UNIVERSITY
OF TWENTE.

Twenty-year-old Vulnerabilities are Back: Firmware Security in the Era of ~~Smart~~ Devices

Andrea Continella
University of Twente

DIMVA’24 - 18/07/2024

Key Takeaways

- Firmware requires **re-thinking** automated security analysis methodologies
- **Significant advances** in firmware analysis. Yet, we often lack **generalizability**
- Vulnerability discovery alone **won’t be enough**

Firmware security before it was cool

FIE (USENIX’13)

firmwa.re (USENIX’14)

Avatar (NDSS’14)

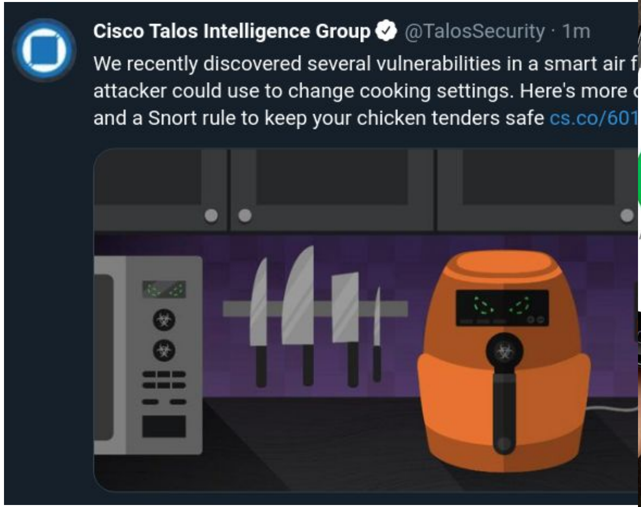
Firmalice (NDSS’15)

Like a bosch

IoT

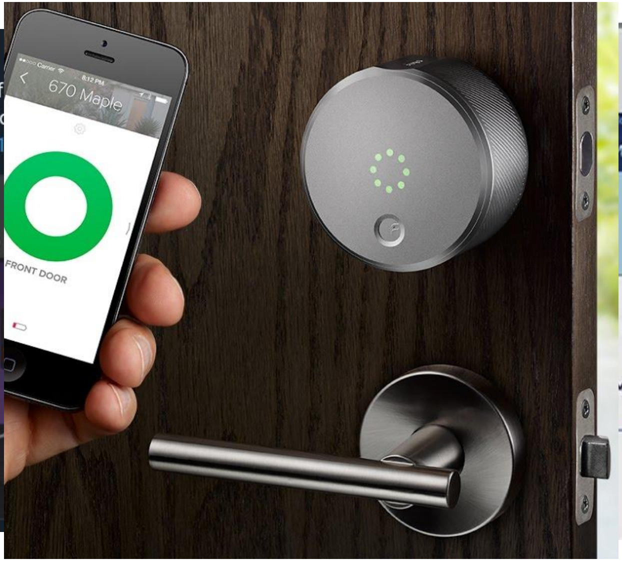
Today's IoT Landscape

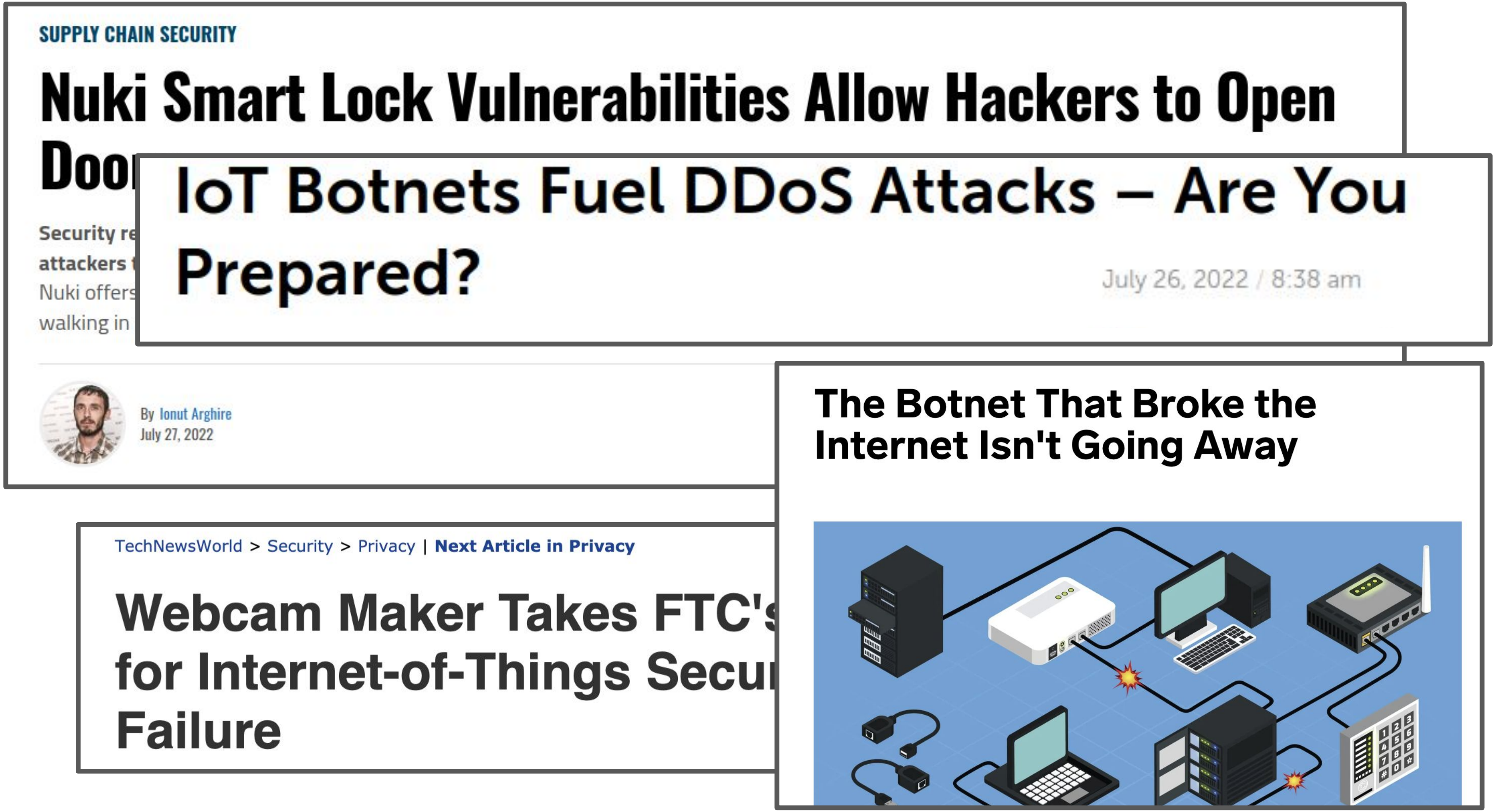
What is the threat model?



Cisco Talos Intelligence Group @TalosSecurity · 1m

We recently discovered several vulnerabilities in a smart air f
attacker could use to change cooking settings. Here's more c
and a Snort rule to keep your chicken tenders safe cs.co/601





SUPPLY CHAIN SECURITY

Nuki Smart Lock Vulnerabilities Allow Hackers to Open Doo

Security re
attackers t
Nuki offers
walking in

By Ionut Arghire
July 27, 2022

IoT Botnets Fuel DDoS Attacks – Are You Prepared?

July 26, 2022 / 8:38 am

The Botnet That Broke the Internet Isn't Going Away

TechNewsWorld > Security > Privacy | **Next Article in Privacy**

Webcam Maker Takes FTC's
for Internet-of-Things Secu
Failure

20y old vulnerabilities are back!

Featuring

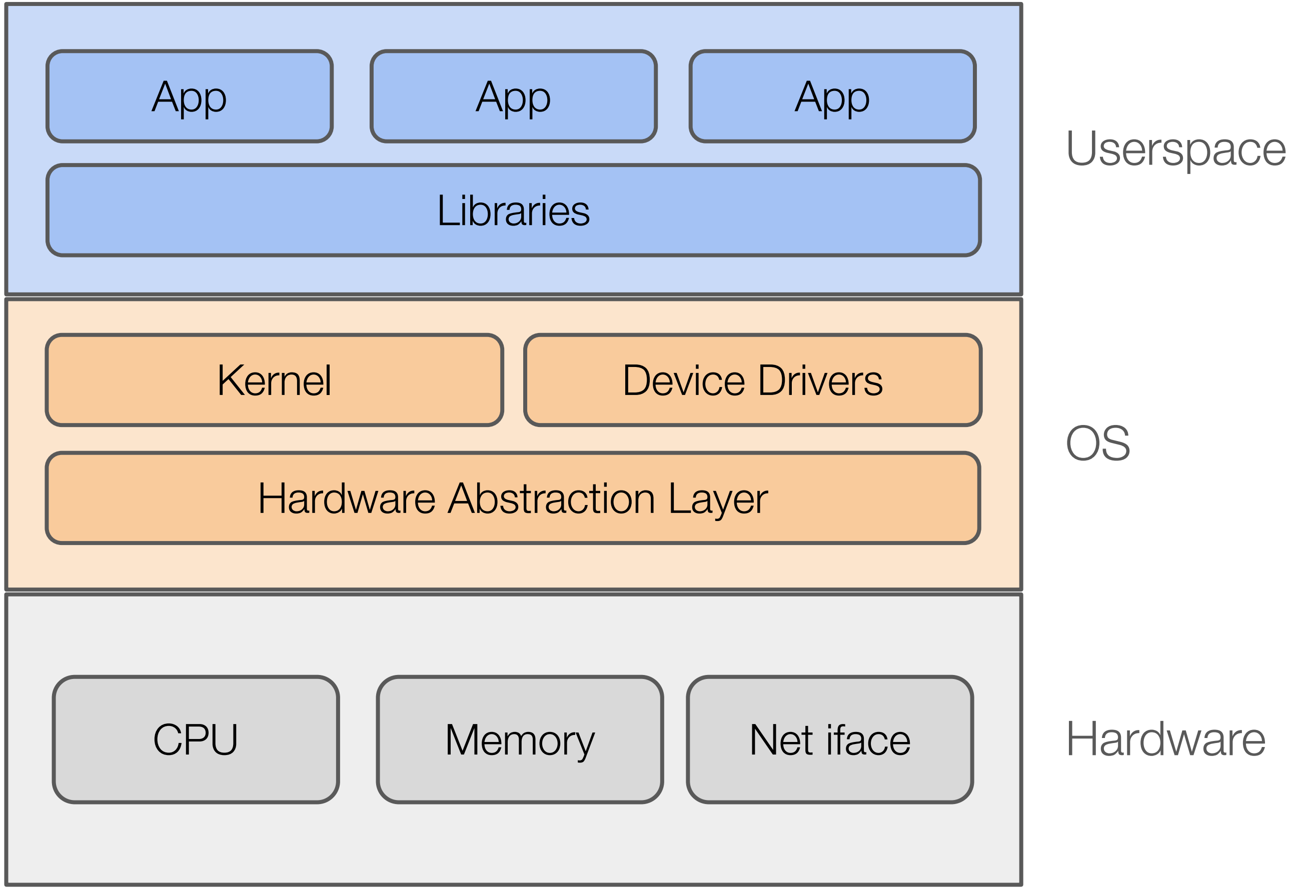
Stack Overflows

Gets()

Scanf()

ASLR who?

Firmware removes many assumptions that software analyses rely on



App

App

App

Libraries

Userspace

Kernel

Device Drivers

Hardware Abstraction Layer

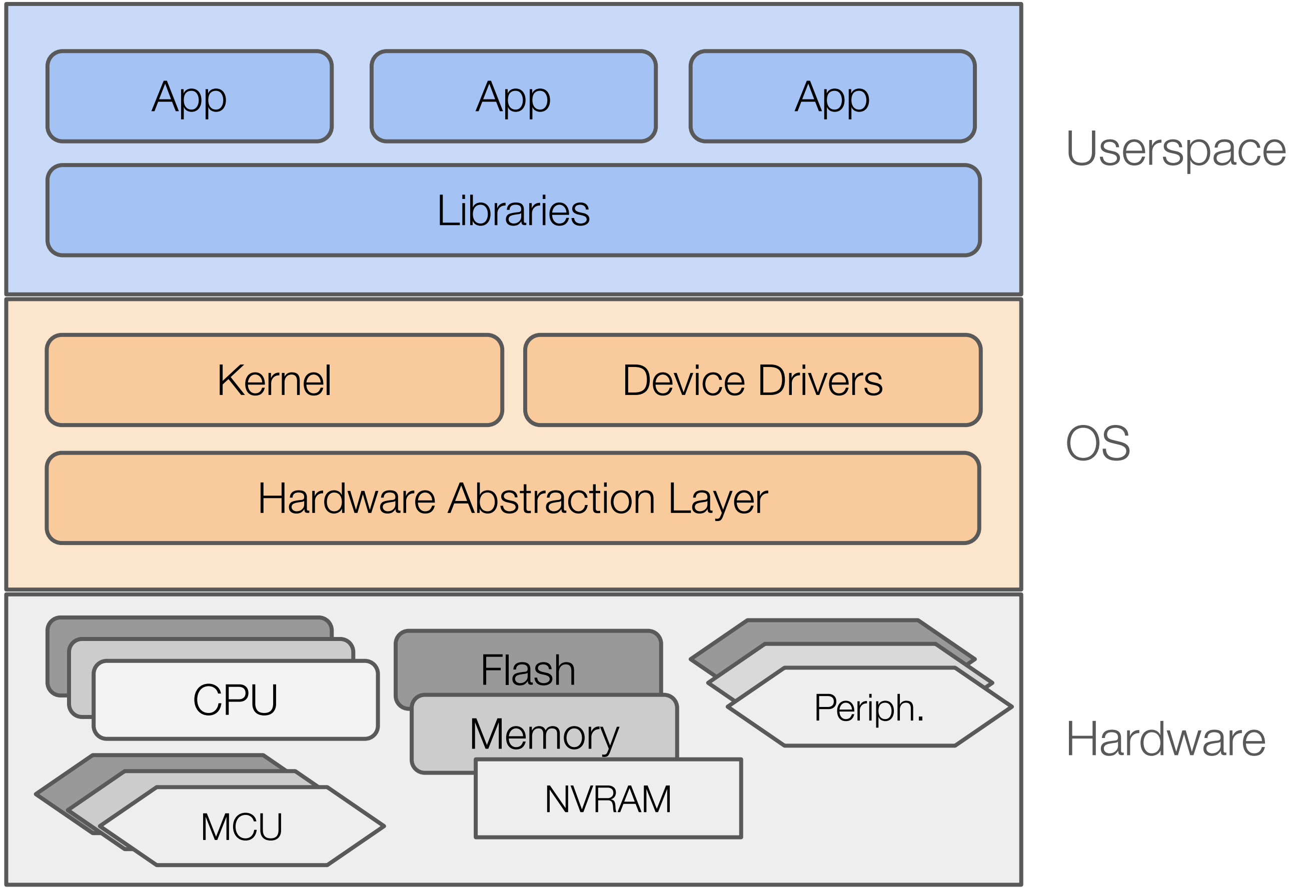
OS

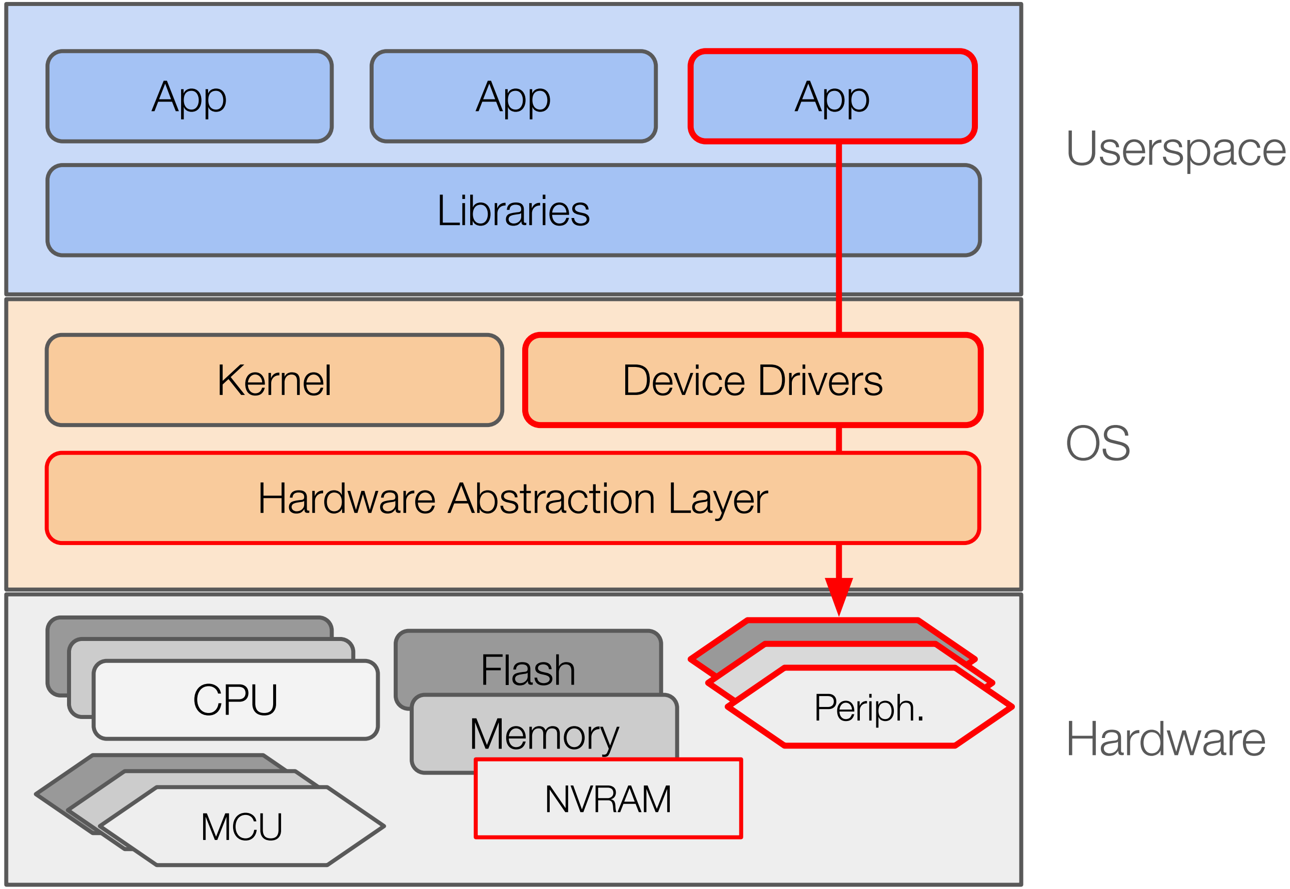
CPU

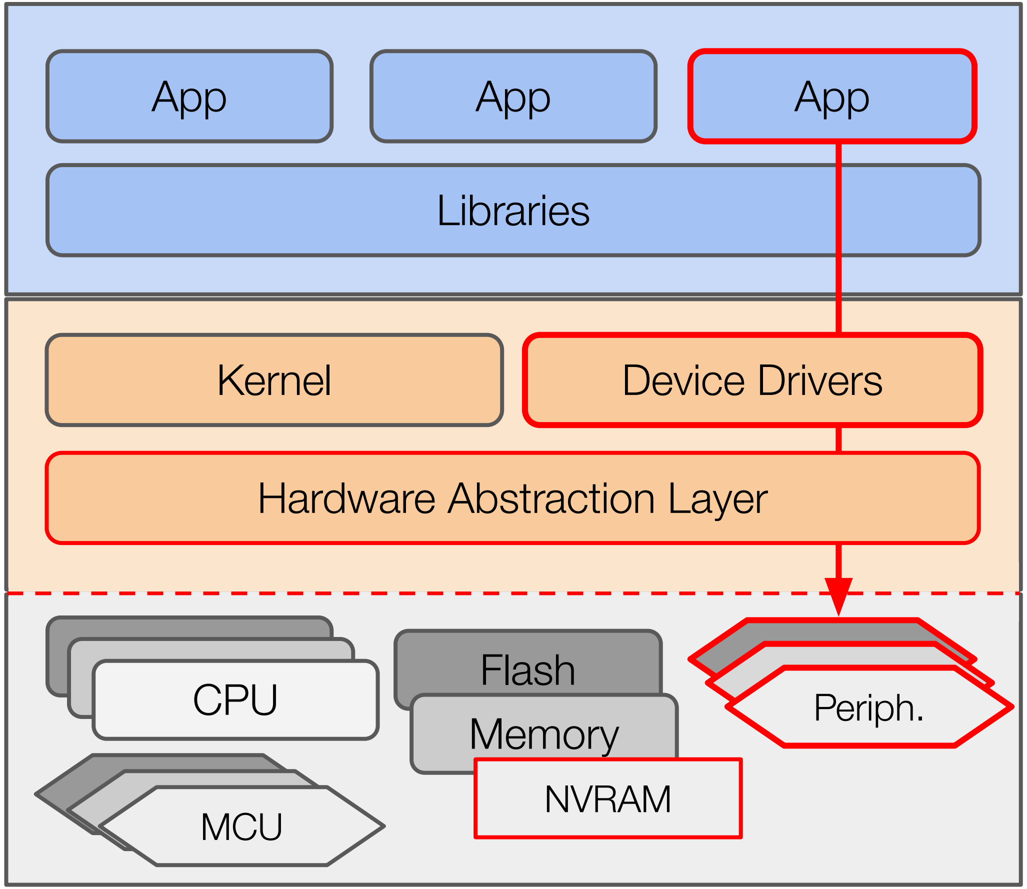
Memory

Net iface

Hardware


App
App
App
Libraries
Userspace
Kernel
Device Drivers
Hardware Abstraction Layer
OS
CPU
Flash
Memory
NVRAM
Periph.
MCU
Hardware


