Blue Team Notes

A collection of one-liners, small scripts, and some useful tips for blue team work.

I've included screenshots where possible so you know what you're getting.

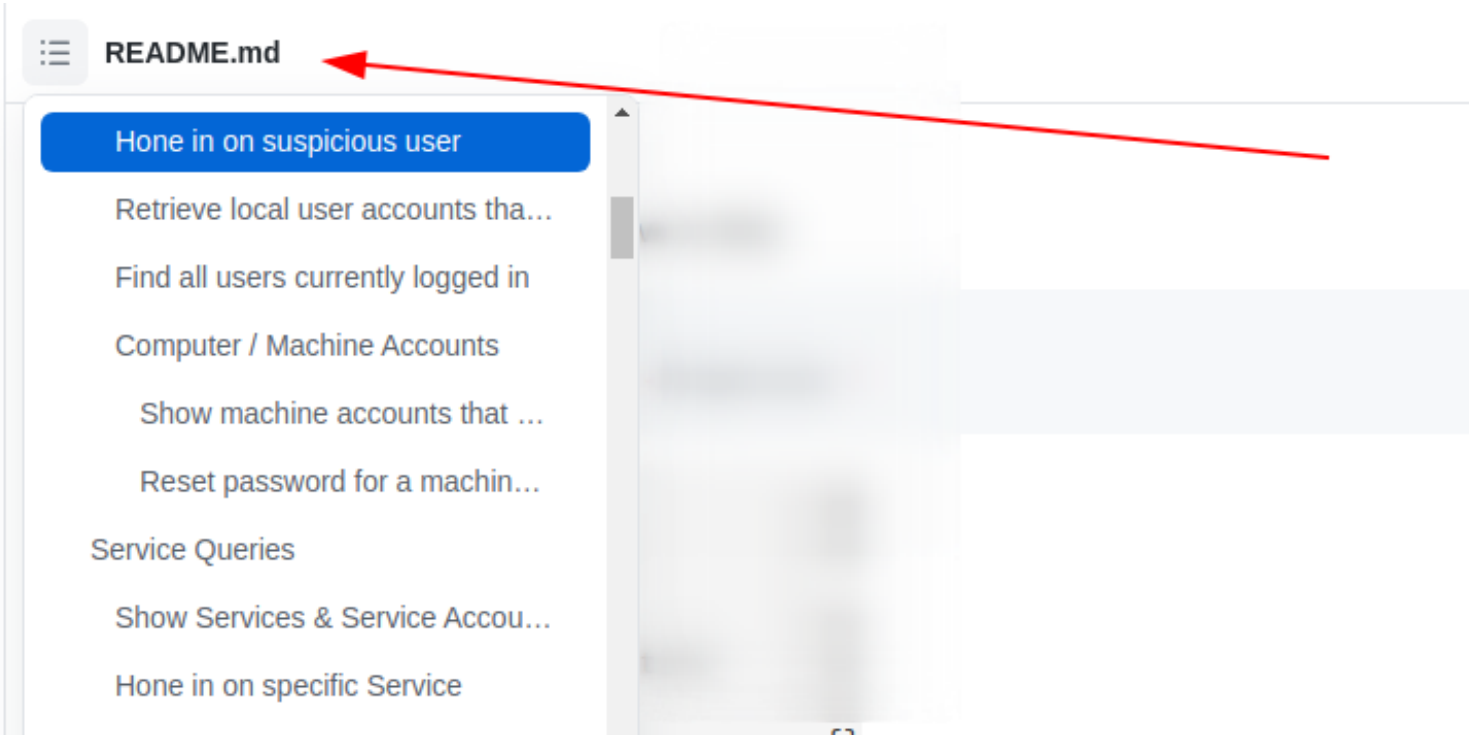
Did the Notes help?

I hope the Blue Team Notes help you catch an adversary, thwart an attack, or even just helps you learn. If you've benefited from the Blue Team Notes, would you kindly consider making a donation to one or two charities.

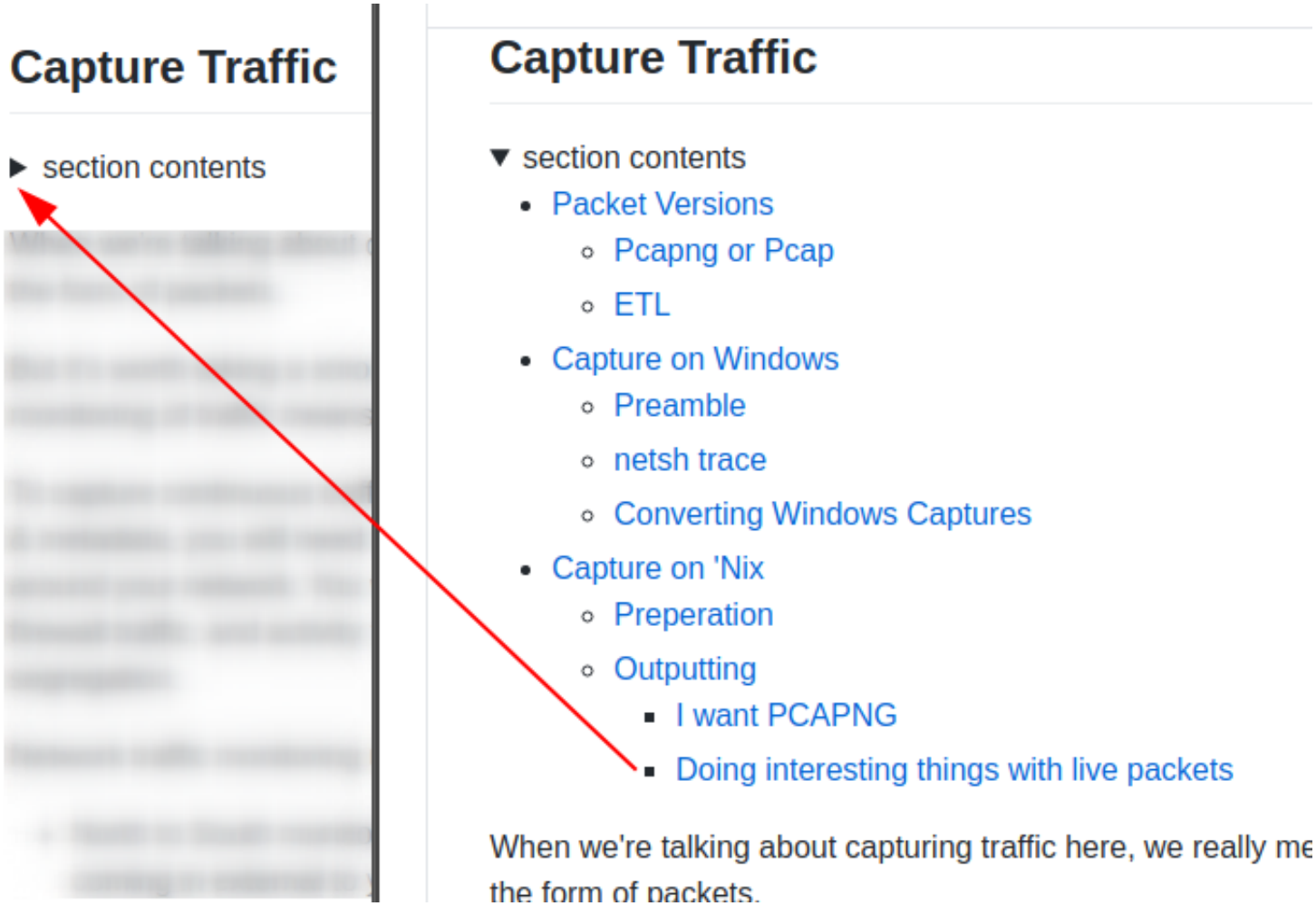
Donate as much or little money as you like, of course. I have some UK charities you could donate to: Great Ormond Street - Children's hospital, Cancer Research, and Feeding Britain - food charity

Table of Contents

As you scroll along, it's easy to lose orientation. Wherever you are in the Blue Team Notes, if you look to the top-left of the readme you'll see a little icon. This is a small table of contents, and it will help you figure out where you are, where you've been, and where you're going



As you go through sections, you may notice the arrowhead that says 'section contents'. I have nestled the sub-headings in these, to make life a bit easier.



Shell Style

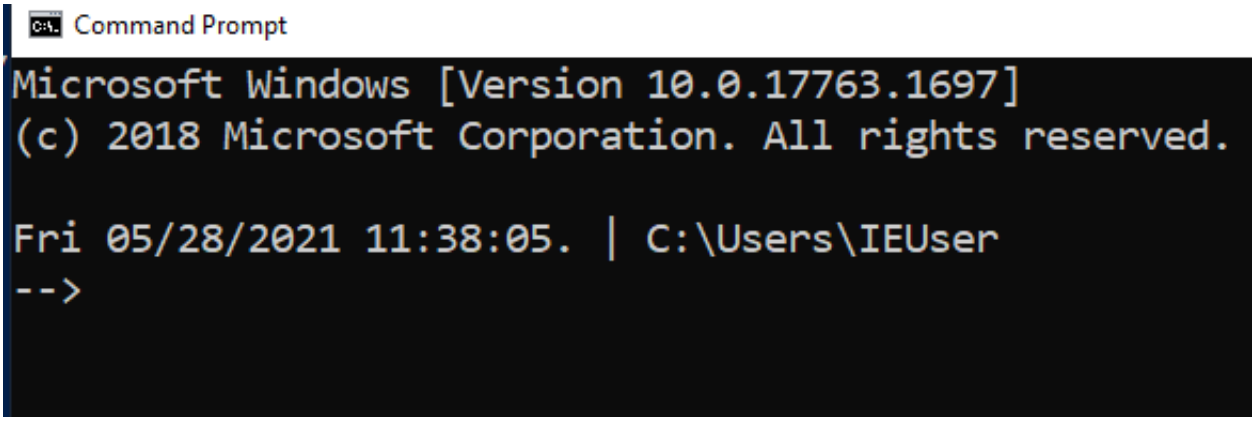
▶ section contents

Give shell timestamp

For screenshots during IR, I like to have the date, time, and sometimes the timezone in my shell

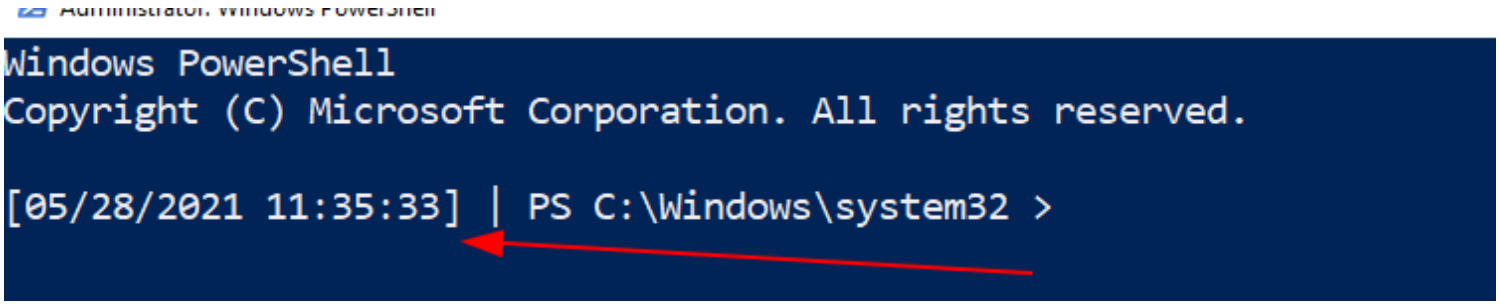
CMD

```
setx prompt $D$S$T$H$H$H$S$B$S$P$_--$g
:: all the H's are to backspace the stupid microsecond timestamp
:: $_ and --$g seperate the date/time and path from the actual shell
:: We make the use of the prompt command: https://docs.microsoft.com/en-us/window
:: setx is in fact the command line command to write variables to the registery
:: We are writing the prompt's new timestamp value in the cmd line into the reg s
```



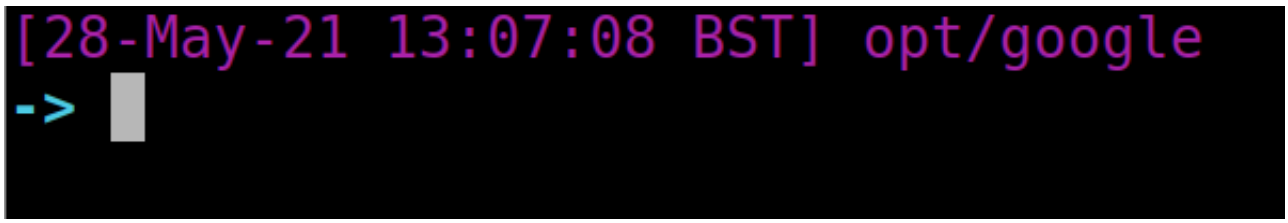
Pwsh

```
###create a powershell profile, if it doesnt exist already
New-Item $Profile -ItemType file -Force
##open it in notepad to edit
function prompt{ "[$(Get-Date)]" +" | PS "+ "$(Get-Location) > "}
##risky move, need to tighten this up. Change your execution policy or it won't
#run the profile ps1
#run as powershell admin
Set-ExecutionPolicy RemoteSigned
```



Bash

```
##open .bashrc
sudo nano .bashrc
#https://www.howtogeek.com/307701/how-to-customize-and-colorize-your-bash-prompt/
##date, time, colour, and parent+child directory only, and -> promptt
PS1='\[\033[00;35m\][`date  +"%d-%b-%y %T %Z"]` ${PWD#"${PWD%/*/*}/"}\n\[\033[01;
      ##begin purple  #year,month,day,time,timezone #show last 2 dir #next line,
#restart the bash source
source ~/.bashrc
```



Windows

▶ section contents

I've generally used these Powershell queries with Velociraptor, which can query thousands of endpoints at once.

OS Queries

▶ section contents

Get Fully Qualified Domain Name

```
([System.Net.Dns]::GetHostByName(($env:computerName))).Hostname

# Get just domain name
```

```
(Get-WmiObject -Class win32_computersystem).domain
```

```
[06/27/2021 10:16:53] PS >([System.Net.Dns]::GetHostByName(($env:computerName))).Hostname
McCerty.JUMPSEC.GB
[06/27/2021 10:16:58] PS >(Get-WmiObject -Class win32_computersystem).domain
JUMPSEC.GB
[06/27/2021 10:16:59] PS >_
```

Get OS and Pwsh info

This will print out the hostname, the OS build info, and the powershell version

```
$Bit = (get-wmiobject Win32_OperatingSystem).OSArchitecture ;
$V = $host | select-object -property "Version" ;
$Build = (Get-WmiObject -class Win32_OperatingSystem).Caption ;
write-host "$env:computername is a $Bit $Build with Pwsh $V
```

```
SP-VM03 is a 64-bit Microsoft Windows 7 Professional with Pwsh @{Version=2.0}
60 is a 64-bit Microsoft Windows 10 Pro with Pwsh @{Version=5.1.18362.752}
983 is a 64-bit Microsoft Windows 10 Enterprise LTSC with Pwsh @{Version=5.1.17763.1007}
005 is a 64-bit Microsoft Windows 10 Pro with Pwsh @{Version=5.1.16299.1004}
16 is a 64-bit Microsoft Windows 7 Professional with Pwsh @{Version=2.0}
25 is a 64-bit Microsoft Windows 7 Professional with Pwsh @{Version=2.0}
49 is a 64-bit Microsoft Windows 7 Professional with Pwsh @{Version=2.0}
975 is a 64-bit Microsoft Windows 10 Enterprise LTSC with Pwsh @{Version=5.1.17763.1007}
is a 32-bit Microsoft Windows 7 Professional with Pwsh @{Version=2.0}
01 is a 64-bit Microsoft Windows 7 Professional with Pwsh @{Version=2.0}
036 is a 64-bit Microsoft Windows 10 Pro with Pwsh @{Version=5.0.10586.1176}
```

Hardware Info

If you want, you can get Hardware, BIOS, and Disk Space info of a machine

```
#Get BIOS Info
gcim -ClassName Win32_BIOS | fl Manufacturer, Name, SerialNumber, Version;
#Get processor info
gcim -ClassName Win32_Processor | fl caption, Name, SocketDesignation;
#Computer Model
```

```
gcim -ClassName Win32_ComputerSystem | fl Manufacturer, Systemfamily, Model, Syst
#Disk space in Gigs, as who wants bytes?
gcim  -ClassName Win32_LogicalDisk |
Select -Property DeviceID, DriveType, @{L='FreeSpaceGB';E={"{0:N2}" -f ($_.FreeSp

## Let's calculate an individual directory, C:\Sysmon, and compare with disk memo
$size = (gci c:\sysmon | measure Length -s).sum / 1Gb;
write-host " Sysmon Directory in Gigs: $size";
$free = gcim  -ClassName Win32_LogicalDisk | select @{L='FreeSpaceGB';E={"{0:N2}"
echo "$free";
$cap = gcim  -ClassName Win32_LogicalDisk | select @{L="Capacity";E={"{0:N2}" -f
echo "$cap"
```

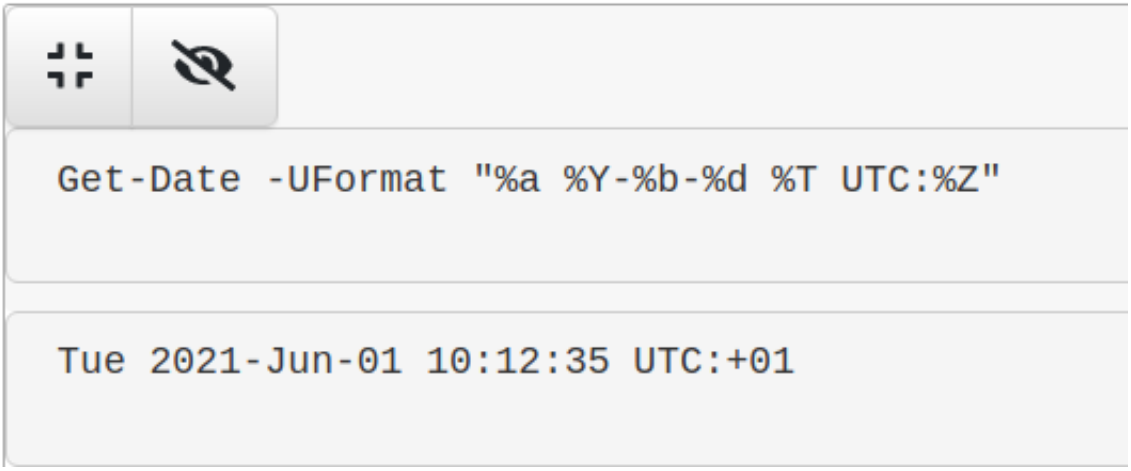
```
Name         : PhoenixBIOS 4.0 Release 6.0
SerialNumber : VMware-42 1d d8 45 49 7b 92 91-ee 1f 91 b4 6
Version      : INTEL  - 6040000




caption           : Intel64 Family 6 Model 85 Stepping 4
Name              : Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GH
SocketDesignation : CPU #000

caption           : Intel64 Family 6 Model 85 Stepping 4
Name              : Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GH
SocketDesignation : CPU #001




Manufacturer : VMware, Inc.
Model        : VMware Virtual Platform
SystemType   : x64-based PC




DeviceID    : C:
DriveType   : 3
FreeSpaceGB : 57.99
Capacity    : 79.66
```

Time info

Human Readable

Get a time that's human readable

```
Get-Date -UFormat "%a %Y-%b-%d %T UTC:%Z"
```



Machine comparable

This one is great for doing comparisons between two strings of time

```
[Xml.XmlConvert]::ToString((Get-Date).ToUniversalTime(), [System.Xml.XmlDateTimeS
```

Compare UTC time from Local time

```
$Local = get-date;$UTC = (get-date).ToUniversalTime();
write-host "LocalTime is: $Local";write-host "UTC is: $UTC"
```

```
 $Local = get-date;$UTC = (get-date).ToUniversalTime();
write-host "LocalTime is: $Local";write-host "UTC is: $UTC"
```

```
 LocalTime is: 06/01/2021 10:34:36
UTC is: 06/01/2021 09:34:36
```

Update Info

Get Patches

Will show all patch IDs and their installation date

```
get-hotfix|
select-object HotFixID,InstalledOn|
Sort-Object  -Descending -property InstalledOn|
format-table -autosize
```

```
HotFixID  InstalledOn
--------  -----------
KB5001078 15/03/2021 00:00:00
KB4598243 15/03/2021 00:00:00
KB4535680 27/01/2021 00:00:00
KB4054590 27/01/2021 00:00:00
KB4132216 04/12/2020 00:00:00
KB4576750 25/11/2020 00:00:00
KB4049065 02/02/2018 00:00:00
```

Find why an update failed

```
$Failures = gwmi -Class Win32_ReliabilityRecords;
$Failures | ? message -match 'failure'  | Select -ExpandProperty message
```

Manually check if patch has taken

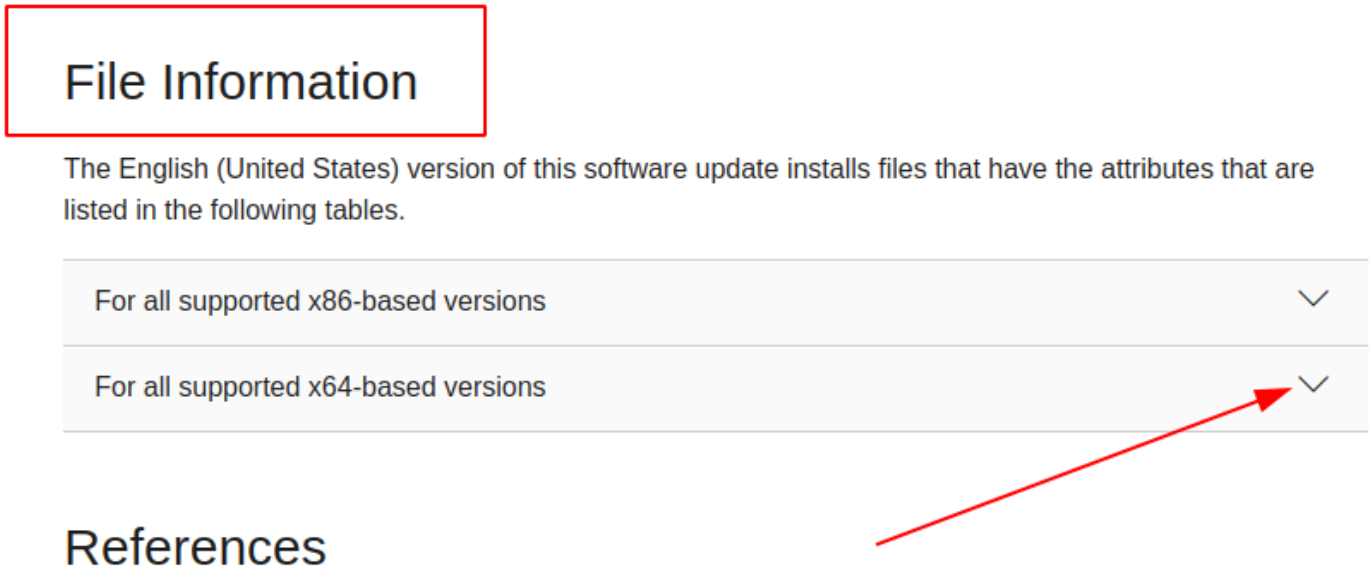
This happened to me during the March 2021 situation with Microsoft Exchange's ProxyLogon. The sysadmin swore blind they had patched the server, but neither `systeminfo` of `get-hotfix` was returning with the correct KB patch.

The manual workaround isn't too much ballache

Microsoft Support Page

First identify the ID number of the patch you want. And then find the dedicated Microsoft support page for it.

For demonstration purposes, let's take `KB5001078` and it's corresponding support page. You'll be fine just googling the patch ID number.



Then click into the dropdown relevant to your machine.

File Information

The English (United States) version of this software update installs files that have the attributes that are listed in the following tables.

For all supported x86-based versions

For all supported x64-based versions

File name	File version	Date	Time	File size
luainstall.dll	10.0.14393.4222	13-Jan-2021	21:11	60,176
appxreg.dll	10.0.14393.4222	13-Jan-2021	21:10	42,776
appxprovisionpackage.dll	10.0.14393.4222	13-Jan-2021	21:20	86,800
EventsInstaller.dll	10.0.14393.4222	13-Jan-2021	21:20	222,488

Here you can see the files that are included in a particular update. The task now is to pick a handful of the patch-files and compare your host machine. See if these files exist too, and if they do do they have similar / same dates on the host as they do in the Microsoft patch list?

On Host

Let us now assume you don't know the path to this file on your host machine. You will have to recursively search for the file location. It's a fair bet that the file will be in `C:\Windows\` (but not always), so lets' recursively look for `EventsInstaller.dll`

```
$file = 'EventsInstaller.dll'; $directory = 'C:\windows' ;
gci -Path $directory -Filter $file -Recurse -force|
sort-object  -descending -property LastWriteTimeUtc | fl *
```

We'll get a lot of information here, but we're really concerned with is the section around the various *times*. As we sort by the `LastWriteTimeUtc`, the top result should in theory be the latest file of that name...but this is not always true.

```
Extension          : .dll
CreationTime       : 02/02/2018 18:14:02
CreationTimeUtc    : 02/02/2018 18:14:02
LastAccessTime     : 17/05/2021 17:34:30
LastAccessTimeUtc  : 17/05/2021 16:34:30
LastWriteTime      : 17/05/2021 17:34:30
LastWriteTimeUtc   : 17/05/2021 16:34:30
Attributes         : Archive
```

Discrepencies

I've noticed that sometimes there is a couple days discrepency between dates.

For example in our screenshot, on the left Microsoft's support page supposes the `EventsInstaller.dll` was written on the 13th January 2021. And yet our host on the right side of the screenshot comes up as the 14th January 2021. This is fine though, you've got that file don't sweat it.

Account Queries

▶ section contents

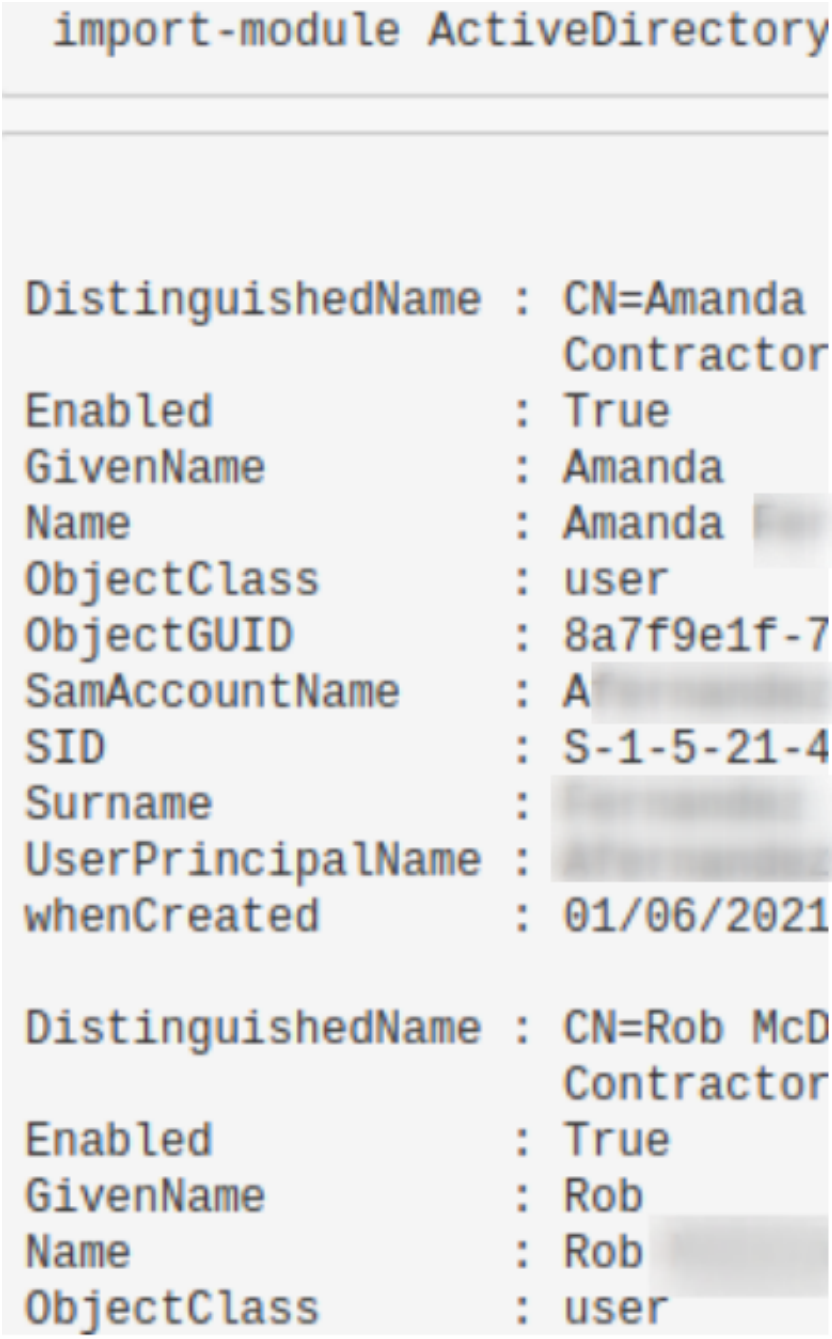
Users recently created in Active Directory

Run on a Domain Controller.

Change the AddDays field to more or less days if you want. Right now set to seven days.

The 'when Created' field is great for noticing some inconsistencies. For example, how often are users created at 2am?

```
import-module ActiveDirectory;
$When = ((Get-Date).AddDays(-7)).Date;
Get-ADUser -Filter {whenCreated -ge $When} -Properties whenCreated |
sort whenCreated -descending
```



Hone in on suspicious user

You can use the `SamAccountName` above to filter

```
import-module ActiveDirectory;
```

```
Get-ADUser -Identity HamBurglar -Properties *
```

```
AccountExpirationDate                : 28
accountExpires                       : 13
AccountLockoutTime                   :
AccountNotDelegated                  : Fa
AllowReversiblePasswordEncryption    : Fa
AuthenticationPolicy                 : {}
AuthenticationPolicySilo             : {}
BadLogonCount                        :
CannotChangePassword                 : Fa
CanonicalName                        : CP
                                       Co
Certificates                         : {}
City                                 :
CN                                   : Am
codePage                             : 0
Company                              : Ex
CompoundIdentitySupported            : {}
Country                              :
countryCode                          : 0
Created                              : 01
createTimeStamp                      : 01
Deleted                              :
Department                           :
Description                          : EL
DisplayName                          : Am
DistinguishedName                    : CN
                                       Fe
                                       Co
Division                             :
DoesNotRequirePreAuth                : Fa
dSCorePropagationData                : {0
EmailAddress                         :
```

Retrieve local user accounts that are enabled

```
Get-LocalUser | ? Enabled -eq "True"
```

```
[06/02/2021 22:48:03] | PS C:\Win

Name   Enabled Description
----   ------- -----------
IEUser True    IEUser
sshd   True
```

Find all users currently logged in

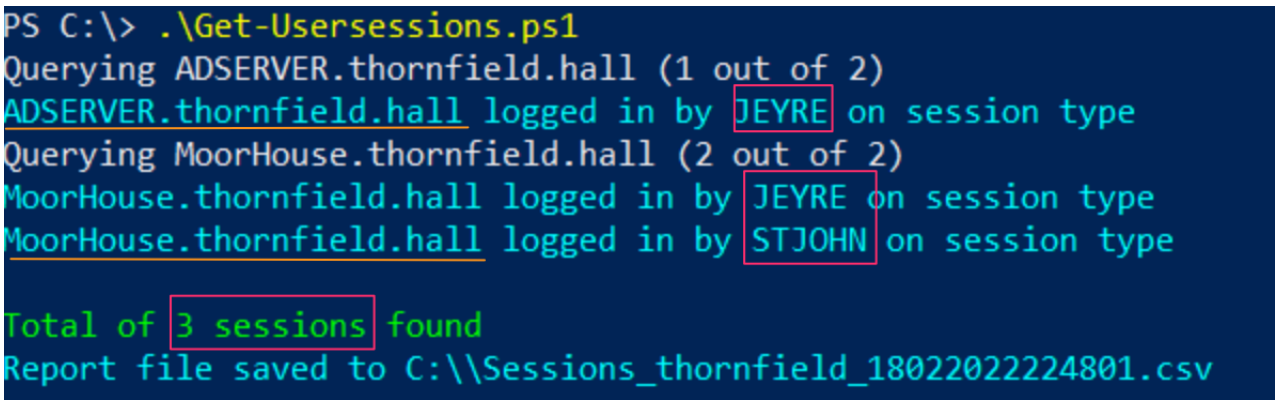
```
qwinsta
#or
quser
```

Find all users logged in across entire AD

If you want to find every single user logged in on your Active Directory, with the machine they are also signed in to.

I can reccomend YossiSassi's Get-UserSession.ps1 and Get-RemotePSSession.ps1.

This will generate a LOT of data in a real-world AD though.

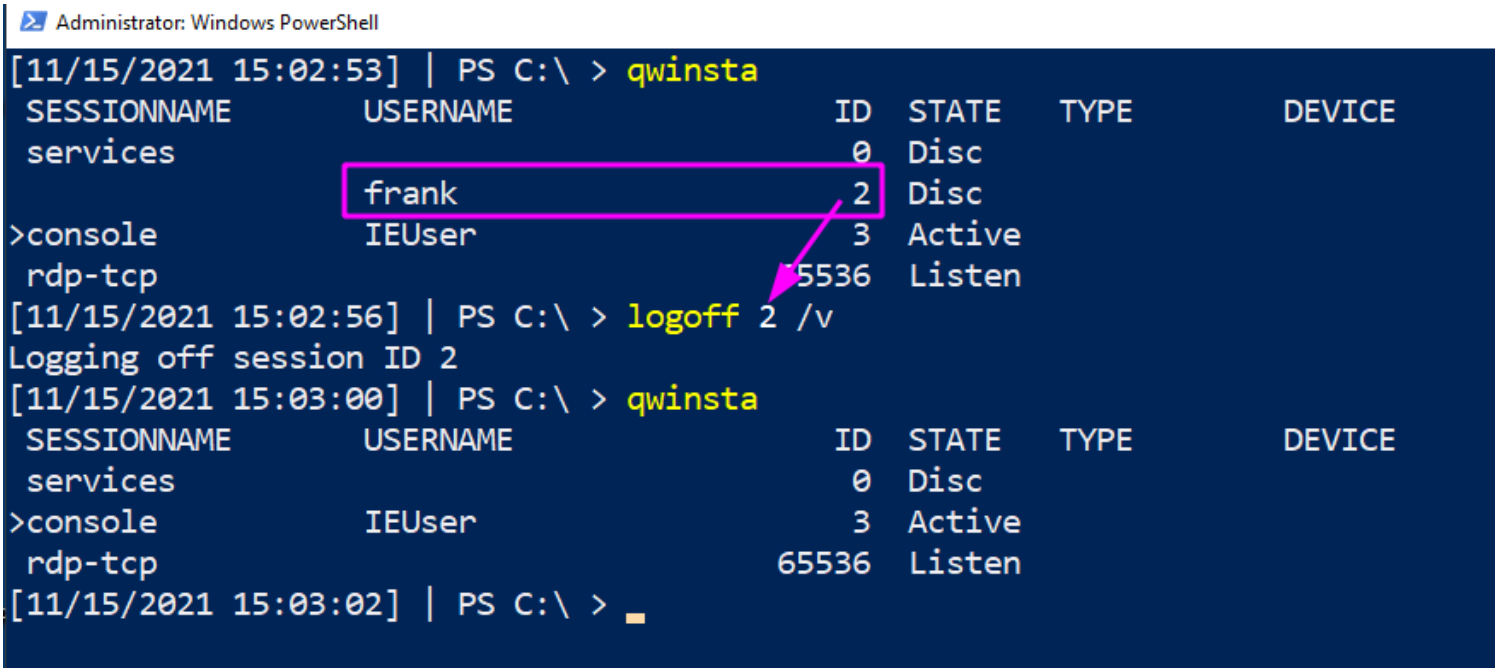


Evict User

Force user logout

You may need to evict a user from a session - perhaps you can see an adversary has been able to steal a user's creds and is leveraging their account to traverse your environment

```
#show the users' session
qwinsta

#target their session id
logoff 2 /v
```

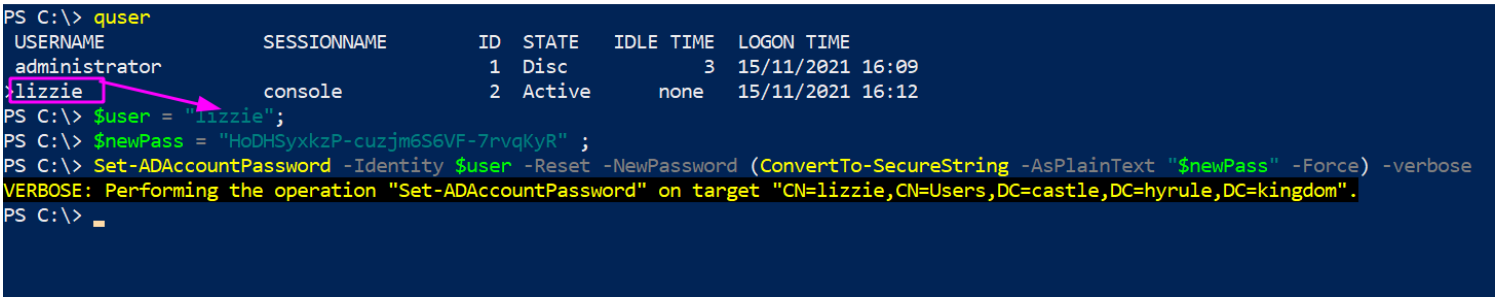


Force user new password

From the above instance, we may want to force a user to have a new password - one the adversary does not have

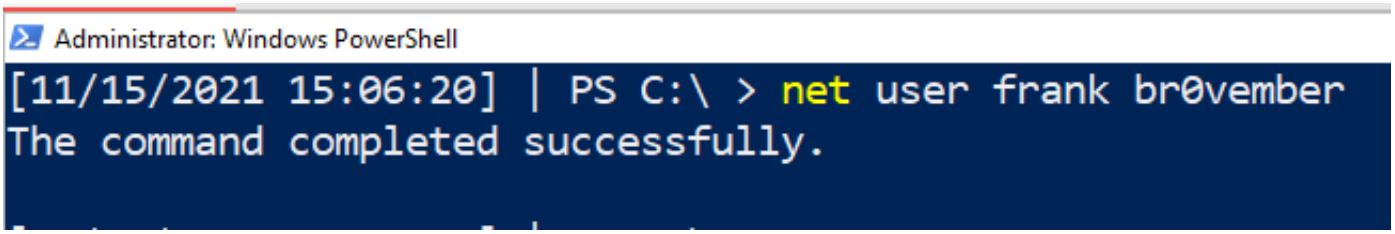
for Active Directory

```
$user = "lizzie" ; $newPass = "HoDHSyxkzP-cuzjm6S6VF-7rvqKyR";

#Change password twice.
#First can be junk password, second time can be real new password
Set-ADAccountPassword -Identity $user -Reset -NewPassword (ConvertTo-SecureString
Set-ADAccountPassword -Identity $user -Reset -NewPassword (ConvertTo-SecureString
```



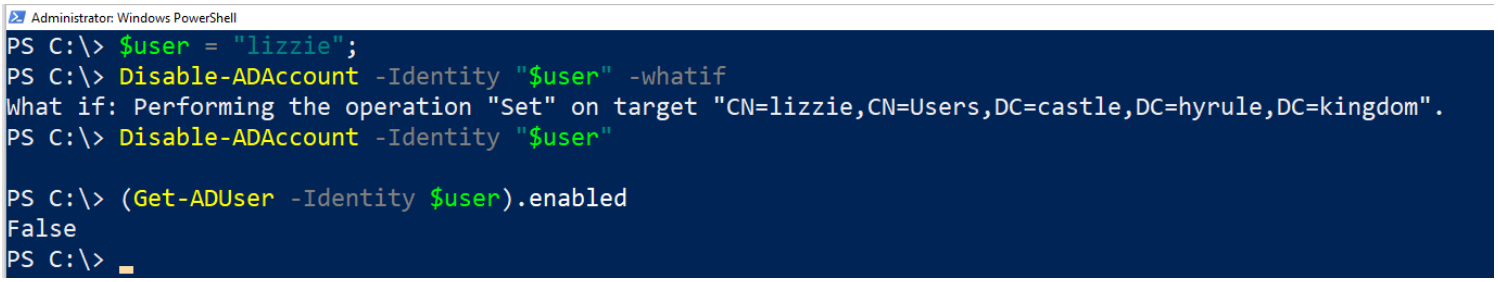
For local non-domain joined machines

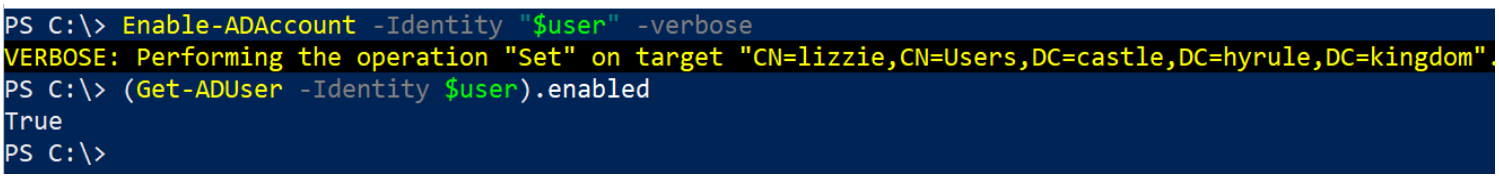
```
#for local users
net user #username #newpass
net user frank "lFjcVR7fW2-HoDHSyxkzP"
```



Disable AD Account

```
#needs the SAMAccountName
$user = "lizzie";
Disable-ADAccount -Identity "$user" #-whatif can be appended

#check its disabled
(Get-ADUser -Identity $user).enabled

#renable when you're ready
Enable-ADAccount -Identity "$user" -verbose
```





Disable local Account

```
# list accounts with  Get-LocalUser
Disable-LocalUser -name "bad_account$"
```

```
PS C:\> Get-LocalUser | ? Name -match bad | fl Name,enabled


Name    : bad_account$
Enabled : True



PS C:\> Disable-LocalUser -name "bad_account$" -whatif
What if: Performing the operation "Disable local user" on target "bad_account$".
PS C:\> Disable-LocalUser -name "bad_account$"
PS C:\> Get-LocalUser | ? Name -match bad | fl Name,enabled


Name    : bad_account$
Enabled : False
```

Evict from Group

Good if you need to quickly eject an account from a specific group, like administrators or remote management.

```
$user = "erochester"
remove-adgroupmember -identity Administrators -members $User -verbose -confirm:$f
```

```
[01/23/2022 22:05:08] | PS C:\ > Remove-ADGroupmember -identity Administrators -members
"erochester" -verbose -confirm:$false
VERBOSE: Performing the operation "Set" on target
"CN=Administrators,CN=Builtin,DC=thornfield,DC=hall".
[01/23/2022 22:05:18] | PS C:\ >
```

Computer / Machine Accounts

Adversaries like to use Machine accounts (accounts that have a $) as these often are overpowered AND fly under the defenders' radar

Show machine accounts that are apart of interesting groups.

There may be misconfigurations that an adversary could take advantadge.

```
Get-ADComputer -Filter * -Properties MemberOf | ? {$_.MemberOf}
```

```
DNSHostName       : [redacted]ocal
Enabled           : False
MemberOf          : {CN=DL_ADAudit_Plus_Permission,OU=Restricted,OU=Groups,DC=C
                    PUK,DC=local}
Name              : [redacted]
ObjectClass       : computer
ObjectGUID        : 126369c2-02ee-4472-98d7-d4663cb2146b
SamAccountName    : [redacted]4$
SID               : S-1-5-21-[redacted]
UserPrincipalName :
```

Reset password for a machine account.

Good for depriving adversary of pass they may have got. Also good for re-establishing trust if machine is kicked out of domain trust for reasons(?)

```
Reset-ComputerMachinePassword
```

All Users PowerShell History

During an IR, you will want to access other users PowerShell history. However, the get-history command only will retrieve the current shell's history, which isn't very useful.

