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

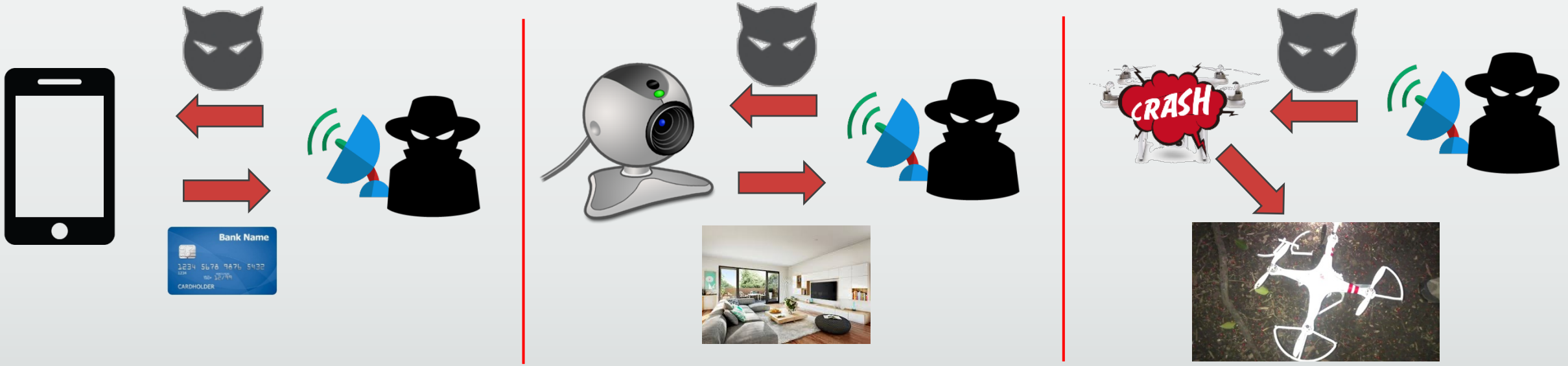
PURDUE UNIVERSITY

* NEC Laboratories America
Relentless passion for innovation

+ Georgia Tech

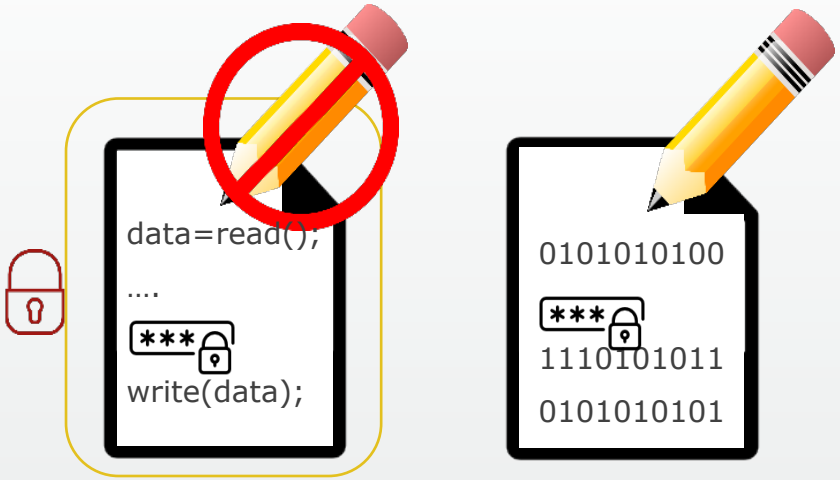
Security of ARM platforms

- 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

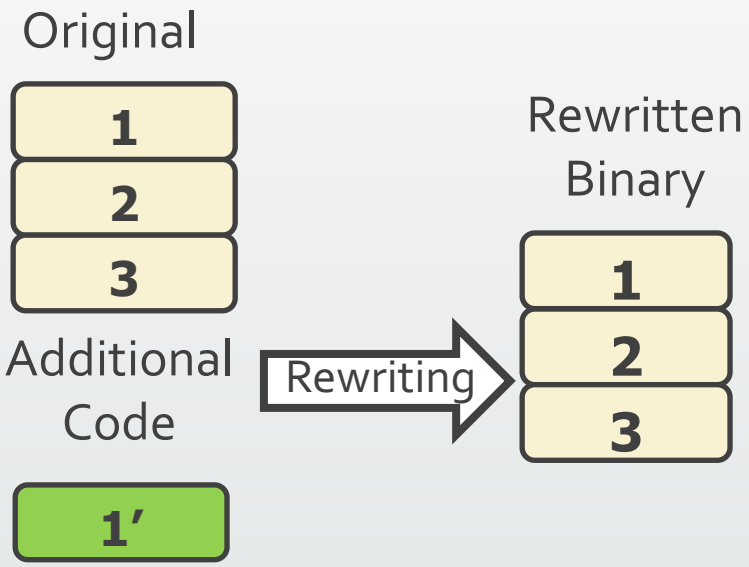
Rewriter	Target Architecture
