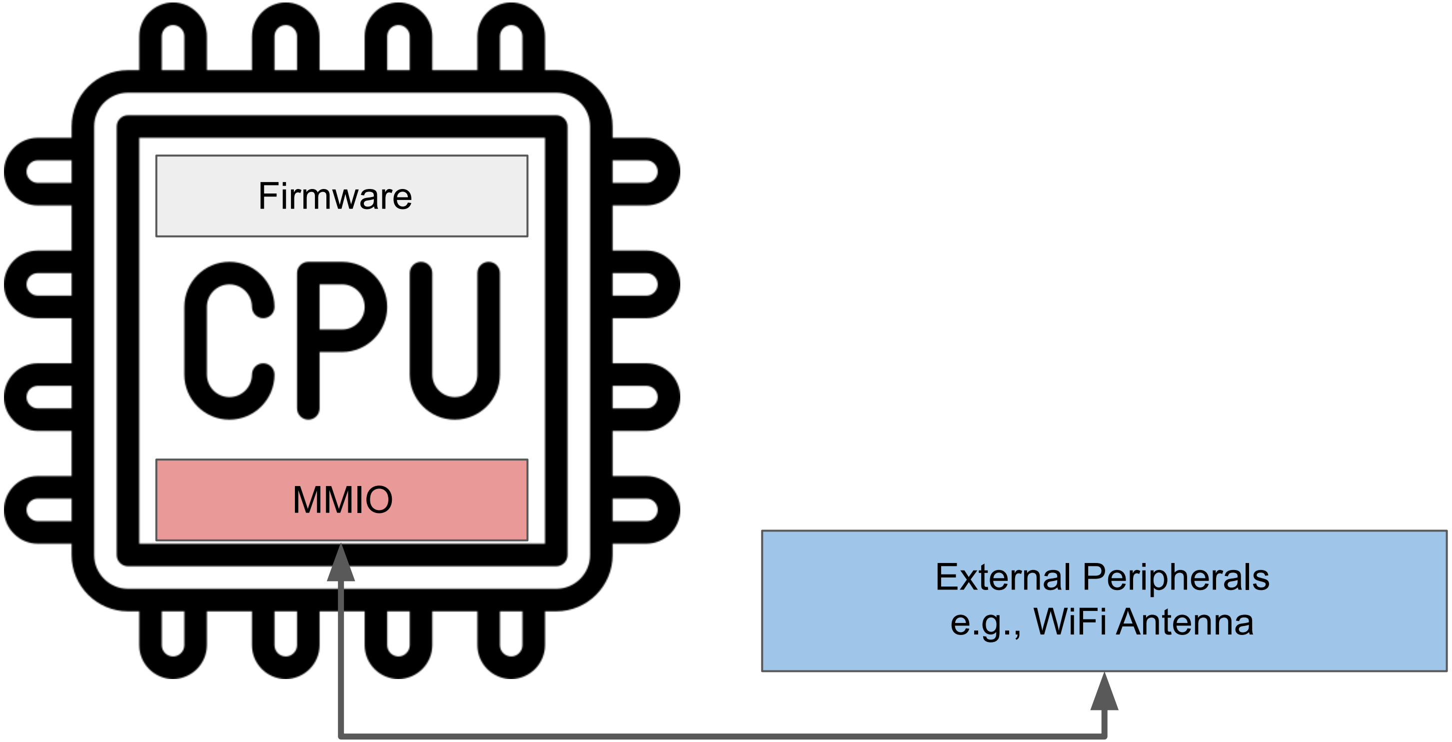
Forming Faster Firmware Fuzzers

***Lukas Seidel**, Qwiet AI and TU Berlin*
Dominik Maier, TU Berlin
Marius Muench, VU Amsterdam and University of Birmingham

