Fine-grained Address Space Layout Randomization on Program Startup

19 november 2017

Ivannikov Institute for System Programming of the RAS

Introduction

Introduction

- Software errors and vulnerabilities are inevitable.

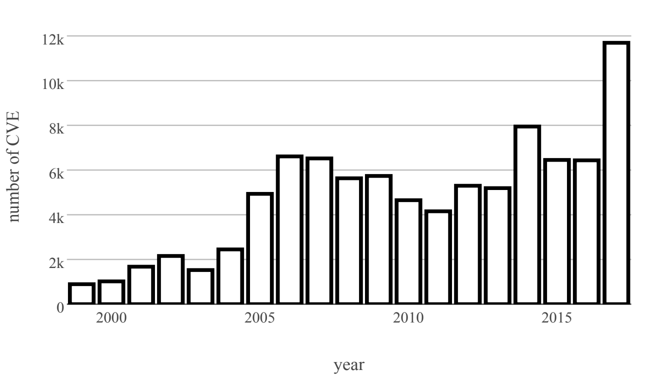


Figure 1: Vulnerabilities By Year

Introduction

- Bad guys always try to exploit vulnerabilities.
- Existing prevention techniques (DEP, ASLR, PaX) are not enough.

CVE-2013-1690 used by the FBI to de-anonymize users of the Tor browser.

Goals

Fine-grained address space layout randomization:

- runs at program startup,
- operates on function level,
- applicable to the whole system,
- for Linux x86-64.

Limitations:

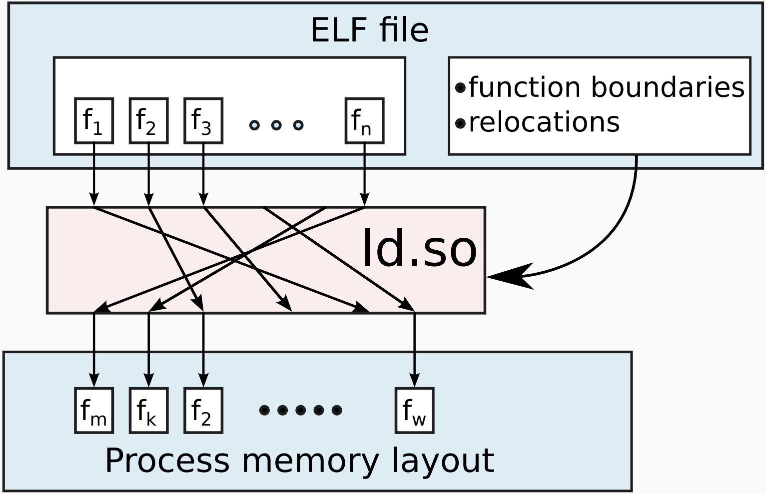
- no runtime re-randomization,
- source code is required,
- user space only randomization.

Related Works

- Selfrando
- Oxymoron, Pagerando
- Runtime re-randomization
- Compile time diversity

Design

Scheme


ELF file
f_1 f_2 f_3 f_n
•function boundaries
•relocations
ld.so
f_m f_k f_2 f_w
Process memory layout

Linking Stage

Linker creates an auxiliary section in the ELF file which contains:

- entry point,
- function boundaries (start, length and alignment),
- relocations (address and type, target and source function ids).

Startup Stage

Modify the dynamic linker/loader to:

- search the special section,
- change permission RW -> RE,
    - to adapt: PaX
- permute the function order.

Experimental Evaluation

Environment

- Intel i7-4790
- 16 GB RAM
- CentOS 7
- Linux 3.10 + PaX
- gcc 4.8.5
- binutils 2.23
- glibc 2.17

