European Research Council
Established by the European Commission

Advances in Memory Forensics

Fabio Pagani

EURECOM
Sophia Antipolis

9th September 2019

Publications List

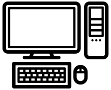
Towards Automated Profile Generation for Memory Forensics **F Pagani**, D Balzarotti	S&P 2020 *(revise)*
Back to the Whiteboard: a Principled Approach for the Assessment and Design of Memory Forensic Techniques **F Pagani**, D Balzarotti	USENIX 2019
Introducing the Temporal Dimension to Memory Forensics **F Pagani**, O Fedorov, D Balzarotti	TOPS 2019
Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis **F Pagani**, M Dell'Amico, D Balzarotti	CODASPY 2018
Taming transactions: Towards hardware-assisted control flow integrity using transactional memory M Muench, **F Pagani**, Y Shoshitaishvili, C Kruegel, G Vigna, D Balzarotti	RAID 2016
Measuring the Role of Greylisting and Nolisting in Fighting Spam **F Pagani**, M De Astis, M Graziano, A Lanzi, D Balzarotti	DSN 2016

Memory Forensics - Introduction

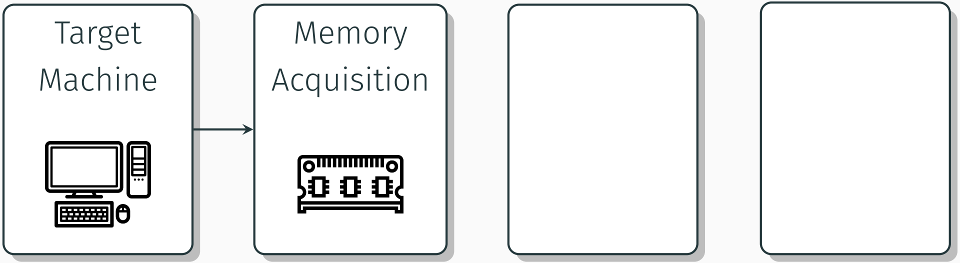
> *Memory forensics is arguably the most* **fruitful**, **interesting**, *and* **provocative** *realm of digital forensics.*
>
> Hale Ligh et al. — The Art of Memory Forensics (2014)

Memory Forensics - Introduction

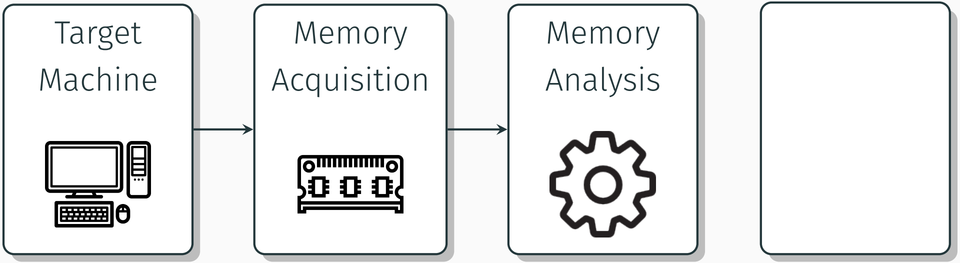
Target Machine



Memory Forensics - Introduction



Memory Forensics - Introduction

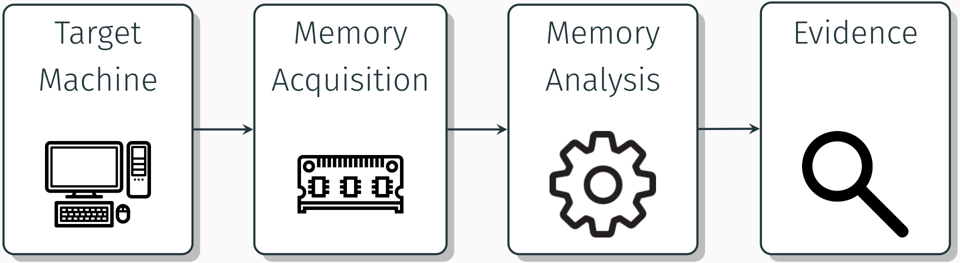


Target Machine

Memory Acquisition

Memory Analysis

Memory Forensics - Introduction



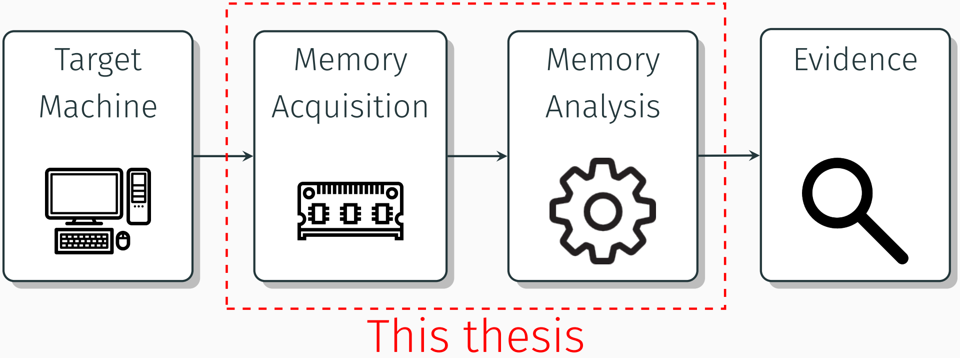
Target Machine

Memory Acquisition

Memory Analysis

Evidence

Memory Forensics - Introduction



Target Machine

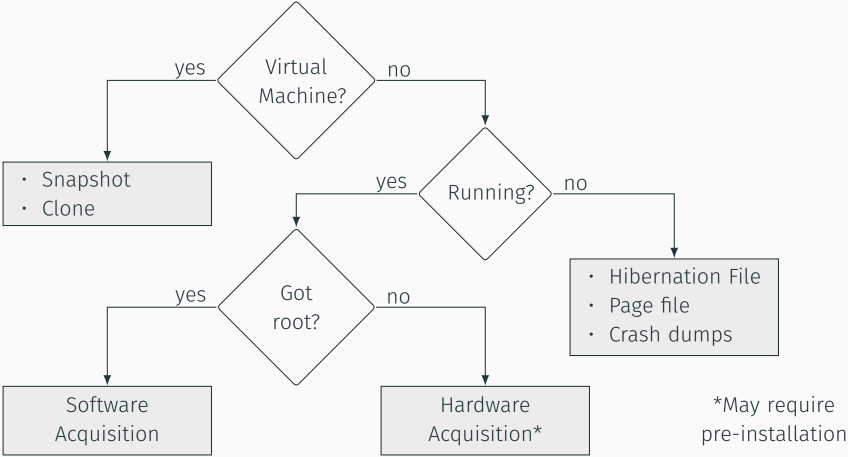
Memory Acquisition

Memory Analysis

Evidence

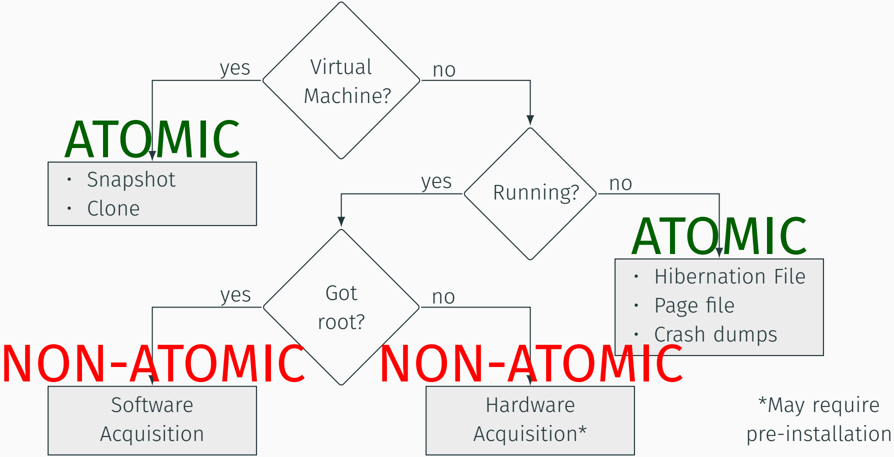
This thesis

Memory Acquisition



Decision tree adapted from The Art of Memory Forensics

Memory Acquisition

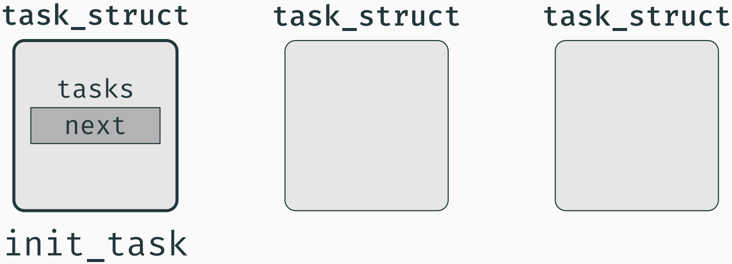


Decision tree adapted from The Art of Memory Forensics

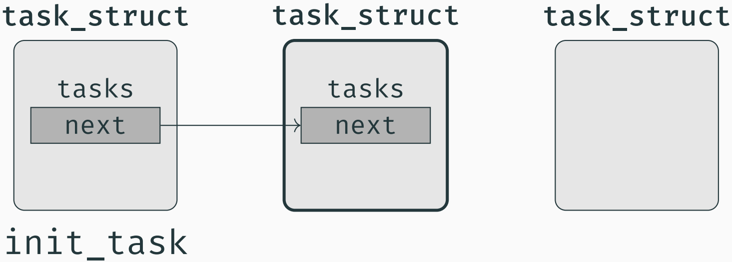
Memory Analysis

- 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
  - Kernel Modules → list, code, unloaded modules
  - Networking → connections, sockets, arp table
  - Windows Registry → keys, password hashes
  - System information → clipboard content, screenshot
- Every analysis task is “organized” in a plugin

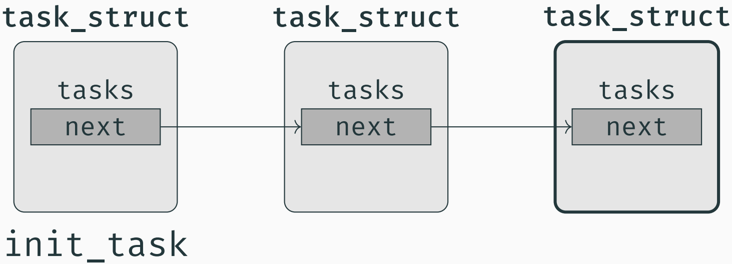
Memory Analysis



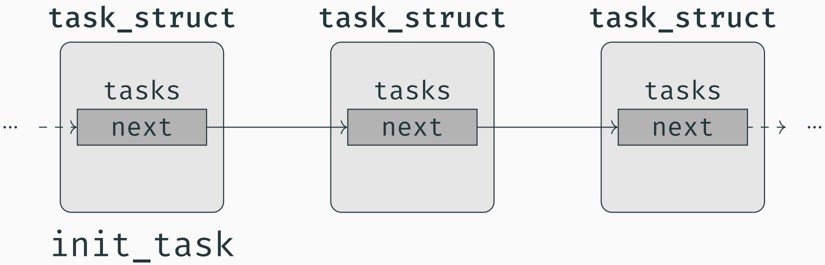
Memory Analysis



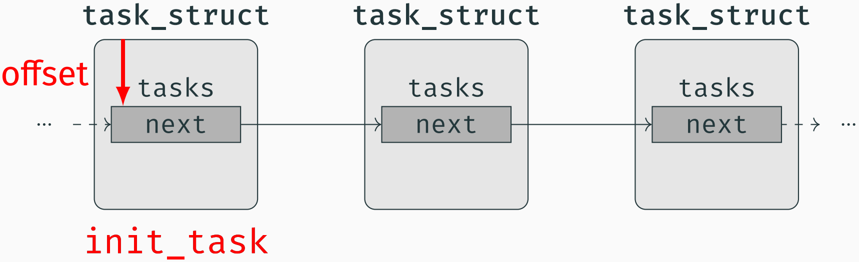
Memory Analysis



Memory Analysis



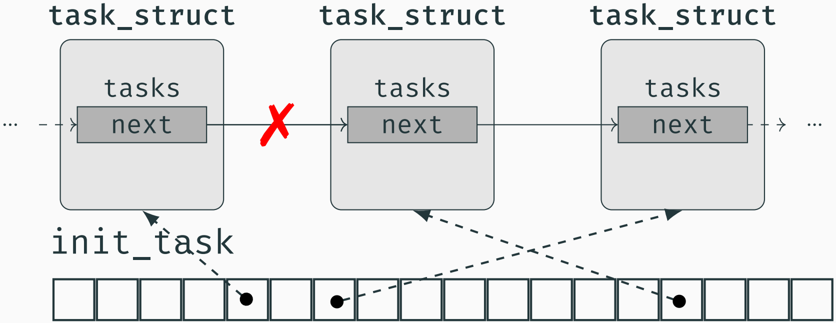
Memory Analysis



Problems

- Bridge the semantic gap (“SOLVED” with a profile)