BISTRO	x86
Uroboros	x86
Dyninst	x86
Pebil	x86
REINS	x86
PSI	x86
SecondWrite	LLVM
Dyninst	ARM 64bit (Experimental)
RevARM	**ARM 32bit**

Binary Instrumentation Approaches

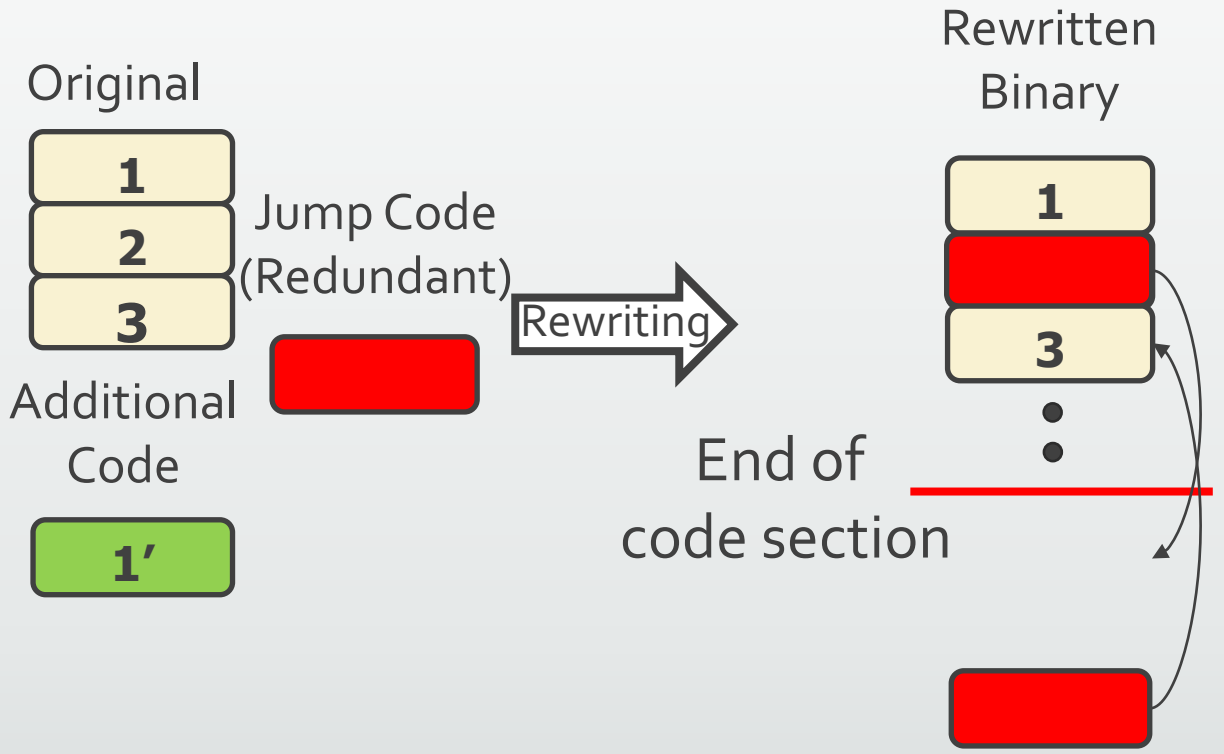
	Insertion-based	**Detour-based**
Instrumentation Type	Insert new codes inline	Jump to new codes
Control Flow	Preserved	Altered
Overhead	Lower	Higher

