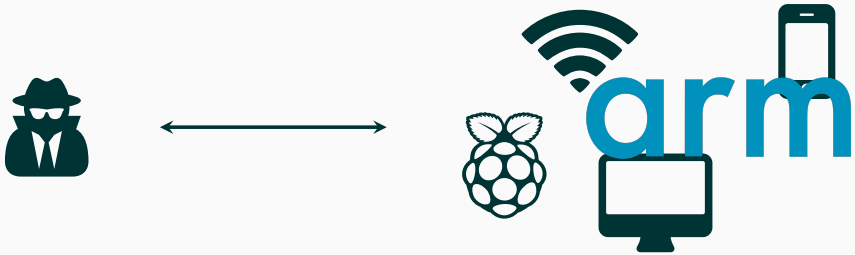
FIPAC: Thwarting Fault- and Software-Induced Control-Flow Attacks with ARM Pointer Authentication

Robert Schilling, Pascal Nasahl, Stefan Mangard
robert.schilling@iaik.tugraz.at

April 12, 2022

IAIK – Graz University of Technology



- **Attacker:**
  - Has remote access to device
  - Has physical access to device

Possible Attacks

- Software Attacks
  - Memory vulnerabilities leading to ROP, JOP, ...
- Physical fault attacks
  - Voltage glitches, clock glitches, EM, ...
- Software Induced Fault Attacks
  - PlunderVolt, CLKscrew, VoltJockey, ...

- Manipulate the control-flow of the program
  - Bypass privilege checks or signature verification

PLUNDER VOLT

- Control-Flow Integrity (CFI) is the foundation of secure software execution
- Determine a control-flow graph at compile-time and enforce it at runtime
- CFI exists on different levels of granularity
  - Indirect function calls → Software CFI
  - Instruction level → Requires intrusive hardware changes
  - Basic-block level → Suitable for software designs

- Why do we need another CFI countermeasure?
  - Combined software- and fault attacks bypass existing schemes
  - Strong CFI requires hardware support → not possible for commodity systems

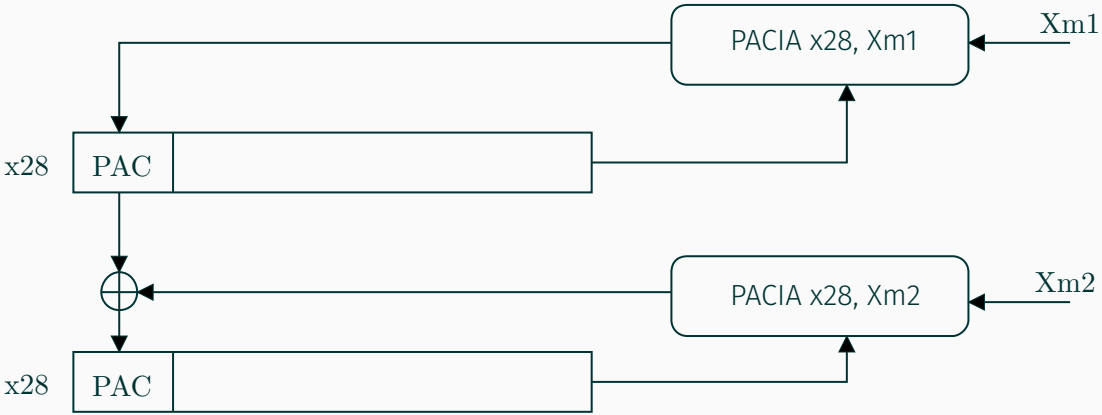
- **Contribution**
  - Basic-block granular CFI protection for ARM-based commodity devices
  - Exploit ARM Pointer Authentication for a cryptographic CFI state update
  - Custom LLVM-based toolchain to automatically compile arbitrary C-code
  - Functional and performance evaluation based on SPEC2017 and Embench

Design of FIPAC

State Updates with Pointer Authentication

- Exploit ARM Pointer Authentication for a cryptographically secure state update
- `PACIA` used to sign a pointer and store a MAC in the upper bits
- What changed in ARMv8.6A?

