Back to the Whiteboard: a Principled Approach for the Assessment and Design of Memory Forensic Techniques

Fabio Pagani and Davide Balzarotti

Usenix Security ’19
Memory Forensics - Introduction

Infected Machine → Memory Dump → Analysis
Memory Forensics - Introduction

Infected Machine → Memory Dump → Analysis → Evidence
Extract the following information:

- List processes, kernel modules
- Open files, memory mappings, sockets..
- System information: routing table, kernel logs..
Extract the following information:

- List processes, kernel modules
- Open files, memory mappings, sockets..
- System information: routing table, kernel logs..

... and much more: Volatility (the most used memory forensic framework) has more than 100 plugins for Windows!
Memory Forensics - Listing Processes

task_struct

init_task

task_struct

task_struct

- tasks
- next
- prev
Memory Forensics - Listing Processes

```
[task_struct]
<table>
<thead>
<tr>
<th>tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
</tr>
<tr>
<td>prev</td>
</tr>
</tbody>
</table>

[task_struct]
<table>
<thead>
<tr>
<th>tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
</tr>
<tr>
<td>prev</td>
</tr>
</tbody>
</table>

init_task

...
Memory Forensics - Listing Processes

init_task

task_struct

- tasks
  - next
  - prev

task_struct

- tasks
  - next
  - prev

task_struct

- tasks
  - next
  - prev
Memory Forensics - Listing Processes

```
<table>
<thead>
<tr>
<th>task_struct</th>
<th>task_struct</th>
<th>task_struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>tasks</td>
<td>tasks</td>
<td>tasks</td>
</tr>
<tr>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>prev</td>
<td>prev</td>
<td>prev</td>
</tr>
</tbody>
</table>

init_task

linux_pslist

pid_hash

linux_pidhashtable

[-1560x-1987] 3
Memory Forensics - Listing Processes

The diagram illustrates the basic data structures involved in listing processes.

- **linux_pslst**: A list of task_structs.
- **tasks**: Pointer to the next task_struct in the list.
- **next**: Pointer to the next task_struct.
- **prev**: Pointer to the previous task_struct.
- **init_task**: The first task_struct in the list.
- **pid_hash**: A hash table mapping process IDs to task_structs.
- **linux_pidhashtable**: The hash table structure.
Forensic analyses are manually created by humans.
Forensic analyses are manually created by humans.

- Are there other techniques to list processes?
  Linux kernel 4.19: ~6000 structures with ~40000 fields
Forensic analyses are manually created by humans.

- Are there other techniques to list processes?
  Linux kernel 4.19: ~6000 structures with ~40000 fields

- How can we compare them?
  Shortest one? Most stable across different kernels?
Build a graph of kernel structures
Contributions

Build a graph of kernel structures

Define metrics to evaluate analyses
Contributions

Build a graph of kernel structures

Define metrics to evaluate analyses

Study analyses as paths on the graph
worklist ← kernel global variables;
while worklist ≠ ∅ do
  s ← worklist.pop();
  new_structs ← Explore(s);
  worklist.push(new_structs);
end while
Kernel Graph - Creation

worklist ← kernel global variables;
while worklist ≠ ∅ do
    s ← worklist.pop();
    new_structs ← Explore(s);
    worklist.push(new_structs);
end while

Challenge

Kernel “abstract data types”
Kernel Graph - ADT Challenge

(task_struct) tasks

list_head

children

list_head

(task_struct) tasks

list_head

children

list_head

(task_struct) tasks

list_head

children

list_head
Kernel Graph - ADT Challenge

task_struct

tasks

list_head

children

list_head

... -> ...

task_struct

tasks

list_head

siblings

list_head

... -> ...

task_struct

tasks

list_head

siblings

list_head

... -> ...

