DIGITAL FORENSICS RESEARCH CONFERENCE

A Study on the Evolution of Kernel Data Types Used in Memory Forensics and Their Dependency on Compilation Options

Andrea Oliveri[1], Nikola Nemes[2], Branislav Andjelic[2], Davide Balzarotti[1]

[1]EURECOM [2]University of Novi Sad

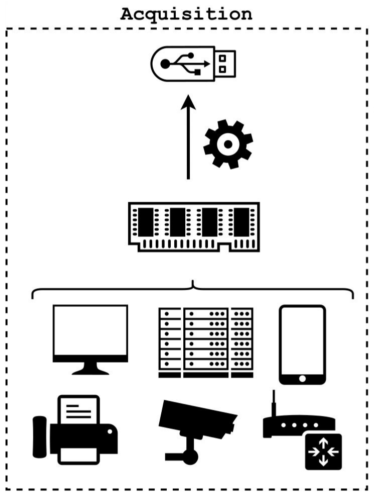
@IridiumXOR

oliveri@eurecom.fr

About OS Profiles

Memory forensics process is composed of 3 main phases:

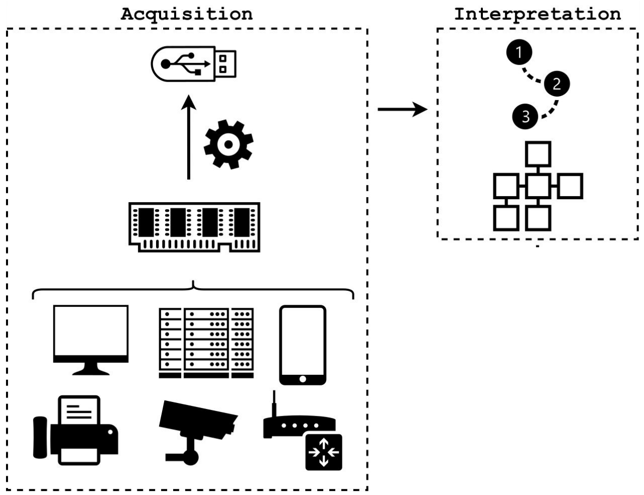
- Acquisition: copy RAM content to a non-volatile storage



About OS Profiles

Memory forensics process is composed of 3 main phases:

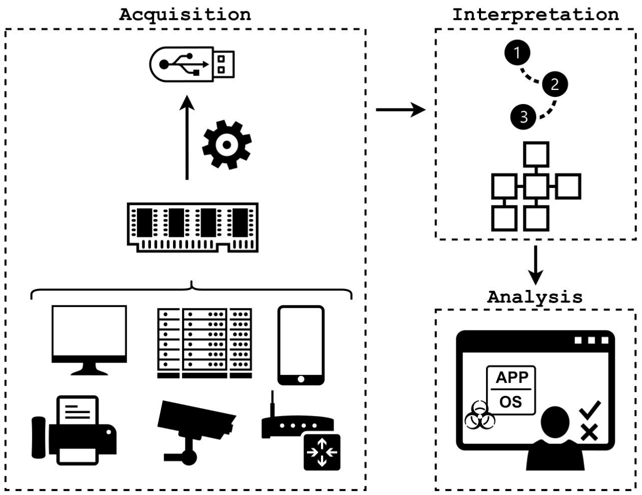
- Acquisition: copy RAM content to a non-volatile storage
- Interpretation: kernel structures locationing and exploration



About OS Profiles

Memory forensics process is composed of 3 main phases:

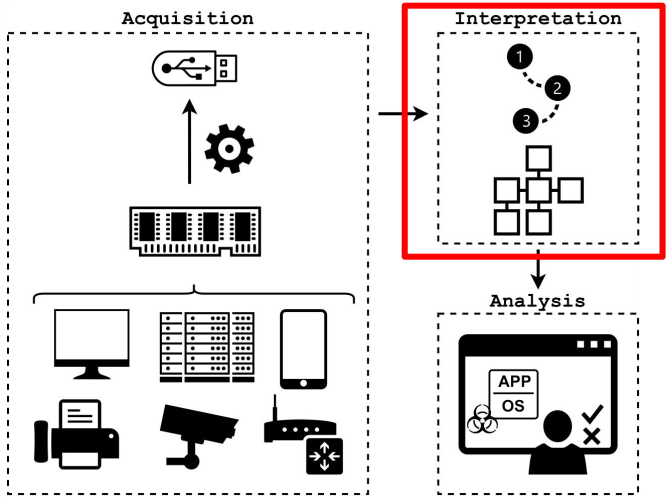
- Acquisition: copy RAM content to a non-volatile storage
- Interpretation: kernel structures locationing and exploration
- Analysis: analysis and correlation of forensics evidences



About OS Profiles

Memory forensics process is composed of 3 main phases:

- Acquisition: copy RAM content to a non-volatile storage
- Interpretation: kernel structures locationing and exploration
- Analysis: analysis and correlation of forensics evidence



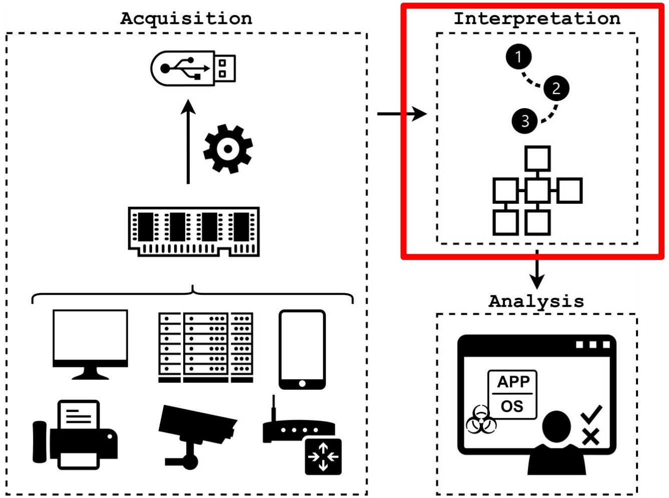
To locate and explore kernel structs we use **profiles**:

- contain location of kernel global variables

About OS Profiles

Memory forensics process is composed of 3 main phases:

- Acquisition: copy RAM content to a non-volatile storage
- Interpretation: kernel structures locationing and exploration
- Analysis: analysis and correlation of forensics evidence



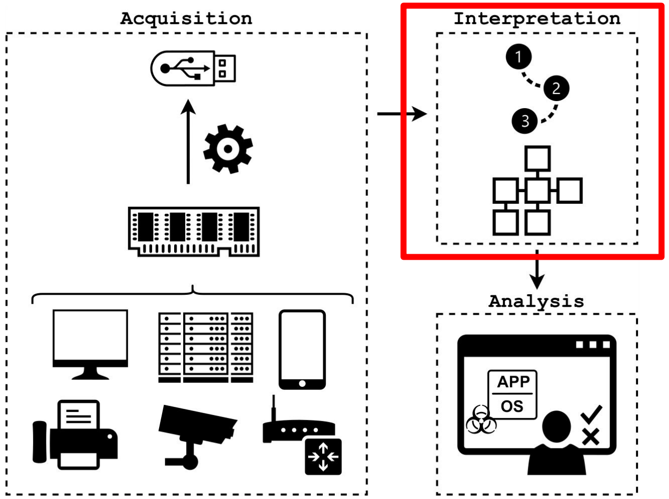
To locate and explore kernel structs we use **profiles**:

- contain location of kernel global variables
- contain description of kernel structs

About OS Profiles

Memory forensics process is composed of 3 main phases:

- Acquisition: copy RAM content to a non-volatile storage
- Interpretation: kernel structures locationing and exploration
- Analysis: analysis and correlation of forensics evidence



To locate and explore kernel structs we use **profiles**:

- contain location of kernel global variables
- contain description of kernel structs
- different OS versions and/or kernel configurations **require different profiles**

Why Different Profiles?

Why we need a different profile for each different kernel version?

Why Different Profiles?

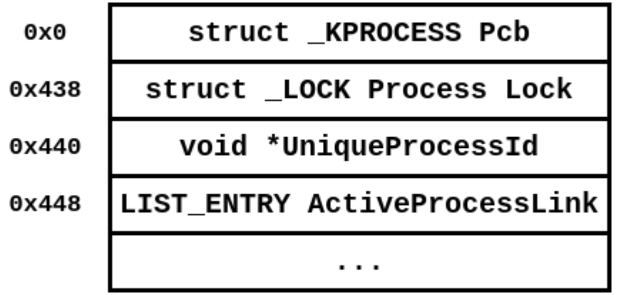
Why we need a different profile for each different kernel version?

- Each OS use different structs to represent same data

Why Different Profiles?

Why we need a different profile for each different kernel version?

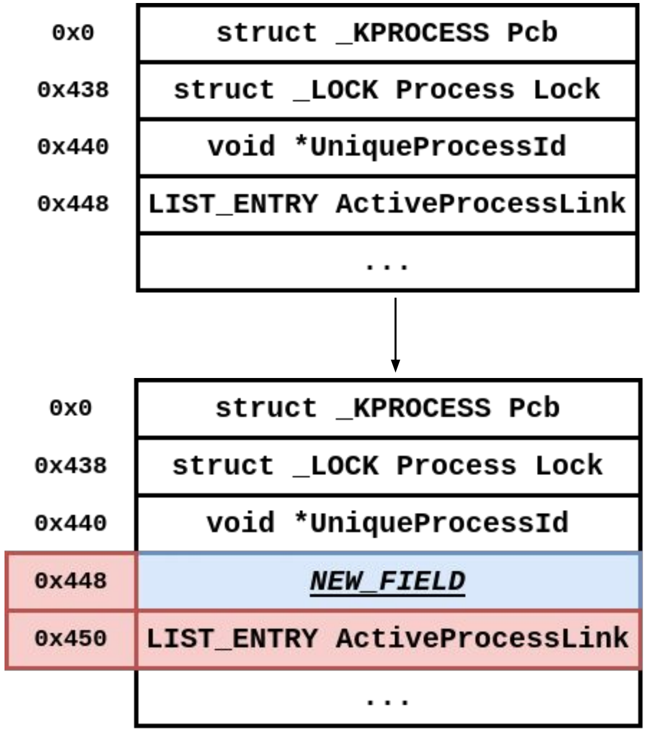
- Each OS use different structs to represent same data
- Kernel structs layout evolves



Why Different Profiles?

Why we need a different profile for each different kernel version?

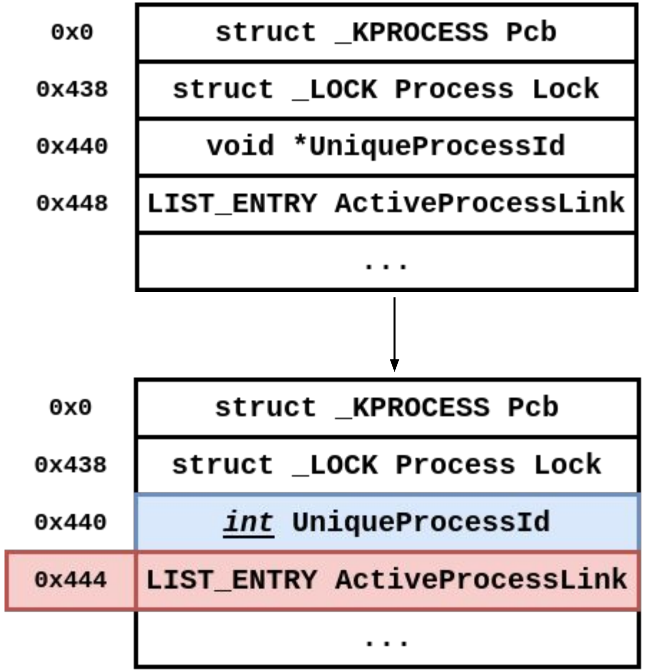
- Each OS use different structs to represent same data
- Kernel structs layout evolves
    - **Add/remove of a field**



Why Different Profiles?

