventFilter -Namespace root\subscription
Get-WmiObject -Class __EventConsumer -Namespace root\subscription
```

Remediate Persistent WMI Subscriptions

The most important aspect is to locate and remove the CommandLineEventConsumer. This has the malicious command stored within the value 'CommandLineTemplate'. The below example searches for commands that contain 'powershell'.

```
Get-WMIObject -Namespace root\subscription -Class __EventFilter -Filter "Name like '%%[Name]%%'" |
Remove-WmiObject
Get-WMIObject -Namespace root\subscription -Class CommandLineEventConsumer -Filter "CommandLineTemplate
like '%%powershell%%'" | Remove-WmiObject
Get-WMIObject -Namespace root\subscription -Class __FilterToConsumerBinding -Filter "__Path like '%%
[Name]%%'" | Remove-WmiObject
```

Enumerate WMI Namespaces (https://learn-powershell.net/2014/05/09/quick-hits-list-all-available-wmi-namespaces-using-powershell/)

```
Function Get-WmiNamespace ($Path = 'root')
{
        foreach ($Namespace in (Get-WmiObject -Namespace $Path -Class __Namespace))
        {
                $FullPath = $Path + "/" + $Namespace.Name
                Write-Output $FullPath
                Get-WmiNamespace -Path $FullPath
        }
}
Get-WMINamespace -Recurse
```

Mimikatz/Credential Extraction Detection

The below represent registry keys which make it more difficult for Mimikatz to work. Modification of these keys may indicate an attacker trying to execute Mimikatz within an environment if they were set to their more secure state. Always test prior to changing registry keys such as these in a production environment to ensure nothing breaks.

```
HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest
        - “UseLogonCredential” should be 0 to prevent the password in LSASS/WDigest
HKLM\SYSTEM\CurrentControlSet\Control\Lsa
        - “RunAsPPL” should be set to dword:00000001 to enable LSA Protection which prevents non-
protected processes from interacting with LSASS.
        - Mimikatz can remove these flags using a custom driver called mimidriver.
                - This uses the command **!+** and then **!processprotect /remove /process:lsass.exe**
by default so tampering of this registry key can be indicative of Mimikatz activity.
```

The Mimikatz Yara rule (https://github.com/gentilkiwi/mimikatz/blob/master/kiwi_passwords.yar) may also prove useful.

Some techniques may involve loading lsasrv.dll or wdigest.dll to extract credentials and may be caught if this is loaded legitimately using:

```
tasklist /m wdigest.dll
tasklist /m lsasrv.dll
```

You may be able to detect changes to the below registry keys which can be used to load an arbitrary DLL and extract credentials, more information from Adam Chester (https://blog.xpnsec.com/exploring-mimikatz-part-1/)

```
reg query HKLM\SYSTEM\CurrentControlSet\Services\NTDS /v LsaDbExtPt
reg query HKLM\SYSTEM\CurrentControlSet\Services\NTDS\DirectoryServiceExtPt
```

An adversary may also tamper with the number of cached logons a system holds (default of 10).

```
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v CachedLogonsCount
```

Password Filter DLL Credential Harvesting Detection

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Lsa" /s /f Notification
```

Network Provider DLL Credential Harvesting Detection

Note: There will be many legitimate ones so verify first.

```
reg query HKLM\SYSTEM\CurrentControlSet /s /f NetworkProvider
gci HKLM:\SYSTEM\CurrentControlSet\Services\*\NetworkProvider
```

Note: This is 100% the research performed by Grzegorz Tworek (https://github.com/gtworek/PSBits/tree/master/PasswordStealing/NPPSpy), and the below script can be used to automate detection.

```
# The script reads information about network providers (possibly acting as password sniffers) from registry, and displays it
# each entry is checked for binary signature and DLL metadata
# no admin privileges needed

# get providers from registry
$providers = Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\NetworkProvider\Order" -Name ProviderOrder

# iterate through entries
$arrExp=@()
foreach ($prov in ($providers.ProviderOrder -split ','))
{
    $row = New-Object psobject
    $row | Add-Member -Name "Name" -MemberType NoteProperty -Value $prov

    $dllPath = (Get-ItemProperty "HKLM:\SYSTEM\CurrentControlSet\Services\$prov\NetworkProvider" -Name ProviderPath).ProviderPath
    $row | Add-Member -Name "DllPath" -MemberType NoteProperty -Value $dllPath
    $signature = Get-AuthenticodeSignature -FilePath $dllPath
    $certSubject = ""
    if ($signature.Status.value__ -eq 0)  #valid
    {
        $certSubject = $signature.SignerCertificate.Subject
    }
    $row | Add-Member -Name "Signer" -MemberType NoteProperty -Value $certSubject
    $row | Add-Member -Name "Version" -MemberType NoteProperty -Value (Get-Command $dllPath).FileVersionInfo.FileVersion
    $row | Add-Member -Name "Description" -MemberType NoteProperty -Value (Get-Command $dllPath).FileVersionInfo.FileDescription

    $arrExp += $row
}

# Let's display the array
if (Test-Path Variable:PSise)
{
    $arrExp | Out-GridView
}
else
{
    $arrExp | Format-List
}
```

Registry key modification timestamp (by Shaun Hess)

```
param(
[parameter(
ValueFromPipeline=$true,
ValueFromPipelineByPropertyName=$true)]
[Alias("CN","__SERVER","Computer","CNAME")]
[string[]]$ComputerName=$env:ComputerName,
[string]$Key = "HKLM",
[string]$SubKey
)

function Get-RegKeyLastWriteTime {
<#
   .SYNOPSIS
       Retrieves the last write time of the supplied registry key
       .DESCRIPTION
       The Registry data that a hive stores in containers are called cells. A cell
       can hold a key, a value, a security descriptor, a list of subkeys, or a
       list of key values.
       Get-RegKeyLastWriteTime retrieves the LastWriteTime through a pointer to the
       FILETIME structure that receives the time at which the enumerated subkey was
       last written. Values do not contain a LastWriteTime property, but changes to
       child values update the parent keys lpftLastWriteTime.

       The LastWriteTime is updated when a key is created, modified, accessed, or
       deleted.
       .PARAMETER ComputerName
       Computer name to query
       .PARAMETER Key
       Root Key to query
       HKCR - Symbolic link to HKEY_LOCAL_MACHINE \SOFTWARE \Classes.
       HKCU - Symbolic link to a key under HKEY_USERS representing a user's profile
       hive.
       HKLM - Placeholder with no corresponding physical hive. This key contains
       other keys that are hives.
       HKU  - Placeholder that contains the user-profile hives of logged-on
       accounts.
       HKCC - Symbolic link to the key of the current hardware profile
       .PARAMETER SubKey
       Registry Key to query
       .EXAMPLE
       Get-RegKeyLastWriteTime -ComputerName testwks -Key HKLM -SubKey Software
       .EXAMPLE
```

```
        Get-RegKeyLastWriteTime -ComputerName testwks1,testwks2 -SubKey Software
        .EXAMPLE
        Get-RegKeyLastWriteTime -SubKey Software\Microsoft
        .EXAMPLE
        "testwks1","testwks2" | Get-RegKeyLastWriteTime -SubKey Software\Microsoft `
        \Windows\CurrentVersion
        .NOTES
        NAME: Get-RegKeyLastWriteTime
        AUTHOR: Shaun Hess
        VERSION: 1.0
        LASTEDIT: 01JUL2011
        LICENSE: Creative Commons Attribution 3.0 Unported License
        (http://creativecommons.org/licenses/by/3.0/)
        .LINK
        http://www.shaunhess.com
        #>

    [CmdletBinding()]

    param(
    [parameter(
    ValueFromPipeline=$true,
    ValueFromPipelineByPropertyName=$true)]
    [Alias("CN","__SERVER","Computer","CNAME")]
    [string[]]$ComputerName=$env:ComputerName,
    [string]$Key = "HKLM",
    [string]$SubKey
    )

    BEGIN {
     switch ($Key) {
      "HKCR" { $searchKey = 0x80000000} #HK Classes Root
      "HKCU" { $searchKey = 0x80000001} #HK Current User
      "HKLM" { $searchKey = 0x80000002} #HK Local Machine
      "HKU"  { $searchKey = 0x80000003} #HK Users
      "HKCC" { $searchKey = 0x80000005} #HK Current Config
      default {
      "Invalid Key. Use one of the following options:
                           HKCR, HKCU, HKLM, HKU, HKCC"}
     }

     $KEYQUERYVALUE = 0x1
     $KEYREAD = 0x19
     $KEYALLACCESS = 0x3F
    }
```

```
 PROCESS {
  foreach($computer in $ComputerName) {

$sig0 = @'
[DllImport("advapi32.dll", SetLastError = true)]
  public static extern int RegConnectRegistry(
        string lpMachineName,
        int hkey,
        ref int phkResult);
'@
  $type0 = Add-Type -MemberDefinition $sig0 -Name Win32Utils -Namespace RegConnectRegistry -Using
System.Text -PassThru

$sig1 = @'
[DllImport("advapi32.dll", CharSet = CharSet.Auto)]
  public static extern int RegOpenKeyEx(
    int hKey,
    string subKey,
    int ulOptions,
    int samDesired,
    out int hkResult);
'@
  $type1 = Add-Type -MemberDefinition $sig1 -Name Win32Utils `
 -Namespace RegOpenKeyEx -Using System.Text -PassThru

$sig2 = @'
[DllImport("advapi32.dll", EntryPoint = "RegEnumKeyEx")]
extern public static int RegEnumKeyEx(
    int hkey,
    int index,
    StringBuilder lpName,
    ref int lpcbName,
    int reserved,
    int lpClass,
    int lpcbClass,
    out long lpftLastWriteTime);
'@
  $type2 = Add-Type -MemberDefinition $sig2 -Name Win32Utils `
 -Namespace RegEnumKeyEx -Using System.Text -PassThru

$sig3 = @'
[DllImport("advapi32.dll", SetLastError=true)]
public static extern int RegCloseKey(
    int hKey);
'@
```

```
  $type3 = Add-Type -MemberDefinition $sig3 -Name Win32Utils -Namespace RegCloseKey -Using System.Text
-PassThru


  $hKey = new-object int
  $hKeyref = new-object int
  $searchKeyRemote = $type0::RegConnectRegistry($computer, $searchKey,   [ref]$hKey)
  $result = $type1::RegOpenKeyEx($hKey, $SubKey, 0, $KEYREAD,   [ref]$hKeyref)

  #initialize variables
  $builder = New-Object System.Text.StringBuilder 1024
  $index = 0
  $length = [int] 1024
  $time = New-Object Long

  #234 means more info, 0 means success. Either way, keep reading
  while ( 0,234 -contains $type2::RegEnumKeyEx($hKeyref, $index++,       $builder, [ref] $length,
$null, $null, $null, [ref] $time) )
  {
      #create output object
      $o = "" | Select Key, LastWriteTime, ComputerName
   $o.ComputerName = "$computer"
      $o.Key = $builder.ToString()
   # TODO Change to use the time api
      #Write-host ((Get-Date $time).ToUniversalTime())
      $timezone=[TimeZoneInfo]::Local
      $Offset=$timezone.BaseUtcOffset.TotalHours

      $o.LastWriteTime = (Get-Date $time).AddYears(1600).AddHours($Offset)
      $o
      #reinitialize for next time through the loop
      $length = [int] 1024
      $builder = New-Object System.Text.StringBuilder 1024
  }

  $result = $type3::RegCloseKey($hKey);
  write-host $builder
  }
 }
} # End Get-RegKeyLastWriteTime function

#write-host $Key $SubKey
```

```
#Get-RegKeyLastWriteTime -Key $Key -SubKey $SubKey

Get-RegKeyLastWriteTime -SubKey System\CurrentControlSet\
```

NetNTLM Downgrade Attack Detection

```
reg query HKLM\SYSTEM\CurrentControlSet\Control\Lsa /v LMCompatibilityLevel
reg query HKLM\SYSTEM\CurrentControlSet\Control\Lsa /v RestrictSendingNTLMTraffic
reg query HKLM\SYSTEM\CurrentControlSet\Control\Lsa /v NTLMMinClientSec
```

DanderSpritz eventlogedit (https://github.com/fox-it/danderspritz-evtx)

Oauth Access Token Theft Detection in Azure

Inversecos - How to Detect OAuth Access Token Theft in Azure (https://www.inversecos.com/2022/08/how-to-detect-oauth-access-token-theft.html)

SANS FOR509 - Cloud Forensics and Azure

- Poster (https://www.sans.org/posters/enterprise-cloud-forensics-incident-response-poster/)
- CrowdStrike CRT (https://github.com/CrowdStrike/CRT)

Putty Detection

```
reg query HKCU\Software\SimonTatham\PuTTY\Sessions /s
```

Installed Updates

(WMI Quick Fix Engineering)

```
wmic qfe
```

Installed Software/Packages

```
reg query HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\ /s /f DisplayName
reg query HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall\ /s /f DisplayName
wmic product get name,version /format:csv
wmic product get /ALL
dism /online /get-packages
get-WmiObject -Class Win32_Product
get-package
```

Powershell: Full List for all users using uninstall keys in registry

```
$(Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*; Get-
ItemProperty HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*;New-PSDrive -Name HKU -
PSProvider Registry -Root Registry::HKEY_USERS| Out-Null;$UserInstalls += gci -Path HKU: | where
{$_.Name -match 'S-\d-\d+-(\d+-){1,14}\d+$'} | foreach {$_.PSChildName };$(foreach ($User in
$UserInstalls){Get-ItemProperty
HKU:\$User\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*});$UserInstalls = $null;try{Remove-
PSDrive -Name HKU}catch{};)|where {($_.DisplayName -ne $null) -and ($_.Publisher -ne $null)} | Select
DisplayName,DisplayVersion,Publisher,InstallDate,UninstallString |FT
```

Process information

(pslist requires sysinternals pslist.exe):

```
tasklist -v
wmic process list full /format:csv
wmic process get name,parentprocessid,processid /format:csv
wmic process get ExecutablePath,processid /format:csv
wmic process get name,ExecutablePath,processid,parentprocessid /format:csv | findstr /I "appdata"
wmic process where processid=[PID] get parentprocessid
wmic process where processid=[PID] get commandline
wmic process where "commandline is not null and commandline!='" get name,commandline /format:csv
gwmi win32_process -Filter "name like 'powershell.exe'" | select name,processId,commandline|FL
gwmi win32_process | select name,processId,path,commandline|FL
gwmi win32_process |FL ProcessID,ParentProcessID,CommandLine,@{e={$_.GetOwner().User}}
gwmi win32_process | Sort-Object -Property ProcessID | FL
ProcessID,Path,CommandLine,ParentProcessID,@{n="User";e=
{$_.GetOwner().User}},@{n="ParentProcessPath";e={gps -Id $_.ParentProcessID|Select -exp Path}}
pslist
Get-Process -IncludeUserName
```

PowerShell Module to show Process Tree (https://gist.github.com/JPMinty/f4d60adafdfbc12b0e4226a27bf1dcb0)

```
import-module .\Get-ProcessTree.ps1
Get-ProcessTree -Verbose | FT Id, Level, IndentedName, ParentId,Path,CommandLine
```

Current Process execution or module loads from temporary directories

Note: This will likely have some false positives as it's just a wildcard. So in this case using 'temp' can come up in words such as 'ItemProvider'.

```
(gps -Module -ea 0).FileName|Select-String "Appdata","ProgramData","Temp","Users","public"|unique
```

Current Process execution or module loads from temporary directories + hash

```
$A=((gps -Module -ea 0).FileName|Select-String
"Appdata","ProgramData","Temp","Users","public"|sort|unique);foreach ($B in $A) {filehash $B};
$A=((gps).Path|Select-String "Appdata","ProgramData","Temp","Users","public"|sort|unique);foreach ($B
in $A) {filehash $B};
```

Scan for malware with Windows Defender

```
"%ProgramFiles%\Windows Defender\MpCmdRun.exe" -Scan -ScanType 1
"%ProgramFiles%\Windows Defender\MpCmdRun.exe" -Scan -ScanType 2
"%ProgramFiles%\Windows Defender\MpCmdRun.exe" -Scan -ScanType 3 -File C:\Users\
[username]\AppData\Local\Temp
```

Note: Types are as follows

- 1: Quick scan
- 2: Full system scan
- 3: File and directory custom scan

Check Windows Defender for excluded files and default actions

```
reg query "HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions" /s
Get-ChildItem 'HKLM:\SOFTWARE\Microsoft\Windows Defender\Exclusions'
Get-MpPreference | Select Exclusion*
Get-MpPreference | Select *DefaultAction
```

Delete Windows Defender excluded files

```
reg delete "HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths" /v "[RegkeyValue]"
reg delete "HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths"
Remove-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths' -Name "Paths"
```

Check Windows Defender Block/Quarantine Logs

```
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-Windows Defender/Operational';
Data='Severe'} | FL TimeCreated,Message
```

Check and Set Access Control Lists

```
Get-Acl -Path 'HKLM:\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths'|FL
Get-Acl -Path [FileWithRequiredAccess] | Set-Acl -Path 'HKLM:\SOFTWARE\Microsoft\Windows
Defender\Exclusions\Paths'
```

Change ACE for “everyone” on folder and subfiles/folders

Grant everyone full access

```
icacls "C:\{DESIREDFOLDERPATH}" /grant everyone:(CI)(OI)F /T
```

Remove ACE entries for “everyone”

```
icacls "C:\{DESIREDFOLDERPATH}" /remove everyone /T
```

Disable unwanted windows binaries (via Base64 encoding and removal)

Note: This is one method, not the only way.

```
certutil -encode C:\windows\system32\mshta.exe C:\windows\system32\mshta.disabled
Get-Acl -Path C:\windows\system32\mshta.exe | Set-Acl -Path C:\windows\system32\mshta.disabled
takeown /f C:\windows\system32\mshta.exe
icacls C:\windows\system32\mshta.exe /grant administrators:F
rm C:\windows\system32\mshta.exe
```

Enable windows binaries (via Base64 decoding and removal)

```
certutil -decode C:\windows\system32\mshta.disabled C:\windows\system32\mshta.exe
Get-Acl -Path C:\windows\system32\mshta.disabled | Set-Acl -Path C:\windows\system32\mshta.exe
takeown /f C:\windows\system32\mshta.disabled
icacls C:\windows\system32\mshta.disabled /grant administrators:F
rm C:\windows\system32\mshta.disabled
```

Make multiple files visible and remove ‘superhidden’

```
gci C:\{DESIREDFOLDERPATH} -force -recurse -ea 0 | foreach {$_.attributes = 'Normal'};
attrib -s -h C:\{DESIREDFOLDERPATH}\*.*
```

Check Security Descriptor Definition Language (SDDL) and Access Control Entries (ACE) for services

```
sc sdshow [servicename]
$A=get-service;foreach ($service in $A){$service;sc.exe sdshow $service.Name}
$A=get-service;foreach ($service in $A){$service;sc.exe sdshow $service.Name|Select-String "A;*DC"}
$A=get-service;foreach ($service in $A){$service;sc.exe sdshow $service.Name|Select-String "A;*WD"}
$A=get-service;foreach ($service in $A){$service;sc.exe sdshow $service.Name|Select-String "A;*WO"}
```

More information on ACE Strings (https://docs.microsoft.com/en-us/windows/win32/secauthz/ace-strings) and the level of access they can provide, and a breakdown is included below.

Syntax of Security Descriptor String (https://learn.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-string-format):

```
owner (O:), primary group (G:), DACL (D:), and SACL (S:).
```

Syntax of ACE String:

```
ace_type;ace_flags;rights;object_guid;inherit_object_guid;account_sid;(resource_attribute)
```

Note: ACE strings are used in the DACL/SACL components of a SDDL.

