Holistic Control-Flow Protection on Real-Time Embedded Systems with Kage

Yufei Du (UNC-CH, UR), Zhuojia Shen (UR), Komail Dharsee (UR), Jie Zhou (UR),

Robert J. Walls (WPI), John Criswell (UR)

Microcontroller (MCU)

Drone

Home IoT

SSD Controller

Automobile

Vulnerable to control-flow hijacking!

Control-Flow Hijacking

- Code injection
- Return to libc
- Return-oriented programming
- Jump-oriented programming

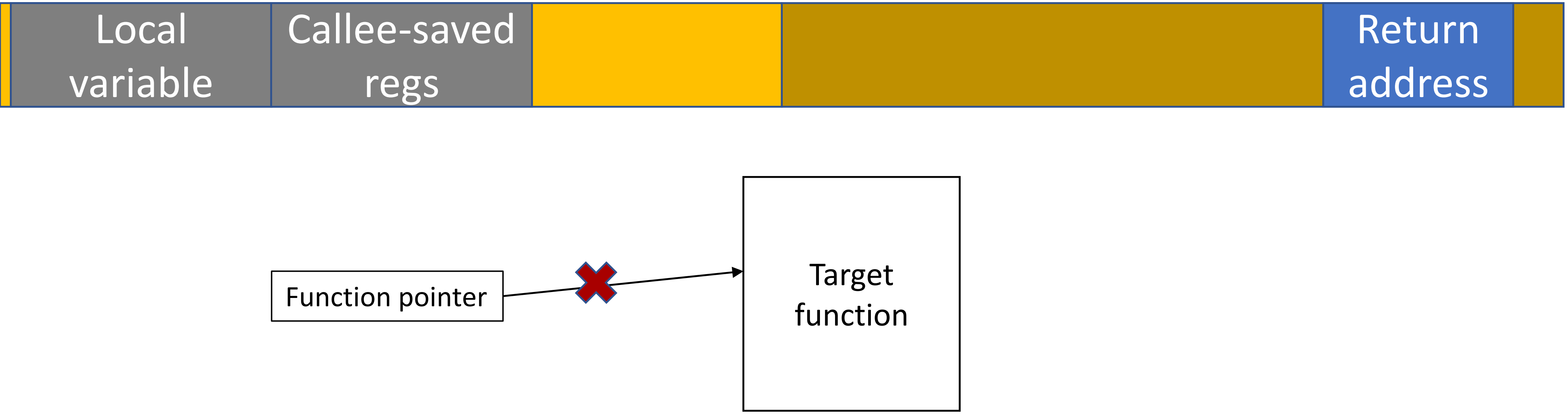
All begins by overwriting control data!

Common Defenses

- Memory safe language
- Address space layout randomization
- Shadow stack + control-flow integrity (CFI) check

Common Defenses

Shadow stack (Chiueh and Hsu. ICDCS 2001) + control-flow integrity (CFI) check (Abadi et al. CCS 2005)



Shadow Stack + CFI Check

Previous work in MCU-based embedded devices

Name	Supports OS	Return Address	Function Pointer	Processor State	Input Validation for Kernel API	Performance Overhead
CaRE[1]	×	✓	✓	—	—	13%-513%*
Silhouette[2]	×	✓	✓	—	—	1.3%
RECFISH[3]	✓	✓	✓	△	×	21%

1. Nyman et al. RAID 2017
2. Zhou et al. USENIX Security 2020
3. Walls et al. ECRTS 2019

Shadow Stack + CFI Check

Previous work in MCU-based embedded devices

Name	Supports OS	Return Address	Function Pointer	Processor State	Input Validation for Kernel API	Performance Overhead
CaRE[1]	×	✓	✓	—	—	13%-513%*
Silhouette[2]	×	✓	✓	—	—	1.3%
RECFISH[3]	✓	✓	✓	△	×	21%
Kage	✓	✓	✓	✓	✓	5.19%

