riscure

The EXFILES project has received funding from the European Union’s Horizon 2020 research and. innovation programme under grant agreement No 883156

Black box fuzzing with side channels

Sergei Volokitin

EX FILES

White box setting

- Vulnerability research:
  - Source code review
  - Reverse engineering
  - Debugging
  - Fuzzing
  - etc.

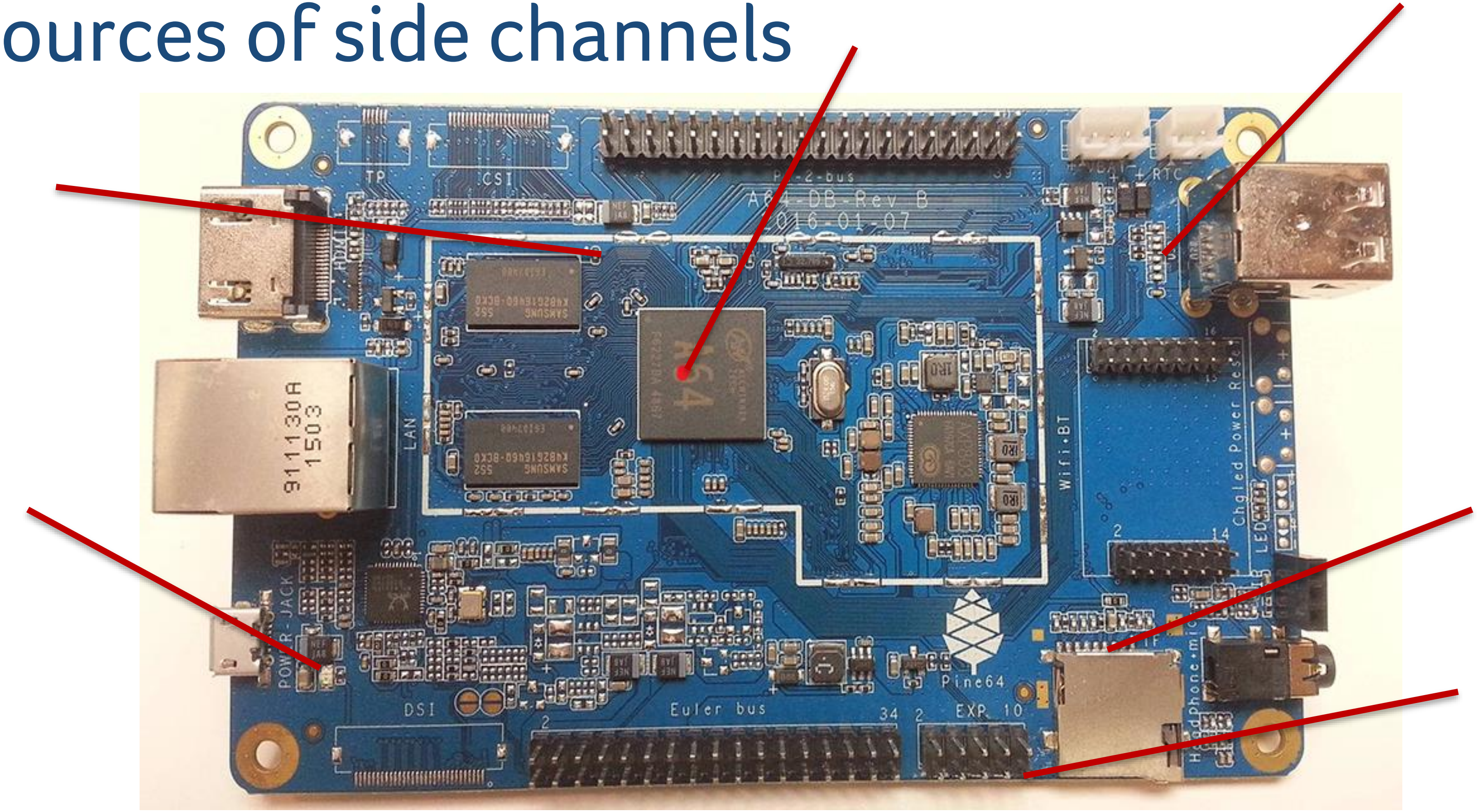
Black box setting

- Targets with limited/no public spec
- No source code
- No available binary
- Encrypted updates
- Protected memory
- No debug

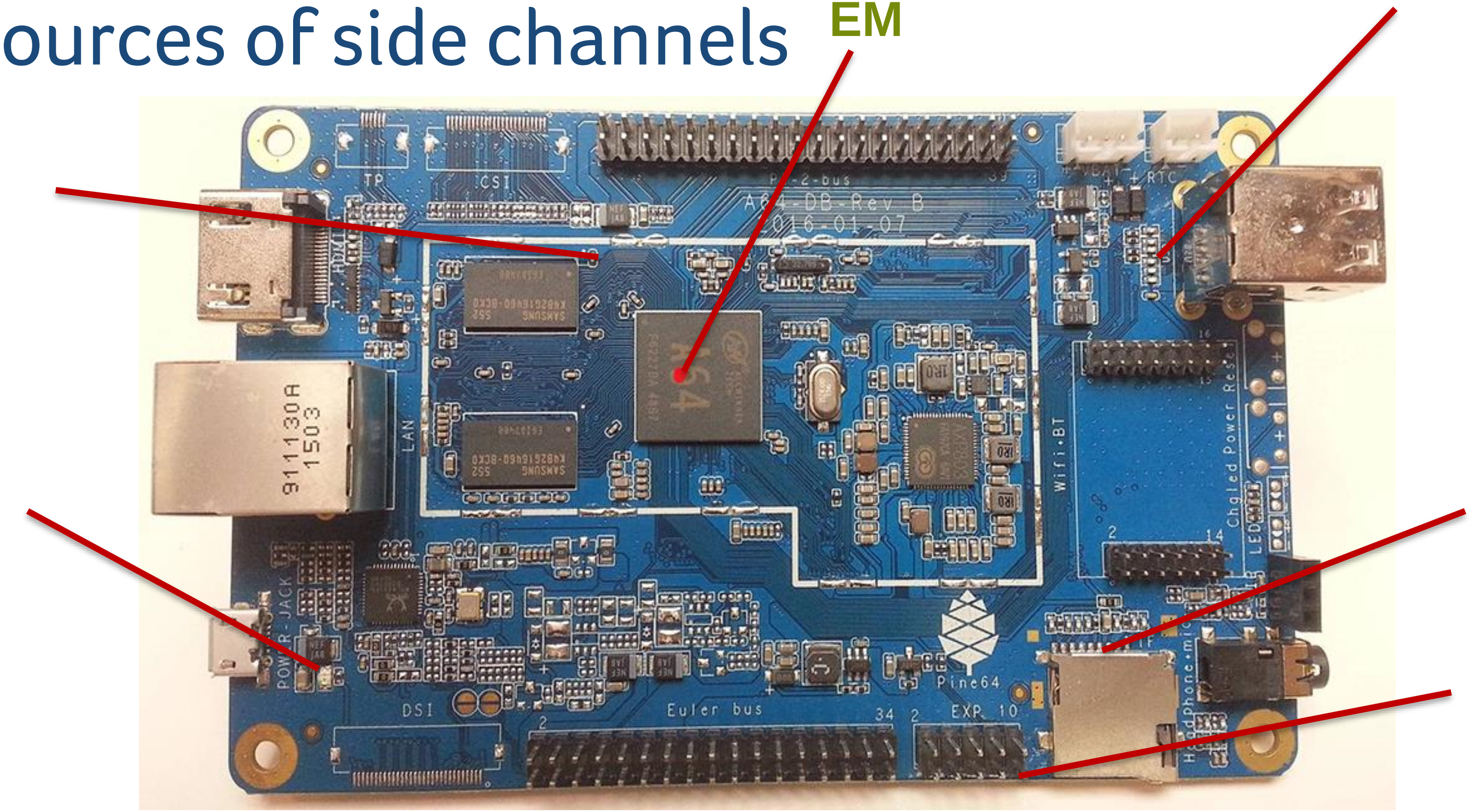
Fuzzing with side channels

Can we explore black box targets more efficiently by leveraging physical access to the target?

