ACFA: Secure Runtime Auditing & Guaranteed Device Healing via Active Control Flow Attestation

Adam Caulfield[*], Norrathep Rattanavipanon[+], Ivan De Oliveira Nunes[*]

[*] Rochester Institute of Technology
[+] Prince of Songkla University, Phuket Campus

32nd USENIX Security Symposium
August 2023

Embedded devices - Smart Spaces & “Internet of Things”

- Low-end, energy efficient, low cost
- Resource constrained — security
- Execute safety-critical tasks in modern systems
  - Sensor/alarm system
  - Modern medical device
- Must monitor device behavior to determine unexpected/malicious activity

Can we achieve ***runtime auditing*** of a remotely deployed (potentially compromised) MCU?

Desired security guarantees for runtime auditing:

1. Generate authentic/accurate evidence of the exact runtime behavior

2. Deliver the evidence to device owner for further analysis

3. After compromise is detected, provide a means to remotely remediate the source of the compromise

Control Flow Attestation (CFA):

Generate evidence of static and runtime integrity of remote device

Verifier (Vrf)

Prover (Prv)

(1) Send Challenge *chal*

(2) Execute Software

$Exec() \rightarrow \mathrm{CF}_{\mathrm{Log}}$

(3) Produce Response

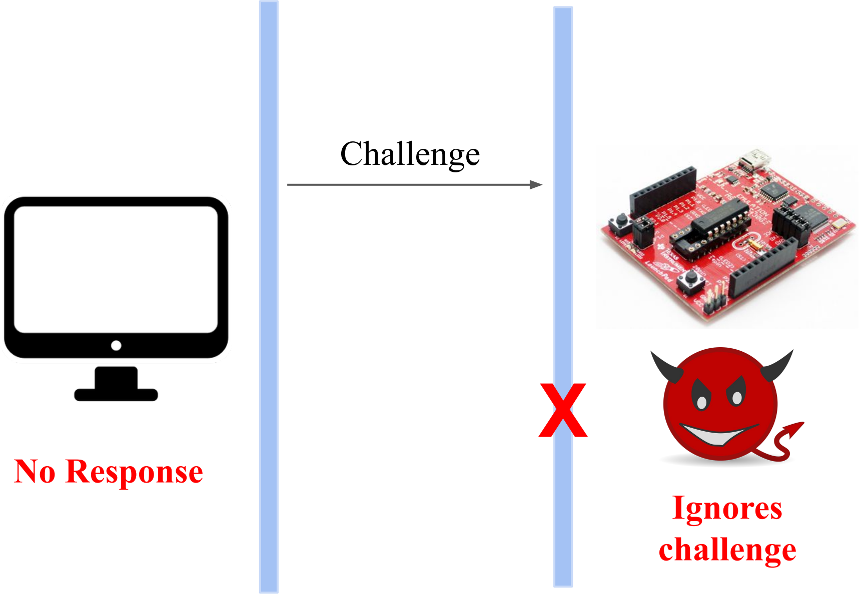
$H = Attest(chal, MEM, CF_{Log})$

(4) Send H and CF_{Log}

(5) Verify the result

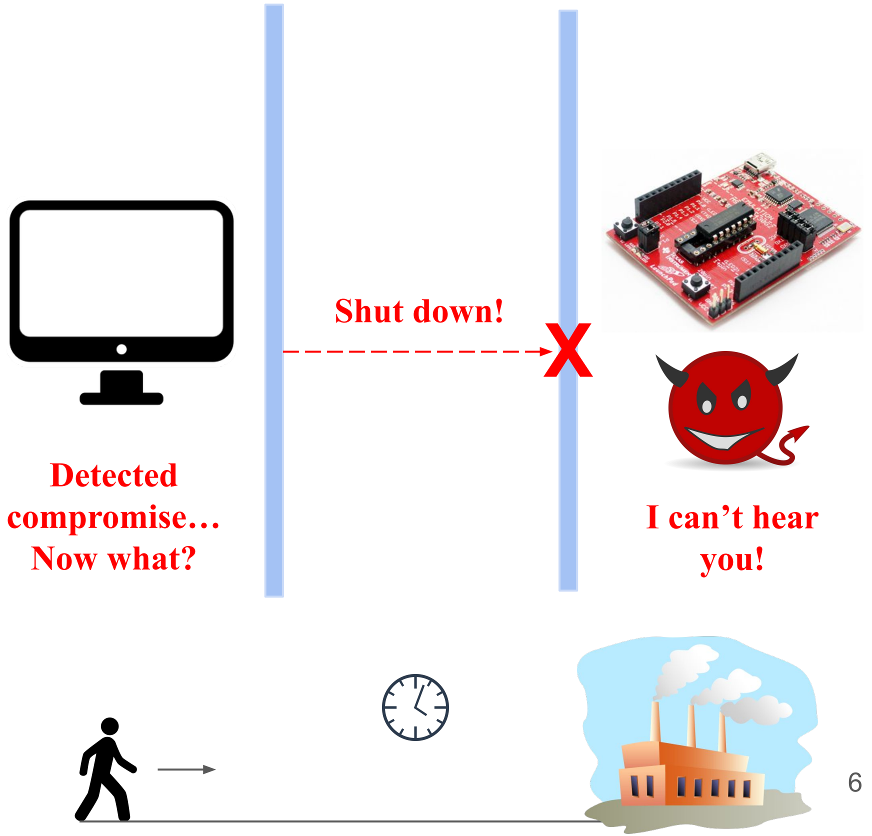
From *Attestation* to *Auditing*

- Attestation is a ***passive*** technique
- No guarantee that Verifier receives the response
- Attestation – *something is wrong*
- Auditing – *what is wrong*



After detection…

- How to resolve compromises?
- Physical intervention



Summary

Current Techniques

✔ Guarantees runtime evidence is accurate/authentic

X Cannot guarantee eventually delivery of runtime evidence to Vrf

X No ability to remotely intervene after compromise detection

To bridge this gap…

Our work, ACFA: Active Control Flow Attestation

- ✔ Guarantees runtime evidence is accurate/authentic
- ✔ Guarantees Vrf eventually receives runtime evidence
- ✔ Enables remote device healing: trusted mechanism executes upon detection

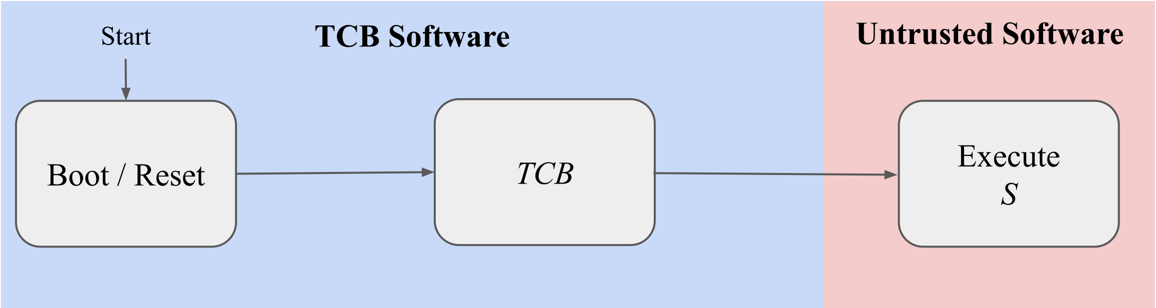
Active CFA (ACFA) Overview

1. Key Idea:
    a. Extend conventional CFA to include communication of evidence (H, CF_{Log}) in TCB
    b. Hardware that generates CF_{Log} and *actively* triggers generation/transmission of response
    c. Hardware support ensures healing mechanism executes the moment compromise is detected