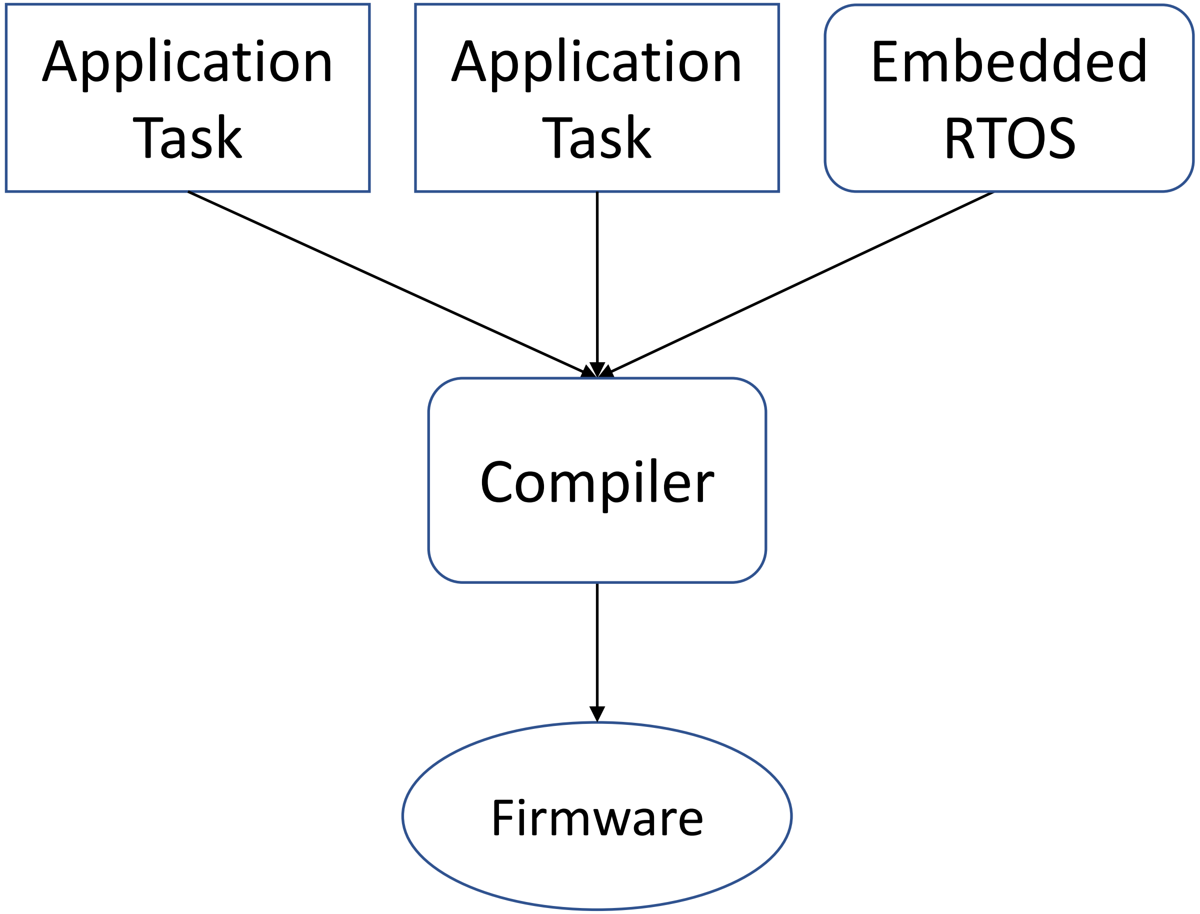
1. Nyman et al. RAID 2017
2. Zhou et al. USENIX Security 2020
3. Walls et al. ECRTS 2019

