Memory Smearing: Myth or Reality?

Fabio Pagani

30th September 2019
Memory Forensics - Introduction

- Target Machine
- Memory Acquisition
- Memory Analysis
- Evidence
Memory Forensics - Introduction

- Target Machine
- Memory Acquisition
- Memory Analysis
Memory Forensics - Introduction

Target Machine → Memory Acquisition → Memory Analysis → Evidence
Memory Forensics - Introduction

This talk
Memory Acquisition

Decision tree adapted from The Art of Memory Forensics
Memory Acquisition

Decision tree adapted from The Art of Memory Forensics
ATOMIC
Memory Acquisition - Introduction

ATOMIC

NON-ATOMIC
Memory Analysis

• The “core” of memory forensics.

• Several frameworks: Volatility, Rekall (Google), Mandiant’s Memoryze.

• Examples of information that can be extracted:
  • Processes → list/tree, open files, memory mappings, extract executable and shared libraries
  • Modules → list, code, unloaded modules
  • Networking → connections, sockets, arp table
  • Windows Registry → keys, password hashes
  • System information → clipboard content, screenshot

• Every task is “organized” in a plugin
Memory Analysis

Some pointers can be inconsistent!
Memory Analysis

Problem: Some pointers can be inconsistent!
Memory Analysis

Some pointers can be inconsistent!
Memory Analysis

Some pointers can be inconsistent!
Problem

Some pointers can be inconsistent!
pagefile. Additional issues include, as you pointed out, that while the imaging process is occurring, the kernel memory (and even user-mode memory) is changing...so what you end up with is a smear, for want of a better term.
Vomel et. al — Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition (DFRWS 2009)
In about every fifth memory dump acquired via kernel-level acquisition we were confronted with inconsistent page tables. While almost the whole virtual address space of our payload application RAMMANGLE.EXE could be reconstructed, a few pages were sporadically mismapped to virtual memory of other processes, unused physical memory or kernel memory. The reason for this is yet unknown to us, however, because all tested kernel-level acquisition tools exhibited the same behavior, regardless of the acquisition method (either using MmMapIoSpace(), the Device\PhysicalMemory device or PTE remapping) we do not consider it to be a tool error. However, on the
In about every fifth memory dump acquired via hardware level acquisition we were confronted with page tables. While the state of memory is harder to capture, page smearing

Current issues — page smearing

The following sections describe current approaches to acquisition across all major operating systems, along with the limitations of these approaches. Each section is structured to describe the state of the art, its limitations, and future directions to improve acquisition techniques and procedures. We start with page smearing as it is one of the most pressing issues.

Case and Richard — Memory forensics: The path forward (DFWRS 2017)
Memory Smearing - History

- To have a good memdump we need to freeze the OS

  *But if you do you can have some troubles*

- You can trigger a blue screen (not really cool)

In some cases (10% to 20%), volatility and windbg can’t analyze them. You can't always just make another dump.
The Problem
Types of Inconsistency

First Acquisition Sequence

T0

P1

Struct A
null

P2

P3

P4

Acquisition Time

T1

Struct A
6400

Struct B
AAAAAAA
AAAAAAA
AAAAAAA

Buffer C
AAAAAAA
AAAAAAA
AAAAAAA

T2

Struct A
6400

Struct B
AAAAAAA
AXXXXXA
AAAAAAA

Buffer C
AAAAAAA
AAAAAAA
AAAAAAA

T3

Struct A
8192

Struct B
BBBBBBB
BBBBBBB
BBBBBBB

Buffer C
BBBBBBB
BBBBBBB
BBBBBBB

Second Acquisition Sequence
### Types of Inconsistency

<table>
<thead>
<tr>
<th>Acquisition Time</th>
<th>Struct A</th>
<th>Buffer C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T0</strong></td>
<td>Struct A null</td>
<td></td>
</tr>
<tr>
<td><strong>T1</strong></td>
<td>Struct A</td>
<td>Buffer C AAAAAAAA AAAAAAAA AAAAAAAA</td>
</tr>
</tbody>
</table>

**First Acquisition Sequence**

<table>
<thead>
<tr>
<th>Acquisition Time</th>
<th>Struct A</th>
<th>Buffer C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T0</strong></td>
<td>Struct A null</td>
<td></td>
</tr>
<tr>
<td><strong>T1</strong></td>
<td>Struct A</td>
<td>Buffer C AAAAAAAA AAAAAAAA</td>
</tr>
</tbody>
</table>

