Malware Detection in Memory Forensics

Ricardo J. Rodríguez

rjrodriguez@unizar.es ※ @RicardoJRdez ※ www.ricardojrodriguez.es

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Spain

October 11, 2021

Université Paris-Saclay, CEA LIST
Paris, France

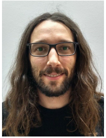
$whoami

- **Associate Professor at the University of Zaragoza**
- **Research lines**:
  - Program binary analysis
  - Digital forensics
  - Offensive security
  - Security and survivability analysis with formal models

Universidad Zaragoza

$whoami

- **Associate Professor at the University of Zaragoza**
- **Research lines**:
  - Program binary analysis
  - Digital forensics
  - Offensive security
  - Security and survivability analysis with formal models
- **Research team** – *we make really good stuff!* ☺
  - `https://reversea.me`
  - `https://twitter.com/reverseame/`
  - `https://t.me/reverseame`



Miguel Martín-Pérez
PhD. student

Daniel Uroz
PhD. student

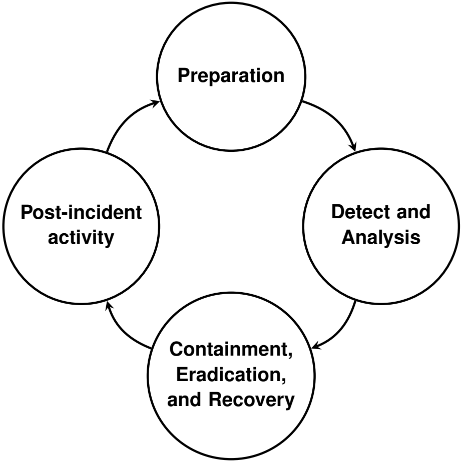
Razvan Raducu
PhD. student

Pedro Fernández
Technician

Universidad Zaragoza

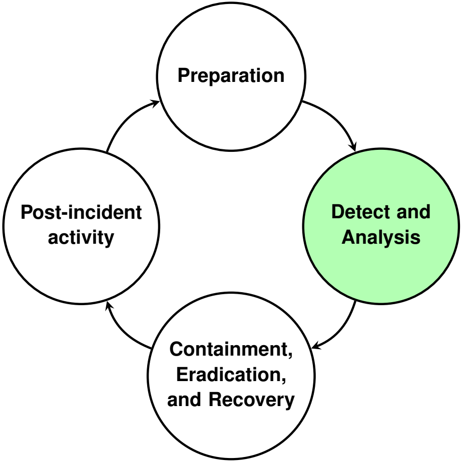
Agenda

Introduction
A little recap...



Incident response as defined by NIST

Introduction
A little recap...

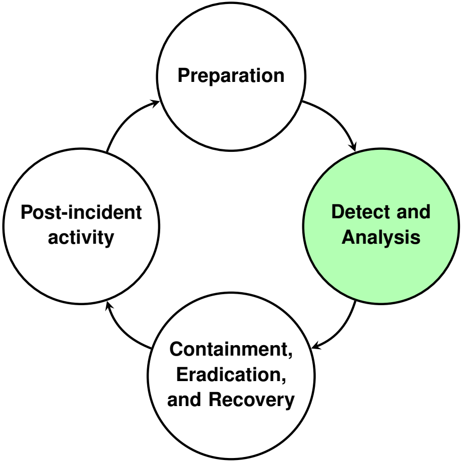


Incident response as defined by NIST

- **Network forensics**
- **Computer forensics**
  - Disk + **memory**

Introduction

A little recap...



Incident response as defined by NIST

- **Network forensics**
- **Computer forensics**
  - Disk + **memory**

Disk vs. memory

- Sometimes, **access to physical drives is difficult to achieve**
- **Current limits of storage capacity vs. memory capacity**
  - Terabytes versus gigabytes
  - **Facilitates initial triage**
- Some data only resides in memory

Memory dump

- **Full of data** to analyze
- **Each item that can be analyzed is called *memory* artifact**
  - Retrieved via appropriate internal structures of the OS or using a pattern-like search
- Snapshot of running processes, logged in users, open files, or open network connections – **everything that was running at the time of acquisition**
- May also contain **recently freed system resources**
  - Normally, memory is not zeroed when freed
- Volatility: **de facto standard** tool for analyzing memory dumps
  - Version 2 vs. version 3 ⇒ Python2 vs. Python3

Introduction

A little more of recap...