Sources of side channels

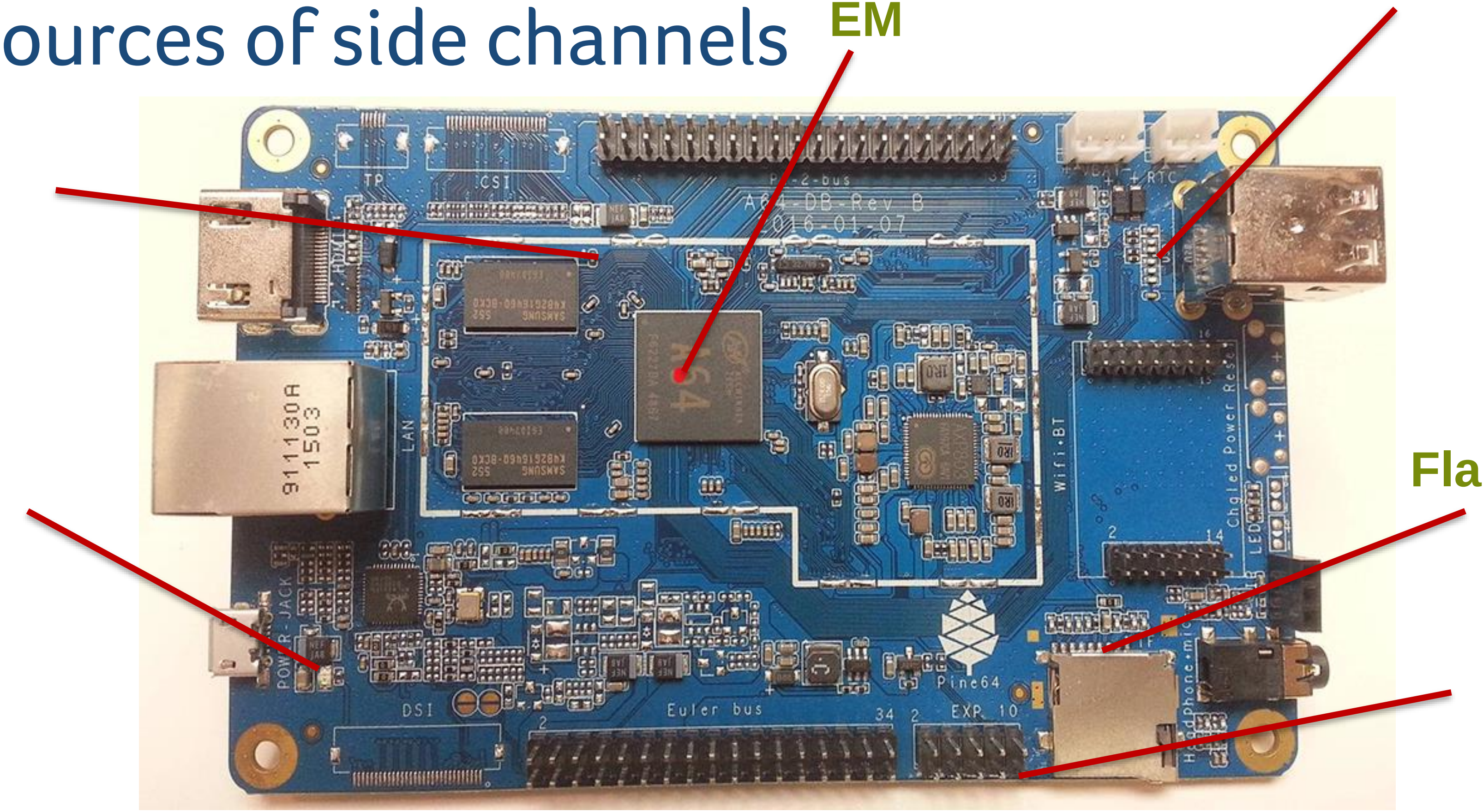


Sources of side channels



EM

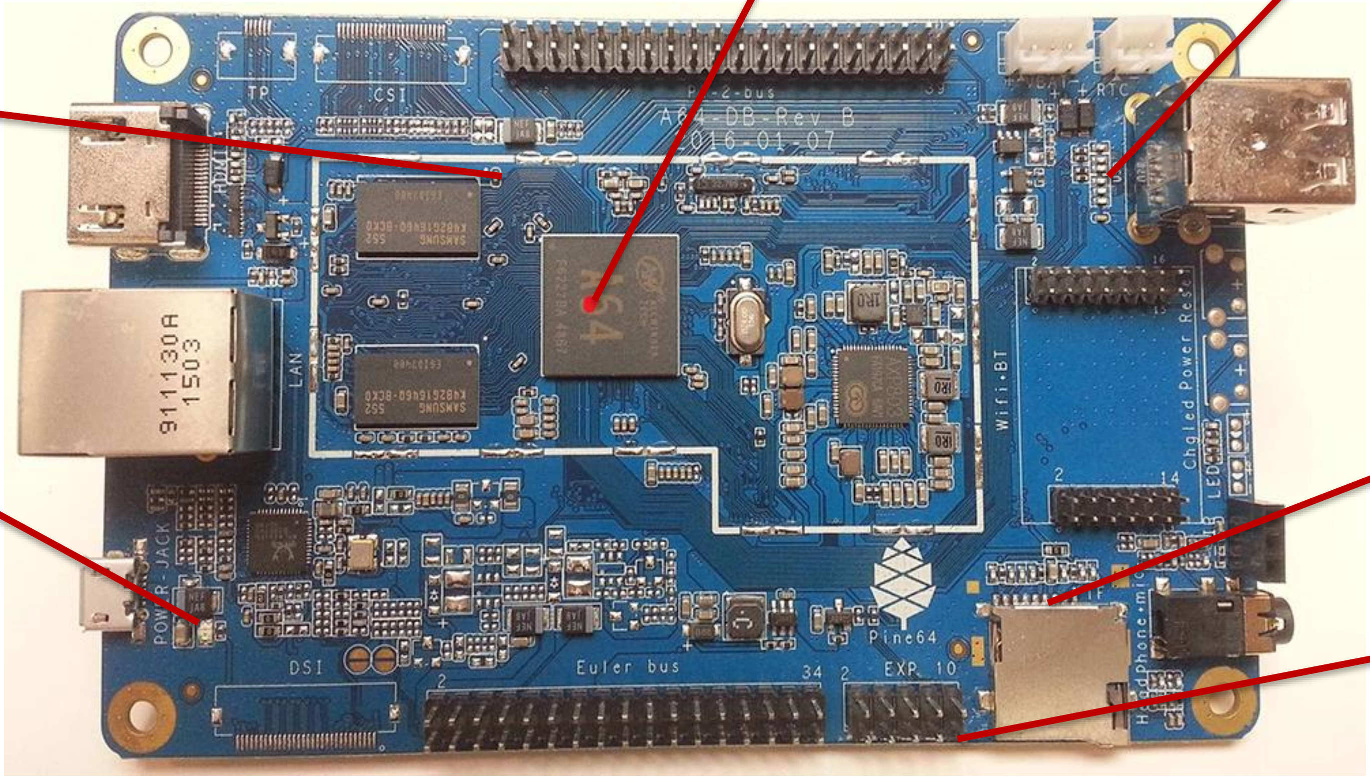
Sources of side channels



EM

Flash

Sources of side channels

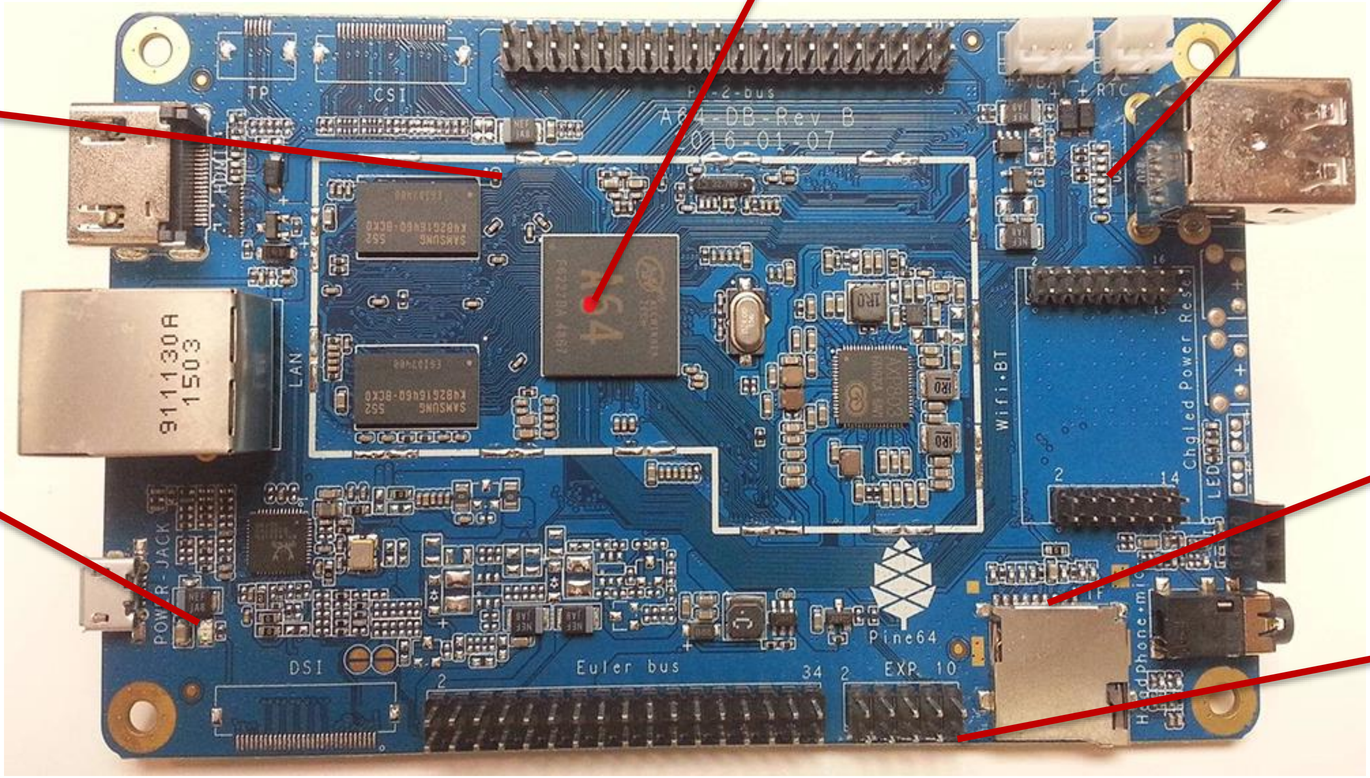


EM

USB

Flash

Sources of side channels