**Second Acquisition Sequence**

<table>
<thead>
<tr>
<th>Acquisition Time</th>
<th>Struct A</th>
<th>Buffer C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T0</strong></td>
<td>Struct A null</td>
<td></td>
</tr>
<tr>
<td><strong>T1</strong></td>
<td>Struct A</td>
<td>Buffer C AAAAAAAA AAAAAAAA</td>
</tr>
</tbody>
</table>

**Diagram Notes:**
- P1, P2, P3, P4 represent different processes or components.
- Struct A, Buffer C are shown in different states across acquisition times T0 and T1.
- The diagram illustrates the flow and changes in data structures over time.
Types of Inconsistency

First Acquisition Sequence

Second Acquisition Sequence
Types of Inconsistency

First Acquisition Sequence

T0
- P1: Struct A
- P2: null

T1
- P1: Struct A
- P2: Struct B
- P3: Buffer C
- P4: null

T2
- P1: Struct A
- P2: Struct B
- P3: Buffer C
- P4: null

T3
- P1: Struct A
- P2: Struct B
- P3: Buffer C
- P4: null

Second Acquisition Sequence

- P1: Struct A
- P2: Struct B
- P3: Buffer C
- P4: null
Types of Inconsistency

First Acquisition Sequence

- **P1@T1**: Struct A
  - Buffer C: AAAAAAAA AAAAAAAA AAAAAAAA

- **P2@T0**: Struct B
  - Buffer C: BBBBBBBB BBBBBBBB BBBBBBBB

- **P3@T3**: Struct B
  - Buffer C: BBBBBBBB BBBBBBBB BBBBBBBB

- **P4@T2**: Buffer C
  - Struct A: AAAAAAAA AAAAAAAA AAAAAAAA

Acquisition Times:
- **T0**: null
- **T1**: 6400
- **T2**: BBBBBBBB
- **T3**: BBBBBBBB

First Acquisition Sequence:
- P1
- P2
- P3
- P4

Second Acquisition Sequence:
- P1
- P2
- P3
- P4
Types of Inconsistency

First Acquisition Sequence

P1@T1
- Struct A
- Buffer C

P2@T0
- Struct B
- Buffer C

P3@T3
- Buffer C

P4@T2
- Buffer C

Second Acquisition Sequence

P1@T1
- Struct A
- Buffer C

P2@T2
- Struct B
- Buffer C

P3@T3
- Buffer C

P4@T0
- Buffer C
Types of Inconsistency

First Acquisition Sequence

P1@T1
Struct A

P2@T0
Struct A

P3@T3
Struct A

P4@T2
Struct A

Fragment Inconsistency

Second Acquisition Sequence

P1@T1
Struct A

P2@T2
Struct A

P3@T3
Struct A

P4@T0
Struct A
Types of Inconsistency

First Acquisition Sequence

```
<table>
<thead>
<tr>
<th>Acquisition Time</th>
<th>Struct A</th>
<th>Struct B</th>
<th>Buffer C</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td></td>
<td>Struct B</td>
<td>Buffer C</td>
</tr>
<tr>
<td></td>
<td>6400</td>
<td>AAAAAAA</td>
<td>AAAAAAA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAAA</td>
<td>AAAA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAAA</td>
<td>AAAA</td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td>Struct B</td>
<td>Buffer C</td>
</tr>
<tr>
<td></td>
<td>6400</td>
<td>AAAAAAA</td>
<td>AAAAAAA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAAA</td>
<td>AAAA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AAAA</td>
<td>AAAA</td>
</tr>
<tr>
<td>T3</td>
<td></td>
<td>Struct B</td>
<td>Buffer C</td>
</tr>
<tr>
<td></td>
<td>8192</td>
<td>BBBBBBBB</td>
<td>BBBBBBBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BBBBBBB</td>
<td>BBBBBBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BBBBBBB</td>
<td>BBBBBBB</td>
</tr>
</tbody>
</table>
```