Malicious software (malware) analysis

- Determine *what the heck* the malware does as harmful activities
- **Static analysis**
  - Executable files are analyzed without being executed
- **Dynamic analysis**
  - Executable files are analyzed when run

Introduction

The Windows memory subsystem

- **Maps a process virtual address space into physical memory**
- **Manages memory paging**: memory pages are...
  - Paged to disk when the demanding memory of running threads exceeds the available physical memory; and
  - Returned to physical memory when needed

Introduction

The Windows memory subsystem

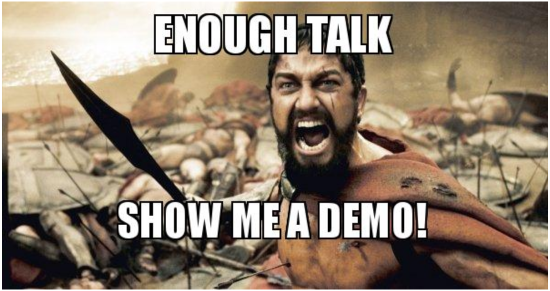
- **Maps a process virtual address space into physical memory**
- **Manages memory paging**: memory pages are...
  - Paged to disk when the demanding memory of running threads exceeds the available physical memory; and
  - Returned to physical memory when needed

Memory page

- **Contiguous fixed-length block of virtual memory**
- Small (4 KiB) and large pages (2 MiB [x86 & x64] to 4 MiB [ARM])
- **Different states**: free, reserved, and committed

Universidad Zaragoza

Introduction



Talk guided by a demo

- Windows 7 x86 machine
- Alina malware (slightly modified for local connection) + system files

Agenda

Universidad Zaragoza

Issue #1: Incompleteness of images

The content of an image is incomplete
(relative to its image file)[1]

Everything happens for a reason...

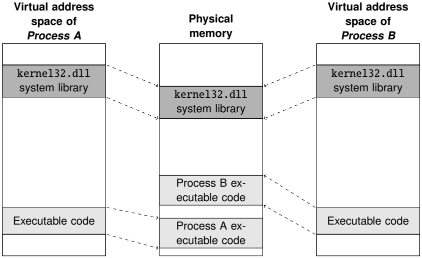
- **Page swapping**
  - The OS stores unused memory pages in a secondary source until those pages are needed again
  - Allows us to use more memory than is actually available in RAM
- **Demand paging** (or *lazy page loading*)
  - The OS does not bring data from files on disk to memory until it is absolutely necessary
  - Optimization issue

(remember to show it with the demo)

[1] Following Windows terminology, an *image file* means a program file that resides on disk, while an *image* means the in-memory representation of an image file. Similarly, an image as well as a process are internally represented by a module

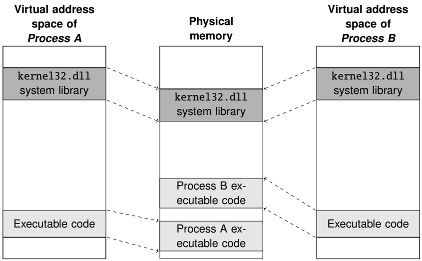
Current Issues and our Contributions

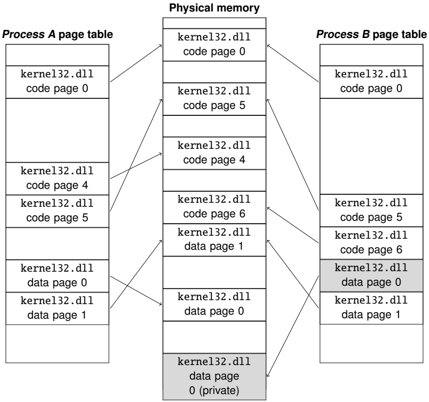
Issue #1: Incompleteness of images



Current Issues and our Contributions

Issue #1: Incompleteness of images





Current Issues and our Contributions

Issue #1: Incompleteness of images

Evaluation of memory paging in Windows 10 [MR-ICDF2C-21]