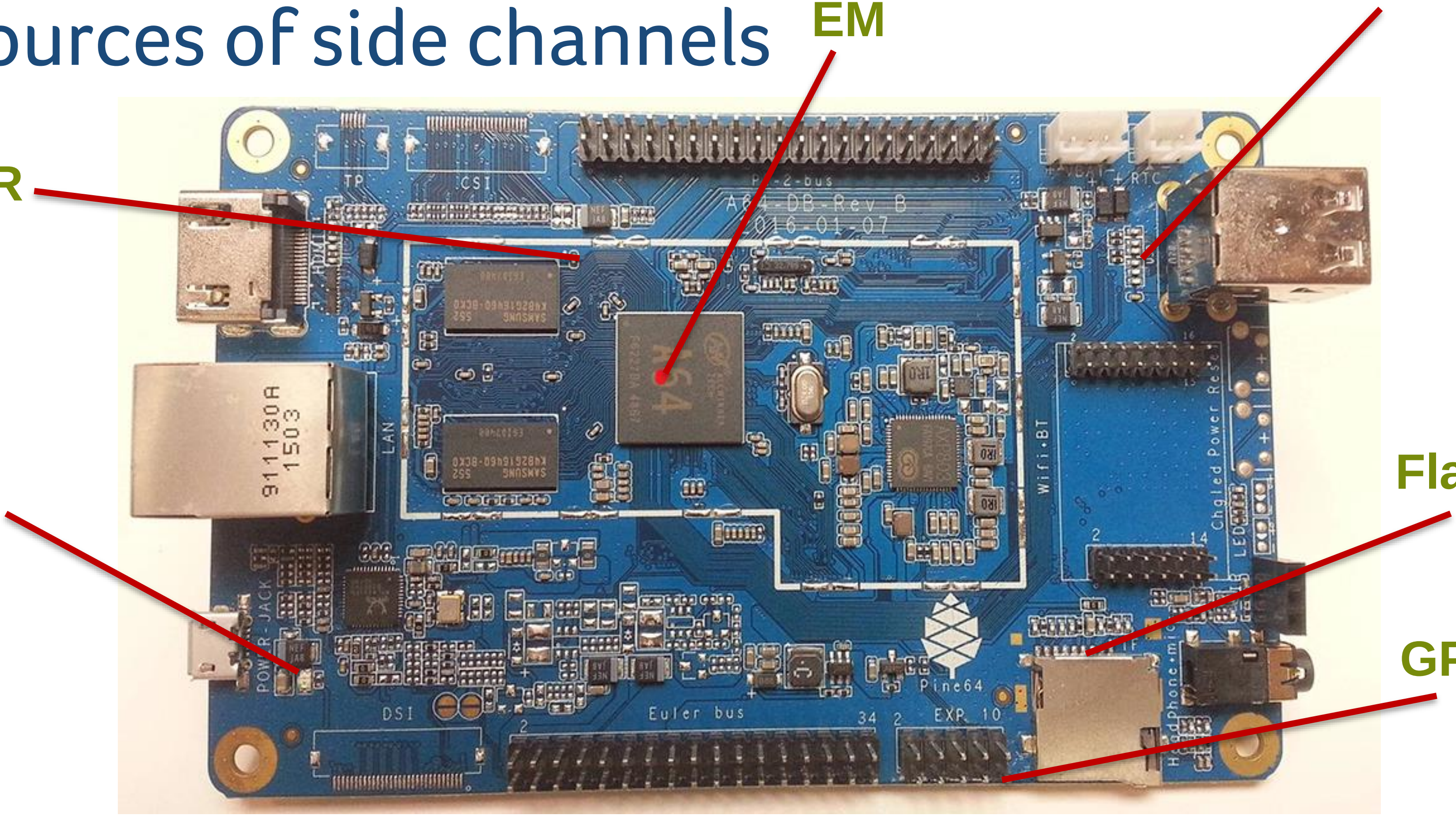
DDR

EM

USB

Flash

Sources of side channels



EM

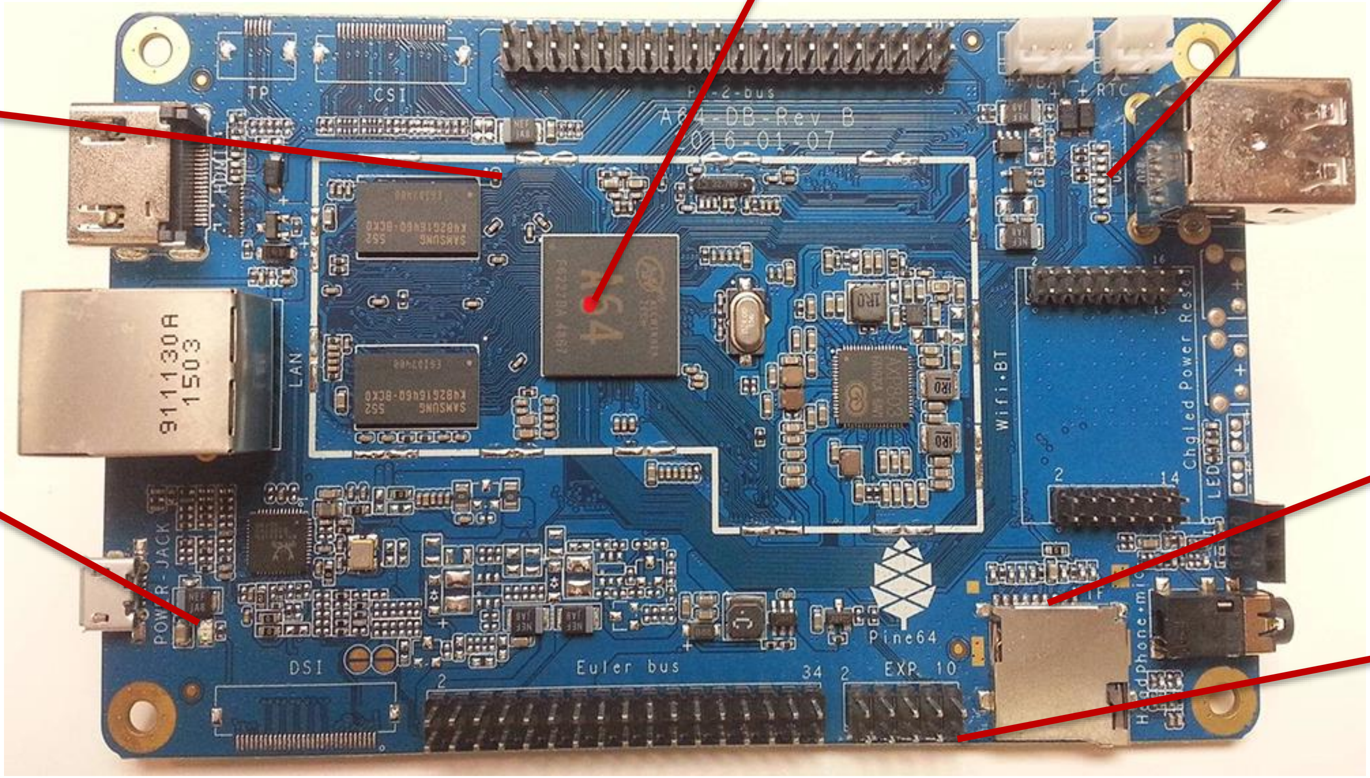
USB

DDR

Flash

GPIO

Sources of side channels



EM

USB

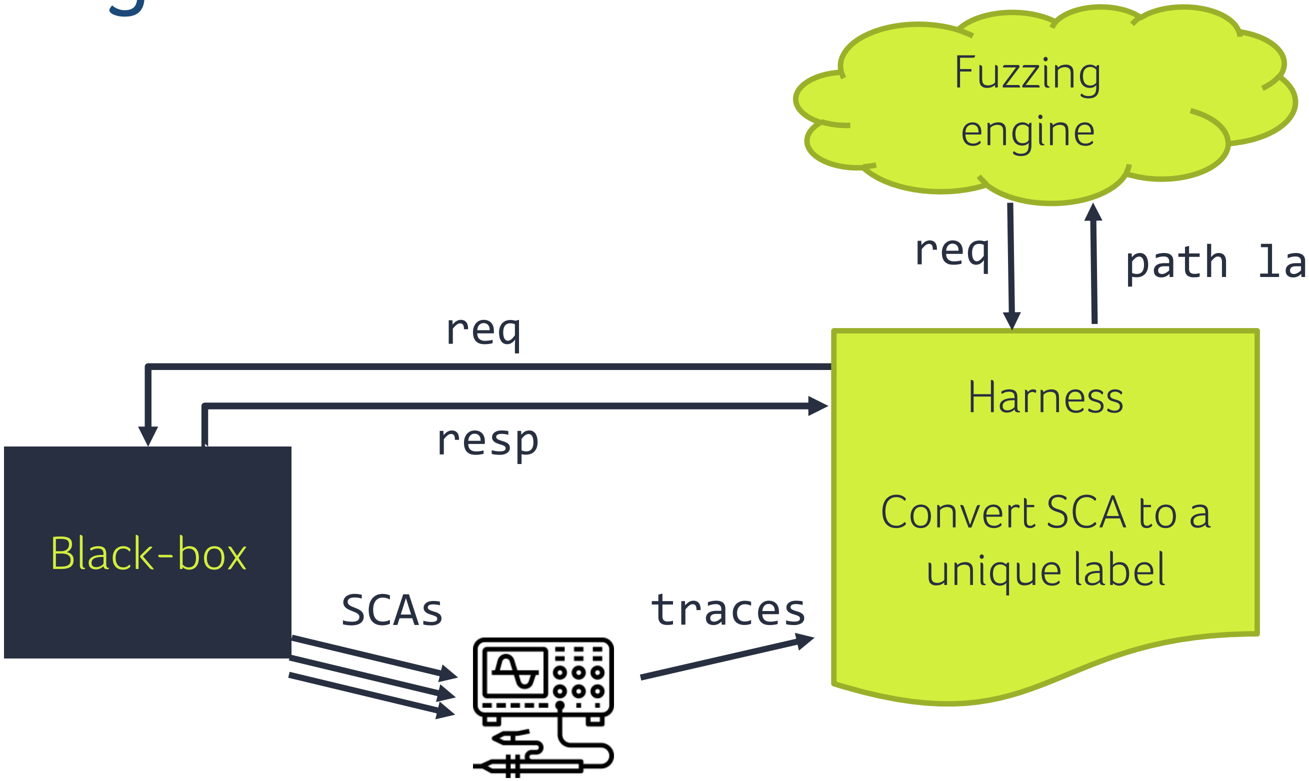
DDR

LED

Flash

GPIO

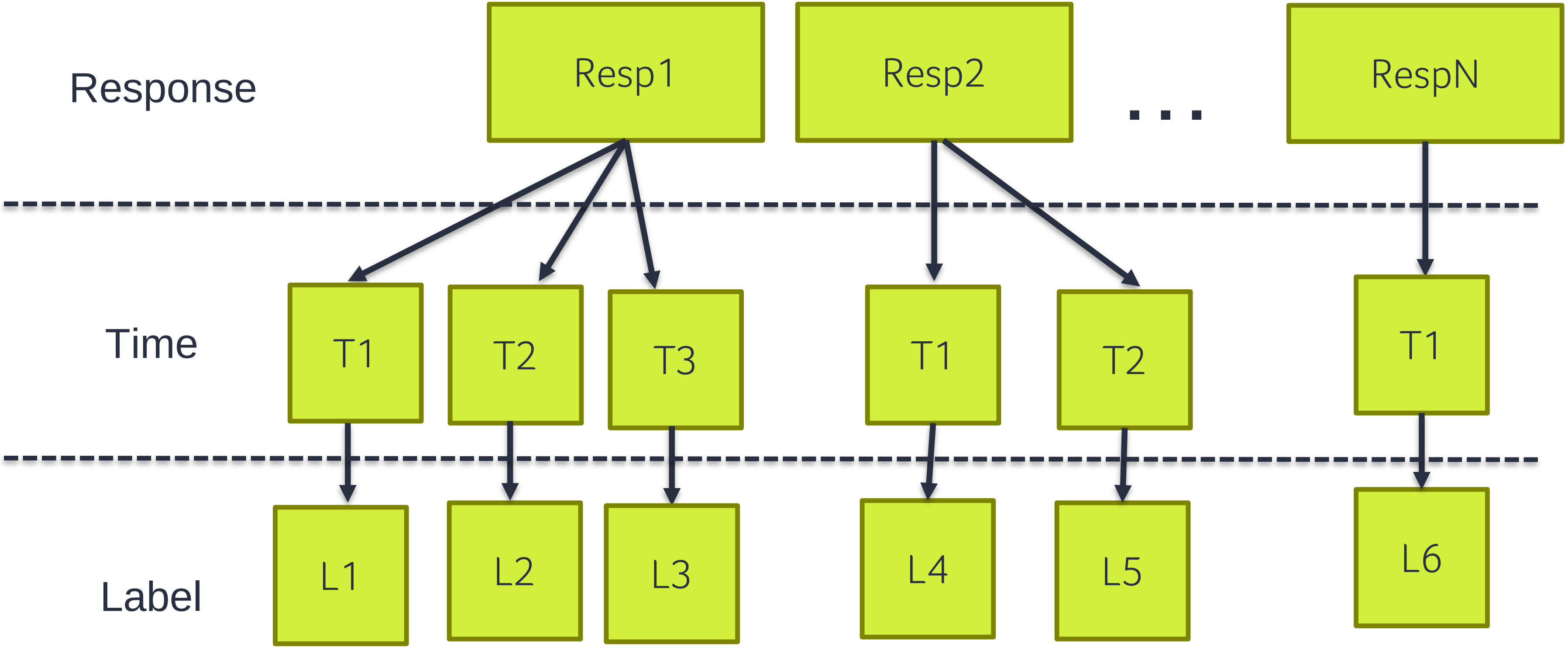
Fuzzing with side channels



