

Welcome to FOR532: Enterprise Memory Forensics In-Depth



- For Class Prep, you will need to find:
 - Course Media "A" (ISO file or USB)
 - Workbook
- Before class starts, please complete:
 - Lab 0: Install SIFT Workstation and 532 Windows VM
- Course Dropbox Link: <https://for532.com/dropbox>

Welcome to the SANS FOR532 Enterprise Memory Forensics In-Depth course. In this class, you'll learn how memory forensics can help your incident response investigations. The goal of this course is to enable you to add memory forensics to your Incident Response tool chest.

It will give you greater insights into attacker activities in your investigations and might make the difference between success and failure in your investigations.

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

SANSForensics

dfir.to/DFIRCast

@SANSForensics

dfir.to/LinkedIn

OPERATING SYSTEM & DEVICE IN-DEPTH

FOR308
Digital Forensics Essentials

FOR498
Battlefield Forensics & Data Acquisition
GBFA

FOR500
Windows Forensic Analysis
GCFA

FOR518
Mac and iOS Forensic Analysis & Incident Response
GIME

FOR585
Smartphone Forensic Analysis In-Depth
GASF

INCIDENT RESPONSE & THREAT HUNTING

FOR508
Advanced Incident Response, Threat Hunting & Digital Forensics
GCFA

FOR509
Enterprise Cloud Forensics & Incident Response
GCFR

FOR528
Ransomware for Incident Responders

FOR572
Advanced Network Forensics: Threat Hunting, Analysis & Incident Response
GNFA

FOR578
Cyber Threat Intelligence
GCTI

FOR608
Enterprise-Class Incident Response & Threat Hunting

FOR630
REM: Malware Analysis Tools & Techniques
GREM

FOR710
Reverse-Engineering Malware: Advanced Code Analysis

SEC504
Hacker Tools, Techniques & Incident Handling
GCIH

This page intentionally left blank.

Volatility 3 Content



As required by the volatility 3 license, all materials related to volatility 3 are made public.

Volatility

- Open-Source memory forensics tool
- Provides a platform for a number of built-in plugins as well as additional custom plugins
- Usually adopts fairly quickly to new OS versions
- Compatible to Windows, Linux and MacOS memory images
- Can work with hibernation files
- Used to be the tool of choice to conduct structured memory analysis
- Current version 3 took very long to be mostly finished; plugin support is still very limited.
- Volatility 3 comes with a new license that seems to be heavily debated in the OS community

Volatility is an open-source memory forensics framework that is used to analyze the contents of a computer's volatile memory (RAM). It can be used to extract artifacts from a memory dump, such as running processes, network connections, and open files, and can help investigators identify suspicious activity and potential security threats. Volatility is a powerful and flexible tool that is widely used by forensic investigators, security researchers, and others who need to analyze the contents of a computer's memory. It is available for a variety of platforms, including Windows, Linux, and Mac OS X, and can be used to analyze memory dumps from a wide range of systems.

The current version is 3, however, it doesn't support all the plugins that exist for volatility 2. The new, home-grown license that is unique to volatility 3 has been debated in the open-source community.

Further reading:

<https://twitter.com/cyb3rops/status/1356328893410635777?lang=de>

Volatility OS Support



- 32-bit Windows XP (Service Pack 2 and 3)
- 32-bit Windows 2003 Server (Service Pack 0, 1, 2)
- 32-bit Windows Vista (Service Pack 0, 1, 2)
- 32-bit Windows 2008 Server (Service Pack 1, 2)
- 32-bit Windows 7 (Service Pack 0, 1)
- 32-bit Windows 8, 8.1, and 8.1 Update 1
- 32-bit Windows 10 (initial support)
- 64-bit Windows XP (Service Pack 1 and 2)
- 64-bit Windows 2003 Server (Service Pack 1 and 2)
- 64-bit Windows Vista (Service Pack 0, 1, 2)
- 64-bit Windows 2008 Server (Service Pack 1 and 2)
- 64-bit Windows 2008 R2 Server (Service Pack 0 and 1)
- 64-bit Windows 7 (Service Pack 0 and 1)
- 64-bit Windows 8, 8.1, and 8.1 Update 1
- 64-bit Windows Server 2012 and 2012 R2
- 64-bit Windows 10 (including at least 10.0.14393)
- 64-bit Windows Server 2016 (including at least 10.0.14393.0)



- 32-bit 10.5.x Leopard (the only 64-bit 10.5 is Server, which isn't supported)
- 32-bit 10.6.x Snow Leopard
- 32-bit 10.7.x Lion
- 64-bit 10.6.x Snow Leopard
- 64-bit 10.7.x Lion
- 64-bit 10.8.x Mountain Lion
- 64-bit 10.9.x Mavericks
- 64-bit 10.10.x Yosemite
- 64-bit 10.11.x El Capitan
- 64-bit 10.12.x Sierra
- 64-bit 10.13.x High Sierra
- 64-bit 10.14.x Mojave
- 64-bit 10.15.x Catalina



- 32-bit Linux kernels 2.6.11 to 5.5
- 64-bit Linux kernels 2.6.11 to 5.5
- OpenSuSE, Ubuntu, Debian, CentOS, Fedora, Mandriva, etc.

