RetTag: Hardware-assisted Return Address Integrity on RISC-V

Yu Wang[1], Jinting Wu[1], Tai Yue[2,1], Zhenyu Ning[1], Fengwei Zhang[1]

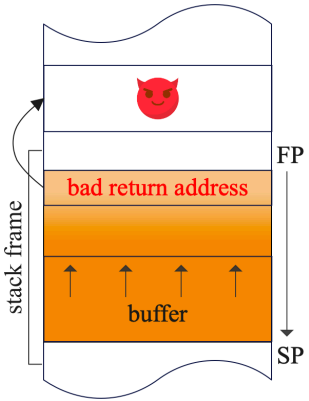
[1] *Southern University of Science and Technology*
[2] *National University of Defense Technology*

April 5, 2022

SUSTech Southern University of Science and Technology

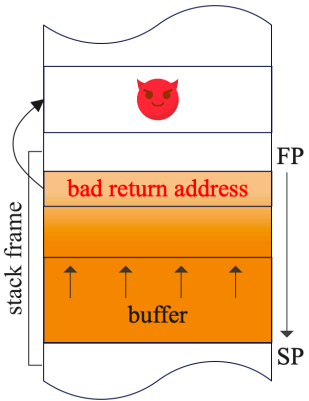
Motivation

Memory corruption vulnerabilities, such as buffer overflows, remain a prominent threat.



Motivation

Memory corruption vulnerabilities, such as buffer overflows, remain a prominent threat.

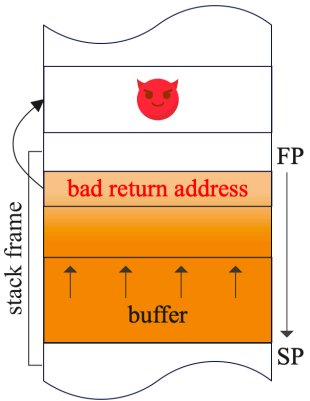


- Code injection: inject shellcode into writable memory
- Code reuse (e.g., ROP): construct gadgets from existing code

SUSTech
Southern University of Science and Technology

Motivation

Memory corruption vulnerabilities, such as buffer overflows, remain a prominent threat.



- Code injection: inject shellcode into writable memory
- Code reuse (e.g., ROP): construct gadgets from existing code

ROP on RISC-V

Some works have proved the availability of ROP attacks on the new RISC-V architecture.

Return address must be protected when stored on the stack!

Motivation

ARMv8.3-A Pointer Authentication

- Protect the integrity of pointers in memory
- Use Pointer Authentication Codes (PAC) embedded in pointers

Introduced in ARMv8.3-A specification (2016)

- Armed COTS devices are limited
  - Recent Apple processors and a few other processors
- Current PA scheme is still vulnerable
  - Face the challenge of reuse attacks

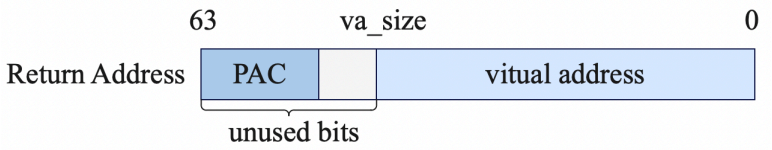
ARM Pointer Authentication. https://www.qualcomm.com/media/documents/files/whitepaper-pointer-authentication-on-armv8-3.pdf

SUSTech Southern University of Science and Technology

Motivation

Main Idea

- Leverage **Pointer Authentication Code (PAC)** embedded into the **unused bits** of return address to ensure return address integrity on RISC-V.



Three paged virtual-memory schemes in 64-bit RISC-V:

- Sv39, Sv48, and Sv57 ✔

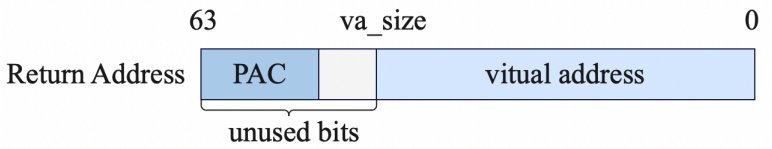
RISC-V®

SUSTech
Southern University of Science and Technology

Motivation

Main Idea

- Leverage **Pointer Authentication Code (PAC)** embedded into the **unused bits** of return address to ensure return address integrity on RISC-V.