Insertion-based vs. Detour-based

Insertion-based approach

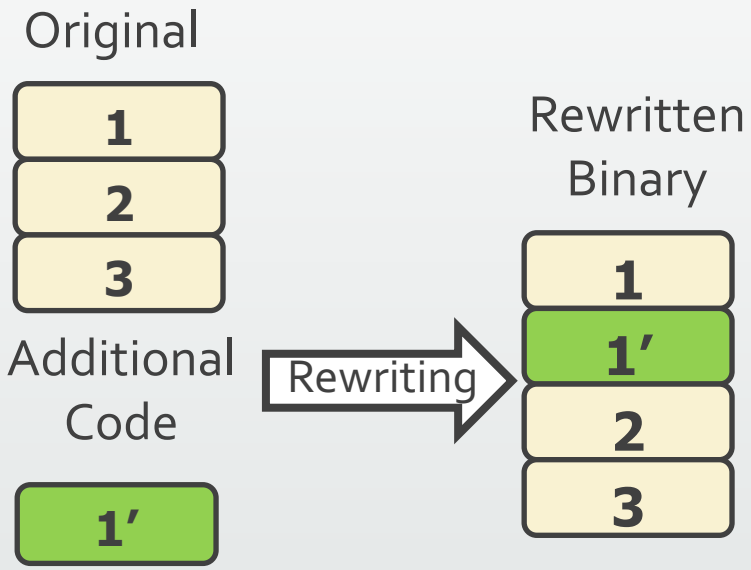


Detour-based approach

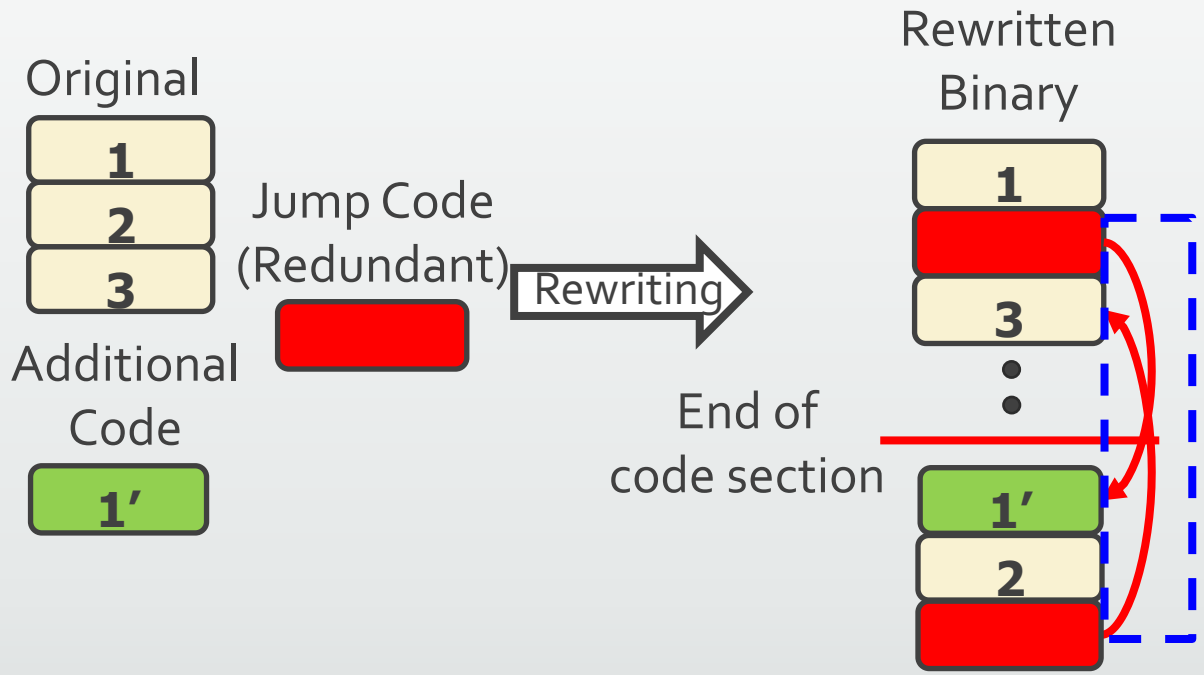


Low Overhead

Insertion-based approach



Detour-based approach



Redundant Jump → **Run-time overhead**

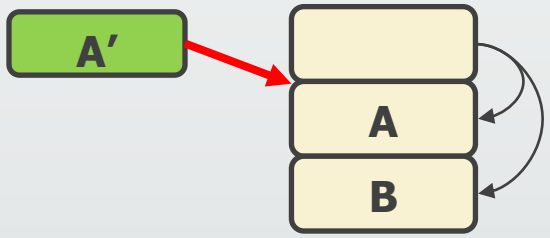
Redundant Jump → **Space overhead**

PURDUE UNIVERSITY CERIAS

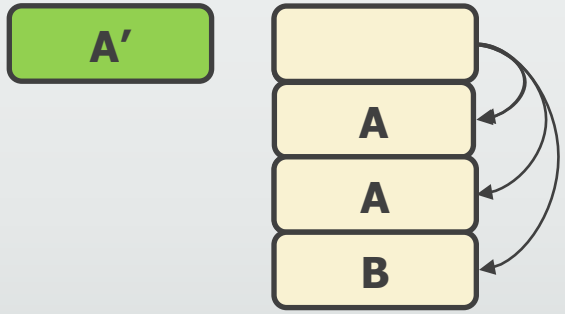
Instrumentation at Arbitrary Code Locations

