

Plugins documentation:

Aviel Zohar

Introduction

This pdf contains documentation on 3 different files – volexp.py, pfn.py and winobj.py. each of them contains multiple plugins (except winobj.py).

1. [Vol3xp.py](#) contains:

- [VolExp](#)
- [StructAnalyzer](#)
- [WinObjGUI](#)
- [FileScanGUI](#)

2. [PFN.py](#) contains:

- [P2V](#)
- [PFNInfo](#)
- [RAMMap](#)

3. [WinObj.py](#) contains:

- [WinObj](#)

:Plugins documentation	1
Introduction.....	1
VolExp.py	5
VolExp – Volatility Explorer.....	5
Introduction.....	5
Motivation	5
Startup Arguments	5
Some Research.....	6
Abilities.....	8
General UI	8
Functionality	10
• Process right click	11
• PE File	13
• Integrate with FileScanGui and WinObjGui.....	14
• Registry Viewer	15
• Integrate With Struct Analyzer.....	16
• Integrate With Other Plugins	16
Plugin Flow:	20
Struct Analyzer.....	21
Introduction.....	21
Motivation	21
Arguments	21
Some Research.....	22
Abilities.....	22
Struct Analyzer Flow (example).....	26
WinObjGui	29
Introduction.....	29
Motivation	29
Startup Arguments	29
Some Research.....	29
Abilities.....	31

• Integrate with Struct Analyzer.....	31
FileScanGui	32
Introduction.....	32
Motivation	32
Startup Arguments	32
Some Research.....	32
Abilities.....	34
• Integrate with Struct Analyzer.....	34
• Dump File/Directory	34
Pfn.py.....	36
Motivation	36
P2V – Physical to Virtual	37
Intruduction.....	37
Startup Arguments:	37
Some research.....	37
Abilities.....	40
PFNInfo – page frame number information.....	41
Introduction.....	41
Motivation	41
Startup Arguments	41
Some research.....	41
Abilities.....	42
RAMMap – Random Access Memory Mapping	43
Introduction.....	43
Startup Arguments	43
Some Research.....	44
Abilities.....	45
WinObj.py.....	46
WinObj – windows (kernel) objects.....	46
Introduction.....	46
Motivation	46

Startup Arguments:	46
Some Research.....	46
Abilities.....	48
Using the tools.....	50
Instructions.....	50
Sahar.vmem.....	50
KSLSample.vmem	60
Closing Statements.....	63
References.....	64

VolExp.py

VolExp – Volatility Explorer

Introduction

This program allows the user to access a Memory Dump. It can also function as a plugin to the Volatility Framework (<https://github.com/volatilityfoundation/volatility>).

This program functions similarly to Process Explorer/Hacker, but additionally it allows the user access to a Memory Dump. This program can run from Windows, Linux and MacOS machines, but can only use Windows memory images.

Motivation

To this day, memory forensics analysts most frequently use CLI-based tools for memory analysis. For the sake of comfort, ease of navigation, and faster, more efficient analysis, I decided to create a tool that we're all familiar with from SysInternals [ProcExp/Process Hacker] as a GUI-based plugin for Volatility.

Additionally, I created it for my own enhanced understanding of the Volatility Framework.

This tool uses multithreading and multiprocessing to run different plugins (svcsan, hivelist, getsids, pslist, handles) and different calculations for automation and efficiency.

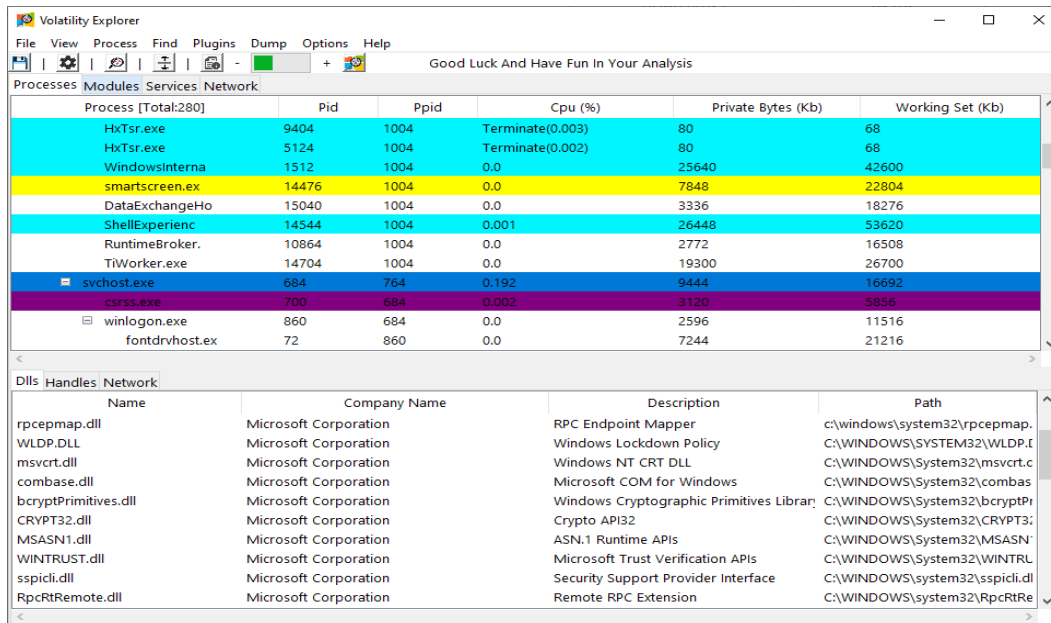
For the functionality of this plugin, I have written several additional plugins to the framework, very similar to volatility2 (Converted to volatility3 for compatibility) – getsids, getservicesids, privileges and envvars and also added to the windows\symbols\extensions__init__.py file of the framework some functions to help me create these plugins.

Startup Arguments

This plugin accepts the following arguments:

- --DUMP-DIR: Directory to dump files.

- --SAVED-FILE: File that contains a previously saved run (you can save your progress at any time, similar to ida .idb file).
- --API-KEY: Virus Total API key to integrate with Virus Total.



The screenshot shows the Volatility Explorer application. The top menu bar includes File, View, Process, Find, Plugins, Dump, Options, and Help. Below the menu is a toolbar with various icons. The main window is divided into two panes. The top pane, titled 'Processes [Total:280]', displays a table of running processes. The bottom pane, titled 'DLLs', displays a table of loaded DLLs.

Process [Total:280]	Pid	Ppid	Cpu (%)	Private Bytes (Kb)	Working Set (Kb)
HxTsr.exe	9404	1004	Terminate(0.003)	80	68
HxTsr.exe	5124	1004	Terminate(0.002)	80	68
WindowsInterna	1512	1004	0.0	25640	42600
smartscreen.ex	14476	1004	0.0	7848	22804
DataExchangeHo	15040	1004	0.0	3336	18276
ShellExperienc	14544	1004	0.001	26448	53620
RuntimeBroker.	10864	1004	0.0	2772	16508
TiWorker.exe	14704	1004	0.0	19300	26700
svchost.exe	684	764	0.192	9444	16692
csrss.exe	700	684	0.002	3120	5856
winlogon.exe	860	684	0.0	2596	11516
fontdrvhost.ex	72	860	0.0	7244	21216

Name	Company Name	Description	Path
rpcepmap.dll	Microsoft Corporation	RPC Endpoint Mapper	c:\windows\system32\vpcepmap.
WLDP.DLL	Microsoft Corporation	Windows Lockdown Policy	C:\WINDOWS\SYSTEM32\WLDP.L
msvcrt.dll	Microsoft Corporation	Windows NT CRT DLL	C:\WINDOWS\System32\msvcrt.c
combase.dll	Microsoft Corporation	Microsoft COM for Windows	C:\WINDOWS\System32\combas
bcryptPrimitives.dll	Microsoft Corporation	Windows Cryptographic Primitives Libran	C:\WINDOWS\System32\bcryptPr
CRYPT32.dll	Microsoft Corporation	Crypto API32	C:\WINDOWS\System32\CRYPT32
MSASN1.dll	Microsoft Corporation	ASN.1 Runtime APIs	C:\WINDOWS\System32\MSASN1
WINTRUST.dll	Microsoft Corporation	Microsoft Trust Verification APIs	C:\WINDOWS\System32\WINTRL
sspicli.dll	Microsoft Corporation	Security Support Provider Interface	C:\WINDOWS\system32\sspicli.dl
RpcRtRemote.dll	Microsoft Corporation	Remote RPC Extension	C:\WINDOWS\system32\RpcRtRe

Some Research

Firstly, I noticed a problem in the implementation of a lot PsTree tools (much more prevalent on later versions of Windows because there is a far greater number of processes).

Since many processes start and/or are terminated, a situation may occur where Process A creates process B as its child process, it is then terminated, and later process C uses process A's PID since it recognizes it as free – resulting in PsTree incorrectly presenting Process C as Process B's parent.

To fix the problem above, I looked for a kernel struct (so User Mode Malware can't trick us) that can give me solid proof of processes' parent-child relationships, and found it in `_EPROCESS.CreateTime` (so I can validate the parent process's creation time and that it is not at a later time than its child).

Secondly, I had to understand how to identify the process type (.Net, Immersive, Protected or Service), so I can visually categorize them using colors:

- Identifying a .Net Process (Yellow) - CrossSessionCreate and WriteWatch flags are on.
- Identifying an Immersive Process (Metro apps - Blue) – must be under a Job and (_EPROCESS.TokenFlags & token flag LOWBOX) or "S-1-15-2-" is inside the token sids. This sid is reserved for Immersive processes.
- Identifying a Protected Process (Purple) – ProtectedProcess flag in Windows Vista to Windows 8.1; in later versions using the Protection struct inside the _EPROCESS.
- Services (Pink) – using the svcsan (plugin) output.

```
# Colored .Net processes. (os is vista or later)
if self.os_version['Major'] > 5 and e_proc.CrossSessionCreate == 1 and e_proc.WriteWatch == 1:
    process_comments[int(pid)] += "(Colored in yellow because this is a .Net process)"
    process_comments['pidColor'][int(pid)] = "yellow"

# Colored Immersive process. (os is vista or later)
TOKEN_LOWBOX = 0x4000 # this flag mean this is AppContainer!.
if proc_token and (self.os_version['Major'] > 5 and e_proc.Job != 0 and proc_token.TokenFlags & TOKEN_LOWBOX or any("S-1-15-2-" in sid for sid in proc_token.get_sids())):
    process_comments[int(pid)] += "(Colored in turquoise because this is a Immersive process)"
    process_comments['pidColor'][int(pid)] = "turquoise1"
    integrity = 'AppContainer' if integrity not in integrity_levels else integrity

protection = 'Disable'
# Colored Protected process. (os is 8.1 or later)
if (self.os_version['Major'] == 10 or (self.os_version['Major'] == 6 and self.os_version['Minor'] == 3)) and e_proc.Protection.Type > 0:
    process_comments[int(pid)] += "(Colored in purple because this is a Protected process)"
    process_comments['pidColor'][int(pid)] = "purple"
    protection = 'Protected {}'.format(e_proc.Protection.Level) if int(e_proc.Protection.Level) not in protect_signer_by_level else protect_signer_by_level[int(e_proc.Protection.Level)]
elif self.os_version['Major'] > 5: # Check if vista or later
    protection = 'Protected' if int(get_right_member(e_proc, ["ProtectedProcess"]) or -1) == 1 else 'Disable'
```

Abilities

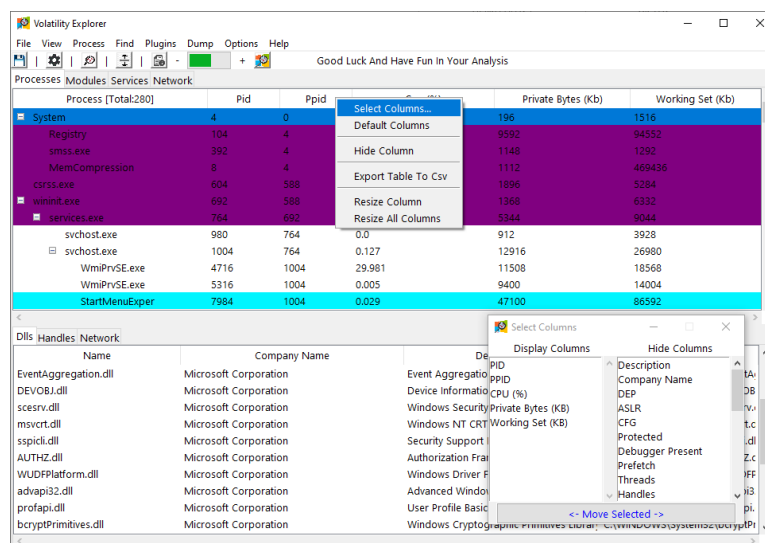
General UI

- The process tree hierarchy Table that is presented in the picture below has a functionality to add or remove columns from the table (using Right Click in the table Headers ->Select Columns, select some columns and click <-Move Selected-> button).

You can also restore the table to the default state, hide selected columns, export the table as to a csv file, and resize specific columns/all of the table (you can also sort-by-column and change the order of the columns by dragging and dropping a column or by changing the order in the Select Column window).

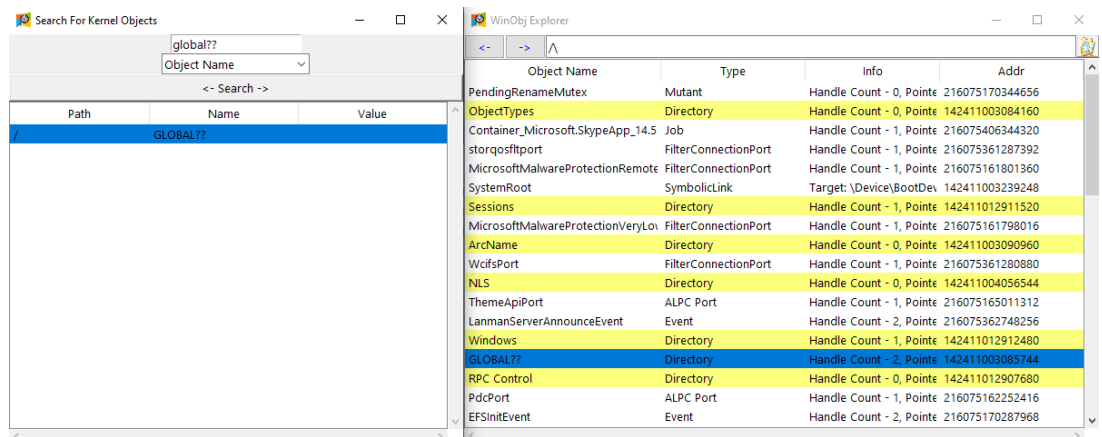
Additionally, you can type something while a row is selected to navigate to the row of the process typed (for example if I type "ser" in the main table below, the services.exe process will selected [or another process that starts with ser, if there are any]).

- The above is true for all the columns in the plugin, including modules, services, dlls, imports, etc.



- The main table also has a search bar (using ctrl+f/Find from the menu) that will search for handles/dlls.

- Options - > Change Themes to change the themes of the program (using tkinter themes, you can add more themes by installing the ttkthemes package [pip install ttkthemes]).
- Options -> Options[ctrl+o] to change configuration of the plugin: Add VirusTotal API key to integrate with VirusTotal, change dump directory, show unnamed handles, show PE string on PE properties, etc.
- By clicking on the progress bar, you are able to see what's currently running and what finished running, so you can track VolExp's current and historical progress (tasks that you run manually will also be added to the progress bar, like VAD information, PE information, etc.).
- You can also save your work to a file (File->Save/Save As) and then reload the plugin (using the -SAVED-FILE option) to restore from that specific point, to save all your work, the progress of the plugin, process comments and colors.
- Another very important UI class of this plugin, is the Explorer window.



This explorer is built upon the table (so everything you can do with the table you can do with the explorer as well).

1. You can enter rows that are colored in yellow (much like folders in Windows File Explorer).
2. Export the entire Explorer to a csv and not just the presented table.
3. Using Ctrl + F to search for a specific string inside a specific column and navigate to that column (by double clicking the specific item).

Functionality

- Process tree hierarchy with dlls, handles and connections to each process (there is also a tab for all the kernel modules, services and network connections).

The screenshot displays the Volatility Explorer interface. The 'Processes' tab is active, showing a list of processes with columns for Process [Total:280], Pid, Ppid, Cpu (%), Private Bytes (Kb), and Working Set (Kb). A 'Select Columns' dialog is open, allowing the user to choose which columns to display. The dialog has two sections: 'Display Columns' and 'Hide Columns'. The 'Display Columns' section includes Name, Company Name, Description, and Path. The 'Hide Columns' section includes File Version, Internal Name, Legal Copyright, Original File Name, Product Name, Product Version, DLL Base, and Ldr Address. The 'Move Selected' button is at the bottom of the dialog. Below the process list, the 'DLLs' tab is visible, showing a table with columns for Name, Company Name, Description, and Path. The table lists various DLLs such as RPCRT4.dll, ucrtbase.dll, umpnpmgr.dll, msvcrt.dll, WLDAP.dll, combase.dll, bcryptPrimitives.dll, CRYPT32.dll, MSASN1.dll, and WINTRUST.dll.

Process [Total:280]	Pid	Ppid	Cpu (%)	Private Bytes (Kb)	Working Set (Kb)
System	4	0		1516	
Registry	104	4		94552	
smss.exe	392	4		1292	
MemCompression	8	4		469436	
csrss.exe	604	588		5284	
wininit.exe	692	588		6332	
services.exe	764	692		9044	
svchost.exe	980	764		3928	
svchost.exe	1004	764		26980	
WmiPrvSE.exe	4716	1004		18568	
WmiPrvSE.exe	5316	1004		14004	
StartMenuExper	7984	1004		86592	

Name	Company Name	Description	Path
RPCRT4.dll	Microsoft Corporation	Remote Procedure Call Runtime	C:\WINDOWS\System32\RPCRT4.dll
ucrtbase.dll	Microsoft Corporation	Microsoft® C Runtime Library	C:\WINDOWS\System32\ucrtbase.dll
umpnpmgr.dll	Microsoft Corporation	User-mode Plug-and-Play Service	c:\windows\system32\umpnpmgr.dll
msvcrt.dll	Microsoft Corporation	Windows NT CRT DLL	C:\WINDOWS\System32\msvcrt.dll
WLDAP.dll	Microsoft Corporation	Windows Lockdown Policy	C:\WINDOWS\SYSTEM32\WLDAP.dll
combase.dll	Microsoft Corporation	Microsoft COM for Windows	C:\WINDOWS\System32\combase.dll
bcryptPrimitives.dll	Microsoft Corporation	Windows Cryptographic Primitives Library	C:\WINDOWS\System32\bcryptPrimitives.dll
CRYPT32.dll	Microsoft Corporation	Crypto API32	C:\WINDOWS\System32\CRYPT32.dll
MSASN1.dll	Microsoft Corporation	ASN.1 Runtime APIs	C:\WINDOWS\System32\MSASN1.dll
WINTRUST.dll	Microsoft Corporation	Microsoft Trust Verification APIs	C:\WINDOWS\System32\WINTRUST.dll

As you can see, you can also see the Company Name and the Description of the dll by default (this will load while the plugin is running). You can also add additional columns such as Internal Name, Copyright, Original Name, Product Name and Product Version, etc.

- Mapping of the meaning behind handles' granted_access (the number representing a handle's permission) to a coherent, literal meaning for each handle type.

Type	Name	File Sha	Handle	Access	Decoded Access
File	\Device\HarddiskVolume1\Windows\Sy	R--r-d	516	1179785	FILE_READ_DATA, FILE_READ_E
Directory	WindowStations		520	4	CREATE_OBJECT
Process	System Pid 396		524	42	PROCESS_CREATE_THREAD, P
Event	UniqueInteractiveSessionIdEvent		528	2031619	EVENT_ALL_ACCESS
Process	System Pid 432		536	42	PROCESS_CREATE_THREAD, P
Thread	Tid 460 Pid 4		540	2097151	THREAD_ALL_ACCESS
Event	EventShutDownCSRSS		556	2031619	EVENT_ALL_ACCESS
Desktop	Disconnect		560	983043	DESKTOP_READOBJECTS, DES
Desktop	Disconnect		564	983043	DESKTOP_READOBJECTS, DES
Session	Session1		568	0	Type: Desktop Decoded Access: DESKTOP_READOBJ

- Process right click will let you view specific data related to this process.

Volatility Explorer

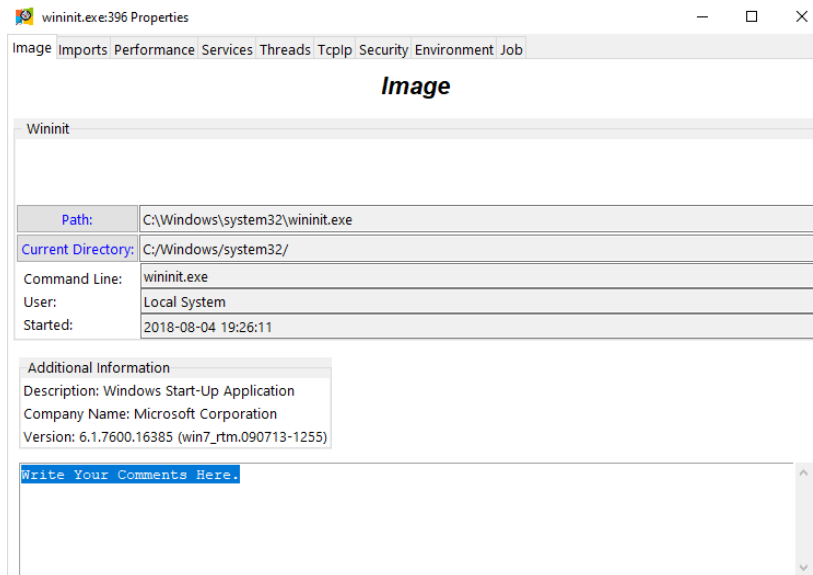
File View Process Find Plugins Dump Options Help

Good Luck And Have Fun In Your Analysis

Processes Modules Services Network

Process [Total:63]	Pid	Ppid	Cpu (%)	Private Bytes (Kb)	Working Set (Kb)
dwm.exe	2704	844	0.000	2184	6068
svchost.exe	868			19048	32108
svchost.exe	1012			6632	11388
svchost.exe	620			13672	13956
spoolsv.exe	1120			7612	13912
svchost.exe	1164			11712	14112
VGAAuthService.	1356			5680	10340
vmtoolsd.exe	1428			10508	18656
cmd.exe	3916			0	20
svchost.exe	1948			1684	4460
dllhost.exe	1324			4308	11208
msdtc.exe	1436			3472	7936
taskhost.exe	2344	492	0.198	7756	8096
sppsvc.exe	2500	492	3.171	5464	11248
SearchIndexer.	3064	492	0.595	18420	13868
PresentationFo	724	492	0.099	25808	16428
mscorsvw.exe	412	492	0.0	2172	5304

For example, let's see the properties and Memory Information:

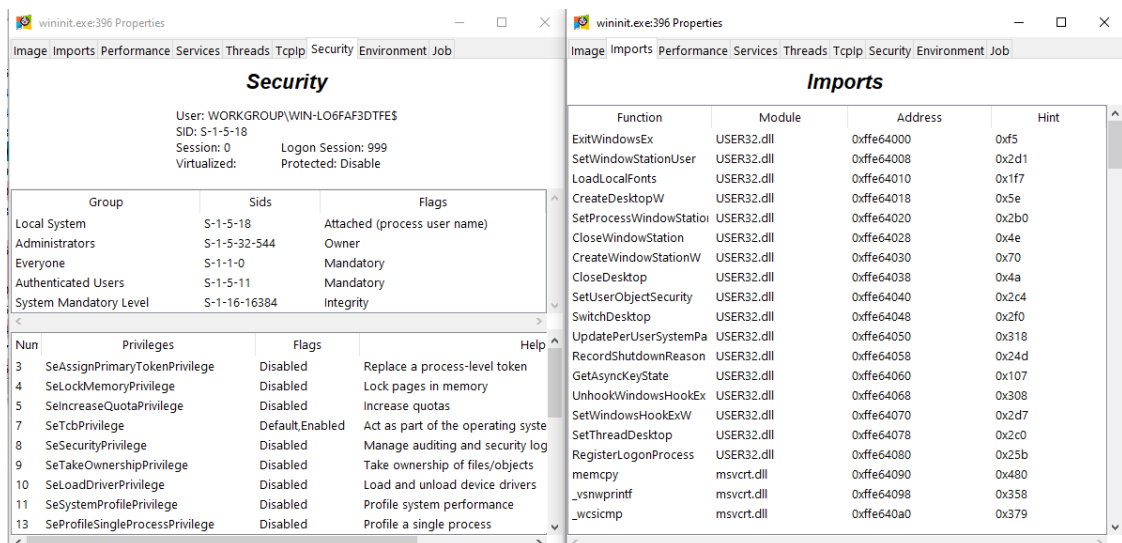


As you can see Process Properties contains a lot of tabs. In the first tab you can view general information.