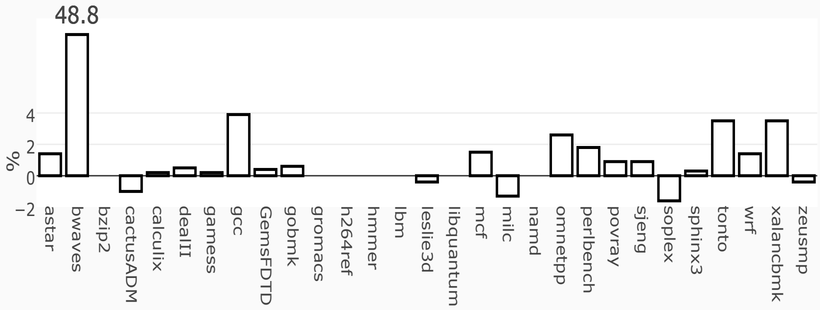


Figure 2: Performance Slowdown for SPEC2006

Startup Time Evaluation

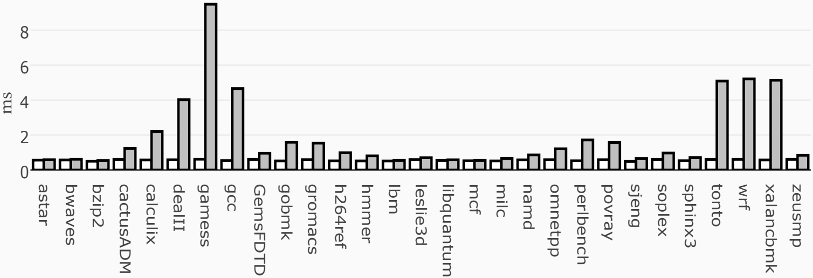


Figure 3: Startup Time Slowdown for SPEC2006

ELF Size Evaluation

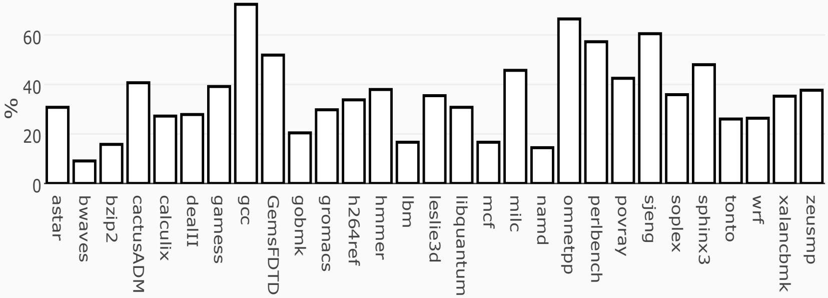


Figure 4: File Size Slowdown for SPEC2006

Comparison of Randomizing Techniques

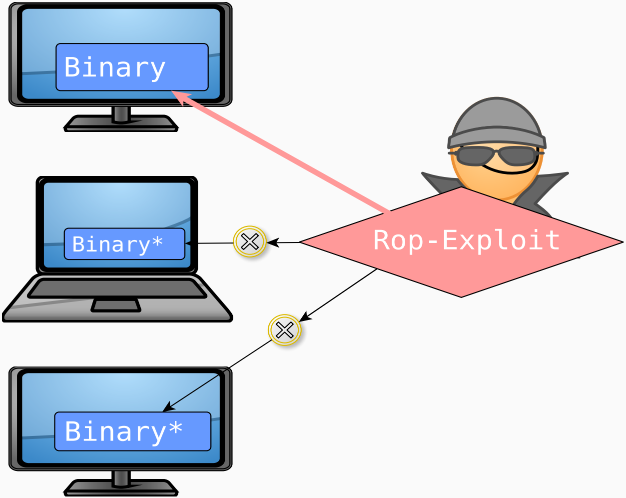
	fgASLR	Selfrando	Oxymoron	Runtime
performance, %	~2	~2	~2	100
startup time	~5 ms	?	0	0
file size, %	30	?	2	?
granularity	function	function	pages	?
sharing	no	no	yes	no

Exploit Prevention Evaluation

Testing Scheme

1. Prepare the test files.
2. Search and classify rop-gadgets.
3. Estimate the survival probability for gadget.
4. Create rop-chains for test files.
5. Check created rop-chains on randomized files.

Threat Model



Test Files

- CentOS 7
- no-PIE ELF
- /usr/bin/*
- /usr/sbin/*

overall 470 files.

Generation of Randomized Files

- Store runtime address space layout in core dumps by modified gcore (gbd).
- 10 core dumps generated for each test files.

Overall 470 * 11 ELF files for analysis.

Rop-gadget Searching and Classifying Tool

- Searches all rop-gadgets and classifies them by semantic types.
- Stores the result in gadget database.

Gadget database stores gadget descriptions:

- address,
- type,
- parameters,
- side effects.

Survival Probability Estimation

$$\frac{\sum_{j=1}^{m} \left(\frac{\sum_{i=1}^{n_j} k_i^j}{10 n_j} \right)}{m} = 0.05 \tag{1}$$

m - number of files,

n_j - number of gadgets in j file,

k_i^j - number of files where g_i^j stayed in place.

Survival Gadgets Metric

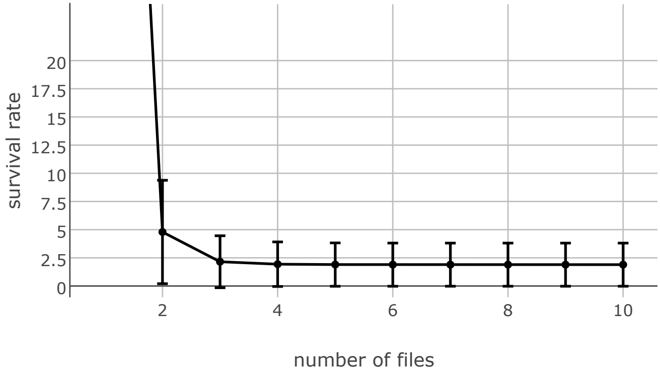


Figure 5: Rate of Survived Gadgets by Population Size

Rop-chains for Testing.

It is possible to create rop-chains with gadget database.

