Unrelocating Windows modules in memory dumps

Miguel Martín-Pérez, **Ricardo J. Rodríguez**, Davide Balzarotti

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

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

December 19, 2020

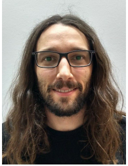
NoConName 2020
The Internet

$whoami

- **Assistant Professor at the University of Zaragoza**
- **Research lines**:
  - Program binary analysis
  - Digital forensics
  - Security and performance system analysis
- Speaker and trainer in various infosec conferences (NcN, HackLU, RootedCON, STIC CCN-CERT, HIP, MalCON, HITB. . .)

$whoami

- **Assistant Professor at the University of Zaragoza**
- **Research lines**:
  - Program binary analysis
  - Digital forensics
  - Security and performance system analysis
- Speaker and trainer in various infosec conferences (NcN, HackLU, RootedCON, STIC CCN-CERT, HIP, MalCON, HITB. . .)
- **Research team** – *we make really good stuff!* ☺
  - `https://reversea.me` / `https://t.me/reverseame`



Miguel Martín-Pérez
PhD. student

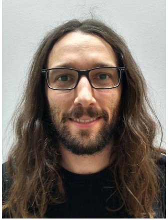
Daniel Uroz
PhD. student

Razvan Raducu
PhD. student

Credits: https://steemit.com/

This work is the result of a research done in collaboration with Miguel Martín-Pérez (PhD. student in University of Zaragoza) and Davide Balzarotti (Professor at EURECOM):

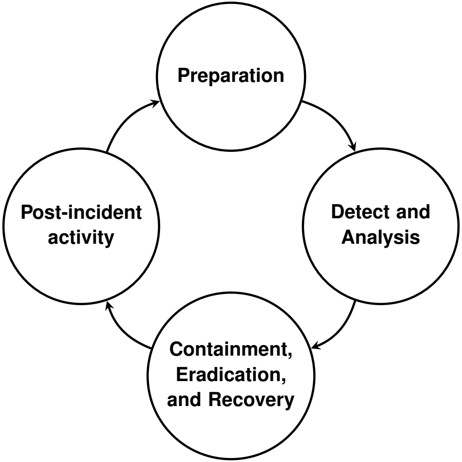
Pre-processing Memory Dumps to Improve Similarity Score of Windows Modules. Miguel Martín-Pérez, Ricardo J. Rodríguez, Davide Balzarotti, *Computers & Security*, vol. 101, pp. 102119, 2021. doi: 10.1016/j.cose.2020.102119 (publicly available here)



Agenda

Agenda

Introduction
A little bit of recap...



Incident response as defined by NIST

Incident response

- **Figure out *what the heck* happened**, while preserving data related to the incident
- **Ask the well-known 6 W's** (what, who, why, how, when, and where)
- Common incident: **presence of malicious software** (malware)
- **Different types of analysis to get hints**:
  - **Computer forensics: disks + memory**
  - Network forensics

Introduction

- **Disk forensics**: analysis of device drives
- **Memory forensics**: analysis of the data contained in the memory of the system under study

Introduction

- **Disk forensics**: analysis of device drives
- **Memory forensics**: analysis of the data contained in the memory of the system under study

Disk vs. memory

- Sometimes, **access to physical device drives are difficult to achieve**
- Think about **current limits of storage capacity versus memory capacity**
  - Terabytes versus gigabytes
  - **Facilitates the initial triage**

Introduction

OK. *Can I use memory forensics for triaging the running processes?*

- **You need to identify processes somehow**
- **Techniques as cryptohashing (used in disk forensics) are unsuitable**
  - *Examples*: MD5, SHA-1, SHA-256...
  - **Avalanche effect**: inputs with slight variations produce radically different outputs