You can also add a comment to this specific process (as well as coloring processes [using right click on the selected process->Change Color]).

There is a lot more information in the other tabs, like the Process Imports, Performance, Services, etc. - Very similarly to Process Explorer.

Moving on to Security, Imports, threads and environment:



wininit.exe:396 Properties

Image Imports Performance Services Threads Tcpip Security Environment Job

Threads

Tid	Flag	Create Time
400	PS_CROSS_THREAD_F	2018-08-04 19:26:11
444	PS_CROSS_THREAD_F	2018-08-04 19:26:11
484	PS_CROSS_THREAD_F	2018-08-04 19:26:12
488	PS_CROSS_THREAD_F	2018-08-04 19:26:12
540	PS_CROSS_TH	2018-08-04 19:26:14
544	PS_CROSS_TH	2018-08-04 19:26:14

Tid : 488

Start Addr : 0x7771c500

Flag : PS_CROSS_THREAD_FLAGS_TERMINATED PS_CROSS_THREAD_

Stack Base : 0xf88002300000

Base Priority : 13

Priority : 14

User Time : 0

Kernel Time : 1

Create Time : 2018-08-04 19:26:12

Offset : 275428110718304

wininit.exe:396 Properties

Image Imports Performance Services Threads Tcpip Security Environment Job

Environment

Variable	Value
ALLUSERSPROFILE	C:\ProgramData
CommonProgramFiles	C:\Program Files\Common Files
CommonProgramFiles(x86)	C:\Program Files (x86)\Common File
CommonProgramW6432	C:\Program Files\Common Files
COMPUTERNAME	WIN-LO6FAF3DTFE
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_CHECK	NO
NUMBER_OF_PROCESSORS	2
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.J
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 60 Stepping
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	3c03
ProgramData	C:\ProgramData

wininit.exe (396): Memory information (from VAD)

Start Size Use Protection Control Flags

0x270000 0xffff Heap (ID 1) PAGE_READWRITE

0x7ffffdb000 0xfff PEB PAGE_READWRITE

0x7efe0000 0xffff Read Only Memory Base PAGE_READONLY Based, Reserve

0x130000 0xfff Stack Base (tid: 400) [Wait (UserRequest)] PAGE_READWRITE

0x2540000 0x7fff Stack Reserved | Stack Guard start from 0x7ffffae010 (tid: 544) [Wait (WrQue PAGE_READWRITE

0x1df0000 0x7fff Stack Reserved | Stack Guard start from 0x7ffffd5010 (tid: 484) [Wait (UserRi PAGE_READWRITE

0xb0000 0x7fff Stack Reserved | Stack Guard start from 0x7ffffde010 (tid: 400) [Wait (UserRi PAGE_READWRITE

0x7fe0000 0xffff User Shared Data PAGE_READONLY

0x21a0000 0x2cefff \Windows\Globalization\Sorting\SortDefault.nls PAGE_READONLY Accessed, File

0x7efd960000 0x6aff \Windows\System32\KernelBase.dll PAGE_EXECUTE_WRITECOPY Accessed, File, Image

Members

Start 0x7efd960000

End 0x7efd9cafff

Size 0x6aff

Tag Vad

Flags 'Protection': 7, 'VadType': 2

Protection PAGE_EXECUTE_WRITECOPY

Type 2

Control Area 0xfa8019f71990

Segment 0xfa8019f71990

As you can see, memory information will display virtual address mapping information for a specific process, with a detailed description for each memory region.

- **PE File:** You can also get information about PE files (from main table modules and from process dlls) by right clicking on a PE file:

Dlls Handles Network

Name Company Name Description Path

msvcrt.dll Corporation Windows NT CRT DLL C:\WINDOWS\System32\msvcrt.c

sspicli.dll Corporation Security Support Provider Interface C:\WINDOWS\system32\sspicli.d

AUTHZ.dll Corporation Authorization Framework C:\WINDOWS\system32\AUTHZ.c

WUDFPlatform.dll Corporation Windows Driver Foundation - User-mode C:\WINDOWS\SYSTEM32\WUDFF

advapi32.dll Corporation Advanced Windows 32 Base API C:\WINDOWS\System32\advapi3

profapi.dll Corporation User Profile Basic API C:\WINDOWS\System32\profapi.

bcryptPrimitives.dll Corporation Windows Cryptographic Primitives Libran C:\WINDOWS\System32\bcryptPr

rmclient.dll Corporation Resource Manager Client C:\WINDOWS\SYSTEM32\rmclien

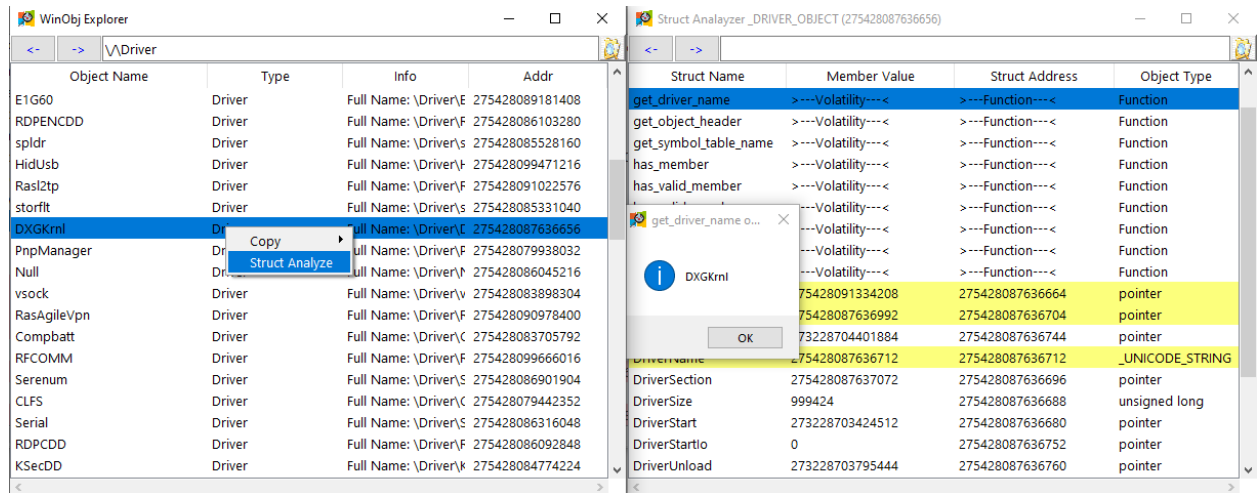
combase.dll Microsoft Corporation Microsoft COM for Windows C:\WINDOWS\System32\combas

kernel.appcore.dll Microsoft Corporation AppModel API Host C:\WINDOWS\System32\kernelAj

Name	Va	Size Of Raw Data	Pointer To Raw Data	Characteristics	Meaning (Characteristics)
.text	4096	1713152	1536	1744830496	IMAGE_SCN_MEM_NOT_PAGED
INITKDE	1720320	14848	1714688		
POOLM	1736704	7680	1729536		
POOLC	1744896	12288	1737216		
RWEXE	1757184	0	0		

- The screenshot shows a File Explorer window with the address bar set to `%ProgramData%\Microsoft\Search\Data\Applications\Windows`. The main pane displays a list of files and folders. The file `Windows.edb` is highlighted in blue. The left pane shows the file's properties, including its location at `%ProgramFiles%\Lavasoft\Windows.edb` and its size of 2103612896 bytes.

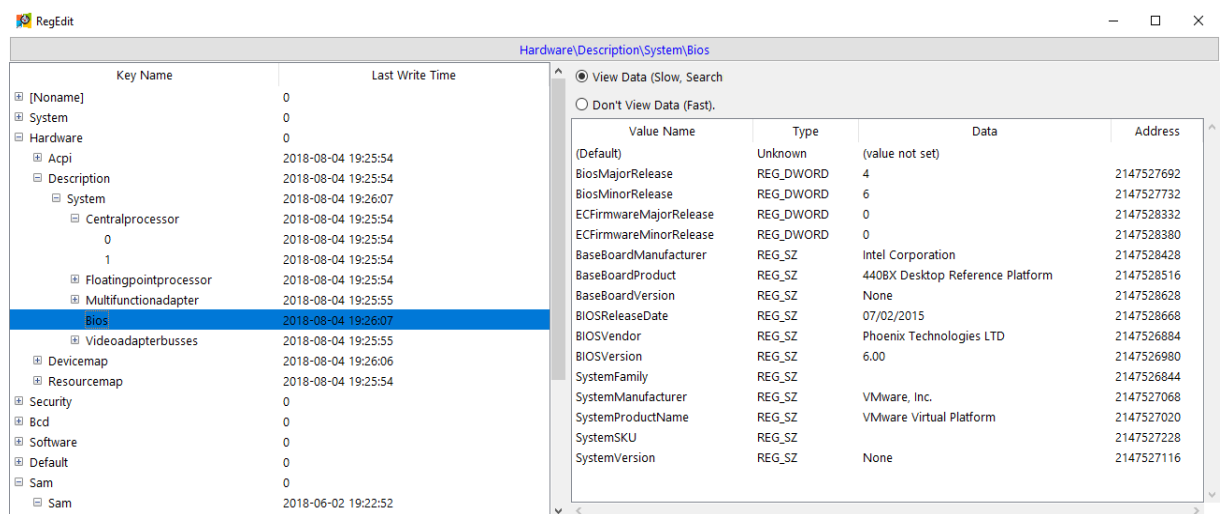
As you can see, the files are collected inside an Explorer UI with user assist data (using userassist plugin). You can jump directly to a specific file from the user assist.



Here you can see WinObj Explorer run Struct Analyze on a specific driver, and then run the volatility function `get_driver_name` on that driver which will result in the popup window DXGKrn.

WinObj Explorer has [Struct Analyzer](#) on a lot of windows kernel objects, like `_OBJECT_DIRECTORY`, `_DRIVER_OBJECT`, `_KMUTANT`, etc.

- [Registry Viewer](#) – will display all the registry keys very similarly to Regedit, and will let you enumerate the values as well as the data.



- **Integrate With Struct Analyzer:** For each process, PE file, thread, etc. you can right click -> Struct Analyzer to view much more information of each struct.

For example running [Struct Analyzer](#) on _EPROCESS:

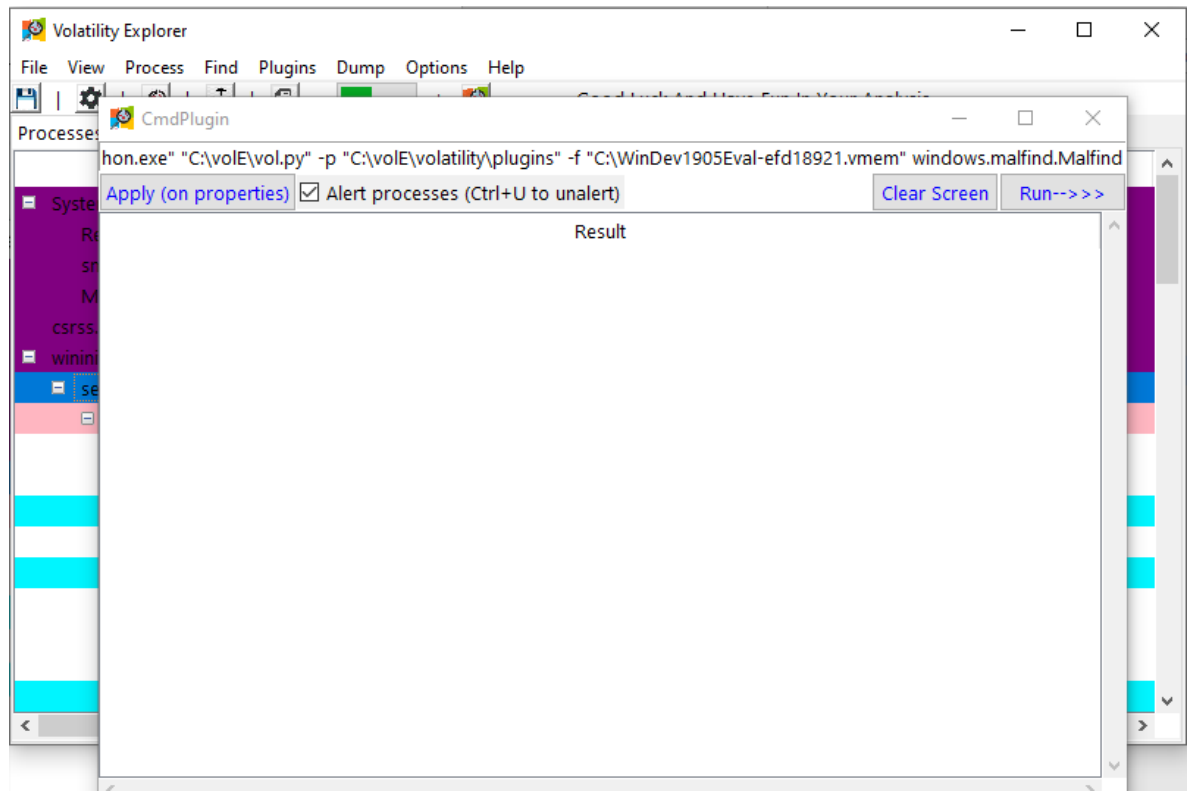
Struct Name	Member Value	Struct Address	Object Type
load_order_modules	>---Volatility---<	>---Function---<	Function
mem_order_modules	>---Volatility---<	>---Function---<	Function
member	>---Volatility---<	>---Function---<	Function
write	>---Volatility---<	>---Function---<	Function
AccountingFolded	0	275428111740444	bitfield
ActiveProcessLinks	275428111739752	275428111739752	_LIST_ENTRY
ActiveThreads	22	275428111740168	unsigned long
ActiveThreadsHighWatermark	24	275428111740572	unsigned long
AddressCreationLock	275428111739896	275428111739896	_EX_PUSH_LOCK
AddressSpaceInitialized	2	275428111740448	bitfield
AffinityPermanent	0	275428111740444	bitfield
AffinityUpdateEnable	0	275428111740444	bitfield
AlpcContext	275428111740520	275428111740520	_ALPC_PROCESS_CONTEXT
AwelInfo	0	275428111740264	pointer
BreakOnTermination	0	275428111740448	bitfield
CloneRoot	0	275428111739936	pointer
CommitCharge	4447	275428111739800	unsigned long long
CommitChargeLimit	0	275428111740248	unsigned long long

- **Integrate With Other Plugins:** This plugin has a lot of capabilities but analysts will probably have to use much more plugins during their work. This plugin allows you to easily integrate with other plugins by navigating to the Plugins tab->All/Well Known> <plugin_name>:

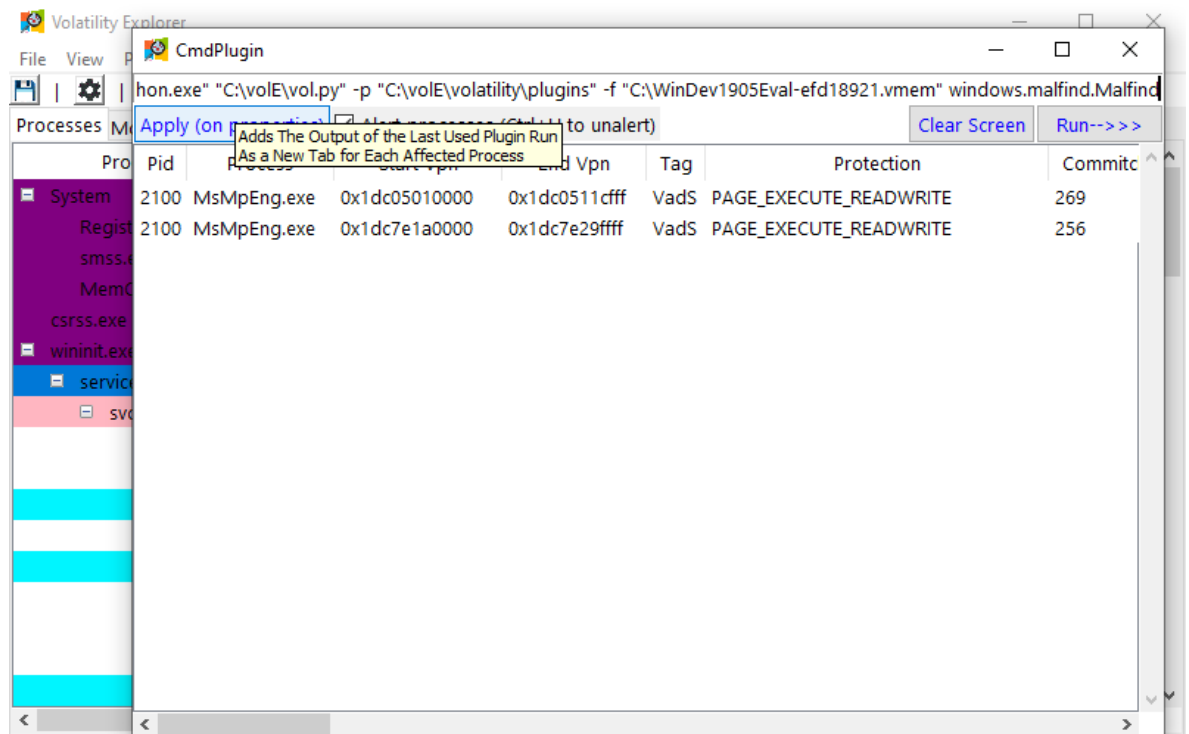
Volatility Explorer Plugins tab -> All/Well Known > <plugin_name>:

Process [Total:99]	Pid	Working Set (Kb)
System	4	4
Registry	68	860
smss.exe	292	16
MemCompression	1640	268672
csrss.exe	392	676
wininit.exe	464	64
services.exe	512	2428
svchost.exe	712	6292
WmiPrvSE.exe	2848	8032
SppExtComObj.E	4088	1564
ShellExperienc	1776	232
dllhost.exe	4212	1688
SearchUI.exe	4288	284
RuntimeBroker.	4520	2528
RuntimeBroker.	4608	6360
ApplicationFra	4880	2520
MicrosoftEdge.	4928	28

It will automatically enter the command line (you can add arguments if you want to).