Memory Analysis



Problems

- Bridge the semantic gap (“SOLVED” with a profile)
- Some data can be inconsistent/tampered (UNSOLVED)

Thesis Contributions

- Unknown effects of non-atomic memory acquisition
    - Introducing the Temporal Dimension to Memory Forensics (TOPS 2019)

Thesis Contributions

- Unknown effects of non-atomic memory acquisition
  - Introducing the Temporal Dimension to Memory Forensics (TOPS 2019)
- A profile is required to analyze a memory dump
  - Towards Automated Profile Generation for Memory Forensics (S&P 2020 - *revise*)

Thesis Contributions

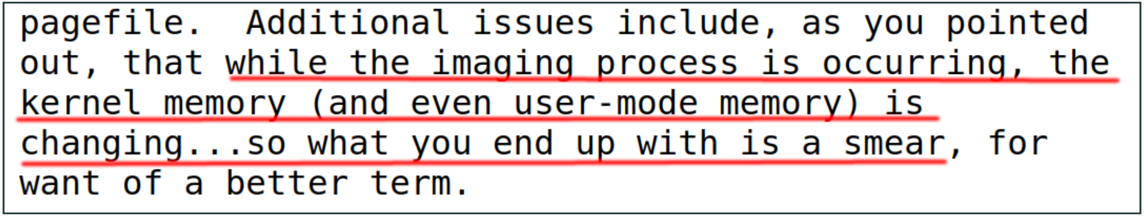
- Unknown effects of non-atomic memory acquisition
    - Introducing the Temporal Dimension to Memory Forensics (TOPS 2019)
- A profile is required to analyze a memory dump
    - Towards Automated Profile Generation for Memory Forensics (S&P 2020 - *revise*)
- Memory forensics heuristics are manually created
    - Back to the Whiteboard: a Principled Approach for the Assessment and Design of Memory Forensic Techniques (USENIX 2019)

Introducing the Temporal Dimension to Memory Forensics (TOPS 2019)

Introduction

- Research in the field has focused on the spatial dimension of memory forensics:
    - Filling the semantic gap
    - Locating and traversing kernel structures
- We propose a second orthogonal dimension, **time**, to study temporal consistency of information

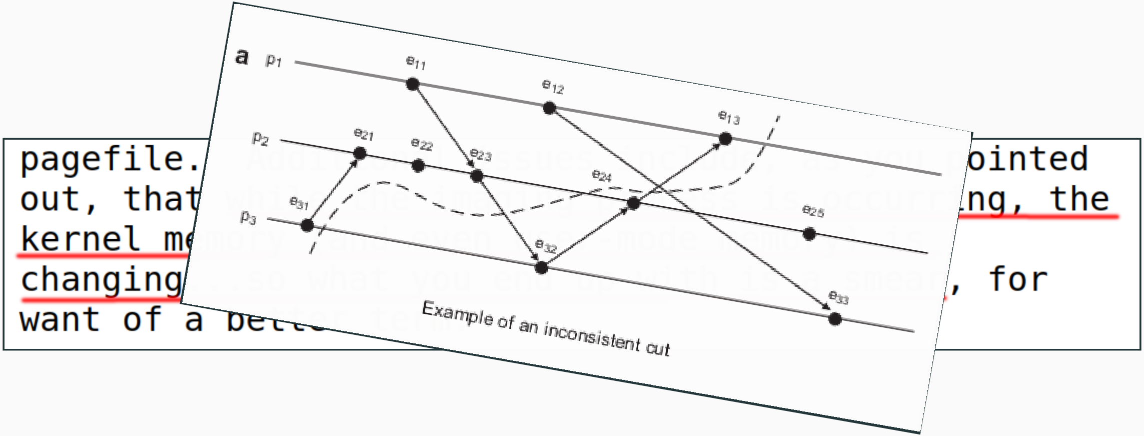
History



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)

History

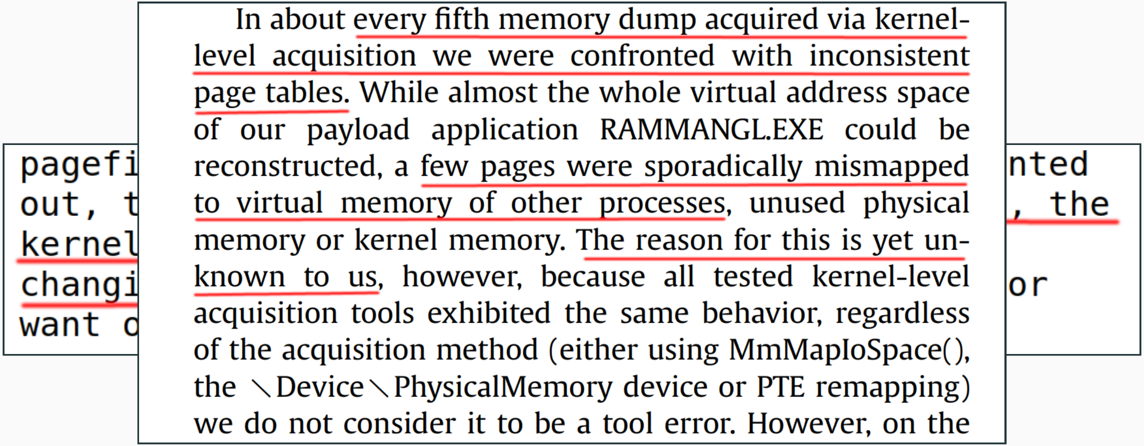


pagefile. Additional issues include, as you pointed out, that while the imaging process is occurring, the kernel memory (and even user-mode memory) is changing...so what you end up with is a smear, for want of a better term.

Example of an inconsistent cut

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

History



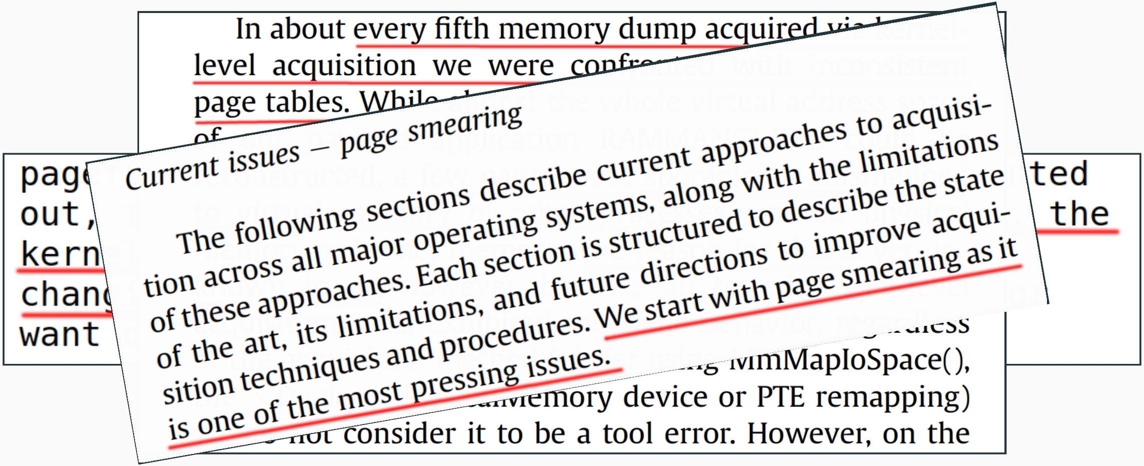
pagefi
out, t
kernel
changi
want o

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

nted
, the
or

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

History



In about every fifth memory dump acquired via kernel-level acquisition we were confronted with inconsistent page tables. While

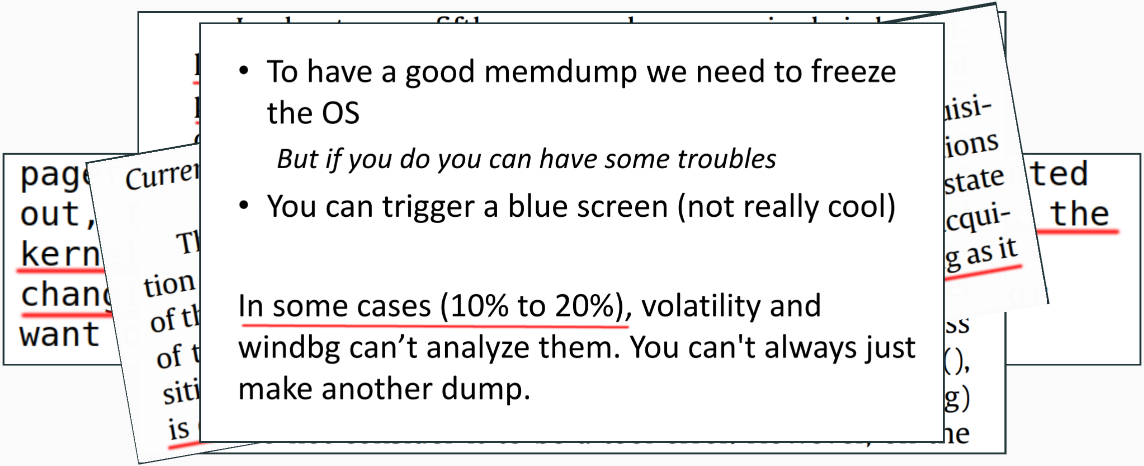
Current issues – page smearing

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.

MmMapIoSpace(), memory device or PTE remapping) not consider it to be a tool error. However, on the

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

History



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

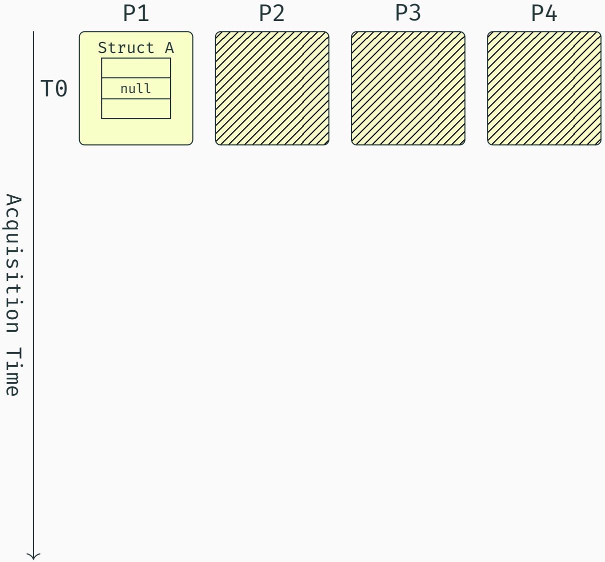
  But if you do you can have some troubles

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

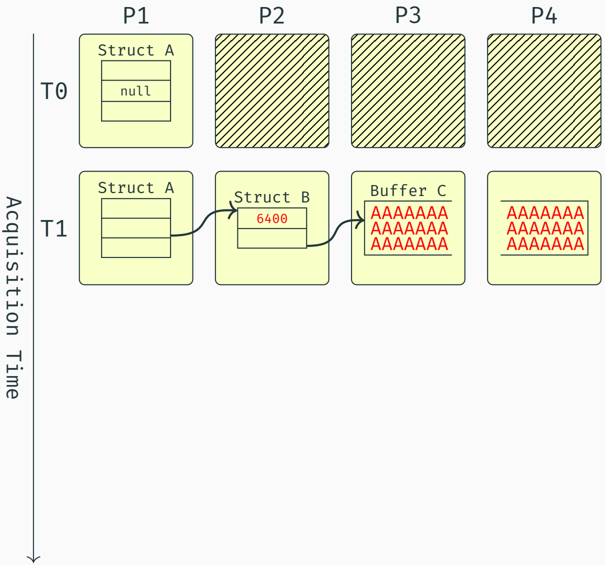
In some cases (10% to 20%), volatility and windbg can’t analyze them. You can't always just make another dump.