Why we need a different profile for each different kernel version?

- Each OS use different structs to represent same data
- Kernel structs layout evolves
  - **Add/remove of a field**
  - **Change of field type**

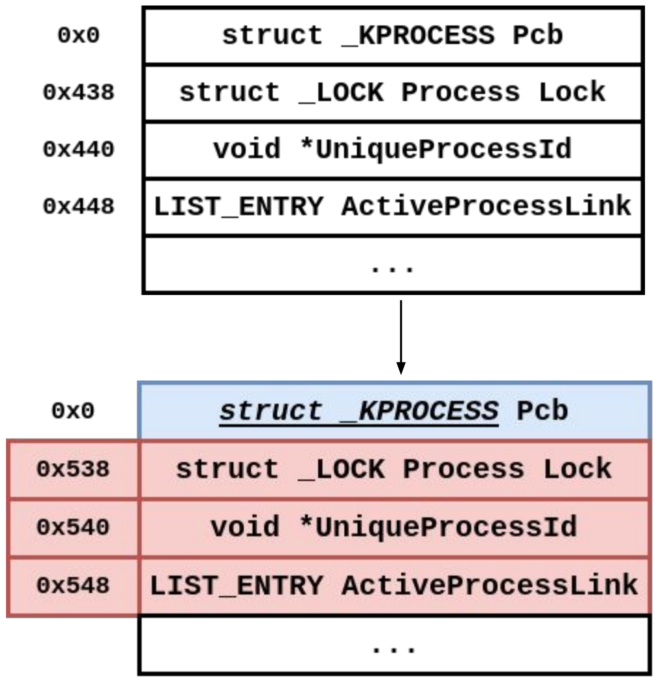


Why Different Profiles?

Why we need a different profile for each different kernel version?

- Each OS use different structs to represent same data
- Kernel structs layout evolves
  - **Add/remove of a field**
  - **Change of field type**
  - **Change in embedded structs**

    ⇒ avalanche effect!



Why Different Profiles?

Why we need a different profile for each different kernel version?

- Each OS use different structs to represent same data
- Kernel structs layout evolves
  - **Add/remove of a field**
  - **Change of field type**
  - **Change in embedded structs**
    ⇒ avalanche effect!
- Kernel global variables change location

Why Different Profiles?

Why we need a different profile for each different kernel version?

- Each OS use different structs to represent same data
- Kernel structs layout evolves
  - **Add/remove of a field**
  - **Change of field type**
  - **Change in embedded structs**
    ⇒ avalanche effect!
- Kernel global variables change location
- Kernel compiled with different options (Linux only)

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?**

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?**

⇒ We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?**

⇒ We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile

Analysis of Volatility 3 Profiles

⇒ Analysis of **Linux**, **macOS** and **Windows** kernels

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a "compatible" profile that contains (partially) correct structure definitions?**

⇒ We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile

Analysis of Volatility 3 Profiles

⇒ Analysis of **Linux**, **macOS** and **Windows** kernels

- How structs change in relation to OS version?

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **<u>Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?</u>**

⇒ We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile

<u>Analysis of Volatility 3 Profiles</u>

⇒ Analysis of **Linux, macOS** and **Windows** kernels

- How structs change in relation to OS version?
- Which Volatility plugins are the most affected by the use of a compatible profile?

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?**

⇒ We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile

Analysis of Volatility 3 Profiles

⇒ Analysis of **Linux, macOS** and **Windows** kernels

- How structs change in relation to OS version?
- Which Volatility plugins are the most affected by the use of a compatible profile?
- Guidelines in case of missing profile

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?**

⇒ **We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile**

Analysis of Volatility 3 Profiles

⇒ Analysis of **Linux**, **macOS** and **Windows** kernels

- How structs change in relation to OS version?
- Which Volatility plugins are the most affected by the use of a compatible profile?
- Guidelines in case of missing profile

Static Analysis of Linux Kernel Source Code

⇒ Analysis of **ALL** the Linux kernel compilation options at the same time

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?**

⇒ **We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile**

Analysis of Volatility 3 Profiles

⇒ Analysis of **Linux**, **macOS** and **Windows** kernels

- How structs change in relation to OS version?
- Which Volatility plugins are the most affected by the use of a compatible profile?
- Guidelines in case of missing profile

Static Analysis of Linux Kernel Source Code

⇒ Analysis of **ALL** the Linux kernel compilation options at the same time

- How Kernel options evolves?

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?**

⇒ We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile

Analysis of Volatility 3 Profiles

⇒ Analysis of **Linux, macOS** and **Windows** kernels

- How structs change in relation to OS version?
- Which Volatility plugins are the most affected by the use of a compatible profile?
- Guidelines in case of missing profile

Static Analysis of Linux Kernel Source Code

⇒ Analysis of **ALL** the Linux kernel compilation options at the same time

- How Kernel options evolves?
- How Kernel options affect structs?

How Kernel structs Evolve?

Unfortunately sometimes the correct profile is not available… **Is it possible to use a “compatible” profile that contains (partially) correct structure definitions?**

⇒ We have to characterize and measure how the kernel structs change among the different kernel releases and how many errors we introduce using a compatible profile

Analysis of Volatility 3 Profiles

⇒ Analysis of **Linux**, **macOS** and **Windows** kernels

- How structs change in relation to OS version?
- Which Volatility plugins are the most affected by the use of a compatible profile?
- Guidelines in case of missing profile

Static Analysis of Linux Kernel Source Code

⇒ Analysis of **ALL** the Linux kernel compilation options at the same time

- How Kernel options evolves?
- How Kernel options affect structs?
- Which class of options modifies data structures the most?

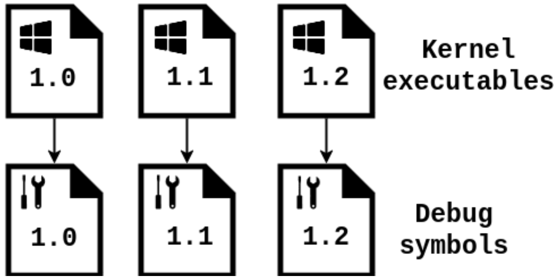
Volatility 3 Profiles Analysis

1. Collect kernel executables (**2298**)
   - **509** Linux (2.6.32 -> 6.5, Debian)
   - **195** macOS (10.6 -> 14.3, all available)
   - **1594** Windows (Vista -> Windows 11)

Volatility 3 Profiles Analysis

1. Collect kernel executables (**2298**)
   - **509** Linux (2.6.32 -> 6.5, Debian)
   - **195** macOS (10.6 -> 14.3, all available)
   - **1594** Windows (Vista -> Windows 11)

2. Retrieve debug symbols

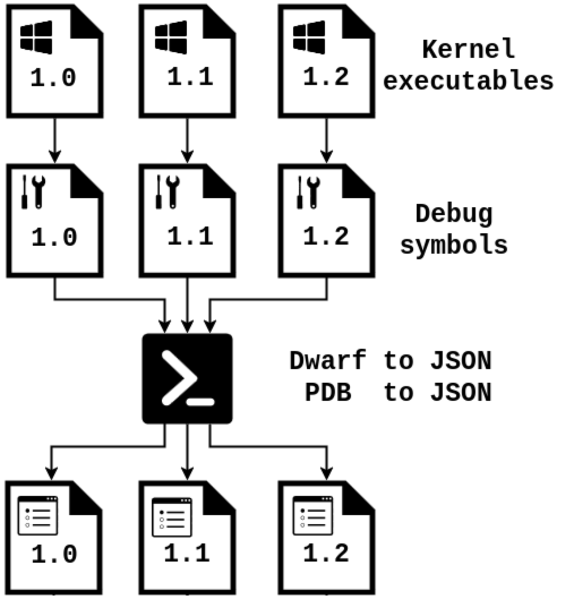


Volatility 3 Profiles Analysis

1. Collect kernel executables (**2298**)
   - **509 Linux** (2.6.32 -> 6.5, Debian)
   - **195 macOS** (10.6 -> 14.3, all available)
   - **1594 Windows** (Vista -> Windows 11)

2. Retrieve debug symbols

3. Create Volatility 3 JSON profiles



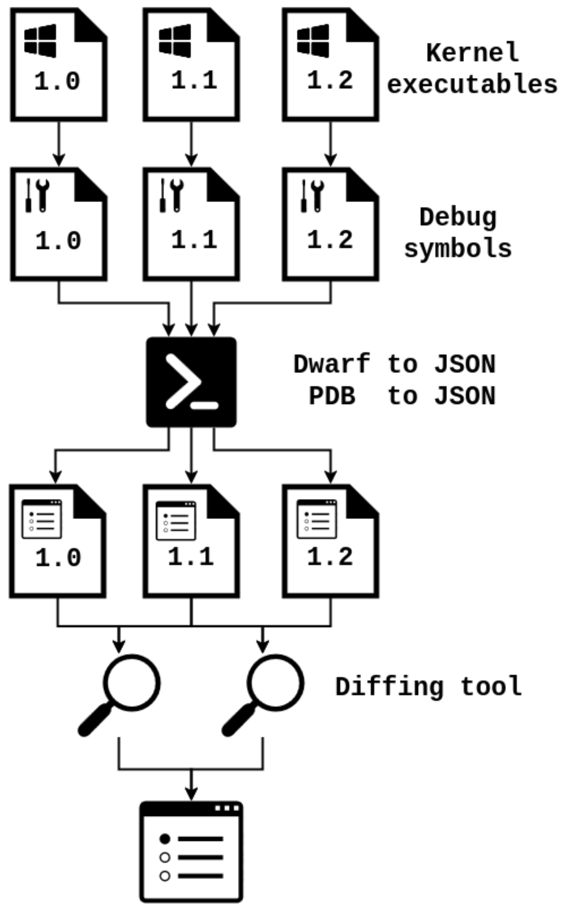
Volatility 3 Profiles Analysis

1. Collect kernel executables (**2298**)
   - **509 Linux** (2.6.32 -> 6.5, Debian)
   - **195 macOS** (10.6 -> 14.3, all available)
   - **1594 Windows** (Vista -> Windows 11)

2. Retrieve debug symbols

3. Create Volatility 3 JSON profiles

4. Diff “adjacent” versions



Volatility 3 Profiles Analysis

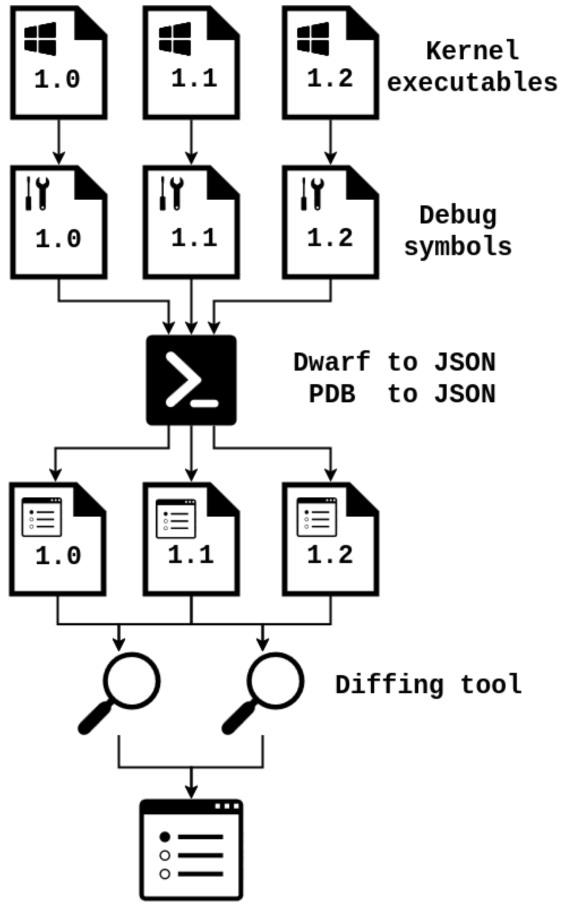
1. Collect kernel executables (**2298**)
   - <u>**509** Linux</u> (2.6.32 -> 6.5, Debian)
   - <u>**195** macOS</u> (10.6 -> 14.3, all available)
   - <u>**1594** Windows</u> (Vista -> Windows 11)

2. Retrieve debug symbols

3. Create Volatility 3 JSON profiles

4. Diff “adjacent” versions

5. Collect and analyze
   - Add/removed struct/unions
   - Add/removed fields
   - Change in embedded structs
   - Change in field offset
   - Change in global variables location



Stats on Changes in Forensics Relevant Data Structures

We track the evolution of the **most used forensics data structures**

- 29 Linux → **1452** changes
- 36 macOS → **333** changes
- 44 Windows → **279** changes

Stats on Changes in Forensics Relevant Data Structures

We track the evolution of the **most used forensics data structures**

- **29 Linux** → **1452** changes
- **36 macOS** → **333** changes
- **44 Windows** → **279** changes

We track three different class of fields: **data fields**, **pointers** and **embedded structs**.