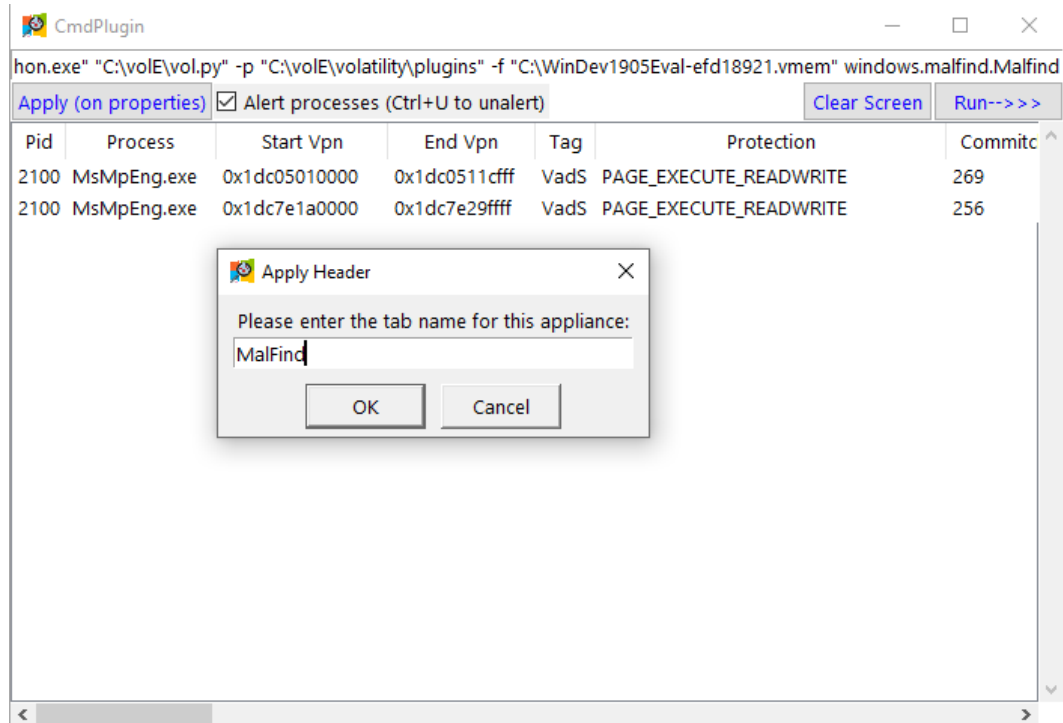


Clicking Run will run the plugin (as a subprocess)

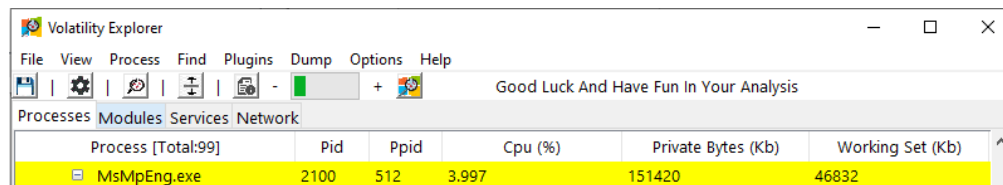
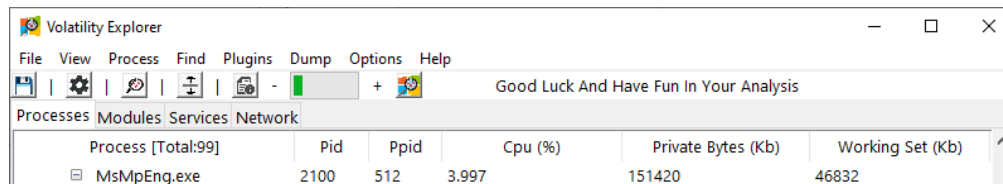
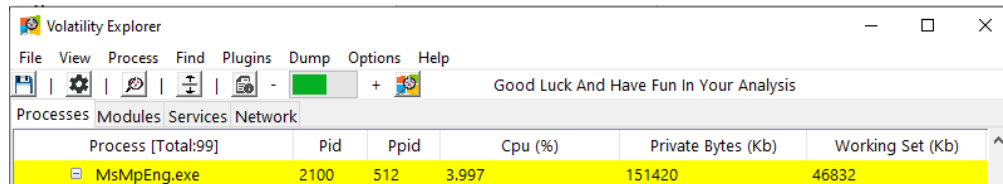


If we will click "Apply (on properties)" it will add the plugin output to the

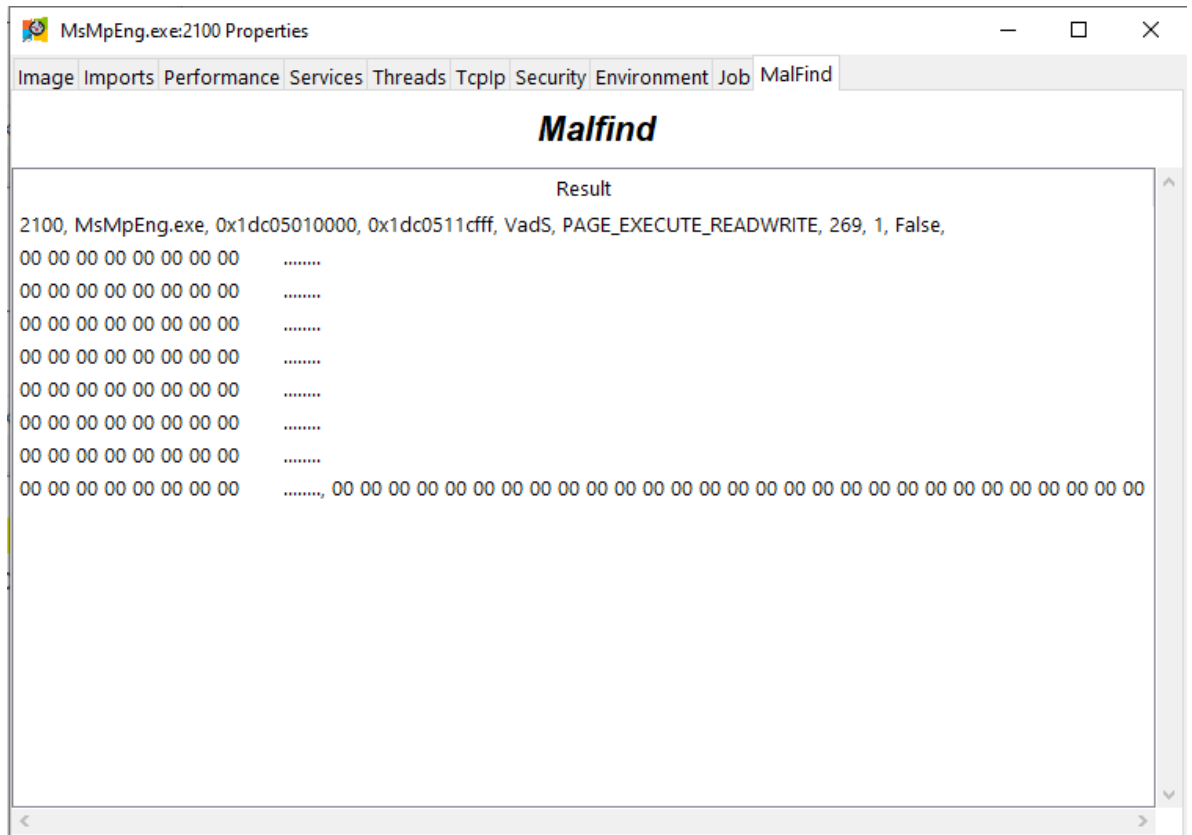
corresponding process properties and make a visual alert for this process. You will then have to enter a name for the new tab that will show in the properties window. For example, let's call it MalFind:



Which will make the process with PID 2100 flash as yellow:



When we double click this process (or Right Click->Properties) we will see a new tab - MalFind (the name we chose) tab with the results of the Malfind:



Once we enter the process the visual alert only for this process will stop (or you can unalert all the processes at once using the Menu->View->Unalert Processes (Ctrl+U)).

This is very strong capability which will let you organize your analysis using only the VolExp plugin, allowing you access to other plugins through it.

Plugin Flow:

This plugin starts by checking the user parameters, if something is missing it will pop up the options menu and ask the user to insert the missing data.

Then the plugin will get all processes data from the main table (PsTree view), and sort the processes in a PsTree view.

Next, it will run 2 additional processes (multiprocessing) to get data from the FileScanGui and WinObjGui plugins (MftParserGui will be added in the future when it will be added to the framework [the code for this already ready]).

At the same time its will start 5 additional threads, one for each of the following tasks:

1. Get Services using the SvcScan plugin and update the table.
2. Get Network Data using the NetScan plugin and update the table.
3. Get Process Handles using the Handles plugin (adding the Decoded Access as well) and update the table.
4. Get Process Environment Variables and Imports.
5. Attempt to translate the process SIDs to Username (using the GetSIDs plugin that I have written), and get additional information about each PE (Company Name, File Description, File Version, Internal Name, Legal Copyright, Original File Name, Product name, Product Version) using pefile module.

After this last thread finishes, it will create X numbers of threads (1 thread for each hive), walk over its keys and get the registry data (to be used by RegEdit).

It is especially important to note the fact that the better your computer's hardware, the program will run faster. (If you try to run VolExp plugin using a very slow computer the UI may freeze sometimes, especially at the beginning, because a slow computer can't really handle 5 different threads and 2 additional processes so quickly.)

Struct Analyzer

Introduction

Struct Analyzer is a nice extension to the other UI plugins (but can also run as a plugin).

It parses structs remarkably similarly to volshell's dt function, but in an Explorer UI, so you can easily analyze structs in a graphical hierarchy.

Additionally, you can run all the volatility function on the struct you analyze and view the results. If the result is another struct, it will open in a new struct analyzer window. If it is a list/generator of structures, it will open a list of structs view with the ability to double click on one or more to open struct analyzer on the chosen struct.

Motivation

Despite the fact that VolExp's utilities show us a lot of information, it cannot display everything - so I created Struct Analyzer, similarly to dt in WinDbg, it can parse structs inside the main struct by clicking at it), so I can view any additional information in any struct I wish, by displaying the struct in another view.

Arguments

This plugin accepts the following arguments:

- --STRUCT: the struct type to analyze.
- --ADDR: the address of the struct.
- --PID: the PID of the address space's process of the struct (enter "--PID=physical" for physical address space).

Struct Analyzer _EPROCESS (275428091444016)

Struct Name	Member Value	Struct Address	Object Type
Outswapped	0	275428091445104	bitfield
OverrideAddressSpace	0	275428091445104	bitfield
PageDirectoryPte	275428091444736	275428091444736	_HARDWARE_PTE
Pcb	275428091444016	275428091444016	_KPROCESS
PdeUpdateNeeded	0	275428091445104	bitfield
PeakVirtualSize	141885440	275428091444480	unsigned long long
Peb	8796092846080	275428091444840	pointer
PhysicalVadRoot	275428102061856	275428091444584	pointer
PrefetchTrace	275428091444848	275428091444848	_EX_FAST_REF
PrimaryTokenFrozen	1	275428091445100	bitfield
PriorityClass	2	275428091444767	unsigned char
ProcessDelete	0	275428091445104	bitfield
ProcessExiting	0	275428091445104	bitfield
ProcessInSession	1	275428091445104	bitfield
ProcessInserted	1	275428091445104	bitfield
ProcessLock	275428091444368	275428091444368	_EX_PUSH_LOCK
ProcessQuotaPeak	21648 284256	275428091444440	array
ProcessQuotaUsage	17240 284256	275428091444424	array

Some Research

The research of this plugin had a lot to do with how Volatility3's framework works and how to parse it correctly into a GUI where you can view all the data and run the functions you need – basically giving full support to all of Volatility's functions for the chosen struct, such as cast and dereference_as.

Abilities

Struct Analyzer is built on top of Explorer so everything you can do using Explorer, you can do with the Struct Analyzer window:

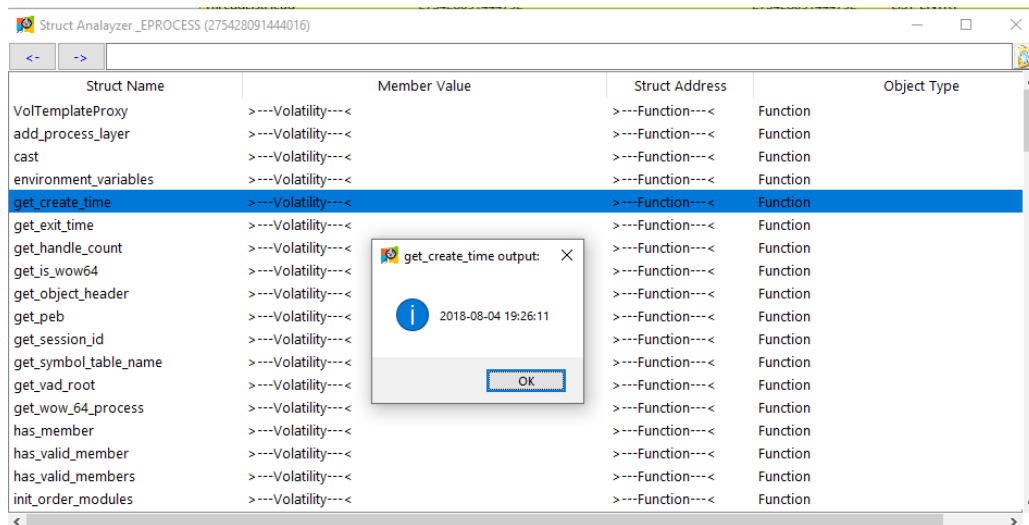
- Display all the struct members.

Struct Analyzer _EPROCESS (275428091444016)

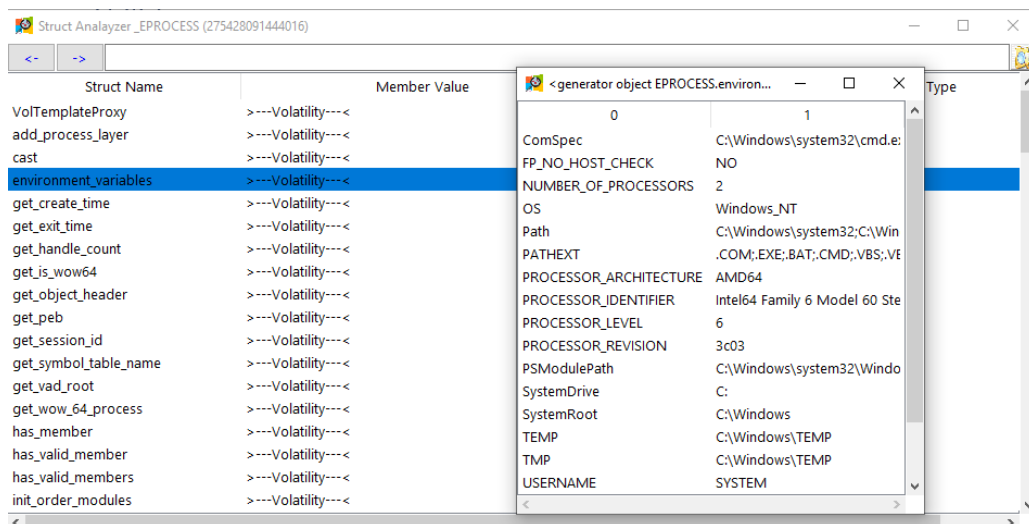
Struct Name	Member Value	Struct Address	Object Type
SessionProcessLinks	275428091444496	275428091444496	_LIST_ENTRY
SetTimerResolution	0	275428091445104	bitfield
SetTimerResolutionLink	0	275428091445104	bitfield
SmallestTimerResolution	0	275428091445232	unsigned long
Spare	0	275428091444688	pointer
StackRandomizationDisabled	0	275428091445100	bitfield
ThreadListHead	275428091444792	275428091444792	_LIST_ENTRY
TimerResolutionLink	275428091445208	275428091445208	_LIST_ENTRY
TimerResolutionStackRecord	0	275428091445240	pointer
Token	275428091444536	275428091444536	_EX_FAST_REF
UmsScheduledThreads	0	275428091444652	unsigned long
UniqueProcessId	388	275428091444400	pointer
VadRoot	275428091445112	275428091445112	_MM_AVL_TABLE
VdmAllowed	0	275428091445104	bitfield
VirtualSize	141885440	275428091444488	unsigned long long
Vm	275428091444936	275428091444936	_MMSUPPORT
VmDeleted	0	275428091445104	bitfield
VmTopDown	0	275428091445104	bitfield

- Display all the struct functions, with the ability to run them (and also insert arguments if necessary). The output will be displayed in a popup window of the following possible types (according to the result):

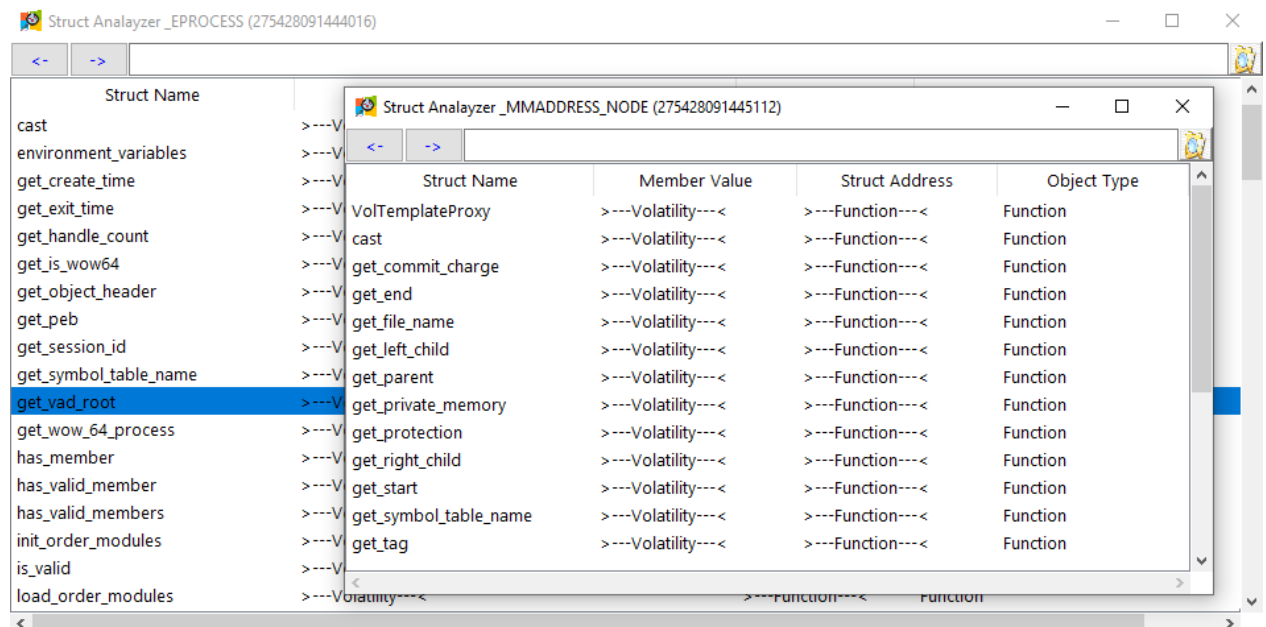
1. As a string:



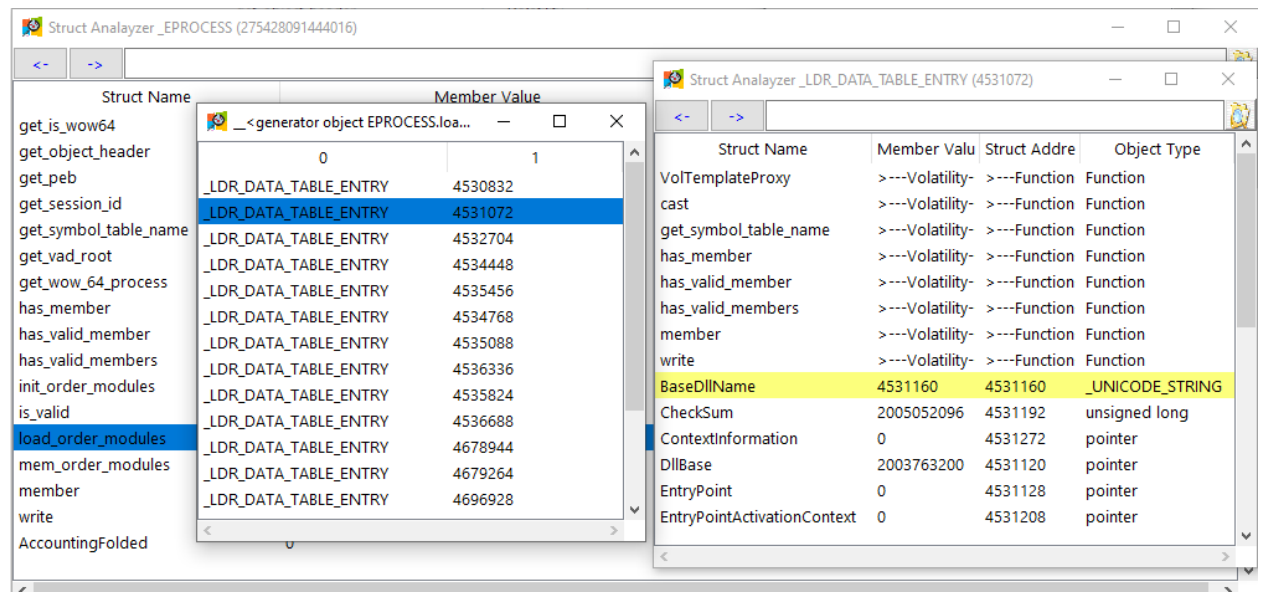
2. A list of strings:



3. Object (display in another Struct Analyzer):



4. A list of objects (Double clicking on an element (struct) will open it in a new Struct Analyzer window):



- By default, Struct Analyzer will parse up to 3 structs inside each struct (every time you get inside another struct it will parse 3 structs recursively inside as well). So you can use the explorer search to find elements 3 hierarchal levels lower.

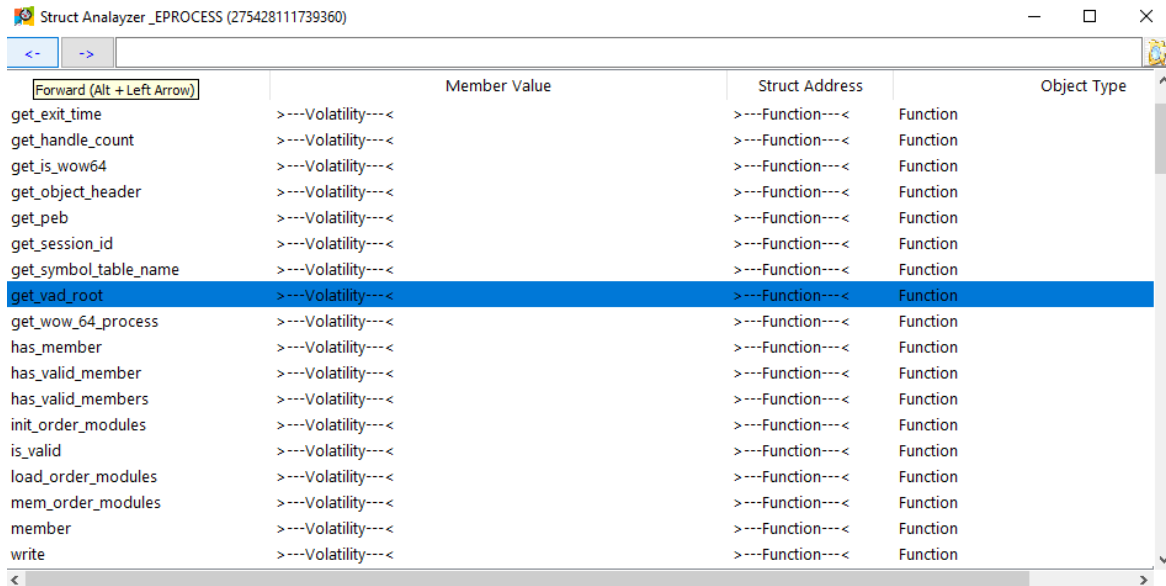
The screenshot shows two windows from the Struct Analyzer tool. The main window, titled 'Struct Analyzer _EPROCESS (275428091444016)', displays a list of structs and their members. The 'Pcb\LdtProcessLock\' path is selected in the breadcrumb. The list includes various structs like 'VolTemplateProxy', 'cast', 'get_symbol_table_name', etc., and a 'Contention' struct which is highlighted in blue. The 'Contention' struct has a 'Member Value' of 0 and a 'Struct Address' of 275428091444320, with a type of 'unsigned long'. Other structs like 'Gate', 'KernelApcDisable', 'Owner', and 'SpecialApcDisable' are highlighted in yellow.

The right-hand pane, titled 'Struct Analyzer Search (deeg 3 insi...', shows search results. It has a search bar with 'Cont' entered and a dropdown for 'Struct Name'. Below the search bar is a table with two columns: 'Path' and 'Name'. The results list various paths and their corresponding names, with 'Pcb\LdtProcessLock' and 'Contention' highlighted in blue.