Instead, PowerShell in Windows 10 saves the last 4096 commands in a particular file. On an endpoint, we can run a quick loop that will print the full path of the history file - showing which users history it is showing - and then show the contents of that users' PwSh commands

```
$Users = (Gci C:\Users\*\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\
$Pasts = @($Users);

foreach ($Past in $Pasts) {
  write-host "`n----User Pwsh History Path $Past---`n" -ForegroundColor Magenta;
  get-content $Past
}
```

```
PS C:\Users\Administrator> $Users = (Gci C:\Users\*\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt).FullName
>> $Pasts = @($Users);
>>
>> foreach ($Past in $Pasts) {
>>     write-host "`n----User Pwsh History Path $Past---`n" -ForegroundColor Magenta;
>> get-content $Past -first 3
>> }

----User Pwsh History Path C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt---

touch evil.exe
echo > evil.exe
echo > evil.ps1

----User Pwsh History Path C:\Users\jimmy\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt---

ls C:\Users
whoami
net users
PS C:\Users\Administrator>
```

And check this one too

```
c:\windows\system32\config\systemprofile\appdata\roaming\microsoft\windows\powers
```

Service Queries

▶ section contents

Show Services

Let's get all the services and sort by what's running

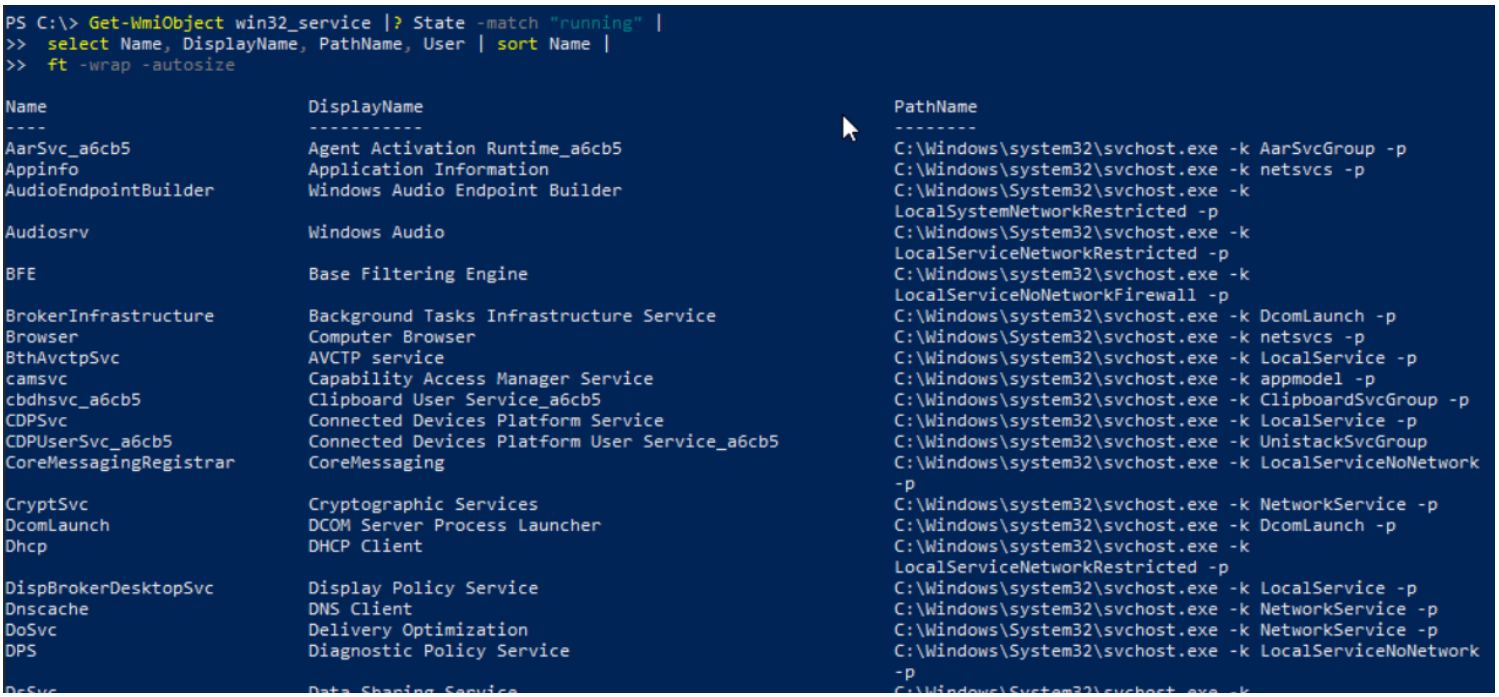
```
get-service|Select Name,DisplayName,Status|
sort status -descending | ft -Property * -AutoSize|
Out-String -Width 4096
```

```
Name                      DisplayName                                   Status
----                      -----------                                   ------
EventLog                  Windows Event Log                             Running
EventSystem               COM+ Event System                             Running
StorSvc                   Storage Service                               Running
Power                     Power                                         Running
WaaSMedicSvc              Windows Update Medic Service                  Running
W32Time                   Windows Time                                  Running
SysMain                   SysMain                                       Running
SystemEventsBroker        System Events Broker                          Running
TabletInputService        Touch Keyboard and Handwriting Panel Service  Running
VMTools                   VMware Tools                                  Running
PlugPlay                  Plug and Play                                 Running
FontCache                 Windows Font Cache Service                    Running
StateRepository           State Repository Service                      Running
netprofm                  Network List Service                          Running
Dnscache                  DNS Client                                    Running
```

Now show the underlying executable supporting that service

```
Get-WmiObject win32_service |? State -match "running" |
```

```
select Name, DisplayName, PathName, User | sort Name |
ft -wrap -autosize
```



```
PS C:\> Get-WmiObject win32_service |? State -match "running" |
>>  select Name, DisplayName, PathName, User | sort Name |
>>  ft -wrap -autosize

Name                      DisplayName                                  PathName
----                      -----------                                  --------
AarSvc_a6cb5              Agent Activation Runtime_a6cb5               C:\Windows\system32\svchost.exe -k AarSvcGroup -p
Appinfo                   Application Information                      C:\Windows\system32\svchost.exe -k netsvcs -p
AudioEndpointBuilder      Windows Audio Endpoint Builder               C:\Windows\System32\svchost.exe -k
                                                                       LocalSystemNetworkRestricted -p
Audiosrv                  Windows Audio                                C:\Windows\System32\svchost.exe -k
                                                                       LocalServiceNetworkRestricted -p
BFE                       Base Filtering Engine                        C:\Windows\system32\svchost.exe -k
                                                                       LocalServiceNoNetworkFirewall -p
BrokerInfrastructure      Background Tasks Infrastructure Service      C:\Windows\system32\svchost.exe -k DcomLaunch -p
Browser                   Computer Browser                             C:\Windows\System32\svchost.exe -k netsvcs -p
BthAvctpSvc               AVCTP service                                C:\Windows\system32\svchost.exe -k LocalService -p
camsvc                    Capability Access Manager Service            C:\Windows\system32\svchost.exe -k appmodel -p
cbdhsvc_a6cb5             Clipboard User Service_a6cb5                 C:\Windows\system32\svchost.exe -k ClipboardSvcGroup -p
CDPSvc                    Connected Devices Platform Service           C:\Windows\system32\svchost.exe -k LocalService -p
CDPUserSvc_a6cb5          Connected Devices Platform User Service_a6cb5 C:\Windows\system32\svchost.exe -k UnistackSvcGroup
CoreMessagingRegistrar    CoreMessaging                                C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork
                                                                       -p
CryptSvc                  Cryptographic Services                       C:\Windows\system32\svchost.exe -k NetworkService -p
DcomLaunch                DCOM Server Process Launcher                 C:\Windows\system32\svchost.exe -k DcomLaunch -p
Dhcp                      DHCP Client                                  C:\Windows\system32\svchost.exe -k
                                                                       LocalServiceNetworkRestricted -p
DispBrokerDesktopSvc      Display Policy Service                       C:\Windows\system32\svchost.exe -k LocalService -p
Dnscache                  DNS Client                                   C:\Windows\system32\svchost.exe -k NetworkService -p
DoSvc                     Delivery Optimization                        C:\Windows\System32\svchost.exe -k NetworkService -p
DPS                       Diagnostic Policy Service                    C:\Windows\System32\svchost.exe -k LocalServiceNoNetwork
                                                                       -p
DsSvc                     Data Sharing Service                         C:\Windows\System32\svchost.exe -k
```

Hone in on specific Service

If a specific service catches your eye, you can get all the info for it. Because the single and double qoutes are important to getting this right, I find it easier to just put the DisplayName of the service I want as a variable, as I tend to fuck up the displayname filter bit

```
$Name = "eventlog";
gwmi -Class Win32_Service -Filter "Name = '$Name' " | fl *

#or this, but you get less information compared to the one about tbh
get-service -name "eventlog" | fl *
```

```
AcceptPause               : False
AcceptStop                : True
Caption                   : Active Directory Web Services
CheckPoint                : 0
CreationClassName         : Win32_Service
DelayedAutoStart          : False
Description               : This service provides a Web Service interface to
                            instances of the directory service (AD DS and AD
                            LDS) that are running locally on this server. If
                            this service is stopped or disabled, client
                            applications, such as Active Directory PowerShell,
                            will not be able to access or manage any directory
                            service instances that are running locally on this
                            server.
DisplayName               : Active Directory Web Services
InstallDate               :
ProcessId                 : 1916
```

Kill a service

```
Get-Service -DisplayName "meme_service" | Stop-Service -Force -Confirm:$false -ve
```

Hunting potential sneaky services

I saw a red team tweet regarding sneaky service install. To identify this, you can deploy the following:

```
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\services\*" |
ft PSChildName, ImagePath -autosize | out-string -width 800

# Grep out results from System32 to reduce noise, though keep in mind adversaries
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\services\*" |
where ImagePath -notlike "*System32*" |
ft PSChildName, ImagePath -autosize | out-string -width 800
```

```
puppet                          "C:\Program Files\Puppet Labs\Puppet\sys\ruby\bin\ruby.exe" -rubygems "C:\Program Files\Puppet
abs\Puppet\service\daemon.rb"
pvscsi

RDMANDK

RDPNP

ReFS

ReFSv1

Sense                           "C:\Program Files\Windows Defender Advanced Threat Protection\MsSense.exe"

sneaky                          C:\sneaky.exe

Sysmon                          C:\Windows\Sysmon.exe

SysmonDrv                       SysmonDrv.sys

TCPIP6TUNNEL

TCPIPTUNNEL

TrustedInstaller                C:\Windows\servicing\TrustedInstaller.exe
```

Network Queries

▶ section contents

Show TCP connections and underlying process

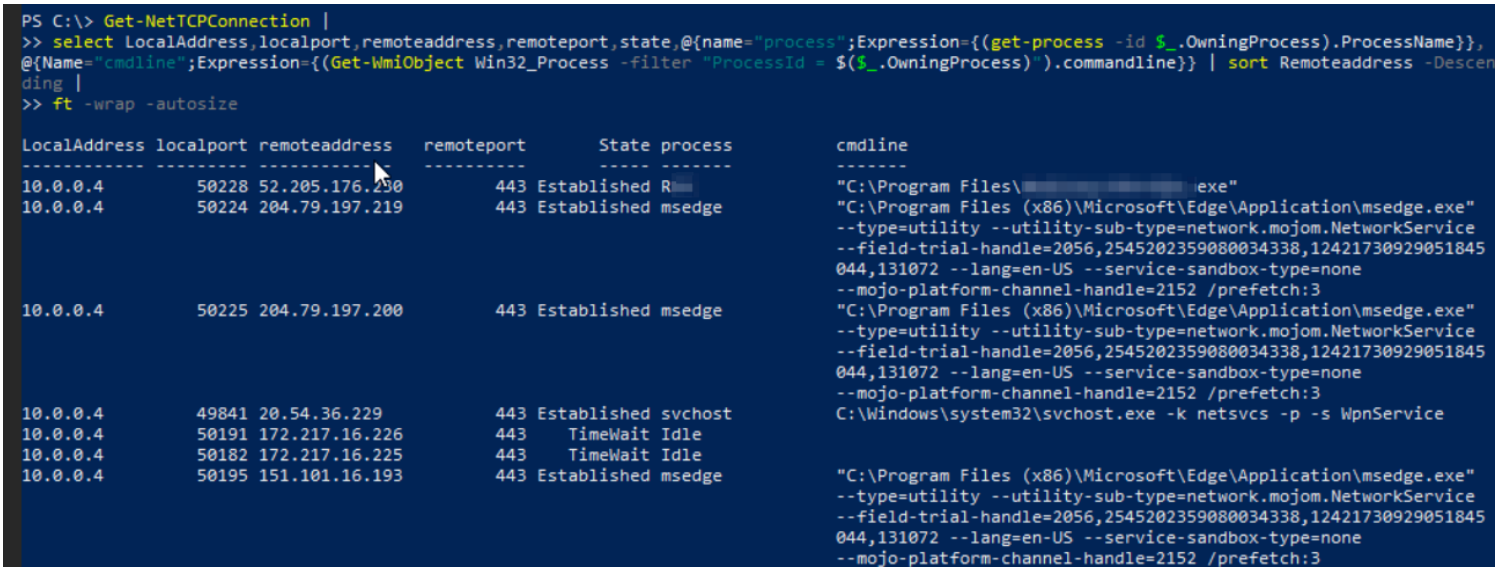
This one is so important, I have it listed twice in the blue team notes

I have a neat one-liner for you. This will show you the local IP and port, the remote IP andport, the process name, and the underlying executable of the process!

You could just use `netstat -b`, which gives you SOME of this data

But instead, try this bad boy on for size:

```
Get-NetTCPConnection |
select LocalAddress,localport,remoteaddress,remoteport,state,@{name="process";Exp
sort Remoteaddress -Descending | ft -wrap -autosize

#### you can search/filter by the commandline process, but it will come out janky
##### in the final field we're searching by `anydesk`
Get-NetTCPConnection |
select LocalAddress,localport,remoteaddress,remoteport,state,@{name="process";Exp
|  Select-String -Pattern 'anydesk'
```



```
PS C:\> Get-NetTCPConnection |
>> select LocalAddress,localport,remoteaddress,remoteport,state,@{name="process";Expression={(get-process -id $_.OwningProcess).ProcessName}},
@{Name="cmdline";Expression={(Get-WmiObject Win32_Process -filter "ProcessId = $($_.OwningProcess)").commandline}} | sort Remoteaddress -Descen
ding |
>> ft -wrap -autosize

LocalAddress localport remoteaddress   remoteport       State process          cmdline
------------ --------- -------------   ----------       ----- -------          -------
10.0.0.4         50228 52.205.176.230         443 Established R[redacted]      "C:\Program Files\[redacted]exe"
10.0.0.4         50224 204.79.197.219         443 Established msedge           "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
                                                                                --type=utility --utility-sub-type=network.mojom.NetworkService
                                                                                --field-trial-handle=2056,2545202359080034338,12421730929051845
                                                                                044,131072 --lang=en-US --service-sandbox-type=none
                                                                                --mojo-platform-channel-handle=2152 /prefetch:3
10.0.0.4         50225 204.79.197.200         443 Established msedge           "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
                                                                                --type=utility --utility-sub-type=network.mojom.NetworkService
                                                                                --field-trial-handle=2056,2545202359080034338,12421730929051845
                                                                                044,131072 --lang=en-US --service-sandbox-type=none
                                                                                --mojo-platform-channel-handle=2152 /prefetch:3
10.0.0.4         49841 20.54.36.229           443 Established svchost          C:\Windows\system32\svchost.exe -k netsvcs -p -s WpnService
10.0.0.4         50191 172.217.16.226         443    TimeWait Idle
10.0.0.4         50182 172.217.16.225         443    TimeWait Idle
10.0.0.4         50195 151.101.16.193         443 Established msedge           "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
                                                                                --type=utility --utility-sub-type=network.mojom.NetworkService
                                                                                --field-trial-handle=2056,2545202359080034338,12421730929051845
                                                                                044,131072 --lang=en-US --service-sandbox-type=none
                                                                                --mojo-platform-channel-handle=2152 /prefetch:3
```

######### **Bound to catch bad guys or your moneyback guaranteed!!!!**

Find internet established connections, and sort by time established

You can always sort by whatever value you want really. CreationTime is just an example

```
Get-NetTCPConnection -AppliedSetting Internet |
select-object -property remoteaddress, remoteport, creationtime |
Sort-Object -Property creationtime |
format-table -autosize
```

```
remoteaddress  remoteport creationtime
-------------  ---------- ------------
10.200.151.66        5079 18/04/2021 16:44:09
10.200.151.66        5079 18/04/2021 16:44:09
                      445 30/04/2021 05:44:58
10.200.154.147        445 09/05/2021 08:22:07
10.200.154.136      49154 15/05/2021 10:32:52
10.200.154.130        445 18/05/2021 15:10:24
10.200.154.144        445 19/05/2021 18:35:18
10.200.150.10         445 25/05/2021 06:34:41
10.200.154.166        445 26/05/2021 08:02:43
10.200.154.119        445 26/05/2021 14:13:06
10.200.154.113        445 27/05/2021 15:03:54
10.200.154.200      49154 28/05/2021 15:50:41
10.200.155.49       49739 28/05/2021 15:50:43
10.200.154.166      49154 28/05/2021 15:50:45
10.200.155.20       50194 28/05/2021 15:50:46
10.200.160.196      49154 28/05/2021 15:50:46
10.200.155.21       49864 28/05/2021 15:50:46
```

Sort remote IP connections, and then unique them

This really makes strange IPs stand out

```
(Get-NetTCPConnection).remoteaddress | Sort-Object -Unique
```

```
remoteaddress
-------------
::
0.0.0.0
1.2.3.4
10.200.150.10
10.200.150.129
10.200.150.130
10.200.150.84
10.200.151.22
10.200.151.40
10.200.151.41
10.200.151.45
10.200.151.66
10.200.154.102
10.200.154.108
10.200.154.109
```

Hone in on a suspicious IP

If you see suspicious IP address in any of the above, then I would hone in on it

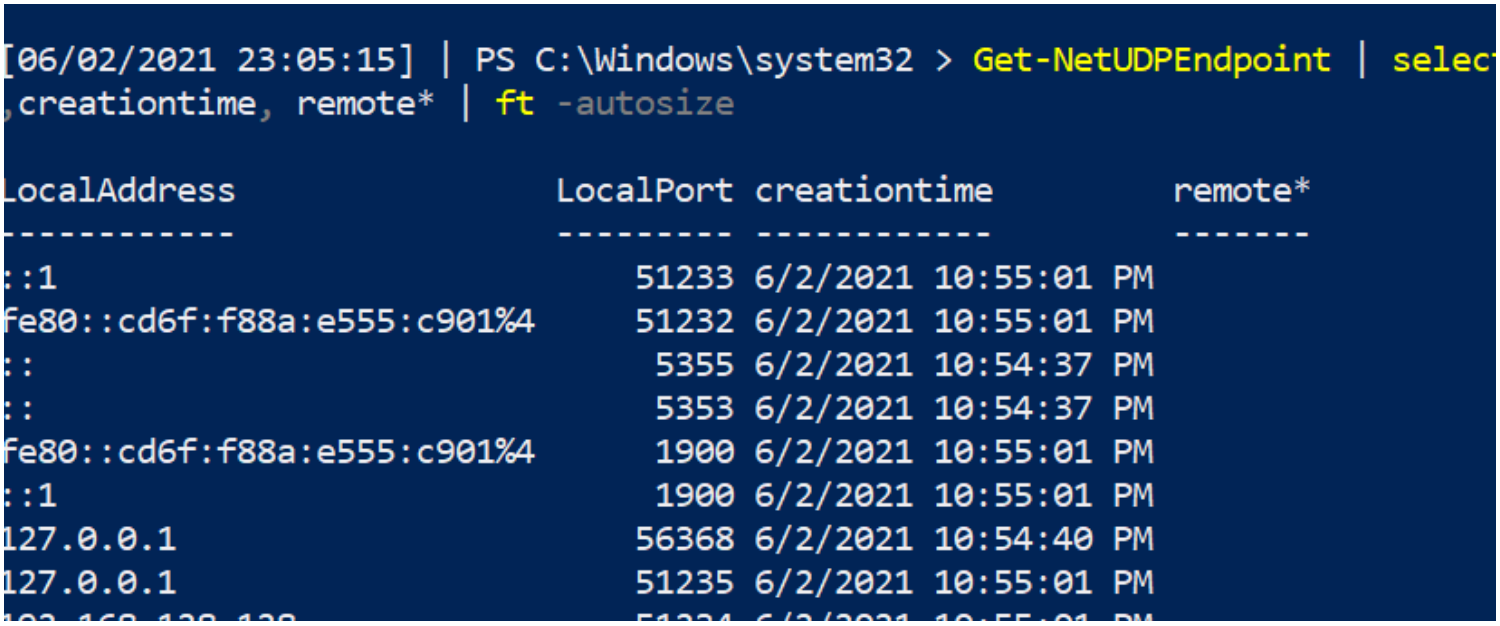
```
Get-NetTCPConnection |
? {($_.RemoteAddress -eq "1.2.3.4")} |
select-object -property state, creationtime, localport,remoteport | ft -autosize

## can do this as well
 Get-NetTCPConnection -remoteaddress 0.0.0.0 |
 select state, creationtime, localport,remoteport | ft -autosize
```

```
      State creationtime        localport remoteport
      ----- ------------        --------- ----------
Established 30/04/2021 05:44:58     61700        445
Established 28/05/2021 16:10:24     61578      49154
   TimeWait 01/01/1601 00:00:00     61500        135
```

Show UDP connections

You can generally filter pwsh UDP the way we did the above TCP

```
Get-NetUDPEndpoint | select local*,creationtime, remote* | ft -autosize
```



Kill a connection

There's probably a better way to do this. But essentially, get the tcp connection that has the specific remote IPv4/6 you want to kill. It will collect the OwningProcess. From here, get-process then filters for those owningprocess ID numbers. And then it will stop said process. Bit clunky

```
stop-process -verbose -force -Confirm:$false (Get-Process -Id (Get-NetTCPConnecti
```

Check Hosts file

Some malware may attempt DNS hijacking, and alter your Hosts file

```
gc -tail 4 "C:\Windows\System32\Drivers\etc\hosts"

#the above gets the most important bit of the hosts file. If you want more, try t
gc "C:\Windows\System32\Drivers\etc\hosts"
```

Check Host file Time

Don't trust timestamps....however, may be interesting to see if altered recently

```
gci "C:\Windows\System32\Drivers\etc\hosts" | fl *Time*
```

```
CreationTime       : 22/08/2013 14:25:43
CreationTimeUtc    : 22/08/2013 13:25:43
LastAccessTime     : 22/08/2013 14:25:41
LastAccessTimeUtc  : 22/08/2013 13:25:41
LastWriteTime      : 22/08/2013 14:25:41
LastWriteTimeUtc   : 22/08/2013 13:25:41
```

DNS Cache

Collect the DNS cache on an endpoint. Good for catching any sneaky communication or sometimes even DNS C2

```
Get-DnsClientCache | out-string -width 1000
```

```
Get-DnsClientCache | out-string -width 1000


Entry                    RecordName               Record Status    Section TimeTo Data   Data
                                                  Type                     Live   Length
-----                    ----------               ------ ------    ------- ------ ------ ----
g[redacted]7             G[redacted]7.[redacted].local      A      Success   Answer     605      4 10.200.155.34
g[redacted]0             G[redacted]0.[redacted].local      A      Success   Answer     458      4 10.200.155.48
g[redacted]7.[redacted].local      g[redacted]7.[redacted].local      A      Success   Answer     898      4 10.200.154.246
c[redacted]0.[redacted].local      c[redacted]0.[redacted].local      A      Success   Answer     914      4 10.200.154.182
g[redacted]1.[redacted].local      g[redacted]1.[redacted].local      A      Success   Answer     914      4 10.200.155.42
c[redacted]8.[redacted].local      c[redacted]8.[redacted].local      A      Success   Answer     914      4 10.200.154.181
```

Investigate DNS

The above command will likely return a lot of results you don't really need about the communication between 'trusted' endpoints and servers. We can filter these 'trusted' hostnames out with regex, until we're left with less common results.

On the second line of the below code, change up and insert the regex that will filter out your machines. For example, if your machines are generally called WrkSt1001.corp.local, or ServStFAX.corp.local, you can regex out that first poriton so it will exclude any and all machines that share this - so `workst|servst` would do the job. You don't need to wildcard here.

Be careful though. If you are too generic and liberal, you may end up filtering out malicious and important results. It's bettter to be a bit specific, and drill down further to amake sure you aren't filtering out important info. So for example, I wouldn't suggest filtering out short combos of

letters or numbers `ae|ou|34|`

```
Get-DnsClientCache |
? Entry -NotMatch "workst|servst|memes|kerb|ws|ocsp" |
out-string -width 1000
```

If there's an IP you're sus of, you can always take it to WHOIS or VirusTotal, as well see for other instances it appears in your network and what's up to whilst it's interacting there.

IPv6

Since Windows Vitsa, the Windows OS prioritises IPv6 over IPv4. This lends itself to man-in-the-middle attacks, you can find some more info on exploitation here

Get IPv6 addresses and networks

```
Get-NetIPAddress -AddressFamily IPv6  | ft Interfacealias, IPv6Address
```

```
Interfacealias                IPv6Address
--------------                -----------
vEthernet (Ethernet 2)        fe80::30c8:c062:82f7:9
vEthernet (WiFi)              fe80::bd68:272d:67f1:c
vEthernet (Ethernet)          fe80::fd26:fd12:4444:c
vEthernet (Default Switch)    fe80::e8ae:b673:259f:6
Ethernet 2                    fe80::e846:9d07:c484:4
Bluetooth Network Connection  fe80::69c7:cf9d:f26a:6
Local Area Connection* 1      fe80::54d1:2838:f6af:c
Ethernet                      fe80::24a1:661c:9a7c:f
WiFi                          fe80::88b7:a761:3f6e:5
Loopback Pseudo-Interface 1   ::1
```

Disable Priority Treatment of IPv6

You probably don't want to switch IPv6 straight off. And if you DO want to, then it's probably better at a DHCP level. But what we can do is change how the OS will prioritise the IPv6 over IPv4.

```
#check if machine prioritises IPv6
ping $env:COMPUTERNAME -n 4 # if this returns an IPv6, the machine prioritises th

#Reg changes to de-prioritise IPv6
New-ItemProperty “HKLM:\SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters\” -Na

#If this reg already exists and has values, change the value
Set-ItemProperty “HKLM:\SYSTEM\CurrentControlSet\Services\Tcpip6\Parameters\” -Na

#you need to restart the computer for this to take affect
#Restart-Computer
```

```
Reply from fe80::bd68:272d:67f1:d29a%47: time<1ms
Reply from fe80::bd68:272d:67f1:d29a%47: time<1ms
Reply from fe80::bd68:272d:67f1:d29a%47: time<1ms
Reply from fe80::bd68:272d:67f1:d29a%47: time<1ms

Ping statistics for fe80::bd68:272d:67f1:d29a%47:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

BITS Queries

```
Get-BitsTransfer|
fl DisplayName,JobState,TransferType,FileList, OwnerAccount,BytesTransferred,Crea

## filter out common bits jobs in your enviro, ones below are just an example, yo
Get-BitsTransfer|
| ? displayname -notmatch "WU|Office|Dell_Asimov|configjson" |
fl DisplayName,JobState,TransferType,FileList, OwnerAccount,BytesTransferred,Crea

## Hunt down BITS transfers that are UPLOADING, which may be sign of data exfil
Get-BitsTransfer|
? TransferType -match "Upload" |
fl DisplayName,JobState,TransferType,FileList, OwnerAccount,BytesTransferred,Crea
```

```
  Get-BitsTransfer | ? displayname -notmatch "WU|Office|Dell_Asimov|configjson"
tionTime
```

```
DisplayName            : UpdateDescriptionXml
FileList               : {https://g.live.com/1rewlive5skydrive/ODSUProduction64}
JobState               : Transferred
TransferType           : Download
OwnerAccount           : NT AUTHORITY\SYSTEM
BytesTransferred       : 726
CreationTime           : 10/31/2021 11:58:15 PM
TransferCompletionTime : 11/1/2021 8:45:15 AM
```

Remoting Queries

▶ section contents

Powershell Remoting

Get Powershell sessions created

```
Get-PSSession
```

Query WinRM Sessions Deeper

You can query the above even deeper.

```
get-wsmaninstance -resourceuri shell -enumerate |
select Name, State, Owner, ClientIP, ProcessID, MemoryUsed,
@{Name = "ShellRunTime"; Expression = {[System.Xml.XmlConvert]::ToTimeSpan($_.She
@{Name = "ShellInactivity"; Expression = {[System.Xml.XmlConvert]::ToTimeSpan($_.
```

```
[localhost]: PS C:\> get-wsmaninstance -resourceuri shell -enumerate |
>> select Name, State, Owner, ClientIP, ProcessID, MemoryUsed,
>> @{Name = "ShellRunTime"; Expression = {[System.Xml.XmlConvert]::ToTimeSpan($_.ShellRunTime)}},
>> @{Name = "ShellInactivity"; Expression = {[System.Xml.XmlConvert]::ToTimeSpan($_.ShellInactivity)}}


Name            : WinRM2
State           : Connected
Owner           : CASTLE\Administrator
ClientIP        : ::1
ProcessId       : 3212
MemoryUsed      : 71MB
ShellRunTime    : 00:04:26
ShellInactivity : 00:00:00
```

The ClientIP field will show the original IP address that WinRM'd to the remote machine. The times under the Shell fields at the bottom have been converted into HH:MM:SS, so in the above example, the remote PowerShell session has been running for 0 hours, 4 minutes, and 26 seconds.

Remoting Permissions

```
Get-PSSessionConfiguration |
fl Name, PSVersion, Permission
```

```
PS C:\Users\Administrator> Get-PSSessionConfiguration |
>> fl Name, PSVersion, Permission


Name        : microsoft.powershell
PSVersion   : 5.1
Permission  : NT AUTHORITY\INTERACTIVE AccessAllowed,
              BUILTIN\Administrators AccessAllowed, BUILTIN\Remote
              Management Users AccessAllowed

Name        : microsoft.powershell.workflow
PSVersion   : 5.1
Permission  : BUILTIN\Administrators AccessAllowed, BUILTIN\Remote
              Management Users AccessAllowed

Name        : microsoft.powershell32
PSVersion   : 5.1
Permission  : NT AUTHORITY\INTERACTIVE AccessAllowed,
              BUILTIN\Administrators AccessAllowed, BUILTIN\Remote
              Management Users AccessAllowed

Name        : microsoft.windows.servermanagerworkflows
PSVersion   : 3.0
Permission  : NT AUTHORITY\INTERACTIVE AccessAllowed,
              BUILTIN\Administrators AccessAllowed
```

Check Constrained Language

To be honest, constrained language mode in Powershell can be trivally easy to mitigate for an adversary. And it's difficult to implement persistently. But anyway. You can use this quick variable to confirm if a machine has a constrained language mode for pwsh.

```
$ExecutionContext.SessionState.LanguageMode
```

```
$ExecutionContext.SessionState.LanguageMode
```

```
FullLanguage
```

RDP settings

You can check if RDP capability is permissioned on an endpoint

```
if ((Get-ItemProperty "hklm:\System\CurrentControlSet\Control\Terminal Server").f
```

If you want to block RDP

```
Set-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Control\Terminal Server' -
#Firewall it out too
Disable-NetFirewallRule -DisplayGroup "Remote Desktop"
```

Query RDP Logs

Knowing who is RDPing in your enviroment, and from where, is important. Unfortunately, RDP logs are balllache. Threat hunting blogs like this one can help you narrow down what you are looking for when it comes to RDP

Let's call on one of the RDP logs, and filter for event ID 1149, which means a RDP connection has been made. Then let's filter out any IPv4 addresses that begin with 10.200, as this is the internal IP schema. Perhaps I want to hunt down public IP addresses, as this would suggest the RDP is exposed to the internet on the machine and an adversary has connected with correct credentials!!!

Two logs of interest

- Microsoft-Windows-TerminalServices-RemoteConnectionManager/Operational
- Microsoft-Windows-TerminalServices-LocalSessionManager%4Operational.evtx

```
# if you acquire a log, change this to get-winevent -path ./RDP_log_you_acquired.
get-winevent -path "./Microsoft-Windows-TerminalServices-RemoteConnectionManager%
? id -match 1149 |
sort Time* -descending |
fl time*, message

get-winevent -path ./ "Microsoft-Windows-TerminalServices-LocalSessionManager%4Op
? id -match 21 |
sort Time* -descending |
fl time*, message
```

```
  get-winevent -logname "Microsoft-Windows-TerminalServices-RemoteConnectionManager/Operational" | ? id -match 1149 | ? message -notmatch '10.200'  |ft message -
wrap
```

```
Message
-------
Remote Desktop Services: User authentication succeeded:

User: vmware.admin
Domain:
Source Network Address: 10.202.202.90
Remote Desktop Services: User authentication succeeded:

User: vmware.admin
Domain:
Source Network Address: 10.202.202.7
```

Current RDP Sessions

You can query the RDP sessions that a system is currently running

```
qwinsta

:: get some stats
qwinsta /counter
```

```
[11/12/2021 11:05:12] | PS C:\Users\IEUser > qwinsta
 SESSIONNAME       USERNAME                 ID  STATE   TYPE        DEVICE
 services                                    0  Disc
>rdp-tcp#2         IEUser                   1  Active
 console                                     2  Conn
 rdp-tcp                                 65536  Listen
[11/12/2021 11:05:14] | PS C:\Users\IEUser > qwinsta /counter
 SESSIONNAME       USERNAME                 ID  STATE   TYPE        DEVICE
 services                                    0  Disc
>rdp-tcp#2         IEUser                   1  Active
 console                                     2  Conn
 rdp-tcp                                 65536  Listen
Total sessions created: 3
Total sessions disconnected: 1
Total sessions reconnected: 1
```

You can read here about how to evict a malicious user from a session and change the creds rapidly to deny them future access

Check Certificates

```
gci "cert:\" -recurse | fl FriendlyName, Subject, Not*
```

```
FriendlyName : Microsoft Root Certificate Authority
Subject      : CN=Microsoft Root Certificate Authority, DC=microsoft, DC=com
NotAfter     : 10/05/2021 00:28:13
NotBefore    : 10/05/2001 00:19:22

FriendlyName : Thawte Timestamping CA
Subject      : CN=Thawte Timestamping CA, OU=Thawte Certification, O=Thawte,
               L=Durbanville, S=Western Cape, C=ZA
NotAfter     : 31/12/2020 23:59:59
NotBefore    : 01/01/1997 00:00:00

FriendlyName :
Subject      : CN=COMODO RSA Certification Authority, O=COMODO CA Limited,
```

Certificate Dates

You will be dissapointed how many certificates are expired but still in use. Use the `-ExpiringInDays` flag

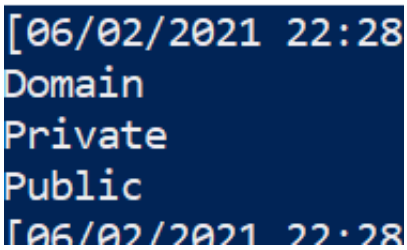
```
gci "cert:\*" -recurse -ExpiringInDays 0 | fl FriendlyName, Subject, Not*
```

Firewall Queries

▶ section contents

Retrieve Firewall profile names

```
(Get-NetFirewallProfile).name
```



Retrieve rules of specific profile

Not likely to be too useful getting all of this information raw, so add plenty of filters

```
Get-NetFirewallProfile -Name Public | Get-NetFirewallRule
##filtering it to only show rules that are actually enabled
Get-NetFirewallProfile -Name Public | Get-NetFirewallRule | ? Enabled -eq "true"
```

```
Name                  : WMI-WINMGMT-In-TCP
DisplayName           : Windows Management Instrumentation (WMI-In)
Description           : Inbound rule to allow WMI traffic for remote Windows
                        Management Instrumentation. [TCP]
DisplayGroup          : Windows Management Instrumentation (WMI)
Group                 : @FirewallAPI.dll,-34251
Enabled               : False
Profile               : Private, Public
Platform              : {}
Direction             : Inbound
Action                : Allow
EdgeTraversalPolicy   : Block
LooseSourceMapping    : False
LocalOnlyMapping      : False
Owner                 :
PrimaryStatus         : OK
Status                : The rule was parsed successfully from the store. (65536)
EnforcementStatus     : NotApplicable
PolicyStoreSource     : PersistentStore
PolicyStoreSourceType : Local
```

Filter all firewall rules

```
#show firewall rules that are enabled
Get-NetFirewallRule | ? Enabled -eq "true"
#will show rules that are not enabled
Get-NetFirewallRule | ? Enabled -notmatch "true"

##show firewall rules that pertain to inbound
Get-NetFirewallRule | ? direction -eq "inbound"
#or outbound
Get-NetFirewallRule | ? direction -eq "outbound"

##stack these filters
Get-NetFirewallRule | where {($_.Enabled -eq "true" -and $_.Direction -eq "inboun
#or just use the built in flags lol
Get-NetFirewallRule -Enabled True -Direction Inbound
```

Code Red

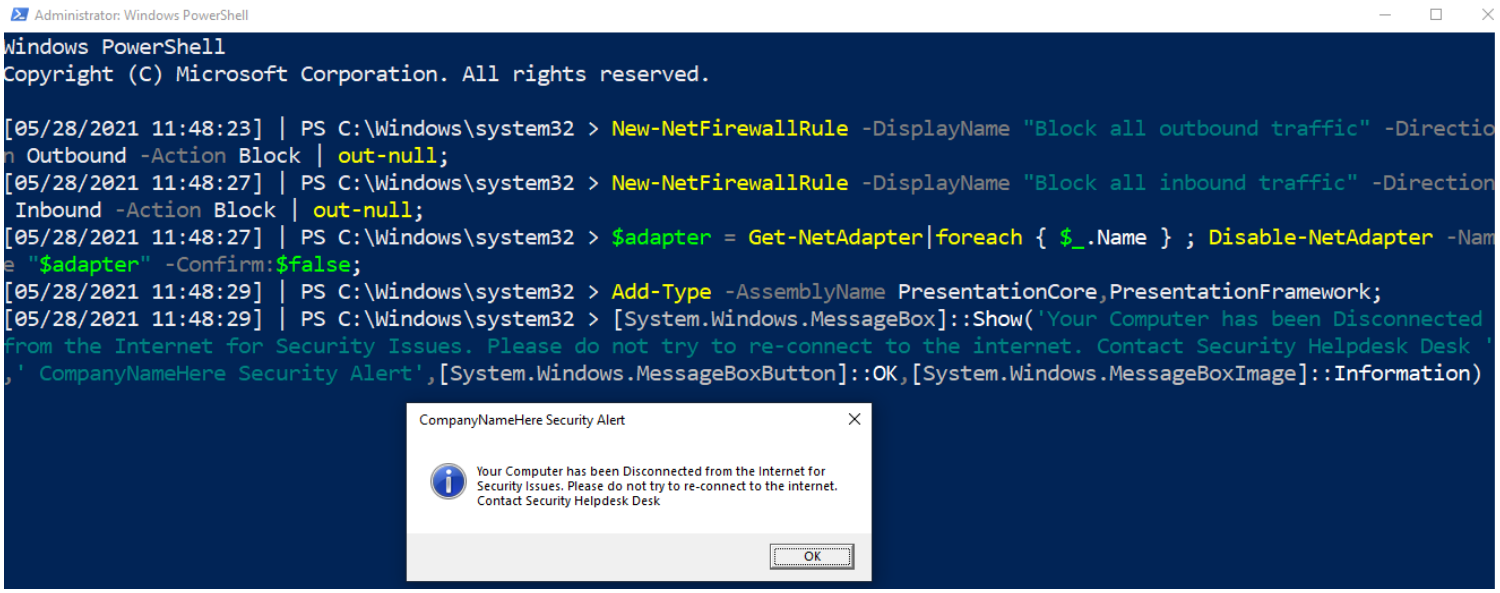
Isolate Endpoint

Disconnect network adaptor, firewall the fuck out of an endpoint, and display warning box

This is a code-red command. Used to isolate a machine in an emergency.

In the penultimate and final line, you can change the text and title that will pop up for the user

```
New-NetFirewallRule -DisplayName "Block all outbound traffic" -Direction Outbound
New-NetFirewallRule -DisplayName "Block all inbound traffic" -Direction Inbound -
$adapter = Get-NetAdapter|foreach { $_.Name } ; Disable-NetAdapter -Name "$adapte
Add-Type -AssemblyName PresentationCore,PresentationFramework;
[System.Windows.MessageBox]::Show('Your Computer has been Disconnected from the I
```



SMB Queries

▶ section contents

List Shares

```
Get-SMBShare
```

```
Get-SMBShare
```

```
Name   ScopeName Path                                    Description
----   --------- ----                                    -----------
ADMIN$ *         C:\Windows                              Remote Admin
C$     *         C:\                                     Default share
IPC$   *                                                 Remote IPC
print$ *         C:\Windows\system32\spool\drivers       Printer Drivers
```

List client-to-server SMB Connections

Dialect just means verison. SMB3, SMB2 etc

```
Get-SmbConnection

#just show SMB Versions being used. Great for enumeration flaws in enviro - i.e,
Get-SmbConnection |
select Dialect, Servername, Sharename | sort Dialect
```

```
Get-SmbConnection | ft
```

```
ServerName ShareName UserName       Credential           Dialect NumOpens
---------- --------- --------       ----------           ------- --------
[redacted]1    Shared    [redacted]ton [redacted]ton 3.0.2   5
[redacted]s1   IPC$      [redacted]ton [redacted]ton 3.0.2   0
[redacted]1    IPC$      [redacted]ton [redacted]ton 2.1     0
```

```
Dialect Servername Sharename
------- ---------- ---------
2.1             1  IPC$
3.0.2           1  Shared
3.0.2           s1 IPC$
```

Remove an SMB Share

```
Remove-SmbShare -Name MaliciousShare -Confirm:$false -verbose
```

Process Queries

▶ section contents

Processes and TCP Connections

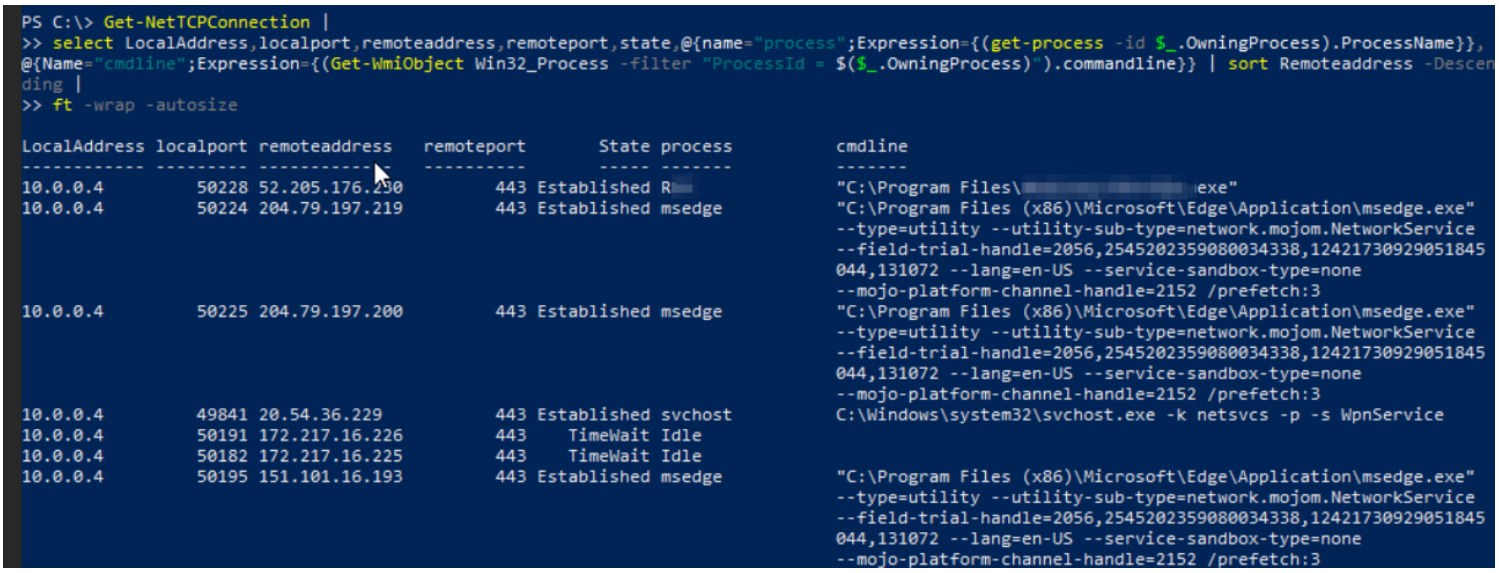
I have a neat one-liner for you. This will show you the local IP and port, the remote IP andport, the process name, and the underlying executable of the process!

You could just use `netstat -b`, which gives you SOME of this data

```
PS C:\> netstat -b

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    10.0.0.4:49841         20.54.36.229:https     ESTABLISHED
  WpnService
 [svchost.exe]
  TCP    10.0.0.4:50257         104.21.54.39:https     ESTABLISHED
 [msedge.exe]
  TCP    10.0.0.4:50284         192.168.1.49:8080      CLOSE_WAIT
 [msedge.exe]
  TCP    10.0.0.4:50286         10.0.0.77:9200         SYN_SENT
 [metricbeat.exe]
  TCP    10.0.0.4:50287         10.0.0.77:9200         SYN_SENT
 [filebeat.exe]
  TCP    10.0.0.4:50288         10.0.0.77:9200         SYN_SENT
```

But instead, try this bad boy on for size:

```
Get-NetTCPConnection |
select LocalAddress,localport,remoteaddress,remoteport,state,@{name="process";Exp
sort Remoteaddress -Descending | ft -wrap -autosize
```



```
PS C:\> Get-NetTCPConnection |
>> select LocalAddress,localport,remoteaddress,remoteport,state,@{name="process";Expression={(get-process -id $_.OwningProcess).ProcessName}},
@{Name="cmdline";Expression={(Get-WmiObject Win32_Process -filter "ProcessId = $($_.OwningProcess)").commandline}} | sort Remoteaddress -Descen
ding |
>> ft -wrap -autosize

LocalAddress localport remoteaddress   remoteport       State process         cmdline
------------ --------- -------------   ----------       ----- -------         -------
10.0.0.4         50228 52.205.176.230         443 Established R               "C:\Program Files\                exe"
10.0.0.4         50224 204.79.197.219         443 Established msedge          "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
                                                                              --type=utility --utility-sub-type=network.mojom.NetworkService
                                                                              --field-trial-handle=2056,2545202359080034338,12421730929051845
                                                                              044,131072 --lang=en-US --service-sandbox-type=none
                                                                              --mojo-platform-channel-handle=2152 /prefetch:3
10.0.0.4         50225 204.79.197.200         443 Established msedge          "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
                                                                              --type=utility --utility-sub-type=network.mojom.NetworkService
                                                                              --field-trial-handle=2056,2545202359080034338,12421730929051845
                                                                              044,131072 --lang=en-US --service-sandbox-type=none
                                                                              --mojo-platform-channel-handle=2152 /prefetch:3
10.0.0.4         49841 20.54.36.229           443 Established svchost         C:\Windows\system32\svchost.exe -k netsvcs -p -s WpnService
10.0.0.4         50191 172.217.16.226         443    TimeWait Idle
10.0.0.4         50182 172.217.16.225         443    TimeWait Idle
10.0.0.4         50195 151.101.16.193         443 Established msedge          "C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
                                                                              --type=utility --utility-sub-type=network.mojom.NetworkService
                                                                              --field-trial-handle=2056,2545202359080034338,12421730929051845
                                                                              044,131072 --lang=en-US --service-sandbox-type=none
                                                                              --mojo-platform-channel-handle=2152 /prefetch:3
```

Show all processes and their associated user

