Leveraging Relocations in ELF-binaries for Linux Kernel Version Identification

Manish Bhatt, Irfan Ahmed

University of New Orleans

DFRWS 2018

Outline

- Introduction
- Design Rationale
- Implementation
- Evaluation
- Possible Optimization Measures
- Conclusion
- Acknowledgement

Introduction

- Forensics Challenges in analysing Linux memory dumps:
  - **Address Randomisation**
  - **Kernel Version Identification**
- Proposed Solution:
  - Using Relocation Entries
  - **codeid-elf**

Design Rationale (1)

- S be one of the obtained signatures. S=<ptr, offset>
- Let P(i) refer to the i^{th} page in memory.
- Base address in Memory be Bm
- Base address in disk be Dm, then the following is true:

$$B_m - D_m = RandomizedOffset = ptr - S.pointer$$

Design Rationale (2)

In Memory Kernel Code

```
1b001000: 8b0d c01e b71b f686 1102 0000 4075 160f      ............@u..
1b001010: 0115 c61e b71b b818 0000 008e d88e c08e      ................
1b001020: e08e e88e d08d a100 0000 40fc 31c0 bf00      ..........@.1...
```

In Disk Kernel Code

```
00000000: 8b0d c01e b701 f686 1102 0000 4075 160f      ............@u..
00000010: 0115 c61e b701 b818 0000 008e d88e c08e      ................
00000020: e08e e88e d08d a100 0000 40fc 31c0 bf00      ..........@.1...
```

Relocation Entries

No.	Offset	Pointer	Symbol
1	c1000002	c1000002	initial_stack
2.	c1000012	c1000002	boot_gdt_descr

Design Rationale

No.	In Disk	In Memory	Difference
1.	0x01b71ec0	0x1bb71ec0	0x1a000000
2.	0x01b71ec6	0x1bb71ec6	0x1a000000

Fingerprinting and Derandomization

- Generate Signatures for all known versions.
- Match **all the signatures** to a memory dump for **version identification**.
- Match **single page** for address derandomization.

Data: output of readelf -r program
Result: List of ⟨*offset*, *pointer*⟩ signature tuples

```
initialization;
filehandler = Open Output File;
while every line in the output do
    if (line contains "R_386_32") then
        filehandler.write(line.offset+ " " + line.value) ;
    else
        continue;
    end
end
```

Algorithm 1: Algorithm to extract signatures.

Implementation (1)

- Leveraging known utilities for signature extraction: **readelf**
- Implemented in a mixture of C and Python
- Only relevant relocation entry is **R_386_32**
  - According to standards, direct replacement of relocated address
  - As opposed to the **PC** relative address

Implementation (2): PAGE DETECTION

- Could have just done the whole executable at once. Why?
  - All pages were not detected
    - Some pages no relocs (less than 2%)
    - Some relocs changed at runtime
  - Therefore, page-level analysis was performed

Implementation (3): Version Detection

- Run for all known versions
- Pick the version with the highest number of page matches
- Just want derandomization?
  - Just extract a single page or a couple of pages
  - Faster as opposed to trying the whole executable

Implementation (4): Workflow

View Relocation Entries using "readelf -r" utility on the Kernel Binary → Extract offsets in ".rel.text" with type R_386_32. → Extract pointer values pertaining to those offsets from .text section of kernel binary. → Group offsets based on page addresses → Use Matching algorithm using signature page-groups and Memory Dump

Gathering Data

- Avoid compiling a lot of kernels
- Easier to create the **vmlinux** executable inside the kernel
- Memory snapshot via vmware **vmss2core** utility
- Created 22 different kernel versions this way
- Standard kernels used to evaluate

Custom Kernels?