Second Acquisition Sequence

```
<table>
<thead>
<tr>
<th>Acquisition Time</th>
<th>Struct A</th>
<th>Struct B</th>
<th>Buffer C</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td></td>
<td>BBBBBBBB</td>
<td>Buffer C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BBBBBBBB</td>
<td>BBBBBBBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BBBBBBB</td>
<td>BBBBBBB</td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td></td>
<td>BBBBBBBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BBBBBBBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BBBBBBBB</td>
</tr>
<tr>
<td>T3</td>
<td></td>
<td>BBBBBBBB</td>
<td>Buffer C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BBBBBBBB</td>
<td>BBBBBBBB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BBBBBBB</td>
<td>BBBBBBB</td>
</tr>
</tbody>
</table>
```

Pointer Inconsistency
Types of Inconsistency

**First Acquisition Sequence**

- **T0**
  - P1: Struct A
  - P2: null

- **T1**
  - P1: Struct A
  - P2: Struct B
  - Buffer C

- **T2**
  - P1: Struct A
  - P2: Struct B
  - Buffer C

- **T3**
  - P1: Struct A
  - P2: Struct B
  - Buffer C

**Second Acquisition Sequence**

- **T1**
  - P1: Struct A
  - P2: Struct B
  - Buffer C

- **T2**
  - P1: Struct A
  - P2: Struct B
  - Buffer C

- **T3**
  - P1: Struct A
  - P2: Struct B
  - Buffer C

**Value Inconsistency**

- **P1@T1**: Struct A
- **P2@T0**: Buffer C
- **P3@T3**: Buffer C
- **P4@T2**: Buffer C

- **P1@T1**: Struct A
- **P2@T2**: Struct B
- **P3@T3**: Buffer C
- **P4@T0**: Buffer C
Impact Estimation - Fragmentation

![Graph showing impact estimation for various processes]

Key:
- Green circle: stack
- Red triangle: heap
- Blue square: code

Processes listed:
- systemd (1)
- systemd-udevd (275)
- snapd (759)
- NetworkManager (825)
- lightdm (1244)
- upstart (1253)
- VBoxClient (1282)
- ibus-daemon (1569)
- upstart-file-br (1584)
- indicator-bluet (1768)
- colord (1853)
- evolution-calen (1876)
- gnome-software (1932)
- sd_cicer (14254)
- firefox (21963)
- vlc (21964)
- htop (22163)

Y-axis: Processes
X-axis: Seconds
Impact Estimation - Fragmentation
Impact Estimation - Fragmentation

DISPERSED

DISPERSED

Seconds

0 20 40 60 80 100 120 140

stack  heap  code

systemd (1)
systemd-udevd (275)
snapd (759)
NetworkManager (825)
lightsdm (1244)
upstart (1253)
VBoxClient (1282)
ibus-daemon (1569)
upstart-file-br (1584)
indicator-bluet (1768)
colord (1853)
evolution-calen (1876)
evolution-software (1932)
gnome-software (1935)
sd_cicero (14254)
firefox (21963)
vlc (21964)
htop (22153)
WHERE ARE FIREFOX CODE PAGES?!?
Kernel-Space Integrity

Task Structure

- `task_struct`
- `mm`
  - `comm` ('firefox')
- `mm_struct`
- `map_count` 50
- `mm_rb`
- `mmap`

VM Area Structure

- `vm_area_struct`
  - `vm_start` 0x55a6bf7a5
  - `vm_end` 0x55a6bf7a7
  - `vm_prot` RW

- `vma` <- `vma` <- `vma` <- `vma`
Kernel-Space Integrity

- task_struct
- mm_struct
  - mm
  - comm: ‘firefox’
  - map_count
    - 50
  - mm_rb
  - mmap
  - vma

- vm_area_struct
  - vm_start: 0x55a6bf7a5
  - vm_end: 0x55a6bf7a7
  - vm_prot: RW

Volatility Plugin:
- map_count == list_len(mmap)
- map_count == tree_len(mm_rb)

![Diagram of kernel-space integrity](image-url)
Kernel-Space Integrity

Volatility Plugin:

map_count == list_len(mmap)
map_count == tree_len(mm_rb)
## Kernel-Space Integrity

<table>
<thead>
<tr>
<th></th>
<th>Scenario 1 (Firefox)</th>
<th>Scenario 2 (Apache)</th>
<th>Scenario 3 (Malware)</th>
</tr>
</thead>
<tbody>
<tr>
<td>List mismatch</td>
<td>100%</td>
<td>71%</td>
<td>80%</td>
</tr>
<tr>
<td>Tree mismatch</td>
<td>100%</td>
<td>73%</td>
<td>80%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100%</strong></td>
<td><strong>78%</strong></td>
<td><strong>80%</strong></td>
</tr>
</tbody>
</table>