- **Paging issues in user-space modules** on a Windows 10 64-bit system (build 19041) with 4GiB and 8GiB RAM memory
- **Different memory workloads**: 25%, 50%, 75%, 100%, 125%, and 150%
  - We developed a naif tool that allocates memory and writes a random byte every 4KiB

Universidad Zaragoza

Current Issues and our Contributions

Issue #1: Incompleteness of images

Evaluation of memory paging in Windows 10 [MR-ICDF2C-21]

- **Paging issues in user-space modules** on a Windows 10 64-bit system (build 19041) with 4GiB and 8GiB RAM memory
- **Different memory workloads**: 25%, 50%, 75%, 100%, 125%, and 150%
  - We developed a naif tool that allocates memory and writes a random byte every 4KiB
- System memory acquired at various runtimes for each memory workload
  - *First observation moment*: every 15 seconds for the first minute, every minute for 4 more minutes, while allocating memory
  - *Second observation moment*: same pattern, after stopping the memory allocator tool

Universidad Zaragoza

Current Issues and our Contributions

Issue #1: Incompleteness of images

Evaluation of memory paging in Windows 10 [MR-ICDF2C-21]

- **Paging issues in user-space modules** on a Windows 10 64-bit system (build 19041) with 4GiB and 8GiB RAM memory
- **Different memory workloads**: 25%, 50%, 75%, 100%, 125%, and 150%
  - We developed a naif tool that allocates memory and writes a random byte every 4KiB
- System memory acquired at various runtimes for each memory workload
  - *First observation moment*: every 15 seconds for the first minute, every minute for 4 more minutes, while allocating memory
  - *Second observation moment*: same pattern, after stopping the memory allocator tool
- **Side product of our research**: `residentmem`
  - Volatility2 plugin, GNU/GPLv3. `https://github.com/reverseame/residentmem`
  - Extracts the number of resident pages (that is, in memory) of each image and each process within a memory dump
  - Provides forensic analysts with information on the amount of binary data that cannot be analyzed correctly

Universidad Zaragoza

Current Issues and our Contributions

Issue #1 – discussion of results

On executable modules

- **Almost 80% of the executable module pages are resident** in memory
- With 100% and 125%, in 0.5 minutes:
  - **Most modules are expelled**
  - The number of resident pages for retrievable modules is drastically reduced
- Modules progressively come back to memory, after memory exhaustion
  - **Ratio of resident pages for retrievable modules** $\leq 25\%$
  - Significant increases in 0.5 minutes and in 3 minutes are observed

Universidad Zaragoza

Current Issues and our Contributions

Issue #1 – discussion of results

On executable modules

- **Almost 80% of the executable module pages are resident** in memory
- With 100% and 125%, in 0.5 minutes:
  - **Most modules are expelled**
  - The number of resident pages for retrievable modules is drastically reduced
- Modules progressively come back to memory, after memory exhaustion
  - **Ratio of resident pages for retrievable modules** $\leq$ 25%
  - Significant increases in 0.5 minutes and in 3 minutes are observed

On shared library modules

- Modules only have **20% of their pages resident**, with a maximum percentage observed of 75%
- With 100% and 125%, **in 0.5 minutes the system starts expelling them**
  - Distribution shape is similar in both memory configurations
  - Aggressive expelling of modules is observed in 8GiB
- **Most modules have only less than 5% of their pages resident, after memory exhaustion**

Universidad Zaragoza

Current Issues and our Contributions

Issue #2: Inaccuracy of the content of memory artifacts

The content of an image is inaccurate
(relative to its image file)

Everything happens for a reason...