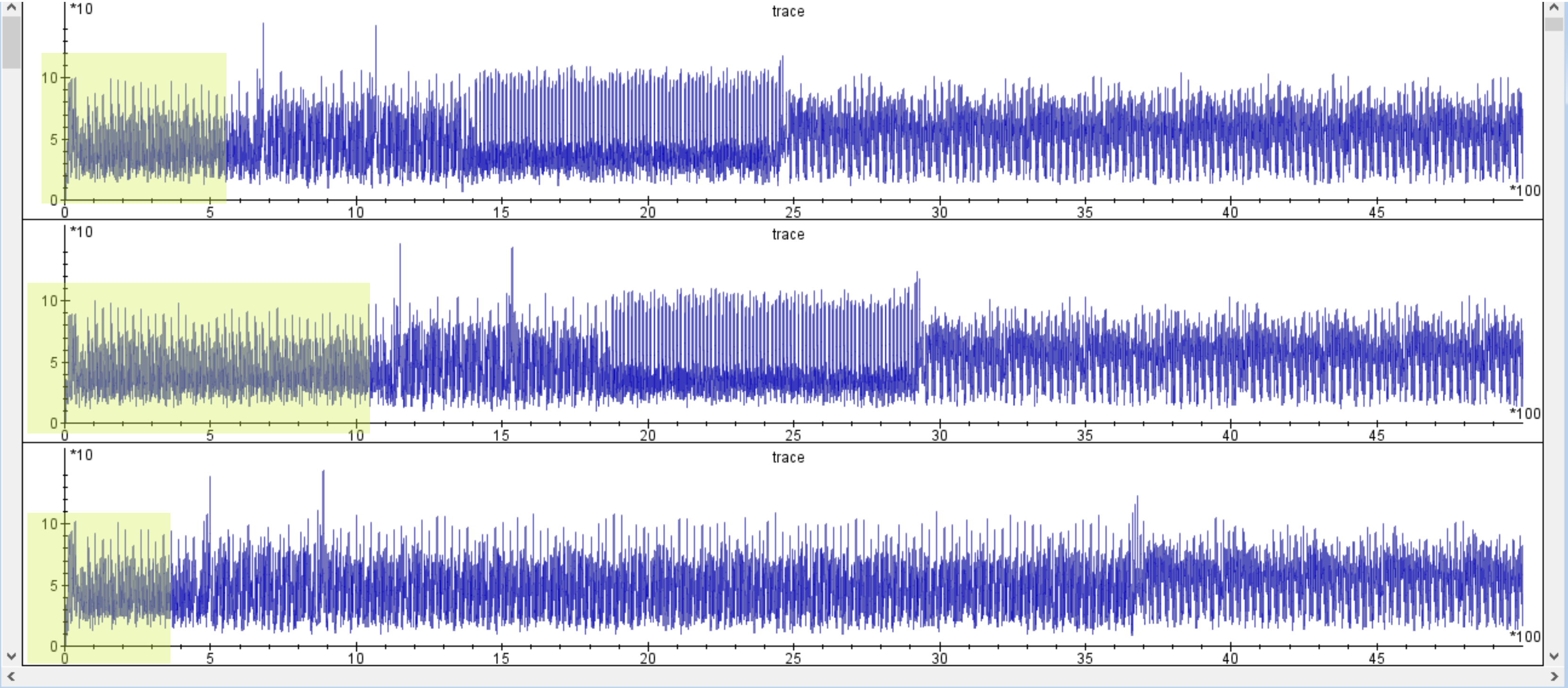
Fuzzing with side channels approach

- Types of side channels:
    - Response data, timing, power trace, EM trace, serial memory access, GPIO activity…
- Hierarchy of sources
    - Not all the SCA data has equal priority of labeling
    - Response → Serial → Timing → Power/EM trace
- Extendable Hierarchical Labeler

Extendable Hierarchical Labeler



Jitter and labelling



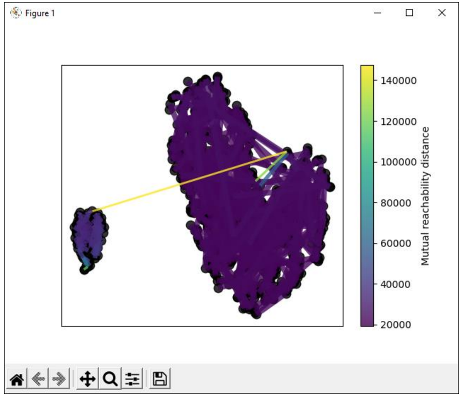
Trace labelling challenges

- Trace labelling
  - We perform clustering
  - The data is noisy
  - Need to cluster traces incrementally
  - Need to do it sufficiently fast

