# ESOF 422:
# Advanced Software Engineering: Cyber Practices

More Volatility

Reese Pearsall

Spring 2025

## Announcements

- HW6 will be posted very soon (sorry)
- We will try to work through some parts on Friday

You will need to have Kali Linux and Volatility installed before Friday's classtime (I'll post an installation video)

Final Exam: Wednesday May 7th 2:00 – 3:50 PM

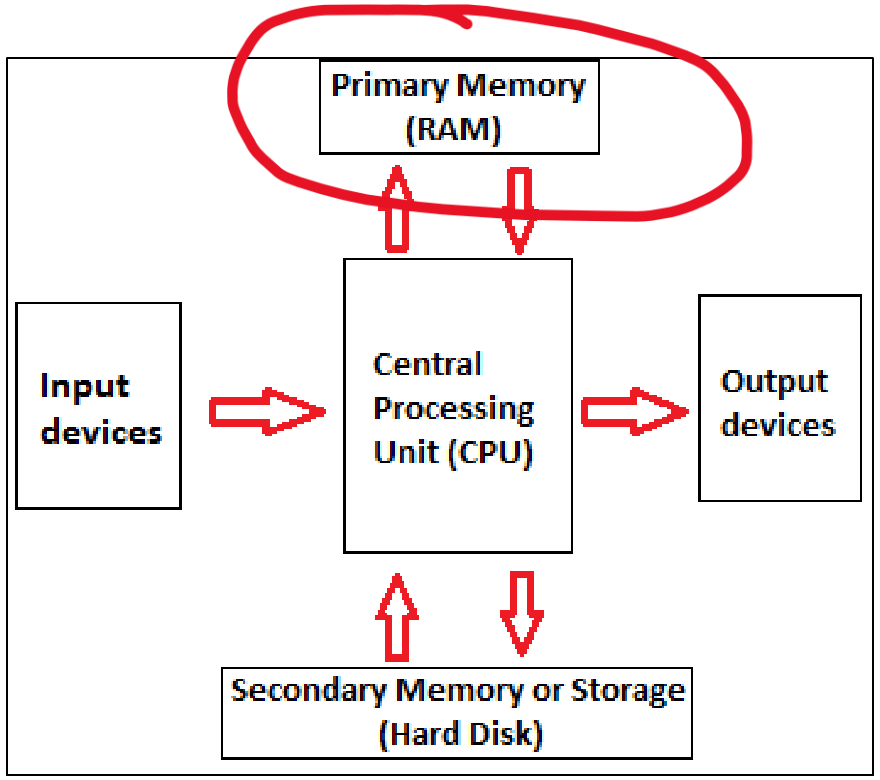
Students with accommodations: Final Exam should be registered at the testing center

## Memory Forensics

Analysis of data sources from a running system’s memory (RAM)

What does RAM contain?

- Programs and files that have been executed
- Running (and sometimes dead) processes
- What programs accessed what files
- Where opens files are/were location on disk
- Information from keyboard (passwords, emails, chats)
- Opened web pages
- Decrypted content
- Network connections
- Content no longer on disk
- Content that was never on disk


Primary Memory (RAM)
Input devices
Central Processing Unit (CPU)
Output devices
Secondary Memory or Storage (Hard Disk)

## Memory Forensics

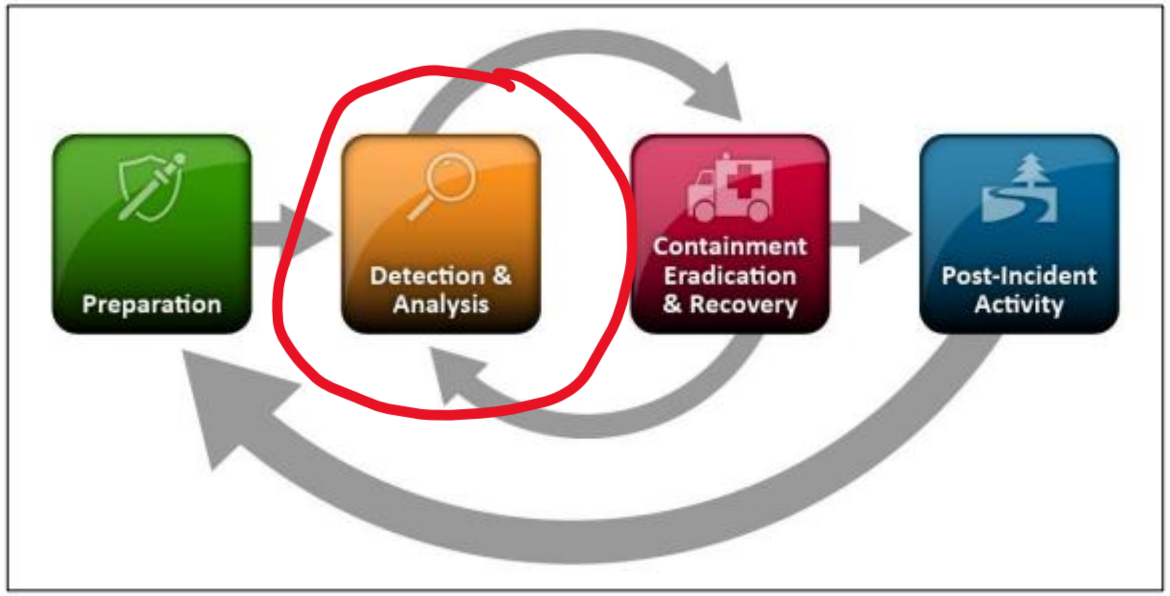


Figure 3-1. Incident Response Life Cycle

The stuff we are talking about for the remainder of the semester are parts of the **analysis** stage

## Volatility

**Volatility** is a popular, modular framework used for memory forensics

- Written in Python
- Works on memory images from Windows, Mac, and Linux systems
- Runs on Windows, Mac, and Linux
- Open source
- Extensible and scriptable API
- Lots of plugins and community modules

Volatility is not:
- A memory acquisition tool
- Not a GUI
- Bug-free
- Supportive of every single OS version

## Executed Code

Any program will typically be in the form of an `.exe` file or a `.dll` file

**`.exe` file**

`.exe` files run by itself, or can be ran by a user (double-click)