- **Linux** → 93% of new fields are **add in the middle of a struct** ⇒ **avalanche effect**!

Stats on Changes in Forensics Relevant Data Structures

We track the evolution of the **most used forensics data structures**

●	29 Linux →	**1452** changes
●	**36** macOS →	**333** changes
●	**44** Windows →	**279** changes

We track three different class of fields: **data fields, pointers** and **embedded structs.**

- **Linux** → 93% of new fields are **add in the middle of a struct** ⇒ **avalanche effect**!
- **macOS** → High ratio of **data fields** that change **location**

Stats on Changes in Forensics Relevant Data Structures

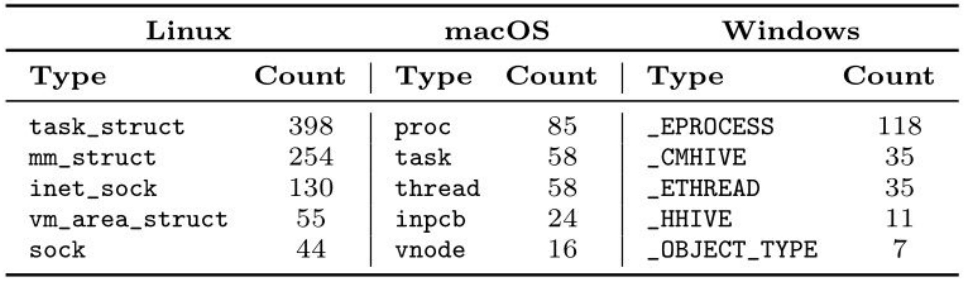
We track the evolution of the **most used forensics data structures**

● <u>29 Linux</u>	→ **1452** changes
● <u>36 macOS</u>	→ **333** changes
● <u>44 Windows</u>	→ **279** changes

We track three different class of fields: **data fields**, **pointers** and **embedded structs**.

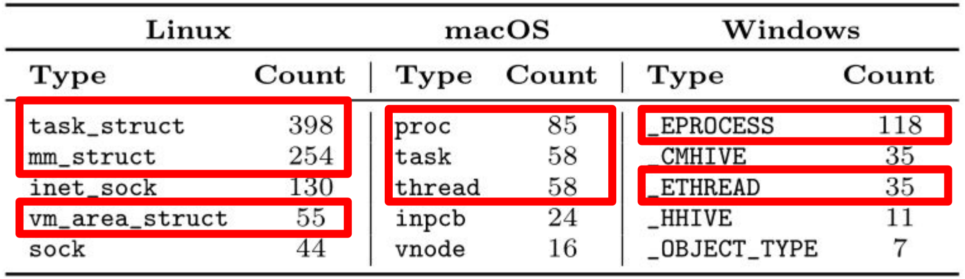
- **<u>Linux</u>** → <u>93%</u> of new fields are **add in the middle of a struct** ⇒ **<u>avalanche effect</u>**!
- **<u>macOS</u>** → High ratio of **data fields** that change **location**
- **<u>Windows</u>** → High ratio of changes in types of **embedded structures** and their **offsets**

Detailed Analysis of Structs Changes



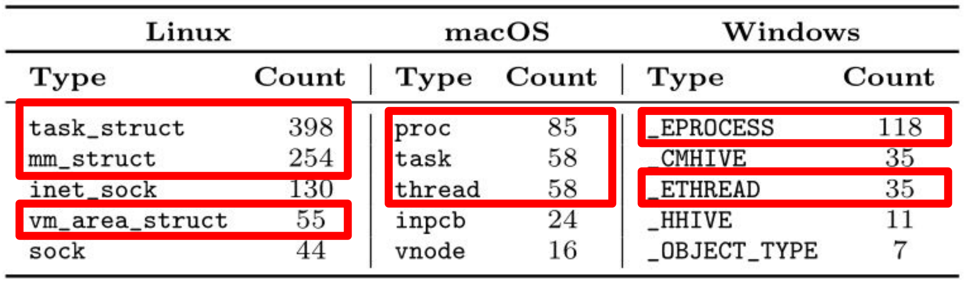
Linux		macOS		Windows	
Type	Count	Type	Count	Type	Count
task_struct	398	proc	85	_EPROCESS	118
mm_struct	254	task	58	_CMHIVE	35
inet_sock	130	thread	58	_ETHREAD	35
vm_area_struct	55	inpcb	24	_HHIVE	11
sock	44	vnode	16	_OBJECT_TYPE	7

Detailed Analysis of Structs Changes

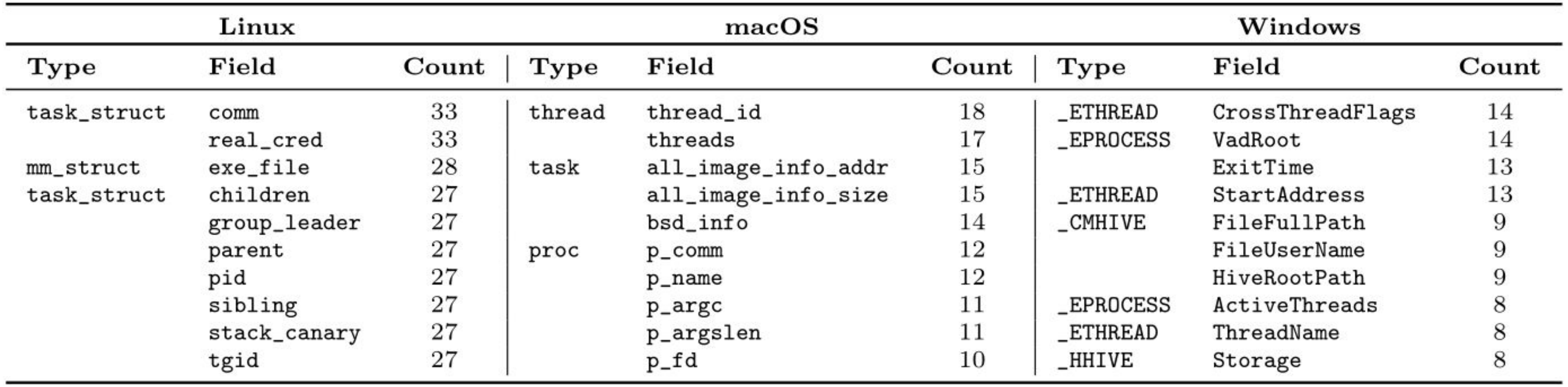


Linux		macOS		Windows	
Type	Count	Type	Count	Type	Count
task_struct	398	proc	85	_EPROCESS	118
mm_struct	254	task	58	_CMHIVE	35
inet_sock	130	thread	58	_ETHREAD	35
vm_area_struct	55	inpcb	24	_HHIVE	11
sock	44	vnode	16	_OBJECT_TYPE	7

Detailed Analysis of Structs Changes



Linux		macOS		Windows	
Type	Count	Type	Count	Type	Count
task_struct	398	proc	85	_EPROCESS	118
mm_struct	254	task	58	_CMHIVE	35
inet_sock	130	thread	58	_ETHREAD	35
vm_area_struct	55	inpcb	24	_HHIVE	11
sock	44	vnode	16	_OBJECT_TYPE	7



Linux			macOS			Windows		
Type	Field	Count	Type	Field	Count	Type	Field	Count
task_struct	comm	33	thread	thread_id	18	_ETHREAD	CrossThreadFlags	14
	real_cred	33		threads	17	_EPROCESS	VadRoot	14
mm_struct	exe_file	28	task	all_image_info_addr	15		ExitTime	13
task_struct	children	27		all_image_info_size	15	_ETHREAD	StartAddress	13
	group_leader	27		bsd_info	14	_CMHIVE	FileFullPath	9
	parent	27	proc	p_comm	12		FileUserName	9
	pid	27		p_name	12		HiveRootPath	9
	sibling	27		p_argc	11	_EPROCESS	ActiveThreads	8
	stack_canary	27		p_argslen	11	_ETHREAD	ThreadName	8
	tgid	27		p_fd	10	_HHIVE	Storage	8

Detailed Analysis of Structs Changes

Linux		macOS		Windows	
Type	Count	Type	Count	Type	Count
task_struct	398	proc	85	_EPROCESS	118
mm_struct	254	task	58	_CMHIVE	35
inet_sock	130	thread	58	_ETHREAD	35
vm_area_struct	55	inpcb	24	_HHIVE	11
sock	44	vnode	16	_OBJECT_TYPE	7

Linux			macOS			Windows		
Type	Field	Count	Type	Field	Count	Type	Field	Count
task_struct	comm	33	thread	thread_id	18	_ETHREAD	CrossThreadFlags	14
	real_cred	33		threads	17	_EPROCESS	VadRoot	14
mm_struct	exe_file	28	task	all_image_info_addr	15		ExitTime	13
task_struct	children	27		all_image_info_size	15	_ETHREAD	StartAddress	13
	group_leader	27		bsd_info	14	_CMHIVE	FileFullPath	9
	parent	27	proc	p_comm	12		FileUserName	9
	pid	27		p_name	12		HiveRootPath	9
	sibling	27		p_argc	11	_EPROCESS	ActiveThreads	8
	stack_canary	27		p_argslen	11	_ETHREAD	ThreadName	8
	tgid	27		p_fd	10	_HHIVE	Storage	8

Affected Volatility Plugins

- **Plugins that list processes** using linked-list walk
  ⇒ **pslist, pstree, psaux, ps_env, threads**