Volatility supports a number of operating systems and versions. The available plugins per platform differ though. You can find a detailed list at the Volatility Foundation's website, which is <https://www.volatilityfoundation.org/26>.

Support of operating systems is subject to change for other volatility versions.

Volatility 3 Under the Hood

This page intentionally left blank.

From Volatility 2 to 3

- Current version = 3
- Moved from Python 2 to Python 3
- Complete rewrite – from monolith to library
- Introduces independent components that every plugin can combine to fulfill its task (Layers, Symbols, ...)
- No profiles anymore
- Automagic improved
- Uses real python objects instead of proxy objects
- Volatility 3 is way faster but loses live memory forensic capabilities
- That reduces its ability to scale significantly and reduces its use to pinpoint forensics

Volatility 2 and Volatility 3 are two different versions of the Volatility memory forensics framework.

Volatility 2 is the older version of the software and is no longer actively developed or maintained. Volatility 3 is the current version of the software and is under active development. There are several key differences between the two versions, including:

- Compatibility: Volatility 3 is compatible with a wider range of operating systems and memory dump formats than Volatility 2.
- Features: Volatility 3 includes a number of new features and improvements that are not available in Volatility 2, such as support for analyzing memory dumps from the Microsoft Hyper-V virtualization platform.
- Performance: Volatility 3 is generally faster and more efficient than Volatility 2 and can handle larger memory dumps more effectively. That comes at the cost of no longer being able to handle live memory.
- Documentation: The documentation for Volatility 3 is more extensive and up-to-date than the documentation for Volatility 2.

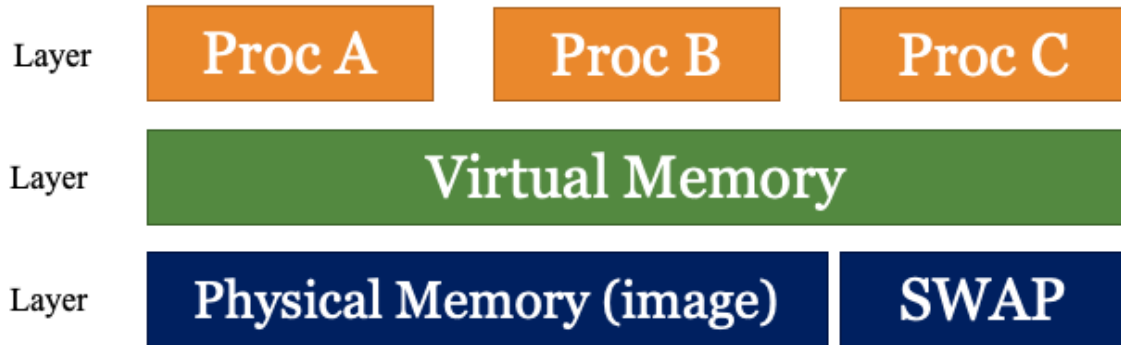
Overall, Volatility 3 is a more powerful and up to date tool than Volatility 2. However, if you need to analyze an older memory dump that is not supported by Volatility 3, or if you are using a platform that is not supported by Volatility 3, you may need to use Volatility 2 instead or switch to MemProcFS (only for Windows memory). Their live analysis also works.

Layers

- Layers replace address spaces in profiles
- Vol2 defined one fixed stack of address spaces
- Vol3 is more flexible which e.g., allows inclusion of swap space as one layer can have multiple dependencies below it (e.g., Virtual Memory consists of physical ram AND swap space)
- We'll later look into more details when you'll write your first own plugin

In Volatility 3, a layer is a modular component that is used to provide support for a specific operating system, memory dump format, or other type of data. Layers are used to abstract the underlying details of a memory dump, allowing the rest of the Volatility framework to access and analyze the data in a consistent and platform-agnostic way. Each layer in Volatility 3 is responsible for providing a specific set of functionality, such as parsing a particular memory dump format, or providing access to data structures and other artifacts from a specific operating system. Layers are used throughout the Volatility framework and are an important part of its modular and extensible architecture.

Layers (VOL3)



Like indicated, layers operate on different levels, closer to the memory image and further on top. Physical memory always refers to images, as volatility 3 does not support live memory analysis.