App
App
App
Userspace
Libraries
Kernel
Device Drivers
OS
Hardware Abstraction Layer
CPU
Flash
Memory
Periph.
Hardware
NVRAM
MCU


App

App

App

Libraries

Kernel

Device Drivers

Hardware Abstraction Layer

CPU

Flash

Memory

NVRAM

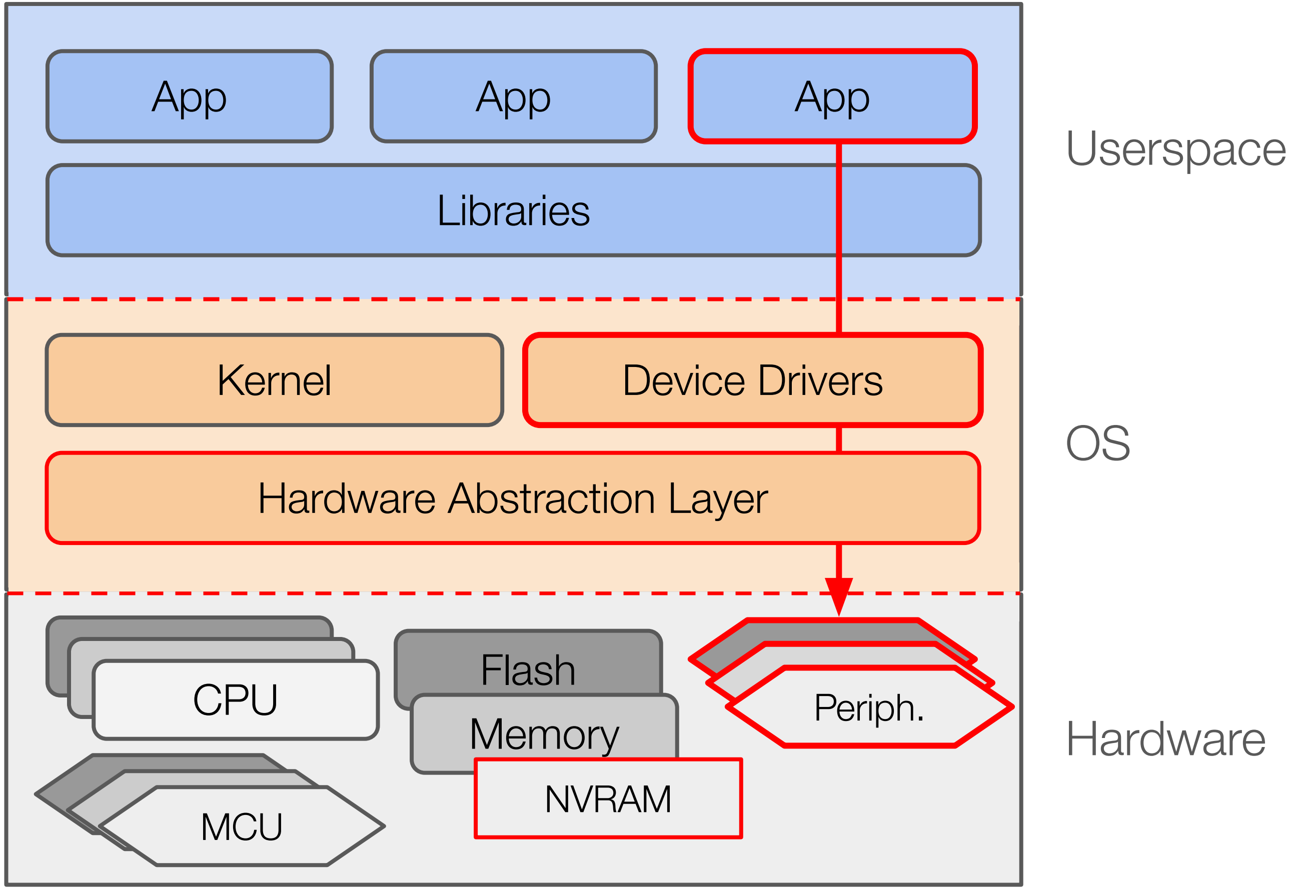
Periph.

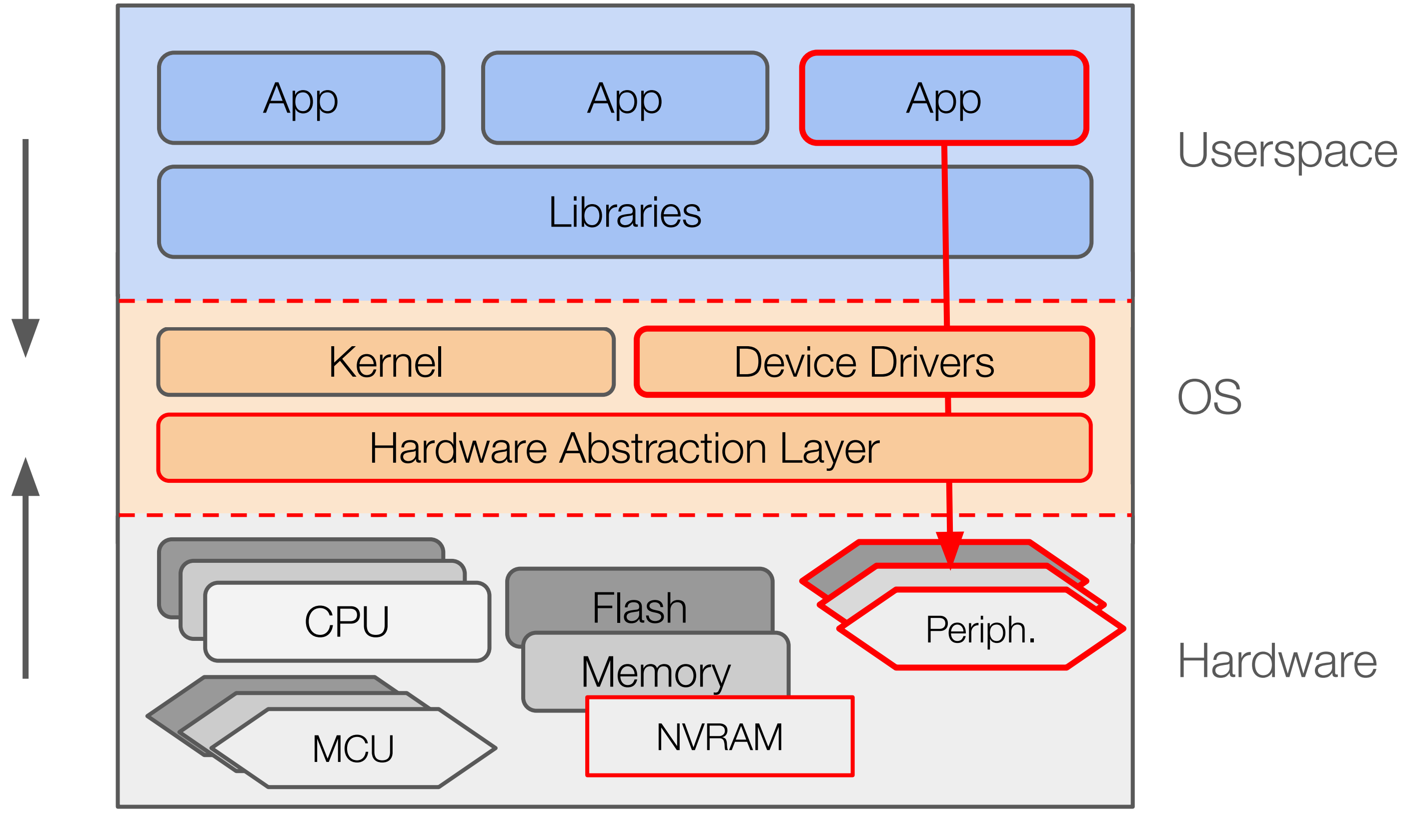
MCU

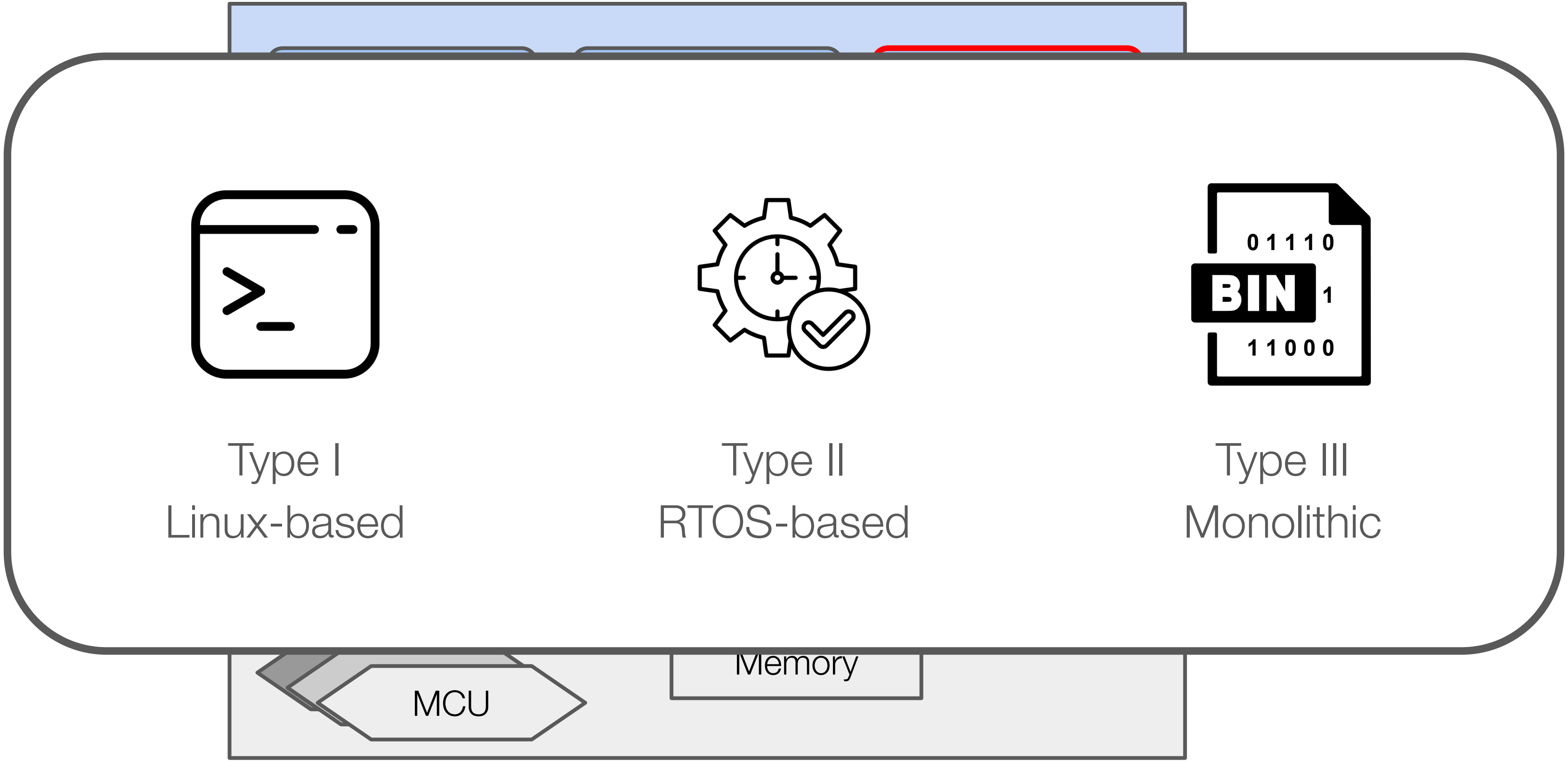
Userspace

OS

Hardware


App
App
App
Userspace
Libraries
Kernel
Device Drivers
OS
Hardware Abstraction Layer
CPU
Flash
Memory
Periph.
Hardware
NVRAM
MCU


App
App
App
Userspace
Libraries
Kernel
Device Drivers
OS
Hardware Abstraction Layer
CPU
Flash
Periph.
Memory
Hardware
MCU
NVRAM



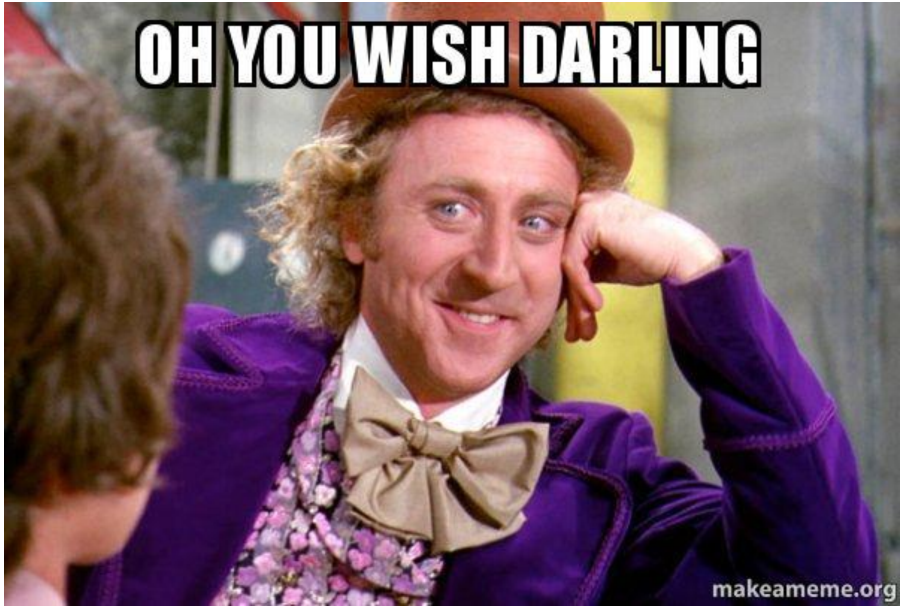
Type I
Linux-based

Type II
RTOS-based

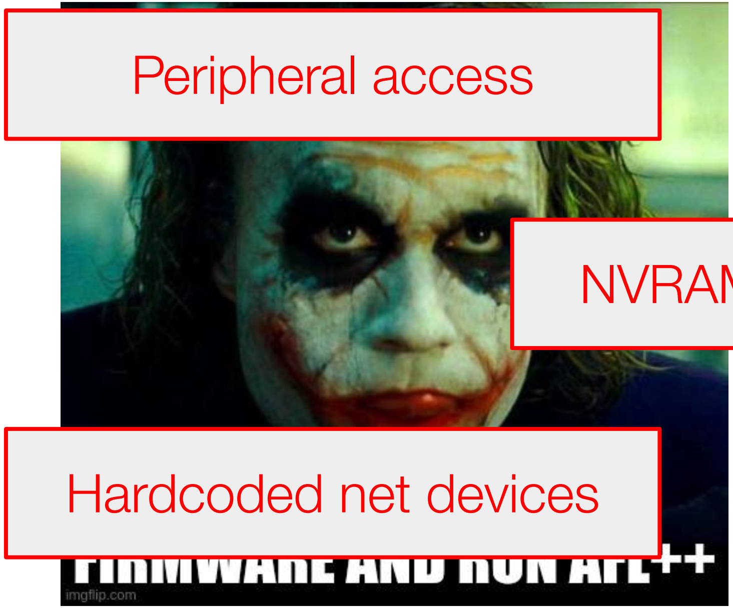
Type III
Monolithic

“What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices” (NDSS’18)

