Malware Detection and Triage with Volatility 3

Andrew Case, Daniel Donze, Lauren Pace

© 2025 Volexity Inc.

Course materials

- Slides (this presentation)
- Lab Guide
- Lab memory samples
- Command Comparison

- Do not distribute these materials outside of workshop attendees

© 2025 Volexity Inc.

Memory Forensics Introduction

- ☑ Definition and benefits of memory analysis
- ☑ History of memory analysis and pioneering research
- ☑ Different memory analysis techniques and approaches



© 2025 Volexity Inc.

What is memory forensics?

- Memory forensics is the process of acquiring and analyzing physical memory (RAM) in order to find artifacts and evidence
- Can be performed in conjunction with disk and network forensics, but many cases solvable with just memory analysis
- **Fileless malware != memory-only malware**
  - Memory-only doesn't survive reboots (but may be re-infected over the network, etc.)
  - Fileless can persist in the registry or elsewhere on the system

© 2025 Volexity Inc.

What can be recovered?

- Running processes
- Active network connections
- Loaded kernel drivers
- Console input and output
- Malware-created artifacts
- Application information (URL history, chat logs, emails)
- Disk encryption keys
- A whole lot more…

© 2025 Volexity Inc.

Why does it matter?

- Can recover the entire state of the operating system and running applications at the time of the capture
- Can also uncover historical information
- Much of the information recovered from memory is never written to disk or the network (i.e. software encryption keys)
  - IM clients that don't log
  - Browsers in private (incognito) mode
- Advanced malware operates only in memory

© 2025 Volexity Inc.

The Prevalence of Memory Only Malware

- Across platforms, memory-only payloads are often used by malware to avoid detection and hinder analysis
- Disk and live forensics generally can find no traces of this malware
- Volatile memory is the ***only*** place to determine that such malware is present and to fully investigate it

Corelump is the main payload and resides exclusively in memory to evade detection. It contains a variety of capabilities including keylogging, capturing screenshots, exfiltrating files, running a remote shell, and running arbitrary plugins downloaded from KNOTWEED's C2 server.

Microsoft Report: https://www.microsoft.com/en-us/security/blog/2022/07/27/untangling-knotweed-european-private-sector-offensive-actor-using-0-day-exploits/

Mark Your Calendar: APT41 Innovative Tactics

May 28, 2025

Malware Infection Chain

This malware has three distinct modules, deployed in series, each with a distinct function. Each module also implements stealth and evasion techniques, including memory-only payloads, encryption, compression, process hollowing, control flow obfuscation, and leveraging Google Calendar for C2.

1. **PLUSDROP** - DLL to decrypt and execute the next stage in memory.
2. **PLUSINJECT** - Launches and performs process hollowing on a legitimate "svchost.exe" process, injecting the final payload.

- https://cloud.google.com/blog/topics/threat-intelligence/apt41-innovative-tactics

CISA Emergency Directives

This emergency directive requires the following actions:

1. *Agencies that have the expertise to take the following actions immediately must do so before proceeding to Action 2.* ***Agencies without this capability shall proceed to Action 2.***

   - Forensically image system memory and/or host operating systems hosting all instances of SolarWinds Orion versions 2019.4 through 2020.2.1 HF1. Analyze for new user or service accounts, privileged or otherwise.

   - Analyze stored network traffic for indications of compromise, including new external DNS domains to which a small number of agency hosts (e.g., SolarWinds systems) have had connections.

2. Affected agencies shall immediately **disconnect or power down SolarWinds Orion products, versions 2019.4 through 2020.2.1 HF1, from their network**.

© 2025 Volexity Inc.

NCCIC – You've Been Compromised?

- Don'ts of Incident Response
  - Do Not Turn Off The System
  - Do Not Immediately Disconnect From Network
  - Do Not Run Anti-Virus Programs
    - Changes last-access time of virtually every file
    - Forces system log rotation
  - Do Not Run Registry Or File Cleaner Programs
  - Do Not Install Or Run Any Additional Tools
- Do's of Incident Response
  - Capture Memory
  - Disk Image

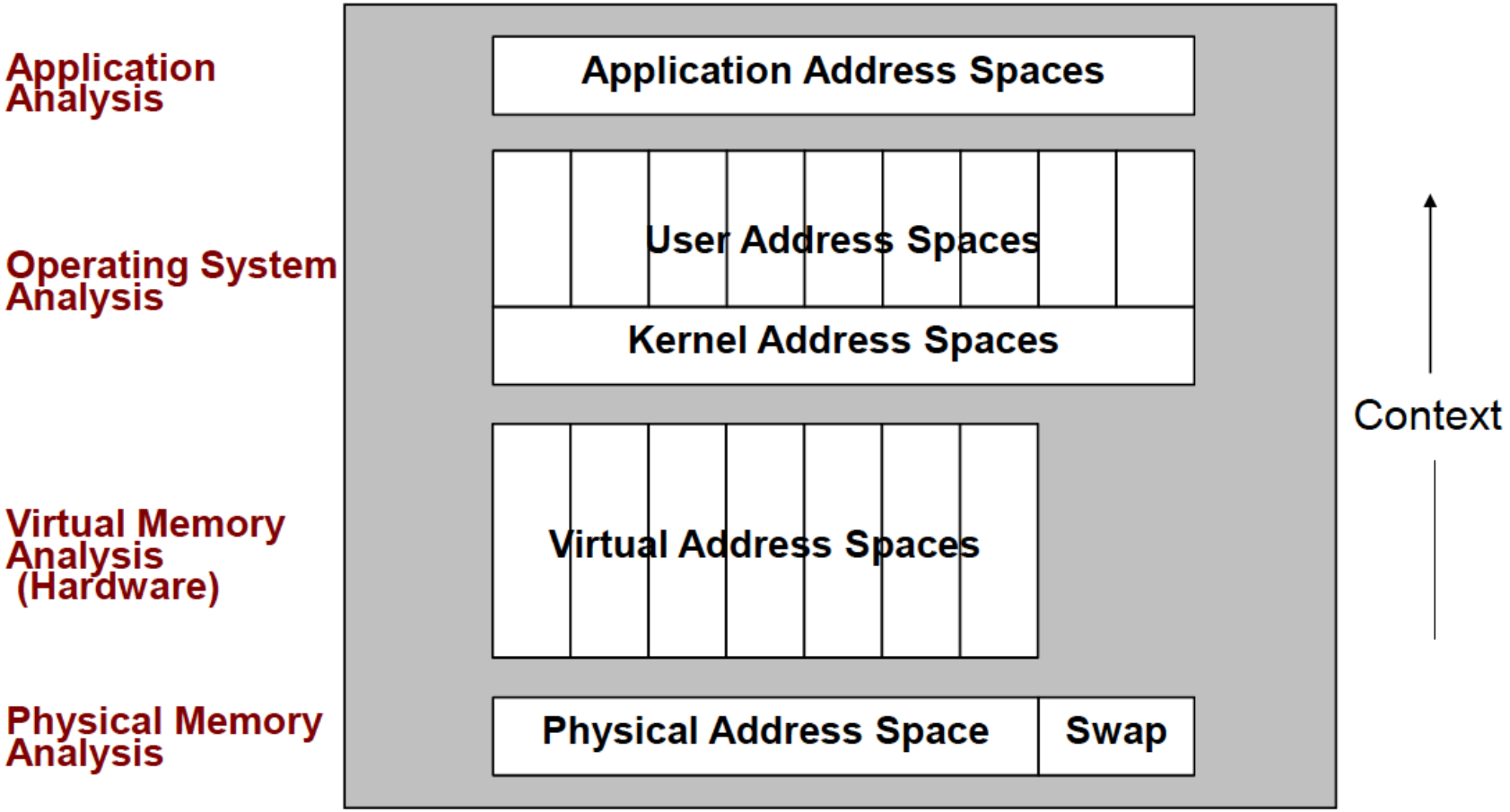
The volatile memory in a system is a gold mine of forensics data, often containing information that cannot be found on the hard drive or anywhere else. Some advanced malware has even evolved to erase any sign of its presence except for the code in memory that it needs to run. For these reasons, those responding to an incident should make every effort to capture a memory image

https://www.cisa.gov/sites/default/files/FactSheets/NCCIC%20ICS_FactSheet_AreYouCompromised_S508C.pdf

© 2025 Volexity Inc.

Memory analysis types

Increasing context allows data “attribution”



© 2025 Volexity Inc.

Analysis Techniques

- Unstructured
- File carving
- Structured
- You don’t have to use one or the other – know the pros/cons of each and use them accordingly

© 2025 Volexity Inc.

Unstructured Analysis

- Strings, grep, less, regexp
- Passwords, domains, chat logs, system logs
- No context (i.e. process attribution) in physical space

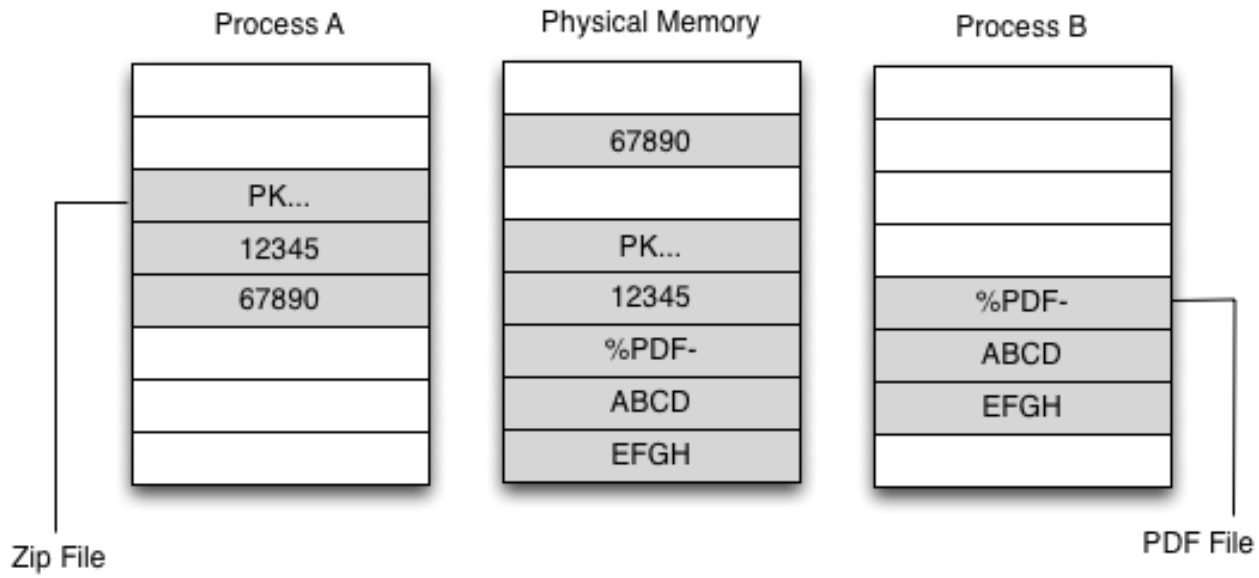
```
$ strings ~/memory.vmem | grep -i password
#define ERROR_CHANGING_PASSWORD
(RASBASE+109)
Enter password for %s in %s at %s: (
       --ftp-password=PAROLA  ftp kullan
       --http-password=PAROLA http kullan
" Language:          passwd(5) password file
challengePassword
```

Process or kernel owned? Which specific process?

© 2025 Volexity Inc.

File Carving

- Foremost, scalpel, PhotoRec, Bulk Extractor
- Office files, images, executables, archives
- Non-contiguous pages/swap in physical space
- Solution: dumpfiles plugin



© 2025 Volexity Inc.

Structured Analysis

- Allows us to reconstruct the views of the operating system and applications
- Interpret data in RAM just like tools on the live system
- Requires offline modeling of data structures
  - Typically through debug symbols and/or reverse engineering
- Most powerful, in-depth type of analysis

© 2025 Volexity Inc.

Getting Started with Volatility

- ☑ Why use Volatility?
- ☑ How to install Volatility
- ☑ Supported platforms/profiles
- ☑ Basic command-line options



© 2025 Volexity Inc.

Volatility2 - Volatility3

- Volatility2 (python2)
  - https://github.com/volatilityfoundation/volatility
  - Previous version of Volatility since ~2008
  - Deprecated in Spring 2025
- Volatility3 (python3)
  - https://github.com/volatilityfoundation/volatility3
  - Current version
  - Active maintenance and new development
  - The focus of this workshop

© 2025 Volexity Inc.

Volatility 3 Format Support

- Raw memory samples
- Windows crash dumps
- Virtualbox/QEMU samples
- VMware samples
- Lime samples
- AVML (lime with awkward padding)
- Leechcore (PCI DMA)

© 2025 Volexity Inc.

How to install Volatility 3

- Python wheel and Releases
  - https://github.com/volatilityfoundation/volatility3/releases
  - pip3 install volatility3
  - Let pip handle dependencies
- Source code
  - Clone the latest repo
  - https://github.com/volatilityfoundation/volatility3
  - Install dependencies through pip (see toml file)

© 2025 Volexity Inc.

Basic usage

$ vol3 -f <file> <plugin> [options]

$ vol3 -h

^ See plugins and global options

$ vol3 <plugin> -h

^ See plugin specific options

- These are like Volatility2, except no --profile
- Volatility3 plugin names include the operating system (i.e., windows.pslist instead of just pslist)

© 2025 Volexity Inc.

Command	Vol2	Vol3
List processes in a table	pslist --profile=PROFILE	windows.pslist
List processes in a tree	pstree --profile=PROFILE	windows.pstree
List process command lines	cmdline --profile=PROFILE	windows.cmdline
List DLLs in a process	dlllist --profile=PROFILE -p PID	windows.dlllist --pid PID
Dump DLL(s)	dlldump [-p PID] [-b BASE] -D DIR	windows.dlllist --dump [--pid PID]
Dump a process	procdump [-p PID] -D DIR	windows.pslist --dump [--pid PID]
Dump process memory	memdump [-p PID] -D DIR	windows.memmap --dump [--pid PID]
Dump a cached file by physical offset	dumpfiles -Q OFFSET -D DIR	windows.dumpfiles --physaddr OFFSET
Scan for MFTs	mftparser	windows.mftscan
List registry hives	hivelist	windows.registry.hivelist
Print registry key	printkey -o OFFSET -K KEY	windows.registry.printkey --offset OFFSET --key KEY

© 2025 Volexity Inc.

