Towards a Practical Solution to Detect Code Reuse Attacks on ARM Mobile Devices

2015.6.14

Yongje Lee*, Ingoo Heo, Dongil Hwang, Kyungmin Kim and Yunheung Paek
Seoul National University, Korea

*Speaker

Contents

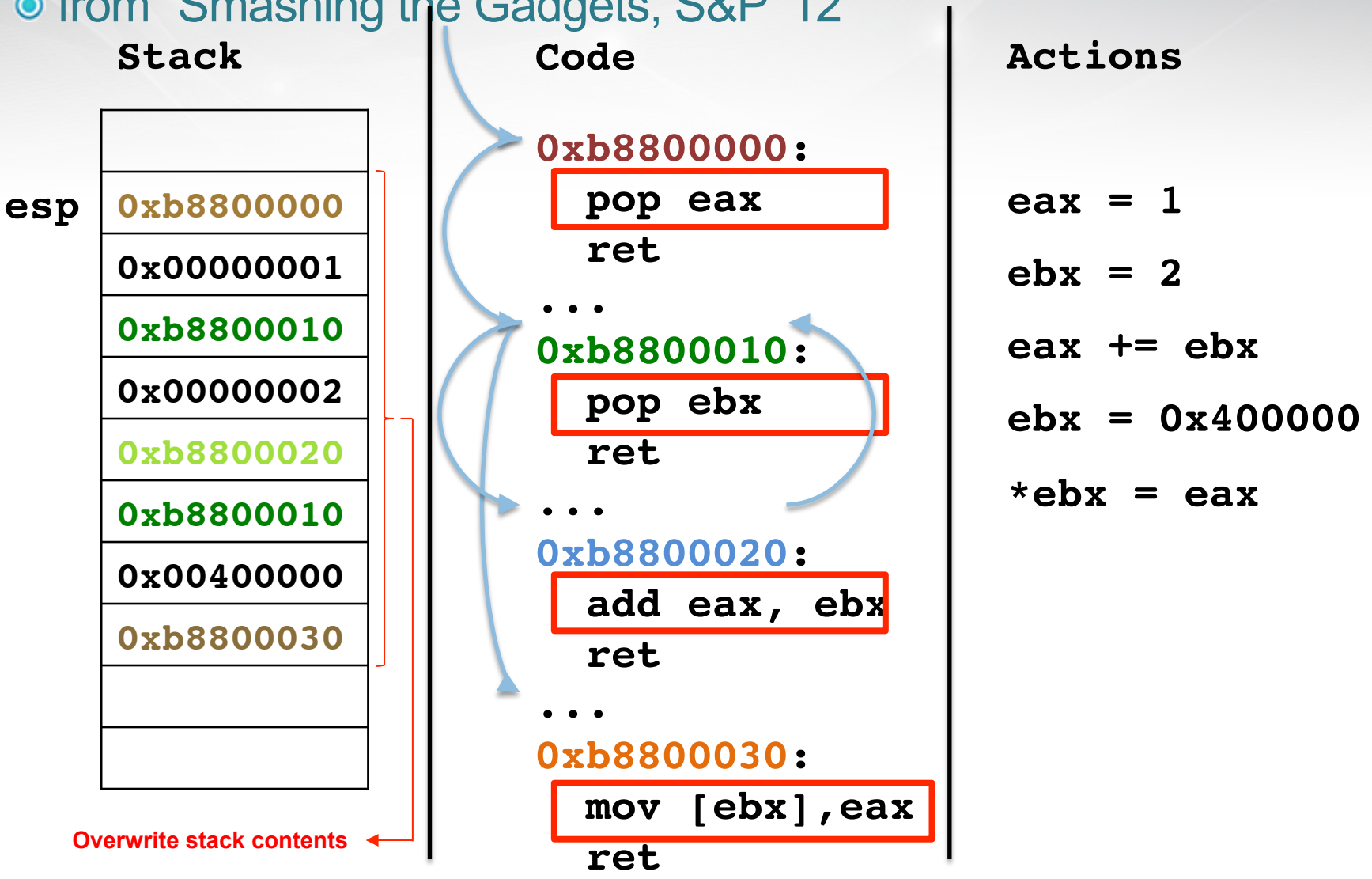
Growing need to protect system …

- Attackers try to control the system behavior in all aspects
  - Capture various system events and manipulate the events for their profits

