Rethinking Emulation for Fu(zzi)n(g) and Profit:

Near-Native Rehosting for Embedded ARM Firmware

Lukas Seidel
Binarly REsearch Team

Overview

- Intro to Embedded Firmware Fuzzing
- Overview of Rehosting Approaches and Systems
- Performance Roadblocks
- Near-Native Rehosting
- The SAFIREFUZZ Engine
- Performance
- Outlook & Conclusion

Overview

- Intro to Embedded Firmware Fuzzing
- Overview of Rehosting Approaches and Systems
- Performance Roadblocks
- Near-Native Rehosting
- The SAFIREFUZZ Engine
- Performance
- Outlook & Conclusion

USENIX Paper: *Forming Faster Firmware Fuzzers* [1] w/ Dominik Maier and Marius Muench

- Framework is open source and results fully reproducible!

```
    X    X    X    X    X
   / \  / \  / \  / \  / \
  +-----------------------+
 /| +-------------------+ |\
| | |     Firmware      | | |
 \| |                   | |/
  | +-------------------+ |
 /|                       |\
| |          CPU          | |
 \|                       |/
  |                       |
 /| +-------------------+ |\        +--------------------------+
| | |       MMIO        | | |       |   External Peripherals   |
 \| +---------^---------+ |/        |   (e.g., WiFi antenna)   |
  +-----------+-----------+         +--------------------------+
   \_/  \_/  \|/  \_/  \_/                        ^
    V    V    |    V    V                         |
              |                                   |
              |                                   |
              +-----------------------------------+
```

```
+-------------------------------------------------------------------------------------------------------+
|              +--------------------------------------------------------------------------+             |
|              |                                                                          |             |
|              |                                           Emulator                       |             |
|              |       X    X    X    X    X                                              |             |
|              |      / \  / \  / \  / \  / \                                 +-----------+-----------+ |
|              |     +-----------------------+                                |                       | |
|              |    /| +-------------------+ |\                               |                       | |
|              |   | | |     Firmware      | | |                     +--------|    Fuzzing Engine     | |
|              |    \| |                   | |/                      |        |                       | |
|              |     | +-------------------+ |                       |        |                       | |
|              |    /|                       |\                      |        +-----------+-----------+ |
|              |   | |          CPU          | |                     |                    |             |
|              |    \|                       |/                      |                    |             |
|              |     |                       |                       v                    |             |
|              |    /| +-------------------+ |\        +--------------------------+       |             |
|              |   | | |       MMIO        | | |       |   External Peripherals   |       |             |
|              |    \| +---------^---------+ |/        |   (e.g., WiFi antenna)   |       |             |
|              |     +-----------+-----------+         +--------------------------+       |             |
|              |      \_/  \_/  \|/  \_/  \_/                        ^                    |             |
|              |       V    V    |    V    V                         |                    |             |
|              |                 |                                   |                    |             |
|              |                 |                                   |                    |             |
|              |                 +-----------------------------------+                    |             |
|              |                                                                          |             |
|              +--------------------------------------------------------------------------+             |
|                                                                                                       |
| Userspace                                                                                             |
+-------------------------------------------------------------------------------------------------------+
```

Rehosting Approaches Overview

1. Option: Peripheral Modeling

Hardware Record-based

- Obtain actual hardware that runs the firmware
- Record hardware feedback (MMIO interactions)
- Replay MMIO values/patterns during emulation
- OR: train ML model on recorded inputs to have a more diverse output space

1. Option: Peripheral Modeling

Hardware Record-based

- Obtain actual hardware that runs the firmware
- Record hardware feedback (MMIO interactions)
- Replay MMIO values/patterns during emulation
- OR: train ML model on recorded inputs to have a more diverse output space

Symbolic Constraints-based

- Handle unknown peripheral reads by returning symbolic values and exploring program paths
- SotA (*Fuzzware* [3]):
  - Program analysis to spot partial uses of hardware-generated values
  - Local dynamic symbolic execution (only execute code in context of specific MMIO access)