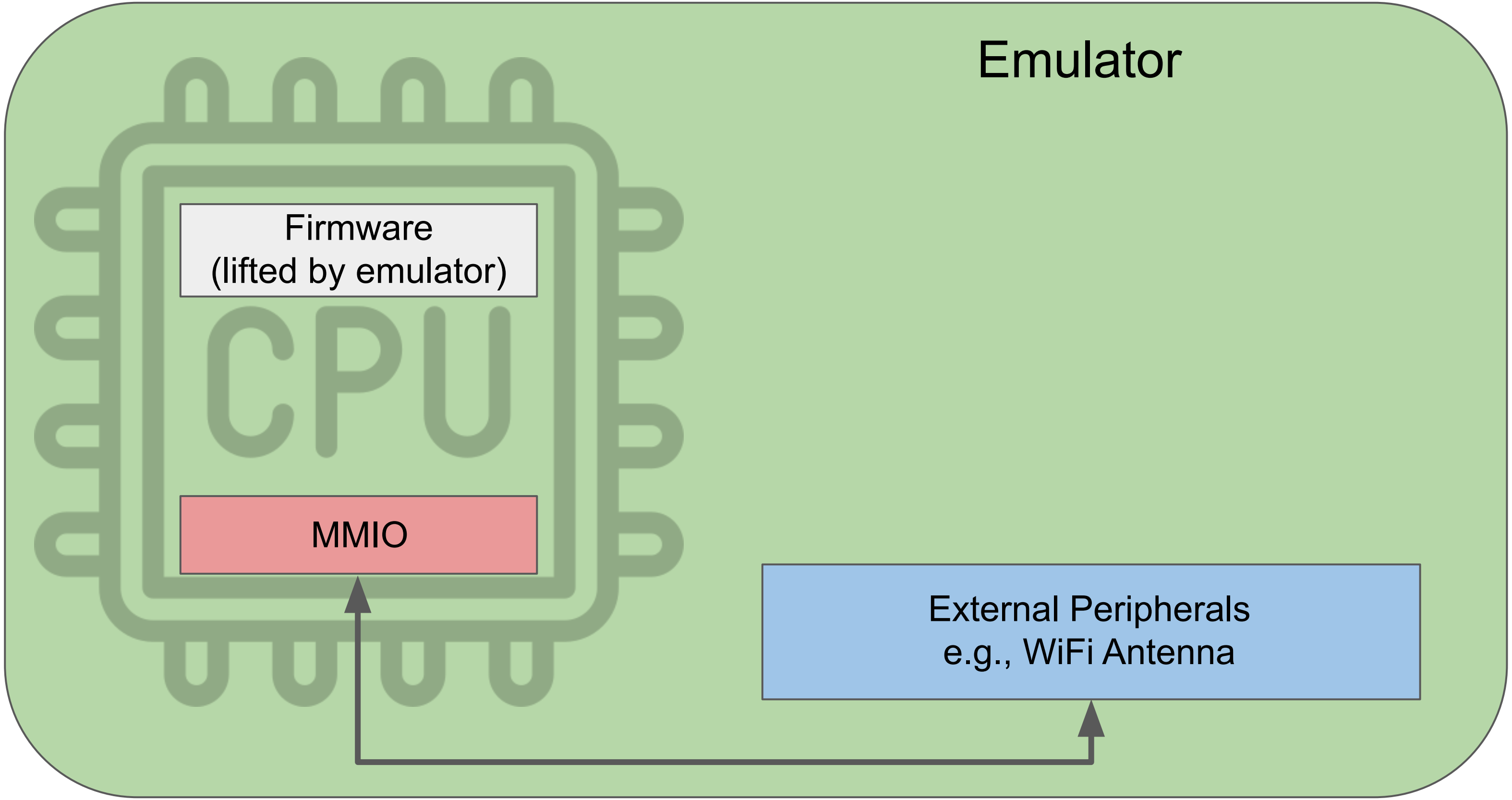
The 32nd USENIX Security Symposium

Our Goal:
Re-Think Firmware Emulation for Fuzzing

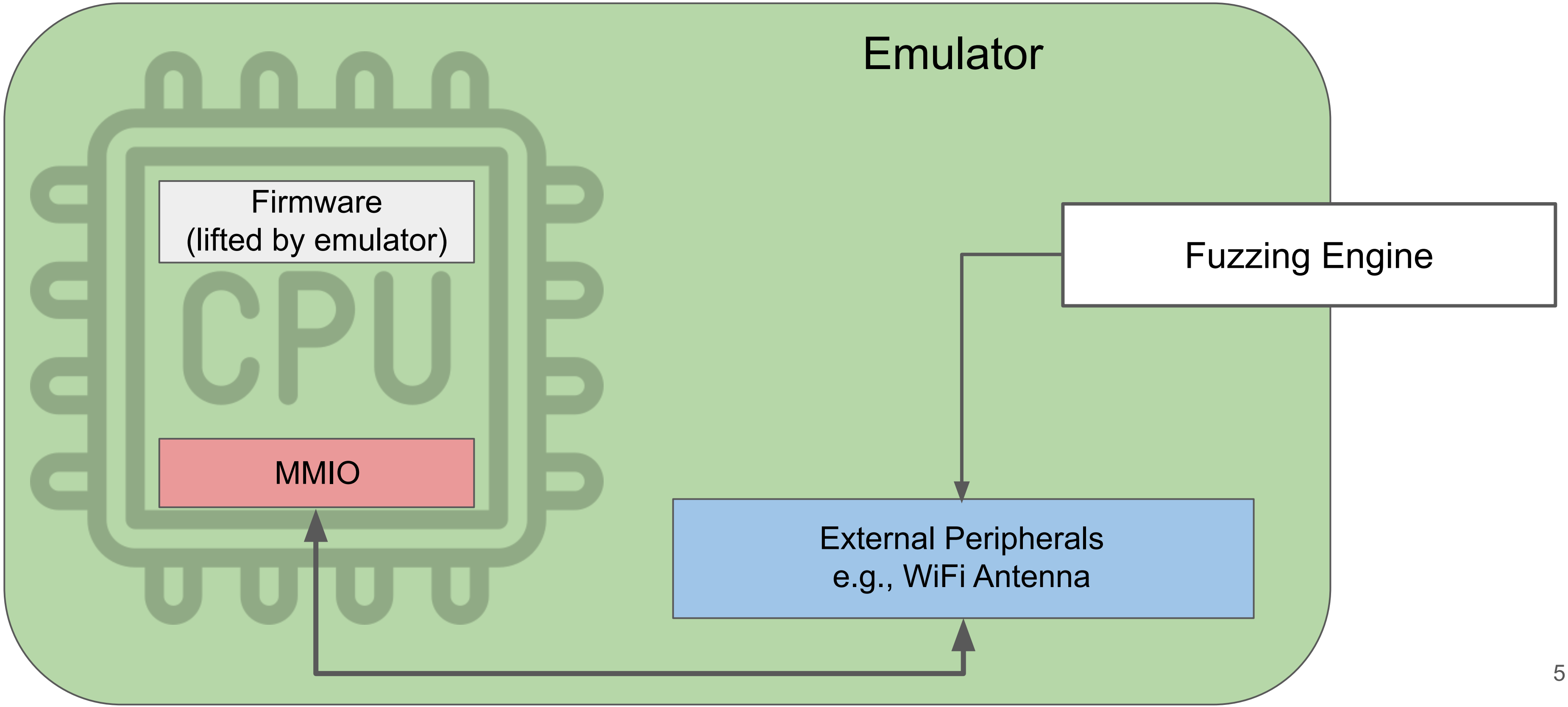
Firmware Fuzzing



Firmware Fuzzing



Firmware Fuzzing



Observations

1) Full Binary lifting / rewriting (even if heavily cached) is expensive. QEMU's advantage is executing diverse architectures but most embedded work focuses on ARM.

2) QEMU was developed for more complex systems, deploying a SoftMMU which dispatches all memory accesses and introduces significant overhead

For more roadblocks that we addressed, please refer to our paper.