Example SDDL:

```
D:(A;;CCLCSWRPWPDTLOCRRC;;;SY)
S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)
```

Example SDDL Breakdown (from above):

```
D: = DACL String
A; = SDDL_ACCESS_ALLOWED (ace_type)
; = Nil (ace_flags)
CCLCSWRPWPDTLOCRRC; = Concatenated rights (rights)
        CC = SDDL_CREATE_CHILD
        LC = SDDL_LIST_CHILDREN
        SW = SDDL_SELF_WRITE
        RP = SDDL_READ_PROPERTY
        WP = SDDL_WRITE_PROPERTY
        DT = SDDL_DELETE_TREE
        LO = SDDL_LIST_OBJECT
        CR = SDDL_CRITICAL
        RC = SDDL_READ_CONTROL
; = Nil (object_guid)
; = Nil (inherit_object_guid)
;SY = SDDL_LOCAL_SYSTEM (account_sid)

S: = SACL String
AU; = SDDL_AUDIT (ace_type)
FA; = SDDL_AUDIT_FAILURE (ace_flags)
CCDCLCSWRPWPDTLOCRSDRCWDWO; = Concatenated rights (rights)
        CC = SDDL_CREATE_CHILD
        DC = SDDL_LIST_CHILDREN
        LC = SDDL_SELF_WRITE
        SW = SDDL_READ_PROPERTY
        RP = SDDL_WRITE_PROPERTY
        WP = SDDL_DELETE_TREE
        DT = SDDL_LIST_OBJECT
        LO = SDDL_CRITICAL
        CR = SDDL_READ_CONTROL
        SD = SDDL_STANDARD_DELETE
        RC = SDDL_READ_CONTROL
        WD = SDDL_WRITE_DAC
        WO = SDDL_WRITE_OWNER
; = Nil (object_guid)
; = Nil (inherit_object_guid)
;WD = SDDL_EVERYONE (account_sid)
```

Kill “Unstoppable” Service/Process

```
reg add HKLM\SYSTEM\CurrentControlSet\Services\{SERVICENAME}\Parameters /V start /T reg_dword /D 4 /f
sc.exe sdset {SERVICENAME} "D:(A;;CCLCSWRPWPDTLOCRRC;;;SY)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;BA)
(A;;CCLCSWLOCRRC;;;IU)(A;;CCLCSWLOCRRC;;;SU)S:(AU;FA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;WD)"
Get-Service -Name {SERVICENAME} | Set-Service -Status Paused
sc.exe config {SERVICENAME} start= disabled
Get-Service -Name {SERVICENAME} | Set-Service -Status Stopped
tasklist /FI "IMAGENAME eq {SERVICEEXENAME}"
taskkill /F /t /IM "{SERVICEEXENAME}"
```

Obtain hash for all running executables

Issues with spaces in names but supports CMD.exe

```
FOR /F %i IN ('wmic process where "ExecutablePath is not null" get ExecutablePath') DO certutil -
hashfile %i SHA256 | findstr -v : >> output.txt
```

Powershell (Special thanks Lee Holmes)

```
(gps|gi -ea SilentlyContinue|filehash).hash|sort -u
```

My less efficient powershell

```
foreach ($process in Get-WmiObject win32_process | where {$_.ExecutablePath -notlike ""}) {Get-FileHash
$process.ExecutablePath | Format-List}

foreach ($process in Get-WmiObject win32_process | where {$_.ExecutablePath -notlike ""}) {Get-FileHash
$process.ExecutablePath | select Hash -ExpandProperty Hash}

$A = $( foreach ($process in Get-WmiObject win32_process | where {$_.ExecutablePath -notlike ""}) {Get-
FileHash $process.ExecutablePath | select Hash -ExpandProperty Hash}) |Sort-Object| Get-Unique;$A
```

Obtain hash and established network connections for running executables with dns cache

```
Get-NetTCPConnection -State Established | Select RemoteAddress, RemotePort, OwningProcess,
@{n="Path";e={(gps -Id $_.OwningProcess).Path}},@{n="Hash";e={(gps -Id
$_.OwningProcess|gi|filehash).hash}}, @{n="User";e={(gps -Id $_.OwningProcess -
IncludeUserName).UserName}},@{n="DNSCache";e={(Get-DnsClientCache -Data $_.RemoteAddress -ea
0).Entry}}|sort|gu -AS|FT
```

Obtain hash and listening network connections for running executables

```
Get-NetTCPConnection -State LISTEN | Select LocalAddress, LocalPort, OwningProcess, @{n="Path";e={(gps
-Id $_.OwningProcess).Path}},@{n="Hash";e={(gps -Id $_.OwningProcess|gi|filehash).hash}}, @{n="User";e=
{(gps -Id $_.OwningProcess -IncludeUserName).UserName}}|sort|gu -AS|FT
```

Obtain hash and possible tunneled network connections for running executables

```
Get-NetTCPConnection -State ESTABLISHED |? LocalAddress -Like "::1" | Select RemoteAddress, RemotePort,
OwningProcess, @{n="Path";e={(gps -Id $_.OwningProcess).Path}},@{n="Hash";e={(gps -Id
$_.OwningProcess|gi|filehash).hash}}, @{n="User";e={(gps -Id $_.OwningProcess -
IncludeUserName).UserName}},@{n="DNSCache";e={(Get-DnsClientCache -Data
$_.RemoteAddress).Entry}}|sort|gu -AS|FT
Get-NetTCPConnection -State Established |? LocalAddress -Like "127.0.0.1"| Select RemoteAddress,
RemotePort, OwningProcess, @{n="Path";e={(gps -Id $_.OwningProcess).Path}},@{n="Hash";e={(gps -Id
$_.OwningProcess|gi|filehash).hash}}, @{n="User";e={(gps -Id $_.OwningProcess -
IncludeUserName).UserName}},@{n="DNSCache";e={(Get-DnsClientCache -Data
$_.RemoteAddress).Entry}}|sort|gu -AS|FT
Get-NetTCPConnection -State LISTEN |? LocalAddress -Like "127.0.0.1" | Select LocalAddress, LocalPort,
OwningProcess, @{n="Path";e={(gps -Id $_.OwningProcess).Path}},@{n="Hash";e={(gps -Id
$_.OwningProcess|gi|filehash).hash}}, @{n="User";e={(gps -Id $_.OwningProcess -
IncludeUserName).UserName}}|sort|gu -AS|FT
```

Obtain workstation name for tunneled authentication

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='::';} | FL TimeCreated,Message
```

Obtain hash of DLLs currently loaded by processes

```
$A = $(foreach ($dll in gps|select -ExpandProperty modules -ea SilentlyContinue|? FileName -NotLike
"C:\Windows\SYSTEM32\*"){Get-FileHash $dll.FileName| select Hash -ExpandProperty Hash})|Sort-Object|
Get-Unique;$A
(gps).Modules.FileName | sort -uniq | foreach {filehash $_ -ea 0}
```

Obtain processes where binaries file version doesn't match OS Release

```
gps -FileVersionInfo -ea 0|? {$_.ProductVersion -notmatch
$([System.Environment]::OSVersion.Version|Select -exp Build)}
```

Obtain process binary file external names

```
gps -FileVersionInfo -ea 0 | sort -uniq | Select OriginalFilename,InternalName,Filename
gps -module -FileVersionInfo -ea 0 | sort -uniq | Select OriginalFilename,InternalName,Filename
gps -module -FileVersionInfo -ea 0 | sort -uniq | FL *name,*version
```

Obtain processes running which are running a DLL

```
$A=(gps|select -ExpandProperty modules -ea SilentlyContinue | where {$_.ModuleName -Like 'sechost.dll'
-or $_.ModuleName -Like 'ntdll.dll'} | sort -u);if($A[0].Size -ge -1) {foreach ($Module in $A){tasklist
/m $Module.ModuleName}};
gps | FL ProcessName, @{l="Modules";e={$_.Modules|Out-String}}
```

Obtain hash of unsigned or invalid DLLs currently loaded by processes

```
$A=$(foreach ($dll in gps|select -ExpandProperty modules -ea SilentlyContinue){Get-
AuthenticodeSignature $dll.FileName |Where-Object Status -NE "Valid"|Select Path});$B=$(foreach ($dll
in $A){Get-FileHash $dll.Path| select Hash -ExpandProperty Hash})|Sort-Object| Get-Unique;$B
```

Obtain list of unsigned DLLs currently loaded by processes

```
gps | select -exp modules -ea 0 | Select -exp FileName | Get-AuthenticodeSignature|Where-Object Status -NE "Valid"
gps | select -exp modules -ea 0 | Select -exp FileName | Get-AuthenticodeSignature | ? Status -NE "Valid" | FL Path
```

Obtain DLL information ListDLLs (https://docs.microsoft.com/en-us/sysinternals/downloads/listdlls)

```
listdlls [-r] [-v | -u] [processname|pid]
listdlls [-r] [-v] [-d dllname]
```

Unsigned DLLs loaded by processes (Using ListDLLs (https://docs.microsoft.com/en-us/sysinternals/downloads/listdlls))

```
listdlls64.exe -u -accepteula
```

Obtain DLLs in use by processes

```
listdlls.exe -v processname -accepteula
listdlls.exe -v -d dllname.dll -accepteula
listdlls.exe -d dllname.dll -accepteula
listdlls.exe -v PID -accepteula
```

Enable logging of non non-Windows module loads via WDAC code integrity

Note 1: Special thanks to Matt Graeber (https://twitter.com/mattifestation/status/1366435525272481799) for this.

Note 2: This is based off of a Windows Defender Application Control system integrity policy (https://gist.github.com/mgraeber-rc/7b9f4d497d75967afc58209df611508b) which has been converted on an enterprise system.

On an enterprise system enable it by creating a module load audit policy:
https://twitter.com/mattifestation/status/1366435525272481799

```
ConvertFrom-CIPolicy Non_Microsoft_UserMode_Load_Audit.xml
C:\Windows\System32\CodeIntegrity\SIPolicy.p7b
```

Store the converted policy on a Win10 system to be monitored at: Windows\System32\CodeIntegrity\SIPolicy.p7b

Extract Module (DLL, SYS and EXE) information from WDAC Audit Events

```
# Extract relevant properties from 3076 events
# Modified by Jai Minton @CyberRaiju, based from original work by Matt Graeber @mattifestation

# On an enterprise system enable it by creating a module load audit policy:
https://twitter.com/mattifestation/status/1366435525272481799
        # ConvertFrom-CIPolicy Non_Microsoft_UserMode_Load_Audit.xml
C:\Windows\System32\CodeIntegrity\SIPolicy.p7b
# Store the converted policy on a Win10 system to be monitored at:
Windows\System32\CodeIntegrity\SIPolicy.p7b
# If you don't have one available you can use a pre-converted one found [here]
(https://github.com/JPMinty/Misc-Tools/blob/main/Windows-Defender-Application-Control-
WDAC/SIPolicy.p7b)

# More information:
#
https://gist.githubusercontent.com/mattifestation/de140831d47e15370ba35c1877f39082/raw/8db18ab36723cc9e
af9770c2cadafe46460ff80e/3076EventExtractor.ps1
# https://posts.specterops.io/threat-detection-using-windows-defender-application-control-device-guard-
in-audit-mode-602b48cd1c11
# https://github.com/mattifestation/WDACTools

$SigningLevelMapping = @{
[Byte] 0 = 'Unchecked'
[Byte] 1 = 'Unsigned'
[Byte] 2 = 'Enterprise'
[Byte] 3 = 'Custom1'
[Byte] 4 = 'Authenticode'
[Byte] 5 = 'Custom2'
[Byte] 6 = 'Store'
[Byte] 7 = 'Antimalware'
[Byte] 8 = 'Microsoft'
[Byte] 9 = 'Custom4'
[Byte] 0xA = 'Custom5'
[Byte] 0xB = 'DynamicCodegen'
[Byte] 0xC = 'Windows'
[Byte] 0xD = 'WindowsProtectedProcessLight'
[Byte] 0xE = 'WindowsTcb'
[Byte] 0xF = 'Custom6'
}

$CIEvents = Get-WinEvent -FilterHashtable @{ LogName = 'Microsoft-Windows-CodeIntegrity/Operational';
```

```
Id = 3076} | ForEach-Object {
        $ScenarioValue = $_.Properties[16].Value.ToString()
        $Scenario = $ScenarioValue
                switch ($Scenario) {
                '0' { $Scenario = 'Kernel-Mode' }
                '1' { $Scenario = 'User-Mode' }
        }
        [PSCustomObject] @{
                TimeCreated = $_.TimeCreated
                MachineName = $_.MachineName
                UserId = $_.UserId
                FileName = $_.Properties[1].Value
                ProcessName = $_.Properties[3].Value
                CertificateSHA1AuthentiCodeHash =
[BitConverter]::ToString($_.Properties[8].Value).Replace('-', '')
                CertificateSHA256AuthentiCodeHash =
[BitConverter]::ToString($_.Properties[10].Value).Replace('-', '')
                ModuleSHA1Hash = [BitConverter]::ToString($_.Properties[12].Value).Replace('-', '')
                ModuleSHA256Hash = [BitConverter]::ToString($_.Properties[14].Value).Replace('-', '')
                OriginalFileName = $_.Properties[24].Value
                InternalName = $_.Properties[26].Value
                FileDescription = $_.Properties[28].Value
                ProductName = $_.Properties[30].Value
                FileVersion = $_.Properties[31].Value
                SISigningScenario = $Scenario
                RequestedSigningLevel = $SigningLevelMapping[$_.Properties[4].Value]
                ValidatedSigningLevel = $SigningLevelMapping[$_.Properties[5].Value]
                PolicyHash = [BitConverter]::ToString($_.Properties[22].Value).Replace('-', '')
        }
}
$CIEvents
```

Determine handles on a file

```
handle [[-a] [-u] | [-c [handle] [-l] [-y]] | [-s]] [-p [processname]|[pid]] [filename]
handle -a -u -s -p exp
handle windows\system
```

Verify EternalBlue Patch (MS17-010) is installed - Microsoft (https://support.microsoft.com/en-us/help/4023262/how-to-verify-that-ms17-010-is-installed)

Note: This impacts the SMB 1.0 Server Driver, if you don't have the below, then it's not installed. If you do you can use the above to determine patch level.

```
get-item C:\Windows\system32\drivers\srv.sys | FL VersionInfo
get-hotfix -id KB<111111>
```

Obtain TXT records from recently resolved domains

```
foreach ($domains in Get-DnsClientCache){Resolve-DnsName $domains.Entry -Type "TXT"|Select Strings|?
Strings -NotLike ""};
```

Check all Appdata files for unsigned or invalid executables

```
Get-ChildItem -Recurse $env:APPDATA\..\*.exe -ea SilentlyContinue| ForEach-object {Get-
AuthenticodeSignature $_ -ea SilentlyContinue} | Where-Object {$_.status -ine "Valid"}|Select
Status,Path
```

Check for execuables in Local System User Profile and Files

```
Get-ChildItem C:\Windows\*\config\systemprofile -recurse -force -ea 0 -include *.exe, *.dll *.lnk
```

Investigate WMI Usage

Note: Requires Strings (https://docs.microsoft.com/en-us/sysinternals/downloads/strings)

```
strings -q C:\windows\system32\wbem\repository\objects.data
```

Find executables and scripts in Path directories ($env:Path)

```
Get-Command * -Type Application | FT -AutoSize
Get-Command -Name * | FL FileVersionInfo
```

PowerShell Command History

```
Get-History
```

Find files created/written based on date

```
Get-ChildItem [file] | Select-Object CreationTime
Get-ChildItem C:\ -recurse -ea SilentlyContinue -force | where-object { $_.CreationTime.Date -match
"12/25/2014"}
Get-ChildItem C:\ -recurse -ea SilentlyContinue -force | where-object { $_.LastWriteTime -match
"12/25/2014"}
Get-ChildItem C:\ -recurse -ea SilentlyContinue -force | where-object { $_.CreationTime.Hour -gt 2 -and
$_.CreationTime.Hour -lt 15}
```

Check running executables for malware via VirusTotal

Note: VT Has a rate limit for the Public API so this won't work if you are using the Public API. All 1 liners require VTAPIKey to be set as your VirusTotal API key

```
foreach ($process in Get-WmiObject win32_process | where {$_.ExecutablePath -notlike ""}) {Invoke-
RestMethod -Method 'POST' -Uri 'https://www.virustotal.com/vtapi/v2/file/report' -Body @{ resource =
(Get-FileHash $process.ExecutablePath | select Hash -ExpandProperty Hash); apikey = "[VTAPIKey]"}}
```

This query uses a 15 second timeout to ensure only 4 queries are submitted a minute

```
foreach ($process in Get-WmiObject win32_process | where {$_.ExecutablePath -notlike ""}) {Invoke-
RestMethod -Method 'POST' -Uri 'https://www.virustotal.com/vtapi/v2/file/report' -Body @{ resource =
(Get-FileHash $process.ExecutablePath | select Hash -ExpandProperty Hash); apikey = "
[VTAPIKey]"};Start-Sleep -Seconds 15;}
```

This query uses a 15 second timeout to ensure only 4 queries are submitted a minute and only unique hashes are queried

```
$A = $( foreach ($process in Get-WmiObject win32_process | where {$_.ExecutablePath -notlike ""}) {Get-FileHash $process.ExecutablePath | select Hash -ExpandProperty Hash}) |Sort-Object| Get-Unique -AsString; foreach ($process in $A) {Invoke-RestMethod -Method 'POST' -Uri 'https://www.virustotal.com/vtapi/v2/file/report' -Body @{ resource =($process); apikey = "[VTAPIKey]"};Start-Sleep -Seconds 15;}
```

Scan systems for IOA/IOC (Yara)

Loki Scanner (https://github.com/Neo23x0/Loki)

```
loki-upgrader.exe
loki.exe -p [Directory]
```

Crowdresponse Scanner (https://www.crowdstrike.com/resources/community-tools/crowdresponse/)

```
CrowdResponse -v -i config.txt -o out.xml
```

IREC Tactical (https://binalyze.com/download/irec/)

```
IREC.exe --triage-memory
IREC.exe -ad "\\MACHINE\IREC-DIR" --triage-ruleset MyYaraRules --triage-memory
```

Yara (https://github.com/virustotal/yara/releases/latest)

```
yara32.exe -d filename=[file defined in ruleset.yar] [ruleset.yar] [file to scan]
yara32.exe -d filename=[svchost.exe] [ruleset.yar] -r [directory to scan]
yara64.exe yararule.yar -r C:
yara64.exe yararule.yar -r C: -f 2> $null
```

Yara Linux

Note: -s shows matching yara strings.

```
yara rule.yara malware.exe -s
yara rule.yara [Directory] -s
```

For more creation and usage of Yara, refer to PMA Writeup (https://www.jaiminton.com/Tutorials/PracticalMalwareAnalysis)

Use Snort to test a created Snort rule over a pcap

```
snort -A fast --pcap-single=./pcap.pcap -c ./strrat.rules -l /var/log/snort
```

Kill malicious process

```
wmic process where name="malware.exe" call terminate
wmic process where processid=[PID] delete
taskkill /IM malware.exe
taskkill /PID [PID] /T
```

Note: Call terminate allows you to specify an exit status in terms of a signed integer or a quoted negative value. Both methods essentially function the same by calling TerminateProcess.

Dump full process memory

(procdump requires systinternals procdump.exe)

```
procdump -ma [processID]
```

Live Triage of Memory

Shout-out to Matt Graeber, Jared Atkinson and Joe Desimone for the awesome work that has gone into these scripts. Note: Not all tested, appears to work with a standard Meterpreter payload, and by default with Cobalt Strike.

- PowerShellArsenal (https://github.com/JPMinty/PowerShellArsenal)
- Get-InjectedThread (https://gist.github.com/JPMinty/beffcd18d8ec06b73643c2f38cde384d)

Locate Possible Shellcode within process via Injected Thread

```
Import-Module .\Get-InjectedThread.ps1
Get-InjectedThread
```

Obtain Possible Shellcode within process as Hex

```
(Get-InjectedThread|Select -exp Bytes|ForEach-Object ToString X2) -join ''
(Get-InjectedThread|? {$_.ThreadId -match '{PID}'}|Select -exp Bytes|ForEach-Object ToString X2) -join
''
```

Obtain Possible Shellcode within process as Hex

```
(Get-InjectedThread|Select -exp Bytes|ForEach-Object ToString X2) -join '\x'
(Get-InjectedThread|? {$_.ThreadId -match '{PID}'}|Select -exp Bytes|ForEach-Object ToString X2) -join
'\x'
```

Basic Memory Analysis via PowerShellArsenal

```
import-module .\PowerShellArsenal.psd1
Find-ProcessPEs
Get-ProcessStrings
Get-ProcessMemoryInfo -ProcessID {PID}
Get-VirtualMemoryInfo
```

Locate Possible Shellcode Address Space

```
Get-ProcessMemoryInfo {PID} | ? {$_.AllocationProtect -eq "PAGE_EXECUTE_READWRITE"}
```

Find Meterpreter in Process Memory:

Ref: Meterpreter Wiki (https://github.com/rapid7/metasploit-framework/wiki/Meterpreter)

```
Find-ProcessPEs {PID} | ?{$_.ModuleName -eq "metsrv.dll" -OR $_.ModuleName -eq "ext_server_stdapi.dll"
-OR $_.ModuleName -like "ext_server_*.dll"} | FL ProcessID,ModuleName,Imports;
$A=$(gps | Select -exp Id); foreach ($process in $A){Find-ProcessPEs $process | ?{$_.ModuleName -eq
"metsrv.dll"} | FL ProcessID,ModuleName,Imports};
$A=$(gps | Select -exp Id);     foreach ($process in $A){Find-ProcessPEs $process | ?{$_.ModuleName -eq
"metsrv.dll" | FL ProcessID,ModuleName,Imports};
$A=$(gps | Select -exp Id);     foreach ($process in $A){Find-ProcessPEs $process | ?{$_.ModuleName -eq
"metsrv.dll" -OR $_.ModuleName -eq "ext_server_stdapi.dll" -OR $_.ModuleName -like "ext_server_*.dll"}
| FL ProcessID,ModuleName,Imports};
```

Find Cobalt Strike in Process Memory:

```
Find-ProcessPEs {PID} | ?{$_.ModuleName -eq "beacon.dll" -OR $_.ModuleName -eq "beacon x64.dll" -OR
$_.ModuleName -eq "beacon.x64.dll"} | FL ProcessID,ModuleName,Imports;
$A=$(gps | Select -exp Id); foreach ($process in $A){Find-ProcessPEs $process | ?{$_.ModuleName -eq
"beacon.dll"} | FL ProcessID,ModuleName,Imports};
```

Network connections

(tcpvcon requires sysintenals tcpvcon.exe):

```
ipconfig /all
netstat –anob
netstat -ano
Tcpvcon -a
```

Routing table and ARP cache

```
route print
arp -a
Get-NetNeighbor
```

Contents of DNS resolver

(useful for recent web history)

```
ipconfig /displaydns
Get-DnsClientCache | FT -AutoSize
```

Currently connected Access Point name (WiFi)

```
reg query HKLM\system\CurrentControlSet\Services\Dnscache\Parameters\DnsActiveIfs\ /s
netsh wlan show interfaces
```

Previously connected Access Point names (WiFi)

```
netsh wlan show profile
```

Current surrounding Access Point names (WiFi)

```
netsh wlan show network mode=bssid
```

Extended network adapter configuration information

```
reg query HKLM\system\CurrentControlSet\Services\Tcpip\Parameters\ /s
reg query HKLM\system\CurrentControlSet\Services\Tcpip6\Parameters\ /s
```

Enable DNS Logging

```
wevtutil set-log "Microsoft-Windows-DNS-Client/Operational" /enabled:true
```

OR

```
$DNSLogs = 'Microsoft-Windows-DNS-Client/Operational'
$DNSContainer = New-Object System.Diagnostics.Eventing.Reader.EventLogConfiguration $logName
$DNSContainer.IsEnabled=$true
$DNSContainer.SaveChanges()
```

Scan DNS Logs

```
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-DNS-Client/Operational'; Id='3010';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-DNS-Client/Operational'; Id='3020';} | FL
TimeCreated,Message
```

Scan DNS Logs and output unique DNS Queries

```
$events=Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-DNS-Client/Operational';
Id='3020';};
$output = @();
foreach ($Event in $events){
$data = New-Object -TypeName PSObject;
$XML = [xml]$Event.ToXml();
$query=$XML.Event.EventData.Data|?{$_.Name -eq 'QueryName'} | Select -exp InnerText;
$result=$XML.Event.EventData.Data|?{$_.Name -eq 'QueryResults'} | Select -exp InnerText;
$data `
| Add-Member NoteProperty Query "$query" -PassThru `
| Add-Member NoteProperty QueryResults "$result" -PassThru | Out-Null
$output += $data;
}
$output = $output | sort Query | unique -AsString;
$output
```

Hostname to corresponding IPs from list

```
$listofhostnames = cat Hostnames.txt;
foreach ($hostname in $listofhostnames){
try{[System.Net.Dns]::gethostaddresses("$hostname")|FT $hostname, IPAddressToString}catch
{}}
```

T1074 Data Staging

Note: Examples of some known staging directories, lots of false positives likely.

