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

Fabio Pagani and Davide Balzarotti

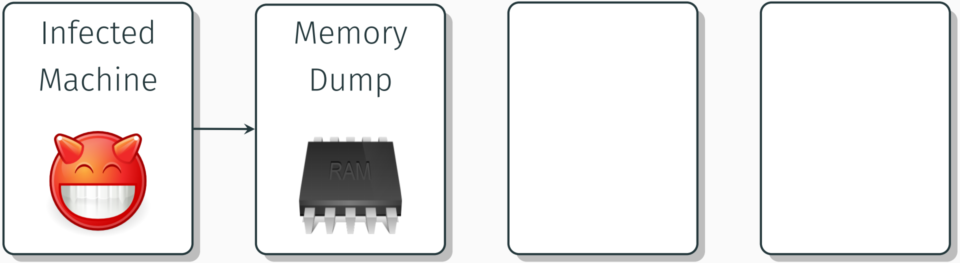
EURECOM
Sophia Antipolis

Usenix Security '19

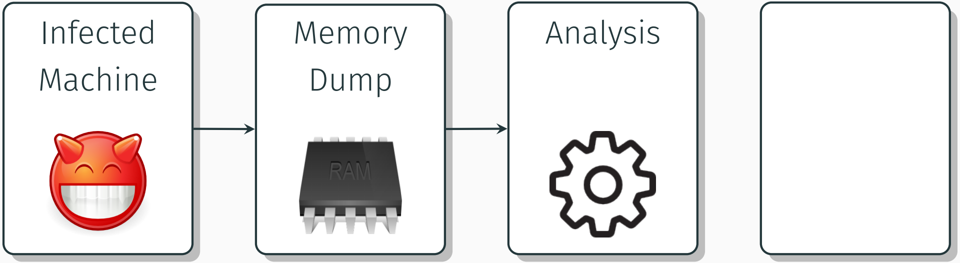
Memory Forensics - Introduction

Infected Machine

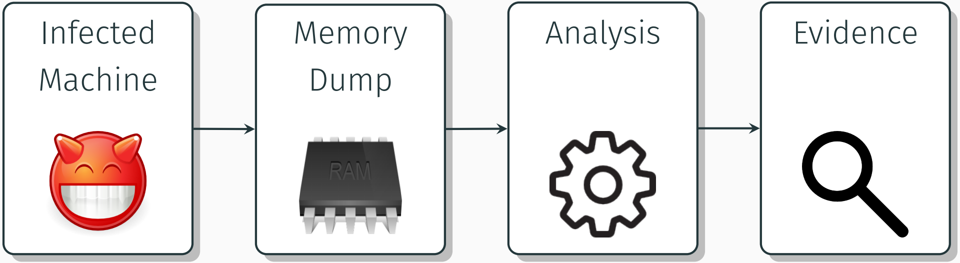
Memory Forensics - Introduction



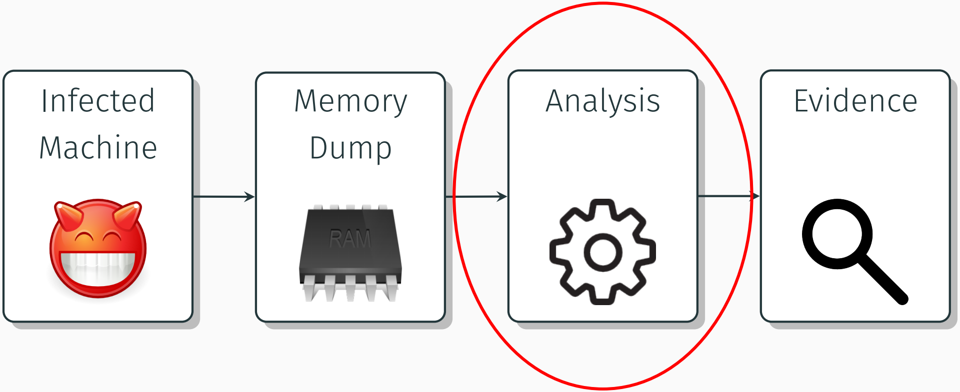
Memory Forensics - Introduction



Memory Forensics - Introduction



Memory Forensics - Introduction



Extract the following information:

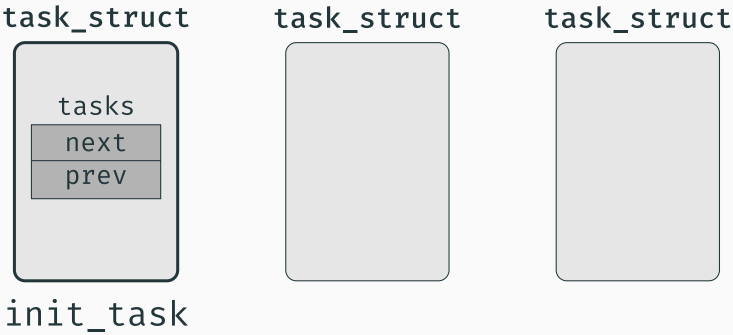
- List processes, kernel modules
- Open files, memory mappings, sockets..
- System information: routing table, kernel logs..

Extract the following information:

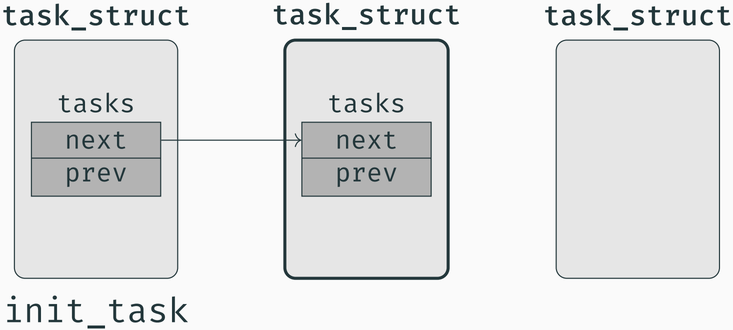
- List processes, kernel modules
- Open files, memory mappings, sockets..
- System information: routing table, kernel logs..

… and much more: Volatility (the most used memory forensic framework) has more than 100 plugins for Windows!