Jitter effects on clustering - HDBSCAN

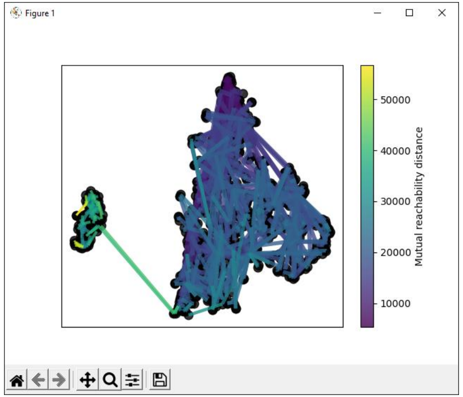
- Synthetic tests of two commands with a different amount of jitter in the signal

0%



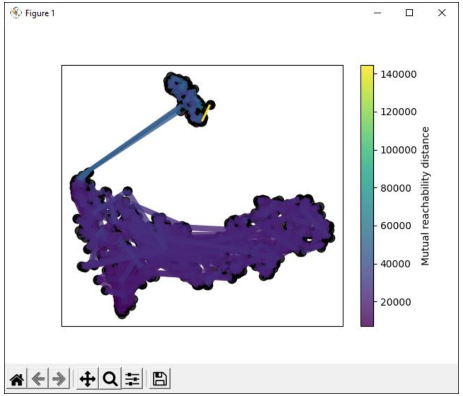
Mislabeled 0.1%

10%



Mislabeled 1.7%

20%

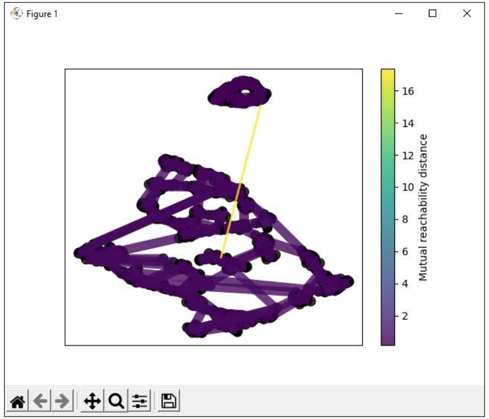


Mislabeled 3.8%

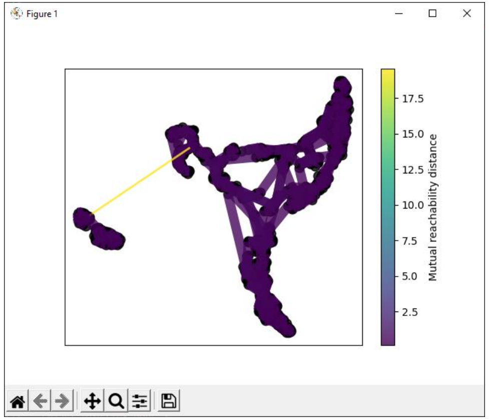
Jitter effects on clustering - UMAP

- **UMAP + HDBSCAN**

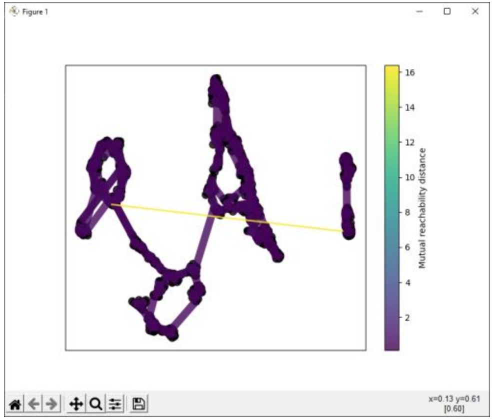