```
gci C:\ProgramData\ -recurse -include .* -ea 0 -force | ?{ $_.PSIsContainer }
gci C:\Windows\Temp -recurse -ea 0 -force | ?{ $_.PSIsContainer }
ls C:\ProgramData\tmp\log.log
ls C:\ProgramData\log.log
ls C:\ProgramData\google\
ls C:\ProgramData\Sun\low
ls env:temp\SMB
ls C:\ProgramData\.rnd
ls C:\inetpub\
```

Latest system activities

(requires Nirsoft's LastActivityView)

```
LastActivityView.exe /shtml "LastActivityView.html"
```

Driver information

```
wmic sysdriver list brief /format:csv
driverquery
driverquery /FO list /v
driverquery /si
wmic sysdriver list full
```

Process and extra information

```
tasklist /m
tasklist /m /fi "pid eq [PID]"
tasklist /svc
wmic process where processid=[PID] get commandline
tasklist /v
```

Hosts file and service>port mapping

```
type %SystemRoot%\System32\drivers\etc\hosts
type %SystemRoot%\System32\drivers\etc\services
```

Recycle Bin Forensics

- Named as $I = Metadata of file (Info)
- Named as $R = The file contents itself (Recovery)
- Located at %SystemDrive%\$Recycle.Bin in win vista and later commonly (C:\$Recycle.Bin)
- Use dir /a via cmd to show recycle bin SID folders and files

DCOM Information + Firewall Rules

```
wmic dcomapp get /all /format:List
netsh advfirewall firewall show rule dir=in name=all | Select-String -Pattern 'dcom' -Context 2,11
```

Service Information

(psservice requires sysinternals psservice.exe):

```
wmic service list full
net start
sc query
wmic loadorder
psservice
```

Stop and disable/delete malicious service

```
net stop [servicename]
sc config [servicename] start= disabled
sc delete [servicename]
```

Disable Internet Explorer

More Information: MS Docs (https://support.microsoft.com/en-us/help/4013567/how-to-disable-internet-explorer-on-windows)

```
dism /online /Disable-Feature /FeatureName:Internet-Explorer-Optional-amd64
```

cmd history

```
doskey /history
```

Linux Subsystem for Windows 10 may have history in a location such as:

```
C:\Users\
[User]\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\hom
e\[user]
```

Files greater than a 10mb

```
FOR /R C:\ %i in (*) do @if %~zi gtr 10000000 echo %i %~zi
```

Temp files greater than 10mb

```
FOR /R C:\Users\[User]\AppData %i in (*) do @if %~zi gtr 10000000 echo %i %~zi
```

Locate process handles (e.g. files open by process)

Note: Requires handles/handles64.exe from sysinternals

```
handle64.exe -p [PID/name] -nobanner
handle64.exe -a -p [PID/name] -nobanner
handle64.exe -a -l -p [PID/name] -nobanner
handle64.exe -a -l -u -p keepass -nobanner
```

Close process handles (e.g. files open by process)

Note: Requires handles/handles64.exe from sysinternals

```
handle64.exe -c [hexhandleref] -p [PID] -nobanner
handle64.exe -c [hexhandleref] -y -p [PID] -nobanner
```

Event logs between a timeframe

This tool is useful for gathering all windows events within a given timeframe: Event Finder2 (https://github.com/BeanBagKing/EventFinder2)

Check audit policies

```
auditpol /get /category:*
```

Set logging on all success/failure events

(WARNING THIS WILL PRODUCE A LOT OF NOISE, TAILOR TO YOUR NEEDS)

```
auditpol /set /category:* /success:enable /failure:enable
```

Enable logging of process creation

```
auditpol /set /subcategory:"Process Creation" /success:enable /failure:enable
```

Scan process creation logs for 'appdata'

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4688';}| ? {$_.Message -match 'appdata'}|FL
TimeCreated, Message
```

Parse process creation logs

```
$ProcessSpawnEvents = Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4688';} | ForEach-Object
{
        [PSCustomObject] @{
                TimeCreated = $_.TimeCreated
                MachineName = $_.MachineName
                UserSid = $_.Properties[0].Value
                UserName = $_.Properties[1].Value
                UserDomainName = $_.Properties[2].Value
                SubjectLogonId = $_.Properties[3].Value
                ProcessId = $_.Properties[4].Value
                ProcessName = $_.Properties[5].Value
                TokenElevationType = $_.Properties[6].Value
                ParentProcessId = $_.Properties[7].Value
                CommandLine = $_.Properties[8].Value
                TargetUserSid = $_.Properties[9].Value
                TargetUserName = $_.Properties[10].Value
                TargetDomainName = $_.Properties[11].Value
                TargetLogonId = $_.Properties[12].Value
                ParentProcessName = $_.Properties[13].Value
                MandatoryLabel = $_.Properties[14].Value
        }
}
$ProcessSpawnEvents
```

Check for Windows Security Logging Bypass

Special thanks to Grzegorz Tworek - 0gtweet (https://twitter.com/0gtweet/status/1182516740955226112)

```
reg query HKLM\System\CurrentControlSet\Control\MiniNt
```

Check group policies

```
gpresult /Z /SCOPE COMPUTER
gpresult /Z /SCOPE USER
gpresult /R /SCOPE COMPUTER
gpresult /R /SCOPE USER
gpresult /r /z
ls C:\Users\[username]\AppData\Local\GroupPolicy\DataStore
ls C:\Windows\system32\GroupPolicy\DataStore
```

Obtain mode settings for ports

```
mode
```

Event Logs for offline analysis

Event logs can be found: %SystemRoot%\System32\winevt\Logs

```
wevtutil epl System [Location]\System.evtx
wevtutil epl Security [Location]\Security.evtx
wevtutil epl Application [Location]\Application.evtx
wevtutil epl "Windows PowerShell" [Location]\Powershell.evtx
```

OR:

```
esentutl.exe /y /vss C:\Windows\System32\winevt\Logs\Security.evtx /d [Location]\Security.evtx
```

Copy all event logs:

```
XCOPY C:\Windows\System32\winevt\Logs [Location] /i
XCOPY C:\WINDOWS\system32\LogFiles\ [Location] /i
```

User Access Logging (UAL) KStrike Parser (https://github.com/brimorlabs/KStrike)

Note: More information can be found here (https://docs.microsoft.com/en-us/windows-server/administration/user-access-logging/manage-user-access-logging). Special thanks to Brimor Labs.

```
KStrike.py SYSTEMNAME\Current.mdb > Current_mdb.txt
```

mdb Files are found at the below:

```
%SystemRoot%\Windows\System32\Logfiles\SUM
```

More information available on the CrowdStrike Blog - Patrick Bennett (https://www.crowdstrike.com/blog/user-access-logging-ual-overview/)

Quickly scan event logs with DeepblueCLI (https://github.com/sans-blue-team/DeepBlueCLI)

```
.\DeepBlue.ps1 .\evtx\psattack-security.evtx | FL
```

Event Tracing for Windows (ETW).

Event tracing is how a Provider (an application that contains event tracing instrumentation) creates items within the Windows Event Log for a consumer. This is how event logs are generated, and is also a way they can be tampered with. More information on this architecture can be found below.

Event Tracing Architecture (https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing)

A great post by Matt Graeber (https://medium.com/palantir/tampering-with-windows-event-tracing-background-offense-and-defense-4be7ac62ac63) goes into some depth on how this works and some common ways of interacting with ETW Traces.

List Running Trace Sessions

```
logman query -ets
```

List Providers That a Trace Session is Subscribed to

```
logman query "EventLog-System" -ets
```

List all ETW Providers

```
logman query providers
reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers\
```

View providers process is sending events to

```
logman query providers -pid {PID}
```

Setup Custom Log Tracing

Special thanks to Spotless (https://twitter.com/spotheplanet) for his crash course (https://www.ired.team/miscellaneous-reversing-forensics/etw-event-tracing-for-windows-101)

Query Providers Available and their keyword values

```
logman query providers
logman query providers Microsoft-Windows-WinHttp
```

Note: Take note of wanted values.

Initiate Tracing Session

```
logman create trace [TRACENAMEHERE] -ets
logman query [TRACENAMEHERE] -ets
```

Update trace with wanted providers

Note: the mask is the combined values wanted. For example if a keyword was 0x1 and another 0x16 and you wanted both you'd use 0x17.

```
logman update [TRACENAMEHERE] -p Microsoft-Windows-WinHttp 0x100000000 -ets
```

Delete Subscription and Providers

```
logman update trace [TRACENAMEHERE] --p Microsoft-Windows-WinHttp 0x100000000 -ets
logman stop [TRACENAMEHERE] -ets
```

Event Log/Tracing Tampering Detection

```
reg query HKLM\SYSTEM\CurrentControlSet\Services\EventLog\ /s /v File
reg query HKLM\SYSTEM\CurrentControlSet\Services\EventLog\ /s /v MaxSize
reg query HKLM\SYSTEM\CurrentControlSet\Services\EventLog\ /s /v Retention
sc.exe query eventlog
gci REGISTRY::HKLM\SYSTEM\CurrentControlSet\Control\WMI\Autologger\ -recurse
reg query HKLM\SYSTEM\CurrentControlSet\control\WMI\AutoLogger\ /s /v enable*
```

Timeline Windows Event Logs.

An easy way to explore Windows event logs is to dump them into a normalized csv format using EvtxExplorer.

EvtxExplorer: (https://ericzimmerman.github.io/#!index.md)

```
EvtxECmd.exe -d "C:\Windows\System32\winevt\Logs" --csv C:\ --csvf AllEvtx.csv
```

From here you can analyse the CSV using Timeline explorer to view relevant information and group by MAPs.

TimelineExplorer: (https://ericzimmerman.github.io/#!index.md)

Super Timeline a host:

This can be done using Plaso (Log2Timeline) (https://plaso.readthedocs.io/en/latest/)

Common IIS logs can often be found in the below locations:

- %SystemDrive%\inetpub\logs\LogFiles
- %SystemRoot%\System32\LogFiles\W3SVC1
- %SystemDrive%\inetpub\logs\LogFiles\W3SVC1
    - Note: replace 1 with the number for your IIS website ID
- %SystemDrive%\Windows\System32\LogFiles\HTTPERR

Common Apache logs can often be found in the below locations:

- /var/log
- /var/log/httpd/access.log
- /var/log/apache/access.log
- /var/log/apache2/access.log
- /var/log/httpd-access.log

Other logs can be found in the below, often using the Event Trace Log (ETL) format:

- C:\Windows\System32\LogFiles
- C:\Windows\Panther

ETL format can be parsed using tracerpt which is included in Windows, some examples below.

```
tracerpt C:\Windows\System32\LogFiles\WMI\Terminal-Services-RPC-Client.etl
tracerpt logfile1.etl logfile2.etl -o logdump.xml -of XML
tracerpt logfile.etl -o logdmp.xml -of XML -lr -summary logdmp.txt -report logrpt.xml
tracerpt logfile1.etl logfile2.etl -o -report
tracerpt logfile.etl counterfile.blg -report logrpt.xml -df schema.xml
tracerpt -rt "NT Kernel Logger" -o logfile.csv -of CSV
```

Software specific logs are often stored in readable formats at any of the following locations.

```
%AppData%\[softwarename] (e.g. C:\Users\[username]\AppData\Roaming\[softwarename]\)
%LocalAppData%\[softwarename] (e.g. C:\Users\[username]\AppData\Local\[softwarename]\)
%programfiles%\[softwarename] (e.g. C:\Program Files\[softwarename]\)
%programfiles(x86)%\[softwarename] (e.g. C:\Program Files (x86)\[softwarename]\)
```

You may also find useful memory crashdumps at the below:

```
C:\Users\[username]\AppData\Local\CrashDumps
C:\Users\[username]\AppData\Local\Microsoft\Windows\WER\
```

Security log information

Note: Logs and their event codes have changed over time. Most of the references here are for Windows Vista and Server 2008 onwards rather than Windows 2000,XP,Server 2003. More information on them may be added in the future if required.

(psloglist requires psloglist.exe from systinternals):

```
wevtutil qe security /f:text
eventquery.vbs /L security
wevtutil qe security /f:text | Select-String -Pattern "Event ID: [EventCode]" -Context 2,20
wevtutil qe security /f:text | Select-String -Pattern "Event ID: [EventCode]" -Context 2,20 | findstr
"Account Name:"
psloglist -s -x security
```

Note: Some suspicious events - “Event log service was stopped”, “Windows File Protection is not active on this system”, “The MS Telnet Service has started successfully”

- Security: 4720 (Account created)
- Security: 4722 (Account enabled)
- Security: 4724 (Password reset)
- Security: 4723 (User changed password)
- Security: 4736 (Account deleted)
- Security: 4781 (Account renamed)
- Security: 4738 (User account change)
- Security: 4688 (A new process has been created)
- Security: 4732 (Account added to a group)
- Security: 4733 (Account removed from a group)
- Security: 1102 (Audit log cleared)
- Security: 4614 (Security System Extension)
- Security: 4672 (Special privileges assigned to new logon)
- Security: 4624 (Account successfully logged on)
- Security: 4698 (Scheduled Task Creation)
- Security: 4702 (Scheduled Task Modified)
- Security: 4699 (Scheduled Task Deleted)
- Security: 4701 (Scheduled Task Disabled)
- Security: 4700 (Scheduled Task Enabled)
- Security: 4697 (Service Installation)
- Security: 4625 (Account failed to log on)
- Security: 4776 (The domain controller attempted to validate credentials for an account)
- Security: 4634 (Account successfully logged off)
- Security: 4740 (A user account was locked out)
- Security: 4767 (A user account was unlocked)
- Security: 4778 (Remote Desktop session reconnected)
- Security: 4779 (Remote desktop session disconnected)
- Security: 4625 (A user account failed to log on)
- Security: 4648 (A logon was attempted using explicit credentials)

- Security: 4768 (A Kerberos authentication ticket (TGT) was requested)
    - 0x6 (The username doesn't exist) - Bad username or not yet replicated to DC
    - 0xC (Start time is later than end time - Restricted workstation)
    - 0x12 (Account locked out, disabled, expired, restricted, or revoked etc)
- Security: 4769 (A Kerberos service ticket was requested)
- Security: 4770 (A Kerberos service ticket was renewed)
- Security: 4771 (Kerberos pre-authentication failed)
    - 0x10 - Smart card logon is being attempted and the proper certificate cannot be located.
    - 0x17 - The user's password has expired.
    - 0x18 - The wrong password was provided.
- Security: Greater than 4720 Eand less than 4764 (Account/group modifications)

Logon type information

- Type: 0 (Used only by System account authentications)
- Type: 2 (Interactive Logon)
    - User is at the keyboard.
- Type: 3 (Network Authentication/SMB Auth Logon)
    - Auth over the network. Note: RDP can fall under this if Network Level Authentication is enabled.
- Type: 4 (Batch Logon)
    - More often than not from a Scheduled Task.
- Type: 5 (Service Logon)
    - More often than not from a Service.
- Type: 7 (Unlock Logon)
    - User is at the keyboard unlocking it after lunch.
- Type: 8 (Network Cleartext Logon)
    - Basically Logon Type 3 but creds are in the clear.
- Type: 9 (New Credentials Logon)
    - More often than not from using ‘RunAs‘ with the ‘/netonly‘ parameter.
- Type: 10 (Terminal/RDP Logon Type)
    - Logon via Terminal Services/RDP.
- Type: 11 (Cached Interactive)
    - Logon when unable to connect to domain (Cached Creds locally).
- Type: 12 (Cached Remote Interactive)
    - Same as RemoteInteractive. This is used for internal auditing.
- Type: 13 (Cached Unlock Logon)
    - Same as Unlock Logon except with cached creds.

Special logon information (4672)

Privilege Name	Description	Notes
SeAssignPrimaryTokenPrivilege	Replace a process-level token	Required to assign the primary token of a process. With this privilege, the user can initiate a process to replace the default token associated with a started subprocess.
SeAuditPrivilege	Generate security audits	With this privilege, the user can add entries to the security log.
SeBackupPrivilege	Back up files and directories	Required to perform backup operations. With this privilege, the user can bypass file and directory, registry, and other persistent object permissions for the purposes of backing up the system. This privilege causes the system to grant all read access control to any file, regardless of the access control list (ACL) specified for the file. Any access request other than read is still evaluated with the ACL.
SeCreateTokenPrivilege	Create a token object	Allows a process to create a token which it can then use to get access to any local resources when the process uses NtCreateToken() or other token-creation APIs. When a process requires this privilege, we recommend using the LocalSystem account (which already includes the privilege), rather than creating a separate user account and assigning this privilege to it.
SeDebugPrivilege	Debug programs	Required to debug and adjust the memory of a process owned by another account.With this privilege, the user can attach a debugger to any process or to the kernel. We recommend that SeDebugPrivilege always be granted to Administrators, and only to Administrators. Developers who are debugging their own applications do not need this user right. Developers who are debugging new system components need this user right. This user right provides complete access to sensitive and critical operating system components.
SeEnableDelegationPrivilege	Enable computer and user accounts to be trusted for delegation	With this privilege, the user can set the Trusted for Delegation setting on a user or computer object.The user or object that is granted this privilege must have write access to the account control flags on the user or computer object.
SeImpersonatePrivilege	Impersonate a client after authentication	With this privilege, the user can impersonate other accounts.
SeLoadDriverPrivilege	Load and unload device drivers	Required to load or unload a device driver.With this privilege, the user can dynamically load and unload device drivers or other code in to kernel mode. This user right does not apply to Plug and Play device drivers.

Privilege Name	Description	Notes
SeRestorePrivilege	Restore files and directories	Required to perform restore operations. This privilege causes the system to grant all write access control to any file, regardless of the ACL specified for the file. Any access request other than write is still evaluated with the ACL. Additionally, this privilege enables you to set any valid user or group SID as the owner of a file. With this privilege, the user can bypass file, directory, registry, and other persistent objects permissions when restoring backed up files and directories and determines which users can set any valid security principal as the owner of an object.
SeSecurityPrivilege	Manage auditing and security log	Required to perform a number of security-related functions, such as controlling and viewing audit events in security event log. With this privilege, the user can specify object access auditing options for individual resources, such as files, Active Directory objects, and registry keys.A user with this privilege can also view and clear the security log.
SeSystemEnvironmentPrivilege	Modify firmware environment values	Required to modify the nonvolatile RAM of systems that use this type of memory to store configuration information.
SeTakeOwnershipPrivilege	Take ownership of files or other objects	Required to take ownership of an object without being granted discretionary access. This privilege allows the owner value to be set only to those values that the holder may legitimately assign as the owner of an object. With this privilege, the user can take ownership of any securable object in the system, including Active Directory objects, files and folders, printers, registry keys, processes, and threads.
SeTcbPrivilege	Act as part of the operating system	This privilege identifies its holder as part of the trusted computer base.This user right allows a process to impersonate any user without authentication. The process can therefore gain access to the same local resources as that user.

System log information:

```
wevtutil qe system /f:text
eventquery.vbs /L system
```

Note: Some useful events -

- System: 7030 (Basic Service Operations)
- System: 7040 (The start type of a service was changed from disabled to auto start)
- System: 7045 (Service Was Installed)
- System: 1056 (DHCP Server Oddities)
- System: 10000 (COM Functionality)
- System: 20001 (Device Driver Installation)
- System: 20002 (Remote Access)
- System: 20003 (Service Installation)

Application log information

Many applications output errors to the Windows Application Event Logs. For example an application crash may generate an event, or an error may generate an event of value. It's worth looking for events with a source relating to a known vulnerable component that may have been exploited. For example the Australian Cyber Security Centre (https://www.cyber.gov.au/acsc/view-all-content/advisories/advisory-2020-004-remote-code-execution-vulnerability-being-actively-exploited-vulnerable-versions-teleriktelerik-ui-sophisticated-actors) makes special note in one of their reports for the following event.

- Event ID: 1309
- Source: ASP.NET [version_number]

In particular instances of this event with reference to Telerik.Web.UI.IAsyncUploadConfiguration, one can help to identify successful exploitation of a vulnerable Telerik instance.

Another example is looking at successful MsiInstaller events, given malicious MSI files are all too common. Some examples of viewing these in PowerShell is given below.

```
Get-WinEvent -FilterHashtable @{LogName='Application';ProviderName='MsiInstaller'} | FL
Get-WinEvent -FilterHashtable @{LogName='Application';ProviderName='MsiInstaller';Id='1042'} | FL
Get-WinEvent -FilterHashtable @{LogName='Application';ProviderName='MsiInstaller';Id='11707'} | FL
```

Sysmon log information

When installed and running the event log is located at: "Applications and Services Logs/Microsoft/Windows/Sysmon/Operational"

Note: Sysmon and a list of up to date events can be found here (https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon)

Note: A WMI consumer is a management application or script that interacts with the WMI infrastructure. Microsoft Docs - WMI Architecture (https://docs.microsoft.com/en-us/windows/desktop/WmiSdk/wmi-architecture)

- Sysmon: 1 (Process create)
- Sysmon: 2 (File creation time)
- Sysmon: 3 (Network connection detected)
- Sysmon: 4 (Sysmon service state changed)
- Sysmon: 5 (Process terminated)
- Sysmon: 6 (Driver loaded)
- Sysmon: 9 (Image loaded)
- Sysmon: 10 (Process accessed)
- Sysmon: 11 (File created)
- Sysmon: 12 (Registry object added or deleted)
- Sysmon: 13 (Registry value set)
- Sysmon: 14 (Registry object renamed)
- Sysmon: 15 (File stream created)
- Sysmon: 16 (Sysmon configuration changed)
- Sysmon: 17 (Named pipe created)
- Sysmon: 18 (Named pipe connected)
- Sysmon: 19 (WMI filter)
- Sysmon: 20 (WMI consumer)
- Sysmon: 21 (WMI consumer filter)
- Sysmon: 22 (DNS Query)
- Sysmon: 23 (File Delete)
- Sysmon: 24 (Clipboard Changed)
- Sysmon: 25 (Process Tampering)
- Sysmon: 26 (File Delete)

Review Sysmon Logs 1-liner (replace ID value as required from above)

```
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-Sysmon/Operational';ID='1'} | FL
TimeCreated, Message
```