Kage

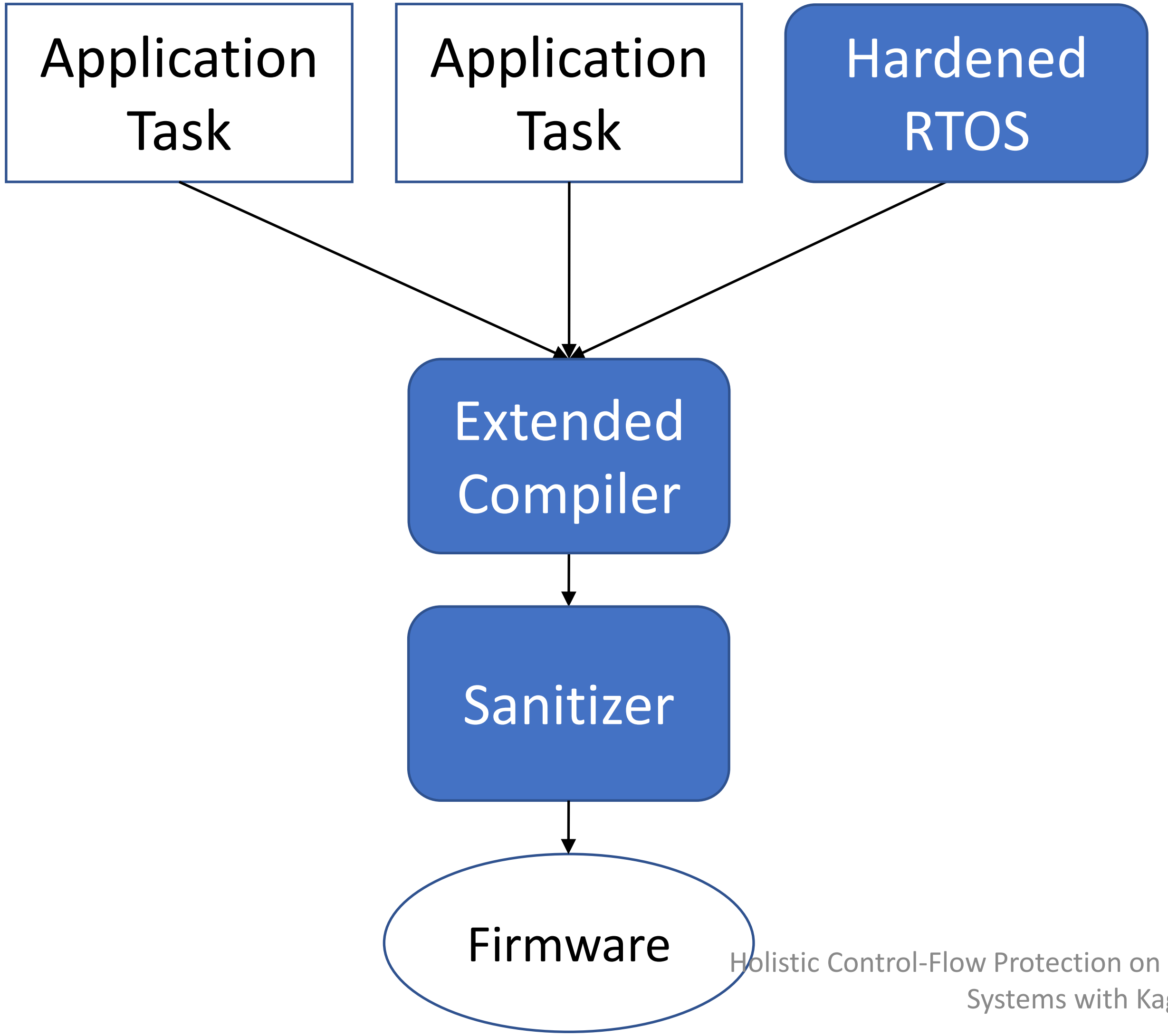
- Efficient control-flow hijacking defense on existing ARMv7 devices with real-time operating system
- FreeRTOS-based Implementation for ARMv7-M devices
  - No ARM TrustZone
  - Memory protection unit (MPU)
    - Read, write, execute

Ka Ge
か げ
影
(Shadow)