Introduction

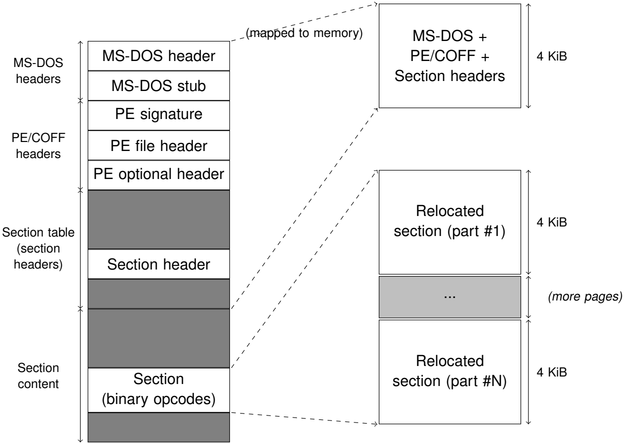
OK. *Can I use memory forensics for triaging the running processes?*

- **You need to identify processes somehow**
- **Techniques as cryptohashing (used in disk forensics) are unsuitable**
  - *Examples*: MD5, SHA-1, SHA-256...
  - **Avalanche effect**: inputs with slight variations produce radically different outputs

Process ≠ executable file (on disk)

Introduction
Process ≠ executable file

- **Relocation process**: ASLR, PIE
- **Memory mapping**
  - Memory page granularity (normally, 4KiB)
- **Page smearing**
- **Demand paging**



Introduction

Similarity digest algorithms

- **Identify similarities between two digital artifacts**
- **Similarity score ranges in** $[0, 100]$, instead of a binary score (yes/no)
- **Useful to find out whether artifacts resemble each other or whether an artifact is contained in another artifact**

Introduction

Similarity digest algorithms

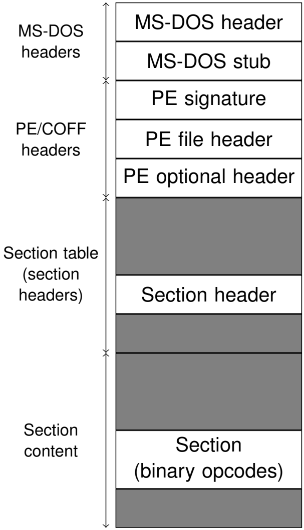
- **Identify similarities between two digital artifacts**
- **Similarity score ranges in** $[0, 100]$, instead of a binary score (yes/no)
- **Useful to find out whether artifacts resemble each other or whether an artifact is contained in another artifact**

Research question:

How do the effects of pagination and relocation affect to similarity score computation?

Agenda

Background
Windows PE file



Background

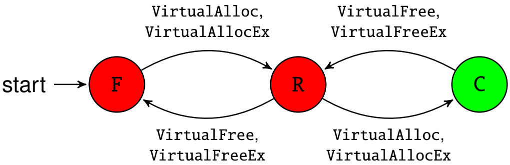
The Windows memory subsystem

- **Virtual size of 32-bit Windows processes**: 2 GiB (prior to Windows 8)
- **Two tasks**:
  - **Maps a process virtual address space into physical memory**
  - **Manages the memory paging**

- **Virtual size of 32-bit Windows processes**: 2 GiB (prior to Windows 8)
- **Two tasks**:
  - **Maps a process virtual address space into physical memory**
  - **Manages the memory paging**

Memory page

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

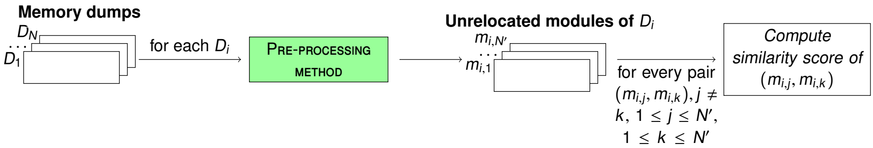


Background
Similarity digest algorithms

- Categories: ***bytewise***, *syntactic*, *semantic*
- **Types of bytewise algorithms**:
  - **Block Based Hashing**
    - Split data into blocks and concatenate the cryptohash of every block
    - *Example*: `dcfldd`
  - **Context Trigger Piecewise Hashing**
    - Parts of the input drive the splitting procedure
    - *Example*: `ssdeep`
  - **Statistically Improbable Features**
    - Most relevant (statistically speaking) blocks are selected
    - *Example*: `sdhash`
  - **Locality Sensitive Hashing**
    - Cluster equivalent elements into buckets, and compare the number of elements in every bucket
    - *Example*: `TLSH`