- Methods to acquire such capabilities
  - Code Injection Attack
    - Attackers first inject their own code in the memory
    - Execute the code after hijacking the normal course of execution
    - *Writable xor eXecutable* (W⊕X) policy effectively prevents the code injection attacks.
  - Code Reuse Attack (CRA)
    - Obey the W⊕X policy: do not rely on injected code
    - Launch an attack by stitching existing code snippets (gadgets) into a new code sequenc e
    - E.g.) Return-oriented programming (ROP), Jump-oriented programming (JOP)

Return-oriented Programming

- from “Smashing the Gadgets, S&P ’12”



Return-oriented Programming

- The return addresses in the stack are manipulated by attackers .
  - Consequently, the return addresses do not point to the original call sites.

- Solution : Shadow Call Stack
  - Basically consists in maintaining a copy of the call stack of the program run ning on the host processor
  - On an identified CALL instruction, the return address is pushed on the shad ow call stack.
  - On an identified RET instruction, the return address is checked against the saved one.
  - This solution is considered to be one of the fine-grained ROP defenses.

The Objectives of This Work

- Detect ROP attacks on ARM-based mobile devices
  - Smart mobile devices continue to gain in popularity among the general publ ic ➔ becoming more appealing targets of software-oriented attacks.
  - ARM is the de-facto standard CPU for diverse mobile devices.
- Pursue hardware-based CRA detection
  - Especially for ROP detection in this work
  - Special hardware modules are added for detecting ROPs to minimize the p erformance overhead.
- Seek for the suitable solution for ARM-based AP design
  - These days, to make AP, device vendors usually buy COTS ARM cores an d integrate them together with supporting IPs.
  - Our solution should not require the modification of internal microarchitectur e of ARM cores. ➔ Exploit built-in ARM CoreSight to extract the program e xecution behaviors outside the host

Related Work

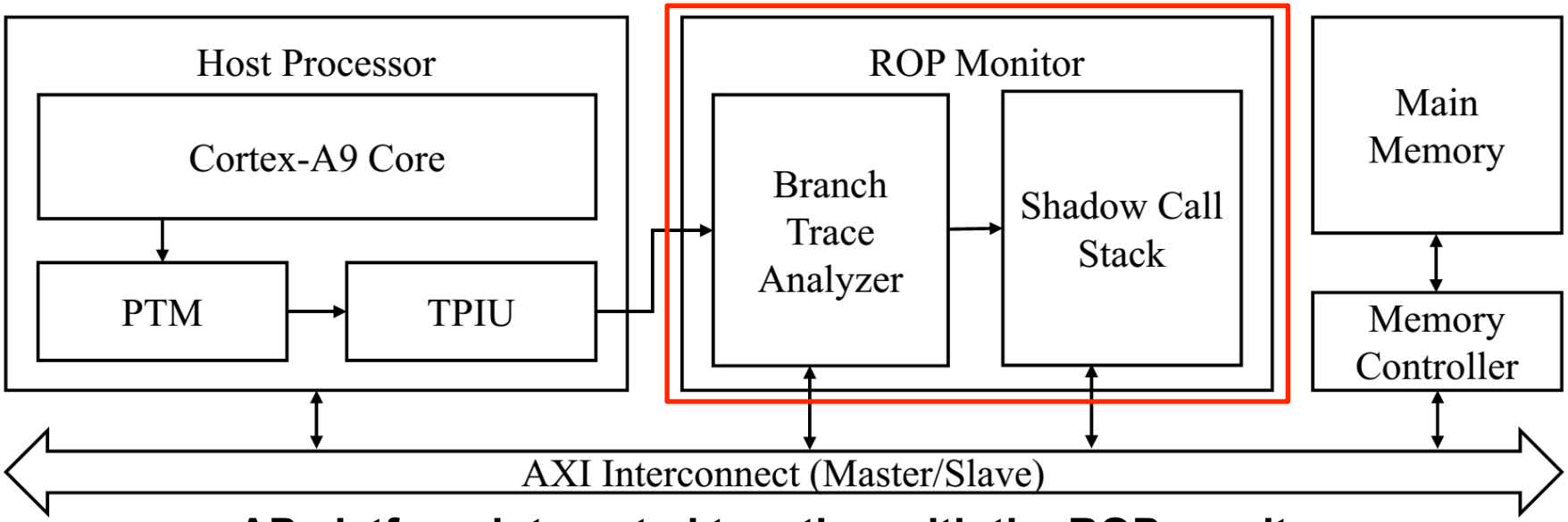
- Hardware-based CRA detection
  - SmashGuard (IEEE Transactions on Computer'07)
    - Hardware shadow stack
  - Branch Regulation (ISCA'12)
    - Thwart ROP and JOP attacks by enforcing a simple invariant ruling the normal behavior s of branches in a programming language.
  - SCRAP (HPCA'13)
    - Signature based JOP defense
  - Hardware-based CFI (DAC'14)
    - Simple backward-edge flow integrity enforcement by checking that the return instruction transfers to the address within an active function.
- Exploiting built-in hardware debug architecture
  - Extrax (DATE'15)
    - A kernel integrity monitor using the core debug interface for SPARC processors
    - First approach that utilizes the debug interface in an effort to thwart security threats
  - No work has been implemented in ARM-based mobile devices.