Workflow for Firmware Developers



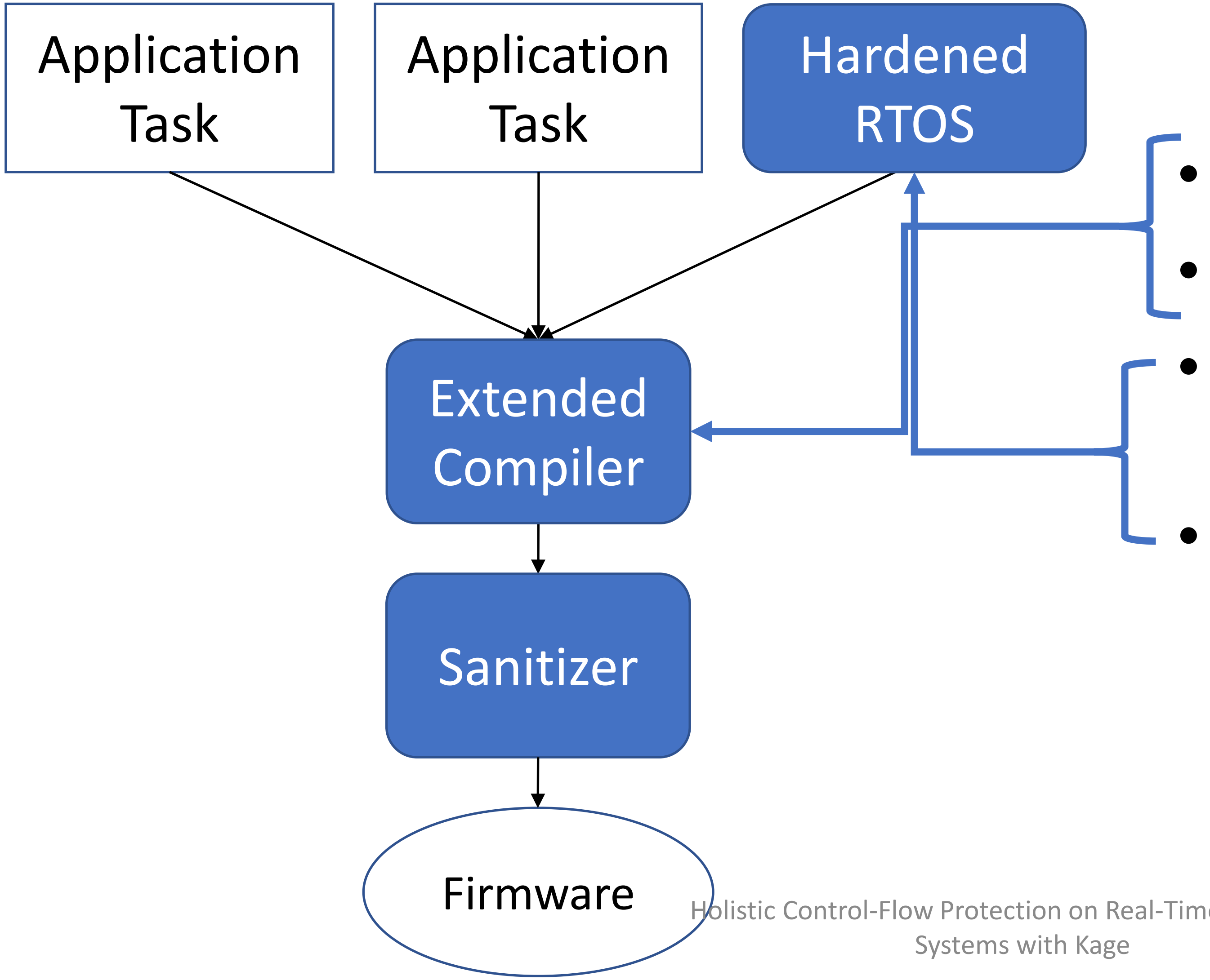
Kage Design



Highlights:

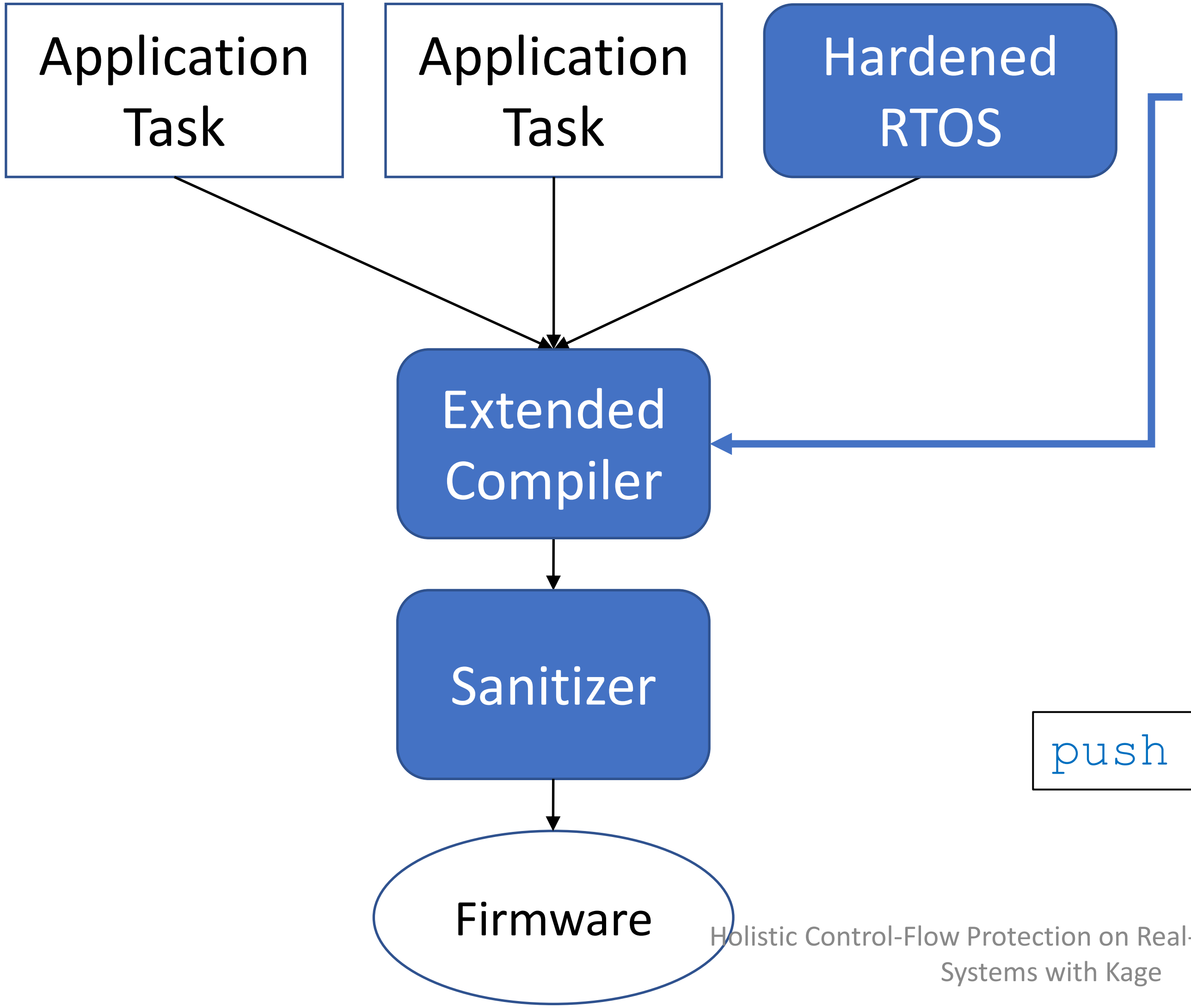
- Protects control data efficiently
- Prevents bypassing the protections

Design: Protecting Control Data



- Return address
- Function pointer
- Processor state during context switch
- Processor state during exception