Near-Native Rehosting

Core Idea:

a) *A lot of embedded firmware runs on ARMv7-M chips*
b) *Certain ARMv8-A cores provide compatibility with AArch32 and Thumb instruction set variants*

⇒ Execute binaries for small embedded devices on their "bigger brothers"!

By this, we

- Heavily reduce the amount of code which needs lifting / rewriting
- outperform rehosting approaches built on top of general-purpose emulators

Reduced Memory Access Overhead

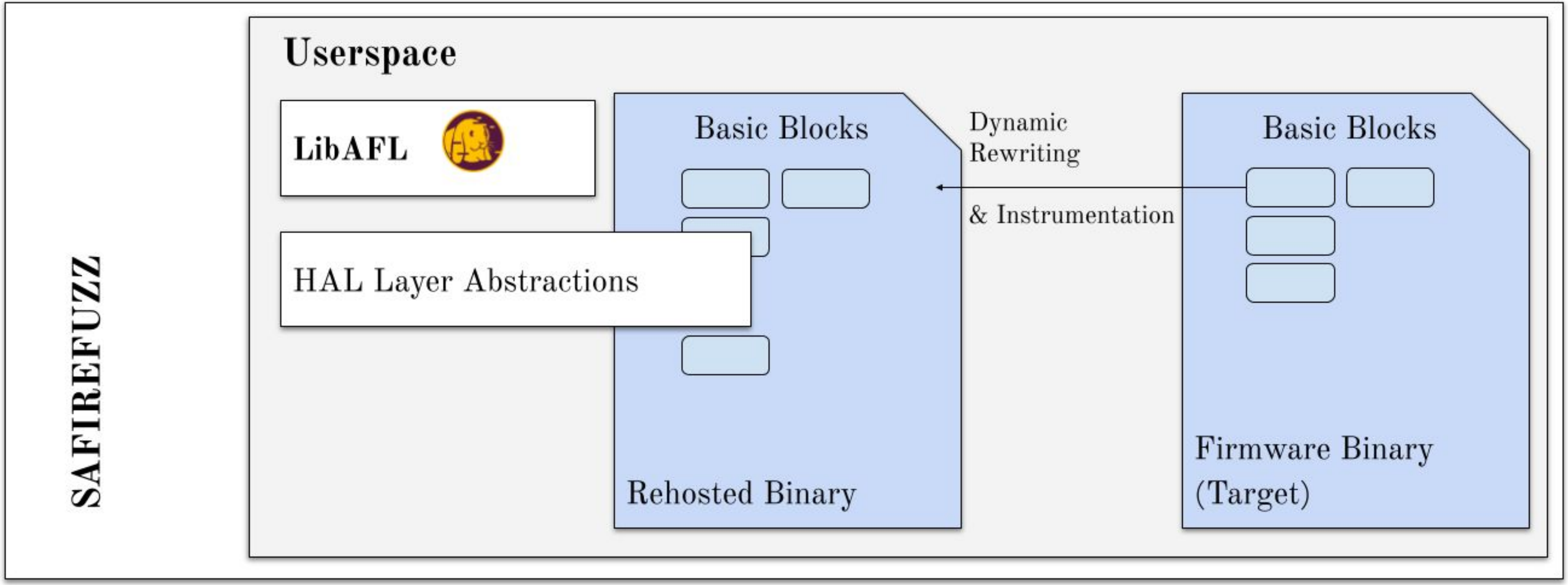
- Mirror memory layout of the embedded device in userspace

  ⇒ rewritten instructions do not need extra logic to dispatch memory accesses

- Use your usual MMU to detect memory violations

  ⇒ no need for overhead-inducing SoftMMU

The Framework



High-Level Emulation

- Search for functions accessing MMIO peripherals (HAL)
- Emulate their behavior in a high-level language (handler)
- Insert hooks to your handler while rewriting