Assumption

- The system enforces the (W⊕X) security protection rule.
- No other security holes which can directly escalate adversary's privilege are assumed.
- Adversaries might exploit memory corruption vulnerabilities.
- Adversaries can bypass the *address space layout randomizati on* (ASLR).
- Self-modifying code is not considered.

Architecture for ROP Detection

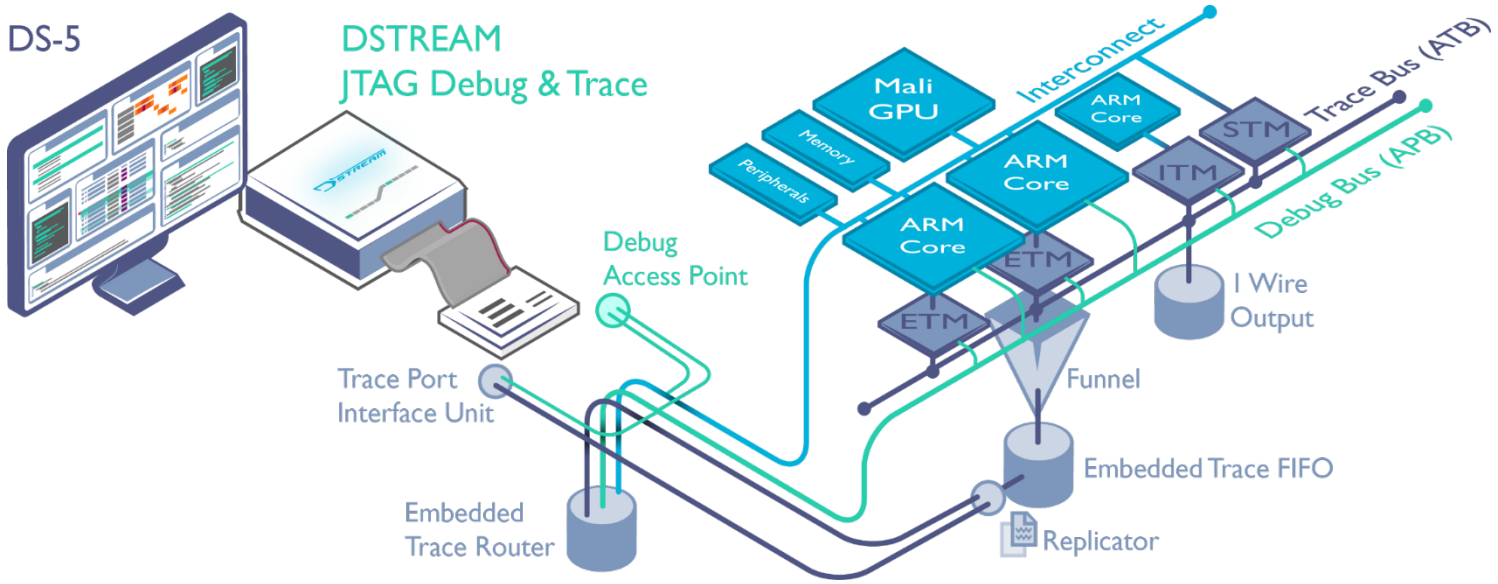
- System Components
  - ROP monitor
    - A subsystem where monitoring modules for ROP detection are integrated together.
    - Branch Trace Analyzer (BTA), Shadow Call Stack (SCS)
  - CPU : Cortex-A9 processor
    - Equipped with PTM, TPIU : ARM CoreSight debug modules
  - Main bus : AMBA3 AXI interconnect