Parse Process Creation Events and Display Data

```
$events=Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-Sysmon/Operational';ID='1'};
$output = @();
foreach ($Event in $events){
$data = New-Object -TypeName PSObject;
$XML = [xml]$Event.ToXml();
$RuleName=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleName'} | Select -exp InnerText;
$UtcTime=$XML.Event.EventData.Data|?{$_.Name -eq 'UtcTime'} | Select -exp InnerText;
$ProcessGuid=$XML.Event.EventData.Data|?{$_.Name -eq 'ProcessGuid'} | Select -exp InnerText;
$ProcessId=$XML.Event.EventData.Data|?{$_.Name -eq 'ProcessId'} | Select -exp InnerText;
$Image=$XML.Event.EventData.Data|?{$_.Name -eq 'Image'} | Select -exp InnerText;
$FileVersion=$XML.Event.EventData.Data|?{$_.Name -eq 'FileVersion'} | Select -exp InnerText;
$Description=$XML.Event.EventData.Data|?{$_.Name -eq 'Description'} | Select -exp InnerText;
$Product=$XML.Event.EventData.Data|?{$_.Name -eq 'Product'} | Select -exp InnerText;
$Company=$XML.Event.EventData.Data|?{$_.Name -eq 'Company'} | Select -exp InnerText;
$OriginalFileName=$XML.Event.EventData.Data|?{$_.Name -eq 'OriginalFileName'} | Select -exp InnerText;
$CommandLine=$XML.Event.EventData.Data|?{$_.Name -eq 'CommandLine'} | Select -exp InnerText;
$CurrentDirectory=$XML.Event.EventData.Data|?{$_.Name -eq 'CurrentDirectory'} | Select -exp InnerText;
$User=$XML.Event.EventData.Data|?{$_.Name -eq 'User'} | Select -exp InnerText;
$LogonGuid=$XML.Event.EventData.Data|?{$_.Name -eq 'LogonGuid'} | Select -exp InnerText;
$LogonId=$XML.Event.EventData.Data|?{$_.Name -eq 'LogonId'} | Select -exp InnerText;
$TerminalSessionId=$XML.Event.EventData.Data|?{$_.Name -eq 'TerminalSessionId'} | Select -exp
InnerText;
$IntegrityLevel=$XML.Event.EventData.Data|?{$_.Name -eq 'IntegrityLevel'} | Select -exp InnerText;
$Hashes=$XML.Event.EventData.Data|?{$_.Name -eq 'Hashes'} | Select -exp InnerText;
$ParentProcessGuid=$XML.Event.EventData.Data|?{$_.Name -eq 'ParentProcessGuid'} | Select -exp
InnerText;
$ParentProcessId=$XML.Event.EventData.Data|?{$_.Name -eq 'ParentProcessId'} | Select -exp InnerText;
$ParentImage=$XML.Event.EventData.Data|?{$_.Name -eq 'ParentImage'} | Select -exp InnerText;
$ParentCommandLine=$XML.Event.EventData.Data|?{$_.Name -eq 'ParentCommandLine'} | Select -exp
InnerText;
$ParentUser=$XML.Event.EventData.Data|?{$_.Name -eq 'ParentUser'} | Select -exp InnerText;
$data `
| Add-Member NoteProperty RuleName "$RuleName" -PassThru `
| Add-Member NoteProperty UtcTime "$UtcTime" -PassThru `
| Add-Member NoteProperty ProcessGuid "$ProcessGuid" -PassThru `
| Add-Member NoteProperty ProcessId "$ProcessId" -PassThru `
| Add-Member NoteProperty Image "$Image" -PassThru `
| Add-Member NoteProperty FileVersion "$FileVersion" -PassThru `
| Add-Member NoteProperty Description "$Description" -PassThru `
| Add-Member NoteProperty Product "$Product" -PassThru `
| Add-Member NoteProperty Company "$Company" -PassThru `
| Add-Member NoteProperty OriginalFileName "$OriginalFileName" -PassThru `
```

```
    | Add-Member NoteProperty CommandLine "$CommandLine" -PassThru `
    | Add-Member NoteProperty CurrentDirectory "$CurrentDirectory" -PassThru `
    | Add-Member NoteProperty User "$User" -PassThru `
    | Add-Member NoteProperty LogonGuid "$LogonGuid" -PassThru `
    | Add-Member NoteProperty LogonId "$LogonId" -PassThru `
    | Add-Member NoteProperty TerminalSessionId "$TerminalSessionId" -PassThru `
    | Add-Member NoteProperty IntegrityLevel "$IntegrityLevel" -PassThru `
    | Add-Member NoteProperty Hashes "$Hashes" -PassThru `
    | Add-Member NoteProperty ParentProcessGuid "$ParentProcessGuid" -PassThru `
    | Add-Member NoteProperty ParentProcessId "$ParentProcessId" -PassThru `
    | Add-Member NoteProperty ParentImage "$ParentImage" -PassThru `
    | Add-Member NoteProperty ParentCommandLine "$ParentCommandLine" -PassThru `
    | Add-Member NoteProperty ParentUser "$ParentUser" -PassThru | Out-Null
    $output += $data;
    }
    $output = $output | sort UtcTime | unique -AsString;
    $output
```

Parse DNS Query Events and Display Data

```
$events=Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-Sysmon/Operational';ID='22'};
$output = @();
foreach ($Event in $events){
$data = New-Object -TypeName PSObject;
$XML = [xml]$Event.ToXml();
$RuleName=$XML.Event.EventData.Data|?{$_.Name -eq 'RuleName'} | Select -exp InnerText;
$UtcTime=$XML.Event.EventData.Data|?{$_.Name -eq 'UtcTime'} | Select -exp InnerText;
$ProcessGuid=$XML.Event.EventData.Data|?{$_.Name -eq 'ProcessGuid'} | Select -exp InnerText;
$ProcessId=$XML.Event.EventData.Data|?{$_.Name -eq 'ProcessId'} | Select -exp InnerText;
$QueryName=$XML.Event.EventData.Data|?{$_.Name -eq 'QueryName'} | Select -exp InnerText;
$QueryResults=$XML.Event.EventData.Data|?{$_.Name -eq 'QueryResults'} | Select -exp InnerText;
$Image=$XML.Event.EventData.Data|?{$_.Name -eq 'Image'} | Select -exp InnerText;
$User=$XML.Event.EventData.Data|?{$_.Name -eq 'User'} | Select -exp InnerText;
$data `
| Add-Member NoteProperty RuleName "$RuleName" -PassThru `
| Add-Member NoteProperty UtcTime "$UtcTime" -PassThru `
| Add-Member NoteProperty ProcessGuid "$ProcessGuid" -PassThru `
| Add-Member NoteProperty ProcessId "$ProcessId" -PassThru `
| Add-Member NoteProperty QueryName "$QueryName" -PassThru `
| Add-Member NoteProperty QueryResults "$QueryResults" -PassThru `
| Add-Member NoteProperty Image "$Image" -PassThru `
| Add-Member NoteProperty User "$User" -PassThru | Out-Null
$output += $data;
}
$output = $output | sort UtcTime | unique -AsString;
$output
```

Active Directory Investigation

Note: Live information can be found using DSQuery (https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/cc732952(v=ws.11)) or Netdom (https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/cc772217(v=ws.11)).

```
dsquery computer
dsquery user
dsquery contact
dsquery domainroot -inactive 4
dsquery group
dsquery ou
dsquery site
dsquery server
dsquery quota
dsquery *
        - dsquery * -limit 999999999
netdom query fsmo
netdom query trust
netdom query pdc
netdom query DC
netdom query server
netdom query workstation
netdom query OU
```

NT Directory Services Directory Information Tree File (ntds.dit)

Active Directory Database file containing all schema, domain, configuration information (e.g. users, IP, computers, domain trusts etc)

- %SystemRoot%\NTDS\ntds.dit
- %SystemRoot%\System32\ntds.dit
    - File created only when promoting certain OS to a DC, and seldom used.

Edb.log

10MB transaction log used to store temporary data before it is sent to the ntds.dit database.

- %SystemRoot%\NTDS\Edb.log

Edbxxxxx.log

Additional transaction log files if the main edb.log file gets larger than 10MB without being flushed to ntds.dit.

- %SystemRoot%\NTDS\edbxxxxx.log

Edb.chk

Checkpoint file used to determine how much of the transaction logs have been sent to the ntdis.dit database.

- %SystemRoot%\NTDS\edb.chk

Resx.log/Resx.jrs

Reserved log files in case the hard drive fills up, at which point these files will be used (ideally they should never be used).

- %SystemRoot%\NTDS\res1.log
- %SystemRoot%\NTDS\res2.log

Temp.edb

Temporary file to store information during in progress transactions.

- %SystemRoot%\NTDS\temp.edb

Schema.ini

Initialises the ntds.dit file when the domain controller is created, and is then never used again.

- %SystemRoot%\NTDS\schema.ini

Investigation of ntds.dit

Obtaining this file can be done using any of the following and also requires the SYSTEM hive to decrypt (note: ntdsutil may not work on older AD servers).

(Output will be under C:\Audit)

ntdsutil

```
ntdsutil "activate instance ntds" ifm "create full C:\Audit" quit quit
```

vssadmin

```
vssadmin create shadow /for=C:
mkdir C:\Audit
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy[Number]\Windows\ntds\ntds.dit C:\Audit\ntds.dit
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy[Number]\Windows\System32\config\SYSTEM
C:\Audit\SYSTEM
vssadmin delete shadows /shadow=[ShadowCopyID]
```

Other 'less legitimate' replication methods can be found detailed on the AD Security Blog by Sean Metcalf (https://adsecurity.org/?p=2398#MimikatzDCSync)

- Or by using Invoke-NinjaCopy (https://github.com/clymb3r/PowerShell/blob/master/Invoke-NinjaCopy/Invoke-NinjaCopy.ps1)

Repair the file if required:

```
esentutl /p /o C:\Audit\ntds.dit
```

Analysing this file offline can be done with tactics such as:

- Ropnop - Extract Hashes and Domain Info (https://blog.ropnop.com/extracting-hashes-and-domain-info-from-ntds-dit/)

Origami-PDF (Malicious PDF Analysis)

Github Download (https://github.com/gdelugre/origami)

```
pdfextract malware.pdf
```

More Malicious PDF/Doc Analysis

```
pdfid.py malware.pdf
pdfparser.py malware.pdf
pdfparser.py malware.pdf --object [number] --filter --raw --dump file.[extension]
oledump.py file.[extension]
oledump.py file.[extension] --select [number] --vbadecompress
```

Exiftool (Image Analysis)

```
exiftool malware.jpeg
```

Dump all thumbnails from a jpg image to a folder

Thanks to Phil Harvey (https://exiftool.org/forum/index.php?topic=7227.0)

Note: This may be useful for finding evidence of photo manipulation

```
exiftool -a -b -W FOLDERNAME/%f_%t%-c.%s -preview:all IMAGENAME.jpg
```

RDP Cache images

This can be used to display some fragments of images which a user could see when operating on a server using the Windows RDP. The cache files are located: %USERPROFILE%\AppData\Local\Microsoft\Terminal Server Client\Cache\

These can be parsed using BMC-Tools (https://github.com/ANSSI-FR/bmc-tools)

```
bmc-tools.py -s ./ -d ./output
bmc-tools.py -s ./ -d ./output -o -b
```

RDP (Terminal Services) Activity

```
reg query 'HKU\{SID}\Software\Microsoft\Terminal Server Client' /s
```

RDP (Terminal Services) Configuration

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server" /s
```

Check if Terminal Services Enabled

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fDenyTSConnections
```

Check if one session per user has been modified

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fSingleSessionPerUser
```

Check if port number has been modified

```
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server\Wds\rdpwd\Tds\tcp" /v PortNumber
reg query "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server\WinStations\RDP-Tcp" /v PortNumber
```

Host Firewall information:

```
netsh firewall show config
advfirewall firewall show rule name=all verbose
```

Model of motherboard and hardware information:

```
wmic baseboard get product,manufacturer
wmic desktopmonitor get /all /format:list
wmic baseboard get /all /format:list
wmic bios get /all /format:list
wmic cpu get /all /format:list
```

Monitoring of open files:

```
openfiles /local on
```

Check Bitlocker Encryption

```
manage-bde -status
```

OR Powershell:

```
Get-BitLockerVolume
```

List open files

(this needs to have been enabled first and the PC rebooted, psfiles requires sysinternals psfile.exe)

```
openfiles /query
psfile
```

Display proxy information

```
netsh winhttp show proxy
```

Disconnect open files based on username:

```
openfiles /disconnect /a username
```

Powershell (some with WMI). Note: Namespace is a group of classes belonging to the same management environment. Most important is the CIMV2 child which is the most common.

Powershell Commands

```
help get-wmiobject
```

Service information

```
Get-WmiObject win32_service | select Name, DisplayName, State, PathName
Get-Service
```

View Named Pipes

```
[System.IO.Directory]::GetFiles("\\.\pipe\")
get-childitem \\.\pipe\
dir \\.\pipe\
```

Harden System from ISO Phishing

Special Thanks - Mubix (https://malicious.link/post/2022/blocking-iso-mounting/)

```
reg add "HKEY_CLASSES_ROOT\Windows.IsoFile\shell\mount" /v ProgrammaticAccessOnly /t REG_SZ
reg add "HKEY_CLASSES_ROOT\Windows.VhdFile\shell\mount" /v ProgrammaticAccessOnly /t REG_SZ
```

Harden System from Lateral Movement/privesc

Note: These may inadvertently break communication of devices and should be tested. It may also require a restart.

Disable remote interaction with services

```
reg add "HKLM\SYSTEM\CurrentControlSet\Control" /v DisableRemoteScmEndpoints /t REG_DWORD /d 1 /f
```

Disable remote interaction with scheduled tasks

```
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule" /v DisableRpcOverTcp /t REG_DWORD
/d 1 /f
```

Disable RDP access

```
reg add "HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fDenyTSConnections /t REG_DWORD /d 1
/f
```

Disable DCOM

```
reg add "HKLM\SOFTWARE\Microsoft\Ole" /v EnableDCOM /t REG_SZ /d N /f
```

Disable Admin Shares

```
reg add "HKLM\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters" /v "AutoShareWks" /t REG_DWORD
/d 0 /f
reg add "HKLM\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters" /v "AutoShareServer" /t
REG_DWORD /d 0 /f
```

Disable Printer Spooler Service (PrintNightmare RCE & LPE Mitigation)

Note: Flow chart (https://twitter.com/gentilkiwi/status/1412483747321192451) kindly provided by Benjamin Delpy

```
Stop-Service -Name Spooler -Force
Set-Service -Name Spooler -StartupType Disabled
reg add "HKLM\SYSTEM\CurrentControlSet\Services\Spooler" /v Start /t REG_DWORD /d 4 /f
```

Prevent SYSTEM from writing new print DLL (PrintNightmare RCE & LPE Mitigation)

Special thanks to truesec (https://blog.truesec.com/2021/06/30/fix-for-printnightmare-cve-2021-1675-exploit-to-keep-your-print-servers-running-while-a-patch-is-not-available/)

```
$Path = "C:\Windows\System32\spool\drivers"
$Acl = (Get-Item $Path).GetAccessControl('Access')
$Ar = New-Object  System.Security.AccessControl.FileSystemAccessRule("System", "Modify",
"ContainerInherit, ObjectInherit", "None", "Deny")
$Acl.AddAccessRule($Ar)
Set-Acl $Path $Acl
```

Disable Remote Printing (PrintNightmare RCE mitigation)

```
reg add "HKLM\SOFTWARE\Policies\Microsoft\Windows NT\Printers" /v RegisterSpoolerRemoteRpcEndPoint /t
REG_DWORD /d 2 /f
```

Enable Warning on PointAndPrint and UAC (PrintNightmare LPE mitigation)

```
reg add "HKLM\SOFTWARE\Policies\Microsoft\Windows NT\Printers\PointAndPrint" /v
NoWarningNoElevationOnInstall /t REG_DWORD /d 0 /f
reg add "HKLM\SOFTWARE\Policies\Microsoft\Windows NT\Printers\PointAndPrint" /v
NoWarningNoElevationOnUpdate /t REG_DWORD /d 0 /f
reg add "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System" /v EnableLUA /t REG_DWORD /d 1
/f
```

Deny vulnerable Netlogon connections (Prevent ZeroLogon CVE-2020-1472)

Note: This should be run on a DC or relevant policy applied. It requires the August 11, 2020 update. Full mitigation advice can be found here (https://support.microsoft.com/en-us/help/4557222/how-to-manage-the-changes-in-netlogon-secure-channel-connections-assoc)

```
reg add "HKLM\SYSTEM\CurrentControlSet\Services\Netlogon\Parameters" /v FullSecureChannelProtection /t
REG_DWORD /d 1 /f
```

It should be noted the following System events relate to this and should be reviewed:

- Event IDs 5827 and 5828 in the System event log, if ZeroLogon connections are denied.
- Event IDs 5830 and 5831 in the System event log, if ZeroLogon connections are allowed by “Domain controller: Allow vulnerable Netlogon secure channel connections” group policy.
- Event ID 5829 in the System event log, if ZeroLogon vulnerable Netlogon secure channel connection is allowed.

Rename mshtml.dll (CVE-2021-40444 Mitigation)

Note: This will render any application which leverages mshtml.dll for rendering HTML content unable to do so (including mshta.exe - yay). At this stage the MSHTML (Trident) engine should not be leveraged by many applications and Microsoft recommends future app development not use the MSHTML (Trident) engine (https://techcommunity.microsoft.com/t5/windows-it-pro-blog/internet-explorer-11-desktop-app-retirement-faq/ba-p/2366549). Some examples of what do use it include .chm files and software mentioned here (https://en.wikipedia.org/wiki/Trident_%28layout_engine%29)

- Run cmd.exe as Administrator.

  takeown /F mshtml.dll icacls mshtml.dll /grant administrators:F move mshtml.dll mshtml2.dll cd ../SysWOW64 takeown /F mshtml.dll icacls mshtml.dll /grant administrators:F move mshtml.dll mshtml2.dll

Delete ms-msdt association (CVE-2022-30190/Follina Mitigation)

From MSRC (https://msrc-blog.microsoft.com/2022/05/30/guidance-for-cve-2022-30190-microsoft-support-diagnostic-tool-vulnerability/)

```
reg delete HKEY_CLASSES_ROOT\ms-msdt /f
```

Stop Server Responsible for Inter-process Communication Calls

```
net stop server
```

Delete Admin Shares

Note: This may break some application communication and admin functionality. It may also be temporary as Windows has been known to recreate them. Always test.

- C$ = Default share on systems ‘C’ drive.
- IPC$ = Default Inter-process communication share (used by named pipes)
- ADMIN$ = Default share for remote administration (used by PsExec)

  net share C$ /delete net share IPC$ /delete net share ADMIN$ /delete

Disable Anonymous Access to Named Pipes

```
reg add "HKLM\SYSTEM\CurrentControlSet\Services\LanManServer\Parameters" /v "RestrictNullSessAccess" /t "REG_DWORD" /d 1 /f
```

Notes on named pipes:

- Named pipes are used for communication between processes, this includes a process from a remote system.
- A named pipe can be created by anyone.
- By enabling 'RestrictNullSessAccess' you stop anonymous network logons from accessing named pipes on your system.
- If a process has the 'SeImpersonatePrivilege' (or equivalent) privilege assigned and creates a named pipe, it may be able to impersonate the user context of anyone who connects to its named pipe if it then acts as the named pipe server.
    - The client of a named pipe, RPC, or DDE connection can control the impersonation level that the server of the named pipe can impersonate, ref: Microsoft (https://docs.microsoft.com/en-us/windows/win32/secauthz/impersonation-levels)
        - This doesn't apply if the connection is remote, in that instance the permissions a set by the server.
- Any service running through the Service Control Manager (SCM), or Component Object Model (COM) specified to run under a certain account, automatically has impersonate privileges.
- When creating a child process using 'CreateProcessWithToken' the secondary logon service 'seclogon' needs to be running or else this will fail.

Impersonation level	Description
SecurityAnonymous	The server cannot impersonate or identify the client.
SecurityIdentification	The server can get the identity and privileges of the client, but cannot impersonate the client.
SecurityImpersonation	The server can impersonate the client's security context on the local system.
SecurityDelegation	The server can impersonate the client's security context on remote systems.

Disable OLE objects in

```
Set-ItemProperty HKCU:\Software\Microsoft\Office\*\*\Security -Name PackagerPrompt -Type DWORD -Value 2
Set-ItemProperty REGISTRY::HKU\*\Software\Microsoft\Office\*\*\Security -Name PackagerPrompt -Type DWORD -Value 2
```

Process WMI objects

```
get-wmiobject -list | where {$_.name -like "*process*"}
```

Process information

```
Get-WmiObject win32_process|select processname,ProcessId,CommandLine
Get-WmiObject win32_process -Filter "name like '%powershell.exe'" | select processId,commandline|FL
Get-Process
```

Baseline processes and services

(Used to compare new process/services)

```
Get-Process | Export-Clixml -Path C:\Users\User\Desktop\process.xml
Get-Service | Export-Clixml -Path C:\Users\User\Desktop\service.xml
$edproc = Import-Clixml -Path C:\Users\User\Desktop\process.xml
$edproc1 = Import-Clixml -Path C:\Users\User\Desktop\process1.xml
$edservice = Import-Clixml -Path C:\Users\User\Desktop\service.xml
$edservice1 = Import-Clixml -Path C:\Users\User\Desktop\service1.xml
Compare-Object $edproc $edproc1 -Property processname
Compare-Object $edservice $edservice1 -Property servicename
```

View and interact with shadow copies (MUST BE RUN FROM ELEVATED CMD.exe)

```
vssadmin list shadows | findstr "VolumeShadowCopy"
mklink /d shadow \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2\
dir shadow
rmdir shadow
```

With a linked shadow copy we can copy pagefile.sys using the below.

```
mkdir C:\Audit
robocopy shadow C:\Audit pagefile.sys
attrib -s -h C:\Audit\pagefile.sys
```

Create Shadow Copy for C: drive

```
vssadmin create shadow /for=C:
```

Other Shadow Copy Techniques

In Windows 7 or certain other OS you may not have access to use 'vssadmin create'. As such some trickery may be required. In Windows 7 we can create a scheduled task (to execute with System privileges) and use it to create a Shadow Copy with Microsoft DLLs, this simulates the activity of creating a 'System Restore Point'. This can also be done with psexec if you wish to install the psexec service.

```
schtasks /ru "SYSTEM" /Create /SC DAILY /ST "00:00" /TN "\Microsoft\Windows\SystemRestore\SR" /TR
"%windir%\system32\rundll32.exe /d srrstr.dll,ExecuteScheduledSPPCreation" /f
schtasks /run /TN \Microsoft\Windows\SystemRestore\SR
vssadmin list shadows
```

If you want to remove the scheduled task so it doesn't run daily, use:

```
schtasks /delete /TN \Microsoft\Windows\SystemRestore\SR /f
```

You can also back it up using wbadmin (https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/wbadmin-start-backup), but it's a bit more intricate. The below example should backup C drive to E drive.