External Plugins (Volatility3)

- Like Volatility2, there is also a repository for Volatility3 plugins submitted by members of the community
- https://github.com/volatilityfoundation/community3
- External or 3rd party plugins can be loaded with the -p/--plugin-dirs options

$ **mkdir ~/myplugins**

$ **cp pluginfile.py ~/myplugins**

$ **vol3 -f data.lime --plugin-dirs ~/myplugins windows.myplugin**

© 2025 Volexity Inc.

No Profiles in Volatility 3

- Volatility3 scans memory for the Windows kernel GUID (unique identifier)
- Downloads the PDB (symbol set) from Microsoft, converts to Volatility3 ISF (Intermediate Symbol Format) which is compressed JSON
- Caches the ISF for later use

© 2025 Volexity Inc.

Profiles Pros/Cons

- Volatility 3 doesn't ship with any ISF out of the box. It requires Internet access, either at run time or in advance (create ISF with pdbconv.py and supply to Volatility 3)
- The scanning and downloading can take a little while longer than Volatility 2 initially, but future commands are faster
- No need to scan for KDBG or other values, as the ISF contains direct symbol addresses
- Relies on Microsoft publishing PDBs (they usually do, but sometimes screw up)

© 2025 Volexity Inc.

Optimizing Analysis

- Use Volatility3 to save a JSON config and re-use it on subsequent commands
- The initial work took ~2 minutes but all others take under 3 seconds

$ **time vol3 -f data.lime --save-config config.json windows.pslist**
Volatility 3 Framework 2.4.2
[...]
real **146.82s**
user 62.94s
sys 9.98s
cpu 49%

$ **time vol3 -f data.lime --config config.json windows.pslist**
Volatility 3 Framework 2.4.2
[...]
real **2.63s**
user 1.66s
sys 0.20s
cpu 70%

© 2025 Volexity Inc.

Output Rendering

- Like Volatility 2's --output-format option, Volatility 3 can render results in many different formats.
  - quick (default)
  - pretty (tabular)
  - csv
  - json (one JSON item)
  - jsonl (one JSON item per result)

$ **vol3 -f data.lime -r json windows.pslist**

© 2025 Volexity Inc.

Volatility 3 Column Filters

$ vol3 -f data.lime windows.pslist

PID	PPID	ImageFileName	Offset(V)	Threads	Handles	SessionId	Wow64	CreateTime	ExitTime	File output
4	0	System	0x823c8830	59	403	N/A	False	N/A	N/A	Disabled
376	4	smss.exe	0x820df020	3	19	N/A	False	2010-10-29 17:08:53.000000	N/A	Disabled
600	376	csrss.exe	0x821a2da0	11	395	0	False	2010-10-29 17:08:54.000000	N/A	Disabled
624	376	winlogon.exe	0x81da5650	19	570	0	False	2010-10-29 17:08:54.000000	N/A	Disabled
668	624	services.exe	0x82073020	21	431	0	False	2010-10-29 17:08:54.000000	N/A	Disabled
1580	668	jqs.exe	0x81e0eda0	5	148	0	False	2010-10-29 17:09:05.000000	N/A	Disabled
1664	668	vmtoolsd.exe	0x81fe52d0	5	284	0	False	2010-10-29 17:09:05.000000	N/A	Disabled

[snip]

$ vol3 –f data.lime --filters=ImageFileName,jqs.exe windows.pslist

PID	PPID	ImageFileName	Offset(V)	Threads	Handles	SessionId	Wow64	CreateTime	ExitTime	File output
1580	668	jqs.exe	0x81e0eda0	5	148	0	False	2010-10-29 17:09:05	N/A	Disabled

Notes

- Ending a filter with an exclamation point (!) turns it into a regular expression
- Entering a blank column name, e.g. “,jqs.exe”, will search all columns for the pattern

© 2025 Volexity Inc.

Volatility 3 Process Filtering

```
$ python3 vol.py -r pretty -f Lab1.lime windows.handles --pid 680 868 1928
```

- Volatility 3 requires the full --pid argument name, not -p like Volatility 2
- Volatility 3 uses space separation between process IDs to list multiple at once, not commas like Volatility 2
- You do not need to put multiple PIDs in quotes like normal bash/terminal commands with spaces in an argument

© 2025 Volexity Inc.

Processes

- ☑ Analyzing processes and their resources in memory
- ☑ Kernel and executive process objects
- ☑ Finding and walking the active process list



© 2025 Volexity Inc.

Processes (_EPROCESS)

- Container for resources used when executing a program
- Private virtual address space (DTB) except shared memory and most of kernel memory
- Executable instructions (PE, DLLs)
- Open handles
- Security context (SIDs, privileges)
- Thread(s) of execution

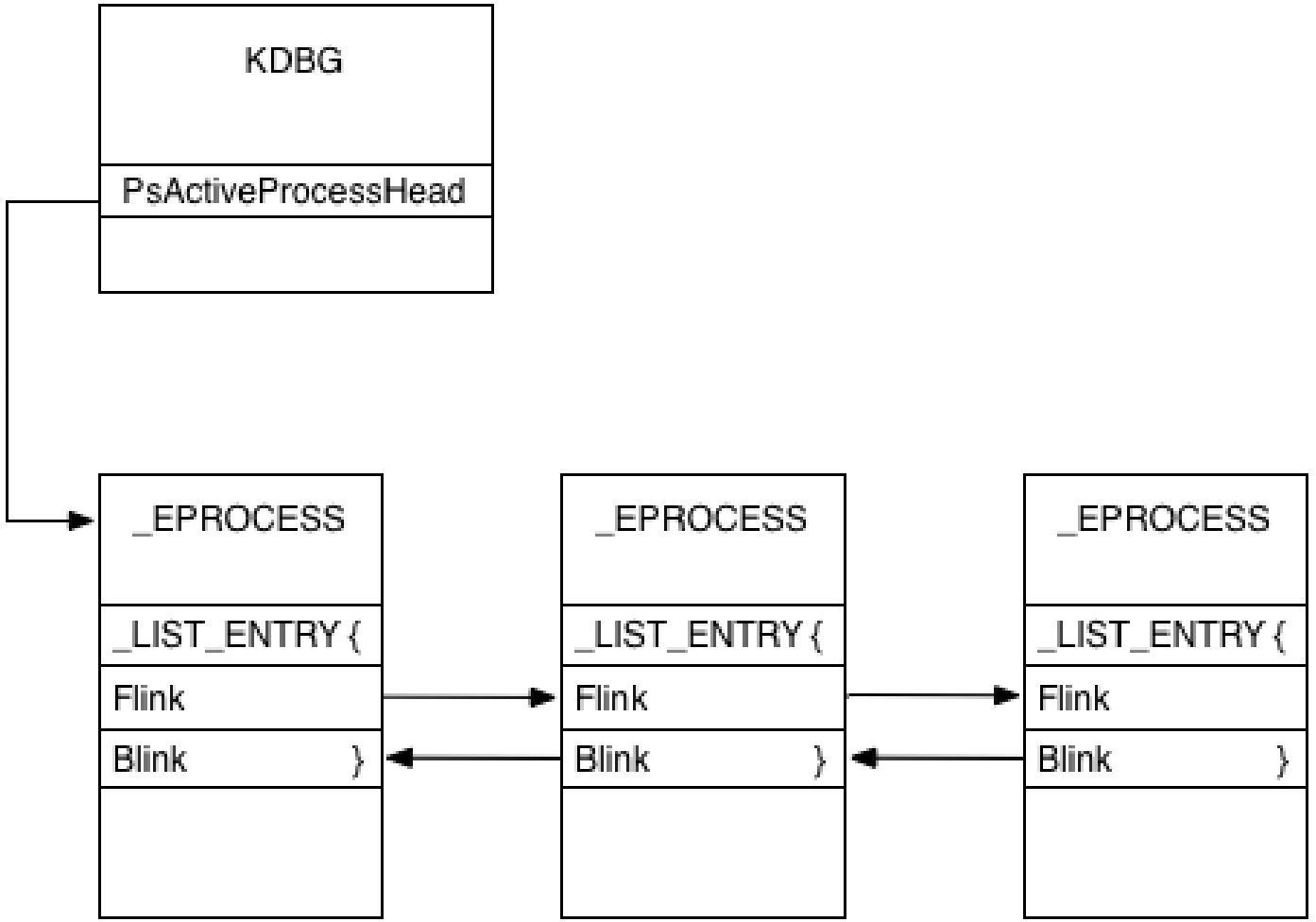
© 2025 Volexity Inc.

PsActiveProcessHead

- Doubly-linked list head pointed to by KDBG
- All live APIs reference this list (indirectly)
  - NtQuerySystemInformation() with SystemInformationProcess class
  - CreateToolhelp32Snapshot() with Process32First()/Process32Next()
  - WMI Win32_Process class
  - EnumProcesses() from psapi.dll
- Kernel mode process list?
  - PsGetCurrentProcess()
  - nt!PsInitialSystemProcess

© 2025 Volexity Inc.

Active process list



© 2025 Volexity Inc.

Listing Processes

$ vol3 –f data.lime windows.pslist

PID	PPID	Image FileName	Offset(V)	Threads	Handles	SessionId	Wow64	CreateTime	ExitTime
3268	844	svchost.exe	0xa0082d3e8080	1	-	0	False	2022-11-22 20:15:31	N/A
4936	2944	userinit.exe	0xa0082d5d1080	0	-	2	False	2022-11-22 20:15:31	**2022-11-22 20:16:03**
1032	4936	explorer.exe	0xa0082d0f4080	43	-	2	False	2022-11-22 20:15:31	N/A
5428	5412	msedge.exe	0xa0082dbc6080	0	-	2	False	2022-11-22 20:15:35	**2022-11-22 20:15:41**
5956	972	TextInputHost.	0xa0082eccb080	10	-	2	False	2022-11-22 20:15:41	N/A
2276	972	StartMenuExper	0xa0082ebe1080	8	-	2	False	2022-11-22 20:15:42	N/A
5360	972	RuntimeBroker.	0xa0082ebe0080	2	-	2	False	2022-11-22 20:15:42	N/A
5108	972	SearchApp.exe	0xa0082ead0380	39	-	2	False	2022-11-22 20:15:43	N/A

[snip]

© 2025 Volexity Inc.

Sessions

- http://volatility-labs.blogspot.com/2012/09/movp-11-logon-sessions-processes-and.html
- Represent user logon sessions; container for processes and objects
- RDP, Terminal Services, Fast-User Switching
- Session 0 isolation since Vista
- _EPROCESS.Session
- _MM_SESSION_SPACE
  - SessionId – unique ID
  - ProcessList - _EPROCESS

© 2025 Volexity Inc.

Sessions Plugin

```
$ vol3 -r pretty -f data.lime windows.sessions
| Session ID | Session Type | Process ID |        Process |                    User Name |                  Create Time
* |        N/A |            - |          4 |         System |                            - | 2024-05-28 00:23:09.000000
* |        N/A |            - |        100 |       Registry |                            - | 2024-05-28 00:23:05.000000
* |        N/A |            - |        444 |       smss.exe |                            - | 2024-05-28 00:23:09.000000
* |          0 |            - |        600 |       csrss.exe |                      /SYSTEM | 2024-05-28 00:23:11.000000
* |          0 |            - |        700 |    wininit.exe |                      /SYSTEM | 2024-05-28 00:23:11.000000
* |          0 |            - |        820 |   services.exe |                      /SYSTEM | 2024-05-28 00:23:12.000000
* |          0 |            - |        828 |      lsass.exe |                      /SYSTEM | 2024-05-28 00:23:12.000000
* |          0 |            - |        544 |    svchost.exe |  ACMECORP/EC2AMAZ-DDPQH3R$ | 2024-05-28 00:23:15.000000
* |          0 |            - |        676 |    svchost.exe |  ACMECORP/EC2AMAZ-DDPQH3R$ | 2024-05-28 00:23:15.000000
* |          0 |            - |        416 |    svchost.exe |  ACMECORP/EC2AMAZ-DDPQH3R$ | 2024-05-28 00:23:15.000000
* |          0 |            - |       1112 |    svchost.exe |  ACMECORP/EC2AMAZ-DDPQH3R$ | 2024-05-28 00:23:15.000000
* |          0 |            - |       1204 |    svchost.exe | NT AUTHORITY/LOCAL SERVICE | 2024-05-28 00:23:15.000000
* |          0 |            - |       1212 |    svchost.exe | NT AUTHORITY/LOCAL SERVICE | 2024-05-28 00:23:15.000000
* |          0 |            - |       1220 |    svchost.exe | NT AUTHORITY/LOCAL SERVICE | 2024-05-28 00:23:15.000000
* |          0 |            - |       1232 |    svchost.exe |  ACMECORP/EC2AMAZ-DDPQH3R$ | 2024-05-28 00:23:15.000000
[snip]
* |          2 |            - |       3060 |    rdpclip.exe |     ACMECORP/Administrator | 2024-05-28 00:28:37.000000
* |          2 |            - |       3628 |    svchost.exe |     ACMECORP/Administrator | 2024-05-28 00:28:37.000000
* |          2 |            - |       1940 |    svchost.exe |     ACMECORP/Administrator | 2024-05-28 00:28:37.000000
* |          2 |            - |       1180 |     sihost.exe |     ACMECORP/Administrator | 2024-05-28 00:28:37.000000
```


© 2025 Volexity Inc.