Design: Efficient Memory Access Control



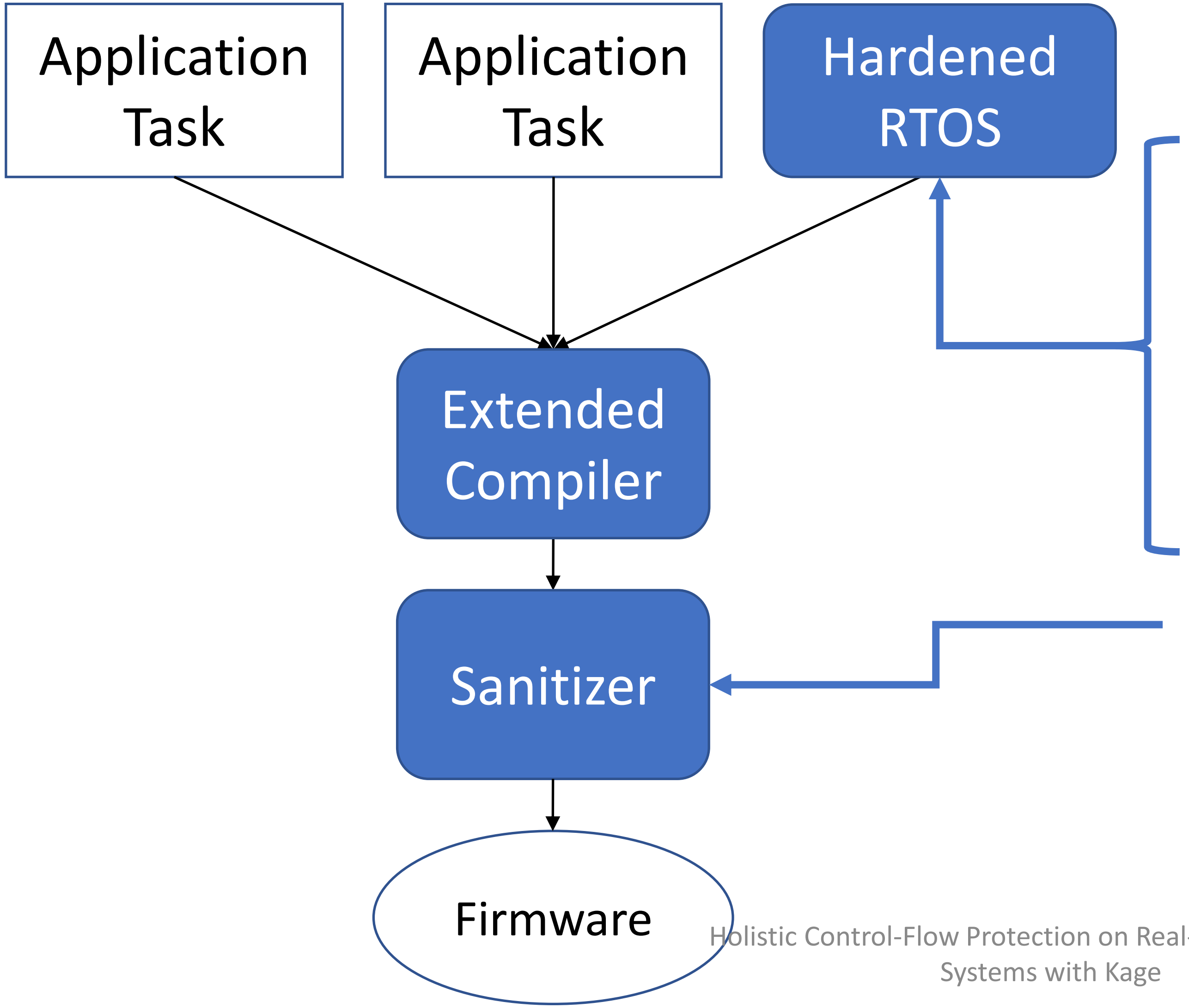
Store Hardening

- Silhouette (Zhou et al. USENIX Security 2020)
- Unprivileged store instructions: STRT, STRBT, STRHT

```
push {r0, r1}
```

```
sub        sp, #8
strt r0, [sp, #0]
strt r1, [sp, #4]
```

Design: Preventing Bypass



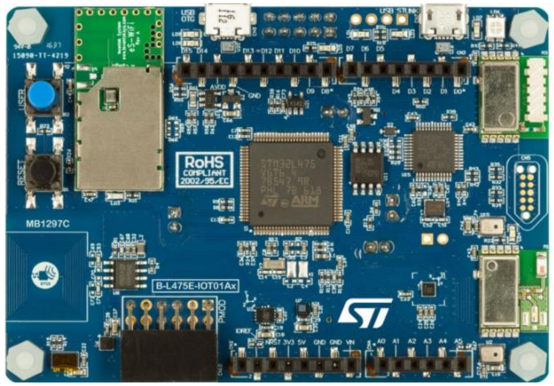
- Isolated memory regions
- Small trusted computing base
- Runtime checks for kernel API calls
- Checking programming mistakes