1. foo();
2. foo(1);
3. foo(1, 2);
4. foo(1, 2, 3);
5. system("/bin/sh");

Created Examples for Sed

foo(1, 2, 3);

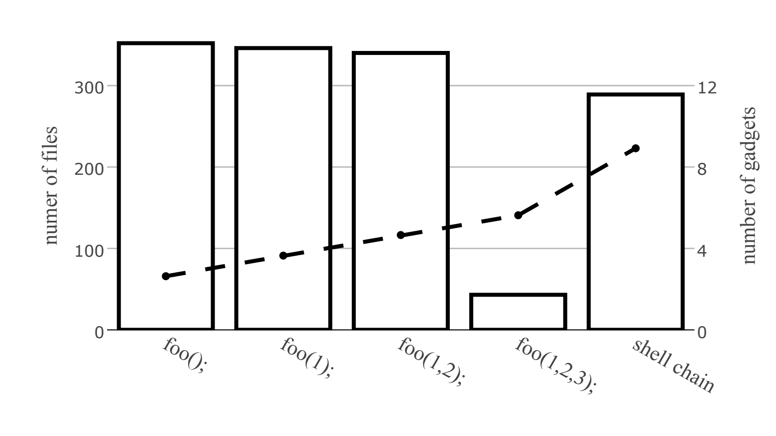
```
0x40b99c -> POP RBX ; RET
0x402e8c -> MOV RAX, RBX ; POP RBX ; RET
0x401de2 -> POP RDX ; RET 0021h
0x40968b -> POP RSI ; RET
0x40bd23 -> POP RDI ; RET
0x4027e7 -> JMP RAX
```

Created Examples for Sed

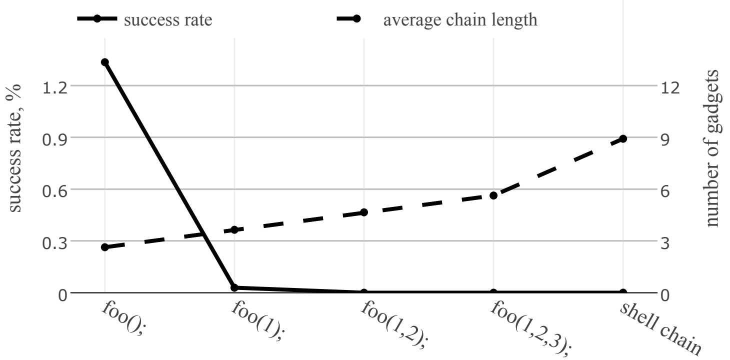
system("/bin/sh");

```
0x401de2 -> POP RDX ; RET 0021h
0x40bd23 -> POP RDI ; RET
0x40ace4 -> MOV QWORD PTR [RDI + 30h], RDX ;
            ADD RSP, 0000000000000008h ; RET
0x40b99c -> POP RBX ; RET
0x402e8c -> MOV RAX, RBX ; POP RBX ; RET
0x40bd23 -> POP RDI ; RET
0x4027e7 -> JMP RAX
```

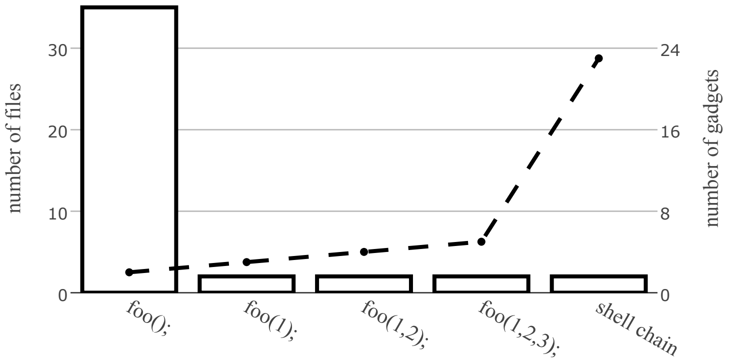
Created Rop-chains Statistics



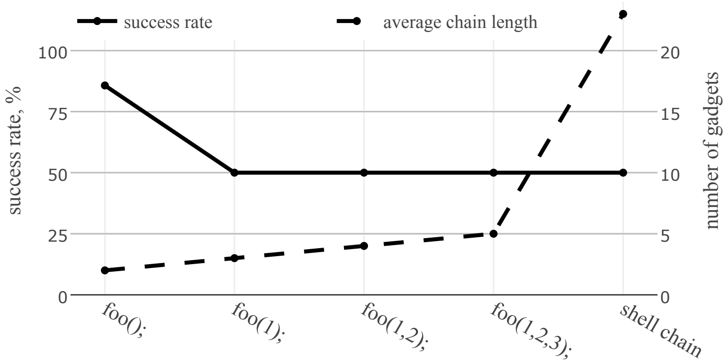
Rop-chain Success Rate for Randomized Files



No .text Created Rop-chains Statistics



No .text Rop-chain Success Rate for Randomized Files



Conclusion

Future Works

- Fix debug information.
- Randomize executable sections outside .text as well.

Conclusion

- Fine-grained ASLR on program startup at function level for Linux x86-64.
- Average performance slowdown ~ 2 %.
- Successfull rop-based attacks mitigation.

Thank You!