Parent/Child Processes – windows.pstree

```
$ vol3 --hide-columns Audit ExitTime Cmd Wow64 Path Threads Handles Offset -f data.lime -r pretty
windows.pstree
        | PID  | PPID |  ImageFileName | Sid | CreateTime
        | 3060 | 1168 |   winlogon.exe |   2 | 2023-04-13 14:18:35
**      | 1728 | 3060 |        dwm.exe |   2 | 2023-04-13 14:18:36
**      | 4324 | 3060 |    LogonUI.exe |   2 | 2023-04-13 14:58:07
**      | 1356 | 3060 | fontdrvhost.ex |   2 | 2023-04-13 14:18:36
**      | 2432 | 3060 |   userinit.exe |   2 | 2023-04-13 14:18:43
***     | 4736 | 2432 |   explorer.exe |   2 | 2023-04-13 14:18:43
****    | 2056 | 4736 |   iexplore.exe |   2 | 2023-04-13 14:49:42
*****   | 2744 | 2056 |   iexplore.exe |   2 | 2023-04-13 14:49:42
****    | 5484 | 4736 | spartandiagnos |   2 | 2023-04-13 14:54:16
*****   | 4400 | 5484 | @WanaDecryptor |   2 | 2023-04-13 14:54:48
*****   | 7100 | 5484 | @WanaDecryptor |   2 | 2023-04-13 14:54:24
******  | 5488 | 7100 |   taskhsvc.exe |   2 | 2023-04-13 14:54:25
******* | 5480 | 5488 |    conhost.exe |   2 | 2023-04-13 14:54:25
```


© 2025 Volexity Inc.

Command Line Arguments - windows.cmdline

	PID	Process	Args
*	4	System	-
*	88	Registry	-
*	272	smss.exe	\SystemRoot\System32\smss.exe
*	556	winlogon.exe	winlogon.exe
*	612	services.exe	C:\Windows\system32\services.exe
*	628	lsass.exe	C:\Windows\system32\lsass.exe
*	720	svchost.exe	C:\Windows\system32\svchost.exe -k DcomLaunch -p -s PlugPlay
*	752	fontdrvhost.ex	"fontdrvhost.exe"
*	760	svchost.exe	C:\Windows\system32\svchost.exe -k DcomLaunch -p
*	904	svchost.exe	C:\Windows\system32\svchost.exe -k RPCSS -p
*	948	svchost.exe	C:\Windows\system32\svchost.exe -k DcomLaunch -p -s LSM
*	1048	svchost.exe	C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork -p
*	1060	svchost.exe	C:\Windows\system32\svchost.exe -k LocalService -p -s bthserv
*	1076	svchost.exe	C:\Windows\system32\svchost.exe -k LocalService -p -s BthAvctpSvc
*	1128	svchost.exe	C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted -p -s NcbService

© 2025 Volexity Inc.

Tokens and Security

- ☑ Key structures for tokens, SIDs, and privileges
- ☑ Mapping SIDs to user names with cached registry hives
- ☑ Determining level of access gained by an attacker
- ☑ Changes in privileges before and after Vista
- ☑ How malware enables privileges
- ☑ Detecting DKOM on process tokens



© 2025 Volexity Inc.

Tokens

- Stores security context, user information, privileges, ACLs
- _EPROCESS.Token points to a _TOKEN
- The token stores an array of SIDs, one for each of the security contexts for which the process or thread can act as
- The privilege structures/format changed starting in Vista
- Often a target of DKOM token replacement (i.e. _EPROCESS.Token = NewToken)

© 2025 Volexity Inc.

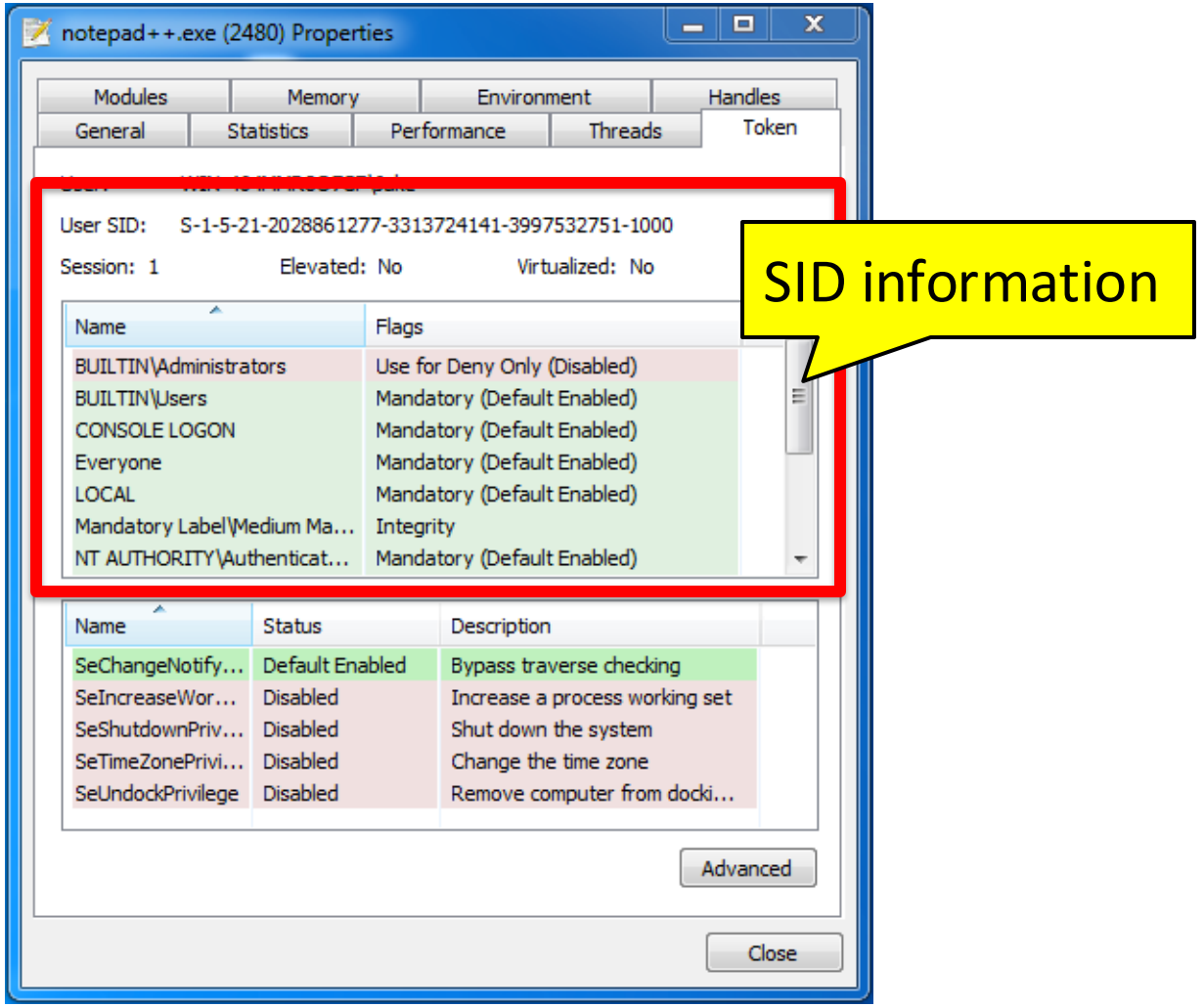
SIDs

- Unique, immutable identifier of a user, group, or a security principal
- Windows grants or denies access to resources based on membership / presence of SIDs in a token
- Example well known sids mapping:

```
(re.compile(r'S-1-5-[0-9-]+-500$'), 'Administrator'),
(re.compile(r'S-1-5-[0-9-]+-501$'), 'Guest'),
(re.compile(r'S-1-5-[0-9-]+-502$'), 'KRBTGT'),
(re.compile(r'S-1-5-[0-9-]+-512$'), 'Domain Admins'),
(re.compile(r'S-1-5-[0-9-]+-513$'), 'Domain Users'),
(re.compile(r'S-1-5-[0-9-]+-514$'), 'Domain Guests'),
(re.compile(r'S-1-5-[0-9-]+-515$'), 'Domain Computers'),
(re.compile(r'S-1-5-[0-9-]+-516$'), 'Domain Controllers'),
[...]
```


© 2025 Volexity Inc.

Sanity check!



© 2025 Volexity Inc.

Mapping SIDs to users

```
$ vol3 -q -f data.lime windows.registry.printkey --key "Microsoft\Windows
NT\CurrentVersion\ProfileList\S-1-5-21-2971954518-3112999052-1389127009-1001"
Last Write Time     Hive Offset   Type Key      Name               Data             Volatile
2023-09-26 03:00:37 0x8a061781a000 REG_EXPAND_SZ ProfileImagePath "C:\Users\absej" False
2023-09-26 03:00:37 0x8a061781a000 REG_DWORD     Flags            0                False
2023-09-26 03:00:37 0x8a061781a000 REG_DWORD     FullProfile       1               False
2023-09-26 03:00:37 0x8a061781a000 REG_DWORD     State            0                False
2023-09-26 03:00:37 0x8a061781a000 REG_BINARY    Sid              "
01 05 00 00 00 00 00 05   ........
15 00 00 00 56 6d 24 b1   ....Vm$.
8c 98 8c b9 61 65 cc 52   ....ae.R"                                                 False
```


© 2025 Volexity Inc.

```
$ vol3 -r pretty -f data.lime windows.getsids --pid 624
  | PID |      Process |          SID |                Name
* | 624 | winlogon.exe |     S-1-5-18 |        Local System
* | 624 | winlogon.exe | S-1-5-32-544 |      Administrators
* | 624 | winlogon.exe |      S-1-1-0 |            Everyone
* | 624 | winlogon.exe |     S-1-5-11 | Authenticated Users
$ vol3 -r pretty -f data.lime windows.getsids --pid 684
  | PID |   Process |          SID |                Name
* | 684 | lsass.exe |     S-1-5-18 |        Local System
* | 684 | lsass.exe | S-1-5-32-544 |      Administrators
* | 684 | lsass.exe |      S-1-1-0 |            Everyone
* | 684 | lsass.exe |     S-1-5-11 | Authenticated Users
$ vol3 -r pretty -f data.lime windows.getsids --pid 1096
  |  PID |      Process |                                                SID |  Name
* | 1096 | explorer.exe |  S-1-5-21-2682149276-1333600406-3352121115-500 |  administrator
* | 1096 | explorer.exe |  S-1-5-21-2682149276-1333600406-3352121115-513 |  Domain Users
* | 1096 | explorer.exe |                                         S-1-1-0 |  Everyone
* | 1096 | explorer.exe |                                      S-1-5-32-545 |  Users
* | 1096 | explorer.exe |                                      S-1-5-32-544 |  Administrators
* | 1096 | explorer.exe |                                          S-1-5-4 |  Interactive
* | 1096 | explorer.exe |                                         S-1-5-11 |  Authenticated Users
* | 1096 | explorer.exe |                                  S-1-5-5-0-206541 |  Logon Session
* | 1096 | explorer.exe |  S-1-5-21-2682149276-1333600406-3352121115-519 |  Enterprise Admins
* | 1096 | explorer.exe | S-1-5-21-2682149276-1333600406-3352121115-1115 |  -
* | 1096 | explorer.exe |  S-1-5-21-2682149276-1333600406-3352121115-518 |  Schema Admins
* | 1096 | explorer.exe |  S-1-5-21-2682149276-1333600406-3352121115-512 |  Domain Admins
* | 1096 | explorer.exe |  S-1-5-21-2682149276-1333600406-3352121115-520 |  Group Policy Creator Owners
```

Grrcon: What type of access did the attacker gain?

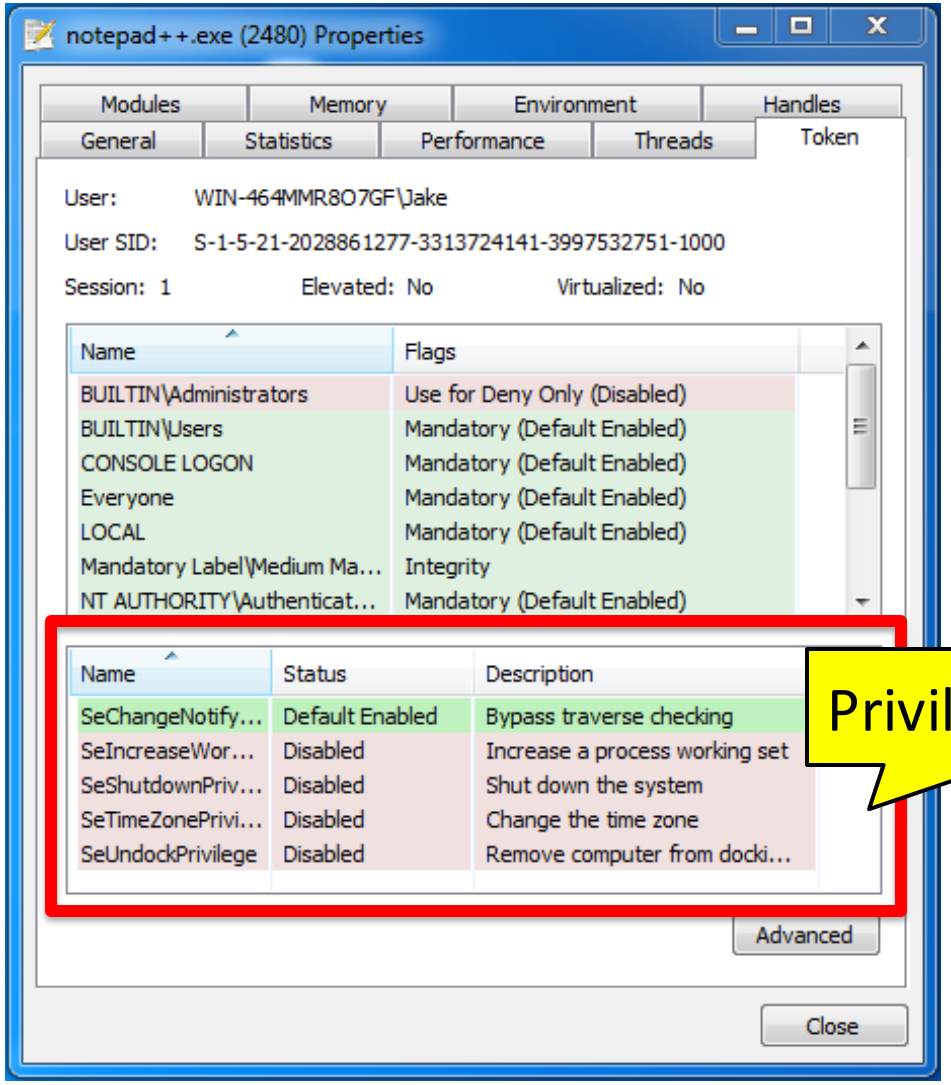
© 2025 Volexity Inc.

