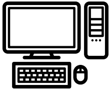
Memory Smearing: Myth or Reality?

Fabio Pagani

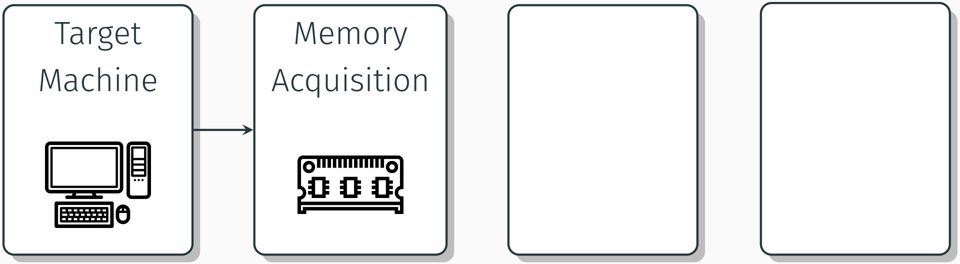
EURECOM
Sophia Antipolis

30th September 2019

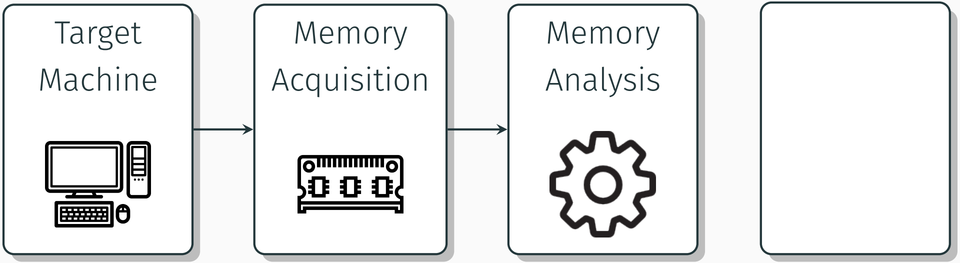
Target Machine



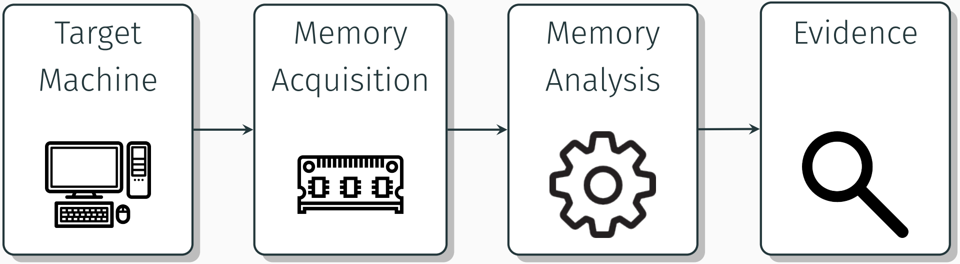
Memory Forensics - Introduction



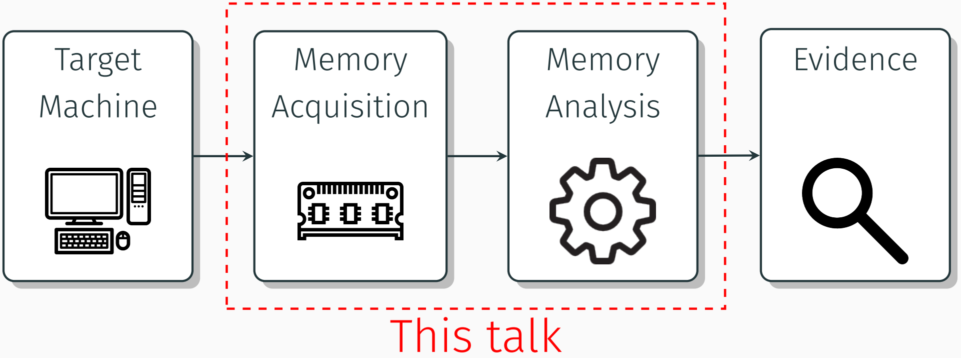
Memory Forensics - Introduction



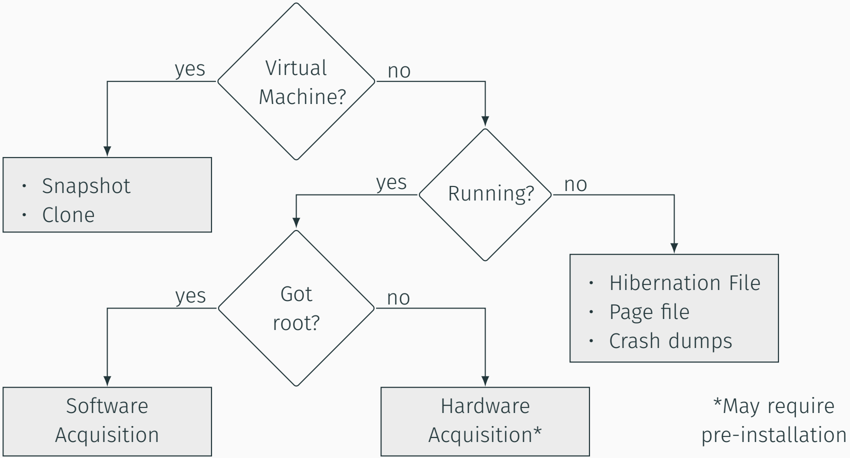
Memory Forensics - Introduction



Memory Forensics - Introduction

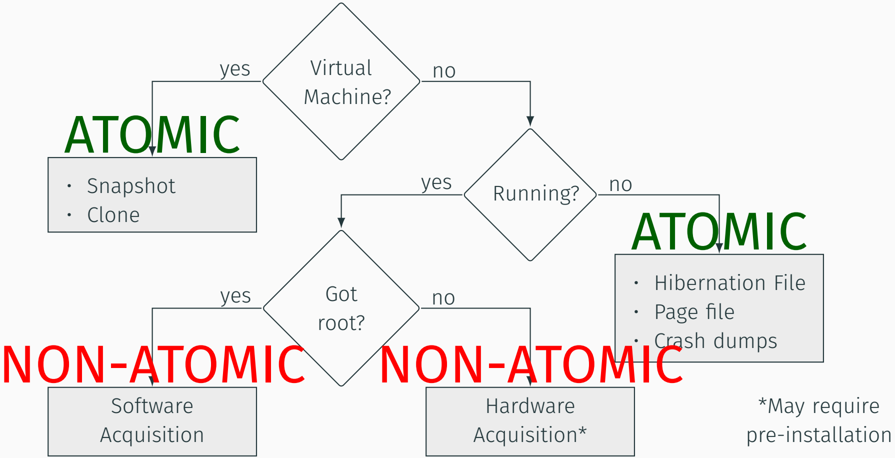


Memory Acquisition



Decision tree adapted from The Art of Memory Forensics

Memory Acquisition



Decision tree adapted from The Art of Memory Forensics

ATOMIC

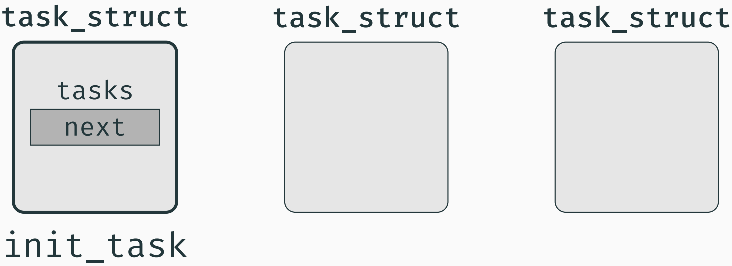
ATOMIC

NON-ATOMIC

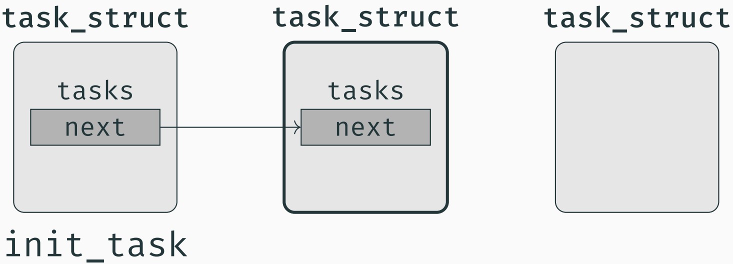
Memory Analysis

- The “core” of memory forensics.