Path	Name
AlpcContext	AlpcContext
ObjectTable	HandleContentionEvent
ObjectTable\QuotaProcess	AlpcContext
ObjectTable\QuotaProcess	AlpcContext
Pcb\Header	ThreadControlFlags
Pcb\Header	TimerControlFlags
Pcb\LdtProcessLock	Contention
Peb	ActivationContextData
Peb	SystemDefaultActivationCor
Peb	pContextData

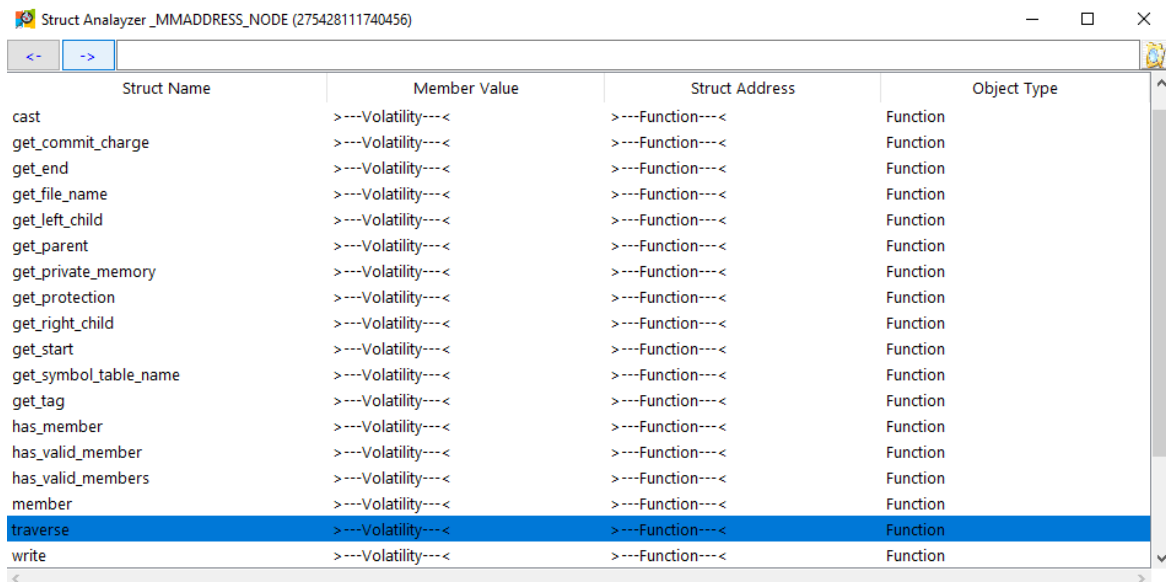
Struct Analyzer Flow (example)

Here is an example of what you can do using Struct Analyzer (_PROCESS at address: 275428111739360):



	Member Value	Struct Address	Object Type
get_exit_time	>---Volatility---<	>---Function---<	Function
get_handle_count	>---Volatility---<	>---Function---<	Function
get_is_wow64	>---Volatility---<	>---Function---<	Function
get_object_header	>---Volatility---<	>---Function---<	Function
get_peb	>---Volatility---<	>---Function---<	Function
get_session_id	>---Volatility---<	>---Function---<	Function
get_symbol_table_name	>---Volatility---<	>---Function---<	Function
get_vad_root	>---Volatility---<	>---Function---<	Function
get_wow_64_process	>---Volatility---<	>---Function---<	Function
has_member	>---Volatility---<	>---Function---<	Function
has_valid_member	>---Volatility---<	>---Function---<	Function
has_valid_members	>---Volatility---<	>---Function---<	Function
init_order_modules	>---Volatility---<	>---Function---<	Function
is_valid	>---Volatility---<	>---Function---<	Function
load_order_modules	>---Volatility---<	>---Function---<	Function
mem_order_modules	>---Volatility---<	>---Function---<	Function
member	>---Volatility---<	>---Function---<	Function
write	>---Volatility---<	>---Function---<	Function

Double Clicking on the get_vad_root function will result in another struct analyzer window. This time, it will be a struct analyzer for _MMADDRESS_NODE (which is the struct type that is returned from the get_vad_root function):



Struct Name	Member Value	Struct Address	Object Type
cast	>---Volatility---<	>---Function---<	Function
get_commit_charge	>---Volatility---<	>---Function---<	Function
get_end	>---Volatility---<	>---Function---<	Function
get_file_name	>---Volatility---<	>---Function---<	Function
get_left_child	>---Volatility---<	>---Function---<	Function
get_parent	>---Volatility---<	>---Function---<	Function
get_private_memory	>---Volatility---<	>---Function---<	Function
get_protection	>---Volatility---<	>---Function---<	Function
get_right_child	>---Volatility---<	>---Function---<	Function
get_start	>---Volatility---<	>---Function---<	Function
get_symbol_table_name	>---Volatility---<	>---Function---<	Function
get_tag	>---Volatility---<	>---Function---<	Function
has_member	>---Volatility---<	>---Function---<	Function
has_valid_member	>---Volatility---<	>---Function---<	Function
has_valid_members	>---Volatility---<	>---Function---<	Function
member	>---Volatility---<	>---Function---<	Function
traverse	>---Volatility---<	>---Function---<	Function
write	>---Volatility---<	>---Function---<	Function

Double clicking "traverse" will result in the following popup, asking you to enter function

parameters, since the traverse function requires parameters to run (this will happen every time you try to run a function that accepts arguments):

traverse Arguments: — □ ×

visited

None

depth

0

[<- Save & Continue ->](#)

Double clicking traverse and entering the parameters (you probably shouldn't enter anything, just click Save & Continue button, since it fills in the default values) will result in the following list:

0	1
._MMVAD	275428111688624
._MMVAD	275428111021584
._MMVAD_SHORT	275428111700176
._MMVAD	275428111701504
._MMVAD_SHORT	275428111719808
._MMVAD	275428111711936
._MMVAD	275428111695360
._MMVAD	275428111740944
._MMVAD	275428111708816
._MMVAD_SHORT	275428111713632
._MMVAD	275428111736512
._MMVAD_SHORT	275428114287200

Double clicking on one of the structs will append another Struct Analyzer window of the chosen struct:

Struct Analyzer _MMVAD (275428111021584) — □ ×

[<-](#) [->](#)

Struct Name	Member Value	Struct Address	Object Type
VolTemplateProxy	>---Volatility---<	>---Function---<	Function
cast	>---Volatility---<	>---Function---<	Function
get_commit_charge	>---Volatility---<	>---Function---<	Function
get_end	>---Volatility---<	>---Function---<	Function
get_file_name	>---Volatility---<	>---Function---<	Function
get_left_child	>---Volatility---<	>---Function---<	
get_parent	>---Volatility---<	>---Function---<	
get_private_memory	>---Volatility---<	>---Function---<	
get_protection	>---Volatility---<	>---Function---<	
get_right_child	>---Volatility---<	>---Function---<	
get_start	>---Volatility---<	>---Function---<	
get_symbol_table_name	>---Volatility---<	>---Function---<	
get_tag	>---Volatility---<	>---Function---<	Function
has_member	>---Volatility---<	>---Function---<	Function
has_valid_member	>---Volatility---<	>---Function---<	Function
has_valid_members	>---Volatility---<	>---Function---<	Function
member	>---Volatility---<	>---Function---<	Function
traverse	>---Volatility---<	>---Function---<	Function

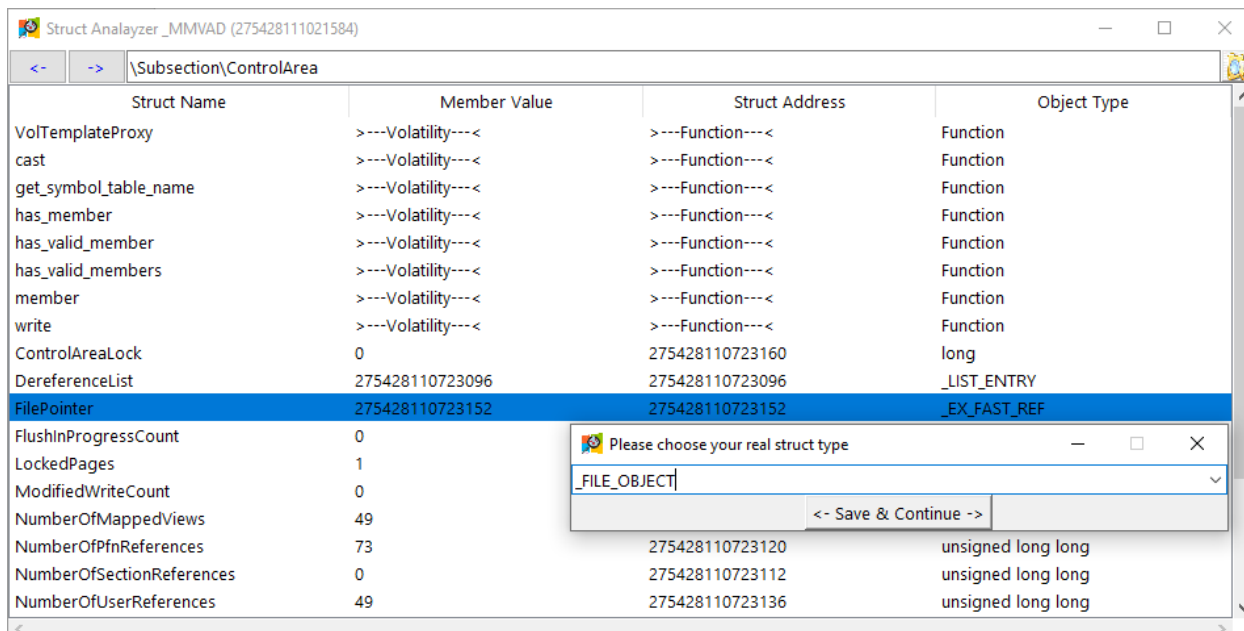
get_file_name output: ×

\\Windows\\Globalization\\Sorting\\SortDefault.nls

OK

Double clicking on get_file_name will append the output of the function (the file name).

Let's go deeper and try to get the file name (without the get_file_name function). Navigating to Subsection->Control Area and then right clicking on the file and dereference it as _FILE_OBJECT:



Struct Analyzer_MMVAD (275428111021584)

Subsection\ControlArea

Struct Name	Member Value	Struct Address	Object Type
VolTemplateProxy	>---Volatility---<	>---Function---<	Function
cast	>---Volatility---<	>---Function---<	Function
get_symbol_table_name	>---Volatility---<	>---Function---<	Function
has_member	>---Volatility---<	>---Function---<	Function
has_valid_member	>---Volatility---<	>---Function---<	Function
has_valid_members	>---Volatility---<	>---Function---<	Function
member	>---Volatility---<	>---Function---<	Function
write	>---Volatility---<	>---Function---<	Function
ControlAreaLock	0	275428110723160	long
DereferenceList	275428110723096	275428110723096	_LIST_ENTRY
FilePointer	275428110723152	275428110723152	_EX_FAST_REF
FlushInProgressCount	0		
LockedPages	1		
ModifiedWriteCount	0		
NumberOfMappedViews	49		
NumberOfPfnReferences	73	275428110723120	unsigned long long
NumberOfSectionReferences	0	275428110723112	unsigned long long
NumberOfUserReferences	49	275428110723136	unsigned long long

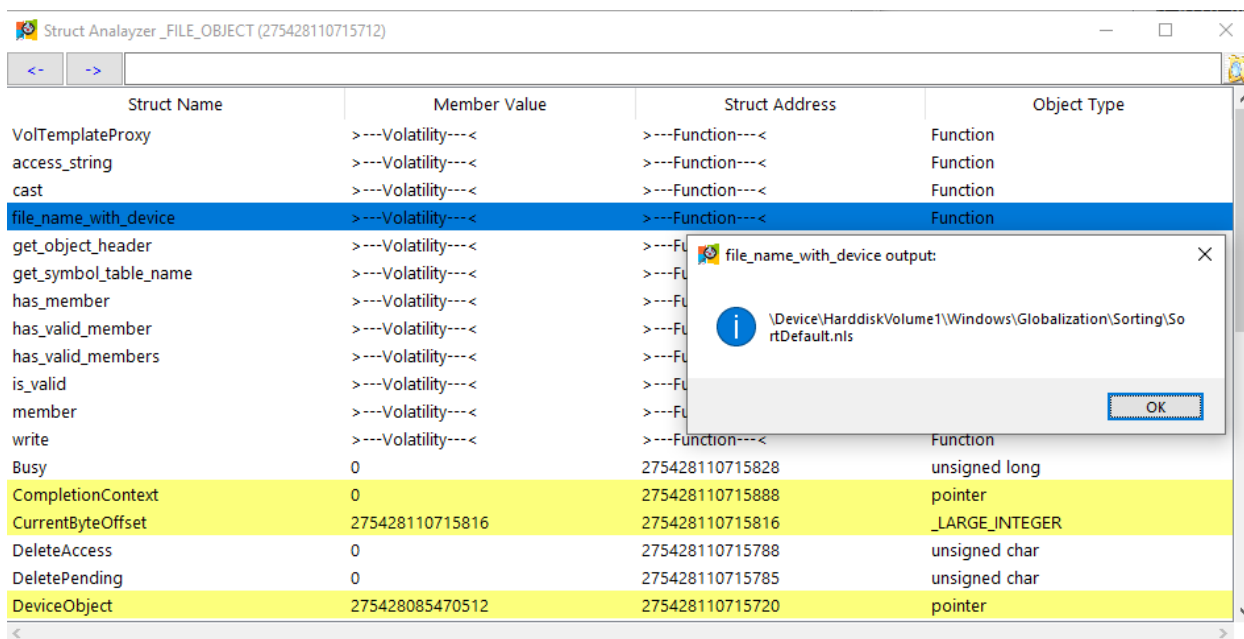
Please choose your real struct type

_FILE_OBJECT

<- Save & Continue ->

This will result in another Struct Analyzer window for _FILE_OBJECT in the dereferenced address.

Now let's click on the function file_name_with_device:



Struct Analyzer_FILE_OBJECT (275428110715712)

Struct Name	Member Value	Struct Address	Object Type
VolTemplateProxy	>---Volatility---<	>---Function---<	Function
access_string	>---Volatility---<	>---Function---<	Function
cast	>---Volatility---<	>---Function---<	Function
file_name_with_device	>---Volatility---<	>---Function---<	Function
get_object_header	>---Volatility---<	>---Fu	
get_symbol_table_name	>---Volatility---<	>---Fu	
has_member	>---Volatility---<	>---Fu	
has_valid_member	>---Volatility---<	>---Fu	
has_valid_members	>---Volatility---<	>---Fu	
is_valid	>---Volatility---<	>---Fu	
member	>---Volatility---<	>---Fu	
write	>---Volatility---<	>---Fu	
Busy	0	275428110715828	unsigned long
CompletionContext	0	275428110715888	pointer
CurrentByteOffset	275428110715816	275428110715816	_LARGE_INTEGER
DeleteAccess	0	275428110715788	unsigned char
DeletePending	0	275428110715785	unsigned char
DeviceObject	275428085470512	275428110715720	pointer

file_name_with_device output:

\Device\HarddiskVolume1\Windows\Globalization\Sorting\SortDefault.nls

OK

WinObjGui

Introduction

WinObjGui is a UI-based plugin which allows you to view all the kernel objects in an Explorer-like GUI.

It uses the WinObj plugin to parse the entire Windows object directory.

In addition to all the Explorer functions, you will also be able to integrate with [Struct Analyzer](#) and analyze structs from the object directory.

Motivation

After creating [VolExp](#), which is remarkably similar to Process Explorer if we compare it to the SysInternals tool, I was looking for other UI tools that could help an analyst. Another SysInternals tool caught my eye: WinObj, so I decided to create this tool.

Furthermore, I realized that it can fit very well with the [Explorer UI](#), and help make sure that the analyst doesn't miss anything (the output of the winobj.py has a lot of verbose data, making it easy to miss critical information using the CLI tool, so a GUI-based tool can help tremendously).

Startup Arguments

This plugin accepts the following argument:

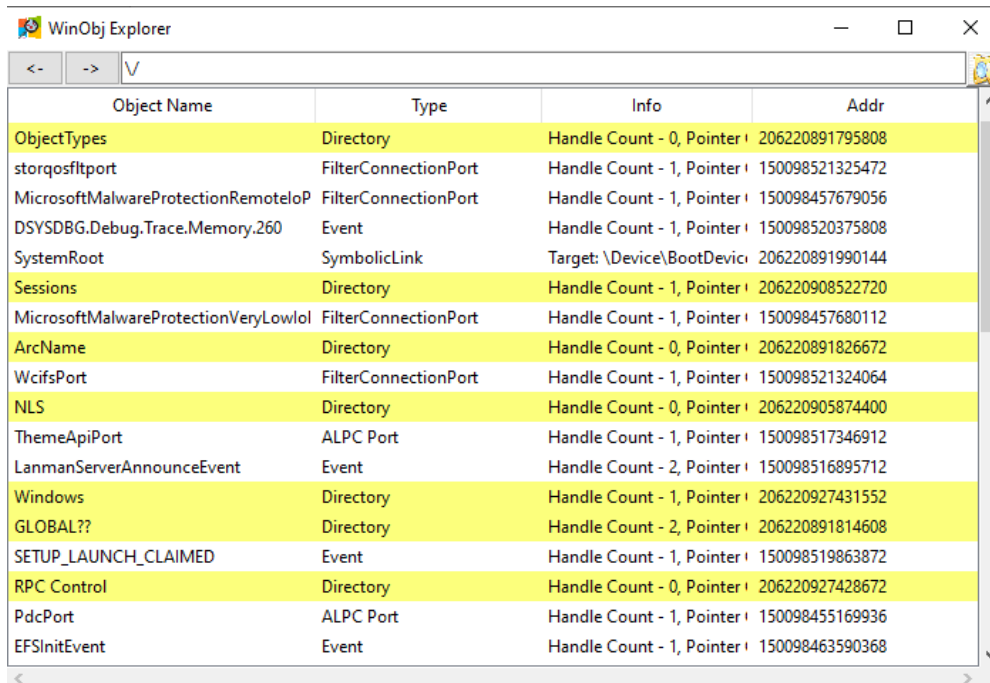
- --GET-DICT: File path to recursively output all the windows directory data in a pickle format (this argument is optional, and is used by [volexp](#). If this argument is present, then no GUI will appear).

Some Research

This Plugin had nearly no research, as it's just a UI view of the WinObj plugin.

The only research that was necessary was to find out the object type of objects referenced by

pointers. For example: driver -> _DRIVER_OBJECT, event -> _KEVENT, mutant -> _KMUTANT, etc.



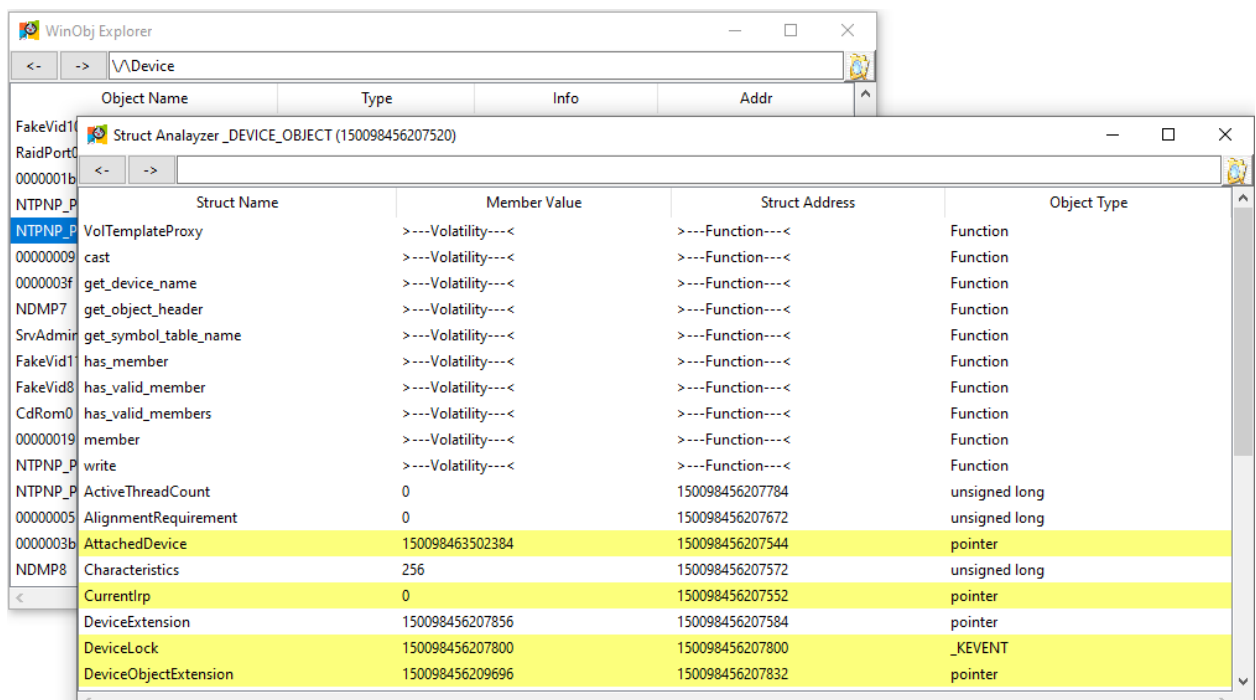
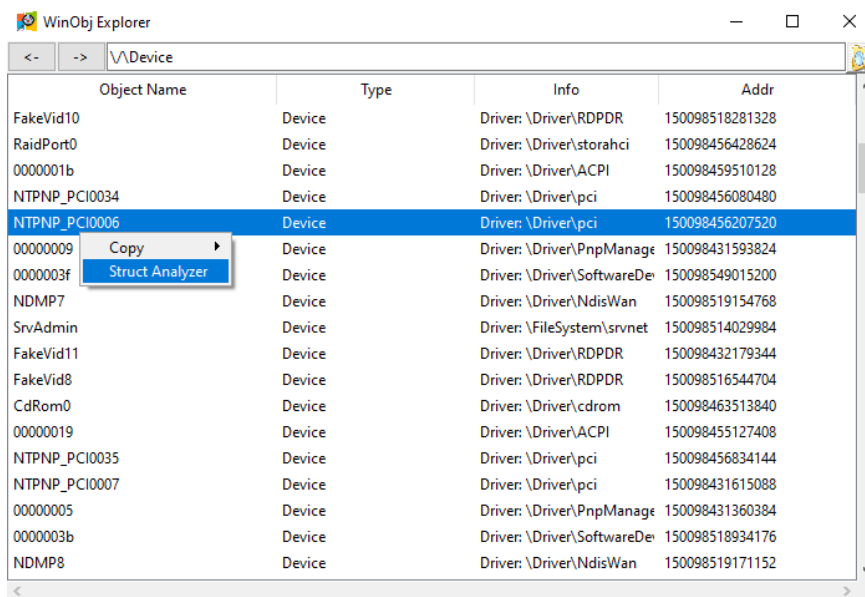
WinObj Explorer

Object Name	Type	Info	Addr
ObjectTypes	Directory	Handle Count - 0, Pointer	206220891795808
storqosfltport	FilterConnectionPort	Handle Count - 1, Pointer	150098521325472
MicrosoftMalwareProtectionRemoteloP	FilterConnectionPort	Handle Count - 1, Pointer	150098457679056
DSYSDBG.Debug.Trace.Memory.260	Event	Handle Count - 1, Pointer	150098520375808
SystemRoot	SymbolicLink	Target: \Device\BootDevice	206220891990144
Sessions	Directory	Handle Count - 1, Pointer	206220908522720
MicrosoftMalwareProtectionVeryLowlo	FilterConnectionPort	Handle Count - 1, Pointer	150098457680112
ArcName	Directory	Handle Count - 0, Pointer	206220891826672
WcifsPort	FilterConnectionPort	Handle Count - 1, Pointer	150098521324064
NLS	Directory	Handle Count - 0, Pointer	206220905874400
ThemeApiPort	ALPC Port	Handle Count - 1, Pointer	150098517346912
LanmanServerAnnounceEvent	Event	Handle Count - 2, Pointer	150098516895712
Windows	Directory	Handle Count - 1, Pointer	206220927431552
GLOBAL??	Directory	Handle Count - 2, Pointer	206220891814608
SETUP_LAUNCH_CLAIMED	Event	Handle Count - 1, Pointer	150098519863872
RPC Control	Directory	Handle Count - 0, Pointer	206220927428672
PdcPort	ALPC Port	Handle Count - 1, Pointer	150098455169936
EFSInitEvent	Event	Handle Count - 1, Pointer	150098463590368

Abilities

WinObjGui is built on top of Explorer so everything you can do using [Explorer](#), you can do with the WinObjGui Window:

- *Integrate with Struct Analyzer* – WinObjGui can integrate with [Struct Analyzer](#) using right click on some struct and then click Struct Analyzer will result a new Struct Analyzer Window of the corresponding struct



FileScanGui

Introduction

FileScanGui is a UI plugin which allows you to view all the files in a Windows Explorer-Like UI. It also shows data pulled from the user assist (so you can know what files a certain user runs).