Privileges

- Permission to perform specific tasks
- A privilege must be *present* before it can be *enabled*
- How to make a privilege present
  - Security policy editor (SecPol.msc)
  - LsaAddAccountRights()
- How to enable a privilege
  - Enabled by default, inheritance
  - AdjustTokenPrivileges()

© 2025 Volexity Inc.

Sanity check!



© 2025 Volexity Inc.

Commonly Exploited Privileges

- SeBackupPrivilege: back up files and directories
  - Used to bypass file read locks
- SeRestorePrivilege: restore files and directories
  - Used to bypass file write locks
- SeDebugPrivilege: debug and adjust memory of a process owned by another account
  - Used as a precursor to code injection
- SeLoadDriverPrivilege: load or unload a kernel driver
  - Used to activate rootkits
- SeShutdownPrivilege: shutdown the local system
  - Used to kick start MBR rootkits, etc.

© 2025 Volexity Inc.

How to enable a privilege

- Obtain a handle to the process or thread token with TOKEN_ADJUST_PRIVILEGES rights
- Translate strName (i.e. "SE_DEBUG_NAME") to a LUID with LookupPrivilegeValue()
- Configure the number of privileges to be modified (PrivilegeCount) and make sure it’s being enabled (SE_PRIVILEGE_ENABLED)
- Pass the token and privilege structure to AdjustTokenPrivileges()

© 2025 Volexity Inc.

C source code

```
void SetPrivilege(LPCSTR strName) {
    HANDLE hToken = NULL;
    TOKEN_PRIVILEGES privs;

    memset(&privs, 0, sizeof(privs));

    if (OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &hToken)) {
        if (LookupPrivilegeValue(NULL, strName, &privs.Privileges[0].Luid)) {
            privs.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
            privs.PrivilegeCount = 1;
            AdjustTokenPrivileges(hToken, FALSE, &privs, 0, NULL, NULL);
        }
        CloseHandle(hToken);
    }
}
```


© 2025 Volexity Inc.

Which privilege(s) were explicitly enabled?

```
$ vol3 -r pretty -qf data.lime windows.privileges --pid 1096
  |  PID |      Process | Value | Privilege               |              Attributes | Description
* | 1096 | explorer.exe |    23 | SeChangeNotifyPrivilege | Present,Enabled,Default | [snip]
* | 1096 | explorer.exe |    19 | SeShutdownPrivilege     |                 Present | Shut down the system
* | 1096 | explorer.exe |    25 | SeUndockPrivilege       |         Present,Enabled | Remove computer from docking
* | 1096 | explorer.exe |     8 | SeSecurityPrivilege     |                 Present | Manage auditing and security log
* | 1096 | explorer.exe |    17 | SeBackupPrivilege       |                 Present | Backup files and directories
* | 1096 | explorer.exe |    18 | SeRestorePrivilege      |                 Present | Restore files and directories
* | 1096 | explorer.exe |    12 | SeSystemtimePrivilege   |                 Present | Change the system time
* | 1096 | explorer.exe |    20 | SeDebugPrivilege        |         Present,Enabled | Debug programs
* | 1096 | explorer.exe |    10 | SeLoadDriverPrivilege   |         Present,Enabled | Load and unload device drivers
* | 1096 | explorer.exe |    28 | SeManageVolumePrivilege |                 Present | Manage the files on a volume
* | 1096 | explorer.exe |    30 | SeCreateGlobalPrivilege | Present,Enabled,Default | Create global objects
* | 1096 | explorer.exe |    29 | SeImpersonatePrivilege  | Present,Enabled,Default | Impersonate a client
```


© 2025 Volexity Inc.

Process Handles

- ☑ The Windows HANDLE data type
- ☑ How handles can assist forensic efforts
- ☑ 3 tiers of handle tables



© 2025 Volexity Inc.

Handles for forensics

- *Which process has a particular file, registry key, mutex currently open?*
- *What resources is a process currently accessing?*
- You can use the handles plugin to locate the address of a specific object in memory for further investigation (i.e. examining the object in volshell)

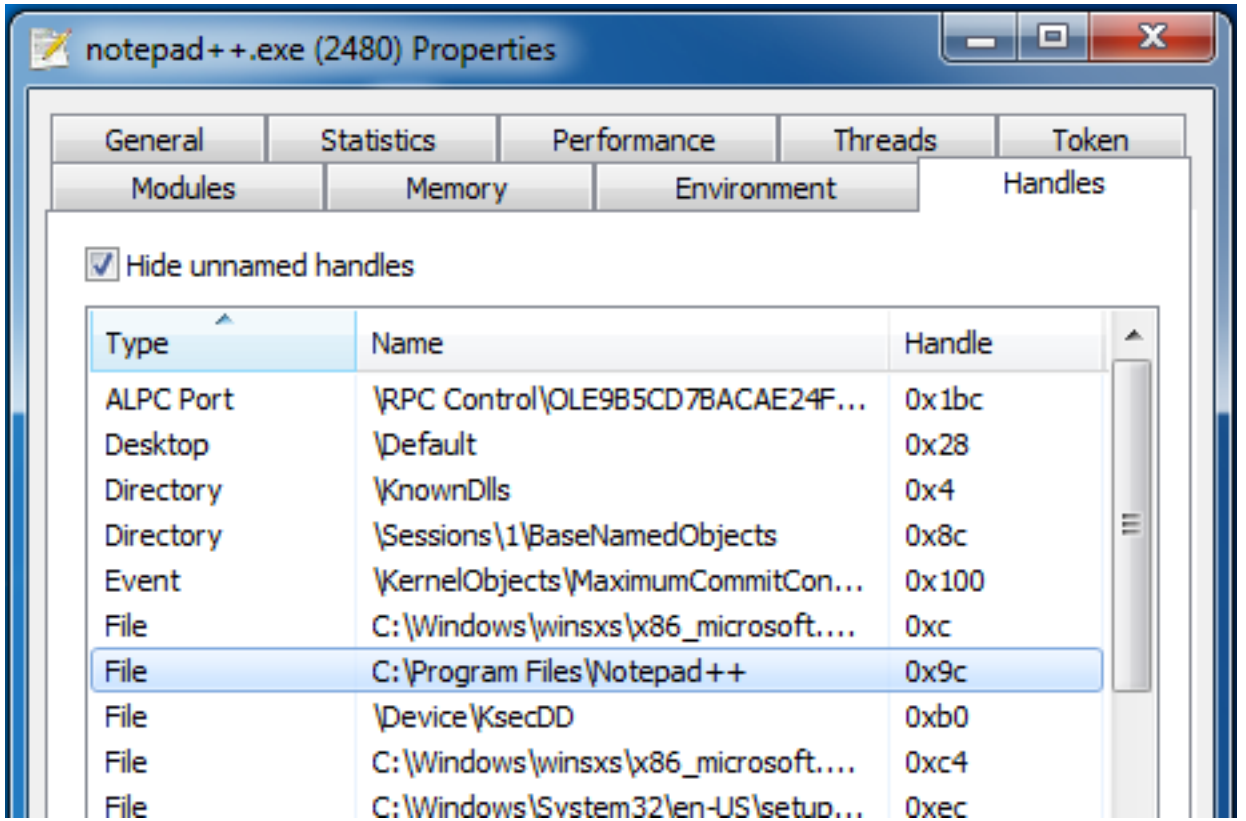
© 2025 Volexity Inc.

What is a handle?

- User-mode applications access objects through handles
- Consistent interface to reference objects
  - Must go through object manager
  - Security and access control is enforced here
- A handle (i.e. HANDLE) is an index into a process-specific handle table
- The HANDLE is valid until the process terminates or it calls CloseHandle()

© 2025 Volexity Inc.

Sanity check!



© 2025 Volexity Inc.

Handles plugin

- Parses the handle table of a process (even those unlinked by rootkits if you specify the --offset parameter)

```
usage: volatility windows.handles.Handles [-h] [--pid [PID [PID ...]]] [--offset OFFSET]

optional arguments:
  -h, --help            show this help message and exit
  --pid [PID [PID ...]]
                        Process IDs to include (all other processes are excluded)
  --offset OFFSET       Process offset in the physical address space
```


© 2025 Volexity Inc.

Volatility 3 Handles

- The windows.handles plugin provides the same functionality as the Volatility2 handles plugin

```
$ vol3 --config config.json windows.handles
Volatility 3 Framework 2.4.2
PID Process Offset           Handle Value Type      GrantedAccess Name
4   System  0x890a9b080480   0x4          Process   0x1fffff      System Pid 4
4   System  0x890a9b129140   0x8          Thread    0x1fffff      Tid 28 Pid 4
4   System  0x890a9b06e0a0   0xc          Mutant    0x1f0001      BcdSyncMutant
4   System  0x9a0656c026a0   0x14         Directory 0xf000f       GLOBAL??
4   System  0x9a0656c0c800   0x1c         Directory 0xf000f       KernelObjects
4   System  0x890a9b087c20   0x20         Event     0x1f0003      LowPagedPoolCondition
4   System  0x890a9b087520   0x24         Event     0x1f0003      HighPagedPoolCondition
4   System  0x890a9b0879a0   0x28         Event     0x1f0003      LowNonPagedPoolCondition
4   System  0x890a9b087a20   0x2c         Event     0x1f0003      HighNonPagedPoolCondition
[...]
```


© 2025 Volexity Inc.

Cobalt Strike via Handles

$ vol3 -r pretty –f data.lime windows.handles --pid 3704

| PID | Process | Offset | HandleValue | Type | GrantedAccess |

[snip]

* | 3704 | https-beacon.e | 0xdf01c2ee0560 | 0x5a0 | File | 0x12019f | **\Device\NamedPipe\postex_3e4e**

* | 3704 | https-beacon.e | 0xdf01c28128f0 | 0x5b4. | File | 0x12019f | **\Device\NamedPipe\postex_868a**

Cobalt Strike uses named pipes for most of its post-exploitation jobs. We use named pipes for post-ex tools that inject into an explicit process (screenshot, keylog). Our fork&run tools largely use named pipes to communicate results back to Beacon too. F-Secure's Detecting Cobalt Strike Default Modules via Named Pipe Analysis discusses this aspect of Cobalt Strike's named pipes. We introduced the ability to change these pipenames in Cobalt Strike 4.2. Set post-ex -> pipename in your Malleable C2 profile. The default name for these pipes is **\\.\pipe\postex_####** in Cobalt Strike 4.2 and later. Prior to 4.2, the default name was random-ish.

© 2025 Volexity Inc.

Remote file system handles

- Detect lateral movement
  - net use […]
  - net share […]
- How does the OS translate drive mappings like Z: to \\192.168.3.2\share?
  - Alias, symbolic links
- Handle prefixes
  - \Device\Mup
    - Multiple UNC Provider
  - \Device\Lanmanredirector

© 2025 Volexity Inc.

Remote mapped shares

```
$ vol3 -f data.lime windows.handles
   PID    Process        Offset           Value  Type   GrantedAccess    Name
* |  976 | svchost.exe | 0xfa800fe8ee50 | 0x610 | File | 0x100000      | \Device\Mup\;:0\EXCHANGE\IPC$
```


© 2025 Volexity Inc.

Other indicators

- \Device\LanmanRedirector***WIN-L8ANB3HN32M***\IPC$
  - This one uses computer name instead of IP
  - Also using a standard share rather than a path
  - ADMIN$ and C$ show up the same way
- symlinkscan plugin
  - Drive letter mapping
  - Connection string (i.e. \Device\Mup\ADDR\PATH)
  - Date and time initiated

© 2025 Volexity Inc.

Symbolic links

```
$ vol3 -f data.lime windows.symlinkscan
0x32032e0   2016-08-24 05:18:44    Global  \GLOBAL??
0x94a9550   2016-08-24 12:51:13    NDPROXY \Device\NDProxy
0xd3bef50   2016-08-24 05:19:37    PEAuth  \Device\PEAuth
0x122c8c30  2016-08-24 12:39:06    Global  \Global??
0x1cf44750  2016-08-24 12:39:06    3   \Sessions\3\BaseNamedObjects
0x1058028e0 2016-08-24 05:18:53    NDISWANIPV6 \Device\NDMP9
0x105b02620 2016-08-24 05:18:53    CdRom0  \Device\CdRom0
0x1a5987f40 2016-08-24 13:30:29    Y:\Device\LanmanRedirector\;Y:00000000003e8906\EXCHANGE\C$
```


© 2025 Volexity Inc.



Exercise 1

- Detecting Privilege Escalation with Volatility 3

© 2025 Volexity Inc.

Process Memory

- ☑ Where is process memory and what does it contain
- ☑ Sources of process memory (working sets, page tables)
- ☑ Mapping and coding with process address spaces
- ☑ Virtual Address Descriptors
- ☑ Translating VAD protections
- ☑ Dumping, scanning, and searching VADs



© 2025 Volexity Inc.

What's in process memory?

- Executable images (exe and dll)
- PEB, TEB
- Stacks and heaps
- Mapped files, shared memory
- Environment variables
- Application data
  - Internet history, passwords, chat logs, etc

© 2025 Volexity Inc.

Virtual Address Descriptors (VADs)

- Describe the layout of process' virtual address space
  - Allow you to reserve or commit a contiguous range of memory that all have the same characteristics
  - PTEs are not created for reserved ranges and the memory cannot be accessed until it’s committed
- Self-balancing AVL tree (for fast lookups)
- What's in a VAD node
  - Range (start and end address)
  - Page protections
  - Full paths of DLLs and files that occupy the range

© 2025 Volexity Inc.