- Several frameworks: Volatility, Rekall (Google), Mandiant’s Memoryze..
- Examples of information that can be extracted:
  - Processes $\rightarrow$ list/tree, open files, memory mappings, extract executable and shared libraries
  - Modules $\rightarrow$ list, code, unloaded modules
  - Networking $\rightarrow$ connections, sockets, arp table
  - Windows Registry $\rightarrow$ keys, password hashes
  - System information $\rightarrow$ clipboard content, screenshot
- Every task is “organized” in a plugin

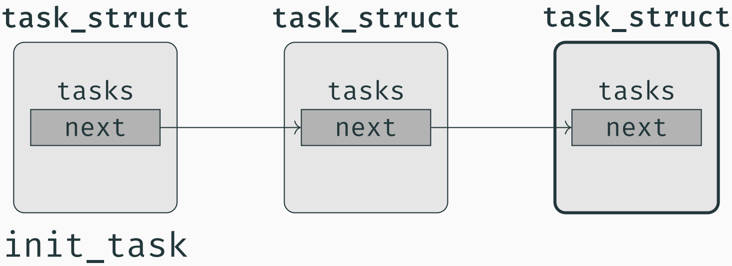
Memory Analysis



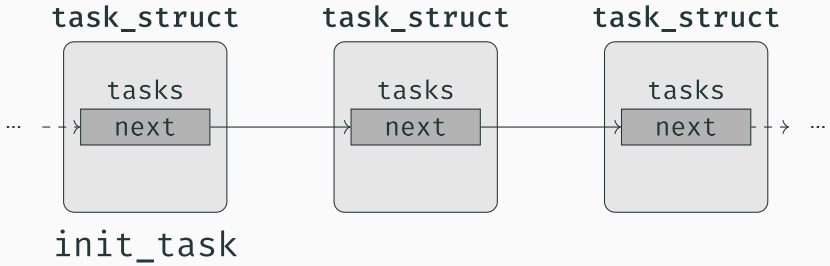
Memory Analysis

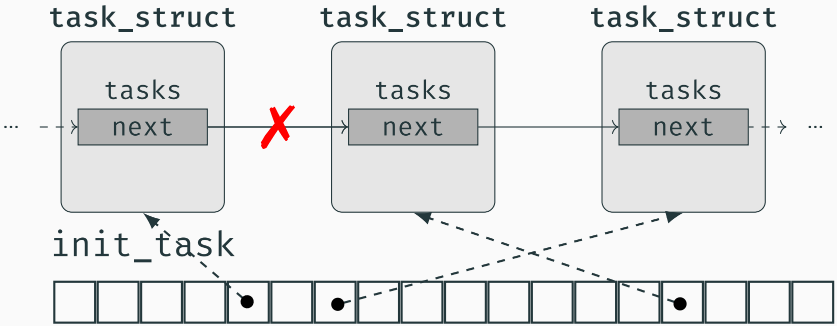


Memory Analysis



Memory Analysis





Problem

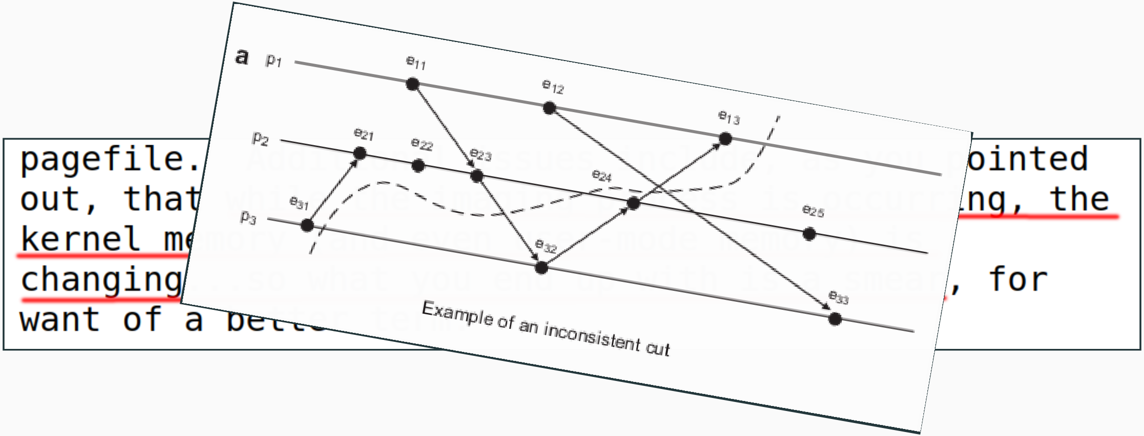
Some pointers can be **inconsistent**!

Memory Smearing - History

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.