```
get-process * -Includeusername
```

```
Handles      WS(K)   CPU(s)     Id UserName               Proc
-------      -----   ------     -- --------               ----
    114       6136     1.33   3620 NT AUTHORITY\SYSTEM    AMPW
    124       9244     6.84   3608 NT AUTHORITY\SYSTEM    clie
    114       7732     0.17    332 NT AUTHORITY\SYSTEM    conh
     94       5548     2.03    860 NT AUTHORITY\SYSTEM    conh
     95       5512     0.98   2368 NT AUTHORITY\SYSTEM    conh
     95       5988     0.02   3820 NT AUTHORITY\SYSTEM    conh
    283       4144     5.41    336                        csrs
    137       7036     8.00    404                        csrs
    250      16228    41.61   1944 NT AUTHORITY\SYSTEM    dfsr
    182       9228    32.67   1228 NT AUTHORITY\SYSTEM    dfss
  10910     188028   744.67   1852 NT AUTHORITY\SYSTEM    dns
      0          4                0                      Idle
```

Try this one if you're hunting down suspicious processes from users

```
gwmi win32_process |
Select Name,@{n='Owner';e={$_.GetOwner().User}},CommandLine |
sort Name -unique -descending | Sort Owner | ft -wrap -autosize
```

```
PS C:\> gwmi win32_process |
>> Select Name,@{n='Owner';e={$_.GetOwner().User}},CommandLine |
>> sort Name -unique -descending | Sort Owner | ft -wrap -autosize

Name                      Owner          CommandLine
----                      -----          -----------
System Idle Process
System
dwm.exe                   DWM-1          "dwm.exe"
SearchApp.exe             Frank          "C:\Windows\SystemApps\Microsoft.Windows.Search_cw5n1h2txyewy\SearchApp.exe"
                                         -ServerName:CortanaUI.AppX8z9r6jm96hw4bsbneegw0kyxx296wr9t.mca
RuntimeBroker.exe         Frank          C:\Windows\System32\RuntimeBroker.exe -Embedding
SecurityHealthSystray.exe Frank          "C:\Windows\System32\SecurityHealthSystray.exe"
ShellExperienceHost.exe   Frank          "C:\Windows\SystemApps\ShellExperienceHost_cw5n1h2txyewy\ShellExperienceHost.exe"
                                         -ServerName:App.AppXtk181tbxbce2qsex02s8tw7hfxa9xb3t.mca
sihost.exe                Frank          sihost.exe
ApplicationFrameHost.exe  Frank          C:\Windows\system32\ApplicationFrameHost.exe -Embedding
ctfmon.exe                Frank          "ctfmon.exe"
dllhost.exe               Frank          C:\Windows\system32\DllHost.exe /Processid:{973D20D7-562D-44B9-B70B-5A0F49CCDF3F}
conhost.exe               Frank          \??\C:\Windows\system32\conhost.exe 0x4
Cortana.exe               Frank          "C:\Program
                                         Files\WindowsApps\Microsoft.549981C3F5F10_3.2111.12605.0_x64__8wekyb3d8bbwe\Cortana.exe"
                                         -ServerName:App.AppX2y379sjp88wjq1y80217mddj3fargf2y.mca
OneDrive.exe              Frank          "C:\Users\Frank\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
powershell.exe            Frank          "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
```

Get specific info about the full path binary that a process is running

```
gwmi win32_process |
Select Name,ProcessID,@{n='Owner';e={$_.GetOwner().User}},CommandLine |
sort name | ft -wrap -autosize | out-string
```

```
Name                      ProcessID Owner  CommandLine
----                      --------- -----  -----------
ApplicationFrameHost.exe       7820 Phoebe C:\WINDOWS\system32\ApplicationFrameHost.exe
cmd.exe                        6312 Phoebe cmd  /C "\\10.10.14.3\\kali\\nc.exe 10.10.14.
cmd.exe                        6464 Phoebe cmd.exe
cmd.exe                        1516 Phoebe cmd.exe /c "cmd /C "\\10.10.14.3\\kali\\nc.ex
                                           cmd.exe" 2>&1"
cmd.exe                        4464 Phoebe C:\WINDOWS\system32\cmd.exe  /K Powerless.bat
conhost.exe                    6588 Phoebe \??\C:\WINDOWS\system32\conhost.exe 0x4
conhost.exe                     872 Phoebe \??\C:\WINDOWS\system32\conhost.exe 0x4
conhost.exe                    2492 Phoebe \??\C:\WINDOWS\system32\conhost.exe 0x4
csrss.exe                       424
csrss.exe                       536
ctfmon.exe                     4596 Phoebe
dllhost.exe                    3240
dllhost.exe                    7408 Phoebe C:\WINDOWS\system32\DllHost.exe
                                           /Processid:{973D20D7-562D-44B9-B70B-5A0F49CCD
dwm.exe                         996
explorer.exe                   5180 Phoebe C:\WINDOWS\Explorer.EXE
```

Get specific info a process is running

```
get-process -name "nc" | ft Name, Id, Path,StartTime,Includeusername -autosize
```

```
Name   Id IncludeUsername Path                       StartTime
----   -- ---------------- ----                       ---------
nc   4312                 \\10.10.14.3\kali\nc.exe 6/5/2021 10:46:04 AM
```

Is a specific process a running on a machine or not

```
$process = "memes";
if (ps |  where-object ProcessName -Match "$process") {Write-Host "$process succe
```

Example of process that is absent

```
 if (get-process | select-object -property ProcessName | where-object {$_.ProcessName -Match "memes*"})
{Write-Host "memes successfully installed on " -NoNewline ; hostname}
else {write-host "memes absent from " -NoNewline ; hostname}
```

```
memes absent from HU      N1
```

Example of process that is present

```
  if (get-process | select-object -property ProcessName | where-object {$_.ProcessName -Match "GoogleUpdate"})
{Write-Host "GoogleUpdate successfully installed on " -NoNewline ; hostname}
else {write-host "GoogleUpdate absent from " -NoNewline ; hostname}
```

```
GoogleUpdate successfully installed on HU      N1
```

Get process hash

Great to make malicious process stand out. If you want a different Algorithm, just change it after `-Algorithm` to something like `sha256`

```
foreach ($proc in Get-Process | select path -Unique){try
{ Get-FileHash $proc.path -Algorithm sha256 -ErrorAction stop |
ft hash, path -autosize -HideTableHeaders | out-string -width 800 }catch{}}
```

```
 foreach ($proc in Get-Process | select path -Unique)
{try { Get-FileHash $proc.path -Algorithm md5 -ErrorAction stop | Select-Obj
```

```
Hash                             Path
----                             ----
75CAF3F6AFCD6E21FBA5DABA97E74C3A C:\Program Files (x86)\Quest\KACE\AMPWatchD
9C3F2E077CC85529DDC8FD2F857F5E0A C:\Program Files (x86)\Trend Micro\Endpoint
11AD39C99B8E8F15B5175EDE9BF7CC38 C:\ProgramData\quest\kace\modules\clientide
09AC6D04F935EFD05AFDAC2733D37598 C:\Program Files (x86)\Trend Micro\Security
5E65FB88E7D005750F777D4D3CCE64A8 C:\Program Files (x86)\Trend Micro\Endpoint
0BCA3F16DD527B4150648EC1E36CB22A C:\Program Files (x86)\Google\Update\Google
```

Show all DLLs loaded with a process

```
get-process -name "memestask" -module
```

```
get-process -name "googleupdate" -module
```

```
   Size(K) ModuleName                                                  FileName
   ------- ----------                                                  --------
       152 GoogleUpdate.exe                                            C:\Program Fil
      1544 ntdll.dll                                                   C:\Windows\SYS
       896 KERNEL32.DLL                                                C:\Windows\Sys
      1672 KERNELBASE.dll                                              C:\Windows\Sys
       476 ADVAPI32.dll                                                C:\Windows\Sys
       760 msvcrt.dll                                                  C:\Windows\Sys
       260 sechost.dll                                                 C:\Windows\Sys
       772 RPCRT4.dll                                                  C:\Windows\Sys
       124 SspiCli.dll                                                 C:\Windows\Sys
        40 CRYPTBASE.dll                                               C:\Windows\Sys
```

Alternatively, pipe `|fl` and it will give a granularity to the DLLs

```
get-process -name "googleupdate" -module | fl
```

```
ModuleName        : GoogleUpdate.exe
FileName          : C:\Program Files (x86)\Google\Update\GoogleUpdate.exe
BaseAddress       : 15007744
ModuleMemorySize  : 155648
EntryPointAddress : 15037506
FileVersionInfo   : File:             C:\Program Files
                    (x86)\Google\Update\GoogleUpdate.exe
                    InternalName:     Google Update
                    OriginalFilename: GoogleUpdate.exe
                    FileVersion:      1.3.35.451
                    FileDescription:  Google Installer
                    Product:          Google Update
                    ProductVersion:   1.3.35.451
                    Debug:            False
                    Patched:          False
                    PreRelease:       False
                    PrivateBuild:     False
                    SpecialBuild:     False
                    Language:         English (United States)

Site              :
Container         :
Size              : 152
Company           : Google LLC
FileVersion       : 1.3.35.451
```

Identify process CPU usage

```
(Get-Process -name "googleupdate").CPU | fl
```

```
(Get-Process -name "googleupdate").CPU | fl
```

```
0.0625
```

I get mixed results with this command but it's supposed to give the percent of CPU usage. I need to work on this, but I'm putting it in here so the world may bare wittness to my smooth brain.

```
$ProcessName = "symon" ;
$ProcessName = (Get-Process -Id $ProcessPID).Name;
```

```
$CpuCores = (Get-WMIObject Win32_ComputerSystem).NumberOfLogicalProcessors;
$Samples = (Get-Counter "\Process($Processname*)\% Processor Time").CounterSample
$Samples | Select `InstanceName,@{Name="CPU %";Expression={[Decimal]::Round(($_.C
```

```
InstanceName CPU %
------------ -----
googleupdate     0
```

Sort by least CPU-intensive processes

Right now will show the lower cpu-using proccesses...useful as malicious process probably won't be as big a CPU as Chrome, for example. But change first line to `Sort CPU –descending` if you want to see the chungus processes first

```
gps | Sort CPU |
Select –Property ProcessName, CPU, ID, StartTime |
ft –autosize –wrap | out-string –width 800
```

```
ProcessName    CPU           Id StartTime
-----------    ---           -- ---------
Idle                          0
conhost        0             156 06/06/2021 12:15:40
smss           0.078125      236 12/03/2021 10:14:28
unsecapp       0.09375      4664 24/05/2021 10:36:48
svchost        0.140625     2704 24/05/2021 10:35:48
svchost        0.1875       2976 12/03/2021 10:15:01
powershell     0.25          284 06/06/2021 12:15:40
winlogon       0.234375      440 12/03/2021 10:14:29
svchost        0.3125       1584 12/03/2021 10:14:32
msdtc          0.328125     1992 12/03/2021 10:14:32
dllhost        0.484375     1764 12/03/2021 10:14:32
wininit        0.5           404 12/03/2021 10:14:29
VGAuthService  0.515625     1120 12/03/2021 10:14:31
svchost        0.546875     1040 12/03/2021 10:14:31
dwm            0.703125      744 12/03/2021 10:14:31
dmgsvc         0.734375      848 12/03/2021 10:16:12
diawp          0.734375      816 06/06/2021 11:44:21
Dmgupgradesvc  0.78125       3532 12/03/2021 10:16:17
csrss          0.8125         412 12/03/2021 10:14:29
```

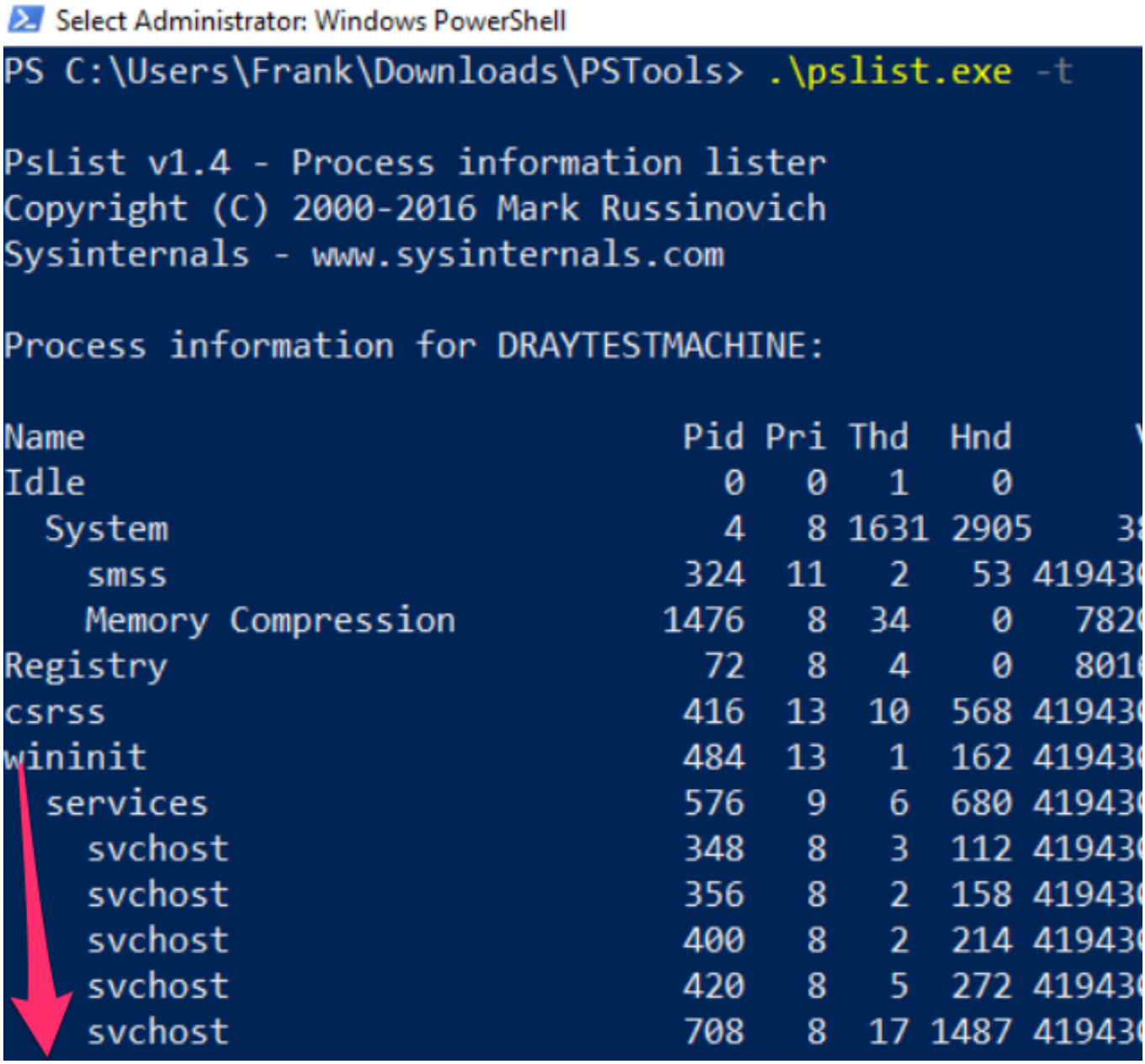
Stop a Process

```
Get-Process -Name "memeprocess" | Stop-Process -Force -Confirm:$false -verbose
```

Process Tree

You can download the PsList exe from Sysinternals

Fire it off with the `-t` flag to create a parent-child tree of the processes



Recurring Task Queries

▶ section contents

Get scheduled tasks

Identify the user behind a command too. Great at catching out malicious schtasks that perhaps are imitating names, or a process name

```
schtasks /query /FO CSV /v | convertfrom-csv |
where { $_.TaskName -ne "TaskName" } |
select "TaskName","Run As User", Author, "Task to Run"|
fl | out-string
```

```
TaskName    : \Microsoft\Windows\Workplace Join\Recovery-Check
Run As User : INTERACTIVE
Author      : N/A
Task To Run : %SystemRoot%\System32\dsregcmd.exe /checkrecovery

TaskName    : \Microsoft\Windows\WwanSvc\NotificationTask
Run As User : INTERACTIVE
Author      : Microsoft Corporation
Task To Run : %SystemRoot%\System32\WiFiTask.exe wwan

TaskName    : \Microsoft\Windows\WwanSvc\OobeDiscovery
Run As User : SYSTEM
Author      : N/A
```

Get a specific schtask

```
Get-ScheduledTask -Taskname "wifi*" | fl *
```

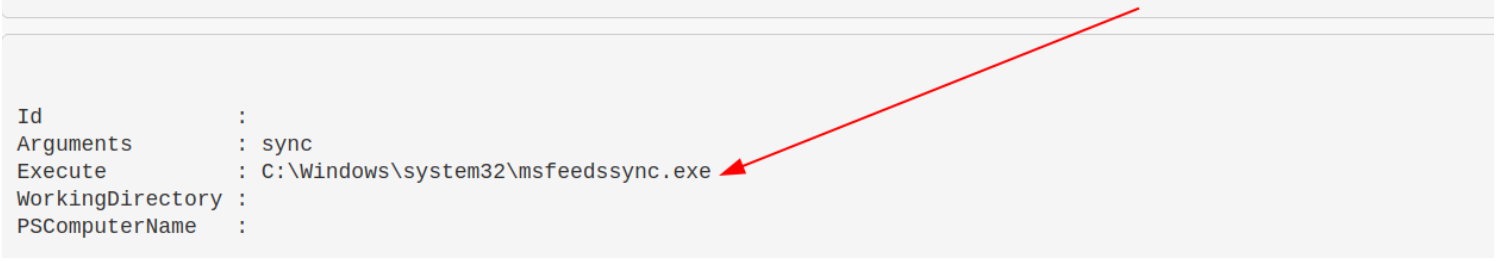
```
[06/02/2021 23:10:26] | PS C:\Windows\system32 > Get-ScheduledTask -Taskname "wifi*"

TaskPath                                    TaskName                    State
--------                                    --------                    -----
\Microsoft\Windows\NlaSvc\                  WiFiTask                    Ready
\Microsoft\Windows\WCM\                     WiFiTask                    Ready
```

To find the commands a task is running

Great one liner to find exactly WHAT a regular task is doing

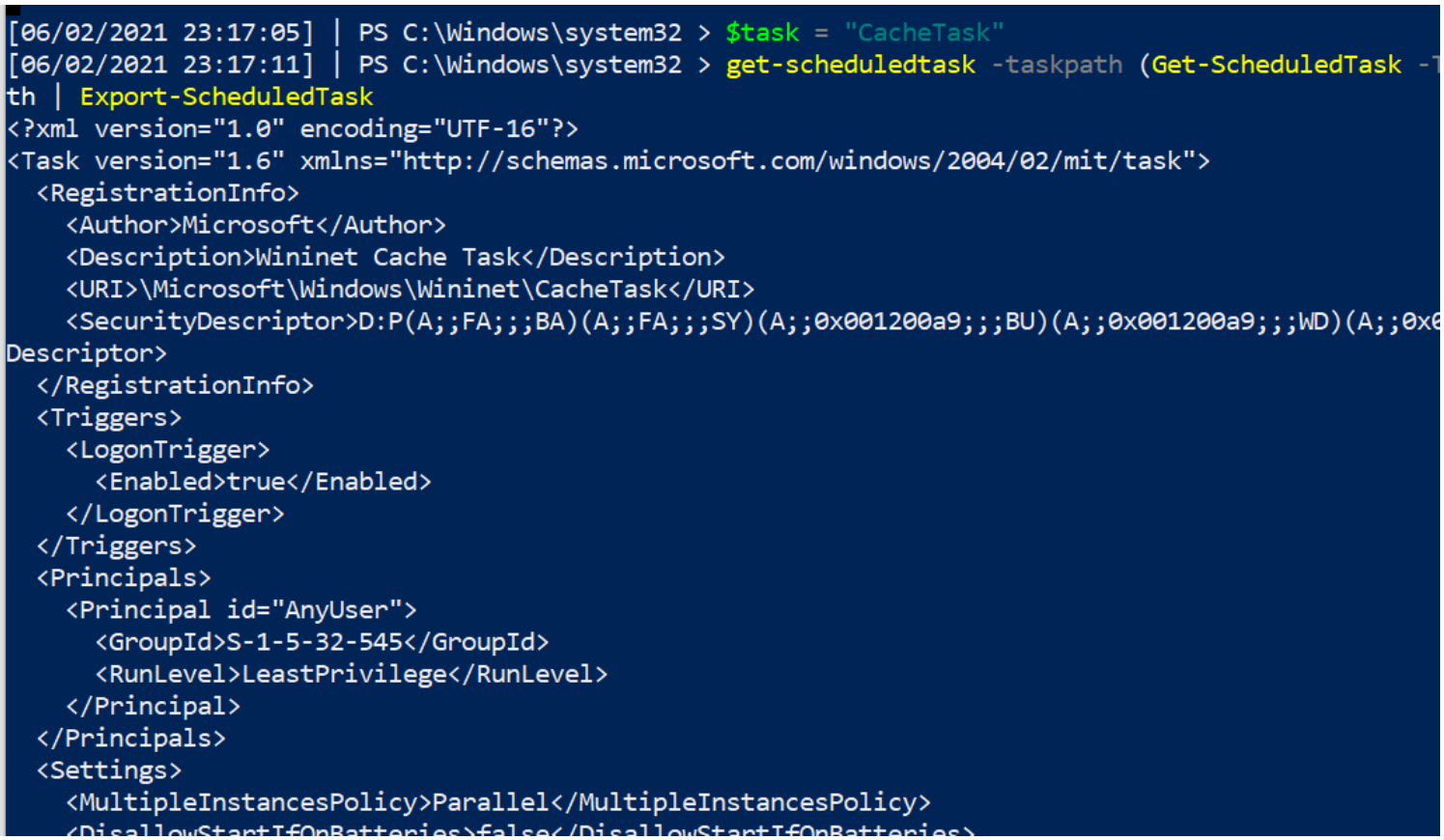
```
$task = Get-ScheduledTask | where TaskName -EQ "meme task";
$task.Actions
```

```
 $task = Get-ScheduledTask | where TaskName -EQ "User_Feed_Synchronization-{74441243-60DC-44AA-A802-5BCC62B97518}";
$task.Actions
```



And a command to get granularity behind the schtask requires you to give the taskpath. Tasks with more than one taskpath will throw an error here

```
$task = "CacheTask";
get-scheduledtask -taskpath (Get-ScheduledTask -Taskname "$task").taskpath | Expo
#this isn't the way the microsoft docs advise.
    ##But I prefer this, as it means I don't need to go and get the taskpath whe
```



To stop the task

```
Get-ScheduledTask "memetask" | Stop-ScheduledTask -Force -Confirm:$false -verbose
```

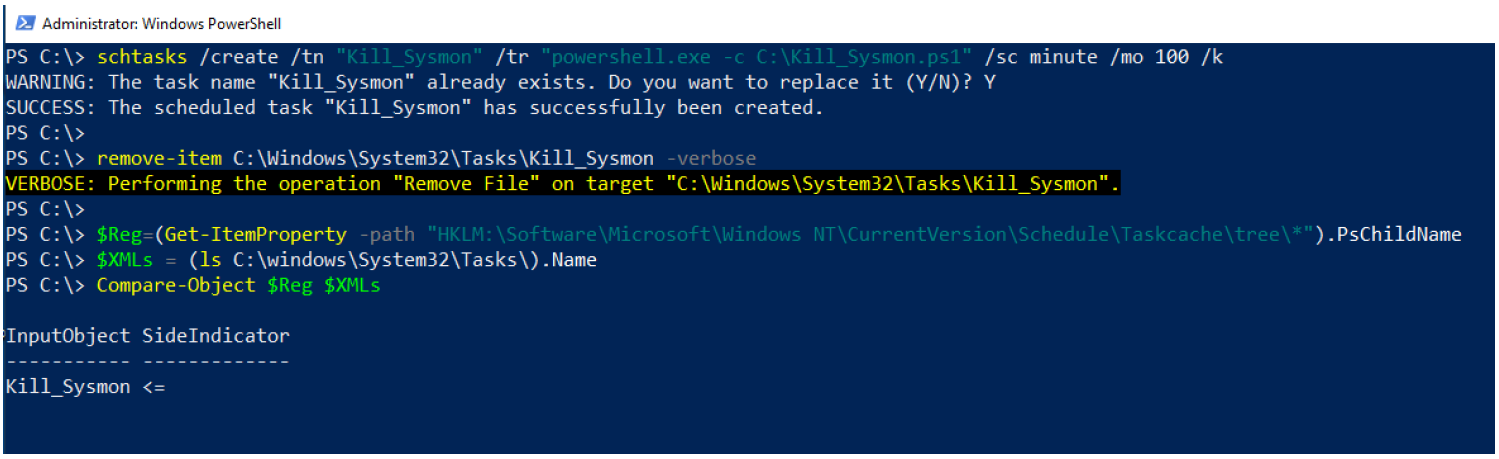
All schtask locations

There's some major overlap here, but it pays to be thorough.

```
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks
C:\Windows\System32\Tasks
C:\Windows\Tasks
C:\windows\SysWOW64\Tasks\
```

You can compare the above for tasks missing from the C:\Windows directories, but present in the Registry.

```
# From my man Anthony Smith - https://www.linkedin.com/in/anthony-c-smith/

$Reg=(Get-ItemProperty -path "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\
$XMLs = (ls C:\windows\System32\Tasks\).Name
Compare-Object $Reg $XMLs
```



Sneaky Schtasks via the Registry

Threat actors have been known to manipulate scheduled tasks in such a way that Task Scheduler no longer has visibility of the recuring task.

However, querying the Registry locations `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree` and `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks`, can reveal a slice of these sneaky tasks.

Shout out to my man @themalwareguy for the $fixedstring line that regexes in/out good/bad characters.

```
# the schtask for our example
# schtasks /create /tn "Find_Me" /tr calc.exe /sc minute /mo 100 /k

# Loop and parse \Taskcache\Tasks Registry location for scheduled tasks
  ## Parses Actions to show the underlying binary / commands for the schtask
  ## Could replace Actions with Trigggers on line 10, after ExpandedProperty
(Get-ItemProperty "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Ta
Foreach-Object {
  write-host "----Schtask ID is $_---" -ForegroundColor Magenta ;
  $hexstring = Get-ItemProperty "HKLM:\Software\Microsoft\Windows NT\CurrentVersi
  $fixedstring = [System.Text.Encoding]::Unicode.GetString($hexstring) -replace '
  write-host $fixedstring
}
```

```
PS C:\> (Get-ItemProperty "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks\*").PSChildName |
>> Foreach-Object {
>>   write-host "----Schtask ID is $_---" -ForegroundColor Magenta ;
>>   $hexstring = (Get-ItemProperty "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks\$_" | Select -ExpandProperty Actions) -join '
' ;
>>  ($hexstring.Split(",",[System.StringSplitOptions]::RemoveEmptyEntries) | ?{$_ -gt '0'} | ForEach{[char][int]"$($_)"}) -join ''
>> }
----Schtask ID is {017020FE-4826-4329-AB78-EA66B95D2AAF}---
□□AllUserswwH±\¿w|□M¥>Ø□pÏt<
----Schtask ID is {02ABD6A5-7C66-4E8A-9BF7-B12354245E8F}---
□□Systemww□ÇDÈØ®ÎM«%M5□ø□$(Arg0)
----Schtask ID is {031C55BC-D055-49F4-B5EF-98F40D8F7841}---
Userww)·1®ýÕ□@¯Bx@t□Zþ&-UserSessionCommand
----Schtask ID is {04622042-F26B-4CCA-815F-E7A8375D87E6}---
□□Systemww)·1®ýÕ□@¯Bx@t□Zþ□-IntegrityCheck
----Schtask ID is {05394804-BDBB-481F-A23B-C7905D714A7A}---
□
UserswwÏóÁ#□Á□E¬®{atìè□
----Schtask ID is {08BFB9C5-8A12-4B63-85A8-0321BE97A32E}---
□□LocalSystemff<%windir%\system32\rundll32.exeJdfdts.dll,DfdGetDefaultPolicyAndSMART
----Schtask ID is {08D81671-F649-4960-878A-41DF593A065E}---
□□LocalSystemffD%windir%\system32\dstokenclean.exe
----Schtask ID is {09996B35-18E3-4A38-BA0A-331A4D6EA3CE}---
8□ÔI§!á□□|äYmwwî1Ã
----Schtask ID is {0C46FA40-EA9F-41B0-8A23-F38E91DECBD4}---
□□LocalSystemww¨>ýÜ`Ù□G□□I
ã□ùO
----Schtask ID is {0C531BCD-530F-4BF9-A057-EEFDDE839C9D}---
□□LocalSystemww□ç~)□º□N□ÓÖçð□Çµ
----Schtask ID is {0C5C92F5-A275-49A6-8F82-00DF4529EC32}---
□□AuthorfffF%systemroot%\system32\usoclient.exe□StartScan
----Schtask ID is {0CAD4002-A672-45EE-B6E3-2EDFECFE05EE}---
□
Usersww¹Ñ¾a□4ì@□A×ñÀc/□□AppLaunch
----Schtask ID is {0DE644A5-9E42-484C-9B9B-C8ED1EB26DDF}---
□□LocalSystemww¹vûX□¬UN¬□Bu□‡Ð`□SYSTEM
----Schtask ID is {10B3FA49-D0E2-4D16-8BB9-3F74ADD128FB}---
□□LocalSystemff:%windir%\system32\bcdboot.exe&%windir% /sysrepair
```

If you don't need to loop to search, because you know what you're gunning for then you can just deploy this

```
$hexstring = (Get-ItemProperty "HKLM:\Software\Microsoft\Windows NT\CurrentVersio
Select -ExpandProperty Actions) -join ',' ; $hexstring.Split(" ")
## can then go to cyberchef, and convert From Decimal with the comma (,) delimine
```

```
PS C:\> $hexstring = (Get-ItemProperty "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks\{8E350038-3475-413A-A1AE-20711DD11C95}" |
Select -ExpandProperty Actions) -join ',' ; $hexstring.Split(" ")
>>
3,0,12,0,0,0,65,0,117,0,116,0,104,0,111,0,114,0,102,102,0,0,0,0,16,0,0,0,99,0,97,0,108,0,99,0,46,0,101,0,120,0,101,0,0,0,0,0,0,0,0,0,0
PS C:\>
```

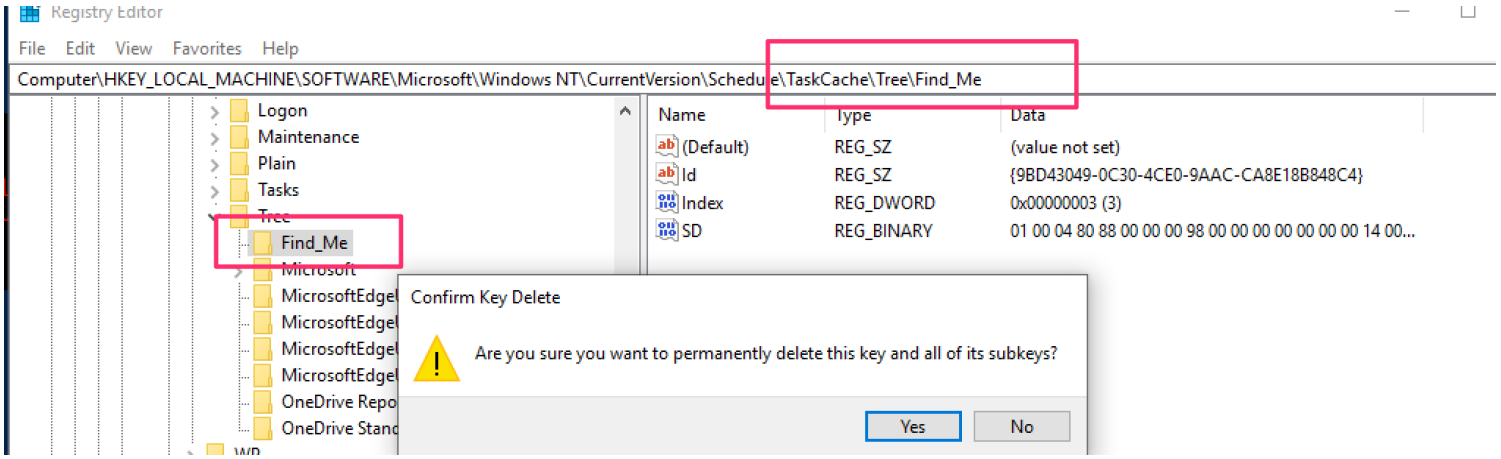
Once you've deployed the above loop, and zoned in on a binary / one-liner that seems sus, you can query it in the other Registry location

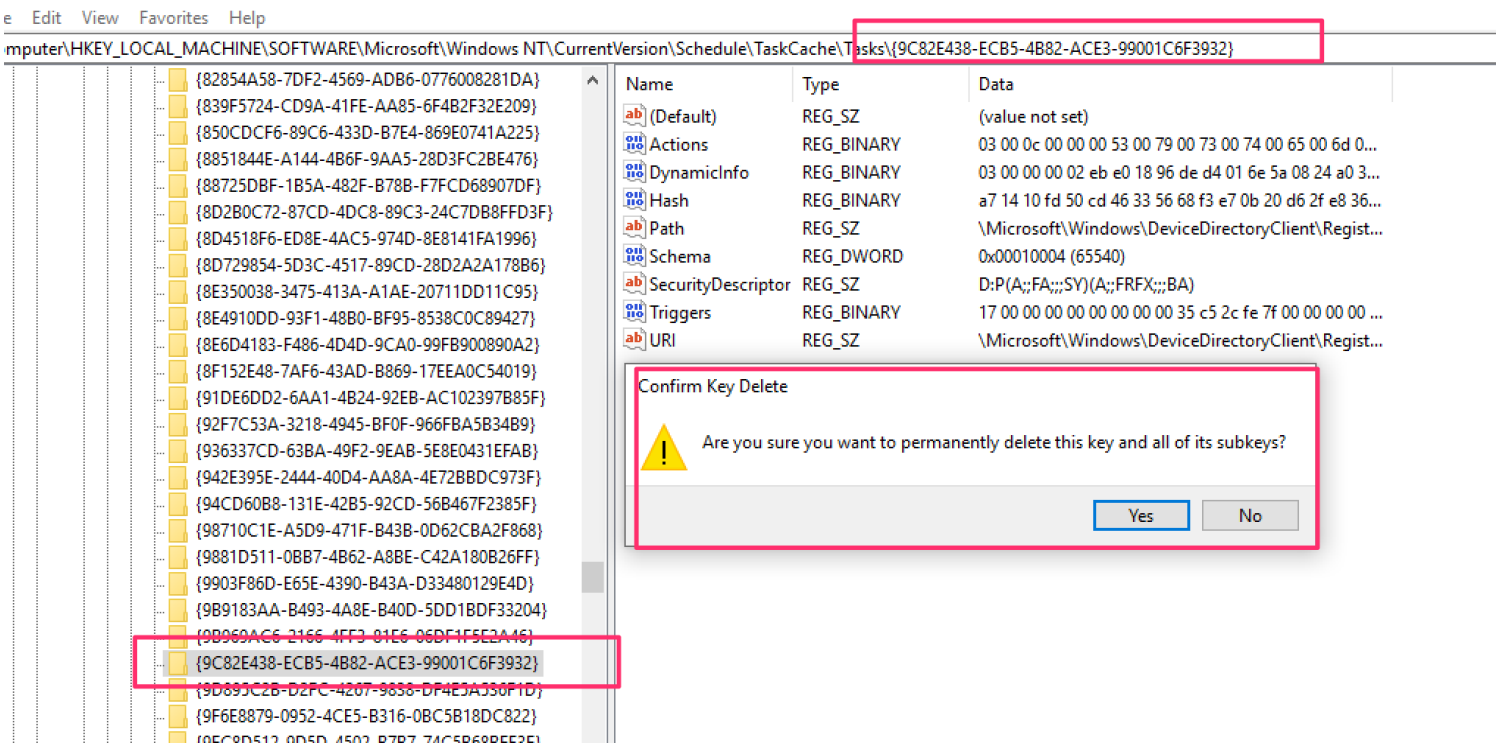
```
# Then for the ID of interest under \Taskcache\Tree subkey
  # Example: $ID = "{8E350038-3475-413A-A1AE-20711DD11C95}" ;
$ID = "{XYZ}" ;
get-itemproperty -path "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedu
? Id -Match "$ID" | fl *Name,Id,PsPath
```

```
PS C:\> $ID = "{F713D4DB-B379-4F42-A207-5C8E3E671345}";
>> get-itemproperty -path "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree\*" | ? Id -Match $ID | fl


SD            : {1, 0, 4, 128...}
Id            : {F713D4DB-B379-4F42-A207-5C8E3E671345}
Index         : 3
PSPath        : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree\Find_Me
PSParentPath  : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree
PSChildName   : Find_Me
PSDrive       : HKLM
PSProvider    : Microsoft.PowerShell.Core\Registry
```

And then eradicating these Registry schtask entries is straight forward via Regedit's GUI, that way you have no permission problems. Delete both:

- HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks\{$ID}
- HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree\$Name





Show what programs run at startup

```
Get-CimInstance Win32_StartupCommand | Select-Object Name, command, Location, Use
```

```
Get-CimInstance Win32_StartupCommand | Select-Object Name, command, Loc
```

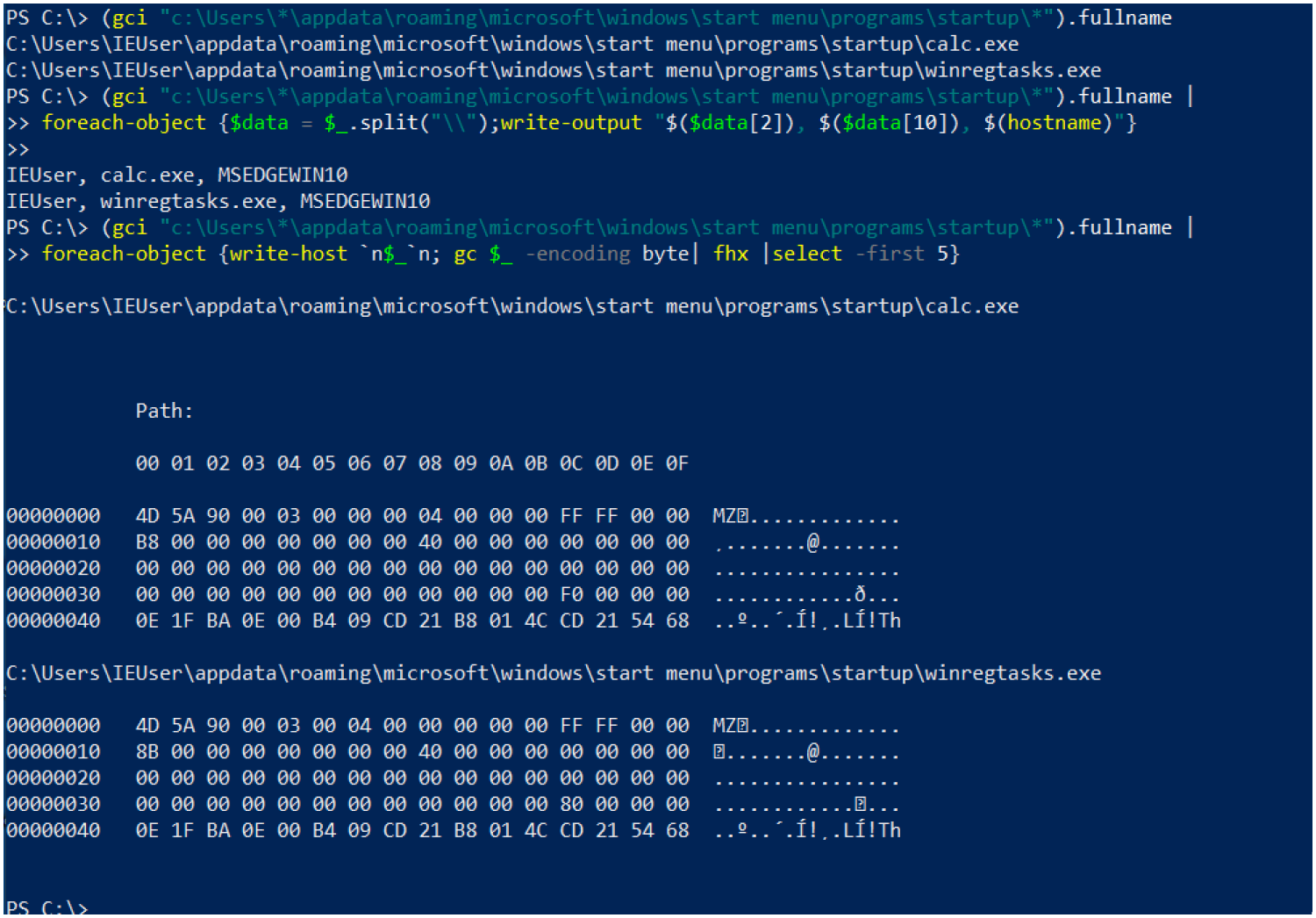
```
Name     : VMware User Process
command  : "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
Location : HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
User     : Public
```

Some direct path locations too can be checked

```
HKLM\software\classes\exefile\shell\open\command
c:\Users\*\appdata\roaming\microsoft\windows\start menu\programs\startup
```

Querying that last one in more detail, you have some interesting options

```
#Just list out the files in each user's startup folder
(gci "c:\Users\*\appdata\roaming\microsoft\windows\start menu\programs\startup\*"

#Extract from the path User, Exe, and print machine name
(gci "c:\Users\*\appdata\roaming\microsoft\windows\start menu\programs\startup\*"
foreach-object {$data = $_.split("\\");write-output "$($data[2]), $($data[10]), $

#Check the first couple lines of files' contents
(gci "c:\Users\*\appdata\roaming\microsoft\windows\start menu\programs\startup\*"
foreach-object {write-host `n$_`n; gc $_ -encoding byte| fhx |select -first 5}
```



Programs at login

Adversaries can link persistence mechanisms to be activated to a users' login via the registry `HKEY_CURRENT_USER\Environment –UserInitMprLogonScript`

```
#Create HKU drive
mount –PSProvider Registry –Name HKU –Root HKEY_USERS

#list all user's enviros
(gp "HKU:\*\Environment").UserInitMprLogonScript

#Collect SID of target user with related logon task
gp "HKU:\*\Environment" | FL PSParentPath,UserInitMprLogonScript

# insert SID and convert it into username
gwmi win32_useraccount |
select Name, SID |
? SID –match "" #insert SID between quotes
```

```
FLARE 09/06/2022 12:54:38
PS C:\Users\Frank\Desktop > (gp "HKU:\*\Environment").UserInitMprLogonScript
>>
C:\Windows\System32\calc.exe
FLARE 09/06/2022 12:54:40
PS C:\Users\Frank\Desktop > gp "HKU:\*\Environment" | FL PSParentPath,UserInitMprLogonScript
>>


PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\.DEFAULT

PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-19

PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-20

PSParentPath           Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001
UserInitMprLogonScript C:\Windows\System32\calc.exe

PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-18



FLARE 09/06/2022 12:54:42
PS C:\Users\Frank\Desktop > gwmi win32_useraccount |
>> select Name, SID |
>> ? SID -match "S-1-5-21-4090064055-3786174766-129191325-1001"

Name  SID
----
Frank S-1-5-21-4090064055-3786174766-129191325-1001
```

You can remove this regsistry entry

```
#confirm via `whatif` flag that this is the right key
remove-itemproperty "HKU:\SID-\Environment\" -name "UserInitMprLogonScript" -what
#delete it
remove-itemproperty "HKU:\SID-\Environment\" -name "UserInitMprLogonScript" -verb
```

```
FLARE 09/06/2022 12:54:50
PS C:\Users\Frank\Desktop > remove-itemproperty "HKU:\S-1-5-21-4090064055-3786174766-129191325-1001\Environment\" -name "UserInitMprLogonScript" -whatif
What if: Performing the operation "Remove Property" on target "Item: HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001\Environment\ Property: UserInitMprLogonScript".
FLARE 09/06/2022 12:57:45
PS C:\Users\Frank\Desktop > remove-itemproperty "HKU:\S-1-5-21-4090064055-3786174766-129191325-1001\Environment\" -name "UserInitMprLogonScript" -verbose
VERBOSE: Performing the operation "Remove Property" on target "Item: HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001\Environment\ Property: UserInitMprLogonScript".
FLARE 09/06/2022 12:57:57
PS C:\Users\Frank\Desktop > gp "HKU:\*\Environment" | FL PSParentPath,UserInitMprLogonScript
>>


PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\.DEFAULT

PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-19

PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-20

PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001

PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-18
```

Programs at Powershell

Adversaries can link their persistence mechanisms to a PowerShell profile, executing their malice every time you start PowerShell

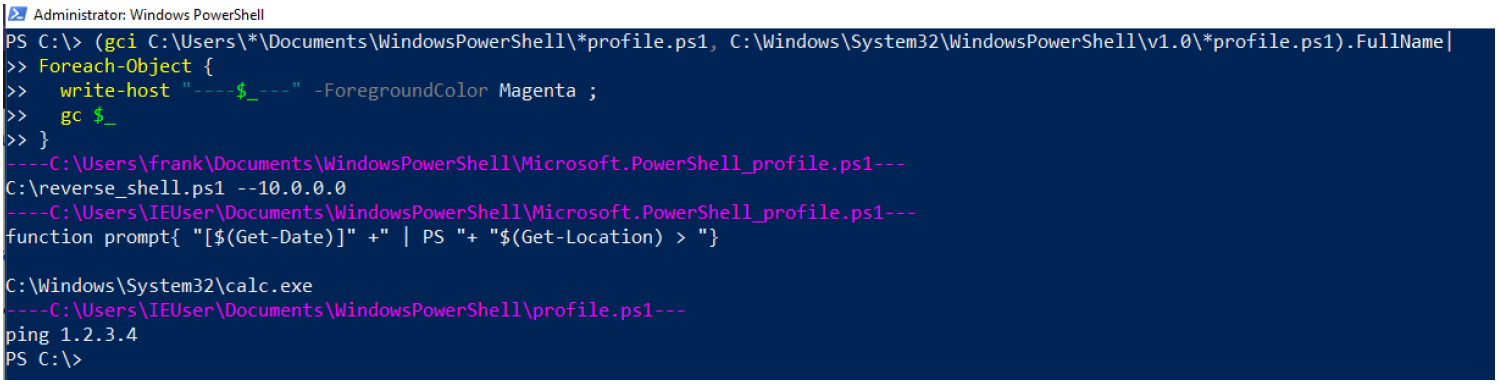
```
#confirm the profile you are querying
echo $Profile
#show PowerShell profile contents
type $Profile
```