Vadinfo plugin

```
$ vol3 -f data.lime windows.vadinfo
PID  Process     Offset             Start VPN   End VPN    Tag   Protection              CommitCharge PrivateMemory
        File
732  svchost.exe 0xfffffa80058b1f40 0x1610000   0x168ffff  VadS PAGE_READWRITE          20           1      N/A
732  svchost.exe 0xfffffa8006261da0 0x76f10000 0x7702efff Vad  PAGE_EXECUTE_WRITECOPY  4            0
     \Windows\System32\kernel32.dll

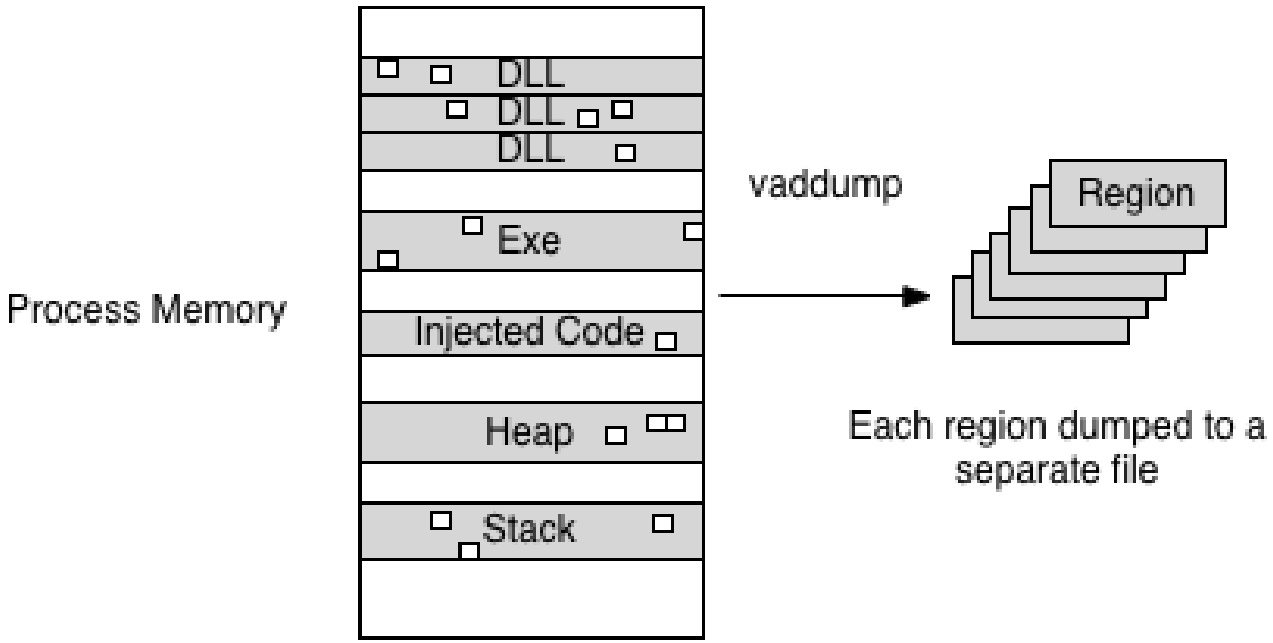
732  svchost.exe 0xfffffa80070f8ad0 0x1a90000   0x1b0ffff  VadS PAGE_READWRITE          40           1      N/A
732  svchost.exe 0xfffffa8006653090 0x1b10000   0x1c0ffff  VadS PAGE_READWRITE          147          1      N/A
732  svchost.exe 0xfffffa8006266850 0x77130000 0x772d9fff Vad  PAGE_EXECUTE_WRITECOPY  14           0
     \Windows\System32\ntdll.dll

732 svchost.exe 0xfffffa80062692b0 0x77030000 0x77129fff Vadm PAGE_EXECUTE_WRITECOPY   3            0
\Windows\System32\user32.dll

732  svchost.exe 0xfffffa80061dfd70 0x7f0e0000 0x7ffdffff VadS PAGE_READONLY           0            1      N/A
```


© 2025 Volexity Inc.

windows.vadinfo --dump



© 2025 Volexity Inc.

Dumping VADs

- Extract each VAD range to separate files
- Zero-padded to maintain proper offsets within each file
- Can be filtered to a specific address with **--address**

```
$ vol3 -f data.lime windows.vadinfo --pid 732 -dump
[cut same output as vadinfo]

$ ls pid.732*dmp
pid.732.vad.0x10000-0x1ffff.dmp                pid.732.vad.0x744e0000-0x744e2fff.dmp
pid.732.vad.0x7fefccd0000-0x7fefccdefff.dmp    pid.732.vad.0x7ffe0000-0x7ffeffff.dmp
pid.732.vad.0x1130000-0x11affff.dmp            pid.732.vad.0x76f10000-0x7702efff.dmp
pid.732.vad.0x7fefcd80000-0x7fefcd93fff.dmp    pid.732.vad.0x7fffff9e000-0x7fffff9ffff.dmp
[snip]

$ ls pid.732*dmp | wc -l
109
```


© 2025 Volexity Inc.

PEB and DLLs

- ☑ The PEB and DLL linked lists
- ☑ Determining which DLLs were loaded by shell code
- ☑ Cross-view DLL detection and advanced hiding techniques
- ☑ Detecting holes in user mode memory
- ☑ ZeroAccess and path name attacks
- ☑ DLLs on wow64



© 2025 Volexity Inc.

Process Environment Block

- _EPROCESS.Peb points to the _PEB
- Why is it useful?
  - Base address of executable in process' memory
  - Process parameters (command-line), current working directory
  - Loader information (DLLs)
  - Heap information
- Exists in user mode of the process AS
  - Easily manipulated without kernel code
  - Can be paged

© 2025 Volexity Inc.

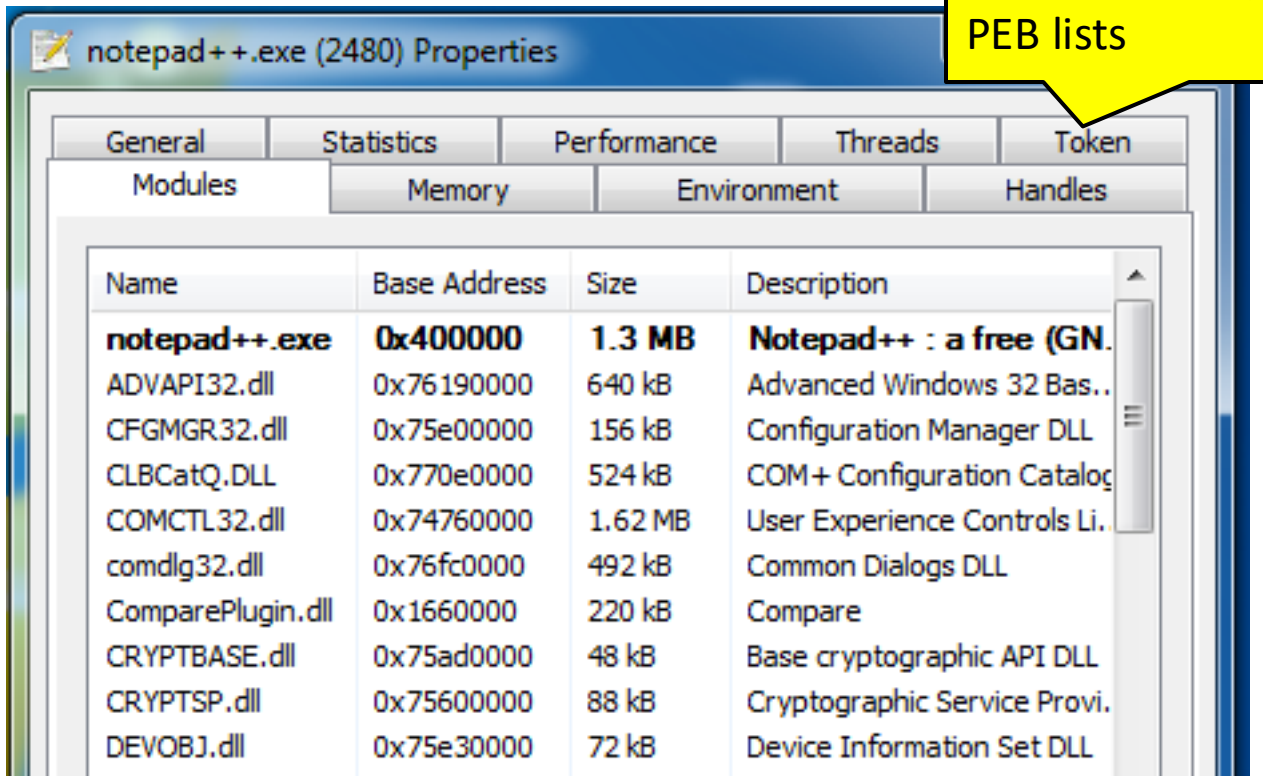
DLLs

- Loaded automatically by the Windows loader for any libraries specified in a process' IAT
- Can be loaded explicitly at any time by code in the process space by calling LoadLibrary() or a derivative
  - LoadLibraryEx() – flexible, more options
  - LdrLoadDll() – native NT API
- Any libraries loaded via these methods are recorded in the PEB's _PEB_LDR_DATA structure

© 2025 Volexity Inc.

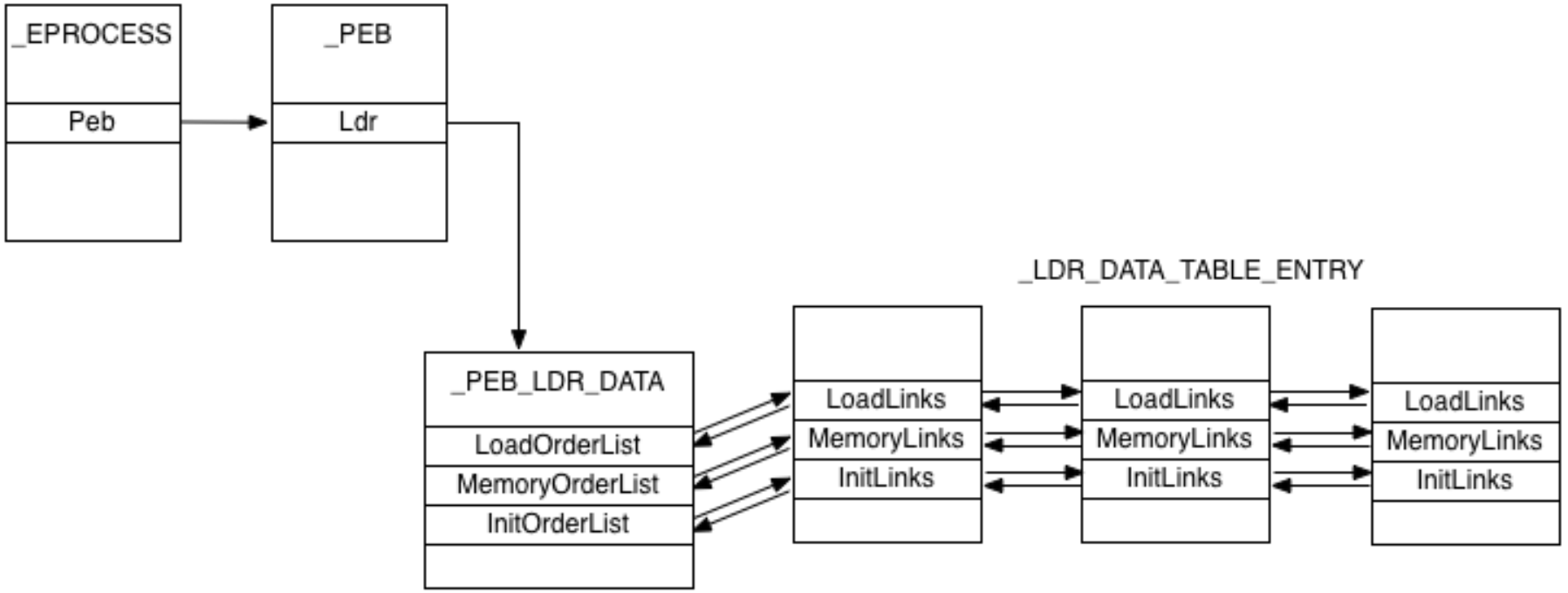
Sanity check!

These entries are enumerated by walking the PEB lists



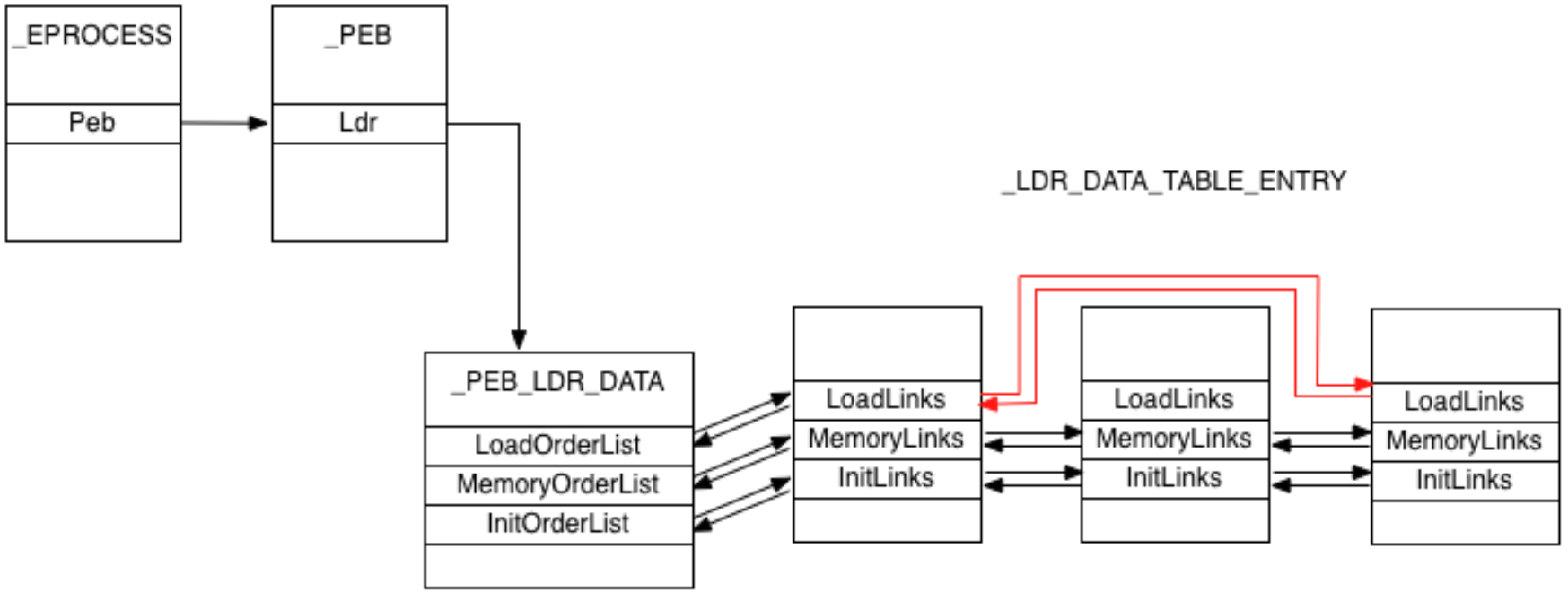
© 2025 Volexity Inc.