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

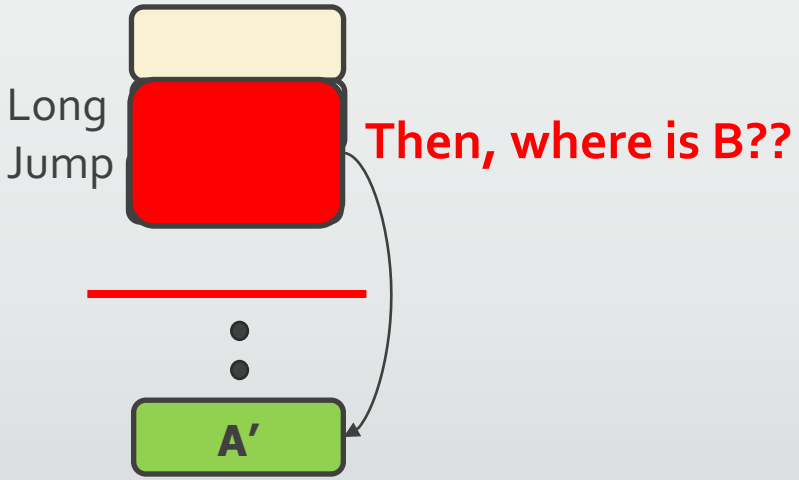
Before instrumentation



Insertion-based approach



Detour-based approach



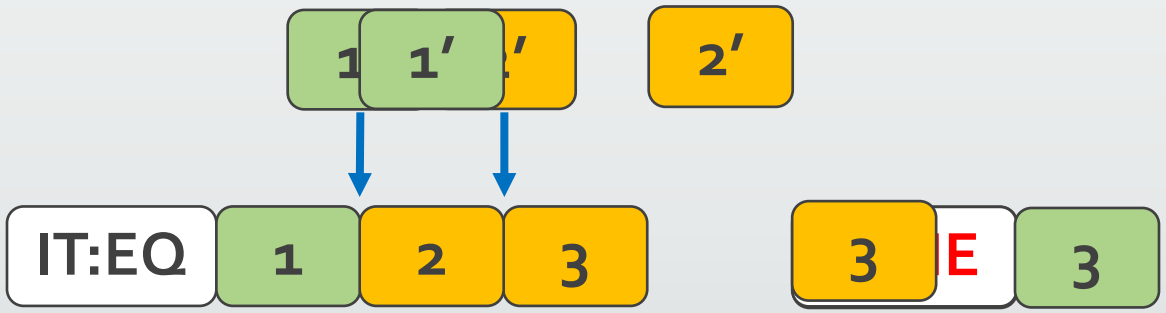
Challenges of Insertion-based Approach

- RevARM: Addresses four ARM-specific challenges
  - C1: If-Then instruction
  - C2: Branch table instruction
  - C3: Direct access to the program counter
  - C4: 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