```
wbadmin start backup -backupTarget:E: -include:c:
```

TCP Connections

```
Get-NetTCPConnection –State Established
```

List of IPV4 addresses who have connected (RDP)

```
Get-WinEvent -Log 'Microsoft-Windows-TerminalServices-LocalSessionManager/Operational' | select -exp
Properties | where {$_.Value -like '*.*.*.*' } | sort Value -u
```

Powershell logs

```
Get-WinEvent -LogName "Windows Powershell"
```

Event logs available

```
Get-EventLog -list
Get-WinEvent -Listlog * | Select RecordCount,LogName
Get-WinEvent -Listlog *operational | Select RecordCount,LogName
wmic nteventlog list brief
```

Event Logs per Application Source

```
Get-EventLog Application | Select -Unique Source
Get-WinEvent -FilterHashtable @{LogName='Application';} -ea 0 | Select -exp ProviderName | sort |
unique
Get-WinEvent -FilterHashtable @{ LogName='Application'; ProviderName='Outlook'}
Get-WinEvent -FilterHashtable @{ LogName='Application'; ProviderName='MsiInstaller'}
Get-WinEvent -FilterHashtable @{ LogName='OAlerts';} | FL TimeCreated, Message
```

Event Logs per Severity Source

Critical Logs

```
Get-WinEvent -FilterHashtable @{ LogName='Application'; Level='1';}
```

Error Logs

```
Get-WinEvent -FilterHashtable @{ LogName='Application'; Level='2';}
```

Warning Logs

```
Get-WinEvent -FilterHashtable @{ LogName='Application'; Level='3';}
```

Information Logs

```
Get-WinEvent -FilterHashtable @{ LogName='Application'; Level='4';}
```

Live Event Log Filtering

```
$Before = Get-Date 01/07/19;
$After = Get-Date 31/05/19;

Get-WinEvent -FilterHashtable @{ LogName='Security'; StartTime=$After; EndTime=$Before; Id='4624';
Data='127.0.0.1'} | Select -ExpandProperty Message

Get-WinEvent -FilterHashtable @{ LogName='Security'; StartTime=$After; EndTime=$Before; Id='4624';
Data='127.0.0.1'} | Select TimeCreated,Message | Select-String -Pattern "0x621EFDC", "0x825225F"

Get-WinEvent -FilterHashtable @{ LogName='Security'; StartTime=$After; EndTime=$Before; Id='4624';
Data='127.0.0.1'} | Select -ExpandProperty Message > [location]\log.txt;
cat [location]\log.txt | Select-String -Pattern "Subject:", "New Logon:", "Process
information","Network Information:" -Context 0,4;

Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-SmbClient/Connectivity';} | Select
Timecreated,LogName,Message | where {$_.message -like "*Failed to establish a network connection*"} |FL

Get-WinEvent -FilterHashtable @{ LogName='*SMB*'; Data="[IP/HostName]"} | Select
Timecreated,LogName,Message |FL

Get-WinEvent -FilterHashtable @{ LogName='*SMB*';} | Select Timecreated,LogName,Message | where
{$_.message -like "*[IP/Hostname]*"} |FL

Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | where {$_.message -match
'0x1F260F3E' } | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | where
{$_.TimeCreated.ToString() -match ('28/10/2019')}|FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='2'} | where
{$_.TimeCreated.ToString() -match ('28/10/2019 11:22')}
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='127.0.0.1'} | where
{$_.TimeCreated.ToString() -match ('28/10/2019') -and $_.Message -match 'user' } | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='NTLM';} -MaxEvents 6 | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='127.0.0.1'} | where
{$_.TimeCreated -ge (get-date).addDays(-3) -and $_.TimeCreated.ToString() -match ('11:04') -and
$_.Message -match 'user' } | FL TimeCreated,Message
```

Find Authenticating user/asset for remote service creation (lateral movement)

```
$After = (get-date).addDays(-3);
$Time = Get-WinEvent -FilterHashtable @{ LogName='System'; Id='7045';} | ? {$_.Message -match
"MALWARE.exe"} | Select -exp TimeCreated;
foreach ($Event in $Time){
        $Event;
        Get-WinEvent -FilterHashtable @{ LogName='Security'; StartTime=$After; Id='4624';} | ?
{$_.TimeCreated -ge $Event.AddSeconds(-2) -AND $_.TimeCreated -le $Event.AddSeconds(+2)} | FL
TimeCreated,Message;
};
```

Locate possible Kerberoast/Kerberos based attacks

Note: When looking at kerberos listing, RC4-HMAC encryption is generally anomalous and may be indicative of kerberoasting.

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4769'; Data='0x17'} | FL TimeCreated, Message
klist
reg query "HKLM\System\CurrentControlSet\Control\Lsa\Kerberos\Parameters" /v "allowtgtsessionkey"
```

Extract useful fields from Legacy Logs

```
$A=Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='127.0.0.1'} | where
{$_.TimeCreated -ge (get-date).addDays(-3) -and $_.Message -match 'INSERT DESIRED INFO HERE' };
ForEach ($Event in $A){$Event.TimeCreated;$Event.Message|findstr /i /C:"Logon
Type:";$Event.Message|findstr /i /C:"Security ID:";$Event.Message|findstr /i /C:"Account
Name:";$Event.Message|findstr /i /C:"Account Domain:";$Event.Message|findstr /i /C:"Process
ID:";$Event.Message|findstr /i /C:"Process Name:";$Event.Message|findstr /i /C:"Workstation
Name:";$Event.Message|findstr /i /C:"Source Network Address:";$Event.Message|findstr /i /C:"Source
Port:";echo "`n";};
```

Note: You can modify the second string to carve out wanted information, some examples below.

Find User Authenticating

```
ForEach ($Event in $A){$Event.TimeCreated;$Event.Message|findstr /i /C:"Account
Name:";$Event.Message|findstr /i /C:"Account Domain:";echo "`n";};
```

Find IP/Port Authenticating

```
ForEach ($Event in $A){$Event.TimeCreated;$Event.Message|(findstr /i /C:"Source Network
Address:";$Event.Message|findstr /i /C:"Source Port:";)|findstr -v "-";echo "`n";};
```

** Note: In the following section filter based on time for reduction of noise Get-Date (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-date).

e.g. get something like the below and add them to the FilterHashTable: StartTime=$After; EndTime=$Before;

```
$Date = (Get-Date).AddDays(-2)
$Before = Get-Date 01/07/19;
$After = Get-Date 31/05/19;
```

Remote Desktop Lateral Movement Detection (Destinations)

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='10'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4778';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4779';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-RemoteDesktopServices-
RdpCoreTS/Operational'; Id='98';} | FL Message,ProcessId,TimeCreated
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-RemoteDesktopServices-
RdpCoreTS/Operational'; Id='131';} | FL Message,ProcessId,TimeCreated
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TerminalServices-
LocalSessionManager/Operational'; Id='21';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TerminalServices-
LocalSessionManager/Operational'; Id='22';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TerminalServices-
LocalSessionManager/Operational'; Id='25';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TerminalServices-
LocalSessionManager/Operational'; Id='41';} | FL TimeCreated,Message
ls C:\Windows\Prefetch\rdpclip.exe*.pf
ls C:\Windows\Prefetch\tstheme.exe*.pf
```

Map Network Shares Lateral Movement Detection (Destinations)

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4672';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4776';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4768';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4769';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='5140';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='5140'; Data='\\*\C$'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='5145';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='5140';} | FL TimeCreated,Message
```

PsExec Lateral Movement Detection (Destinations)

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='2'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4672';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='5140'; Data='\\*\ADMIN$'} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='System'; Id='7045'; Data='PSEXESVC'} | FL TimeCreated,Messag
reg query HKLM\SYSTEM\CurrentControlSet\Services\PSEXESVC
reg query HKLM\SYSTEM\CurrentControlSet\Services\
ls C:\Windows\Prefetch\psexesvc.exe*.pf
```

Scheduled Tasks Lateral Movement Detection (Destinations)

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4672';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4698';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4702';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4699';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4700';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4701';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TaskScheduler/Maintenance'; Id='106';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TaskScheduler/Maintenance'; Id='140';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TaskScheduler/Maintenance'; Id='141';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TaskScheduler/Maintenance'; Id='200';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-TaskScheduler/Maintenance'; Id='201';} | FL
TimeCreated,Message
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks" /s
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks" /s /v Actions
Get-ChildItem -path 'registry::HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Schedule\TaskCache\Tasks\' | Get-ItemProperty | FL Path, Actions
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree"
gci -path C:\Windows\System32\Tasks\ -recurse -File
```

Services Lateral Movement Detection (Destinations)

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4697';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='System'; Id='7034';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='System'; Id='7035';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='System'; Id='7036';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='System'; Id='7040';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='System'; Id='7045';} | FL TimeCreated,Message
reg query 'HKLM\SYSTEM\CurrentControlSet\Services\'
```

WMI/WMIC Lateral Movement Detection (Destinations)

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4672';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-WMI-Activity/Operational'; Id='5857';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-WMI-Activity/Operational'; Id='5860';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-WMI-Activity/Operational'; Id='5861';} | FL
TimeCreated,Message
C:\Windows\System32\wbem\Repository
ls C:\Windows\Prefetch\wmiprvse.exe*.pf
ls C:\Windows\Prefetch\mofcomp.exe*.pf
```

PowerShell Lateral Movement Detection (Destinations)

```
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4624'; Data='3'} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Security'; Id='4672';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-PowerShell/Operational'; Id='4103';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-PowerShell/Operational'; Id='4104';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-PowerShell/Operational'; Id='53504';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Windows PowerShell'; Id='400';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Windows PowerShell'; Id='403';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Windows PowerShell'; Id='800';} | FL TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-WinRM/Operational'; Id='91';} | FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{ LogName='Microsoft-Windows-WinRM/Operational'; Id='168';} | FL
TimeCreated,Message
ls C:\Windows\Prefetch\wsmprovhost.exe*.pf
cat $env:APPDATA\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt
```

Extra Information

Program Compatibility Assistant (PCA) (Windows 11 Pro 22H2 and later)

```
C:\Windows\appcompat\pca\PcaAppLaunchDic.txt
C:\Windows\appcompat\pca\PcaGeneralDb0.txt
C:\Windows\appcompat\pca\PcaGeneralDb1.txt
```

Note: Special thanks to Andrew Rathburn, and Lucas Gonzalez who publicised this (https://aboutdfir.com/new-windows-11-pro-22h2-evidence-of-execution-artifact/), and `rancio#4162` who first discovered and raised the question on it.

AmCache

```
C:\Windows\AppCompat\Programs\Amcache.hve
Amcache.hve\Root\File\{Volume GUID}\#######
```

ShimCache C:\Windows\System32\config\SYSTEM
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\AppCompatCache

Prefetch

```
ls C:\Windows\Prefetch\evil.exe*.pf
```

Connected Devices Platform (Timelining)

```
gci C:\Users\*\AppData\Local\ConnectedDevicesPlatform -recurse
```

User accounts and logon information

```
Get-WmiObject Win32_UserProfile
```

Share information

```
Get-WmiObject Win32_Share
net share
wmic share list brief
wmic netuse get Caption, DisplayType, LocalName, Name, ProviderName, Status
```

List Alternate Data Streams in current Dir and view them

```
gi * -s *
gc [FILENAME] -s [ADSNAME]
```

List Alternate Data Streams in text files within AppData

```
Get-ChildItem -Recurse -Path $env:APPDATA\..\ -include *.txt -ea SilentlyContinue|gi -s *|Select Stream
-ea SilentlyContinue| Where-Object {$_.Stream -ine ":`$DATA"}
```

Use Alternate Data Streams to find download location

```
get-item * -stream *|Where-Object {$_.Stream -ine ":`$DATA"}|cat
get-item C:\Users\Username\Downloads\* -stream *|Where-Object {$_.Stream -ine ":`$DATA"}|cat
$a=(gci -rec -path C:\users\user\downloads -ea 0 | gi -s Zone.Identifier -ea 0 | ? {$_.Length -ge
'27'});foreach ($b in $a){$b.FileName;$b|cat}
$a=(get-item * -stream Zone.Identifier -ea 0 | ? {$_.Length -ge '27'});foreach ($b in $a)
{$b.FileName;$b|cat}
gci -Recurse -Path $env:APPDATA\..\ -include *.txt -ea SilentlyContinue |gi -s *| Where-Object
{$_.Stream -ine ":`$DATA"}|cat
```

General Notes

Under **%SystemRoot%\System32\config** the below registry hives are some of the most important to obtain. Additionally taking these files from within the RegBack directory also assists in comprehensive analysis should any anti-forensics activities have modified these registries.

- DEFAULT
- SAM
- SECURITY
- SOFTWARE
- SYSTEM

Under **%SystemDrive%\Users[name]** there is also a **NTUSER.DAT** file which becomes HKEY_CURRENT_USER into the Registry when a user logs on, and this is very important to obtain. There's also a UsrClass.dat file which can be found:
%USERPROFILE%\AppData\Local\Microsoft\Windows\UsrClass.dat

Gather artifacts

```
reg save HKLM\SAM [LOCATION]\SAM
reg save HKLM\SYSTEM [LOCATION]\SYSTEM
reg save HKLM\SECURITY [LOCATION]\SECURITY
reg save HKLM\SOFTWARE [LOCATION]\SOFTWARE
```

Powershell execution log

```
cat C:\Users\[name]\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline
```

Analyse document for macros

Using olevba (https://github.com/decalage2/oletools/wiki/olevba)

```
olevba [Document]
```

Capture powershell memdump and analyse

Using Procdump from sysinternals:

```
procdump -ma [PowershellPID]
```

Recent execution of programs

- Prefetch Located at : %SystemRoot%\Prefetch\
- RecentFileCache.bcf Located at : %SystemRoot%\AppCompat\Programs\
- Amcache.hve (reg hive) Located at : %SystemRoot%\AppCompat\Programs\
- Program Compatibility Assistant (PCA) (Windows 11 Pro 22H2 and later) Located at:
    - %SystemRoot%\appcompat\pca\PcaAppLaunchDic.txt
    - %SystemRoot%\appcompat\pca\PcaGeneralDb0.txt
    - %SystemRoot%\appcompat\pca\PcaGeneralDb1.txt

Or query a lot of run programs from program compatibility assistant:

```
Get-ItemProperty "REGISTRY::HKCU\Software\Microsoft\Windows
NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store"
Get-ItemProperty "REGISTRY::HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers"
```

Programs specifically set to run as admin

```
reg query "HKU\{SID}\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers" /s /f
RUNASADMIN
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers" /s /f RUNASADMIN
```

Windows Indexing Service

```
C:\ProgramData\Microsoft\Search\Data\Applications\Windows\windows.edb
```

- ESE Database View (https://www.nirsoft.net/utils/ese_database_view.html)
- View ESE Database (http://www.edgemanage.emmet-gray.com/Articles/ViewESE.html)

Programs Accessing Windows Features such as Webcam and Microphone

Special thanks to Zack (svch0st) for his Medium Post (https://medium.com/@7a616368/can-you-track-processes-accessing-the-camera-and-microphone-7e6885b37072)

All Windows Features Including Start and Stop Timestamps

```
$a=$(gci REGISTRY::HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\CapabilityAccessManager\ConsentStore\
-recurse | FT -AutoSize | Out-String);$a.replace("#","\")
reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\CapabilityAccessManager\ConsentStore\ /s /f
LastUsed
```

Programs Using Webcam

```
$a=$(gci
REGISTRY::HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\CapabilityAccessManager\ConsentStore\webcam -
recurse | Select PSChildName | Out-String);$a.replace("#","\")
```

Programs Using Microphone

```
$a=$(gci
REGISTRY::HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\CapabilityAccessManager\ConsentStore\microphon
e -recurse | Select PSChildName | Out-String);$a.replace("#","\")
```

USN Journal (any changes to NTFS volume)

```
fsutil usn readjournal C: > USN.txt
```

NTFS Index Attributes ($I30 file made up of $INDEX_ROOT, $INDEX_ALLOCATION attributes)

INDXParse (https://github.com/williballenthin/INDXParse)

INDXRipper (https://github.com/harelsegev/INDXRipper)

Indx2Csv (https://github.com/jschicht/Indx2Csv)

Link File Analysis

LNK Files Located at:

```
C:\Users\*\AppData\Roaming\Microsoft\Windows\Recent
```

Using LECmd (https://ericzimmerman.github.io/#!index.md) to parse Link metadata.

```
LECmd.exe -f {fileDirectory}\filename.lnk
```

Of interest is information such as:

- MachineID (NetBIOS name)
- MAC Address
- MAC Vendor
- Timestamps
- Volume Droid
- Volume Droid Birth
- File Droid
- File Droid Birth

File format specification (https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-shllink/16cb4ca1-9339-4d0c-a68d-bf1d6cc0f943)

Jump Lists Analysis

Jump List Files Located at:

```
C:\Users\*\AppData\Roaming\Microsoft\Windows\Recent\AutomaticDestinations
```

A rough PowerShell 1-liner to gather information on previous opened directories and files is below

```
$Files=$(cat C:\Users\*\AppData\Roaming\Microsoft\Windows\Recent\*Destinations\*.*Destinations-ms);$Files.Split("``")|Select-String "Storage" | findstr -v "1SPSU"|findstr -v "?"
```

SRUM Analysis

System Resource Usage Monitor Located at: %systemroot%\System32\sru\SRUDB.dat

Great tool to parse to csv: SRUM-Dump (https://github.com/MarkBaggett/srum-dump)

Background Activity Moderator (BAM/DAM)

```
reg query "HKLM\SYSTEM\CurrentControlSet\Services\bam\UserSettings" /s
reg query "HKLM\SYSTEM\CurrentControlSet\Services\dam\UserSettings" /s
reg query "HKLM\SYSTEM\CurrentControlSet\Services\bam\UserSettings" /s /v *.exe
reg query "HKLM\SYSTEM\CurrentControlSet\Services\dam\UserSettings" /s /v *.exe
```

Windows 10 Mail App Forensics

```
%LocalAppData%\Comms\Unistore\data\0 - Windows phone data
%LocalAppData%\Comms\Unistore\data\2 - Contact lists
%LocalAppData%\Comms\Unistore\data\3 - Contents/body of email
%LocalAppData%\Comms\Unistore\data\5 - Calendar invitations
%LocalAppData%\Comms\Unistore\data\7 - Email attachments
```

Capture packets with netsh

Note: You will need to use something like etl2pcapng to convert these captures to a cap file for analysis with Wireshark Download (https://github.com/microsoft/etl2pcapng/releases)

```
netsh trace start persistent=yes capture=yes tracefile=c:\temp\packetcapture.etl
netsh trace stop
```

Capture Packets with PowerShell

```
New-NetEventSession -Name "Capture" -CaptureMode SaveToFile -LocalFilePath "c:\temp\packetcapture.etl"
Add-NetEventProvider -Name "Microsoft-Windows-TCPIP" -SessionName "Capture"
Add-NetEventPacketCaptureProvider -SessionName "Capture"
Start-NetEventSession -Name "Capture"
```

Stop Capturing Packets with PowerShell

```
Get-NetEventSession
Stop-NetEventSession -Name Capture
Remove-NetEventSession
```

Convert ETL File to PCAP

```
etl2pcapng.exe c:\temp\packetcapture.etl c:\temp\packetcapture.pcapng
```

NTUSER.DAT Important Registry entries:

Recent execution of programs (GUI)

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer
```

- \RecentDocs *(Notes recent files run, most commonly .lnk files)*
- \UserAssist *(Notes files run and number of times run. Values are ROT13 encoded)*,
    - CEBFF5CD is Executable File Execution
    - F4E57C4B is Shortcut File Execution
- \TypedPaths *(Notes file locations visited using Windows Explorer address bar)*
- \RunMRU *(Notes recent commands executed through the 'run' program)*
- \ComDlg32 *(Last file path visited)*
    - \LastVisitedPidlMRU *(Last PID which was 'Most Recently Used', e.g. the binaries used to open a file)*
    - \OpenSavePidlMRU *(Last Saved PID file which was 'Most Recently Used', location of a file opened by a binary)*
- \WordWheelQuery *(Keywords searched for from the START menu bar)*
- \FeatureUsage\AppBadgeUpdated (Every Time Task Bar Application Gets Notification and Badge Updates)
- \FeatureUsage\AppLaunch (Every Time Task Bar Application Which is Pinned is Launched)
- \FeatureUsage\AppSwitched (Every Time Task Bar Application Left Clicked)
- \FeatureUsage\ShowJumpView (Every Time Task Bar Application Right Clicked)
- \FeatureUsage\TrayButtonClicked (Every Time Relevant Button on Task Bar is Clicked)

You can get the names of recently run files from UserAssist by using ROT13 across them, we can do this quickly in Powershell by using a script from BornToBeRoot and some scriptfu:

Convert-ROT13.ps1 (https://github.com/JPMinty/PowerShell/blob/master/Scripts/Convert-ROT13.ps1)

```
$A=$(gci
REGISTRY::HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\*\Count |
Select -exp Property);foreach ($B in $A){.\Convert-ROT13.ps1 -Rot 13 $B|Select -exp Text}
```

Or for those who don't want to run it over every entry individually.

```
$A=$(gci
REGISTRY::HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\*\Count |
Select -exp Property)|Out-String;.\Convert-ROT13.ps1 -Rot 13 $A|Select -exp Text
```

Recent Apps/Last Visited MRU

Note: OpenSavePidlMRU is in hex and will need to be decoded

```
reg query "HKCU\Software\Microsoft\Windows\Current Version\Search\RecentApps"
reg query "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\OpenSavePidlMRU" /s
```

Execution of Sysinternals Tool

```
reg query HKCU\Software\Sysinternals\ /s /v EulaAccepted
reg query HKU\SID\Software\Sysinternals\ /s /v EulaAccepted
```

Recent Internet Explorer History

```
reg query "HKCU\Software\Microsoft\Internet Explorer\TypedURLs"
C:\Users\username\AppData\Local\Microsoft\Windows\History\Low\History.IE5\
C:\Users\username\AppData\Local\Microsoft\Windows\History\
C:\Users\username\AppData\Roaming\Microsoft\Internet Explorer\UserData\Low
C:\Users\username\AppData\Local\Microsoft\Windows\WebCache\Internet Explorer\WebCacheV01.dat
```

IE Cache Viewer (https://www.nirsoft.net/utils/ie_cache_viewer.html) Browser History Viewer (https://www.foxtonforensics.com/browser-history-viewer/) ESE Database View (https://www.nirsoft.net/utils/ese_database_view.html) Browsing History View (https://www.nirsoft.net/utils/browsing_history_view.html)

```
esentutl.exe /y /vss C:\Users\Username\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat /d
C:\Location\WebCacheV01.dat
```

Google Chrome Service Workers

“A service worker is a script that your browser runs in the background, separate from a web page, opening the door to features that don’t need a web page or user interaction.” - Service Worker Reference (https://developers.google.com/web/fundamentals/primers/service-workers/)

Service Workers control popups and push notifications received from websites. As this can be used to trick users into taking actions such as opening pages with redirects, installing malware and more these have been included and can be found at the below:

```
In Browser: chrome://serviceworker-internals/
C:\Users\[username]\AppData\Local\Google\Chrome\User Data\Default\Service Worker\ScriptCache
C:\Users\[username]\AppData\Local\Google\Chrome\User Data\Default\Preferences
```

Recent Chrome History

```
'C:\Users\Username\AppData\Local\Google\Chrome\User Data\Default\history'
```

Note: This is stored as a SQL database which can be easily examined using a tool like sqlite3 which is available on MacOS and Linux. Example below to scan user downloads:

```
sqlite3 -line [HISTORYFILE] 'SELECT * FROM Downloads;' | grep -i '[keyword to search for]' -A 10
```

Recent Firefox History

More Information (https://support.mozilla.org/en-US/kb/profiles-where-firefox-stores-user-data#w_what-information-is-stored-in-my-profile)

```
C:\Users\username\AppData\Roaming\Mozilla\Firefox\Profiles\*\
```

Recent Edge History

```
C:\Users\Username\AppData\Local\Packagaes\Microsoft.MicrosoftEdge_xxxx\AC\#!001\MicrosoftEdge\Cache\
C:\Users\Username\AppData\Local\Packagaes\Microsoft.MicrosoftEdge_xxxx\AC\MicrosoftEdge\User\Default\Recovery\Active\
C:\Users\Username\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat
```

Check Root Certificate Store

```
reg query HKLM\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates\
reg query HKCU\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates\
reg query HKU\{SID}\SOFTWARE\Microsoft\SystemCertificates\ROOT\Certificates\
reg query HKLM\Software\Policies\Microsoft\SystemCertificates\Root\Certificates\
reg query HKCU\Software\Policies\Microsoft\SystemCertificates\Root\Certificates\
reg query HKU\{SID}\Software\Policies\Microsoft\SystemCertificates\Root\Certificates\
```

Thumbnail Cache

Thumbcacheviewer (https://github.com/thumbcacheviewer/thumbcacheviewer)

```
C:\Users\Username\AppData\Local\Microsoft\Windows\Explorer
```

Shellbags

Shellbags can be used to verify the previous existance of files which have been deleted. This is used by the OS to store information about a file location's customisation e.g. look, feel, size, sorting files method, colour etc and resides after files have been deleted. Shellbags Explorer or ShellBagsView can be used to parse this information.

HKCU\SOFTWARE\Microsoft\Windows\Shell

- \BagMRU
- \Bags

BagsMRU contains the database of folders and their saved settings by windows.

UsrClass.dat Shellbags

Additional shellbags files can be found in UsrClass.dat