Symbols and Types

- Volatility 3 can build objects based on symbol tables
- Automatically fetches symbol tables for Windows from the official symbol server
- Authors can more easily adopt types quickly

In Volatility 3, symbols and types are two related concepts that are used to describe the structure and layout of data in a computer's memory. Symbols are used to represent named values or entities in memory, such as variables, functions, or data structures. Types are used to describe the data type of a symbol, such as an integer, floating-point number, or string. Together, symbols and types are used to provide a high-level view of the data in memory, allowing the Volatility framework to understand and analyze the data in a more meaningful way. Symbols and types are an important part of the Volatility framework, as they help to provide a clear and consistent way to access and manipulate data in memory.

Plugins

- Plugins leverage different layers and instantiated objects
- They query data from the memory representation to present it to users in a specific way
- Plugins have certain requirements to run (specified by the author)



In Volatility 3, a plugin is a modular component that adds a specific feature or capability to the Volatility framework. Plugins are used to extend the functionality of Volatility, allowing it to perform new types of analysis or support additional data sources. Each plugin in Volatility 3 is responsible for providing a specific set of functionality, such as extracting a particular type of artifact from a memory dump or analyzing a specific aspect of the data in memory. Plugins are a key part of the Volatility framework's extensible architecture and can be used to add new features and capabilities to the tool without modifying its core code.

General Command Structure: Volatility 2

```
vol.py -f image.mem --profile some_profile pslist
```



Usefull command	Description
<code>vol.py -h</code>	Shows all plugins and profiles
<code>vol.py <plugin> -h</code>	Shows helpfile specific to <plugin>

Volatility uses a set of plugins. In Volatility 2 we need to specify a profile that matches the image. In a nutshell, that tells volatility where to look for certain kernel structures and what exactly they look like. This might vary significantly between different versions of the same operating system family.

The general structure is:

```
vol.py -f <image> --profile <profile> <plugin>
```

Some plugins allow or even require further parameters. Certain extraction plugins for example require further input that specifies what they should extract. Every plugin is required to have a help page. The plugin specific help page can be invoked by using the plugin name and the switch **-h**.

One plugin that doesn't require a profile is **imageinfo**. This plugin is used to identify the required profile for an image.

General Command Structure: Volatility 3

```
vol.py -f image.mem windows.pslist
```

Memory image OS Plugin

Useful command	Description
<code>vol.py -h</code>	Shows all plugins and profiles
<code>vol.py <OS>.<plugin> -h</code>	Shows helpfile specific to <plugin>

Volatility 3 works differently. Profiles are gone and for Windows it autodetects the OS version of the image and downloads the symbols from the Microsoft symbol server accordingly.

Also, the naming schema for the plugins has changed. Now all Windows plugins are prefixed with windows.

The general command line schema is:

vol.py <OS>.<plugin>

Again the **-h** switch shows helpful information.

Linux plugins are prefixed with **Linux** and MacOS plugins with **mac**.

Tools

- MemProcFS gives access to VAD information via `./<pid>/memmap/vad.txt`
- There are a number of volatility plugins to inspect the VAD tree
 - **Volatility 3**
 - `windows.vadinfo.VadInfo`
 - `windows.vadyarascan.VadYaraScan`
 - **Volatility 2**
 - `vadwalk`
 - `vadtree`
 - `vadinfo`
 - `vaddump`
- VAD trees can be manually walked with volshell or windbg.

MemProcFS walks the tree for you and gives you a per-process view on the tree at “./<pid>/memmap/vad.txt”.

Volatility also offers a few plugins to walk the VAD tree. You can also manually do so by using volshell or windbg.

Volatility 3 Plugin Map (I)

Volatility 2	Volatility 3
<code>vol.py -f image.mem --profile some_profile pslist</code>	<code>vol3 -f image.mem windows.pslist</code>
<code>vol.py -f image.mem --profile some_profile pstree</code>	<code>vol3 -f image.mem windows.pstree</code>
<code>vol.py -f image.mem --profile some_profile psscan</code>	<code>vol3 -f image.mem windows.psscan</code>
<code>vol.py -f image.mem --profile some_profile dlllist -p <pid></code>	<code>vol3 -f image.mem windows.dlllist --pid <pid></code>
<code>vol.py -f image.mem --profile some_profile getsids</code>	<code>vol3 -f image.mem windows.getsids</code>
<code>vol.py -f image.mem --profile some_profile psxview</code>	Does not exist.

This page intentionally left blank.

Volatility 3 Plugin Map (2)

Volatility 2	Volatility 3
<code>vol.py -f image.mem --profile some_profile netscan</code>	<code>vol3 -f image.mem windows.netscan</code>

This page intentionally left blank.