Advantages of using unused bits:

- Requiring no additional memory
- Introducing less performance overhead
- Requiring no additional hardware components

SUSTech
Southern University of Science and Technology

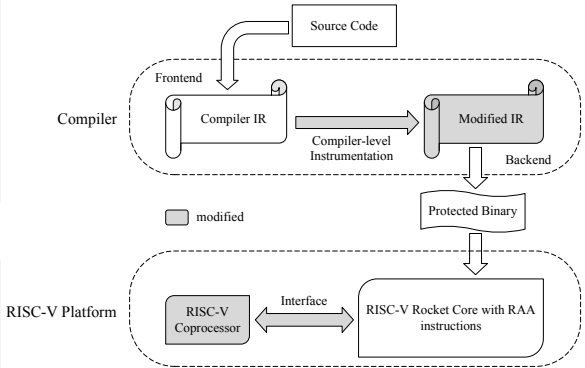
Our Proposal - RetTag

RetTag

- A hardware-assisted and crypto-based defense scheme:
  - Custom Return Address Authentication (RAA) instructions
  - RetTag-enhanced compiler
  - RetTag-enabled RISC-V platform

How Does It work?

- Compiler-level instrumentation to transparently emit RAA instructions
- RISC-V platform to support RAA instructions



PAC Generation

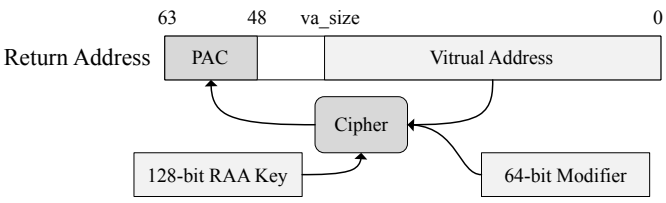
RAA Instructions

- `pac`: create a PAC of the return address when the protected program calls a function
- `aut`: calculate a new PAC and verify it at the end of the function call

PAC Generation

RAA Instructions

- `pac`: create a PAC of the return address when the protected program calls a function
- `aut`: calculate a new PAC and verify it at the end of the function call



RetTag generates PAC by using:

- the value of return address
- a 128-bit RAA key
- a 64-bit modifier

Resist Reuse Attacks

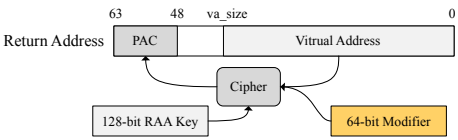
Reuse Attacks

The adversary reuse previously observed valid PACs!

Solution

- A **unique modifier** can effectively prevent the pointer from being arbitrarily interchangeable with a malicious one

Resist Reuse Attacks

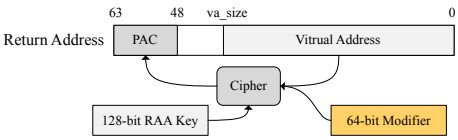


Modifier

The 64-bits modifier contains:

- the 32-bit SP value
- the 32 most significant bits of function-specific id

Resist Reuse Attacks



Modifier

The 64-bits modifier contains:

- the 32-bit SP value
- the 32 most significant bits of function-specific id

Note

- Only SP value as the tweak (ARM PA)
  - The same modifier will be generated when different functions execute with the same SP value
- Only function-id as the tweak
  - The same modifier will be generated when a function is called by different callers

SUSTech
Southern University of Science and Technology

Compiler-level Instrumentation

```
Function:
Prologue:
  lui    tmp, func-id[0:19]
  addi   tmp, tmp, func-id[20:31]  ① get 32-bit function-id
  slli   tmp, tmp, 0x20
  add    tmp, tmp, sp              ② get modifier
  pac    ra,  ra,  tmp             ③ PAC generation
  ...
Function body
Epilogue:
  ...
  lui    tmp, func-id[0:19]
  addi   tmp, tmp, func-id[20:31]  ④ get 32-bit function-id
  slli   tmp, tmp, 0x20
  add    tmp, tmp, sp              ⑤ get modifier
  aut    ra,  ra,  tmp             ⑥ PAC authentication
  ret
```

Instrumentation

- Prologue:
  - ① ② Construct a 64-bit modifier
  - ③ PAC generation
    - generate the PAC
    - store it in the unused bits
- Epilogue :
  - ④ ⑤ Construct a 64-bit modifier
  - ⑥ PAC authentication
    - regenerate the PAC
    - remove the PAC (SUCCESS) or trigger an interrupt (FAIL)

SUSTech
Southern University of Science and Technology

Compiler-level Instrumentation

```
Function:
Prologue:
  lui    tmp, func-id[0:19]
  addi   tmp, tmp, func-id[20:31]   ① get 32-bit function-id
  slli   tmp, tmp, 0x20
  add    tmp, tmp, sp               ② get modifier
  pac    ra,  ra,  tmp              ③ PAC generation
  ...
Function body
Epilogue:
  ...
  lui    tmp, func-id[0:19]
  addi   tmp, tmp, func-id[20:31]   ④ get 32-bit function-id
  slli   tmp, tmp, 0x20
  add    tmp, tmp, sp               ⑤ get modifier
  aut    ra,  ra,  tmp              ⑥ PAC authentication
  ret
```

Instrumentation

- Prologue:

① ② Construct a 64-bit modifier

③ PAC generation

  - generate the PAC
  - store it in the unused bits

- Epilogue :