Alan Carvey — Security Incidents ML (2005)

Memory Smearing - History

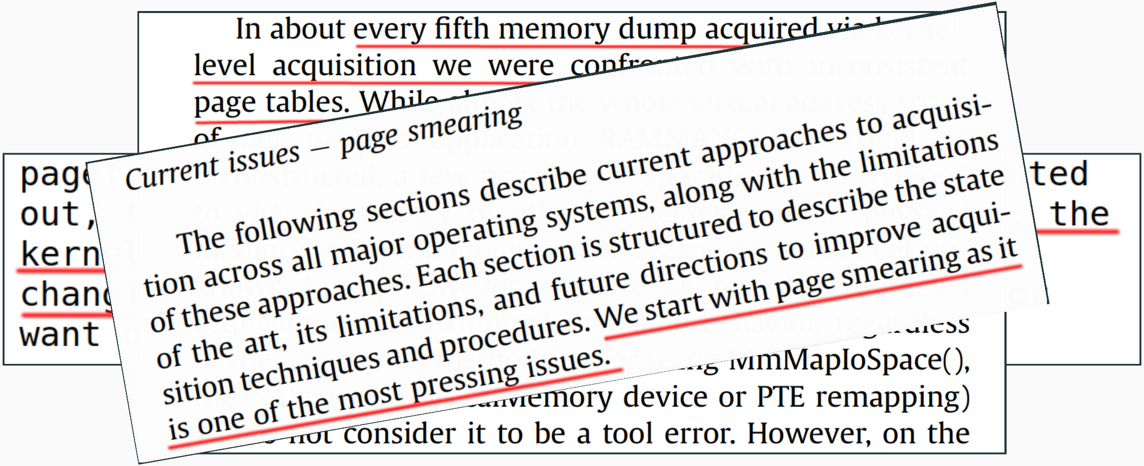


Vomel et. al — Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition (DFRWS 2009)

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 RAMMANGL.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

Gruhn et. al — Evaluating atomicity, and integrity of correct memory acquisition methods (DFRWS 2016)

Memory Smearing - History



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)

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

  But if you do you can have some troubles

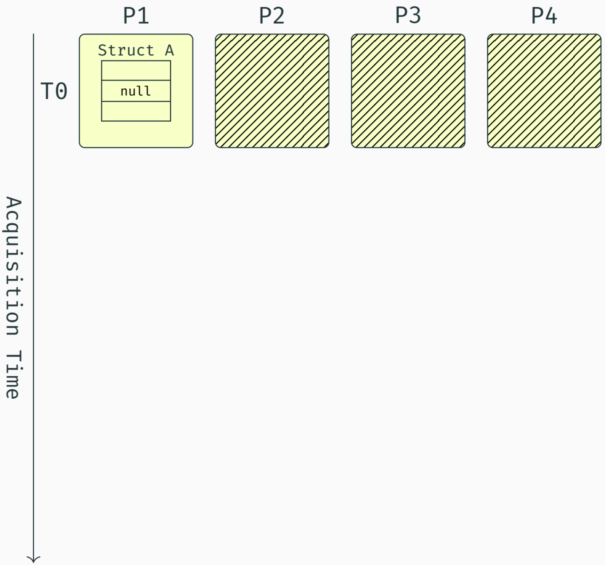
- You can trigger a blue screen (not really cool)

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

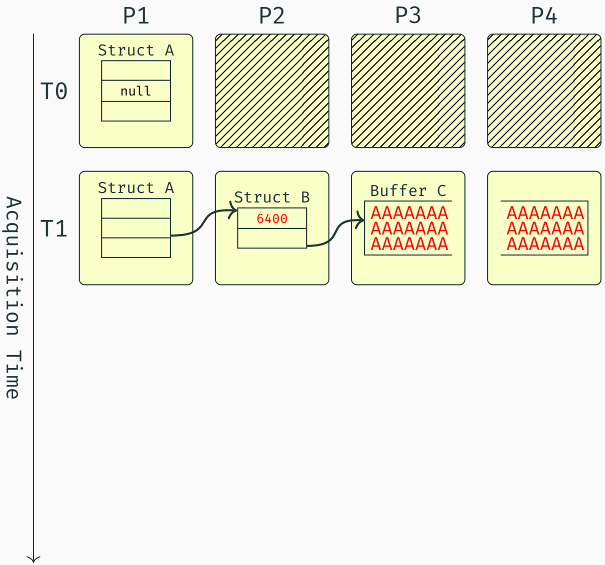
Le Berre — From corrupted memory dump to rootkit detection (NDH 2018)