Volatility 3 Plugin Map I

Volatility 2	Volatility 3
<code>vol.py -f <image> --profile <profile> malfind -d <dump directory></code>	<code>vol3 -f image.mem windows.pslist</code>
<code>vol3 -f <image> windows.malfind --dump</code>	<code>vol3 -f image.mem windows.pstree</code>
<code>vol.py -f <image> --profile <profile> ldrmodules -p <pid></code>	<code>vol3 -f image.mem windows.psscan</code>
<code>vol.py -f <image> --profile <profile> hollowfind</code>	Does not exist.
<code>vol.py -f <image> --profile <profile> atoms</code>	Does not exist.

This page intentionally left blank.

Volatility 3 Plugin Map II

Volatility 2	Volatility 3
<code>vol.py -f <image> --profile <profile> ssdt egrep -iv '(ntoskrnl win32k)'</code>	<code>Vol3 -f <image> windows.ssdt egrep -iv '(ntoskrnl win32k)'</code>
<code>vol.py -f <image> --profile <profile> apihooks</code>	Does not exist.

This page intentionally left blank.

Volatility 3 Plugin Map III

Volatility 2	Volatility 3
<code>vol.py -f <image> --profile <profile> -p <pid> --dump-dir . procdump</code>	<code>vol.py -f mydump.vmem windows.pslist.PsList --pid <PID> --dump</code>
<code>vol.py -f <image> --profile <profile> -p <pid> --dump-dir . procdump</code>	<code>vol.py -f mydump.vmem -o <output directory> windows.memmap.Memmap --pid <PID> --dump</code>
<code>vol.py -f <image> --profile <profile> -o <offset> --dump-dir . dlldump</code>	<code>vol3 -f c2.vmem windows.dlllist --pid <PID> --dump</code>
<code>vol.py -f <image> --profile <profile> -o <offset> --dump-dir . moddump</code>	<code>vol3 -f c2.vmem windows.modules --pid <PID> --dump</code>
<code>vol.py -f <image> --profile <profile> filescan > filelist</code>	<code>vol3 -f <image> filescan > filelist</code>

This page intentionally left blank.

Volatility 3 Plugin Map IV

Volatility 2	Volatility 3
<code>vol.py -f <image> --profile <profile> -o <offset> --dump-dir . dumpfiles</code>	<code>vol3 -f <image> windows.dumpfiles --virtaddr <virtual address></code>
<code>vol.py -f <image> --profile <profile> hivescan</code>	<code>vol3 -f <image> windows.registry.hivescan</code>
<code>vol.py -f <image> --profile <profile> hivescan</code>	<code>vol3 -f <image> windows.registry.hivescan</code>
<code>vol.py -f <image> --profile <profile> printkey</code>	<code>vol3 -f <image> windows.registry.printkey</code>
<code>vol.py -f <image> --profile <profile> userassist</code>	<code>vol3 -f <image> windows.registry.userassist</code>

This page intentionally left blank.

Volatility 3 Plugin Map V

Volatility 2	Volatility 3
<code>vol.py -f <image> --profile <profile> autoruns</code>	n/a
<code>vol.py -f <image> --profile <profile> getservicesids</code>	n/a
<code>vol.py -f <image> --profile <profile> shimcachemem</code>	n/a
<code>vol.py -f <image> --profile <profile> amcache</code>	n/a
<code>vol.py -f <image> --profile <profile> usbstor</code>	n/a
<code>vol.py -f <image> --profile <profile> prefetchparser</code>	n/a

This page intentionally left blank.

Volatility 3 plugin Map VI

Volatility 2	Volatility 3
<code>vol.py -f <image> --profile <profile> linux_bash</code>	<code>vol3 -f <image> linux.bash</code>
<code>vol.py -f <image> --profile <profile> linux_check_afinfo</code>	<code>vol3 -f <image> linux.check_afinfo</code>
<code>vol.py -f <image> --profile <profile> linux_check_idt</code>	<code>vol3 -f <image> linux.check_idt</code>
<code>vol.py -f <image> --profile <profile> linux_check_modules</code>	<code>vol3 -f <image> linux.check_modules</code>
<code>vol.py -f <image> --profile <profile> linux_lsmod</code>	<code>vol3 -f <image> linux.lsmod</code>

This page intentionally left blank.

Volatility 3 plugin Map VII

Volatility 2	Volatility 3
<code>vol.py -f <image> --profile <profile> linux_lsof</code>	<code>vol3 -f <image> linux.lsof</code>
<code>vol.py -f <image> --profile <profile> linux_malfind</code>	<code>vol3 -f <image> linux.malfind</code>
<code>vol.py -f <image> --profile <profile> linux_proc_maps</code>	<code>vol3 -f <image> linux.proc_maps</code>
<code>vol.py -f <image> --profile <profile> linux_pslist</code>	<code>vol3 -f <image> linux.pslist</code>
<code>vol.py -f <image> --profile <profile> linux_pstree</code>	<code>vol3 -f <image> linux.pstree</code>
<code>vol.py -f <image> --profile <profile> linux_psxview</code>	Does not exist.