```
HKCU\SOFTWARE\Classes
%USERPROFILE%\AppData\Local\Microsoft\Windows\UsrClass.dat
```

SOFTWARE Hive Registry Entries

Common startup locations

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Runonce
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunonceEx
```

Network Information

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList
        \Signatures
                \Unmanaged
                        (record DefaultGatewayMac, DnsSuffix, FirstNetwork(SSID), ProfileGUID)
                \Managed
        \Nla\Cache
        Profiles

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\HomeGroup
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Profiles\[GUID]
        0x06 = Wired
        0x17 = Broadband
        0x47 = Wireless
```

Gather information via Live Queries:

```
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures\Unmanaged" /s
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Profiles" /s
```

Gather WiFi Passwords

```
netsh wlan show profile
netsh wlan show profile name={NAME} key=clear
netsh wlan export profile interface=* key=clear
ls C:\ProgramData\Microsoft\Wlansvc\Profiles\Interfaces\*\* | cat
```

Networks connected\disconnected to\from and mac address

```
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-NetworkProfile/Operational'; Id='10000';}|FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-NetworkProfile/Operational'; Id='10001';}|FL
TimeCreated,Message
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-DHCP*'; Id='50067';}|FL TimeCreated,Message
Get-WmiObject win32_networkadapterconfiguration | FL description, macaddress
```

Get host Mac Addresses

```
getmac
```

Lookup MAC Address/Organizationally Unique Identifier (OUI)

A number of ways to do this but one of the most accurate is from the IEEE Webpage (https://regauth.standards.ieee.org/standards-ra-web/pub/view.html#registries). You can interact directly with their REST API to get the results you want in JSON format. To do so via PowerShell, just change the 'text' top the first 3 digits of the MAC Address (The OUI):

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12;Invoke-WebRequest -Uri "https://services13.ieee.org/RST/standards-ra-web/rest/assignments/?registry=MAC&text=08-00-27&sortby=organization&sortorder=asc&size=10" | Select -exp content;
```

Useful Wireshark filters

All traffic to or from an IP

```
ip.addr == {IP}
```

All TCP traffic on a port

```
tcp.port == {port}
```

All traffic from an IP

```
ip.src == {IP}
```

All traffic to an IP

```
ip.dst == {IP}
```

HTTP or DNS Traffic

```
http || dns
```

Client>DC traffic filtering noise

```
smb || nbns || dcerpc || nbss || dns
```

TCP issues (flags)

```
tcp.analysis.flags
```

TCP protocol flags (present)

```
tcp.flags.syn == 1
tcp.flags.ack == 1
```

Encrypted Traffic

```
(ssh || tls)
```

Follow TCP Stream

```
tcp.stream == {StreamNum}
```

TCP packets for string

```
tcp contains {String}
```

HTTP codes

```
http.response.code == 400
http.response.code == 200
http.response.code == 404
http.response.code == 500
```

User agent, without/with case sensitivity

```
http.user_agent matches "firefox"
http.user_agent matches "(?-i)firefox"
```

Addresses on subnet by IP/Hostname

```
ip.addr == 172.217.167.78/16
ip.addr eq hostname/24
```

TLS Handshakes (Show's certificate information)

```
tls.handshake.type == 11
```

Filter based on server type

```
http.server == "Apache"
```

Wireshark Key Tips and Tricks by Brad Duncan

- General Wireshark Filter Reference (https://www.wireshark.org/docs/man-pages/wireshark-filter.html)
- Full Wireshark Display Filter Reference (https://www.wireshark.org/docs/dfref/)
- Customizing Wireshark – Changing Your Column Display (https://unit42.paloaltonetworks.com/unit42-customizing-wireshark-changing-column-display/)
- Using Wireshark – Display Filter Expressions (https://unit42.paloaltonetworks.com/using-wireshark-display-filter-expressions/)
- Using Wireshark: Identifying Hosts and Users (https://unit42.paloaltonetworks.com/using-wireshark-identifying-hosts-and-users/)
- Using Wireshark: Exporting Objects from a Pcap (https://unit42.paloaltonetworks.com/using-wireshark-exporting-objects-from-a-pcap/)
- Wireshark Tutorial: Examining Trickbot Infections (https://unit42.paloaltonetworks.com/wireshark-tutorial-examining-trickbot-infections/)
- Wireshark Tutorial: Examining Ursnif Infections (https://unit42.paloaltonetworks.com/wireshark-tutorial-examining-ursnif-infections/)

Decrypting Encrypted Packets

This can be done in a few ways:

Man-in-the-middle (MITM)

- MITM Through SSLStrip (https://github.com/moxie0/sslstrip)
- MITM Through mitmproxy (https://mitmproxy.org/)

Using the (Pre)-Master-Secret SSLKEYLOGFILE (https://wiki.wireshark.org/TLS#Using_the_.28Pre.29-Master-Secret)
Using an RSA Private Key (https://docs.microsoft.com/en-us/archive/blogs/nettracer/decrypting-ssltls-sessions-with-wireshark-reloaded)

```
tshark -nr pcapfile -o ssl.keylog_file:./random.keylog -T fields -e http2.headers.cookie
```

Using tshark to analyse pcaps

Note: The above filters can be used with the below techniques by specifying '-Y' and using a capture filter, or '-R' and using a read/display filter (only use if doing multiple passes of file). Different outputs are specified using '-T'.

Merging multiple pcap files

Note: mergecap (https://www.wireshark.org/docs/man-pages/mergecap.html)

```
mergecap /[directory]/*.pcap -w /[directory]/capture.pcap
```

List Unique IP Sources in Pcap

```
tshark -T fields -r 'capture.pcap' -e ip.src | sort -u
```

List Unique IP Sources and Destination for HTTP traffic

```
tshark -T fields -r 'capture.pcap' -e ip.src -e ip.dst -Y "http" | sort -u
```

Live DNS Request and Responses on WiFi

```
tshark -i wlan0 -T fields -f "src port 53" -n -e dns.qry.name -e dns.resp.addr
```

Extract All Objects/Files from Supported Protocols

Note: This will create a folder called 'exported' and put the results in there

```
tshark -r 'capture.pcap' --export-objects http,exported
tshark -r 'capture.pcap' --export-objects dicom,exported
tshark -r 'capture.pcap' --export-objects imf,exported
tshark -r 'capture.pcap' --export-objects smb,exported
tshark -r 'capture.pcap' --export-objects tftp,exported
```

List URIs Accessed

```
tshark -T fields -r capture.pcap -e http.host -e ip.dst -e http.request.full_uri -Y "http.request"
```

Get HTTP POST Requests and Output to JSON

```
tshark -T json -r capture.pcap -Y "http.request.method == POST"
```

Export objects

```
tshark -r capture.pcap --export-objects http,[folderName]
tshark -r capture.pcap --export-objects dicom,[folderName]
tshark -r capture.pcap --export-objects tftp,[folderName]
tshark -r capture.pcap --export-objects imf,[folderName]
tshark -r capture.pcap --export-objects smb,[folderName]
```

List POST Request Parameters

```
tshark -r victim.pcap -Y "ip.src == [IP] && http.request.method == POST" -T fields -e frame.time -e
ip.src -e http.host -e http.request.uri -e urlencoded-form.key -e urlencoded-form.value
```

Using tcpdump to analyse (cut out) pcaps

Packets containing Syn & Ack Flags

```
tcpdump -r file.pcap "tcp[tcpflags] & (tcp-syn|tcp-ack) == tcp-syn|tcp-ack" -w /targetfile.pcap
```

Packets containing only Syn Flag

```
tcpdump -r file.pcap "tcp[tcpflags] == tcp-syn" -w /targetfile.pcap
```

Packets containing only Syn and Ack Flags

```
tcpdump -r file.pcap "tcp[tcpflags] == tcp-syn|tcp-ack" -w /targetfile.pcap
```

SYSTEM Hive Registry Entries

USB Information

USB information is commonly found within the Windows Event Logs from Storage Service and Partition diagnostics on Windows 10 and later:

```
Microsoft-Windows-Storsvc/Diagnostic
Microsoft-Windows-Partition/Diagnostic
```

Example query last USB inserted:

```
$a=(Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-Storsvc/Diagnostic';Id='1001'} -
MaxEvents 1);
($a) | select -exp TimeCreated; ($a).Properties | select -exp Value;
```

Example query USB removed:

```
$a=(Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-Partition/Diagnostic';} -MaxEvents 1);
($a) | select -exp TimeCreated; ($a).Properties | select -exp Value;
```

Outside of this, the registry has a number of artifacts:

- HKLM\SOFTWARE\Microsoft\Windows Portable Devices\Devices Note: Find Serial # and then look for FriendlyName to obtain the Volume Name of the USB device

- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\EMDMgmt
    - Key will ONLY be present if system drive is NOT SSD
    - Traditionally used for ReadyBoost
    - Find Serial # to obtain the Volume Serial Numberof the USB deviceoThe Volume Serial Number will be in decimal - convert to hex
    - You can find complete history of Volume Serial Numbers here, even if the device has been formatted multiple times. The USB device's Serial # will appear multiple times, each with a different Volume Serial Number generated on each format.
- HKLM\SYSTEM\MountedDevices
    - Find Serial # to obtain the Drive Letter of the USB device
    - Find Serial # to obtain the Volume GUID of the USB device

Using the VolumeGUID found in SYSTEM\MountedDevices, you can find the user that actually mounted the USB device:

```
NTUSER.DAT\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Mountpoints2
```

USB Times:

- First time device is connected
- Last time device is connected
- Removal time

Live System

```
HKLM\SYSTEM\CurrentControlSet\Enum\USBSTOR (Class ID/Serial Number)
HKLM\SYSTEM\CurrentControlSet\Enum\USB (VID/PID)
HKLM\SYSTEM\CurrentControlSet\Control\DeviceContainers\*
```

Some quick 1-liners:

```
reg query HKLM\SYSTEM\CurrentControlSet\Enum\SWD\WPDBUSENUM\ /s /f FriendlyName
reg query HKLM\SYSTEM\CurrentControlSet\Enum\USBSTOR\ /s /f FriendlyName
reg query HKLM\SYSTEM\CurrentControlSet\Enum\USB\ /s /f FriendlyName
reg query HKLM\SYSTEM\CurrentControlSet\Enum\USBSTOR\
reg query HKLM\SYSTEM\CurrentControlSet\Enum\USB\
reg query HKLM\SYSTEM\CurrentControlSet\Enum\SWD\WPDBUSENUM\
reg query HKLM\SYSTEM\CurrentControlSet\Control\DeviceContainers\ /s /f "USB"
```

Live System pnputil for all devices and interfaces

```
pnputil /enum-devices
pnputil /enum-interfaces
```

Forensic Image (Determine Control Set Number from HKLM\SYSTEM\Select\ -> Current Value)
HKLM\SYSTEM\ControlSet00x\Enum\USBSTOR (Class ID/Serial Number)
HKLM\SYSTEM\ControlSet00x\Enum\USB (VID/PID)

Note: VID/PID information can be found online. Subdirectories under USB and USBSTOR provide unique USB identifiers (if the & is near the end), if it is near the start they do not conform to MS standards and it is unique to the given PC only.

```
HKLM\SYSTEM\CurrentControlSet\Enum\USBSTOR\Ven_Prod_Version\USB iSerial#\Properties\{GUID}\####
```

- 0064 = First Install
- 0066 = Last Connected
- 0067 = Last Removal

More Information (https://github.com/woanware/usbdeviceforensics)

OS Information

```
HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation
HKLM\SYSTEM\CurrentControlSet\Control\ComputerName\ComputerName
        HKLM\SYSTEM\CurrentControlSet\services\LanmanServer\Shares
HKLM\SYSTEM\CurrentControlSet\FileSystem
        NtfsDisableLastAccessUpdate set at 0x1 means that access time stamps are turned OFF by default
```

Network Information

```
wmic nic get /all /format:list
wmic nicconfig get /all /format:list

HKLM\SYSTEM\CurrentControlSet\services\Tcpip\Parameters\Interfaces
        Display interfaces and their IP address configuration (using interface GUID)
```

Prefetch Information

```
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PrefetchParameters
        0=Disabled
        1=Application prefetching enabled
        3=Application and Boot prefetching enabled (default)
```

Prefetch Parser (https://ericzimmerman.github.io/#!index.md)

```
PECmd.exe -d "C:\Windows\Prefetch"
PECmd.exe -d "C:\Windows\Prefetch" --csv "c:\temp" --csvf Prefetch.csv
gci C:\Windows\Prefetch\
```

analyzePF (https://github.com/analyzeDFIR/analyzePF) WinPrefetchView (https://www.nirsoft.net/utils/win_prefetch_view.html)

Superfetch Information

- HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PrefetchParameters /v EnableSuperfetch
    - 0=Disabled
    - 1=Booting only
    - 2=Applications
    - 3=Application and Boot

Find relevant files:

```
gci C:\Windows\Prefetch\ -recurse -include *.db, *.trx
```

PowerShell Host Based Investigation and Containment Techniques

Note: I thoroughly recommend looking at maintaining an accurate log of all actions taken through the use of PowerShell host based IR and Containment. To do so I recommend reading PowerShell ♥ the Blue Team (https://devblogs.microsoft.com/powershell/powershell-the-blue-team/). This will allow you to log all actions taken through this type of IR Framework. Other alternatives for larger scale response include:

PowerForensics (https://powerforensics.readthedocs.io/en/latest/)

Google Rapid Response (https://github.com/google/grr)

Kansa PowerShell IR Framework (https://github.com/davehull/Kansa)

Google Rapid Response comes in the form of a Server > Client architecture but is very flexible.

- GRR Docs (https://grr-doc.readthedocs.io/en/latest/index.html)

Kansa is a modular PowerShell IR Framework which can be used across multiple hosts in parallel.

Enable PS Remoting using PsExec

```
psexec.exe \\TARGET -s powershell Enable-PSRemoting -Force;
```

OR for public network setup (less security)

```
psexec.exe \\TARGET -s powershell Enable-PSRemoting -SkipNetworkProfileCheck -Force;
```

Confirm trusted hosts list if required

```
Get-Item WSMan:\localhost\Client\TrustedHosts
Set-Item WSMan:\localhost\Client\TrustedHosts -Value 'ASSET1,ASSET2'
```

Quick Remote Response (no audit/logging)

```
Enter-PSSession -ComputerName SERVER -Credential [name]
```

Setup logging for IR

Note: If you enter a PSSession, the logging won't persist, so you will need to enable it on the remote host and pull the file back afterwards. Otherwise refer to PowerShell ♥ the Blue Team mentioned above.

```
Start-Transcript -Path "C:\[location]\investigation-1.log" -NoClobber
```

Thanks Barnaby Skeggs (https://b2dfir.blogspot.com/2018/11/windows-powershell-remoting-host-based.html)

Establish Remote Session

```
$s1 = New-PSsession -ComputerName remotehost -SessionOption (New-PSSessionOption -NoMachineProfile) -
ErrorAction Stop
```

Enter or exit remote session

```
Enter-PSSession -Session $s1
Exit-PSSEssion
```

Issuing remote command/shell

```
Invoke-Command -ScriptBlock {whoami} -Session $s1
Invoke-Command -file file.ps1 -Session $s1
```

Retrieving/downloading files

```
Copy-Item -Path "[RemoteHostFilePath]" -Destination "[LocalDestination]" -FromSession $s1
```

Checking for running processes

```
Invoke-Command -ScriptBlock {Get-Process} -Session $s1
```

Query Registry Keys

```
Invoke-Command -ScriptBlock {Get-ItemProperty -Path
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run} -Session $s1
```

PCAP collection

*Note: Script and pcap should be located under: C:\Windows\System32 or your user directory.

```
Invoke-Command -ScriptBlock {ipconfig} -Session $s1

Invoke-Command -ScriptBlock {
$url = "https://raw.githubusercontent.com/nospaceships/raw-socket-sniffer/master/raw-socket-
sniffer.ps1"
Invoke-WebRequest -Uri $url `
        -OutFile "raw-socket-sniffer.ps1"
PowerShell.exe -ExecutionPolicy bypass .\raw-socket-sniffer.ps1 `
        -InterfaceIp "[RemoteIPv4Address]
        -CaptureFile "capture.cap"
        } -Session $s1
```

Blocking a domain

```
Invoke-Command -ScriptBlock { Add-Content C:\Windows\System32\drivers\etc\hosts "`n127.0.0.1 bad.com"}
-Session $s1
```

Blocking an IP

```
Invoke-Command -ScriptBlock {New-NetFirewallRule -DisplayName "Block_Malicious_IP" -Direction Outbound
–LocalPort Any -Protocol TCP -Action Block -RemoteAddress 173.182.192.43}  -Session $s1
```

Unblocking an IP

```
Invoke-Command -ScriptBlock {Remove-NetFirewallRule -DisplayName "Block_Malicious_IP"} -Session $s1
```

Quarantining a host using Firewall

```
Invoke-Command -ScriptBlock {New-NetFirewallRule -DisplayName InfoSec_Quarantine -Direction Outbound -
Enabled True -LocalPort Any -RemoteAddress Any -Action Block} -Session $s1
```

Creating an OU to quarantine a host into

```
import-module ActiveDirectory
New-ADOrganizationalUnit -Name "Quarantined" -Path "DC=CORP,DC=COM"
```

Or

```
dsadd ou "ou=Quarantined,dc=CORP,dc=COM"
```

Moving an AD Object into a quarantine OU

```
Move-ADObject -Identity "CN=[USERNAME],OU=Marketing,DC=CORP,DC=com" -TargetPath
"OU=Quarantined,DC=CORP,DC=com"
```

Quarantining a host using DCOM

Note: Another method which is a little more DANGEROUS is to disable DCOM on this host and restart which will stop something using DCOM to spread TO this host. This will likely break some aspects of it communicating on the domain, and also your ability to respond so it's not recommended, but this is a possible solution for a host which is to be thoroughly contained befor being investigated once an asset is returned (for example for re-imaging). This can be reversed by changing this registry key back to Y.

```
Invoke-Command -ScriptBlock {reg add HKLM\SOFTWARE\Microsoft\Ole /v EnableDCOM /t REG_SZ /d N /f;
shutdown /r /f;} -Session $s1
```

Remove a quarantined host

```
Invoke-Command -ScriptBlock {Remove-NetFirewallRule -DisplayName InfoSec_Quarantine} -Session $s1
```

Disable Admin Shares

```
Invoke-Command -ScriptBlock {Red add "HKLM\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters"
/V "AutoShareWks" /T "REG_DWORD" /D "0" /F } -Session $s1
Invoke-Command -ScriptBlock {restart-service Lanmanserver -Force} -Session $s1
```

Credentials and Exposure

When investigating a compromised asset, it's important to know what remote triage methods leave your credentials on the infected endpoint, and what ones don't. Reference can be found on Microsoft Documentation (https://docs.microsoft.com/en-us/windows-server/identity/securing-privileged-access/securing-privileged-access-reference-material#administrative-tools-and-logon-types)

Connection Method	Logon Type	Reusable credentials on destination	Notes
Logon via console	Interactive	Y	Includes hardware remote access/network KVM/lights-out cards
RUNAS	Interactive	Y	Nil
RUNAS/NETWORK	NewCredentials	Y	Clones LSA session, but uses new creds when connecting to network resources.
Remote Desktop	RemoteInteractive	Y	Nil
Remote Desktop Failure	RemoteInteractive	N	Only stored briefly
Net Use * \SERVER	Network	N	Nil
Net Use * \ SERVER /user	Network	N	Nil
MMC snap-ins to remote computer	Network	N	Nil
PowerShell WinRM	Network	N	e.g. Enter-PSSession SERVER
PowerShell WinRM with CredSSP	NetworkClearText	Y	e.g. New-PSSession SERVER -Authentication Credssp -Credential PWD
PsExec without explicit creds	Network	N	e.g. PsExec \SERVER cmd
PsExec with explicit creds	Network&Interactive	Y	PsExec \SERVER -u USER -p PWD cmd
Remote Registry	Network	N	Nil
Remote Desktop Gateway	Network	N	Authenticating to Remote Desktop Gateway
Scheduled Task	Batch	Y	Also saved as LSA secret on disk
Tools as Service	Service	Y	Also saved as LSA secret on disk
Vuln Scanners	Network	N	Most use Network logons; however, those that don't have the risk of creds on destination.
IIS "Basic Authentication"	NetworkCleartext	Y	Nil
IIS "Integrated Windows Authentication"	Network	N	NTLM/Kerberos Providers

Cached Credentials

Cached credentials are located within a system's registry at:

```
HKEY_LOCAL_MACHINE\SECURITY\Cache
```

Kerberos Tickets and Exposure

Special thanks to Cert EU (https://cert.europa.eu/static/WhitePapers/CERT-EU_SWP_17-002_Lateral_Movements.pdf) for this. When comparing Pass-the-Hash to Pass-the-Ticket attacks, the following key differences apply:

- Administrative privileges are required to steal credentials, but NOT to use an obtained Kerberos ticket.
- A password change does NOT make Kerberos tickets invalid. By default Kerberos has a max lifetime of 10hrs before the ticket must be renewed, and a max renewal time of 7 days after being granted.

Due to this disabling accounts may not be enough to prevent ongoing compromise, and you may have to purge the users kerberos ticket. Locate the user in question using 'sessions' and purge by specifying the user session prior to logging them off.

```
klist.exe sessions
klist purge -li 0x2e079217
query user
logoff [session id]
```

Windows Memory Forensics

Volatility 2.x Basics

(Note: Depending on what version of volatility you are using and where you may need to substitute volatility with vol.py if there's no alias setup)

Find out what profiles you have available

```
volatility --info
```

Find out the originating OS profile to be used from the memory dump.

```
volatility -f memorydump.mem imageinfo
volatility -f memorydump.mem kdbgscan
```

Determine what plugins are available for use.

```
volatility -f memorydump.mem --profile=[profilename] -h
```

Check what processes were running.