Agenda

Pre-Processing Methods



Development and evaluation of two pre-processing methods to undo the work performed by the Windows relocation process

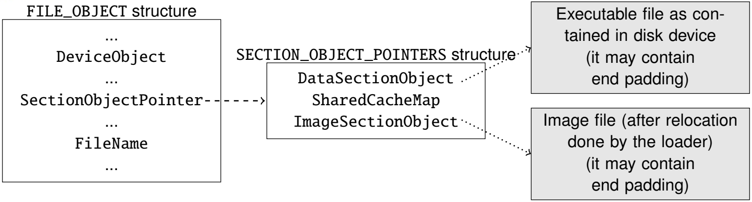
- **Identifies and changes every byte affected by the relocation process, relying on the section `.reloc` of a Windows PE**
    - **Data is divided into blocks. Every block tells the adjustments for a 4KiB memory page**
    - **`IMAGE_BASE_RELOCATION` structure**: contains 2-byte entries indicating what base relocation type is applied (first 4 bits) + the address offset (12 bits)
    - *Further reading*: `http://research32.blogspot.com/2015/01/base-relocation-table.html`

```
typedef struct _IMAGE_BASE_RELOCATION {
    DWORD   VirtualAddress;
    DWORD   SizeOfBlock;
//  WORD    TypeOffset[1];
} IMAGE_BASE_RELOCATION;
```

Problem ahead

this section is stripped off from the image file once it is relocated

Pre-Processing Methods
Guided De-Relocation – File Objects



- **Logical interface between kernel and user-mode processes and the corresponding file data stored in the physical disk**
- Stores a **pointer to a** `SECTION_OBJECT_POINTERS` **structure**
  - Stores file-mapping and cache-related information for a file stream
  - **Three opaque pointers**: `DataSectionObject`, `SharedCacheMap`, and `ImageSectionObject`
  - `DataSectionObject` and `ImageSectionObject` may point to a memory zone where the program binary was mapped either as a data file or as an image file, both containing `.reloc` section

Note: not all processes have a corresponding `File Object` *representation in memory...*

Input: A memory dump $\mathcal{M}$
Output: Set of unrelocated modules $\mathcal{U}$

$\mathcal{U} = \emptyset$
Get list of file objects $\mathcal{F}$ from $\mathcal{M}$
foreach *module m in* $\mathcal{M}$ **do**
    Let $\mathcal{A}$ be the range of virtual memory addresses of *m*
    Walk through every field *p* of the PE header and data directories of *m*. If $p \in \mathcal{A}$, de-relocate *p*
    if $\exists f \in \mathcal{F}$ *such that f corresponds to m and f has* `.reloc` *section* **then**
        Create m' as a copy of *m*
        foreach *block b in* `.reloc` *section of f* **do**
            Get the RVA of the page a_m from the the block *b*
            foreach *entry e in the block b* **do**
                Get the offset *o* from the the entry block *e*
                De-relocate $[a_m + o]$ in m'
            end
        end
        $\mathcal{U} = \mathcal{U} \bigcup \{m'\}$
    end
end

De-relocation process: *the two-less significant bytes of an address are left unmodified, while zeroing the others* (we assume that the relocation always takes place with 64KiB alignment, as ASLR does)