```
[01/11/2022 09:33:03] | PS C:\ > echo "calc.exe" > $PROFILE
[01/11/2022 09:33:16] | PS C:\ > echo $Profile
C:\Users\IEUser\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
[01/11/2022 09:33:23] | PS C:\ > type $Profile
calc.exe
[01/11/2022 09:33:27] | PS C:\ > _
```

To fix this one, I'd just edit the profile and remove the persistence (so `notepad $Profile` will be just fine)

You can get a bit more clever with this if you want

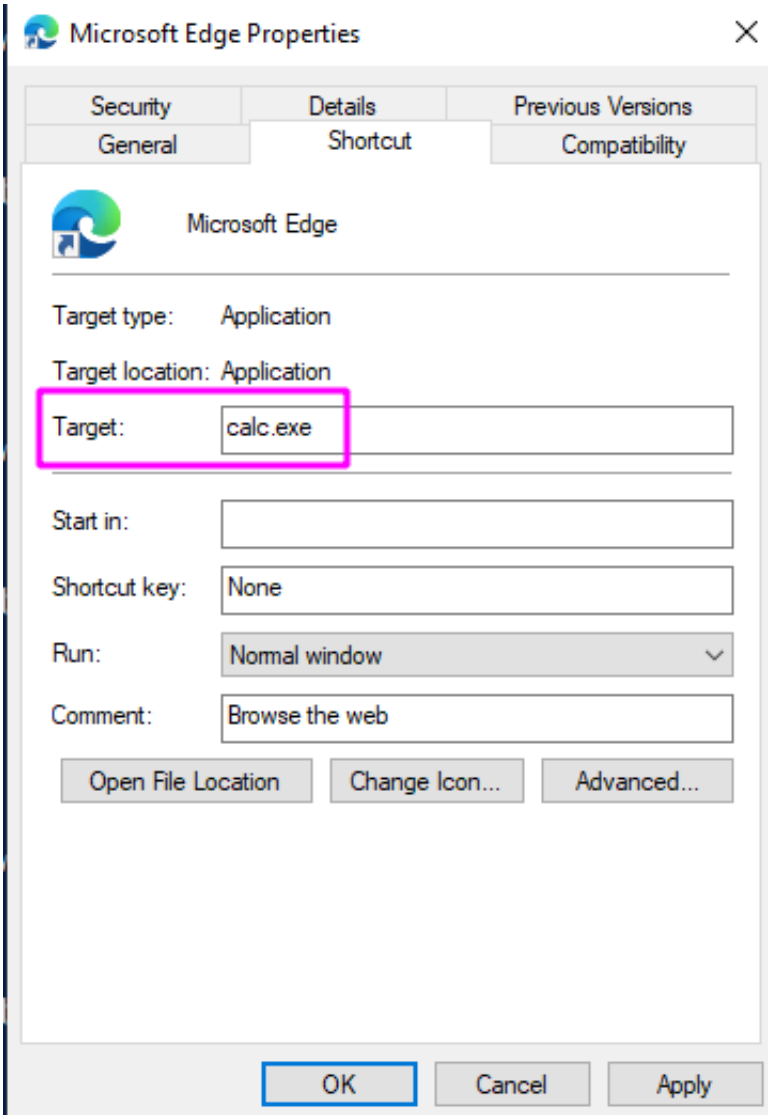
```
(gci C:\Users\*\Documents\WindowsPowerShell\*profile.ps1, C:\Windows\System32\Win
Foreach-Object {
  write-host "----$_---" -ForegroundColor Magenta ;
  gc $_ # | select-string -notmatch function ## if you want to grep out stuff you
}
```



Stolen Links

Adversaries can insert their malice into shortcuts. They can do it in clever ways, so that the application will still run but at the same time their malice will also execute when you click on the application

For demo purposes, below we have Microsoft Edge that has been hijacked to execute calc on execution.



We can specifically query all Microsoft Edge's shortcuts to find this

```
Get-CimInstance Win32_ShortcutFile |
? FileName -match 'edge' |
fl FileName,Name,Target, LastModified
```

```
[01/11/2022 09:55:24] | PS C:\ > Get-CimInstance Win32_ShortcutFile | ? FileName -match 'edge' | fl FileName,Name,Target, LastModified


FileName     : Microsoft Edge
Name         : C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Microsoft Edge.lnk
Target       : C:\Windows\System32\calc.exe
LastModified : 1/11/2022 9:56:09 AM
```

This doesn't scale however, as you will not know the specific shortcut that the adversary has manipulated. So instead, sort by the `LastModified` date

```
Get-CimInstance Win32_ShortcutFile |
sort LastModified -desc |
fl FileName,Name,Target, LastModified
```

```
[01/11/2022 09:58:47] | PS C:\ > Get-CimInstance Win32_ShortcutFile | sort LastModified -desc | fl FileName,Name,Target, LastModif


FileName     : Microsoft Edge
Name         : C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Microsoft Edge.lnk
Target       : C:\Windows\System32\calc.exe
LastModified : 1/11/2022 9:56:09 AM
```

Hunt LNKs at scale

This above will output a LOT, however. You may want to only show results for anything LastModified after a certain date. Lets ask to only see things modified in the year 2022 onwards

```
Get-CimInstance Win32_ShortcutFile |
where-object {$_.lastmodified -gt [datetime]::parse("01/01/2022")} |
sort LastModified -desc | fl FileName,Name,Target, LastModified
```

```
PS C:\> Get-CimInstance Win32_ShortcutFile |
>> where-object {$_.lastmodified -gt [datetime]::parse("01/01/2022")} | fl FileName,Name,Target, LastModified


FileName     : Microsoft Edge
Name         : C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Microsoft Edge.lnk
Target       : C:\Windows\System32\calc.exe
LastModified : 1/11/2022 9:56:09 AM

FileName     : Microsoft Edge
Name         : C:\Users\IEUser\AppData\Roaming\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar\Microsoft Edge.lnk
Target       : C:\Windows\System32\calc.exe
LastModified : 1/11/2022 9:45:55 AM

FileName     : onedrive
Name         : C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\OneDrive.lnk
Target       : C:\evil.exe
LastModified : 1/11/2022 10:04:55 AM
```

Scheduled Jobs

Surprisingly, not many people know about Scheduled Jobs. They're not anything too strange or different, they're just scheduled tasks that are specificially powershell.

I've written about a real life encounter I had during an incident, where the adversary had leveraged a PowerShell scheduled job to execute their malice at an oppertune time

Find out what scheduled jobs are on the machine

```
Get-ScheduledJob
# pipe to | fl * for greater granularity
```

```
[06/02/2021 23:27:09] | PS C:\Users\IEUser\Desktop > Get-ScheduledJob

Id        Name            JobTriggers     Command                                  Enabled
--        ----            -----------     -------                                  -------
1         GPS             1               GPS                                      True
2         EVIL            1               &evilshell.exe                           True
```

Get detail behind scheduled jobs

```
Get-ScheduledJob | Get-JobTrigger |
Ft -Property @{Label="ScheduledJob";Expression={$_.JobDefinition.Name}},ID,Enable
#pipe to fl or ft, whatever you like the look of more in the screenshot
```

```
ScheduledJob : GPS
Id           : 1
Enabled      : True
At           : 6/2/2021 1:45:00 PM
Frequency    : Once
DaysOfWeek   :

ScheduledJob : EVIL
Id           : 1
Enabled      : True
At           : 6/2/2021 1:45:00 PM
Frequency    : Once
DaysOfWeek   :



[06/02/2021 23:34:36] | PS C:\Users\IEUser\Desktop > Get-ScheduledJo
ob";Expression={$_.JobDefinition.Name}},ID,Enabled, At, frequency, Da

ScheduledJob Id Enabled At                   Frequency DaysOfWeek
------------ -- ------- --                   --------- ----------
GPS           1    True 6/2/2021 1:45:00 PM      Once
EVIL          1    True 6/2/2021 1:45:00 PM      Once
```

Kill job

The following all work.

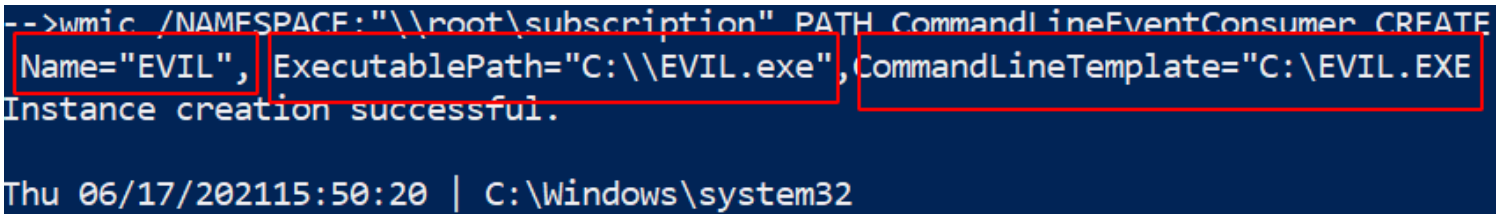
```
Disable-ScheduledJob -Name evil_sched
```

```
Unregister-ScheduledJob -Name eviler_sched
Remove-Job -id 3
#then double check it's gone with Get-ScheduledJob

#if persists, tack on to unregister or remove-job
-Force -Confirm:$false -verbose
```

Hunt WMI Persistence

WMIC can do some pretty evil things 1 & 2. One sneaky, pro-gamer move it can pull is *persistence*

In the image below I have included a part of setting up WMI persistence



Finding it

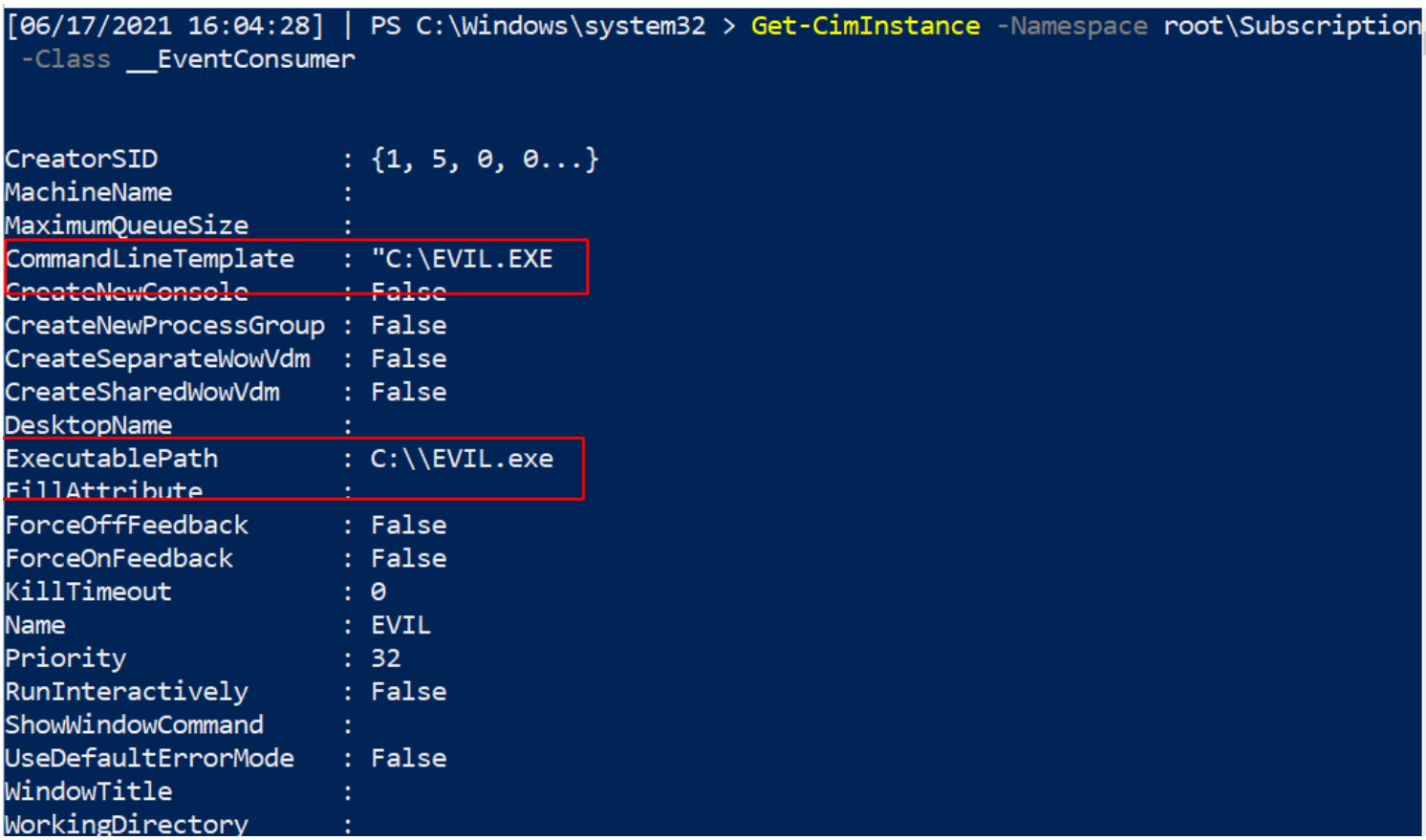
Now, our task is to find this persistent evil.

Get-CimInstance comes out cleaner, but you can always rely on the alternate Get-WMIObject

```
Get-CimInstance -Namespace root\Subscription -Class __FilterToConsumerBinding
Get-CimInstance -Namespace root\Subscription -Class __EventFilter
Get-CimInstance -Namespace root\Subscription -Class __EventConsumer

## OR

Get-WMIObject -Namespace root\Subscription -Class __EventFilter
Get-WMIObject -Namespace root\Subscription -Class __FilterToConsumerBinding
Get-WMIObject -Namespace root\Subscription -Class __EventConsumer
```

```
[06/17/2021 16:01:50] | PS C:\Windows\system32 > Get-CimInstance -Namespace root\Subscription
 -Class __FilterToConsumerBinding


Consumer                : CommandLineEventConsumer (Name = "EVIL")
CreatorSID              : {1, 5, 0, 0...}
DeliverSynchronously    : False
DeliveryQoS             :
Filter                  : __EventFilter (Name = "EVIL")
MaintainSecurityContext : False
SlowDownProviders       : False
PSComputerName          :
```

```
[06/17/2021 16:03:42] | PS C:\Windows\system32 > Get-CimInstance -Namespace root\Subscriptio
 -Class __EventFilter


CreatorSID     : {1, 2, 0, 0...}
EventAccess    :
EventNamespace : root\cimv2
Name           : SCM Event Log Filter
Query          : select * from MSFT_SCMEventLogEvent
QueryLanguage  : WQL
PSComputerName :

CreatorSID     : {1, 5, 0, 0...}
EventAccess    :
EventNamespace : root\cimv2
Name           : EVIL
Query          : SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance
                 ISA 'Win32_PerfFormattedData_PerfOS_System'
QueryLanguage  : WQL
PSComputerName :
```



Removing it

Now we've identified the evil WMI persistence, let us be rid of it!

We can specify the Name as `EVIL` as that's what it was called across the three services. Whatever your persistence calls itself, change the name for that

```
#notice this time, we use the abbrevated version of CIM and WMI

gcim -Namespace root\Subscription -Class __EventFilter |
? Name -eq "EVIL" | Remove-CimInstance -verbose

gcim -Namespace root\Subscription -Class __EventConsumer|
? Name -eq "EVIL" | Remove-CimInstance -verbose

#it's actually easier to use gwmi here instead of gcim
gwmi -Namespace root\Subscription -Class __FilterToConsumerBinding |
? Consumer -match "EVIL" | Remove-WmiObject -verbose
```

```
[06/17/2021 16:22:40] | PS C:\Windows\system32 > gcim -Namespace root\Subscription -Class
_EventFilter | ? Name -eq "EVIL" | Remove-CimInstance -verbose
VERBOSE: Performing the operation "Remove-CimInstance" on target "__EventFilter (Name =
"EVIL")".
VERBOSE: Perform operation 'Delete CimInstance' with following parameters,
''namespaceName' = ROOT/Subscription,'instance' = __EventFilter (Name = "EVIL")'.
VERBOSE: Operation 'Delete CimInstance' complete.
[06/17/2021 16:22:58] | PS C:\Windows\system32 >
```

A note on CIM

You may see WMI and CIM talked about together, whether on the internet or on in the Blue Team Notes here.

CIM is a standard for language for vendor-side management of a lot of the physical and digital mechanics of what makes a computer tick. WMIC was and is Microsoft's interpretation of CIM.

However, Microsoft is going to decommision WMIC soon. So using `Get-Ciminstance` versions rather than `get-wmiobject` is probably better for us to learn in the long term. I dunno man, It's complicated.

Run Keys

What are Run Keys

I've written in depth about run keys, elsewhere

Run and RunOnce registry entries will run tasks on startup. Specifically:

- Run reg keys will run the task every time there's a login.
- RunOnce reg kgeys will run the taks once and then self-delete keys.
  - If a RunOnce key has a name with an exclemation mark (!likethis) then it will self-delete
  - IF a RunOnce key has a name with an asterik (* LikeDIS) then it can run even in Safe Mode.

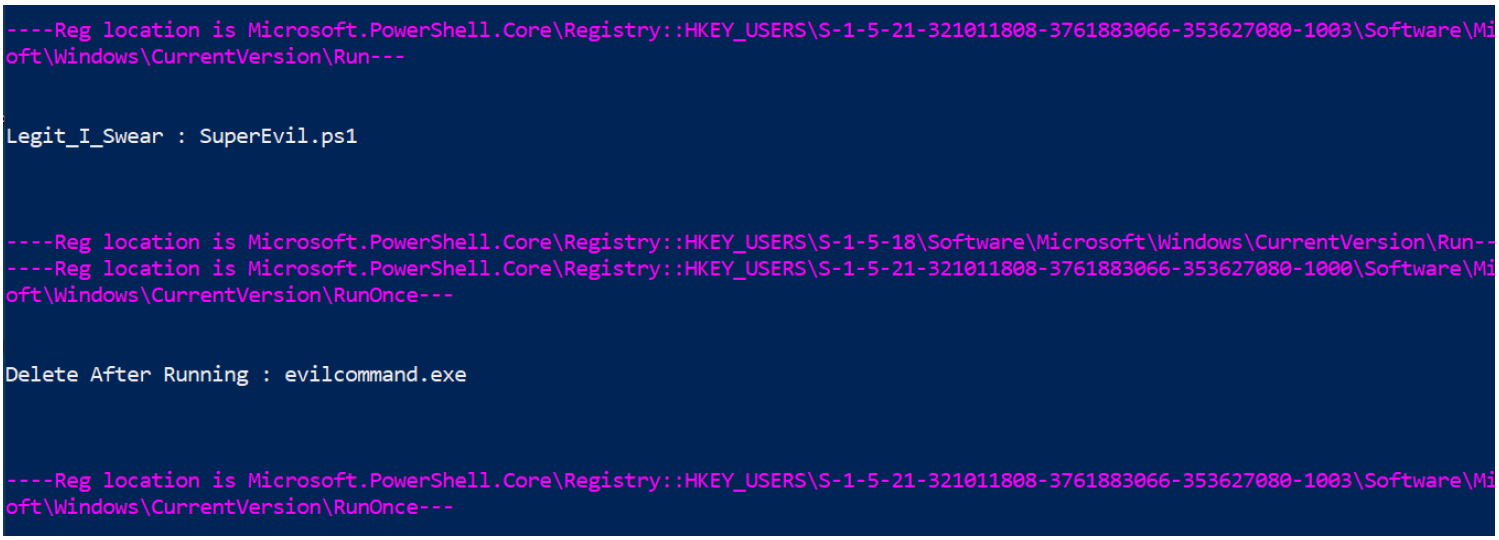
If you look in the reg, you'll find some normal executables.

```
[07/02/2021 20:37:56] PS> get-itemproperty -path "HKLM:\Software\Microsoft\Windows\Current
Version\Run" | select -property * -exclude PS* | fl


SecurityHealth      : C:\Windows\system32\SecurityHealthSystray.exe
bginfo              : C:\BGinfo\Bginfo.exe /accepteula /ic:\bginfo\bgconfig.bgi /timer:0
VMware User Process : "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
```

Finding Run Evil

A quick pwsh *for loop* can collect the contents of the four registry locations.

```
#Create HKU drive
mount -PSProvider Registry -Name HKU -Root HKEY_USERS

(gci HKLM:\Software\Microsoft\Windows\CurrentVersion\Run, HKLM:\Software\Microsof
Foreach-Object {
  write-host "----Reg location is $_---" -ForegroundColor Magenta ;
  gp $_ |
  select -property * -exclude PS*, One*, vm* | #exclude results here
  FL
}

#you can squish that all in one line if you need to
(gci HKLM:\Software\Microsoft\Windows\CurrentVersion\Run, HKLM:\Software\Microsof
```



You can also achieve the same thing with these two alternative commands, but it isn't as cool as the above for loop

```
get-itemproperty "HKU:\*\Software\Microsoft\Windows\CurrentVersion\Run*" |
  select -property * -exclude PSPR*,PSD*,PSC*,PSPAR*  | fl
get-itemproperty "HKLM:\Software\Microsoft\Windows\CurrentVersion\Run*" |
  select -property * -exclude PSPR*,PSD*,PSC*,PSPAR*  | fl
```

```
>> get-itemproperty "HKLM:\Software\Microsoft\Windows\CurrentVersion\Run*" |
>>   select -property * -exclude PSPR*,PSD*,PSC*,PSPAR*  | fl


OneDriveSetup : C:\Windows\SysWOW64\OneDriveSetup.exe /thfirstsetup
PSPath        : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-19\Software\Microsoft\Windows\CurrentVersion\Run

OneDriveSetup : C:\Windows\SysWOW64\OneDriveSetup.exe /thfirstsetup
PSPath        : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-20\Software\Microsoft\Windows\CurrentVersion\Run

OneDrive : "C:\Users\IEUser\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
PSPath   : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-321011808-3761883066-353627080-1000\Software\Microsoft\Wind
           ows\CurrentVersion\Run

Delete_After_Running : evilcommand.exe
PSPath               : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-321011808-3761883066-353627080-1000\Software\Mi
                       crosoft\Windows\CurrentVersion\RunOnce

OneDrive      : "C:\Users\toby\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
Legit_I_Swear : SuperEvil.ps1
PSPath        : Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-321011808-3761883066-353627080-1003\Software\Microsoft
                \Windows\CurrentVersion\Run
```

Removing Run evil

Be surgical here. You don't want to remove Run entries that are legitimate. It's important you remove with -verbose too and double-check it has gone, to make sure you have removed what you think you have.

Specify the SID

```
#Create HKU drive
mount -PSProvider Registry -Name HKU -Root HKEY_USERS

#List the malicious reg by path
get-itemproperty "HKU:\SID\Software\Microsoft\Windows\CurrentVersion\RunOnce" | s

#Then pick the EXACT name of the Run entry you want to remove. Copy paste it, inc
Remove-ItemProperty -Path "HKU:\SID-\Software\Microsoft\Windows\CurrentVersion\Ru

#Then check again to be sure it's gone
get-itemproperty "HKU:\*\Software\Microsoft\Windows\CurrentVersion\RunOnce" | sel
```

```
[07/02/2021 21:56:46] PS> get-itemproperty "HKCU:\Software\Microsoft\Windows\CurrentVersion\RunOnce" | select -property * -exclude PS* | f
l


!EvilRunOnce  : c:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe -noexit -command 'EVILCOMMAND.exe'
*EvilerRunOnce : c:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe -noexit -command 'EVILERCOMMAND.exe'



[07/02/2021 21:56:50] PS> Remove-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\RunOnce" -Name "*EvilerRunOnce" -verb
ose
VERBOSE: Performing the operation "Remove Property" on target "Item: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
Property: *EvilerRunOnce".
[07/02/2021 21:57:02] PS> Remove-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\RunOnce" -Name "!EvilRunOnce" -verbos
e
VERBOSE: Performing the operation "Remove Property" on target "Item: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
Property: !EvilRunOnce".
[07/02/2021 21:57:17] PS> get-itemproperty "HKCU:\Software\Microsoft\Windows\CurrentVersion\RunOnce" | select -property * -exclude PS* | f
l
[07/02/2021 21:57:20] PS>
```

Other Malicious Run Locations

Some *folders* can be the locations of persistence.

```
#Create HKU drive
mount -PSProvider Registry -Name HKU -Root HKEY_USERS

$folders = @("HKU:\*\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shel
foreach ($folder in $folders) {
  write-host "----Reg key is $folder--- -ForegroundColor Magenta ";
  get-itemproperty -path "$folder"  |
  select -property * -exclude PS* | fl
}
```

```
[07/02/2021 21:49:43] PS> foreach ($folder in $folders) {
>> write-host "----Reg key is $folder---";
>> get-itemproperty -path "$folder"  |
>> select -property * -exclude PS* | fl
>> }
----Reg key is HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders---


AppData                                 : C:\Users\IEUser\AppData\Roaming
Cache                                   : C:\Users\IEUser\AppData\Local\Microsoft\Windows\INetCache
Cookies                                 : C:\Users\IEUser\AppData\Local\Microsoft\Windows\INetCookies
Desktop                                 : C:\Users\IEUser\Desktop
Favorites                               : C:\Users\IEUser\Favorites
History                                 : C:\Users\IEUser\AppData\Local\Microsoft\Windows\History
Local AppData                           : C:\Users\IEUser\AppData\Local
My Music                                : C:\Users\IEUser\Music
My Pictures                             : C:\Users\IEUser\Pictures
My Video                                : C:\Users\IEUser\Videos
NetHood                                 : C:\Users\IEUser\AppData\Roaming\Microsoft\Windows\Network Shortcuts
Personal                                : C:\Users\IEUser\Documents
```

Svchost startup persistence

```
get-itemproperty -path "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchos
```

```
[07/02/2021 21:50:32] PS> get-itemproperty -path "HKLM:\SOFTWARE\Microsoft\Windows NT\Curre


DcomLaunch                          : {Power, LSM, BrokerInfrastructure, PlugPlay...}
defragsvc                           : {defragsvc}
LocalServiceNetworkRestricted       : {TimeBrokerSvc, WarpJITSvc, eventlog, AudioSrv...}
rdxgroup                            : {RetailDemo}
RPCSS                               : {RpcEptMapper, RpcSs}
sdrsvc                              : {sdrsvc}
utcsvc                              : {DiagTrack}
WepHostSvcGroup                     : {WepHostSvc}
Camera                              : {FrameServer}
LocalService                        : {nsi, WdiServiceHost, w32time, EventSystem...}
LocalServiceNoNetworkFirewall       : {BFE, mpssvc}
NetworkServiceAndNoImpersonation    : {KtmRm}
diagnostics                         : {DiagSvc}
AxInstSVGroup                       : {AxInstSV}
smphost                             : {smphost}
PrintWorkflow                       : {PrintWorkflowUserSvc}
```

Winlogon startup persistence

```
#Create HKU drive
mount -PSProvider Registry -Name HKU -Root HKEY_USERS

(gci "HKU:\*\Software\Microsoft\Windows NT\CurrentVersion\Winlogon").PSPath |
Foreach-Object {
  write-host "----Reg location is $_---" -ForegroundColor Magenta ;
  gp $_ |
  select -property * -exclude PS* |
  FL
}
```

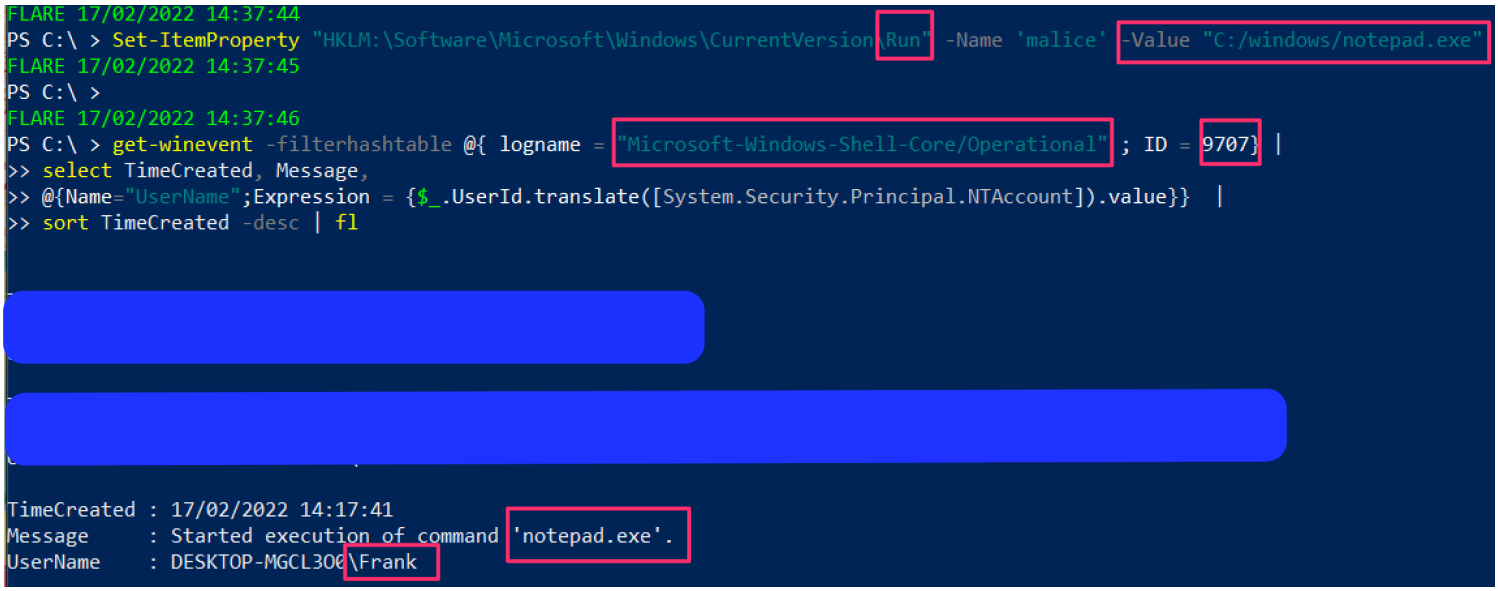
```
----Reg location is Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-321011808-3761883066-353627080-1000\Software\Microsoft\Windows NT\CurrentVersion\Winlogon---


ExcludeProfileDirs : AppData\Local;AppData\LocalLow;$Recycle.Bin;OneDrive;Work Folders
BuildNumber        : 17763
FirstLogon         : 0
PUUActive          : {91, 152, 205, 63...}
DP                 : {210, 0, 232, 0...}
ParseAutoexec      : 1



----Reg location is Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-321011808-3761883066-353627080-1003\Software\Microsoft\Windows NT\CurrentVersion\Winlogon---


ExcludeProfileDirs : AppData\Local;AppData\LocalLow;$Recycle.Bin;OneDrive;Work Folders
BuildNumber        : 17763
FirstLogon         : 0
ParseAutoexec      : 1
PUUActive          : {91, 152, 205, 63...}
DP                 : {210, 0, 232, 0...}
```

Find more examples of Run key evil from Mitre ATT&CK

Evidence of Run Key Execution

You can query the 'Microsoft-Windows-Shell-Core/Operational' log to find evidence if a registry run key was successful in executing.

```
get-winevent -filterhashtable @{ logname = "Microsoft-Windows-Shell-Core/Operatio
select TimeCreated, Message,
@{Name="UserName";Expression = {$_.UserId.translate([System.Security.Principal.NT
sort TimeCreated -desc| fl
```



Screensaver Persistence

It can be done, I swear. Mitre ATT&CK has instances of .SCR's being used to maintain regular persistence

```
#Create HKU drive
mount -PSProvider Registry -Name HKU -Root HKEY_USERS

gp "HKU:\*\Control Panel\Desktop\" | select SCR* | fl
# you can then go and collect the .scr listed in the full path, and reverse engin

#you can also collect wallpaper info from here
gp "HKU:\*\Control Panel\Desktop\" | select wall* | fl
```

```
[07/02/2021 22:21:51] PS> gp "HKCU:\Control Panel\Desktop\" | select SCR* | fl


ScreenSaveActive    : 1
SCRNSAVE.EXE        : c:\windows\system32\TotallyLegit.scr
ScreenSaveTimeOut   : 420
ScreenSaverIsSecure : 1
```

Query Group Policy

The group policy in an Windows can be leveraged and weaponised to propogate malware and even ransomware across the entire domain

You can query the changes made in the last X days with this line

```
#collects the domain name as a variable to use later
$domain = (Get-WmiObject -Class win32_computersystem).domain;
Get-GPO -All -Domain $domain |
?{ ([datetime]::today - ($_.ModificationTime)).Days -le 10 } | sort
# Change the digit after -le to the number of days you want to go back for
```

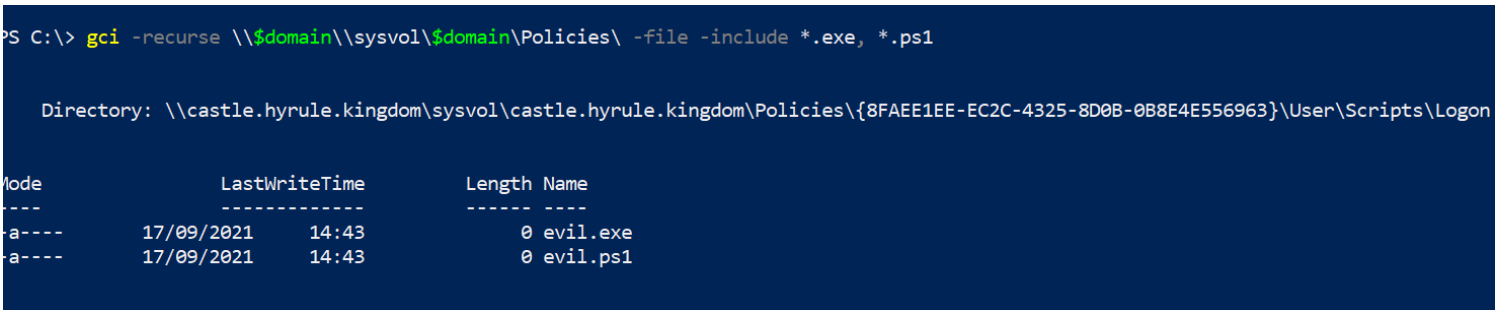
```
PS C:\> $domain = (Get-WmiObject -Class win32_computersystem).domain;
>> Get-GPO -All -Domain $domain |
>> ?{ ([datetime]::today - ($_.ModificationTime)).Days -le 10 } | sort


DisplayName      : EvilLogon
DomainName       : castle.hyrule.kingdom
Owner            : CASTLE\Domain Admins
Id               : 8faee1ee-ec2c-4325-8d0b-0b8e4e556963
GpoStatus        : AllSettingsEnabled
Description      :
CreationTime     : 17/09/2021 14:41:24
ModificationTime : 17/09/2021 14:41:24
UserVersion      : AD Version: 0, SysVol Version: 0
ComputerVersion  : AD Version: 0, SysVol Version: 0
WmiFilter        :

DisplayName      : Default Domain Controllers Policy
DomainName       : castle.hyrule.kingdom
```

Query GPO Scripts

We can hunt down the strange thinngs we might see in our above query

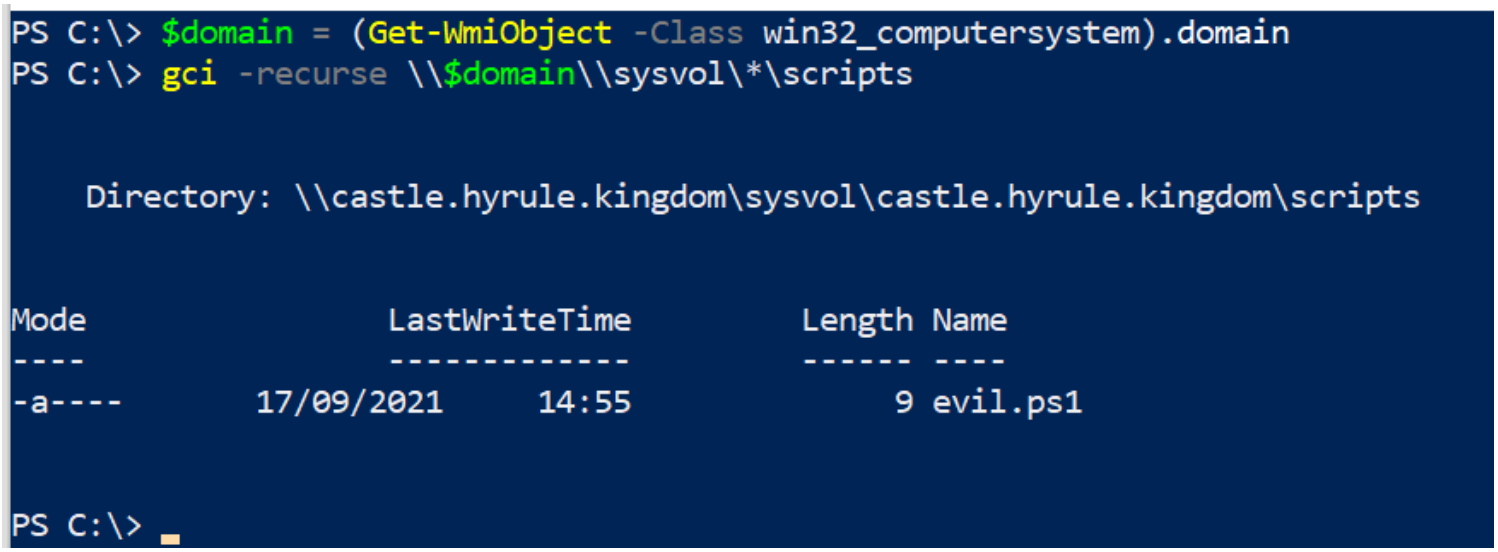
We can list all of the policies, and see where a policy contains a script or executable. You can change the `include` at the end to whatever you want

```
$domain = (Get-WmiObject -Class win32_computersystem).domain;
gci -recurse \\$domain\\sysvol\$domain\Policies\ -file -include *.exe, *.ps1
```



We can hunt down where GPO scripts live

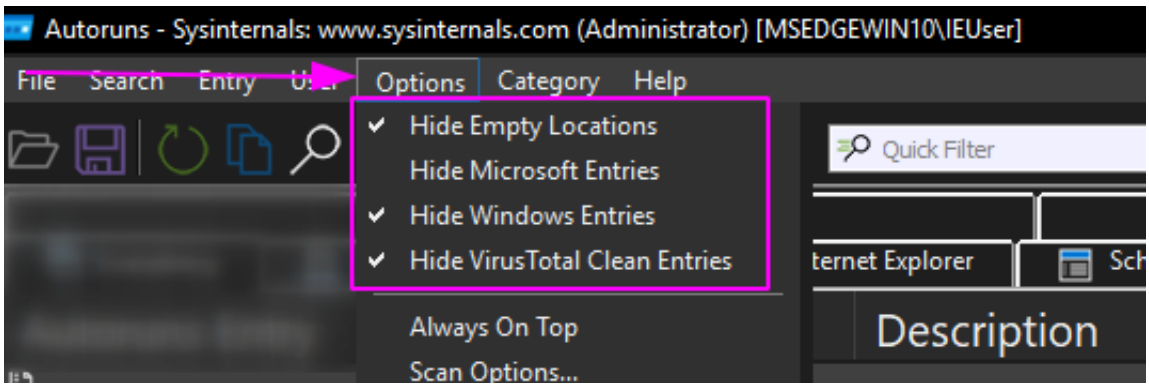
```
$domain = (Get-WmiObject -Class win32_computersystem).domain;
gci -recurse \\$domain\\sysvol\*\scripts
```



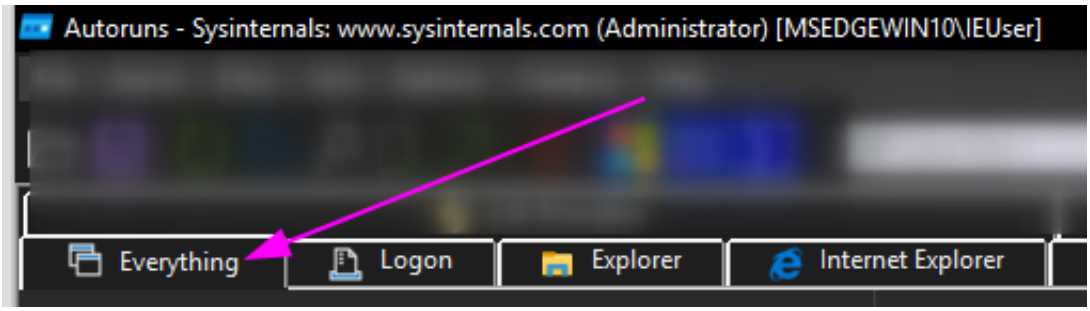
Autoruns

Autoruns is a Sysinternals tool for Windows. It offers analysts a GUI method to examine the recurring tasks that an adversary might use for persistence and other scheduled malice.

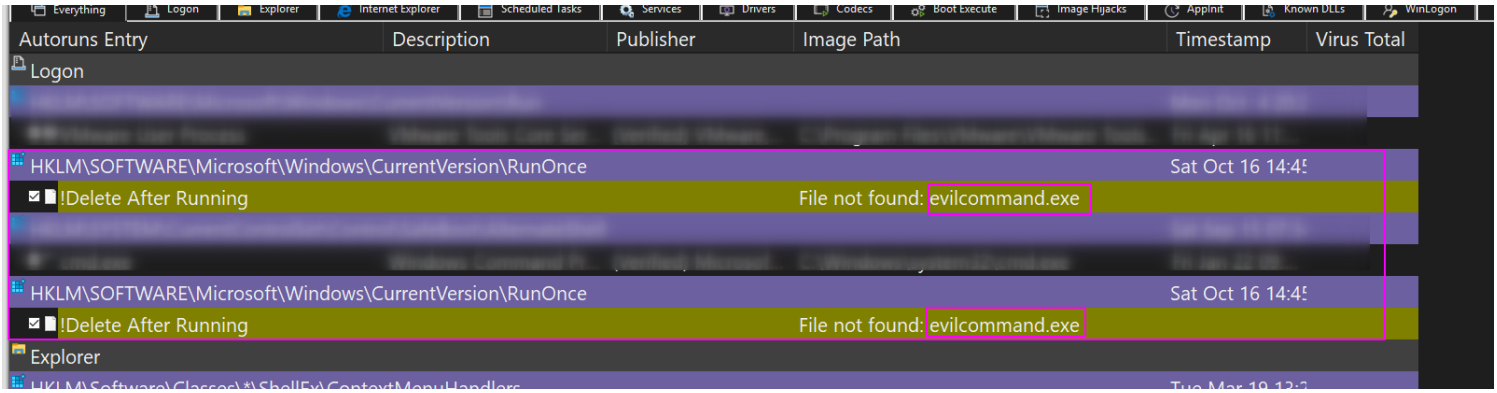
Before you go anywhere cowboy, make sure you've filtered out the known-goods under options. It makes analysis a bit easier, as you're filtering out noise. Don't treat this as gospel though, so yes hide the things that VirusTotal and Microsoft SAY are okay.....but go and verify that those auto-running tasks ARE as legitimate as they suppose they are



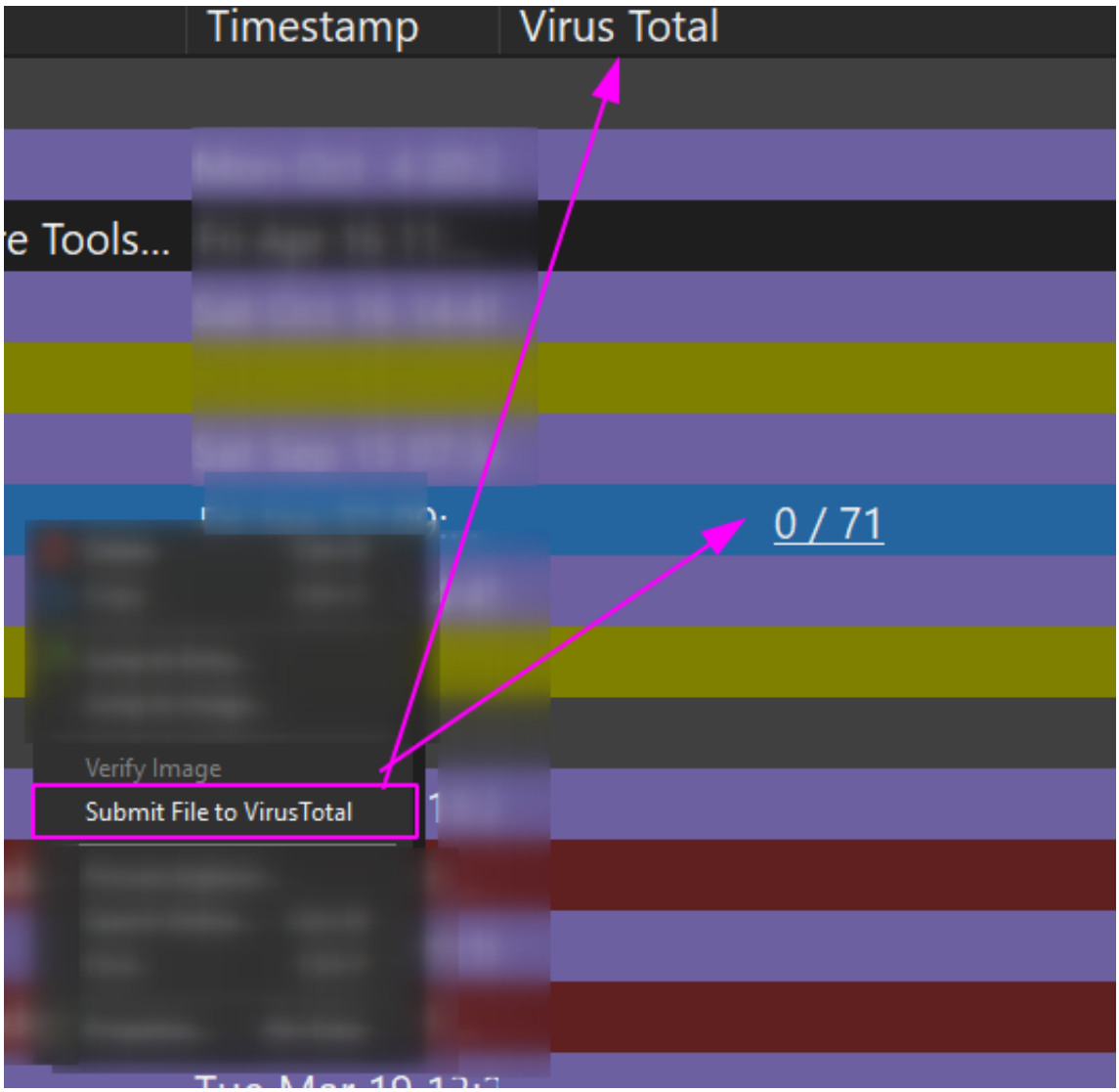
I personally just stick to the 'Everything' folder, as I like to have full visibility rather than go into the options one by one



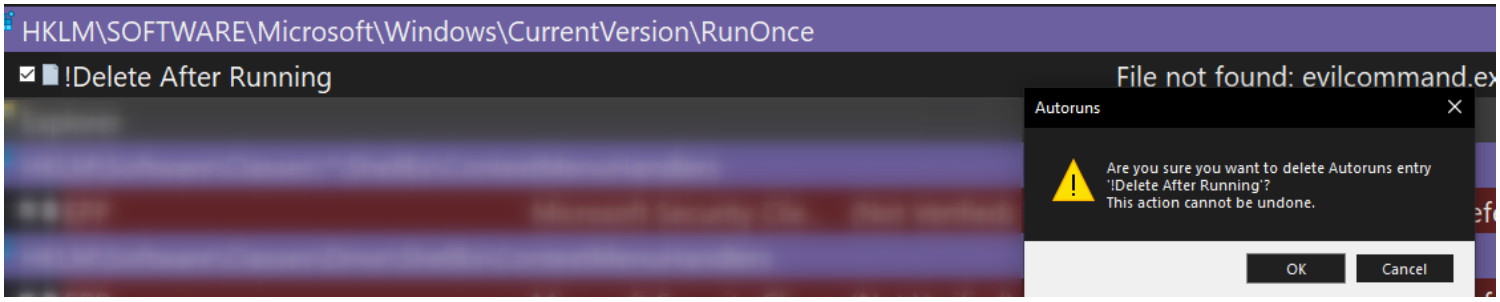
Some things in autorun may immediately stick out to you as strange. Take for example the malicious run key I inserted on the VM as an example:



You can right-click and ask Virus Total to see if the hash is a known-bad



And you can right-click and ask autoruns to delete this recurring task from existence



I like autoruns for digital forensics, where you take it one machine at a time. But - in my uneducated opinion - it does not scale well. A tool like Velociraptor that allows orchestration across thousands of machines can be leveraged to query things with greater granularity than Autoruns allows.

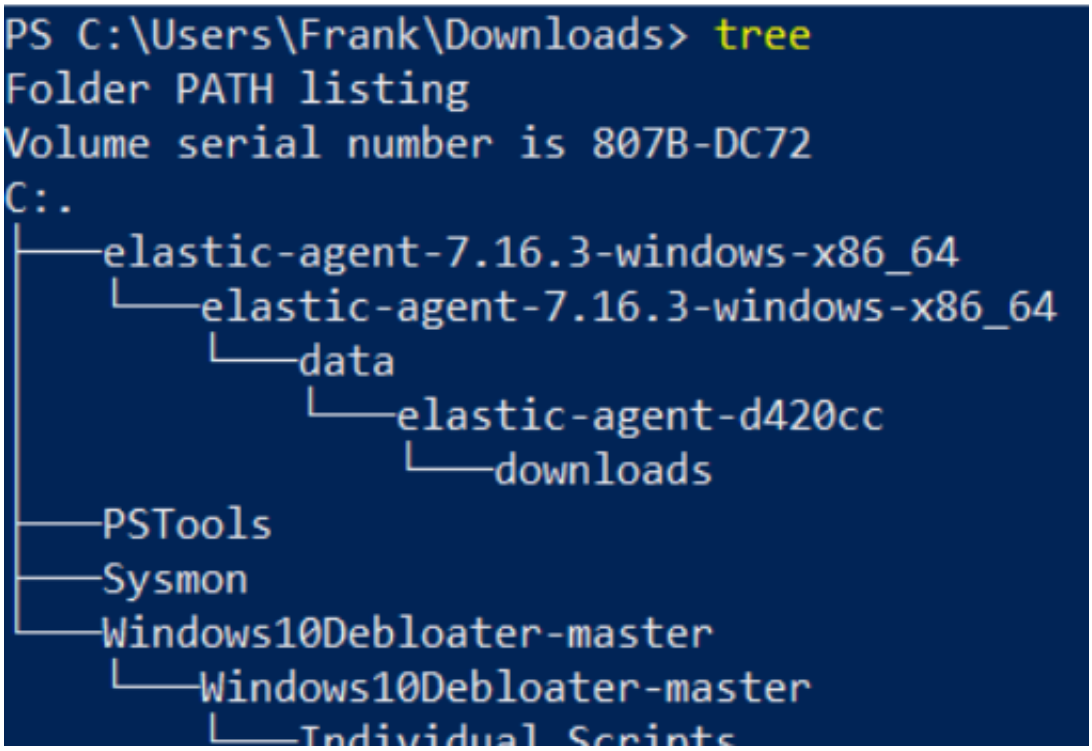
This is why I like to use PowerShell for much of my blue team work on a Windows machine, where possible. I can pre-filter my queries so I don't get distraced by noise, but moreover I can run that fine-tuned PowerShell query network-wide across thosuands of machines and recieve the results back rapidly.

File Queries

▶ section contents

File tree

Fire off `tree` to list the directories and files underneath your current working directory, nestled under each other



Wildcard paths and files

You can chuck wildcards in directories for gci, as well as wildcard to include file types.

Let's say we want to look in all of the Users \temp\ directories. We don't want to put their names in, so we wildcard it.

We also might only be interested in the pwsh scripts in their \temp, so let's filter for those only

