RandCompile Removing Forensic Gadgets from the Linux Kernel to Combat its Analysis

Fabian Franzen, Andreas Chris Wilhelmer, Jens Grossklags
Technical University of Munich

December 7, 2023

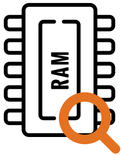
Outline

1. Current State of Memory Forensics
2. How to combat Modern Forensic Tools: RandCompile
3. Evaluation
4. Future Research & Conclusion

Current State of Memory Forensics

Intro

- **Memory-Forensic:** The science of deducting information about an operating system state out of a memory dump
- Allows to reason about
  - Process List
  - (Cryptographic-)Secrets
  - IPs/MAC-Addresses of devices in proximity
  - ...
- Complexity depends on available information.
  - Debugging Symbols of operating system



Recent Developments in Linux Memory Forensics

New Challenges for analysts:

- **Structure Layout Randomization** (since 2017)
  - Binary Layout of data structures is modified at compile time.
  - Primarily a Binary Exploitation defense, but effective against forensic tools

Recent Developments in Linux Memory Forensics

New Challenges for analysts:

- **Structure Layout Randomization** (since 2017)
  - Binary Layout of data structures is modified at compile time.
  - Primarily a Binary Exploitation defense, but effective against forensic tools

Research Progress:

- Tools are capable to deal with **Structure Layout Randomization**
- OS-agnostic tools
  - Certain implementation characteristics are shared between OSes
  - Operate with minimal additional information on MacOS, Linux, Windows, and other operating systems

Forensic Gadgets

We systematized and grouped last-generation tools by the essential OS artifacts used to enable their analysis (Forensic Gadgets):

- **Special Comm** - ”Special” strings allowing easy identification of the process list in the memory dump.

Forensic Gadgets

We systematized and grouped last-generation tools by the essential OS artifacts used to enable their analysis (Forensic Gadgets):

- **Special Comm** - ”Special” strings allowing easy identification of the process list in the memory dump.
- **Symbol Tables** - The OS maintaines a list of its own functions/global variables for dynamic loading of drivers and or additional functionality (eBPF, ftrace, …).

Forensic Gadgets

We systematized and grouped last-generation tools by the essential OS artifacts used to enable their analysis (Forensic Gadgets):

- **Special Comm** - ”Special” strings allowing easy identification of the process list in the memory dump.
- **Symbol Tables** - The OS maintaines a list of its own functions/global variables for dynamic loading of drivers and or additional functionality (eBPF, ftrace, ...).
- **ABI Constraints** - The compiled kernel code follows predictable patterns revealing location/layout of data structures (i.e. using offset revealing instructions).

Forensic Gadgets

We systematized and grouped last-generation tools by the essential OS artifacts used to enable their analysis (Forensic Gadgets):

- **Special Comm** - ”Special” strings allowing easy identification of the process list in the memory dump.
- **Symbol Tables** - The OS maintaines a list of its own functions/global variables for dynamic loading of drivers and or additional functionality (eBPF, ftrace, ...).
- **ABI Constraints** - The compiled kernel code follows predictable patterns revealing location/layout of data structures (i.e. using offset revealing instructions).
- **Order of Fields** - The data structure layout (especially without Structure Layout Randomization) is forseeable.

Forensic Gadgets

We systematized and grouped last-generation tools by the essential OS artifacts used to enable their analysis (Forensic Gadgets):

- **Special Comm** - ”Special” strings allowing easy identification of the process list in the memory dump.
- **Symbol Tables** - The OS maintaines a list of its own functions/global variables for dynamic loading of drivers and or additional functionality (eBPF, ftrace, ...).
- **ABI Constraints** - The compiled kernel code follows predictable patterns revealing location/layout of data structures (i.e. using offset revealing instructions).
- **Order of Fields** - The data structure layout (especially without Structure Layout Randomization) is forseeable.
- **Pointer Graph** - The pointers between the kernel objects form a characteristic graph revealing e.g. the process list uniquely out of the set of objects.

Systematization of Last Generation Tools

Tool	Year	Analysis Subject	FG 1: Special comm	FG 2: Symbol Tables	FG 3: ABI Constraints	FG 4: Order of Fields	FG 5: Pointer Graph	Recovery Scope
Linux-specific								
KATANA	2022	Offset Revealing Instructions		✗	✗			All structures
Trustzone Rootkit	2022	Kernel Runtime Data	✗					Selected structures
LOGICMEM	2022	Kernel Runtime Data	✗	✗		✗	✗	Selected structures
AUTOPROFILE	2021	Offset Revealing Instructions		✗	✗	✗		All structures
OS-agnostic								
FOSSIL	2023	Kernel Runtime Data	✗				✗	All structures
HYPERLINK	2016	Kernel Runtime Data	✗				✗	Selected structures

How to Combat Modern Memory Forensic Tools?