In addition to all the Explorer function you will also be able to integrate with Struct Analyzer and analyze `_FILE_OBJECT` structs and also dump files/directories or view a file's HexDump.

Motivation

While working on the [VolExp](#) plugin, I realized that VolExp is not enough.

During a regular system analysis, when using Process Explorer\Hacker you are also using another UI that Microsoft has created, which also seems to get the least credit – Windows Explorer, since it increases comfortability and ease of use tremendously. So I decide to create one.

It could help the user to navigate directly to the directory/file they want to and view/dump it easily.

Startup Arguments

This plugin accepts the following arguments:

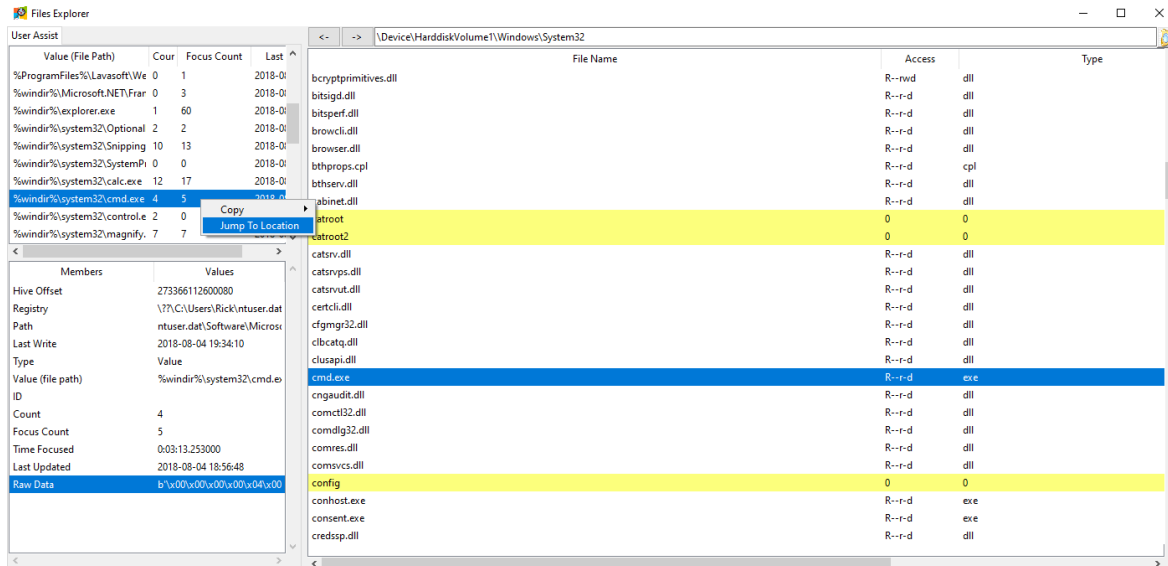
- `--DUMP-DIR`: Directory to dump files.
- `--GET-DICT`: File path to output all the filescan data, ordered by directories, in a pickle format (this argument is optional, and is used by [volexp](#). If this argument is present, then no GUI will appear).

Some Research

My research for this plugin was realizing how to dump a file from memory to disk.

This is not something new and it was present in Volatility2 as well as in Rekall - but not in

Volatility3, so I had to recreate it in my plugin. So I create this functionality using Volatility2's dumpfiles and some additional resources (Supports dump from both `_control_area` and `_shared_cache_map`).

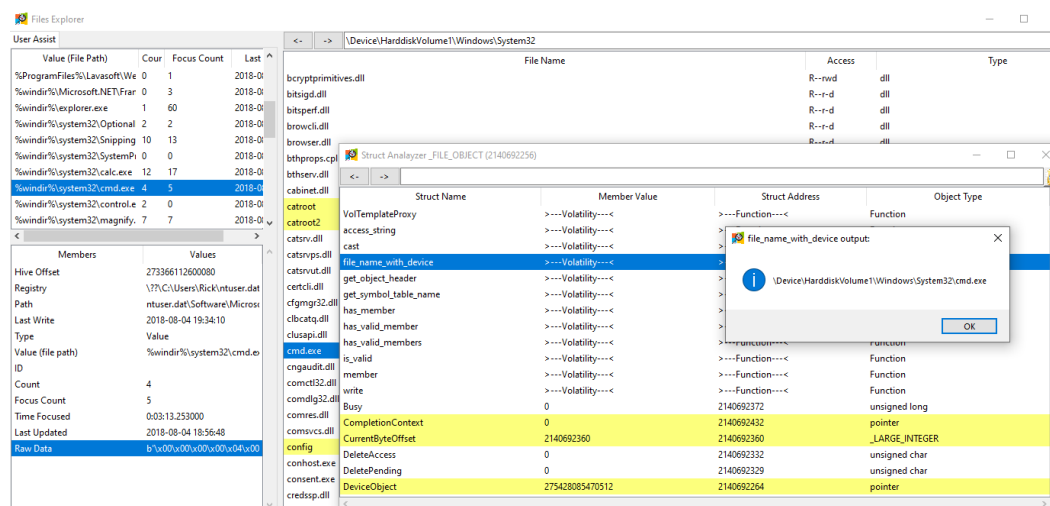


Abilities

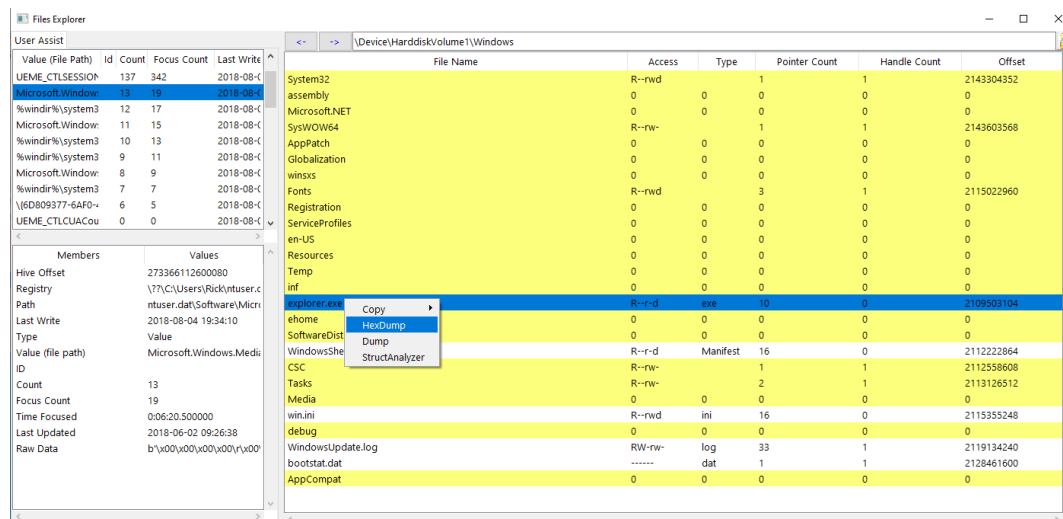
FileScanGui is built on top of Explorer so everything you can do using [Explorer](#), you can do with the FileScanGui Window:

- **Integrate with Struct Analyzer** – FileScanGui can integrate with Struct Analyzer using right click on a file and then clicking Struct Analyzer will result a new Struct Analyzer Window of the corresponding `_FILE_OBJECT`

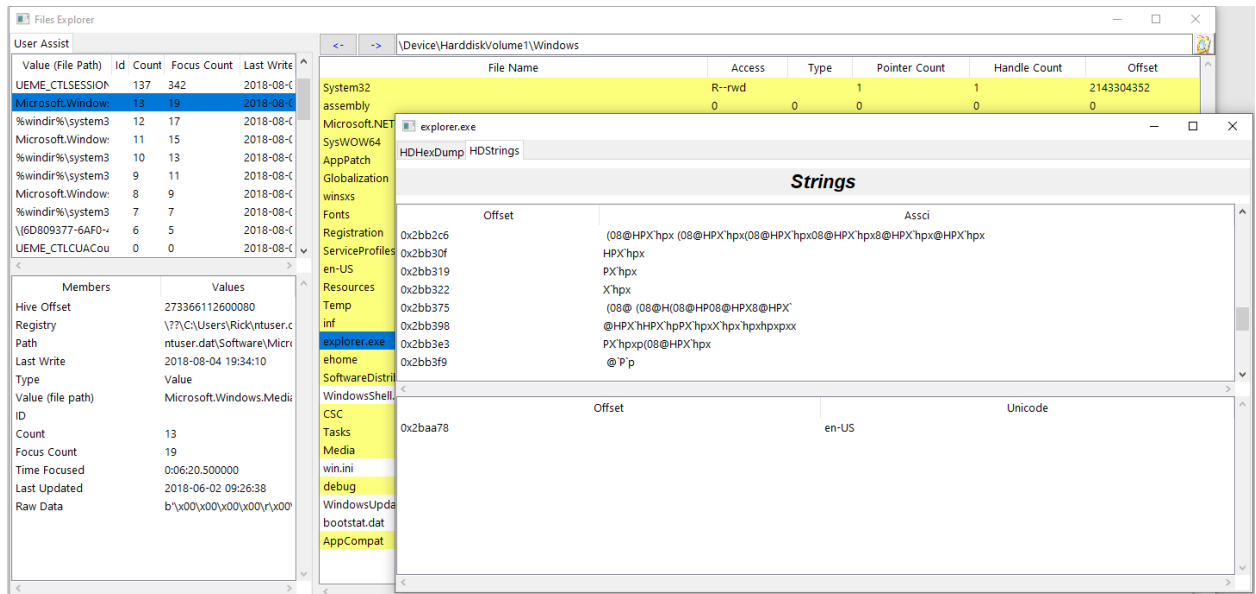
For example, running Struct Analyzer on `cmd.exe` and running the `file_name_with_device` function inside the Struct Analyzer window:



- **Dump File/Directory** – FileScanGui also has a feature to dump files or directories using right click and "Dump" or "HexDump":



Let's see HexDump Strings view for example:



Pfn.py

Motivation

Most of memory forensics tools will create a view of something from the virtual address space of a process. Most, like Volatility, Rekall, etc. have a built-in function to translate virtual address to physical address, like the processor always does. But except for rammap from SysInternals there is not a working tool (that works on newer versions of Windows as well) that can translate physical address to virtual address or can give us forensics artifacts based on a given physical address.

Rekall does have this ability, but it fully works only for Windows version 8.0 and prior, and only on 64bit systems.

I realize that Windows still needs the ability to translate physical address to virtual address so I decide to take the challenge, do a lot of research and create this plugin (fully supported from Windows XP to Windows 10!), I decided to make it a suite of plugins: P2V, PFNInfo (that uses P2V) and rammap, that uses them both.

Rammap will give information for each physical page in the memory dump, so make sure to use the 64bit version of python if you run rammap on a large memory dump.

P2V – Physical to Virtual

Intruduction

This plugin translates a physical address to a virtual one using the pfndb.

It also tries to find the owner of the page, as well as if there is any file mapped to this address - and if there is, which processes mapped it.

Startup Arguments:

This plugin accepts only one argument:

- -- ADDRESS: the physical address to translate.

```
c:\volE>vol.py -f C:\Users\██████\Downloads\OtterCTF\OtterCTF.vmem windows.pfn.P2V -
-ADDRESS 0x20000
Volatility 3 Framework 1.2.1-beta.1
Progress: 41.02 Scanning primary2 using PdbSignatureScanner
Owner Loading To File Name File Offset Physical Virtual
WebCompanionIn (3880), WebCompanion.e (3856) 3880, 3856 \Device\HarddiskVol
ume1\Windows\assembly\NativeImages_v2.0.50727_32\mscorlib\62a0b3e4b40ec0e8c5cfaa0c8
848e64a\mscorlib.ni.dll 0x399400 0x20000 0x7336b000, 0x7336b000
c:\volE>
```

Some research

To understand how this plugin works we need to understand the _MMPFN struct.

To help you better understand, let's define PFN as "VAD for the physical layer" – it's a struct that has a lot of data about the physical page.

```
(primary2) >>> dt("_MMPFN")
nt_symbols1!_MMPFN (48 bytes)
 0x0 : u1 nt_symbols1!_unnamed_152f
 0x8 : u2 nt_symbols1!_unnamed_1531
 0x10 : Lock nt_symbols1!long
 0x10 : PteAddress nt_symbols1!pointer
 0x10 : PteLong nt_symbols1!unsigned long long
 0x10 : VolatilePteAddress nt_symbols1!pointer
 0x18 : u3 nt_symbols1!_unnamed_1536
 0x1c : UsedPageTableEntries nt_symbols1!unsigned short
 0x1e : VaType nt_symbols1!unsigned char
 0x1f : ViewCount nt_symbols1!unsigned char
 0x20 : AweReferenceCount nt_symbols1!long
 0x20 : OriginalPte nt_symbols1!_MMPTE
 0x28 : u4 nt_symbols1!_unnamed_153e
```

This struct doesn't come alone. The OS has an array of _MMPFNs that we can find in MmPfnDatabase (a pointer to the start of the array).

The first entry of the array describes the first physical page, the second describe the second and

so on... In short, the array length = ram size / page_size. In 64 bit systems this struct 0x30 bytes large, meaning that for every page the size of 0x1000, 0x30 of memory will be immediately taken.

Thus, this struct takes more 1% from your memory, and never pages out, but there are some nonvalid _MMPFNs and we can check whether a PFN is valid or not using the MiPfnBitMap (on some versions).

```
addr = nt.get_symbol('MiPfnBitMap').address
rtl = nt.object("_RTL_BITMAP", addr)

# Check if the page index not out of range
if rtl.SizeOfBitMap < pfn_index:
    return False

# Check if the page is valid
c_byte = context.layers['primary'].read(rtl.Buffer + (pfn_index >> 3), 1)[0]
if c_byte & (2 ** (pfn_index % 8)) == (2 ** (pfn_index % 8)):
    return True
return False
```

Or check the _MMPFN.u4.PfnExists flag (on Windows versions that the bitmap is not present).

So how do we make use of this struct?

The OS uses this struct to track the physical pages – whether the page is in use, the address of the page, the virtual(_MMPFN.PteAddress) and physical address(_MMPFN.u4.PteFrame << PAGE_BITS | (_MMPFN.PteAddress & 0xFFF)) of the PTE (that manages this page), the OriginalPte (to handle soft page faults) and much more.

So if each _MMPFN contains information about the PTE that manages the page, we can just go to the PTE that manages our _MMPFN, the PTE that manages the PTE that manages the _MMPFN and so on... Basically, we have to reverse the entire steps of the virtual address translation.

After we go through all that, we end in the DirectoryTableBase of the process, so I created a dictionary that maps DTB address to the process and follows this routine, which allows me to get both the virtual address and the process that mapped this page.

To make sure that this works, let's go to volshell and check the values from the example on the start:

```
c:\volE>vol.py -f C:\Users\... \Downloads\OtterCTF\OtterCTF.vmem windows.pfn.P2V -
-ADDRESS 0x20000
Volatility 3 Framework 1.2.1-beta.1
Progress: 41.02 Scanning primary2 using PdbSignatureScanner
Owner Loading To File Name File Offset Physical Virtual
WebCompanionIn (3880), WebCompanion.e (3856) 3880, 3856 \Device\HarddiskVol
ume1\Windows\assembly\NativeImages_v2.0.50727_32\mscorlib\62a0b3e4b40ec0e8c5cfaa0c8
848e64a\mscorlib.ni.dll 0x399400 0x20000 0x7336b000, 0x7336b000
c:\volE>

(primary2) >>> pr.UniqueProcessId
3880
(primary2) >>> layer = pr.add_process_layer()
(primary2) >>> context.layers[layer].translate(0x7336b000)
(131072, 'memory_layer')
(primary2) >>> hex(131072)
'0x20000'
(primary2) >>> _
```

And we got what we expected, the address 0x7336b000 translates to the physical address 0x20000 that we checked before (in the example on top).

But I actually lied to you guys, the example above didn't work in the algorithm explained above, because that method only works for pages that map to a specific process; it would not work for shared pages - if we try the method above we end up with the System (4) process, not with PID3880 and PID3856. So how did I get this process?

When more than one process wants to use the same page, the OS creates a "prototype" PTE (for files for example). We can check whether a PFN points to a prototype PTE using the `_MMPFN.u3.PrototypePte` flag. This type of PTE points to a `_SUBSECTION` struct.

By iterating through all the processes and collecting all the subsections that loaded into each process, we can then determine which process belongs to each subsection we find.

```

def map_process_vads_subsections(self):
    """
    get all vads and subsections from a processes
    """
    self.subsections = {}
    self.vads = {}
    # Go all over the processes and collect all the subsections.
    for c_proc in pslist.Pslist.list_processes(context = self.context,
                                              layer_name = self.config['primary'],
                                              symbol_table = self.config['nt_symbols']):
        # Go all over vads.
        for vad in c_proc.get_vad_root().traverse():
            if hasattr(vad, "Subsection") and vad.Subsection:
                try:
                    subsection = vad.Subsection
                    seen_subs = []
                    # Walk the subsection list
                    while subsection not in seen_subs and subsection:
                        seen_subs.append(subsection)
                        start_addr = subsection.SubsectionBase.real

                        # Vads should not be in userspace
                        if start_addr > self.HighestUserAddress:
                            end_addr = start_addr + (subsection.PtesInSubsection * self.ntkrnlmp.get_type('_MMPTTE').size)
                            range = (start_addr, end_addr)
                            if not range in self.subsections:
                                self.subsections[range] = []
                                self.subsections[range].append((c_proc, vad, subsection))
                            subsection = subsection.NextSubsection
                        except exceptions.InvalidAddressException:
                            pass
                    if not int(c_proc.UniqueProcessId) in self.vads:
                        self.vads[int(c_proc.UniqueProcessId)] = []
                    self.vads[int(c_proc.UniqueProcessId)].append((vad.get_start(), vad.get_end(), vad))

```

(The code above shows how we map all the subsection ranges to the self.subsection dictionary.)

Abilities

- Translate physical address to virtual address
- Find the processes that map a given physical address
- Find a file mapped in a given physical address (if present)

PFNInfo – page frame number information

Introduction

As its name suggests, its main goal is to bring as much information as possible from a given physical address using the PFN Database.

Motivation

After creating P2V, RAMMap from SysInternals was yet stronger than my own tool. I needed to find much more information, like the use of a page, page priority, page list, etc...

For that reason, I decided to create a new plugin with this goal in mind: To give additional information about the page, not just the owner and the virtual address like P2V already does, but additional possible information we can get.

Startup Arguments

This plugin accepts one the following arguments:

- --ADDRESS: Address of the _MMPFN struct
- --INDEX: Index of the _MMPFN struct (inside the array [the page frame number])
- --PAGE: Address of a page

Note: --INDEX and --PAGE are relatively similar. If you use --PAGE={address} you can use --INDEX={address}/PAGE_SIZE.

Some research

In the previous plugin ([P2V](#)), a part of the research was to investigate the _MMPFN struct and how the OS uses it. Here, I had to do much more research, in different Windows versions, such as how to extract the page list, priority, reference and share count, page color(numa), PTE type, NxBit, etc.

I had to find how to locate general memory areas, like NonPagePool and PagePool (for session space as well) in order to identify the use of a page.

```

c:\vol>vol.py -f C:\Users\...Downloads\OtcCtf\OtcCtf.vmem windows.pfn.pfninfo --INDEX 100000
Volatility 3 Framework 1.2.1-beta.1
Progress: 41.02 Scanning primary2 using PdbSignatureScanner
Pfn Index      Pfn Address      Page List      Priority      Reference      Share Count      Page Color      Pte Type      Nxbt  Use      File Name      Offset  Image  Pool Tags
100000 18446738026480398848 Active 5      1      1      0      Prototype PTE  1      Mapped File  \Device\HarddiskVolume1\Windows\System32\winevt\Logs\Microsoft-Windows-Application-Experience\Program-Compatibi
lity-Troubleshooter.evtx 65536 False ([],)
c:\vol>

```

Abilities

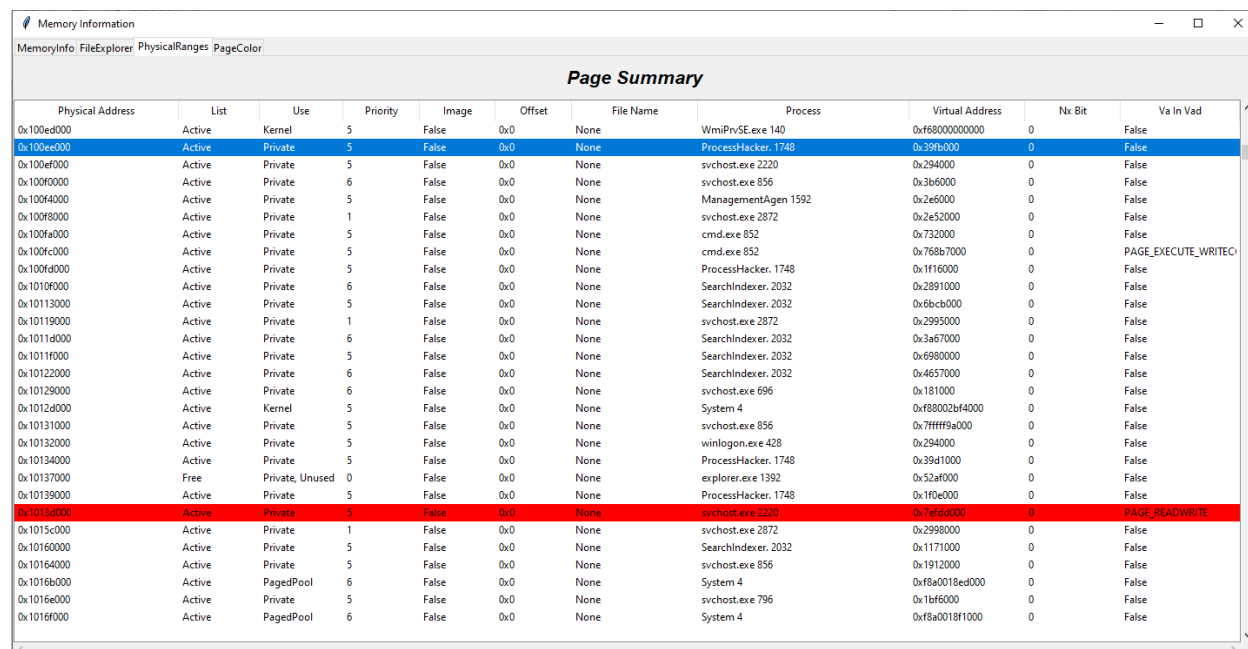
This plugin lists a lot of information related to a physical page, including:

PFN index, PFN address, page list, priority, reference and share count, page color(numa), PTE type, NxBit, the use of a page, file name (if any), file offset, is the file an image and pool tags (scan the physical address from built in pool tags list)

RAMMap – Random Access Memory Mapping

Introduction

This plugin is the reason that P2V and PFNInfo exists. RAMMap uses them both to get every information we can get from a given physical page, in addition to attempting to mark pages it finds suspicious.