Pre-Processing Methods
Linear Sweep De-Relocation

Input: A memory dump $\mathcal{M}$
Output: Set of unrelocated modules $\mathcal{U}$

```
U = ∅
foreach module m in M do
    Identify empty 4KB-length memory pages, tagging every byte as visited
    Let A be the range of virtual memory addresses of m
    /* Process the structured information */
    Walk through every field f of the PE header and data directories of m, tagging as
      visited bytes. In addition, if f is a virtual memory address, de-relocate f
    /* Process the unstructured information */
    Retrieve the memory space S ⊂ A of the code section of m
    /* Tag lookup tables */
    Identify lookup tables in S and de-relocate the entries of lookup tables that target
      to A
    if m is a 32-bit image file then
        /* Tag strings */
        Identify UNICODE and ASCII strings in S, as well as padding bytes, tagging
          as visited bytes
        /* Tag lookup tables */
        Identify lookup tables in S, tagging as visited bytes, and de-relocate the
          entries of lookup tables that target to A
        /* Tag byte patterns */
        Identify common byte patterns in S, and if subsequent bytes to a pattern p
          conform a memory address a_m and a_m ∈ A, tag p and the subsequent bytes
          as visited bytes and de-relocate a_m
        /* Process the rest of bytes in S */
        foreach byte b ∈ S such that b is not tagged as visited do
            Get the sequences of valid assembly instructions, considering as first
              byte of each sequence b_i, 0 ≤ i ≤ 14, b_0 = b
            Select the longest sequence (in bytes) of valid assembly instructions I
            Identify each assembly instruction in I, and if the instruction contains a
              memory operand which targets to A, de-relocate the operand
            Tag all bytes of the sequence of instructions I as visited bytes
        end
    end
    U = U ∪ {m}
end
```

- **Works in all bytes** in two phases:
  - *Structured* information
  - *Unstructured* information
- **The longest sequence of valid assembly instructions is chosen, considering a maximum of 14-byte length instructions**
  - Slices of 15 bytes
  - **We rely on the Capstone engine**

Pre-Processing Methods
Linear Sweep De-Relocation – example

(consider this slice begins at `0x1000`*, for simplicity)*

```
FC              CLD
FEFF            ???
FFFF            ???
E8 39000000     CALL 0x1043
8B45 08         MOV EAX,DWORD PTR SS:[EBP+0x8]
E8 A487FFFF     CALL KernelBa.752917F0
C2 0C00         RETN 0xC
90              NOP
FE              ???
FFFF            ???
FF00            INC DWORD PTR DS:[EAX]
0000            ADD BYTE PTR DS:[EAX],AL
00CC            ADD AH,CL
FFFF            ???
```

`0x1000: cld`

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
slice	FC	FE	FF	FF	FF	E8	39	00	00	00	8B	45	08	E8	A4
length	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Pre-Processing Methods
Linear Sweep De-Relocation – example

(consider this slice begins at `0x1000`*, for simplicity)*

```
FC                CLD
FEFF              ???
FFFF              ???
E8 39000000       CALL 0x1043
8B45 08           MOV EAX,DWORD PTR SS:[EBP+0x8]
E8 A487FFFF       CALL KernelBa.752917F0
C2 0C00           RETN 0xC
90                NOP
FE                ???
FFFF              ???
FF00              INC DWORD PTR DS:[EAX]
0000              ADD BYTE PTR DS:[EAX],AL
00CC              ADD AH,CL
FFFF              ???
```

`0x1001: ????`

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
slice	FC	FE	FF	FF	FF	E8	39	00	00	00	8B	45	08	E8	A4
length	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0

Pre-Processing Methods
Linear Sweep De-Relocation – example

(consider this slice begins at `0x1000`*, for simplicity)*

```
FC              CLD
FEFF            ???
FFFF            ???
E8 39000000     CALL 0x1043
8B45 08         MOV EAX,DWORD PTR SS:[EBP+0x8]
E8 A487FFFF     CALL KernelBa.752917F0
C2 0C00         RETN 0xC
90              NOP
FE              ???
FFFF            ???
FF00            INC DWORD PTR DS:[EAX]
0000            ADD BYTE PTR DS:[EAX],AL
00CC            ADD AH,CL
FFFF            ???
```

`0x1004: ????`

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
slice	FC	FE	FF	FF	FF	E8	39	00	00	00	8B	45	08	E8	A4
length	1	-1	-1	-1	-1	0	0	0	0	0	0	0	0	0	0