0% **10%** **20%**



Mislabeled 0.1%

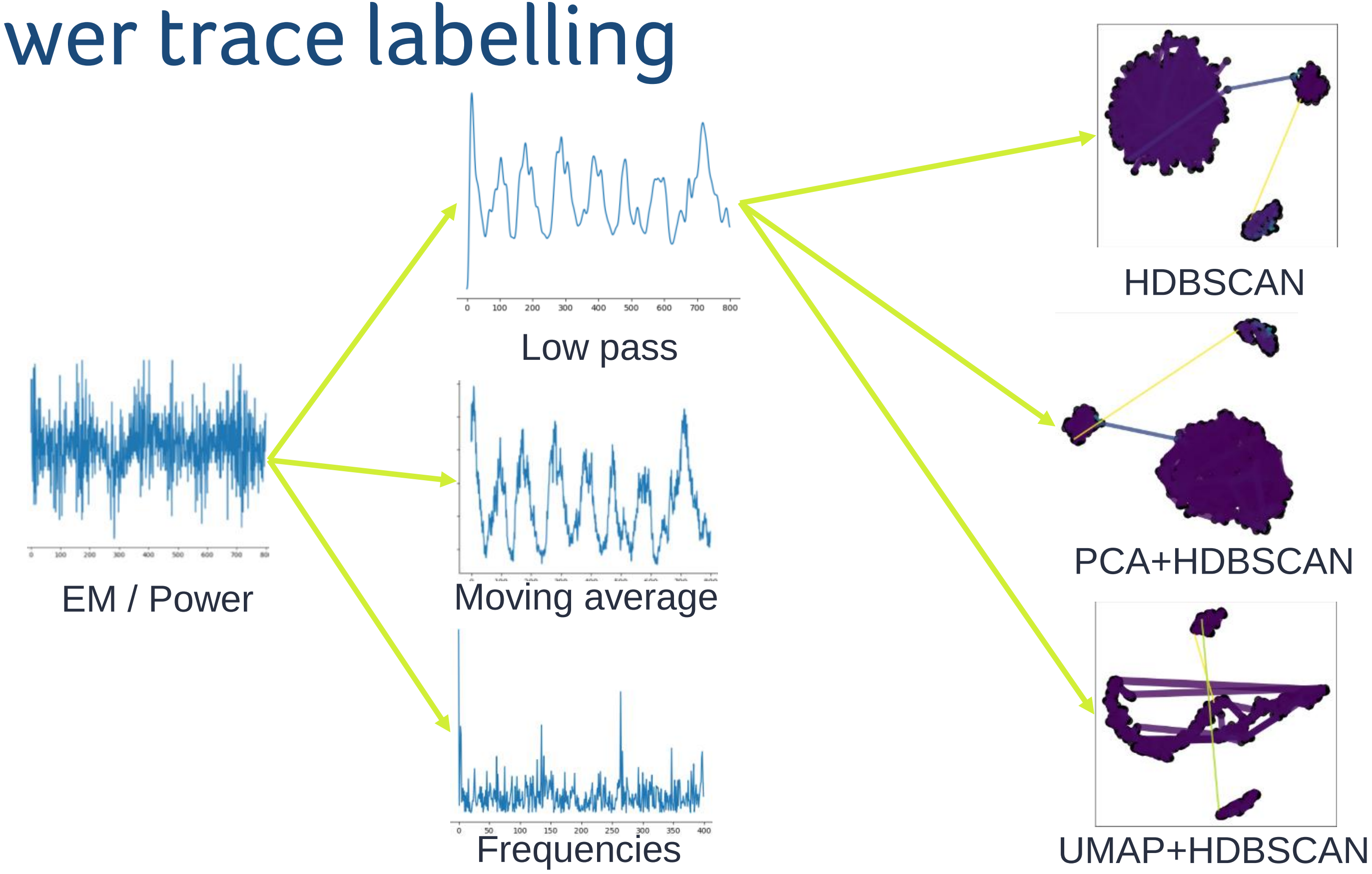


Mislabeled 0.8%



Mislabeled 0.9%

Power trace labelling



EM / Power

Low pass

Moving average

Frequencies

HDBSCAN

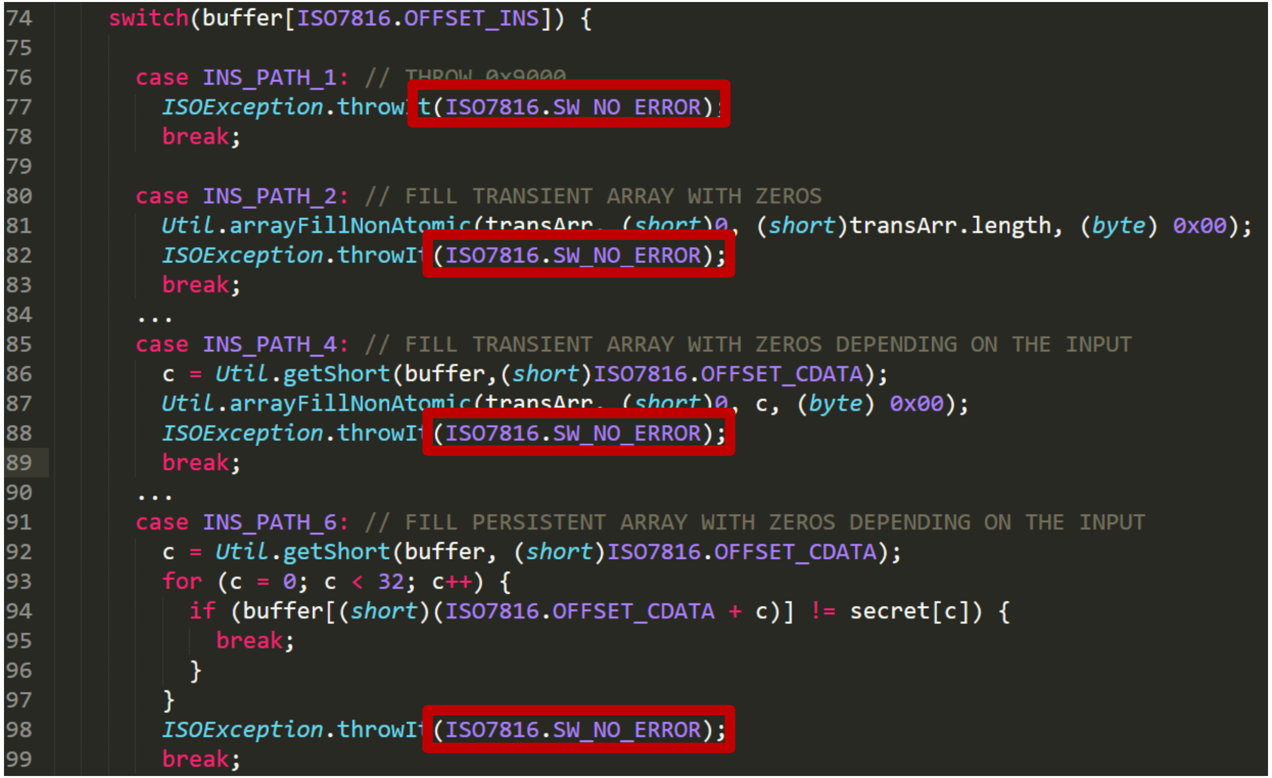
PCA+HDBSCAN

UMAP+HDBSCAN

Smart Card Use case

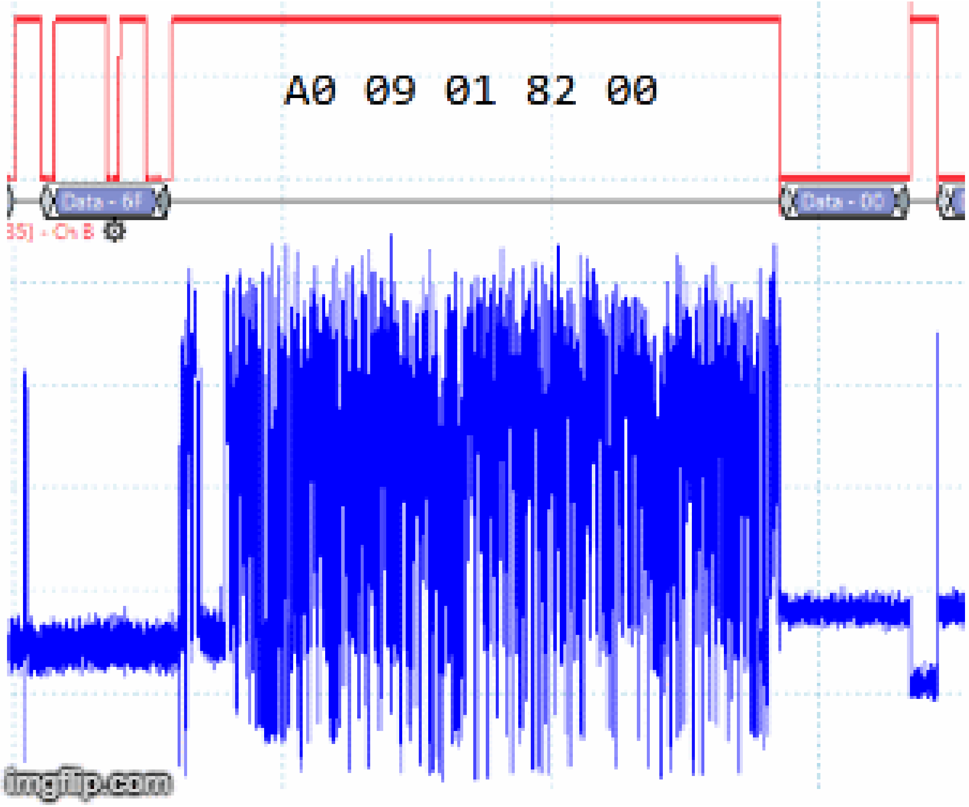
- Apply the fuzzing method to a smart card applet
- Self written code, can assess the coverage
- Only 0x9000 SW is returned, no data