Starts its own process when launched

Every program has an `.exe` of some kind

**`.dll` file (Dynamic Linked Library)**

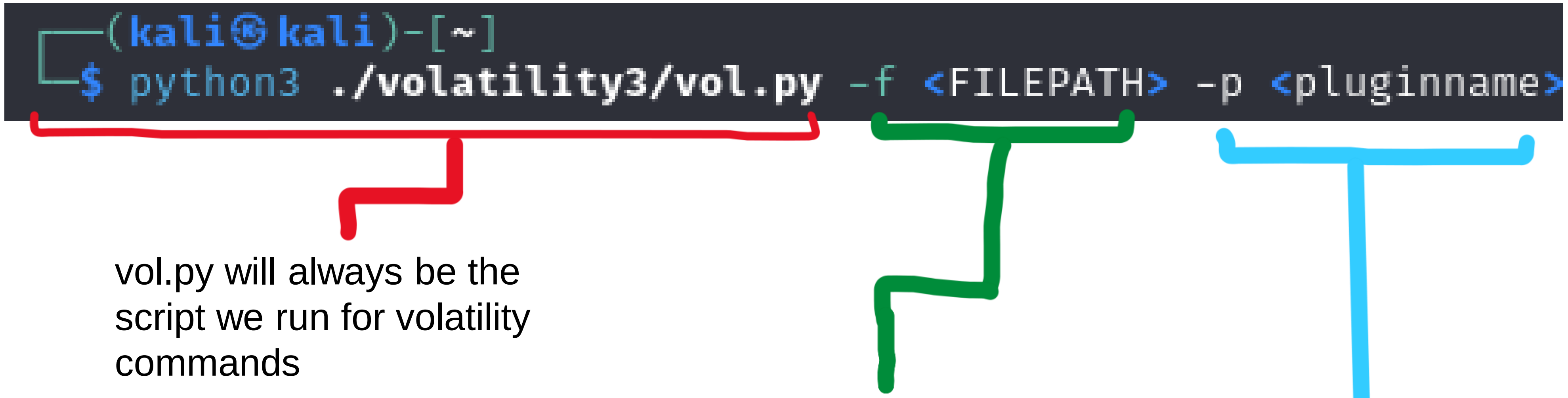
Cannot be ran by itself. Must be called by another process

Contains library code (reuseable code, classes, and objects)

Can be difficult to find usage in memory

Can be injected with malicious code (DLL Hijacking)

## Getting Started



```
┌──(kali㉿kali)-[~]
└─$ python3 ./volatility3/vol.py -f <FILEPATH> -p <pluginname>
```

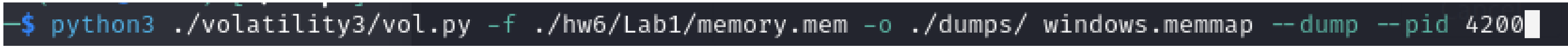
vol.py will always be the script we run for volatility commands

We provide the path to the `.mem` or `.vmem` files for analysis with the –f flag

The name of the plugin to run

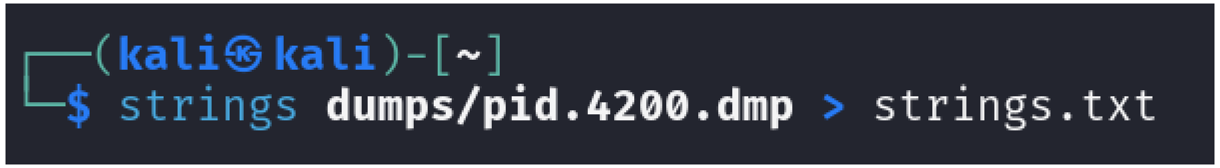
## Process Dumping

We can provide a process ID, and the memap plugin will dump the raw contents of the process space (this may take awhile)



```
─$ python3 ./volatility3/vol.py -f ./hw6/Lab1/memory.mem -o ./dumps/ windows.memmap --dump --pid 4200
```

There will be a lot of data (in hexadecimal) that is dumped. There are several different tools



```
┌──(kali㉿kali)-[~]
└─$ strings dumps/pid.4200.dmp > strings.txt
```

The **`strings`** command can be used to identify possible strings that existed in the process space

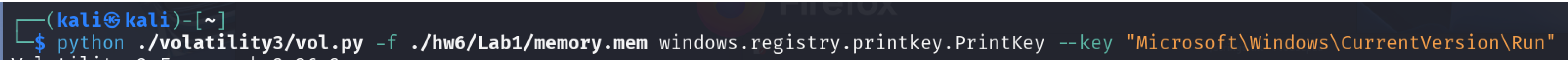
90% of the strings generated will likely be irrelevant, but some might provide some insight!

If a malicious payload is executed, that string should be located somewhere as a String

**Malware authors typically execute their code through some programming-level system call**

.exec(), .execve(), .popen()  .system()

# Printing Windows Registry Values



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab1/memory.mem windows.registry.printkey.PrintKey --key "Microsoft\Windows\CurrentVersion\Run"
```

**Software\Microsoft\Windows\CurrentVersion\Run** has items that execute when the user logs in

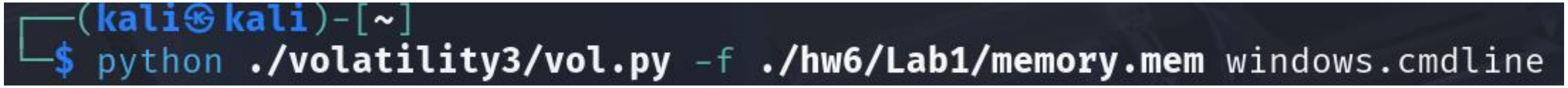
**Software\Microsoft\Internet Explorer\TypedURLs** has a list of typed URLs

**Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs** shows recently opened documents per file extension

**SYSTEM\CurrentControlSet\Control\DeviceClasses** shows detailed USB device information

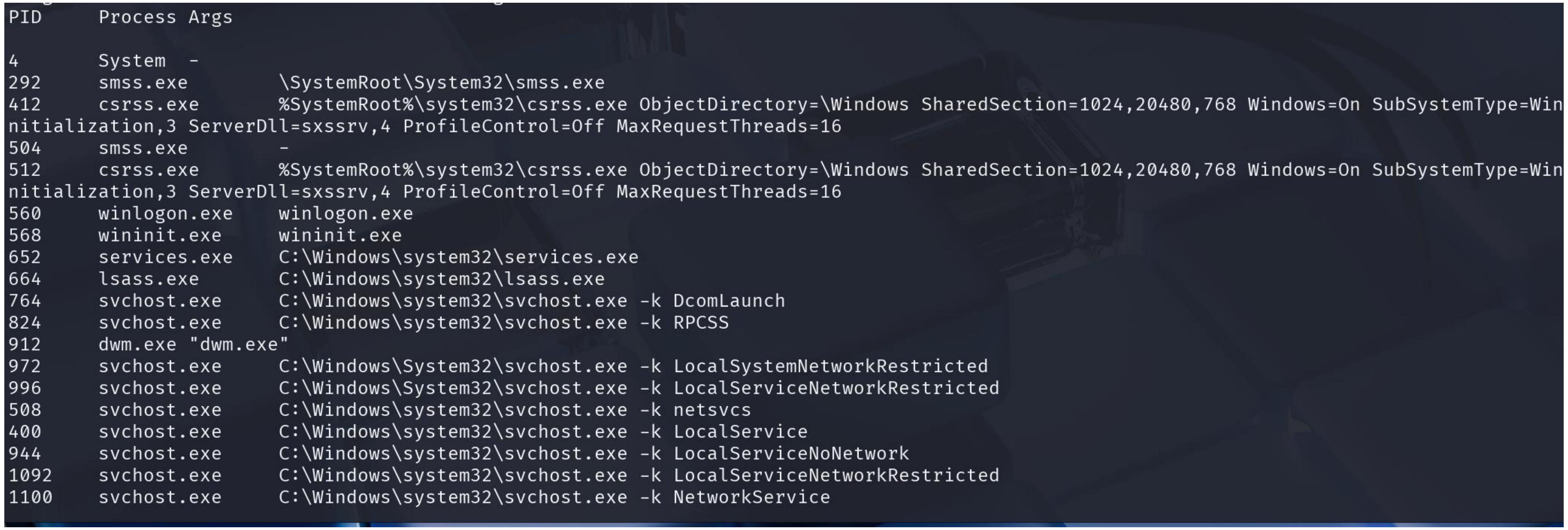
…and so much more. Some keys may not have a value yet

## Command Line History



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab1/memory.mem windows.cmdline
```