DLL linked lists



© 2025 Volexity Inc.

Unlinking DLLs



© 2025 Volexity Inc.

```
//http://www.openrce.org/blog/view/844/How_to_hide_dll

#define CUT_LIST(item) \
    item.Blink->Flink = item.Flink; \
    item.Flink->Blink = item.Blink

int CRogue::UnlinkDll(HANDLE hModToHide)
{
    PPEB_LDR_DATA LdrData = NULL;
    PLDR_MODULE Module = NULL;

    __asm {
        mov     eax, fs:30h             //Find the PEB
        mov     ebx, [eax+0Ch]          //PEB.Ldr
        mov     [LdrData], ebx
    };

    for (
        Module = (PLDR_MODULE)LdrData->InLoadOrderModuleList.Flink;
        Module->DllBase != 0;
        Module = (PLDR_MODULE)Module->InLoadOrderModuleList.Flink
    ) {
        if ((HMODULE)Module->DllBase == hModToHide) {
            CUT_LIST(Module->InLoadOrderModuleList);
            CUT_LIST(Module->InInitializationOrderModuleList);
            CUT_LIST(Module->InMemoryOrderModuleList);
            ZeroMemory(Module, sizeof(LDR_MODULE));
            break;
        }
    }

    return 0;
}
```

Unlinking all 3 lists and zeroing the hidden module structure.

© 2025 Volexity Inc.

Cross referencing VADs and DLLs

- LdrModules plugin references the memory-mapped image files in the VAD nodes with the 3 PEB lists of DLLs
  - Basically looks for any VADs with FilePointer and "MZ" at the base address
- There are some rules/exceptions
  - The *.exe is never in the InInitializationOrderList
  - It’s possible to memory-map PE files that legitimately won't end up in the PEB lists (*.fon, *.tlb, *.cpl, *.mui)
    - MapViewOfFileEx()
    - LoadLibraryEx() with LOAD_LIBRARY_AS_DATAFILE

© 2025 Volexity Inc.

LdrModules Plugin

Application executables do not have a DLL initialization

```
$ vol3 -f data.lime windows.malware.ldrmodules --pid 1912
Pid   Process        Base       InLoad InInit InMem MappedPath
1912 VMwareTray.exe 0x400000   True   False  True  \Program Files\VMware\VMware Tools\VMwareTray.exe
1912 VMwareTray.exe 0x360000   True   True   True  \Program Files\VMware\VMware Tools\intl.dll
1912 VMwareTray.exe 0x7c800000 True   True   True  \WINDOWS\system32\kernel32.dll
1912 VMwareTray.exe 0x77dd0000 True   True   True  \WINDOWS\system32\advapi32.dll
1912 VMwareTray.exe 0x4f0000   True   True   True  \Program Files\VMware\VMware Tools\iconv.dll
[snip]
```


© 2025 Volexity Inc.

Ldrmodules vs Injected Code

```
$ vol3 -f data.lime windows.malware.ldrmodules --pid 1928
Pid   Process    Base        InLoad InInit InMem MappedPath
1928 lsass.exe 0x80000     False  False  False N/A
1928 lsass.exe 0x7c900000 True   True   True  \WINDOWS\system32\ntdll.dll
1928 lsass.exe 0x1000000  True   False  True  N/A
1928 lsass.exe 0x870000   True   True   True  N/A
1928 lsass.exe 0x7c800000 True   True   True  \WINDOWS\system32\kernel32.dll

$ vol3 -f data.lime windows.vadinfo --pid 1928 --address 0x80000
PID Process    Start   End     Tag Protection             CommitCharge  PrivateMemory       File
1928 lsass.exe 0x80000 0xf9fff Vad PAGE_EXECUTE_READWRITE 0             0                   N/A


$ vol3 -q -f data.lime windows.dlllist --pid 1928 | grep 0x80000
$
```


© 2025 Volexity Inc.

Code Injection

- ☑ Remote library injection
- ☑ Remote shell code injection
- ☑ Detecting reflective DLL loaders
- ☑ An analysis of process hollowing
- ☑ Using Malfind to automatically detect injected code segments



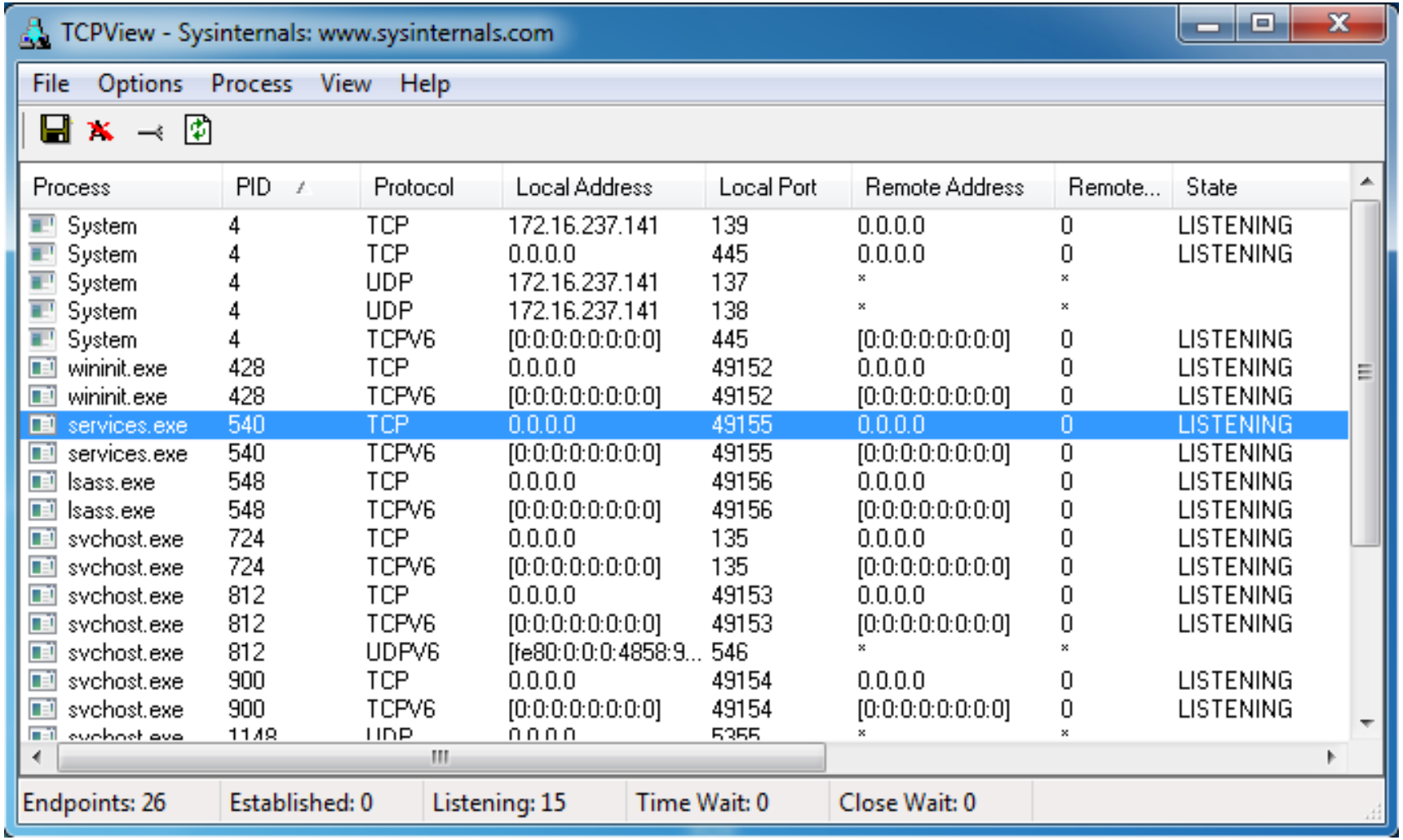
© 2025 Volexity Inc.

Code Injection

- Typically refers to staging code in the memory of another process and then executing it from within the target process' context
- It is a means of stealth and persistence
  - Makes activity appear to originate from the target process
  - If a malicious process is found it’s easy to terminate
- How you detect it depends on the method used to perform the injection

© 2025 Volexity Inc.

Who's the culprit?



© 2025 Volexity Inc.

Types of Code Injection

- Remote DLL injection
- Remote shell code injection
- Reflective DLL loading
- Process hollowing

© 2025 Volexity Inc.

Remote DLL Injection via CreateRemoteThread

1. Process A (malware) enables Debug privilege
2. Process A opens a handle to Process B (victim) by calling OpenProcess() and specifies at least the following privileges
   - PROCESS_CREATE_THREAD
   - PROCESS_VM_OPERATION
   - PROCESS_VM_WRITE
3. Process A allocates memory in Process B using VirtualAllocEx() to store a string with the path to the DLL to be injected
   - PAGE_READWRITE
4. Process A writes "C:\path\to\bad.dll" into Process B's memory with WriteProcessMemory()
5. Process A calls CreateRemoteThread() and points the thread's start address at LoadLibrary(). The thread's parameter points to the allocated region in Process B's memory that contains the DLL path
6. Process A calls VirtualFree() to free the memory in process B, it calls CloseHandle() on Process B's process and thread.

© 2025 Volexity Inc.

Remote DLL Injection via QueueUserAPC

- Starts out the same as before
- Process A calls OpenThread() to gain a reference to a running thread in Process B
- Process A calls QueueUserAPC() with address to execute set to LoadLibrary() and the parameter set to address of the bad.dll string

© 2025 Volexity Inc.

Detection (LoadTime/Count and PEB vs VAD)

- *Typically, no special detection is necessary*
- The DLL was loaded via LoadLibrary() so it should be in the 3 PEB linked lists
  - Suspicious LoadTime
  - Suspicious LoadCount
- However, if the DLL unlinked itself after loading, you can find it with ldrmodules (cross-reference with VAD)

© 2025 Volexity Inc.

Remote Shell code Injection

- Same two steps as before
  - Enable DEBUG
  - Open a process handle
- Instead of allocating a PAGE_READWRITE region for a DLL path, use PAGE_EXECUTE_READWRITE for shellcode
  - Optional regions may be allocated for the thread's parameters (transfer of data from process A to process B)
- When calling CreateRemoteThread or QueueUserAPC, the start address points at the shellcode

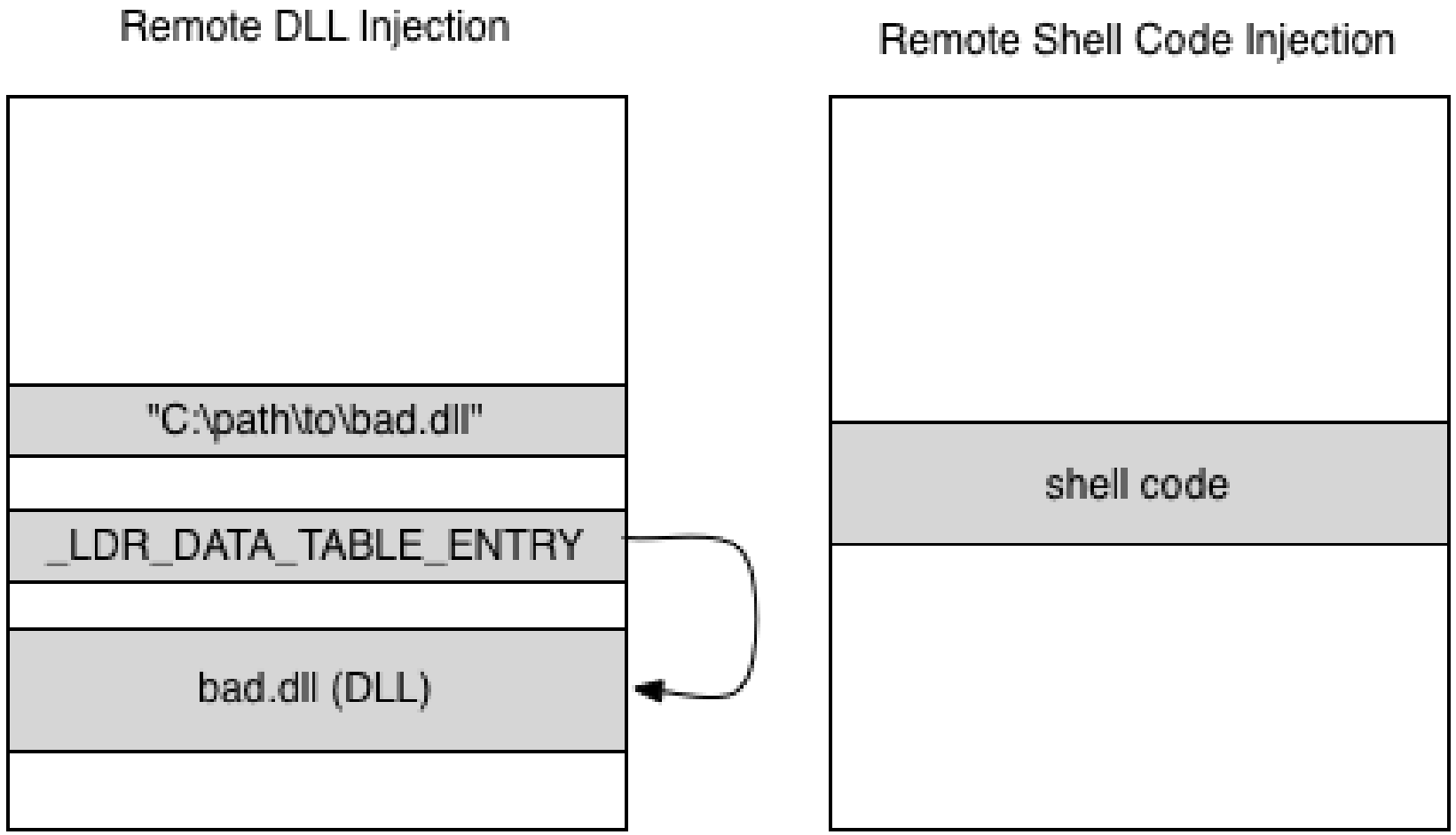
© 2025 Volexity Inc.