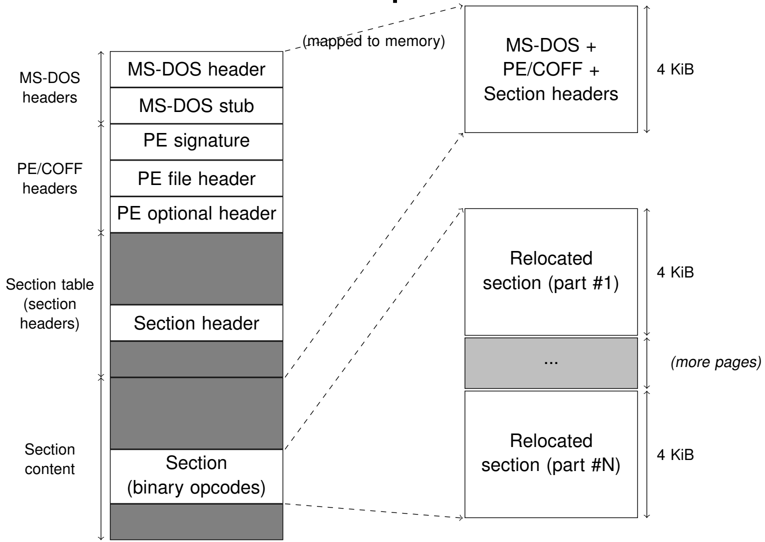
- **Paging effect**
  - Image file mapped into 4KiB aligned memory regions (assuming small pages)
  - As a consequence, a zero padding may appear
- **Relocation**
  - Addresses of external functions resolved (e.g., IAT functions)
  - PE sections removed (e.g., `.reloc` or Authenticode signatures)

(remember to show it with the demo)

Current Issues and Contributions

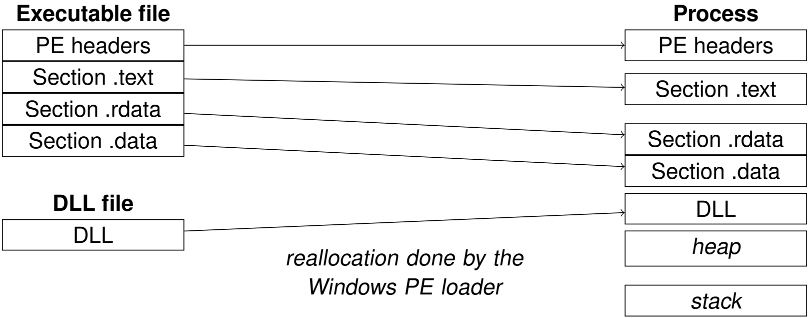
Issue #2: Inaccuracy of the content of memory artifacts

Windows PE file vs. Windows process



Memory Issues and Contributions

Issue #2: Inaccuracy of the content of memory artifacts



Memory Issues and Contributions

Issue #2: Inaccuracy of the content of memory artifacts

Similarity Digest Algorithms (SDAs)

- Identify similarities between different digital artifacts **using an intermediate representation** (i.e., a digest/fingerprint)
- **Bytewise granularity level:** based on byte stream
- **Similarity measure:** typically, $m \in [0, 1]$ ($m \in \mathbb{R}$)
  - In cryptographic hashes we have $m \in \{0, 1\}$ ($m \in \mathbb{Z}$)

Memory Issues and Contributions

Issue #2: Inaccuracy of the content of memory artifacts

Similarity Digest Algorithms (SDAs)

- Identify similarities between different digital artifacts **using an intermediate representation** (i.e., a digest/fingerprint)
- **Bytewise granularity level:** based on byte stream
- **Similarity measure:** typically, $m \in [0, 1]$ ($m \in \mathbb{R}$)
  - In cryptographic hashes we have $m \in \{0, 1\}$ ($m \in \mathbb{Z}$)

Classification of SDAs [MRB-FSIDI-21]

- **Two working stages**:
  - Artifact processing and digest generation phases (feature generation, feature processing, feature selection phase, features deduplication, and digest generation phase)
  - Digest comparison phase
- **Different dimensions and characteristics in each phase**
- Attacks and desirable properties of a SDA to be robust against attacks