- **Is this actually a problem?**
  - List: Firefox stack and code never present
  - Tree: Firefox stack present 10%, code present 30%

- Key recovery for WannaCry and NotPetya
Kernel-Space Integrity

<table>
<thead>
<tr>
<th></th>
<th>Scenario 1 (Firefox)</th>
<th>Scenario 2 (Apache)</th>
<th>Scenario 3 (Malware)</th>
</tr>
</thead>
<tbody>
<tr>
<td>List mismatch</td>
<td>100%</td>
<td>71%</td>
<td>80%</td>
</tr>
<tr>
<td>Tree mismatch</td>
<td>100%</td>
<td>73%</td>
<td>80%</td>
</tr>
<tr>
<td>Total</td>
<td>100%</td>
<td>78%</td>
<td>80%</td>
</tr>
</tbody>
</table>

Is this actually a problem?
## Kernel-Space Integrity

<table>
<thead>
<tr>
<th></th>
<th>Scenario 1 (Firefox)</th>
<th>Scenario 2 (Apache)</th>
<th>Scenario 3 (Malware)</th>
</tr>
</thead>
<tbody>
<tr>
<td>List mismatch</td>
<td>100%</td>
<td>71%</td>
<td>80%</td>
</tr>
<tr>
<td>Tree mismatch</td>
<td>100%</td>
<td>73%</td>
<td>80%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>100%</td>
<td>78%</td>
<td>80%</td>
</tr>
</tbody>
</table>

**Is this actually a problem?**

- List → Firefox stack and code **never** present
- Tree → Firefox stack present **10%**, code present **30%**
Kernel-Space Integrity

<table>
<thead>
<tr>
<th></th>
<th>Scenario 1 (Firefox)</th>
<th>Scenario 2 (Apache)</th>
<th>Scenario 3 (Malware)</th>
</tr>
</thead>
<tbody>
<tr>
<td>List mismatch</td>
<td>100%</td>
<td>71%</td>
<td>80%</td>
</tr>
<tr>
<td>Tree mismatch</td>
<td>100%</td>
<td>73%</td>
<td>80%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>100%</td>
<td>78%</td>
<td>80%</td>
</tr>
</tbody>
</table>

Is this actually a problem?

- List → Firefox stack and code **never** present
- Tree → Firefox stack present **10%**, code present **30%**
- Key recovery for WannaCry and NotPetya
User-Space Integrity
User-Space Integrity

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
<th>$D_5$</th>
<th>$D_6$</th>
<th>$D_7$</th>
<th>$D_8$</th>
<th>$D_9$</th>
<th>$D_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>6</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>Physical Pages</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Acquisition Time (s)</td>
<td>3.2</td>
<td>30.0</td>
<td>37.8</td>
<td>31.0</td>
<td>0.25</td>
<td>26.0</td>
<td>28.6</td>
<td>1.0</td>
<td>27.6</td>
<td>39.9</td>
</tr>
<tr>
<td>rbp delta (s)</td>
<td>7.7</td>
<td>38.8</td>
<td>49.6</td>
<td>43.7</td>
<td>7.3</td>
<td>43.4</td>
<td>4.3</td>
<td>4.0</td>
<td>15.1</td>
<td>5.64</td>
</tr>
<tr>
<td>Corrupted (registers)</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>Corrupted (frame pointers)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Inconsistent data</td>
<td>N/A</td>
<td>✓</td>
<td>N/A</td>
<td>✓</td>
<td>–</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>✓</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## User-Space Integrity