State Update with ARM Pointer Authentication

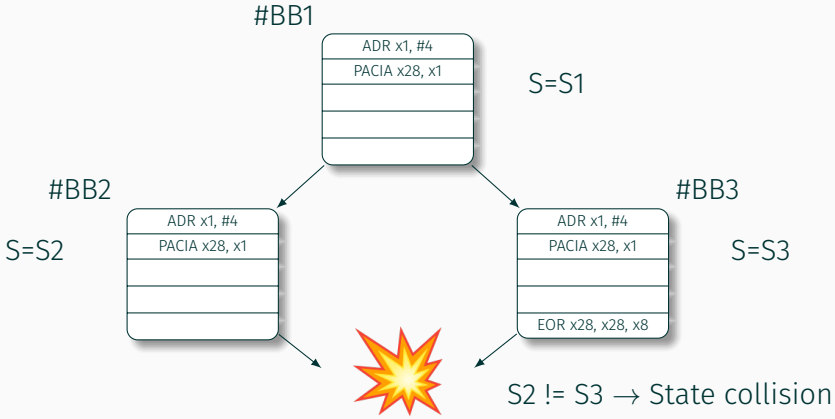


Design of FIPAC

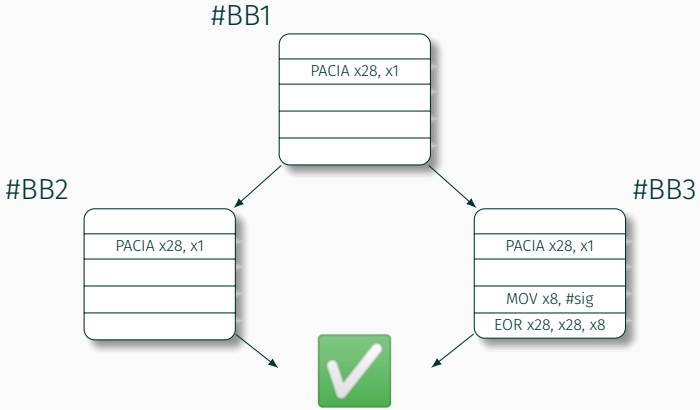
- Reserve register `x28` for the global CFI state
- Exploit PAC to implement cryptographically enforced CFI in software
  - Every basic block updates a global CFI state with a PAC instruction
  - Check instructions at different locations
  - Enforcement of the control-flow graph at basic-block level
- CFI state update with `PACIA`

```
adr   x2 ,  #4
pacia x28 , x2
```

State Updates in FIPAC



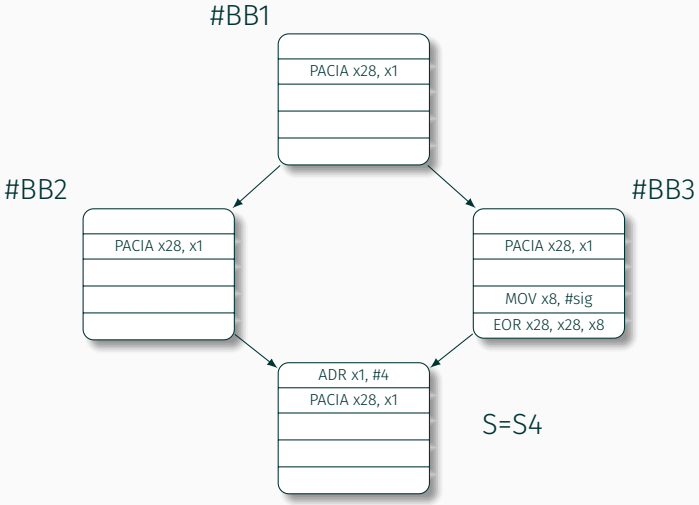
State Updates in FIPAC



Insert a justifying signature

Sig = S2 $\oplus$ S3

State Updates in FIPAC



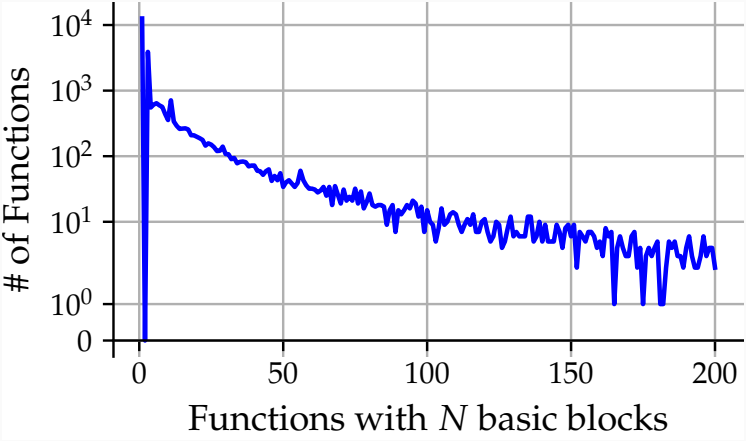
- Conditional branches
  - See example before
- Direct calls
  - Patch CFI state to a new beginning state
  - Continue execution with the callee's return state
- Indirect calls
  - Patch CFI state to an intermediate state at function entry/return
  - Custom entrypoint for indirect calls

State Checking with ARM Pointer Authentication

- `AUTIZA` used to verify the correctness of a signed pointer
- Not applicable → CFI state is not a valid pointer with PAC
- CFI state known for every program locations
  1. Transform CFI state to a valid PAC
  2. Use `AUTIZA` to validate the correctness
- CFI check sequence