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

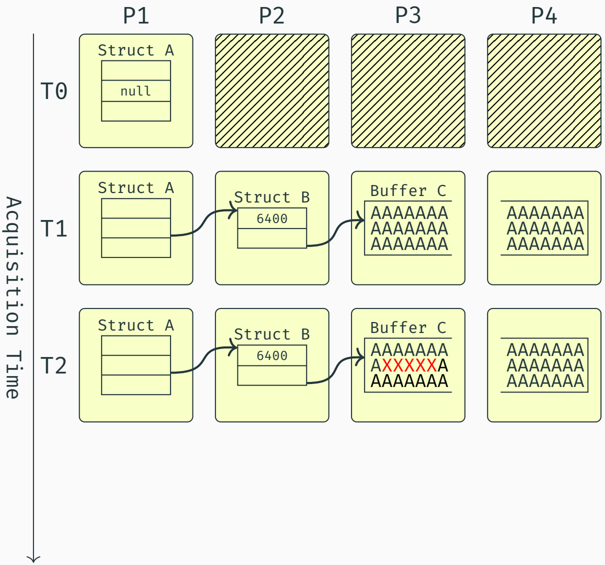
Types of Inconsistency



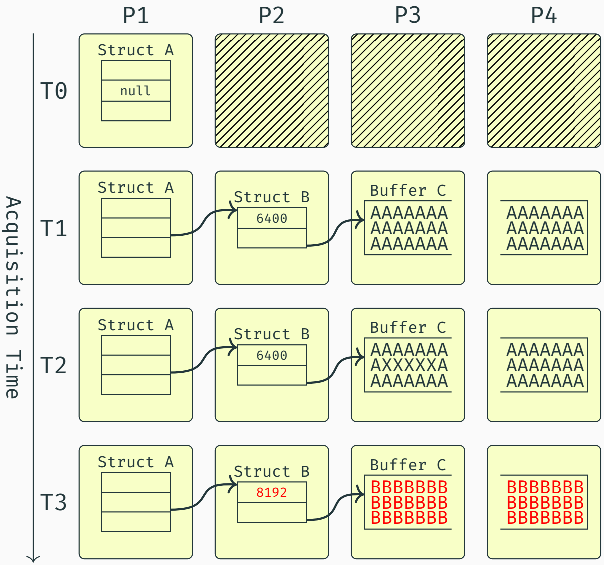
Types of Inconsistency



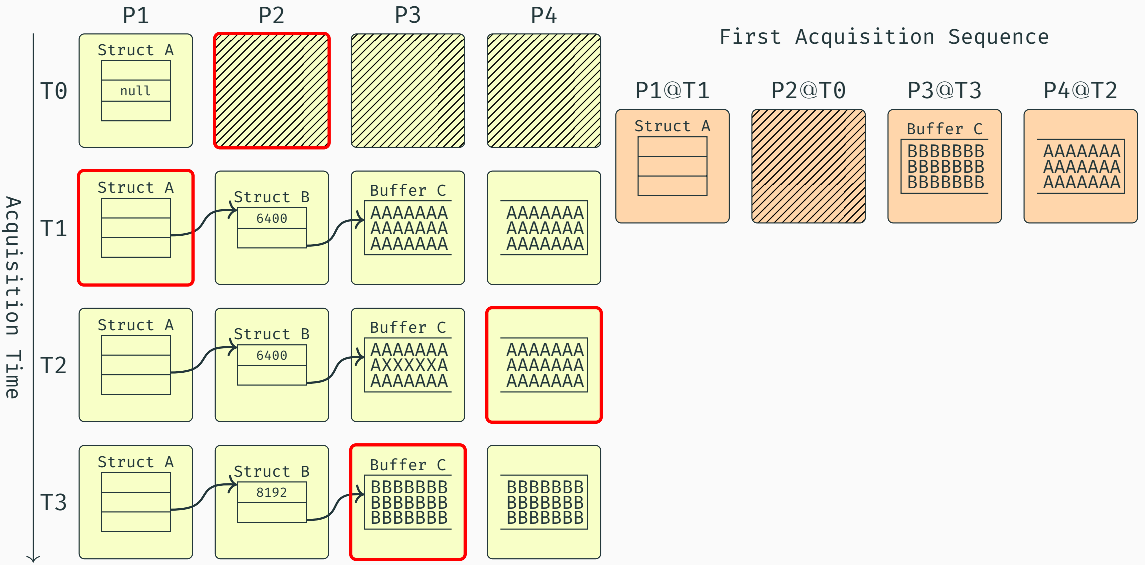
Types of Inconsistency



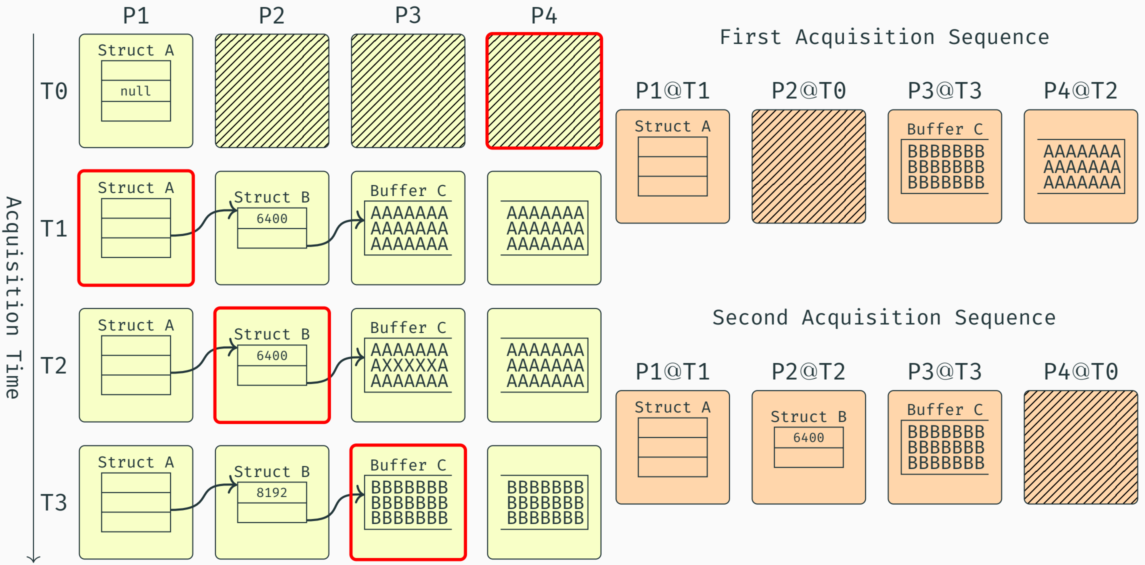
Types of Inconsistency



Types of Inconsistency



Types of Inconsistency



P1 P2 P3 P4

Acquisition Time

T0 Struct A null

T1 Struct A Struct B 6400 Buffer C AAAAAAA AAAAAAA AAAAAAA AAAAAAA AAAAAAA AAAAAAA

T2 Struct A Struct B 6400 Buffer C AAAAAAA AXXXXXA AAAAAAA AAAAAAA AAAAAAA AAAAAAA

T3 Struct A Struct B 8192 Buffer C BBBBBBB BBBBBBB BBBBBBB BBBBBBB BBBBBBB BBBBBBB

First Acquisition Sequence

P1@T1 P2@T0 P3@T3 P4@T2

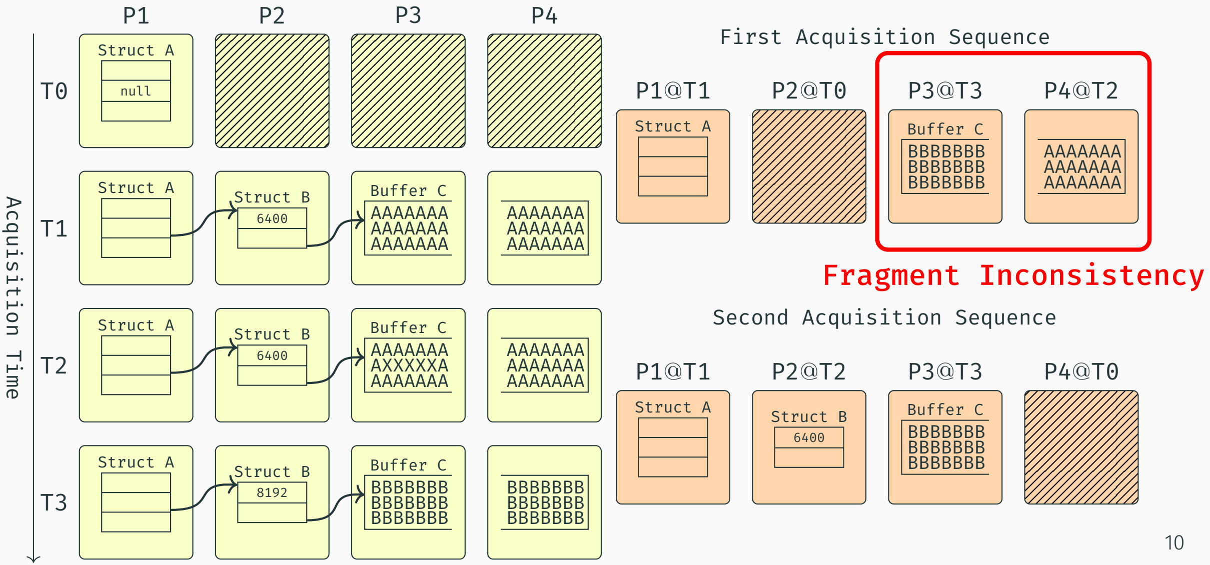
Struct A Buffer C BBBBBBB BBBBBBB BBBBBBB AAAAAAA AAAAAAA AAAAAAA

Second Acquisition Sequence

P1@T1 P2@T2 P3@T3 P4@T0

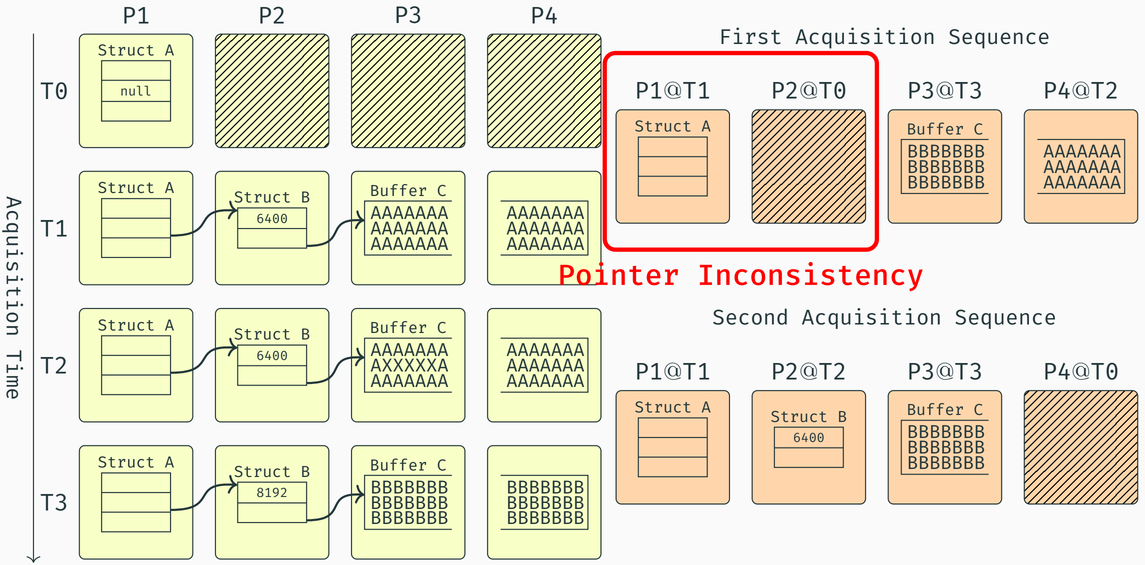
Struct A Struct B 6400 Buffer C BBBBBBB BBBBBBB BBBBBBB

