Preparations

- Get the material
  - Install vol.py and rip.pl
    - https://github.com/volatilityfoundation/volatility
    - https://github.com/who1s/install_regripper
  - Material.7z → Unpack it
  - Finalboss.7z → Save it to your disk
- Import OVA to your VMWare or VBox
- Power on the machine
  - Click “I copied it” if asked anything

```
$ git clone
https://github.com/volatilityfoundation/volatility
$ cd volatility
$ sudo pyth[illegible] setup.py install
```

$ 7z x Material.7z -p pystyyvetaa

Disclaimer

DO NOT run anything you’re able to extract from the memory. There’s a real malware and you can easily infect your own machine, if you don’t know what you’re doing.


Memory
FORENSICS

Authors

whois

@JuhoJauhiainen

/in/jauhiainen/

timiett1

@timiett1

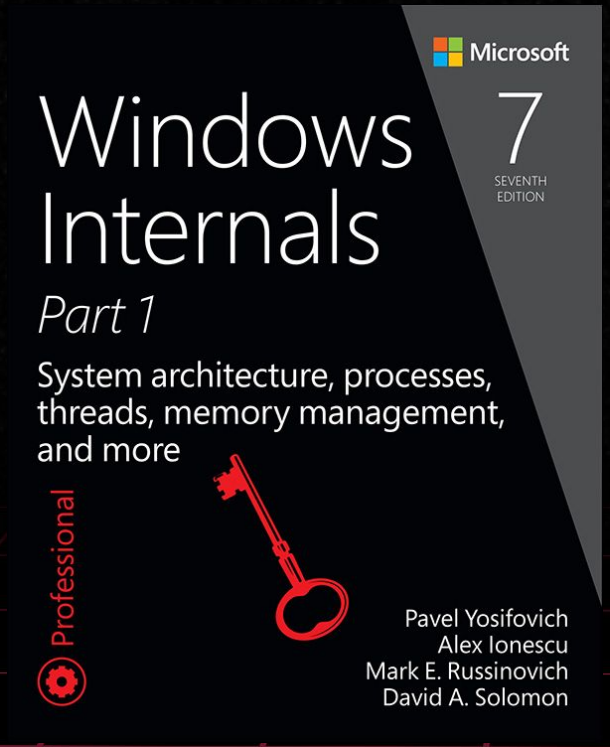
/in/timo-miettinen-b743a896/

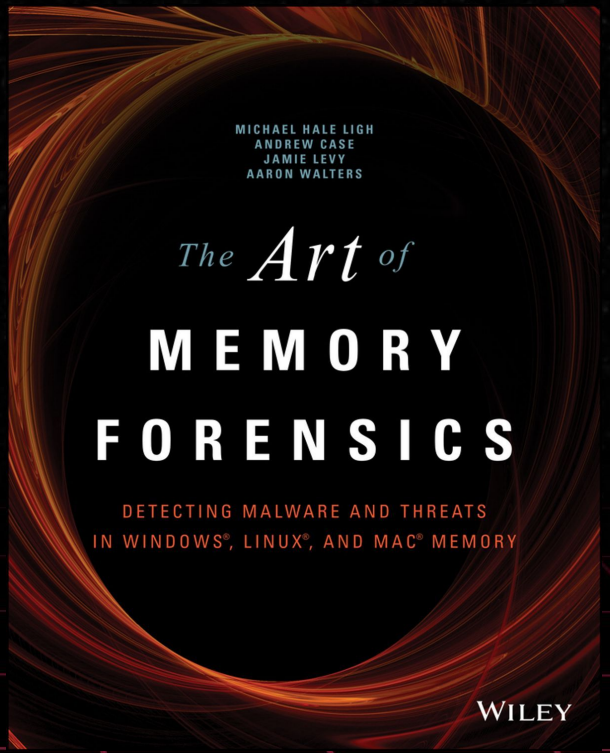
mika

N/A

N/A

Book recommendations





Syllabus

- Why memory forensics?
- Windows internals introduction
- Evidence collection
- How to get started w/ memory forensics
- … [6 labs + CTF]
- Profit