2. Option: High-Level Emulation

Catch MMIO accesses before they occur:

- Hook functions that handle MMIO / file system accesses
    - Typically at the Hardware Abstraction Layer (HAL)
- Emulate behavior in high-level language
- Supply fuzzing input via these hooks

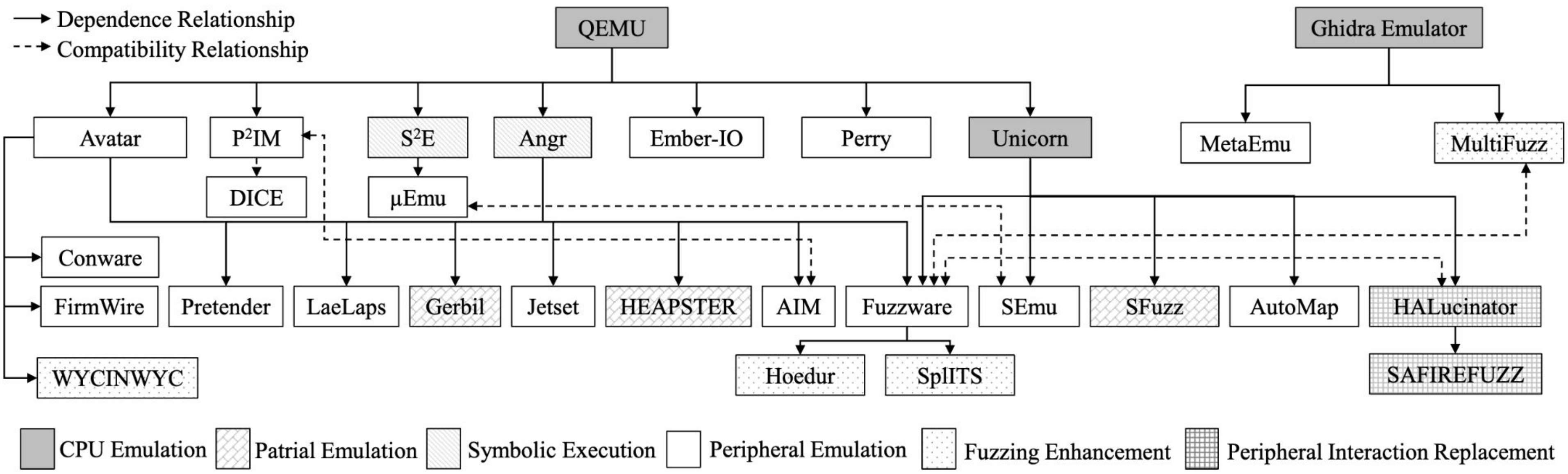
```rust
/// Return fake FatFs FILE object
pub unsafe fn f_open(file_ptr: u32, _path_ptr: u32, _mode_byte: u32) -> u32 {
    let buf_ptr: u32 = crate::handlers::malloc(size: FUZZ_LEN);

    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
```

Taxonomy of Firmware Rehosting Systems

Source: [4]



Performance Roadblocks in Emulation

[R1] Binary Lifting

- QEMU: guest code -> TinyCode -> Instrumentation -> JIT compilation -> host code

Performance Roadblocks in Emulation

[R1] Binary Lifting

- QEMU: guest code -> TinyCode -> Instrumentation -> JIT compilation -> host code

[R2] Basic Block Caching

- Multiple rehosting systems developed on top of of AFL-QEMU:
  No caching and execution chaining

Performance Roadblocks in Emulation

[R1] Binary Lifting

- QEMU: guest code -> TinyCode -> Instrumentation -> JIT compilation -> host code

[R2] Basic Block Caching

- Multiple rehosting systems developed on top of of AFL-QEMU:
  No caching and execution chaining