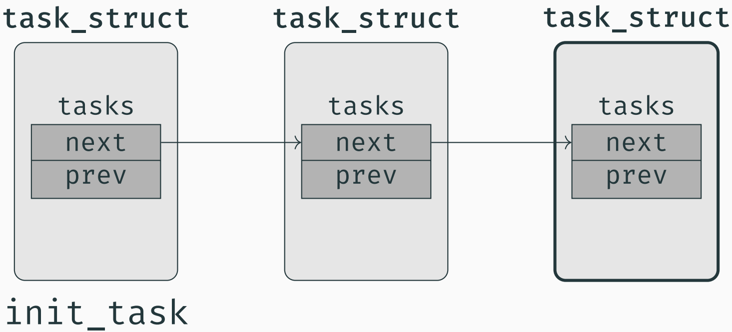
Memory Forensics - Listing Processes



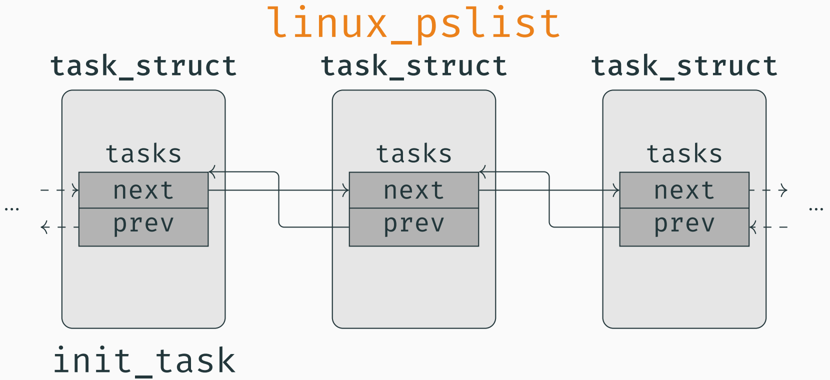
Memory Forensics - Listing Processes



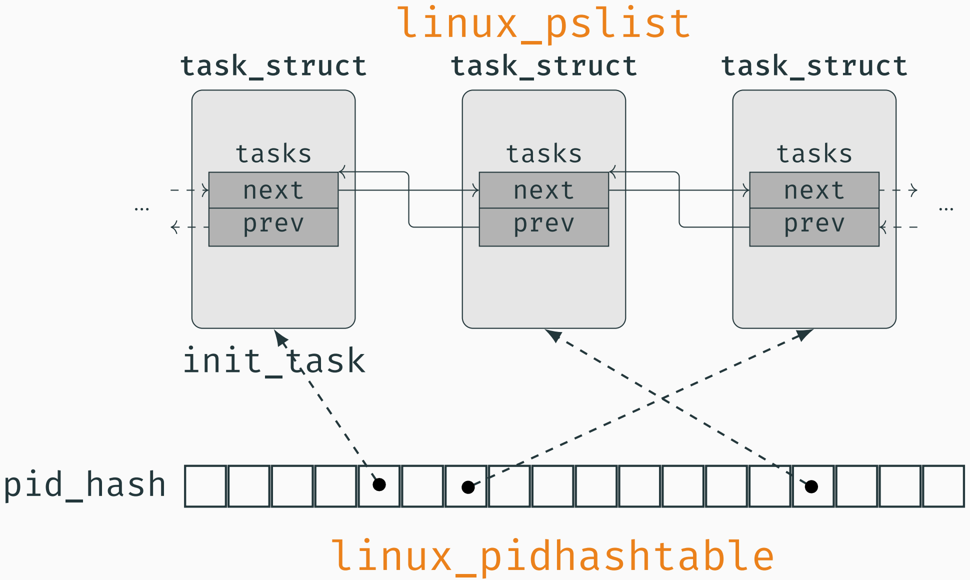
Memory Forensics - Listing Processes



Memory Forensics - Listing Processes



Memory Forensics - Listing Processes



Motivations

Forensic analyses are manually created by humans.

Motivations

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?

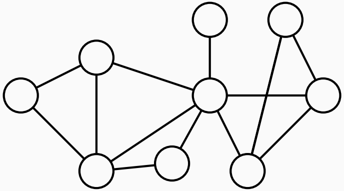
  Linux kernel 4.19: ~6000 structures with ~40000 fields

- How can we compare them?

  Shortest one? Most stable across different kernels?