Design Goals of RandCompile

Harden **Linux** systems against automated forensic analysis

Design

	Forensic Gadgets					Transformation	
	FG-1	FG-2	FG-3	FG-4	FG-5	GCC Plugin	Manual
String and Pointer Encryption	✓				✓		✓
Better Data-Order Randomization				✓			✓
Externalize `printk` Format Strings			✓			✓	
Adding Bogus Parameters with Artificial Memory Accesses			✓			✓	

- Perform selected transformations on the kernel to remove four out five forensic gadgets.
  - two are automatically applied (by a compiler plugin)
  - two applied manually in form of kernel patch
- **Disclaimer:** Perfect Obfuscation is in general not possible! This is a hardening mechanism against automated tools.

ABI Randomization

Katana and AutoProfile target FG 3

- Offset Revealing Instructions reveal layout of data structures
- ABI mandates calling convention
  - Allows a structural matching of generated machine code with the source code

Example:

```
do_stuff(current->mm ❶, current->
    ↪ cred ❷, &g ❸);
```

↓

```
mov     rdx❸,0xffffffff82019c60
mov     rax,QWORD PTR gs:0x16d00
mov     rsi❷,QWORD PTR [rax+0x10]
mov     rdi❶,QWORD PTR [rax+0x440]
call    ffffffff811bacd0 <do_stuff>
```

ABI Randomization

Countermeasures by RandCompile

- Shuffle the order of the arguments at call site and implementation site
- Applied automatically to all functions through a compiler plugin.

Issues

- Functions with few parameters have few possibilities for randomization

Example:

```
do_stuff(current->cred ❶, current->mm
    ↪ ❷, &g ❸);
```

↓

```
mov     rdx❸,0xffffffff82019c60
mov     rax,QWORD PTR gs:0x16d00
mov     rsi❷,QWORD PTR [rax+0x440]
mov     rdi❶,QWORD PTR [rax+0x10]
call    ffffffff811bacd0 <do_stuff>
```

ABI Randomization

We can add bogus parameters to functions with few parameters

- This can be undone by an analysis tool that has access to the source code
- Also add bogus assembly code hurting performance

Example:

```
int64_t bogusstuff[6];
do_stuff(current->cred❶, current->mm ❷, bogusstuff[0]❸, &g❹, bogusstuff[3]❺, bogusstuff[5] ❻);
```

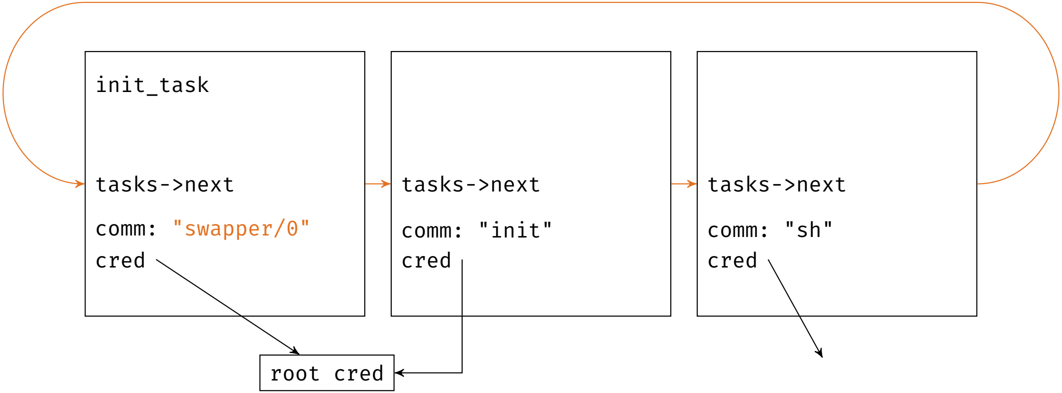
↓

```
mov     rcx❹,0xffffffff82019c60
mov     r8❺,QWORD PTR [rsp+0x18]
mov     r9❻,QWORD PTR [rsp+0x28]
mov     rax,QWORD PTR gs:0x16d00
mov     rsi❷,QWORD PTR [rax+0x440]
mov     rdx❸,QWORD PTR [rsp]
mov     rdi❶,QWORD PTR [rax+0x10]
call    ffffffff811bacd0 <do_stuff>
```

Pointer & String Encryption

HyperLink and Fossil analyse the pointer graph of kernel objects (FG 5).