[R3] Dispatch of Memory Accesses

- Full-system emulation developed for general purpose / complex systems
- SoftMMU dispatches memory accesses
- In many embedded systems, we do not need / want to emulate an MMU

Performance Roadblocks in Emulation

[R1] Binary Lifting

- QEMU: guest code -> TinyCode -> Instrumentation -> JIT compilation -> host code

[R2] Basic Block Caching

- Multiple rehosting systems developed on top of of AFL-QEMU:
  No caching and execution chaining

[R3] Dispatch of Memory Accesses

- Full-system emulation developed for general purpose / complex systems
- SoftMMU dispatches memory accesses
- In many embedded systems, we do not need / want to emulate an MMU

[R4] Fuzzer in Separate Process

- No in-process communication: high overhead due to context switches

Intuition:

A lot of embedded firmware runs on ARMv7-M chips

Certain ARMv8-A cores provide compatibility with AArch32 and Thumb instruction set variants

Near-Native Rehosting

- Execute binaries for small embedded devices on their "bigger brothers"
- Heavily reduce the amount of code which needs lifting / rewriting

- Outperform rehosting approaches built on top of general-purpose emulators!

SAFIREFUZZ

```
+--------------------------------------------------------------------------------------------------------+
|                                                                                                        |
|            +--------------------------------------------------------------------------------------+    |
|            |   Userspace                                                                          |    |
|            |                                                                                      |    |
|            |   +--------------+    +-------------------+                   +-------------------+  |    |
|            |   |    LibAFL    |    |   Basic  Blocks   | Dynamic           |   Basic  Blocks   |  |    |
|            |   +--------------+    |   .---.   .---.   | Rewriting         |   .---.   .---.   |  |    |
|            |                       |  (     ) (     )<-+-------------------+--(     ) (     )  |  |    |
|            |                       |   `---'   `---'   | & Instrumentation |   `---'   `---'   |  |    |
|            |                       |   .---.   .---.   |                   |   .---.   .---.   |  |    |
|            |   +-------------------+------+ ) (     )  |                   |  (     ) (     )  |  |    |
|            |   |                          +'   `---'   |                   |   `---'   `---'   |  |    |
|            |   |  HAL Layer Abstractions  |            |                   |                   |  |    |
|            |   |                          |            |                   |                   |  |    |
|            |   +-------------------+------+            |                   |                   |  |    |
|            |                       |                   |                   | Firmware Binary   |  |    |
|            |                       | Rehosted Binary   |                   | (Target)          |  |    |
|            |                       +-------------------+                   +-------------------+  |    |
|            |                                                                                      |    |
|            |                                                                                      |    |
|            +--------------------------------------------------------------------------------------+    |
|                                                                                                        |
| SAFIREFUZZ                                                                                             |
+--------------------------------------------------------------------------------------------------------+
```

Design

- Basic Block Rewriting
  - Runtime instrumentation
  - Majority of all instructions require no rewriting at all, only:
    - PC-modifying instructions
    - PC-relative memory accesses

- Function Hooking
  - On block rewrite, check if hook is registered at address
  - Emits jump to user-supplied code, executes at block execution time
  - Switch to engine: 5 instructions

- Interrupt Approximation
  - Observation: interrupts commonly triggered by peripherals
    - MMIO access by ISR needs a hook anyway
  - Tick-based approx. by call-level counter and clock-update hook