**`windows.cmdline`** is used to see how processes used the command line and arguments for commands



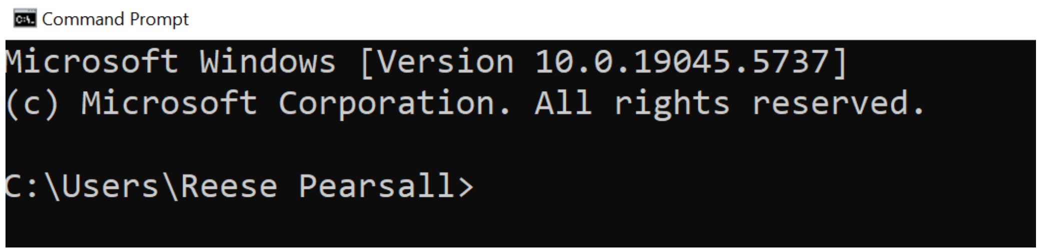
```
PID     Process Args

4       System  -
292     smss.exe        \SystemRoot\System32\smss.exe
412     csrss.exe       %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,20480,768 Windows=On SubSystemType=Win
nitialization,3 ServerDll=sxssrv,4 ProfileControl=Off MaxRequestThreads=16
504     smss.exe        -
512     csrss.exe       %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,20480,768 Windows=On SubSystemType=Win
nitialization,3 ServerDll=sxssrv,4 ProfileControl=Off MaxRequestThreads=16
560     winlogon.exe    winlogon.exe
568     wininit.exe     wininit.exe
652     services.exe    C:\Windows\system32\services.exe
664     lsass.exe       C:\Windows\system32\lsass.exe
764     svchost.exe     C:\Windows\system32\svchost.exe -k DcomLaunch
824     svchost.exe     C:\Windows\system32\svchost.exe -k RPCSS
912     dwm.exe "dwm.exe"
972     svchost.exe     C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted
996     svchost.exe     C:\Windows\System32\svchost.exe -k LocalServiceNetworkRestricted
508     svchost.exe     C:\Windows\system32\svchost.exe -k netsvcs
400     svchost.exe     C:\Windows\system32\svchost.exe -k LocalService
944     svchost.exe     C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork
1092    svchost.exe     C:\Windows\system32\svchost.exe -k LocalServiceNetworkRestricted
1100    svchost.exe     C:\Windows\system32\svchost.exe -k NetworkService
```

Many of it will be benign Windows services, but if a malicious process spawns a new process via command line, it will show up here

## Powershell and Command Line (`cmd`)

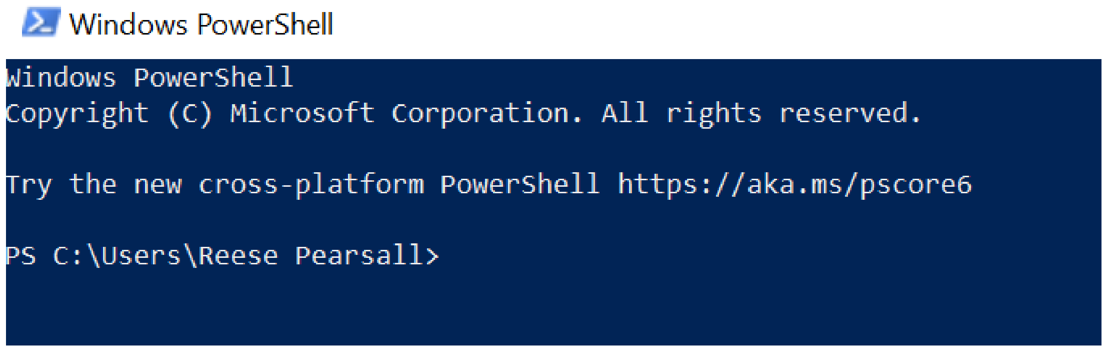
On Windows, Powershell and Command Line are both command-line interfaces, but are much different in design and power

**Command Line**
- Limited scripting capabilities
- Very old
- Cannot manage windows services/users
- More challenging to communicate with other processes



```
Microsoft Windows [Version 10.0.19045.5737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Reese Pearsall>
```

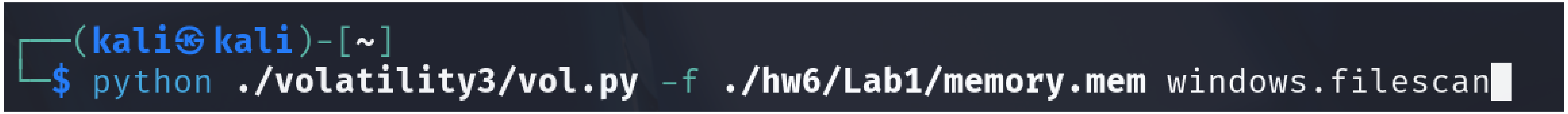
**Powershell**
- Robust scripting capabilities, access to .NET framework
- Modern
- Can manage windows services/users
- Able to communicate with processes easier



