RevARM: A Platform-Agnostic ARM Binary Rewriter for Security Applications

Taegyu Kim, Chung Hwan Kim, Hongjun Choi, Yonghwi Kwon, Brendan Saltaformaggio, Xiangyu Zhang, Dongyan Xu

Purdue University
*NEC Laboratories America
+Georgia Tech
Security of ARM platforms

• ARM platforms have recently gained popularity
  • Mobile phones, IoT, CPS, and etc
• However, many security needs arise
ARM Binary Instrumentation

• How to make systems secure?
  • Add security logics via instrumentation

• Source instrumentation
  • Source codes are not always available
    • Legacy program, closed sources..

• Binary instrumentation
  • Limited capabilities of existing techniques

We need a solid ARM binary rewriting technique
Requirements of ARM Binary Rewriter

• Address ARM-specific instrumentation challenges

• Low overhead for resource-scarce systems
  • Most ARM-based platforms have 1) small memory + 2) low computing power

• Instrumentation at arbitrary code locations
ARM-specific Challenges

• Compare with state-of-the-art rewriters
  • Most works focus on x86
  • SecondWrite
    • Requirement: Binary → LLVM IR
    • IR Transformation is not maintained in the recent LLVM
    • Due to high failure rate
  • Dyninst
    • Support the 64 bit architecture
    • Available version = Experimental version

• RevARM overcome ARM-specific challenges

<table>
<thead>
<tr>
<th>Rewriter</th>
<th>Target Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>BISTRO</td>
<td>x86</td>
</tr>
<tr>
<td>Uroboros</td>
<td>x86</td>
</tr>
<tr>
<td>Dyninst</td>
<td>x86</td>
</tr>
<tr>
<td>Pebil</td>
<td>x86</td>
</tr>
<tr>
<td>REINS</td>
<td>x86</td>
</tr>
<tr>
<td>PSI</td>
<td>x86</td>
</tr>
<tr>
<td>SecondWrite</td>
<td>LLVM</td>
</tr>
<tr>
<td>Dyninst</td>
<td>ARM 64bit (Experimental)</td>
</tr>
<tr>
<td>RevARM</td>
<td>ARM 32bit</td>
</tr>
</tbody>
</table>
## Binary Instrumentation Approaches

<table>
<thead>
<tr>
<th></th>
<th>Insertion-based</th>
<th>Detour-based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instrumentation Type</strong></td>
<td>Insert new codes inline</td>
<td>Jump to new codes</td>
</tr>
<tr>
<td><strong>Control Flow</strong></td>
<td>Preserved</td>
<td>Altered</td>
</tr>
<tr>
<td><strong>Overhead</strong></td>
<td>Lower</td>
<td>Higher</td>
</tr>
</tbody>
</table>
Insertion-based vs. Detour-based

Insertion-based approach

Original

1
2
3

Additional Code

1'

Rewritten Binary

1
2
3

Rewriting

Detour-based approach

Original

1
2
3

Jump Code (Redundant)

Rewriting

Additional Code

1'

End of code section

Rewritten Binary

1
3

End of code section
Low Overhead

Insertion-based approach

Original
1
2
3
Additional Code
1’

Rewritten Binary
1’
2
3

Rewriting

Detour-based approach

Original
1
2
3
Additional Code
1’

Rewritten Binary
1
2
3

Jump Code (Redundant)

Rewriting

End of code section

Redundant Jump

Run-time overhead

Space overhead
Instrumentation at Arbitrary Code Locations

- A four-byte jump instruction used to alter an original control flow
- Jump instruction may overwrite multiple original instructions
  - Incorrect control flows

Before instrumentation

- Insertion-based approach
  - A four-byte jump instruction
  - Jump instruction may overwrite multiple original instructions
  - Incorrect control flows

- Detour-based approach
  - Long Jump
  - Then, where is B??

Then, where is B??
Challenges of Insertion-based Approach

- RevARM: Addresses four ARM-specific challenges
  - C_1: If-Then instruction
  - C_2: Branch table instruction
  - C_3: Direct access to the program counter
  - C_4: Run-time instruction mode switching
C1: If-Then Instruction

- Conditionally execute following instructions
- Work like if-else statement

First following instruction cannot take “else” condition

![Diagram showing if-then instruction with conditions and actions](image)
C2: Branch Table Instruction

- TBB, TBH, LDR PC represent “switch statement”
- Reference range: TBB < TBH < LDR PC

One-byte relative address for each case

Two-byte relative address for each case

---

Branch Table

8 bytes were inserted

---
C2: Branch Table Instruction

- Q: What if even TBH range is insufficient?

Two-byte relative address for each case

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8E60</td>
<td>ADR R4,PC,#0x8</td>
<td>default</td>
</tr>
<tr>
<td>0x8E62</td>
<td>LDR PC,[R4,R2,LSL#2]</td>
<td></td>
</tr>
<tr>
<td>0x8E64</td>
<td>ALIGN 4</td>
<td></td>
</tr>
<tr>
<td>0x8E68</td>
<td>DCD 0x8E78+1</td>
<td>case0</td>
</tr>
<tr>
<td>0x8E6C</td>
<td>DCD 0x8E80+1</td>
<td>case1</td>
</tr>
<tr>
<td>0x8E70</td>
<td>DCD 0x8E88+1</td>
<td>case2</td>
</tr>
<tr>
<td>0x8E74</td>
<td>DCD 0x908C+1</td>
<td>default</td>
</tr>
<tr>
<td>0x8E78</td>
<td>LDR R4,[R1]</td>
<td>case0</td>
</tr>
<tr>
<td>0x908C</td>
<td>LDR R4,[R3]</td>
<td>default</td>
</tr>
</tbody>
</table>