SUCH FORENSICS
WOW
MUCH SKILLZ
DOPE
SO GOOD
imgflip.com

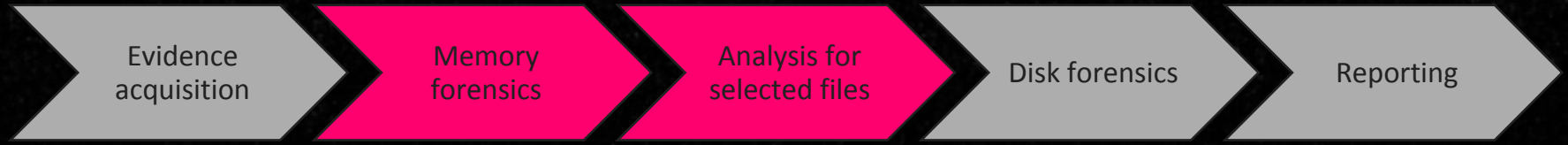
Why u forensicate memory?

What can you find from memory?

- Windows registry and log files
- Opened files
- Secrets
- Configuration files and data
- All running processes (including malware)
- Network artifacts

- Memory is the best place in analyzing malicious software activity
- Some evidence can’t be found elsewhere
  - “Fileless malware”
  - Malware can EITHER success in
    - Hiding
    - Executing

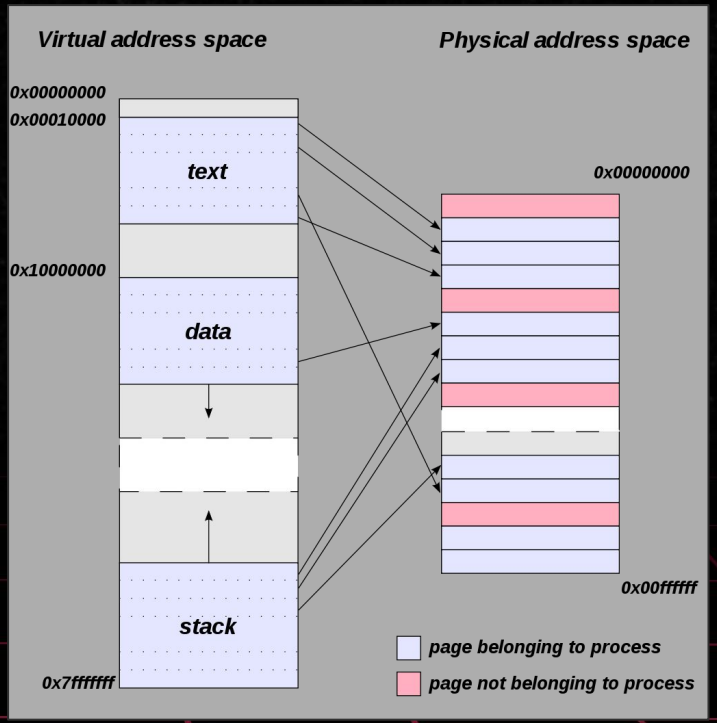
Digital Forensics process



Virtual vs. physical address space

Briefly:

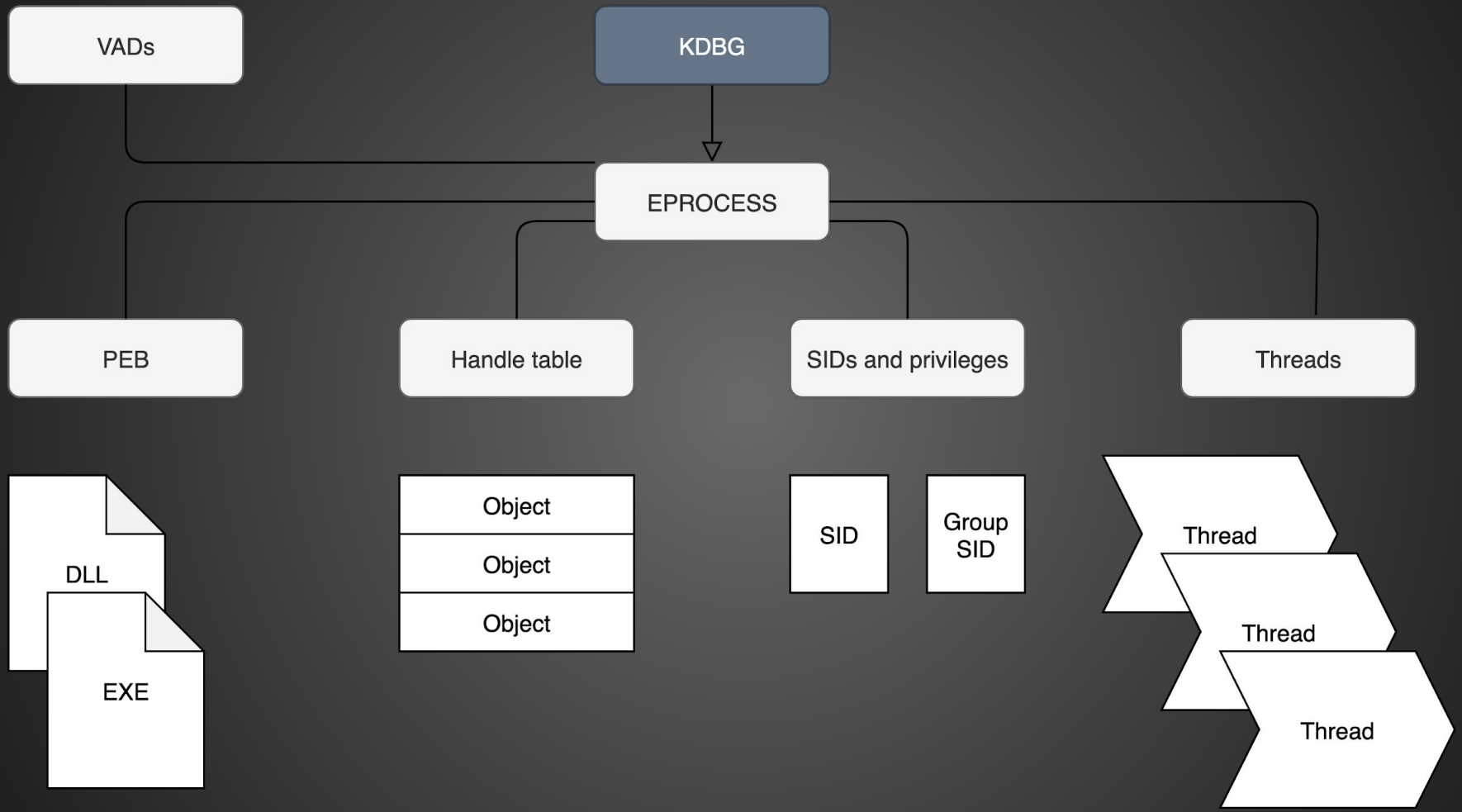
- Virtual address space
  - Every process has their own
  - System and process address spaces reside here
- Physical address space
  - Processes have no clue what's going on here
  - The "real" location in the memory
  - System only