Evaluation

- Performance impact on embedded application
- Performance overhead of each individual mechanism
- Effectiveness against return-oriented programming

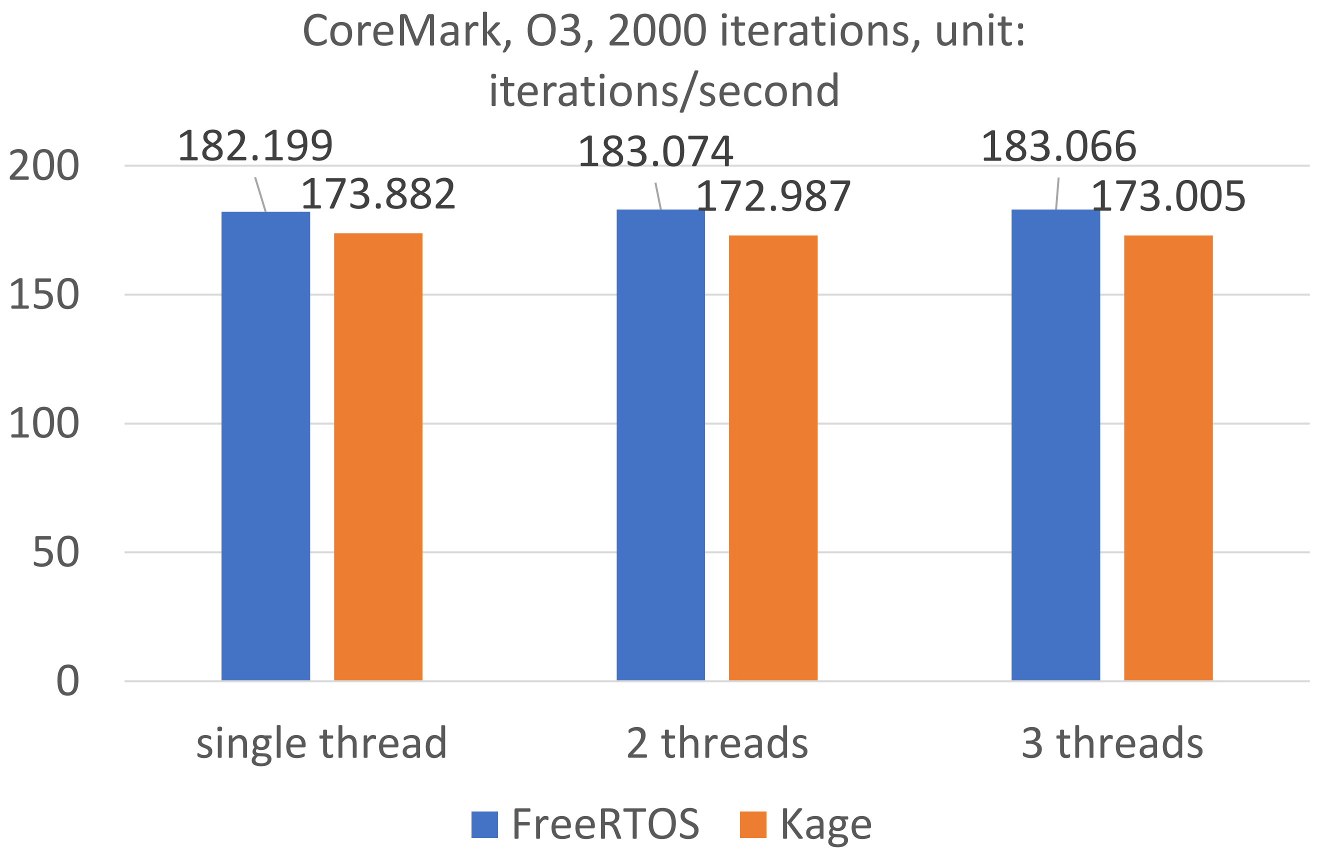
Experimental Setup

- Hardware: STM32L475 IoT Node
- Benchmark: CoreMark
- Baseline: Default FreeRTOS compiled with LLVM 9
- Average of 3 runs