Pre-Processing Methods
Linear Sweep De-Relocation – example

(consider this slice begins at `0x1000`*, for simplicity)*

```
FC              CLD
FEFF            ???
FFFF            ???
E8 39000000     CALL 0x1043
8B45 08         MOV EAX,DWORD PTR SS:[EBP+0x8]
E8 A487FFFF     CALL KernelBa.752917F0
C2 0C00         RETN 0xC
90              NOP
FE              ???
FFFF            ???
FF00            INC DWORD PTR DS:[EAX]
0000            ADD BYTE PTR DS:[EAX],AL
00CC            ADD AH,CL
FFFF            ???
```

```
0x1005: call    0x1043
0x100a: mov     eax, dword ptr [rbp + 8]
0x100d: call    0xffffffffffff97b1
0x1012: ret     0xc
0x1015: nop
```

- Instructions out of the window are not considered

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
slice	FC	FE	FF	FF	FF	E8	39	00	00	00	8B	45	08	E8	A4
length	1	-1	-1	-1	-1	17	0	0	0	0	-1	0	0	-1	0

Pre-Processing Methods
Linear Sweep De-Relocation – example

(consider this slice begins at `0x1000`*, for simplicity)*

```
FC              CLD
FEFF            ???
FFFF            ???
E8 39000000     CALL 0x1043
8B45 08         MOV EAX,DWORD PTR SS:[EBP+0x8]
E8 A487FFFF     CALL KernelBa.752917F0
C2 0C00         RETN 0xC
90              NOP
FE              ???
FFFF            ???
FF00            INC DWORD PTR DS:[EAX]
0000            ADD BYTE PTR DS:[EAX],AL
00CC            ADD AH,CL
FFFF            ???
```

```
0x1006: cmp     dword ptr [rax], eax
0x1008: add     byte ptr [rax], al
0x100a: mov     eax, dword ptr [rbp + 8]
0x100d: call    0xffffffffffff97b0
0x1012: ret     0xc
0x1015: nop
```

- As the instruction starting at `0x100a` was already considered in a previous sequence, **the processing of this sequence is skipped**
- −1 **value is set in the length vector**, instead of the current sequence length

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
slice	FC	FE	FF	FF	FF	E8	39	00	00	00	8B	45	08	E8	A4
length	1	-1	-1	-1	-1	17	-1	0	-1	0	-1	0	0	-1	0

Pre-Processing Methods

Linear Sweep De-Relocation – example

(consider this slice begins at `0x1000`*, for simplicity)*

```
FC                     CLD
FEFF                   ???
FFFF                   ???
E8 39000000            CALL 0x1043
8B45 08                MOV EAX,DWORD PTR SS:[EBP+0x8]
E8 A487FFFF            CALL KernelBa.752917F0
C2 0C00                RETN 0xC
90                     NOP
FE                     ???
FFFF                   ???
FF00                   INC DWORD PTR DS:[EAX]
0000                   ADD BYTE PTR DS:[EAX],AL
00CC                   ADD AH,CL
FFFF                   ???
```

```
0x1007: add     byte ptr [rax], al
0x1009: add     byte ptr [rbx - 0x5b17f7bb], cl
0x100f: xchg    edi, edi
0x1011: inc     edx
0x1013: or      al, 0
0x1015: nop
```

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
slice	FC	FE	FF	FF	FF	E8	39	00	00	00	8B	45	08	E8	A4
length	1	-1	-1	-1	-1	17	-1	-1	-1	-1	-1	0	0	-1	0

Pre-Processing Methods

Linear Sweep De-Relocation – example

(consider this slice begins at `0x1000`*, for simplicity)*

```
FC                  CLD
FEFF                ???
FFFF                ???
E8 39000000         CALL 0x1043
8B45 08             MOV EAX,DWORD PTR SS:[EBP+0x8]
E8 A487FFFF         CALL KernelBa.752917F0
C2 0C00             RETN 0xC
90                  NOP
FE                  ???
FFFF                ???
FF00                INC DWORD PTR DS:[EAX]
0000                ADD BYTE PTR DS:[EAX],AL
00CC                ADD AH,CL
FFFF                ???
```

```
0x100b: inc ebp
0x100c: or      r8b, r13b
0x100e: movsb   byte ptr [rdi], byte ptr [rsi]
0x100f: xchg    edi, edi
0x1011: inc     edx
0x1013: or      al, 0
0x1015: nop
```