Dynamic Firmware Testing



Dynamic Firmware Testing



Peripheral access

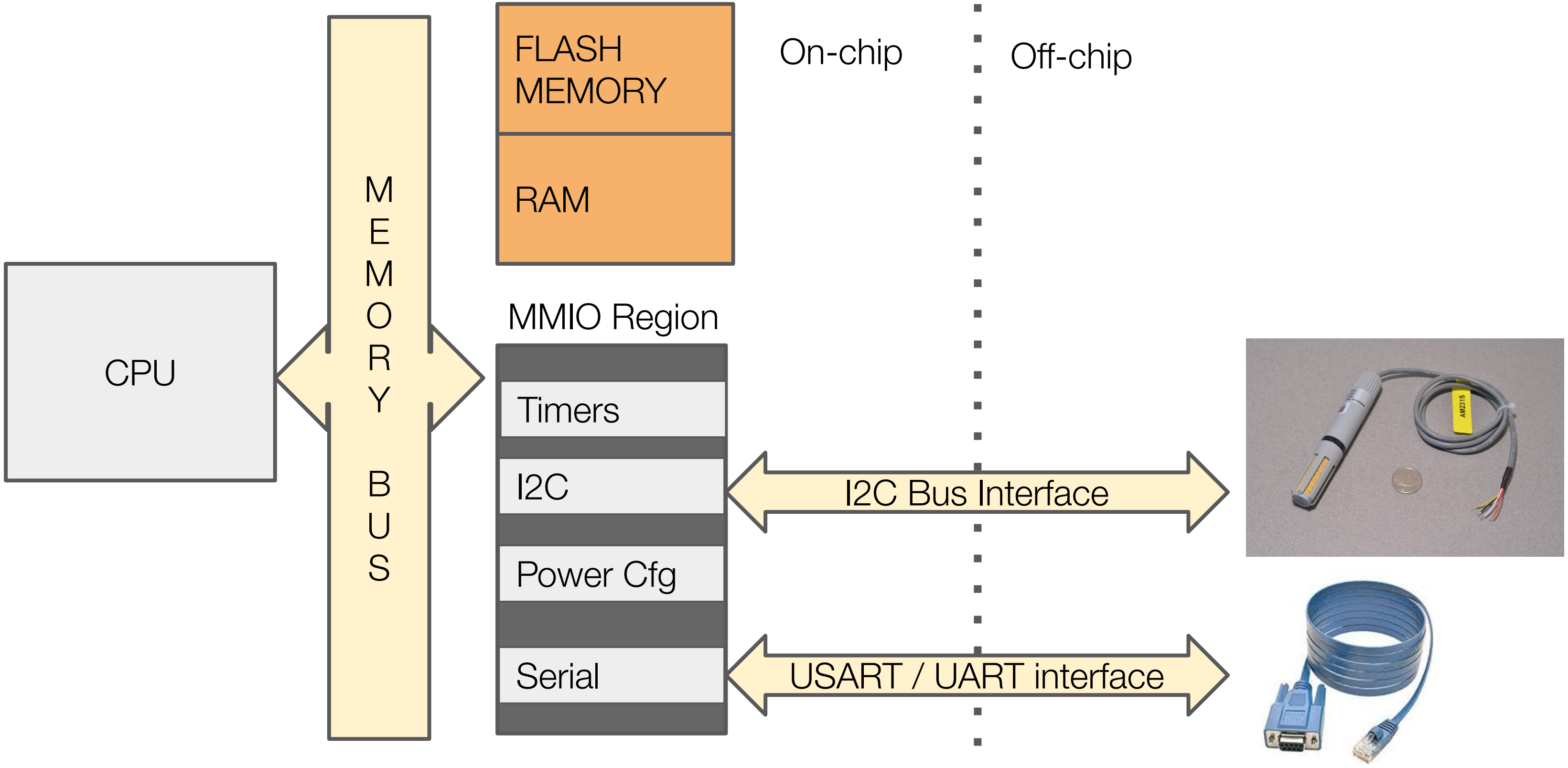
Missing (unpacked) paths

NVRAM configurations

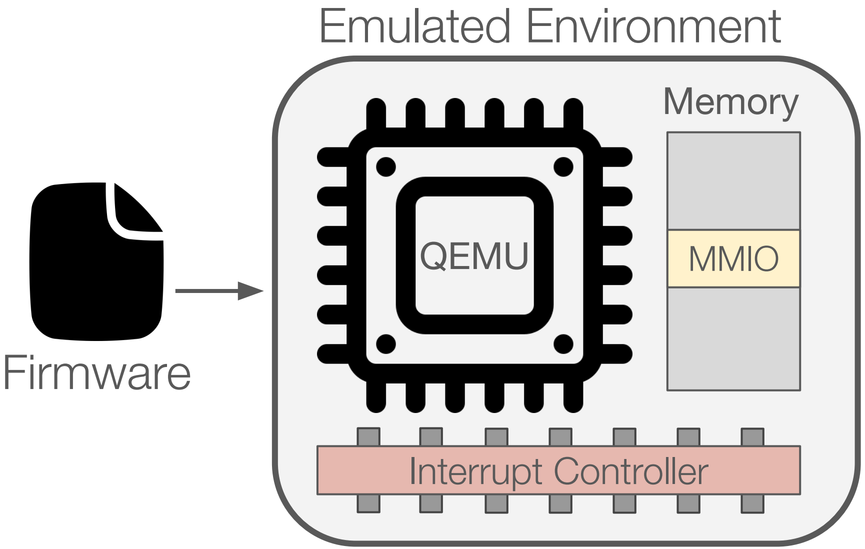
Hardcoded net devices

Environment checks

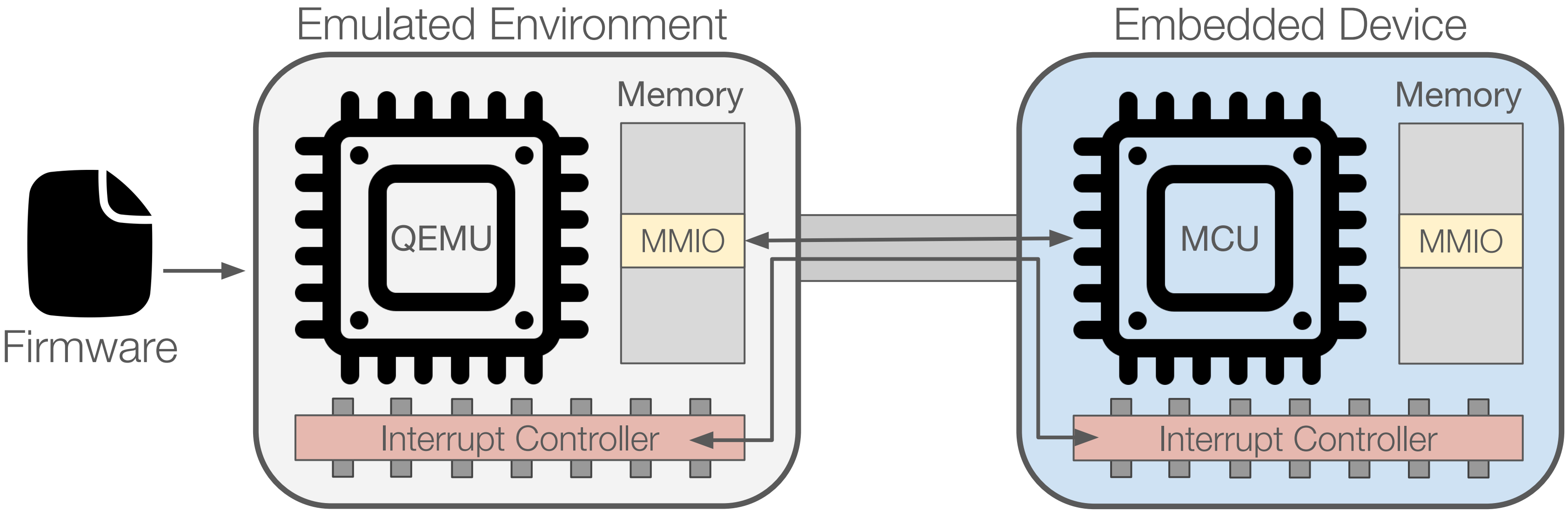
Interacting with Peripherals



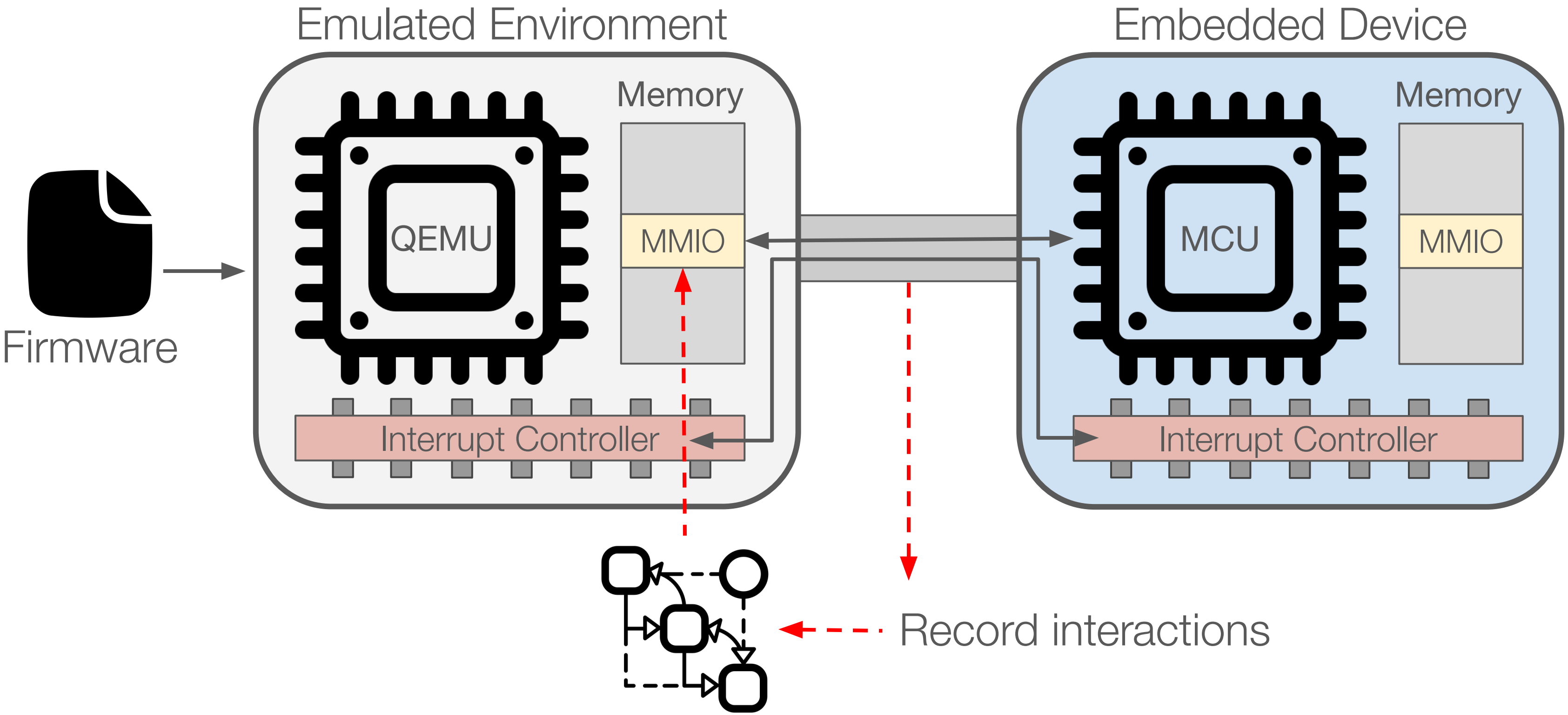
Firmware Rehosting



Firmware Rehosting



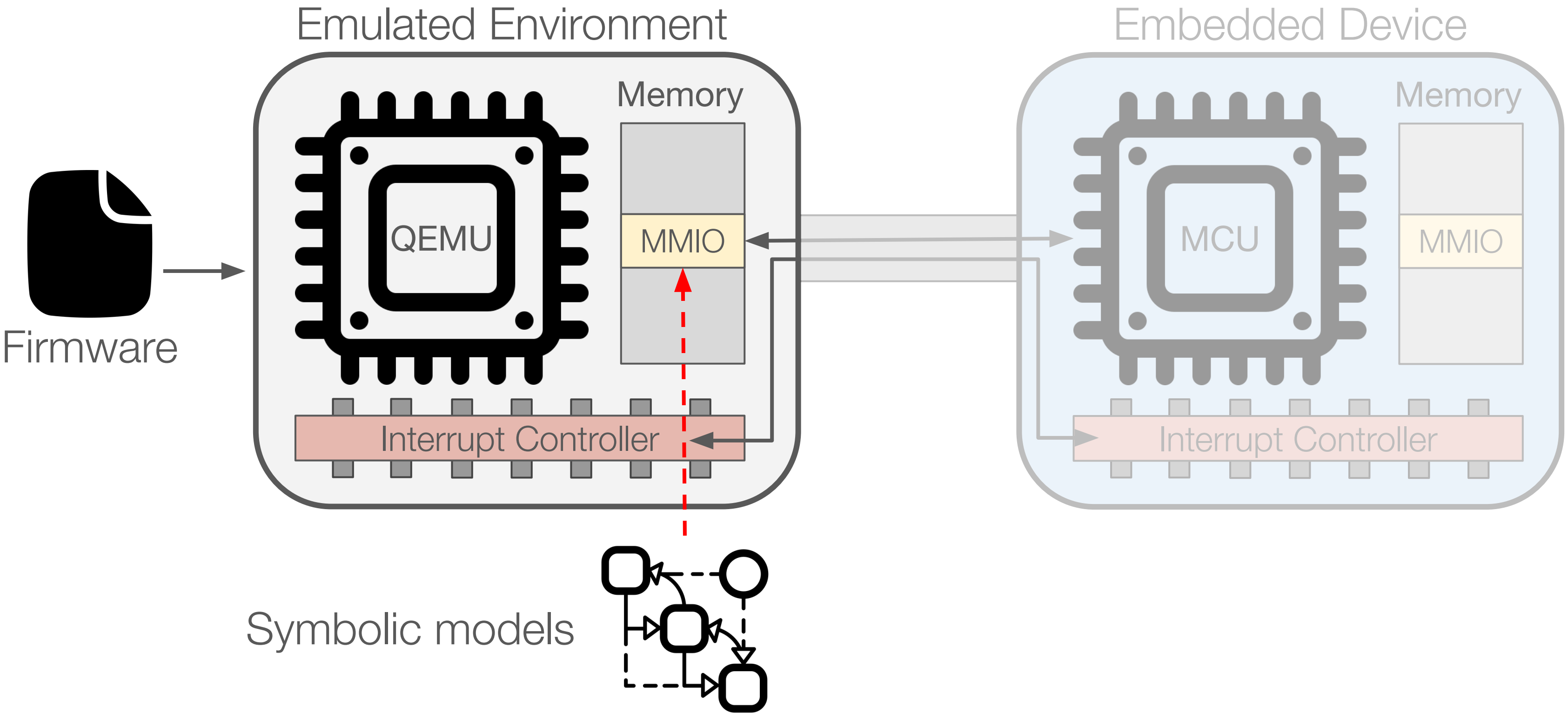
Firmware Rehosting



Pretender (RAID’19)

P2IM (USENIX’20)

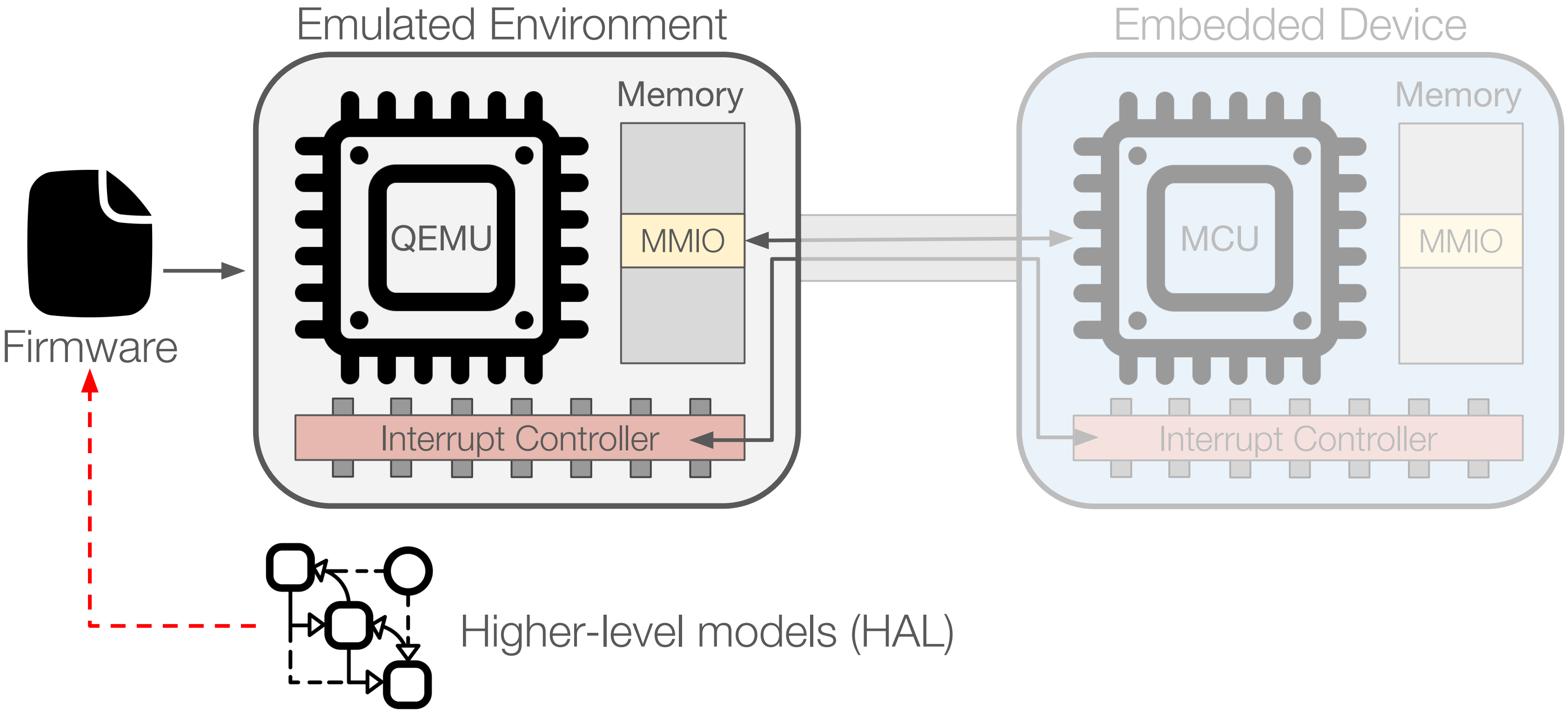
Firmware Rehosting



µEmu (USENIX’21)

Fuzzware (USENIX’22)

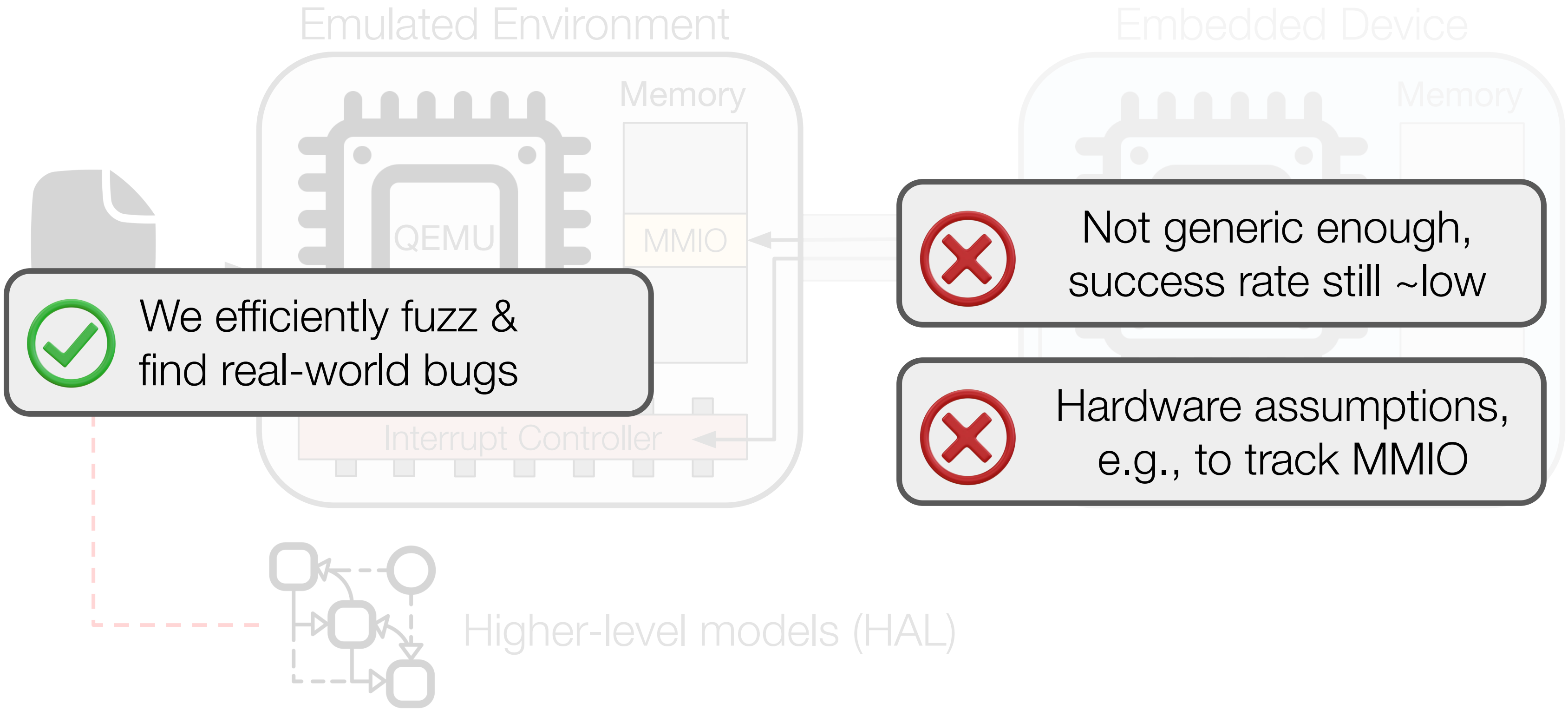
Firmware Rehosting



HALucinator (USENIX'20)

SafireFuzz (USENIX'23)

Firmware Rehosting



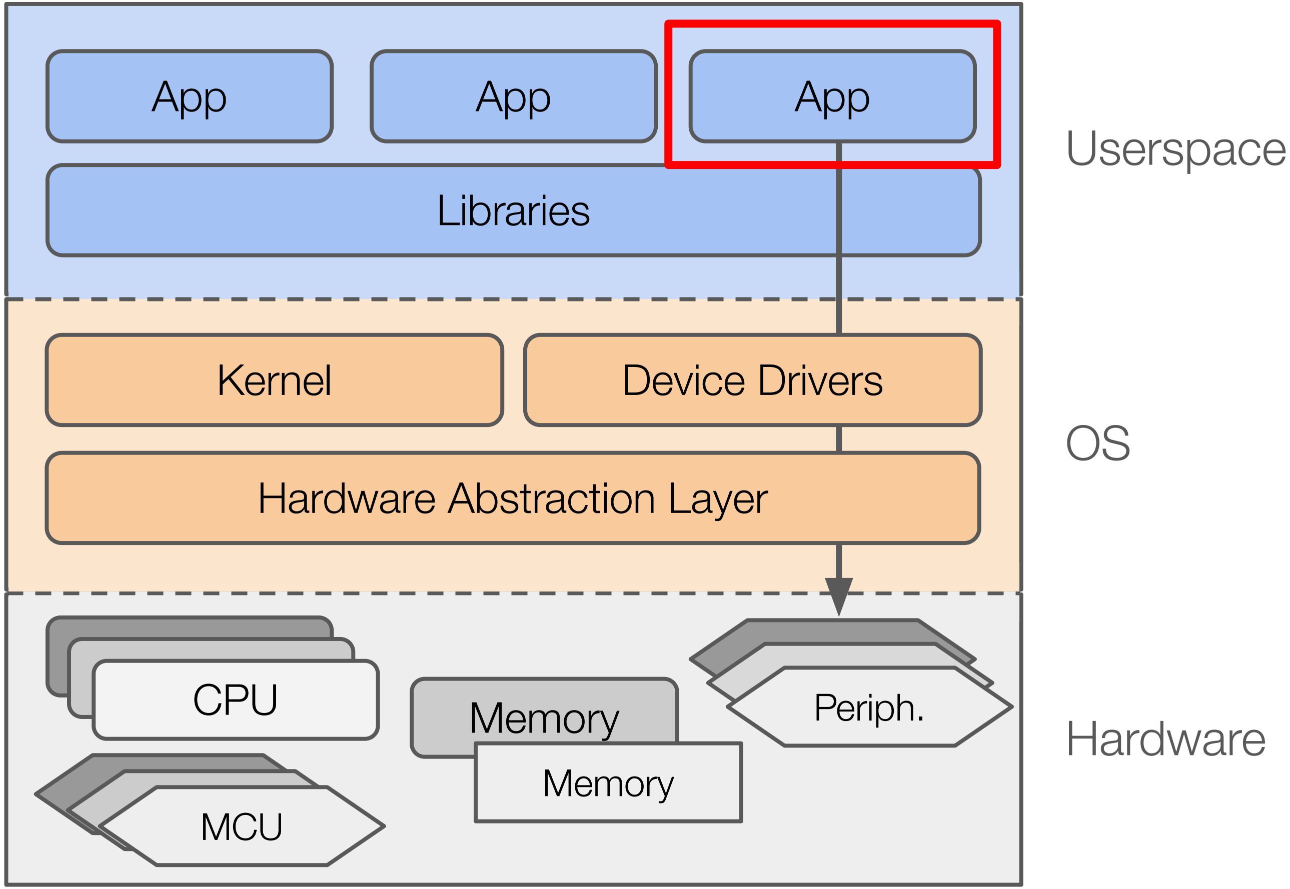
✓ We efficiently fuzz & find real-world bugs