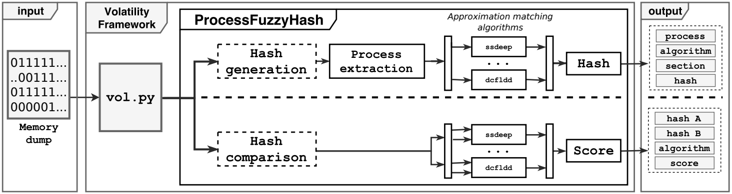
Universidad Zaragoza

Memory Issues and Contributions

Issue #2: Inaccuracy of the content of memory artifacts

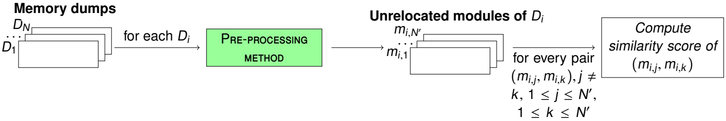
Plugin `ProcessFuzzyHash` [RMA-ISDFS-18]

- Integrates 4 different algorithms for similarity digest calculation
- Bytewise granularity and resemblance (similarity of objects of similar size)
  - `dcfldd`, `ssdeep`, `SDhash`, and `TLSH`
- Allows an (easy) extension to support other algorithms
- Included in the official Volatility2 Framework (under GNU/GPLv3 license)



Memory Issues and Contributions

Issue #2: Inaccuracy of the content of memory artifacts



Pre-processing methods [MRB-COSE-21]

- New plugin: `Similarity Unrelocated Module`
- Volatility2 plugin, GNU/GPLv3. `https://github.com/reverseame/similarity-unrelocated-module`
- **Unrelocates modules** from a given memory dump. Two algorithms:
  - Guided de-relocation (based on `.reloc` sections)
  - Linear sweep de-relocation (decompiles binary code in sliding windows and undo the relocation of instructions with memory addresses)
- **Evaluation of the accuracy of the similarity score in modules**
  - It improves when using any of the pre-processing methods
  - Smart arbitrary byte modifications can drastically affect it, for some of these algorithms (e.g., ssdeep)

Memory Issues and Contributions

Issue #2: Inaccuracy of the content of memory artifacts

Yet another problem related to inaccuracy...

- **Page smearing**
  - Memory inconsistency due to acquired page tables referencing physical pages whose contents changed during the acquisition process
  - **Commonly found on systems with +8GB of RAM or under heavy load**
  - Of course, **only occurs in acquisitions done in live systems**

Solutions (*we are not dealing with this at this time*)

- Freeze memory
- Cause a crash dump
- Check the temporal consistency of acquired data: temporal forensics!

Universidad Zaragoza

Memory Issues and Contributions

Issue #2: Inaccuracy of the content of memory artifacts

Introducing Temporal forensics

- Idea from by Pagani et al.[2]
  - *"we argue that memory forensics should also consider the time in which each piece of data was acquired. This new temporal dimension provides a preliminary way to assess the reliability of a given result and opens the door to new research directions that can minimize the effect of the acquisition time or detect inconsistencies"*
- **Volatility is modified to accurately record time data in a memory dump**
  - Publicly available at `https://github.com/pagabuc/atomicity_tops`

Output example (extracted from [PFB19])

```
$ ./vol.py -f dump.raw --profile=... --pagetime pslist
<original pslist output>
Accessed physical pages: 171
Acquisition time window: 72s
[XX-------------XxX---xXXX--xX-xX---Xxx-xx-X-XxxX-XXX]
```

[2] [PFB19] Pagani, F.; Fedorov, O. & Balzarotti, D. *Introducing the Temporal Dimension to Memory Forensics*. ACM Trans. Priv. Secur., ACM, 2019, 22 , 9:1-9:21

Current Issues and our Contributions

Issue #3: Initial triage for malware detection

Detection of persistence points is difficult

As a consequence of previous issues...

- **Windows Registry contains volatile hives**
- Furthermore, **not all registry keys are in memory**[3]
  - Affected by demand paging and page swapping
  - Some on-disk hives are mapped to memory during Windows start-up, but not all content is in memory

[3]Dolan-Gavitt, B. *Forensic analysis of the Windows registry in memory*. Digital Investigation, 2008, 5, S26-S32

Current Issues and our Contributions

Issue #3: Initial triage for malware detection

Detection of suspicious Auto-Start Extensibility Points [UR-DIIN-19]