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
```

Challenges

- PC-relative memory accesses
  - Data chunks interleaved with instructions
    - Finding bounds is a hard problem
  - Resolve absolute address of memory access and patch instruction with static load

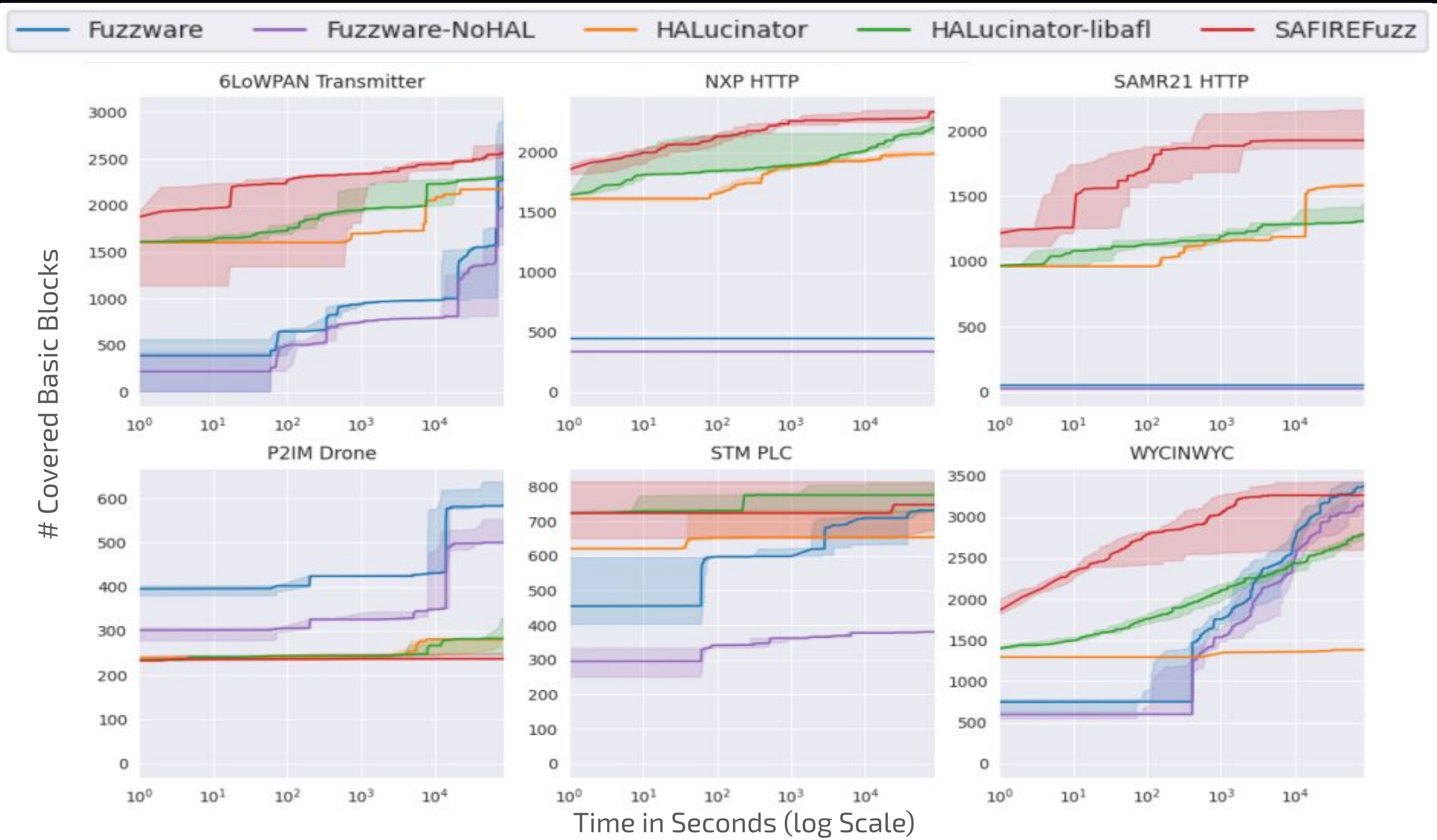
Challenges

- PC-relative memory accesses
  - Data chunks interleaved with instructions
    - Finding bounds is a hard problem
  - Resolve absolute address of memory access and patch instruction with static load

- Processor Cache Maintenance
  - ARMv7-A cores have separate caches for instructions and data
    - Non-coherent
  - (non-deterministically) leads to inconsistencies with self-modifying code
    - You can overwrite instructions in memory at runtime, but you have to tell your CPU!
  - After rewrite: invalidate cache for corresponding range

Performance Evaluation

Coverage over Time



binarly

Execution Speeds

Evaluation:

- 12 targets
- 5 runs each
- 24 hours per run

binarly

Execution Speeds

Evaluation:

- 12 targets
- 5 runs each
- 24 hours per run

On average:

- 690x faster than HALucinator [5]
- 145x faster than Fuzzware

Outlook

Potential Drawbacks:

- Manual Effort
  - Identify MMIO accesses / functions at HAL
  - Implement realistic handlers

- Fidelity
  - Interrupt approximation covers most code paths but is not ‘realistic’
  - Usually a trade-off against automation

Outlook

Potential Drawbacks:

- Manual Effort
  - Identify MMIO accesses / functions at HAL
  - Implement realistic handlers
- Fidelity
  - Interrupt approximation covers most code paths but is not ‘realistic’