✗ Not generic enough, success rate still ~low

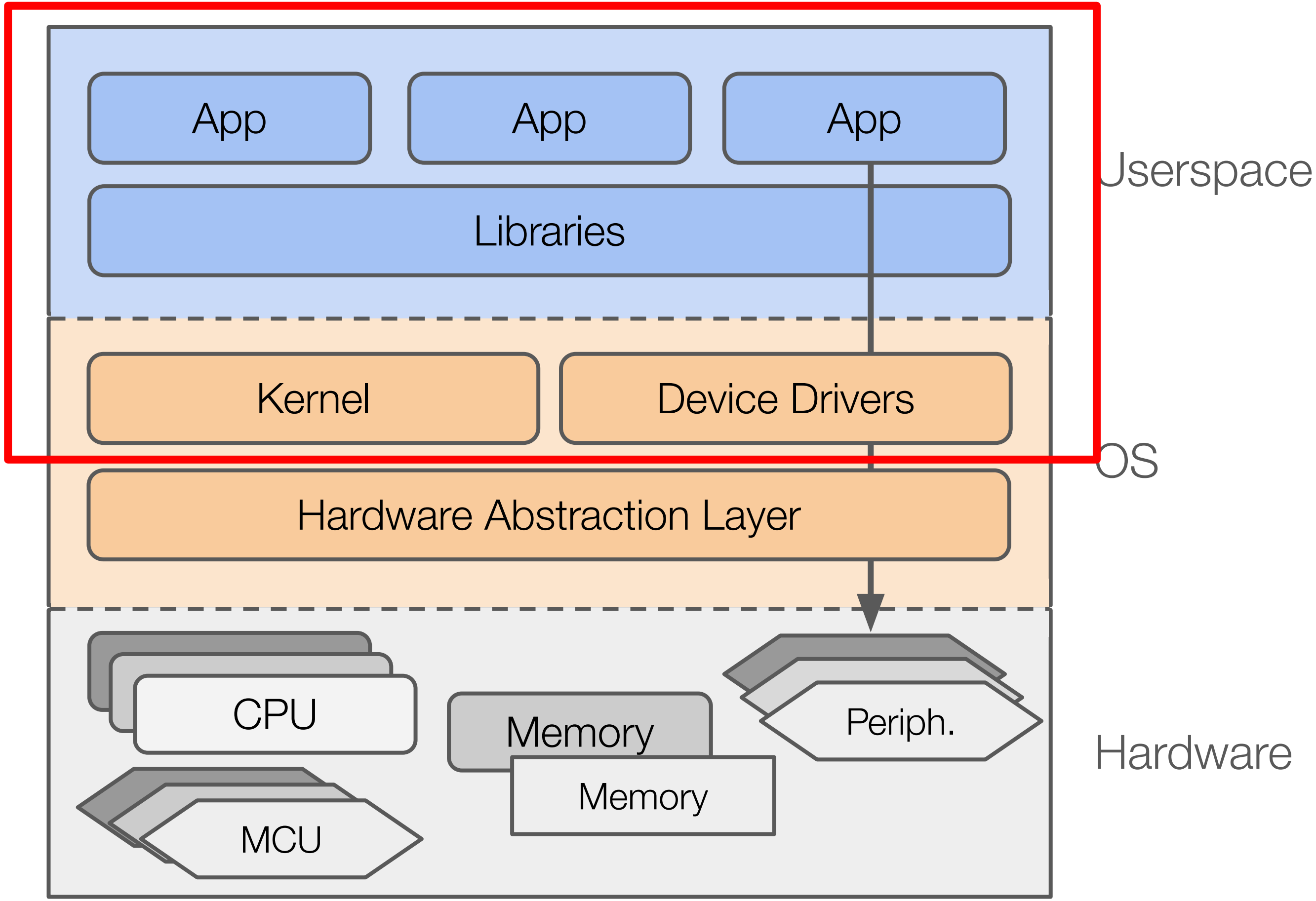
✗ Hardware assumptions, e.g., to track MMIO

HALucinator (USENIX'20)

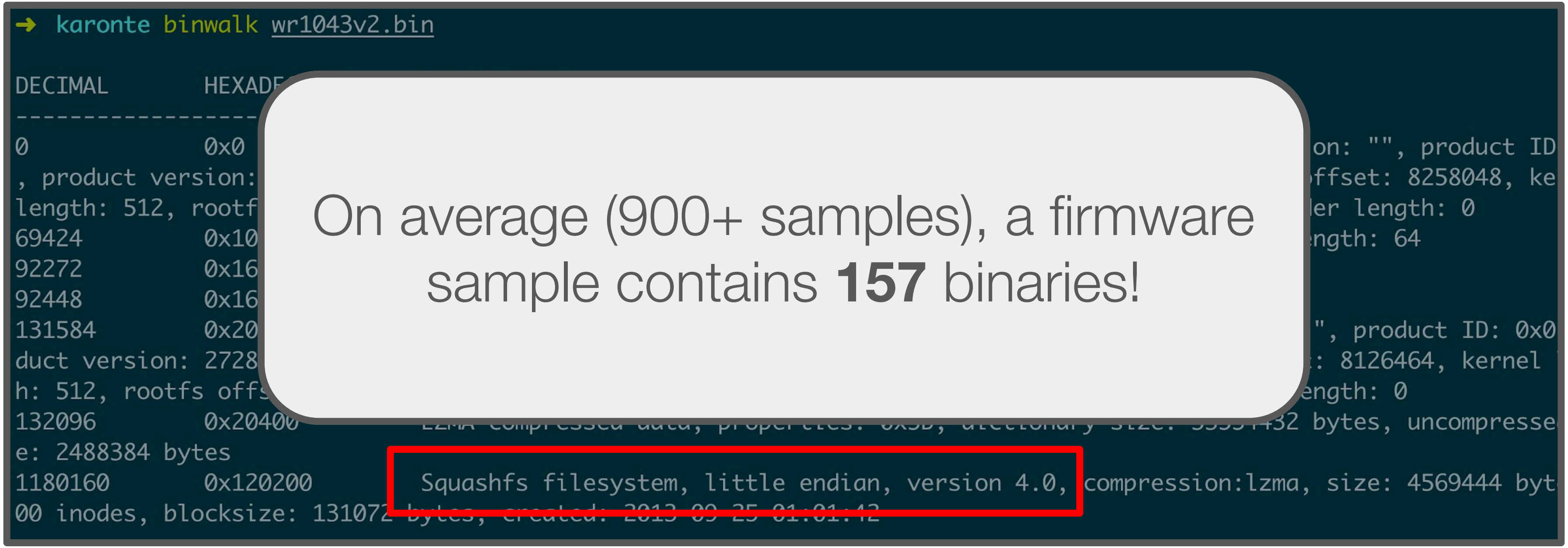
SafireFuzz (USENIX'23)


App
App
App
Userspace
Libraries
Kernel
Device Drivers
OS
Hardware Abstraction Layer
CPU
Memory
Periph.
Hardware
Memory
MCU

Type I
Linux-based

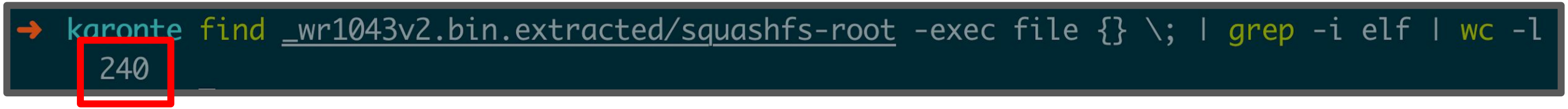


Linux-based Firmware is Multi-binary



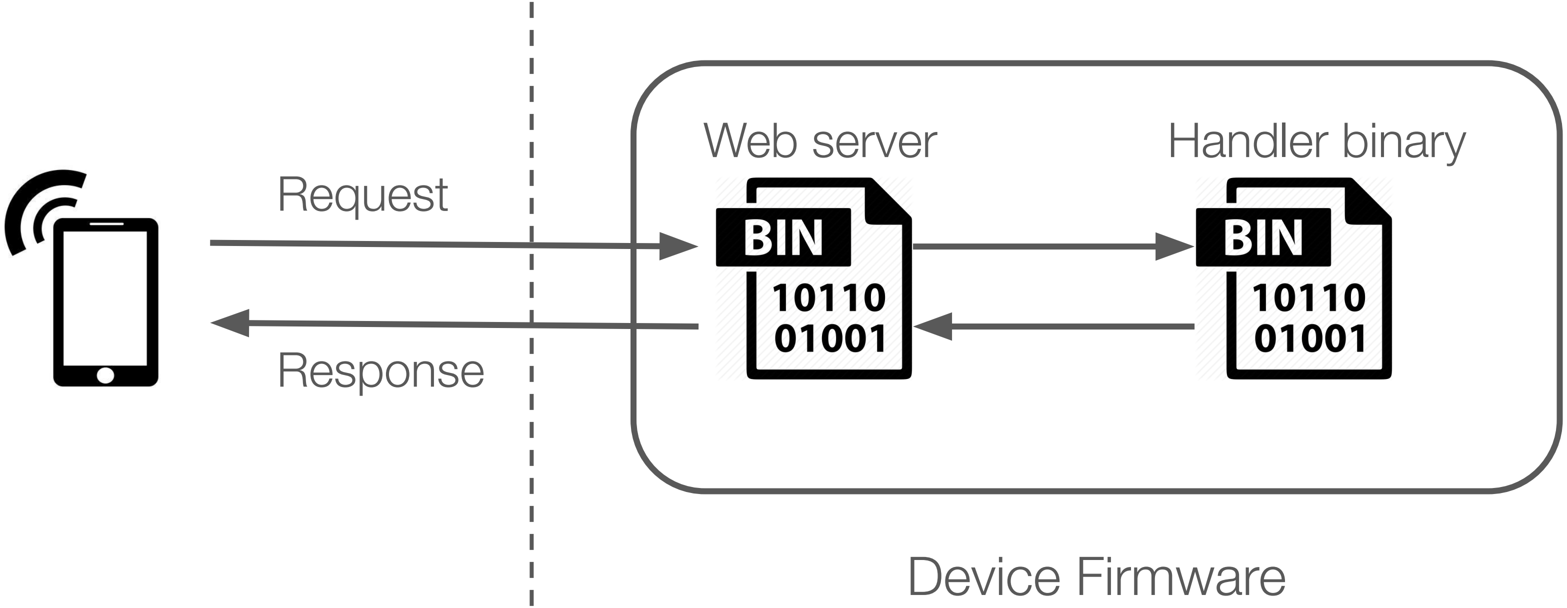
```
➜  karonte binwalk wr1043v2.bin

DECIMAL       HEXADE
--------------------
0             0x0                                          on: "", product ID
, product version:                                        ffset: 8258048, ke
length: 512, rootf                                         ler length: 0
69424         0x10                                         ngth: 64
92272         0x16
92448         0x16
131584        0x20                                         ", product ID: 0x0
duct version: 2728                                        : 8126464, kernel
h: 512, rootfs offs                                       ength: 0
132096        0x20400         LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompresse
e: 2488384 bytes
1180160       0x120200        Squashfs filesystem, little endian, version 4.0, compression:lzma, size: 4569444 byt
00 inodes, blocksize: 131072 bytes, created: 2013-09-25 01:01:42
```

On average (900+ samples), a firmware sample contains **157** binaries!



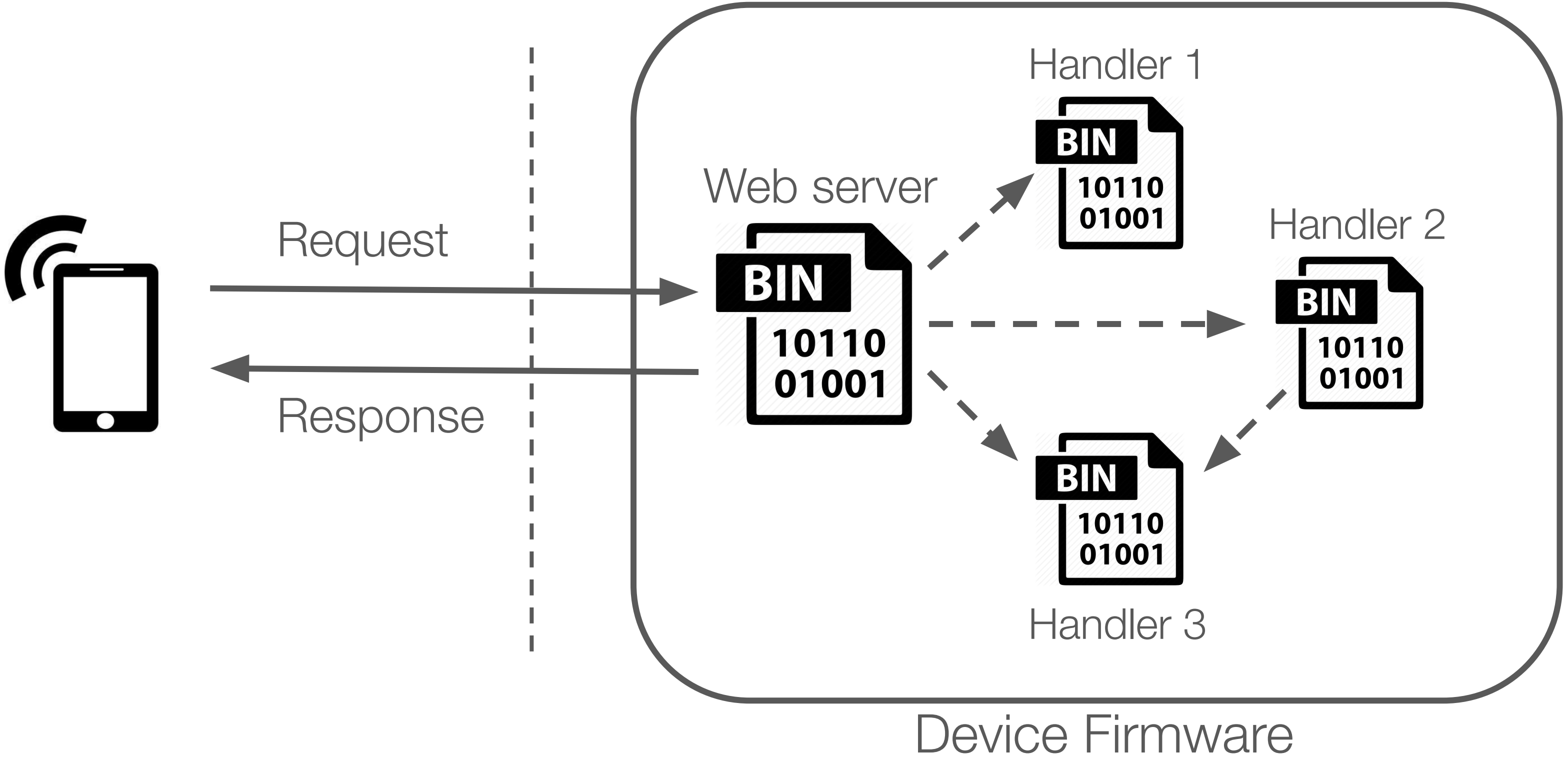
```
➜  karonte find _wr1043v2.bin.extracted/squashfs-root -exec file {} \; | grep -i elf | wc -l
     240
```

Linux-based Firmware Architecture

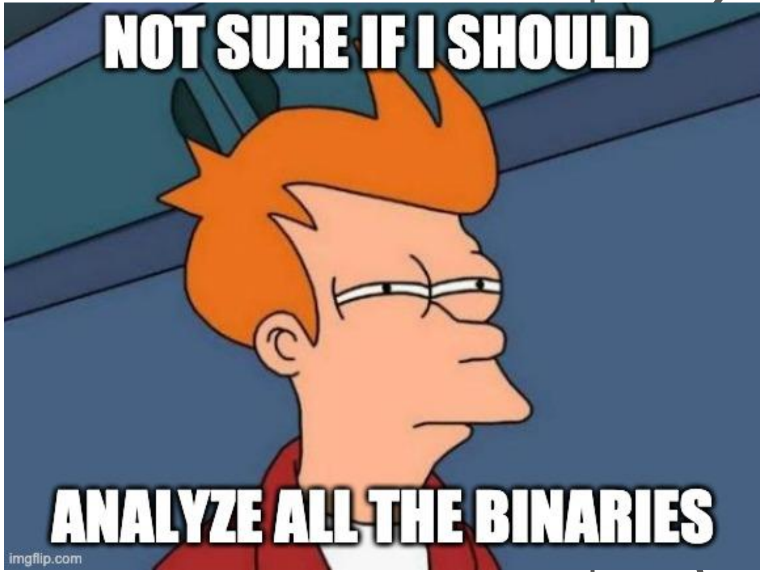


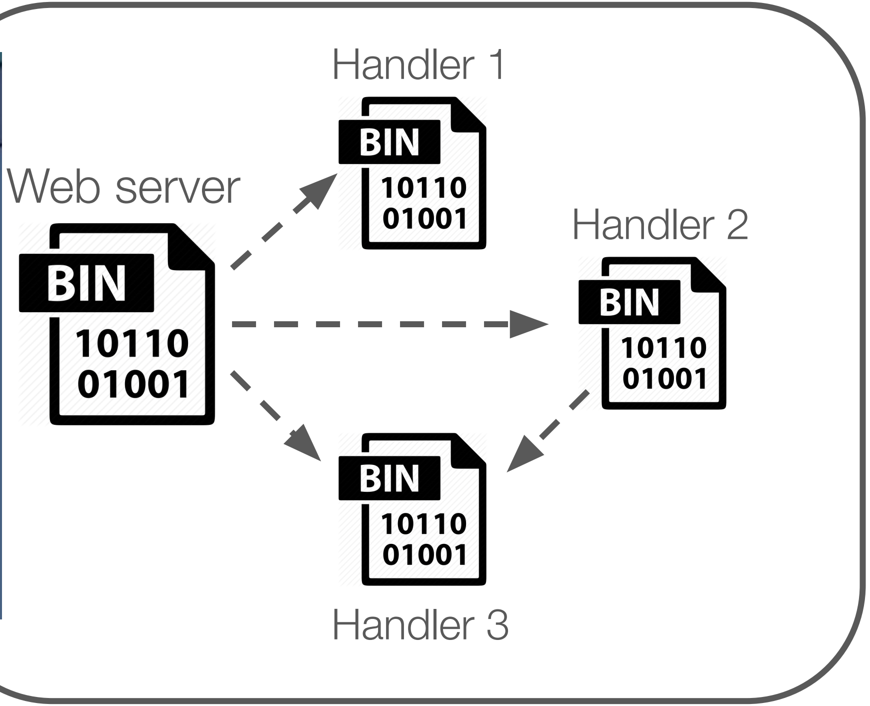
Device Firmware

Linux-based Firmware Architecture



Which binaries do we analyze?