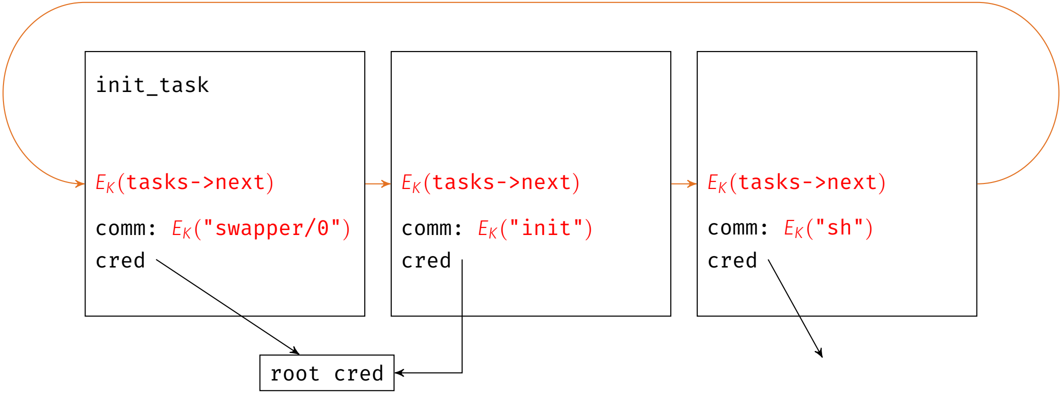
- e.g. the process information objects are connected by a linked list.
- first process in list contains well-known string (FG 1).



Pointer & String Encryption

HyperLink and Fossil analyse the pointer graph of kernel objects (FG 5).

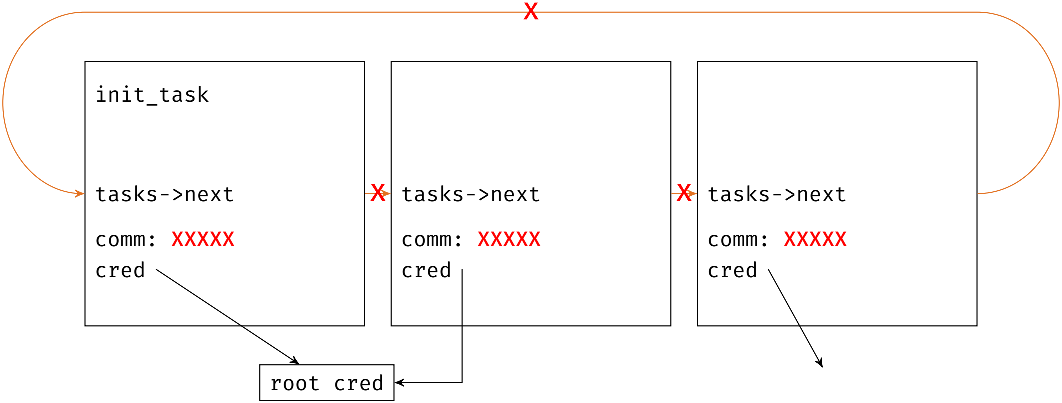
- e.g. the process information objects are connected by a linked list.
- first process in list contains well-known string (FG 1).
- Encrypt Pointers and Strings in process information objects



Pointer & String Encryption

HyperLink and Fossil analyse the pointer graph of kernel objects (FG 5).

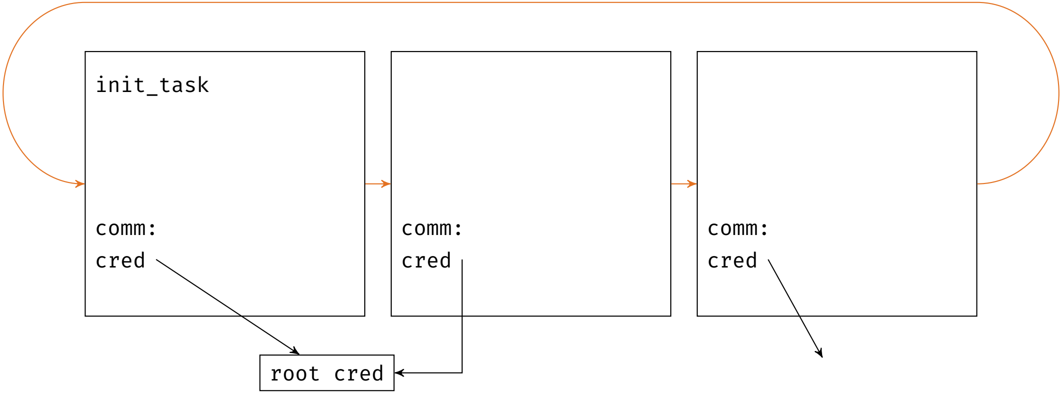
- e.g. the process information objects are connected by a linked list.
- first process in list contains well-known string (FG 1).
- Encrypt Pointers and Strings in process information objects



Pointer & String Encryption

HyperLink and Fossil analyse the pointer graph of kernel objects (FG 5).