Detection (VAD Attributes)

- We look for the following characteristics
  - PAGE_EXECUTE_READWRITE allocation
  - PrivateMemory
  - VadS
  - Not backed by a file
  - Fully committed
  - Non-zeroed

© 2025 Volexity Inc.

Injection Artifacts



© 2025 Volexity Inc.

Reflective DLL Injection

- DLLs can be loaded in memory without ever existing on disk
- Bypasses LoadLibrary() and the Windows loader entirely
- Thus it never enters the PEB lists in the first place

© 2025 Volexity Inc.

Detection

- Artifacts are identical to that of Remote Shell Code Injection
- Why?
  - Because you still need to allocate memory to hold the DLL
  - The memory still needs to be R/W/E
  - If the DLL is not on disk, it cannot be mapped
  - Thus _MMVAD_SHORT (VadS) will be the structure used
  - It will be private to the process (not shareable)

© 2025 Volexity Inc.

Malfind Plugin

- Malfind is designed to pick out VAD segments with this matching criteria
- False positives are possible, weed them out by looking at the hex dump and disassembly
  - MZ at the base is almost always bad
  - Function prologue at the base is almost always bad
  - Assembly instructions like PUSH, POP, MOV, JMP, CALL at the base are likely bad
  - All zeros or non-sense/garbage assembly instructions are probably false positives

© 2025 Volexity Inc.

Malfind vs Reflective Injection

```
$ vol3 -q -f data.lime windows.malware.malfind
1928lsass.exe 0x870000 0x9a7fff   Vad  PAGE_EXECUTE_READWRITE    0    0    Disabled MZ header
4d 5a 90 00 03 00 00 00   MZ......
04 00 00 00 ff ff 00 00   ........
b8 00 00 00 00 00 00 00   ........
40 00 00 00 00 00 00 00   @.......
00 00 00 00 00 00 00 00   ........
00 00 00 00 00 00 00 00   ........
00 00 00 00 00 00 00 00   ........
00 00 00 00 08 01 00 00   ........
0x870000:    dec ebp
0x870001:    pop edx
0x870002:    nop
0x870003:    add byte ptr [ebx], al
0x870005:    add byte ptr [eax], al
0x870007:    add byte ptr [eax + eax], al
0x87000a:    add byte ptr [eax], al
```


© 2025 Volexity Inc.

Malfind picking up API hook stubs

```
$ vol3 -q -f data.lime windows.malware.malfind
6176 RasTls.exe  0x1e70000 0x1f6ffff   VadS PAGE_EXECUTE_READWRITE    256 1    Disabled N/A
8b 0c 24 49 81 f7 da 57   ..$I...W
35 90 72 03 73 01 e8 b8   5.r.s...
e0 b8 12 18 81 c9 6f 59   ......oY
16 ca ba 13 32 2f f9 e9   ....2/..
01 00 00 00 e9 0d 44 ae   ......D.
c6 26 4f 74 03 75 01 73   .&Ot.u.s
42 73 03 72 01 e8 7a 03   Bs.r..z.
7b 01 e9 e9 01 00 00 00   {.......
0x1e70000:   mov ecx, dword ptr [esp]
0x1e70003:   dec ecx
0x1e70004:   xor edi, 0x903557da
0x1e7000a:   jb  0x1e7000f
0x1e7000c:   jae 0x1e7000f
0x1e7000e:   call 0x149fe0cb
```


© 2025 Volexity Inc.

$ vol3 -f Emotet.lime windows.cmdline

```
PID     Process         Args
4       System          Required memory at 0x20 is not valid (process exited?)
312     smss.exe        \SystemRoot\System32\smss.exe
[snip]
4780    rundll32.exe    C:\Windows\SysWOW64\rundll32.exe "C:\Users\Christian\AppData\Local\Wladsodhvynb\zfrwcmnngjl.jkk",Control_RunDLL
3508    svchost.exe     C:\Windows\system32\svchost.exe -k PrintWorkflow -s PrintWorkflowUserSvc
```

$ vol3 -f Emotet.lime windows.malware.malfind

```
4780  rundll32.exe  0x48d0000  0x48d2fff  VadS  PAGE_EXECUTE_READWRITE  3  1  Disabled  MZ header
4d 5a 80 00 01 00 00 00   MZ……
04 00 00 00 ff ff 00 00   ……..
b8 00 00 00 00 00 00 00   ……..
40 00 00 00 00 00 00 00   @…….
4780  rundll32.exe  0x4900000  0x4900fff  VadS  PAGE_EXECUTE_READWRITE  1  1  Disabled  Function prologue
55 89 e5 90 90 90 9a 11   U…….
00 90 04 33 00 89 ec 5d   …3…]
c3 48 83 ec 20 e8 e6 0f   .H……
fe ff 48 83 c4 20 cb 00   ..H…..
0x4900000:  push    ebp
0x4900001:  mov     ebp, esp
[snip]
```

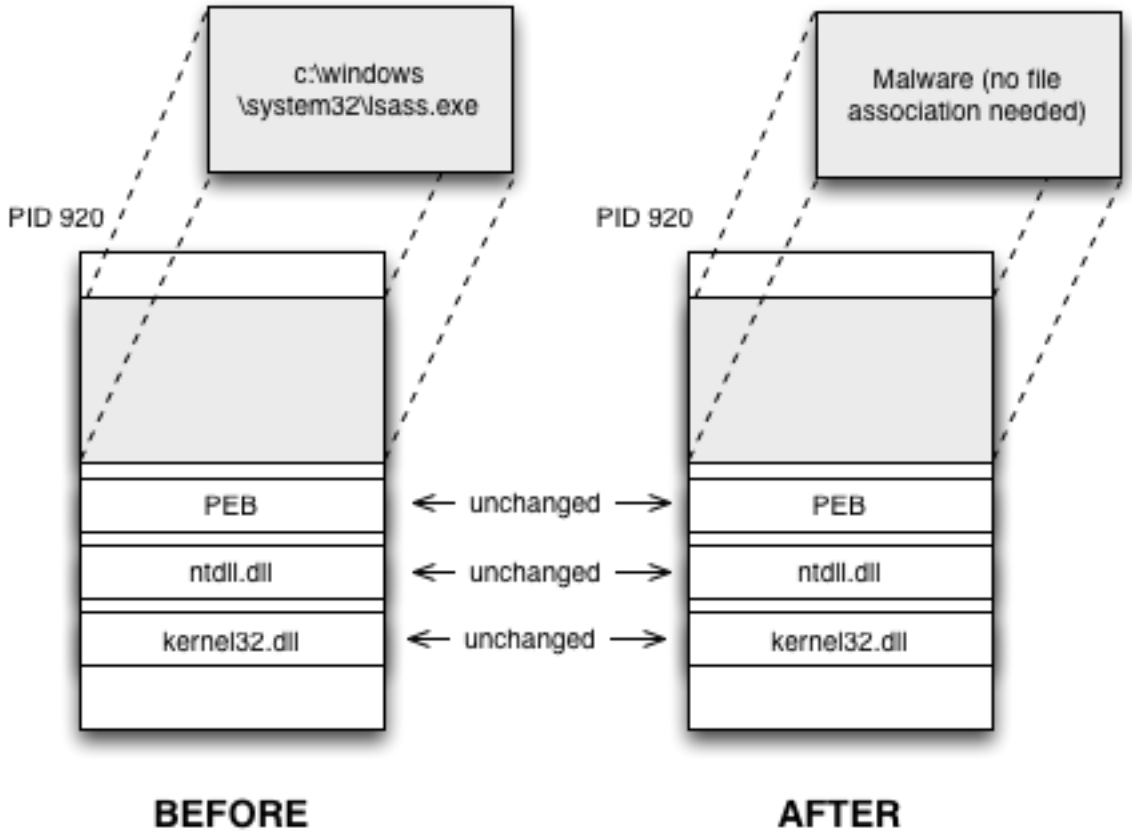

© 2025 Volexity Inc.

Process Hollowing

- Start a new process (typically a well-known one like lsass.exe) in suspended state
- lsass.exe shows up in process listings, PEB, its full path on disk is %system%\lsass.exe as expected, etc.
- Hollow the process
  - Free the memory block with the executable's base address, re-allocate a new one at the same address
  - Call NtUnmapViewOfFile to disassociate lsass.exe then overwrite instructions
- Before resuming the thread, change the context to point at the new code

© 2025 Volexity Inc.

Visualizing Early Process Hollowing



© 2025 Volexity Inc.

lsass.exe Hollowing

```
$ vol3 -q -f data.lime windows.pslist | grep lsass.exe
680   624   lsass.exe 0x81e70020     19    342   0     False     2010-10-29 17:08:54.000000    N/A   Disabled
868   668   lsass.exe 0x81c498c8     2     23    0     False     2011-06-03 04:26:55.000000    N/A   Disabled
1928  668   lsass.exe 0x81c47c00     4     65    0     False     2011-06-03 04:26:55.000000    N/A   Disabled

$ vol3 -q -f data.lime windows.cmdline --pid 680 868 1928
PID   Process    Args
680   lsass.exe C:\WINDOWS\system32\lsass.exe
868   lsass.exe "C:\WINDOWS\\system32\\lsass.exe"
1928  lsass.exe "C:\WINDOWS\\system32\\lsass.exe”

$ vol3 -q -r pretty -f data.lime --filters "Start VPN",16777216 windows.vadinfo --pid 680 868 1928
    |   PID |    Process | Start VPN |    End VPN | Protection             | CommitCharge | File                          |
*   |   680 | lsass.exe | 0x1000000 | 0x1005fff | PAGE_EXECUTE_WRITECOPY |            1 | \WINDOWS\system32\lsass.exe |
*   |   868 | lsass.exe | 0x1000000 | 0x1005fff | PAGE_EXECUTE_READWRITE |            2 |                           N/A |
*   |  1928 | lsass.exe | 0x1000000 | 0x1005fff | PAGE_EXECUTE_READWRITE |            2 |                           N/A |
```


© 2025 Volexity Inc.

Hollow process variations

- We have seen variations of process hollowing that are designed to be extra tricky and difficult to detect
- Example:
  - Malware starts (suspended) a legit copy of a signed Microsoft executable
  - Malware "hollows" it out, but replaces the buffer with a copy of the signed file from disk, rather than a totally different payload
  - The payload is injected elsewhere in process memory
  - The AddressOfEntryPoint (first instruction) of the hollowed process is then changed to point at the injected code
  - In this case, dumping the VADs as we did before will yield practically the same content
    - The strings will match, at least. The code will differ slightly, but that should be expected

© 2025 Volexity Inc.

Automation

- hollowfind plugin
  - Submitted to the 2016 Volatility Plugin Contest
  - https://cysinfo.com/detecting-deceptive-hollowing-techniques/
- Compares PEB and VAD information:
  - File mapping in PEB vs no file & RWX in VAD
- Disassembles the beginning bytes for spotting control flow redirection

© 2025 Volexity Inc.

hollowprocesses vs Stuxnet

668 services.exe Unexpected protection (PAGE_EXECUTE_READWRITE) for DLL in the PEB's load order list (0x13f0000) with path C:\WINDOWS\system32\KERNEL32.DLL.ASLR.0360c5e2

940 svchost.exe Unexpected protection (PAGE_EXECUTE_READWRITE) for DLL in the PEB's load order list (0xd00000) with path C:\WINDOWS\system32\KERNEL32.DLL.ASLR.0360c8ee

868 lsass.exe **Unexpected protection (PAGE_EXECUTE_READWRITE) for VAD hosting the process executable (0x1000000) with path <Non-File Backed Region>**

868 lsass.exe **Unexpected protection (PAGE_EXECUTE_READWRITE) for DLL in the PEB's load order list (0x1000000) with path C:\WINDOWS\system32\lsass.exe**

1928 lsass.exe Unexpected protection (PAGE_EXECUTE_READWRITE) for VAD hosting the process executable (0x1000000) with path <Non-File Backed Region>

1928 lsass.exe Unexpected protection (PAGE_EXECUTE_READWRITE) for DLL in the PEB's load order list (0x1000000) with path C:\WINDOWS\system32\lsass.exe

© 2025 Volexity Inc.

hollowprocesses vs Cobalt Strike

$ vol3 -f data.lime windows.malware.hollowprocesses

PID Process Notes

4636 svchost.exe Unexpected protection (**PAGE_EXECUTE_READWRITE**) for VAD hosting the process executable (0x880000) with path <Non-File Backed Region>

4636 svchost.exe Unexpected protection (**PAGE_EXECUTE_READWRITE**) for DLL in the PEB's load order list (0x880000) with path C:\Windows\SysWOW64\svchost.exe

© 2025 Volexity Inc.

windows.malware.hollowprocess vs Non-RWX Hollowing

$ python3 vol.py -f data.lime windows.malware.hollow

100 MTF.exe Unexpected protection (**PAGE_READWRITE**) for DLL in the PEB's load order list (0x610000) with path C:\Users\L490\AppData\Local\FXSEXT32\MTF.exe