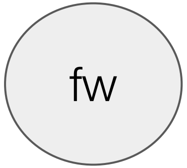




Device Firmware

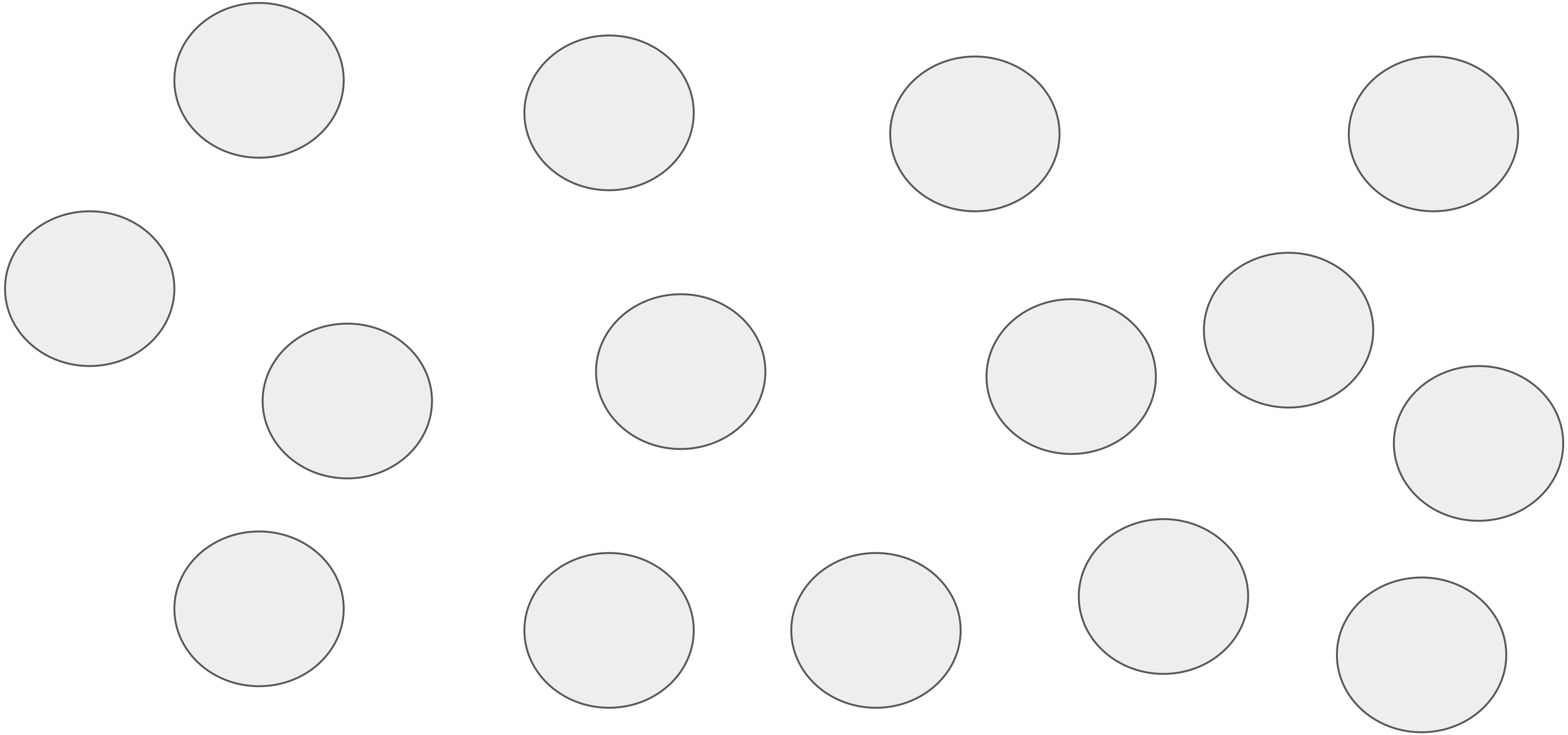
Karonte

Karonte in a Nutshell



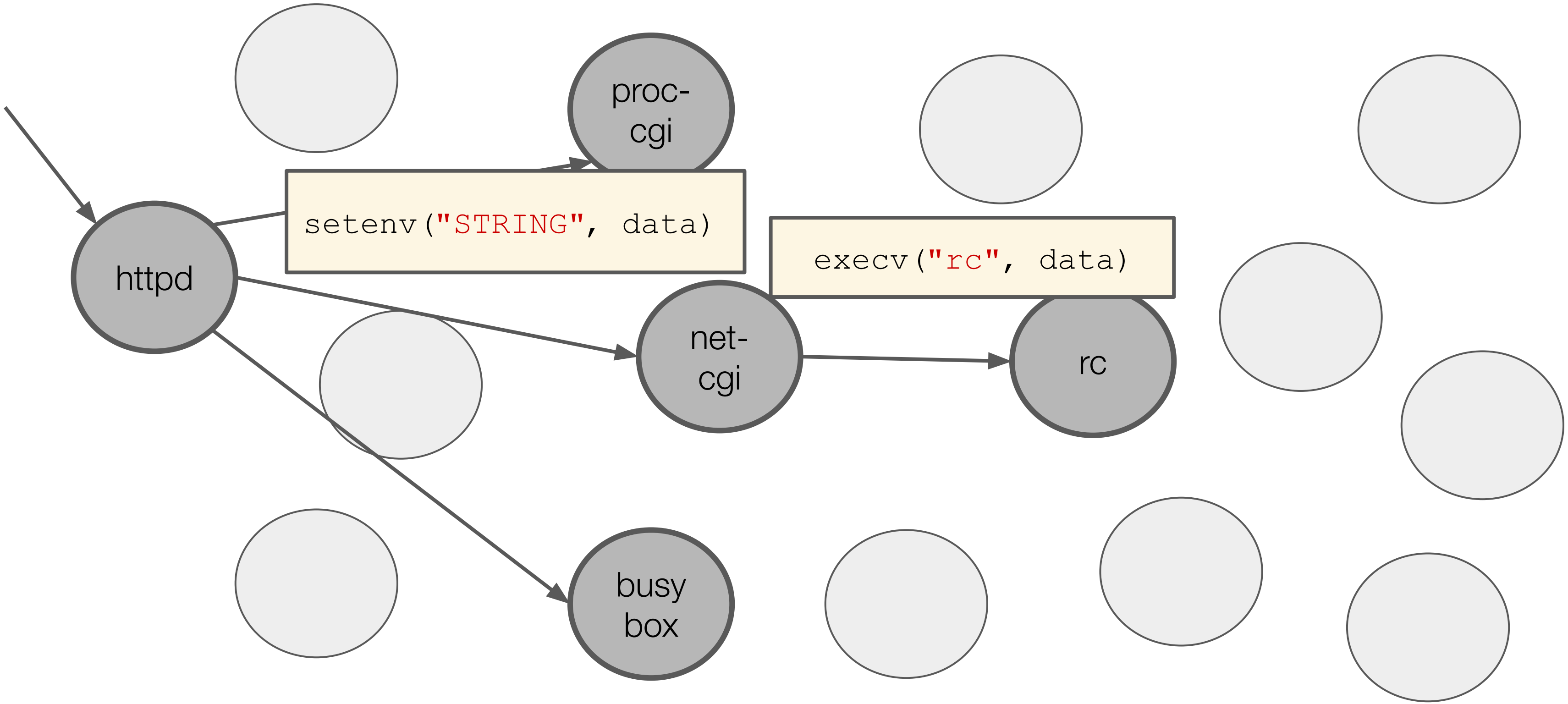
KARONTE: Detecting Insecure Multi-binary Interactions in Embedded Firmware. *IEEE S&P*, 2020

Karonte in a Nutshell



KARONTE: Detecting Insecure Multi-binary Interactions in Embedded Firmware. *IEEE S&P*, 2020

Karonte in a Nutshell

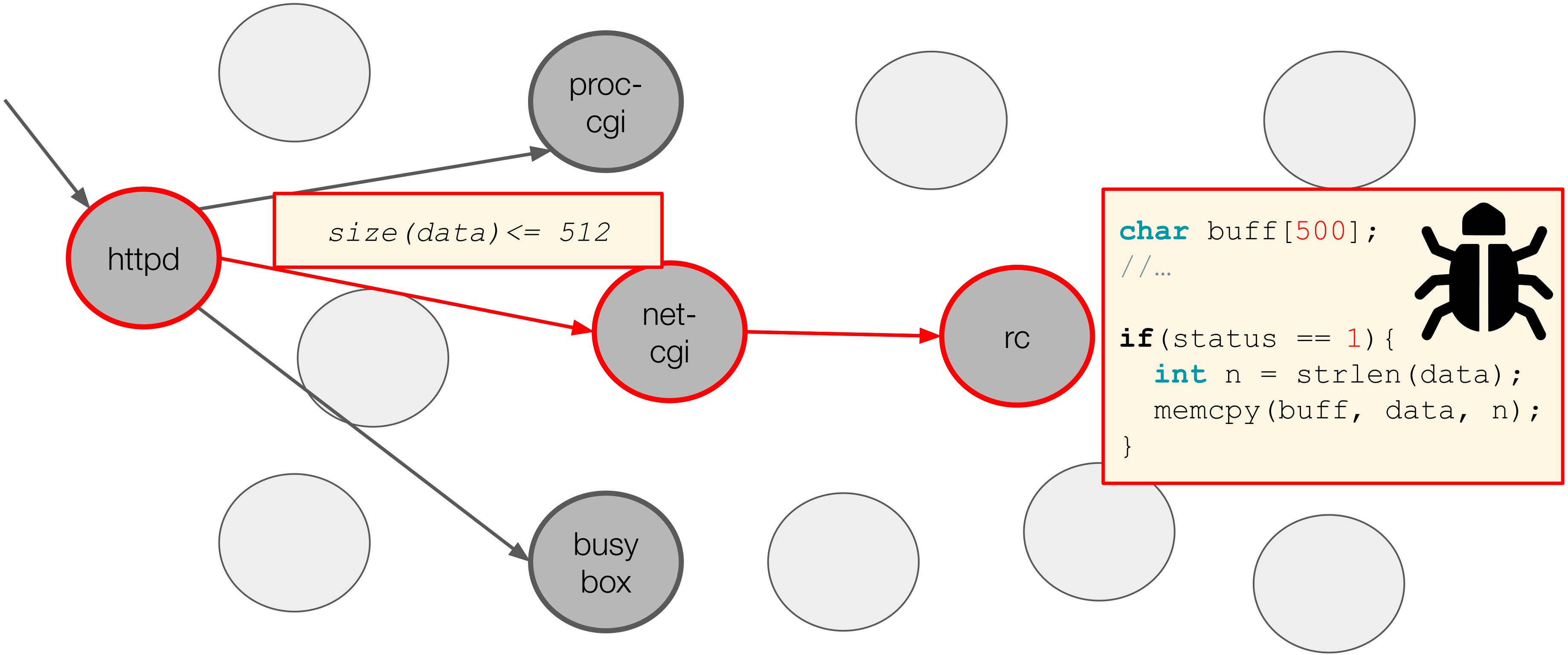


```
setenv("STRING", data)
```

```
execv("rc", data)
```

KARONTE: Detecting Insecure Multi-binary Interactions in Embedded Firmware. *IEEE S&P*, 2020

Karonte in a Nutshell



```
char buff[500];
//…

if(status == 1){
  int n = strlen(data);
  memcpy(buff, data, n);
}
```

KARONTE: Detecting Insecure Multi-binary Interactions in Embedded Firmware. *IEEE S&P*, 2020

Karonte: Evaluation

Firmware from 53 devices from 7 different vendors

46 new zero-day software bugs and rediscover another 5

Number alerts decreased from an average of **945** to an average of **5** per firmware

Alert reduction of **two orders of magnitude** and a **low false-negative rate**

	ALL			**Karonte**		
Vendor	**No. Bins**	**No. Alerts**	**Avg Time**	**No. Bins**	**No. Alerts**	**Avg Time**
NETGEAR	280	12,393	7 days	8	36	17 hours
D-Link	143	7,299	3 days	6	24	14 hours
TP-Link	110	13,104	3 days	5	2	1.5 hours
Tenda	105	3,318	5 days	6	12	1 hour
Total	**638**	**36,114**	**18 days**	**25**	**74**	**34 hours**

Follow-up Research

Use the front-end to locate the back-end code that handles the user-supplied data

- Reduce false positives

SaTC (USENIX’21)

Lightweight, on-demand, context-sensitive Reaching Definition Analysis

- Improve detection && 4x faster

HermeScan (NDSS’24)

Context-sensitive static data-flow analysis with string values reasoning

- Improve scalability

Mango (USENIX’24)

Follow-up Research

Use the front-end to locate the back-end code that handles the user-supplied data

- Reduce false positives

SaTC (USENIX'21)

✗ Alert validation and reproducibility

Lightweight, on-demand, context-sensitive Reac

✓ Scalability in multi-binary setting

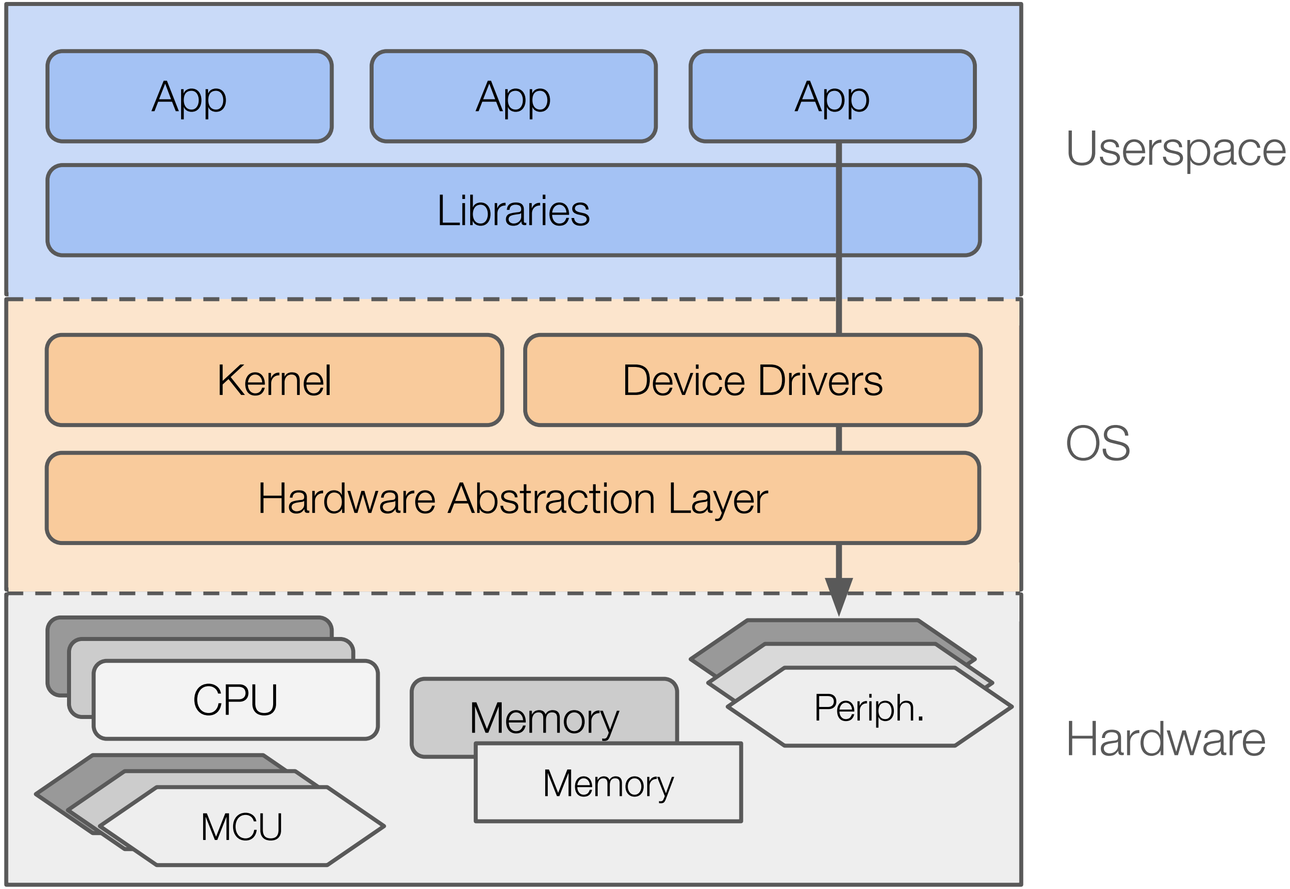
HermeScan (NDSS'24)

✗ Custom (un)packing?

Context-sensitive static data-flow analysis with s

- Improve scalability

Mango (USENIX'24)


App

App

App

Libraries

Kernel

Device Drivers

Hardware Abstraction Layer

CPU

MCU

Memory

Memory

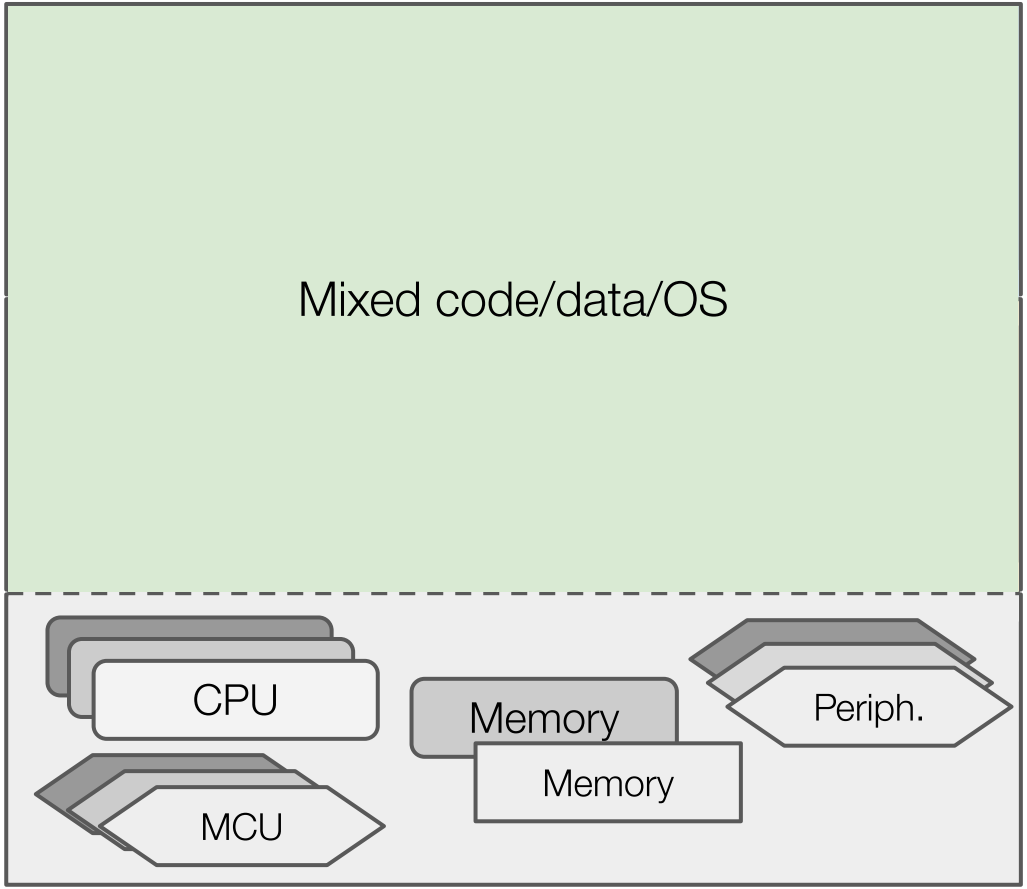
Periph.

Userspace

OS

Hardware

Type III
Monolithic



Mixed code/data/OS

CPU

Memory

Memory

Periph.