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
<th>$D_5$</th>
<th>$D_6$</th>
<th>$D_7$</th>
<th>$D_8$</th>
<th>$D_9$</th>
<th>$D_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>6</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>Physical Pages</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Acquisition Time (s)</td>
<td>3.2</td>
<td>30.0</td>
<td>37.8</td>
<td>37.0</td>
<td>0.25</td>
<td>26.0</td>
<td>28.6</td>
<td>1.0</td>
<td>27.6</td>
<td>39.9</td>
</tr>
<tr>
<td>rbp delta (s)</td>
<td>7.7</td>
<td>38.8</td>
<td>49.6</td>
<td>43.7</td>
<td>7.3</td>
<td>43.4</td>
<td>4.3</td>
<td>4.0</td>
<td>15.1</td>
<td>5.64</td>
</tr>
<tr>
<td>Corrupted (registers)</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>Corrupted (frame pointers)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Inconsistent data</td>
<td>N/A</td>
<td>✓</td>
<td>N/A</td>
<td>✓</td>
<td>–</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>✓</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Is this actually a problem?
### User-Space Integrity

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
<th>$D_5$</th>
<th>$D_6$</th>
<th>$D_7$</th>
<th>$D_8$</th>
<th>$D_9$</th>
<th>$D_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>6</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>Physical Pages</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Acquisition Time (s)</td>
<td>3.2</td>
<td>30.0</td>
<td>37.8</td>
<td>37.0</td>
<td>0.25</td>
<td>26.0</td>
<td>28.6</td>
<td>1.0</td>
<td>27.6</td>
<td>39.9</td>
</tr>
<tr>
<td>rbp delta (s)</td>
<td>7.7</td>
<td>38.8</td>
<td>49.6</td>
<td>43.7</td>
<td>7.3</td>
<td>43.4</td>
<td>4.3</td>
<td>4.0</td>
<td>15.1</td>
<td>5.64</td>
</tr>
<tr>
<td>Corrupted (registers)</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Corrupted (frame pointers)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Inconsistent data</td>
<td>N/A</td>
<td>✓</td>
<td>N/A</td>
<td>✓</td>
<td>-</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>✓</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Is this actually a problem?

- Dissecting the user space process heap (DFRWS 2017)
- Building stack traces from memory dump of Windows x64 (DFRWS 2018)
- Chrome Ragamuffin (Volatility plugin for Chrome)
Can we study smear in a more generic way?

Build a graph of kernel structures
Can we study smear in a more generic way?

Build a graph of kernel structures

Define metrics to evaluate analyses
Can we study smear in a more generic way?

Build a graph of kernel structures

Define metrics to evaluate analyses

Study analyses as paths on the graph
The Graph

- 100k Structures (Nodes)
- 840k Pointers (Edges)
Proposed Metrics

• Atomicity
• Stability
• Consistency
**Atomicity**: distance in memory between two connected structures
**Metrics**

**Stability**: how long an edge remains stable in a running machine
- 25 snapshots at [0s, 1s, 5s, ..., 3h]
Consistency: Atomicity + Stability
Evaluation of Current Analyses

<table>
<thead>
<tr>
<th>Volatility Plugin</th>
<th># Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux_arp</td>
<td>13</td>
</tr>
<tr>
<td>linux_check_creds</td>
<td>248</td>
</tr>
<tr>
<td>linux_check_modules</td>
<td>151</td>
</tr>
<tr>
<td>linux_check_tty</td>
<td>13</td>
</tr>
<tr>
<td>linux_find_file</td>
<td>14955</td>
</tr>
<tr>
<td>linux_ifconfig</td>
<td>12</td>
</tr>
<tr>
<td>linux_lsmod</td>
<td>12</td>
</tr>
<tr>
<td>linux_lsof</td>
<td>821</td>
</tr>
<tr>
<td>linux_mount</td>
<td>495</td>
</tr>
<tr>
<td>linux_pidhashtable</td>
<td>469</td>
</tr>
<tr>
<td>linux_proc_maps</td>
<td>4722</td>
</tr>
<tr>
<td>linux_pslist</td>
<td>124</td>
</tr>
</tbody>
</table>
Evaluation of Current Analyses