  - Usually a trade-off against automation

Future Work:

- Better MMIO Model instead of HLE
  - Improvements on top of Fuzzware allow for more detailed MMIO reg <-> fuzz input mappings [6]
  - Better automation
- Snapshotting
  - Faster resets & stateful fuzzing

binarly

Conclusion

- Optimize your emulator and fuzzer setup for your target!

- Near-native execution for ARMv7-M embedded firmware
    - Rehosting in Linux Userspace
    - same idea, other authors: *SURGEON* [2]

- Vastly increased execution speeds
  ⇒ Less time to achieve more coverage!

GITHUB

@pr0me
@binarly_io

binarly

Sources

[1] Seidel, L., Maier, D.C. and Muench, M., 2023, August. Forming Faster Firmware Fuzzers. In USENIX Security Symposium (pp. 2903-2920).

[2] Hofhammer, F., Busch, M., Wang, Q., Egele, M. and Payer, M., 2024, March. SURGEON: Performant, Flexible, and Accurate Re-Hosting via Transplantation. Workshop on Binary Analysis Research (BAR'24).

[3] Scharnowski, T., Bars, N., Schloegel, M., Gustafson, E., Muench, M., Vigna, G., Kruegel, C., Holz, T. and Abbasi, A., 2022. Fuzzware: Using precise {MMIO} modeling for effective firmware fuzzing. In 31st USENIX Security Symposium (USENIX Security 22) (pp. 1239-1256).

[4] Zhou, W., Shen, S. and Liu, P., 2025. IoT Firmware Emulation and Its Security Application in Fuzzing: A Critical Revisit. Future Internet, 17(1), p.19.

[5] Clements, A.A., Gustafson, E., Scharnowski, T., Grosen, P., Fritz, D., Kruegel, C., Vigna, G., Bagchi, S. and Payer, M., 2020. HALucinator: Firmware re-hosting through abstraction layer emulation. In 29th USENIX Security Symposium (USENIX Security 20) (pp. 1201-1218).