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
slice	FC	FE	FF	FF	FF	E8	39	00	00	00	8B	45	08	E8	A4
length	1	-1	-1	-1	-1	17	-1	-1	-1	-1	-1	-1	-1	-1	-1

(consider this slice begins at `0x1000`, for simplicity)

```
FC              CLD
FEFF            ???
FFFF            ???
E8 39000000     CALL 0x1043
8B45 08         MOV EAX,DWORD PTR SS:[EBP+0x8]
E8 A487FFFF     CALL KernelBa.752917F0
C2 0C00         RETN 0xC
90              NOP
FE              ???
FFFF            ???
FF00            INC DWORD PTR DS:[EAX]
0000            ADD BYTE PTR DS:[EAX],AL
00CC            ADD AH,CL
FFFF            ???
```

Longest sequence found:

```
0x1005: call   0x1043
0x100a: mov    eax, dword ptr [rbp + 8]
0x100d: call   0xffffffffffff97b1
0x1012: ret    0xc
0x1015: nop
```

- **Bytes in the slice are marked as visited**
- **Bytes of the sequence starting at byte `E8` are also marked as visited**
  - If any instruction has a memory operand targeting the virtual memory range of the process, its address is de-relocated
- **Next slice starts at the byte `FE`** (in `0x1016`)

address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
slice	FC	FE	FF	FF	FF	E8	39	00	00	00	8B	45	08	E8	A4
length	1	-1	-1	-1	-1	17	-1	-1	-1	-1	-1	-1	-1	-1	-1

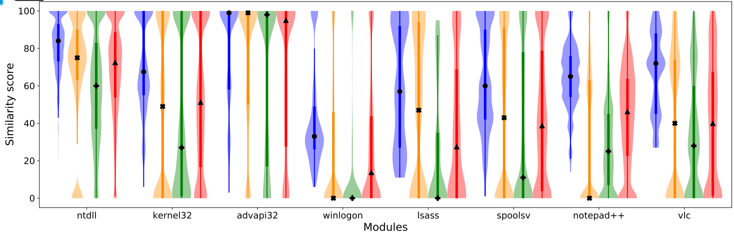
Agenda

Evaluation and Tool Support
Description of experiments

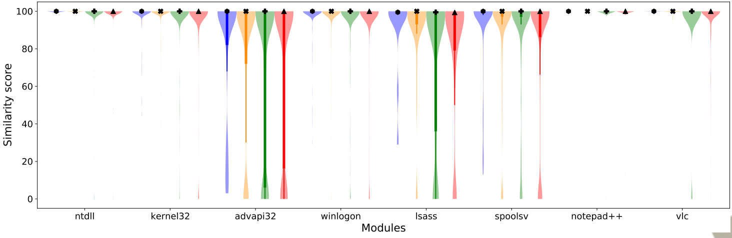
- **Windows 7 6.1.7601, Windows 8.1 6.3.9600, and Windows 10 10.0.14393**
- **x86 and x86-64 versions**, on top of VirtualBox hypervisor
- **Ten memory acquisitions in ten minutes after a fresh boot**
- **Three sets of modules for comparison**:
  - System libraries: `ntdll.dll`, `kernel32.dll`, and `advapi32.dll`
  - System programs: `winlogon.exe`, `lsass.exe`, and `spoolsv.exe`
  - Workstation programs: `Notepad++ 7.5.8`, `vlc 3.0.4`
- **Three scenarios:**
  - No pre-processing (RAW scenario)
  - Application of the Guided De-relocation pre-processing method (GUIDED DE-RELOCATION scenario)
  - Application of the Linear Sweep De-relocation method (LINEAR SWEEP DE-RELOCATION scenario)