- Are they different? As long as
  - They have elf binaries
  - They have relocs of type **R_386_32**
- Fingerprinting the executable using executable structure and properties
- Expectedly kernel independent

Evaluation (1): Metrics

- **Relocation Prevalence** and **Code Coverage**? Snip:

Vmlinux Kernel Version	Relocation Prevalence	Ratio of Pages with no Relocation Entries
4.4.0-22	55.4482	0.0050
4.4.0-24	55.4314	0.0056
4.10.0-14	54.0464	0.0047
4.10.0-14-lowlatency	53.2292	0.0051
4.10.0-19	54.0825	0.0042
4.10.0-19-lowlatency	53.2381	0.0061
4.10.0-20	54.0825	0.0042
4.10.0-21	53.9475	0.0070
4.10.0-22	53.9418	0.0051
4.10.0-24	53.9442	0.0042
4.10.0-26	53.9584	0.0037
4.10.0-27	53.9584	0.0037

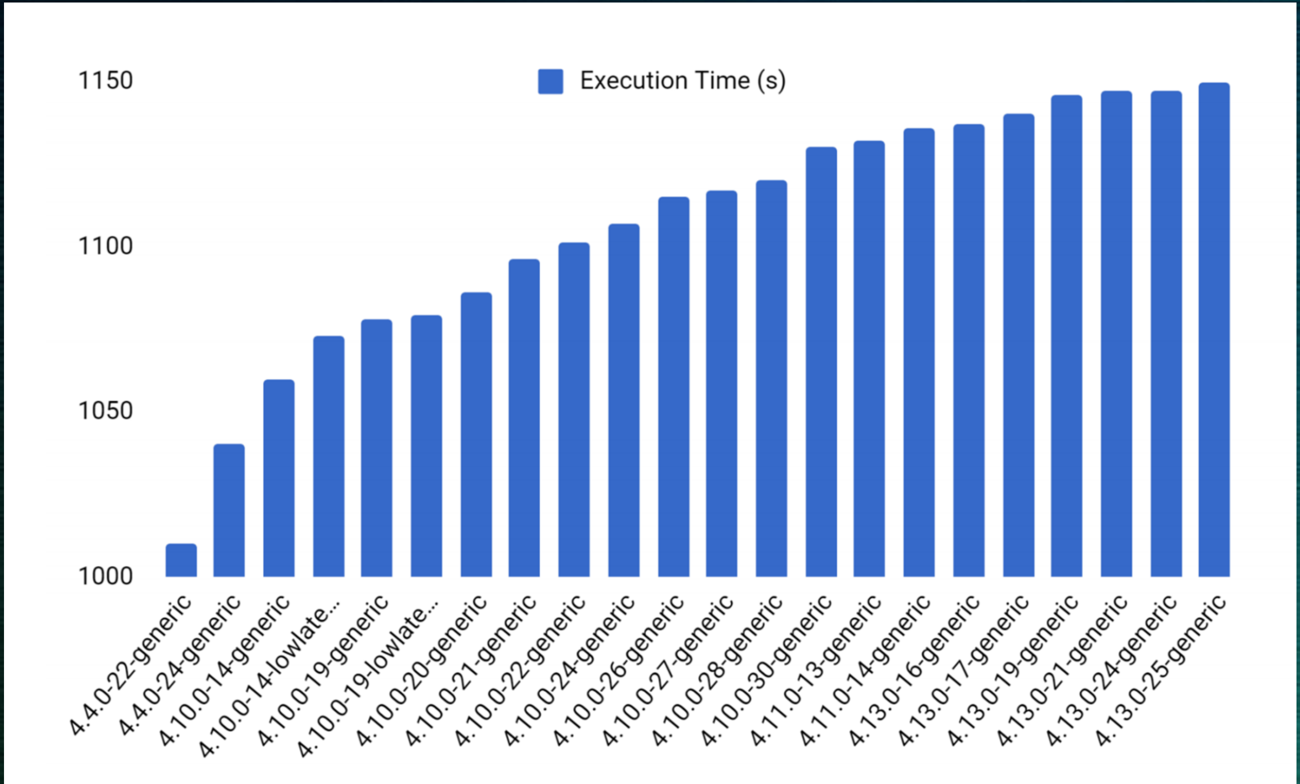
Evaluation (2): Page Hit Rate and Detection