Physical Address	List	Use	Priority	Image	Offset	File Name	Process	Virtual Address	Nx Bit	Va In Vad
0x100ed000	Active	Kernel	5	False	0x0	None	WmiPrvSE.exe 140	0xf68000000000	0	False
0x100ee000	Active	Private	5	False	0x0	None	ProcessHacker. 1748	0x39fb000	0	False
0x100ef000	Active	Private	5	False	0x0	None	svchost.exe 2220	0x294000	0	False
0x100f0000	Active	Private	6	False	0x0	None	svchost.exe 856	0x3b6000	0	False
0x100f4000	Active	Private	5	False	0x0	None	ManagementAgen 1592	0x2e6000	0	False
0x100f8000	Active	Private	1	False	0x0	None	svchost.exe 2872	0x2e52000	0	False
0x100fa000	Active	Private	5	False	0x0	None	cmd.exe 852	0x732000	0	False
0x100fc000	Active	Private	5	False	0x0	None	cmd.exe 852	0x768b7000	0	PAGE_EXECUTE_WRITECOM
0x100fd000	Active	Private	5	False	0x0	None	ProcessHacker. 1748	0x1f16000	0	False
0x1010f000	Active	Private	6	False	0x0	None	SearchIndexer. 2032	0x2891000	0	False
0x10113000	Active	Private	5	False	0x0	None	SearchIndexer. 2032	0x6bcb000	0	False
0x10119000	Active	Private	1	False	0x0	None	svchost.exe 2872	0x2995000	0	False
0x1011d000	Active	Private	6	False	0x0	None	SearchIndexer. 2032	0x3a67000	0	False
0x1011f000	Active	Private	5	False	0x0	None	SearchIndexer. 2032	0x6980000	0	False
0x10122000	Active	Private	6	False	0x0	None	SearchIndexer. 2032	0x4657000	0	False
0x10129000	Active	Private	6	False	0x0	None	svchost.exe 696	0x181000	0	False
0x1012d000	Active	Kernel	5	False	0x0	None	System 4	0xf88002b4000	0	False
0x10131000	Active	Private	5	False	0x0	None	svchost.exe 856	0x7ffff9a000	0	False
0x10132000	Active	Private	5	False	0x0	None	winlogon.exe 428	0x294000	0	False
0x10134000	Active	Private	5	False	0x0	None	ProcessHacker. 1748	0x39d1000	0	False
0x10137000	Free	Private, Unused	0	False	0x0	None	explorer.exe 1392	0x52af000	0	False
0x10139000	Active	Private	5	False	0x0	None	ProcessHacker. 1748	0x1f0e000	0	False
0x1013a000	Active	Private	5	False	0x0	None	svchost.exe 2220	0x7ef4d000	0	PAGE_READWRITE
0x1015c000	Active	Private	1	False	0x0	None	svchost.exe 2872	0x2998000	0	False
0x10160000	Active	Private	5	False	0x0	None	SearchIndexer. 2032	0x1171000	0	False
0x10164000	Active	Private	5	False	0x0	None	svchost.exe 856	0x1912000	0	False
0x1016b000	Active	PagedPool	6	False	0x0	None	System 4	0xf8a0018ed000	0	False
0x1016e000	Active	Private	5	False	0x0	None	svchost.exe 796	0x1bf6000	0	False
0x1016f000	Active	PagedPool	6	False	0x0	None	System 4	0xf8a0018f1000	0	False

(Colored in red because the VAD is marked as PAGE_READWRITE but the NxBit is unset).

Startup Arguments

This plugin accepts the following arguments:

- --ADDRESS: Starting address (defaults to 0)
- --SIZE: The size to analyze (the default is the memory dump size minus the starting address)
- --COLORED: Display only colored pages (defaults to false)

Some Research

I researched and looked for “weird” page states, categorizing them and giving each of them a different color.

For example, it will catch a conflict between the VAD protection to the PTE protection.

```
NDLE WINAPI LoadRemoteLibraryR( HANDLE hProcess, LPVOID lpBuffer, DWORD dwLeng
    lpRemoteLibraryBuffer = VirtualAllocEx( hProcess, NULL, dwLength, MEM_RESERVE|MEM_COMMIT, PAGE_EXECUTE_READWRITE );
    lpRemoteLibraryBuffer = VirtualAllocEx(hProcess, NULL, dwLength, MEM_RESERVE | MEM_COMMIT, PAGE_READONLY);
    DWORD oldPrection;
    VirtualProtectEx(hProcess, lpRemoteLibraryBuffer, dwLength, PAGE_EXECUTE_READWRITE, &oldPrection);
```

#BHEU 🐦@BLACKHATEVENTS

(Taken from BlackHat 2019, an hiding technique from plugins that check the VAD’s protection, like MalFind)

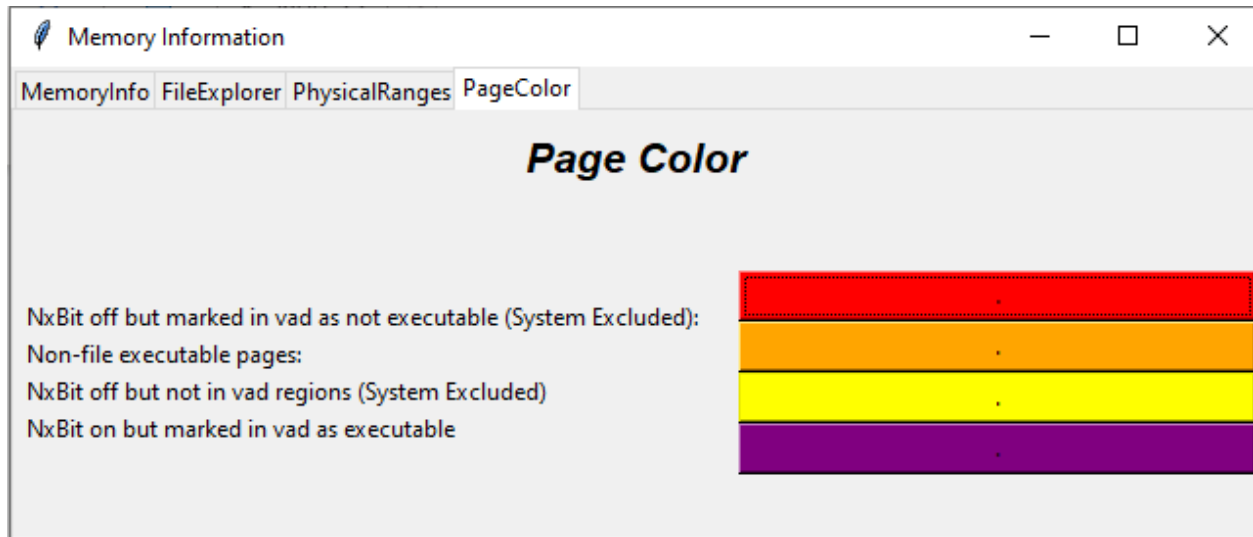
RAMMap will catch this technique and color it in red (if NxBit is off but the VAD shows it has no execute rights). For the opposite (NxBit is set but VAD shows it has execute rights), it will color in purple by default.

During the research I have found that on some Windows versions the OS will map a file with a PAGE_EXECUTE_WRITECOPY permission, but set the NxBit (so Windows will unset it manually when the CPU throw an exception).

This is acceptable because when DEP is disabled and the CPU throws an exception that a code attempted to run from a non-executable page (because the NxBit is on) - the OS handles the exception by unsetting the NxBit, so the processor could run this - which could indicate malicious injection into a process where DEP is disabled.

Abilities

You can always change the default colors by using the PageColor tab (you can also disable the coloring by changing the color to white like the other tabs)



- Color pages as you can see in the picture above
- Find if page is a part of a file (and VAD protection).
- Find the process that loaded a physical page (if it is a mapped file, then it will find all the processes that mapped this page).
- Get additional information like how the system uses this page (Use tab), the page priority, NxBit state and much more (as you can see in the picture above).

WinObj.py

Contains only one plugin: WinObj

WinObj – windows (kernel) objects

Introduction

The WinObj plugin parses the entire Windows object directory.

Motivation

After I created VolExp, while working on FileScanGui I was looking for a way to translate “harddrivevolume{number}” to the drive letter, and what is a better way than the way Windows itself does it?

So I was able to do so using WinObj (<https://github.com/kslggroup/WinObj>), but after using it a bit, I realized that this plugin works very well, but could sometimes miss some objects from the object manager because it doesn't parse it correctly so not all the data is present.

Thus, I decided to update the plugin. I also noticed that it has a lot of exceptions in Windows 10, so I fixed that as well.

Startup Arguments:

This plugin accepts the following arguments:

- --PARSE-ALL: Parse the entire directory (Boolean)
- --SUPPLY-ADDR: Parse _OBJECT_DIRECTORY at a specific address
- --FULL-PATH: Parse the specified directory from the object directory

Some Research

Firstly, I was looking for a better way to find the object directory, and since Volatility3 parses the kernel symbols as well, we should just take the ObpRootDirectoryObject symbol which holds a pointer to the root directory of the object manager (and if it failed to get it this way, which should never happen, we can try taking this pointer from the kdbg instead).

```

def get_root_directory(self):
    """
    :return
    : a pointer to the root directory
    """
    # gets the pointer
    # if for some reason ObpRootDirectoryObject not exist lets take the value from ObpRootDirectoryObject
    try:
        import struct
        _pointer_struct = struct.Struct("<Q") if self.ntkrnlmp.get_type('pointer').size == 8 else struct.Struct('<I')
        root_dir_addr = int(_pointer_struct.unpack(
            self.context.layers['primary'].read(self.ntkrnlmp.get_symbol('ObpRootDirectoryObject').address + self.ntkrnlmp.offset, 8))[0])
    except:
        root_dir_addr = info.Info.get_kdbg_structure(self.context, self.config_path, self.config['primary'], self.config['nt_symbols']).ObpRootDirectoryObject
        root_dir_addr = self.ntkrnlmp.object("pointer", offset=root_dir_addr - self.ntkrnlmp.offset)
    return root_dir_addr

```

Secondly, we parse the directory as we should parse it, as an `_OBJECT_DIRECTORY` struct, and make sure that we don't follow the same pointer twice (could be a while true problem in broken images):

```

def parse_directory(self, addr, l):
    """
    :param addr
    : long, pointer the the driectory
    :param l
    : list
    :return
    : None
    the function will parse the directory and add every valid object to the received list
    """
    seen = set()
    layer_name = self.config['primary']
    kvo = self.context.layers[layer_name].config["kernel_virtual_offset"]
    directory_array = self.ntkrnlmp.object('_OBJECT_DIRECTORY', addr - self.ntkrnlmp.offset)
    for pointer_addr in directory_array.HashBuckets:
        if not pointer_addr or pointer_addr == 0xffffffff:
            continue

        # Walk the ChainLink foreach item inside the directory.
        while pointer_addr not in seen:
            try:
                myObj = self.ntkrnlmp.object("pointer", offset=pointer_addr+self.POINTER_SIZE - kvo)
                self.AddToList(myObj, l)
            except exceptions.InvalidAddressException:
                pass

            seen.add(pointer_addr)
            try:
                pointer_addr = pointer_addr.ChainLink
            except exceptions.InvalidAddressException:
                break
            if not pointer_addr:
                break

```

Abilities

- Example of listing drivers from the Driver directory inside the root directory, finding some very old malware drivers (Stuxnet):

```
c:\volE>vol.py -f C:\sample007.bin windows.winobj --FULL-PATH \Driver
Volatility 3 Framework 1.2.1-beta.1
Progress: 0.00 Scanning primary2 using PdbSignatureScanner
Object Address(V) Name str Additional Info
0x820d7418 Beep Driver Full Name: \Driver\Beep
0x81ea1ca0 NDIS Driver Full Name: \Driver\NDIS
0x8239a240 KSecDD Driver Full Name: \Driver\KSecDD
0x8210cf38 Raspti Driver Full Name: \Driver\Raspti
0x81ed39e8 es1371 Driver Full Name: \Driver\es1371
0x8210cdc0 Mouclass Driver Full Name: \Driver\Mouclass
0x8232f4d0 vmx_svga Driver Full Name: \Driver\vmx_svga
0x82077dc8 Fips Driver Full Name: \Driver\Fips
0x82282b00 Kbdclass Driver Full Name: \Driver\Kbdclass
0x82079410 VgaSave Driver Full Name: \Driver\VgaSave
0x822a9750 NDPProxy Driver Full Name: \Driver\NDPProxy
0x8229db98 Compbatt Driver Full Name: \Driver\Compbatt
0x81edd030 Ptilink Driver Full Name: \Driver\Ptilink
0x822ec8c0 MountMgr Driver Full Name: \Driver\MountMgr
0x81da6f38 wdmaud Driver Full Name: \Driver\Wdmaud
0x822a3168 dmload Driver Full Name: \Driver\dmload
0x823c8278 isapnp Driver Full Name: \Driver\isapnp
0x8208a030 redbook Driver Full Name: \Driver\redbook
0x82282150 vmmouse Driver Full Name: \Driver\vmmouse
0x82320308 atapi Driver Full Name: \Driver\atapi
0x81ea1a08 vm SCSI Driver Full Name: \Driver\vm SCSI
0x82061880 RasAcid Driver Full Name: \Driver\RasAcid
0x81ebcea0 PSched Driver Full Name: \Driver\PSched
0x8226c710 dmio Driver Full Name: \Driver\dmio
0x820786e8 NPF Driver Full Name: \Driver\NPF
0x821a12c0 IpNat Driver Full Name: \Driver\IpNat
0x82214c68 audstub Driver Full Name: \Driver\audstub
0x81ee1c28 usbuhci Driver Full Name: \Driver\usbuhci
0x81d9c978 mouhid Driver Full Name: \Driver\mouhid
0x821a2b10 Win32k Driver Full Name: \Driver\Win32k
0x82305a68 swenum Driver Full Name: \Driver\swenum
0x8208b798 rdpr Driver Full Name: \Driver\rdpr
0x82085f38 usbbus Driver Full Name: \Driver\usbbus
0x821a1458 RDPDOD Driver Full Name: \Driver\RDPDOD
0x822f3e48 Update Driver Full Name: \Driver\Update
0x82277850 RasPppoe Driver Full Name: \Driver\RasPppoe
0x81ed3da0 usbccgp Driver Full Name: \Driver\usbccgp
0x81e7cc08 HTTP Driver Full Name: \Driver\HTTP
0x82305f38 TermDD Driver Full Name: \Driver\TermDD
0x8233cee8 Ftdisk Driver Full Name: \Driver\Ftdisk
0x8205ab10 sysaudio Driver Full Name: \Driver\sysaudio
0x82214518 Rasl2tp Driver Full Name: \Driver\Rasl2tp
0x8208b0d0 Fdc Driver Full Name: \Driver\Fdc
0x82325030 ParVdm Driver Full Name: \Driver\ParVdm
0x8232fb60 vmci Driver Full Name: \Driver\vmci
0x81e816c8 PptpMiniport Driver Full Name: \Driver\PptpMiniport
0x81ee15b0 vmxnet Driver Full Name: \Driver\vmxnet
0x8208b620 serenum Driver Full Name: \Driver\serenum
0x823afce8 WMIxMDM Driver Full Name: \Driver\WMIxMDM
0x822e8160 ACPI_HAL Driver Full Name: \Driver\ACPI_HAL
0x81f26870 MRxCLs Driver Full Name: \Driver\MRxCLs
0x820e54f8 MRxNet Driver Full Name: \Driver\MRxNet
0x820dba18 NetBT Driver Full Name: \Driver\NetBT
0x8226c0a8 agp440 Driver Full Name: \Driver\agp440
0x823058d8 Cdrom Driver Full Name: \Driver\Cdrom
```

- Example of results in vol3 - 1.2.1 (Windows XP) for driverscan and drivers (so the only way to get correct results at the moment is my way):

```
c:\volE>vol.py -f C:\sample007.bin windows.driverscan
Volatility 3 Framework 1.2.1-beta.1
Progress: 0.00 Scanning primary2 using PdbSignatureScanner
Offset Start Size Service Key Driver Name Name
c:\volE>
```

```
c:\volE>vol.py -f C:\sample007.bin windows.drivers
Volatility 3 Framework 1.2.1-beta.1
Progress: 0.00 Scanning primary2 using PdbSignatureScanner
Offset Start Size Service Key Driver Name Name
c:\volE>
```


- Using WinObj, you can enumerate everything from the object directory itself:
As we saw already, you can enumerate drivers, but there is much more you can do. You can enumerate devices, callbacks, mutexes, events, semaphores, jobs, ALPC ports (including the WSL ALPC ports, from \PSXSS), symbolic links, section objects, windows stations, sessions, desktops, etc..
You can recreate almost any plugin results from WinObj:
For example wintree (from volatility2) -> using WinObj to parse \Windows\WindowStations and find all the windowstations objects, then parse all the processes from that window, which will show all the information that wintree gives us.
- The kernel uses the object directory to translate objects as well so it could be very unsafe for a malware to change data there and try to hide, because it could result in unwanted changes.

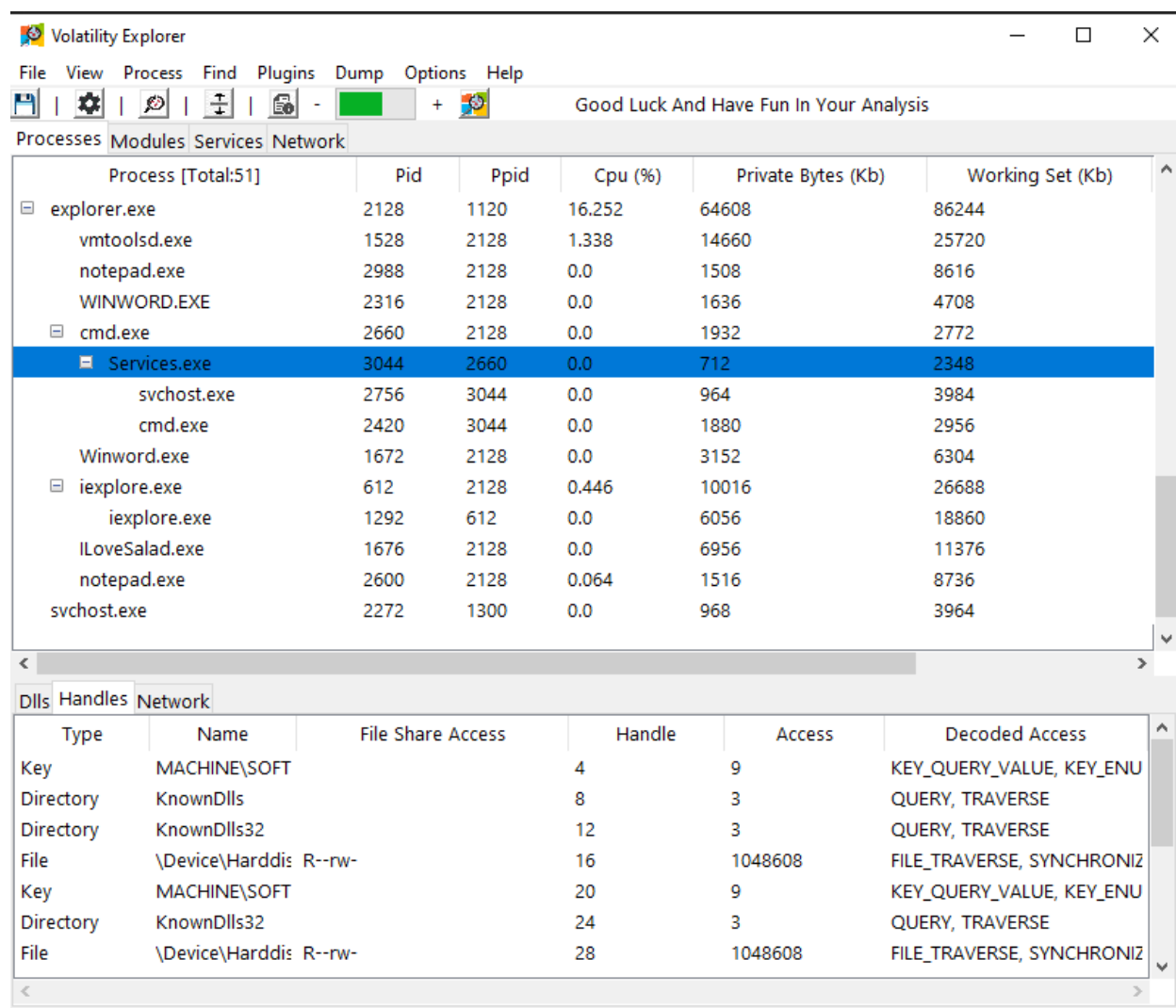
Using the tools

Instructions

In this demo I am going to use two different dumps. One that has been analyzed before (<https://drive.google.com/file/d/0B7v1Owo0v5SYZ016VmVoVFV1eIE/view>), the KSLSample.vmem - the sample of the threadmap plugin in the contest of 2017, as well as sahar.vmem, which I am not allowed to upload.

Sahar.vmem

This memory image belongs to a cloud VMWare user that downloaded a malicious program.



The screenshot shows the Volatility Explorer application window. The 'Processes' tab is selected, displaying a list of processes. The 'Services.exe' process is highlighted. Below the process list, the 'Handles' tab is selected, showing a list of handles.