```
gci "C:\Users\*\AppData\Local\Temp\*" -Recurse -Force -File  -Include *.ps1, *.ps
ft lastwritetime, name -autosize |
out-string -width 800
```

```
gci "C:\Users\*\AppData\Local\Temp\*" -Recurse -Force -File  -Include *.ps1, *.psm1 |
ft lastwritetime, name -autosize |
out-string -width 80
```

```
LastWriteTime       Name
-------------       ----
19/10/2016 14:40:54 NewApplication.ps1
```

Check if a specific file or path is alive.

I've found that this is a great one to quickly check for specific vulnerabilities. Take for example, CVE-2021-21551. The one below this one is an excellent way of utilising the 'true/false' binary results that test-path can give

```
test-path -path "C:\windows\temp\DBUtil_2_3.Sys"
```

False	GI SP-VM16
False	GA 095
False	CI 19
True	CI 42
True	GA 163
True	G 248

test if files and directories are present or absent

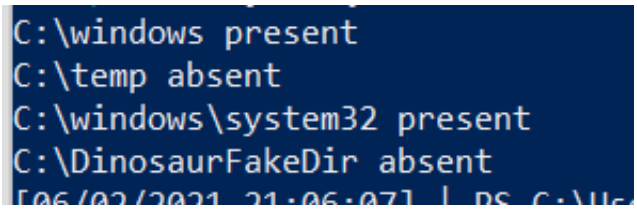
This is great to just sanity check if things exist. Great when you're trying to check if files or directories have been left behind when you're cleaning stuff up.

```
$a = Test-Path "C:\windows\sysmon.exe"; $b= Test-Path "C:\Windows\SysmonDrv.sys";
IF ($a -eq 'True') {Write-Host "C:\windows\sysmon.exe present"} ELSE {Write-Host
IF ($b -eq 'True') {Write-Host "C:\Windows\SysmonDrv.sys present"} ELSE {Write-Ho
IF ($c -eq 'True') {Write-Host "C:\Program Files (x86)\sysmon present"} ELSE {Wri
IF ($d -eq 'True') {Write-Host "C:\Program Files\sysmon present"} ELSE {Write-Hos
```

```
 C:\windows\sysmon.exe absent
C:\Windows\SysmonDrv.sys present
C:\Program Files (x86)\sysmon absent
C:\Program Files\sysmon absent
```

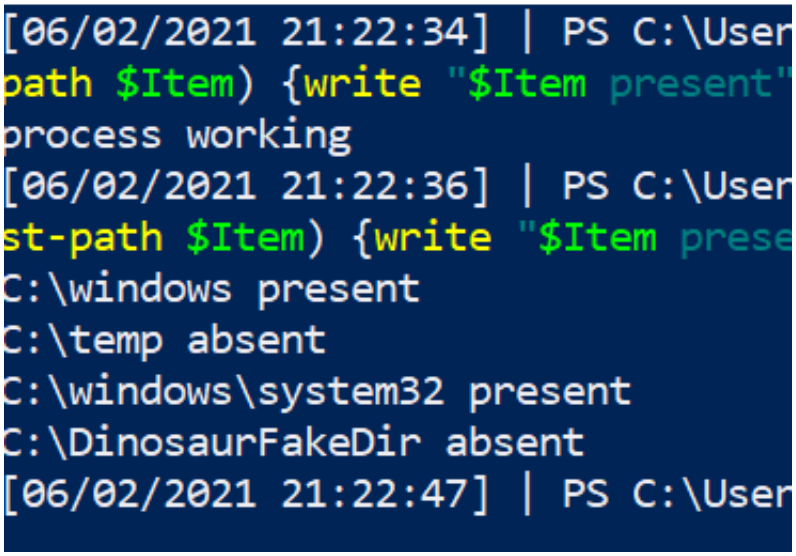
^ The above is a bit over-engineered. Here's an an abbrevated version

```
$Paths = "C:\windows" , "C:\temp", "C:\windows\system32", "C:\DinosaurFakeDir" ;
foreach ($Item in $Paths){if
(test-path $Item) {write "$Item present"}else{write "$Item absent"}}
```



We can also make this conditional. Let's say if Process MemeProcess is NOT running, we can then else it to go and check if files exist

```
$Paths = "C:\windows" , "C:\temp", "C:\windows\system32", "C:\DinosaurFakeDir" ;
if (Get-Process | where-object Processname -eq "explorer") {write "process workin
foreach ($Item in $Paths){if (test-path $Item) {write "$Item present"}else{write
```



You can use `test-path` to query Registry, but even the 2007 Microsoft docs say that this can give inconsistent results, so I wouldn't bother with test-path for reg stuff when it's during an IR

Query File Contents

Seen a file you don't recognise? Find out some more about it! Remember though: don't trust timestamps!

```
Get-item C:\Temp\Computers.csv |
select-object -property @{N='Owner';E={$_.GetAccessControl().Owner}}, *time, vers
```

```
Owner          : [redacted]Green[redacted]
CreationTime   : 08/01/2021 14:21:39
LastAccessTime : 08/01/2021 14:21:39
LastWriteTime  : 08/01/2021 14:21:58
VersionInfo    : File:             C:\Temp\Computers.csv
                 InternalName:
                 OriginalFilename:
                 FileVersion:
                 FileDescription:
                 Product:
                 ProductVersion:
                 Debug:            False
                 Patched:          False
                 PreRelease:       False
                 PrivateBuild:     False
                 SpecialBuild:     False
                 Language:
```

Alternate data streams

```
# show streams that aren't the normal $DATA
get-item evil.ps1 -stream "*" | where stream -ne ":$DATA"
# If you see an option that isn't $DATA, hone in on it
get-content evil.ps1 -steam "evil_stream"
```

Read hex of file

```
gc .\evil.ps1 -encoding byte |
Format-Hex
```

```
06/02/2021 23:46:45] | PS C:\Users\IEUser\Desktop > gc .\evil.ps1 -encoding byte | Format-H

          Path:

          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   65 63 68 6F 20 22 65 76 69 6C 22                  echo "evil"
```

Recursively look for particular file types, and once you find the files get their hashes

This one-liner was a godsend during the Microsoft Exchange ballache back in early 2021

```
Get-ChildItem -path "C:\windows\temp" -Recurse -Force -File -Include *.aspx, *.js
Get-FileHash |
format-table hash, path -autosize | out-string -width 800
```

```
Hash                                                             Path
----                                                             ----
919F49DDEC686768B09A3D6B59174B998AC16184FF64C14B17049B0F6F826573 C:\windows\temp\af397ef28e484961ba48646a5d38cf54.db
D7A991F392BFE9DB7ADF8510BFDBD1899560669684396442C0ED131F40E33C48 C:\windows\temp\af397ef28e484961ba48646a5d38cf54.db.ses
A342536B384355A5B1E05BD23A3BD9550E03255D15F129BC75F8100834504B1A C:\windows\temp\dd_vcredist_amd64_20190301101524.log
70D48E83DACDEE5BE185530879885B42CD75427EA056A2C4ADEBAEEB9B1F25CB C:\windows\temp\dd_vcredist_amd64_20190301101524_000_vcRuntimeMinimum_x64.log
D91C7B4C6944929CF022862CCE927872A25F146BD12B58F041312AE6A68B01B0 C:\windows\temp\dd_vcredist_amd64_20190301101524_001_vcRuntimeAdditional_x64.log
4C143D93943C4210C2605533AC24DE75FBF1AE897ED9577A1DB3497F7A210906 C:\windows\temp\dd_vcredist_amd64_20210310133207.log
53CB26C3189D68329A3EFF3D4479CE87851C42CFB50244ECCC482364EF3A0183 C:\windows\temp\dd_vcredist_amd64_20210310133207_000_vcRuntimeMinimum_x64.log
D6101C2F5E1CF0FB7337C74B1A60EDAF3640741926D0E61ECEE0DAF89255EE16 C:\windows\temp\dd_vcredist_amd64_20210310133207_001_vcRuntimeAdditional_x64.log
2D4184C3BC234451DCF9F328457F7CA41DC5BDE222262CA3D9FD0A1ADFC9EA72 C:\windows\temp\dd_vcredist_amd64_20210310133209.log
215FA713AE389D9456EBD6B2920DB965FDB3E6F4B968E2F3AD94E68442181C5E C:\windows\temp\dd_vcredist_x86_20190301101520.log
6D03E5B8970207EEA5867EB13558945378381DE084617E8C444B11C290728785 C:\windows\temp\dd_vcredist_x86_20190301101520_000_vcRuntimeMinimum_x86.log
```

Compare two files' hashes

```
get-filehash "C:\windows\sysmondrv.sys" , "C:\Windows\HelpPane.exe"
```

```
get-filehash "C:\windows\sysmondrv.sys" , "C:\Windows\HelpPane.exe"
```

```
Algorithm       Hash                                                             Path
---------       ----                                                             ----
SHA256          E074F2AD824A09400E6B5C6DC2F504C01FC60B5BE37CD6361DE822B3C4F18BFB C:\windows\sysmondrv.sys
SHA256          A1AD9018DB52A951D7E80B998DE7D6EE6B388D4AA1B46535E317662484186826 C:\Windows\HelpPane.exe
```

Find files written after X date

I personally wouldn't use this for DFIR. It's easy to manipulate timestamps....plus, Windows imports the original compiled date for some files and binaries if I'm not mistaken

Change the variables in the first time to get what you're looking. Remove the third line if you

want to include directories

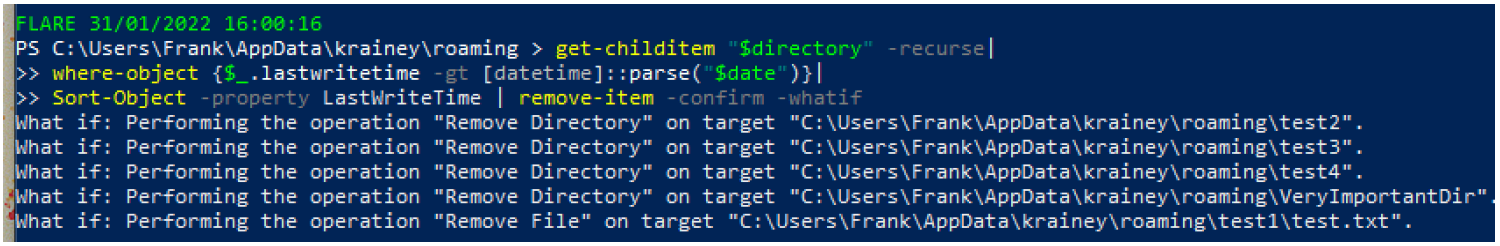
```
$date = "12/01/2021"; $directory = "C:\temp"
get-childitem "$directory" -recurse|
where-object {$_.mode -notmatch "d"}|
where-object {$_.lastwritetime -gt [datetime]::parse("$date")}|
Sort-Object -property LastWriteTime | format-table lastwritetime, fullname -autos
```

```
LastWriteTime       FullName
-------------       --------
12/01/2021 15:10:05 C:\temp\1
12/01/2021 15:27:21 C:\temp\1
12/01/2021 15:37:41 C:\temp\1
12/01/2021 15:43:53 C:\temp\S
12/01/2021 15:46:00 C:\temp\1
14/01/2021 11:50:07 C:\temp\K
14/01/2021 13:23:02 C:\temp\K
20/01/2021 13:24:48 C:\temp\v
```

Remove items written after x date

And then you can recursively remove the files and directories, in case malicious

```
$date = "31/01/2022"; $directory = "C:\Users\Frank\AppData\"
get-childitem "$directory" -recurse|
where-object {$_.lastwritetime -gt [datetime]::parse("$date")}|
Sort-Object -property LastWriteTime | remove-item -confirm -whatif
```



Remove the last -whatif flag to actaully detonate. Will ask you one at a time if you want to delete items. Please A to delete all

```
PS C:\Users\Frank\AppData\krainey\roaming > get-childitem "$directory" -recurse|
>> where-object {$_.lastwritetime -gt [datetime]::parse("$date")}|
>> Sort-Object -property LastWriteTime | remove-item -confirm

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove Directory" on target "C:\Users\Frank\AppData\krainey\roaming\test2".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove Directory" on target "C:\Users\Frank\AppData\krainey\roaming\test3".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): Y
```

copy multiple files to new location

```
copy-item "C:\windows\System32\winevt\Logs\Security.evtx", "C:\windows\System32\w
```

Grep in Powershell

Change the string in the second line. You should run these one after another, as it will grep for things in unicode and then ascii.

I like to use these as really lazy low-key yara rules. So grep for the string "educational purposes only" or something like that to catch malicious tooling - you'd be surprised how any vendors take open-source stuff, re-brand and compile it, and then sell it to you.....

```
ls C:\Windows\System32\* -include '*.exe', '*.dll' |
select-string 'RunHTMLApplication' -Encoding unicode |
select-object -expandproperty path -unique

#and with ascii
ls C:\Windows\System32\* -include '*.exe', '*.dll' |
select-string 'RunHTMLApplication' -Encoding Ascii |
select-object -expandproperty path -unique
```

```
[10/19/2021 14:59:51] | PS C:\ > ls C:\Windows\System32\* -include '*.exe', '*.dll' |
>> select-string 'RunHTMLApplication' -Encoding unicode |
>> select-object -expandproperty path -unique
[10/19/2021 15:00:24] | PS C:\ > ls C:\Windows\System32\* -include '*.exe', '*.dll' |
>> select-string 'RunHTMLApplication' -Encoding Ascii |
>> select-object -expandproperty path -unique
C:\Windows\System32\mshta.exe
C:\Windows\System32\mshtml.dll
[10/19/2021 15:00:58] | PS C:\ >
```

Registry Queries

▶ section contents

A note on HKCU

Just a note: Anywhere you see a reg key does HKCU - this is Current User. Your results will be limited to the user you are.

To see more results, you should change the above from HKCU, to HKU.

You often need the SID of the users you want to go and look at their information.

So for example, a query like this:

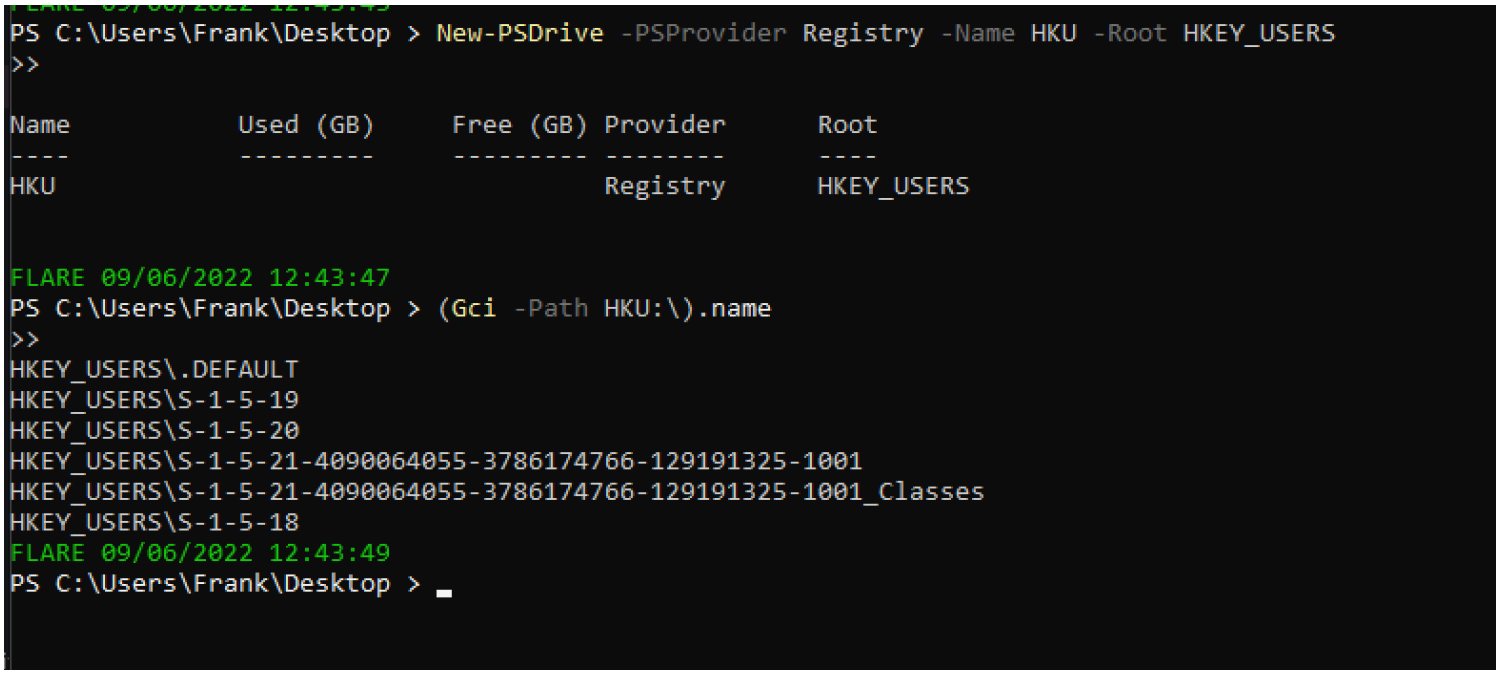
```
HKCU:\Control Panel\Desktop\
```

Becomes:

```
HKU\s-1-12-1-707864876-1224890504-1467553947-2593736053\Control Panel\Desktop
```

HKU needs to be set up to work

```
New-PSDrive -PSProvider Registry -Name HKU -Root HKEY_USERS;
(Gci -Path HKU:\).name
```



Show reg keys

Microsoft Docs detail the regs: their full names, abbrevated names, and what their subkeys generally house

```
##show all reg keys
(Gci -Path Registry::).name

# show HK users
mount -PSProvider Registry -Name HKU -Root HKEY_USERS;(Gci -Path HKU:\).name

##lets take HKEY_CURRENT_USER as a subkey example. Let's see the entries in this
(Gci -Path HKCU:\).name

# If you want to absolutely fuck your life up, you can list the names recursively
(Gci -Path HKCU:\ -recurse).name
```

```
[05/28/2021 14:20:39] | PS C:\Windows\system32 > (Gci -Path Registry::).name
HKEY_LOCAL_MACHINE
HKEY_CURRENT_USER
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_USERS
HKEY_PERFORMANCE_DATA
[05/28/2021 14:20:46] | PS C:\Windows\system32 > (Gci -Path HKCU:\).name
HKEY_CURRENT_USER\AppEvents
HKEY_CURRENT_USER\Console
HKEY_CURRENT_USER\Control Panel
HKEY_CURRENT_USER\Environment
HKEY_CURRENT_USER\EUDC
HKEY_CURRENT_USER\Keyboard Layout
HKEY_CURRENT_USER\Network
HKEY_CURRENT_USER\Printers
HKEY_CURRENT_USER\Software
HKEY_CURRENT_USER\System
HKEY_CURRENT_USER\Volatile Environment
[05/28/2021 14:21:51] | PS C:\Windows\system32 >
```

Read a reg entry

```
Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\SysmonDrv"
```

```
Get-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\SysmonDrv"
```

```
Type         : 1
Start        : 0
ErrorControl : 1
ImagePath    : SysmonDrv.sys
DisplayName  : SysmonDrv
Description  : System Monitor driver
PSPath       : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SYSTEM\Cu
               rrentControlSet\Services\SysmonDrv
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SYSTEM\Cu
               rrentControlSet\Services
PSChildName  : SysmonDrv
PSDrive      : HKLM
PSProvider   : Microsoft.PowerShell.Core\Registry
```

Quick useful reg keys

Query timezone on an endpoint. Look for the TimeZoneKeyName value

- `HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation`

Query the drives on the endpoint

- `HKLM\SYSTEM\MountedDevices`

Query the services on this machine, and if you want to see more about one of the results just add it to the path

- `HKLM\SYSTEM\CurrentControlSet\Services`
- `HKLM\SYSTEM\CurrentControlSet\Services\ACPI`

Query software on this machine

- `HKLM\Software`
- `HKLM\Software\PickOne`

Query SIDs

- `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList`
- `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\[Long-SID-Number-HERE]`

Query user's wallpaper. Once we know a user's SID, we can go and look at these things:

- `HKU\S-1-5-18\Control Panel\Desktop\`

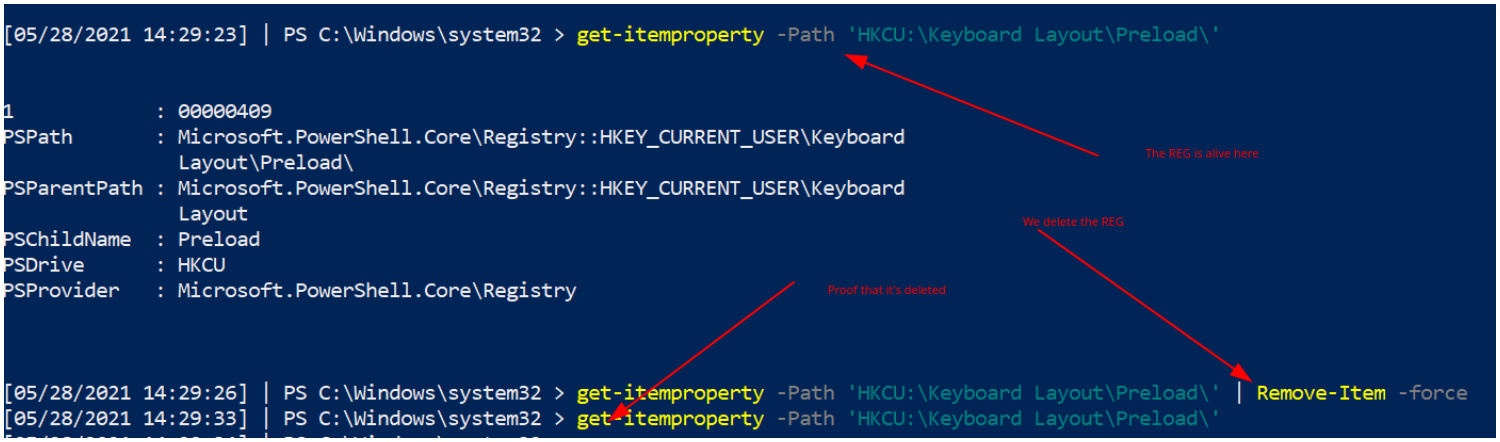
Query if credentials on a machine are being cached maliciously

```
# can run this network-wide
if ((Get-ItemProperty "HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\W

#remediate the malice with this
reg add "HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest" /v UseL
```

Remove a reg entry

If there's a malicious reg entry, you can remove it this way

```
#Create HKU drive
mount -PSProvider Registry -Name HKU -Root HKEY_USERS

# Read the reg to make sure this is the bad boy you want
get-itemproperty -Path 'HKU:\*\Keyboard Layout\Preload\'
#remove it by piping it to remove-item
get-itemproperty -Path 'HKU:\*\Keyboard Layout\Preload\' | Remove-Item -Force -Co
# double check it's gone by trying to re-read it
get-itemproperty -Path 'HKU:\*\Keyboard Layout\Preload\'
```



Removing HKCurrentUser Keys

If a Registry is under `HKCU`, it's not clear exactly WHO it can belong to.

```
================== File Contents ==================
iex ([System.Text.Encoding]::ASCII.GetString(( gp "HKCU:\Software\AppDataLow\Software\Microsoft\FDBC3F8C-385A-37D8-2A81-EC5BFE45E0BF").AJRoM1M0))
```

If a Registry is under `HKCU`, you can figure out WHICH username it belongs to but you can't just go into HKCU in your PwSh to delete it....because YOU are the current user.

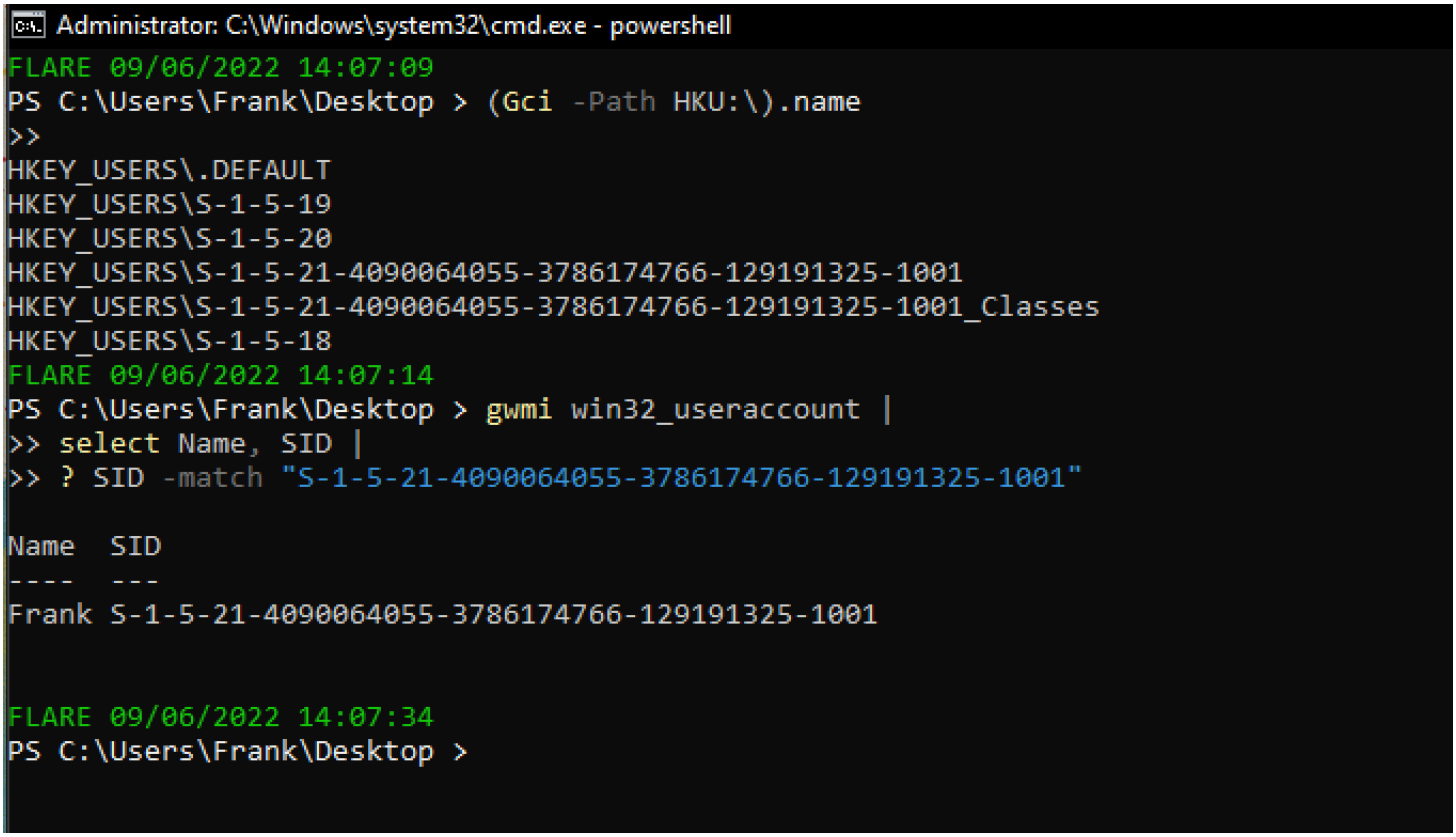
Instead, get the SID of the user

And then you can traverse to that as the path as HKU. So for example, under User_Alfonso's reg keys

```
#this
HKCU:\Software\AppDataLow\Software\Microsoft\FDBC3F8C-385A-37D8-2A81-EC5BFE45E0BF

#must become this. Notice the reg changes in the field field, and the SID gets sa
HKU:\S-1-5-21-912369493-653634481-1866108234-1004\Software\AppDataLow\Software\Mi
```

To just generally convert them

```
mount -PSProvider Registry -Name HKU -Root HKEY_USERS
```



Understanding Reg Permissions

Reg permissions, and ACL and SDDL in general really, are a bit long to understand. But worth it, as adversaries like using the reg.

Adversaries will look for registries with loose permissions, so let's show how we first can identify loose permissions

Get-ACl

The Access Control List (ACL) considers the permissions associated with an object on a Windows machine. It's how the machine understands privileges, and who is allowed to do what.

Problem is, if you get and `get-acl` for a particular object, it ain't a pretty thing

```
Get-Acl -Path hklm:\System\CurrentControlSet\services\ | fl
```

There's a lot going on here. Moreover, what the fuck is that SDDL string at the bottom?

The Security Descriptor Definition Language (SDDL) is a representation for ACL permissions, essentially

```
Get-Acl -Path hklm:\System\CurrentControlSet\services\ | fl
```

```
Path   : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\services\
Owner  : NT AUTHORITY\SYSTEM
Group  : NT AUTHORITY\SYSTEM
Access : BUILTIN\Users Allow  ReadKey
         BUILTIN\Administrators Allow  FullControl
         NT AUTHORITY\SYSTEM Allow  FullControl
         CREATOR OWNER Allow  FullControl
         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow  ReadKey
         S-1-15-3-1024-1065365936-1281604716-3511738428-1654721687-432734479-3232135806-4053264122-3456934681 Allow
         ReadKey
Audit  :
Sddl   : O:SYG:SYD:AI(A;CIID;KR;;;BU)(A;CIID;KA;;;BA)(A;CIID;KA;;;SY)(A;CIIOID;KA;;;CO)(A;CIID;KR;;;AC)(A;CIID;KR;;;S-1
         -15-3-1024-1065365936-1281604716-3511738428-1654721687-432734479-3232135806-4053264122-3456934681)
```

Convert SDDL

You could figure out what the wacky ASCII chunks mean in SDDL....but I'd much rather convert the permissions to something human readable

Here, an adversary is looking for a user they control to have permissions to maniptulate the service, likely they want *Full Control*

```
$acl = Get-Acl -Path hklm:\System\CurrentControlSet\services\;
ConvertFrom-SddlString -Sddl $acl.Sddl | Foreach-Object {$_.DiscretionaryAcl[0]};
ConvertFrom-SddlString -Sddl $acl.Sddl -Type RegistryRights | Foreach-Object {$_.
# bottom one specifices the  registry access rights when you create RegistrySecur
```

```
 BUILTIN\Users: AccessAllowed Inherited (ExecuteKey, ListDirectory, ReadExtendedAttributes, ReadPermissions, WriteExtendedAttributes)
BUILTIN\Users: AccessAllowed Inherited (EnumerateSubKeys, ExecuteKey, Notify, QueryValues, ReadPermissions)
```

What could they do with poor permissions?

An adversary in control of a loosely permissioned registry entry for a service, for example, could give themselves a privesc or persistence. For example:

```
#don't actually run this
Set-ItemProperty -path HKLM:\System\CurrentControlSet\services\example_service -n
```

Hunting for Reg evil

Now we know how reg entries are compromised, how can we search?

The below takes the services reg as an example, and searches for specifically just the reg-key

Name and Image Path.

```
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\services\*" |
ft PSChildName, ImagePath -autosize | out-string -width 800

#You can search recursively with this, kind of, if you use wildcards in the path
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\**\*" |
ft PSChildName, ImagePath -autosize | out-string -width 800

# This one-liner is over-engineered. # But it's a other way to be recursive if yo
# will take a while though
$keys = Get-ChildItem -Path "HKLM:\System\CurrentControlSet\" -recurse -force ;
$Items = $Keys | Foreach-Object {Get-ItemProperty $_.PsPath };
ForEach ($Item in $Items) {"{0,-35} {1,-10} " -f $Item.PSChildName, $Item.ImagePa
```

```
PSChildName                         ImagePath
-----------                         ---------
1394ohci                            \SystemRoot\System32\drivers\1394ohci.sys
3ware                               System32\drivers\3ware.sys
ACPI                                System32\drivers\ACPI.sys
acpiex                              System32\Drivers\acpiex.sys
acpipagr                            \SystemRoot\System32\drivers\acpipagr.sys
AcpiPmi                             \SystemRoot\System32\drivers\acpipmi.sys
acpitime                            \SystemRoot\System32\drivers\acpitime.sys
ADP80XX                             System32\drivers\ADP80XX.SYS
AeLookupSvc                         C:\Windows\system32\svchost.exe -k netsvcs
AFD                                 \SystemRoot\system32\drivers\afd.sys
agp440                              System32\drivers\agp440.sys
ahcache                             system32\DRIVERS\ahcache.sys
ALG                                 C:\Windows\System32\alg.exe
AmdK8                               \SystemRoot\System32\drivers\amdk8.sys
```

Filtering Reg ImagePath

Let's continue to use the \Services\ reg as our example.

Remember in the above example of a malicious reg, we saw the ImagePath had the value of C:\temp\evil.exe. And we're seeing a load of .sys here. So can we specifically just filter for .exes in the ImagePath.

I have to mention, don't write .sys files off as harmless. Rootkits and bootkits weaponise .sys, for example.

If you see a suspicious file in reg, you can go and collect it and investigate it, or collect it's hash. When it comes to the ImagePath, \SystemRoot\ is usually C:\Windows, but you can confirm with

```
$Env:systemroot .
```

```
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\services\*" |
where ImagePath -like "*.exe*" |
ft PSChildName, ImagePath -autosize | out-string -width 800

# if you notice, on line two we wrap .exe in TWO in wildcards. Why?
  # The first wildcard is to ensure we're kind of 'grepping' for a file that ends
    # Without the first wildcard, we'd be looking for literal .exe
  # The second wildcard is to ensure we're looking for the things that come after
     # This is to make sure we aren't losing the flags and args of an executable

# We can filter however we wish, so we can actively NOT look for .exes
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\services\*" |
where ImagePath -notlike "*.exe*" |
ft PSChildName, ImagePath -autosize | out-string -width 800

#fuck it, double stack your filters to not look for an exe or a sys...not sure wh
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\services\*" |
? {($_.ImagePath -notlike "*.exe*" -and $_.Imagepath -notlike "*.sys*")} |
ft PSChildName, ImagePath -autosize | out-string -width 800

#If you don't care about Reg Entry name, and just want the ImagePath
(Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\services\*").ImagePath
```

```
PSChildName                          ImagePath
-----------                          ---------
AJRouter                             C:\Windows\system32\svchost.exe -k
                                     LocalServiceNetworkRestricted
ALG                                  C:\Windows\System32\alg.exe
AM[redacted]                         "C:\Program Files
                                     ([redacted]).exe"
AppIDSvc                             C:\Windows\system32\svchost.exe -k
                                     LocalServiceNetworkRestricted
Appinfo                              C:\Windows\system32\svchost.exe -k
                                     netsvcs
```

Query Background Activity Moderator

BAM only in certain Windows 10 machines. Provides full path of the executabled last execution time

```
reg query "HKLM\SYSTEM\CurrentControlSet\Services\bam\state\UserSettings" /s
```

```
# or HKLM\SYSTEM\CurrentControlSet\Services\bam\UserSettings\
```

OR BAMParser.ps1

```
PS C:\ > reg query "HKLM\SYSTEM\CurrentControlSet\Services\bam\state\UserSettings" /s

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\bam\state\UserSettings\S-1-5-18
    Version    REG_DWORD    0x1
    SequenceNumber    REG_DWORD    0x1

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\bam\state\UserSettings\S-1-5-21-4090064055-3786174766-129191325-1000
    Version    REG_DWORD    0x1
    SequenceNumber    REG_DWORD    0x2
    Microsoft.Windows.CloudExperienceHost_cw5n1h2txyewy    REG_BINARY    9544B6A83E0DD80100000000000000000001000000002000000
    MicrosoftWindows.Client.CBS_cw5n1h2txyewy    REG_BINARY    390BA8A83E0DD8010000000000000000001000000002000000

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\bam\state\UserSettings\S-1-5-21-4090064055-3786174766-129191325-1001
    Version    REG_DWORD    0x1
    Microsoft.Windows.StartMenuExperienceHost_cw5n1h2txyewy    REG_BINARY    6AA7AC98D925D80100000000000000000001000000002000000
    SequenceNumber    REG_DWORD    0x15
    Microsoft.Windows.Search_cw5n1h2txyewy    REG_BINARY    90283299D925D80100000000000000000001000000002000000
    Microsoft.Windows.ShellExperienceHost_cw5n1h2txyewy    REG_BINARY    F08EE480DB25D80100000000000000000001000000002000000
    MicrosoftWindows.Client.CBS_cw5n1h2txyewy    REG_BINARY    12C93DA2D925D80100000000000000000001000000002000000
    Microsoft.WindowsStore_8wekyb3d8bbwe    REG_BINARY    D3F96AF4400DD80100000000000000000001000000002000000
    Microsoft.Windows.SecHealthUI_cw5n1h2txyewy    REG_BINARY    D3F96AF4400DD80100000000000000000001000000002000000
    windows.immersivecontrolpanel_cw5n1h2txyewy    REG_BINARY    A9AD7EA1540DD80100000000000000000001000000002000000
    \Device\HarddiskVolume3\Windows\explorer.exe    REG_BINARY    63C58C99D925D80100000000000000000000000000002000000
    \Device\HarddiskVolume3\Program Files\VMware\VMware Tools\vmtoolsd.exe    REG_BINARY    CEBFEAFADD25D80100000000000000000000000000
    \Device\HarddiskVolume3\Windows\System32\WindowsPowerShell\v1.0\powershell.exe    REG_BINARY    D0C66789DB25D80100000000000000000000
    \Device\HarddiskVolume3\Windows\System32\conhost.exe    REG_BINARY    D0C66789DB25D80100000000000000000000000000002000000
    \Device\HarddiskVolume3\Windows\System32\mmc.exe    REG_BINARY    004946A0CE25D80100000000000000000000000000002000000
    \Device\HarddiskVolume3\Windows\System32\notepad.exe    REG_BINARY    C94E96F4EB25D80100000000000000000000000000002000000
    \Device\HarddiskVolume3\Windows\System32\notepad.exe    REG_BINARY    2A30E9C30E24D80100000000000000000000000000002000000
    \Device\HarddiskVolume3\Users\Frank\Desktop\maybe_keylogger.exe    REG_BINARY    5E90EFC0DD24D80100000000000000000000000000000200000
    \Device\HarddiskVolume3\Program Files (x86)\Microsoft\Edge\Application\msedge.exe    REG_BINARY    83BD64E8D925D80100000000000000000
    \Device\HarddiskVolume3\Program Files\Process Hacker 2\ProcessHacker.exe    REG_BINARY    B1ACC51AF224D80100000000000000000000000000
    \Device\HarddiskVolume3\Windows\System32\OpenWith.exe    REG_BINARY    A084624DF224D80100000000000000000000000000002000000
    \Device\HarddiskVolume3\Program Files\7-Zip\7zFM.exe    REG_BINARY    4CA53121CE25D80100000000000000000000000000002000000
    \Device\HarddiskVolume3\Users\Frank\AppData\Local\Temp\Procmon64.exe    REG_BINARY    3513DA1CE225D80100000000000000000000000000002
    \Device\HarddiskVolume3\Windows\System32\Magnify.exe    REG_BINARY    DE3E3976E925D80100000000000000000000000000002000000
```

```
FLARE 19/02/2022 23:54:03
PS C:\ > .\updated_BAMparser.ps1

TimeUTC             Item                                                                                    User                 Sid
-------             ----                                                                                    ----                 ---
2022-02-19 23:53:55Z \Device\HarddiskVolume3\Windows\System32\notepad.exe                                   DESKTOP-MGCL3O0\Frank S-1-5-21-4090064055-3786174766-1
                                                                                                                                 29191325-1001
2022-02-19 23:36:04Z \Device\HarddiskVolume3\Windows\System32\Magnify.exe                                   DESKTOP-MGCL3O0\Frank S-1-5-21-4090064055-3786174766-1
                                                                                                                                 29191325-1001
2022-02-19 22:43:28Z \Device\HarddiskVolume3\Users\Frank\AppData\Local\Temp\Procmon64.exe                   DESKTOP-MGCL3O0\Frank S-1-5-21-4090064055-3786174766-1
                                                                                                                                 29191325-1001
2022-02-19 22:13:53Z \Device\HarddiskVolume3\Program Files\VMware\VMware Tools\vmtoolsd.exe                DESKTOP-MGCL3O0\Frank S-1-5-21-4090064055-3786174766-1
                                                                                                                                 29191325-1001
2022-02-19 21:56:24Z \Device\HarddiskVolume3\Windows\System32\conhost.exe                                   DESKTOP-MGCL3O0\Frank S-1-5-21-4090064055-3786174766-1
                                                                                                                                 29191325-1001
2022-02-19 21:56:24Z \Device\HarddiskVolume3\Windows\System32\WindowsPowerShell\v1.0\powershell.exe         DESKTOP-MGCL3O0\Frank S-1-5-21-4090064055-3786174766-1
                                                                                                                                 29191325-1001
2022-02-19 21:56:09Z Microsoft.Windows.ShellExperienceHost_cw5n1h2txyewy                                    DESKTOP-MGCL3O0\Frank S-1-5-21-4090064055-3786174766-1
```

Driver Queries

▶ section contents

Drivers are an interesting one. It isn't everyday you'll see malware sliding a malicious driver in ; bootkits and rootkits have been known to weaponise drivers. But it's well worth it, because it's an excellent method for persistence if an adversary can pull it off without blue-screening a

machine. You can read more about it here

You can utilise Winbindex to investigate drivers, and compare a local copy you have with the indexed info. Malicious copies may have a hash that doesn't match, or a file size that doesn't quite match.

Printer Drivers

```
Get-PrinterDriver | fl Name, *path*, *file*
```

```
Get-PrinterDriver | fl Name, *path*, *file*
```

```
Name           : Send to Microsoft OneNote 16 Driver
InfPath        : C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc975b614\prnms006.inf
Path           : C:\windows\System32\DriverStore\FileRepository\ntprint.inf_amd64_18b0d38ddfaee729\Amd64\mxdwdrv.dll
ColorProfiles  :
ConfigFile     : C:\windows\System32\DriverStore\FileRepository\prnms003.inf_amd64_7699f2338e4df80f\Amd64\PrintConfig.d
                 ll
DataFile       : C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc975b614\SendToOneNote.gpd
DependentFiles : {C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc975b614\SendToOneNote-mani
                 fest.ini, C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc975b614\SendToOne
                 Note-pipelineconfig.xml, C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc97
                 5b614\SendToOneNoteNames.gpd, C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb
                 6fc975b614\SendToOneNoteFilter.dll...}
HelpFile       :

Name           : Microsoft XPS Document Writer v4
InfPath        : C:\windows\System32\DriverStore\FileRepository\prnms001.inf_amd64_f340cb58fcd23202\prnms001.inf
Path           : C:\windows\System32\DriverStore\FileRepository\ntprint.inf_amd64_18b0d38ddfaee729\Amd64\mxdwdrv.dll
```

System Drivers

If drivers are or aren't signed, don't use that as the differentiation for what is legit and not legit. Some legitimate drivers are not signed ; some malicious drivers sneak a signature.

Unsigned

Get unsigned drivers. Likely to not return much

```
gci C:\Windows\*\DriverStore\FileRepository\ -recurse -include *.inf|
Get-AuthenticodeSignature |
? Status -ne "Valid" | ft -autosize

gci -path C:\Windows\System32\drivers -include *.sys -recurse -ea SilentlyContinu
Get-AuthenticodeSignature |
? Status -ne "Valid" | ft -autosize
```

Signed

Get the signed ones. Will return a lot.