<table>
<thead>
<tr>
<th>Volatility Plugin</th>
<th># Nodes</th>
<th>Stability (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux_arp</td>
<td>13</td>
<td>12,000</td>
</tr>
<tr>
<td>linux_check_creds</td>
<td>248</td>
<td>2</td>
</tr>
<tr>
<td>linux_check_modules</td>
<td>151</td>
<td>700</td>
</tr>
<tr>
<td>linux_check_tty</td>
<td>13</td>
<td>30</td>
</tr>
<tr>
<td>linux_find_file</td>
<td>14955</td>
<td>0</td>
</tr>
<tr>
<td>linux_ifconfig</td>
<td>12</td>
<td>12,000</td>
</tr>
<tr>
<td>linux_lsmod</td>
<td>12</td>
<td>700</td>
</tr>
<tr>
<td>linux_lsof</td>
<td>821</td>
<td>0</td>
</tr>
<tr>
<td>linux_mount</td>
<td>495</td>
<td>10</td>
</tr>
<tr>
<td>linux_pidhashtable</td>
<td>469</td>
<td>30</td>
</tr>
<tr>
<td>linux_proc_maps</td>
<td>4722</td>
<td>0</td>
</tr>
<tr>
<td>linux_pslist</td>
<td>124</td>
<td>30</td>
</tr>
</tbody>
</table>

Stability: 3 paths **never** changed in over 3 hours
11 paths **changed** in less than 1 minute
## Evaluation of Current Analyses

<table>
<thead>
<tr>
<th>Volatility Plugin</th>
<th># Nodes</th>
<th>Stability (s)</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fast</td>
</tr>
<tr>
<td>linux_arp</td>
<td>13</td>
<td>12,000</td>
<td>✓</td>
</tr>
<tr>
<td>linux_check_creds</td>
<td>248</td>
<td>2</td>
<td>✓</td>
</tr>
<tr>
<td>linux_check_modules</td>
<td>151</td>
<td>700</td>
<td>✓</td>
</tr>
<tr>
<td>linux_check_tty</td>
<td>13</td>
<td>30</td>
<td>✓</td>
</tr>
<tr>
<td>linux_find_file</td>
<td>14955</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>linux_ifconfig</td>
<td>12</td>
<td>12,000</td>
<td>✓</td>
</tr>
<tr>
<td>linux_lsmod</td>
<td>12</td>
<td>700</td>
<td>✓</td>
</tr>
<tr>
<td>linux_lsof</td>
<td>821</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>linux_mount</td>
<td>495</td>
<td>10</td>
<td>✓</td>
</tr>
<tr>
<td>linux_pidhashtable</td>
<td>469</td>
<td>30</td>
<td>✓</td>
</tr>
<tr>
<td>linux_proc_maps</td>
<td>4722</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>linux_pslist</td>
<td>124</td>
<td>30</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Consistency:** 5 inconsistent plugins when fast acquisition
7 inconsistent plugins when slow acquisition
Solutions
• Given a physical page we must be able to tell *when* it was acquired!

• Modified LiME to record timing information

• Overhead:
  • Every 100$\mu$s $\rightarrow$ 0.7%
  • Every page $\rightarrow$ 2.4%
• Transparently add the timing information to Volatility

• Intercept object creation to create a *timeline*:

```
./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]
Locality-Based Acquisition

- Every memory acquisition tool treats pages equally:
  - **Independently** if it is used by the OS
  - **Independently** if it contains forensics data
  - From lowest → highest physical address

- Can we do better?
- Why not acquiring forensics/interconnected data first, and then rest of memory?
Locality-Based Acquisition

Two phases:

1. *Smart* dump:
   - Process and module list
   - For each process: page tables, memory mappings, open files, stack, heap, kernel stack..

2. Traditional acquisition of the remaining pages

Impact

- Negligible overhead in time and memory footprint
- No inconsistency in kernel and user space integrity tests!
Locality-Based Acquisition

Two phases:

1. *Smart* dump:
   - Process and module list
   - For each process: page tables, memory mappings, open files, stack, heap, kernel stack..

2. Traditional acquisition of the remaining pages

**Impact**

- Negligible overhead in time and memory footprint
Locality-Based Acquisition

Two phases:

1. *Smart* dump:
   - Process and module list
   - For each process: page tables, memory mappings, open files, stack, heap, kernel stack..

2. Traditional acquisition of the remaining pages

**Impact**

- Negligible overhead in time and memory footprint
- No inconsistency in kernel and user space integrity tests!
More details on our papers:

• Introducing the Temporal Dimension to Memory Forensics (ACM TOPS 2019)
• Back to the Whiteboard: a Principled Approach for the Assessment and Design of Memory Forensic Techniques (USENIX 2019)

All the code and artifacts developed are open-source!

• https://github.com/pagabuc/atomicity_tops
• https://github.com/pagabuc/kernographer
Questions?

Twitter: @pagabuc
Email: pagani@eurecom.fr