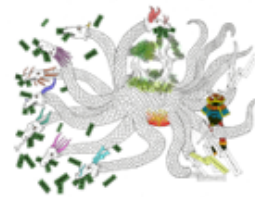
This page intentionally left blank.

Multiple Analysts on the Same Dataset

- Even with automation place, no analyst will know if someone else already analyzed a certain data set
- Collaboration saves resources



VolUtility



Orochi

For a long time, the standard for volatility collaboration was volUtility. It is a web-based platform that supports collaborative analysis of memory images. It stores already processed results. It only works with volatility 2 as of now which limits its use more and more.

For volatility 3 Orochi can be seen as an alternative to volUtility when using volatility 3. It's also quite more powerful and uses Elasticsearch as data backend.

Further Reading:

<https://github.com/kevthehermit/VolUtility>

<https://github.com/LDO-CERT/orochi>

Custom Volatility Content



This page intentionally left blank.

Volatility 3

- Volatility 2 will be discontinued soon
- Yet more than half of the plugins have not been ported to volatility 3 so far
- The volatility 3 developers offer all the bits and pieces needed to write new plugins; however, documentation is a bit limited.
- By the end of that section, you'll understand how to write volatility 3 plugins, so the limited number of plugins baked into vol3 will not be a problem any longer

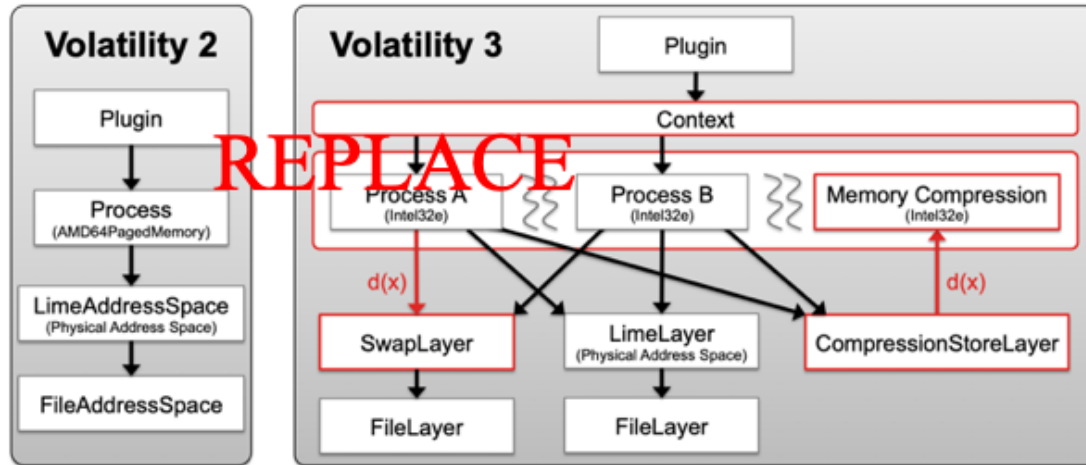
Volatility 2 will most likely be discontinued soon. At the same time the majority of plugins available in Volatility 2 have not yet been ported to Volatility 3. The issue tracker on the Volatility 3 GitHub repo does have quite some feature requests asking for plugins that have not been implemented yet. In general, it seems to be rare, that the requests get fulfilled. Mostly the developers refer to volshell and other capabilities that can be used to get the data the plugins would produce out manually.

For that reason, it might be necessary that you write your own plugin if you want to stick with volatility.

There is little information on how to write plugins, but you can always look at preexisting plugins to understand the workflow.

Please be aware, that the volatility 3 license is new and one of a kind. It has been pretty much debated when it first came out and might impact your decision to write content for Volatility 3.

Layers



Source: Volatility Foundation

While volatility 2 required plugins to depend on a very hierarchical structure, volatility is more flexible. Plugins can combine different layers in a context as needed and are not restricted to operate on one layer only. As volatility supports loading swap files (and, also, compression), the accessibility of data, especially in newer Windows versions, is way better than in volatility 2.