```
Get-WmiObject Win32_PnPSignedDriver |
fl DeviceName, FriendlyName, DriverProviderName, Manufacturer, InfName, IsSigned,

# alternatives
gci -path C:\Windows\System32\drivers -include *.sys -recurse -ea SilentlyContinu
Get-AuthenticodeSignature |
? Status -eq "Valid" | ft -autosize
#or
gci C:\Windows\*\DriverStore\FileRepository\ -recurse -include *.inf|
Get-AuthenticodeSignature |
? Status -eq "Valid" | ft -autosize
```

```
DeviceName          : Motherboard resources
FriendlyName        :
DriverProviderName : Microsoft
Manufacturer        : (Standard system devices)
InfName             : machine.inf
IsSigned            : True
DriverVersion       : 10.0.17763.771

DeviceName          : ACPI Thermal Zone
FriendlyName        :
DriverProviderName : Microsoft
Manufacturer        : (Standard system devices)
InfName             : machine.inf
IsSigned            : True
DriverVersion       : 10.0.17763.771

DeviceName          : HID-compliant system controller
```

```
  gci -path C:\Windows\System32\drivers -include *.sys -recurse -ea S
e
```

```


    Directory: C:\Windows\System32\drivers\wd


SignerCertificate                         Status Path
-----------------                         ------ ----
14865CDDB19535A58A2D16F388E49DC2C255F956 Valid  WdBoot.sys
F7C2F2C96A328C13CDA8CDB57B715BDEA2CBD1D9 Valid  WdDevFlt.sys
F7C2F2C96A328C13CDA8CDB57B715BDEA2CBD1D9 Valid  WdFilter.sys
F7C2F2C96A328C13CDA8CDB57B715BDEA2CBD1D9 Valid  WdNisDrv.sys


    Directory: C:\Windows\System32\drivers


SignerCertificate                         Status Path
-----------------                         ------ ----
AE9C1AE54763822EEC42474983D8B635116C8452 Valid  1394ohci.sys
AE9C1AE54763822EEC42474983D8B635116C8452 Valid  3ware.sys
```

Other Drivers

Gets all 3rd party drivers

```
Get-WindowsDriver -Online -All |
fl Driver, ProviderName, ClassName, ClassDescription, Date, OriginalFileName, Dri
```

```
Driver           : 1394.inf
ProviderName     : Microsoft
ClassName        : 1394
ClassDescription : IEEE 1394 host controllers
Date             : 21/06/2006 00:00:00
OriginalFileName : C:\Windows\System32\DriverStore\FileRepository\1394.inf_amd64_4fad51adb157038a\1394.inf
DriverSignature  : Signed

Driver           : 3ware.inf
ProviderName     : LSI
ClassName        : SCSIAdapter
ClassDescription : Storage controllers
Date             : 11/04/2013 00:00:00
OriginalFileName : C:\Windows\System32\DriverStore\FileRepository\3ware.inf_amd64_408ceed6ec8ab6cd\3ware.inf
DriverSignature  : Signed

Driver           : 61883.inf
```

Drivers by Registry

You can also leverage the Registry to look at drivers

```
#if you know the driver, you can just give the full path and wildcard the end if
get-itemproperty -path "HKLM:\System\CurrentControlSet\Services\DBUtil*"

#You'll likely not know the path though, so just filter for drivers that have \dr
get-itemproperty -path "HKLM:\System\CurrentControlSet\Services\*"  |
? ImagePath -like "*drivers*" |
fl ImagePath, DisplayName
```

```
ImagePath   : \SystemRoot\System32\drivers\1394ohci.sys
DisplayName : @1394.inf,%PCI\CC_0C0010.DeviceDesc%;1394 OHCI Compliant Host
              Controller

ImagePath : System32\drivers\3ware.sys

ImagePath   : System32\drivers\ACPI.sys
DisplayName : @acpi.inf,%ACPI.SvcDesc%;Microsoft ACPI Driver

ImagePath   : \SystemRoot\System32\drivers\AcpiDev.sys
DisplayName : @acpidev.inf,%AcpiDev.SvcDesc%;ACPI Devices driver

ImagePath   : System32\Drivers\acpiex.sys
DisplayName : Microsoft ACPIEx Driver
```

(

Drivers by Time

Look for the drivers that exist via directory diving.. We can focus on .INF and .SYS files, and sort by the time last written.

```
#change to LastWriteTimeUtc if you need to.

# first directory location
gci C:\Windows\*\DriverStore\FileRepository\ -recurse -include *.inf |
sort-object LastWriteTime -Descending |
ft FullName,LastWriteTime | out-string -width 850

# second driver location
gci -path C:\Windows\System32\drivers -include *.sys -recurse -ea SilentlyContinu
sort-object LastWriteTime -Descending |
ft FullName,LastWriteTime | out-string -width 850
```

```
 gci C:\Windows\*\DriverStore\FileRepository\ -recurse -include *.inf |
sort-object LastWriteTime -Descending |
ft Name,LastWriteTimeUtc | out-string -width 850
```

```
Name                                           LastWriteTimeUtc
----                                           ----------------
ntprint.inf                                    20/11/2020 12:08:07
prnms003.inf                                   20/11/2020 12:08:07
usb.inf                                        20/11/2020 12:08:07
iscsi.inf                                      20/11/2020 12:08:07
ntprint.inf                                    20/11/2020 12:08:07
prnms003.inf                                   20/11/2020 12:08:07
```

DLL Queries

▶ section contents

DLLs Used in Processes

We've already discussed how to show DLLs used in processes

But what about getting *granular*. Well, let's pick on a specific process we can see running, and let's get the DLLs involved, their file location, their size, and if they have a company name

```
get-process -name "google*" |
Fl @{l="Modules";e={$_.Modules | fl FileName, Size, Company | out-string}}

#alterntive version, just print filepath of specific process' DLL
(gps -name "google*").Modules.FileName
```

```
FileName : C:\WINDOWS\SYSTEM32\ntdll.dll
Size     : 1972
Company  : Microsoft Corporation

FileName : C:\WINDOWS\System32\KERNEL32.DLL
Size     : 716
Company  : Microsoft Corporation

FileName : C:\WINDOWS\System32\KERNELBASE.dll
Size     : 2644
Company  : Microsoft Corporation

FileName : C:\WINDOWS\System32\ADVAPI32.dll
Size     : 652
Company  : Microsoft Corporation

FileName : C:\WINDOWS\System32\msvcrt.dll
Size     : 632
Company  : Microsoft Corporation

FileName : C:\WINDOWS\System32\sechost.dll
Size     : 632
Company  : Microsoft Corporation
```

You can in theory run this without specifying a process, and it will just retrieve all of the DLLs involved in all the processes. But this will be LONG man.

Investigate Process Dlls

We can zero in on the DLLs that a process may call on

```
(gps -name "google").Modules.FileName | Get-AuthenticodeSignature
```

```
SignerCertificate                         Status                                 Path
-----------------                         ------                                 ----
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  ntdll.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  KERNEL32.DLL
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  KERNELBASE.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  ADVAPI32.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  msvcrt.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  sechost.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  RPCRT4.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  CRYPT32.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  ucrtbase.dll
AE9C1AE54763822EEC42474983D8B635116C8452  Valid                                  MSASN1.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  ole32.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  combase.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  bcryptPrimitives.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  GDI32.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  gdi32full.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  msvcp_win.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  USER32.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  dbghelp.dll
A4341B9FD50FB9964283220A36A1EF6F6FAA7840  Valid                                  win32u.dll
```

Investigate DLLs

Generically

This will return a lot of DLLs and their last write time. I personally would avoid this approach

```
gci -path C:\Windows\*, C:\Windows\System32\*  -file -force -include *.dll | fl N

#to get signature codes for these pipe it
gci -path C:\Windows\*, C:\Windows\System32\*  -file -force -include *.dll | Get-
#to get hashes for these, pipe it too
gci -path C:\Windows\*, C:\Windows\System32\*  -file -force -include *.dll | get-
```

```
LastWriteTime : 15/09/2018 08:29:28

Name          : aadauthhelper.dll
LastWriteTime : 15/09/2018 08:28:30

Name          : aadcloudap.dll
LastWriteTime : 14/08/2019 02:21:19

Name          : aadjcsp.dll
LastWriteTime : 15/09/2018 08:28:38

Name          : aadtb.dll
LastWriteTime : 28/04/2020 02:31:22

Name          : aadWamExtension.dll
LastWriteTime : 15/09/2018 08:28:30

Name          : AboutSettingsHandlers.dll
LastWriteTime : 15/09/2018 08:28:56

Name          : AboveLockAppHost.dll
LastWriteTime : 07/08/2020 02:19:32
```

Invalid

Like drivers, if a DLL is signed or un-signed, it doesn't immediately signal malicious. There are plenty of official files on a Windows machine that are unsigned. Equally, malicious actors can get signatures for their malicious files too.

You'll get a lot of results if you look for VALID, signed DLLs. So maybe filter for INVALID ones first. Both will take some time

```
#get invalid
gci -path C:\Windows\*, C:\Windows\System32\*  -file -force -include *.dll |
Get-AuthenticodeSignature | ? Status -ne "Valid"

#collect valid ones with this command
gci -path C:\Windows\*, C:\Windows\System32\*  -file -force -include *.dll |
Get-AuthenticodeSignature | ? Status -eq "Valid"
```

```
   Directory: C:\Windows\System32


SignerCertificate                         Status                                 Path
-----------------                         ------                                 ----
                                          NotSigned                              cpuidsdk64.dll
                                          NotSigned                              cwbad1.dll
                                          NotSigned                              cwbcore.dll
                                          NotSigned                              cwbdc.dll
                                          NotSigned                              cwbdq.dll
                                          NotSigned                              cwbdt.dll
                                          NotSigned                              cwbodbc.dll
                                          NotSigned                              cwbrc.dll
                                          NotSigned                              cwbrw.dll
                                          NotSigned                              cwbsof.dll
                                          NotSigned                              cwbunpla.dll
                                          NotSigned                              cwbunpls.dll
                                          NotSigned                              cwbunssl.dll
                                          NotSigned                              cwbzzodb.dll
                                          NotSigned                              qxdaedrs.dll
```

Specifically

We can apply all of the above to individual DLLs. If I notice something strange during the process' DLL hunt, or if I had identified a DLL with an invalid signature. I'd then hone in on that specific DLL.

```
gci -path C:\Windows\twain_32.dll | get-filehash
gci -path C:\Windows\twain_32.dll | Get-AuthenticodeSignature
```

```
gci -path C:\Windows\twain_32.dll | Get-AuthenticodeSignature
```

```
    Directory: C:\Windows


SignerCertificate                         Status                                 Path
-----------------                         ------                                 ----
AE9C1AE54763822EEC42474983D8B635116C8452  Valid                                  twain_32.dll
```

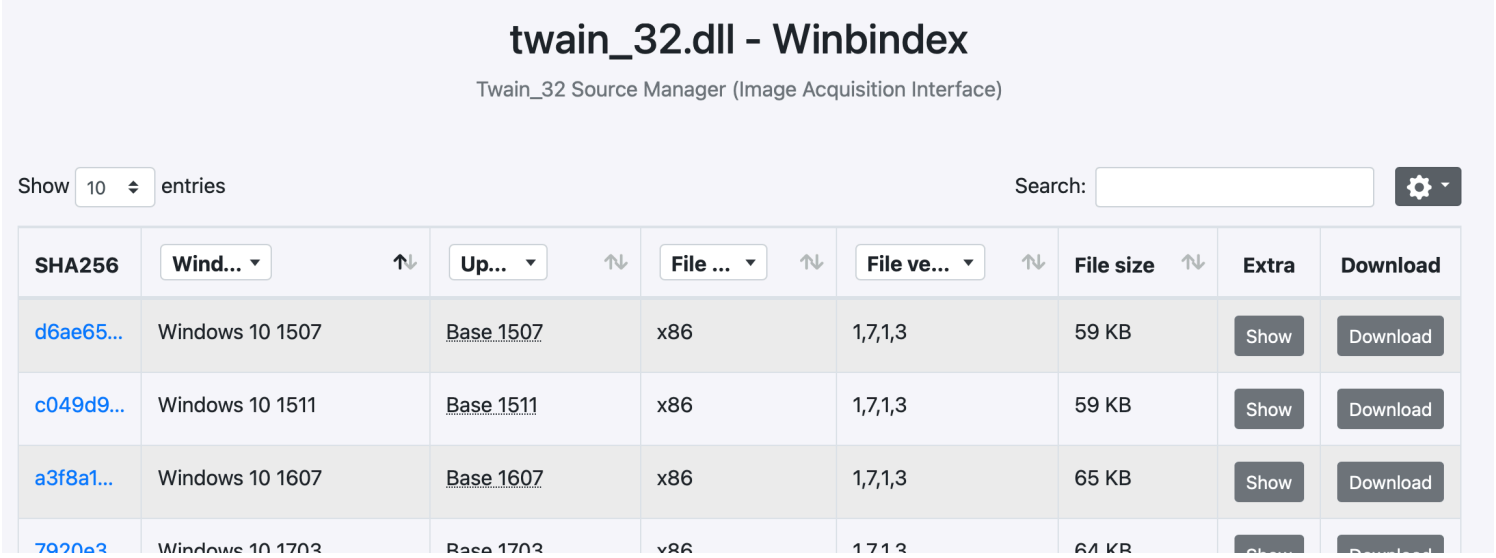
```
gci -path C:\Windows\twain_32.dll | get-filehash
```

```
Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          FD293C4A8B44BAEE2EFCB5FD19080620ECA07D3FF3F4A693701F354951A68F40       C:\Windows\twain_32.dll
```

Verify

If you need to verify what a DLL is, you have a myriad of ways. One way is through Winbindex

Here, you can put the name of a DLL (or many of other filetypes), and in return get a whole SLUETH of data. You can compare the file you have locally with the Winbindex info, which may highlight malice - for example, does the hash match ? Or, is your local copy a much larger file size than the suggested size in the index?



If not Windex, you have the usual Google-Fu methods, and having the file hash will aid you here

AV Queries

▶ section contents

Query Defender

If you have Defender active on your windows machine, you can leverage PowerShell to query what threats the AV is facing

This simple command will return all of the threats. In the screenshot below, it shows someone attempted to download mimikatz.

```
Get-MpThreatDetection
```

```
[11/02/2021 12:58:21] | PS C:\ > Get-MpThreatDetection


ActionSuccess                 : True
AdditionalActionsBitMask      : 0
AMProductVersion              : 4.18.2109.6
CleaningActionID              : 3
CurrentThreatExecutionStatusID : 0
DetectionID                   : {5259D447-0F3B-4738-BDC8-9372406683B5}
DetectionSourceTypeID         : 4
DomainUser                    : MSEDGEWIN10\IEUser
InitialDetectionTime          : 11/2/2021 12:58:14 PM
LastThreatStatusChangeTime    : 11/2/2021 12:58:21 PM
ProcessName                   : Unknown
RemediationTime               : 11/2/2021 12:58:21 PM
Resources                     : {file:_C:\Users\IEUser\Downloads\mimikatz_trunk.zip, webfile:_C:\Users\IEUser\Downloads
                                \mimikatz_trunk.zip|https://github-releases.githubusercontent.com/18496166/bfc2b8f2-26e
                                7-4893-9a4e-4d26a676794b?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4
                                CSVEH53A%2F20211102%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20211102T125812Z&X-Amz-E
                                xpires=300&X-Amz-Signature=43c2e845ff6ccb268bda75050305a9b414900a01e329fee56e2a220ef8d0
                                5df3&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=18496166&response-content-dis
                                position=attachment%3B%20filename%3Dmimikatz_trunk.zip&response-content-type=applicatio
                                n%2Foctet-stream|pid:5112,ProcessStart:132803314939264143}
ThreatID                      : 2147768041
ThreatStatusErrorCode         : 0
ThreatStatusID                : 4
PSComputerName                :
```

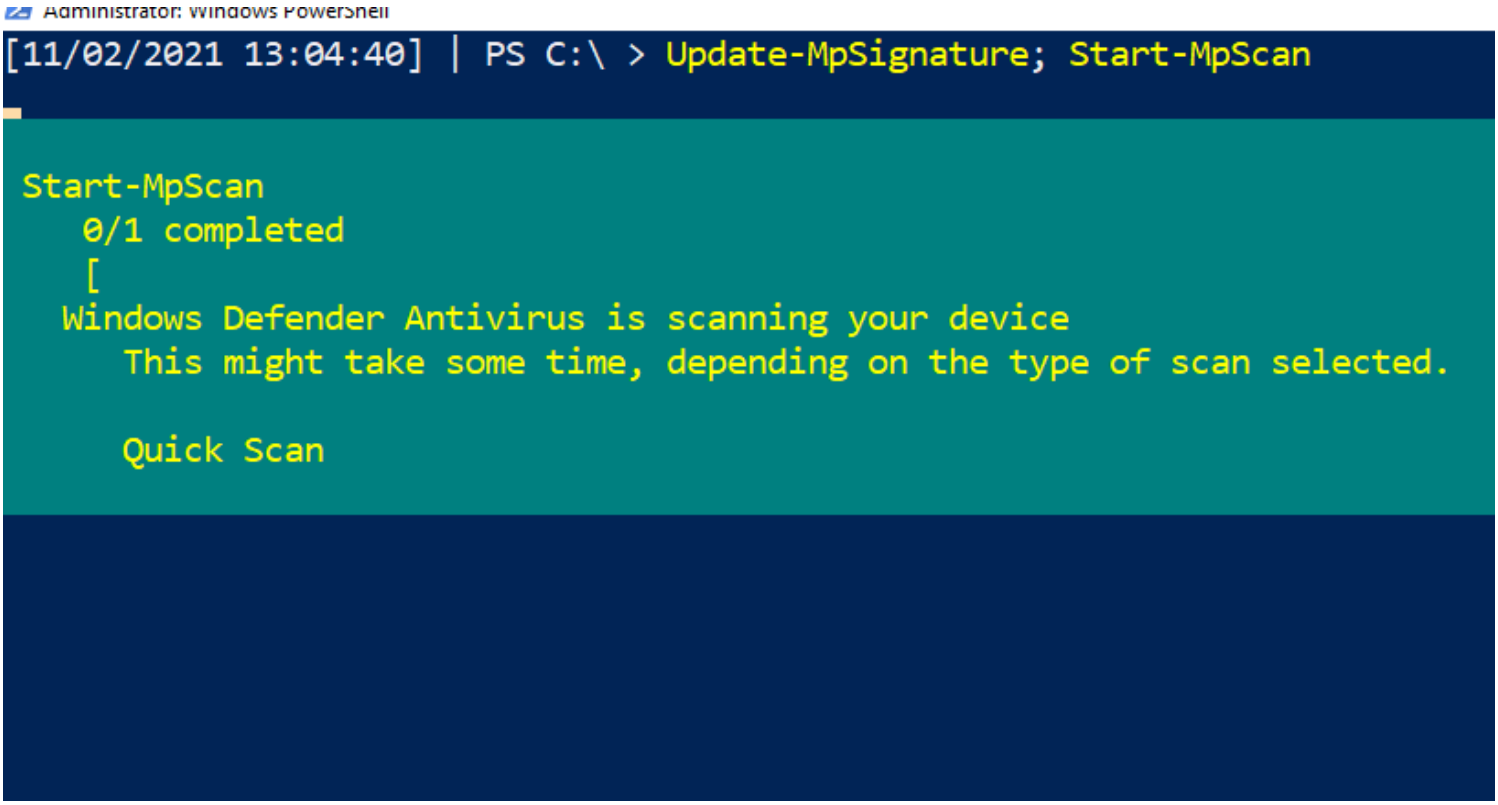
However, if you have numerous threat alerts, the above command may be messy to query. Let's demonstrate some augmentations we can add to make our hunt easier

```
Get-MpThreatDetection | Format-List threatID, *time, ActionSuccess
#Then, take the ThreatID and drill down further into that one
Get-MpThreat -ThreatID
```

```
[11/02/2021 12:58:47] | PS C:\ > Get-MpThreatDetection | Format-List threatID, *time, ActionSuccess


threatID                  : 2147768041
InitialDetectionTime      : 11/2/2021 12:58:14 PM
LastThreatStatusChangeTime : 11/2/2021 12:58:21 PM
RemediationTime           : 11/2/2021 12:58:21 PM
ActionSuccess             : True



[11/02/2021 13:00:59] | PS C:\ > Get-MpThreat -ThreatID 2147768041


CategoryID       : 8
DidThreatExecute : False
IsActive         : False
Resources        : {file:_C:\Users\IEUser\Downloads\mimikatz_trunk.zip, webfile:_C:\Users\IEUser\Downlo
                   k.zip|https://github-releases.githubusercontent.com/18496166/bfc2b8f2-26e7-4893-9a4e
```

Trigger Defender Scan

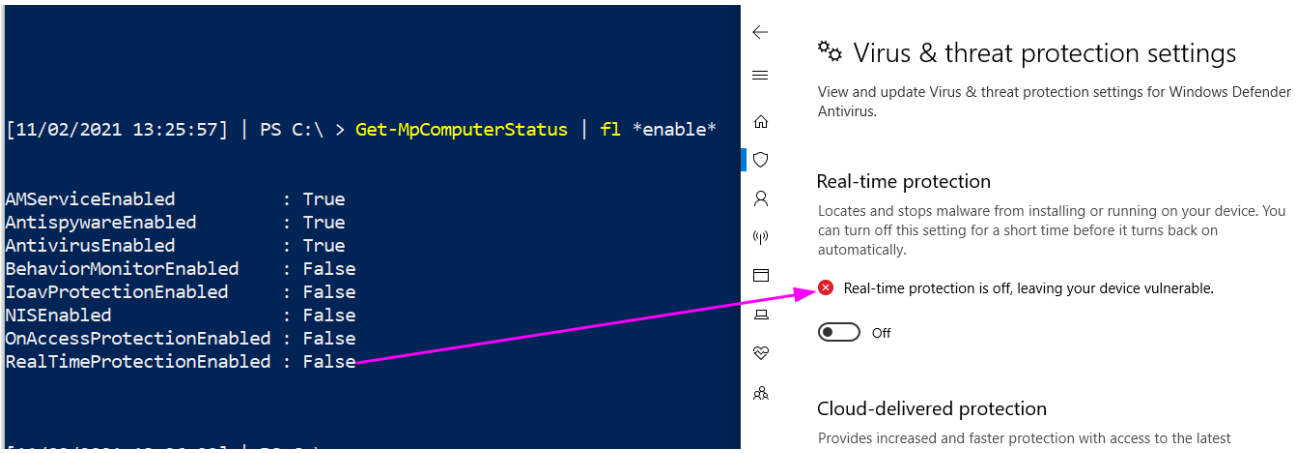
```
Update-MpSignature; Start-MpScan

#or full scan
Start-MpScan -ScanType FullScan

#Specify path
Start-MpScan -ScanPath "C:\temp"
```



Check if Defender has been manipulated

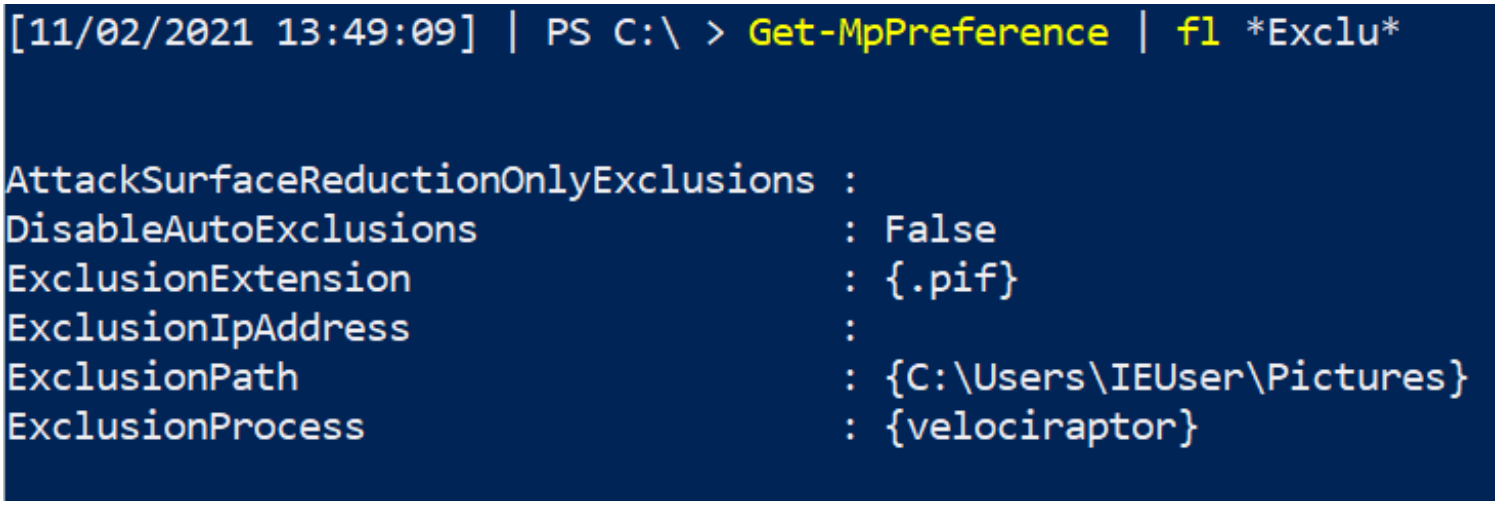
Adversaries enjoy simply turning off / disabling the AV. You can query the status of Defender's various detections

```
Get-MpComputerStatus | fl *enable*
```



Adversaries also enjoy adding exclusions to AVs....however please note that some legitimate tooling and vendors ask that some directories and executables are placed on the exclusion list

```
Get-MpPreference | fl *Exclu*
```



Enable Defender monitoring

If you see some values have been disabled, you can re-enable with the following:

```
Set-MpPreference -DisableRealtimeMonitoring $false -verbose
```

```
[11/02/2021 13:35:23] | PS C:\ > Set-MpPreference -DisableRealtimeMonitoring $false -verbose
VERBOSE: Performing operation 'Update MSFT_MpPreference' on Target 'ProtectionManagement'.
[11/02/2021 13:35:33] | PS C:\ > Get-MpComputerStatus | fl *enable*


AMServiceEnabled          : True
AntispywareEnabled        : True
AntivirusEnabled          : True
BehaviorMonitorEnabled    : True
IoavProtectionEnabled     : True
NISEnabled                : True
OnAccessProtectionEnabled : True
RealTimeProtectionEnabled : True
```

And get rid of the exclusions the adversary may have gifted themselves

```
Remove-MpPreference -ExclusionProcess 'velociraptor' -ExclusionPath 'C:\Users\IEU
```

```
[11/02/2021 14:02:05] | PS C:\ > Remove-MpPreference -ExclusionProcess 'velociraptor' -ExclusionPath 'C:\Users\IEUser\Pictures' -ExclusionExtension '.pif' -force -verbose
VERBOSE: Performing operation 'Update MSFT_MpPreference' on Target 'ProtectionManagement'.
[11/02/2021 14:03:46] | PS C:\ > Get-MpPreference | fl *Excl*


AttackSurfaceReductionOnlyExclusions :
DisableAutoExclusions                : False
ExclusionExtension                   :
ExclusionIpAddress                   :
ExclusionPath                        :
ExclusionProcess                     :
```

Log Queries

▶ section contents

From a security perspective, you probably don't want to query logs on the endpoint itself....endpoints after a malicious event can't be trusted. You're better to focus on the logs that have been forwarded from endpoints and centralised in your SIEM.

If you REALLY want to query local logs for security-related instances, I can recommend this awesome repo

I've tended to use these commands to troubleshoot Windows Event Forwarding and other log related stuff.

Show Logs

Show logs that are actually enabled and whose contents isn't empty.

```
Get-WinEvent -ListLog *|
where-object {$_.IsEnabled -eq "True" -and $_.RecordCount -gt "0"} |
```

```
sort-object -property LogName |
format-table LogName -autosize -wrap
```

```
LogName
-------
Application
Microsoft-Client-Licensing-Platform/Admin
Microsoft-Windows-AAD/Operational
Microsoft-Windows-Application-Experience/Program-Compatibility-Assistant
Microsoft-Windows-Application-Experience/Program-Telemetry
Microsoft-Windows-ApplicationResourceManagementSystem/Operational
Microsoft-Windows-AppModel-Runtime/Admin
Microsoft-Windows-AppReadiness/Admin
Microsoft-Windows-AppReadiness/Operational
Microsoft-Windows-AppXDeployment/Operational
Microsoft-Windows-AppXDeploymentServer/Operational
Microsoft-Windows-BackgroundTaskInfrastructure/Operational
Microsoft-Windows-Biometrics/Operational
Microsoft-Windows-Bits-Client/Operational
Microsoft-Windows-CertificateServicesClient-Lifecycle-System/Operational
Microsoft-Windows-CertificateServicesClient-Lifecycle-User/Operational
Microsoft-Windows-CodeIntegrity/Operational
Microsoft-Windows-Containers-Wcifs/Operational
```

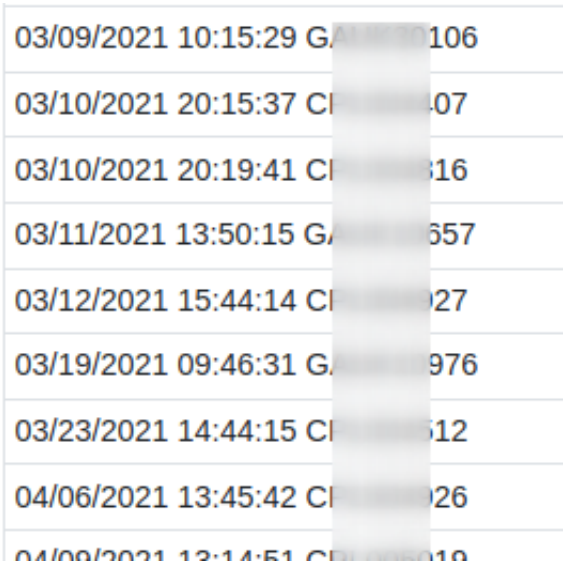
Overview of what a specific log is up to

```
Get-WinEvent -ListLog Microsoft-Windows-Sysmon/Operational | Format-List -Propert
```

```
FileSize                 : 67112960
IsLogFull                : False
LastAccessTime           : 08/03/2021 20:41:33
LastWriteTime            : 01/06/2021 15:12:50
OldestRecordNumber       : 23136464
RecordCount              : 84703
LogName                  : Microsoft-Windows-Sysmon/Operational
LogType                  : Operational
LogIsolation             : Custom
IsEnabled                : True
IsClassicLog             : False
SecurityDescriptor       : O:BAG:SYD:(A;;0xf0007;;;SY)(A;;0x7;;;BA)(A;;0x
                           1;;;BO)(A;;0x1;;;SO)(A;;0x1;;;S-1-5-32-573)(A;
                           ;0x1;;;NS)
LogFilePath              : %SystemRoot%\System32\Winevt\Logs\Microsoft-Wi
                           ndows-Sysmon%4Operational.evtx
MaximumSizeInBytes       : 67108864
LogMode                  : Circular
```

Specifically get the last time a log was written to

```
(Get-WinEvent -ListLog Microsoft-Windows-Sysmon/Operational).lastwritetime
```



Compare the date and time a log was last written to

Checks if the date was written recently, and if so, just print *sysmon working* if not recent, then print the date last written. I've found sometimes that sometimes sysmon bugs out on a machine, and stops committing to logs. Change the number after `-ge` to be more flexible than the one day it currently compares to

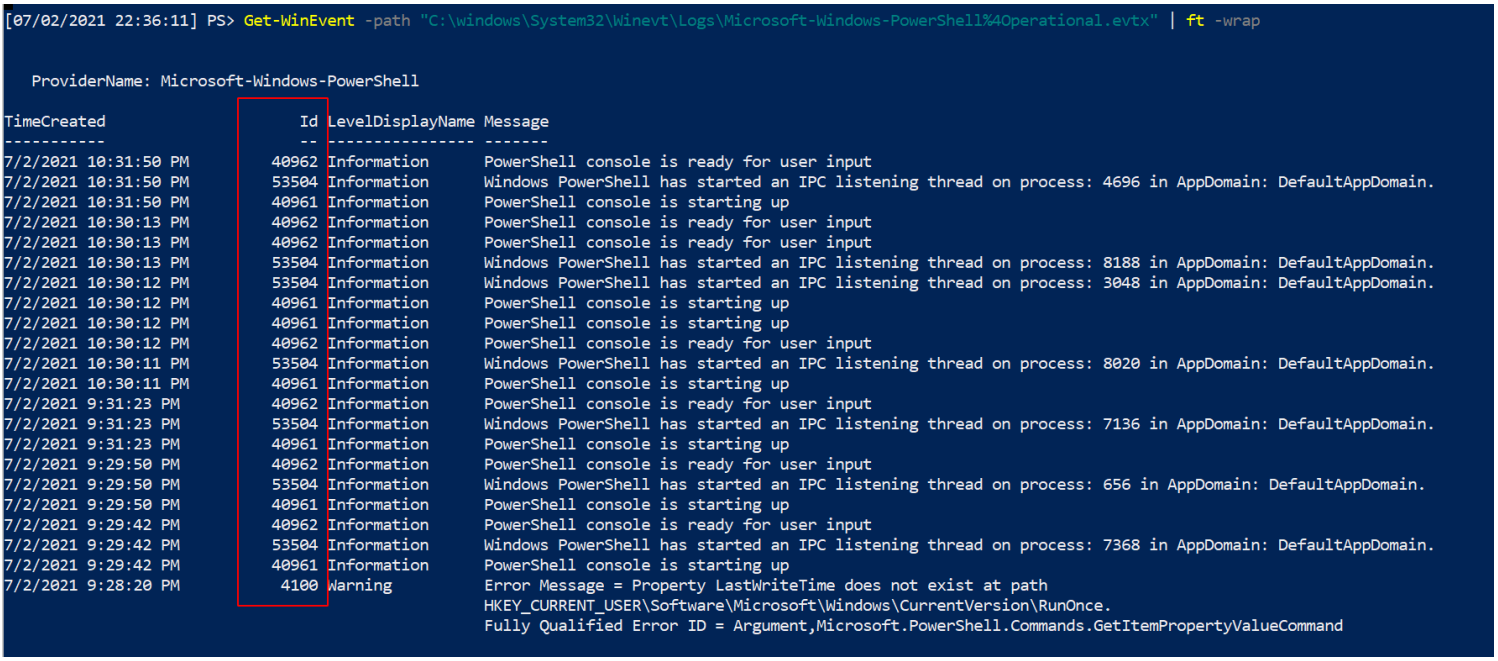
```
$b = (Get-WinEvent -ListLog Microsoft-Windows-Sysmon/Operational).lastwritetime;
$a = Get-WinEvent -ListLog Microsoft-Windows-Sysmon/Operational| where-object {(n
if ($a -eq $null){Write-host "sysmon_working"} else {Write-host "$env:computernam
```

Read a Log File

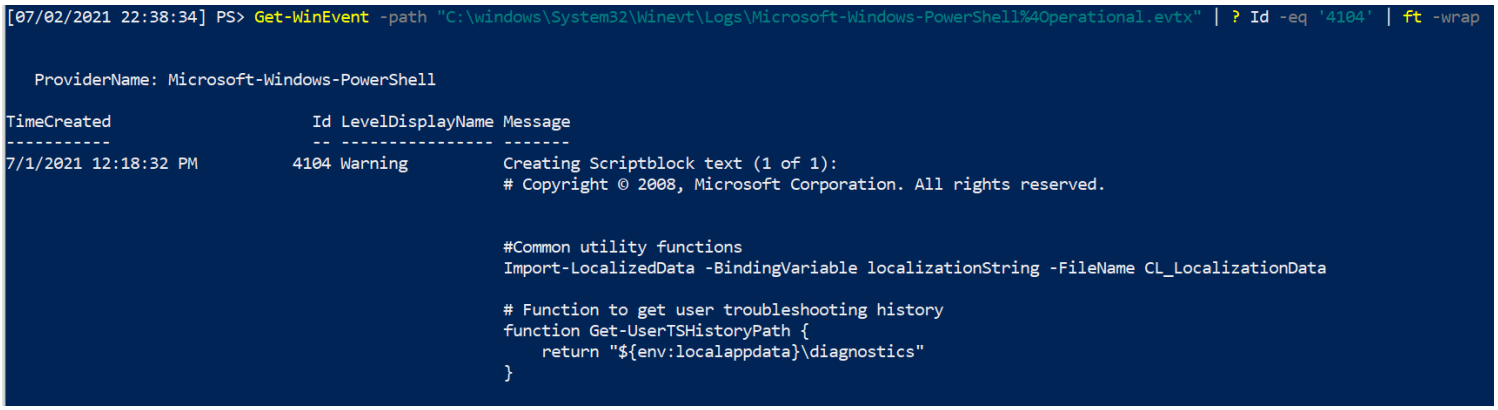
Again, trusting the logs of an endpoint is a dangerous game. An adversary can evade endpoint logging. It's better to utilise logs that have been taken to a central point, to trust EVENT IDs from Sysmon, or trust network traffic if you have it.

Nonetheless, you can read the EVTX file you are interesting in

```
Get-WinEvent -path "C:\windows\System32\Winevt\Logs\Microsoft-Windows-PowerShell%

#Advisable to filter by Id to filter out noise
Get-WinEvent -path "C:\windows\System32\Winevt\Logs\Microsoft-Windows-PowerShell%
? Id -eq '4104' | ft -wrap
#this is an example ID number.
```



```
[07/02/2021 22:36:11] PS> Get-WinEvent -path "C:\windows\System32\Winevt\Logs\Microsoft-Windows-PowerShell%4Operational.evtx" | ft -wrap


   ProviderName: Microsoft-Windows-PowerShell

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
7/2/2021 10:31:50 PM         40962 Information      PowerShell console is ready for user input
7/2/2021 10:31:50 PM         53504 Information      Windows PowerShell has started an IPC listening thread on process: 4696 in AppDomain: DefaultAppDomain.
7/2/2021 10:31:50 PM         40961 Information      PowerShell console is starting up
7/2/2021 10:30:13 PM         40962 Information      PowerShell console is ready for user input
7/2/2021 10:30:13 PM         40962 Information      PowerShell console is ready for user input
7/2/2021 10:30:13 PM         53504 Information      Windows PowerShell has started an IPC listening thread on process: 8188 in AppDomain: DefaultAppDomain.
7/2/2021 10:30:12 PM         53504 Information      Windows PowerShell has started an IPC listening thread on process: 3048 in AppDomain: DefaultAppDomain.
7/2/2021 10:30:12 PM         40961 Information      PowerShell console is starting up
7/2/2021 10:30:12 PM         40961 Information      PowerShell console is starting up
7/2/2021 10:30:12 PM         40962 Information      PowerShell console is ready for user input
7/2/2021 10:30:11 PM         53504 Information      Windows PowerShell has started an IPC listening thread on process: 8020 in AppDomain: DefaultAppDomain.
7/2/2021 10:30:11 PM         40961 Information      PowerShell console is starting up
7/2/2021 9:31:23 PM          40962 Information      PowerShell console is ready for user input
7/2/2021 9:31:23 PM          53504 Information      Windows PowerShell has started an IPC listening thread on process: 7136 in AppDomain: DefaultAppDomain.
7/2/2021 9:31:23 PM          40961 Information      PowerShell console is starting up
7/2/2021 9:29:50 PM          40962 Information      PowerShell console is ready for user input
7/2/2021 9:29:50 PM          53504 Information      Windows PowerShell has started an IPC listening thread on process: 656 in AppDomain: DefaultAppDomain.
7/2/2021 9:29:50 PM          40961 Information      PowerShell console is starting up
7/2/2021 9:29:42 PM          40962 Information      PowerShell console is ready for user input
7/2/2021 9:29:42 PM          53504 Information      Windows PowerShell has started an IPC listening thread on process: 7368 in AppDomain: DefaultAppDomain.
7/2/2021 9:29:42 PM          40961 Information      PowerShell console is starting up
7/2/2021 9:28:20 PM           4100 Warning          Error Message = Property LastWriteTime does not exist at path
                                                    HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce.
                                                    Fully Qualified Error ID = Argument,Microsoft.PowerShell.Commands.GetItemPropertyValueCommand
```



```
[07/02/2021 22:38:34] PS> Get-WinEvent -path "C:\windows\System32\Winevt\Logs\Microsoft-Windows-PowerShell%4Operational.evtx" | ? Id -eq '4104' | ft -wrap


   ProviderName: Microsoft-Windows-PowerShell

TimeCreated                     Id LevelDisplayName Message
-----------                     -- ---------------- -------
7/1/2021 12:18:32 PM          4104 Warning          Creating Scriptblock text (1 of 1):
                                                    # Copyright © 2008, Microsoft Corporation. All rights reserved.