Active CFA (ACFA) Overview

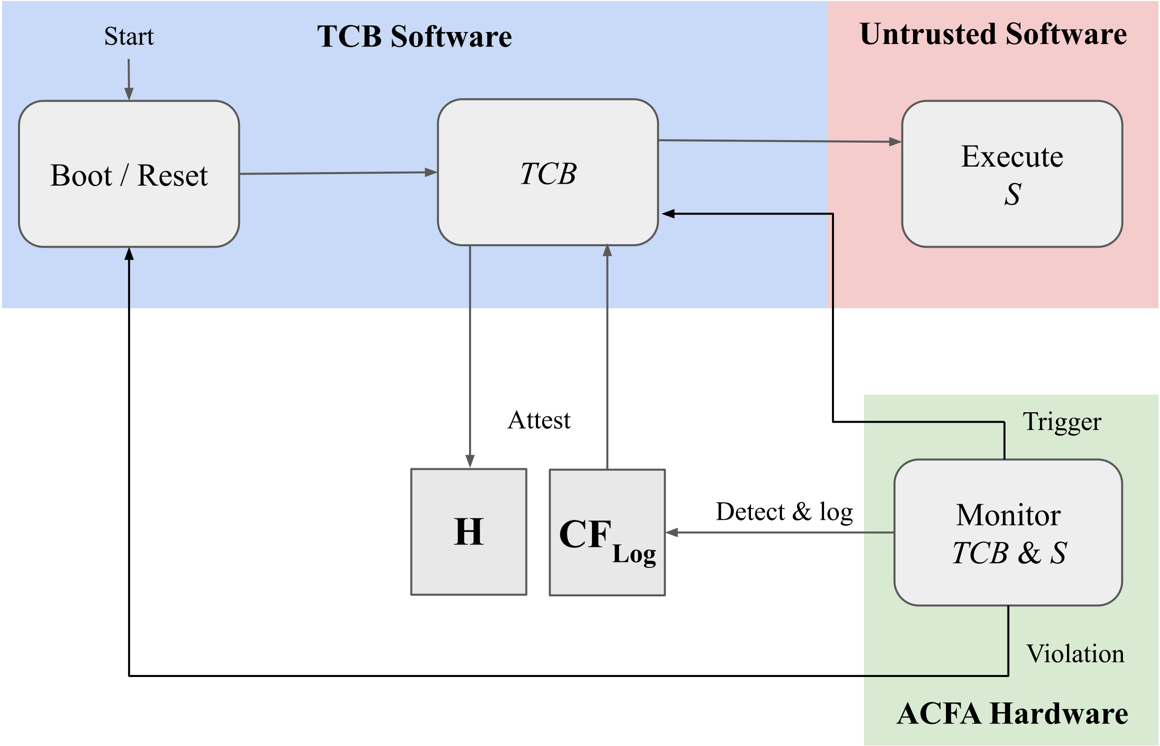
1. Key Idea:
    a. Extend conventional CFA to include communication of evidence (H, CF_{Log}) in TCB
    b. Hardware that generates CF_{Log} and ***actively*** triggers generation/transmission of response
    c. Hardware support ensures healing mechanism executes the moment compromise is detected
2. Low-cost hybrid (software/hardware) architecture for MCUs
    a. Low-cost hardware extension to protect memory, trigger TCB, and record MCU's control flow
    b. Software for attestation/communication of evidence and healing mechanism

ACFA Workflow



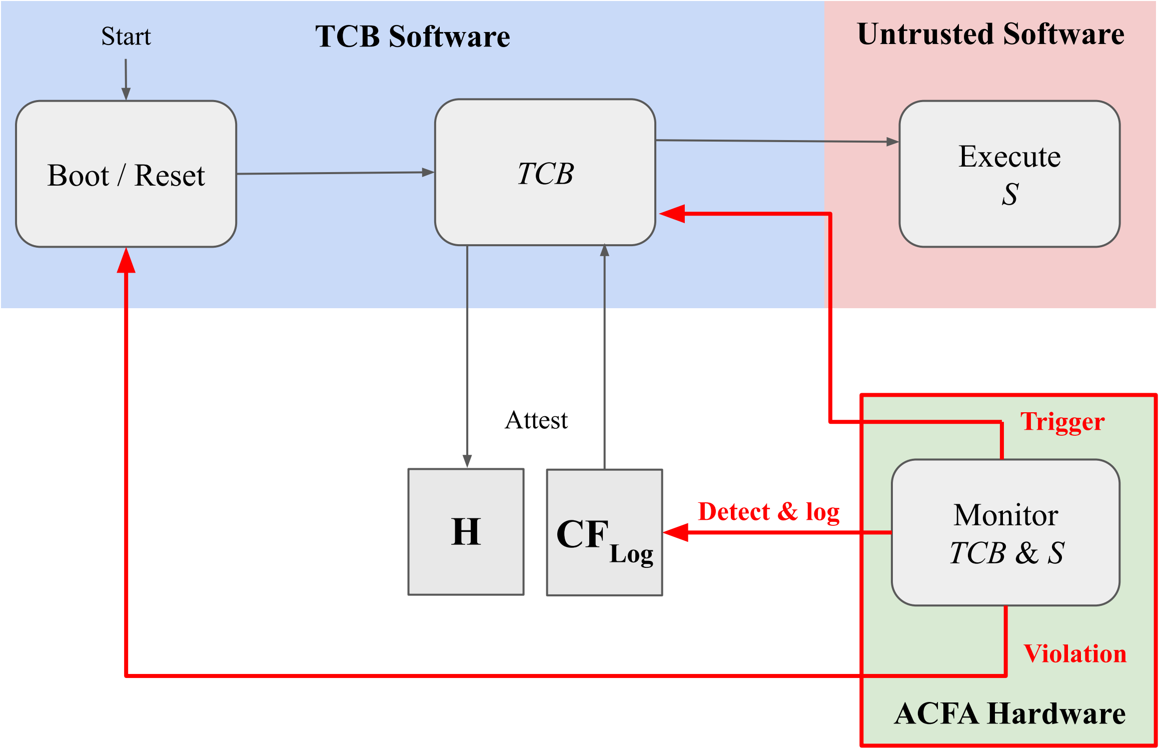
- MCU program memory contains ACFA's TCB and the application software (S)