Process [Total:51]	Pid	Ppid	Cpu (%)	Private Bytes (Kb)	Working Set (Kb)
explorer.exe	2128	1120	16.252	64608	86244
vmtoolsd.exe	1528	2128	1.338	14660	25720
notepad.exe	2988	2128	0.0	1508	8616
WINWORD.EXE	2316	2128	0.0	1636	4708
cmd.exe	2660	2128	0.0	1932	2772
Services.exe	3044	2660	0.0	712	2348
svchost.exe	2756	3044	0.0	964	3984
cmd.exe	2420	3044	0.0	1880	2956
Winword.exe	1672	2128	0.0	3152	6304
iexplore.exe	612	2128	0.446	10016	26688
iexplore.exe	1292	612	0.0	6056	18860
ILoveSalad.exe	1676	2128	0.0	6956	11376
notepad.exe	2600	2128	0.064	1516	8736
svchost.exe	2272	1300	0.0	968	3964

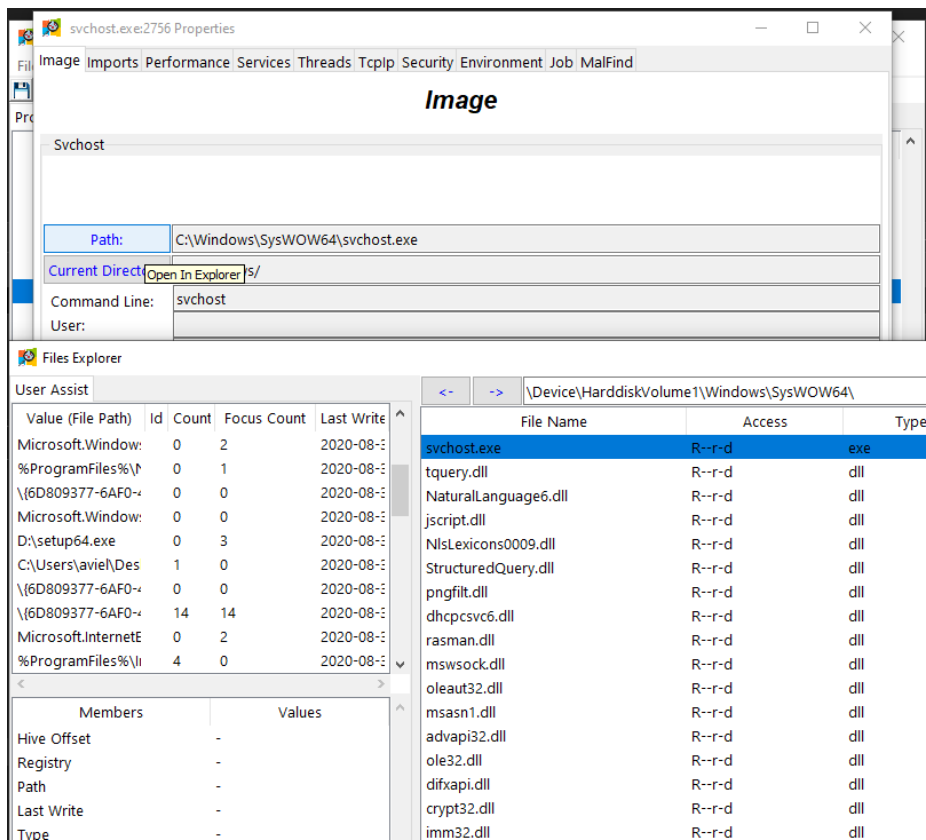
Type	Name	File Share Access	Handle	Access	Decoded Access
Key	MACHINE\SOFT		4	9	KEY_QUERY_VALUE, KEY_ENU
Directory	KnownDlls		8	3	QUERY, TRAVERSE
Directory	KnownDlls32		12	3	QUERY, TRAVERSE
File	\Device\Harddis R--rw-		16	1048608	FILE_TRAVERSE, SYNCHRONIZ
Key	MACHINE\SOFT		20	9	KEY_QUERY_VALUE, KEY_ENU
Directory	KnownDlls32		24	3	QUERY, TRAVERSE
File	\Device\Harddis R--rw-		28	1048608	FILE_TRAVERSE, SYNCHRONIZ

From the process tree already, we can see a lot of weird stuff going on here.

Services.exe is running under a cmd, svchost is running under Services, two instances of differently spelled WinWord variants (lower and upper case).

Opening svchost.exe's (2756) properties and clicking at Path to open in Explorer validated that we are in fact dealing with the authentic svchost image.

In that case, what is going on here??



To make sure, Let's run Malfind against this image (you can run Malfind on one specific process by using Process->Plugins->WellKnown->Malfind)

CmdPlugin

\Python\Python38\python.exe "c:\vole\vol.py" -p "C:\vole\volatility\plugins" -f "C:\sahar.vmem" windows.malfind.Malfind

Apply (on properties) ☒ Alert processes (Ctrl+U to unalert) Clear Screen Run-->>>

Pid	Process	Start Vpn	End Vpn	Tag	Protection	Committed Charge
1580	svchost.exe	0x2490000	0x250ffff	VadS	PAGE_EXECUTE_READWRITE	128
1580	svchost.exe	0x4d30000	0x4e2ffff	VadS	PAGE_EXECUTE_READWRITE	256
2128	explorer.exe	0x2a60000	0x2a60fff	VadS	PAGE_EXECUTE_READWRITE	1
2128	explorer.exe	0x3940000	0x394ffff	VadS	PAGE_EXECUTE_READWRITE	16
2272	svchost.exe	0x2e0000	0x2edfff	VadS	PAGE_EXECUTE_READWRITE	14
2916	svchost.exe	0x2e0000	0x2edfff	VadS	PAGE_EXECUTE_READWRITE	14
2756	svchost.exe	0x2e0000	0x2edfff	VadS	PAGE_EXECUTE_READWRITE	14
612	ieexplore.exe	0xbf0000	0xbf1fff	VadS	PAGE_EXECUTE_READWRITE	2
612	ieexplore.exe	0x3cd0000	0x3cd0fff	VadS	PAGE_EXECUTE_READWRITE	1
612	ieexplore.exe	0x5fff0000	0x5fffffff	VadS	PAGE_EXECUTE_READWRITE	16
1292	ieexplore.exe	0x830000	0x831fff	VadS	PAGE_EXECUTE_READWRITE	2
1292	ieexplore.exe	0x5fff0000	0x5fffffff	VadS	PAGE_EXECUTE_READWRITE	16
2348	SearchFilterHo	0xb40000	0xbbffff	VadS	PAGE_EXECUTE_READWRITE	2

Using the Apply (on properties) button will color the svchost in yellow:

Volatility Explorer

File View Process Find Plugins Dump Options Help

Good Luck And Have Fun In Your Analysis

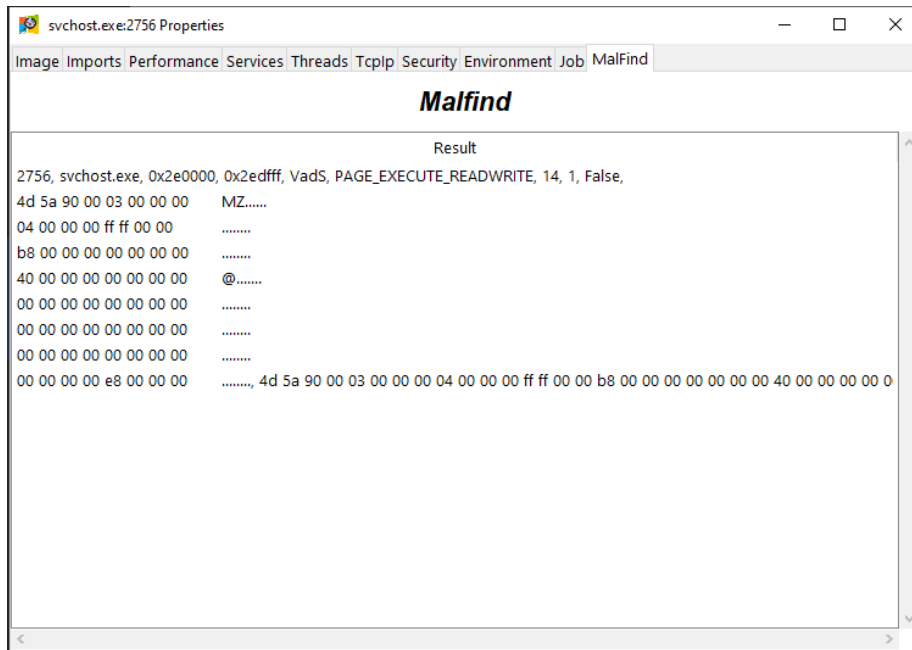
Processes Modules Services Network

Process [Total:51]	Pid	Ppid	Cpu (%)	Private Bytes (Kb)	Working Set (Kb)
explorer.exe	2128	1120	16.252	64608	86244
vmtoolsd.exe	1528	2128	1.338	14660	25720
notepad.exe	2988	2128	0.0	1508	8616
WINWORD.EXE	2316	2128	0.0	1636	4708
cmd.exe	2660	2128	0.0	1932	2772
Services.exe	3044	2660	0.0	712	2348
svchost.exe	2756	3044	0.0	964	3984
cmd.exe	2420	3044	0.0	1880	2956
Winword.exe	1672	2128	0.0	3152	6304
ieexplore.exe	612	2128	0.446	10016	26688
ieexplore.exe	1292	612	0.0	6056	18860
ILoveSalad.exe	1676	2128	0.0	6956	11376
notepad.exe	2600	2128	0.064	1516	8736
svchost.exe	2272	1300	0.0	968	3964

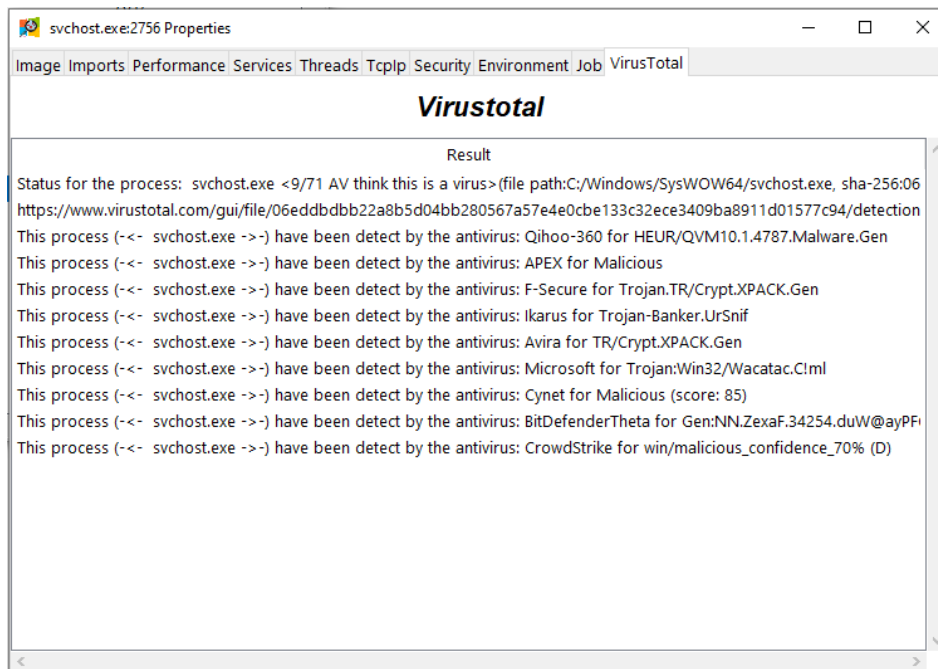
DLLs Handles Network

Type	Name	File Share Access	Handle	Access	Decoded Access
Key	MACHINE\SOFTWARE		4	9	KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS
Directory	KnType: Key				TRVERSE
File	Name: MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\IMAGE FILE EXECUTION OPTIONS		12	1048608	FILE_EXECUTE, SYNCHRONIZE
File	\Device\Harddisk R-rw-		16	1048608	FILE_EXECUTE, SYNCHRONIZE
Key	MACHINE\SYSTEM		20	131097	KEY_READ
Key	MACHINE		32	131097	KEY_READ
Key	MACHINE\SYSTEM		40	1	KEY_QUERY_VALUE

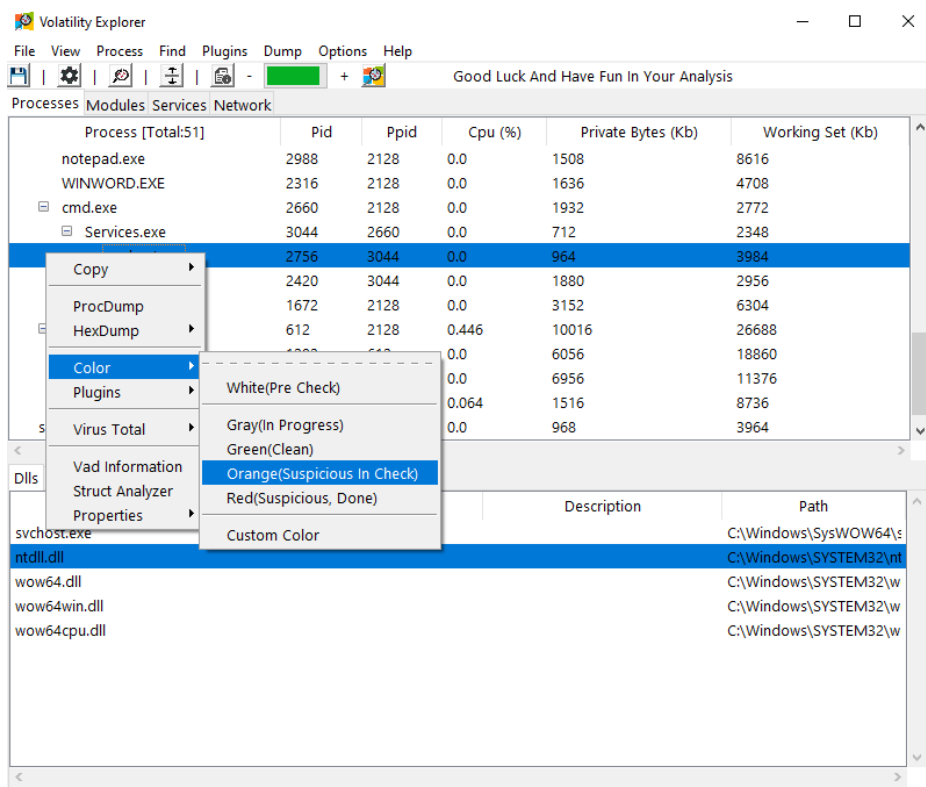
By looking at svchost's properties we can understand that we've got an MZ injection in our hands.



By uploading this process to VirusTotal we can see that nine different antiviruses detected this file as malicious (none of them detected the real svchost):

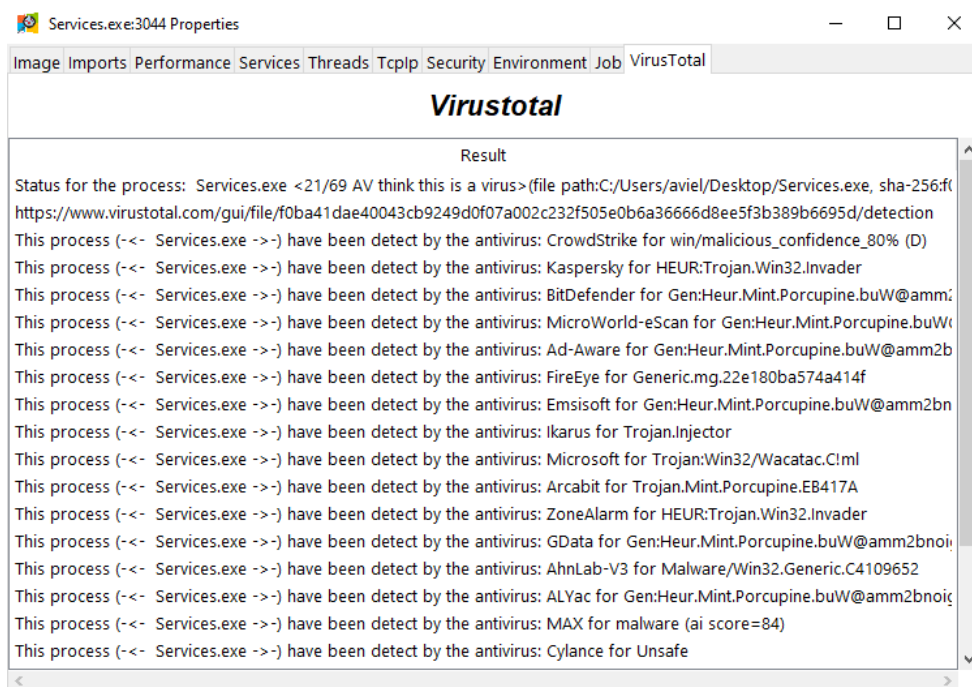


Let's add a comment on this in the Comment tab and color this process as suspicious

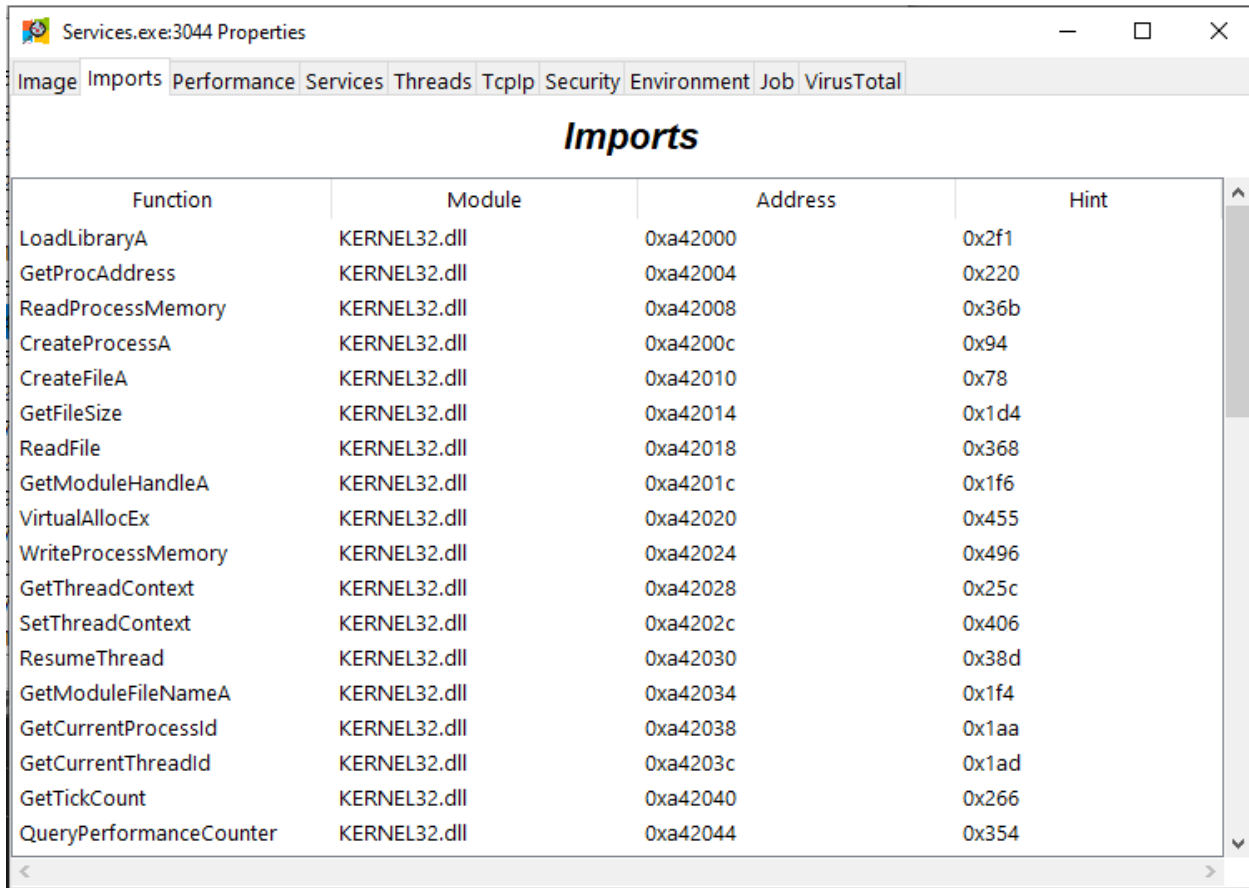


As we can see, this svchost loads only 4 dlls. Coincidentally, services.exe loads the same dlls exactly.

In that case, let's take a look at Services.exe:



A quick view of the process' imports, and we can already know what we are dealing with:



Function	Module	Address	Hint
LoadLibraryA	KERNEL32.dll	0xa42000	0x2f1
GetProcAddress	KERNEL32.dll	0xa42004	0x220
ReadProcessMemory	KERNEL32.dll	0xa42008	0x36b
CreateProcessA	KERNEL32.dll	0xa4200c	0x94
CreateFileA	KERNEL32.dll	0xa42010	0x78
GetFileSize	KERNEL32.dll	0xa42014	0x1d4
ReadFile	KERNEL32.dll	0xa42018	0x368
GetModuleHandleA	KERNEL32.dll	0xa4201c	0x1f6
VirtualAllocEx	KERNEL32.dll	0xa42020	0x455
WriteProcessMemory	KERNEL32.dll	0xa42024	0x496
GetThreadContext	KERNEL32.dll	0xa42028	0x25c
SetThreadContext	KERNEL32.dll	0xa4202c	0x406
ResumeThread	KERNEL32.dll	0xa42030	0x38d
GetModuleFileNameA	KERNEL32.dll	0xa42034	0x1f4
GetCurrentProcessId	KERNEL32.dll	0xa42038	0x1aa
GetCurrentThreadId	KERNEL32.dll	0xa4203c	0x1ad
GetTickCount	KERNEL32.dll	0xa42040	0x266
QueryPerformanceCounter	KERNEL32.dll	0xa42044	0x354

CreateProcessA, CreateFileA, GetFileSize, ReadFile, VirtualAllocEx, WriteProcessMemory, Get/SetThreadContext and ResumeThread is almost everything we need to perform a process hollowing injection.

If we take a look at the VAD information (using process Right Click -> VAD information):

Start	Size	Use	Pr	Control Flags
0x7ffe0000	0xffff	User Shared Data	PA	
0xa40000	0x5fff	\Users\...\Desktop\Services.exe	PA	Accessed, File, Image
0x76ad0000	0x45fff	\Windows\SysWOW64\KernelBase.dll	PA	File, Image
0x74540000	0x4bfff	\Windows\SysWOW64\apphelp.dll	PA	Accessed, File, Image
0x76d30000	0x10ffff	\Windows\SysWOW64\kernel32.dll	PA	Accessed, File, Image
0x775f0000	0x17ffff	\Windows\SysWOW64\ntdll.dll	PA	Accessed, File, Image
0x40000	0xfff	\Windows\System32\apisetschema.dll	PA	File, Image
0x430000	0x66fff	\Windows\System32\locale.nls	PA	Accessed, File
0x77410000	0x1a8fff	\Windows\System32\ntdll.dll	PA	Accessed, File, Image
0x746a0000	0x3efff	\Windows\System32\wow64.dll	PA	Accessed, File, Image

Members	Values
Start	0xa40000
End	0xa45fff
Size	0x5fff
Tag	Vad
Flags	'Protection': 7, 'VadType': 2
Protection	PAGE_EXECUTE_WRITECOPY
Type	2
Control Area	0xfa800348a010
Segment	0xfa800348a010

We can see that this process was loaded from the desktop, and also loads ntdll.dll dynamically (using the LoadLibraryA and GetProcAddress that we saw in the imports above).