Types of Inconsistency



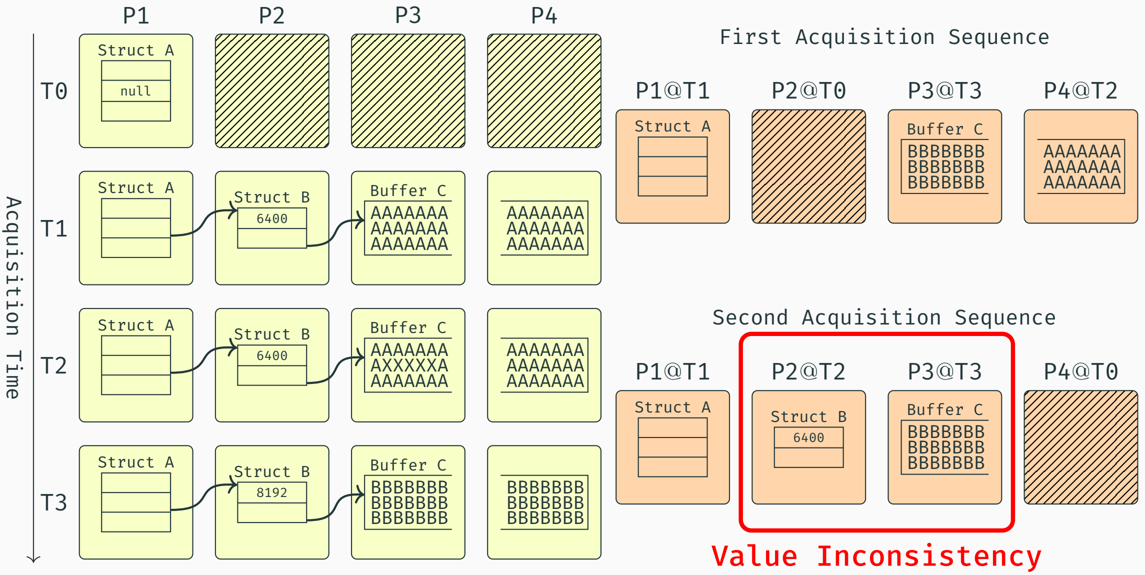
Fragment Inconsistency

Types of Inconsistency

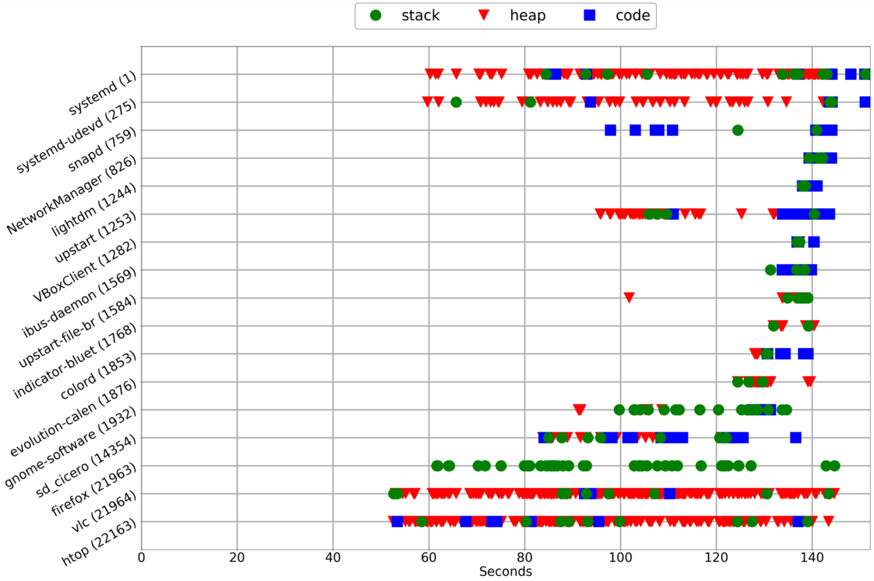


Pointer Inconsistency

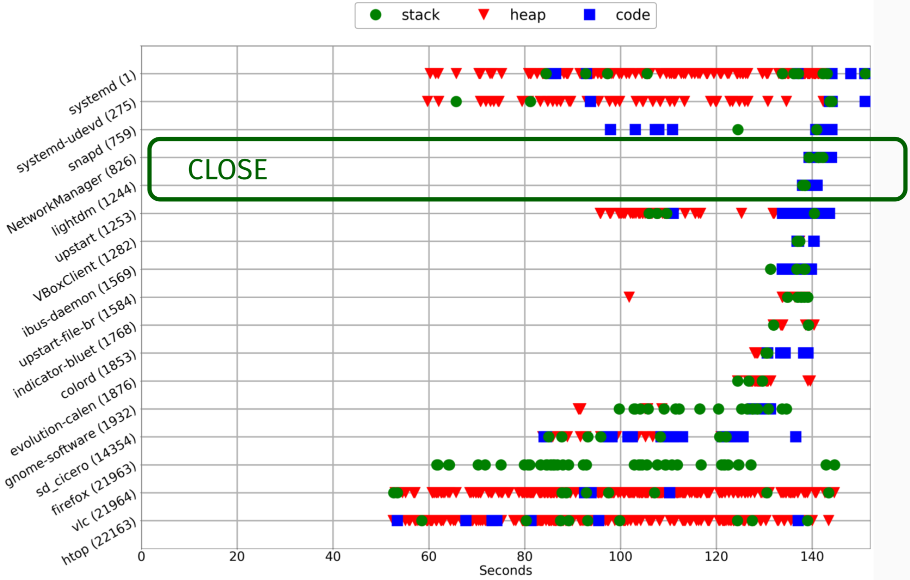
Types of Inconsistency



Impact Estimation - Fragmentation

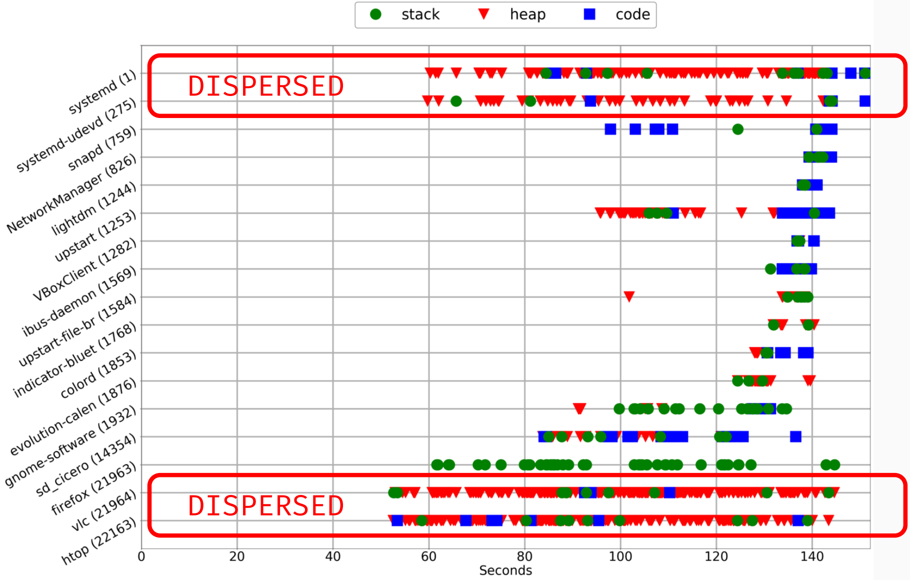


Impact Estimation - Fragmentation

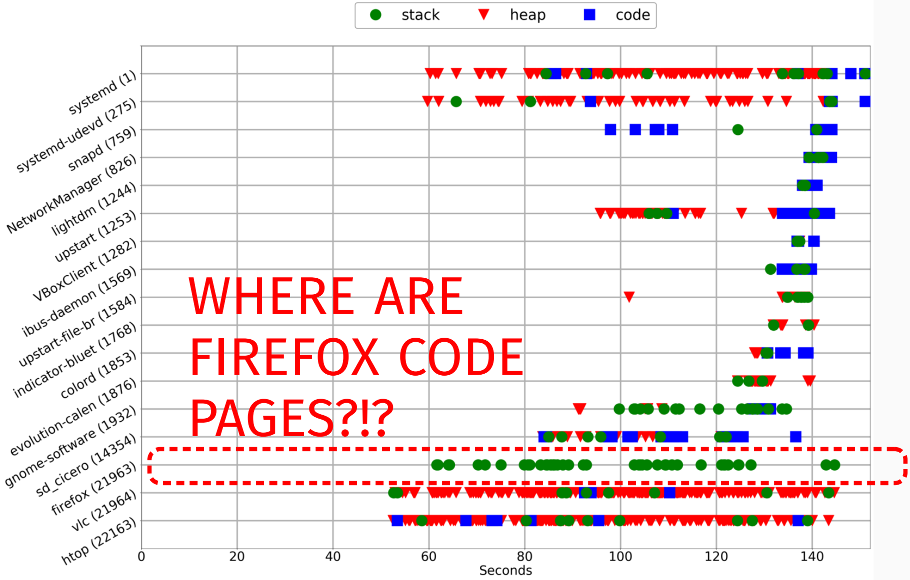


CLOSE

Impact Estimation - Fragmentation

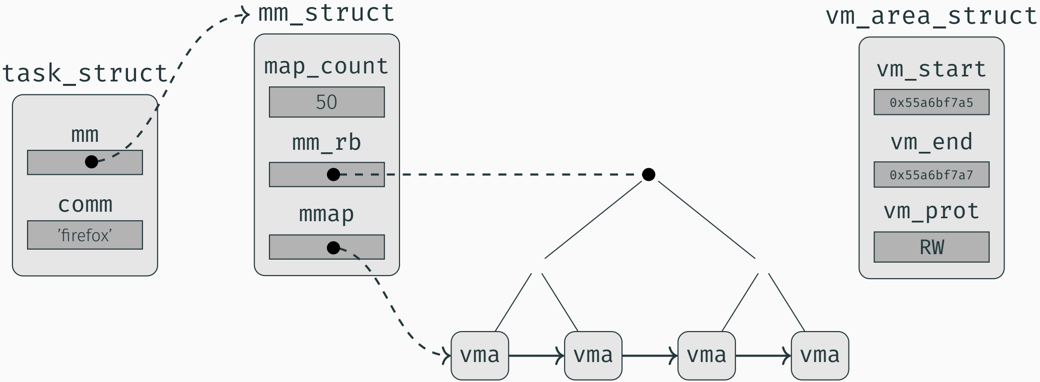


Impact Estimation - Fragmentation

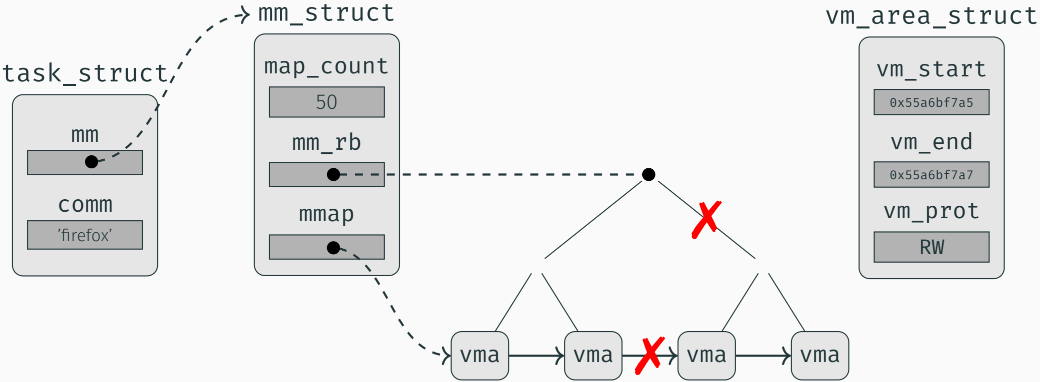


WHERE ARE FIREFOX CODE PAGES?!?