                                                    #Common utility functions
                                                    Import-LocalizedData -BindingVariable localizationString -FileName CL_LocalizationData

                                                    # Function to get user troubleshooting history
                                                    function Get-UserTSHistoryPath {
                                                        return "${env:localappdata}\diagnostics"
                                                    }
```

WinRM & WECSVC permissions

Test the permissions of winrm - used to see windows event forwarding working, which uses winrm usually on endpoints and wecsvc account on servers

```
netsh http show urlacl url=http://+:5985/wsman/ && netsh http show urlacl url=htt
```

```
netsh http show urlacl url=http://+:5985/wsman/ && netsh http show urlacl url=https://+:5986/wsman/
```

```
URL Reservations:
-----------------

    Reserved URL            : http://+:5985/wsman/
        User: NT SERVICE\WinRM
            Listen: Yes
            Delegate: No
            SDDL: D:(A;;GX;;;S-1-5-80-569256582-2953403351-2909559716-1301513147-412116970)


URL Reservations:
-----------------

    Reserved URL            : https://+:5986/wsman/
        User: NT SERVICE\WinRM
            Listen: Yes
            Delegate: No
            SDDL: D:(A;;GX;;;S-1-5-80-569256582-2953403351-2909559716-1301513147-412116970)
```

Usage Log

These two blogs more or less share how to possibly prove when a C#/.net binary was executed 1, 2

The log's contents itself is useless. But, the file name of the log may be telling as it will be named after the binary executed.

A very basic way to query this is

```
gci "C:\Users\*\AppData\Local\Microsoft\*\UsageLogs\*", "C:\Windows\System32\conf
```

```
PS C:\> gci "C:\Users\*\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\*",
>> "C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\"


    Directory: C:\Users\IEUser\AppData\Local\Microsoft\CLR_v4.0\UsageLogs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        11/19/2022   8:38 PM            642 NGenTask.exe.log
-a----        11/24/2022   1:18 PM           3857 powershell.exe.log
-a----        11/11/2022   4:14 PM           5924 sdiagnhost.exe.log
-a----        11/24/2022   1:18 PM            659 SharpKatz.exe.log
-a----        11/11/2022   3:05 PM            425 Suborner64.exe.log


    Directory: C:\Users\toby\AppData\Local\Microsoft\CLR_v4.0\UsageLogs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        11/19/2022   8:38 PM            642 NGenTask.exe.log
-a----        11/19/2022   8:32 PM           5924 sdiagnhost.exe.log


    Directory: C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\CLR_v4.0\UsageLogs


Mode                 LastWriteTime         Length Name
```

If you wanted to query this network wide, you've got some options:

```
#Show usage log's created after a certain day
  #use american date, probably a way to convert it but meh
gci "C:\Users\*\AppData\Local\Microsoft\*\UsageLogs\*",
"C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\*\UsageLogs\*"
where-object {$_.LastWriteTime -gt [datetime]::parse("11/22/2022")} |
? Name -notmatch Powershell #can ignore and filter some names

# Show usage log but split to focus on the username, executable, and machine name
(gci "C:\Users\*\AppData\Local\Microsoft\*\UsageLogs\*").fullname |
ForEach-Object{$data = $_.split("\\");write-output "$($data[8]), $($data[2]), $(h
Select-String -notmatch "powershell", "NGenTask","sdiagnhost"

#For SYSTEM, you don't need to overcomplicate this
(gci "C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\*\UsageLog
ForEach-Object{ write-host "$_, SYSTEM, $(hostname)"}
```

```
PS C:\> gci "C:\Users\*\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\*",
>> "C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\" |
>> where-object {$_.LastWriteTime -gt [datetime]::parse("11/22/2022")} |
>> ? Name -notmatch Powershell


    Directory: C:\Users\IEUser\AppData\Local\Microsoft\CLR_v4.0\UsageLogs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----       11/24/2022   1:56 PM            642 NGenTask.exe.log
-a----       11/24/2022   1:18 PM            659 SharpKatz.exe.log


    Directory: C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\CLR_v4.0\UsageLogs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----       11/24/2022   1:56 PM            642 NGenTask.exe.log
-a----       11/24/2022   1:56 PM            226 taskhostw.exe.log
```

```
PS C:\> (gci "C:\Users\*\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\*").fullname |
>> ForEach-Object{$data = $_.split("\\");write-output "$($data[8]), $($data[2]), $(hostname)"} |
>> Select-String -notmatch "powershell", "NGenTask","sdiagnhost"

SharpKatz.exe.log, IEUser, MSEDGEWIN10
Suborner64.exe.log, IEUser, MSEDGEWIN10


PS C:\> (gci "C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\CLR_v4.0\UsageLogs\").name |
>> ForEach-Object{ write-host "$_, SYSTEM, $(hostname)"}
NGenTask.exe.log, SYSTEM, MSEDGEWIN10
powershell.exe.log, SYSTEM, MSEDGEWIN10
Suborner64.exe.log, SYSTEM, MSEDGEWIN10
taskhostw.exe.log, SYSTEM, MSEDGEWIN10
tzsync.exe.log, SYSTEM, MSEDGEWIN10
PS C:\>
```

But keep in mind, an adversary changing the file name is easy and therefore this is a meh telemetry source

```
    Directory: C:\Users\IEUser\AppData\Local\Microsoft\CLR_v4.0\UsageLogs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----       11/24/2022   1:56 PM            642 NGenTask.exe.log
-a----       11/24/2022   1:18 PM            659 SharpKatz.exe.log
-a----       11/24/2022   2:43 PM            425 totally_legit.exe.log
```

Powershell Tips

▶ section contents

Get Alias

PwSh is great at abbreviating the commands. Unfortunately, when you're trying to read someone else's abbreviated PwSh it can be ballache to figure out exactly what each weird abbrevation does.

Equally, if you're trying to write something smol and cute you'll want to use abbrevations!

Whatever you're trying, you can use `Get-Alias` to figure all of it out

```
#What does an abbrevation do
get-alias -name gwmi
#What is the abbrevation for this
get-alias -definition write-output
#List all alias' and their full command
get-alias
```

```
[06/02/2021 20:53:11] | PS C:\Users\IEUser > get-alias -name gwmi

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           gwmi -> Get-WmiObject


[06/02/2021 20:53:23] | PS C:\Users\IEUser > get-alias -definition write-output

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           echo -> Write-Output
Alias           write -> Write-Output
```

Get Command and Get Help

This is similar to `apropos` in Bash. Essentially, you can search for commands related to keywords you give.

Try to give singulars, not plural. For example, instead of `drivers` just do `driver`

```
get-command *driver*

## Once you see a particular command or function, to know what THAT does use get-
# get-help [thing]
Get-Help Get-SystemDriver
```

```
CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Function        Add-PrinterDriver                                  1.1        PrintManagement
Function        Get-OdbcDriver                                     1.0.0.0    Wdac
Function        Get-PrinterDriver                                  1.1        PrintManagement
Function        Remove-PrinterDriver                               1.1        PrintManagement
Function        Set-OdbcDriver                                     1.0.0.0    Wdac
Cmdlet          Add-WindowsDriver                                  3.0        Dism
Cmdlet          Export-WindowsDriver                               3.0        Dism
Cmdlet          Get-SystemDriver                                   1.0        ConfigCI
Cmdlet          Get-WindowsDriver                                  3.0        Dism
Cmdlet          Remove-WindowsDriver                               3.0        Dism
Application     driverquery.exe                                    10.0.17... C:\windows\system3
```

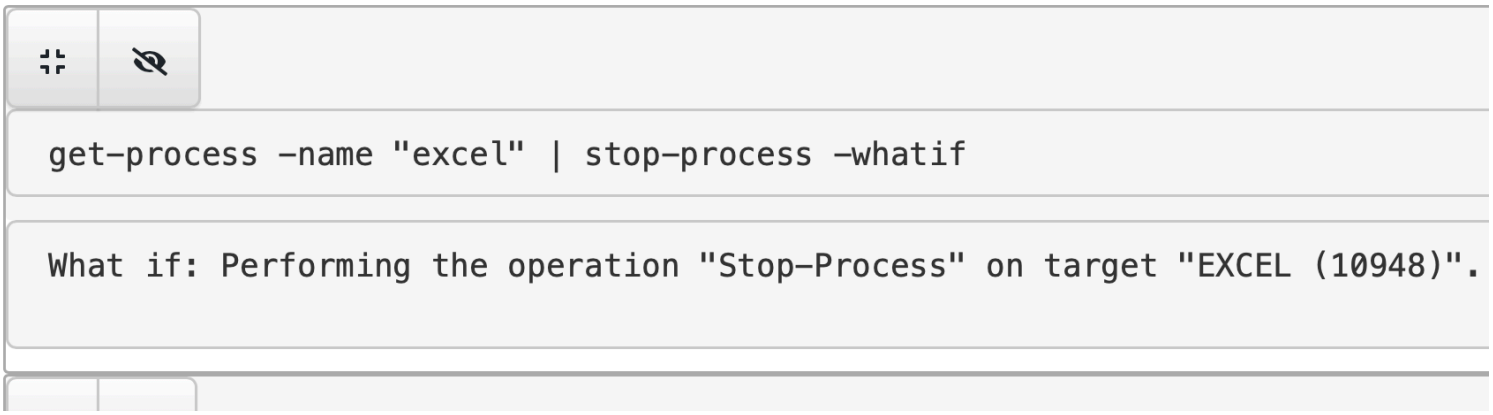
```
Get-Help Get-SystemDriver
```

```
NAME
    Get-SystemDriver

SYNTAX
    Get-SystemDriver [-Audit] [-ScanPath <string>] [-UserPEs] [-NoScript] [-NoShadowCopy] [-OmitPaths <string[]>]
    [-PathToCatroot <string>] [-ScriptFileNames]  [<CommonParameters>]


ALIASES
    None


REMARKS
    Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
        -- To download and install Help files for the module that includes this cmdlet, use Update-Help.
```

WhatIf

`-WhatIf` is quite a cool flag, as it will tell you what will happen if you run a command. So before you kill a vital process for example, if you include whatif you'll gain some insight into the irreversible future!

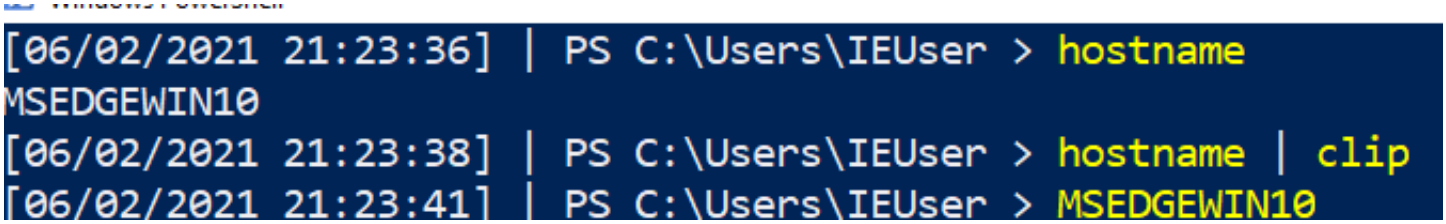
```
get-process -name "excel" | stop-process -whatif
```



Clip

You can pipe straight to your clipboard. Then all you have to do is paste

```
# this will write to terminal
hostname
# this will pipe to clipboard and will NOT write to terminal
hostname | clip
# then paste to test
#ctrl+v
```



Output Without Headers

You may just want a value without the collumn header that comes. We can do that with `-ExpandProperty`

```
# use the -expandproperty before the object you want. IN this case, ID
 select -ExpandProperty id

# so for example
get-process -Name "google*" | select -ExpandProperty id
# lets stop the particular google ID that we want
$PID =  get-process -Name "google" | ? Path -eq $Null | select -ExpandProperty id
Stop-Process -ID $PID -Force -Confirm:$false -verbose
```

```
get-process -Name "google*" | select -ExpandProperty id
```

```
10160
```

```
get-process -Name "google*" | select id
```

```
   Id
   --
10160
```

If you pipe to `|` `format-table` you can simply use the `-HideTableHeaders` flag

```
gps -name "google*" | ft -HideTableHeaders
```

```
   189      13     2144       3192       0.08  10160   0 GoogleUpdate
```

Re-run commands

If you had a command that was great, you can re-run it again from your powershell history!

```
##list out history
get-history
#pick the command you want, and then write down the corresponding number
#now invoke history
Invoke-History -id 38
```

```
## You can do the alias / abbrevated method for speed
h
r 43
```

```
 35 Clear-Content
 36 clear
 37 Get-History
 38 echo "howdy partner!"
 39 cls
 40 Get-History
 41 cls


[06/02/2021 22:11:38] | PS C:\Users\IEUser > Invoke-History -id 38
echo "howdy partner!"
howdy partner!
```

```
[06/02/2021 22:16:09] | PS C:\Users\IEUser > h

  Id CommandLine
  -- -----------
  50 clear-history
  51 cls
  52 history
  53 echo "howdy partner!"
  54 cls


[06/02/2021 22:16:13] | PS C:\Users\IEUser > r 53
echo "howdy partner!"
howdy partner!
```

Stop Trunction

Out-String

For reasons(?) powershell truncates stuff, even when it's really unhelpful and pointless for it to do so. Take the below for example: our hash AND path is cut off....WHY?! :rage:

```
Hash                                     Path
----                                     ----
18B16797CEC719FB6863BF3FAF2FAD6675E7... C:\Program Files\Microsoft Integrati...


Hash                                     Path
----                                     ----
022A1F9E9DA097C77698713A907F8AC81DCD... C:\Program Files\Microsoft Integrati...
```

To fix this, use `out-string`

```
#put this at the very end of whatever you're running and is getting truncated
| outstring -width 250
# or even more
| outstring -width 4096
#use whatever width number appropiate to print your results without truncation

#you can also stack it with ft. For example:
Get-ItemProperty -Path "HKLM:\System\CurrentControlSet\services\*" |
ft PSChildName, ImagePath -autosize | out-string -width 800
```

Look no elipses!

```
F30686DD09B81D4080AB58DEF209173772FA132FA3762688274270AFA6407872 C:\Windows\system32\conhost.exe

18B16797CEC719FB6863BF3FAF2FAD6675E79BF384EA4859B91764C22A352A6B C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diahost.exe

022A1F9E9DA097C77698713A907F8AC81DCD38461C2872EEB32001518A24A8B6 C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diawp.exe
```

-Wrap

In some places, it doesn't make sense to use out-string as it prints strangely. In these instances, try the `-wrap` function of `format-table`

This, for example is a mess because we used out-string. It's wrapping the final line in an annoying and strange way. ans

```
csrss.exe                   340 SYSTEM

csrss.exe                   412 SYSTEM

diahost.exe                3604 DIAHostService  "C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diahost.exe" -h 944

diawp.exe                  2240 DIAHostService  "C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diawp.exe" -K 1bc5309c-d
erProcess/WorkerProcessManagement -G False

diawp.exe                  3040 DIAHostService  "C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diawp.exe" -K 1e020d73-5
erProcess/WorkerProcessManagement -G False

diawp.exe                  2736 DIAHostService  "C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diawp.exe" -K 5bdcba3e-a
erProcess/WorkerProcessManagement -G False
```

```
| ft -property * -autosize -wrap
#you don't always need to the -property * bit. But if you find it isn't printing
| ft -autosize -wrap
```

Isn't this much better now?

```
conhost.exe                   5012 SYSTEM             \??\C:\Windows\syst
                                                      em32\conhost.exe
                                                      0x4
csrss.exe                      340 SYSTEM
csrss.exe                      412 SYSTEM
diahost.exe                   3604 DIAHostService      "C:\Program
                                                      Files\Microsoft
                                                      Integration Runtime
                                                      \5.0\Shared\diahost
                                                      .exe" -h 944
diawp.exe                     2240 DIAHostService      "C:\Program
                                                      Files\Microsoft
                                                      Integration Runtime
                                                      \5.0\Shared\diawp.e
                                                      xe" -K 1bc5309c-d1d
                                                      4-4db9-be4b-5de65e1
                                                      7e0c4 -U net.pipe:/
                                                      /localhost/WorkerPr
                                                      ocess/WorkerProcess
                                                      Management -G False
diawp.exe                     3040 DIAHostService      "C:\Program
                                                      Files\Microsoft
                                                      Integration Runtime
                                                      \5.0\Shared\diawp.e
                                                      xe" -K 1e020d73-510
```

Directories

For some investigations, I need to organise my directories or everything will get messed up. I enjoy using Year-Month-Date in my directory names!

```
mkdir -p "C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%Z")"
```

```
# your working directory for today will be
echo "C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%Z")"

##move to the working director
cd "C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%Z")"

##save outputs to
echo 'test' > C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%Z")\test.t
```

```
[12/01/2021 21:53:13] | PS C:\Malware_Analysis > mkdir -p "C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%Z")"


    Directory: C:\Malware_Analysis


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        12/1/2021   9:53 PM                2021_Dec_01_Wed_UTC+00
```

```
[12/01/2021 21:53:16] | PS C:\Malware_Analysis > echo "C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%Z")"
C:\Malware_Analysis\2021_Dec_01_Wed_UTC+00
[12/01/2021 21:53:27] | PS C:\Malware_Analysis > cd "C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%Z")"
[12/01/2021 21:53:37] | PS C:\Malware_Analysis\2021_Dec_01_Wed_UTC+00 > echo 'test' > C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%Z")\test.txt
[12/01/2021 21:53:44] | PS C:\Malware_Analysis\2021_Dec_01_Wed_UTC+00 > dir


    Directory: C:\Malware_Analysis\2021_Dec_01_Wed_UTC+00


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        12/1/2021   9:53 PM             14 test.txt
```

Transcripts

Trying to report back what you ran, when you ran, and the results of your commands can become a chore. If you forget a pivotal screenshot, you'll kick yourself - I know I have.

Instead, we can ask PowerShell to create a log of everything we run and see on the command line.

```
# you can pick whatever path you want, this is just what I tend to use it for
Start-Transcript -path "C:\Malware_Analysis\$(Get-Date -UFormat "%Y_%b_%d_%a_UTC%

## At the end of the malware analysis, we will then need to stop all transcripts
Stop-transcript

#you can now open up your Powershell transcript with notepad if you want
```

```
[12/01/2021 21:56:07] | PS C:\ > Start-Transcript -path "C:\Malware_Analysis\$(Get-Date -UFormat
"%Y_%b_%d_%a_UTC%Z")\PwSh_transcript.log" -noclobber -IncludeInvocationHeader
Transcript started, output file is C:\Malware_Analysis\2021_Dec_01_Wed_UTC+00\PwSh_transcript.log
```

```
[12/01/2021 21:57:15] | PS C:\ > stop-transcript
Transcript stopped, output file is C:\Malware_Analysis\2021_Dec_01_Wed_UTC+00\PwSh_transcript.log
```

```
[12/01/2021 21:57:08] | PS C:\ >
**********************
Command start time: 20211201215715
**********************
PS>get-service 'velociraptor'

Status   Name               DisplayName
------   ----               -----------
Running  Velociraptor       velociraptor


[12/01/2021 21:57:15] | PS C:\ >
**********************
Command start time: 20211201215722
**********************
PS>stop-transcript
**********************
Windows PowerShell transcript end
End time: 20211201215722
**********************
```

Linux

This section is a bit dry, forgive me. My Bash DFIR tends to be a lot more spontaneous and therefore I don't write them down as much as I do the Pwsh one-liners

Bash History

▶ section contents

Checkout the SANS DFIR talk by Half Pomeraz called You don't know jack about .bash_history. It's a terrifying insight into how weak bash history really is by default

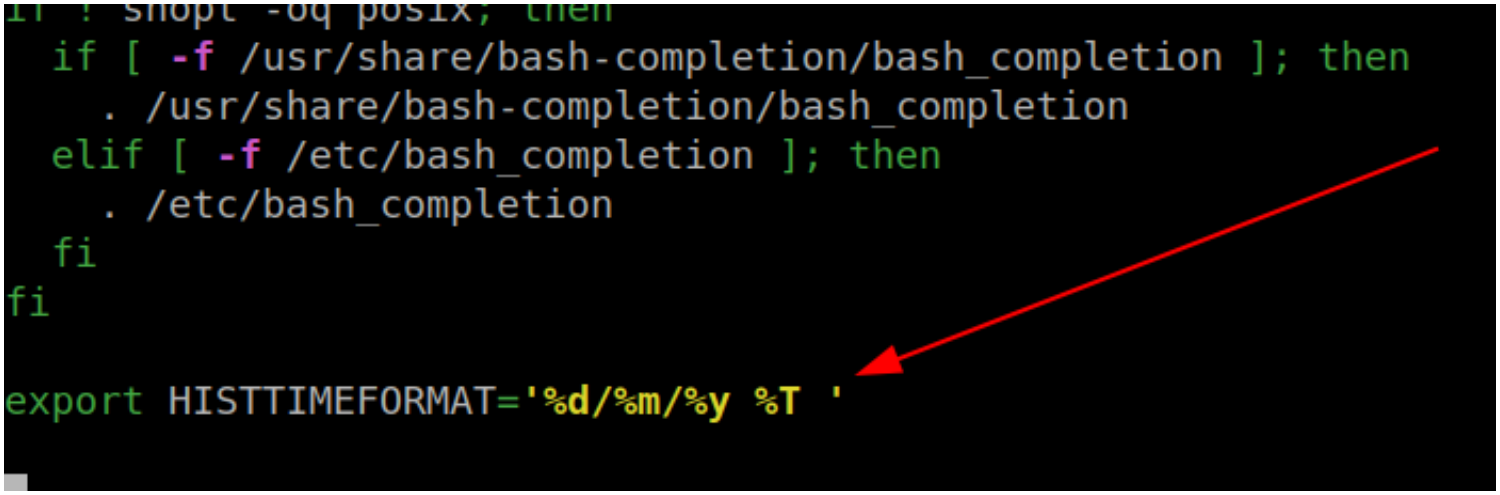
Add add timestamps to `.bash_history`

Via .bashrc

```
nano ~/.bashrc
#at the bottom
export HISTTIMEFORMAT='%d/%m/%y %T '
#expand bash history size too

#save and exit
source ~/.bashrc
```

Or by /etc/profile

```
nano /etc/profile
export HISTTIMEFORMAT='%d/%m/%y %T '

#save and exit
source /etc/profile
```



Then run the `history` command to see your timestamped bash history

```
3295  28/05/21 12:31:50 find . type f
3296  28/05/21 12:32:20 find . type f
3297  28/05/21 12:32:28 find . type f
3298  28/05/21 12:32:33 find . type f
3299  28/05/21 12:32:39 find . type f
3300  28/05/21 12:32:41 clear
3301  28/05/21 12:32:49 find . type f
3302  28/05/21 12:32:53 find . type f
3303  28/05/21 12:33:01 clear
3304  28/05/21 12:33:06 find . type f
3305  28/05/21 12:34:03 exit
3306  28/05/21 13:07:05 cd /opt
```

Grep and Ack

▶ section contents

Grep Regex extract IPs

IPv4

```
grep -E -o "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-
```

IPv6

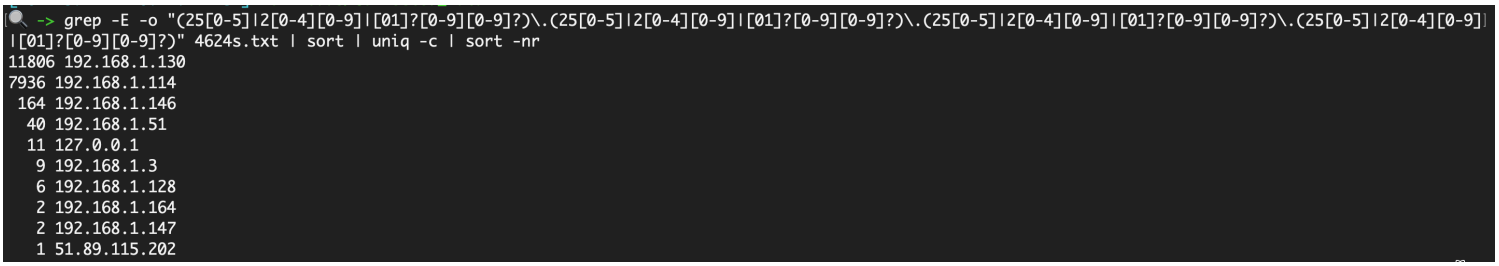
```
egrep '(([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,7}:|([0-9a
```

Stack up IPv4s

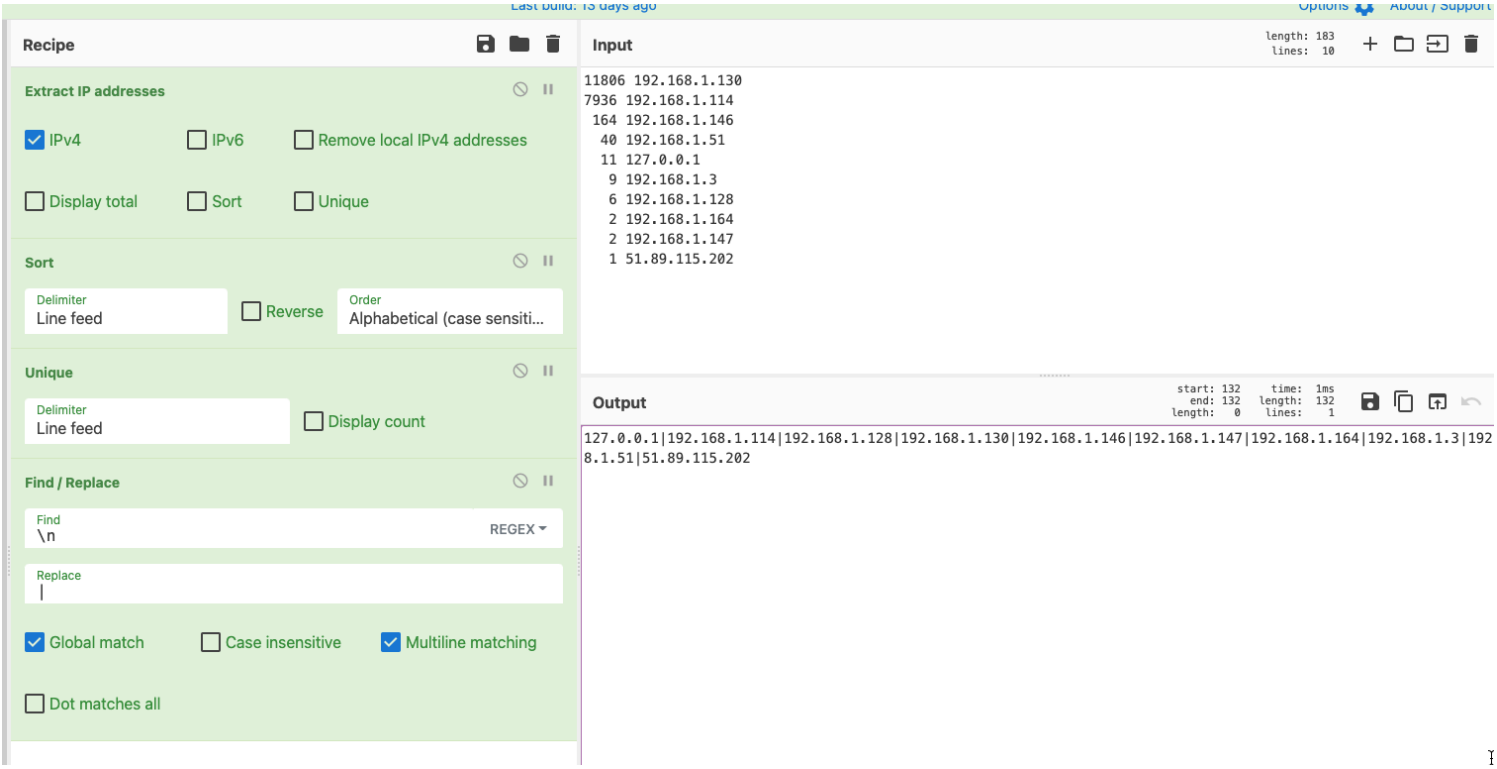
Great for parsing 4625s and 4624s in Windows world, and seeing the prelevence of the IPs trying to brute force you. Did a thread on this

So for example, this is a txt of all 4654s for an external pereimter server

```
grep -E -o "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-
```



To then prepare this to compare to the 4624s, I find it easiest to use this [cyberchef recipe](https://gchq.github.io/CyberChef/#recipe=Extract_IP_addresses(true,false,false,false,false,false)Sort('Line%20feed',false,'Alphabetical%20(case%20sensitive)')Unique('Line%20feed',false)Find_/_Replace(%7B'option':'Regex','string':'%5C%5Cn'%7D,'%7C',true,false,true,false))



And now, compare the brute forcing IPs with your 4624 successful logins, to see if any have successfully compromised you

```
grep -iEo '192.168.1.114|192.168.1.128|192.168.1.130|192.168.1.146|192.168.1.147|
```

Use Ack to highlight

One thing I really like about Ack is that it can highlight words easily, which is great for screenshots and reporting. So take the above example, let's say we're looking for two specific IP, we can have ack filter and highlight those

Ack is like Grep's younger, more refined brother. Has some of greps' flags as default, and just makes life a bit easier.

```
#install ack if you need to: sudo apt-get install ack
ack -i '127.0.0.1|1.1.1.1' --passthru file.txt
```

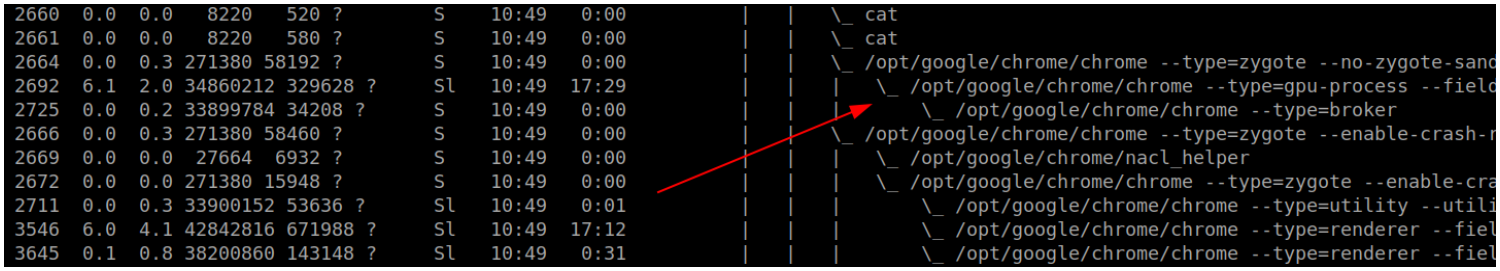
```
[02-Jun-21 10:41:31 BST] d/Desktop
-> ack '127.0.0.1|1.1.1.1' --passthru file.txt
3.3.3.3
1.1.1.1
127.0.0.1
10.10.10.10
20.20.20.20
192.192.192.192
127.0.0.1
```

Processes and Networks

▶ section contents

Track parent-child processes easier

```
ps -aux --forest
```



Get an overview of every running process running from a non-standard path

```
sudo ls -l /proc/[0-9]*/exe 2>/dev/null | awk '/ -> / && !/\/usr\/(lib(exec)?|s?b
```

```
[11-Jan-22 09:10:30 GMT] /
🔍 -> sudo ls -l /proc/[0-9]*/exe 2>/dev/nul
22343 -> /opt/google/chrome/chrome
22791 -> /opt/google/chrome/chrome
24350 -> /opt/google/chrome/chrome
24381 -> /opt/google/chrome/chrome
3597 -> /opt/google/chrome/chrome
3604 -> /opt/google/chrome/chrome_crashpad_h
3606 -> /opt/google/chrome/chrome_crashpad_h
3612 -> /opt/google/chrome/chrome
3614 -> /opt/google/chrome/chrome
3617 -> /opt/google/chrome/nacl_helper
3620 -> /opt/google/chrome/chrome
3640 -> /opt/google/chrome/chrome
3643 -> /opt/google/chrome/chrome
3651 -> /opt/google/chrome/chrome
3669 -> /opt/google/chrome/chrome
3677 -> /opt/google/chrome/chrome
4069 -> /opt/google/chrome/chrome
4084 -> /opt/google/chrome/chrome
```

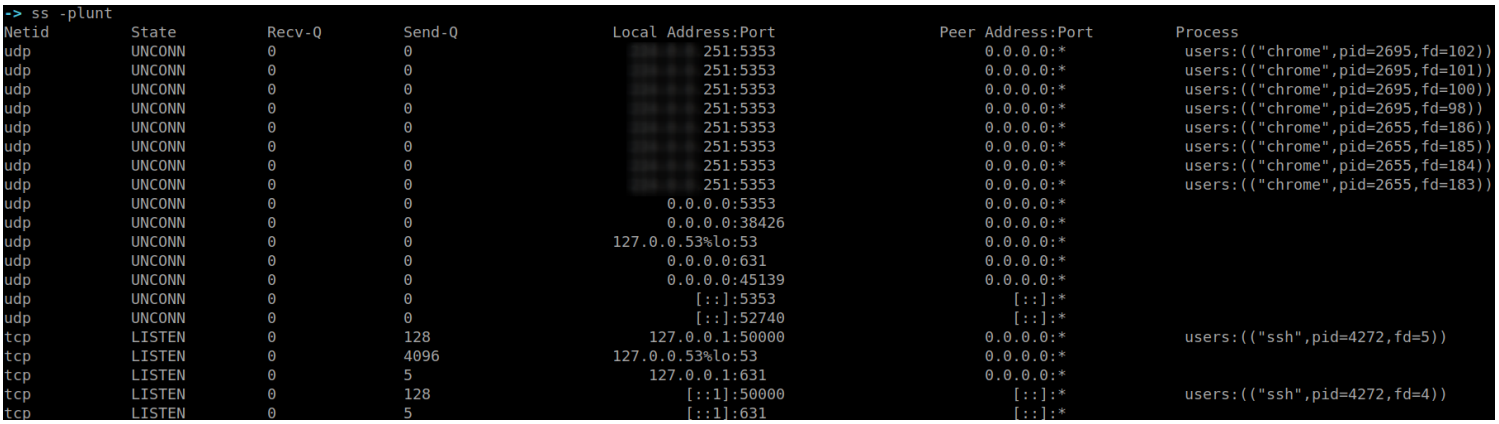
Or list every process full stop

```
sudo ls -l /proc/[0-9]*/exe 2>/dev/null | awk '/ -> / {print $NF}' | sort | tac
```

```
[11-Jan-22 09:11:24 GMT] /
🔍 -> sudo ls -l /proc/[0-9]*/exe 2>/dev/null | awk '/ -> / {print $NF}' | sort | tac
/usr/sbin/wpa_supplicant
/usr/sbin/vmware-authdlauncher
/usr/sbin/thermald
/usr/sbin/sshd
/usr/sbin/rsyslogd
/usr/sbin/openvpn
/usr/sbin/NetworkManager
/usr/sbin/ModemManager
/usr/sbin/lightdm
/usr/sbin/lightdm
/usr/sbin/kerneloops
/usr/sbin/kerneloops
/usr/sbin/irqbalance
/usr/sbin/cupsd
/usr/sbin/cups-browsed
/usr/sbin/cron
/usr/sbin/avahi-daemon
/usr/sbin/avahi-daemon
/usr/sbin/anacron
/usr/sbin/agetty
/usr/sbin/acpid
/usr/lib/x86_64-linux-gnu/xfce4/xfconf/xfconfd
```

Get a quick overview of network activity

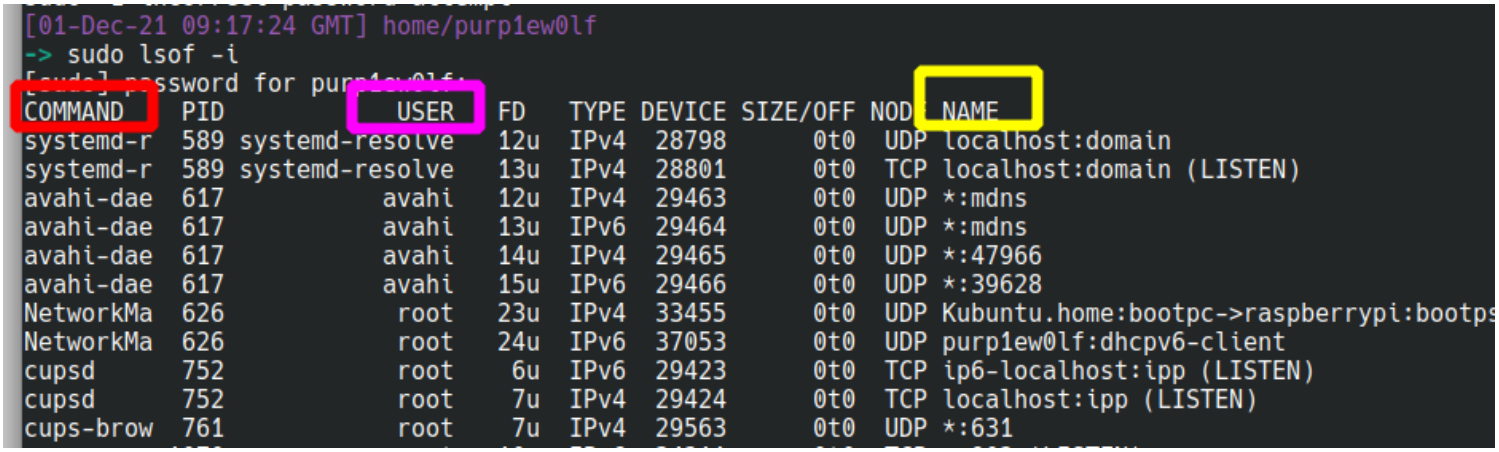
```
netstat -plunt
#if you don't have netstat, try ss
ss -plunt
```



```
-> ss -plunt
Netid        State         Recv-Q        Send-Q               Local Address:Port                Peer Address:Port        Process
udp          UNCONN        0             0                          [::]:5353                        [::]:*            users:(("chrome",pid=2695,fd=102))
udp          UNCONN        0             0                          [::]:5353                        [::]:*            users:(("chrome",pid=2695,fd=101))
udp          UNCONN        0             0                          [::]:5353                        [::]:*            users:(("chrome",pid=2695,fd=100))
udp          UNCONN        0             0                          [::]:5353                        [::]:*            users:(("chrome",pid=2695,fd=98))
udp          UNCONN        0             0                          [::]:5353                        [::]:*            users:(("chrome",pid=2655,fd=186))
udp          UNCONN        0             0                          [::]:5353                        [::]:*            users:(("chrome",pid=2655,fd=185))
udp          UNCONN        0             0                          [::]:5353                        [::]:*            users:(("chrome",pid=2655,fd=184))
udp          UNCONN        0             0                          [::]:5353                        [::]:*            users:(("chrome",pid=2655,fd=183))
udp          UNCONN        0             0                     0.0.0.0:5353                     0.0.0.0:*
udp          UNCONN        0             0                     0.0.0.0:38426                    0.0.0.0:*
udp          UNCONN        0             0                127.0.0.53%lo:53                      0.0.0.0:*
udp          UNCONN        0             0                     0.0.0.0:631                      0.0.0.0:*
udp          UNCONN        0             0                     0.0.0.0:45139                    0.0.0.0:*
udp          UNCONN        0             0                        [::]:5353                        [::]:*
udp          UNCONN        0             0                        [::]:52740                       [::]:*
tcp          LISTEN        0             128                 127.0.0.1:50000                  0.0.0.0:*            users:(("ssh",pid=4272,fd=5))
tcp          LISTEN        0             4096             127.0.0.53%lo:53                    0.0.0.0:*
tcp          LISTEN        0             5                   127.0.0.1:631                     0.0.0.0:*
tcp          LISTEN        0             128                     [::1]:50000                      [::]:*            users:(("ssh",pid=4272,fd=4))
tcp          LISTEN        0             5                       [::1]:631                        [::]:*
```

This alternative also helps re-visualise the originating command and user that a network connection belongs to

```
sudo lsof -i
```



```
[01-Dec-21 09:17:24 GMT] home/purp1ew0lf
-> sudo lsof -i
[sudo] password for purp1ew0lf:
COMMAND    PID            USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
systemd-r  589 systemd-resolve   12u  IPv4  28798      0t0  UDP localhost:domain
systemd-r  589 systemd-resolve   13u  IPv4  28801      0t0  TCP localhost:domain (LISTEN)
avahi-dae  617           avahi   12u  IPv4  29463      0t0  UDP *:mdns
avahi-dae  617           avahi   13u  IPv6  29464      0t0  UDP *:mdns
avahi-dae  617           avahi   14u  IPv4  29465      0t0  UDP *:47966
avahi-dae  617           avahi   15u  IPv6  29466      0t0  UDP *:39628
NetworkMa  626            root   23u  IPv4  33455      0t0  UDP Kubuntu.home:bootpc->raspberrypi:bootps
NetworkMa  626            root   24u  IPv6  37053      0t0  UDP purp1ew0lf:dhcpv6-client
cupsd      752            root    6u  IPv6  29423      0t0  TCP ip6-localhost:ipp (LISTEN)
cupsd      752            root    7u  IPv4  29424      0t0  TCP localhost:ipp (LISTEN)
cups-brow  761            root    7u  IPv4  29563      0t0  UDP *:631
```

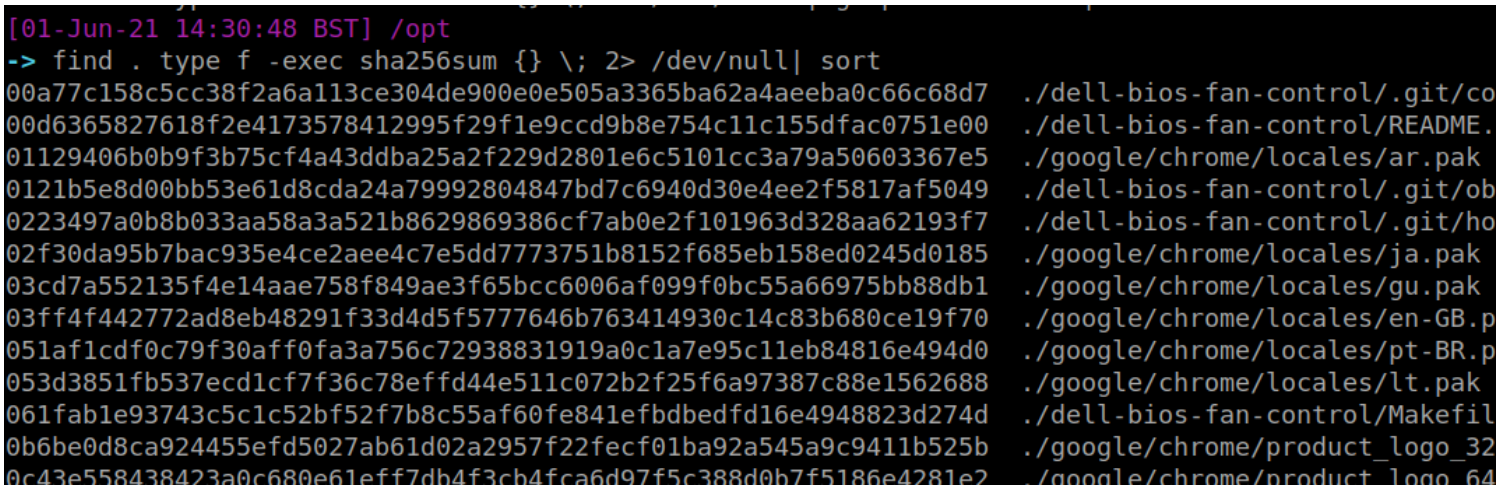
Files

▶ section contents

Recursively look for particular file types, and once you find the files get their hashes

Here's the bash alternative

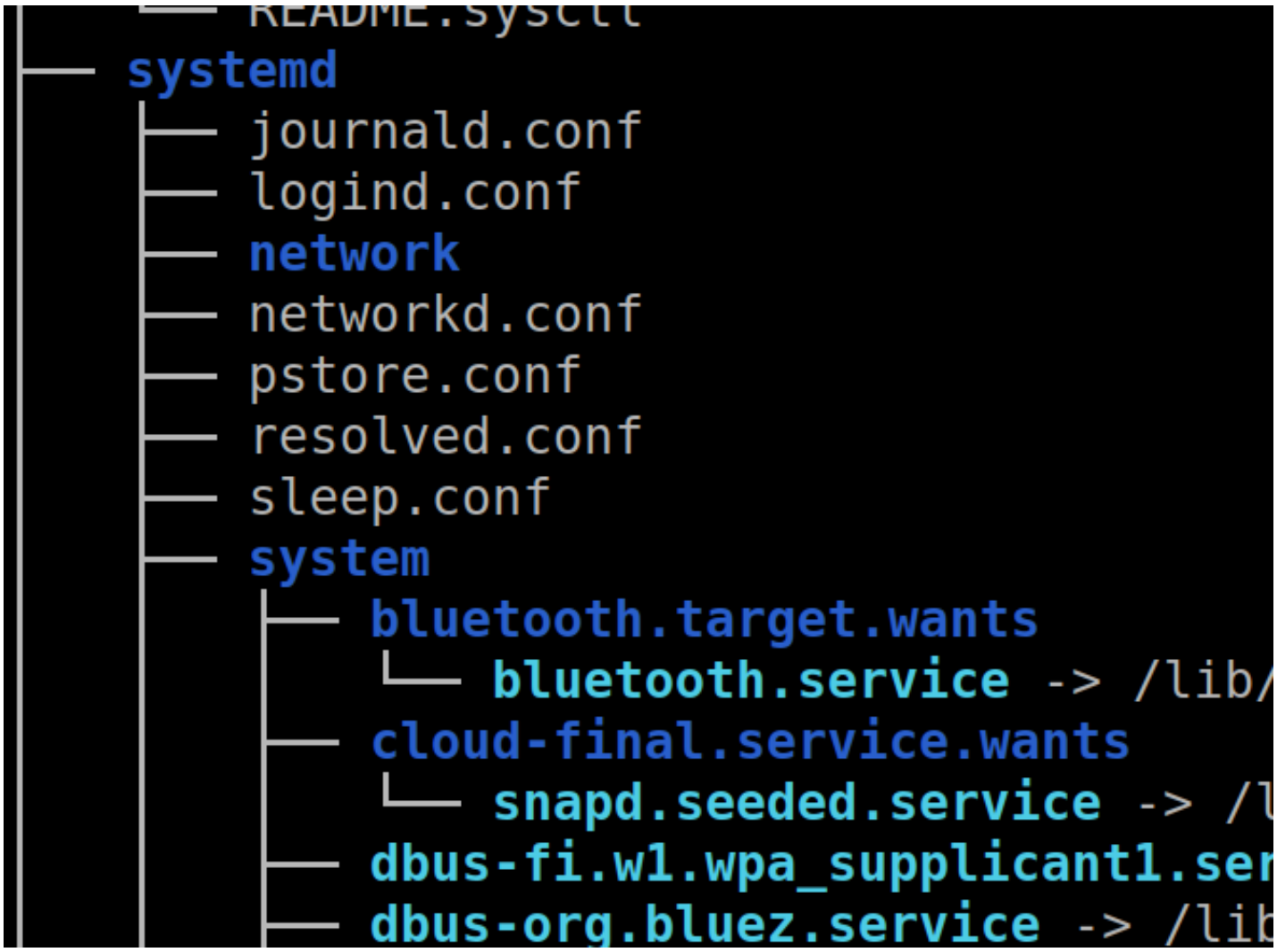
```
find . type f -exec sha256sum {} \; 2> /dev/null | grep -Ei '.asp|.js' | sort
```



Tree

`Tree` is an amazing command. Please bask in its glory. It will recursively list out folders and filders in their parent-child relationship.....or tree-branch relationship I suppose?

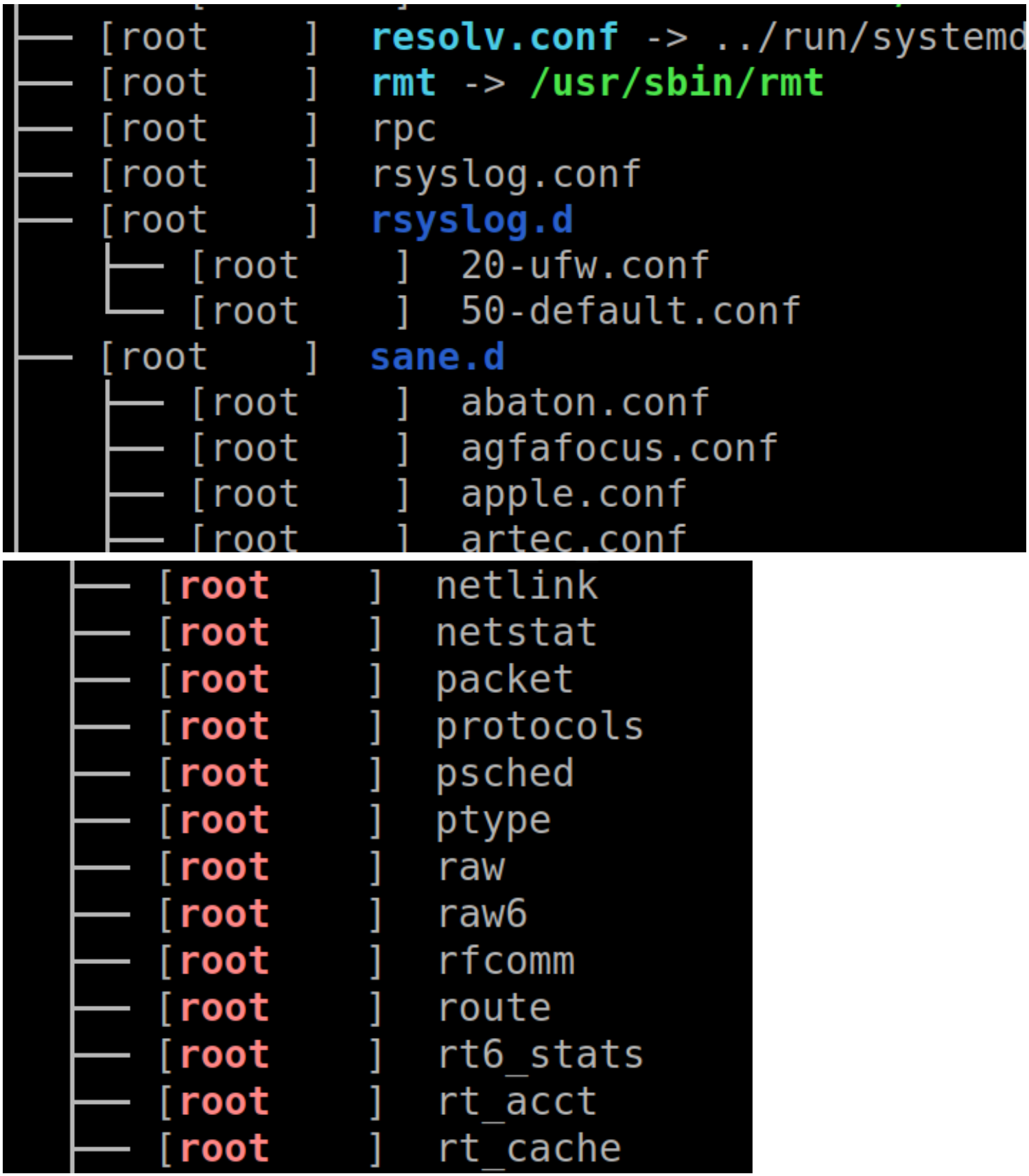
```
#install sudo apt-get install tree
tree
```



But WAIT! There's more!

Tree and show the users who own the files and directories

```
tree -u
#stack this with a grep to find a particular user you're looking for
tree -u | grep 'root'
```



If you find it a bit long and confusing to track which file belongs to what directory, this flag on tree will print the fullpath

```
tree -F
# pipe with | grep 'reports' to highlight a directory or file you are looking for
```

```
/opt/nessus/var/nessus/users # tree -f
.
└── ./scanner
    ├── ./scanner/auth
    │   ├── ./scanner/auth/admin
    │   ├── ./scanner/auth/hash
    │   └── ./scanner/auth/rules
    ├── ./scanner/policies.db
    └── ./scanner/reports
        ├── ./scanner/reports/03009591-f3bf-cbef-132a-3ccfbf6248294194baaf8cc50b83
        ├── ./scanner/reports/03009591-f3bf-cbef-132a-3ccfbf6248294194baaf8cc50b83.name
        ├── ./scanner/reports/03009591-f3bf-cbef-132a-3ccfbf6248294194baaf8cc50b83.nessus
        ├── ./scanner/reports/03009591-f3bf-cbef-132a-3ccfbf6248294194baaf8cc50b83.ts
        ├── ./scanner/reports/07c71c4d-5fe6-138c-6917-02a8e0c6e3637b09c1def5887669
        ├── ./scanner/reports/07c71c4d-5fe6-138c-6917-02a8e0c6e3637b09c1def5887669.name
        ├── ./scanner/reports/07c71c4d-5fe6-138c-6917-02a8e0c6e3637b09c1def5887669.nessus
        ├── ./scanner/reports/07c71c4d-5fe6-138c-6917-02a8e0c6e3637b09c1def5887669.ts
```

Get information about a file

`stat` is a great command to get lots of information about a file

```
stat file.txt
```

```
  File: scanner/policies.db
  Size: 10240           Blocks: 24         IO Block: 4096   regular
Device: 801h/2049d      Inode: 2364689     Links: 1
Access: (0600/-rw-------)  Uid: (    0/    root)   Gid: ( 1004/
Access: 2021-06-03 15:42:55.432366977 +0100
Modify: 2021-02-04 11:17:28.425879349 +0000
Change: 2021-05-17 22:00:18.169319109 +0100
 Birth: -
```

Files and Dates

Be careful with this, as timestamps can be manipulated and can't be trusted during an IR

This one will print the files and their corresponding timestamp

```
find . -printf "%T+ %p\n"
```

```
[03-Jun-21 15:44:44 BST] /opt
-> find . -printf "%T+ %p\n" | sort
2021-01-11+13:32:22.5462045820 ./google
2021-01-11+13:32:22.5462045820 ./google/chrome/WidevineCdm/_platfo
2021-01-12+20:35:20.7083955270 .
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/branch
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/descri
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks/
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks/
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks/
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks/
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks/
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks/
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks/
2021-01-12+20:35:20.7083955270 ./dell-bios-fan-control/.git/hooks/
```

Show all files created between two dates

I've got to be honest with you, this is one of my favourite commands. The level of granularity you can get is crazy. You can find files that have changed state by the MINUTE if you really wanted.