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

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Reese Pearsall>
```

For more complex tasks, malware authors will try to summon Powershell to execute their payload

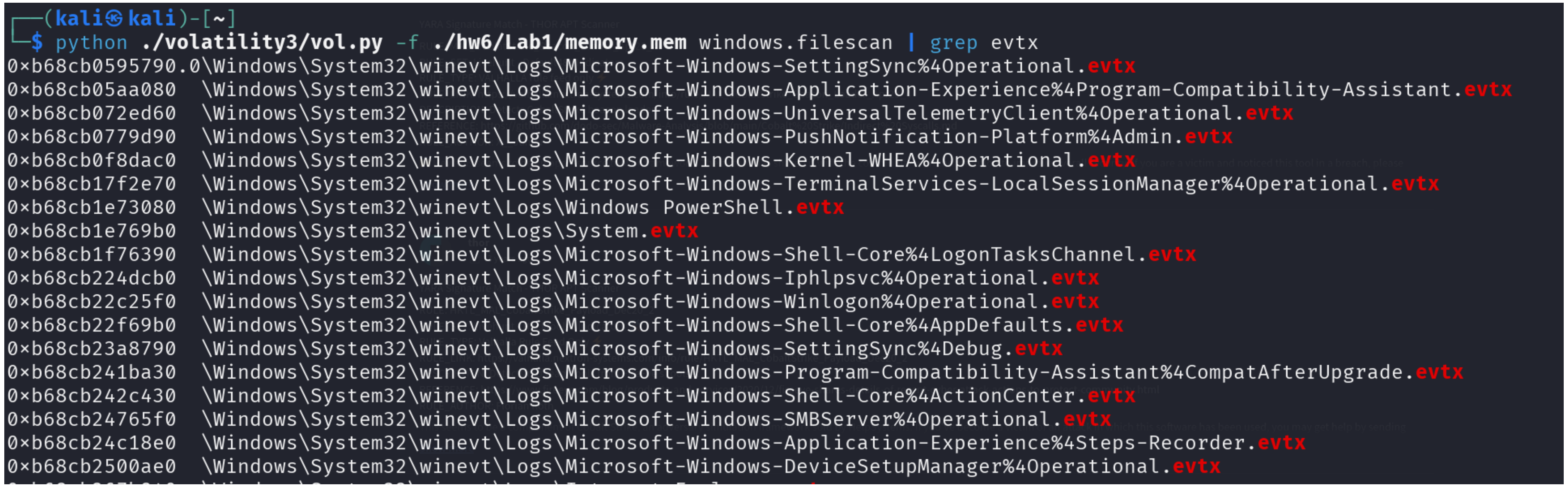
## File Scanning



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab1/memory.mem windows.filescan
```

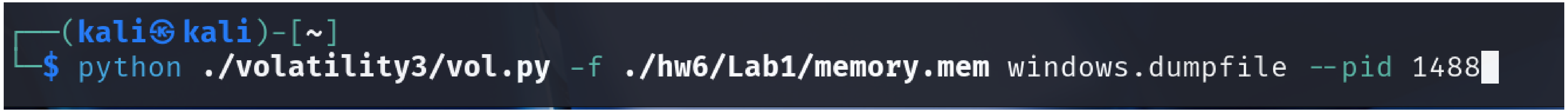
`windows.filescan` will scan for files that are found in the memory image

- exe files
- dll files
- Documents
- .evtx files (indows XML Event Log) – stores system log information



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab1/memory.mem windows.filescan | grep evtx
0xb68cb0595790.0\Windows\System32\winevt\Logs\Microsoft-Windows-SettingSync%4Operational.evtx
0xb68cb05aa080  \Windows\System32\winevt\Logs\Microsoft-Windows-Application-Experience%4Program-Compatibility-Assistant.evtx
0xb68cb072ed60  \Windows\System32\winevt\Logs\Microsoft-Windows-UniversalTelemetryClient%4Operational.evtx
0xb68cb0779d90  \Windows\System32\winevt\Logs\Microsoft-Windows-PushNotification-Platform%4Admin.evtx
0xb68cb0f8dac0  \Windows\System32\winevt\Logs\Microsoft-Windows-Kernel-WHEA%4Operational.evtx
0xb68cb17f2e70  \Windows\System32\winevt\Logs\Microsoft-Windows-TerminalServices-LocalSessionManager%4Operational.evtx
0xb68cb1e73080  \Windows\System32\winevt\Logs\Windows PowerShell.evtx
0xb68cb1e769b0  \Windows\System32\winevt\Logs\System.evtx
0xb68cb1f76390  \Windows\System32\winevt\Logs\Microsoft-Windows-Shell-Core%4LogonTasksChannel.evtx
0xb68cb224dcb0  \Windows\System32\winevt\Logs\Microsoft-Windows-Iphlpsvc%4Operational.evtx
0xb68cb22c25f0  \Windows\System32\winevt\Logs\Microsoft-Windows-Winlogon%4Operational.evtx
0xb68cb22f69b0  \Windows\System32\winevt\Logs\Microsoft-Windows-Shell-Core%4AppDefaults.evtx
0xb68cb23a8790  \Windows\System32\winevt\Logs\Microsoft-Windows-SettingSync%4Debug.evtx
0xb68cb241ba30  \Windows\System32\winevt\Logs\Microsoft-Windows-Program-Compatibility-Assistant%4CompatAfterUpgrade.evtx
0xb68cb242c430  \Windows\System32\winevt\Logs\Microsoft-Windows-Shell-Core%4ActionCenter.evtx
0xb68cb24765f0  \Windows\System32\winevt\Logs\Microsoft-Windows-SMBServer%4Operational.evtx
0xb68cb24c18e0  \Windows\System32\winevt\Logs\Microsoft-Windows-Application-Experience%4Steps-Recorder.evtx
0xb68cb2500ae0  \Windows\System32\winevt\Logs\Microsoft-Windows-DeviceSetupManager%4Operational.evtx
```

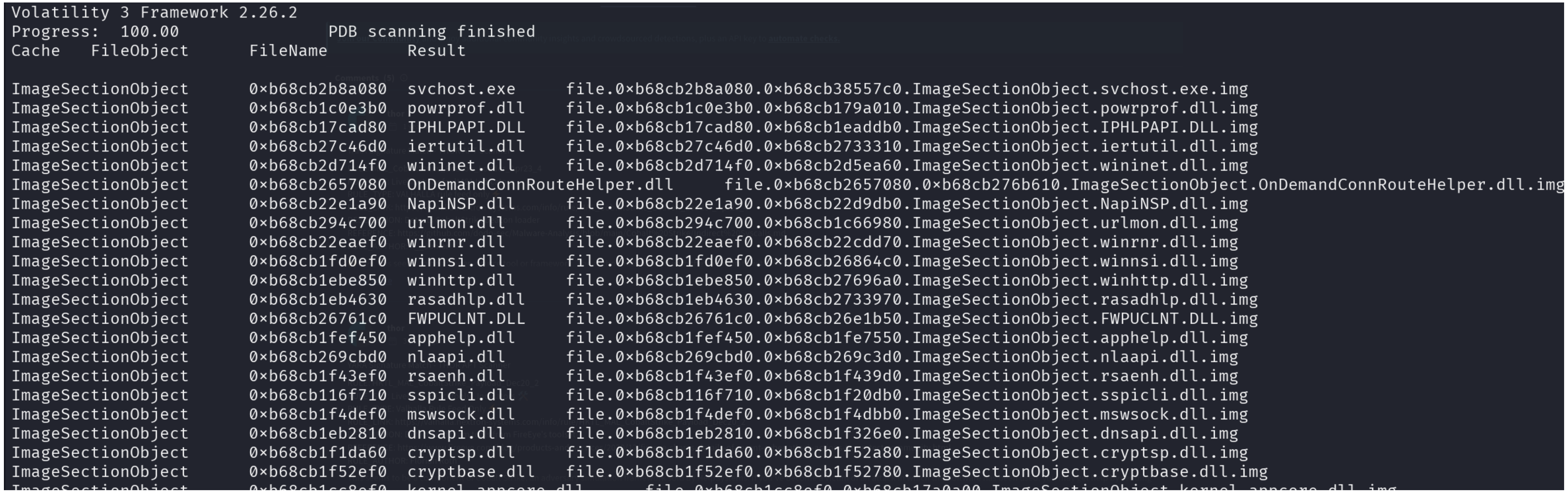
# Process File Dumping



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab1/memory.mem windows.dumpfile --pid 1488
```