7
Solved with a Clang plugin that analyzes the kernel AST

```c
list_add(&p->tasks, &init_task.tasks);
list_add(&p->sibling, &p->children);
```

↓

```c
struct task_struct.tasks -> struct task_struct.tasks
struct task_struct.children -> struct.task_struct.siblings
```
The Graph

- 100k Structures (Nodes)
- 840k Pointers (Edges)
Metrics should capture different *aspects* of memory forensics:
Metrics should capture different aspects of memory forensics:

- Non-atomic memory acquisition (i.e. kernel driver)
Metrics should capture different aspects of memory forensics:

- Non-atomic memory acquisition (i.e. kernel driver)
- Layout of kernel structures changes across different kernel versions and configurations
Metrics should capture different aspects of memory forensics:

- Non-atomic memory acquisition (i.e. kernel driver)
- Layout of kernel structures changes across different kernel versions and configurations
- Attackers can modify kernel structures
Proposed Metrics

• Atomicity
• Stability
• Consistency
• Generality
• Reliability
Proposed Metrics

• Atomicity
• Stability
• Consistency
• Generality
• Reliability
**Metrics**

**Atomicity**: distance in memory between two connected structures
**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

## Volatility Plugin

<table>
<thead>
<tr>
<th>Volatility Plugin</th>
<th>Fast</th>
<th>Slow</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>
### Evaluation of Current Analyses

**Volatility Plugin**

<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>

96% of the nodes → giant strongly connected component (contains on average 53% of total nodes)
### 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>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fast</td>
<td>Slow</td>
</tr>
<tr>
<td>linux_arp</td>
<td>13</td>
<td>12,000</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>linux_check_creds</td>
<td>248</td>
<td>2</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>linux_check_modules</td>
<td>151</td>
<td>700</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>linux_check_tty</td>
<td>13</td>
<td>30</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>linux_find_file</td>
<td>14955</td>
<td>0</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>linux_ifconfig</td>
<td>12</td>
<td>12,000</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>linux_lsmod</td>
<td>12</td>
<td>700</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>linux_lsof</td>
<td>821</td>
<td>0</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>linux_mount</td>
<td>495</td>
<td>10</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>linux_pidhashtable</td>
<td>469</td>
<td>30</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>linux_proc_maps</td>
<td>4722</td>
<td>0</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>linux_pslist</td>
<td>124</td>
<td>30</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

**Consistency:** 5 inconsistent plugins when fast acquisition

7 inconsistent plugins when slow acquisition
Finding New Ways to List Processes

Much harder than expected!

- Hundreds of millions of paths when considering the shortest paths from every root node to every `task_struct`
- Not every path represent an heuristics, because heuristics must be generated by an `algorithm`
Much harder than expected!

- Hundreds of millions of paths when considering the shortest paths from every root node to every `task_struct`
- Not every path represent an heuristic, because heuristics must be generated by an algorithm

To limit the path explosion problem:

- Removed every root node that is not connected to every `task_struct`
- Remove edges used by known techniques (i.e. `tasks` field)
- Remove similar edges (parallel edges with same weights)
- Merge similar paths into `templates` (struct type + remove adjacent same type nodes)

Resulted in 4000 path templates!
# Kernel Graph - New Heuristics Results

<table>
<thead>
<tr>
<th>Category</th>
<th>Root Node</th>
<th>Nodes</th>
<th>task_struct</th>
<th>Stability</th>
<th>Generality</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>cgroup</td>
<td>css_set_table</td>
<td>172</td>
<td>156</td>
<td>10.00</td>
<td>29/85</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>cgrp_df_root</td>
<td>186</td>
<td>156</td>
<td>10.00</td>
<td>29/85</td>
<td>✓</td>
</tr>
<tr>
<td>memory/fs</td>
<td>dentry_hash</td>
<td>58383</td>
<td>23</td>
<td>0.00</td>
<td>36/85</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>inode_hash</td>
<td>14999</td>
<td>23</td>
<td>1.00</td>
<td>36/85</td>
<td>✗</td>
</tr>
<tr>
<td>workers</td>
<td>wq_workqueues</td>
<td>427</td>
<td>69</td>
<td>200.00</td>
<td>39/85</td>
<td>✓</td>
</tr>
</tbody>
</table>

All implemented as Volatility plugins!
Forensics analyses can be extracted and evaluated in a principled way!
A Principled Approach to Memory Forensics

Forensics analyses can be extracted and evaluated in a principled way!

- Kernel graph to model kernel structures
- Set of metrics to capture memory forensics aspects
- Experiments to study current and future techniques
Our framework enables more future research!

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

Twitter: @pagabuc
Email: pagani@eurecom.fr
Examples

struct hlist_head [128] - struct css_set - struct
task_struct

struct hlist_bl_head *- struct dentry - struct inode -
struct vm_area_struct - struct mm_struct - struct
task_struct