ACFA Workflow



- Establish an active root of trust with additional guarantees through hardware extension

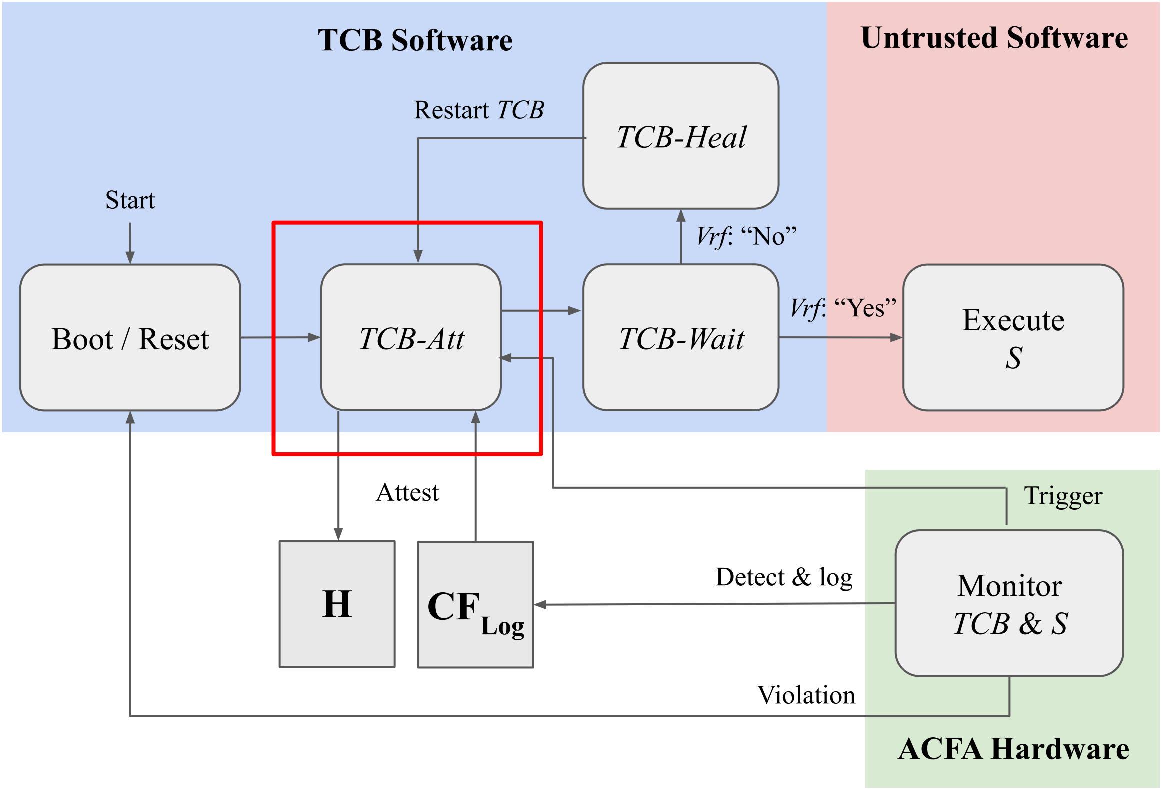
ACFA Workflow



ACFA Hardware:

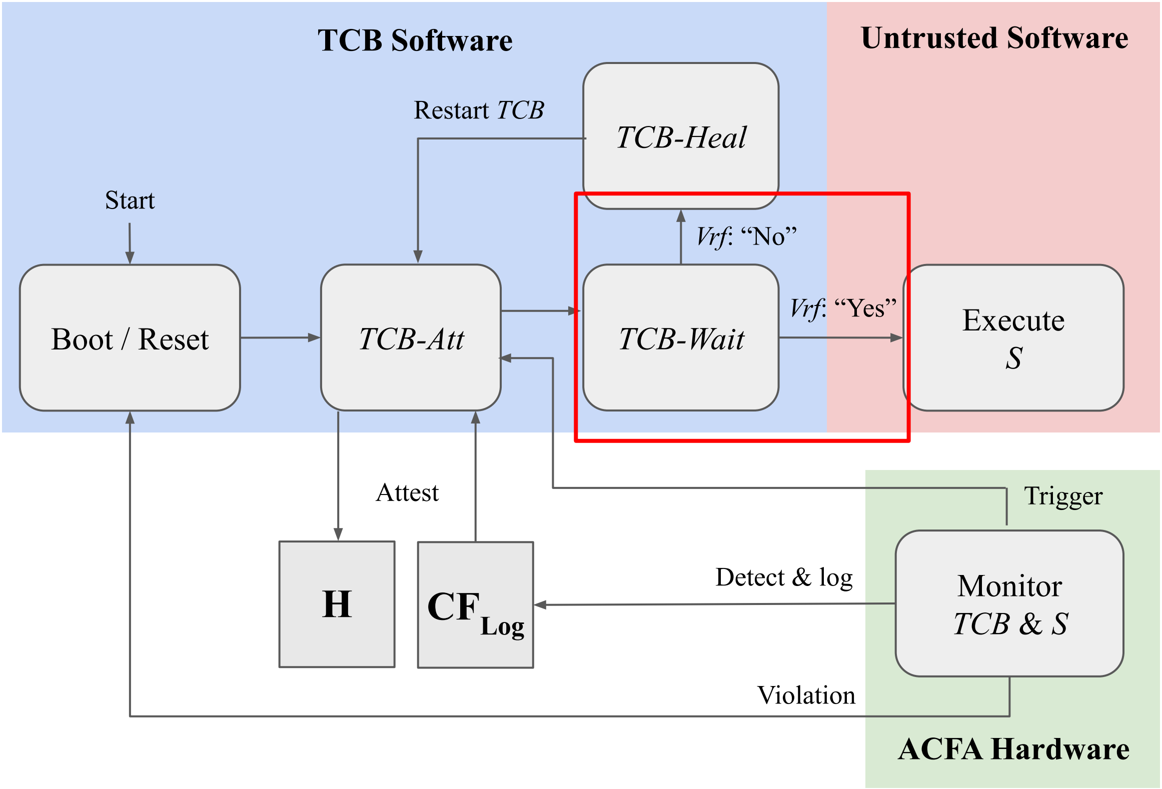
- Detects branching instructions
- Appends CF_{Log}
- Protects software & critical data from illegal modifications
- Generates custom **non-maskable interrupt** to trigger TCB

ACFA Workflow



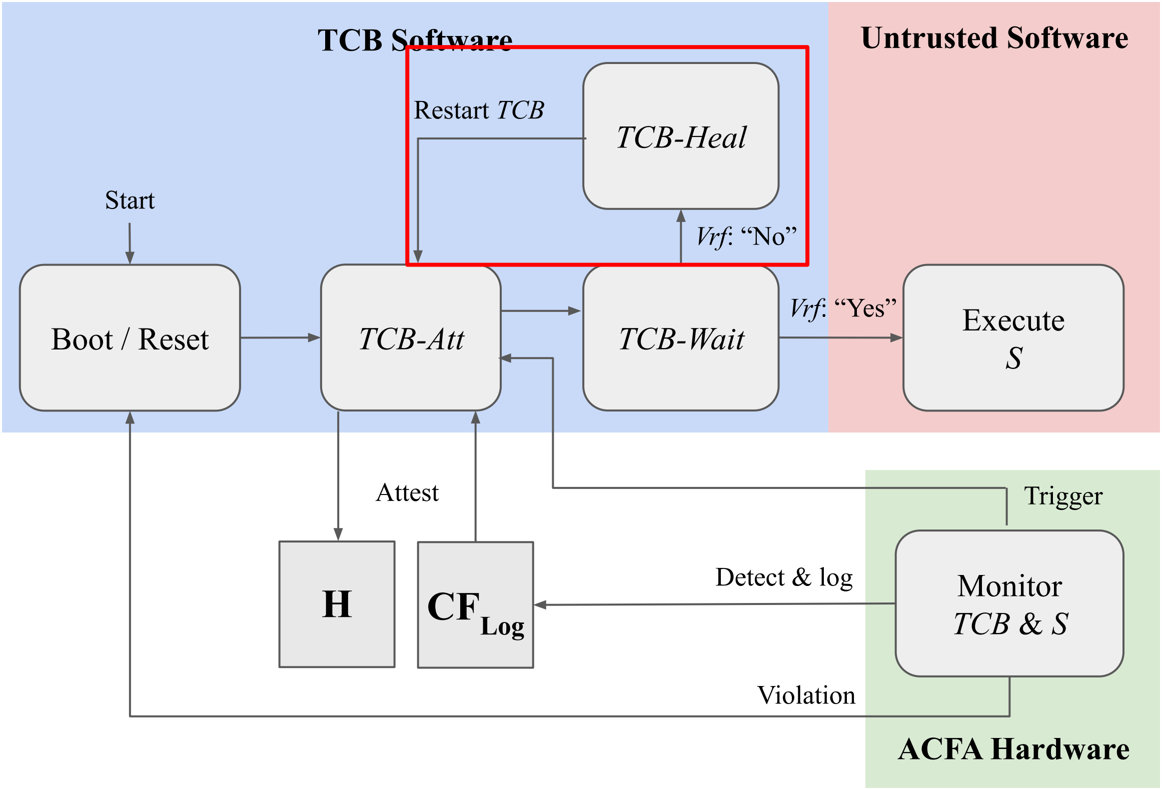
- TCB Executes three submodules atomically:
- Attestation (*TCB-Att*) always executes first
- Computes *H* over the CF_{Log} and significant memory regions