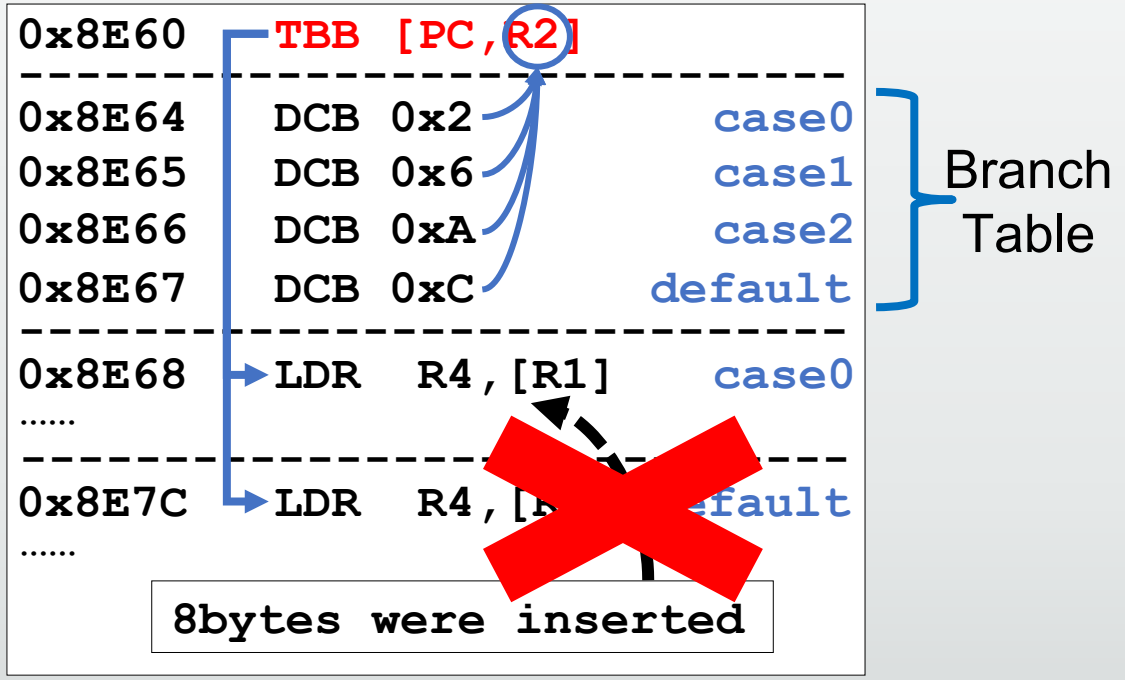


: If condition is true

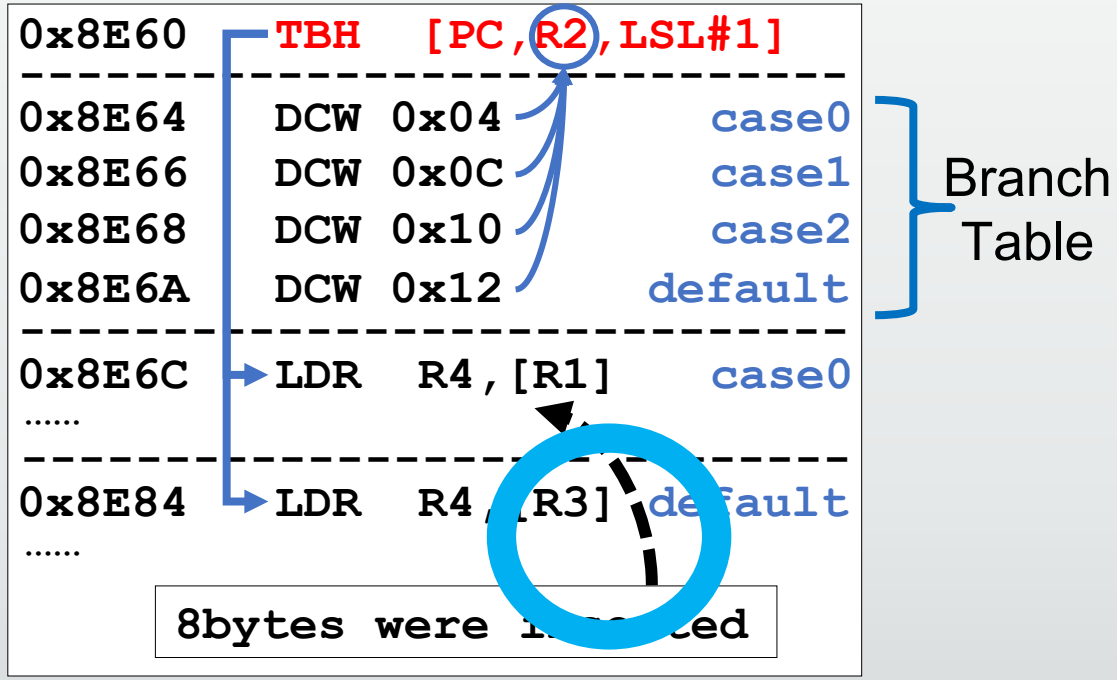
: Else

C2: Branch Table Instruction

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



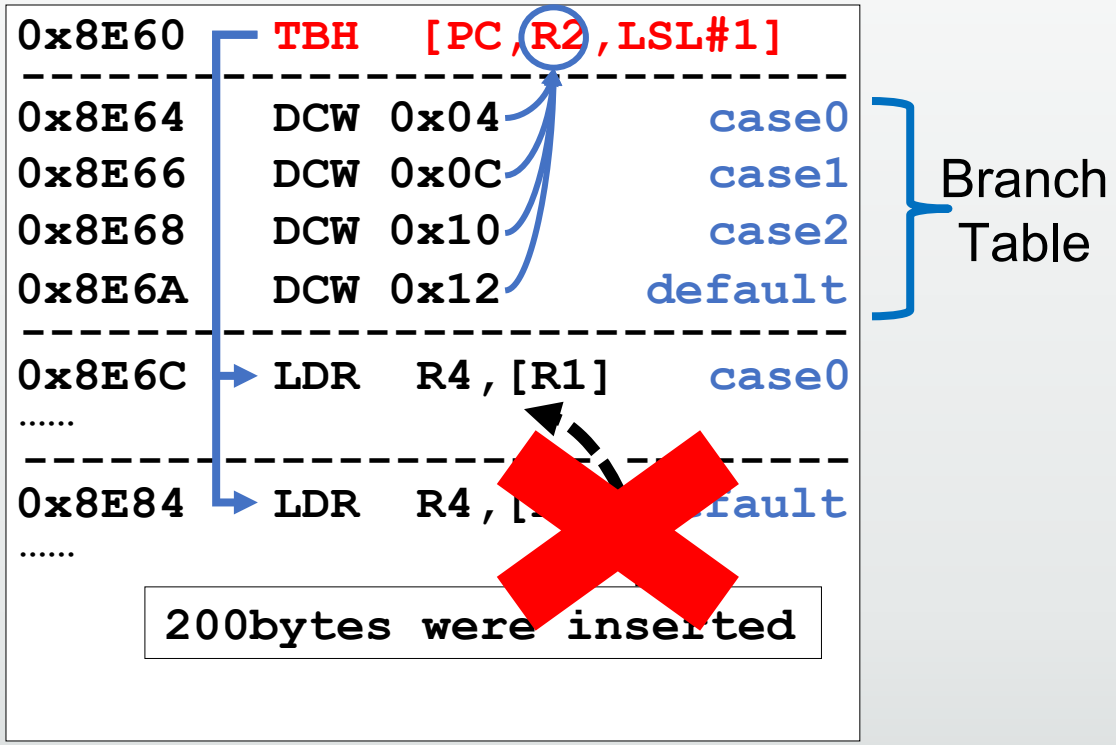
One-byte relative address for each case



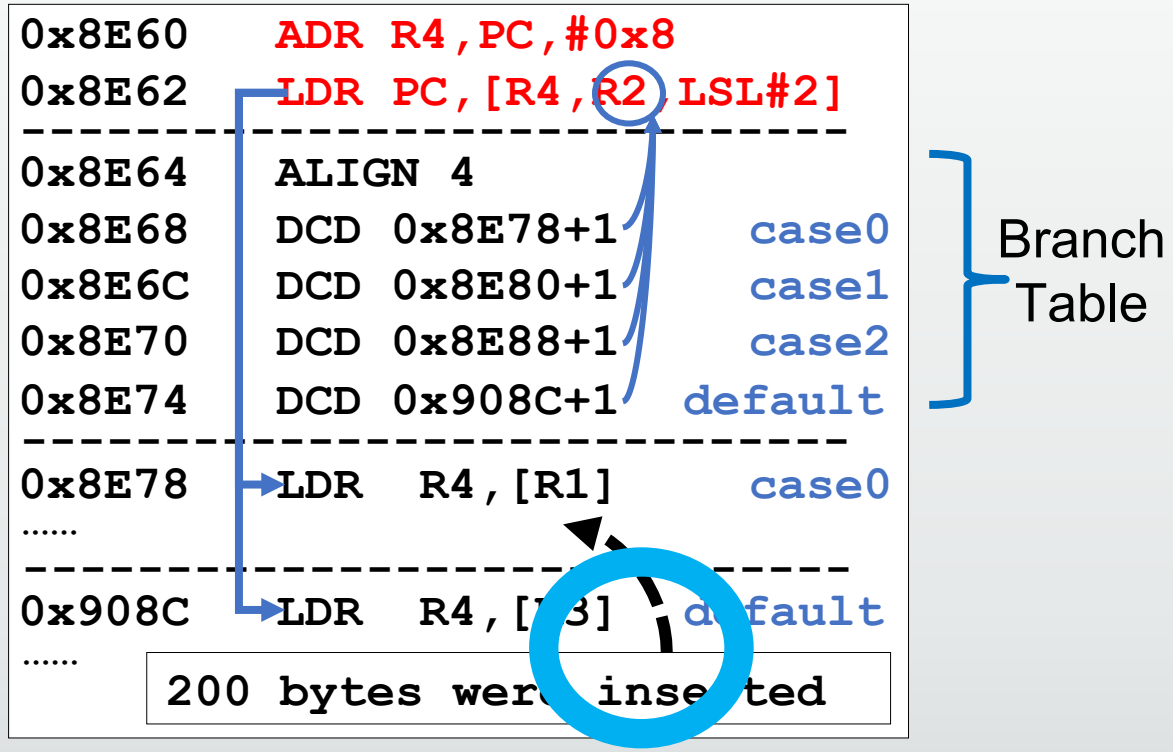
Two-byte relative address for each case

C2: Branch Table Instruction

- Q: What if even TBH range is insufficient?



Two-byte relative address for each case

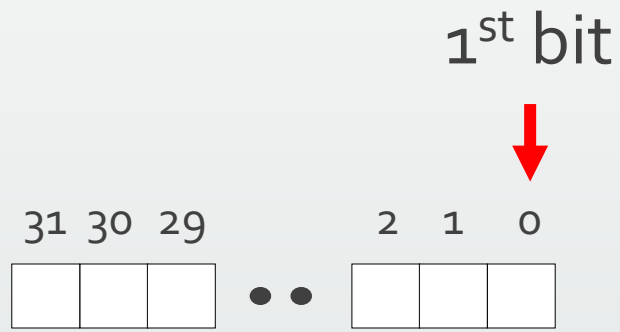


Four-byte absolute address for each case

Other Challenges

- C3: Direct access to the program counter
  - PC can be used as a general register
    - e.g., MOV, PC ← 0x08000000 / 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



Jump to a thumb function

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

Evaluation of RevARM

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

Board	Pixhawk (STM32F427 with FPU)
Processor	ARM Cortex-M4 168Mhz
Memory	256KB SRAM, 2MB flash memory