[6] Scharnowski, T., Wörner, S., Buchmann, F., Bars, N., Schloegel, M. and Holz, T., 2023. Hoedur: Embedded Firmware Fuzzing using Multi-Stream Inputs.

Backup Slides

All Tables from our Paper

Firmware	SafireFuzz		HALucinator		HALucinator - libAFL		Fuzzware	
	exec/s	# basic blocks	exec/s	# basic blocks	exec/s	# basic blocks	exec/s	# basic blocks
6LoWPAN Receiver	581.4	2840	1.2	2354	2.5	2724	73.6	1812 / 1618
6LoWPAN Transmitter	1877.0	2563	1.8	2176	2.6	2307	66.4	2460 / 2101
NXP HTTP	5216.8	2341	4.8	1990	4.5	2209	22.5	447 / 337
SAMR21 HTTP	2894.6	1927	3.1	1581	1.7	1310	1018.4	52 / 26
P2IM PLC	772.1	202	19.5	228	6.3	249	24.5	87 / 70
P2IM Drone	7279.7	237	9.3	281	2.8	283	9.7	583 / 500
STM PLC	7193.8	748	10.8	654	2.0	776	15.5	732 / 381
WYCINWYC	3083.1	3263	9.4	1384	12.3	2795	41.0	3375 / 3166
TCP Echo Client	3401.3	2403	4.8	1679	4.0	2290	87.2	460 / 375
TCP Echo Server	2762.1	2177	5.0	1563	4.7	1710	88.4	459 / 229
UDP Echo Client	4485.3	1613	5.0	1188	4.7	1594	90.2	460 / 229
UDP Echo Server	4636.7	1450	5.9	1045	5.1	1485	85.1	460 / 229

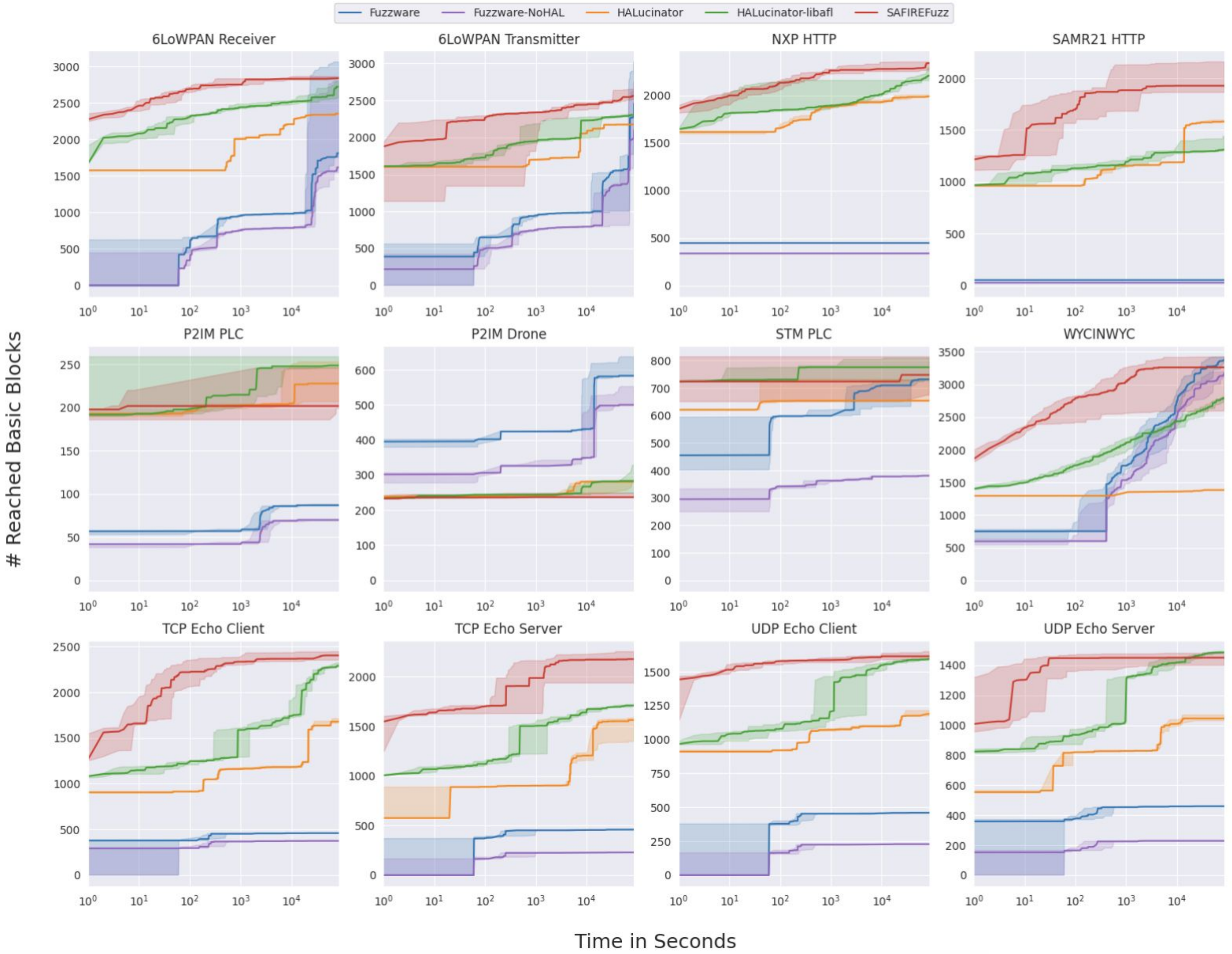
Table 3: Results of fuzzing the targets over 24 hours. Reported numbers are median values from the five runs. For Fuzzware, we report reached basic blocks both with and without considering HAL functions.

A.1 Comparison to other Papers

Firmware	SafireFuzz			HALucinator - Paper			Para-Rehosting		
	exec/s	*Time*	*Crashes*	*exec/s*	*Time*	*Crashes*	*exec/s*	*Time*	*Crashes*
WYCINWYC	3083.1	24h	16	17.92	24h	5	647.86	11h:43m	909
SAMR21 HTTP	2894.6	24h	2	22.92	19d:04h	273	902.95	12h:33m	219
NXP HTTP	5216.8	24h	0	154.5	14d:0h	0	1443.22	12h:39m	0