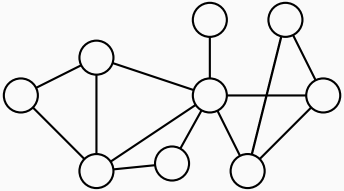
Contributions

Build a graph of kernel structures

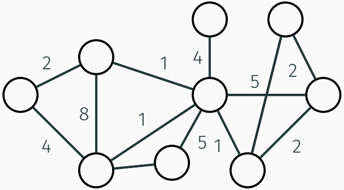


Contributions

Build a graph of kernel structures

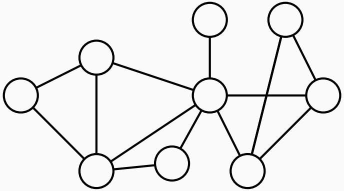


Define metrics to evaluate analyses

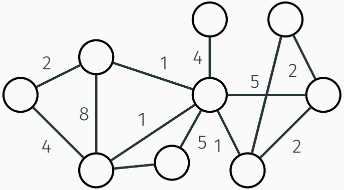


Contributions

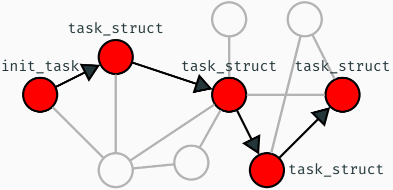
Build a graph of kernel structures



Define metrics to evaluate analyses



Study analyses as paths on the graph



Kernel Graph - Creation

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

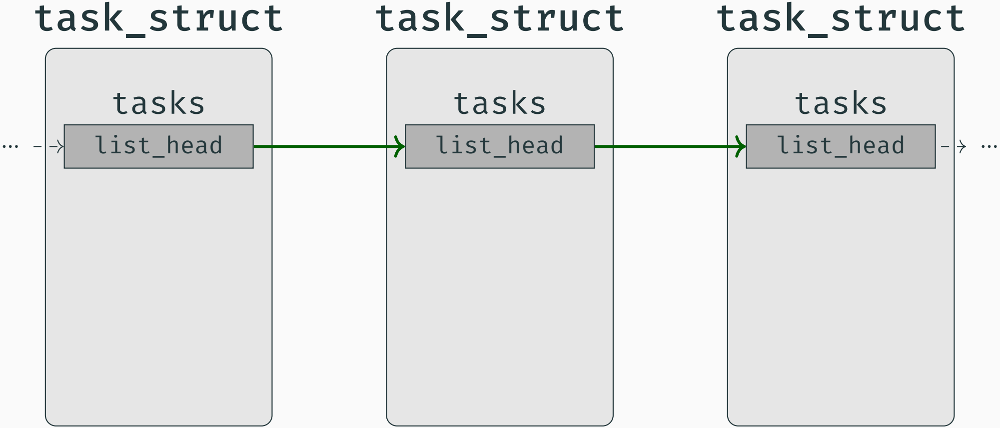
Kernel Graph - Creation

$worklist \leftarrow$ kernel global variables;
while $worklist \neq \emptyset$ **do**
    $s \leftarrow worklist.pop()$;
    $new\_structs \leftarrow Explore(s)$;
    $worklist.push(new\_structs)$;
end while

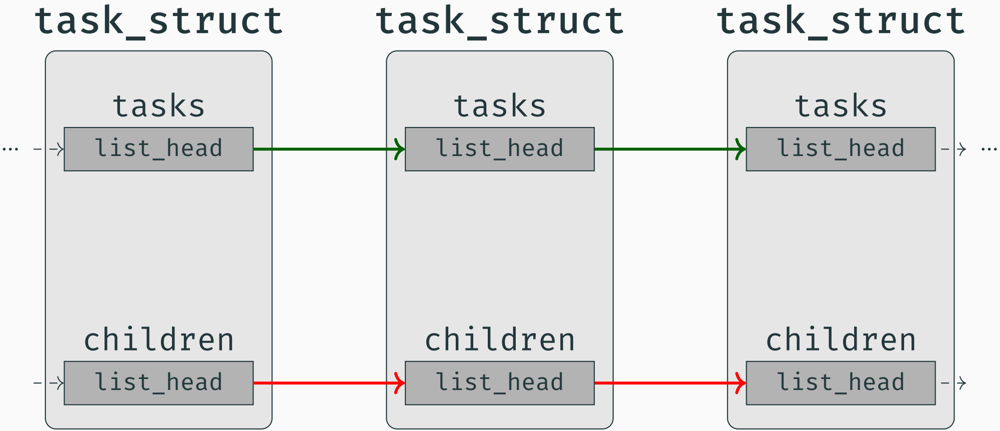
Challenge

Kernel “abstract data types”

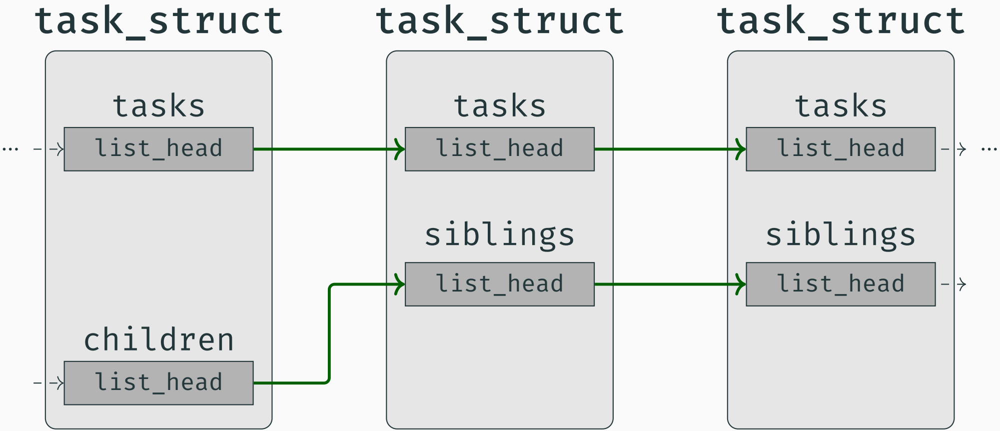
Kernel Graph - ADT Challenge



Kernel Graph - ADT Challenge



Kernel Graph - ADT Challenge



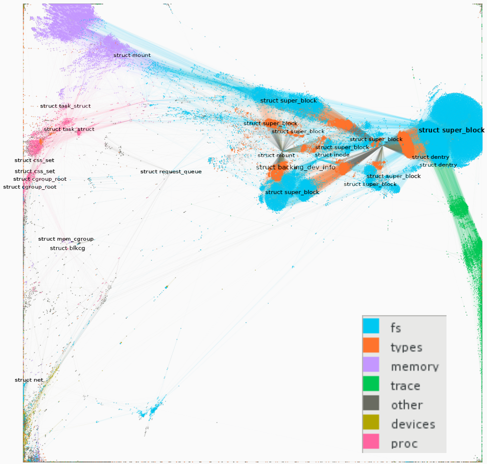
Solved with a Clang plugin that analyzes the kernel AST