Templates & Objects

```
56 65 72 79 20 6e 69 63 65 20 66 69 6e 64 2e 20 59 6f 75
20 72 65 61 6c 6c 79 20 77 65 6e 74 20 74 68 65 20 65 78
74 72 61 20 6d 69 6c 65 2e 20 44 69 64 20 79 6f 75 20 72
65 61 6c 6c 79 20 74 79 70 65 20 61 6c 6c 20 74 68 6f 73
65 20 68 65 78 20 63 68 61 72 61 63 74 65 72 73 3f 20 41
77 65 73 6f 6d 65 2e 20 0a 0a 59 6f 75 20 74 61 6b 65 20
74 68 65 20 62 6c 75 65 20 70 69 6c 6c 2c 20 74 68 65 20
73 74 6f 72 79 20 65 6e 64 73 2c 20 79 6f 75 20 77 61 6b
65 20 75 70 20 69 6e 20 79 6f 75 72 20 62 65 64 20 61 6e
64 20 62 65 6c 69 65 76 65 20 77 68 61 74 65 76 65 72 20
79 6f 75 20 77 6c 69 65 76 65
2e 20 59 6f 75 20 72 65 64 20
70 69 6c 6c 2c 20 69 6e 20 77
6f 6e 64 65 72 6c 61 6e 64 2c 20 61 6e 64 20 49 20 73 68
6f 77 20 79 6f 75 20 68 6f 77 20 64 65 65 70 20 74 68 65
20 72 61 62 62 69 74 20 68 6f 6c 65 20 67 6f 65 73 0a 0a
52 65 64 20 50 69 6c 6c 20 2d 3e 20 68 74 74 70 73 3a 2f
2f 66 6f 72 35 33 32 2e 63 6f 6d 2f 63 75 72 69 6f 75 73
69 6e 76 65 74 69 67 61 74 6f 72 0a 42 6c 75 65 20 50 69
6c 6c 20 2d 3e 20 68 74 74 70 73 3a 2f 2f 66 6f 72 35 33
32 2e 63 6f 6d 2f 68 61 70 70 79 77 68 65 72 65 69 61 6d
0a 0a 0a 0a 49 20 73 74 69 6c 6c 20 6e 65 65 64 20 73 6f
6d 65 20 6d 6f 72 65 20 63 68 61 72 61 63 74 65 72 73 20
74 6f 20 6d 61 6b 65 20 74 68 65 20 73 6c 69 64 65 20 77
6f 72 6b 20 2e 20 c2 af 5c 5f 28 e3 83 84 29 5f 2f c2 af
```

Template

Memory dump

Object

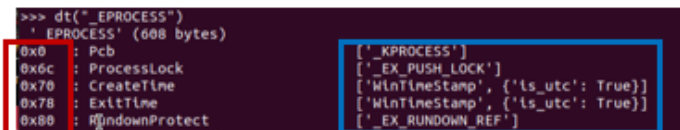
Attribute 1: 55 65 72 79
Attribute 2: 20 6e 69 63 65 20 66
...

There are quite a few changes to the object model between volatility 2 and volatility 3. While volatility 2 kind of created its own presentation even of simple data types like integer, volatility 3 inherits these types from the original python types. The advantage of this is, that these objects now have all the functions that the native python representation would have.

Furthermore, volatility differentiates between templates and objects. A template is like a construction worker and a building plan that is able to build objects. An object is the actual representation of an entity in memory. One of the important differences between volatility 2 and volatility 3 is that in volatility 2 templates were missing certain meta data points like for example size information. This meant that to get the size of a certain data point, one would need to initiate an empty object first and then get the size out of the object. Now that data is rightly available in the template.

Symbols

- Symbols tell how templates for certain objects need to look like plus where to find it (offset)
- They are different for every operating system
- Volatility managed symbols in a structure called **SymbolSpace**
- **SymbolSpaces** are the successor of volatility 2's profiles
- Profiles couldn't handle wow64 processes very well as all symbols lived in the same namespace (name collision)



The screenshot shows a command prompt with the following output:

```
>>> dt("_EPROCESS")
'_EPROCESS' (608 bytes)
0x0 : Pcb
0x6c : ProcessLock
0x70 : CreateTime
0x78 : ExitTime
0x80 : RundownProtect
```

On the right side of the screenshot, there is a list of symbols for the `_KPROCESS` structure:

```
['_KPROCESS']
['_EX_PUSH_LOCK']
['WlnTimeStamp', {'ls_utc': True}]
['WlnTimeStamp', {'ls_utc': True}]
['_EX_RUNDOWN_REF']
```

The word "offset" is written in red text to the left of the first column of the first table, and the word "symbol" is written in blue text to the right of the second column of the second table.

One of the major changes in volatility 3 is that you no longer need to specify a profile to run plugins. For Windows images, volatility 3 downloads the official Microsoft Symbol Tables from Microsoft's symbol server. Symbols tell volatility how certain objects look like and sometimes where to find them for a very particular operating system build.

Volatility 3 maintains these symbols in a so-called SymbolSpace. In that regard, SymbolSpaces could be seen as the successor to profiles.