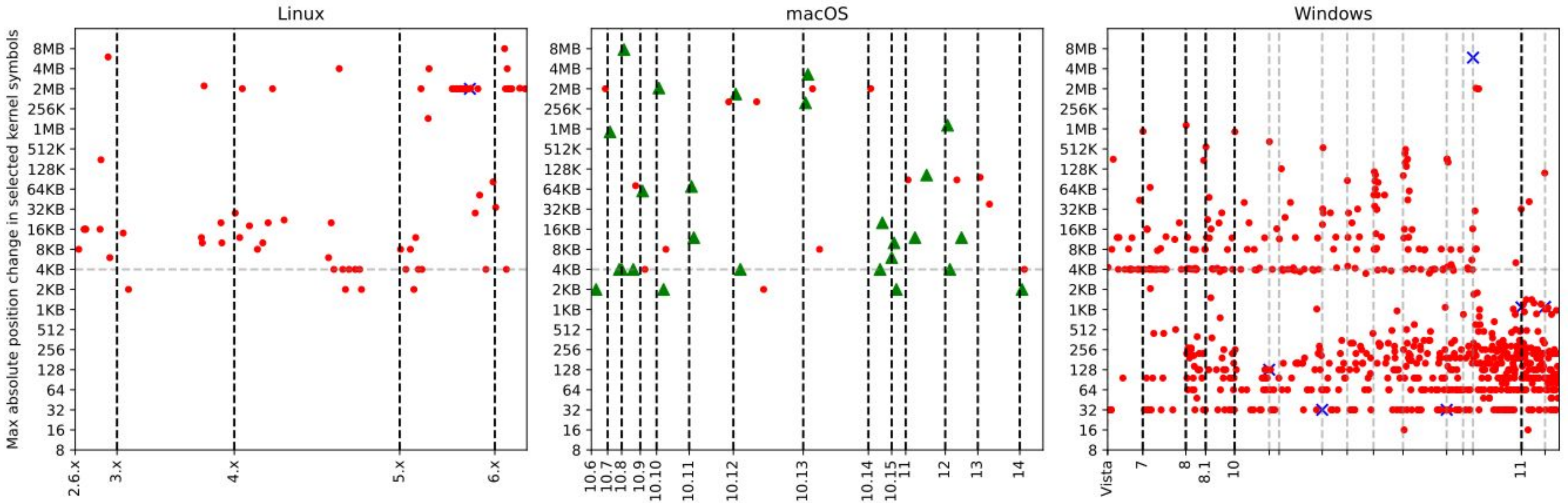
Affected Volatility Plugins

- <u>**Plugins that list processes**</u> using linked-list walk
  ⇒ **pslist, pstree, psaux, ps_env, threads**
- <u>**Plugins that analyze processes**</u> are indirectly affected
  ⇒ **dump_map, elf, procdump, librarylist, dump**

Affected Volatility Plugins

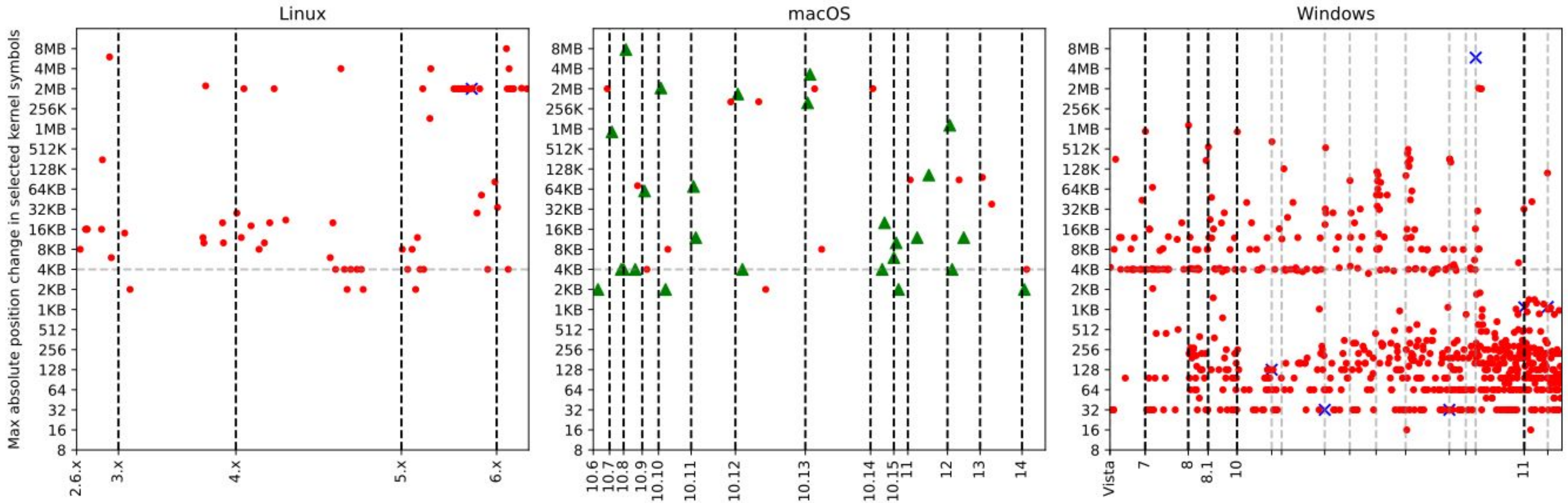
- <u>**Plugins that list processes**</u> using linked-list walk
  ⇒ **pslist, pstree, psaux, ps_env, threads**
- <u>**Plugins that analyze processes**</u> are indirectly affected
  ⇒ **dump_map, elf, procdump, librarylist, dump**
- Other affected plugins:
  - <u>**macOS**</u> ⇒ **lsof, listraw**
  - <u>**Windows**</u> ⇒ all **Registry plugins**

Kernel Global Variables Offsets Variability



Three **Kernel Global Variables** are **essential to** start a forensics analysis:

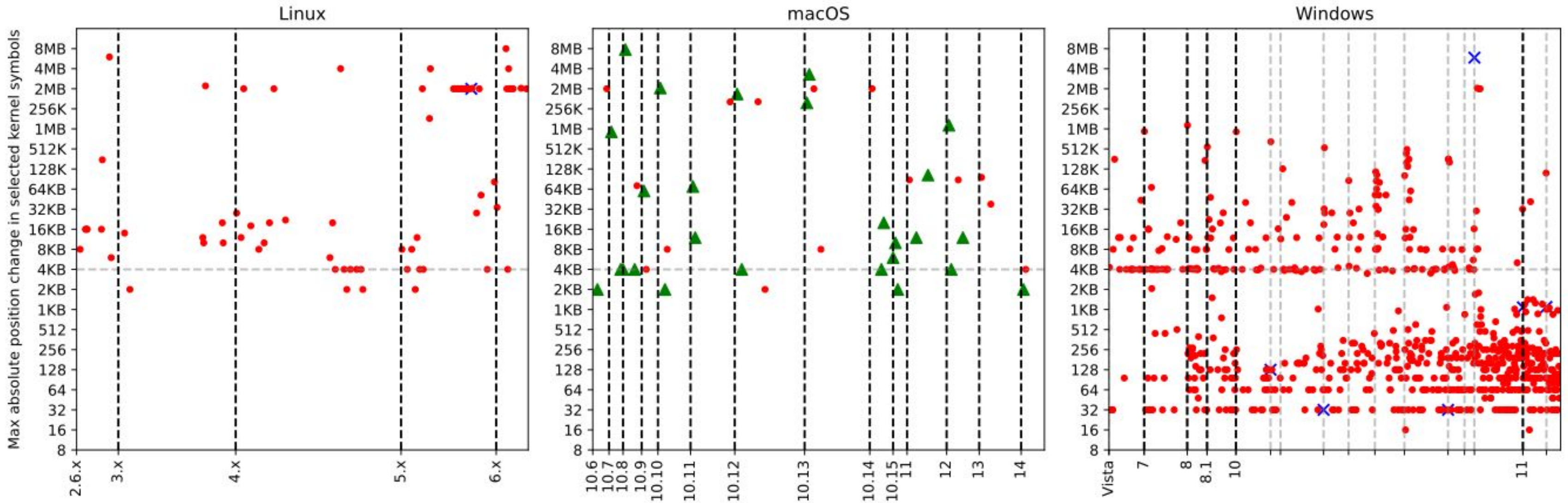
Kernel Global Variables Offsets Variability



Three **Kernel Global Variables** are **essential to** start a forensics analysis:

- To determine the **KASLR offset**

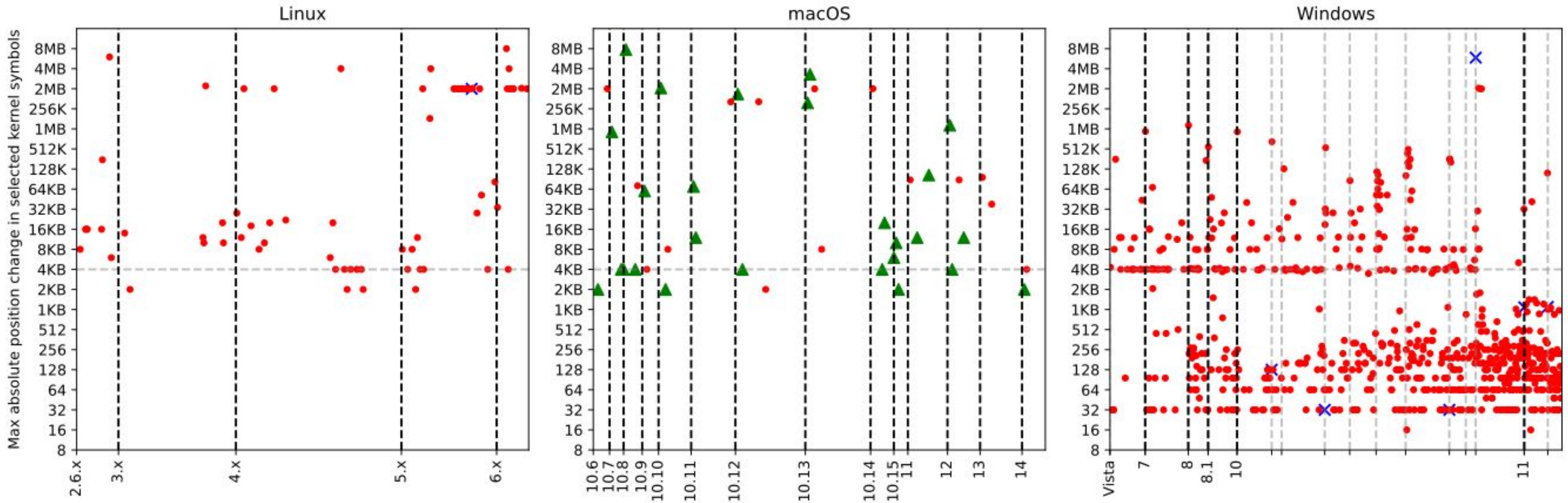
Kernel Global Variables Offsets Variability



Three **Kernel Global Variables** are **essential to** start a forensics analysis:

- To determine the **KASLR offset**
- To identify the **processes linked list**

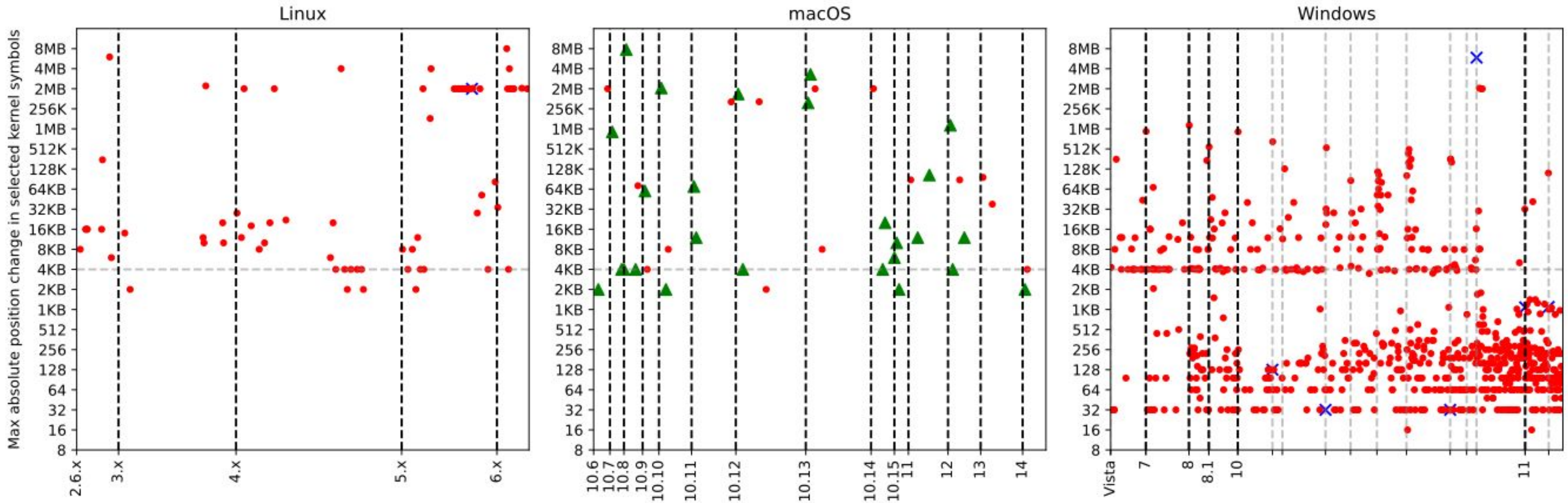
Kernel Global Variables Offsets Variability