The Problem

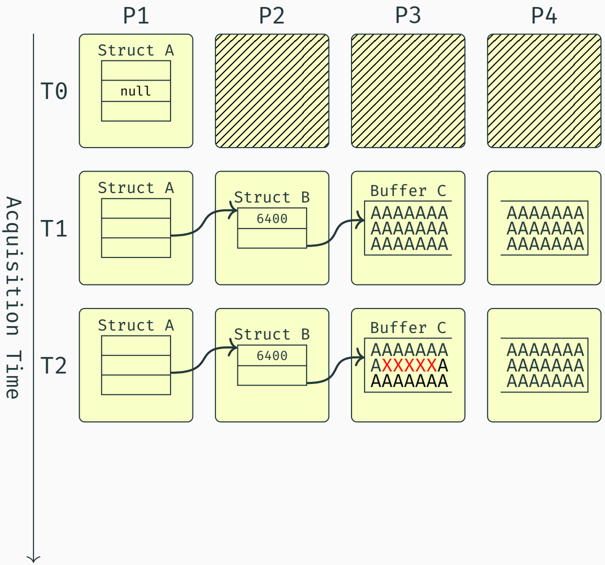
Types of Inconsistency



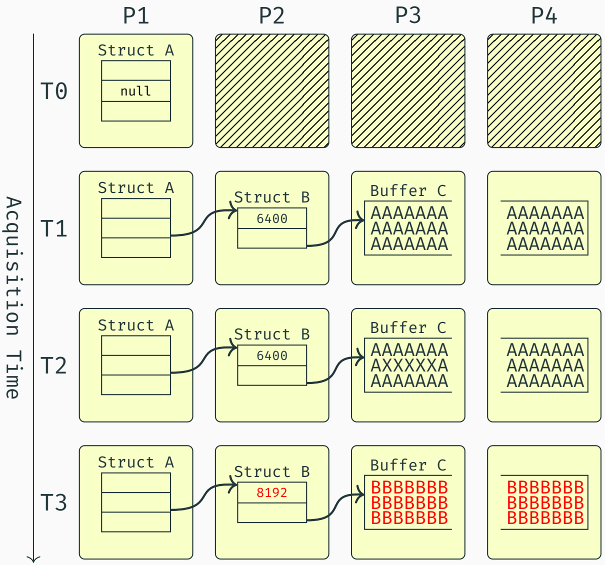
Types of Inconsistency



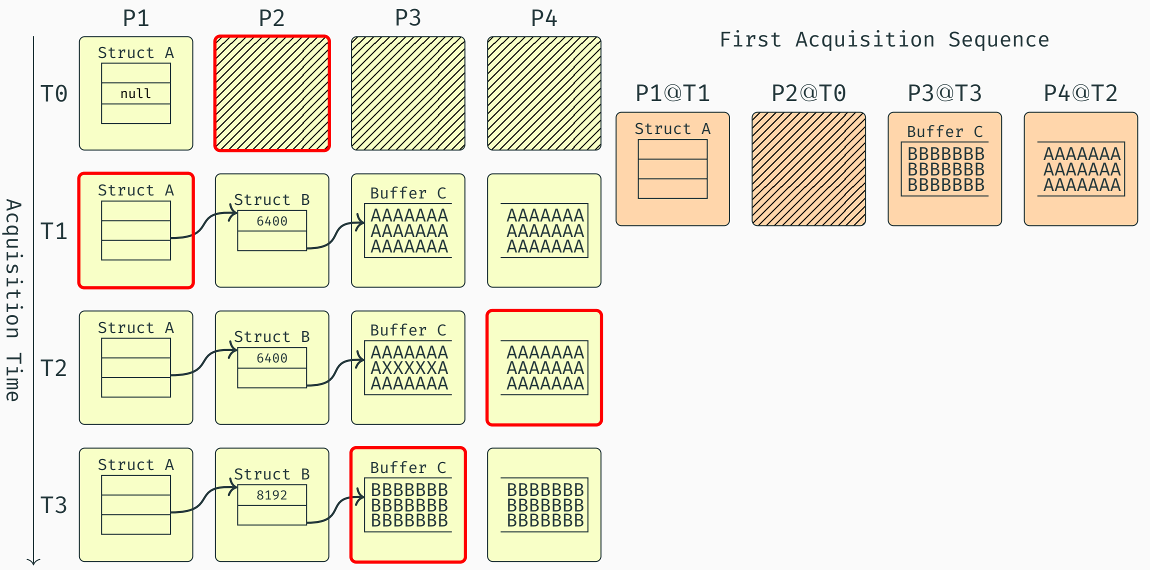
Types of Inconsistency



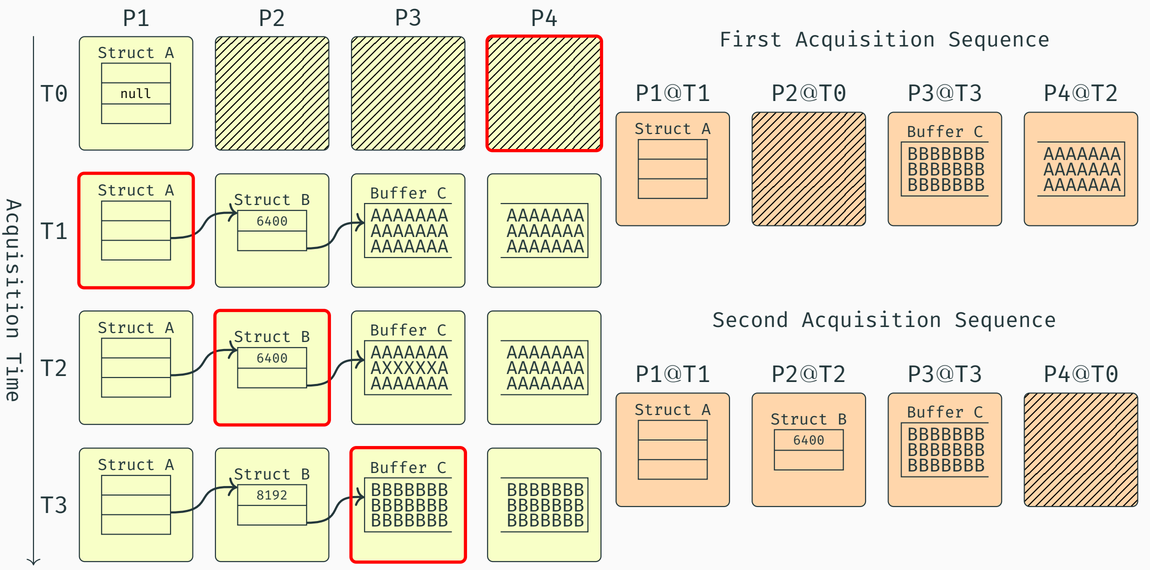
Types of Inconsistency



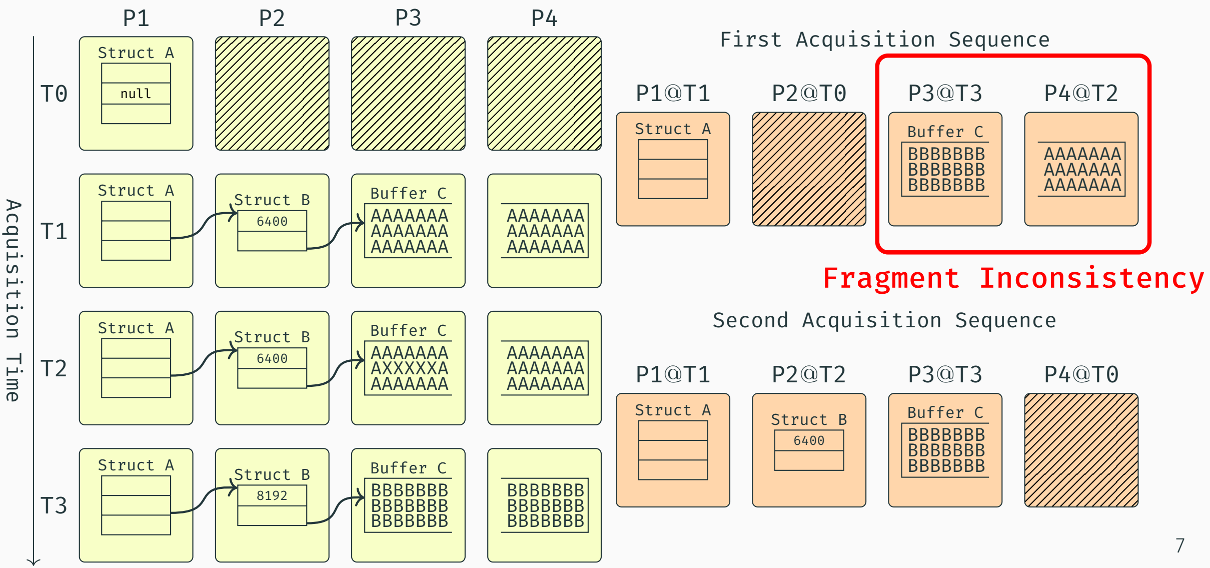
Types of Inconsistency



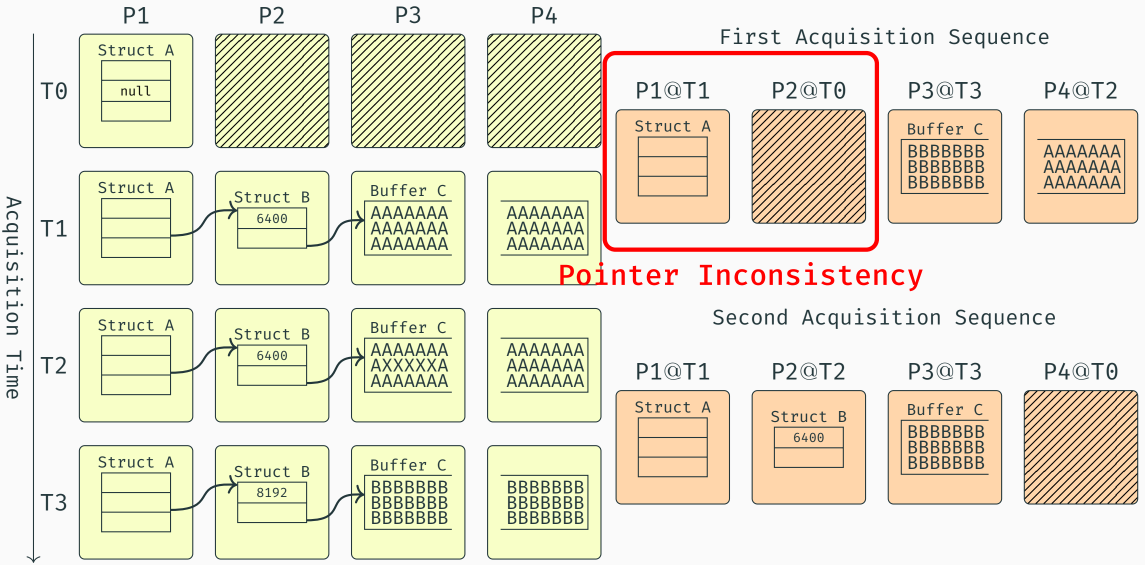
Types of Inconsistency



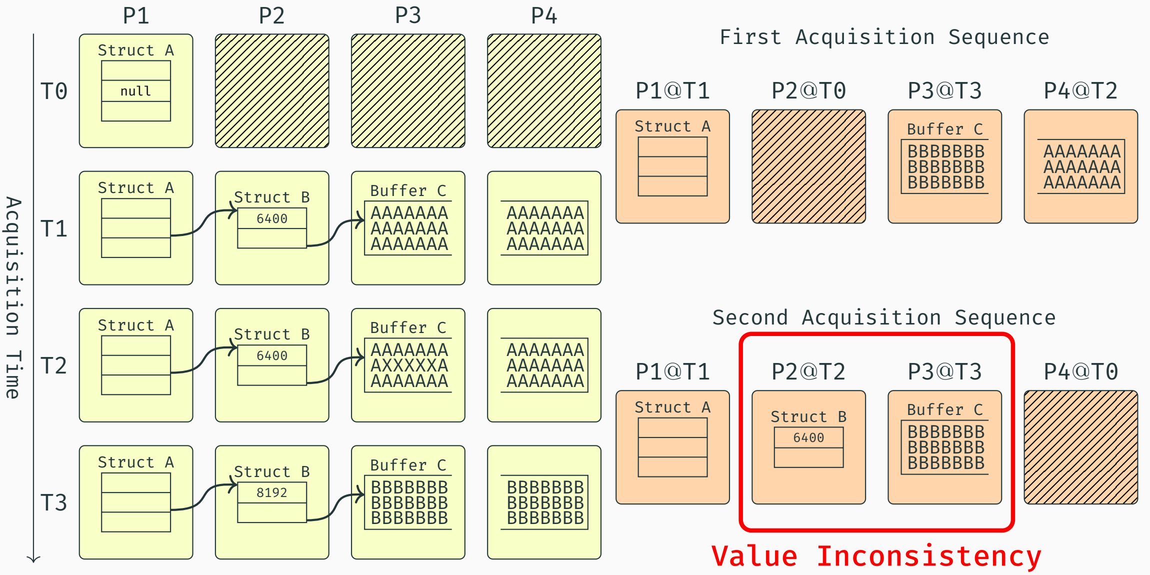
Types of Inconsistency



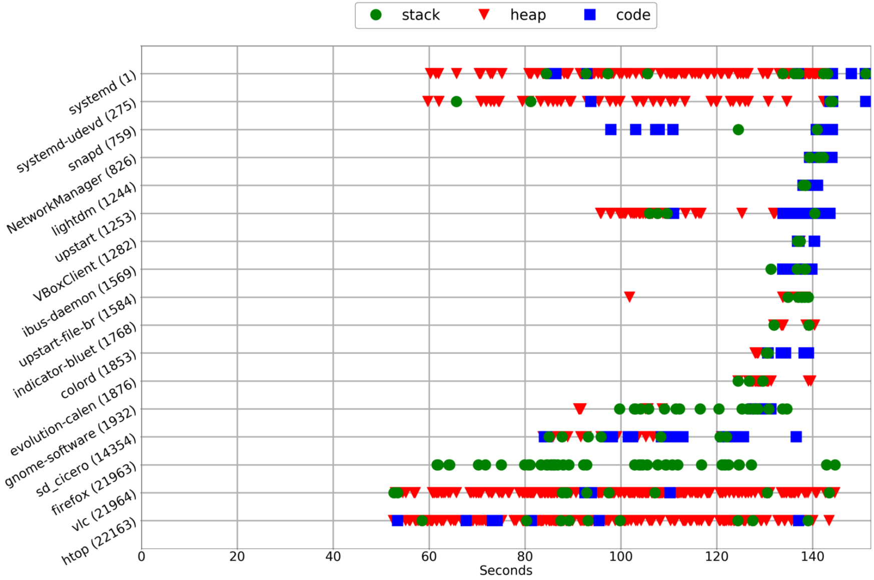
Types of Inconsistency



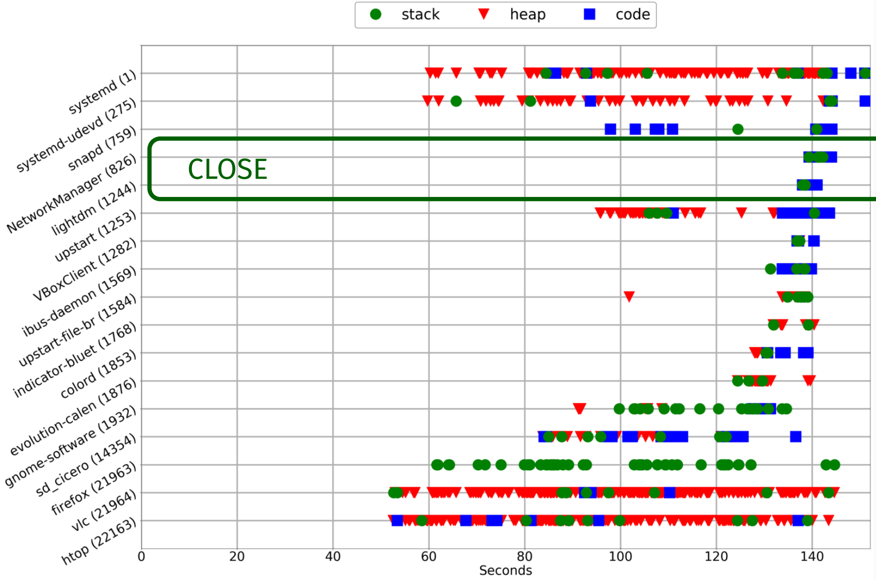
Types of Inconsistency



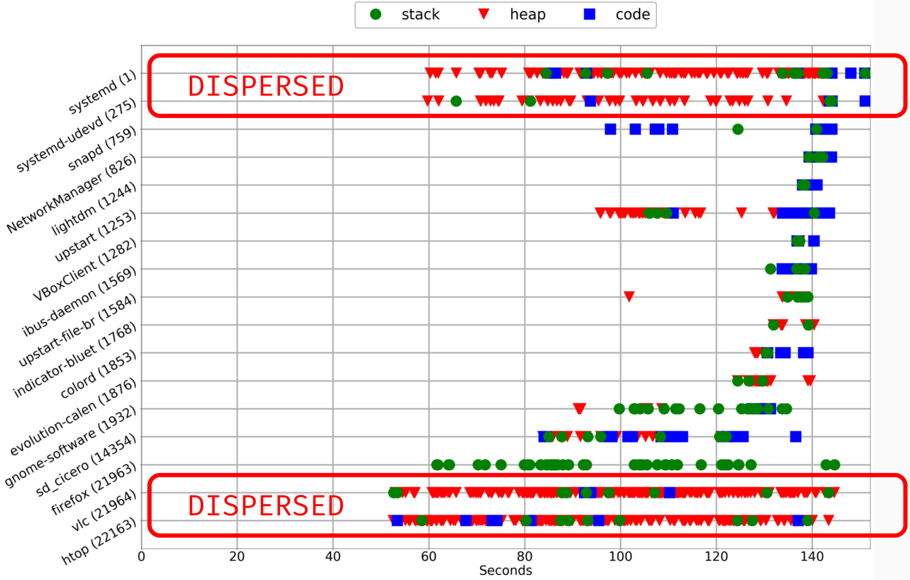
Impact Estimation - Fragmentation



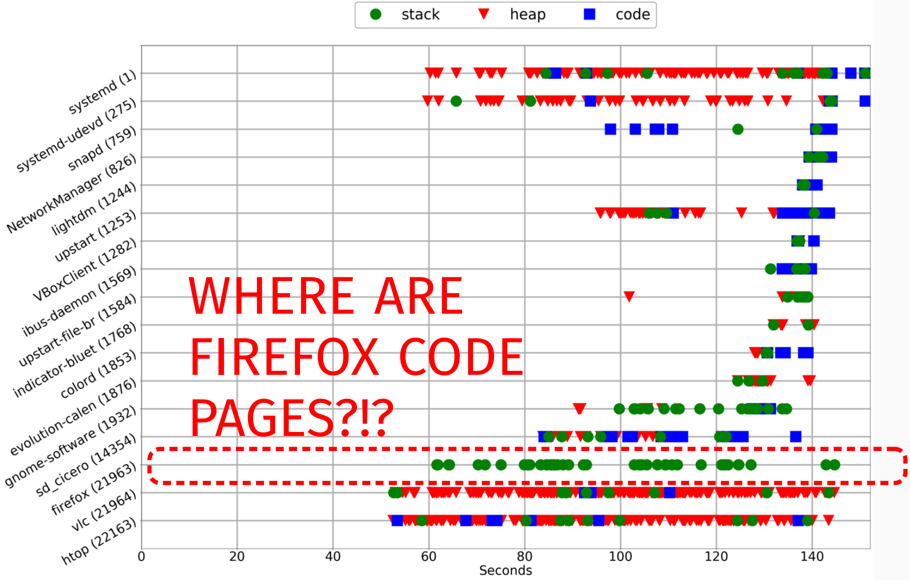
Impact Estimation - Fragmentation



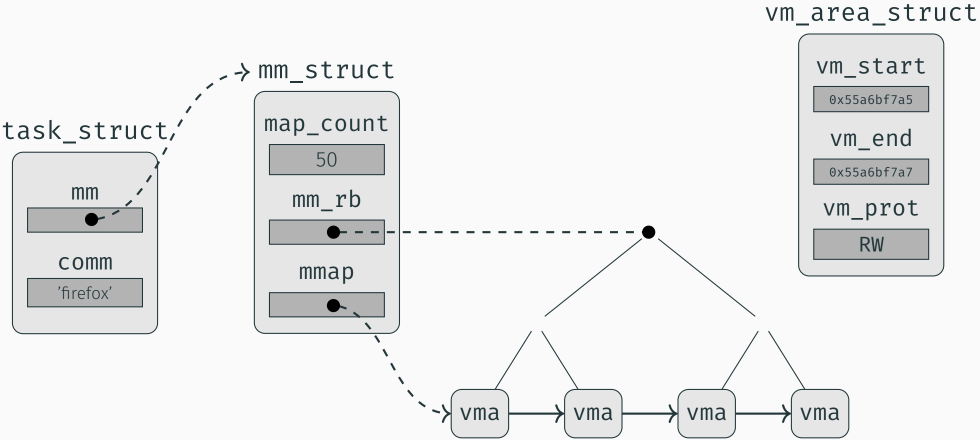