- Volatility2 plugin `winesap`, GNU/AGPLv3.
  `https://github.com/reverseame/winesap`
- Flags suspicious activity based on the Windows registry value:
  - `REG_BINARY` or `REG_NONE`, when contains a PE header
  - `REG_SZ`, `REG_EXPAND_SZ`, or `REG_LINK`, when contain suspicious paths or well-known shell commands that indirectly run programs (e.g., `rundll32.exe shell32.dll,ShellExecute_RunDLL <filepath>`)

Output example

```
WARNING:
Suspicious path file
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\firefox.exe
Debugger: REG_SZ: C:\Users\me\AppData\Roaming\Yztrpxpt\cmd.exe
------------------------------
WARNING:
Suspicious path file
HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows
AppInit_DLLs: REG_SZ: C:\Users\me\AppData\Roaming\Uxkgoeaoqbf\autoplay.dll
```

Universidad Zaragoza

Current Issues and our Contributions

Issue #3: Initial triage for malware detection – taxonomy of ASEPs

[UR-DIIN-19]

	Characteristics					
Windows Auto-Start Extensibility Points	**Write permissions**	**Execution privileges**	**Tracked down in memory forensics**[†]	**Freshness of system**	**Execution scope**	**Configuration scope**
System persistence mechanisms						
Run keys (HKLM root key)	yes	user	yes	user session	application	system
Run keys (HKCU root key)	no	user	yes	user session	application	user
Startup folder (%ALLUSERSPROFILE%)	yes	user	no	user session	application	system
Startup folder (%APPDATA%)	no	user	no	user session	application	user
Scheduled tasks	yes	any	no	not needed[‡]	application	system
Services	yes	system	yes	not needed[‡]	application	system
Program loader abuse						
Image File Execution Options	yes	user	yes	not needed	application	system
Extension hijacking (HKLM root key)	yes	user	yes	not needed	application	system
Extension hijacking (HKCU root key)	no	user	yes	not needed	application	user
Shortcut manipulation	no	user	no	not needed	application	user
COM hijacking (HKLM root key)	yes	any	yes	not needed	system	system
COM hijacking (HKCU root key)	no	user	yes	not needed	system	user
Shim databases	yes	any	yes	not needed	application	system
Application abuse						
Trojanized system binaries	yes	any	no	not needed	system	system
Office add-ins	yes	user	yes	not needed	application	user
Browser helper objects	yes	user	yes	not needed	application	system
System behavior abuse						
Winlogon	yes	user	yes	user session	application	system
DLL hijacking	yes	any	no	not needed	system	system
AppInit DLLs	yes	any	yes	not needed	system	system
Active setup (HKML root key)	yes	user	yes	user session	application	system
Active setup (HKCU root key)	no	user	yes	user session	application	application

[†] *If the memory is paging to disk, it would be not possible to track down these ASEPs in memory forensics.*

[‡] *Depends on the trigger conditions defined to launch the program.*

Universidad Zaragoza

Current Issues and our Contributions

Issue #3: Initial triage for malware detection

Prioritize modules to analyze considering digital signatures

Universidad Zaragoza

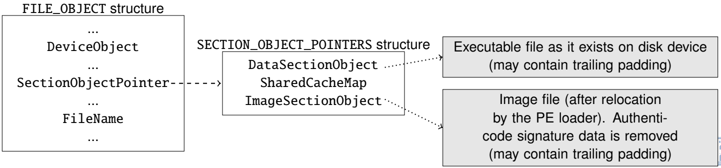
Current Issues and our Contributions

Issue #3: Initial triage for malware detection

Prioritize modules to analyze considering digital signatures

Digital signature verification of retrievable modules [UR-FSIDI-20]

- Volatility2 plugin `sigcheck`, GPLv3. `https://github.com/reverseame/sigcheck`
- **Calculates digital signatures of retrievable modules** (if feasible)
  - In particular, it calculates the Microsoft Authenticode signature
  - Stored in the image file (as a PE section) or in a catalog file
- Relies on `FILE_OBJECT` structures
  - Represent memory mapped files in kernel memory
  - Logical interface between kernel and user-mode processes and the corresponding file data stored in the physical disk