Three **Kernel Global Variables** are **essential to** start a forensics analysis:

- To determine the **KASLR offset**
- To identify the **processes linked list**
- To identify the **kernel modules linked list**

Kernel Global Variables Offsets Variability



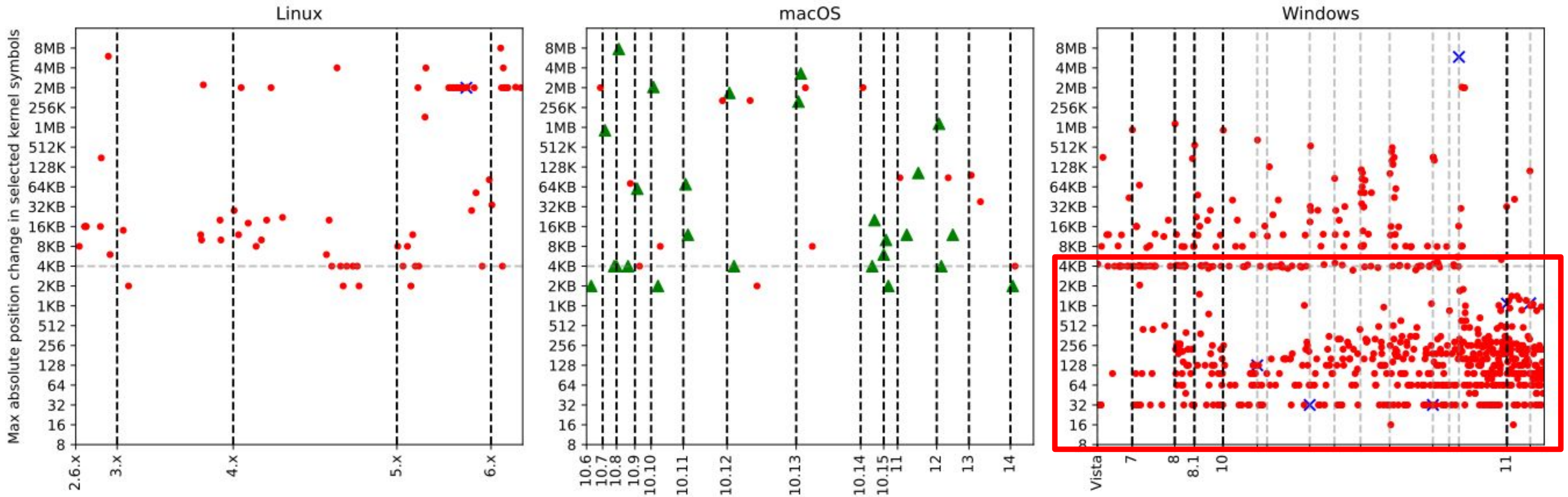
Three **Kernel Global Variables** are **essential to** start a forensics analysis:

- To determine the **KASLR offset**
- To identify the **processes linked list**
- To identify the **kernel modules linked list**

In <u>Windows:</u>

- change in **44% of adjacent kernel** versions (vs 14% on Linux and 21% on macOS)

Kernel Global Variables Offsets Variability



Three **Kernel Global Variables** are **essential to** start a forensics analysis:

- To determine the **KASLR offset**
- To identify the **processes linked list**
- To identify the **kernel modules linked list**

In <u>Windows:</u>

- change in **44% of adjacent kernel** versions (vs 14% on Linux and 21% on macOS)
- **76%** of the **offset** shift is **less than the page size** (4KB)

Select The most Compatible Profile

Suppose that we don’t have the correct profile,
how can we select the most compatible one?

Select The most Compatible Profile

Suppose that we don’t have the correct profile,
how can we select the most compatible one?

We can reach an **high compatibility** between kernel data structures if we use structure definitions taken by the previous

Linux ⇒ **minor release**

macOS ⇒ **major release (or first minor near the major one)**

Windows ⇒ **patch release**

Select The most Compatible Profile

Suppose that we don't have the correct profile,
<u>how can we select the most compatible one</u>?

We can reach an **<u>high compatibility</u>** between kernel data structures if we use structure definitions taken by the previous

<u>Linux</u>	⇒ **minor release**
<u>macOS</u>	⇒ **major release (or first minor near the major one)**
<u>Windows</u>	⇒ **patch release**

Global variables can require to be **brute forced**…

Select The most Compatible Profile

Suppose that we don’t have the correct profile,
<u>how can we select the most compatible one</u>?

We can reach an **<u>high compatibility</u>** between kernel data structures if we use structure definitions taken by the previous

- **<u>Linux</u> ⇒ minor release**
- **<u>macOS</u> ⇒ major release (or first minor near the major one)**
- **<u>Windows</u> ⇒ patch release**

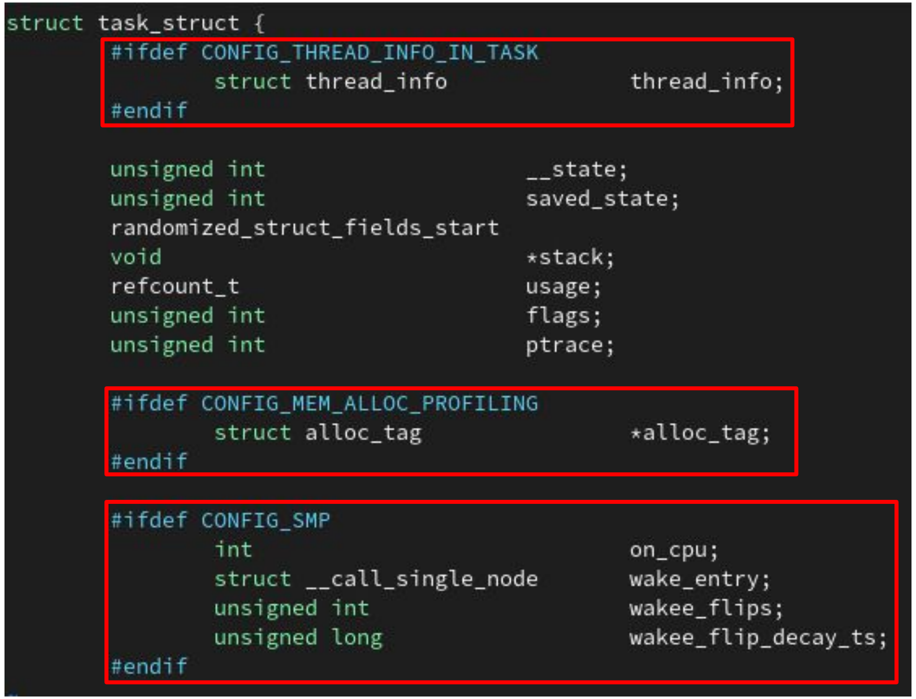
Global variables can require to be **brute forced**…

⇒ **Is it possible to automate** the creation of a profile from a near one?

<u>RESEARCH IN PROGRESS</u>

Linux Kernel Compilation Option Dependency

- The **KConfig** options **influence**:
  - the kernel behaviour
  - **the layout of data structures**



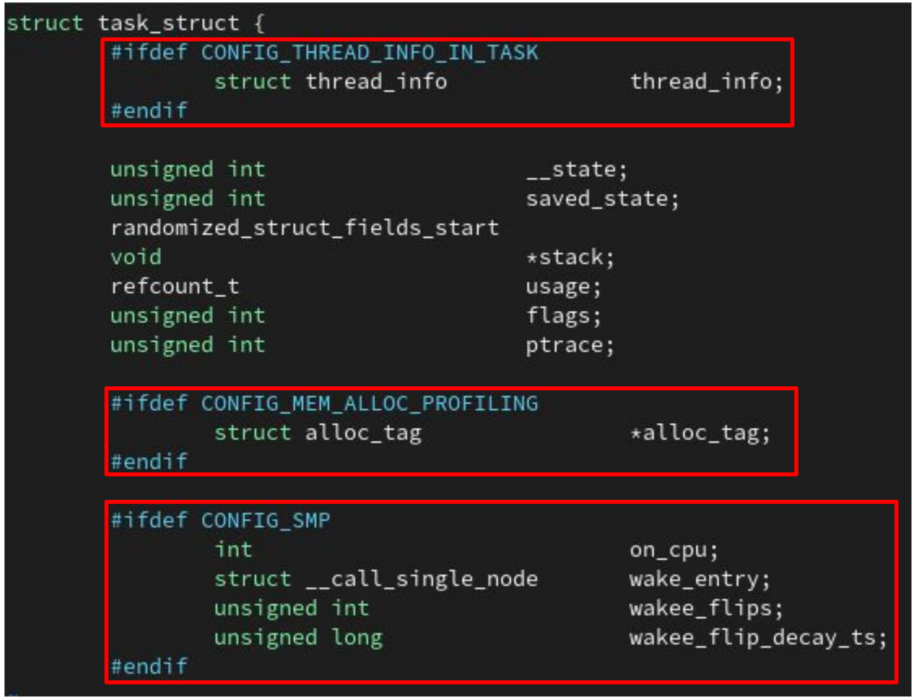
```
struct task_struct {
	#ifdef CONFIG_THREAD_INFO_IN_TASK
		struct thread_info		thread_info;
	#endif

	unsigned int			__state;
	unsigned int			saved_state;
	randomized_struct_fields_start
	void				*stack;
	refcount_t			usage;
	unsigned int			flags;
	unsigned int			ptrace;

	#ifdef CONFIG_MEM_ALLOC_PROFILING
		struct alloc_tag		*alloc_tag;
	#endif

	#ifdef CONFIG_SMP
		int				on_cpu;
		struct __call_single_node	wake_entry;
		unsigned int			wakee_flips;
		unsigned long			wakee_flip_decay_ts;
	#endif
```

Linux Kernel Compilation Option Dependency

- The **KConfig** options **influence**:
  - the kernel behaviour
  - **the layout of data structures**

How the **KConfig** options **influence** the forensics data **structures layout?**

Which KConfig options have the **major impact?**



```
struct task_struct {
	#ifdef CONFIG_THREAD_INFO_IN_TASK
		struct thread_info		thread_info;
	#endif

	unsigned int			__state;
	unsigned int			saved_state;
	randomized_struct_fields_start
	void				*stack;
	refcount_t			usage;
	unsigned int			flags;
	unsigned int			ptrace;

	#ifdef CONFIG_MEM_ALLOC_PROFILING
		struct alloc_tag		*alloc_tag;
	#endif

	#ifdef CONFIG_SMP
		int				on_cpu;
		struct __call_single_node	wake_entry;
		unsigned int			wakee_flips;
		unsigned long			wakee_flip_decay_ts;
	#endif
```