<AP platform integrated together with the ROP monitor>

Branch Trace Analyzer (BTA)

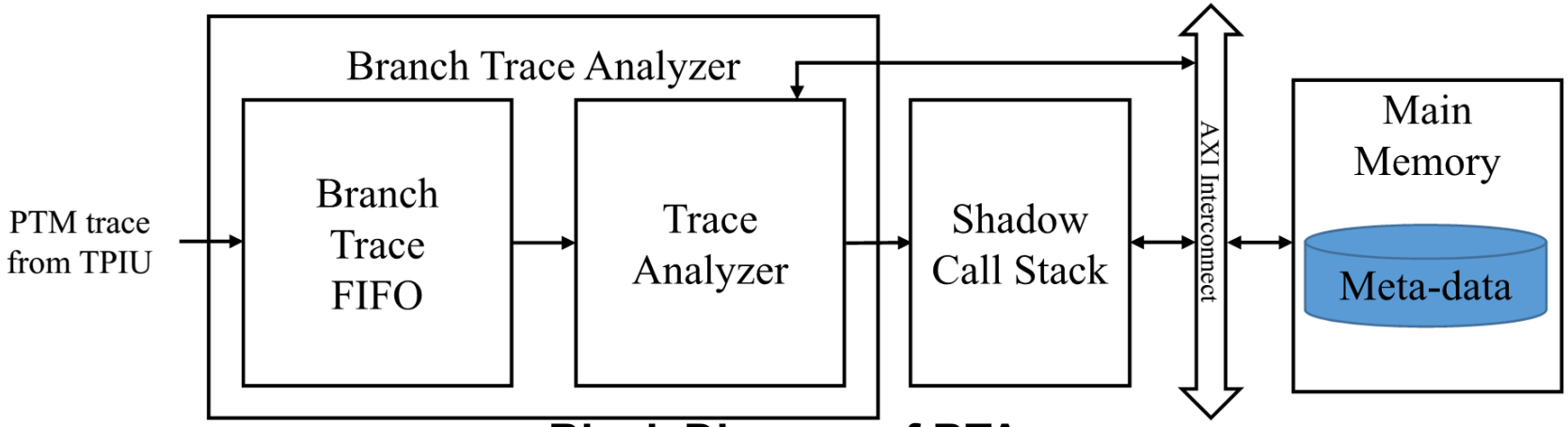
- ARM CoreSight PTM/TPIU
  - PTM captures diverse debug information for the ARM CPU.
    - Branch target addresses, exceptions, current PID, instruction set mode change (ARM/T HUMB) and so on
    - Produce the generic form of the tracing data
  - Generated PTM traces are routed to TPIU, and then forwarded to the exter nal debuggers via off-chip pins.



<ARM CoreSight Debug Architecture (here, ETMs are used)>

Branch Trace Analyzer (BTA)

- In our work, the TPIU output signals are routed to BTA
  - BTA uses these signals to extract useful information for ROP detection.
- Submodules of BTA
  - Trace Analyzer
    - Decode the PTM traces to extract branch types and target addresses
    - Generate necessary information used by the Shadow Call Stack
      - (call, return, source address, target address)
  - Branch Trace FIFO
    - Bridge the frequency gap between CPU and the ROP monitor



<Block Diagram of BTA>

Branch Trace Analyzer (BTA)

- PTM traces are insufficient to interpret the branch behaviors on the host CPU.
  - PTM traces do not carry branch types and target addresses of direct branch instructions.
- To supplement lacking information, we perform offline binary analysis and generate the set of meta-data.
  - Branch type (e.g., jump type : b, bl, call type : bx, blx)
  - Source and/or target addresses of branch Instructions