Impact Estimation - Fragmentation



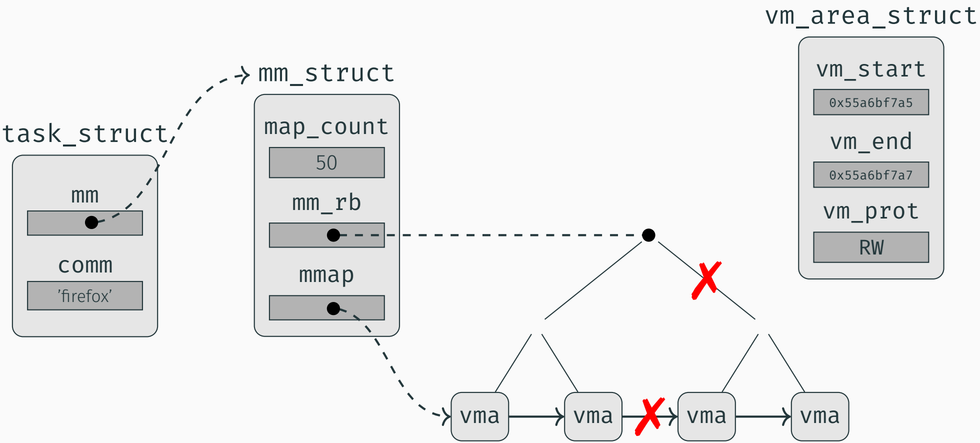
Impact Estimation - Fragmentation



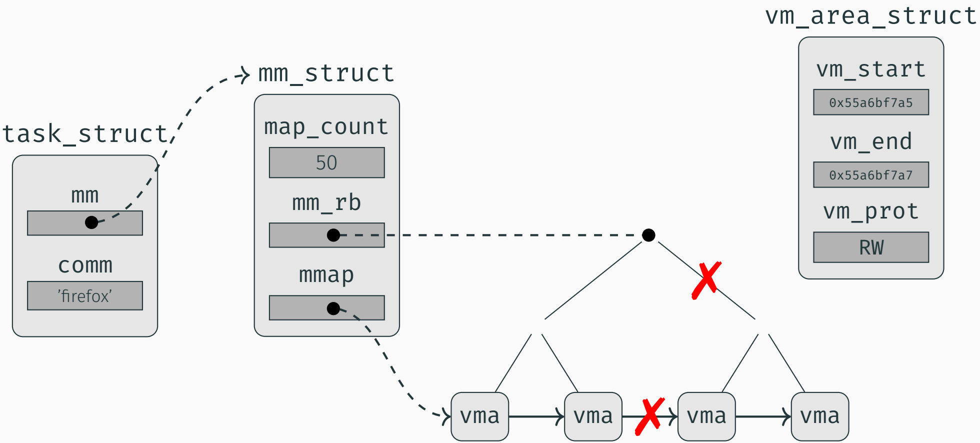
Kernel-Space Integrity



Kernel-Space Integrity



Kernel-Space Integrity



Volatility Plugin: `map_count == list_len(mmap)`

`map_count == tree_len(mm_rb)`

Kernel-Space Integrity

	Scenario 1 (Firefox)	Scenario 2 (Apache)	Scenario 3 (Malware)
List mismatch	100%	71%	80%
Tree mismatch	100%	73%	80%
Total	100%	78%	80%

	Scenario 1 (Firefox)	Scenario 2 (Apache)	Scenario 3 (Malware)
List mismatch	100%	71%	80%
Tree mismatch	100%	73%	80%
Total	100%	78%	80%

Is this actually a problem?

Kernel-Space Integrity

	Scenario 1 (Firefox)	Scenario 2 (Apache)	Scenario 3 (Malware)
List mismatch	100%	71%	80%
Tree mismatch	100%	73%	80%
Total	100%	78%	80%

Is this actually a problem?