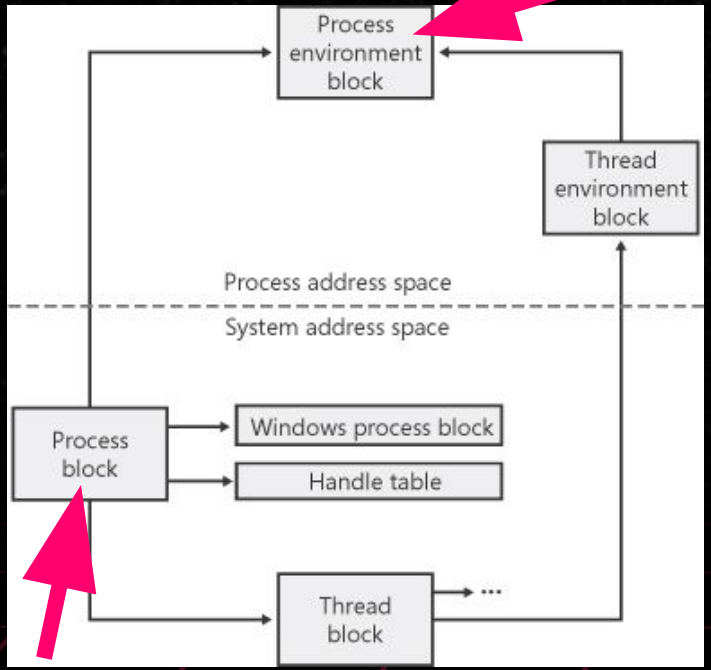


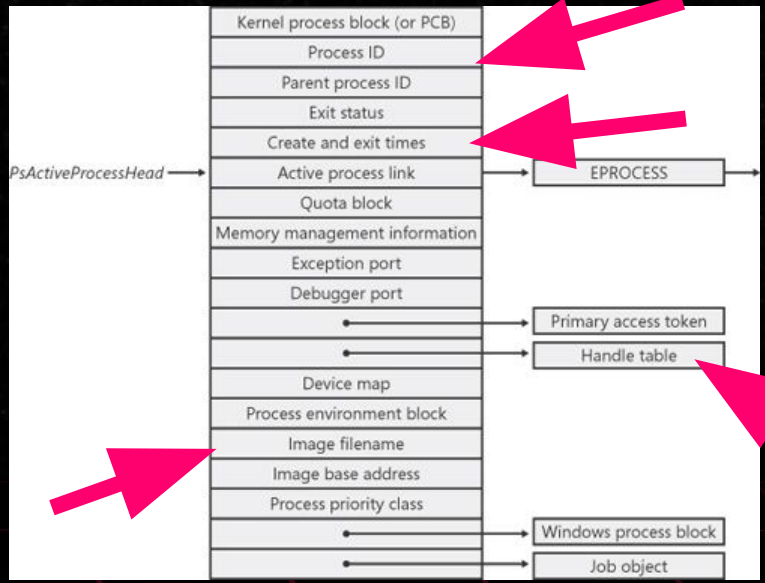
Source: Wikipedia, https://en.wikipedia.org/wiki/Virtual_address_space#/media/File:Virtual_address_space_and_physical_address_space_relationship.svg

Windows process internals

- Kernel Processor Control Region (KPCR) data structure that contains list of KDBG structures
- Kernel Debugger Block (KDBG) is a Windows debugging module
  - Loaded kernel modules, running processes → OS detection
- Each Windows process is represented by an executive process (EPROCESS) block
  - An EPROCESS block contains and points to a number of other related data structures
- The EPROCESS block and most of its related data structures exist in system address space (systemland, kernel)
- Process Environment Block (PEB) being an exception
  - Exists in the process address space (userland), because it contains information that needs to be accessed by user-mode code
- Virtual Address Descriptors (VAD)
  - Memory manager maintains a set of VADs that describes the status of the process's address space


VADs
KDBG
EPROCESS
PEB
Handle table
SIDs and privileges
Threads
DLL
EXE
Object
Object
Object
SID
Group SID
Thread
Thread
Thread





Source: Mark E. Russinovich and David A. Solomon, Windows Internals, 5th Edition,
https://www.microsoftpressstore.com/articles/article.aspx?p=2233328

Analysis overview (SANS)

1. Identify Rogue Processes
2. Analyze Process DLLs and Handles
3. Review Network Artifacts
4. Look for Evidence of Code Injection
5. Check for Signs of a Rootkit
6. Extract Processes, Drivers, and Objects