Linux Kernel Compilation Option (KConfig) Dependency

- Volatility 3 profiles approach not applicable …

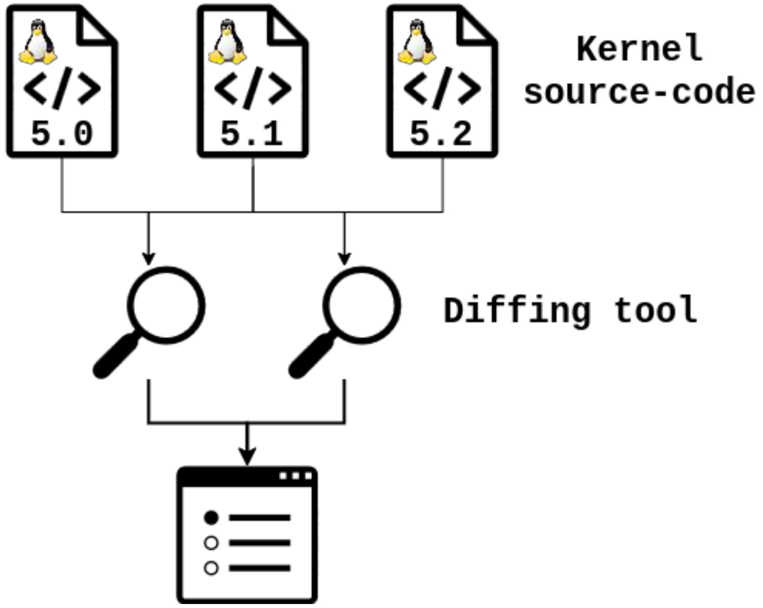
  ⇒ ..<u>compare directly the source code</u>!

Linux Kernel Compilation Option (KConfig) Dependency

- Volatility 3 profiles approach not applicable …

  ⇒ ..**<u>compare directly the source code</u>!**

- We **parse and compare source code of adjacent kernel** versions
  - We exclude Hardware drivers
  - Focus on x64 and ARM64



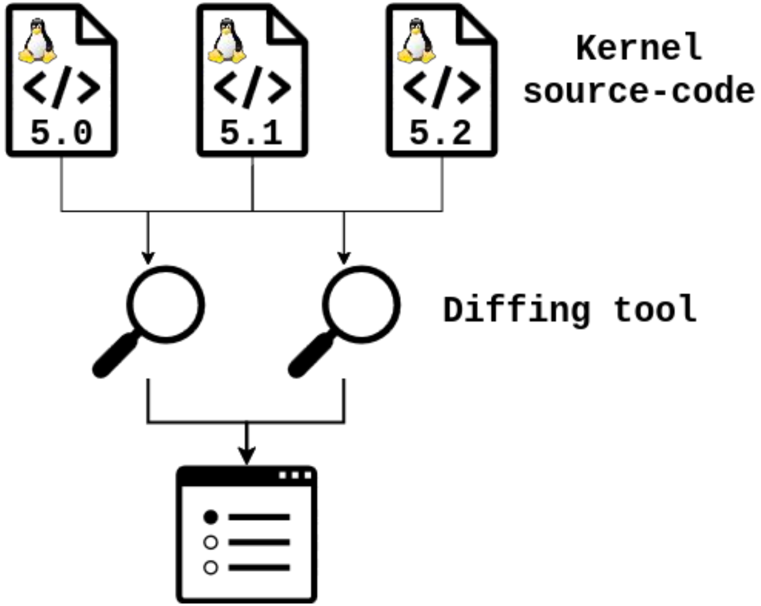
Linux Kernel Compilation Option (KConfig) Dependency

- Volatility 3 profiles approach not applicable …

  ⇒ ..compare directly the source code!

- We **parse and compare source code of adjacent kernel** versions
  - We exclude Hardware drivers
  - Focus on x64 and ARM64

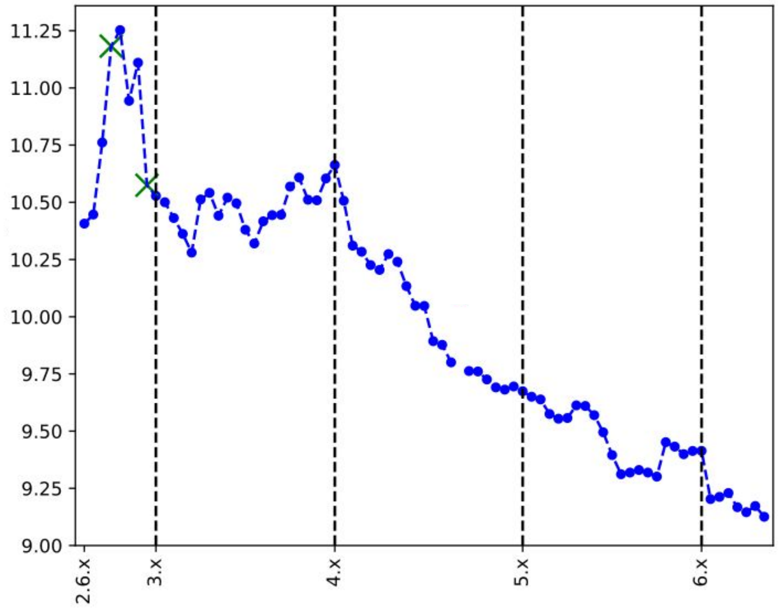
- 77 different minor **versions**
  - from **2.6.32** up to **6.7**
  - covering **96.6% of C structs**



Structs KConfig Dependency: General Stats

Percentage of **structs affected** by KCONFIGs

⇒ **~10%** in continue decrease

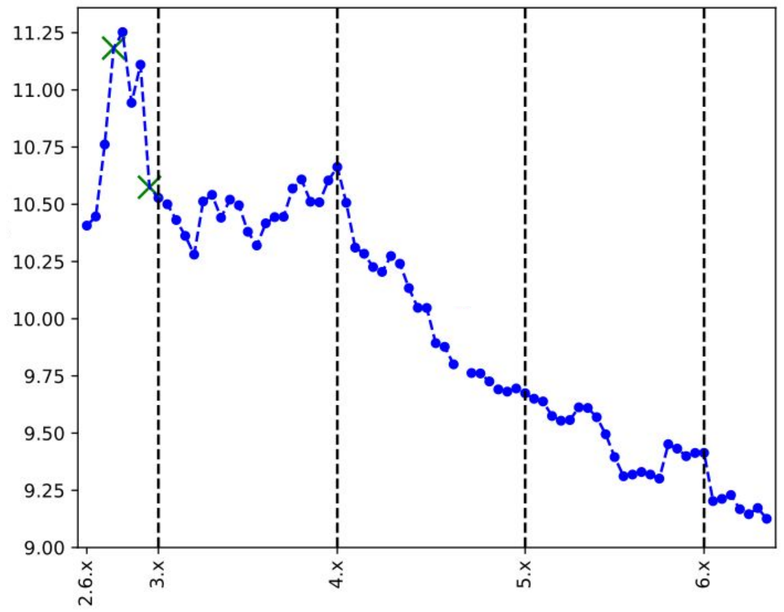


Structs KConfig Dependency: General Stats

Percentage of **structs affected** by KCONFIGs

⇒ **~10%** in continue decrease

… **<u>however can be local spikes</u>**!



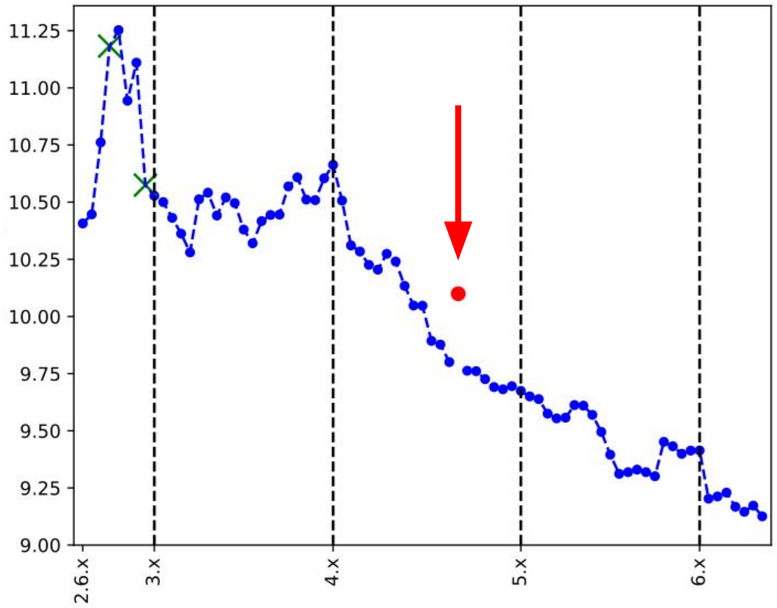
Structs KConfig Dependency: General Stats

Percentage of **structs affected** by KCONFIGs

⇒ ***~10%*** in continue decrease

… **<u>however can be local spikes</u>**!

- **CONFIG_LOCKDEP_CROSSRELEASE** in 4.14 in **lockdep_map** struct



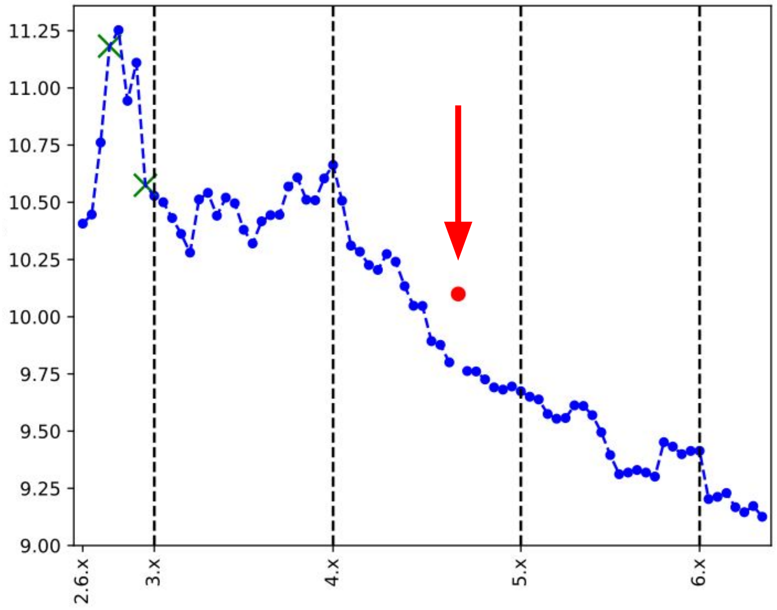
Structs KConfig Dependency: General Stats

Percentage of **structs affected** by KCONFIGs

⇒ ***~10%*** in continue decrease

… **<u>however can be local spikes</u>**!