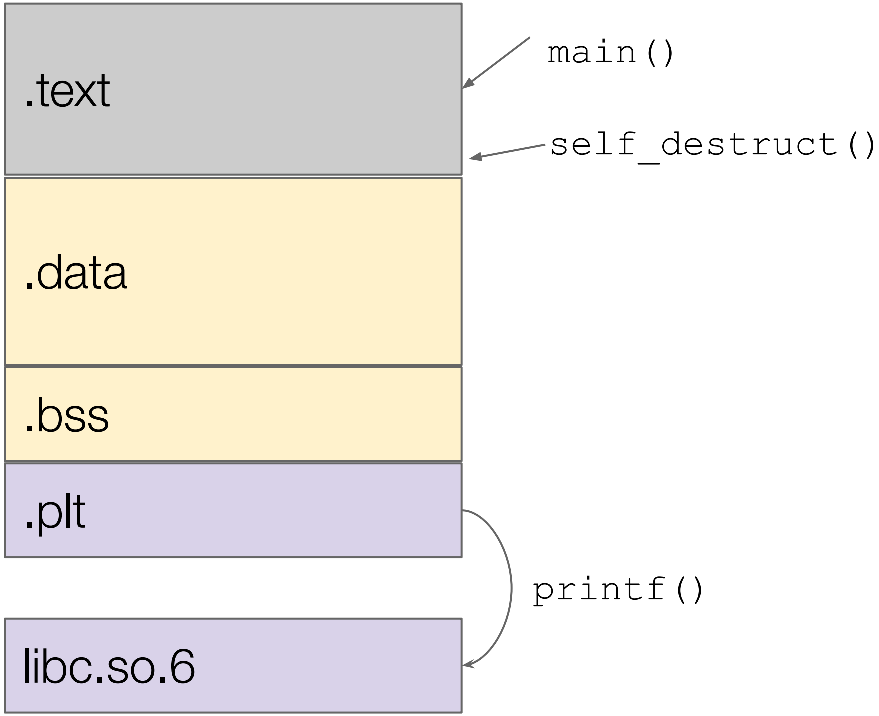
MCU

Userspace

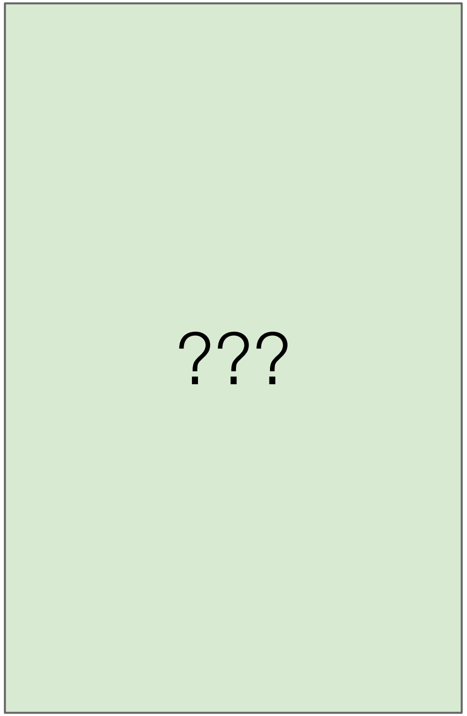
OS

Hardware

Monolithic Firmware Format



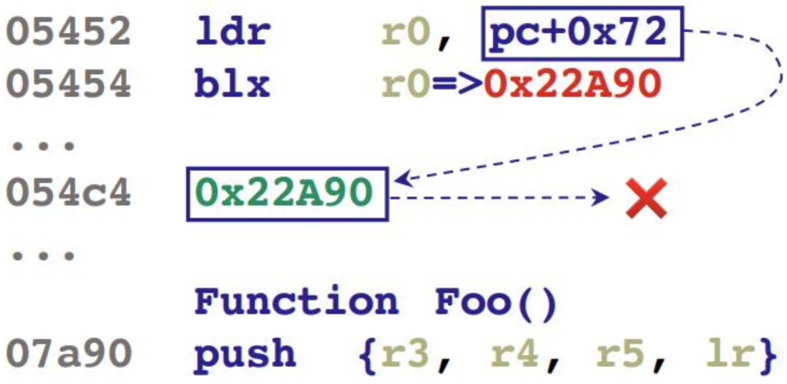
ELF file (e.g,. on Linux)



Monolithic Firmware

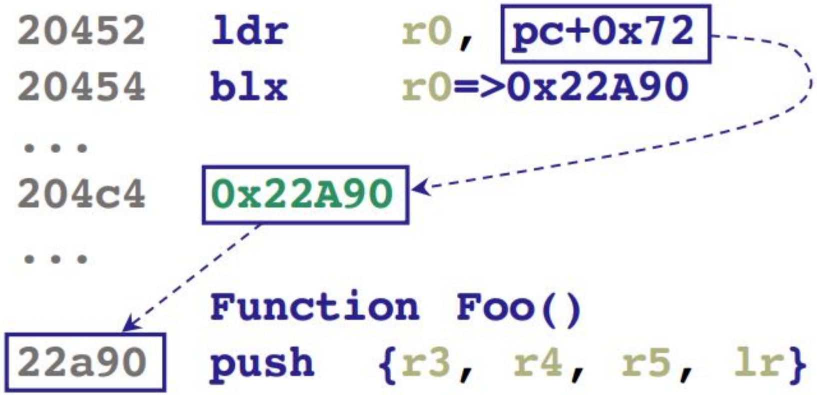
Base Address Identification

Base address: the starting memory address where programs get mapped to



```
05452   ldr     r0, pc+0x72
05454   blx     r0=>0x22A90
...
054c4   0x22A90 - - - - - - - -> ✗
...
        Function  Foo()
07a90   push   {r3, r4, r5, lr}
```

Incorrect Base: 0x0

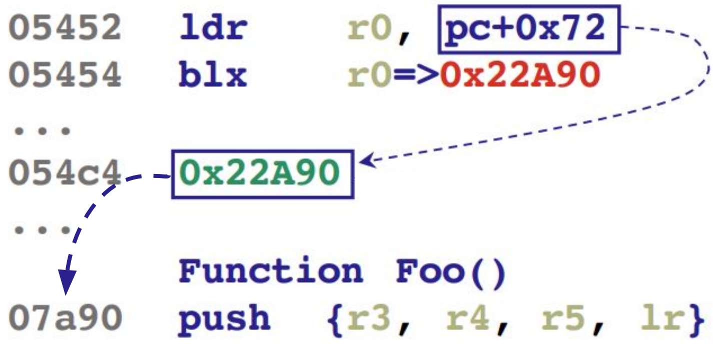


```
20452   ldr     r0, pc+0x72
20454   blx     r0=>0x22A90
...
204c4   0x22A90
...
        Function  Foo()
22a90   push   {r3, r4, r5, lr}
```

Correct Base: 0x1B000

Base Address Identification

- Load firmware at `0x0`
- Identify absolute pointers
- Solve point-to constraints of absolute pointers



```
05452   ldr      r0, pc+0x72
05454   blx      r0=>0x22A90
...
054c4   0x22A90
...
        Function Foo()
07a90   push     {r3, r4, r5, lr}
```

```
0x22A90 - 0x07a90 = 0x1b000
```

- Looking at one pointer is not enough
- Combine all the absolute pointers
- Select candidate that satisfies the most number of constraints

FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities From Bare-Metal Firmware. *CCS*, 2020

Base Address Identification

- Load firmware at 0x0
- Identify absolute pointers
- Solve point-to constraints of absolute pointers

```
05452   ldr      r0, pc+0x72
05454   blx      r0=>0x22A90
...
054c4   0x22A90
```

✓ Unlocked automation in monolithic analyses

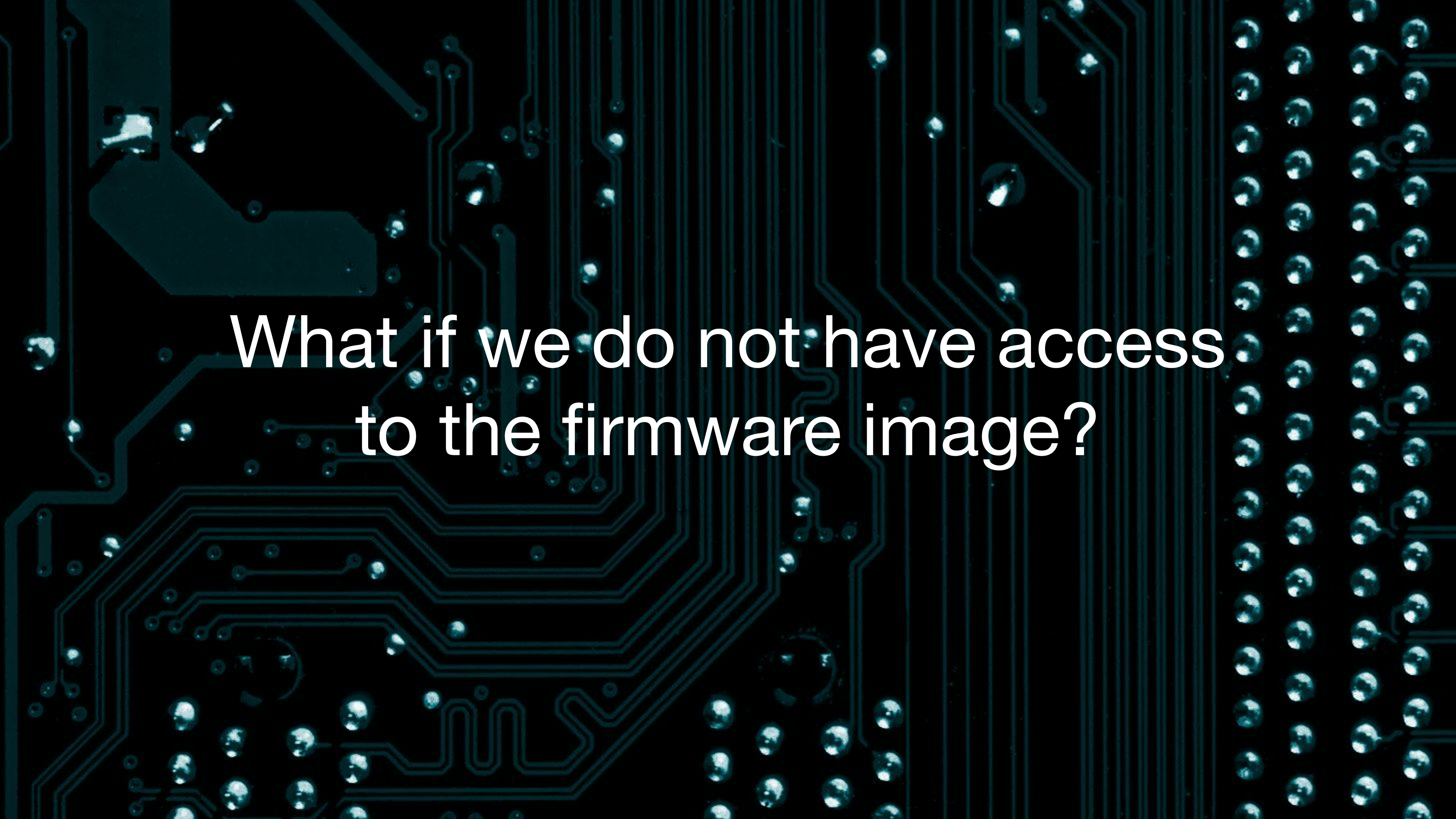
☒ Based on heuristics of hardware architecture

- L h
- Combine all the absolute pointers
- Select candidate that satisfies the most number of constraints

r}

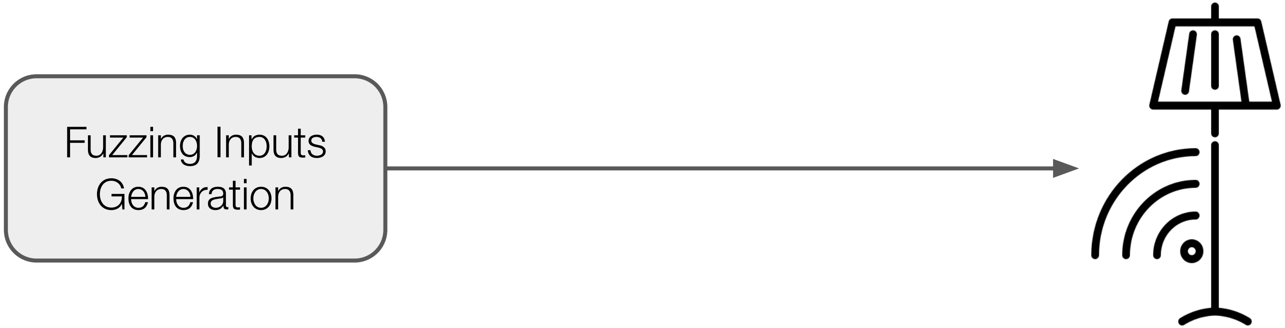
```
0x22A90 - 0x07a90 = 0x1b000
```

FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities From Bare-Metal Firmware. *CCS*, 2020



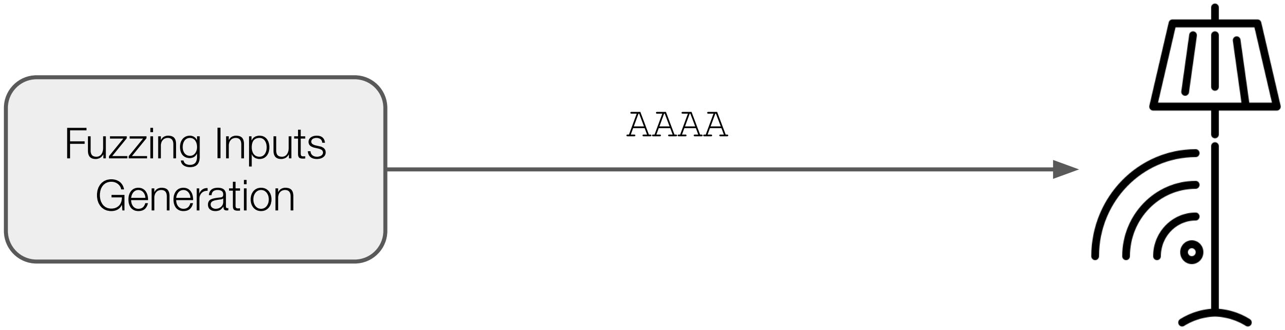
What if we do not have access to the firmware image?

Black-box Fuzzing



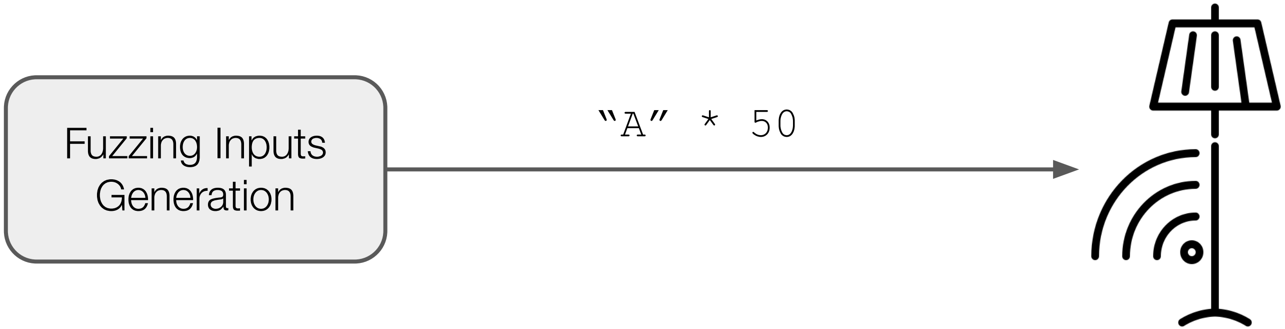
Fuzzing Inputs Generation

Black-box Fuzzing



```
char data[100];
...
read(socket, buff, 500);
...
strcpy(data, buff);
```

Black-box Fuzzing

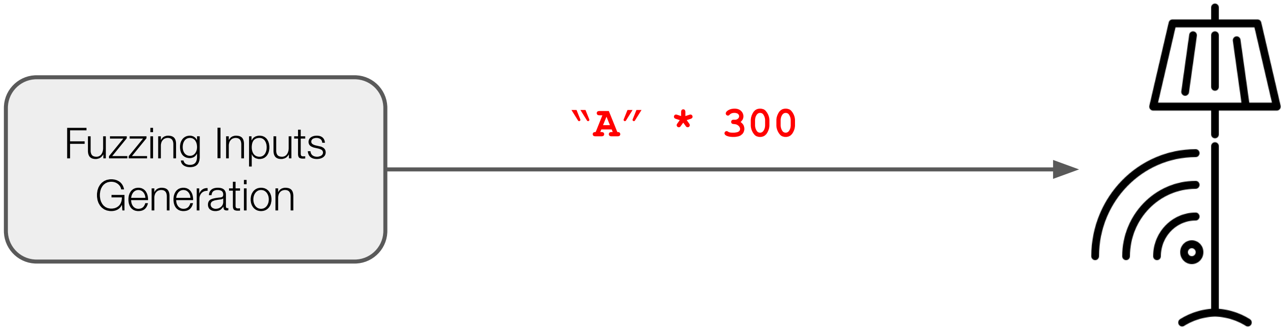


Fuzzing Inputs Generation

“A” * 50

```
char data[100];
...
read(socket, buff, 500);
...
strcpy(data, buff);
```

Black-box Fuzzing

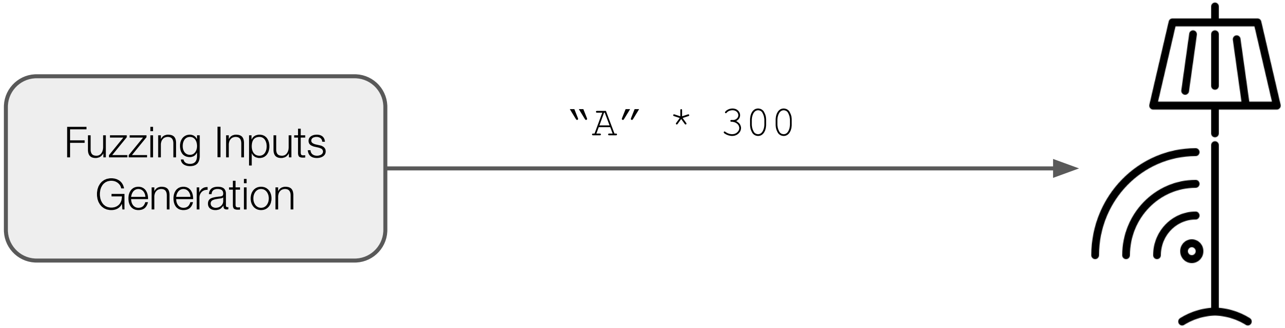


Fuzzing Inputs Generation

“A” * 300

```
char data[100];
...
read(socket, buff, 500);
...
strcpy(data, buff);
```

Black-box Fuzzing

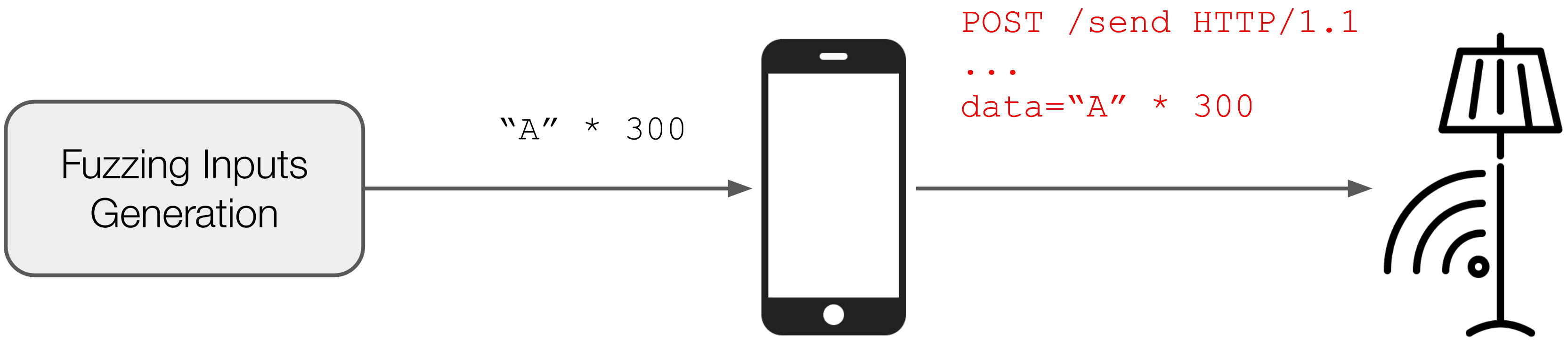


Fuzzing Inputs Generation

“A” * 300

```
char data[100];
...
read(socket, buff, 500);
if (!valid_http_req(buff))
     return;
...
strcpy(data, buff);
```