```
find -newerct "01 Jun 2021 18:30:00" ! -newerct "03 Jun 2021 19:00:00" -ls | sort
```

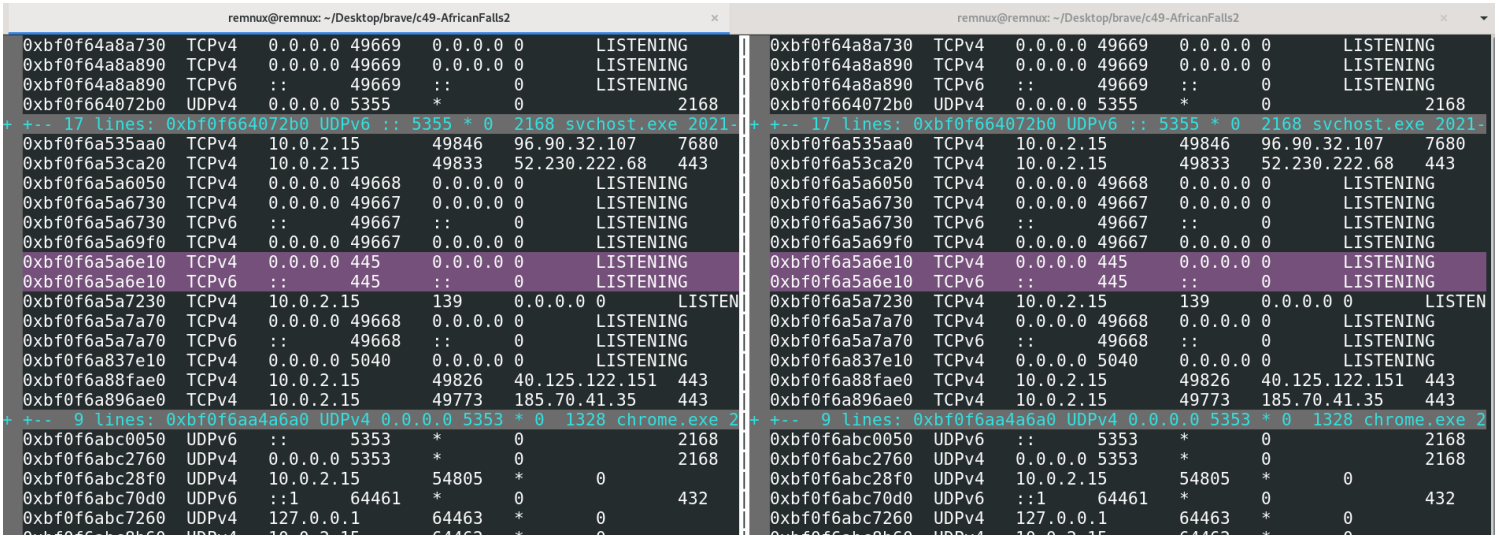
```
[03-Jun-21 15:49:04 BST] d/Downloads
-> find -newerct "01 Jun 2021 18:30:00" ! -newerct "03 Jun 2021 19:00:00" -ls | sort
 28573725      4 drwxr-xr-x   4 d        d            4096 Jun  3 11:38 .
 28573979    324 -rw-rw-r--   1 d        d          328704 Apr  9 17:05 ./update.exe
 28582111    156 -rw-rw-r--   1 d        d          159261 Jun  3 09:32 ./keylogger_
hbrain.exe.zip
 28582363    240 -rw-rw-r--   1 d        d          242688 Jul 27  2020 ./.text
 28582364     68 -rw-rw-r--   1 d        d           66560 Jul 27  2020 ./.rdata
 28582366      8 -rw-rw-r--   1 d        d            5632 Jul 27  2020 ./.data
 28582367     12 -rw-rw-r--   1 d        d           12288 Jul 27  2020 ./.reloc
 29360437      4 drwx------   3 d        d            4096 Jun  3 11:36 ./.rsrc
 29360438      4 drwx------   2 d        d            4096 Jun  3 11:36 ./.rsrc/MANI
 29360439      4 -rw-rw-r--   1 d        d             381 Apr  9 17:05 ./.rsrc/MANI
[03-Jun-21 15:49:06 BST] d/Downloads
```

Compare Files

`vimdiff` is my favourite way to compare two files

```
vimdiff file1.txt file2.txt
```

The colours highlight differences between the two. When you're done, use vim's method of exiting on both files: `:q!` . Do this twice



`diff` is the lamer, tamer version of `vimdiff` . However it does have some flags for quick analysis:

```
#are these files different yes or no?
diff -q net.txt net2.txt

#quickly show minimal differences
diff -d net.txt net2.txt
```

```
[22-Jun-21 22:49:57 BST] brave/c49-AfricanFalls2
-> diff -q net.txt net2.txt
Files net.txt and net2.txt differ
[22-Jun-21 22:50:01 BST] brave/c49-AfricanFalls2
-> diff -d net.txt net2.txt
1,2d0
< Volatility 3 Framework 1.0.1
<
32,33c30,31
< 0xbf0f6a5a6e10        TCPv4   0.0.0.0 445     0.0.0.0 0       LISTENING       4
< 0xbf0f6a5a6e10        TCPv6   ::      445     ::      0       LISTENING       4
---
> 0xbf0f6a5a6e10        TCPv4   0.0.0.0 445     0.0.0.0 0       LISTENING       4Sy
> 0xbf0f6a5a6e10        TCPv6   ::      445     ::      0       LISTENING       4Sy
55c53
< 0xbf0f6abc8cf0        UDPv6   fe80::417e:4ac4:e8ea:c3fb       64460   *       0
0
---
> 0xbf0f6abc8cf0        UDPv6   fe80::417e:4ac4:e8ea:c3fb       64460   *       043
61,62c59,60
< 0xbf0f6bfb7890        UDPv4   10.0.2.15       138     *       0               4
< 0xbf0f6bfb9640        UDPv4   10.0.2.15       137     *       0               4
```

Bash Tips

▶ section contents

Fixing Mistakes

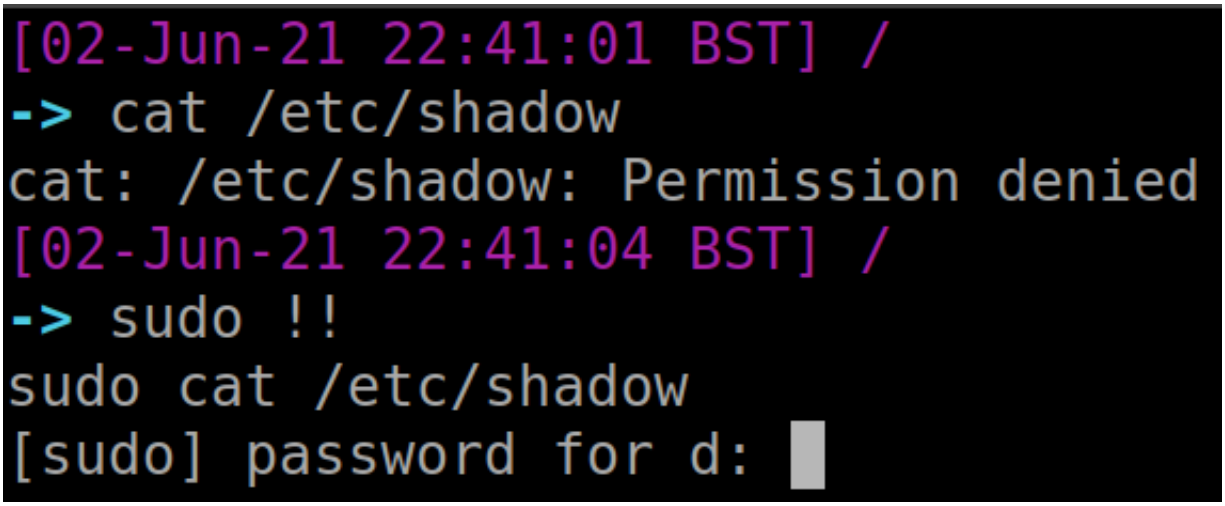
We all make mistakes, don't worry. Bash forgives you

Forget to run as sudo?

We've all done it mate. Luckily, `!!` has your back. The exclamation mark is a history related bash thing.

Using two exclamations, we can return our previous command. By prefixing `sudo` we are bringing our command back but running it as sudo

```
#for testing, fuck up a command that needed sudo but you forgot
cat /etc/shadow
# fix it!
sudo !!
```

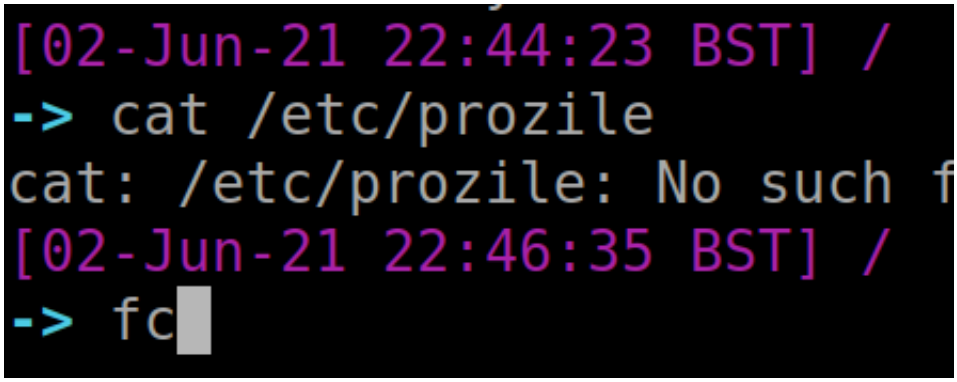


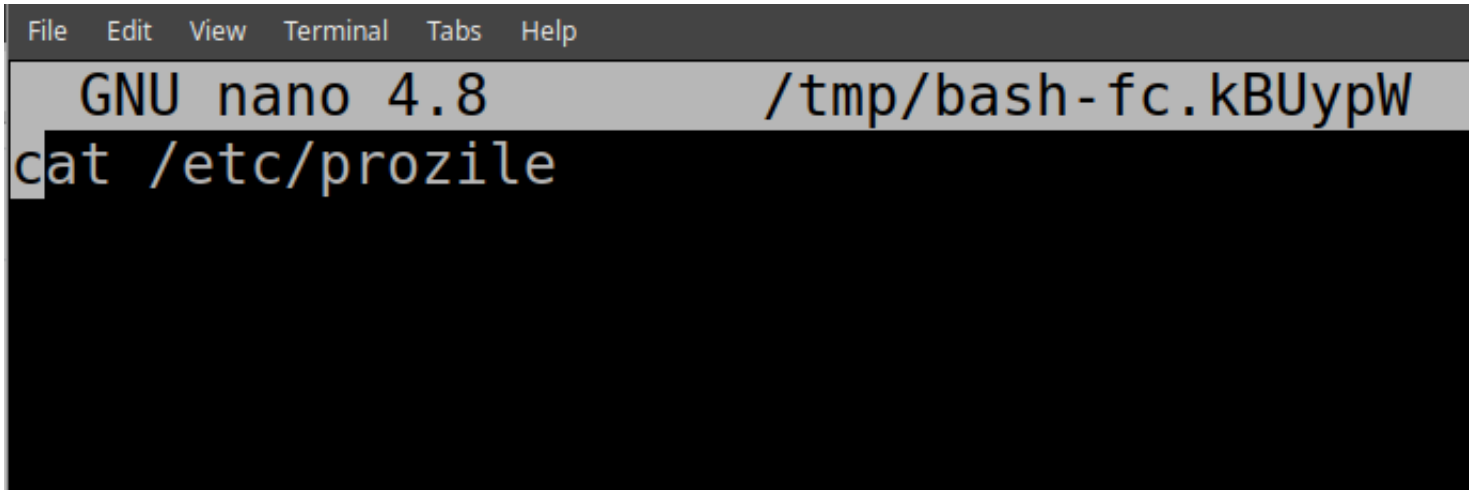
Typos in a big old one liner?

The `fc` command is interesting. It gets what was just run in terminal, and puts it in a text editor environment. You can the ammend whatever mistakes you may have made. Then if you save and exit, it will execute your newly ammended command

```
##messed up command
cat /etc/prozile
#fix it
fc
```

```
#then save and exit
```





Re-run a command in History

If you had a beautiful command you ran ages ago, but can't remember it, you can utilise `history` . But don't copy and paste like a chump.

Instead, utilise exclamation marks and the corresponding number entry for your command in the history file. This is highlighted in red below

```
#bring up your History
history
#pick a command you want to re-run.
# now put one exclamation mark, and the corresponding number for the command you
!12
```

```
 3591  02/06/21 22:47:08 cat /etc/profile
 3592  02/06/21 22:48:01 eclear
 3593  02/06/21 22:48:02 clear
 3594  02/06/21 22:48:13 echo "bringing up old shit"
 3595  02/06/21 22:48:16 history
[02-Jun-21 22:48:16 BST] /
-> !3594
echo "bringing up old shit"
bringing up old shit
```

MacOS

▶ section contents

Reading .plist files

Correct way to just read a plist is `plutil -p` but there are multiple different methods so do whatever, I'm not the plist police

```
[2022-May-24 12:04:29 BST] Downloads/Collected_Data
[🔍 -> sudo plutil -p /var/db/locationd/clients.plist | head -n 10
{
  "com.apple.locationd.bundle-/System/Library/LocationBundles/Routine.bundle" => {
    "BundleId" => "com.apple.locationd.bundle-/System/Library/LocationBundles/Routine.bundle"
    "BundlePath" => "/System/Library/LocationBundles/Routine.bundle"
    "Registered" => ""
    "Whitelisted" => 0
  }
  "com.apple.locationd.bundle-/System/Library/LocationBundles/WifiCalling.bundle" => {
    "Authorized" => 0
    "BundleId" => "com.apple.locationd.bundle-/System/Library/LocationBundles/WifiCalling.bundle"
```

If the plist is in binary format, you can convert it to a more readable xml: `plutil -convert xml1 <path_to_binary_plist>`

Quarantine Events

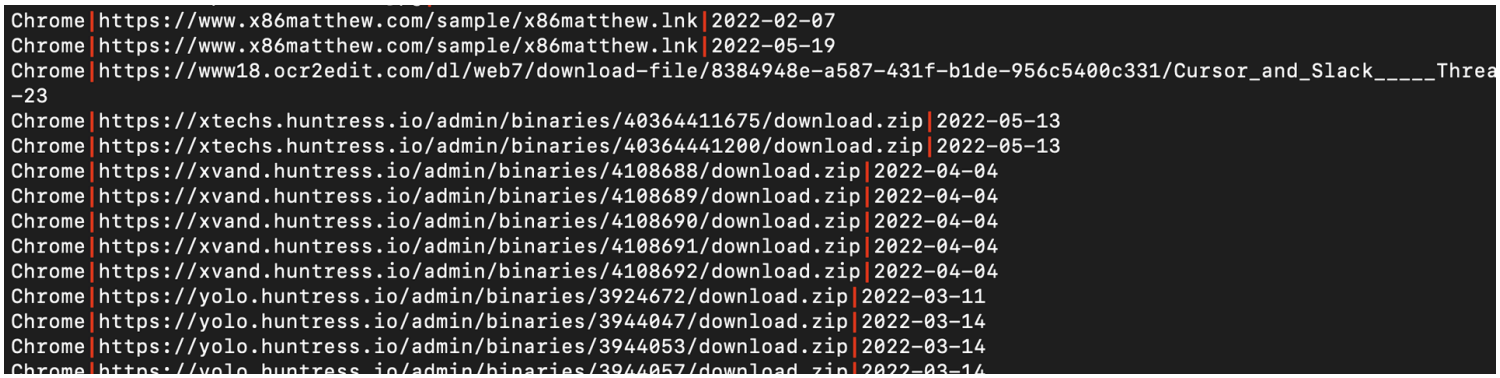
Files downloaded from the internet

The db you want to retrieve will be located here with a corresponding username: `/Users/*/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV2`

Here's a dope one-liner that organises the application that did the downloading, the link to

download, and then the date it was downloaded, via sqlite

```
sqlite3 /Users/dray/Library/Preferences/com.apple.LaunchServices.QuarantineEvents
'select LSQuarantineAgentName, LSQuarantineDataURLString, date(LSQuarantineTimeSt
| sort -u | grep '|' --color
```

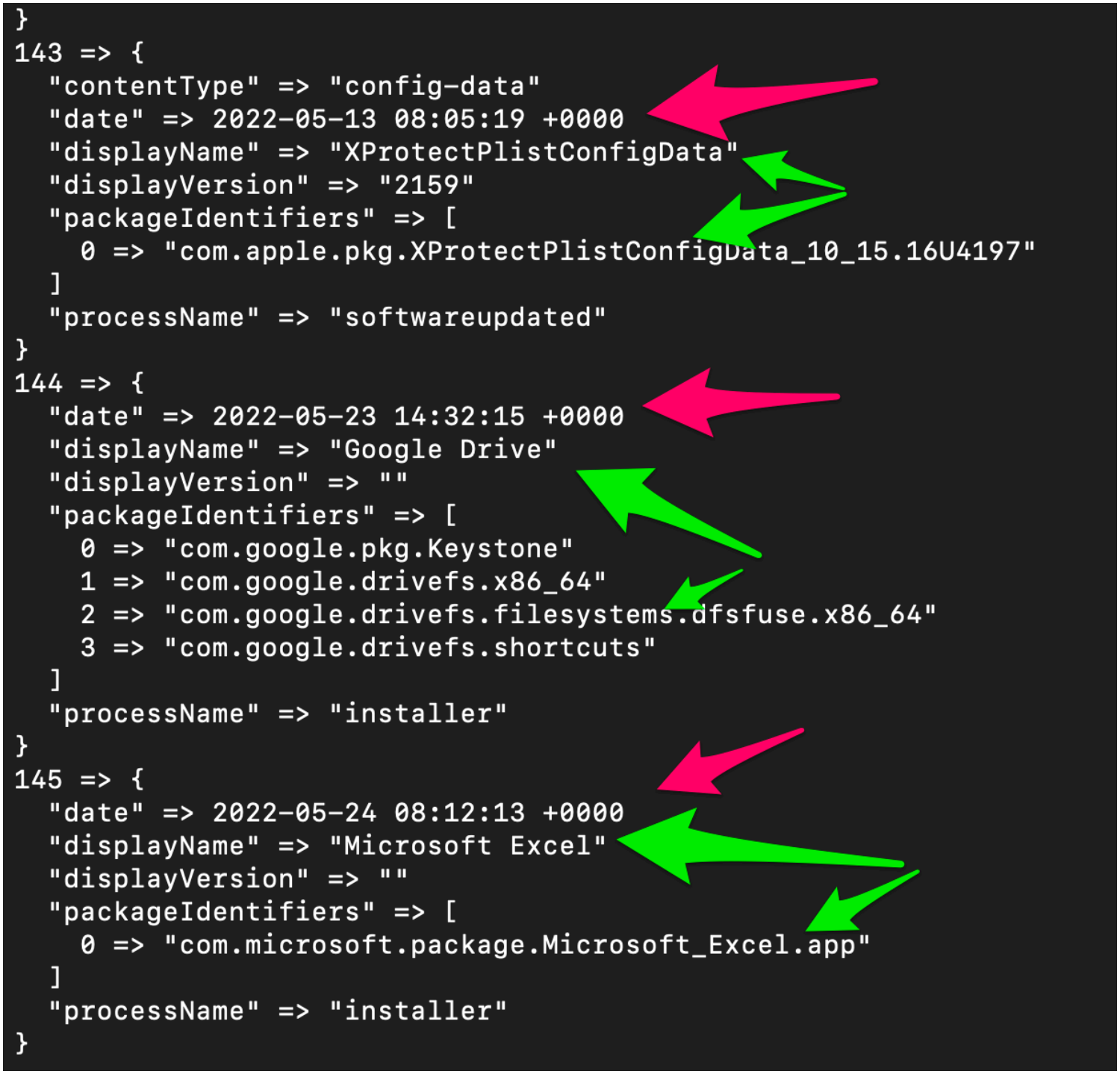


Install History

Find installed applications and the time they were installed from : `/Library/Receipts/InstallHistory.plist`

Annoyingly doesn't show corresponding user ? However, it does auto sort the list by datetime which is helpful

```
plutil -p /Library/Receipts/InstallHistory.plist
```



Location Tracking

Some malware can do creeper stuff and leverage location tracking Things you see here offer an insight into the programs and services allowed to leverage location stuff on mac

```
#plain read
sudo plutil -p /var/db/locationd/clients.plist

#highlight the path of these applications
sudo plutil -p /var/db/locationd/clients.plist | ack --passthru 'BundlePath'
# or sudo plutil -p /var/db/locationd/clients.plist | grep 'BundlePath'
```

```
[2022-May-24 12:11:51 BST] Downloads/Collected_Data
🔍 -> sudo plutil -p /var/db/locationd/clients.plist | ack --passthru 'BundlePath'
{
  "com.apple.locationd.bundle-/System/Library/LocationBundles/Routine.bundle" => {
    "BundleId" => "com.apple.locationd.bundle-/System/Library/LocationBundles/Routine.bundle"
    "BundlePath" => "/System/Library/LocationBundles/Routine.bundle"
    "Registered" => ""
    "Whitelisted" => 0
  }
  "com.apple.locationd.bundle-/System/Library/LocationBundles/WifiCalling.bundle" => {
    "Authorized" => 0
    "BundleId" => "com.apple.locationd.bundle-/System/Library/LocationBundles/WifiCalling.bundle"
    "BundlePath" => "/System/Library/LocationBundles/WifiCalling.bundle"
    "Registered" => ""
    "Whitelisted" => 0
  }
  "com.apple.locationd.bundle-/System/Library/PrivateFrameworks/CoreParsec.framework" => {
    "BundleId" => "com.apple.locationd.bundle-/System/Library/PrivateFrameworks/CoreParsec.framework"
    "BundlePath" => "/System/Library/PrivateFrameworks/CoreParsec.framework"
    "Registered" => ""
  }
  "com.apple.locationd.bundle-/System/Library/PrivateFrameworks/FindMyDevice.framework" => {
    "SLC" => {
      "distanceThreshold" => 500
      "powerBudget" => 0
    }
  }
  "com.apple.locationd.bundle-/System/Library/PrivateFrameworks/HomeKitDaemon.framework" => {
    "Authorized" => 0
    "BundleId" => "com.apple.locationd.bundle-/System/Library/PrivateFrameworks/HomeKitDaemon.framework"
    "BundlePath" => "/System/Library/PrivateFrameworks/HomeKitDaemon.framework"
    "Registered" => ""
  }
  "com.apple.locationd.executable-" => {
  }
  "com.apple.sharingd" => {
  }
  "com.google.Chrome" => {
    "BundleId" => "com.google.Chrome"
    "BundlePath" => "/Applications/Google Chrome.app"
    "Registered" => ""
```

```
[2022-May-24 12:12:24 BST] Downloads/Collected_Data
🔍 -> sudo plutil -p /var/db/locationd/clients.plist | grep 'BundlePath'
    "BundlePath" => "/System/Library/LocationBundles/Routine.bundle"
    "BundlePath" => "/System/Library/LocationBundles/WifiCalling.bundle"
    "BundlePath" => "/System/Library/PrivateFrameworks/CoreParsec.framework"
    "BundlePath" => "/System/Library/PrivateFrameworks/HomeKitDaemon.framework"
    "BundlePath" => "/Applications/Google Chrome.app"
    "BundlePath" => "/Applications/Slack.app"
    "BundlePath" => "/Applications/Obsidian.app"
[2022-May-24 12:12:26 BST] Downloads/Collected_Data
```

Most Recently Used (MRU)

Does what it says.....identifies stuff most recently used

The directory with all the good stuff is here

```
/Users/*/Library/Application Support/com.apple.sharedfilelist/
```

```
#full path to this stuff
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
/Users/*/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedF
```

```
[2022-May-24 13:22:33 BST] Downloads/Collected_Data
-> ls -lash '/users/Dray/Library/Application Support/com.apple.sharedfilelist/'
total 136
 0 drwxr-xr-x  11 dray  staff   352B 24 May 10:58 .
 0 drwx------+ 39 dray  staff   1.2K 12 Apr 09:14 ..
 0 drwxr-xr-x  12 dray  staff   384B 24 May 09:57 com.apple.LSSharedFileList.ApplicationRecentDocuments
16 -rw-r--r--   1 dray  staff   4.8K 19 Jan 17:18 com.apple.LSSharedFileList.FavoriteItems.sfl2
32 -rw-r--r--   1 dray  staff    12K 19 Apr 09:16 com.apple.LSSharedFileList.FavoriteVolumes.sfl2
16 -rw-r--r--   1 dray  staff   4.2K 19 Jan 10:59 com.apple.LSSharedFileList.ProjectsItems.sfl2
24 -rw-r--r--   1 dray  staff   8.1K 24 May 10:58 com.apple.LSSharedFileList.RecentApplications.sfl2
24 -rw-r--r--   1 dray  staff   8.9K 24 May 09:57 com.apple.LSSharedFileList.RecentDocuments.sfl2
 8 -rw-r--r--   1 dray  staff   1.2K  5 Apr 21:54 com.apple.LSSharedFileList.RecentHosts.sfl2
 8 -rw-r--r--   1 dray  staff   2.7K  1 Apr 10:50 com.apple.LSSharedFileList.RecentServers.sfl2
 8 -rw-r--r--   1 dray  staff   322B 19 Jan 17:19 com.apple.LSSharedFileList.iCloudItems.sfl2
[2022-May-24 13:22:38 BST] Downloads/Collected_Data
->
```

Another useful subdirectory here containing stuff relevant to recent applicatioons

```
/Users/users/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSha
```

```
[2022-May-24 13:25:29 BST] Downloads/Collected_Data
-> ls -lash '/users/Dray/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedFileList.ApplicationRecentDocuments/'
total 144
 0 drwxr-xr-x  12 dray  staff   384B 24 May 09:57 .
 0 drwxr-xr-x  11 dray  staff   352B 24 May 10:58 ..
 8 -rw-r--r--   1 dray  staff   789B 19 Jan 12:58 com.addigy.macmanagehelper.sfl2
24 -rw-r--r--   1 dray  staff   9.4K 19 May 11:46 com.apple.console.sfl2
24 -rw-r--r--   1 dray  staff   8.9K 19 May 18:44 com.apple.preview.sfl2
16 -rw-r--r--   1 dray  staff   4.8K 23 May 16:55 com.apple.quicktimeplayerx.sfl2
 8 -rw-r--r--   1 dray  staff   618B 19 Jan 17:48 com.apple.storeuid.sfl2
24 -rw-r--r--   1 dray  staff   9.1K 23 May 16:46 com.apple.textedit.sfl2
 8 -rw-r--r--   1 dray  staff   742B  5 May 11:08 com.microsoft.excel.sfl2
 8 -rw-r--r--   1 dray  staff   781B  7 Feb 20:52 com.microsoft.word.sfl2
 8 -rw-r--r--   1 dray  staff   796B 20 Jan 09:14 com.tinyspeck.slackmacgap.sfl2
16 -rw-r--r--   1 dray  staff   7.5K 24 May 09:57 com.vmware.fusion.sfl2
[2022-May-24 13:25:31 BST] Downloads/Collected_Data
```

There are legitimate ways to parse whats going on here……but that just ain't me chief - I strings these bad boys

```
[2022-May-24 13:24:24 BST] Downloads/Collected_Data
🔍 -> strings '/users/Dray/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedFileList.RecentServers.sfl2' | sort -u
 !"#$
$4B5807A1-59B5-4CC8-80B0-BD1EC812EC0D
$9636EB00-9CF0-423F-ABCD-AE39941E6330
$com.apple.LSSharedFileList.MaxAmount
&'()*
&HIJK
,-./0126<=>?MSTUVWX[abU$null
.com.apple.LSSharedFileList.OverrideIcon.OSTypeTsrvr_
/Volumes/Library_of_Alexandria
789:Z$classnameX$classes\NSDictionary
78YZWNSArray
9;XNSObject_
A5B0DD28-CE8B-46AB-B874-1194A3B5F12C
CustomItemPropertiesTNameXBookmarkTuuid
Library_of_Alexandria
Library_of_AlexandriaO
Macintosh HD
NSKeyedArchiver
Troot
UitemsZproperties
Volumes
WNS.keysZNS.objectsV$class
X$versionY$archiverT$topX$objects
Zvisibility_
book
bplist00
file:///

file:///Volumes/Library_of_Alexandria/
smb://pi@192.168.1.49/Library_of_Alexandria
smb://pi@RASPBERRYPI._smb._tcp.local/Library_of_Alexandria
```

```
[2022-May-24 13:30:11 BST] Downloads/Collected_Data
🔍 -> strings '/users/Dray/Library/Application Support/com.apple.sharedfilelist/com.apple.LSSharedFileList.FavoriteVolumes.sfl2' |sort -u | tail -n 20
]Google ChromeO
ansible
book
bplist00
com-apple-sfl://com.apple.LSSharedFileList.IsComputer
com-apple-sfl://com.apple.LSSharedFileList.IsICloudDrive$
com-apple-sfl://com.apple.LSSharedFileList.IsRemoteDisc
dray
file:///
file:///

file:///Volumes/Google%20Chrome/
file:///Volumes/Install%20Google%20Drive/
file:///Volumes/Obsidian%200.13.19-universal/
file:///Volumes/VMware%20Fusion/
file:///Volumes/flameshot/
flameshot
flameshot.dmg
googlechrome.dmg
packages$
```

Audit Logs

`praudit` command line tool will let you read the audit logs in `/private/var/audit/`

```
sh-3.2# praudit /private/var/audit/current | head -n 40
header,138,11,SecSrvr AuthEngine,0,Mon May 16 16:55:02 2022, + 99 msec
subject,-1,root,wheel,root,wheel,2791,100000,8683454,0.0.0.0
text,begin evaluation
return,success,0
identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705
trailer,138
header,162,11,SecSrvr AuthEngine,0,Mon May 16 16:55:02 2022, + 99 msec
subject,-1,root,wheel,root,wheel,2791,100000,8683454,0.0.0.0
text,system.preferences
text,system.preferences
return,success,0
identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705
trailer,162
header,276,11,SecSrvr AuthEngine,0,Mon May 16 16:55:02 2022, + 99 msec
subject,-1,root,wheel,root,wheel,2791,100000,8683454,0.0.0.0
text,system.preferences
text,client /System/Library/PrivateFrameworks/SystemAdministration.framework/XPCServices/writeconfig.xpc
text,creator /usr/sbin/systemsetup
return,success,0
identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705
```

Play around with the different printable formats of `praudit`

```
sh-3.2# praudit -l /private/var/audit/current  | head -n 5
header,138,11,SecSrvr AuthEngine,0,Mon May 16 16:55:02 2022, + 99 msec,subject,-1,root,wheel,root,wheel,2791,100000,8683454,0.0.0.0,text,begin evaluation,return,success,0,identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705,trailer,138,
header,162,11,SecSrvr AuthEngine,0,Mon May 16 16:55:02 2022, + 99 msec,subject,-1,root,wheel,root,wheel,2791,100000,8683454,0.0.0.0,text,system.preferences,text,system.preferences,return,success,0,identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705,trailer,162,
header,276,11,SecSrvr AuthEngine,0,Mon May 16 16:55:02 2022, + 99 msec,subject,-1,root,wheel,root,wheel,2791,100000,8683454,0.0.0.0,text,system.preferences,text,client /System/Library/PrivateFrameworks/SystemAdministration.framework/XPCServices/writeconfig.xpc,text,creator /usr/sbin/systemsetup,return,success,0,identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705,trailer,276,
header,136,11,SecSrvr AuthEngine,0,Mon May 16 16:55:02 2022, + 99 msec,subject,-1,root,wheel,root,wheel,2791,100000,8683454,0.0.0.0,text,end evaluation,return,success,0,identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705,trailer,136,
header,138,11,SecSrvr AuthEngine,0,Mon May 16 16:55:03 2022, + 454 msec,subject,-1,root,wheel,root,wheel,2852,100000,8683578,0.0.0.0,text,begin evaluation,return,success,0,identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705,trailer,138,
sh-3.2# praudit -x /private/var/audit/current  | head -n 5
<?xml version='1.0' encoding='UTF-8'?>
<audit>
<record version="11" event="SecSrvr AuthEngine" modifier="0" time="Mon May 16 16:55:02 2022" msec=" + 99 msec" >
<subject audit-uid="-1" uid="root" gid="wheel" ruid="root" rgid="wheel" pid="2791" sid="100000" tid="8683454 0.0.0.0" />
<text>begin evaluation</text>
sh-3.2# praudit -s /private/var/audit/current  | head -n 5
header,138,11,AUE_ssauthorize,0,Mon May 16 16:55:02 2022, + 99 msec
subject,-1,root,wheel,root,wheel,2791,100000,8683454,0.0.0.0
text,begin evaluation
return,success,0
identity,1,com.apple.authd,complete,,complete,0x5e55013b2fd96b6a51075985102fee591fd72705
sh-3.2#
```

And then leverage `auditreduce` to look for specific activity (man page).

Examples

What was the user dray up to on 13th May 2022: `auditreduce -d 20220513 -u dray /var/audit/* | praudit`

```
sh-3.2# auditreduce -d 20220513 -u dray  /var/audit/* | praudit | head -n 10
header,197,11,user authentication,0,Fri May 13 09:04:02 2022, + 840 msec
subject,dray,dray,staff,dray,staff,5269,100003,15883,0.0.0.0
text,Verify password for record type Users 'dray' node '/Local/Default'
return,failure: Unknown error: 255,5000
identity,1,com.apple.opendirectoryd,complete,,complete,0xf6f10721b26a944984b27b8f919e3cea3eb7960a
trailer,197
header,197,11,user authentication,0,Fri May 13 09:04:03 2022, + 88 msec
subject,dray,dray,staff,dray,staff,5269,100003,15883,0.0.0.0
text,Verify password for record type Users 'dray' node '/Local/Default'
return,failure: Unknown error: 255,5000
```

Show user logins and outs auditreduce -c lo /var/audit/* | praudit

```
sh-3.2# auditreduce -c lo /var/audit/* | praudit | head -n 10
header,122,11,logout - local,0,Fri May  6 20:55:28 2022, + 747 msec
subject_ex,dray,root,staff,dray,staff,5270,5270,268435456,0.0.0.0
return,success,0
identity,1,com.apple.login,complete,,complete,0x70fa05694023773f73e50cdd1850e0c852144e23
trailer,122
header,122,11,logout - local,0,Fri May  6 20:55:31 2022, + 196 msec
subject_ex,dray,root,staff,dray,staff,10342,10342,268435457,0.0.0.0
return,success,0
identity,1,com.apple.login,complete,,complete,0x70fa05694023773f73e50cdd1850e0c852144e23
trailer,122
```

What happened between two dates: auditreduce /var/audit/* -a 20220401 -b 20220501 | praudit

Command line history

A couple places to retrieve command line activity

```
#will be zsh or bash
```

```
/Users/*/.zsh_sessions/*
/private/var/root/.bash_history
/Users/*/.zsh_history
```

```
[2022-May-24 13:59:14 BST] Downloads/Collected_Data
🔍 -> ls /Users/dray/.zsh_sessions/*
/Users/dray/.zsh_sessions/1926E51B-789D-4E7C-9802-447F86488E72.history
/Users/dray/.zsh_sessions/1926E51B-789D-4E7C-9802-447F86488E72.session
/Users/dray/.zsh_sessions/1FFA66CD-C39C-4A06-9A0D-23F9D4FDF393.history
/Users/dray/.zsh_sessions/1FFA66CD-C39C-4A06-9A0D-23F9D4FDF393.session
/Users/dray/.zsh_sessions/31EA6B96-F3CD-4D76-A514-A7DFB4C72195.historynew
/Users/dray/.zsh_sessions/B02C2BAF-8E8A-4726-A478-994C7E3EE1EC.historynew
/Users/dray/.zsh_sessions/C7596F48-54EC-4796-A80E-29F74118FB6C.history
/Users/dray/.zsh_sessions/C7596F48-54EC-4796-A80E-29F74118FB6C.session
/Users/dray/.zsh_sessions/F7A8DEE8-C2BC-489A-9162-11E88A837A8E.history
/Users/dray/.zsh_sessions/F7A8DEE8-C2BC-489A-9162-11E88A837A8E.session
/Users/dray/.zsh_sessions/_expiration_check_timestamp
```

```
[2022-May-24 14:03:11 BST] Downloads/Collected_Data
🔍 -> sudo ls /private/var/root/
.CFUserTextEncoding     .bash_history           .forward                Library
[2022-May-24 14:03:15 BST] Downloads/Collected_Data
🔍 -> sudo cat /private/var/root/.bash_history | head -n5
pwd
cd /Users/dray/Desktop/louis-durrant-kde-1080.jpg .
cp /Users/dray/Desktop/louis-durrant-kde-1080.jpg .
cp /Users/dray/Desktop/louis-durrant-kde-1080.jpg .
nettop -m
```

WHOMST is in the Admin group

Identify if someone has added themselves to the admin group

```
plutil -p /private/var/db/dslocal/nodes/Default/groups/admin.plist
```

```
[2022-May-24 14:06:32 BST] ~
[🔍 -> sudo plutil -p /private/var/db/dslocal/nodes/Default/groups/admin.plist
{
  "generateduid" => [
    0 => "ABCDEFAB-CDEF-ABCD-EFAB-CDEF00000050"
  ]
  "gid" => [
    0 => "80"
  ]
  "groupmembers" => [
    0 => "FFFFEEEE-DDDD-CCCC-BBBB-AAAA00000000"
    1 => "B59E9F32-DCF5-4340-9A24-8A58867B5087"
    2 => "79E069CC-9C62-45A8-917A-A0EC90A5EFC7"
  ]
  "name" => [
    0 => "admin"
    1 => "BUILTIN\Administrators"
  ]
  "passwd" => [
    0 => "*"
  ]
  "realname" => [
    0 => "Administrators"
  ]
  "smb_sid" => [
    0 => "S-1-5-32-544"
  ]
  "users" => [
    0 => "root"
    1 => "dray"
    2 => "AddigySSH"
  ]
}
```

Persistence locations

Not complete, just some easy low hanging fruit to check.

Can get a more complete list here

```
# start up / login items
/var/db/com.apple.xpc.launchd/disabled.*.plist
/System/Library/StartupItems
/Users/*/Library/Application Support/com.apple.backgroundtaskmanagementagent/back
/var/db/launchd.db/com.apple.launchd/*

# scripts
/Users/*/Library/Preferences/com.apple.loginwindow.plist
/etc/periodic/[daily, weekly, monthly]

# cronjobs / like scheduled tasks
/private/var/at/tabs/
/usr/lib/cron/jobs/
```

```
# system extensions
/Library/SystemExtensions/

# loads of places for annoying persistence amongst daemons
/System/Library/LaunchDaemons/*.plist
/System/Library/LaunchAgents/*.plist
/Library/LaunchDaemons/*.plist
/Library/LaunchAgents/*.plist
/Users/*/Library/LaunchAgents/*.plist
```

```
[2022-May-24 14:12:30 BST] ~
🔍 -> sudo ls /private/var/at/tabs
dray
[2022-May-24 14:12:34 BST] ~
🔍 -> sudo cat /private/var/at/tabs/dray
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.kf2YqyYzUH installed on Tue Feb  8 20:40:09 2022)
# (Cron version -- $FreeBSD: src/usr.sbin/cron/crontab/crontab.c,v 1.24 2006/09/03 17:52:19 ru Exp $)
@reboot sudo spctl --master-disable
[2022-May-24 14:12:43 BST] ~
🔍 ->
```

Essentially a Scheduled Task, like in Windows

```
[2022-May-24 14:10:24 BST] ~
🔍 -> cat /var/db/com.apple.xpc.launchd/disabled.*.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.if.Amphetamine</key>
        <false/>
        <key>com.apple.ManagedClientAgent.enrollagent</key>
        <true/>
        <key>com.apple.Siri.agent</key>
        <true/>
        <key>com.google.keystone.system.agent</key>
        <false/>
        <key>com.microsoft.update.agent</key>
        <false/>
        <key>com.apple.FolderActionsDispatcher</key>
        <true/>
        <key>J8RPQ294UB.com.skitch.SkitchHelper</key>
        <false/>
        <key>com.google.keystone.system.xpcservice</key>
        <false/>
        <key>com.apple.appleseed.seedusaged.postinstall</key>
        <true/>
        <key>com.apple.ScriptMenuApp</key>
        <true/>
</dict>
</plist>
[2022-May-24 14:10:27 BST] ~
```

Transparency, Consent, and Control (TCC)

The TCC db (Transparency, Consent, and Control) offers insight when some applications have made system changes. There are at least two TCC databases on the system - one per user, and one root.

```
/Library/Application Support/com.apple.TCC/TCC.db
/Users/*/Library/Application Support/com.apple.TCC/TCC.db
```