- **CONFIG_LOCKDEP_CROSSRELEASE** in 4.14 in **lockdep_map** struct
- Introduce runtime deadlock detection
  ⇒ **enabled by all major distributions**



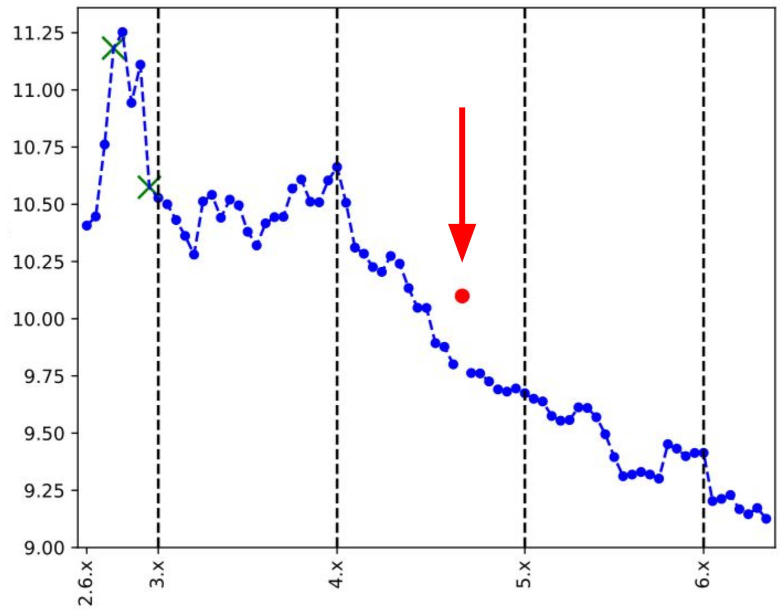
Structs KConfig Dependency: General Stats

Percentage of **structs affected** by KCONFIGs

⇒ ***~10%*** in continue decrease

… <u>**however can be local spikes**</u>!

- **CONFIG_LOCKDEP_CROSSRELEASE** in 4.14 in **lockdep_map** struct
- Introduce runtime deadlock detection
  ⇒ **enabled by all major distributions**
- lockdep_map **embedded in 1451 structs**,
  ⇒ **12 forensics** structures
  e.g. **task_struct**, **module**, and **inode**



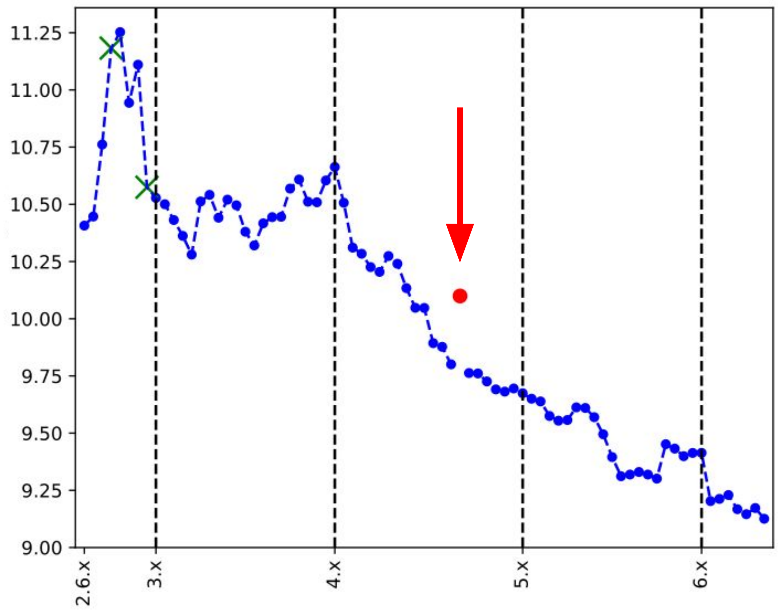
Structs KConfig Dependency: General Stats

Percentage of **structs affected** by KCONFIGs

⇒ ***~10%*** in continue decrease

... **<u>however can be local spikes</u>**!

- **CONFIG_LOCKDEP_CROSSRELEASE** in 4.14 in **lockdep_map** struct
- Introduce runtime deadlock detection
  ⇒ **enabled by all major distributions**
- lockdep_map **embedded in 1451 structs**,
  ⇒ **12 forensics** structures
  e.g. **task_struct**, **module**, and **inode**
- ... **<u>avalanche effect</u>**!



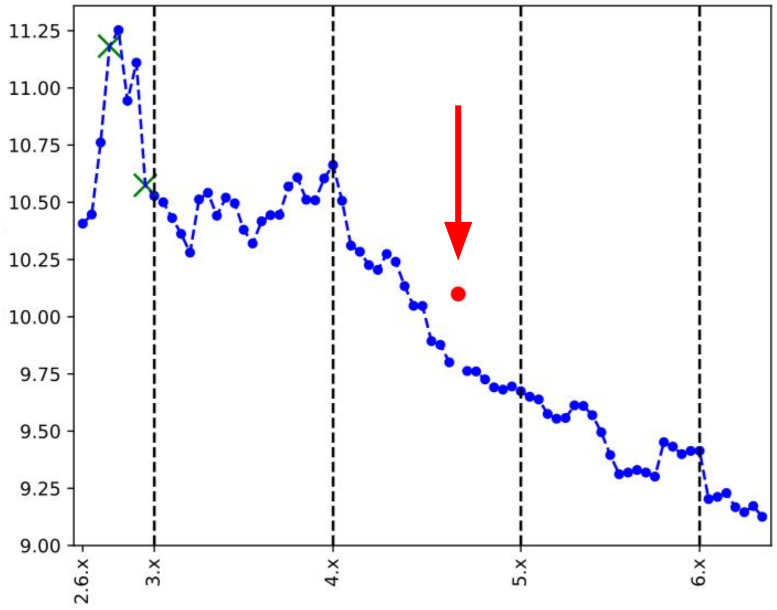
Structs KConfig Dependency: General Stats

Percentage of **structs affected** by KCONFIGs

⇒ **~10%** in continue decrease

… **<u>however can be local spikes</u>**!

- **CONFIG_LOCKDEP_CROSSRELEASE** in 4.14 in **lockdep_map** struct
- Introduce runtime deadlock detection
  ⇒ **enabled by all major distributions**
- lockdep_map **embedded in 1451 structs**,
  ⇒ **12 forensics** structures
  e.g. **task_struct**, **module**, and **inode**
- … **<u>avalanche effect</u>**!
- CONFIG_LOCKDEP_CROSSRELEASE **removed in 4.15** due to huge performance penalty



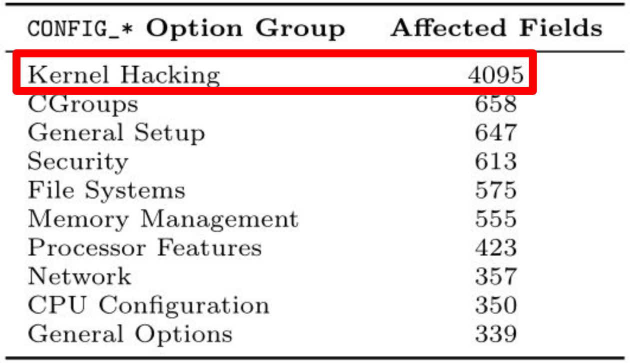
Most Relevant KConfigs for Forensics Structures

Which type of KConfig affect more forensics data structures?

Most Relevant KConfigs for Forensics Structures

Which type of KConfig affect more forensics data structures?

- "**Kernel Hacking**" ⇒ support to debug and performance tools. Rarely used in embedded devices.

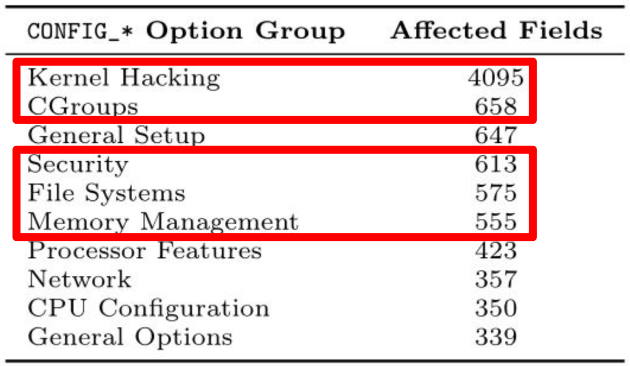


CONFIG_* **Option Group**	**Affected Fields**
Kernel Hacking	4095
CGroups	658
General Setup	647
Security	613
File Systems	575
Memory Management	555
Processor Features	423
Network	357
CPU Configuration	350
General Options	339

Most Relevant KConfigs for Forensics Structures

Which type of KConfig affect more forensics data structures?

- "**Kernel Hacking**" ⇒ support to debug and performance tools. Rarely used in embedded devices.
- Other important groups:
  ⇒ **CGroups, Security, FS, MM**

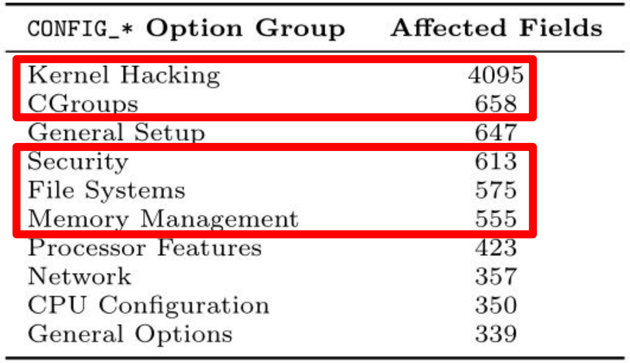


CONFIG_* **Option Group**	**Affected Fields**
Kernel Hacking	4095
CGroups	658
General Setup	647
Security	613
File Systems	575
Memory Management	555
Processor Features	423
Network	357
CPU Configuration	350
General Options	339

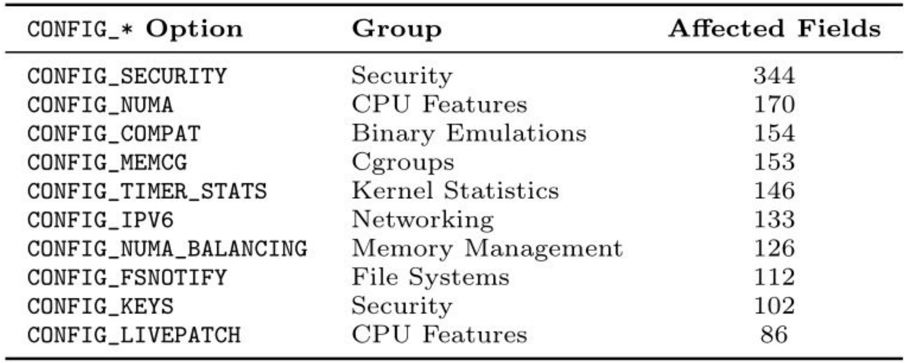
Most Relevant KConfigs for Forensics Structures

Which type of KConfig affect more forensics data structures?