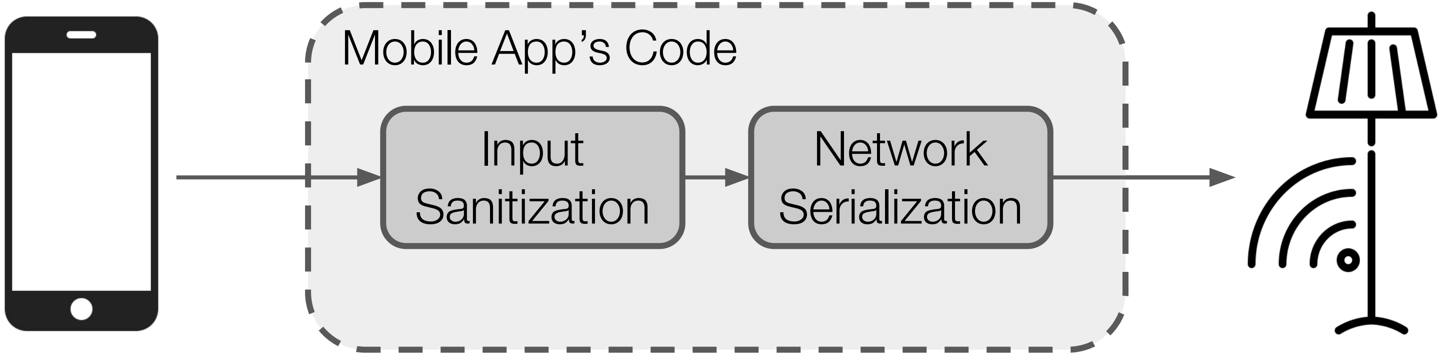
IoTFuzzer



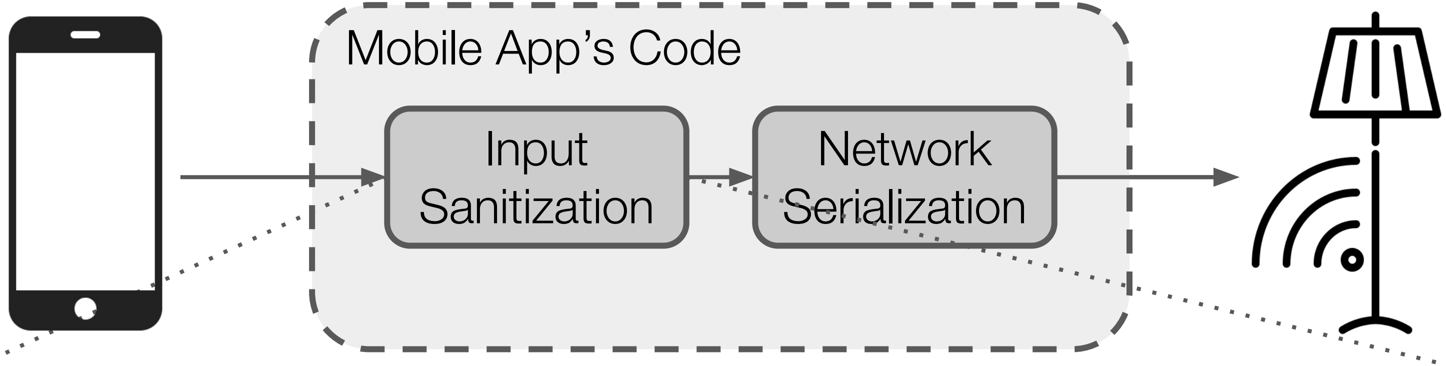
```
char data[100];
...
read(socket, buff, 500);
if (!valid_http_req(buff))
     return;
...
strcpy(data, buff);
```

IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing, NDSS'18

Fuzzing Triggers

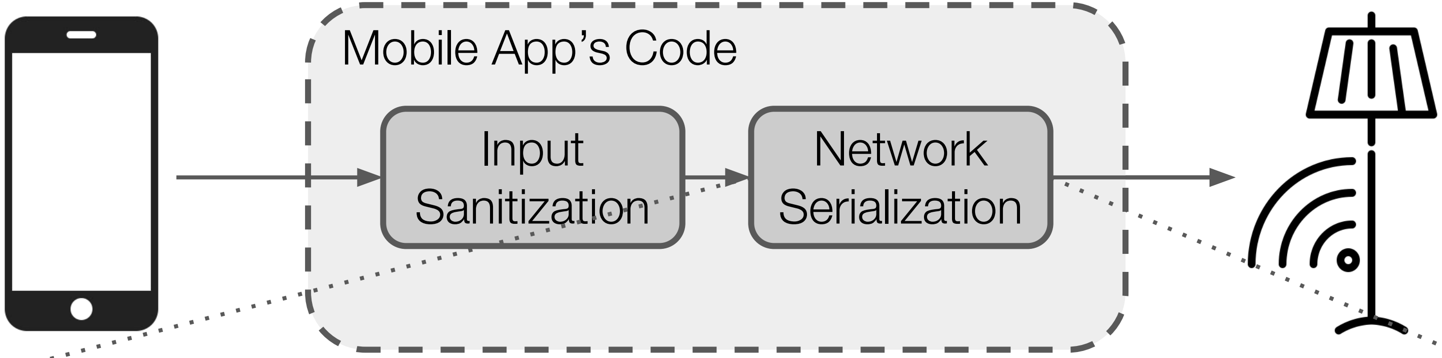


Fuzzing Triggers



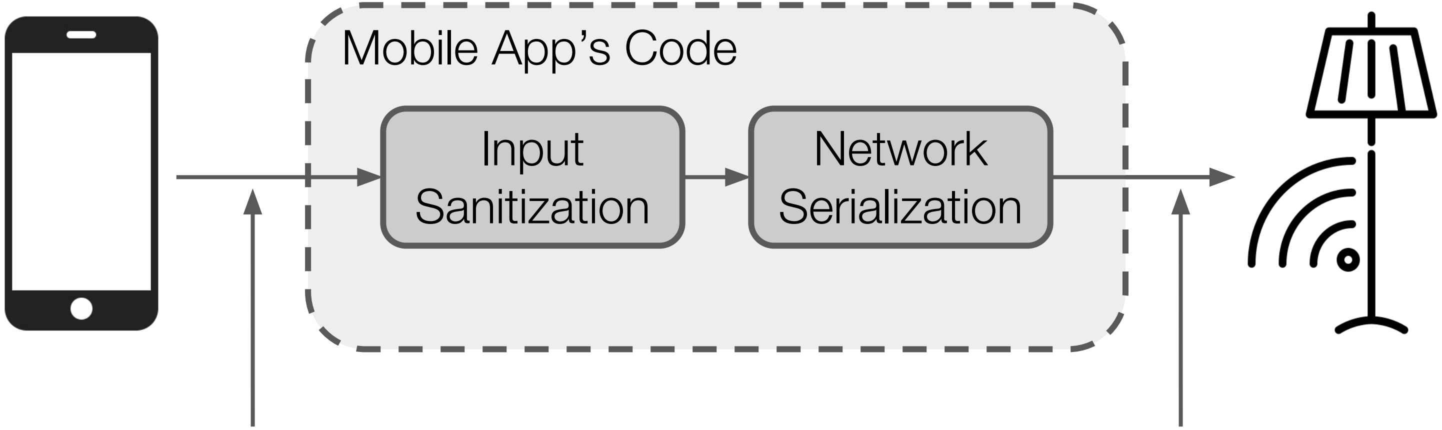
```
if (!adminPwd.contains("&") && adminPwd.length() < 32){
    SendMsg(adminPwd, camId, data);
}
else
    return false;
```

Fuzzing Triggers



```
...
String json = "{\"op\": \"auth\", \"pass\":" + adminPwd "}";
String encoded = Base64.encode(json);

httpSend(DEVICE_IP; encoded);
```

Fuzzing Triggers



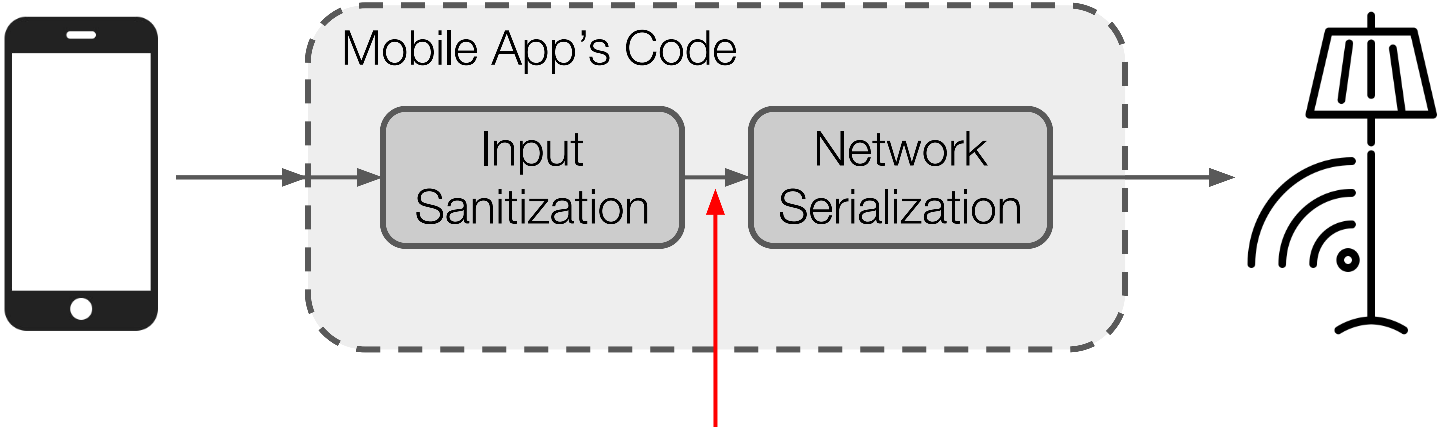
UI-level

Limited by app-side sanitization ❌

Network-level

Invalid inputs ❌

Fuzzing Triggers

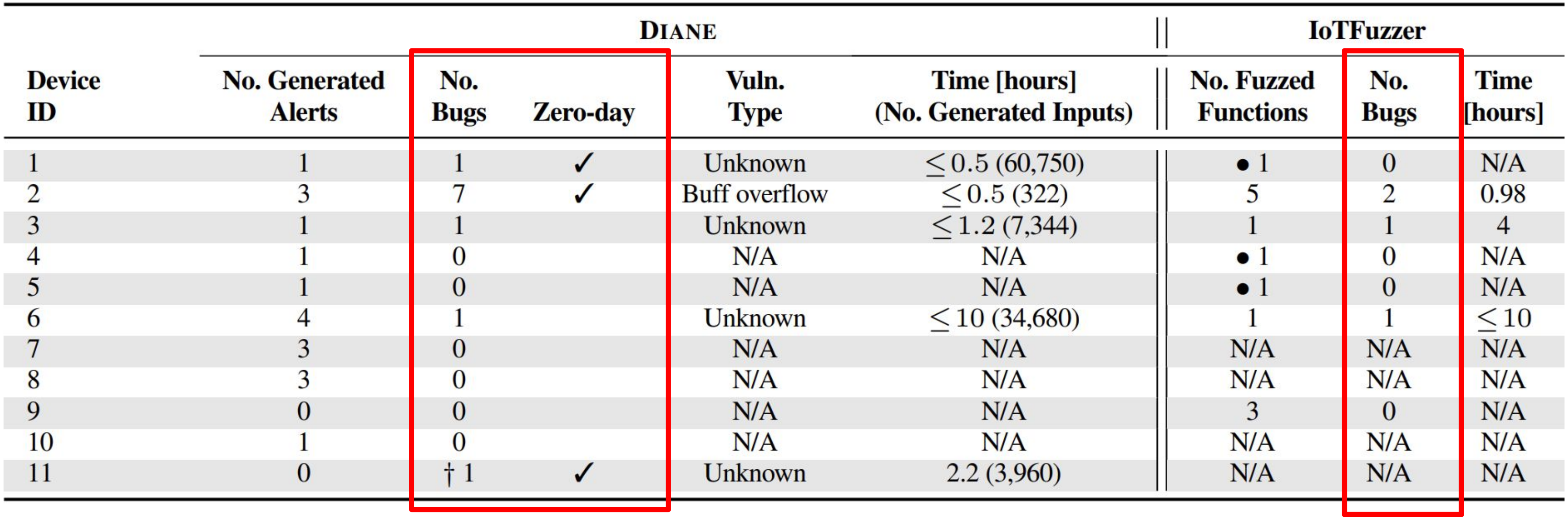


Our Approach = Fuzzing Trigger

Valid inputs ✓

Not limited by app-side input sanitization ✓

Diane: Evaluation

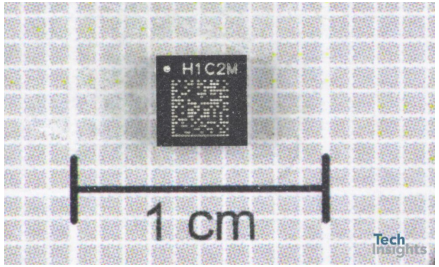


	DIANE					IoTFuzzer		
Device ID	No. Generated Alerts	No. Bugs	Zero-day	Vuln. Type	Time [hours] (No. Generated Inputs)	No. Fuzzed Functions	No. Bugs	Time [hours]
1	1	1	✓	Unknown	≤0.5 (60,750)	• 1	0	N/A
2	3	7	✓	Buff overflow	≤0.5 (322)	5	2	0.98
3	1	1		Unknown	≤1.2 (7,344)	1	1	4
4	1	0		N/A	N/A	• 1	0	N/A
5	1	0		N/A	N/A	• 1	0	N/A
6	4	1		Unknown	≤10 (34,680)	1	1	≤10
7	3	0		N/A	N/A	N/A	N/A	N/A
8	3	0		N/A	N/A	N/A	N/A	N/A
9	0	0		N/A	N/A	3	0	N/A
10	1	0		N/A	N/A	N/A	N/A	N/A
11	0	† 1	✓	Unknown	2.2 (3,960)	N/A	N/A	N/A

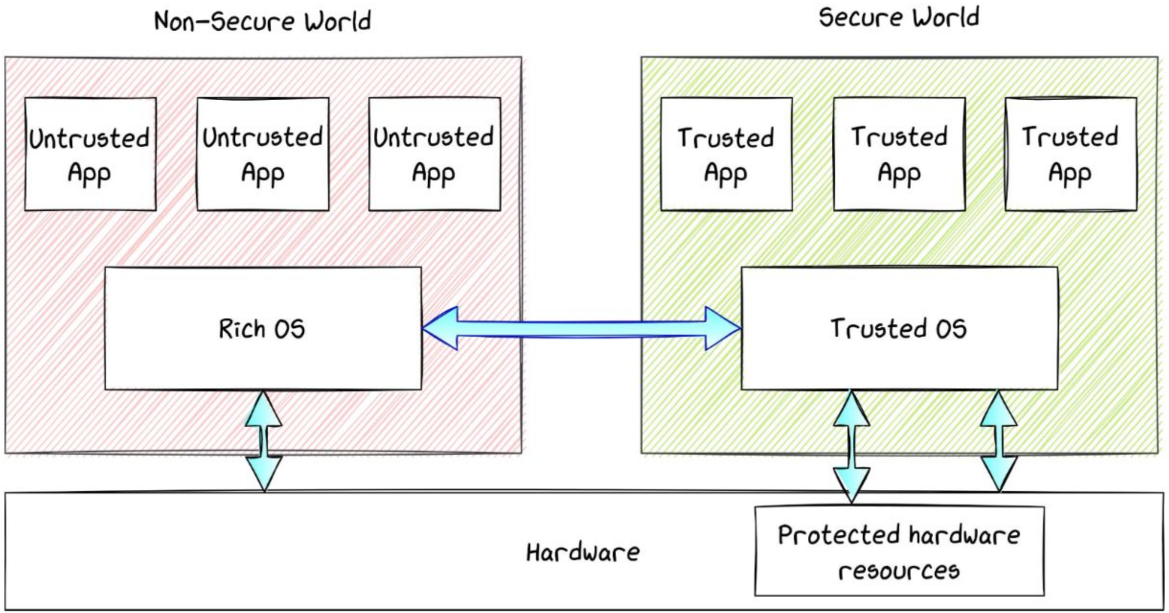
"DIANE: Identifying Fuzzing Triggers in Apps to Generate Under-constrained Inputs for IoT Devices", *IEEE S&P* 2021

Use case: Popular Smart Lock

Google Titan M Chip



External Coprocessor: Trusted Execution Environment (TEE)



Results & Outcomes

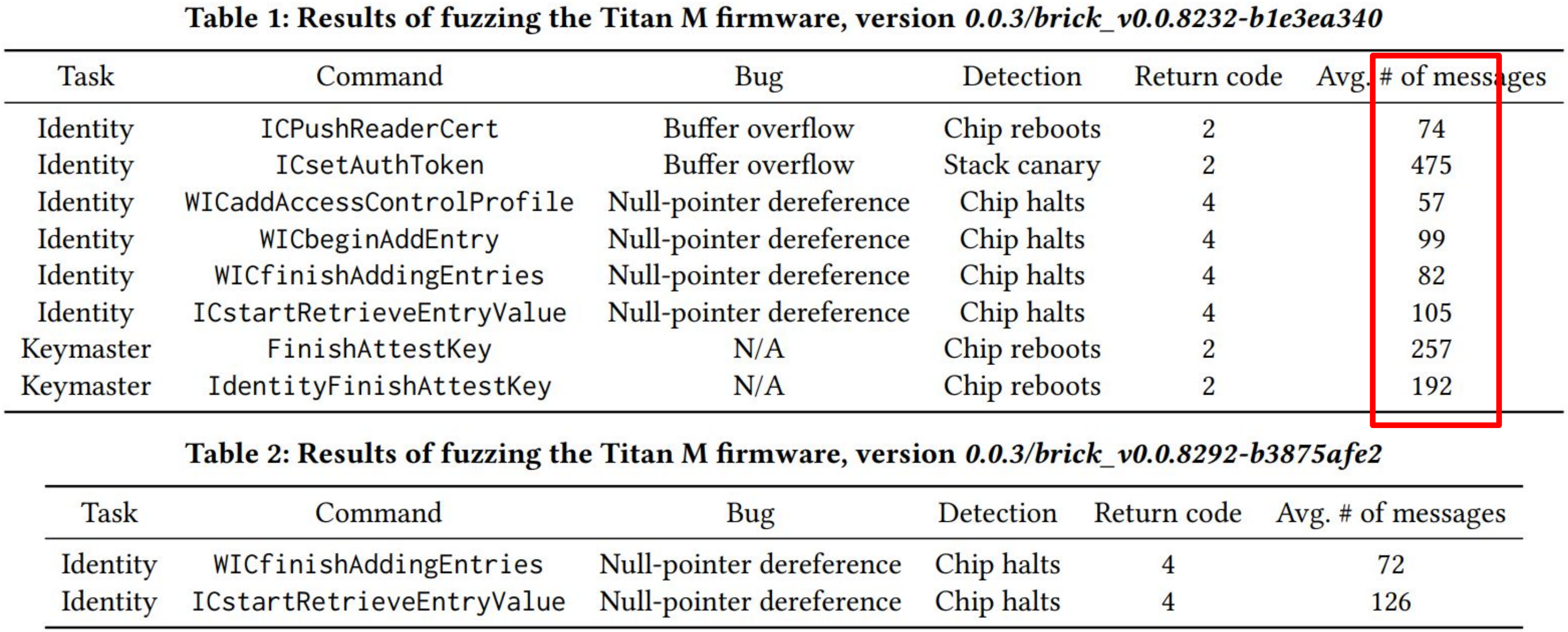


Table 1: Results of fuzzing the Titan M firmware, version *0.0.3/brick_v0.0.8232-b1e3ea340*