(Note: Any entires for svchost.exe should always have services.exe as a parent process and parameters such as /k should always be present)

```
volatility -f memorydump.mem --profile=[profilename] pslist
volatility -f memorydump.mem --profile=[profilename] psscan
volatility -f memorydump.mem --profile=[profilename] tree
```

Check what commands have been run and their output.

```
volatility -f memorydump.mem --profile=[profilename] cmdscan
volatility -f memorydump.mem --profile=[profilename] consoles
```

Dump process files which were running from memory.

```
volatility -f memorydump.mem --profile=[profilename] procdump -p [processid] --dump-dir=./
```

Dump the memory associated with a process file.

```
volatility -f memorydump.mem --profile=[profilename] memdump -p [processid] --dump-dir=./
```

Dump all cached files from memory.

```
volatility -f memorydump.mem --profile=[profilename] dumpfiles --dump-dir=./
```

Check what drivers or kernal modules were unloaded or hidden.

```
volatility -f memorydump.mem --profile=[profilename] modscan
```

Check what network connectivity has occurred.

```
volatility -f memorydump.mem --profile=[profilename] netscan
```

Check what network connectivity has occurred (Windows XP/Server 2003).

```
volatility -f memorydump.mem --profile=[profilename] connections
volatility -f memorydump.mem --profile=[profilename] conscan
volatility -f memorydump.mem --profile=[profilename] sockets
volatility -f memorydump.mem --profile=[profilename] sockscan
```

Check what information exists within registry from memory.

```
volatility -f memorydump.mem --profile=[profilename] hivelist
volatility -f memorydump.mem --profile=[profilename] hivescan
volatility -f memorydump.mem --profile=[profilename] hivedump --dump-dir=./
volatility -f memorydump.mem --profile=[profilename] userassist
volatility -f memorydump.mem --profile=[profilename] shellbags
volatility -f memorydump.mem --profile=[profilename] shimcache
volatility -f memorydump.mem --profile=[profilename] shimcachemem
```

Check internet explorer browsing history

```
volatility -f memorydump.mem --profile=[profilename] iehistory
```

Check for files in memory dump

```
volatility -f memorydump.mem --profile=[profilename] filescan
```

Dump files based on offset

```
volatility -f memorydump.mem --profile=[profilename] dumpfiles -Q [offsetfromfilescan] --dump-dir=./
```

Scan memory with Yara Rule

```
volatility -f memorydump.mem --profile=[profilename] yarascan -y rule.yara
```

Duplicate image space out as a raw DD file

(e.g. dump files such as hiberfil.sys memory from memory).

```
volatility -f memorydump.mem --profile=[profilename] imagecopy
```

Dump timelined artifacts from memory.

```
volatility -f memorydump.mem --profile=[profilename] timeliner
```

Detect persistence mechanisms using Winesap

- Research Paper (http://www.dfrws.org/sites/default/files/session-files/characteristics_and_detectability_of_windows_auto-start_extensibility_points_in_memory_forensics.pdf)
- Volatility Plugin - Winesap (https://gitlab.unizar.es/rrodrigu/winesap)

```
volatility -f memdump.mem --profile=[profile] autoruns
volatility --plugins=./winesap/plugin -f memdump.mem --profile=[profile] autoruns
volatility --plugins=./winesap/plugin -f memdump.mem --profile=[profile] autoruns --match
```

Compare memory dump to known good memory dump.

- csababarta plugins (https://github.com/csababarta/volatility_plugins)

volatility -f infected.mem –profile=[profilename] processbl -B clean.mem -U 2>/dev/null
volatility -f infected.mem –profile=[profilename] servicebl -B clean.mem -U 2>/dev/null
volatility -f infected.mem –profile=[profilename] driverbl -B clean.mem -U 2>/dev/null

Output visual .dot file to view process tree

```
volatility -f memorydump.mem --profile=[profilename] psscan --output=dot --output-file=psscan.dot
volatility -f memorydump.mem --profile=[profilename] tree --output=dot --output-file=pstree.dot
dot -Tpng pstree.dot -o pstree.png
dot -Tpng pstree.dot -o psscan.png
```

Volatility 3.x Basics

Note: Version 3 of Volatility (https://github.com/volatilityfoundation/volatility3/) was released in November 2019 which changes the Volatility usage and syntax. More information on V3 of Volatility can be found on ReadTheDocs (https://volatility3.readthedocs.io/en/latest/basics.html).

A list of common plugins are:

- linux.bash.Bash
- linux.check_afinfo.Check_afinfo
- linux.check_syscall.Check_syscall
- linux.elfs.Elfs
- linux.lsmod.Lsmod
- linux.lsof.Lsof
- linux.malfind.Malfind
- linux.proc.Maps
- linux.pslist.PsList
- linux.pstree.PsTree
- mac.bash.Bash
- mac.check_syscall.Check_syscall
- mac.check_sysctl.Check_sysctl
- mac.check_trap_table.Check_trap_table
- mac.ifconfig.Ifconfig
- mac.lsmod.Lsmod
- mac.lsof.lsof
- mac.malfind.Malfind
- mac.netstat.Netstat
- mac.proc_maps.Maps
- mac.psaux.Psaux
- mac.pslist.PsList
- mac.pstree.PsTree
- mac.tasks.Tasks
- mac.timers.Timers
- mac.trustedbsd.trustedbsd
- windows.cmdline.CmdLine
- windows.dlldump.DllDump
- windows.dlllist.DllList
- windows.driverirp.DriverIrp

- windows.driverscan.DriverScan
- windows.filescan.FileScan
- windows.handles.Handles
- windows.info.Info
- windows.malfind.Malfind
- windows.moddump.ModDump
- windows.modscan.ModScan
- windows.modules.Modules
- windows.mutantscan.MutantScan
- windows.poolscanner.PoolScanner
- windows.procdump.ProcDump
- windows.pslist.PsList
- windows.psscan.PsScan
- windows.pstree.PsTree
- windows.registry.certificates.Certificates
- windows.registry.hivedump.HiveDump
- windows.registry.hivelist.HiveList
- windows.registry.hivescan.HiveScan
- windows.registry.printkey.PrintKey
- windows.registry.userassist.UserAssist
- windows.ssdt.SSDT
- windows.statistics.Statistics
- windows.strings.Strings
- windows.symlinkscan.SymlinkScan
- windows.vaddump.VadDump
- windows.vadinfo.VadInfo
- windows.virtmap.VirtMap
- timeliner.Timeliner

Check Memory Image Information

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.info.Info
```

Check List of Kernel Drivers

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.modules.Modules
```

Check List of Kernel Drivers (incl previously unloaded and hidden)

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.modscan.ModScan
```

Dump List of Kernel Drivers to Files

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.moddump.ModDump
```

Dump List of Running Processes to Files

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.procdump.ProcDump
```

Check Process List of Running Processes

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.pslist.PsList
```

Check Process Tree of Running Processes

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.pstree.PsTree
```

Check Running Processes from EPROCESS blocks

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.psscan.PsScan
```

Check Running Processes for possible shellcode/injection via PAGE_EXECUTE_READWRITE

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.malfind.Malfind
```

Check processes and their command lines

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.cmdline.CmdLine
```

Check for files which exist in memory

```
/usr/bin/python3.6 vol.py -f /home/user/samples/mem.bin windows.filescan.FileScan
```

Miscellaneous Tools and Notes

Eric Zimmerman has excellent widely used libraries and tools (https://ericzimmerman.github.io/#!index.md)

RegRipper (https://github.com/keydet89/RegRipper3.0)

```
rip.pl -r NTUSER.DAT -f ntuser | less.
rip.pl -r SAM -f sam | less
rip.exe -l
rip.exe -r C:\Users\User\ntuser.dat -p userassist
```

Kape (https://learn.duffandphelps.com/kape)

Note: Video Tutorial (https://www.youtube.com/watch?v=L9H1uj2HSb8)

```
kape.exe --tsource C --target RegistryHives --tdest "[location]"
kape.exe --tsource \\server\directory --target !ALL --tdest "[location]" --vhdx LOCALHOST
```

Chainsaw (https://github.com/cyb3rpeace/chainsaw)

```
./chainsaw search mimikatz -i evtx_attack_samples/
./chainsaw search -t 'Event.System.EventID: =4104' evtx_attack_samples/
./chainsaw search -e "DC[0-9].insecurebank.local" evtx_attack_samples --json
./chainsaw hunt evtx_attack_samples/ -s sigma_rules/ --mapping mappings/sigma-event-logs-all.yml
```

Hayabusa (https://github.com/Yamato-Security/hayabusa)

```
hayabusa.exe csv-timeline -l -o results.csv
hayabusa.exe json-timeline -l -o results.json
hayabusa.exe logon-summary -l -o results.csv
hayabusa.exe csv-timeline -d D:\capture\EVTX_Files -o results.csv
hayabusa.exe json-timeline -d D:\capture\EVTX_Files -o results.json
hayabusa.exe logon-summary -d D:\capture\EVTX_Files -o results.csv
```

ShimCaheParser (https://github.com/mandiant/ShimCacheParser)

```
ShimCacheParser.py -h
ShimCacheParser.py -i SYSTEM --BOM
```

AppCompatCacheParser (https://ericzimmerman.github.io/#!index.md)

```
AppCompatCacheParser.exe --csv .\ -t
```

AmCacheParser (https://ericzimmerman.github.io/#!index.md)

```
AmcacheParser.exe --csv .\ -f .\Amcache.hve
```

Windows 10 Timeline Database Parser (https://ericzimmerman.github.io/#!index.md)

```
WxTCmd.exe -f "C:\Users\[username]\AppData\Local\ConnectedDevicesPlatform\L.
[username]\ActivitiesCache.db" --csv .
```

Bulk Extractor (http://downloads.digitalcorpora.org/downloads/bulk_extractor/)

```
bulk_extractor64.exe -o [outputdir] memdump.mem
```

ForensicDots (https://www.forensicdots.de/)

Note: Can be used to determine the Machine Identification Code (https://en.wikipedia.org/wiki/Machine_Identification_Code) of a Printer.

Cyber Chef (https://gchq.github.io/CyberChef/)

The Cyber Swiss Army Knife - a web app for encryption, encoding, compression and data analysis. Note: This was created by an analyst at the GCHQ which is part of the UKs National Cyber Security Centre. The source is actively maintained on Github (https://github.com/gchq/CyberChef)

URLScan (https://urlscan.io/)

A “sandbox for the web”. One of the most widely used, freely available URL scanners which provides a breakdown of technologies used on a website, safebrowsing score, screenshots, redirects, hosting information and certificates, and much more.

Redirect Tracker (https://www.websiteplanet.com/webtools/redirected/?url=)

Note: Thanks to Alejandra for sharing this.

Can be used to see any redirects that will likely occur with a description on each response code to help with analysis when accessing a URL e.g. a shortened URL such as bitly links.

Unshorten Me (https://unshorten.me/)

Can be used to see the end result of a shortened URL that will likely occur along with a screenshot.

Malware Hash Registry - Team Cymru (https://hash.cymru.com/)

Check malware hashes against the Team Cymru database

VirusTotal (https://www.virustotal.com/gui/home/search)

Check malware hashes against the VirusTotal database

MITRE ATT&CK® Framework (https://attack.mitre.org/)

Globally-accessible knowledge base of adversary tactics and techniques based on real-world observations.

LOLBAS Project (https://lolbas-project.github.io/)

The goal of the LOLBAS project is to document every binary, script, and library that can be used for Living Off The Land techniques.

GTFOBins (https://gtfobins.github.io/)

GTFOBins is a curated list of Unix binaries that can be used to bypass local security restrictions in misconfigured systems.

MalAPI.io (https://malapi.io/)

MalAPI.io maps Windows APIs to common techniques used by malware.

LOTS-Project (https://lots-project.com/)

Attackers are using popular legitimate domains when conducting phishing, C&C, exfiltration and downloading tools to evade detection. LOTS-Project aims to list websites that allow attackers to use their domain or subdomain.

Filesec.io (https://filesec.io/)

Repository of file extensions being used by attackers.

LOLDrivers.io (https://www.loldrivers.io/)

Sometimes referred to as BYOVD, Living Off The Land Drivers is a community-driven project that provides a curated list of all Windows drivers that have been found abused by adversaries to bypass security controls and execute malicious code.

EchoTrail (https://www.echotrail.io/insights)

Look up description, commonality, behavior, and more associated with an executable file name or hash

Winbindex (https://winbindex.m417z.com/)

An index of Windows binaries and files including: location, metadata of file, what build it's found in and much more

OSQuery (https://www.osquery.io/)

Open source agent used to query endpoints in a live response fashion.

Velociraptor (https://www.velocidex.com/docs/)

Digital forensic and Incident Response tool to enhance visibility on endpoints.

ViperMonkey (https://github.com/kirk-sayre-work/ViperMonkey)

Visual Basic emulation engine used to analyze and deobfuscate malicious VBA macros.

PcapXray (https://github.com/Srinivas11789/PcapXray)

Plot network traffic from pcap file into visual diagram

Parse and interpret VBA macros

```
vmonkey phishing.docm
```

Faster output

```
pypy vmonkey.py -s phishing.docm
```

Less verbose output

```
vmonkey -l warning phishing.docm
```

XLM Macro Deobfuscator (https://github.com/DissectMalware/XLMMacroDeobfuscator)

Parse and interpret hidden XLM Macros (Excel 4.0 Macros)

```
xlmdeobfuscator --file malware.xlsm
```

Note: Using a tool such as BiffView (http://b2xtranslator.sourceforge.net/snapshots/BiffView.zip) we're able to view BOUNDSHEET records to find hidden sheets. This is indicated by '02 01' hex values at the 5th and 6th offset.

Changing 02 to 00 makes these hidden spreadsheets visible without having to run any macros.

Mounting image files in linux

```
mkdir /mnt/windows
imageMounter.py
ImageMounter.py -s [imagefile] /mnt/windows
cd /mnt/windows
```

OR

```
mkdir /mnt/windows
sudo apt install libguestfs-tools
sudo virt-list-filesystems [vhdx file]
sudo guestmount -a [vhdx file] -m /dev/[filesystemabove] -r /mnt/windows -o allow_other
```

Mounting image files in Windows

- Arsenal Image Mounter (https://arsenalrecon.com/downloads/)
- FTK Imager (https://accessdata.com/product-download/ftk-imager-version-4-2-1)
- Autopsy (https://www.sleuthkit.org/autopsy/download.php)

Unpack binary packed with UPX

```
upx -d PackedProgram.exe
```

Scan exchange for phishing emails

Disclaimer: Always test before running against live systems. For those running Office365 this documentation (https://docs.microsoft.com/en-us/office365/securitycompliance/search-for-and-delete-messages-in-your-organization) may be more useful.

```
# This is used to authenticate yourself and connect to the exchange server
$UserCredential = Get-Credential
$Session = New-PSSession -ConfigurationName Microsoft.Exchange -ConnectionUri
http://EXCHANGEHOSTFQDN/PowerShell/ -Credential $UserCredential
Import-PSSession $Session -DisableNameChecking

# This is used to confirm the mailboxes accessible and modules available
Get-Mailbox
Get-Module

# This is used to remove emails from a mailbox and move them to an administrator mailbox as a backup
Search-Mailbox -Identity "NAME" | Search-Mailbox -SearchQuery 'Subject:"SUBJECT LINE"' -TargetMailbox
"ADMINBACKUPMAILBOX" -TargetFolder "BACKUPFOLDER" -DeleteContent

# This is used to run a report on anyone who received an email with a malicious attachment and log this
information in an administrator mailbox
Get-Mailbox -ResultSize unlimited | Search-Mailbox -SearchQuery attachment:trojan* -TargetMailbox
"ADMINBACKUPMAILBOX" -TargetFolder "BACKUPFOLDER" -LogOnly -LogLevel Full

# This is used to disconnect from the established powershell session
Remove-PSSession $Session
```

Common DLL Information

DLL	Description
Kernel32.dll	(Windows Kernel) This is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware.
Advapi32.dll	(Advanced API) This DLL provides access to advanced core Windows components such as the Service Manager and Registry.
Ntdll.dll	(NT Layer) This DLL is the interface to the Windows kernel. Executables rarely import this file directly, although it is always imported indirectly by Kernel32.dll. If an executable deliberately imports this, it means that the author wanted to use functionality not normally available to Windows programs. Some tasks, such as hiding functionality or manipulating processes, will use this interface.
User32.dll	(Windows User) This DLL contains all the user interface components, such as buttons, scroll bars, and components for controlling and responding to user actions.
Wininet.dll	(Windows Internet API) This DLL contains high level networking functions. These implement protocols such as FTP, HTTP, and NTP.
Gdi32.dll	(Graphics Device Interface) This DLL contains functions used for displaying and manipulating graphics.
WSock32.dll and Ws2_32.dll	(Windows Sockets API) These are networking DLLs. A program that accesses either of these will likely connect to a network or perform network related tasks.
urlmon.dll	(OLE32 Extensions for Win32) This DLL contains functions for COM-based networking. It is used by Internet Explorer and many other applications similar to the Windows Sockets API; however, it is at a higher level than the Windows Sockets API and Windows Internet API and has many elements prefilled.

- When analysing a binary, small amount of strings present and minimal imported functions help confirm that it is a packed binary.

Show processes with networking functions

```
tasklist /m WSock32.dll
tasklist /m Ws2_32.dll
tasklist /m Wininet.dll
tasklist /m winhttp.dll
gps | ?{$_.Modules -like '*WSock32.dll*' -OR $_.Modules -like '*Ws2_32.dll*' -OR $_.Modules -like
'*Wininet.dll*' -OR $_.Modules -like '*winhttp.dll*' }|FL Id, ProcessName
```

Show processes importing the Remote Access API

```
tasklist /m RASAPI32.dll
```

Show processes importing the task scheduler API

```
tasklist /m taskschd.dll
tasklist /m mstask.dll
```

Show processes importing the Windows Media Instrumentation API

```
tasklist /m wbem*
tasklist /m wmi*
```

Windows Memory Analysis (Example Process with Volatility)

Identify memory OS information

```
volatility -f memorydump.mem imageinfo
```

Identify suspicious running processes

```
volatility -f memorydump.mem --profile=[profilename] pstree
```

Show suspicious running processes based on names.

```
volatility -f memorydump.mem --profile=[profilename] pstree | egrep 'winlogon|lsass|services'
volatility -f memorydump.mem --profile=[profilename] psscan
```

Show any malicious or suspicious processes requiring investigation

```
volatility -f memorydump.mem --profile=[profilename] malfind
```

Show any Process Hollowing (Hollow Process Injection)

```
volatility -f memorydump.mem --profile=[profilename] hollowfind
```

Dump suspicious process executables from memory

```
volatility -f memorydump.mem --profile=[profilename] procdump -p [processid] --dump-dir=./
```

Parse the Master File Table

```
volatility -f [memoryDump] mftparser -C --output-file=output.txt
```

Reassemble raw hex of file under $DATA back into original file from dump.raw file.

```
xxd -r dump.raw > [filename.originalextension]
```

Compare hashes with known detections e.g. VirusTotal.

```
sha256 [filename]
https://www.virustotal.com
```

Create a timeline of events.

```
volatility -f memorydump.mem --profile=[profilename] timeliner
volatility -f memorydump.mem --profile=[profilename] timeliner --hive=SECURITY
volatility -f memorydump.mem --profile=[profilename] timeliner --type=Registry
```

Windows Memory Analysis (dump) using Windbg

Using Comaeio SwishDbgExt (https://github.com/comaeio/SwishDbgExt) you are able to better analyse Windows Crash (DMP) files using Windbg. To do this, download the latest release, run windbg, load the correct dll and then run a command. At the time of writing there are:

```
!load X:\FullPath\SwishDbgExt.dll

!help             - Displays information on available extension commands
!ms_callbacks     - Display callback functions
!ms_checkcodecave - Look for used code cave
!ms_consoles      - Display console command's history
!ms_credentials   - Display user's credentials (based on gentilwiki's mimikatz)
!ms_drivers       - Display list of drivers
!ms_dump          - Dump memory space on disk
!ms_exqueue       - Display Ex queued workers
!ms_fixit         - Reset segmentation in WinDbg (Fix "16.kd>")
!ms_gdt           - Display GDT
!ms_hivelist      - Display list of registry hives
!ms_idt           - Display IDT
!ms_lxss          - Display lsxx entries
!ms_malscore      - Analyze a memory space and returns a Malware Score Index (MSI) - (based on Frank
Boldewin's work)
!ms_mbr           - Scan Master Boot Record (MBR)
!ms_netstat       - Display network information (sockets, connections, ...)
!ms_object        - Display list of object
!ms_process       - Display list of processes
!ms_readkcb       - Read key control block
!ms_readknode     - Read key node
!ms_readkvalue    - Read key value
!ms_regcheck      - Scan for suspicious registry entries
!ms_scanndishook  - Scan and display suspicious NDIS hooks
!ms_services      - Display list of services
!ms_ssdt          - Display service descriptor table (SDT) functions
!ms_store         - Display information related to the Store Manager (ReadyBoost)
!ms_timers        - Display list of KTIMER
!ms_vacbs         - Display list of cached VACBs
!ms_verbose       - Turn verbose mode on/off
!ms_yarascan      - Scan process memory using yara rules
```

Other inbuilt WindBG commands (Useful for single-process memory dump analysis)

Get Address memory information

```
!address
```

Get All Thread Information

```
~*
```

Get Current Thread Information

```
~.
```

Get information from each frame of thread stack

```
!for_each_frame db
!for_each_frame dv -t -v
!for_each_frame dt -b
```

Recursively dump PEB of process

```
dt ntdll!_PEB @$peb -r
```

Create Dump

```
.dump D:\Downloads\test\test.dmp
```

DLL/Module Information

```
!dlls
lm
x *!
!lmi [ModuleName]
```

Get stacks for all threads

```
!uniqstack
```

Search Heap for String to view in memory

```
!address -f:Heap -c:"s -u %1 %2 \"http\""
```

Search for String across all memory

```
s -a 0 L?FFFFFFFF "StringToFind"
s -u 0 L?FFFFFFFF "StringToFind"
```

List nearest module to address

```
ln [address]
ln 7c951782
```

Quick analysis

```
!analyze -v
```

Normal Process Relationship Hierarchy (Geneology)

Excellent SANS Reference (https://digital-forensics.sans.org/media/DFPS_FOR508_v4.6_4-19.pdf)

Old:

System

- smss.exe
  - winlogon.exe (upon smss.exe exiting)
    - userinit.exe
      - explorer.exe (upon userinit.exe exiting)
  - wininit.exe (upon smss.exe exiting)
    - lsass.exe
    - services.exe
      - svchost.exe
      - taskhost.exe
  - csrss.exe

Windows 10:

System

- smss.exe
    - winlogon.exe (upon smss.exe exiting)
        - userinit.exe
            - explorer.exe (upon userinit.exe exiting)
    - wininit.exe (upon smss.exe exiting)
        - lsass.exe
        - lsaiso.exe (credential guard only)
        - services.exe
            - svchost.exe
                - taskhostw.exe
                - runtimebroker.exe
    - csrss.exe

Extra notes

Be mindful of the below:

- svchost.exe should always have services.exe pid as ppid and a service associated with it.
- there should never be more than 1 lsass.exe process.
- lsass.exe should always have a parent of winlogon.exe (WinXP and older) or Wininit.exe (Vista or newer).
- pslist and pstree follow a 'Double Linked List' which malware can 'unlink' itself from thus hiding the process.
- psscan looks instead for 'EPROCESS blocks' which is memory associated with a windows process.
- Discrepencies between these 2 areas can indicate the process hollowing has occurred.
    - VAD = Virtual Address Descriptor which lives in kernel memory.
    - PEB = Process Environment Block which lives in process memory.
- PAGE_EXECUTE_READWRITE protection indicates memory marked as executable, which may indicate potential shellcode was used or injected.
- Process hollowing essentially pauses and duplicates a legitimate process, replaces the executable memory with something malicious, and then resumes the process. Process Injection on the other hand injects malicious code into an already running process which causes that process to execute the code.