- "**Kernel Hacking**" ⇒ support to debug and performance tools. Rarely used in embedded devices.
- Other important groups:
  ⇒ **CGroups, Security, FS, MM**



CONFIG_* Option Group	Affected Fields
Kernel Hacking	4095
CGroups	658
General Setup	647
Security	613
File Systems	575
Memory Management	555
Processor Features	423
Network	357
CPU Configuration	350
General Options	339



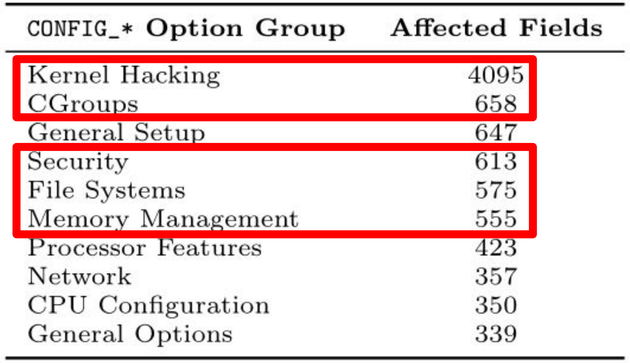
CONFIG_* Option	Group	Affected Fields
CONFIG_SECURITY	Security	344
CONFIG_NUMA	CPU Features	170
CONFIG_COMPAT	Binary Emulations	154
CONFIG_MEMCG	Cgroups	153
CONFIG_TIMER_STATS	Kernel Statistics	146
CONFIG_IPV6	Networking	133
CONFIG_NUMA_BALANCING	Memory Management	126
CONFIG_FSNOTIFY	File Systems	112
CONFIG_KEYS	Security	102
CONFIG_LIVEPATCH	CPU Features	86

Most Relevant KConfigs for Forensics Structures

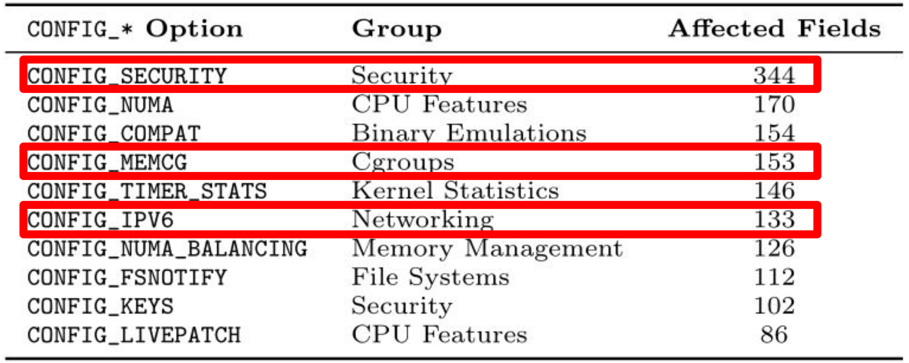
Which type of KConfig affect more forensics data structures?

- "**Kernel Hacking**" ⇒ support to debug and performance tools. Rarely used in embedded devices.
- Other important groups:
  ⇒ **CGroups, Security, FS, MM**

- **CONFIG_SECURITY** ⇒ most influencing one
  - Support for AppArmor and SELinux
- **CONFIG_MEMCG**
  - Support for Memory CGroup
- **CONFIG_IPV6**
  - Support for IPv6



CONFIG_* Option Group	Affected Fields
Kernel Hacking	4095
CGroups	658
General Setup	647
Security	613
File Systems	575
Memory Management	555
Processor Features	423
Network	357
CPU Configuration	350
General Options	339



CONFIG_* Option	Group	Affected Fields
CONFIG_SECURITY	Security	344
CONFIG_NUMA	CPU Features	170
CONFIG_COMPAT	Binary Emulations	154
CONFIG_MEMCG	Cgroups	153
CONFIG_TIMER_STATS	Kernel Statistics	146
CONFIG_IPV6	Networking	133
CONFIG_NUMA_BALANCING	Memory Management	126
CONFIG_FSNOTIFY	File Systems	112
CONFIG_KEYS	Security	102
CONFIG_LIVEPATCH	CPU Features	86

Conclusions

In this study we have:

- Shown **how the structure are influenced** by the different development cycle of the OS.

Conclusions

In this study we have:

- Shown **how the structure are influenced** by the different development cycle of the OS.
- Shown **which Volatility plugins are affected** more by the changing in the structure offsets.

Conclusions

In this study we have:

- Shown **how the structure are influenced** by the different development cycle of the OS.
- Shown **which Volatility plugins are affected** more by the changing in the structure offsets.
- Introduced **minimal guidelines to help analysts** to use the most compatible profile when the correct one is not available.

Conclusions

In this study we have:

- Shown **how the structure are influenced** by the different development cycle of the OS.
- Shown **which Volatility plugins are affected** more by the changing in the structure offsets.
- Introduced **minimal guidelines to help analysts** to use the most compatible profile when the correct one is not available.
- Shown the **avalanche effect of KConfigs** on Linux kernel structs.

Conclusions

In this study we have:

- Shown **how the structure are influenced** by the different development cycle of the OS.
- Shown **which Volatility plugins are affected** more by the changing in the structure offsets.
- Introduced **minimal guidelines to help analysts** to use the most compatible profile when the correct one is not available.
- Shown the **avalanche effect of KConfigs** on Linux kernel structs.
- Identified **most influence KConfigs on forensics structs**.

Conclusions

In this study we have:

- Shown **how the structure are influenced** by the different development cycle of the OS.
- Shown **which Volatility plugins are affected** more by the changing in the structure offsets.
- Introduced **minimal guidelines to help analysts** to use the most compatible profile when the correct one is not available.
- Shown the **avalanche effect of KConfigs** on Linux kernel structs.
- Identified **most influence KConfigs on forensics structs**.

… and **we release Volatility 3 profile dataset and code to the community!**
https://s3.eurecom.fr/datasets/

Questions?

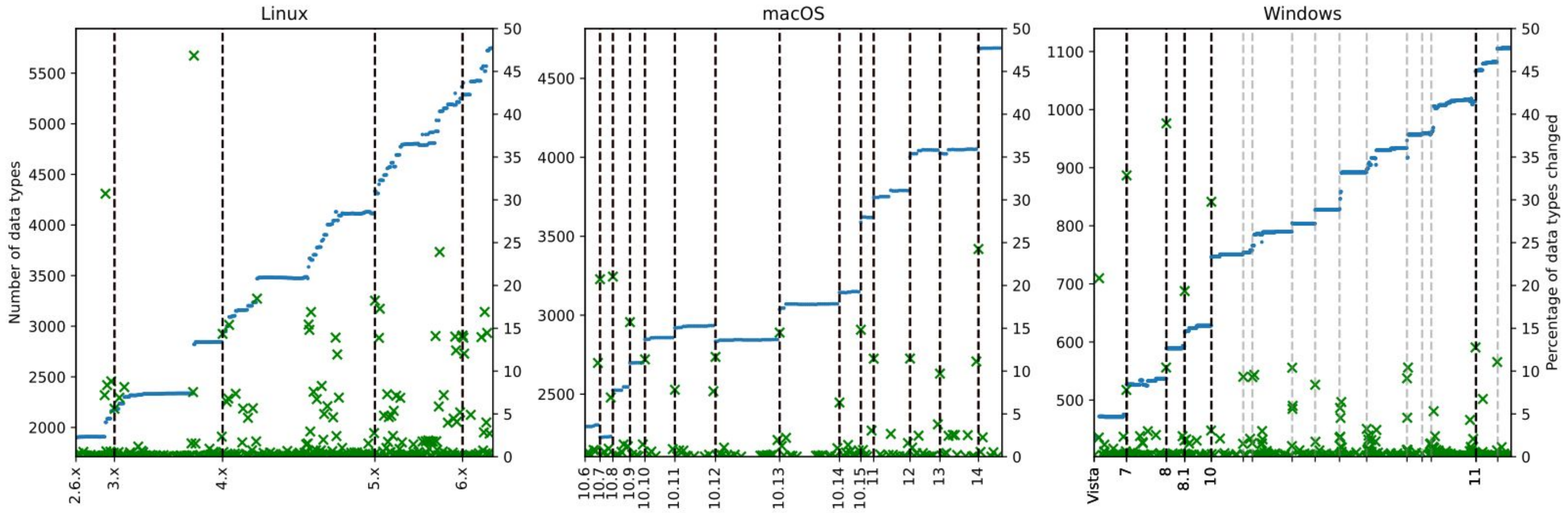
Appendix: Forensics Structures

Linux: address_space, anon_vma, cred, dentry, file, file_system_type, inet_sock, inode, kmem_cache, mm_struct, module, module_kobject, neighbour, neigh_table, nf_hook_ops, path, proto, resource, rtable, seq_operations, sk_buff, sock, sock_common, super_block, task_struct, timespec, tty_struct, vm_area_struct, vfsmount

macOS: cpu_data, dyld_all_image_infos, fileglob, fileproc, fs_event_watcher, inpcb, ifnet, kauth_scope, kmod_info, kmod_info_t, mac_policy_list, mac_policy_list_element, memory_object_control, mount, pmap, proc, protosw, queue_entry, session, sockaddr, socket, socket_filter, sysctl_oid, task, thread, ubc_info, _vm_map, vm_map_header, vm_map_links, vm_map_entry, vm_map_object, vm_object, vm_page, vnode, vnodeopv_desc, zone

Windows: _CM_KEY_NODE, _CMHIVE, _CONTROL_AREA, _DEVICE_OBJECT, _DISPATCHER_HEADER, _DRIVER_OBJECT, _EPROCESS, _ETHREAD, _FILE_OBJECT, _HHIVE, _HANDLE_TABLE, _HANDLE_TABLE_ENTRY, _HEAP_ENTRY, _KMUTANT, _KPCR, _KPRCB, _KPROCESS, _LDR_DATA_TABLE_ENTRY, _LUID_AND_ATTRIBUTES, _MM_AVL_TABLE, _MMADDRESS_NODE, _MMPFN, _MM_SESSION_SPACE, _MMVAD, _MMVAD_FLAGS, _OBJECT_HEADER, _OBJECT_SYMBOLIC_LINK, _OBJECT_TYPE, _PEB, _PEB_LDR_DATA, _POOL_HEADER, _POOL_TRACKER_BIG_PAGES, _POOL_TRACKER_TABLE, _RTL_USER_PROCESS_PARAMETERS, _SUBSECTION, _SECTION_OBJECT_POINTERS, _SEP_TOKEN_PRIVILEGES, _SHARED_CACHE_MAP, _SID_AND_ATTRIBUTES, _SID, _SID_IDENTIFIER_AUTHORITY, _TEB, _TOKEN, _VACB

The Development Cycle Influence the Struct Definitions



Blue dots: total number of C structs/unions

Green crosses: percentage of structs changed between two consecutive kernel versions

Linux: **after 4.x** random and changes, **rolling release model**

macOS: new types at **major releases**

Windows: “commercial code names” are major releases

Structs KConfig Dependency: Forensics Structs

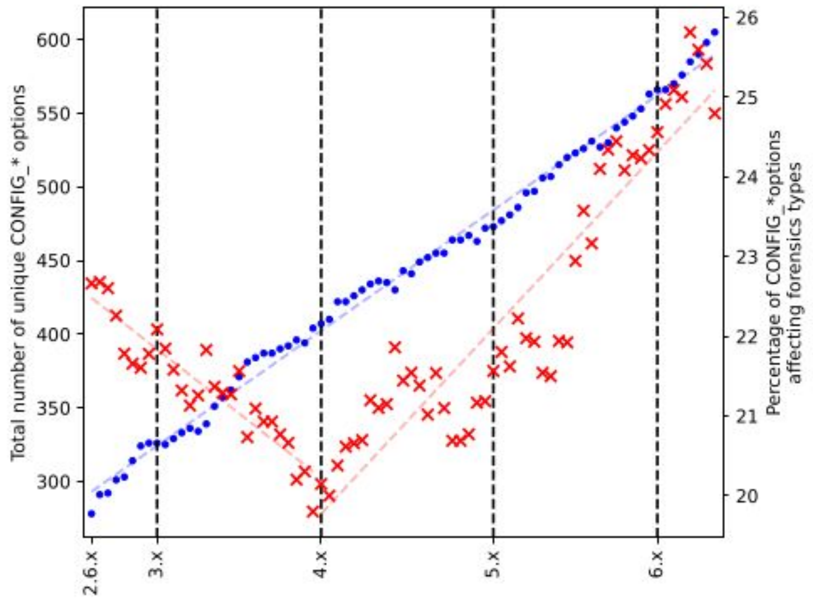
How the KConfig options affect forensics structures?

In blue: Total number of KConfig options

- ⇒ **Linearly increase** across the entire kernel history

In red: KConfig percentage affecting forensics structs

- ⇒ **Decreasing before 4.x and increase after it.**
- **85 (4.0) –> 150 (6.7)** different options


Total number of unique CONFIG_* options
Percentage of CONFIG_*options affecting forensics types
2.6.x
3.x
4.x
5.x
6.x