- e.g. the process information objects are connected by a linked list.
- first process in list contains well-known string (FG 1).
- Encrypt Pointers and Strings in process information objects
  - Store Encryption Key as immediate value in the compiled machine code.



Evaluation

Effectiveness against Offset Revealing Instruction based Analysis

	Base	RandCompile (no bogus)	RandCompile (-printk, -memref)	RandCompile (-printk)	RandCompile (full)
List Modules	✓	✓	✓	✓	✓
Members reconstructed	2	2	2	2	2
Task Listing	✓	✗	✗	✗	✗
Members reconstructed	6	5	5	4	4
List Files	✓	✗	✗	✗	✗
Members reconstructed	16	15	8	7	7
Dmesg Log	✓	✓	✓	✓	✗

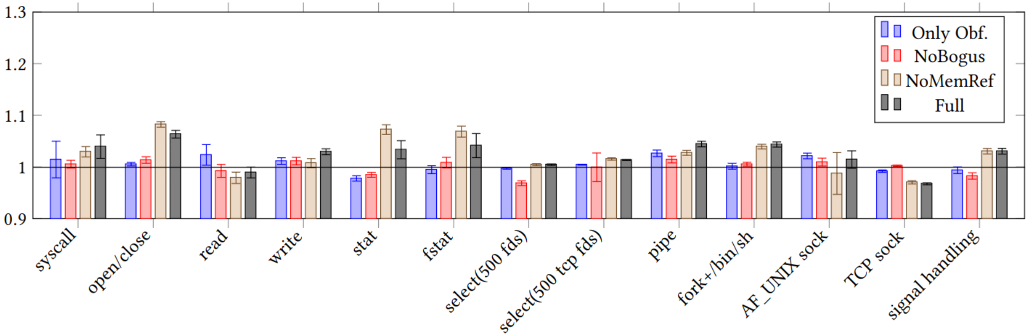
- We perform the core analysis of KATANA with and without RandCompile.
  - Already a single fault during reconstruction causes a fault!

Effectiveness against Kernel Runtime Data Analysis

- Encryption of the string `"swapper/0"` (FG-1) is most effective.
  - Stops LogicMem, Trustzone Rootkit, and HyperLink from operating
  - Fossil analysis performance is degraded. It depends on the analysts queries.
- Pointer Encryption
  - Degrades analysis opportunities of LogicMem, Trustzone Rootkit, and HyperLink further
  - Further degrades attack possibilities of Fossil
  - Future Work: Encrypt also other kernel pointers

Performance

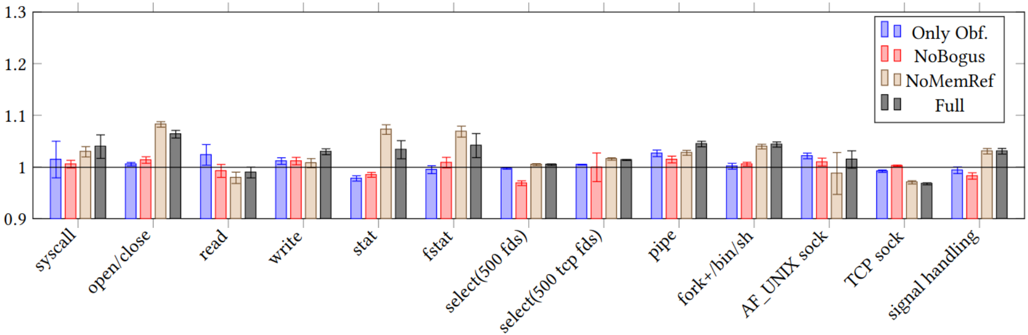
Results using the lmbench Microbenchmark (runtimes are normalized to 1):



Performance

Results using the lmbench Microbenchmark (runtimes are normalized to 1):

- Less than 1-3 percent overhead on average



Discussion

- Are you applying only sound transformations?
  - Yes. RandCompile does not change the semantic/core functionality of the Linux kernel.
- Does not confidential computing (CC) (like AMD-SEV) mitigate this problem?
  - RandCompile complements protection of CC approaches. I.e. AMD-SEV expects a Linux kernel to not trust his drivers.
- Can this be used as a binary exploitation defense?
  - Yes. In combination with Control Flow Integrity protections, it makes abusing existing kernel functions in ROP chains harder.
- Is it a problem that the defenses are applied at compile time?
  - Partially. Applying them during runtime would allow for more widespread use. Applying them at compile time adds diversity to the binary layout.

Conclusion

Conclusion

- RandCompile is an obfuscation tool for the Linux Kernel to harden it various memory forensic tools.
- It is effective against modern forensic analysis tools.
- It completes and extends the Structure Layout Randomization, a mainlined Linux kernel feature.

We have source code!