ACFA Workflow



- Next, the TCB communicates with the Verifier and waits for a response (*TCB-Wait*)
- It continues to transmit the response until it receives an authenticated message back from Verifier
- Next module depends on detection

ACFA Workflow



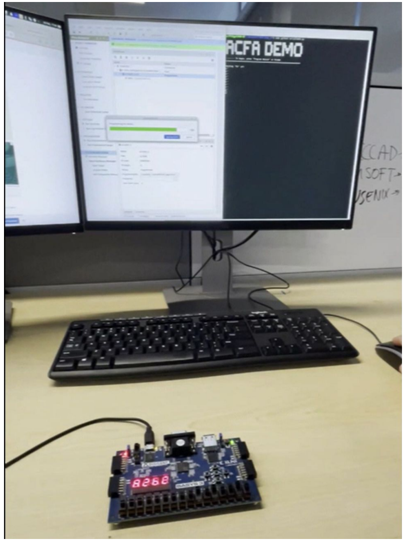
- If verification fails, TCB automatically executes a healing action (*TCB-Heal*)
- This is configurable by the Verifier prior to device deployment
- Always followed by *TCB-Att*

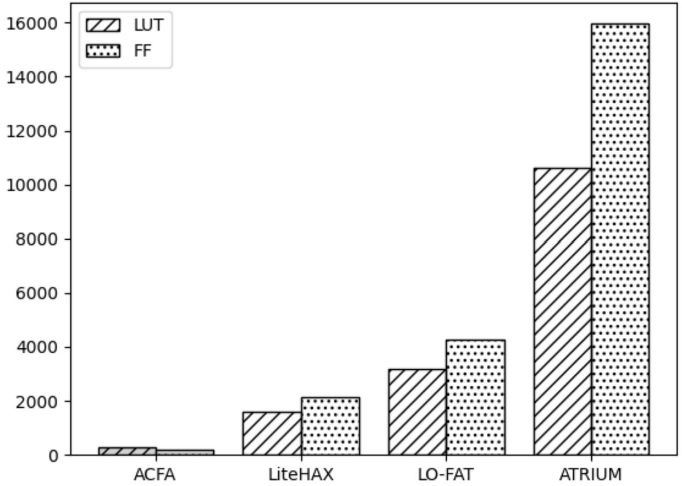
Cost Evaluation

- No runtime overhead to log control flow transfers

- Evaluate hardware cost compared to hardware-based CFA

- Hardware Cost: 275 LUTs, 202 FFs
    - 5.8x less LUTs, 10.5x less FFs than LiteHAX





Thank you

Paper link:

- Available on arXiv
- https://arxiv.org/abs/2303.16282

ACFA Repository:

- Available from RIT-CHAOS-Sec on Github:
- www.github.com/RIT-CHAOS-SEC/ACFA

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Contact information:

- Email: ac7717@rit.edu