796 MTF.exe Unexpected protection **(**PAGE_READWRITE) for DLL in the PEB's load order list (0x5c0000) with path C:\Users\L490\AppData\Local\FXSEXT32\MTF.exe

8924 AppVTerminator Unexpected protection (PAGE_READWRITE) for DLL in the PEB's load order list (0xa20000) with path C:\Users\L490\AppData\Local\KBDTH0\AppVTerminator.exe

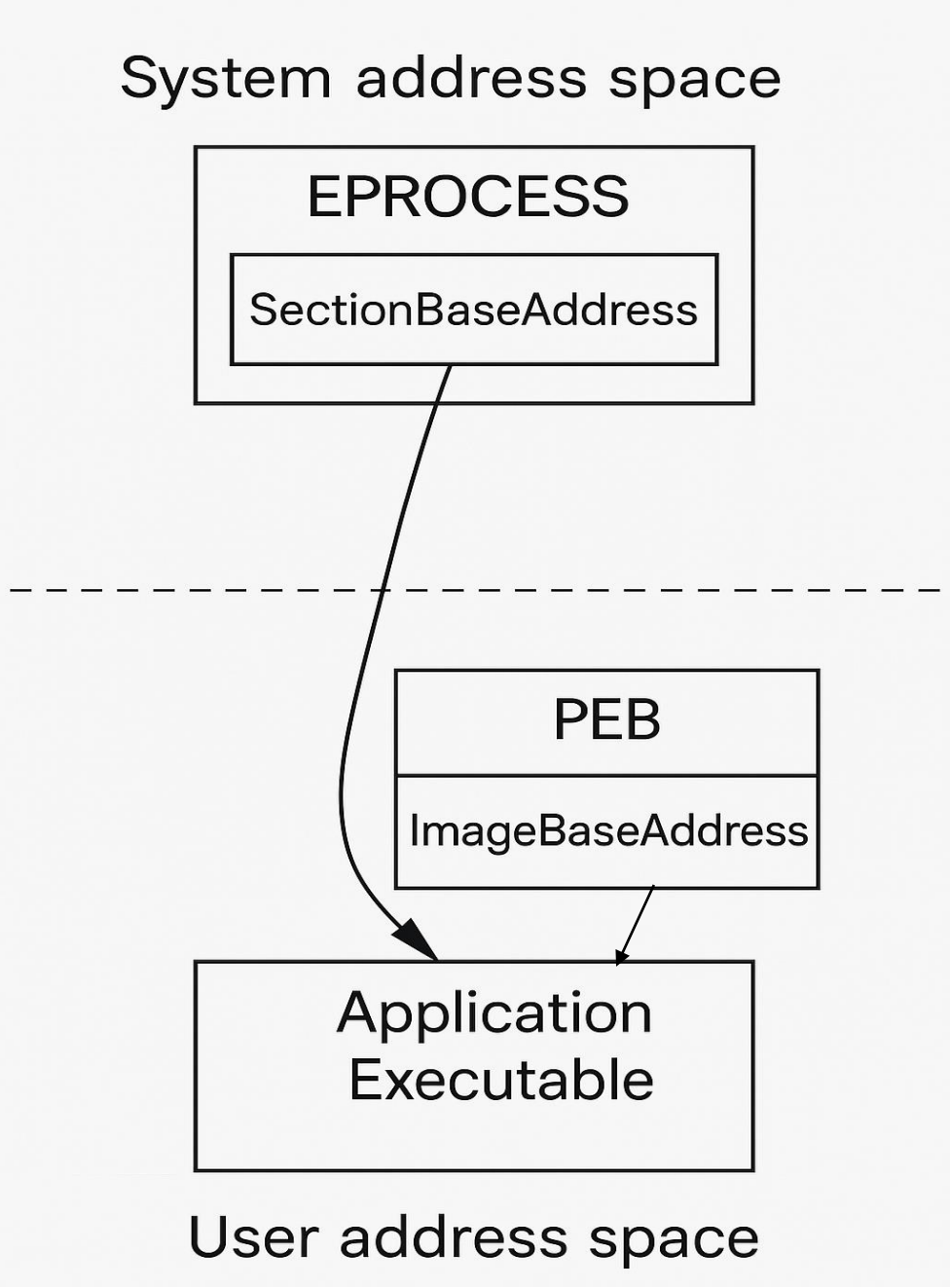
3308 PBDTUQ.exe Unexpected protection (PAGE_READWRITE) for DLL in the PEB's load order list (0x550000) with path C:\Users\L490\AppData\Local\GameBarPresenceWriter\PBDTUQ.exe

6356 msvcr120_clr04 Unexpected protection (PAGE_READWRITE) for DLL in the PEB's load order list (0x580000) with path C:\Users\L490\AppData\Local\nlmgp\msvcr120_clr0400.exe

6236 autoconv.exe Unexpected protection (PAGE_READWRITE) for DLL in the PEB's load order list (0x4b0000) with path C:\WINDOWS\SysWOW64\mbsmsapi\autoconv.exe

SectionBaseAddress vs ImageBaseAddress

- The SectionBaseAddress is set during process creation to the load address of the main application executable
- ImageBaseAddress inside the PEB is later set to the same address
- Userland malware often changes ImageBaseAddress but cannot access SectionBaseAddress



hollowprocess vs Base Address Tampering

$ python3 vol.py -f data.lime windows.malware.hollow

612 manage-bde.exe The ImageBaseAddress reported from the PEB (**0x7ff63f2d0000**) does not match the process SectionBaseAddress (**0x7ff6db320000**)

612 manage-bde.exe There is no VAD starting at the base address of the process executable (**0x7ff6db320000**)

Sample infected with: https://github.com/AzAgarampur/PsForge

Summary So Far…

- If an executable is at the beginning of a VAD (memory region), and it is not PAGE_EXECUTE_WRITECOPY OR does not map a file, then it will be detected

- If malware tampers with the DLL data structures, it will be detected (windows.ldrmodules)

- Bypasses?
  - Overwrite PE header?
  - Map injected malware as PAGE_EXECUTE_WRITECOPY?

Manually Map PAGE_EXECUTE_WRITECOPY?

```
LPVOID WINAPI VirtualAlloc(
  _In_opt_  LPVOID lpAddress,
  _In_      SIZE_T dwSize,
  _In_      DWORD flAllocationType,
  _In_      DWORD flProtect
);

LPVOID WINAPI VirtualAllocEx(
  _In_      HANDLE hProcess,
  _In_opt_  LPVOID lpAddress,
  _In_      SIZE_T dwSize,
  _In_      DWORD flAllocationType,
  _In_      DWORD flProtect
);
```

```
#define PAGE_NOACCESS          0x01
#define PAGE_READONLY          0x02
#define PAGE_READWRITE         0x04
#define PAGE_WRITECOPY         0x08
#define PAGE_EXECUTE           0x10
#define PAGE_EXECUTE_READ      0x20
#define PAGE_EXECUTE_READWRITE 0x40
#define PAGE_EXECUTE_WRITECOPY 0x80
#define PAGE_GUARD            0x100
#define PAGE_NOCACHE          0x200
#define PAGE_WRITECOMBINE     0x400
```

PAGE_EXECUTE_WRITECOPY
0x80

Enables execute, read-only, or copy-on-write access to a mapped view of a file mapping object. An attempt to write to a committed copy-on-write page results in a private copy of the page being made for the process. The private page is marked as **PAGE_EXECUTE_READWRITE**, and the change is written to the new page.

This flag is not supported by the VirtualAlloc or VirtualAllocEx functions.

Windows Vista, Windows Server 2003 and Windows XP: This attribute is not supported by the CreateFileMapping function until Windows Vista with SP1 and Windows Server 2008.

Manually Map PAGE_EXECUTE_WRITECOPY?

- NtCreateSection + MapViewOfSection can be used to manually place a PAGE_EXECUTE_WRITECOPY VAD into a target process
- Detections:
  - No file association/mapping
  - Not in the DLL lists
  - The code needs to run, likely by pointing a thread to it

Code: https://github.com/Offensive-Panda/ProcessInjectionTechniques/tree/main/NtCreateSection_MapViewOfSection

Overwritten PE headers

- It is somewhat common for malware to overwrite its PE header after loading
  - The header is never referenced again after load, so this is safe
- Detecting VADs mapping executables with bad permissions relies on spotting the PE header at the VAD base
- Solution: Threads tell us where code is executing!



- https://www.fortinet.com/blog/threat-research/deep-dive-into-a-dumped-malware-without-a-pe-header

Listing Threads

```
$ python3 vol.py -f data.lime -r pretty --filters PID,836 windows.threads
```

	PID	TID	StartAddress	StartAddressPath	Win32StartAddress	Win32StartAddressPath	CreateTime
*	836	840	0xbf0574ed5518	-	0x7ff7d5554a90	\Windows\System32\lsass.exe	2022-11-10 21:03:12
*	836	852	0x7ffcf185e3e0	…\ntdll.dll	0x7ff7d55521f0	\Windows\System32\lsass.exe	2022-11-10 21:03:12
*	836	872	0x7ffcf185e3e0	…\ntdll.dll	0x7ffcee865240	\Windows\System32\lsasrv.dll	2022-11-10 21:03:12
*	836	876	0x998d6ba7dac0	-	0x7ffcf17eb4a0	\Windows\System32\ntdll.dll	2022-11-10 21:03:12
*	836	880	0xbf05746884d8	-	0x7ffcf17eb4a0	\Windows\System32\ntdll.dll	2022-11-10 21:03:12
*	836	884	0xbf057465a4d8	-	0x7ffcf17eb4a0	\Windows\System32\ntdll.dll	2022-11-10 21:03:13
*	836	4296	0x7ffcf185e3e0	…\ntdll.dll	0x7ffcef1fe910	\Windows\System32\ucrtbase.dll	2022-11-10 21:05:04
*	836	2544	0xbf05747114d8	-	0x7ffcf17eb4a0	\Windows\System32\ntdll.dll	2022-11-10 21:53:12
*	836	5624	0x998d7043b440	-	0x7ffcf17eb4a0	\Windows\System32\ntdll.dll	2022-11-10 22:03:08
*	836	4216	0x7ffcf185e3e0	…\ntdll.dll	0x7ffcf17eb4a0	\Windows\System32\ntdll.dll	2022-11-10 22:04:44
*	836	1320	0x7ffcf185e3e0	…\ntdll.dll	0x7ffcf17eb4a0	\Windows\System32\ntdll.dll	2022-11-10 22:05:24

Signs of a Suspicious Thread

- One or both start addresses point to a memory region (VAD):
  - Not backed by a file
  - Not PAGE_EXEC_WRITECOPY
  - To a file path ending in .exe that is not the main application executable

- Is suspended
  - Malware starts a process in suspended mode and never resumes the main thread

windows.malware.suspicious_threads vs Various Malware

chrome.exe 5720 2616 Win32Start 0xc65410 <Non-File Backed Region> VAD at base address (0xbc0000) hosting this thread has **an unexpected starting protection PAGE_NOACCESS**

rundll32.exe 4288 4352 Win32Start 0x20fb25e14bc <Non-File Backed Region> VAD at base address (0x20fb25e0000) hosting this thread has an **unexpected starting protection PAGE_EXECUTE_READWRITE**

notepad.exe 6276 1480 Win32Start 0x249c2610000 <Non-File Backed Region> VAD at base address (0x249c2610000) hosting this thread has an **unexpected starting protection PAGE_READWRITE**

south.exe 5400 5468 Win32Start 0x22493430000 <Non-File Backed Region> This thread started execution in the VAD starting at base address (0x22493430000), **which is not backed by a file**

dllhost.exe 4024 4068 Win32Start 0x7ffc5495afb0 <Non-File Backed Region> This thread started execution in the VAD starting at base address (0x7ffc54920000), **which is not backed by a file**

Detecting Dirty Vanity and its Suspended Thread

- Detecting the cloned process due to its suspended thread

```
$ python3 vol.py  -r pretty -f Dirty-Vanity.lime windows.suspended_threads
Volatility 3 Framework 2.7.0
  |       Process |  PID |  TID |    StartAddress | Win32StartAddress
* |   FakeExe.exe | 6752 | 1964 | 0x7ffc13722680 |    0x7ffc137a6390
```

- Parent (PID 7472) and cloned child (PID 6752)

```
**** 7472 8152 FakeExe.exe    "C:\FakeExe.exe"
***** 6752   7472 FakeExe.exe "C:\FakeExe.exe"
****** 7188   6752 cmd.exe /k msg * Hello from Dirty Vanity
```

Dirty Vanity: https://i.blackhat.com/EU-22/Thursday-Briefings/EU-22-Nissan-DirtyVanity.pdf

Extracting the containing regions

- We can extract with vaddump (not a PE file) or dlldump (a PE file)

```
$ vol3 -f FILE
       -o <output directory>
       vadinfo
      --pid PID
      --address=ADDRESS
      --dump

$ vol3 -f FILE pedump
      --pid PID
      --base=ADDRESS
```

For a non-PE or smeared PE

For a PE

Note: For malfind, just add --dump

© 2025 Volexity Inc.

strings analysis

- Windows users (GUI)
  - https://mh-nexus.de/en/hxd/
- Mac users (GUI)
  - https://apps.apple.com/us/app/hex-fiend/id1342896380?mt=12
- Linux users
  - strings command

© 2025 Volexity Inc.

Linux (GNU) strings

- strings –a ‘path to sample’ > strings.txt
- strings –a –e l ‘path to sample’ >> strings.txt

- Two passes are required to get ascii (default) and Unicode (-e l)
- Be sure to have >> on the second run or you will overwrite the results of the first

© 2025 Volexity Inc.



Exercise #2

- Detecting Memory-Only Code and DLLs

© 2025 Volexity Inc.



Exercise #3

- Process Hollowing Investigation

© 2025 Volexity Inc.

The End

- Thank you for taking the class!
- Join the Volatility Slack Server:
  - https://www.volatilityfoundation.org/slack
- Stay in touch:
  - acase@volexity.com / @attrc
  - lpace9@lsu.edu
  - ddonze1@lsu.edu
  - @volatility