⇒ Eliminate problematic MMIO accesses

```
/// Return fake FatFs FILE object
pub unsafe fn f_open(file_ptr: u32, _path_ptr: u32, _mode_byte: u32) →
u32 {
    let buf_ptr: u32 = crate::handlers::malloc(size: FUZZ_LEN);

    if FUZZ_INDEX == 0 {
        ptr::copy_nonoverlapping(src: FUZZ_INPUT.as_ptr(), dst: buf_ptr
        as *mut u8, count: FUZZ_LEN as usize);
        FUZZ_INDEX += FUZZ_LEN;
    } else {
        #[cfg(feature = "dbg_prints")]
        println!("Ran out of fuzz after populating one file with f_read");
        utils::exit_hook_ok();
        unreachable!();
    }

    let mut dummy_obj: FDID = FDID::default();
    dummy_obj.objsize = FUZZ_LEN as _;
    let new_file: File = File {
        obj: dummy_obj,
        flag: 0x1,
        err: 0,
        fptr: 0,
        clust: 1,
        sect: 0,
    };
    ptr::copy_nonoverlapping(src: &new_file as *const _, dst: file_ptr as
    *mut File, count: 1);
    0
} fn f_open
```

Basic Block Rewriting

Original Basic Block

```
0x10000: movs   r0, #0
0x10002: movs   r1, #0
0x10004:
    ldr r3, [pc, #0x30]
0x10006: cmp    r3, #1
0x10008: beq    #0x20e
```

PC-relative:

rewrite to load from absolute address

Rewritten Basic Block

```
movs     r0, #0
movs     r1, #0
movt     r3, #0x1
movw     r3, #0x34
ldr      r3, [r3]
cmp      r3, #1
push {r0-r12, lr}
mov r0, #SUCC_0_ADDR
blx rewrite_bb
mov r0, #SUCC_1_ADDR
blx rewrite_bb
blx resolve_branch
pop {r0-r12, lr}
nop
```

Rewritten Basic Block after first Execution

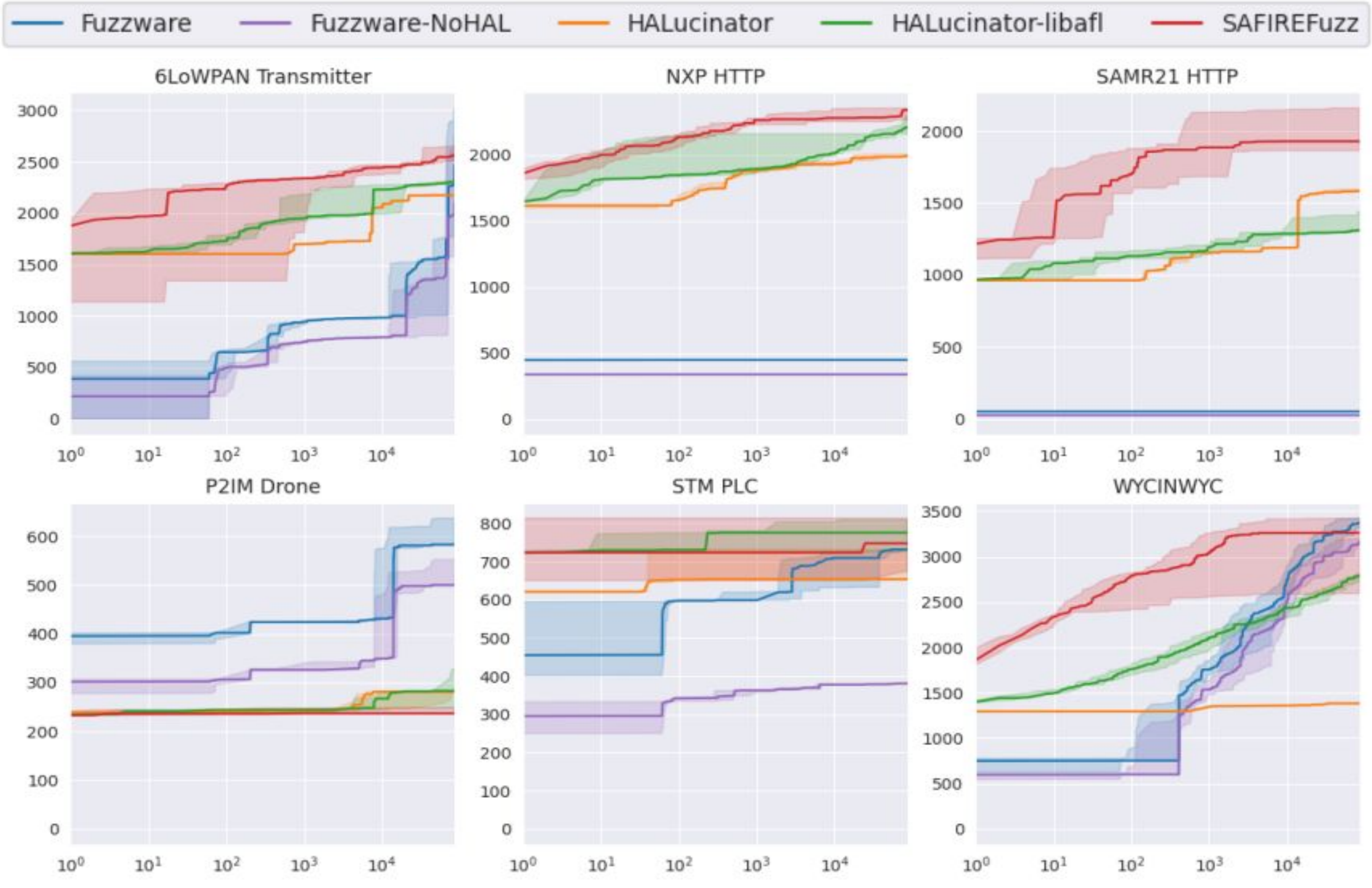
```
movs     r0, #0
movs     r1, #0
movt     r3, #0x1
movw     r3, #0x33
ldr      r3, [r3]
cmp      r3, #1
b        #12
mov r0, #SUCC_0_ADDR
blx rewrite_bb
mov r0, #SUCC_1_ADDR
blx rewrite_bb
blx resolve_branch
pop {r0-r12, lr}
beq #RESOLVED_ADDR
```