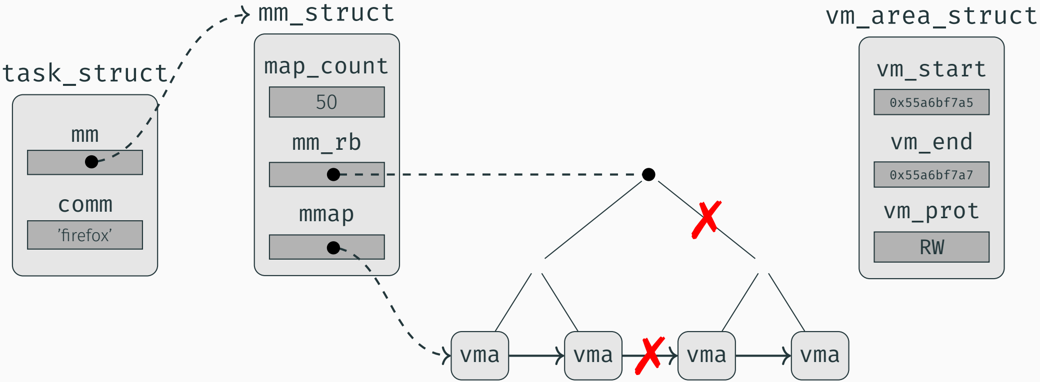
Kernel-Space Integrity



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%

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?

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%**

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%**
- Key recovery for WannaCry and NotPetya

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 100μs $\rightarrow$ 0.7%
    - Every page $\rightarrow$ 2.4%

A New Temporal Dimension - Time Analysis

- 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 they are used by the OS
  - Independently if they contain forensics data
  - From lowest → highest physical address

Locality-Based Acquisition

- Every memory acquisition tool treats pages equally:
  - Independently if they are used by the OS
  - Independently if they contain 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

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!

Conclusions

- We show that inconsistencies do not affect only page tables
- Categorization of inconsistencies: Fragment, Pointer and Value
- Kernel and user-space integrity examples
- Introduced the temporal dimension in memory forensics
- Novel technique to acquire the memory

Towards Automated Profile Generation for Memory Forensics (S&P 2020 - revise)

Introduction

- A profile is needed to overcome the semantic gap
  - Address of kernel global variables
  - Layout of kernel structures
- Building a profile:
  - Easy for Windows: few releases, debug symbols server
  - Not easy for Linux: IoT devices, Android, servers..

Building a profile

Manual effort to create a profile:

- Build a kernel module → Layout of kernel structures
- Grab `System.map` → Address of kernel global variables

Building a profile

Manual effort to create a profile:

- Build a kernel module → Layout of kernel structures
- Grab `System.map` → Address of kernel global variables

Requirements

- Kernel headers + config
- RANDSTRUCT seed (if enabled)

Why we need the kernel config?

```
struct task_struct {
   #ifdef CONFIG_THREAD_INFO_IN_TASK
     struct thread_info  thread_info;
   #endif
     volatile long   state;
     randomized_struct_fields_start
     unsigned int   ptrace;
   #ifdef CONFIG_SMP
     struct llist_node  wake_entry;
     int    on_cpu;
  #ifdef CONFIG_THREAD_INFO_IN_TASK
     unsigned int   cpu;
   #endif
     unsigned int   wakee_flips;
     struct task_struct  *last_wakee;
     int    wake_cpu;
   #endif
     int    on_rq;
   #ifdef CONFIG_CGROUP_SCHED
     struct task_group
         *sched_task_group;
   #endif
      struct sched_dl_entity  dl;
      ....
    }
```

Why we need the kernel config?

```
struct task_struct {
   #ifdef CONFIG_THREAD_INFO_IN_TASK
     struct thread_info  thread_info;
   #endif
     volatile long   state;
     randomized_struct_fields_start
     unsigned int   ptrace;
   #ifdef CONFIG_SMP
     struct llist_node  wake_entry;
     int    on_cpu;
  #ifdef CONFIG_THREAD_INFO_IN_TASK
     unsigned int   cpu;
   #endif
     unsigned int   wakee_flips;
     struct task_struct  *last_wakee;
     int    wake_cpu;
   #endif
     int    on_rq;
   #ifdef CONFIG_CGROUP_SCHED
     struct task_group
         *sched_task_group;
   #endif
      struct sched_dl_entity  dl;
      ....
    }
```

More than 60 `#ifdef` !!

Challenge

Can we reconstruct a profile from a memory dump?

Challenge

Can we reconstruct a profile from a memory dump?

- Phase I: Symbols Recovery + Kernel Version Identification
  - Several past attempts: **ALL** fail on modern `X86_64` platforms with KASLR

Challenge

Can we reconstruct a profile from a memory dump?

- Phase I: Symbols Recovery + Kernel Version Identification
  - Several past attempts: **ALL** fail on modern `X86_64` platforms with KASLR
- Phase II: Source Code Analysis
- Phase III: Profile Generation

Phase I: Symbols Recovery

Problem

Kernel symbols are stored in a compressed form ← not easy to carve!

Phase I: Symbols Recovery

Problem

Kernel symbols are stored in a compressed form ← not easy to carve!

Solution

We locate, extract and execute:

```
/* Call a function on each kallsyms symbol in the core kernel */
int kallsyms_on_each_symbol(int (*fn)(void *, const char *,
                                      struct module *, unsigned long),
                                      void *data);
```

Phase II: Source Code Analysis - Overview

- Download and compile the kernel

Phase II: Source Code Analysis - Overview

- Download and compile the kernel
- Pre-processor activity:
  - Position of `#ifdef` and `macro` statements
- Abstract Syntax Tree analysis
  - Type Definition (`struct foo {...}`) → Fields Position
  - Function Definition (`free_next(task *){...}`) → Access Chains

Phase II: Source Code Analysis - Access Chains

```c
void free_next(struct task *task){
  struct task *t = task->next;
  if (strcmp(t->name, "init")){
    free(t);
  }
}
```

Phase II: Source Code Analysis - Access Chains

```
1   void free_next(struct task *task){
2     struct task *t = task->next;
3     if (strcmp(t->name, "init")){
4       free(t);
5     }
6   }
```

Access chains are triples:

- Location → `free_next`:3
- Transition → `struct task->next|struct task->name`
- Source → `PARAM[0]`

  (other valid sources: global variable, function return)

Phase III: Profile Generation

We have all the ingredients we need:

- Binary code of kernel functions
- Where and how struct fields are used
  (minus those accesses contained in an `#ifdef`)

Phase III: Profile Generation

We have all the ingredients we need:

- Binary code of kernel functions
- Where and how struct fields are used
  (minus those accesses contained in an `#ifdef`)

Given a field, to extract its offset:

- Load kernel functions where the field is used in `angr`

Phase III: Profile Generation

We have all the ingredients we need:

- Binary code of kernel functions
- Where and how struct fields are used
  (minus those accesses contained in an `#ifdef`)

Given a field, to extract its offset:

- Load kernel functions where the field is used in `angr`
- Taint source and symbolically explore the function

Phase III: Profile Generation

We have all the ingredients we need:

- Binary code of kernel functions
- Where and how struct fields are used
  (minus those accesses contained in an `#ifdef`)

Given a field, to extract its offset:

- Load kernel functions where the field is used in `angr`
- Taint source and symbolically explore the function
- "Breakpoint" on memory accesses ← **list of candidate offsets!**

Phase III: Profile Generation

To find the correct offset of a field, we create a `z3` model for every structure:

Phase III: Profile Generation

To find the correct offset of a field, we create a `z3` model for every structure:

- Hard constraints:

```
OffsetField1 < OffsetField2
OffsetField2 < OffsetField3
```

- Soft constraints:

```
OffsetField1 == {0, 10, 20}
OffsetField2 == {20, 50}
OffsetField3 == {20, 60}
```

Results

Version	Release Date	Configuration	Used Fields	Extracted Fields
4.19.37	04/2019	`Debian`	230	205 (89%)
4.19.37	04/2019	`Debian` + `RANDSTRUCT`	230	172 (74%)
4.4.71	06/2017	`OpenWrt`	231	198 (86%)
3.18.94	05/2018	`Goldfish` (`Android`)	236	204 (86%)
2.6.38	03/2011	`Ubuntu`	220	191 (87%)

Results

	Debian 4.19			RANDSTRUCT			Openwrt			Android			Ubuntu 2.6		
	Working	Correct	Wrong	Working	Correct	Wrong	Working	Correct	Wrong	Working	Correct	Wrong	Working	Correct	Wrong
`linux_arp`	◑	10	2	○	6	6	◑	10	2	◑	11	1	●	12	0
`linux_banner`	●	0	0	●	0	0	●	0	0	●	0	0	●	0	0
`linux_check_afinfo`	—	5	1	—	5	1	●	37	3	●	40	2	○	34	5
`linux_check_creds`	●	9	0	●	9	0	●	9	0	●	9	0	●	9	0
`linux_check_fop`	○	77	4	○	65	16	○	75	4	○	76	2	○	67	3
`linux_check_modules`	○	17	1	○	14	4	○	15	2	○	17	0	○	15	2
`linux_check_syscall`	●	36	1	●	32	5	◑	31	5	●	33	3	●	32	3
`linux_check_tty`	○	11	3	○	8	6	○	9	4	○	9	4	○	11	2
`linux_cpuinfo`	○	0	2	○	0	2	○	0	2	○	0	2	●	2	0
`linux_dump_map`	●	10	0	○	6	4	●	10	0	●	10	0	●	9	1
`linux_dynamic_env`	●	29	0	●	23	6	●	6	0	●	29	0	●	27	1
`linux_elfs`	●	26	0	○	20	6	●	25	0	●	26	0	●	23	4
`linux_lsof`	●	24	0	◑	22	2	●	24	0	●	24	0	●	23	0
`linux_malfind`	●	17	0	○	16	1	○	16	1	○	17	0	○	16	1
`linux_mount`	●	20	0	○	18	2	●	20	0	●	20	0	●	19	0
`linux_netscan`	●	16	1	●	16	1	●	15	2	●	15	2	○	14	3
`linux_proc_maps`	●	37	0	○	30	7	◑	36	1	●	37	0	◑	34	1
`linux_psaux`	●	13	0	●	12	1	●	13	0	○	12	1	●	11	0
`linux_psenv`	●	8	0	●	8	0	●	8	0	●	8	0	●	8	0
`linux_pslist`	◑	17	1	◑	16	2	●	19	0	◑	16	3	◑	12	1
`linux_psscan`	●	12	1	●	11	2	●	13	0	●	13	0	●	12	1
`linux_pstree`	●	13	0	●	11	2	○	11	2	○	11	2	●	11	0
`linux_threads`	●	6	0	●	6	0	●	6	0	●	6	0	●	6	0
`linux_tmpfs -L`	●	20	0	○	18	2	●	20	0	●	20	0	●	19	0
`linux_truecrypt_passphrase`	●	3	0	●	3	0	●	3	0	●	3	0	●	3	0