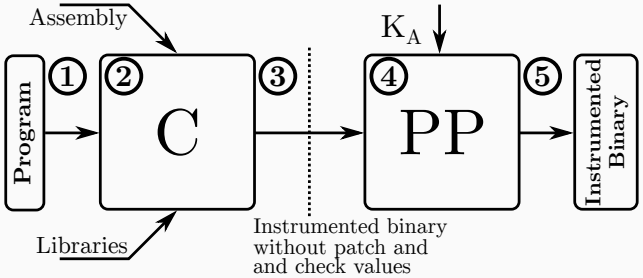
```
mov    x2 , #const
eor    x2 , x28 , x2
autiza x2
```

- Where to place a check?
  - More checks → better detection latency ✔
  - More checks → more overhead ✖
- 3 strategies implemented
  1. Single check at program end
  2. Check at every basic block
  3. Check at function end
- Custom strategies possible


# of Functions
10^4
10^3
10^2
10^1
10^0
0
0
50
100
150
200
Functions with N basic blocks

- Where to place a check?
  - More checks → better detection latency ✔
  - More checks → more overhead ✖
- 3 strategies implemented
  1. Single check at program end
  2. Check at every basic block
  3. Check at function end
- Custom strategies possible

- LLVM-based toolchain with backend passes in the AArch64 backend
- Post-processing tool for computing justifying signatures, CFI checks, and binary patching

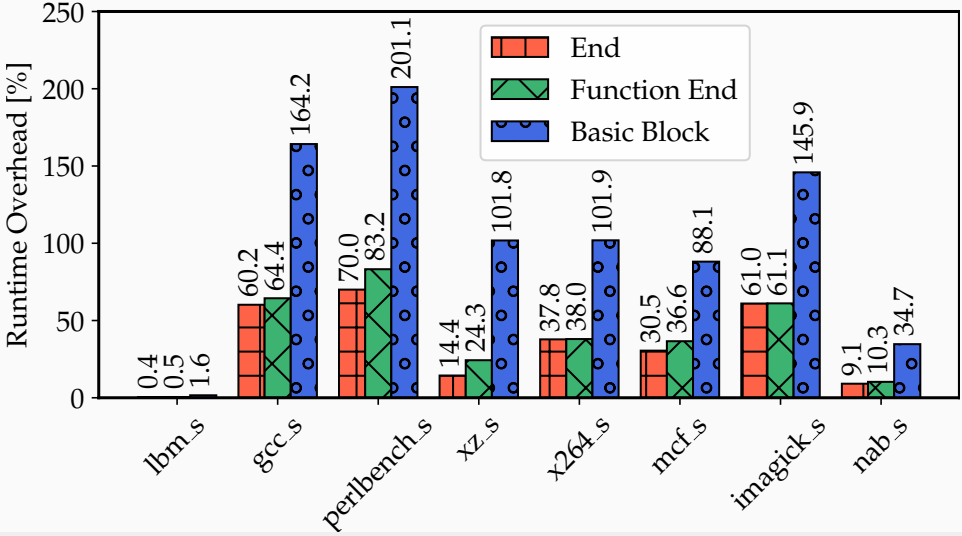


Evaluation Setup

- Functional evaluation with QEMU → We added support for ARMv8.6A's Pointer Authentication
- Performance evaluation with Raspberry Pi 4
  - No hardware support for Pointer Authentication
  - Performance overhead of `PACIA` and `AUTIZA` instructions is emulated

```
pacia x28 , x2
```

```
eor x28 , x28 , #1
eor x28 , x28 , #2
eor x28 , x28 , #3
eor x28 , x28 , x2
```


Runtime Overhead [%]
250
200
150
100
50
0
End
Function End
Basic Block
0.4
0.5
1.6
60.2
64.4
164.2
70.0
83.2
201.1
14.4
24.3
101.8
37.8
38.0
101.9
30.5
36.6
88.1
61.0
61.1
145.9
9.1
10.3
34.7
lbm_s
gcc_s
perlbench_s
xz_s
x264_s
mcf_s
imagick_s
nab_s

- Runtime overhead on SPEC2017 (geometric mean)
  - 19 %: Single check at program end
  - 63 %: Check on every basic block
  - **22 %**: Check on function end
- Checking at function end has small impact on overhead
- Better security properties as functions are typically small

Conclusion

- Basic-block granular CFI with ARM Pointer Authentication
- Pure software design with no hardware changes
- Cryptographically secure state update function
  - Protection against software- and fault attacks
- Custom LLVM-based toolchain to instrument arbitrary programs
  - `https://github.com/Fipac/Fipac`
- Support for different checking strategies

FIPAC: Thwarting Fault- and Software-Induced Control-Flow Attacks with ARM Pointer Authentication

Robert Schilling, Pascal Nasahl, Stefan Mangard
robert.schilling@iaik.tugraz.at

April 12, 2022

IAIK – Graz University of Technology