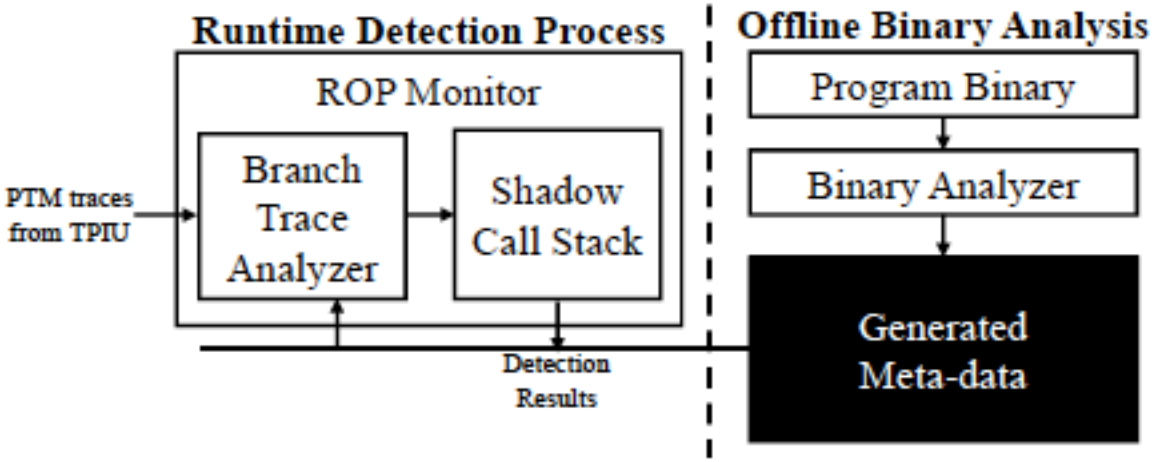
	Address	Code	Decoded instruction	
√	1c496:	f7f2 fd13	bl	eec0 <__errno>
	1c49a:	2416	movs	r4, #22
	1c49c:	6004	str	r4, [r0, #0]
	1c49e:	2400	movs	r4, #0
√	1c4a0:	e078	b.n	1c594 <popen+0x134>
	1c4a2:	200c	movs	r0, #12
√	1c4a4:	f7f0 fd26	bl	cef4 <malloc>
	1c4a8:	9001	str	r0, [sp, #4]
	1c4aa:	2800	cmp	r0, #0

Omitted information in PTM traces

Meta-data
eec0
.
.
.
1c594
.
.
.
cef4

ROP Detection Process

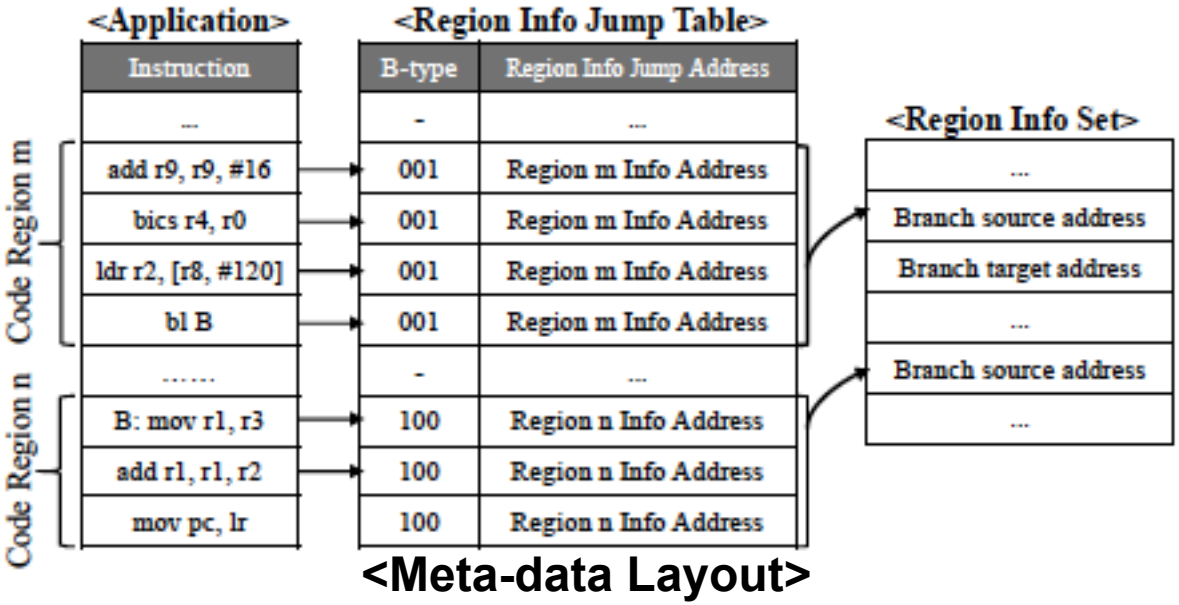
- 1st phase (SW Binary Analyzer)
  - Generate the static information of the target system for ROP detection
  - Resulting information is summarized in the form of meta-data
- 2nd phase (HW ROP Monitor)
  - Runtime detection of ROP attacks
  - Using the generated meta-data, the ROP monitor gets to know the executio n behaviors of the target program.