Current Issues and our Contributions

Issue #3: Initial triage for malware detection – digital signatures

Evaluation

- 32-bit and 64-bit Windows 7, plus additional signed software
- Memory acquired in four moments: at startup and after 10, 20, and 30 min of user activity
  - Best number of retrievable file objects with full data at startup
  - None of the retrieved file objects contained the Authenticode signature as full content
  - Some 32-bit DLLs only contained the certificate header
- Limitations:
  - Data incompleteness and data changes caused by PE relocation: affect calculation of Authenticode signature
  - Catalog-signed files
  - Process hollowing is undetected

Universidad Zaragoza

Current Issues and our Contributions

Issue #3: Initial triage for malware detection

Malicious injected code detection can be tricky

Malicious code in memory regions with execute permissions

- Volatility2 plugin `malscan`, AGPLv3. `https://github.com/reverseame/malscan`
- **Integrated with** `clamav-daemon`. Limitation: only works for Linux
- **Two working modes**:
  - **Normal mode**: scans each memory region with W+X permission, each executable module (to detect process hollowing), and private memory regions of type VadS
  - **Full-scan mode**: scans each memory region with +X permission

Malicious injected code detection can be tricky

Malicious code in memory regions with execute permissions

- Volatility2 plugin `malscan`, AGPLv3. `https://github.com/reverseame/malscan`
- **Integrated with** `clamav-daemon`. Limitation: only works for Linux
- **Two working modes**:
  - **Normal mode**: scans each memory region with W+X permission, each executable module (to detect process hollowing), and private memory regions of type VadS
  - **Full-scan mode**: scans each memory region with +X permission
- **Additional detection mechanisms**:
  - When a VAD exists without an associated image file
  - Common function prologues (e.g., `push ebp;mov ebp, esp`)
  - Empty page followed by a function prologue (e.g., a process which has intentionally removed its header)

Agenda

Future Work

In memory forensics

- **Improvement of completeness**
  - Content enrichment of dumped modules
- **Improvement of accuracy**
  - Robust similarity digest algorithm against attacks
  - New pre-processing methods, with better coverage and results
- **Improvement of initial triage for malware detection**
  - Ways to detect code injection techniques
- **Explore the same issues on other desktop and mobile platforms**

Future Work

In other areas of research

- **Offensive security**: `rop3` + ROPLang
- **Vulnerability scan**: race conditions and heap overflow
- **Network protocol RE**
- **Evasive malware**

References

MRB-COSE-21 Martín-Pérez, M. ; Rodríguez, R. J. & Balzarotti, D. *Pre-processing Memory Dumps to Improve Similarity Score of Windows Modules*. Computers & Security, 2021, vol. 101, p. 102119, Elsevier.

MRB-FSIDI-21 Martín-Pérez, M. ; Rodríguez, R. J. & Breitinger, F. *Bringing Order to Approximate Matching: Classification and Attacks on Similarity Digest Algorithms*. Forensic Science International: Digital Investigation, 2021, 36, 301120

MR-ICDF2C-21 Martín-Pérez, M. & Rodríguez, R.J. *Quantifying Paging on Recoverable Data from Windows User-Space Modules*. Proceedings of the 12th EAI International Conference on Digital Forensics & Cyber Crime, Springer. To appear.

RMA-ISDFS-18 Rodríguez, R. J.; Martín-Pérez, M. & Abadía, I. *A Tool to Compute Approximation Matching between Windows Processes*. Proceedings of the 2018 6th International Symposium on Digital Forensic and Security (ISDFS), 2018, 313-318

UR-DIIN-19 Uroz, D. & Rodríguez, R. J. *Characteristics and Detectability of Windows Auto-Start Extensibility Points in Memory Forensics*. Digital Investigation, 2019, 28, S95-S104

UR-FSIDI-20 Uroz, D. & Rodríguez, R. J. *On Challenges in Verifying Trusted Executable Files in Memory Forensics*. Forensic Science International: Digital Investigation, 2020, 32, 300917

Universidad Zaragoza

Malware Detection in Memory Forensics

Ricardo J. Rodríguez

rjrodriguez@unizar.es ※ @RicardoJRdez ※ www.ricardojrodriguez.es

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Spain

October 11, 2021

Université Paris-Saclay, CEA LIST
Paris, France