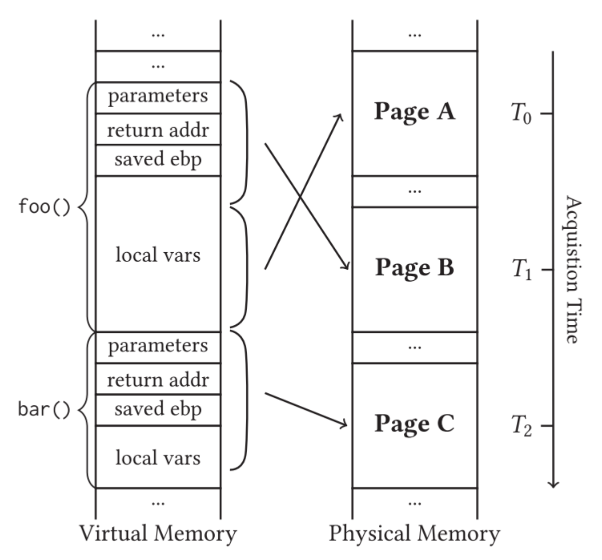
- List → Firefox stack and code **never** present
- Tree → Firefox stack present **10%**, code present **30%**

	Scenario 1 (Firefox)	Scenario 2 (Apache)	Scenario 3 (Malware)
List mismatch	100%	71%	80%
Tree mismatch	100%	73%	80%
Total	100%	78%	80%

Is this actually a problem?

- List $\rightarrow$ Firefox stack and code **never** present
- Tree $\rightarrow$ Firefox stack present **10%**, code present **30%**
- Key recovery for WannaCry and NotPetya

User-Space Integrity



User-Space Integrity

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

User-Space Integrity

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
Frames	-	6	-	6	8	-	-	-	6	-
Physical Pages	4	5	5	4	4	5	4	4	5	5
Acquisition Time (s)	3.2	30.0	37.8	37.0	0.25	26.0	28.6	1.0	27.6	39.9