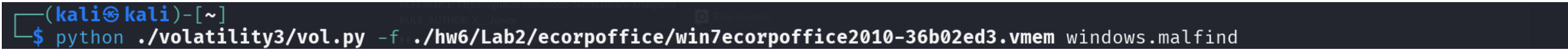
`windows.dumpfile` will scan and extract any files used *by a certain process*

- exe files
- dll files
- documents



```
Volatility 3 Framework 2.26.2
Progress:  100.00               PDB scanning finished
Cache   FileObject      FileName        Result

ImageSectionObject      0xb68cb2b8a080  svchost.exe     file.0xb68cb2b8a080.0xb68cb38557c0.ImageSectionObject.svchost.exe.img
ImageSectionObject      0xb68cb1c0e3b0  powrprof.dll    file.0xb68cb1c0e3b0.0xb68cb179a010.ImageSectionObject.powrprof.dll.img
ImageSectionObject      0xb68cb17cad80  IPHLPAPI.DLL    file.0xb68cb17cad80.0xb68cb1eaddb0.ImageSectionObject.IPHLPAPI.DLL.img
ImageSectionObject      0xb68cb27c46d0  iertutil.dll    file.0xb68cb27c46d0.0xb68cb2733310.ImageSectionObject.iertutil.dll.img
ImageSectionObject      0xb68cb2d714f0  wininet.dll     file.0xb68cb2d714f0.0xb68cb2d5ea60.ImageSectionObject.wininet.dll.img
ImageSectionObject      0xb68cb2657080  OnDemandConnRouteHelper.dll     file.0xb68cb2657080.0xb68cb276b610.ImageSectionObject.OnDemandConnRouteHelper.dll.img
ImageSectionObject      0xb68cb22e1a90  NapiNSP.dll     file.0xb68cb22e1a90.0xb68cb22d9db0.ImageSectionObject.NapiNSP.dll.img
ImageSectionObject      0xb68cb294c700  urlmon.dll      file.0xb68cb294c700.0xb68cb1c66980.ImageSectionObject.urlmon.dll.img
ImageSectionObject      0xb68cb22eaef0  winrnr.dll      file.0xb68cb22eaef0.0xb68cb22cdd70.ImageSectionObject.winrnr.dll.img
ImageSectionObject      0xb68cb1fd0ef0  winnsi.dll      file.0xb68cb1fd0ef0.0xb68cb26864c0.ImageSectionObject.winnsi.dll.img
ImageSectionObject      0xb68cb1ebe850  winhttp.dll     file.0xb68cb1ebe850.0xb68cb27696a0.ImageSectionObject.winhttp.dll.img
ImageSectionObject      0xb68cb1eb4630  rasadhlp.dll    file.0xb68cb1eb4630.0xb68cb2733970.ImageSectionObject.rasadhlp.dll.img
ImageSectionObject      0xb68cb26761c0  FWPUCLNT.DLL    file.0xb68cb26761c0.0xb68cb26e1b50.ImageSectionObject.FWPUCLNT.DLL.img
ImageSectionObject      0xb68cb1fef450  apphelp.dll     file.0xb68cb1fef450.0xb68cb1fe7550.ImageSectionObject.apphelp.dll.img
ImageSectionObject      0xb68cb269cbd0  nlaapi.dll      file.0xb68cb269cbd0.0xb68cb269c3d0.ImageSectionObject.nlaapi.dll.img
ImageSectionObject      0xb68cb1f43ef0  rsaenh.dll      file.0xb68cb1f43ef0.0xb68cb1f439d0.ImageSectionObject.rsaenh.dll.img
ImageSectionObject      0xb68cb116f710  sspicli.dll     file.0xb68cb116f710.0xb68cb1f20db0.ImageSectionObject.sspicli.dll.img
ImageSectionObject      0xb68cb1f4def0  mswsock.dll     file.0xb68cb1f4def0.0xb68cb1f4dbb0.ImageSectionObject.mswsock.dll.img
ImageSectionObject      0xb68cb1eb2810  dnsapi.dll      file.0xb68cb1eb2810.0xb68cb1f326e0.ImageSectionObject.dnsapi.dll.img
ImageSectionObject      0xb68cb1f1da60  cryptsp.dll     file.0xb68cb1f1da60.0xb68cb1f52a80.ImageSectionObject.cryptsp.dll.img
ImageSectionObject      0xb68cb1f52ef0  cryptbase.dll   file.0xb68cb1f52ef0.0xb68cb1f52780.ImageSectionObject.cryptbase.dll.img
ImageSectionObject      0xb68cb1cc8ef0  kernel.appcore.dll      file.0xb68cb1cc8ef0.0xb68cb17a0a00.ImageSectionObject.kernel.appcore.dll.img
```