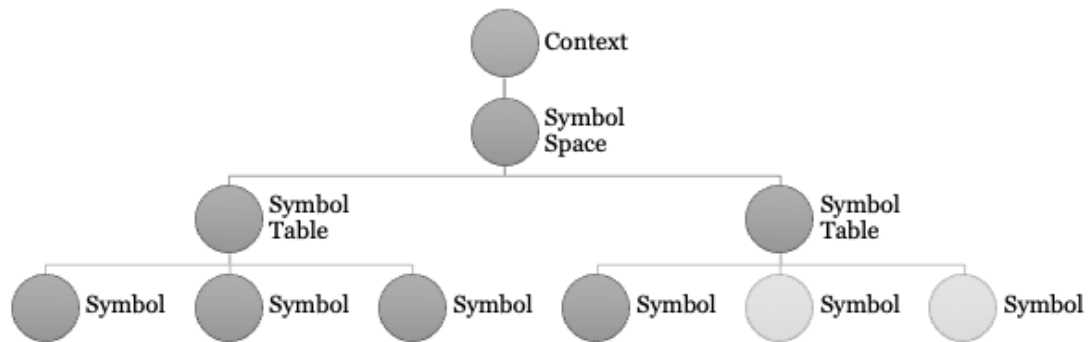
One of the issues you could come across when analyzing 64-bit Windows memory images was a name collision. For instance, structures with the exact same name existed for the 64-bit space but also for the 32-bit (wow64) space. That is one of the reasons why in volatility 3, analyzing wow64 processes does not always work as expected.

In volatility 3 64, the naming convention for symbols changed. They now follow the following convention to prevent name collisions.

module!symbol

That syntax might look very familiar to you if you happen to be a WinDBG user.

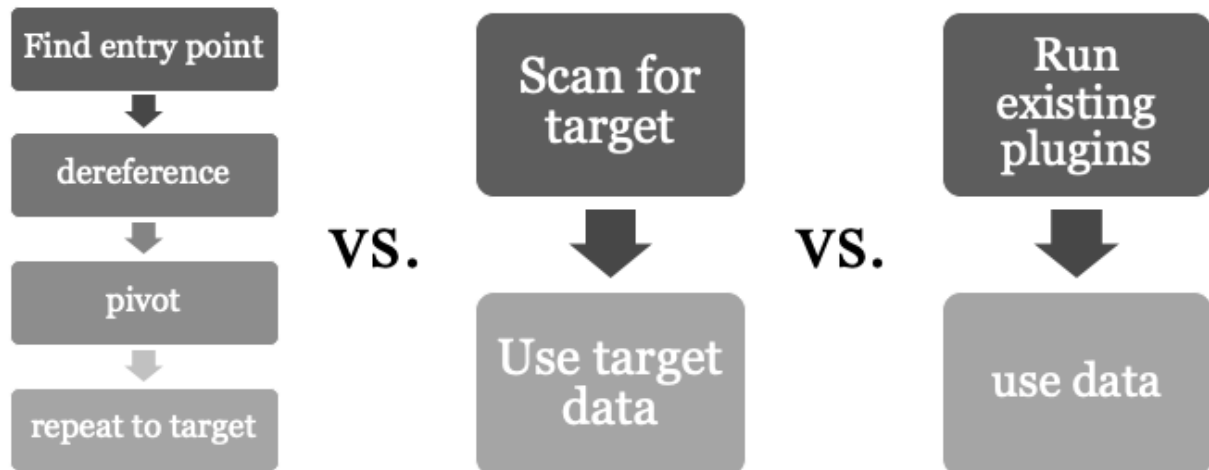
Object Hierarchy



Volatility 3 comes with a certain hierarchy of objects. Right on top is the Context. A context can only have one SymbolSpace at a time, but the SymbolSpace can have as many SymbolTables as needed. Every SymbolTable can store a number of symbols.

Plugin Iterative vs. Scanners vs. Nested

* Multiple approaches may be combined

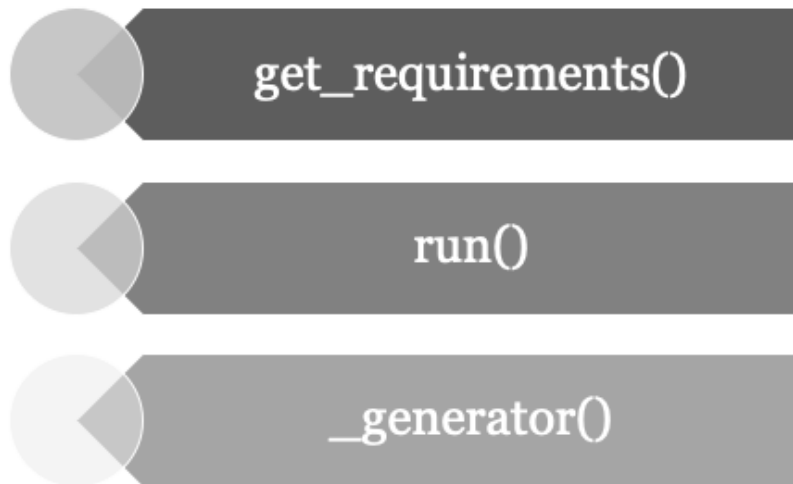


There are different approaches when writing a plugin. For instance, a pslist plugin works way different than a facebook_profile plugin.