Results

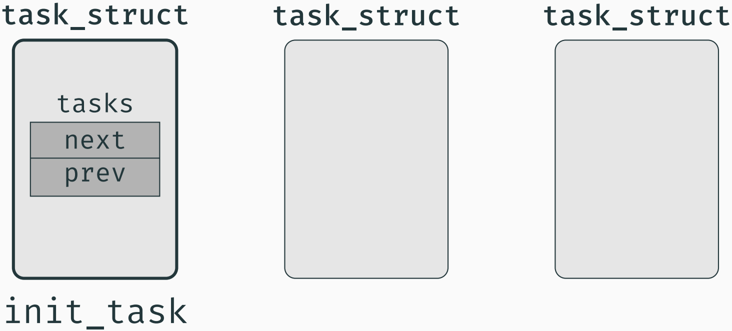
- On non randomized memory dumps: 57% to 64% of plugins work correctly
- Hard constraints play an important role → only 35% of plugins works when `RANDSTRUCT`
- For the 41% of missing fields, there are models with 2 or 3 offsets

Conclusions

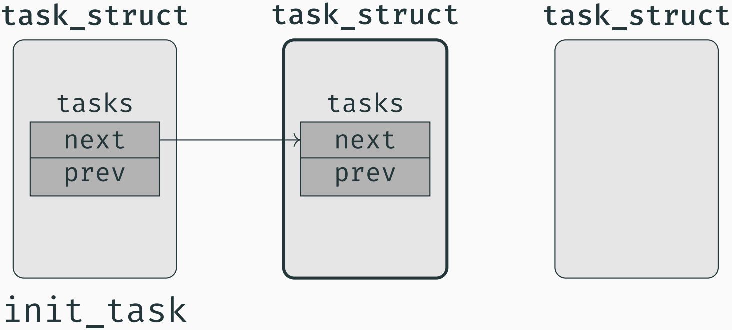
- Creating a profile for Linux is manual, error prone and not always possible
- Three phases to reconstruct a profile from a memory dump:
  - Phase I: Symbols Recovery + Kernel Version Identification
  - Phase II: Source Code Analysis
  - Phase III: Profile Generation
- The extracted profile supports many fundamental forensics plugins

Back to the Whiteboard: a Principled Approach for the Assessment and Design of Memory Forensic Techniques (Usenix 2019)

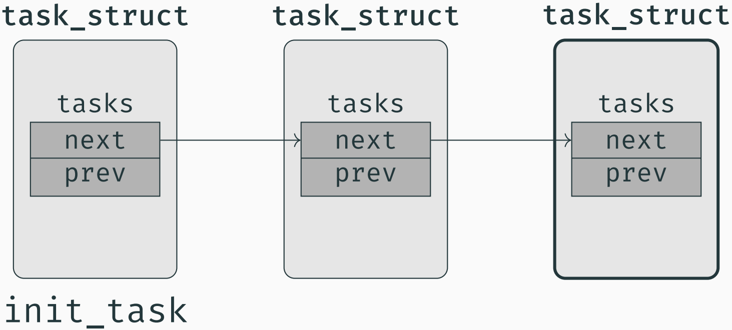
Memory Forensics - Listing Processes



Memory Forensics - Listing Processes

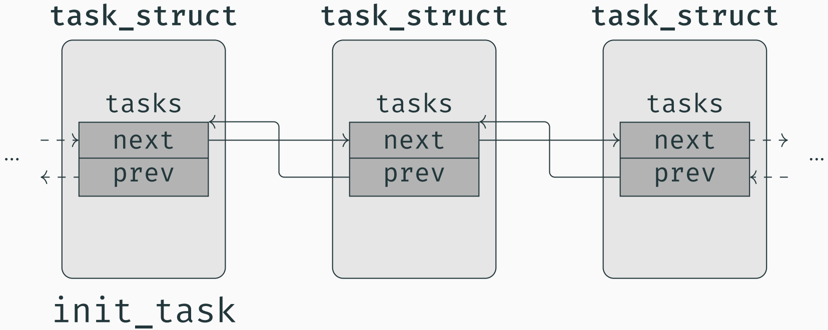


Memory Forensics - Listing Processes

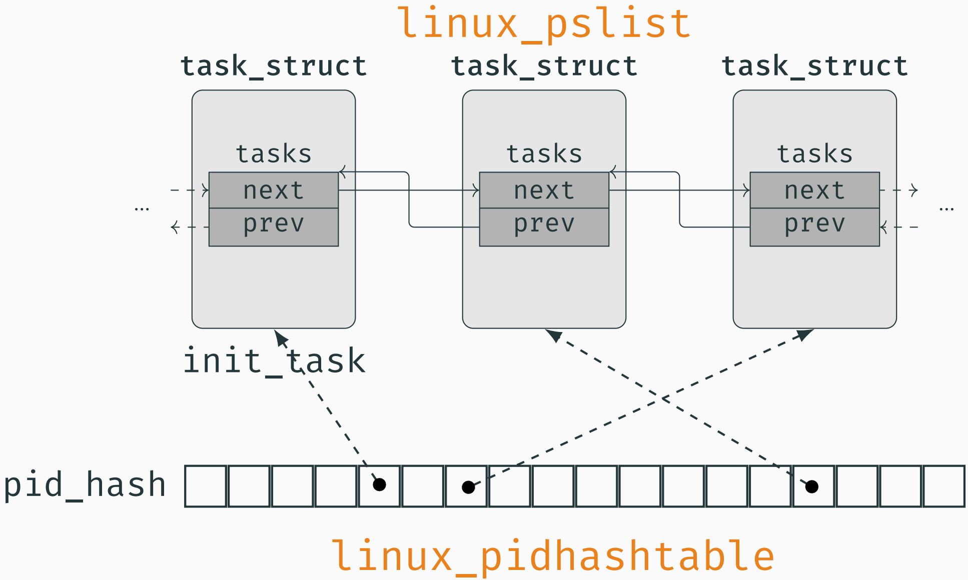


Memory Forensics - Listing Processes

`linux_pslist`



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

Motivations

Forensic analyses are manually created by humans.

- Are there other techniques to list processes?

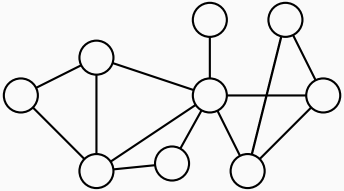
  Linux kernel 4.19: ~6000 structures with ~40000 fields

- How can we compare them?

  Shortest one? Most stable across different kernels?

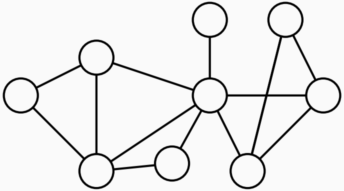
Contributions

Build a graph of kernel structures

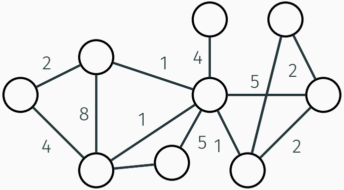


Contributions

Build a graph of kernel structures

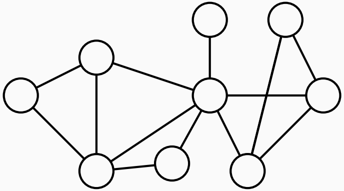


Define metrics to evaluate analyses

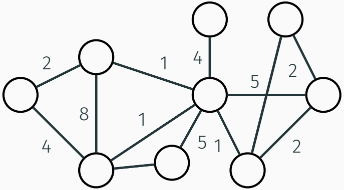


Contributions

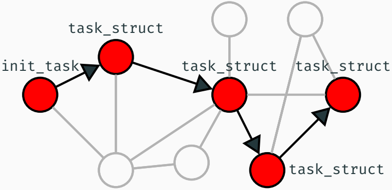
Build a graph of kernel structures



Define metrics to evaluate analyses



Study analyses as paths on the graph



Kernel Graph - Creation

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

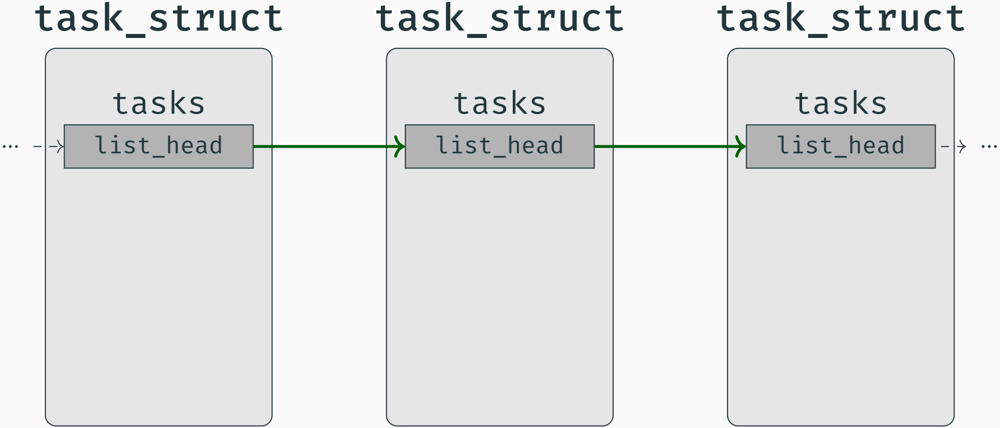
Kernel Graph - Creation

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

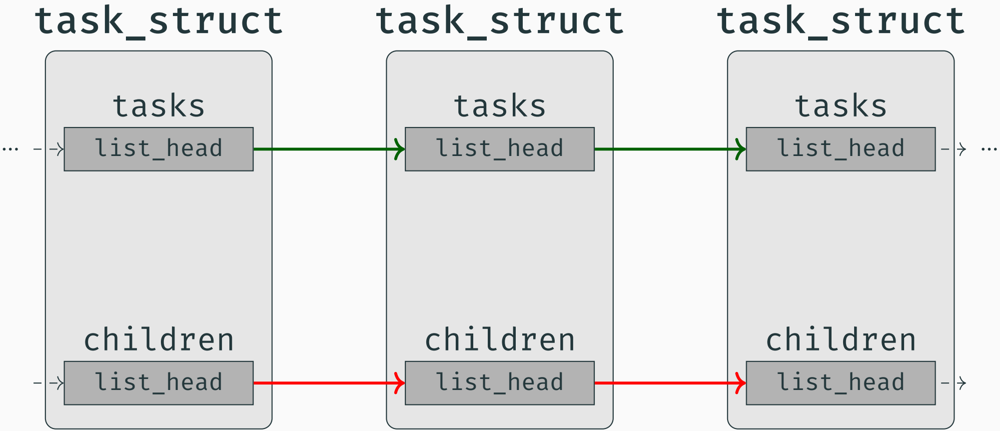
Challenge

Kernel “abstract data types”