Hah!, NtUnmapViewOfSection (from ntdll) is the only missing function to perform process hollowing.

We can also dump Services.exe and view it in our favorite disassembly tool to confirm.

Here it is creating a process named svchost:

```

loc_A41195:                ; lpProcessInformation
push     ebx
push     esi
push     0
push     0
push     4
push     0
push     0
push     0
push     0
push     offset CommandLine ; "svchost"
push     0
push     0
call     ds:CreateProcessA
mov     esi, [ebp]
test    esi, esi
jnz     short loc_A411C7

loc_A411C7:
push     ebp
push     esi
call     sub_A41800
push     100h
mov     edi, eax
call     ??2@YAPAXI@Z
mov     ebp, eax
add     esp, 8
test    ebp, ebp
jz      short loc_A411F5

```



```

mov     edi, [esp+48h+lpFileName]
add     esp, 0Ch
push    0                ; hTemplateFile
push    0                ; dwFlagsAndAttributes
push    4                ; dwCreationDisposition
push    0                ; lpSecurityAttributes
push    0                ; dwShareMode
push    80000000h        ; dwDesiredAccess
push    edi              ; lpFileName
call    ds:CreateFileA
mov     ebp, eax
cmp     ebp, 0FFFFFFFFh
jnz     short loc_A41261

```

Here it opens the malicious file that will be used to replace svchost:

```

push    offset ModuleName ; "ntdll"
call    ds:GetModuleHandleA
push    offset aNtunmapviewofs ; "NtUnmapViewOfSection"
push    eax                ; hModule
call    ds:GetProcAddress
mov     ecx, [esi+8]
mov     edx, [ebx]
push    ecx
push    edx
call    eax
test    eax, eax
jz      short loc_A412F3

```

It then unmaps svchost's sections:

```

mov     eax, [ebp+50h]
mov     ecx, [esi+8]
mov     edx, [ebx]
add     esp, 4
push    40h              ; flProtect
push    3000h            ; flAllocationType
push    eax              ; dwSize
push    ecx              ; lpAddress
push    edx              ; hProcess
call    ds:VirtualAllocEx
test    eax, eax
jnz     short loc_A41333

```

It continues to allocate additional memory:

```

mov     ecx, [esp+48h+Buffer]
mov     edx, [ecx+18h]
mov     ecx, [edi+edx+10h]
lea     eax, [edi+edx]
mov     edx, [eax+14h]
add     edx, [esp+48h+var_1C]
mov     eax, [ebx]
add     esp, 0Ch
push    0                ; lpNumberOfBytesWritten
push    ecx              ; nSize
push    edx              ; lpBuffer
push    ebp              ; lpBaseAddress
push    eax              ; hProcess
call    ds:WriteProcessMemory
test    eax, eax
jz      short loc_A4138A

```

Rewrite the headers:

```

mov     edx, [ebx+4]
add     esp, 4
push    esi              ; lpContext
push    edx              ; hThread
call    ds:GetThreadContext
test    eax, eax
jnz     short loc_A41626

```

Gets thread context:

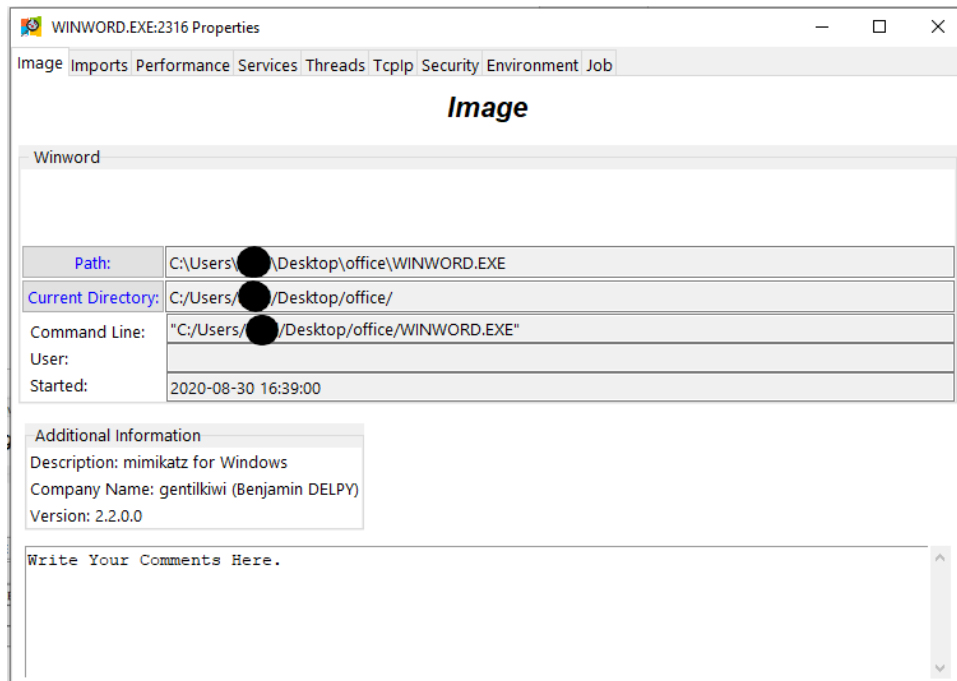
```

push    ecx              ; hThread
call    ds:ResumeThread
test    eax, eax
jnz     short loc_A41681

```

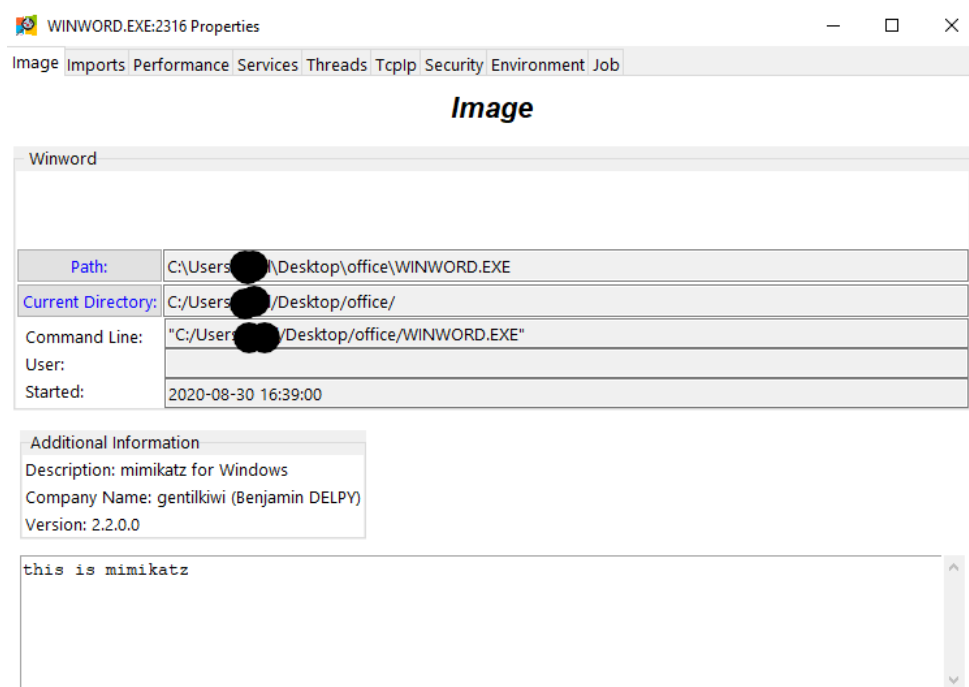
It then resumes the thread, and that's it.

If we also disassemble the svchost, we can find out that it does a lot of abnormal stuff as well, such as running one of our WinWord.exes. Let's look at it:



And just by viewing the file description we can immediately see that this is a mimikatz.

Adding a comment to make our analysis easier:



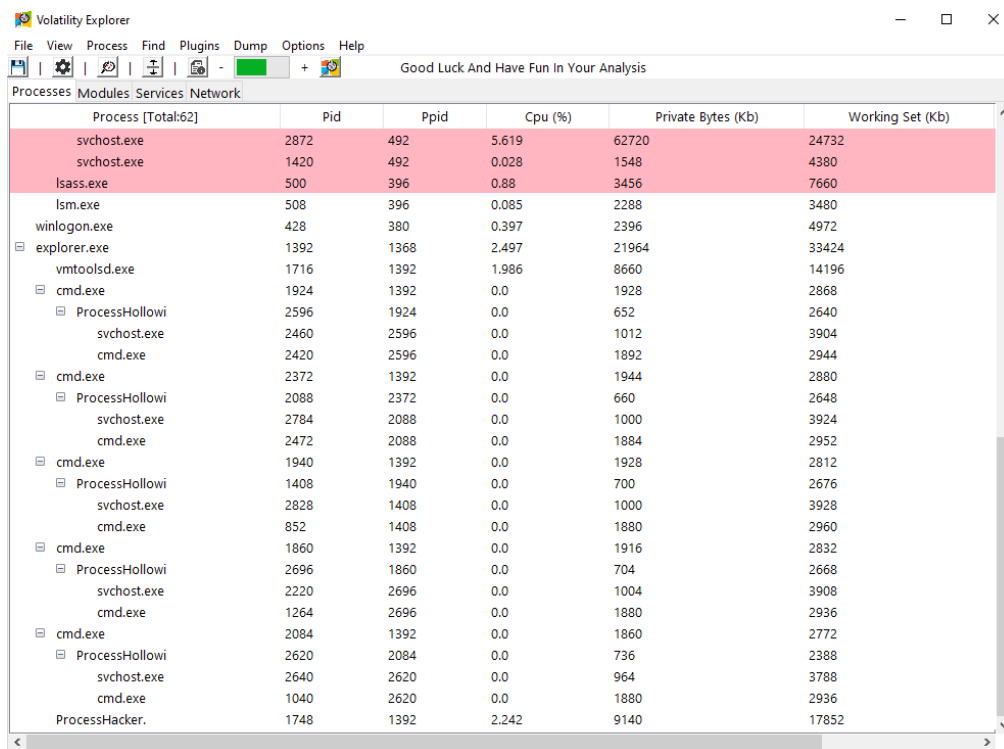
KSLSample.vmem

This is the dump used by kslgroup who created the threadmap.py plugin:

<https://drive.google.com/file/d/0B7v1Owo0v5SYZ016VmVoVFV1eIE/view>

Here, the process hollowing is trying to hide itself by allocating the memory as PAGE_READWRITE and changing it later to be executable.

Let's take a look in VolExp:



Process [Total:62]	Pid	Ppid	Cpu (%)	Private Bytes (Kb)	Working Set (Kb)
svchost.exe	2872	492	5.619	62720	24732
svchost.exe	1420	492	0.028	1548	4380
lsass.exe	500	396	0.88	3456	7660
lsm.exe	508	396	0.085	2288	3480
winlogon.exe	428	380	0.397	2396	4972
explorer.exe	1392	1368	2.497	21964	33424
vmttoolsd.exe	1716	1392	1.986	8660	14196
cmd.exe	1924	1392	0.0	1928	2868
ProcessHollowi	2596	1924	0.0	652	2640
svchost.exe	2460	2596	0.0	1012	3904
cmd.exe	2420	2596	0.0	1892	2944
cmd.exe	2372	1392	0.0	1944	2880
ProcessHollowi	2088	2372	0.0	660	2648
svchost.exe	2784	2088	0.0	1000	3924
cmd.exe	2472	2088	0.0	1884	2952
cmd.exe	1940	1392	0.0	1928	2812
ProcessHollowi	1408	1940	0.0	700	2676
svchost.exe	2828	1408	0.0	1000	3928
cmd.exe	852	1408	0.0	1880	2960
cmd.exe	1860	1392	0.0	1916	2832
ProcessHollowi	2696	1860	0.0	704	2668
svchost.exe	2220	2696	0.0	1004	3908
cmd.exe	1264	2696	0.0	1880	2936
cmd.exe	2084	1392	0.0	1860	2772
ProcessHollowi	2620	2084	0.0	736	2388
svchost.exe	2640	2620	0.0	964	3788
cmd.exe	1040	2620	0.0	1880	2936
ProcessHacker.	1748	1392	2.242	9140	17852

This is just a research dump, so they did not make any special effort to try and hide the process hollowing. But can Malfind find it?

CmdPlugin

.exe" "c:\vol\vol.py" -p "C:\vol\volatility\plugins" -f "C:\Users\avie\Downloads\KSLSample\KSLSample.vmem" windows.malfind.Malfind

Apply (on properties) ☒ Alert processes (Ctrl+U to unalert) Clear Screen Run-->>>

Pid	Process	Start Vpn	End Vpn	Tag	Protection	Commitcharge	Privatememory	Dump
1392	explorer.exe	0x2970000	0x297ffff	VadS	PAGE_EXECUTE_READWRITE	16	1	False
1392	explorer.exe	0x39b0000	0x39b0fff	VadS	PAGE_EXECUTE_READWRITE	1	1	False
2872	svchost.exe	0x2490000	0x250fff	VadS	PAGE_EXECUTE_READWRITE	128	1	False
2872	svchost.exe	0x4d30000	0x4e2fff	VadS	PAGE_EXECUTE_READWRITE	256	1	False
2640	svchost.exe	0x430000	0x43dfff	VadS	PAGE_EXECUTE_READWRITE	14	1	False

As we can see, Malfind only found that process 2640 is suspicious because it has started with the PAGE_EXECUTE_READWRITE rights. They probably did that as a way of comparison.

Let's see now how RAMMap can catch this as well.

Firstly, the plugin won't mark 2640 as suspicious because it has both the NXBit off and has Execute rights. RAMMap only marks pages with a weird state, like a page where the VAD protection is Non executable but the NXBit is not set (so the page can execute):

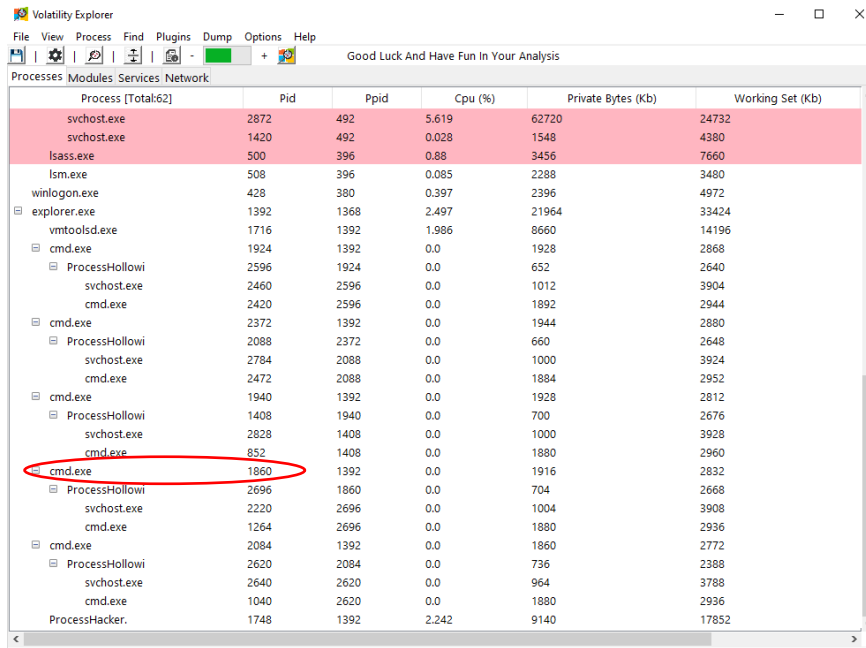
Memory Information

MemoryInfo FileExplorer PhysicalRanges PageColor

Page Summary

Physical Address	List	Use	Priority	Image	Offset	File Name	Process	Virtual Address	Nx Bit	Va In Vad
0x100ed000	Active	Kernel	5	False	0x0	None	WmiPrvSE.exe 140	0xf68000000000	0	False
0x100ee000	Active	Private	5	False	0x0	None	ProcessHacker. 1748	0x39fb000	0	False
0x100ef000	Active	Private	5	False	0x0	None	svchost.exe 2220	0x294000	0	False
0x100f0000	Active	Private	6	False	0x0	None	svchost.exe 856	0x3b6000	0	False
0x100f4000	Active	Private	5	False	0x0	None	ManagementAgen 1592	0x2e6000	0	False
0x100f8000	Active	Private	1	False	0x0	None	svchost.exe 2872	0x2e52000	0	False
0x100fa000	Active	Private	5	False	0x0	None	cmd.exe 852	0x732000	0	False
0x100fc000	Active	Private	5	False	0x0	None	cmd.exe 852	0x768b7000	0	PAGE_EXECUTE_WRITECOM
0x100fd000	Active	Private	5	False	0x0	None	ProcessHacker. 1748	0x1f16000	0	False
0x1010f000	Active	Private	6	False	0x0	None	SearchIndexer. 2032	0x2891000	0	False
0x10113000	Active	Private	5	False	0x0	None	SearchIndexer. 2032	0x6bcb000	0	False
0x10119000	Active	Private	1	False	0x0	None	svchost.exe 2872	0x2995000	0	False
0x1011d000	Active	Private	6	False	0x0	None	SearchIndexer. 2032	0x3a57000	0	False
0x1011f000	Active	Private	5	False	0x0	None	SearchIndexer. 2032	0x6980000	0	False
0x10122000	Active	Private	6	False	0x0	None	SearchIndexer. 2032	0x4657000	0	False
0x10129000	Active	Private	6	False	0x0	None	svchost.exe 696	0x181000	0	False
0x1012d000	Active	Kernel	5	False	0x0	None	System 4	0xf88002bf4000	0	False
0x10131000	Active	Private	5	False	0x0	None	svchost.exe 856	0x7ffff9a000	0	False
0x10132000	Active	Private	5	False	0x0	None	winlogon.exe 428	0x294000	0	False
0x10134000	Active	Private	5	False	0x0	None	ProcessHacker. 1748	0x39d1000	0	False
0x10137000	Free	Private, Unused	0	False	0x0	None	explorer.exe 1392	0x52af000	0	False
0x10139000	Active	Private	5	False	0x0	None	ProcessHacker. 1748	0x1f0e000	0	False
0x1013a000	Active	Private	5	False	0x0	None	svchost.exe 2220	0x2e6a000	0	PAGE_READWRITE
0x1015c000	Active	Private	1	False	0x0	None	svchost.exe 2872	0x2998000	0	False
0x10160000	Active	Private	5	False	0x0	None	SearchIndexer. 2032	0x1171000	0	False
0x10164000	Active	Private	5	False	0x0	None	svchost.exe 856	0x1912000	0	False
0x1016b000	Active	PagedPool	6	False	0x0	None	System 4	0xf8a0018ed000	0	False
0x1016e000	Active	Private	5	False	0x0	None	svchost.exe 796	0x1bf6000	0	False
0x1016f000	Active	PagedPool	6	False	0x0	None	System 4	0xf8a0018f1000	0	False

Here, we can see the PAGE_READWRITE as we expected, but the NxBit is unset, so this page can execute, meaning it is trying to hide, thus marking it in red.



Volatility Explorer

File View Process Find Plugins Dump Options Help

Good Luck And Have Fun In Your Analysis

Processes Modules Services Network

Process [Total:62]	Pid	Ppid	Cpu (%)	Private Bytes (Kb)	Working Set (Kb)
svchost.exe	2872	492	5.619	62720	24732
svchost.exe	1420	492	0.028	1548	4380
lsass.exe	500	396	0.88	3456	7660
lsmd.exe	508	396	0.085	2288	3480
winlogon.exe	428	380	0.397	2396	4972
explorer.exe	1392	1368	2.497	21964	33424
vmtoolsd.exe	1716	1392	1.986	8660	14196
cmd.exe	1924	1392	0.0	1928	2868
ProcessHollowi	2596	1924	0.0	652	2640
svchost.exe	2460	2596	0.0	1012	3904
cmd.exe	2420	2596	0.0	1892	2944
cmd.exe	2372	1392	0.0	1944	2880
ProcessHollowi	2088	2372	0.0	660	2648
svchost.exe	2784	2088	0.0	1000	3924
cmd.exe	2472	2088	0.0	1884	2952
cmd.exe	1940	1392	0.0	1928	2812
ProcessHollowi	1408	1940	0.0	700	2676
svchost.exe	2828	1408	0.0	1000	3928
cmd.exe	852	1408	0.0	1880	2960
cmd.exe	1860	1392	0.0	1916	2832
ProcessHollowi	2696	1860	0.0	704	2668
svchost.exe	2220	2696	0.0	1004	3908
cmd.exe	1264	2696	0.0	1880	2936
cmd.exe	2084	1392	0.0	1860	2772
ProcessHollowi	2620	2084	0.0	736	2388
svchost.exe	2640	2620	0.0	964	3788
cmd.exe	1040	2620	0.0	1880	2936
ProcessHacker.	1748	1392	2.242	9140	17852

Closing Statements

There used to be a time where the SysInternals tools were open source. You could understand the basics of how Windows manages its memory just by reading them. Those were easier times.

About 10 years ago they decided to make the SysInternals suite closed source. Since then, to understand the theoretics behind Windows you need to read, books like The Art of Memory Forensics and Windows Internals.

Then Windows 10 shows up and since a lot of things changed, Volatility starts to move to Volatility3, and Rekall is no longer supported... and I just wanted to understand how the memory manager changes in Windows 10 and the forensics aspects of it.

Therefore, I decided to create an improved open source tool of the most powerful tools from SysInternals (like Pavel Yosifovich started doing) , and added the memory analysis aspect to them.

In the beginning, I only created the VolExp suite for Volatility2 but it wasn't perfect.

My then mentor, Shachaf Atun, competed in the contest in previous years, so I decide to wait one year with the contest.

Later, you decided to make the contest for Volatility3, so I decided to convert the plugin and also add much more information to it, taking inspiration from the SysInternals tools.

This plugin was supposed to let other people understand the intricacy of the operating system's memory management, but I myself have learned a lot by making these plugins; just by running RAMMap and seeing the different colors in different OS versions makes you understand a bit. Slowly all the pieces combine together into the big puzzle, as I hope they will for others.

This last month of researching was intensive but I had a lot of fun, now I will recreate the friendships and connections that I have neglected, and back to normal.

Thanks to my lovely girlfriend for the understanding during this time.

I hope these tools will help you during your memory analysis.

References

1. Gitub: <https://github.com/memoryforensics1/Vol3xp>
2. VolExp for volatility 2: <https://github.com/memoryforensics1/VolExp>
3. Memory sample:
<https://drive.google.com/file/d/0B7v1Owo0v5SYZ016VmVoVFV1eIE/view>
4. Hale Ligh Michael, Case Andres, Levy Jamie, Walter Aaron, The Art of Memory Forensics, Wiley, 2014
5. Russinovich Mark, Solomon A. David, Ionescu Alex, Windows Internals 6th Edition, Microsoft Press, 2012
6. RAMMap: <https://docs.microsoft.com/en-us/sysinternals/downloads/rammap>
7. Process Explorer: <https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer>
8. WinObj: <https://docs.microsoft.com/en-us/sysinternals/downloads/winobj>
9. Threadmap plugin: <https://github.com/kslggroup/threadmap>
10. WinInternals tools: <https://github.com/zodiacon/AllTools>