<ROP Detection Process>

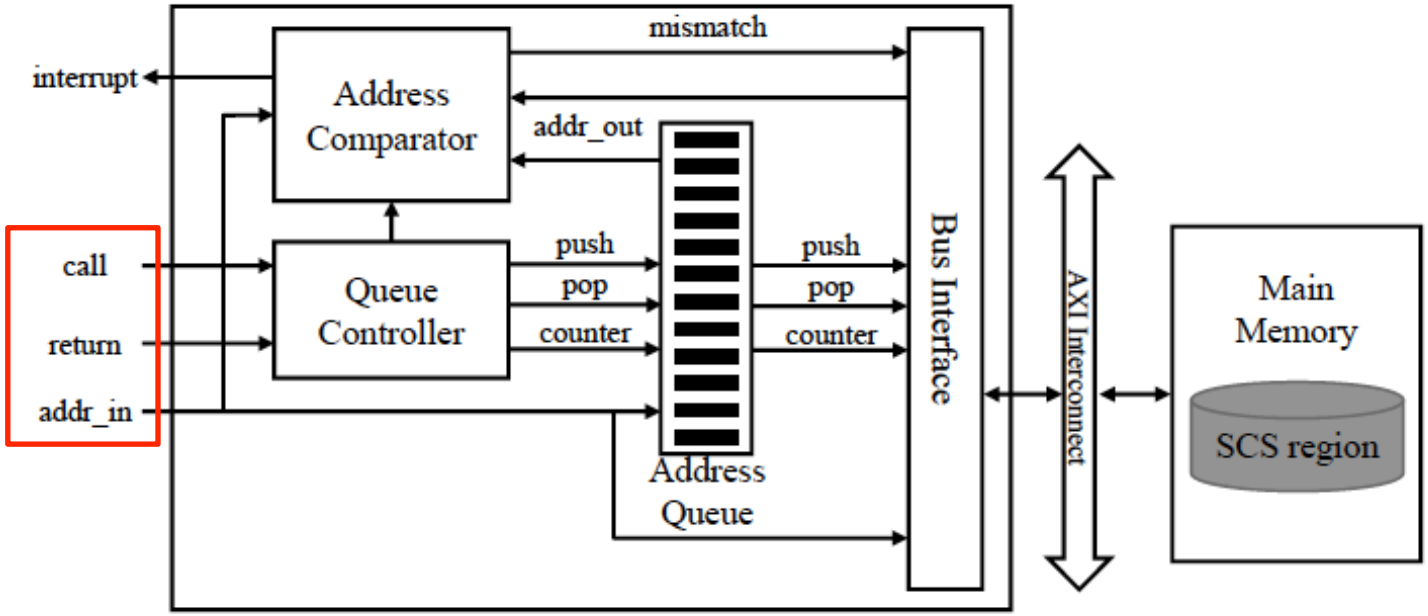
Meta-data Layout

- Binary Analyzer
  - Divide the application code into multiple code regions on every control trans fer instruction ➔ Unique region number is given.
  - Extract branch types according to the ARM's function calling convention
    - Call : bl (branch with link) or blx (branch with link and exchange)
    - Return : branch instruction with the link register (LR)
  - Branch source and/or target addresses should be saved.