6LoWPAN RX	581.4	24h	93	18.84	1d:10h	3	–	–	–
6LoWPAN TX	1877.0	24h	27	15.3	1d:10h	0	–	–	–
P2IM Drone	7279.7	24h	0	11.8	9d:01h	0	–	–	–
P2IM PLC	772.1	24h	14	215	9d:01h	634	–	–	–
ST-PLC	7193.8	24h	325	3.73	1d:10h	27	2552.8	12h:15m	41
STM32 TCP Client	3401.3	24h	0	58.0	3d:08h	0	1092.4	12h:00m	58
STM32 TCP Server	2762.1	24h	0	56.7	3d:08h	0	1466.7	12h:00m	129
STM32 UDP Client	4636.7	24h	0	44.1	3d:08h	0	1245.0	12h:00m	65
STM32 UDP Server	3803.2	24h	0	66.7	3d:08h	0	902.3	12h:00m	16

Table 5: Throughput Comparison with experiments reported in HALucinator [20] and Para-Rehosting [38]. For SafireFuzz, we report values of the median run based on the number of executions. We minimized *Crashes* with AFL's `cmin` for our own experiments, for the other numbers it is not known whether or which minimization the authors applied.

binarly



Fuzzware
Fuzzware-NoHAL
HALucinator
HALucinator-libafl
SAFIREFuzz
6LoWPAN Receiver
6LoWPAN Transmitter
NXP HTTP
SAMR21 HTTP
P2IM PLC
P2IM Drone
STM PLC
WYCINWYC
TCP Echo Client
TCP Echo Server
UDP Echo Client
UDP Echo Server
Reached Basic Blocks
Time in Seconds

Crashes

- All crashes on explored targets reproduced

Firmware	Minimized Crashes
WYCINWYC	16
SAMR21 HTTP	2
6LoWPAN Receiver	93
6LoWPAN Transmitter	27
P2IM PLC	14
STM PLC	325
JPEG Decoder	2
STM32Sine	1

Table 4: Crashes found in targets under test. We minimized crashes with AFL's `cmin`.

- 3 new Bugs in 2 previously unfuzzed targets
  - Sine (OS Inverter):
    - Arbitrary write by corrupted config value (probably not exploitable)
  - Libjpeg Firmware:
    - Segfault after accessing uninitialized struct
    - Out-of-bounds write