Evaluation

- 12 targets previously fuzzed by other firmware fuzzing work, e.g.,
  - STM32-based PLC firmware
  - HTTP Server for Atmel SAM R21 microcontrollers
  - Contiki OS-based WiFi Receiver/Transmitter
  - A fuzzing benchmark firmware with artificial vulnerabilities (*What You Corrupt Is Not What You Crash*)

Evaluation

- 12 targets previously fuzzed by other firmware fuzzing work, e.g.,
  - STM32-based PLC firmware
  - HTTP Server for Atmel SAM R21 microcontrollers
  - Contiki OS-based WiFi Receiver/Transmitter
  - A fuzzing benchmark firmware with artificial vulnerabilities (*What You Corrupt Is Not What You Crash*)
- 4 baseline configurations
  - HALucinator (state-of-the-art HLE-based)
  - HALucinator-LibAFL
  - FuzzWare (state-of-the-art symbolic execution-based)
  - FuzzWare-NoHAL



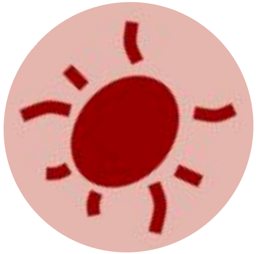
Basic Block Coverage

Performance 📈

690x faster than HALucinator

145x faster than FuzzWare

New Targets



- 2 previously unfuzzed targets
  - Sine: open-source firmware for electric motor inverters
  - STMicroelectronics firmware example for image processing (libjpeg)
- 3 new Bugs
  - Sine:
    - Arbitrary write by corrupted config value (probably not exploitable)
  - Libjpeg:
    - Segfault after accessing uninitialized struct
    - Out-of-bounds write

Conclusion

⇒ Near-native execution, minimal rewriting

⇒ Rehosting of embedded firmware in Linux userspace

⇒ Vastly increased execution speeds

⇒ Less time to achieve (more) coverage

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

SCAN ME

pr0me