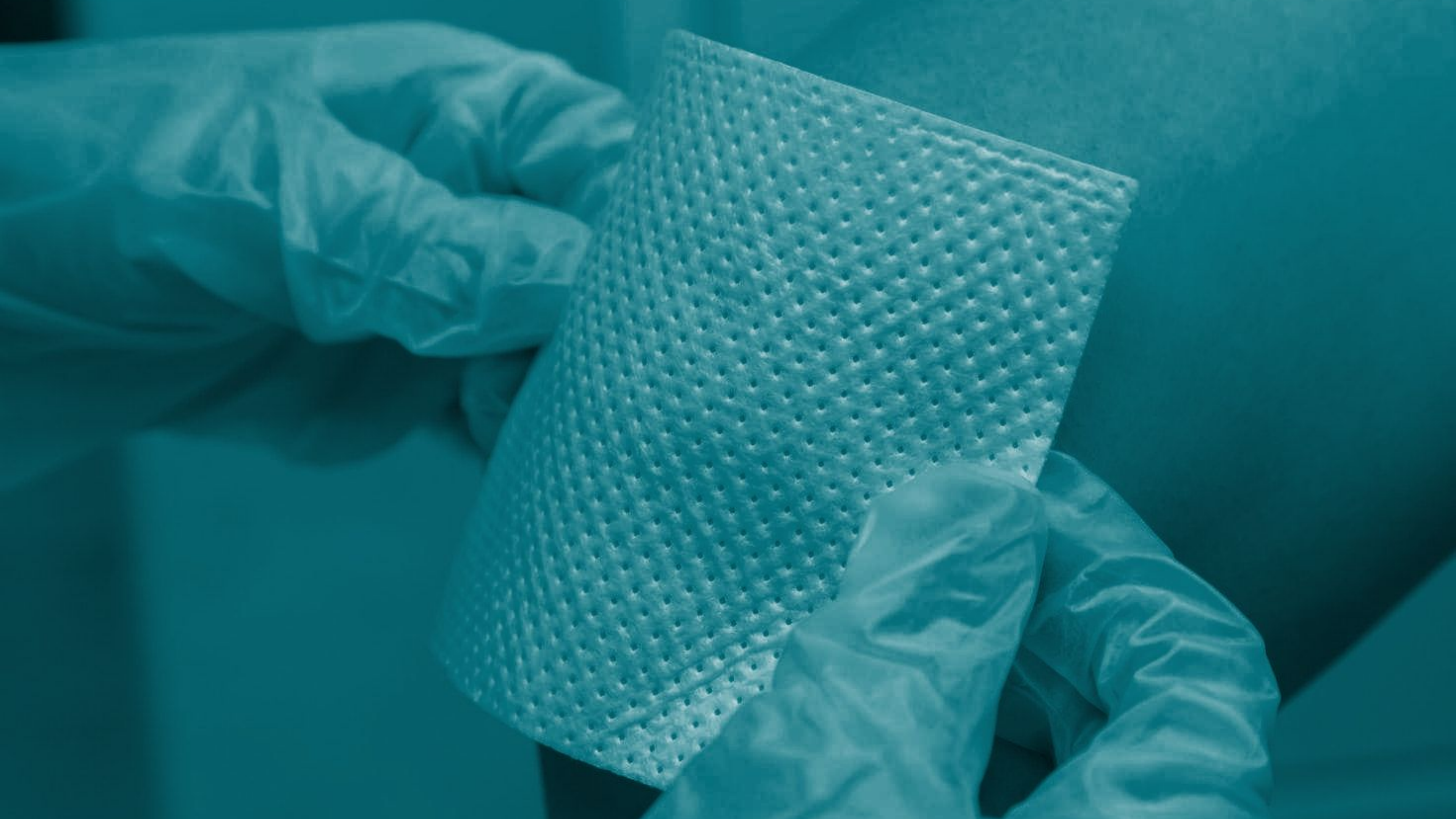
Task	Command	Bug	Detection	Return code	Avg. # of messages
Identity	`ICPushReaderCert`	Buffer overflow	Chip reboots	2	74
Identity	`ICsetAuthToken`	Buffer overflow	Stack canary	2	475
Identity	`WICaddAccessControlProfile`	Null-pointer dereference	Chip halts	4	57
Identity	`WICbeginAddEntry`	Null-pointer dereference	Chip halts	4	99
Identity	`WICfinishAddingEntries`	Null-pointer dereference	Chip halts	4	82
Identity	`ICstartRetrieveEntryValue`	Null-pointer dereference	Chip halts	4	105
Keymaster	`FinishAttestKey`	N/A	Chip reboots	2	257
Keymaster	`IdentityFinishAttestKey`	N/A	Chip reboots	2	192

Table 2: Results of fuzzing the Titan M firmware, version *0.0.3/brick_v0.0.8292-b3875afe2*

Task	Command	Bug	Detection	Return code	Avg. # of messages
Identity	`WICfinishAddingEntries`	Null-pointer dereference	Chip halts	4	72
Identity	`ICstartRetrieveEntryValue`	Null-pointer dereference	Chip halts	4	126

"Reversing and Fuzzing the Google Titan M Chip", *ROOTS*, 2021

CRASHES
CRASHES EVERYWHERE
imgflip.com



Patching Injection for Monolithic Firmware

Third-party automated patching, without source code, is particularly hard

Creating a Patch

What’s the input? No standard sources of input, numerous hardware peripherals

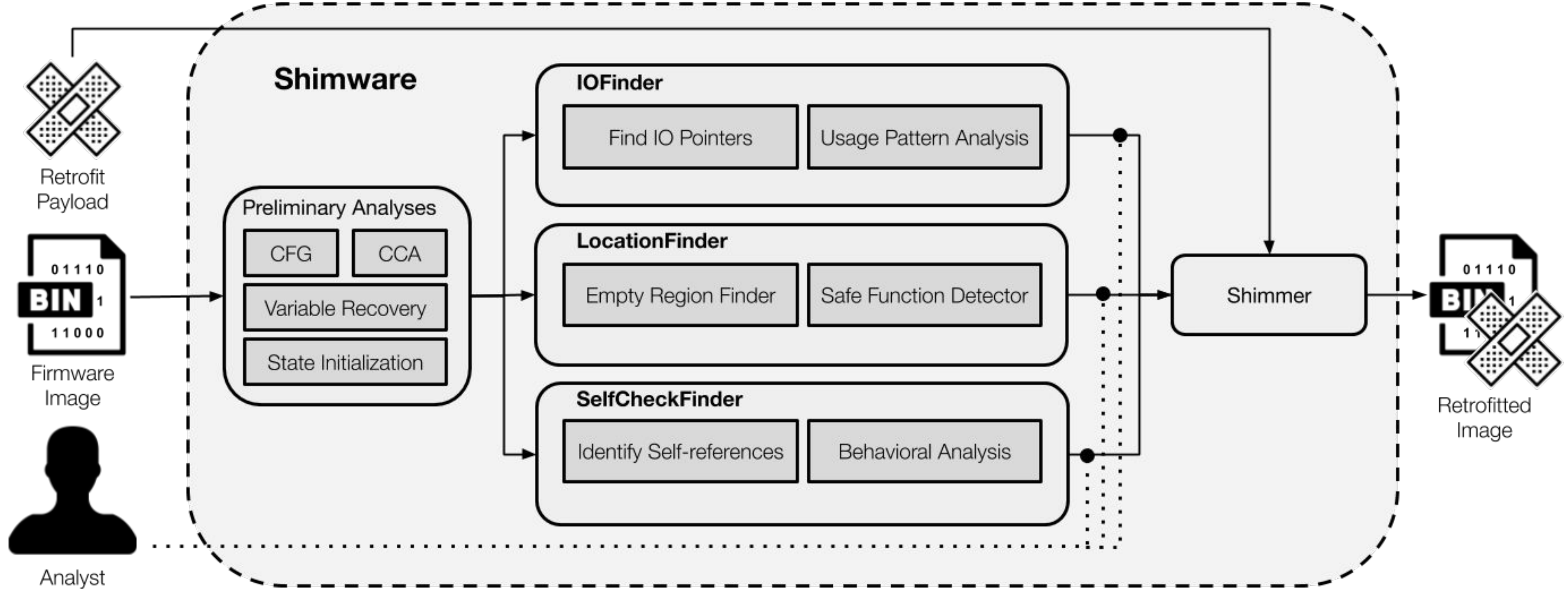
Inserting a Patch

Where? We cannot simply inject & shift && we have space issues

Deploying a Patch

How? Verification mechanism to preserve integrity

Retrofitting Monolithic Firmware



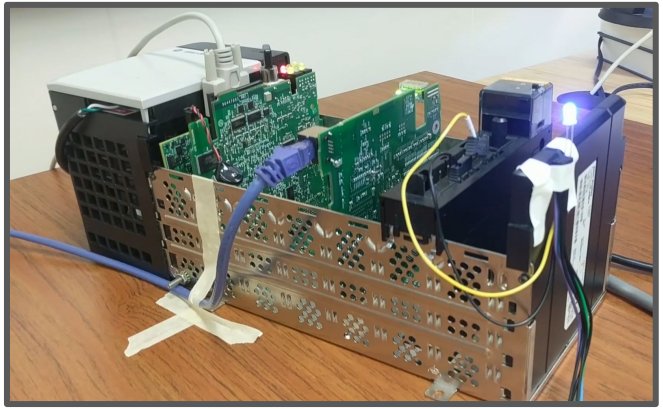
Shimware: Evaluation

Synthetic datasets: 17 + 50 firmware samples

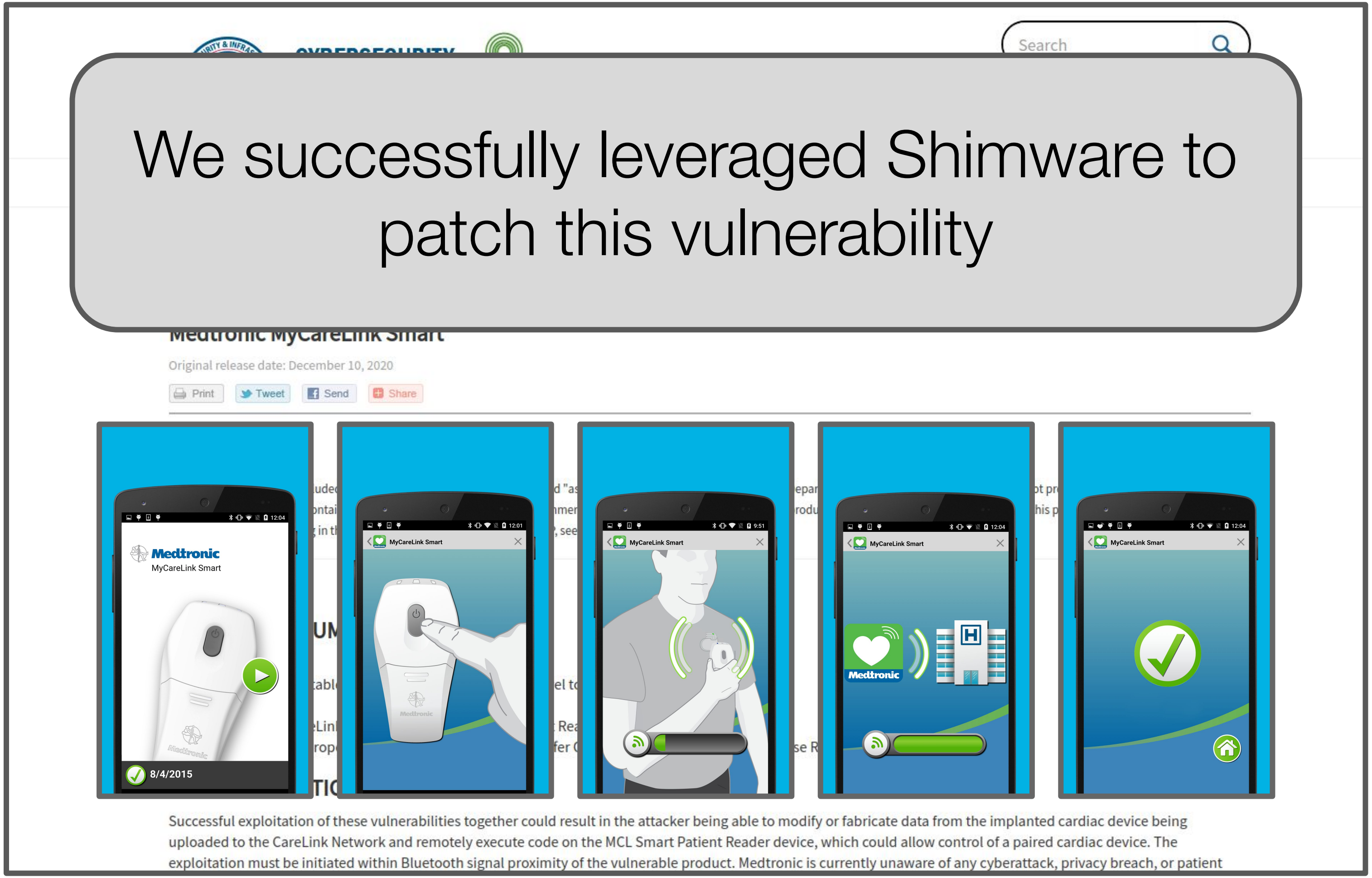
- Recovered **244** useful I/O operations; **8** false negatives
- On average, Shimware detected **375 bytes** of available patching space
  - Max: 75K bytes; Min: 28 bytes

Real-world use cases

- Power Supply device
- Programmable Logic Controller (PLC)
- Pacemaker Monitor



“Shimware: Toward Practical Security Retrofitting for Monolithic Firmware Images”, *RAID, 2023* https://us-cert.cisa.gov/ics/advisories/icsma-20-345-01



We successfully leveraged Shimware to patch this vulnerability

Medtronic MyCareLink Smart

Original release date: December 10, 2020

Print Tweet Send Share

Successful exploitation of these vulnerabilities together could result in the attacker being able to modify or fabricate data from the implanted cardiac device being uploaded to the CareLink Network and remotely execute code on the MCL Smart Patient Reader device, which could allow control of a paired cardiac device. The exploitation must be initiated within Bluetooth signal proximity of the vulnerable product. Medtronic is currently unaware of any cyberattack, privacy breach, or patient

Toward Automated Patching?

- ✗ Lack of automated vulnerability analysis techniques
- ✗ How to automatically generate and test patches?
- ✓ (Semi-)Automated patch injection
- ✗ Verifying secure firmware update mechanisms?

AoT (EuroS&P'23)

Coordinated Vulnerability Disclosure

We established a university-wide policy on coordinated vulnerability disclosure

- Clear to researchers & students how to behave (+ guidelines)
- Leverage in demanding that researchers follow these procedures
- Provides researchers with assurance that they will be protected
- Clear to recipients of disclosure notices how we handle the process

https://www.utwente.nl/en/digital-society/research/tuccr/impact/vulnerability_disclosure/

"Operationalizing Cybersecurity Research Ethics Review: From Principles and Guidelines to Practice", *EthiCS, 2023*

✓ The Good

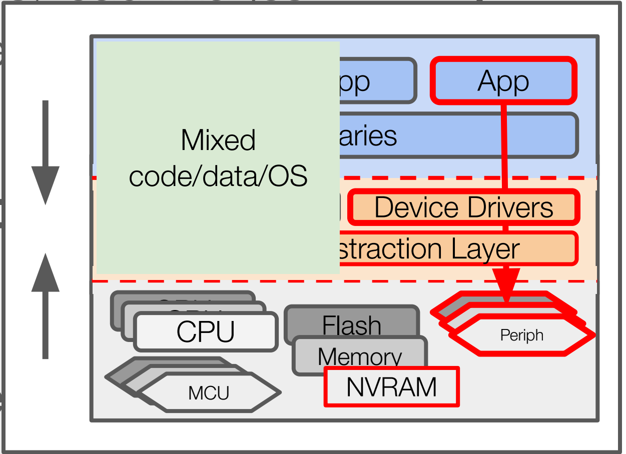
- Better real-world datasets
- Good tools, we can deal with multiple binaries, load images
- Good (use-case-specific) emulators && fuzzers
- We automatically find vulnerabilities!

✗ The Bad

- One size does not fit all, firmware is heterogeneous
- Lots of heuristics, dependent on limited hardware configurations
- Unclear static vs. dynamic analysis trade-off
- Limited focus on mitigation
- What after bug detection? No vulnerability analysis

The Good

- Better real-world datasets
- Good tools, we can deal with multiple binaries, load images
- Good (use-case-specific) emulators && fuzze
- We automatically find vulnerabilities!

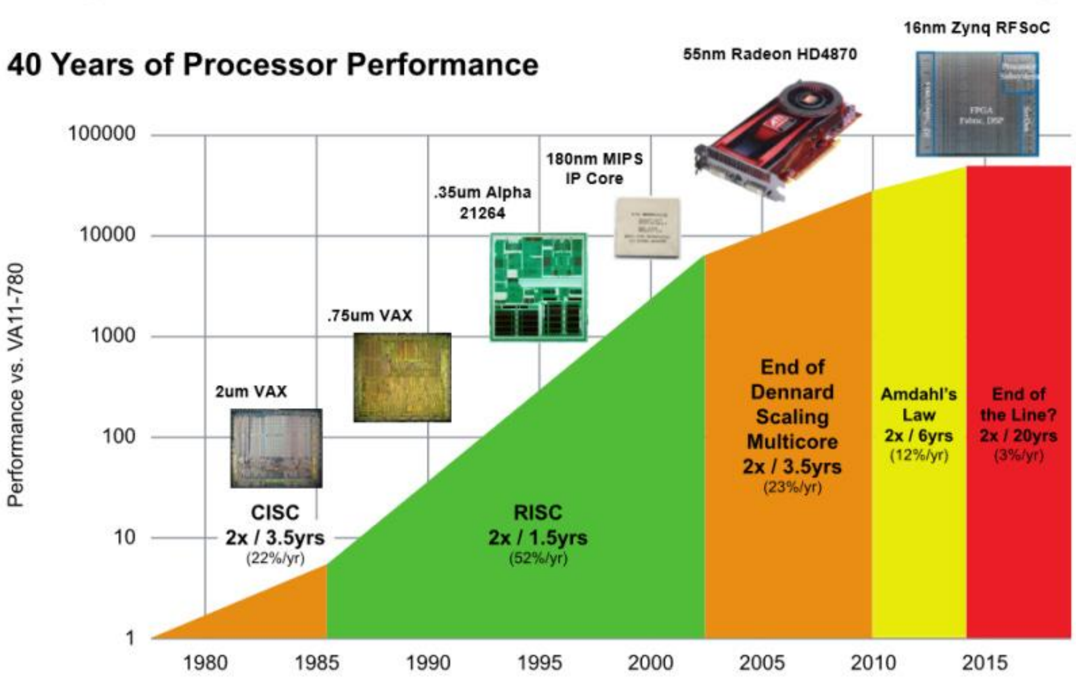


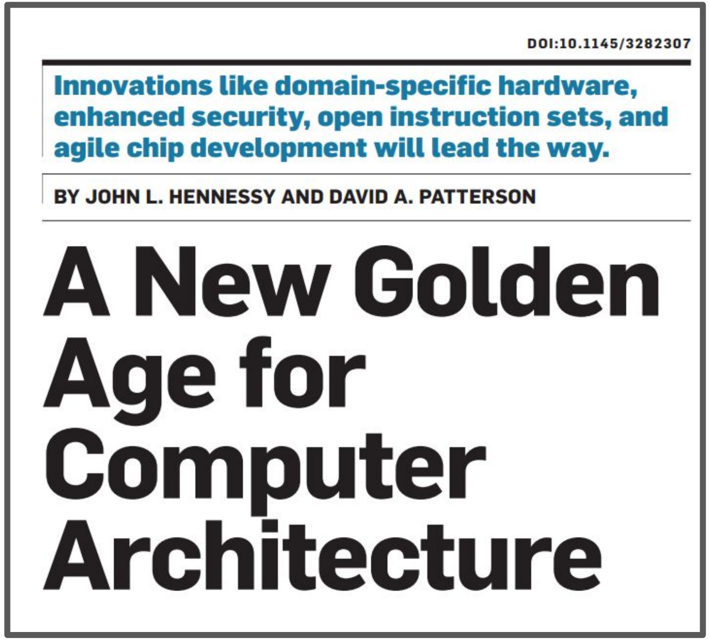
The Bad

- One size does not fit all, firmware is heteroge
- Lots of heuristics, dependent on limited hardware configurations
- Unclear static vs. dynamic analysis trade-off
- Limited focus on mitigation
- What after bug detection? No vulnerability analysis

or isn’t it?

How does the future look like?




DOI:10.1145/3282307

Innovations like domain-specific hardware, enhanced security, open instruction sets, and agile chip development will lead the way.

BY JOHN L. HENNESSY AND DAVID A. PATTERSON

A New Golden Age for Computer Architecture

How does the future look like?

40 Years of Processor Performance
100000
10000
1000
vs. VA11-780
55nm Radeon HD4870
16nm Zynq RFSoC
End of Dennard Scaling Multicore 2x / 3.5yrs (23%/yr)
Amdahl's Law 2x / 6yrs (12%/yr)
2005
2010
DOI:10.1145/3282307
Innovations like domain-specific hardware,
sets, and
way.
en
re
Google
Tensor Processing Unit
AI
ARTIFICIAL INTELLIGENCE

What if in 10-20 years it is all “firmware”?

Thanks!
Questions?

Andrea Continella
<a.continella@utwente.nl>
https://conand.me
🐦 @_conand

- Firmware requires re-thinking automated security analysis methodologies
- Significant advances in firmware analysis. Yet, we often lack generalizability
- Vulnerability discovery alone won’t be enough. We need to automate patching
- What if in 10-20 years it is all “firmware”?

UNIVERSITY
OF TWENTE.