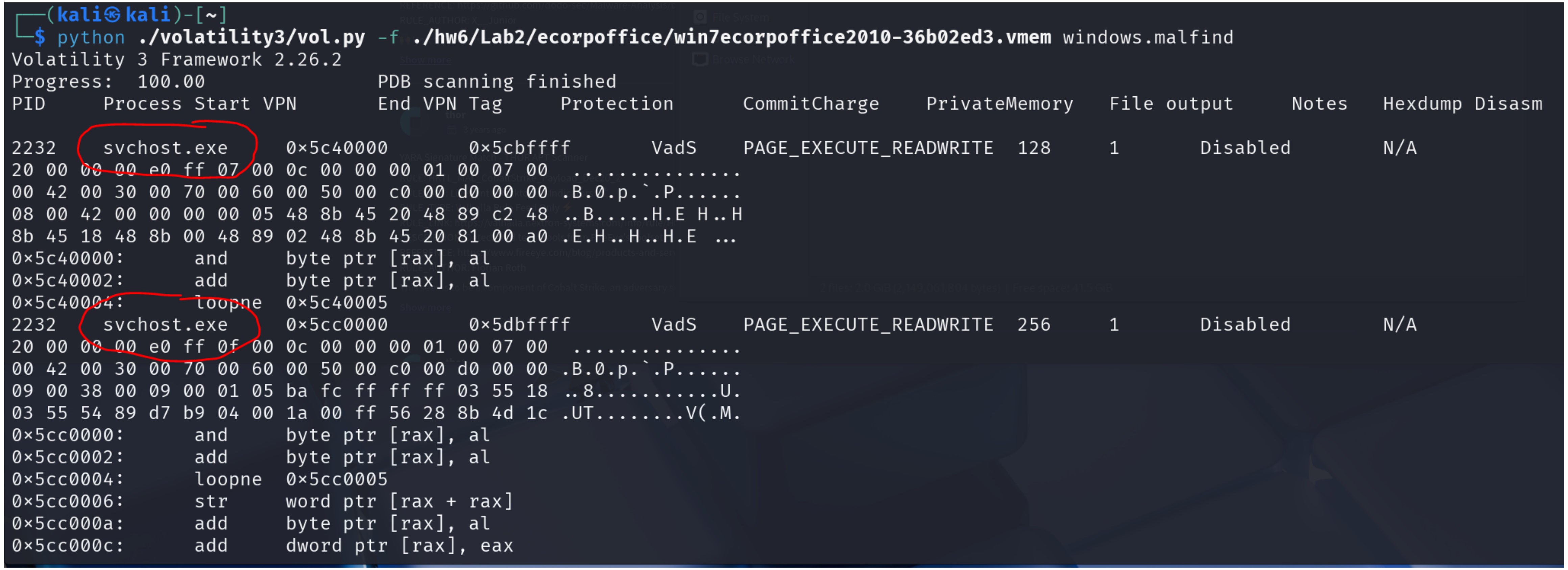
# Malfind



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab2/ecorpoffice/win7ecorpoffice2010-36b02ed3.vmem windows.malfind
```

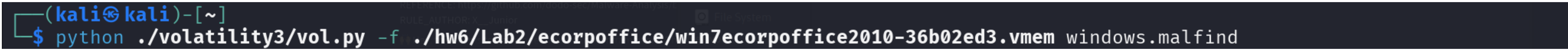
`windows.malfind` will identify malicious process information

False positives are possible, but malfind can be a great place to start searching



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab2/ecorpoffice/win7ecorpoffice2010-36b02ed3.vmem windows.malfind
Volatility 3 Framework 2.26.2
Progress:  100.00               PDB scanning finished
PID     Process Start VPN       End VPN Tag     Protection      CommitCharge    PrivateMemory   File output     Notes   Hexdump Disasm

2232    svchost.exe     0x5c40000       0x5cbffff       VadS    PAGE_EXECUTE_READWRITE  128     1       Disabled        N/A
20 00 00 00 e0 ff 07 00 0c 00 00 00 01 00 07 00  ................
00 42 00 30 00 70 00 60 00 50 00 c0 00 d0 00 00 .B.0.p.`.P......
08 00 42 00 00 00 00 05 48 8b 45 20 48 89 c2 48 ..B.....H.E H..H
8b 45 18 48 8b 00 48 89 02 48 8b 45 20 81 00 a0 .E.H..H..H.E ...
0x5c40000:      and     byte ptr [rax], al
0x5c40002:      add     byte ptr [rax], al
0x5c40004:      loopne  0x5c40005
2232    svchost.exe     0x5cc0000       0x5dbffff       VadS    PAGE_EXECUTE_READWRITE  256     1       Disabled        N/A
20 00 00 00 e0 ff 0f 00 0c 00 00 00 01 00 07 00  ................
00 42 00 30 00 70 00 60 00 50 00 c0 00 d0 00 00 .B.0.p.`.P......
09 00 38 00 09 00 01 05 ba fc ff ff ff 03 55 18 ..8...........U.
03 55 54 89 d7 b9 04 00 1a 00 ff 56 28 8b 4d 1c .UT........V(.M.
0x5cc0000:      and     byte ptr [rax], al
0x5cc0002:      add     byte ptr [rax], al
0x5cc0004:      loopne  0x5cc0005
0x5cc0006:      str     word ptr [rax + rax]
0x5cc000a:      add     byte ptr [rax], al
0x5cc000c:      add     dword ptr [rax], eax
```