④ ⑤ Construct a 64-bit modifier

⑥ PAC authentication

  - regenerate the PAC
  - remove the PAC (SUCCESS) or trigger an interrupt (FAIL)

Note

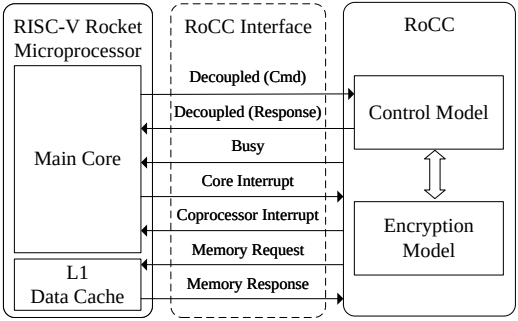
- Not all returns need to be authenticated

SUSTech
Southern University of Science and Technology

Implementation

Hardware extension

- Rocket Custom Coprocessor (RoCC) executes custom instructions
  - Interfaced with the in-order RISC-V Rocket core
- AES128 for PAC encryption
- Prototyped on Xilinx Kintex-7 FPGA KC705 and Rocket emulator



Implementation

Compiler integration

- GCC v10.1
  - function-id: generated by using a pseudo-random, non-repetitive sequence
- LLVM v11.0
  - function-id: generated by calling `getFunctionNumber`

- Bootloader
  - bbl for supporting custom instructions and RoCC interrupt
- Kernel
  - Linux kernel v4.20 and pk for handling RoCC interrupt

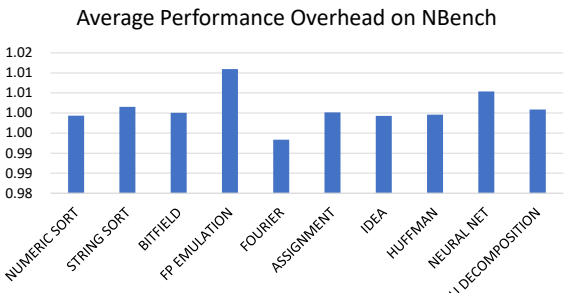
Evaluation

Security Analysis

- Perform attacks that **corrupt return address** in RIPE test suite
- All attacks that can be recurred on RISC-V are resisted successfully
- Out of scope currently
  - Attacks targeting other code pointers (e.g., function pointers)
  - Non-control data attacks
  - Data-oriented Programming (DOP) attacks

SUSTech
Southern University of Science and Technology

Evaluation



Hardware Resource Cost

- Vivado 2018.3
- Less than 0.8%

Performance Overhead

- 0.11% on NBench benchmark
- 7.69% on Coremark benchmark
- Performance Factor
  - Encryption module
  - Ratio of instrumented instructions

SUSTech
Southern University of Science and Technology

Evaluation

Performance Factor

- Encryption module
  - AES128
    - the CPU cycles of `lui,addi,slli,add,pac/aut`: 57
    - the CPU cycles of encryption: 20
    - performance overhead on Coremark: 7.69%

Evaluation

Performance Factor

- Encryption module
  - AES128 $\rightarrow$ faster hardware encryption?
    - the CPU cycles of `lui,addi,slli,add,pac/aut`: 57
    - the CPU cycles of encryption: 20
    - performance overhead on Coremark: 7.69%

```
addi    tmp, tmp, 0
addi    tmp, tmp, 0       four instructions to
addi    tmp, tmp, 0       account for the 4 cycles
addi    tmp, tmp, 0
```

SUSTech
Southern University of Science and Technology

Evaluation

Performance Factor

- Encryption module
  - AES128 → faster hardware encryption?
    - the CPU cycles of `lui,addi,slli,add,pac/aut`: 57 → 12
    - the CPU cycles of encryption: 20 → 4
    - performance overhead on Coremark: 7.69% → 3.42%

```
addi    tmp, tmp, 0
addi    tmp, tmp, 0        four instructions to
addi    tmp, tmp, 0        account for the 4 cycles
addi    tmp, tmp, 0
```

SUSTech
Southern University of Science and Technology

Evaluation

Performance Factor

- Encryption module
- Ratio of instrumented instructions
  - A fixed amount of instrumented instructions
  - The number of instructions in each function
  - Loops and code reuse

Number of Executed Instructions in Benchmarks with RetTag

Benchmarks	All Instructions	Instrumented Ins	Ratio
NBench	256,101,997,912	13,916,804	0.005%
Coremark	19,225,103,401	870,957,759	4.530%

SUSTech
Southern University of Science and Technology

Conclusion

Enhance return address integrity on RISC-V

Integrate RetTag into mainstream compilers

https://github.com/Compass-All/RetTag

Thanks! You can reach me at 12032879@mail.sustech.edu.cn for follow-up questions.

Implement a prototype on emulator and FPGA

More information

SUSTech