Evaluation and Tool Support

Related comparison – RAW scenario



(a) x86

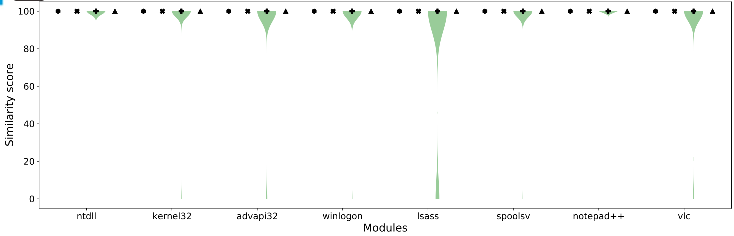


(b) x86-64

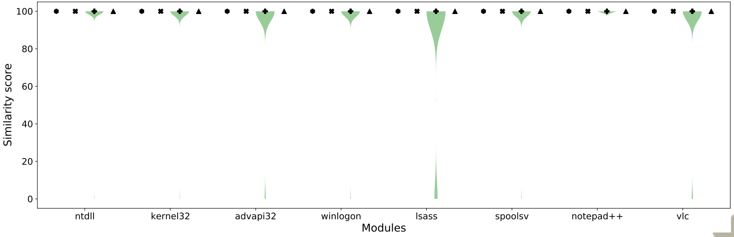
⬢ dcfldd ✖ ssdeep ✚ sdhash ▲ TLSH

Evaluation and Tool Support

Related comparison – GUIDED DE-RELOCATION scenario



(a) x86

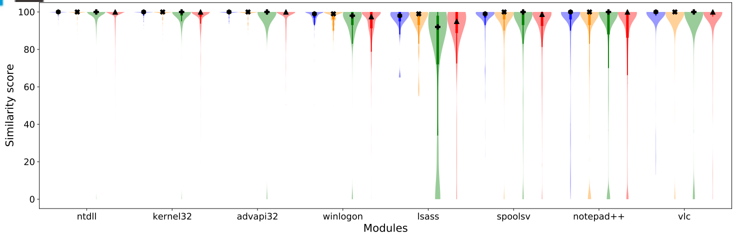


(b) x86-64

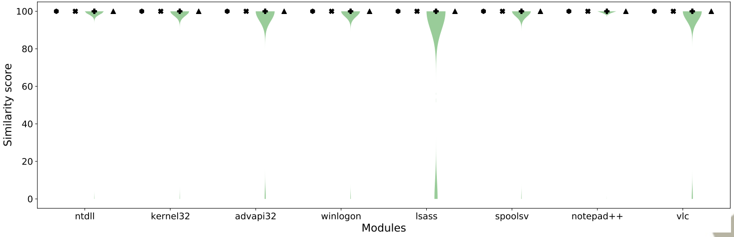
⬢ dcfldd ✖ ssdeep ✚ sdhash ▲TLSH

Evaluation and Tool Support

Related comparison – Linear Sweep De-relocation scenario



(a) x86

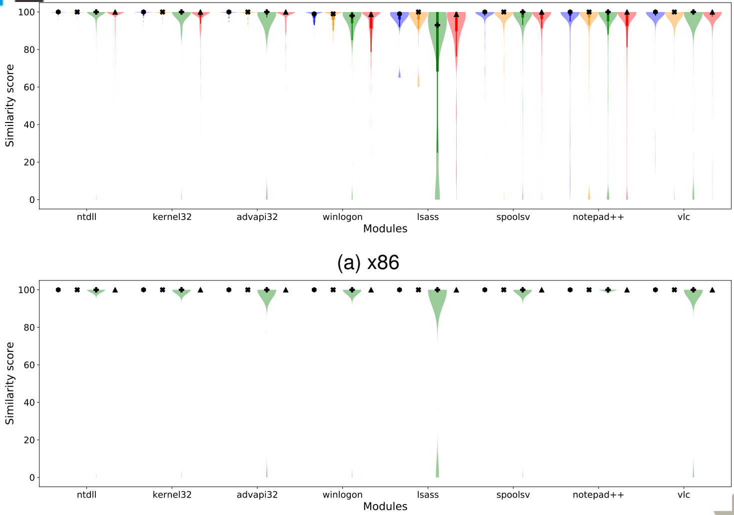


(b) x86-64

⬢ dcfldd ✖ ssdeep ✚ sdhash ▲ TLSH

Evaluation and Tool Support

Related comparison with cross pre-processing methods

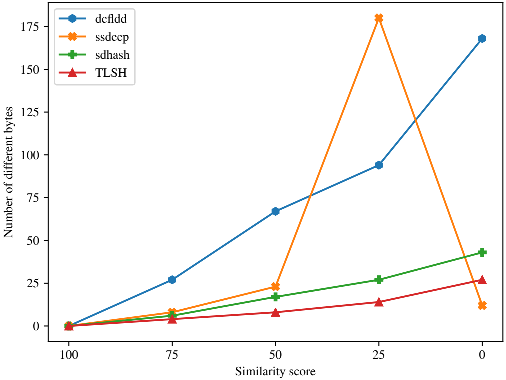


(a) x86

(b) x86-64

Evaluation and Tool Support

Effect of byte changes on the similarity score



- **`dcfldd` needs more byte changes**
- Byte modifications in `ssdeep` were affecting an arbitrary number of features, provoking **no seven consecutive features in common between inputs were found** (and thus the similarity score becomes zero)

`Similarity Unrelocated Module` (`SUM`)

- **Volatility plugin released under GNU/AGPL version 3**
- **Supports both methods**. By default, it applies none
- It also supports:
  - **To use more than one similarity digest algorithm at once**
  - **To select only specific sections of the modules for similarity comparison**
  - **To select process by PID or process and libraries by name**
- **Publicly available in GitHub**

`https://github.com/reverseame/similarity-unrelocated-module`

Agenda

Related Work

- **Performance and robustness of similarity digest algorithms against random byte modification attacks are largely studied** in the literature
- **Some others proposed pre-processing methods aiming to exclude common features and thus enhance the performance** of `sdhash` and `mrsh-v2`
  - Our methods are independent of the particular digest algorithm
  - Our methods work in the input of the algorithm, rather than in internal working details of the algorithm