Shadow Call Stack (SCS)

- SCS receives input signals from BTA
  - addr_in, call, return
- Main Submodules
  - Queue Controller : maintain a shadow copy of the call stack
  - Address Comparator : compare the runtime return address against the add ress saved in the address queue

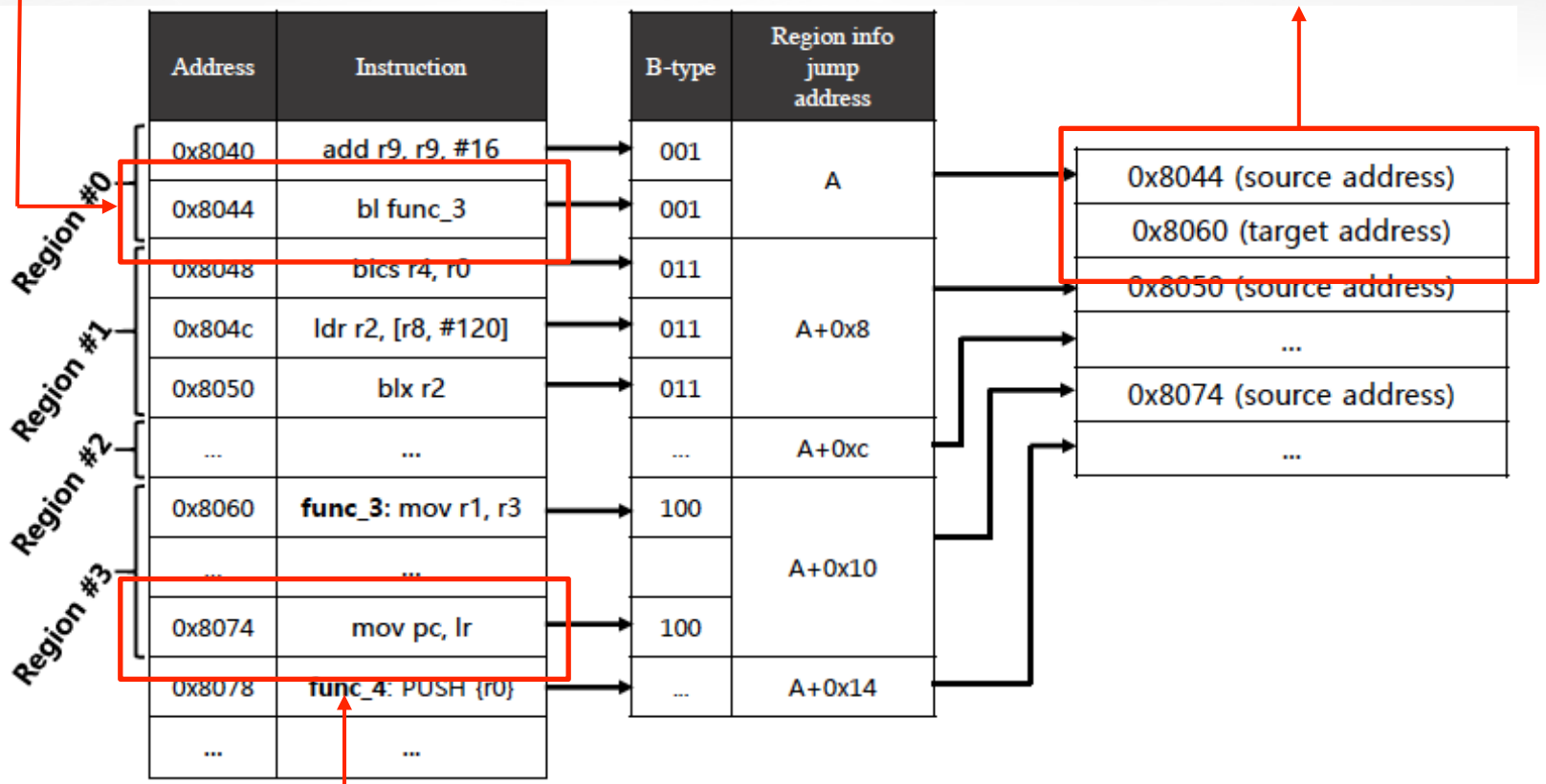


<Block Diagram of SCS>

ROP Detection Procedure with Meta-data

1) func_3 is invoked

2) Source address is saved in SCS.