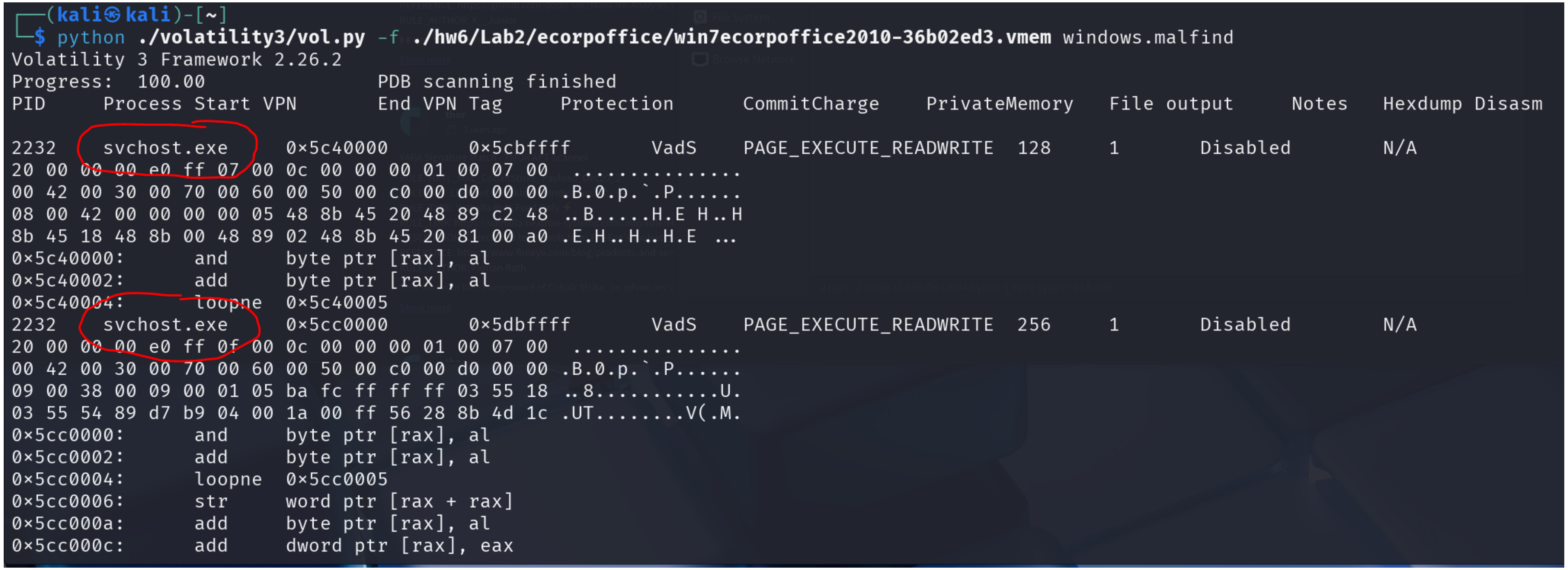
## Malfind



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab2/ecorpoffice/win7ecorpoffice2010-36b02ed3.vmem windows.malfind
```

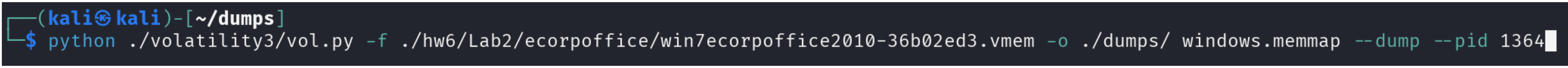
`windows.malfind` will identify malicious process information

False positives are possible, but malfind can be a great place to start searching



```
┌──(kali㉿kali)-[~]
└─$ python ./volatility3/vol.py -f ./hw6/Lab2/ecorpoffice/win7ecorpoffice2010-36b02ed3.vmem windows.malfind
Volatility 3 Framework 2.26.2
Progress:  100.00               PDB scanning finished
PID     Process Start VPN       End VPN Tag     Protection      CommitCharge    PrivateMemory   File output     Notes   Hexdump Disasm

2232    svchost.exe     0x5c40000       0x5cbffff       VadS    PAGE_EXECUTE_READWRITE  128     1       Disabled        N/A
20 00 00 00 e0 ff 07 00 0c 00 00 00 01 00 07 00  ................
00 42 00 30 00 70 00 60 00 50 00 c0 00 d0 00 00 .B.0.p.`.P......
08 00 42 00 00 00 00 05 48 8b 45 20 48 89 c2 48 ..B.....H.E H..H
8b 45 18 48 8b 00 48 89 02 48 8b 45 20 81 00 a0 .E.H..H..H.E ...
0x5c40000:      and     byte ptr [rax], al
0x5c40002:      add     byte ptr [rax], al
0x5c40004:      loopne  0x5c40005
2232    svchost.exe     0x5cc0000       0x5dbffff       VadS    PAGE_EXECUTE_READWRITE  256     1       Disabled        N/A
20 00 00 00 e0 ff 0f 00 0c 00 00 00 01 00 07 00  ................
00 42 00 30 00 70 00 60 00 50 00 c0 00 d0 00 00 .B.0.p.`.P......
09 00 38 00 09 00 01 05 ba fc ff ff ff 03 55 18 ..8...........U.
03 55 54 89 d7 b9 04 00 1a 00 ff 56 28 8b 4d 1c .UT........V(.M.
0x5cc0000:      and     byte ptr [rax], al
0x5cc0002:      add     byte ptr [rax], al
0x5cc0004:      loopne  0x5cc0005
0x5cc0006:      str     word ptr [rax + rax]
0x5cc000a:      add     byte ptr [rax], al
0x5cc000c:      add     dword ptr [rax], eax
```

# Identifying Suspicious Activity

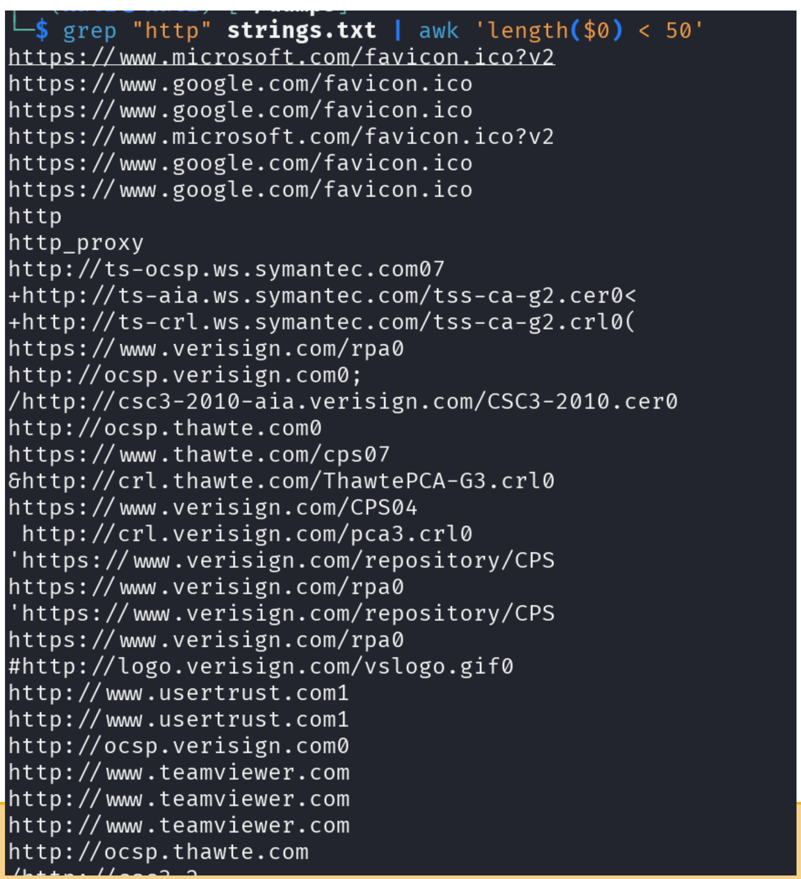