Collection

- Windows
  - WinPMEM (opensource)
  - FTK Imager (free to download)
  - Comae DumpIt, Magnet RAM Capture, Belkasoft Live RAM Capturer, Redline,
- Linux
  - LiME https://github.com/504ensicslabs/lime
  - Linux Memory Grabber https://github.com/halpomeranz/lmg/
- Virtual machines
  - VMWare snapshot, saved state (vmem, vmss, vmsn)
  - Vbox partial memory image (.sav)
  - Hyper-V memory image (.bin) and save state (.vsv)

strings & grep

Sooooo good together (+ awk, sed, cut) <3

ASCII strings

$ strings -a memory.dmp > strings_a.txt

Unicode strings

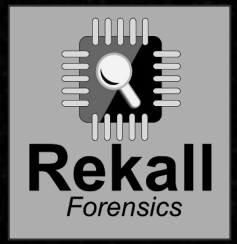
$ strings -e l memory.dmp > strings_u.txt

Grep all the things

$ grep -ihra keyword

$ grep -Ero '(http|https)://[^/"']+'

Toolink



- Open source <3
- Support
  - Win (x86/x64)
    - XP/2003, Vista, 2008/2008R2, 7, 8, 2012, 10, 2016
  - Linux 2.6.11 – 4.2.3 (x86/x64)
    - OpenSuSE, Ubuntu, Debian, CentOS, Fedora, Mandriva
  - Mac OS X 10.5+ (x86/x64)
  - FreeBSD (!!!!)
- https://github.com/volatilityfoundation/volatility
- OS version detection: Kernel Debugger Block

- Open source <3
- Support
  - Win (x86/x64)
    - XP, 7, 8, 10
  - Linux 2.6.24+ (x86/x64)
  - Mac OS X 10.7-10.12.X (x64)
- https://github.com/google/rekall
- OS version detection: Microsoft PDB files

Volatility - basics

$ vol.py –f [image] --profile=[PROFILE] [plugin]

- Esimerkiksi
  - ***$ vol.py –f MEMDUMP.RAW --profile=Win10x86_10586 pslist***

Using local variables:

$ export VOLATILITY_LOCATION=file:///path/to/memory.img

$ export VOLATILITY_PROFILE=Win7SP1x64

$ vol.py pslist

Volatility - basics

$ vol.py --help
$ vol.py --info
$ vol.py [plugin] --help

$ vol.py imageinfo

$ vol.py kdbgscan

1. Identify Rogue Processes

- What to look for?
  - Processes with odd parent processes
    - Example: Word.exe running PowerShell.exe
  - Image of the process is in odd location
    - Example: Usually there's nothing good under C:\Users\Public
  - Process has been executed with odd privileges or accounts
    - Example: SYSTEM is running PowerShell.exe
  - Start time of the process doesn't add up
  - Process name is typosquattered
    - Example: lsaas.exe, cssrs.exe
- Only process relations can tell us a lot
  - What was the attack vector? Did the malware require actions from user?

1. Identify Rogue Processes

- How to investigate with Volatility?
  - ***$ vol.py pslist***
    - Process list
  - ***$ vol.py psscan***
    - Scans the memory image and tries to find EPROCESS blocks
    - Should find some already exited processes
  - ***$ vol.py pstree***
    - Process tree
    - -v stands for verbose and might help in some cases ;---)

LAB #1

LAB #1

1. Detect correct profile for memory dump
2. When was the memory dump taken?
3. Find and list suspicious processes

git clone https://github.com/volatilityfoundation/volatility.git

cd volatility

python2 setup.py install

vol.py …

LAB #1

1. Detect correct profile for memory dump
    a. kdbgscan, imageinfo
    b. Ensure the profile is correct with psscan and netscan
2. Find and list suspicious processes
    a. pslist, psscan, pstree

LAB #1 answers

1. Win10x64_18362
2. 2020-02-13 08:25:53
3. TuqKirkgeTwYpj, iexplore.exe, calc.exe.exe, WmiPrvSE.exe

2. Analyze Process DLLs and Handles

- Let’s look the handle table
- Loaded DLL files
  - Can tell some characteristics of the malware
  - Are they all legit? Any typoed names like User32.dll
  - Where those files have been loaded from?
- What privileges the process has and which account is running the process?