Performance Evaluation Results

- Average: 5.19% overhead
- Slightly higher overhead with more threads
  - 4.56% for single thread
  - 5.50% for 3 threads

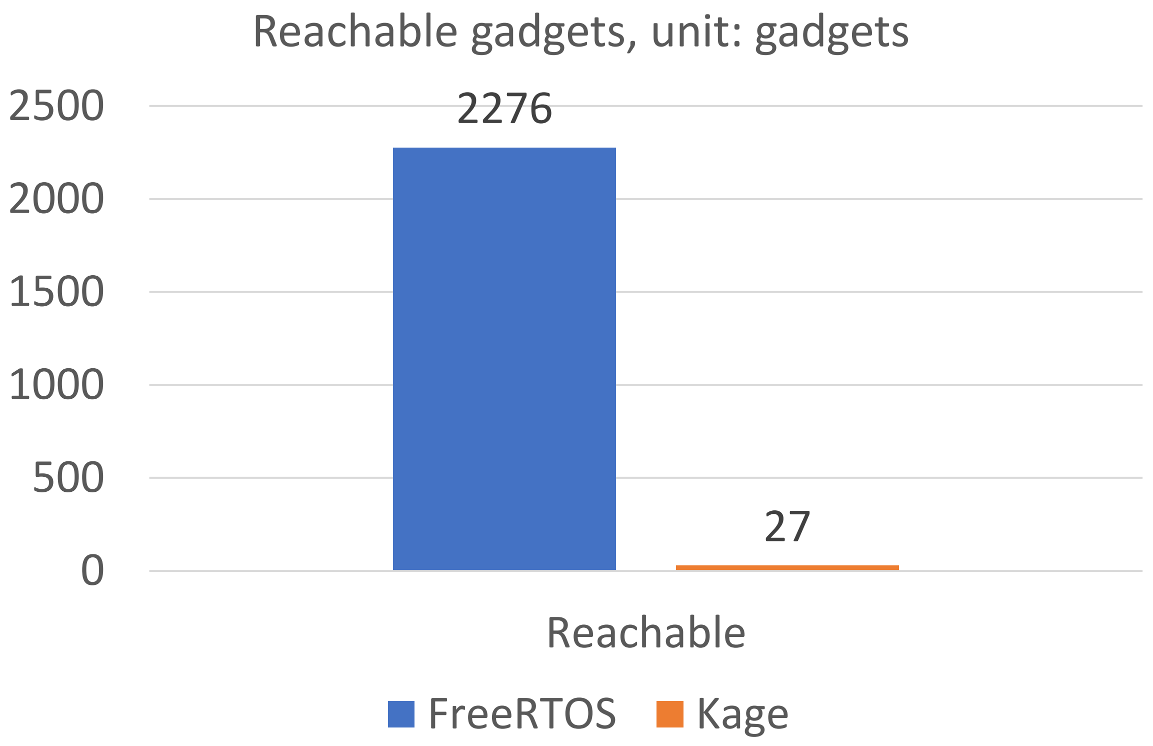


Security Evaluation Setup

- CoreMark firmware with 3 threads
- ROPGadget in ARMv7-M instruction set
- Analyzed reachable gadgets for effectiveness

Security Evaluation Results

- 98.8% less reachable gadgets
- 27 remaining gadgets
  - 17 in the beginning of application function
  - 10 immediately after direct call in application
  - All ends in return or direct call
  - Cannot be stitched together



Conclusion

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

- We presented Kage, an efficient control-flow hijacking defense for ARMv7 architectures with RTOS
- Kage protects control data including return address, function pointer, and processor state, and Kage prevents bypassing the protections
- Kage incurs only 5.19% performance overhead in CoreMark benchmark, lower than previous work
- Open source: https://github.com/URSec/Kage