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

↓

```
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 - Rationale

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

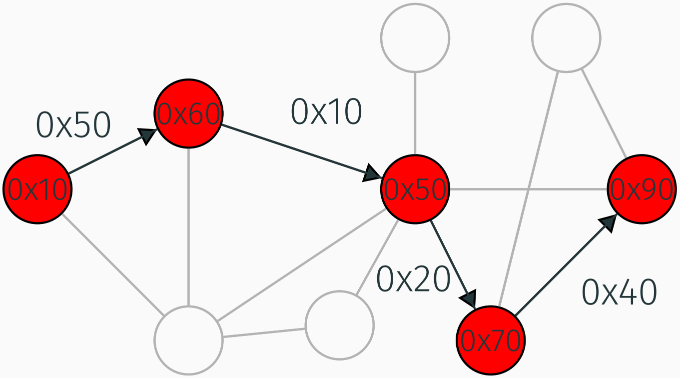
- Atomicity
- Stability
- Consistency
- Generality
- Reliability

Proposed Metrics

- Atomicity
- Stability
- Consistency
- Generality
- Reliability

Metrics

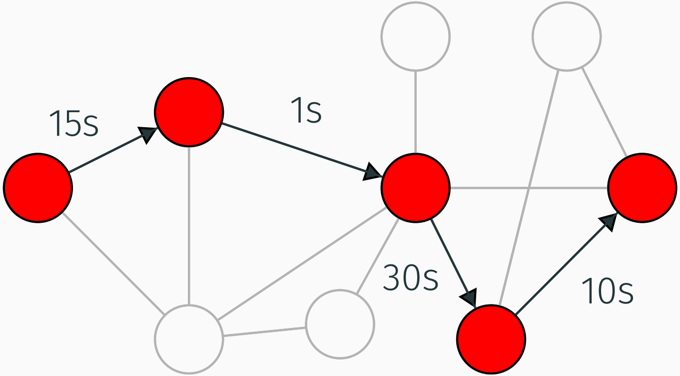
Atomicity: distance in memory between two connected structures



Metrics

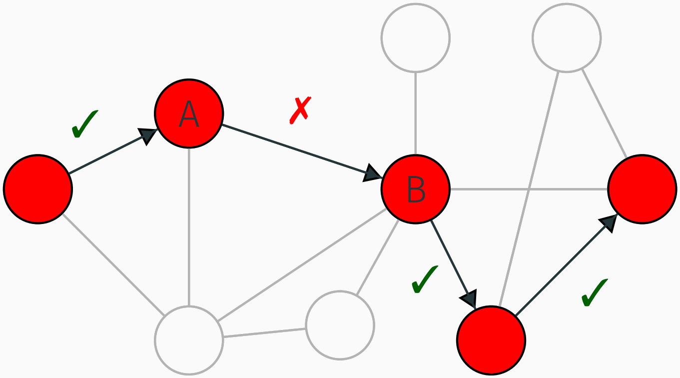
Stability: how long an edge remains stable in a running machine

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



Metrics

Consistency: Atomicity + Stability



Evaluation of Current Analyses

Volatility Plugin			
linux_arp			
linux_check_creds			
linux_check_modules			
linux_check_tty			
linux_find_file			
linux_ifconfig			
linux_lsmod			
linux_lsof			
linux_mount			
linux_pidhashtable			
linux_proc_maps			
linux_pslist			

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		

96% of the nodes $\rightarrow$ giant strongly connected component (contains on average 53% of total nodes)

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

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*

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*

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

Category	Root Node	# Nodes	# task_struct	Stability	Generality	Consistency
cgroup	css_set_table	172	156	10.00	29/85	✗
	cgrp_dfl_root	186	156	10.00	29/85	✓
memory/fs	dentry_hash	58383	23	0.00	36/85	✗
	inode_hash	14999	23	1.00	36/85	✗
workers	wq_workqueues	427	69	200.00	39/85	✓

All implemented as Volatility plugins!

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

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

Future Work

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`