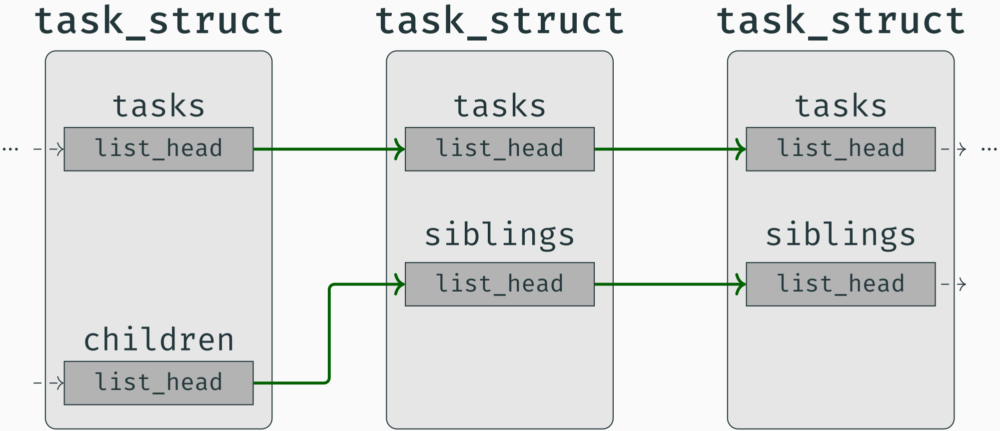
Kernel Graph - ADT Challenge



Kernel Graph - ADT Challenge



Kernel Graph - ADT Challenge



Kernel Graph - ADT Challenge

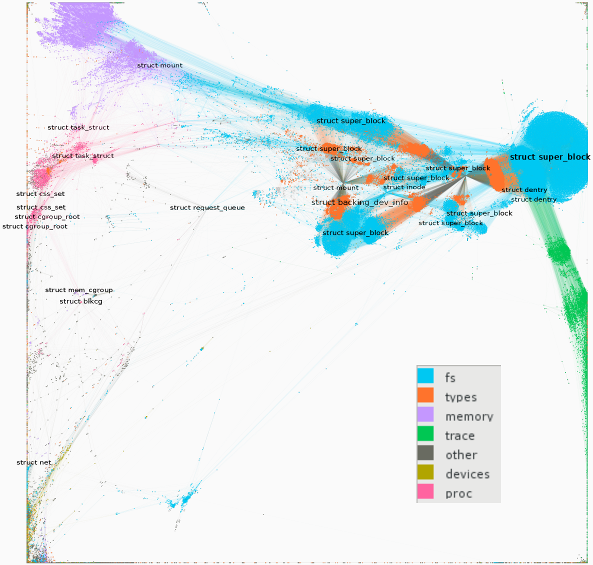
Solved with a Clang plugin that analyzes the kernel AST

```
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 - Rationale

Metrics should capture different *aspects* of memory forensics:

Metrics - Rationale

Metrics should capture different *aspects* of memory forensics:

- Data can be inconsistent in non-atomic memory dumps

Metrics - Rationale

Metrics should capture different *aspects* of memory forensics:

- Data can be inconsistent in non-atomic memory dumps
- Layout of kernel structures changes across different kernel versions and configurations

Metrics - Rationale

Metrics should capture different *aspects* of memory forensics:

- Data can be inconsistent in non-atomic memory dumps
- Layout of kernel structures changes across different kernel versions and configurations
- Attackers can modify kernel structures

Proposed Metrics

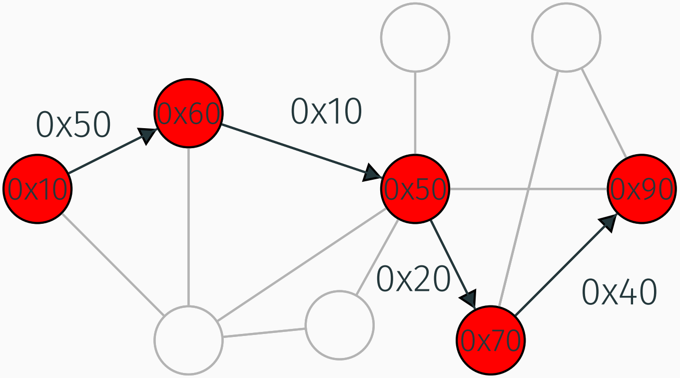
- Atomicity
- Stability
- Consistency
- Generality
- Reliability

Proposed Metrics

- Atomicity
- Stability
- Consistency
- Generality
- Reliability

Metrics

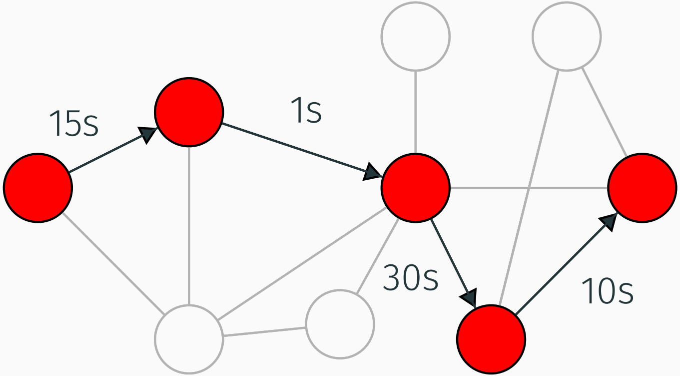
Atomicity: distance in memory between two connected structures



Metrics

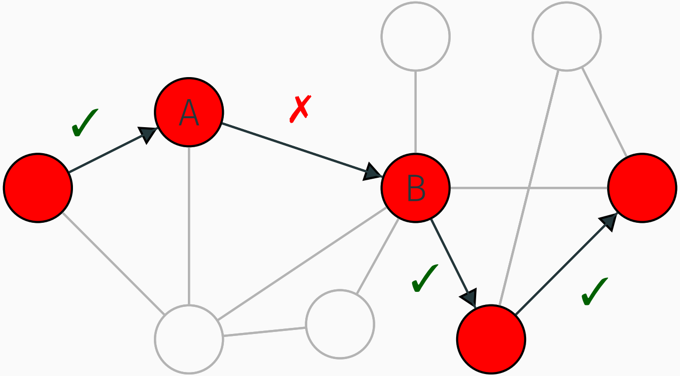
Stability: how long an edge remains stable in a running machine

- 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 → 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

Kernel Graph - New Heuristics Results

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!

A Principled Approach to Memory Forensics

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

Conclusions

Conclusions

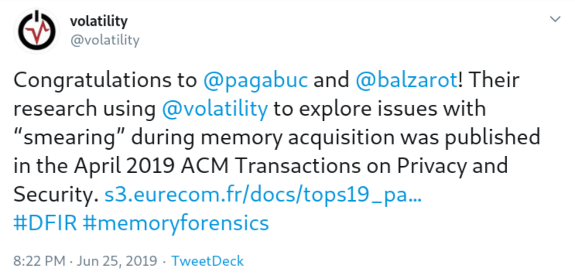
- Interest in memory forensics is growing in industry
- Hopefully academia will follow ;-)
- Thesis contributions:
  - Documented effects of non atomic memory acquisition and proposed solutions
  - Showed how to reconstruct a profile from a memory dump
  - Built a framework to study forensics techniques in a principled way

Open source

All the code and artifacts developed during this thesis are open-source!

- `https://github.com/pagabuc/atomicity_tops`
- `https://github.com/pagabuc/kernographer`

Volatility Tweets



volatility
@volatility

Congratulations to @pagabuc and @balzarot! Their research using @volatility to explore issues with “smearing” during memory acquisition was published in the April 2019 ACM Transactions on Privacy and Security. s3.eurecom.fr/docs/tops19_pa… #DFIR #memoryforensics

8:22 PM · Jun 25, 2019 · TweetDeck

volatility
@volatility

After 13 years, it’s amazing to see all the interesting academic and industry research still being built on @Volatility! Congrats to @pagabuc and @balzarot. We are excited to see the new plugins. #DFIR #memoryforensics #VolPluginContest bit.ly/2HGXeeV

6:44 PM · May 21, 2019 · TweetDeck

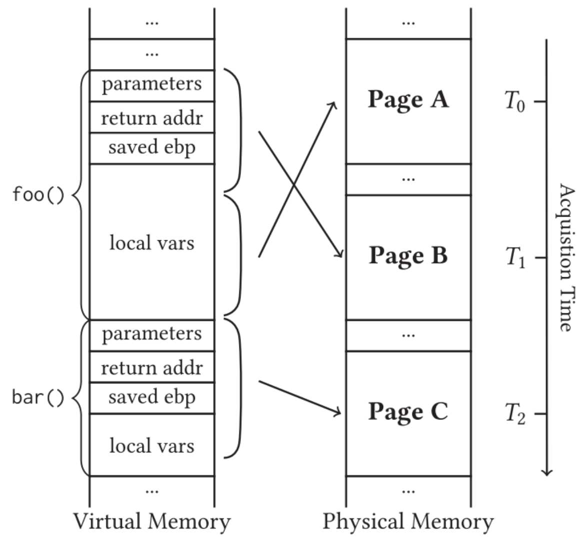
Questions?

THE BEST THESIS DEFENSE IS A GOOD THESIS OFFENSE.

Backup Slides

Backup Slides - Introducing the Temporal Dimension to Memory Forensics (TOPS 2019)

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)

Backup Slides - Towards Automated Profile Generation for Memory Forensics (S&P 2020 - revise)

Why we need the kernel config?

```
struct creds {
  uint32_t uid;
  uint32_t gid;
};

struct task {
  struct task *next;
  struct creds cred;
#ifdef CONFIG_TIME
  uint64_t start_time;
#endif
  char *name;
};

void setup_task(struct task *t,
                char *new_name,
                int gid){
  t->name = new_name;
  t->cred.gid = gid;
#ifdef CONFIG_TIME
  t->start_time = time(NULL);
#endif
}
```

Why we need the kernel config?

```
struct creds {
  uint32_t uid;
  uint32_t gid;
};

struct task {
  struct task *next;
  struct creds cred;
#ifdef CONFIG_TIME
  uint64_t start_time;
#endif
  char *name;
};

void setup_task(struct task *t,
                char *new_name,
                int gid){
  t->name = new_name;
  t->cred.gid = gid;
#ifdef CONFIG_TIME
  t->start_time = time(NULL);
#endif
}
```

① `CONFIG_TIME defined`

```
push    rbx
mov     rbx,rdi
mov     QWORD PTR [rdi+0x18],rsi
mov     DWORD PTR [rdi+0xc],edx
xor     edi,edi
call    0x1030 <time@plt>
mov     QWORD PTR [rbx+0x10],rax
pop     rbx
ret
```

② `CONFIG_TIME not defined`

```
mov     QWORD PTR [rdi+0x10],rsi
mov     DWORD PTR [rdi+0xc],edx
ret
```

Why we need the kernel config?

```
struct creds {
  uint32_t uid;
  uint32_t gid;
};

struct task {
  struct task *next;
  struct creds cred;
#ifdef CONFIG_TIME
  uint64_t start_time;
#endif
  char *name;
};

void setup_task(struct task *t,
                char *new_name,
                int gid){
  t->name = new_name;
  t->cred.gid = gid;
#ifdef CONFIG_TIME
  t->start_time = time(NULL);
#endif
}
```

① CONFIG_TIME defined

```
push    rbx
mov     rbx,rdi
mov     QWORD PTR [rdi+0x18],rsi
mov     DWORD PTR [rdi+0xc],edx
xor     edi,edi
call    0x1030 <time@plt>
mov     QWORD PTR [rbx+0x10],rax
pop     rbx
ret
```

② CONFIG_TIME not defined

```
mov     QWORD PTR [rdi+0x10],rsi
mov     DWORD PTR [rdi+0xc],edx
ret
```

Why we need the kernel config?