Vmlinux Kernel Version	Number of Generated Signatures	Size of Kernel Code in MB	Total Number of Pages in code	Page Hit Rate	Base Address of Kernel	Derandomisation	Kernel Identification
4.4.0-22	108734	7.6640	1961	99.47	0xc100000	√	√
4.4.0-24	108812	7.6708	1963	99.43	0xc100000	√	√
4.10.0-14	113984	8.2387	2109	99.52	0xdc00000	√	√
4.10.0-14-lowlatency	113059	8.2976	2124	99.48	0xc600000	√	√
4.10.0-19	114060	8.2419	2109	99.57	0xd300000	√	√
4.10.0-19-lowlatency	113131	8.3010	2125	99.38	0xc800000	√	√
4.10.0-20	114060	8.2419	2109	99.57	0xd100000	√	√
4.10.0-21	114207	8.2701	2117	99.29	0xd000000	√	√
4.10.0-22	114195	8.2702	2117	99.48	0xcc00000	√	√
4.10.0-24	114200	8.2704	2117	99.57	0xc900000	√	√
4.10.0-26	114230	8.2707	2117	99.62	0xc800000	√	√
4.10.0-27	114230	8.2707	2117	99.62	0xda00000	√	√
4.10.0-28	114230	8.2708	2117	99.62	0xdb00000	√	√
4.10.0-30	114232	8.2713	2117	99.43	0xca00000	√	√

Evaluation (3): Fingerprint Similarity

Kernel **A**	Kernel **B**	Number of Common Signatures	Sim_{AB}
4.10.0-21-generic	4.10.0-28-generic	23	0.0201
4.10.0-21-generic	4.10.0-22-generic	2107	1.8449
4.10.0-21-generic	4.10.0-14-generic	69	0.0604
4.10.0-21-generic	4.10.0-24-generic	2097	1.8361
4.13.0-25-generic	4.13.0-21-generic	99	0.0882
4.10.0-28-generic	4.10.0-30-generic	32607	28.545
4.10.0-28-generic	4.10.0-20-generic	20	0.0175
4.10.0-28-generic	4.10.0-27-generic	29530	25.8514
4.10.0-28-generic	4.10.0-26-generic	2273	1.9898
4.10.0-30-generic	4.10.0-27-generic	29285	25.6364
4.10.0-30-generic	4.10.0-24-generic	102	0.0893
4.10.0-30-generic	4.10.0-26-generic	2163	1.8935
4.13.0-16-generic	4.13.0-21-generic	60	0.0534
4.10.0-20-generic	4.10.0-22-generic	149	0.1306
4.10.0-27-generic	4.10.0-22-generic	105	0.0919
4.10.0-27-generic	4.10.0-14-lowlatency	43	0.0376
4.13.0-19-generic	4.13.0-17-generic	214	0.1904

Evaluation (4): Performance Measures



Optimisation Measures

- Resample Fingerprints
  - Intelligently create a subset of signatures
    - Decreasing the number of signatures
    - Without sacrificing code coverage
    - Decreasing relocation prevalence
- Implementing completely in C
- For kernels, scan just areas where kernels are likely to be found. (Kernel Ranges)

Takeaways

- Very simple to understand and implement
- Low number of pages with no relocation entries
- High code coverage and prevalence
- Fingerprints are mostly unique
  - In experiments, except for **4.11.0-14-generic** and **4.11.0-13-generic**
- Technique works for both derandomization and version identification

Comparison to Similar Works

Method	Similarity of Derived Signatures	Signatures Derived From Memory Dumps	ASLR Support	Version Detection Accuracy
OS-Sommelier	Low	Yes	√	100%
k-BinID	High	No, derived at Run-Time using VMI	√	100%
Image-Based Kernel-Fingerprinting	High/Low depending on Similarity in implementation	No	χ	100%
codeid-elf	Low	No	√	100%

Conclusion

- High page hit rate
- Always derandomized addresses correctly
- Over 99% page hit rate
- Similarity of Relocations Low
- Versions detected correctly every time
- Extendable to other executables

Future Works

- Extend the relocs method to other user-space binaries
- Test it on other custom kernels
- Extend this to other architectures like ARM
- Use this in preliminary Malware Analysis

Acknowledgements

This work was supported by NSF grant #1623276

Questions?