`rbp` delta (s)	7.7	38.8	49.6	4[illegible].7	7.3	43.4	4.3	4.0	15.1	5.64
Corrupted (`registers`)	✓	–	✓	–	–	✓	✓	✓	–	✓
Corrupted (`frame pointers`)	–	–	–	–	–	–	✓	–	–	–
Inconsistent data	N/A	✓	N/A	✓	–	N/A	N/A	N/A	✓	N/A

Is this actually a problem?

User-Space Integrity

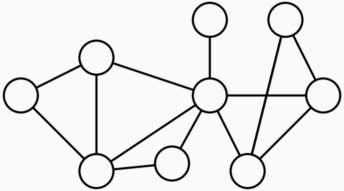
	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
Frames	-	6	-	6	8	-	-	-	6	-
Physical Pages	4	5	5	4	4	5	4	4	5	5
Acquisition Time (s)	3.2	30.0	37.8	37.0	0.25	26.0	28.6	1.0	27.6	39.9
`rbp` delta (s)	7.7	38.8	49.6	4[illegible].7	7.3	43.4	4.3	4.0	15.1	5.64
Corrupted (`registers`)	✓	–	✓	–	–	✓	✓	✓	–	✓
Corrupted (`frame pointers`)	–	–	–	–	–	–	✓	–	–	–
Inconsistent data	N/A	✓	N/A	✓	–	N/A	N/A	N/A	✓	N/A

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)

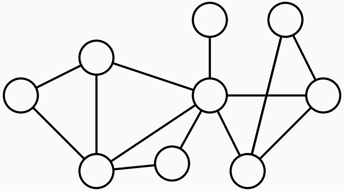
Can we study smear in a more generic way?