```
struct creds {
  uint32_t uid;
  uint32_t gid;
};

struct task {
  struct task *next;
  struct creds cred;
#ifdef CONFIG_TIME
  uint64_t start_time;
#endif
  char *name;
};

void setup_task(struct task *t,
                char *new_name,
                int gid){
  t->name = new_name;
  t->cred.gid = gid;
#ifdef CONFIG_TIME
  t->start_time = time(NULL);
#endif
}
```

① CONFIG_TIME defined

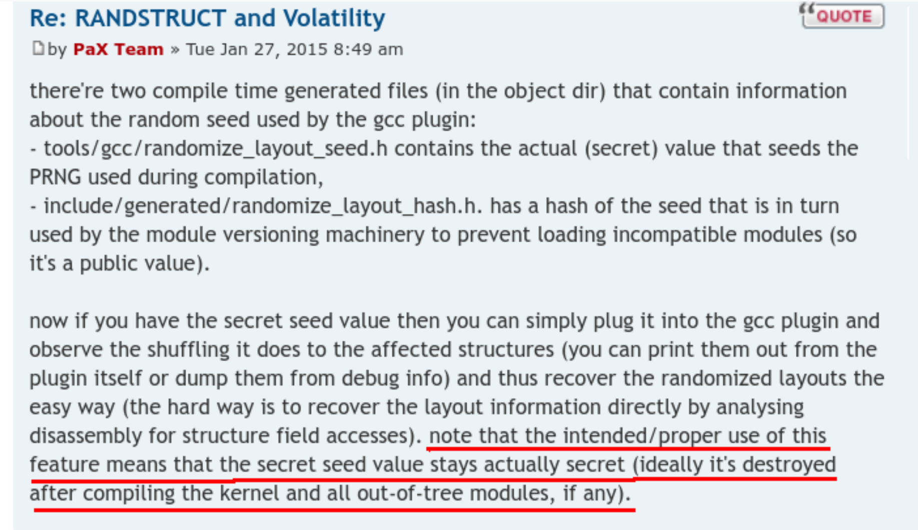
```
push    rbx
mov     rbx,rdi
mov     QWORD PTR [rdi+0x18],rsi
mov     DWORD PTR [rdi+0xc],edx
xor     edi,edi
call    0x1030 <time@plt>
mov     QWORD PTR [rbx+0x10],rax
pop     rbx
ret
```

---

② CONFIG_TIME not defined

```
mov     QWORD PTR [rdi+0x10],rsi
mov     DWORD PTR [rdi+0xc],edx
ret
```

Why we need the RANDSTRUCT seed?



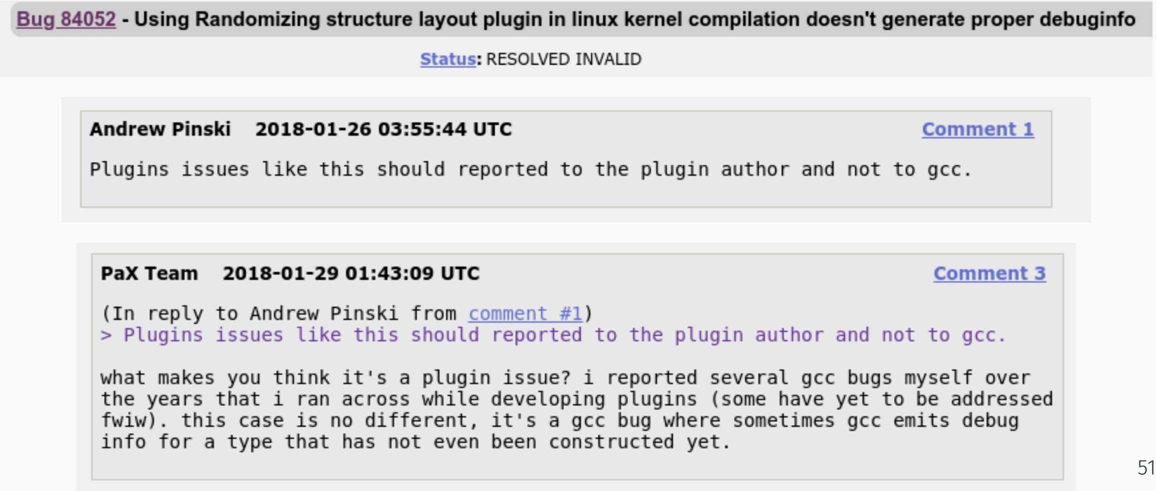
Re: RANDSTRUCT and Volatility

by **PaX Team** » Tue Jan 27, 2015 8:49 am

there're two compile time generated files (in the object dir) that contain information about the random seed used by the gcc plugin:
- tools/gcc/randomize_layout_seed.h contains the actual (secret) value that seeds the PRNG used during compilation,
- include/generated/randomize_layout_hash.h. has a hash of the seed that is in turn used by the module versioning machinery to prevent loading incompatible modules (so it's a public value).

now if you have the secret seed value then you can simply plug it into the gcc plugin and observe the shuffling it does to the affected structures (you can print them out from the plugin itself or dump them from debug info) and thus recover the randomized layouts the easy way (the hard way is to recover the layout information directly by analysing disassembly for structure field accesses). note that the intended/proper use of this feature means that the secret seed value stays actually secret (ideally it's destroyed after compiling the kernel and all out-of-tree modules, if any).

Why we need the RANDSTRUCT seed?



Bug 84052 - Using Randomizing structure layout plugin in linux kernel compilation doesn't generate proper debuginfo

Status: RESOLVED INVALID

Andrew Pinski **2018-01-26 03:55:44 UTC** **Comment 1**

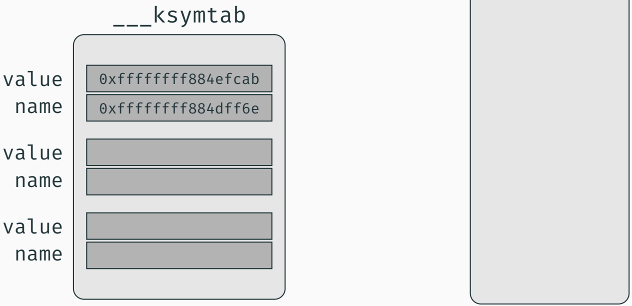
```
Plugins issues like this should reported to the plugin author and not to gcc.
```

PaX Team **2018-01-29 01:43:09 UTC** **Comment 3**

```
(In reply to Andrew Pinski from comment #1)
> Plugins issues like this should reported to the plugin author and not to gcc.

what makes you think it's a plugin issue? i reported several gcc bugs myself over
the years that i ran across while developing plugins (some have yet to be addressed
fwiw). this case is no different, it's a gcc bug where sometimes gcc emits debug
info for a type that has not even been constructed yet.
```

Phase I: Symbols Recovery

```
EXPORT_SYMBOL(``foo'');

struct kernel_symbol {
  unsigned long value;
  const char *name;
};
```



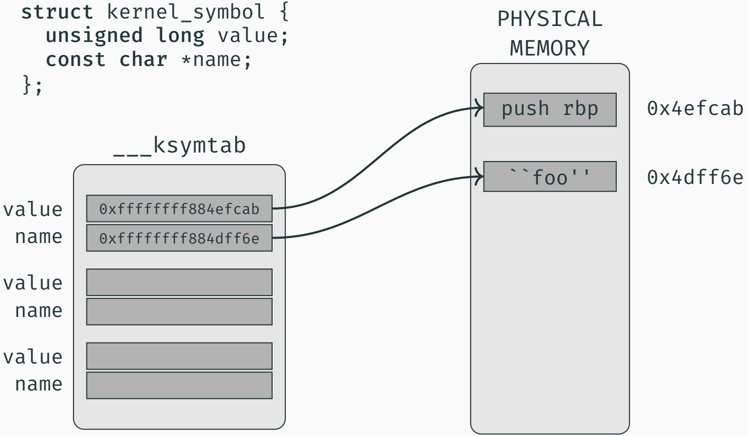
___ksymtab

value 0xffffffff884efcab
name 0xffffffff884dff6e
value
name
value
name

PHYSICAL
MEMORY

Phase I: Symbols Recovery

```
EXPORT_SYMBOL(``foo'');

struct kernel_symbol {
  unsigned long value;
  const char *name;
};
```



PHYSICAL MEMORY

push rbp 0x4efcab

``foo'' 0x4dff6e

___ksymtab

value 0xffffffff884efcab

name 0xffffffff884dff6e

value

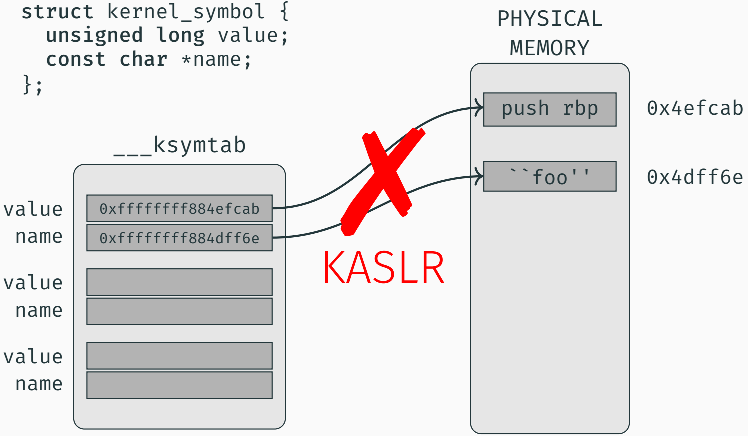
name

value

name

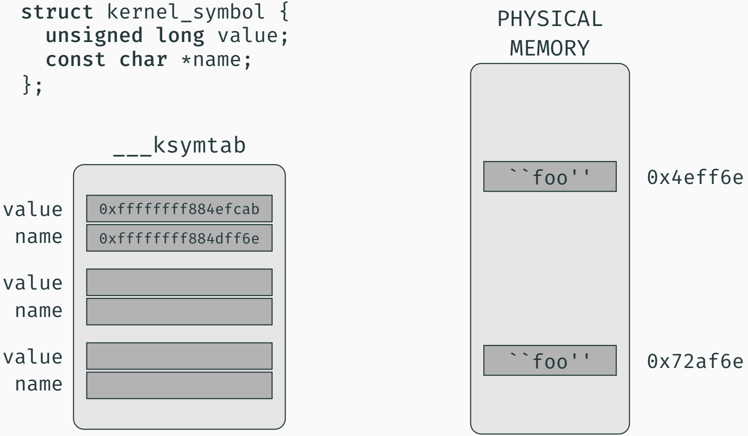
Phase I: Symbols Recovery

```
EXPORT_SYMBOL(``foo'');

struct kernel_symbol {
  unsigned long value;
  const char *name;
};
```



Phase I: Symbols Recovery

```
EXPORT_SYMBOL(``foo'');

struct kernel_symbol {
  unsigned long value;
  const char *name;
};
```

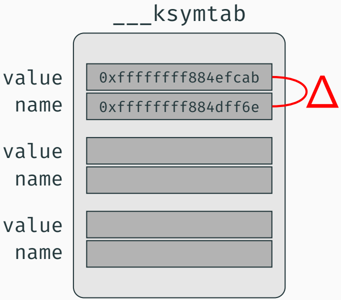


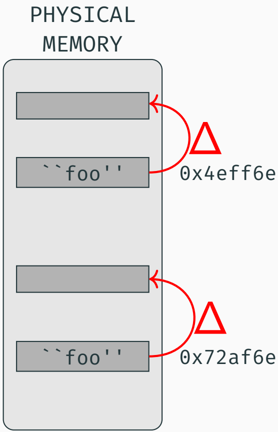
Our approach:

- Match “foo”

Phase I: Symbols Recovery

```
EXPORT_SYMBOL(``foo'');

struct kernel_symbol {
  unsigned long value;
  const char *name;
};
```



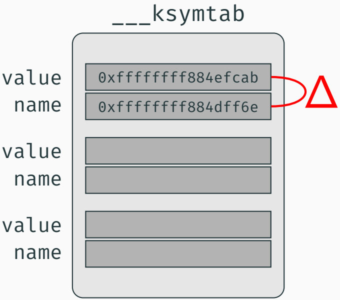


Our approach:

- Match “foo”
- Subtract distance
  (Δ = `name` - `value`)

Phase I: Symbols Recovery

```
EXPORT_SYMBOL(``foo'');

struct kernel_symbol {
  unsigned long value;
  const char *name;
};
```



___ksymtab

value 0xffffffff884efcab

name 0xffffffff884dff6e

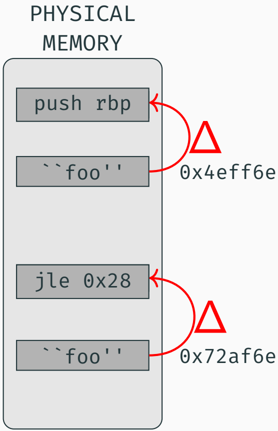
Δ

value

name

value

name



PHYSICAL MEMORY

push rbp

``foo''

Δ 0x4eff6e

jle 0x28

``foo''

Δ 0x72af6e

Our approach:

- Match “foo”
- Subtract distance (Δ = `name` - `value`)
- Extract and execute

Backup Slides - Back to the Whiteboard: a Principled Approach for the Assessment and Design of Memory Forensic Techniques (Usenix 2019)

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!