While the first one follows the structure of memory to dereference objects (in that case _EPROCESS objects), the other one looks for patterns in memory. This could be the first part or the structure of the json object a Facebook webserver transfers when you open a profile page. You could also combine these two processes and only look for Facebook data in the process spaces of typical browser processes.

The third approach is running already existing plugins and combining their output in a useful way. This is what we will do in the lab.

Plugin Structure



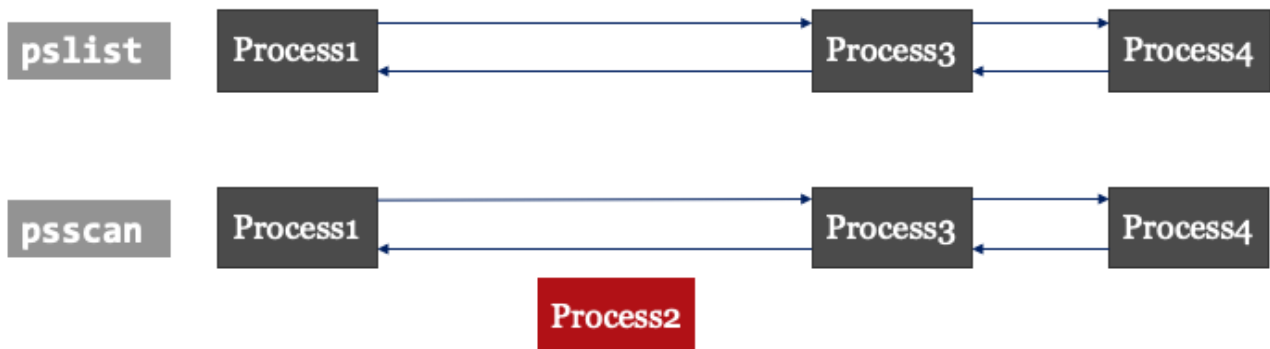
Volatility plugins follow a certain train of actions. First, the plugin needs to ascertain that its requirements are met. Plugin authors do that by implementing the `get_requirements()` function.

The actual payload executes in the `run()` function. This is where most of the magic happens.

Finally, in the `_generator()` function, the result data is prepared to be displayed.

Our psxview

- For starters we will write a minimalistic version of **psxview**
- **psxview** is one of the plugins that doesn't exist in volatility 3 (yet)



As there is no psxview plugin for volatility 3 yet, we will implement a very simple plugin that at least does some of the thing the volatility 2 psxview plugin would do. You can always extend the plugin later if you like.

Our psxview: Limitations

- Our psxview will only compare **pslist** to **psscan** output
- **psscan** will also find legitimately exited processes which will not show up in **pslist**
- The vol 2 plugin has a few more features to check if a process is running or not

```
root@siftworkstation:/memory# vol.py -f rootkit.ing psxview -R
```

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcid	csrss
0x01ab1a20	services.exe	940	True	True	True	True	True
0x01a8db68	svchost.exe	1120	True	True	True	True	True
0x019bc590	alg.exe	1924	True	True	True	True	True
0x01666a70	winlogon.exe	896	True	True	True	True	True
0x0169bda0	svchost.exe	1188	True	True	True	True	True
0x01617600	explorer.exe	1288	True	True	True	True	True
0x015eb270	svchost.exe	1320	True	True	True	True	True
0x019887f0	spoolsv.exe	1824	True	True	True	True	True
0x019922c0	svchost.exe	1608	False	True	True	True	True
0x01838c88	csrss.exe	868	False	True	False	False	Okay
0x01a69a40	lsass.exe	952	True	True	True	True	True
0x01bcc830	System	4	True	True	True	True	Okay
0x01750020	csrss.exe	872	True	True	True	True	Okay
0x01a385a0	smss.exe	824	True	True	True	True	Okay

We will compare the output of pslist and psscan to find processes that are in memory but missing in the pslist output that could be existing processes but, also, unlinked processes. To make a clear determination, psxview in volatility 2 also looks at some other data points like for instance running threads. We won't implement that part today.

Our Workflow



So, here's the high-level workflow of what we will do. Please note, that the actual implementation you'll soon start writing could be optimized but has been written in a very sequential way to make it easier for non-programmers to understand.

1. First, we run pslist and store the results into an array.
2. Secondly, we run psscan on the same dataset.
3. Finally, we match the processes based on their pids and render a list of processes and their presence in the respective lists.

Lab 4.2

Write Volatility Plugin

This page intentionally left blank.