Build a graph of kernel structures

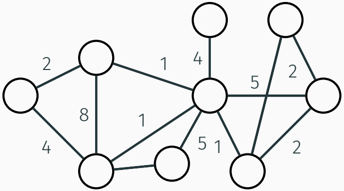


Can we study smear in a more generic way?

Build a graph of kernel structures

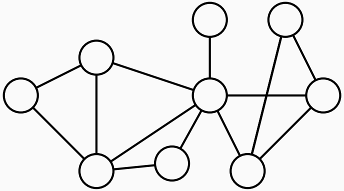


Define metrics to evaluate analyses

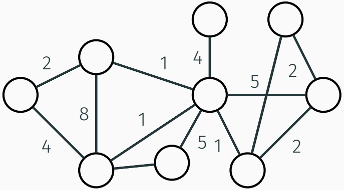


Can we study smear in a more generic way?

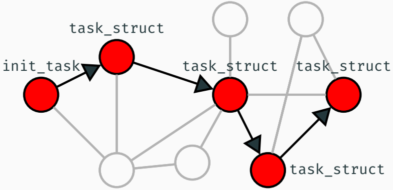
Build a graph of kernel structures



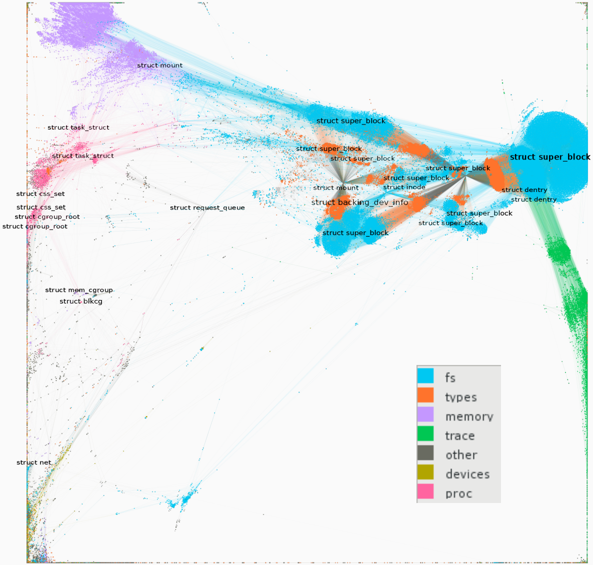
Define metrics to evaluate analyses



Study analyses as paths on the graph



The Graph

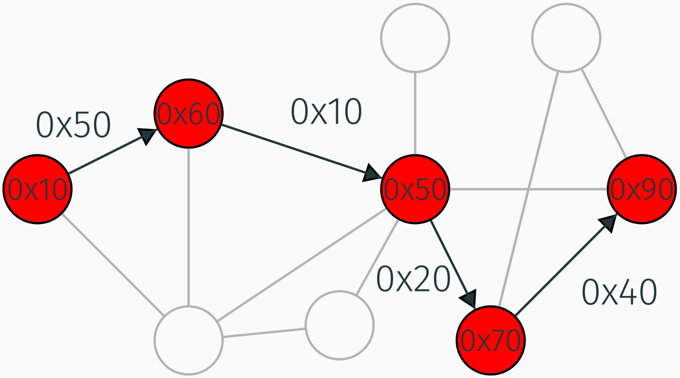


- 100k Structures (Nodes)
- 840k Pointers (Edges)

Proposed Metrics

- Atomicity
- Stability
- Consistency

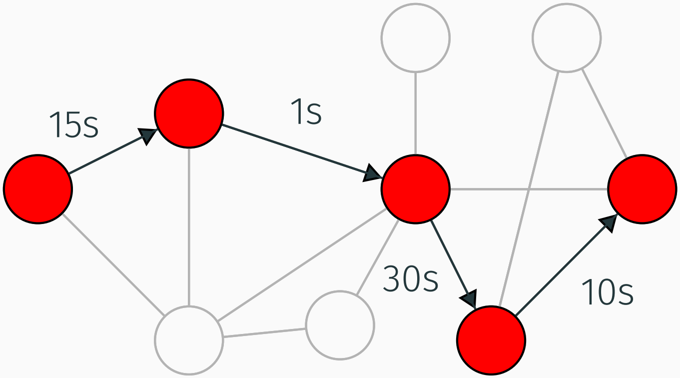
Atomicity: distance in memory between two connected structures



Metrics

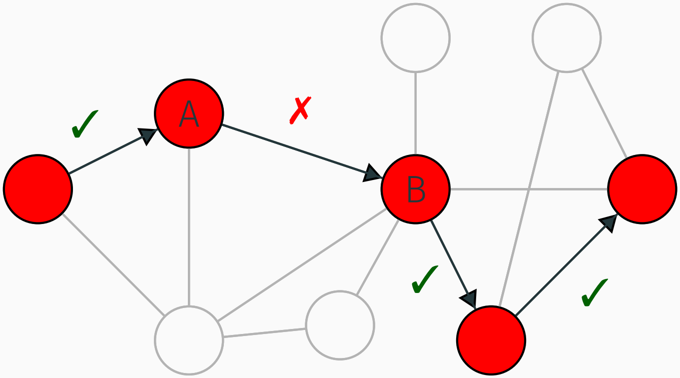
Stability: how long an edge remains stable in a running machine

- 25 snapshots at [0s, 1s, 5s, ..., 3h]



Metrics

Consistency: Atomicity + Stability



Evaluation of Current Analyses

Volatility Plugin	# Nodes		
linux_arp	13		
linux_check_creds	248		
linux_check_modules	151		
linux_check_tty	13		
linux_find_file	14955		
linux_ifconfig	12		
linux_lsmod	12		
linux_lsof	821		
linux_mount	495		
linux_pidhashtable	469		
linux_proc_maps	4722		
linux_pslist	124		

Evaluation of Current Analyses

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

Stability: 3 paths never changed in over 3 hours

11 paths changed in less than 1 minute

Evaluation of Current Analyses

Volatility Plugin	# Nodes	Stability (s)	Consistency Fast	Consistency Slow
linux_arp	13	12,000	✓	✓
linux_check_creds	248	2	✓	✓
linux_check_modules	151	700	✓	✓
linux_check_tty	13	30	✓	✓
linux_find_file	14955	0	✗	✗
linux_ifconfig	12	12,000	✓	✓
linux_lsmod	12	700	✓	✓
linux_lsof	821	0	✗	✗
linux_mount	495	10	✓	✗
linux_pidhashtable	469	30	✓	✗
linux_proc_maps	4722	0	✗	✗
linux_pslist	124	30	✓	✓

Consistency: 5 inconsistent plugins when fast acquisition

7 inconsistent plugins when slow acquisition

Solutions

A New Temporal Dimension - Recording Time

- 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 $\rightarrow$ 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

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!

DEMO

Papers + Code

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`