LR²: Leakage-Resilient Layout Randomization for Mobile Devices

Kjell Braden†§, Stephen Crane‡, Lucas Davi†, Michael Franz*, Per Larsen*‡, Christopher Liebchen†, Ahmad-Reza Sadeghi†

†TU Darmstadt  §EURECOM  §EURECOM  ‡Immunant, Inc.
PROTECT

ALL THE NATIVE

CODES
Today’s Exploits & Mitigations

Application

- Code
- Call *ptr
- Function A
- Data
- Code Pointer
Today’s Exploits & Mitigations

Application

Code

call *ptr

Function A

Data

Code Pointer

Overwrite
Today’s Exploits & Mitigations

- Address Space Layout Randomization

Application

Data

Code Pointer

Overwrite
Today’s Exploits & Mitigations

- Address Space Layout Randomization

![Diagram](image)

- Code-Pointer Disclosure [Serna BH USA’12]

- Code Pointer Disclosure

Read then Overwrite
Today’s Exploits & Mitigations

- Address Space Layout Randomization
- Fine-grained Code Randomization

- Code-Pointer Disclosure [Serna BH USA’12]

Data

Code Pointer

Read then Overwrite
Today’s Exploits & Mitigations

- Address Space Layout Randomization
- Fine-grained Code Randomization

- Code-Pointer Disclosure [Serna BH USA’12]
- JIT-ROP [Snow et al. IEEE S&P’13]
Today’s Exploits & Mitigations

• Address Space Layout Randomization
• Fine-grained Code Randomization
• Execute-only Memory

Application

Execute only

Data

Code Pointer

- Code-Pointer Disclosure [Serna BH USA’12]
- JIT-ROP [Snow et al. IEEE S&P’13]

Read then Overwrite
Today’s Exploits & Mitigations

- Address Space Layout Randomization
- Fine-grained Code Randomization
- Execute-only Memory

- Code-Pointer Disclosure [Serna BH USA’12]
- JIT-ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

Read then Overwrite
Today’s Exploits & Mitigations

- Address Space Layout Randomization
- Fine-grained Code Randomization
- Execute-only Memory
- Code-Pointer Hiding

- Execute only

- Code-Pointer Disclosure [Serna BH USA’12]
- JIT-ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

Read then Overwrite
Today’s Exploits & Mitigations

Readactor [IEEE S&P’15]
Readactor++ [CCS’15]

- Address Space Layout Randomization
- Fine-grained Code Randomization
- Execute-only Memory
- Code-Pointer Hiding

Application

- Execute only
- Execute only

- Code-Pointer Disclosure [Serna BH USA’12]
- JIT-ROP [Snow et al. IEEE S&P’13]
- Isomeron (Attack) [Davi et al. NDSS’15]

Code Pointer

Read then Overwrite
Execute-only Memory

Desktop/Server

- Readactor [IEEE S&P’15]
- XnR [CCS’14]
- HideM [CODASPY’15]

Memory Virtualization

- MMU
- TLB-Splitting

Execute-Only Memory Support
Execute-only Memory

Application

Application

Execute-Only Memory Support

Mobile
Execute-only Memory

This Talk:
Execute-only Memory without Hardware Support
Threat Model

- Read Memory (Information Disclosure)
- Write Memory (Memory Corruption Vulnerability)
- Perform Computations (Scripting Engine or Locally)
- Cannot Inject New Code (DEP, W^X)
LR²: Leakage-Resilient Layout Randomization
LR² Overview

• Fine-grained Code Randomization

• Software eXecute-only Memory (XoM)

• Code-Pointer Hiding
  • Return Addresses
  • Forward Pointers
LR² Overview

• Fine-grained Code Randomization

• Software eXecute-only Memory (XoM)

• Code-Pointer Hiding
  • Return Addresses
  • Forward Pointers
Software XoM: Idea

- Read [address]
- Data A
- Randomized Code

Application
Software XoM: Idea

Sandboxing

Read [address]

Application

Randomized Code

Data A
Software XoM: Design
Software XoM: Design

2 GB

2 GB

Code

Data

Application

Kernel

Code B

Code A

Trampolines

Guard

Data A

Data B

Stack

Heap

Sandboxing Read Instructions

r1 <- addr
r1 <- r1 & 0x7FFFFFFF
r0 <- [r1]
Code-Pointer Hiding: Return Addresses
Code-Pointer Hiding: Return Addresses

Function B
- PUSH LR
- CALL Function C
- POP LR
- Return

B1

Stack
- Caller RA

LR
Address B1
Code-Pointer Hiding: Return Addresses

- Return Addresses
  - LR
  - PUSH LR
  - Dec(LR, Key_B)
  - CALL Function C
  - POP LR
  - Return
  - Stack
  - Enc(LR, Key_B)
  - Address B1
Sandboxing Reads: Optimizations
Optimizations: Loops

\[ \text{r0} \leftarrow \text{address} \]

\[ \text{For } i \leftarrow 0 ; i < X ; ++i \]

\[ \text{Mask } \text{r0} \]

\[ \text{r1} \leftarrow [\text{r0}] \]
Optimizations: Loops

r0 <- address
Mask r0
For i <- 0 ; i < X ; ++i
r1 <- [r0]
Optimizations: Loops

\[ \text{For } i < 0 ; i < X ; ++i \]

\[ r1 <- [r0 + i] \]
Optimizations: Loops

For $i < 0; i < X; ++i$

$r0 \leftarrow \text{address}$

Mask $r0$

$r1 \leftarrow [r0 + i]$
Implementation

• Kernel
  • Stack and Heap Allocations

• Loader
  • Code and Data Sections

• Compiler
  • Sandbox Read Instructions
Evaluation

• Security:
  • Code-Reuse Attacks: Function Permutation
  • Direct disclosure: Execute-only Memory
  • Indirect disclosure:
    • Code-pointer Hiding
    • Code/Data section decoupling

• CPU: Nvidia Tegra Logan K1
• Performance:
  • 6.6% run-time overhead
  • 5.6% space overhead
SPEC CPU 2006

- Pointer Hiding
- Restricted Register-Register Addressing
- Software XoM
- Code and Data Section Decoupling
- Full LR2
LR\(^2\) and Software Fault Isolation (SFI)

- Different Threat Models
  - SFI isolates untrusted code
  - LR\(^2\) protects trusted code

- LR\(^2\) can protect multiple load instructions by masking one address

- SFI sandboxes write and branch instructions
Conclusion

• First pure software execute-only memory technique

• Optimized return address protection scheme

• Performance and security matches state-of-the-art solutions requiring special, high-end hardware