Linux Cheat Sheet

Dumping Memory

```
dd if=/dev/kmem of=/root/kmem
dd if=/dev/mem of=/root/mem
```

LiME (https://github.com/504ensicsLabs/LiME/releases)

```
sudo insmod ./lime.ko "path=./Linmen.mem format=raw"
```

LinPMem (https://github.com/Velocidex/c-aff4/releases/)

```
./linpmem -o memory.aff4
./linpmem memory.aff4 -e PhysicalMemory -o memory.raw
```

Taking Image

```
fdisk -l
dd if=/dev/sda1 of=/[outputlocation]
```

Misc Useful Tools

FastIR (https://github.com/SekoiaLab/Fastir_Collector_Linux)

```
python ./fastIR_collector_linux.py
```

LinEnum (https://github.com/rebootuser/LinEnum/blob/master/LinEnum.sh)

```
./linenum.sh
./linenum.sh -t
```

Live Triage

System Information

```
date
uname –a
lsb_release -a
hostname
cat /proc/version
lsmod
```

Account Information

```
cat /etc/passwd
cat /etc/shadow
cat /etc/sudoers
cat /etc/sudoers.d/*
cut -d: -f1 /etc/passwd
getent passwd | cut -d: -f1
compgen -u
```

Current user

```
whoami
who
```

Last logged on users

```
last
lastb
cat /var/log/auth.log
```

Initialisation Files

```
cat /etc/bash.bashrc
cat ~/.bash_profile
cat ~/.bashrc
```

Environment and Startup Programs

```
env
cat /etc/profile
ls /etc/profile.d/
cat /etc/profile.d/*
```

Scheduled Tasks

```
ls /etc/cron.*
ls /etc/cron.*/*
cat /etc/cron.*/*
cat /etc/crontab
crontab -l
```

Commands That Can Be Run As Root or User

```
sudo -l
```

SSH Keys and Authorised Users

```
cat /etc/ssh/sshd_config
```

Note: This specifies where the SSH daemon will look for keys. Generally this will be as below.

```
ls /home/*/.ssh/*
cat /home/*/.ssh/id_rsa.pub
cat /home/*/.ssh/authorized_keys
```

Sudoers File (who who can run commands as a different user)

```
cat /etc/sudoers
```

Configuration Information

```
ls /etc/*.d
cat /etc/*.d/*
```

Network Connections / Socket Stats

```
netstat
netstat -apetul
netstat -plan
netstat -plant
netstat -naote
ss
ss -l
ss -ta
ss -tp
```

DNS Information for Domain

```
dig www.jaiminton.com a
dig www.jaiminton.com any
dig www.jaiminton.com ns
dig www.jaiminton.com soa
dig www.jaiminton.com hinfo
dig www.jaiminton.com txt
dig +short www.jaiminton.com
```

IPs Allowed to Perform Domain Transfer

```
cat /etc/bind/named.conf.local
```

Specify IP To Use For Domain Transfer

```
dig @127.0.0.1 domain.here axfr -b [IP]
```

IP Table Information

```
ls /etc/iptables
cat /etc/iptables/*.v4
cat /etc/iptables/*.v6
iptables -L
```

Use IPTables For Filtering

AndreaFortuna Cheatsheet (https://andreafortuna.org/2019/05/08/iptables-a-simple-cheatsheet/)

Network Configuration

```
ifconfig -a
```

Difference Between 2 Files

```
diff [file1] [file2]
```

Browser Plugin Information

```
ls -la ~/.mozilla/plugins
ls -la /usr/lib/mozilla/plugins
ls -la /usr/lib64/mozilla/plugins
ls -la ~/.config/google-chrome/Default/Extensions/
```

Kernel Modules and Extensions/

```
ls -la /lib/modules/*/kernel/*
```

File Permissions

```
-rw-r-x-wt 1 fred fred 0 Aug 10 2019 /home/fred/malware
|[-][-][-]- [---][---]
| |  |  | |   |    |
| |  |  | |   |    *-----------------> 7. Group
| |  |  | |   *----------------------> 6. Owner
| |  |  | *--------------------------> 5. Alternate Access Method
| |  |  *---------------------------> 4. Others Permissions
| |  *-----------------------------> 3. Group Permissions
| *-------------------------------> 2. Owner Permissions
*---------------------------------> 1. File Type
```

File Permissions in Linux (https://www.guru99.com/file-permissions.html)

Decode base64 Encoded File

```
base64 -d [filename]
echo [b64stream] | base64 -d
```

Process Information

ps [simple list output threads misc all]'

```
ps -s
ps -l
ps -o
ps -t
ps -m
ps -a
ps -ax
top
```

Size Of File (Bytes)

```
wc -c [file]
```

IP Making Most Requests in Access Log

```
cut -d " " -f 1 access.log | sort | uniq -c | sort -n -k 1,1
```

Count of Unique IPs in Access Log

```
cut -d " " -f 1 access.log | sort -u | wc -l
```

Unique User Agents in Access Log

```
awk -F \" '{ print $6 }' access.log | sort -u
```

Most Requested URLs For POST Request in Access Log

```
awk -F \" '{ print $2 }' access.log | grep "POST" | sort | uniq -c | sort -n -k 1,1
```

Lines In File

```
wc -l [file]
```

Search files recursively in directory for keyword

```
grep -H -i -r "password" /
```

Process Tree

```
ps -auxwf
```

Open Files and space usage

```
lsof
du
```

Pluggable Authentication Modules (PAM)

```
cat /etc/pam.d/sudo
cat /etc/pam.conf
ls /etc/pam.d/
```

Disk / Partition Information

```
fdisk -l
```

Fulle Path For Command in environment variables

```
which [softwarename]
```

System Calls / Network Traffic (https://bytefreaks.net/gnulinux/how-to-capture-all-network-traffic-of-a-single-process)

```
strace -f -e trace=network -s 10000 [PROCESS WITH ARGUMENTS];
strace -f -e trace=network -s 10000 -p [PID];
```

Strings Present In File

```
strings [filepath]
strings -e b [filepath]
```

Note: Below material with thanks to Craig Rowland - Sandfly Security (https://blog.apnic.net/2019/10/14/how-to-basic-linux-malware-process-forensics-for-incident-responders/)

Detailed Process Information

```
ls -al /proc/[PID]
```

Note:

- CWD = Current Working Directory of Malware
- EXE = Binary location and whether it has been deleted
- Most Common Timestamp = When process was created

Recover deleted binary which is currently running

```
cp /proc/[PID]/exe /[destination]/[binaryname]
```

Capture Binary Data for Review

```
cp /proc/[PID]/ /[destination]/[PID]/
```

Binary hash information

```
sha1sum /[destination]/[binaryname]
md5sum /[destination]/[binaryname]
```

Process Command Line Information

```
cat /proc/[PID]/cmdline
cat /proc/[PID]/comm
```

Note:

- Significant differences in the above 2 outputs and the specified binary name under /proc/[PID]/exe can be indicative of malicious software attempting to remain undetected.

Process Environment Variables (incl user who ran binary)

```
strings /proc/[PID]/environ
cat /proc/[PID]/environ
```

Process file descriptors/maps (what the process is 'accessing' or using)

```
ls -al /proc/[PID]/fd
cat /proc/[PID]/maps
```

Process stack/status information (may reveal useful elements)

```
cat /proc/[PID]/stack
cat /proc/[PID]/status
```

Deleted binaries which are still running

```
ls -alr /proc/*/exe 2> /dev/null |  grep deleted
```

Process Working Directories (including common targeted directories)

```
ls -alr /proc/*/cwd
ls -alr /proc/*/cwd 2> /dev/null | grep tmp
ls -alr /proc/*/cwd 2> /dev/null | grep dev
ls -alr /proc/*/cwd 2> /dev/null | grep var
ls -alr /proc/*/cwd 2> /dev/null | grep home
```

Using JQ To Analyse JSON

```
cat people.json | jq '.[] | select((.name.first == "Fred") or (.name.last == "John"))'
cat people.json | jq '.[] | select((.name.first == "Fred") or (.name.last == "John"))'|@csv
```

Hidden Directories and Files

```
find / -type d -name ".*"
```

Immutable Files and Directories (Often Suspicious)

```
lsattr / -R 2> /dev/null | grep "\----i"
```

SUID/SGID and Sticky Bit Special Permissions

```
find / -type f \( -perm -04000 -o -perm -02000 \) -exec ls -lg {} \;
```

File and Directories with no user/group name

```
find / \( -nouser -o -nogroup \) -exec ls -lg  {} \;
```

File types in current directory

```
file * -p
```

Executables on file system

```
find / -type f -exec file -p '{}' \; |  grep ELF
```

Hidden Executables on file system

```
find / -name ".*" -exec file -p '{}' \; | grep ELF
```

Files modified within the past day

```
find / -mtime -1
```

Find files for a particular user

```
find /home/ -user fred -type f
```

Persistent Areas of Interest

```
/etc/rc.local
/etc/initd
/etc/rc*.d
/etc/modules
/etc/cron*
/var/spool/cron/*
/usr/lib/cron/
/usr/lib/cron/tabs
```

Audit Logs

```
ls -al /var/log/*
ls -al /var/log/*tmp
utmpdump /var/log/btmp
utmpdump /var/run/utmp
utmpdump /var/log/wtmp
```

Installed Software Packages

```
ls /usr/bin/
ls /usr/local/bin/
```

MacOS Cheat Sheet

- Sarah Edwards (https://twitter.com/iamevltwin)
- Mac4n6 (https://www.mac4n6.com/)
- SANS FOR518 Reference Sheet (https://sansorg.egnyte.com/dl/zngSJguNqj)
- Mac OS X 10.9 Forensics Wiki (https://forensics.wiki/mac_os_x_10.9_artifacts_location/)
- Mac OS X Forensics Wiki (https://forensics.wiki/mac_os_x/)
- Mac OS X Forensics Artifacts Spreadsheet (https://docs.google.com/spreadsheets/d/1X2Hu0NE2ptdRj023OVWIGp5dqZOw-CfxHLOW_GNGpX8/edit#gid=1317205466)

Dumping Memory

OSXPMem (https://github.com/wrmsr/pmem/tree/master/OSXPMem)

MacPmem (https://github.com/google/rekall/releases/download/1.7.2rc1/rekall-OSX-1.7.2rc1.zip)

```
sudo kextload MacPmem.kext
sudo dd if=/dev/pmem of=memorydump.raw
```

Live Mac IR / Triage

System Information

```
date
sw_vers
uname -a
hostname
cat /System/Library/CoreServices/SystemVersion.plist
cat /private/var/log/daily.out
cat /Library/preferences/.Globalpreferences.plist
sysadminctl -filesystem status
```

Network Connections

```
netstat -an
netstat -anf
lsof -i
```

Routing Table

```
netstat -rn
```

Network Information

```
arp -an
ndp -an
ifconfig
```

Open Files

```
lsof
```

File System Usage

```
sudo fs_usage
sudo fs_usage [process]
sudo fs_usage -f network
sudo fs_usage pid [PID]
```

Bash History

```
cat ~/.bash_history
history
```

User Logins

```
who -a
w
last
```

List of users

```
dscl . -list /Users
```

User information

```
dscl . -read /Users/[username]
dscl . -readall /Users
```

Running Processes

```
ps aux
```

Extended Running Process Information

```
launchctl procinfo [PID]
```

System Profiler

```
system_profiler -xml -detaillevel full > systemprofiler.spx
```

Persistent Locations

Quick Overview (KnockKnock) (https://www.objective-see.com/products/knockknock.html)

```
./KnockKnock.app/Contents/MacOS/KnockKnock -whosthere > /path/to/some/file.json
```

XPC Services

```
ls Applications/[application].app/Contents/XPCServices/
cat Applications/[application].app/Contents/XPCServices/*.xpc/Contents/Info.plist
ls ~/System/Library/XPCServices/
```

Launch Agents & Launch Daemons

```
ls /Library/LaunchAgents/
ls /System/Library/LaunchAgents/
ls /System/Library/LaunchDaemons/
ls /Library/LaunchDaemons/
ls /users/*/Library/LaunchAgents/
launchctl list
launchctl print gui/[UID]
launchctl print system
```

LoginItems

```
cat ~/Library/Preferences/com.apple.loginitems.plist
ls [application].app/Contents/Library/LoginItems/
```

Disable Persistent Launch Daemon

```
sudo launchctl unload -w /Library/LaunchDaemons/[name].plist
sudo launchctl stop /Library/LaunchDaemons/[name].plist
```

Web Browsing Preferences

```
cat ~/Library/Preferences/com.apple.Safari.plist
ls ~/Library/Application Support/Google/Chrome/Default/Preferences
ls ~/Library/Application Support/Firefox/Profiles/********.default/prefs.js
```

Safari Internet History

```
cat ~/Library/Safari/Downloads.plist
cat ~/Library/Safari/History.plist
cat ~/Library/Safari/LastSession.plist
ls ~/Library/Caches/com.apple.Safari/Webpage Previews/
sqlite3 ~/Library/Caches/com.apple.Safari/Cache.db
```

Chrome Internet History

```
ls ~/Library/Application Support/Google/Chrome/Default/History
ls ~/Library/Caches/Google/Chrome/Default/Cache/
ls ~/Library/Caches/Google/Chrome/Default/Media Cache/
```

Firefox Internet History

```
sqlite3 ~/Library/Application Support/Firefox/Profiles/********.default/places.sqlite
sqlite3 ~/Library/Application Support/Firefox/Profiles/********.default/downloads.sqlite
sqlite3 ~/Library/Application Support/Firefox/Profiles/********.default/formhistory.sqlite
ls ~/Library/Caches/Firefox/Profiles/********.default/Cache
```

Apple Email

```
cat ~/Library/Mail/V2/MailData/Accounts.plist
ls ~/Library/Mail/V2/
ls ~/Library/Mail Downloads/
ls ~/Downloads
cat ~/Library/Mail/V2/MailData/OpenAttachments.plist
```

Temporary / Cached

```
ls /tmp
ls /var/tmp
ls /Users/[user]/Library/Caches/Java/tmp
ls /Users/[user]/Library/Caches/Java/cache
        /Applications/Utilities/Java Preferences.app
```

System and Audit Logs

```
ls /private/var/log/asl/
ls /private/var/audit/
cat /private/var/log/appfirewall.log
ls ~/Library/Logs
ls /Library/Application Support/[app]
ls /Applications/
ls /Library/Logs/
```

Specific Log Analysis

```
bzcat system.log.1.bz2
system.log.0.bz2 >> system_all.log
cat system.log >> system_all.log
syslog -f [file]
syslog –T utc –F raw –d /asl
syslog -d /asl
praudit –xn /var/audit/*
sudo log collect
log show
log stream
```

Files Quarantined

```
ls ~/Library/Preferences/com.apple.LaunchServices.QuarantineEvents.V2
ls ~/Library/Preferences/com.apple.LaunchServices.QuarantineEvents
```

User Accounts / Password Shadows

```
ls /private/var/db/dslocal/nodes/Default/users/
ls /private/var/db/shadow/[User GUID]
```

Pluggable Authentication Modules (PAM)

```
cat /etc/pam.d/sudo
cat /etc/pam.conf
ls /etc/pam.d/
```

File Fingerprinting/Reversing

```
file [filename]
xxd [filename]
nm -arch x86_64 [filename]
otool -L [filename]
sudo vmmap [pid]
sudo lsof -p [pid]
xattr –xl [file]
```

Connected Disks and Partitions

```
diskutil list
diskutil info [disk]
diskutil cs
ap list
gpt –r show
gpt -r show -l
```

Disk File Image Information

```
hdiutil imageinfo *.dmg
```

User Keychain Information

```
security list-keychains
security dump-keychains -d [keychain]
```

Spotlight Metadata

```
mdimport –X | -A
mdls [file]
```

Extract download location from Extended Attribute

Note: This is essentially the ‘ADS’ of the MacOS world.

```
xattr -p com.apple.metadata:kMDItemWhereFroms filename.dmg | xxd -r -p | plutil -p -
```

Locate historical file names from Extended Attribute

```
xattr -p com.apple.genstore.origdisplayname filename
```

Bonus Valuable Links

- MITRE ATT&CK™ (https://attack.mitre.org/)
- MITRE Cyber Analytics Repository (https://car.mitre.org/)
- Atomic Red Team (https://redcanary.com/atomic-red-team/)
- Awesome Incident Response (https://github.com/meirwah/awesome-incident-response)
- Awesome Forensics (https://github.com/Cugu/awesome-forensics)
- Mac OSX Forensics (https://github.com/n0fate/mac-osx-forensics)
- Unofficial Mac 4n6 Resources (https://github.com/pstirparo/mac4n6)
- Apple macOS command line (OS X bash) (https://ss64.com/osx/)
- Mac4n6 (https://www.mac4n6.com/)

Special Thanks

- 13Cubed (https://www.youtube.com/13cubed)
- John Strand Windows Live Forensics (https://www.youtube.com/watch?v=HcUMXxyYsnw)
- Blue Team Field Manual - Alan White & Ben Clark
- DFIR Training Windows Registry (https://www.dfir.training/resources/downloads/windows-registry)
- Commandlinekungfu (http://blog.commandlinekungfu.com/)
- Microsoft Audit Logon Events (https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc787567(v=ws.10))
- Windows Dev Win32_LogonSession (https://docs.microsoft.com/en-us/windows/desktop/CIMWin32Prov/win32-logonsession)
- Black Hills Information Security Windows Memory Forensics (https://www.youtube.com/watch?v=cYphLiySAC4)
- Forensics Wiki (https://forensicswiki.org/wiki/Main_Page)
- Wireshark Wiki (https://wiki.wireshark.org/DisplayFilters)
- Microsoft Sysmon (https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon)
- ACSC Github (https://github.com/AustralianCyberSecurityCentre/windows_event_logging)
- Windows Defender Docs (https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-antivirus/command-line-arguments-windows-defender-antivirus)
- Mikefrobbins (https://mikefrobbins.com/2013/12/19/using-powershell-to-remove-phishing-emails-from-user-mailboxes-on-an-exchange-server/)
- Microsoft Office365 (https://docs.microsoft.com/en-us/office365/securitycompliance/search-for-and-delete-messagesadmin-help)
- Microsoft Exchange (https://docs.microsoft.com/en-us/powershell/module/exchange/mailboxes/search-mailbox)
- Microsoft Threat Protection (https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4768)
- ADsecurity (https://adsecurity.org/?page_id=1821)
- Windows cmd fu (https://twitter.com/wincmdfu)
- Crowdstrike (https://www.crowdstrike.com/blog/kovter-killer-how-to-remediate-the-apt-of-clickjacking/)
- Technet Blog (https://blogs.technet.microsoft.com/tip_of_the_day/2015/03/28/tip-of-the-day-reading-the-usn-journal/)
- IETF RFC3227 (https://tools.ietf.org/html/rfc3227#section-2.1)

- Cybereason Adobe Worm (https://www.cybereason.com/blog/adobe-worm-faker-uses-lolbins-and-dynamic-techniques-to-deliver-customized-payloads)
- Melanijan93 Windows 10 mail forensics (https://medium.com/@melanijan93/windows-10-mail-app-forensics-39025f5418d2)
- Cybereason Trickbot (https://www.cybereason.com/blog/triple-threat-emotet-deploys-trickbot-to-steal-data-spread-ryuk-ransomware)
- Bryan Ambrose (https://www.datadigitally.com/2019/06/retrieving-memory-image-remotely.html)
- Lee Holmes (https://twitter.com/Lee_Holmes)
- Florian Roth (https://www.bsk-consulting.de/2014/08/28/scan-system-files-manipulations-yara-inverse-matching-22/)
- Matt Graeber (https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf)
- Vasily Gusev (https://social.technet.microsoft.com/wiki/contents/articles/4242.wmi-discovery-using-powershell.aspx)
- MS Docs 4672 (https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4672)
- Kris (https://www.pdq.com/blog/modifying-the-registry-users-powershell/)
- Barnaby Skeggs (https://twitter.com/barnabyskeggs)
- Sarah Edwards (https://twitter.com/iamevltwin)
- Fahim Hossain (https://medium.com/@fahimhossain_16989/adding-startup-scripts-to-launch-daemon-on-mac-os-x-sierra-10-12-6-7e0318c74de1)
- Surendra Anne (https://www.linuxnix.com/linux-directory-structure-explainedetc-folder/)
- SANS Hunt Evil (https://digital-forensics.sans.org/media/SANS_Poster_2018_Hunt_Evil_FINAL.pdf)
- Craig Rowland - Sandfly Security (https://blog.apnic.net/2019/10/14/how-to-basic-linux-malware-process-forensics-for-incident-responders/)
- Habibar Rahmen - MSDN Blog (https://blogs.msdn.microsoft.com/servergeeks/2014/10/14/active-directory-files-and-their-functions/)
- Mari DeGrazia (https://az4n6.blogspot.com/2014/10/timestomp-mft-shenanigans.html)
- Markus Piéton (https://www.a12d404.net/windows/2019/10/30/schedsvc-persist-without-task.html)
- Samir (https://github.com/sbousseaden/Slides/blob/master/Windows%20DFIR%20Events.pdf)
- Samir - Persistence (https://twitter.com/SBousseaden/status/1243711784101515274/photo/1)
- FireEye (https://www.fireeye.com/blog/threat-research/2019/12/breaking-the-rules-tough-outlook-for-home-page-attacks.html)

- Microsoft-Mitigating Pass-the-Hash Attacks (https://www.microsoft.com/en-us/download/details.aspx?id=36036)
- Brent Muir - Windows 10 Forensics (https://www.slideshare.net/bsmuir/windows-10-forensics-os-evidentiary-artefacts)
- Mike Carey - Locked File Access Using ESENTUTL.exe (https://dfironthemountain.wordpress.com/2018/12/06/locked-file-access-using-esentutl-exe/)
- BornToBeRoot (https://github.com/BornToBeRoot)
- Malware-Traffic-Analysis - Brad Duncan (https://malware-traffic-analysis.net/tutorials/index.html)
- ADsecurity - Detecting Kerberoasting Activity (https://adsecurity.org/?p=3458)
- eladshamir - Internal-Monologue (https://github.com/eladshamir/Internal-Monologue)
- Jared Atkinson - Defenders think in graphs too (https://posts.specterops.io/defenders-think-in-graphs-too-part-1-572524c71e91)
- Adam Chester (https://blog.xpnsec.com/exploring-mimikatz-part-1/)
- Diana Lopera - Trustwave (https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/more-excel-4-0-macro-malspam-campaigns/)
- svch0st (https://medium.com/@7a616368/can-you-track-processes-accessing-the-camera-and-microphone-7e6885b37072)
- AustralianCyberSecurityCentre Advisory 2020-004 (https://www.cyber.gov.au/acsc/view-all-content/advisories/advisory-2020-004-remote-code-execution-vulnerability-being-actively-exploited-vulnerable-versions-telerik-ui-sophisticated-actors)
- Microsoft - Where's the Macro? (https://www.microsoft.com/security/blog/2016/06/14/wheres-the-macro-malware-author-are-now-using-ole-embedding-to-deliver-malicious-files/?source=mmpc)

Tags: Cheatsheet DFIR Forensics Incident IR Response

Categories: Cheatsheet DFIR

Updated: March 23, 2024