```
┌──(kali㉿kali)-[~/dumps]
└─$ python ./volatility3/vol.py -f ./hw6/Lab2/ecorpoffice/win7ecorpoffice2010-36b02ed3.vmem -o ./dumps/ windows.memmap --dump --pid 1364
```

`windows.memmap` to dump a process

VirusTotal?

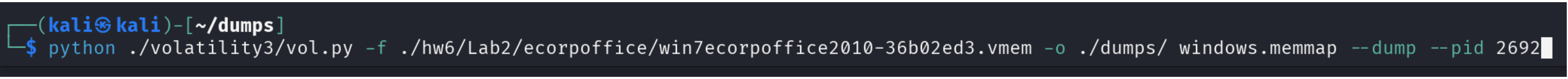
Let's check strings. `http://` is a code string to search for finding what websites the user may have visited



```
└─$ grep "http" strings.txt | awk 'length($0) < 50'
https://www.microsoft.com/favicon.ico?v2
https://www.google.com/favicon.ico
https://www.google.com/favicon.ico
https://www.microsoft.com/favicon.ico?v2
https://www.google.com/favicon.ico
https://www.google.com/favicon.ico
http
http_proxy
http://ts-ocsp.ws.symantec.com07
+http://ts-aia.ws.symantec.com/tss-ca-g2.cer0<
+http://ts-crl.ws.symantec.com/tss-ca-g2.crl0(
https://www.verisign.com/rpa0
http://ocsp.verisign.com0;
/http://csc3-2010-aia.verisign.com/CSC3-2010.cer0
http://ocsp.thawte.com0
https://www.thawte.com/cps07
&http://crl.thawte.com/ThawtePCA-G3.crl0
https://www.verisign.com/CPS04
 http://crl.verisign.com/pca3.crl0
'https://www.verisign.com/repository/CPS
https://www.verisign.com/rpa0
'https://www.verisign.com/repository/CPS
https://www.verisign.com/rpa0
#http://logo.verisign.com/vslogo.gif0
http://www.usertrust.com1
http://www.usertrust.com1
http://ocsp.verisign.com0
http://www.teamviewer.com
http://www.teamviewer.com
http://www.teamviewer.com
http://ocsp.thawte.com
```

**`awk`** is a command-line utility/scripting framework used to process and manipulate data

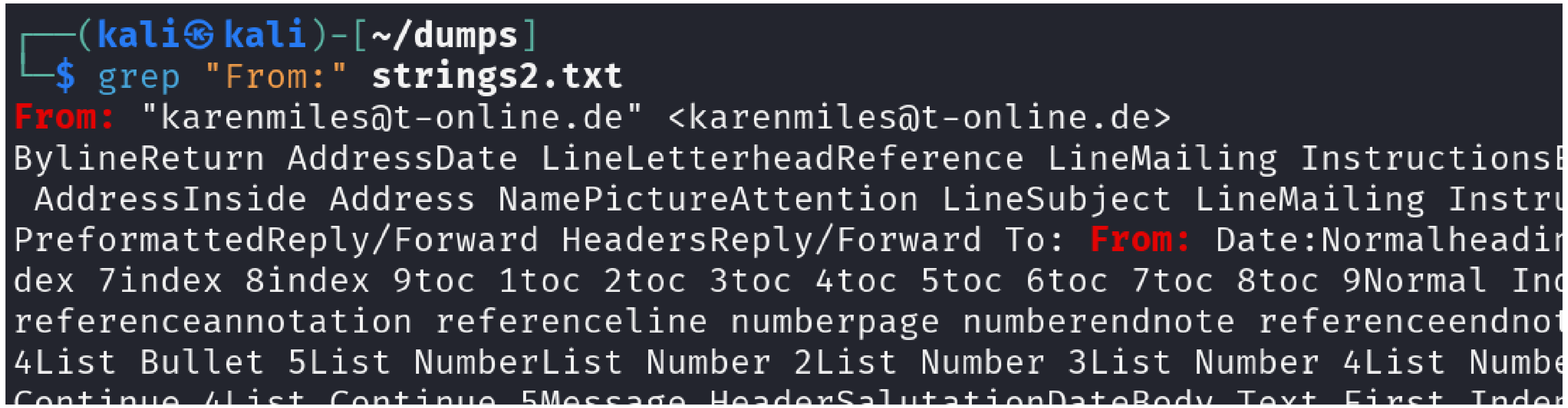
# Email Activity



```
┌──(kali㉿kali)-[~/dumps]
└─$ python ./volatility3/vol.py -f ./hw6/Lab2/ecorpoffice/win7ecorpoffice2010-36b02ed3.vmem -o ./dumps/ windows.memmap --dump --pid 2692
```

Outlook (email agent) is a process in memory, lets dump it with `windows.memmap`!

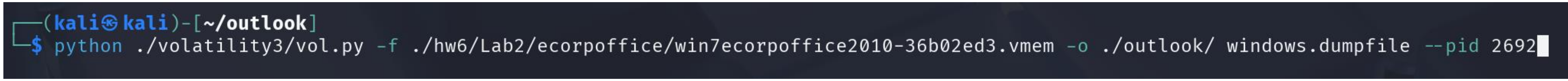
We can grep through it to see email messages that were in memory



```
┌──(kali㉿kali)-[~/dumps]
└─$ grep "From:" strings2.txt
From: "karenmiles@t-online.de" <karenmiles@t-online.de>
BylineReturn AddressDate LineLetterheadReference LineMailing InstructionsE
 AddressInside Address NamePictureAttention LineSubject LineMailing Instru
PreformattedReply/Forward HeadersReply/Forward To: From: Date:Normalheadir
dex 7index 8index 9toc 1toc 2toc 3toc 4toc 5toc 6toc 7toc 8toc 9Normal Inc
referenceannotation referenceline numberpage numberendnote referenceendnot
4List Bullet 5List NumberList Number 2List Number 3List Number 4List Numbe
Continue 4List Continue 5Message HeaderSalutationDateBody Text First Inder
```

If this was a phishing attack, knowing the email is came from is valuable information

Copies of emails are stored in a .PST file



```
┌──(kali㉿kali)-[~/outlook]
└─$ python ./volatility3/vol.py -f ./hw6/Lab2/ecorpoffice/win7ecorpoffice2010-36b02ed3.vmem -o ./outlook/ windows.dumpfile --pid 2692
```

PST Viewer

**GoldFynch has a great online tool for viewing PST files**

(And will let you download email attachments)