2. Analyze Process DLLs and Handles

- How to investigate with Volatility?
  - ***$ vol.py dlllist***
    - Lists all loaded DLL-files per process
  - ***$ vol.py getsids***
    - Shows SIDs (Security Identifier) for each process
    - https://support.microsoft.com/en-us/help/243330/well-known-security-identifiers-in-windows-operating-systems
  - ***$ vol.py handles***

LAB #2

LAB #2

1. Who is running the processes we identified suspicious?
2. What privileges the user has?

LAB #2 answers

1. Dustin Henderson
2. User belongs to *Administrators* group

3. Review Network Artifacts

- To identify processes with suspicious network connections
- To identify command & control connections, data exfiltration, uncommon management connections etc.
- To collect and identify IOCs
  - IP addresses
  - Known ports

3. Review Network Artifacts

- How to investigate with Volatility?
  - ***$ vol.py netscan***
    - Connections and sockets (Vista onwards)
  - ***$ vol.py connscan***
    - TCP connections (Win 7 only)

LAB #3

+ short break

LAB #3

1. What is IP address of the host we are analyzing?
2. Can you spot suspicious network connections?

LAB #3 answers

1. 192.168.87.176
2. Not w/ netscan

4. Look for Evidence of Code Injection

- The attacker may want to write a malicious code into memory of a process, for example to open reverse shell or to ensure their access
  - Motivation may be to evade antivirus software
- Look for running processes with writable memory (PAGE_EXECUTE_READWRITE)
- Might be some times be simple as searching for magic byte of PE files
  - MZ

4. Look for Evidence of Code Injection

- How to investigate with Volatility?
  - ***$ vol.py malfind***
    - Scans processes and tries to find memory sections with ERW permissions
    - Can be used to dump found memory sections
  - ***$ vol.py ldrmodules***
  - ***$ vol.py hollowfind***

LAB #4

LAB #4

Check the memory if there’s signs of code injection

1. Which process has been injected?
2. Can you figure out what the attacker might have done?

LAB #4 answers

1. Explorer.exe (5008)
2. Most likely migration from process 7964

5. Check for Signs of a Rootkit

- Malware designed to enable access to a system as well as to hide itself from the user and other applications
- There are several ways to hide

5. Check for Signs of a Rootkit

- How to investigate with Volatility?
  - ***$ vol.py psxview***
    - Shows a crossview where it is easy to detect hidden processes
  - ***$ vol.py modscan***
  - ***$ vol.py apihooks***
  - ***$ vol.py ssdt***
  - ***$ vol.py driverirp***
  - ***$ vol.py idt***

6. Extract Processes, Drivers, and Objects

- Basically you dump from the memory processes, files, Windows registry hives etc, that were identified to be suspicious in the previous phases of the investigation
- When analysing a piece of malware, you may need to do for example
  - Signature scanning (YARA, AV signatures)
  - Static analysis
  - Hybrid analysis
  - Reverse engineering

6. Extract Processes, Drivers, and Objects

- How to investigate with Volatility?
  - ***$ vol.py dlldump***
    - Dumps all DLL files of certain process
  - ***$ vol.py moddump***
  - ***$ vol.py procdump***
    - Dumps the process image
  - ***$ vol.py memdump***
    - Dumps memory section as a separate dump
  - ***$ vol.py filescan***
    - Finds all files from the memory
  - ***$ vol.py dumpregistry***
    - Dumps all registry files (NTUSER.DAT, SYSTEM, SAM etc.)
  - ***$ vol.py dumpfiles***
    - Dumps file from given location
  - ***$ vol.py svcscan***
  - ***$ vol.py cmdscan***
  - ***$ vol.py consoles***