3) func_3 returns

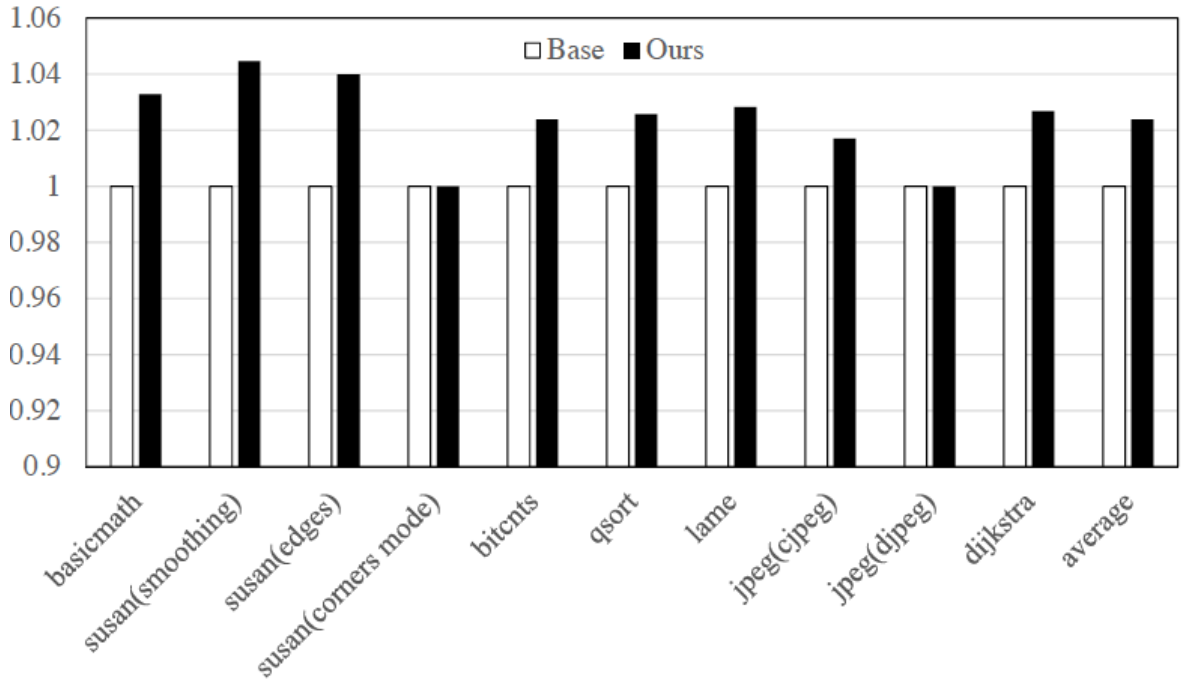
4) SCS checks the return address is 0x8048 (the next address of the call site).

Experimental Environment

- Full-system prototype implemented on Xilinx Zynq-7000 XC7020 platform
  - Cortex-A9 host processor
    - PTM, TPIU included
    - Running at 200MHz
  - ROP monitor
    - Running at 90MHz
    - Occupying 13.8% of LUTs (7,362/53,200) and 3.1% of BRAMs (539/17,400)
    - 86,714 GC by Synopsys DC using a commercial 45-nm library
  - Linux 3.8 kernel
- Tested with ten applications in Mibench test suite

Performance

- Configurations
  - Base : native host program
  - Ours : host program with PTM/PTIU and ROP monitor enabled
- About 2.39% overhead on average
  - Caused by resource(memory) conflicts between the host CPU and the RO P monitor.



Conclusion and Future Extension

- This paper introduces a hardware ROP monitor for ARM-based smart mobile devices.
- The proposed monitor shows negligible performance overhead, and can be implemented without any modifications of the processor internal.
- Consequently, the proposed architecture would become an attractive CRA defense solution to ARM-based AP platforms.
- The proposed architecture can be further applied to thwart control flow hijacking attacks by slightly modifying the meta-data layout and adding additional hardware elements.

Thank You