- 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

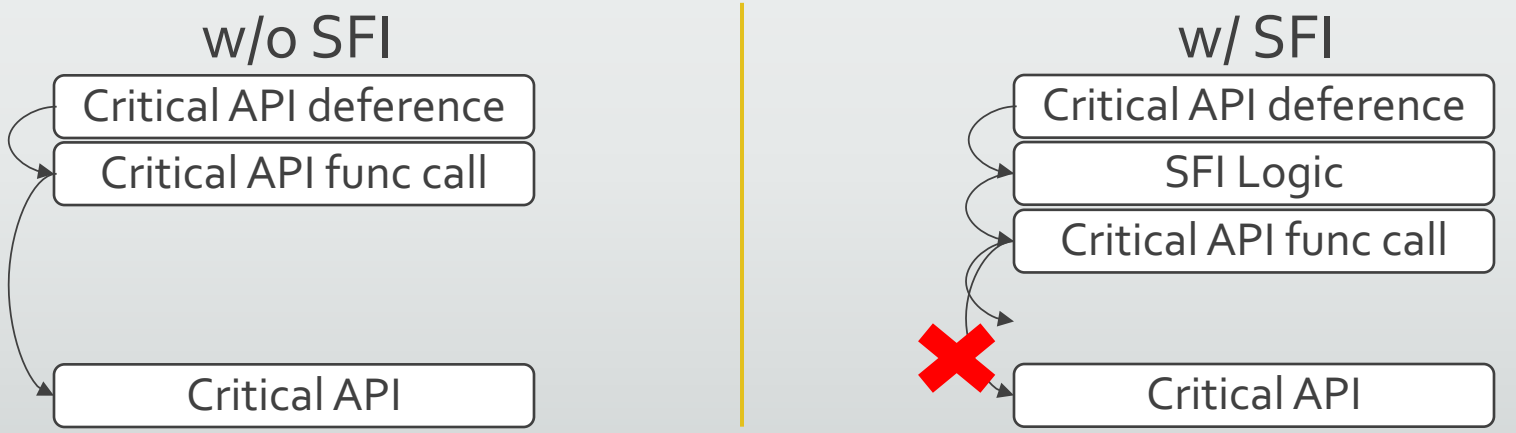
- Simple ROP defense
  - Making gadget locations unpredictable

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

Instrumentation

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

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



Case Study 3

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

Prototype of target function

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

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

Instrument

```
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)
```

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

Instrument

```
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

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

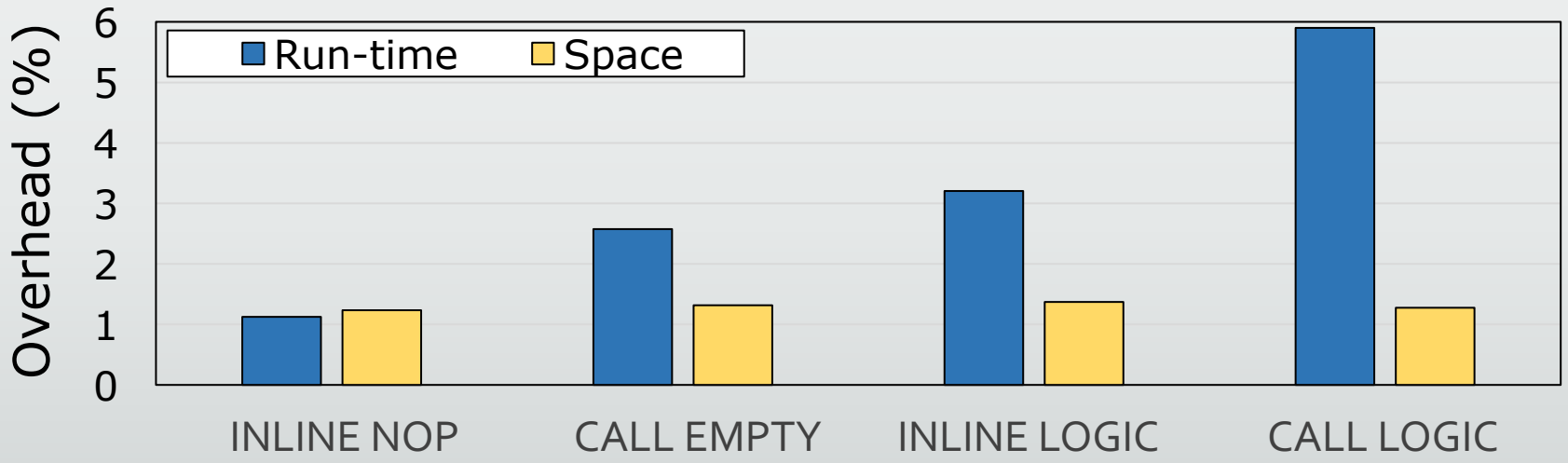
Replacement

```
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;
        }
        ....
```

Stretched

Performance Impact

- CoreMark benchmark
  - Run-time overhead: 3.2%,
  - Space overhead: 1.3%
- Instrumentation
  - Location: function start address
  - Logic: function call counter



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…