Four-byte absolute address for each case

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8E60</td>
<td>TBH [PC,R2,LSL#1]</td>
<td></td>
</tr>
<tr>
<td>0x8E64</td>
<td>DCW 0x04</td>
<td>case0</td>
</tr>
<tr>
<td>0x8E66</td>
<td>DCW 0x0C</td>
<td>case1</td>
</tr>
<tr>
<td>0x8E68</td>
<td>DCW 0x10</td>
<td>case2</td>
</tr>
<tr>
<td>0x8E6A</td>
<td>DCW 0x12</td>
<td>default</td>
</tr>
<tr>
<td>0x8E6C</td>
<td>LDR R4,[R1]</td>
<td>case0</td>
</tr>
<tr>
<td>0x8E84</td>
<td>LDR R4,[R3]</td>
<td>default</td>
</tr>
</tbody>
</table>

200 bytes were inserted
Other Challenges

• C3: Direct access to the program counter
  • PC can be used as a general register
    • e.g., MOV, PC ← 0x080000000 / LDR, PC [R1]
  • Handle all PC access instructions

• C4: Run-time instruction mode switching
  • All code addresses are aligned in 2
  • 1st bit indicates the instruction mode
    • 1st bit = 1 → Thumb mode
    • 1st bit = 0 → ARM mode
Evaluation of RevARM

• Experimental setup
  • iPhone 5S (iOS 10.0.2)
  • 3DR iRiS+ (ArduPilot with NuttX)

<table>
<thead>
<tr>
<th>Board</th>
<th>Pixhawk (STM32F427 with FPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>ARM Cortex-M4 168Mhz</td>
</tr>
<tr>
<td>Memory</td>
<td>256KB SRAM, 2MB flash memory</td>
</tr>
</tbody>
</table>

• Disassembler
  • IDA Pro 6.8
Evaluation of RevARM

• Effectiveness
  • Case 1: ROP defense
  • Case 2: Software fault isolation
  • Case 3: Run-time status monitoring for flight controllers
  • Case 4: Function patching

• Performance impact
  • Run-time overhead: 3.2%
  • Space overhead: 1.3%
Case Study 1&2

- Simple ROP defense
  - Making gadget locations unpredictable

```
0x8E60  MOV  R1, PC
0x8E62  ADD  R1, #1
0x8E64  BLX  R1
```

- Software fault isolation (SFI)
  - Prevent invalid security-critical API access

**Instrumentation**

```
0x8E60  MOV  R1, PC
0x8E62  NOP
0x8E64  ADD  R1, #1
0x8E66  NOP
0x8E68  BLX  R1
0x8E6A  NOP
```

**w/o SFI**
- Critical API deference
- Critical API func call
- Critical API

**w/ SFI**
- Critical API deference
- SFI Logic
- Critical API func call
- Critical API

![Image of diagram showing the process of instrumenting code with SFI and the impact on critical API calls](image-url)
Case Study 3

- Run-time status monitoring for flight controllers
- Monitoring
  - Various flight control status
  - Shell commands

Prototype of target function

```c
float AP_InertialNav_NavEKF::get_altitude(AP_InertialNav_NavEKF *this)
```

Instrument

```
0xDC08 FLDS  S0, [R0,#0C]
0xDC0C BX    LR
```

```
0xDC08 FLDS  S0, [R0,#0C]
0xDC0C BL    MonitorFunc
0xDC0E BX    LR
```
Case Study 3

• Run-time status monitoring for flight controllers

Prototype of target function

```
int nsh_parse(FAR struct nsh_vtbl_s *vtbl, char *cmdline)
```

Instrumentation:

```
0xD4E0  PUSH  {R4-R11,LR}
0xD4E4  SUB   SP, SP, #0x74
0xD4E6  MOV   R4, R0
```

```
0xD4E0  PUSH  {R4-R11,LR}
0xD4E4  PUSH  {R0}
0xD4E6  MOV   R0, R1
0xD4E8  BL   MonitorFunc
0xD4EC  POP   {R0}
0xD4EE  SUB   SP, SP, #0x74
0xD4F0  MOV   R4, R0
```
Case Study 4

- Function patching for real existing vulnerabilities
  - Replace an unpatched function with a patched function

```cpp
int I2C::init()
{
    ...
    if (_bus_clocks[bus_index] > _frequency) {
        (void)up_i2cuninitialize(_dev);
        ...
        goto out;
    }
    ....
    out:
    if ((ret != OK) && (_dev != nullptr)) {
        up_i2cuninitialize(_dev);
    }
    ....
}
```

Stretched Replacement

```cpp
int I2C::init()
{
    ...
    if (_bus_clocks[bus_index] > _frequency) {
        (void)up_i2cuninitialize(_dev);
        _dev = nullptr;
        ...
        goto out;
    }
    ....
    out:
    if ((ret != OK) && (_dev != nullptr)) {
        up_i2cuninitialize(_dev);
        _dev = nullptr;
    }
    ....
}
```
Performance Impact

• CoreMark benchmark
  • Run-time overhead: 3.2%,
  • Space overhead: 1.3%

• Instrumentation
  • Location: function start address
  • Logic: function call counter

![Bar chart showing overhead for different operations: INLINE NOP, CALL EMPTY, INLINE LOGIC, CALL LOGIC]
Conclusion

• One of the new practical ARM binary rewriters
  • Low run-time/space overhead
  • Instrumentation at arbitrary locations
  • Overcome ARM-specific challenges

• Applicable to multiple platforms
  • Smartphone, microcontroller...