LAB #5

LAB #5

1. Extract all registry hives from memory
2. Check if commands run in cmd.exe can be retrieved from memory
3. Can you extract emails from memory? Who send it?
4. Check if you can extract the malicious document or process

LAB #5 answers

1. $ vol.py -f memdump.mem --profile=Win10x64_18362 dumpregistry -D registry
2. Not with consoles or cmdscan plugins → strings
3. strings
4. Process extracted (Win.Trojan.MSShellcode-7), document can’t be extracted

Windows registry

- Registry hives
  - HKEY_CLASSES_ROOT (HKCR)
  - HKEY_CURRENT_USER (HKCU)
  - HKEY_LOCAL_MACHINE (HKLM)
  - HKEY_USERS (HKU)
  - HKEY_CURRENT_CONFIG (HCU)
- Every registry hive has keys and values where computer configuration has been saved to
    - OS settings
    - User settings
    - Software configuration

Where the hives are stored to?

- SYSTEM, SAM, SECURITY, SOFTWARE…
  - *%SystemRoot%\System32\config*
- NTUSER.DAT
  - *C:\Users\%USERNAME%\NTUSER.dat (Vista/7/8/10)*
  - *C:\Documents and settings\%USERNAME%\NTUSER.dat (XP)*
- UsrClass.DAT
  - *%USERPROFILE%\AppData\Local\Microsoft\Windows\Usrclass.dat*

Tools for analysis

RegRipper

- https://github.com/keydet89/RegRipper2.8
- OpenSource <3
- CLI and GUI versions
  - Works on both, *nix and Windows
- Install on Linux: https://raw.githubusercontent.com/who1s/install_regripper/master/install.sh

Registry Explorer/RECmd

- https://ericzimmerman.github.io/#!index.md
- Eric Zimmerman tools
  - Check them out, lot of other cool stuff as well
  - Windows only :(

RegRipper 101

Syntax is easy:

$ rip.pl -r <registryfile> -p <plugin>

How to list all plugins:

$ rip.pl -c -l

Pro-tip: grep <3

$ rip.pl -c -l | grep -i system

$ rip.pl -c -l | grep -i software

$ rip.pl -c -l | grep -i ntuser.dat

LAB #6

LAB #6

1. How many times Dustin has signed on?
2. What is hostname of the system?
3. What has Dustin ran on Windows with the “run” command (win+r) ?

LAB #6 answers

1. $ rip.pl -r registry.0xffffbb02f1b77000.SAM.reg -p samparse
   Answer: 5
2. $ rip.pl -r registry.0xffffbb02ee40d000.SYSTEM.reg -p compname
   Answer: DESKTOP-2446B53
3. $ rip.pl -r registry.0xffffbb02f36f2000.ntuserdat.reg -p runmru
   Answer: inetcpl.cpl, calc and explore

Please give us feedback at

https://forms.gle/haJpZoNcXEibkmta6


IT IS CTF O'CLOCK
imgflip.com

Memory forensics 101 - Capture the Flag

- No need to attack anything, everything you need is in the memory image
- Multiple levels, certain amount of percents required to unlock next level
- No hints available… ¯_(ツ)_/¯
- Only three attempts per question (no bruteforcing)
- UTC or GTFO
- Wrong answer does not give you penalty points
- If you think you got the right answer but the system says it’s wronk, raise your hand and we’ll check if we screwed up [which is btw highly unlikely]

THE FINAL BOSS

Platform:
https://ctf.dfir.fi

Token:
whois can give one if requested :--)

Memory image:
https://files.dfir.fi/mf101/Finalboss.7z

7z password:
hunter2020

Top-3 at Disobey2020:
1. Petteri
2. Ade
3. Kettusec