- **Other works, as (White et al., 2013), proposed a normalization process of relocated bytes by setting constant values**
  - Their approach recreates the Windows PE loader
  - Our methods do not need binary files and are less conservative, as we only normalize the bytes considering 64-KiB memory alignment

Agenda

Conclusions and Future Work

- **Two pre-processing methods to undo the Windows relocation** process
  - Guided De-relocation, **which relies on** `File Objects`
  - Linear Sweep De-relocation, **which performs a linear sweep** of the binary code to identify instructions that contain (absolute) memory addresses as operands
- **Assessment in different scenarios with different similarity digest algorithms** (in particular, `dcfldd`, `ssdeep`, `sdhash`, and `TLSH`)
  - **Similarity score are improved when pre-processing methods are applied**
- **Evaluation of the sensitivity to byte modifications**
  - ***Intelligent* arbitrary byte modifications can dramatically affect the similarity score**

Conclusions and Future Work

- **Two pre-processing methods to undo the Windows relocation** process
  - GUIDED DE-RELOCATION, **which relies on** `File Objects`
  - LINEAR SWEEP DE-RELOCATION, **which performs a linear sweep** of the binary code to identify instructions that contain (absolute) memory addresses as operands
- **Assessment in different scenarios with different similarity digest algorithms** (in particular, `dcfldd`, `ssdeep`, `sdhash`, and `TLSH`)
  - **Similarity score are improved when pre-processing methods are applied**
- **Evaluation of the sensitivity to byte modifications**
  - ***Intelligent* arbitrary byte modifications can dramatically affect the similarity score**

Future work

- **Improve the disassembling process**
- **Extend `SUM` to contemplate also other PE sections**

Unrelocating Windows modules in memory dumps

Miguel Martín-Pérez, **Ricardo J. Rodríguez**, Davide Balzarotti

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

Universidad Zaragoza 1542

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

December 19, 2020

NoConName 2020
The Internet