Applet code

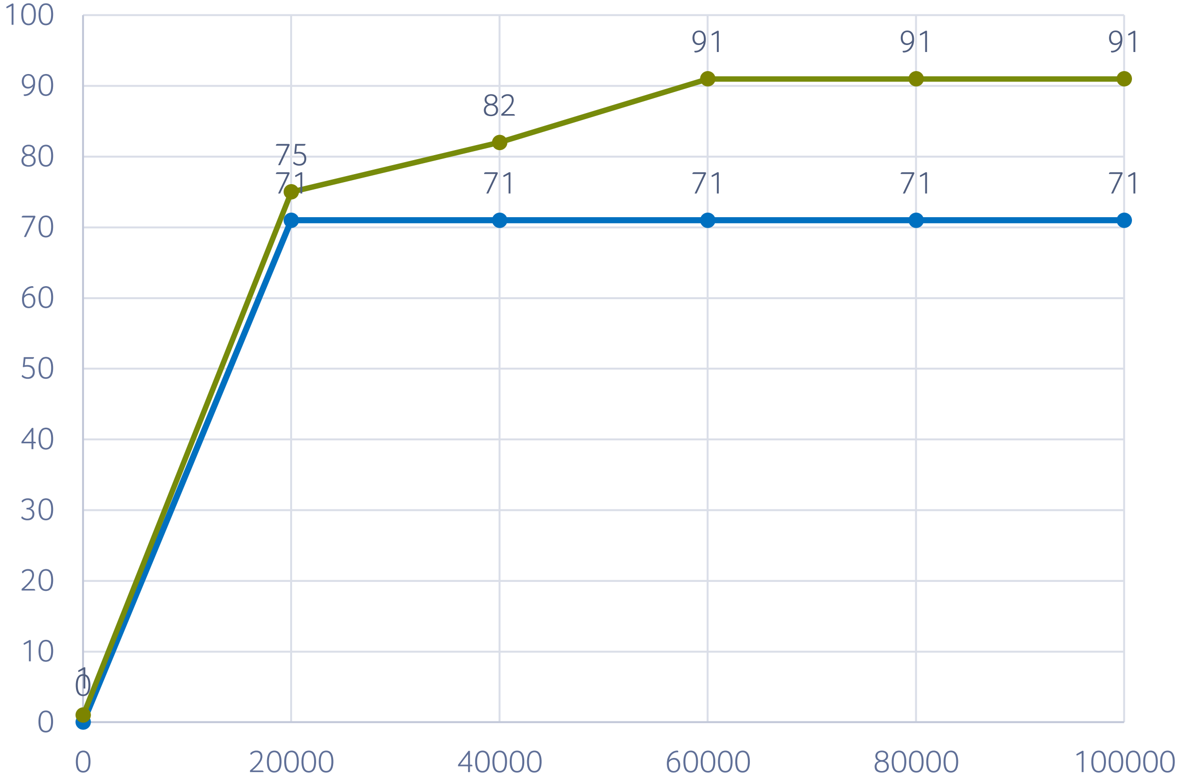


```
switch(buffer[ISO7816.OFFSET_INS]) {

  case INS_PATH_1: // THROW 0x9000
    ISOException.throwIt(ISO7816.SW_NO_ERROR);
    break;

  case INS_PATH_2: // FILL TRANSIENT ARRAY WITH ZEROS
    Util.arrayFillNonAtomic(transArr, (short)0, (short)transArr.length, (byte) 0x00);
    ISOException.throwIt(ISO7816.SW_NO_ERROR);
    break;
  ...
  case INS_PATH_4: // FILL TRANSIENT ARRAY WITH ZEROS DEPENDING ON THE INPUT
    c = Util.getShort(buffer,(short)ISO7816.OFFSET_CDATA);
    Util.arrayFillNonAtomic(transArr, (short)0, c, (byte) 0x00);
    ISOException.throwIt(ISO7816.SW_NO_ERROR);
    break;
  ...
  case INS_PATH_6: // FILL PERSISTENT ARRAY WITH ZEROS DEPENDING ON THE INPUT
    c = Util.getShort(buffer, (short)ISO7816.OFFSET_CDATA);
    for (c = 0; c < 32; c++) {
      if (buffer[(short)(ISO7816.OFFSET_CDATA + c)] != secret[c]) {
        break;
      }
    }
    ISOException.throwIt(ISO7816.SW_NO_ERROR);
    break;
```

Power traces

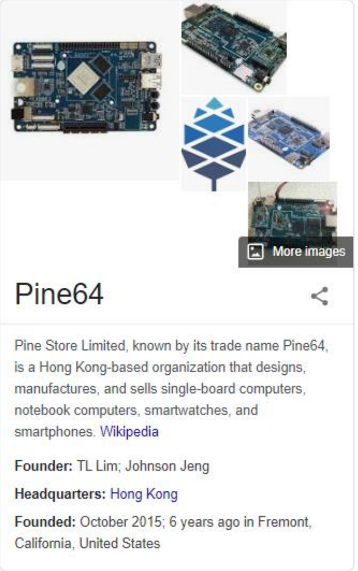


Coverage: Random vs Fuzzer



Smart phone use case

- **Apply the fuzzing method to Uboot of PinePhone**



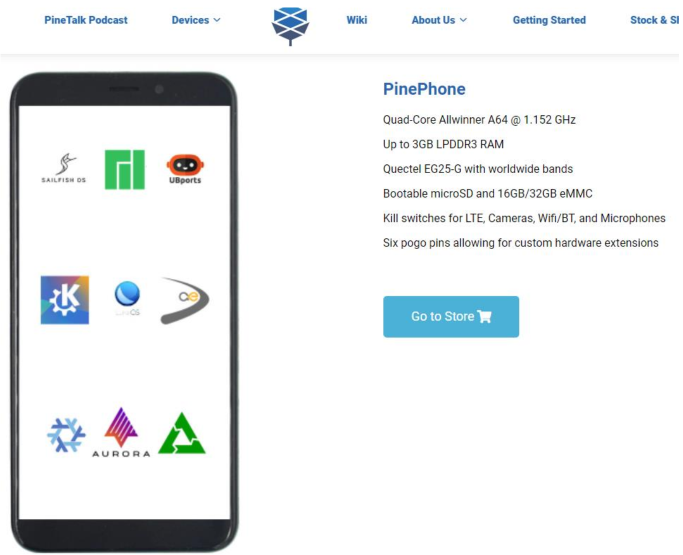
Pine64

Pine Store Limited, known by its trade name Pine64, is a Hong Kong-based organization that designs, manufactures, and sells single-board computers, notebook computers, smartwatches, and smartphones. Wikipedia

Founder: TL Lim; Johnson Jeng

Headquarters: Hong Kong

Founded: October 2015; 6 years ago in Fremont, California, United States



PinePhone

Quad-Core Allwinner A64 @ 1.152 GHz

Up to 3GB LPDDR3 RAM

Quectel EG25-G with worldwide bands

Bootable microSD and 16GB/32GB eMMC

Kill switches for LTE, Cameras, Wifi/BT, and Microphones

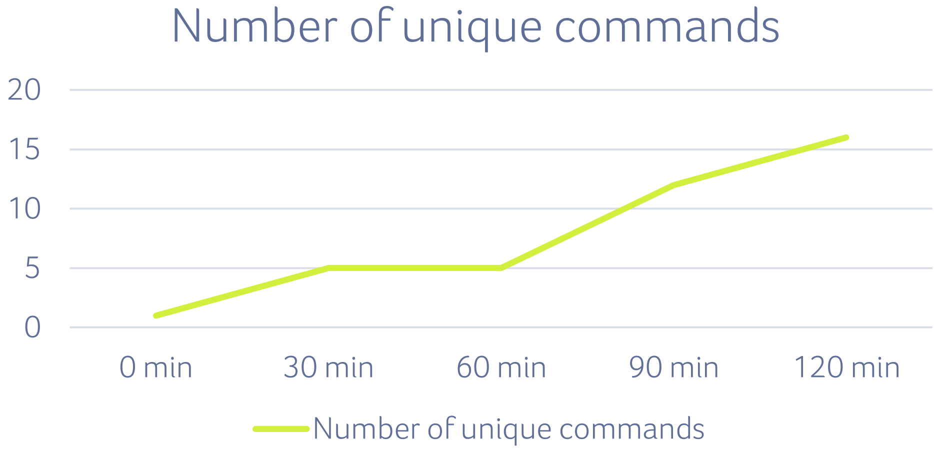
Six pogo pins allowing for custom hardware extensions

Smart phone use case

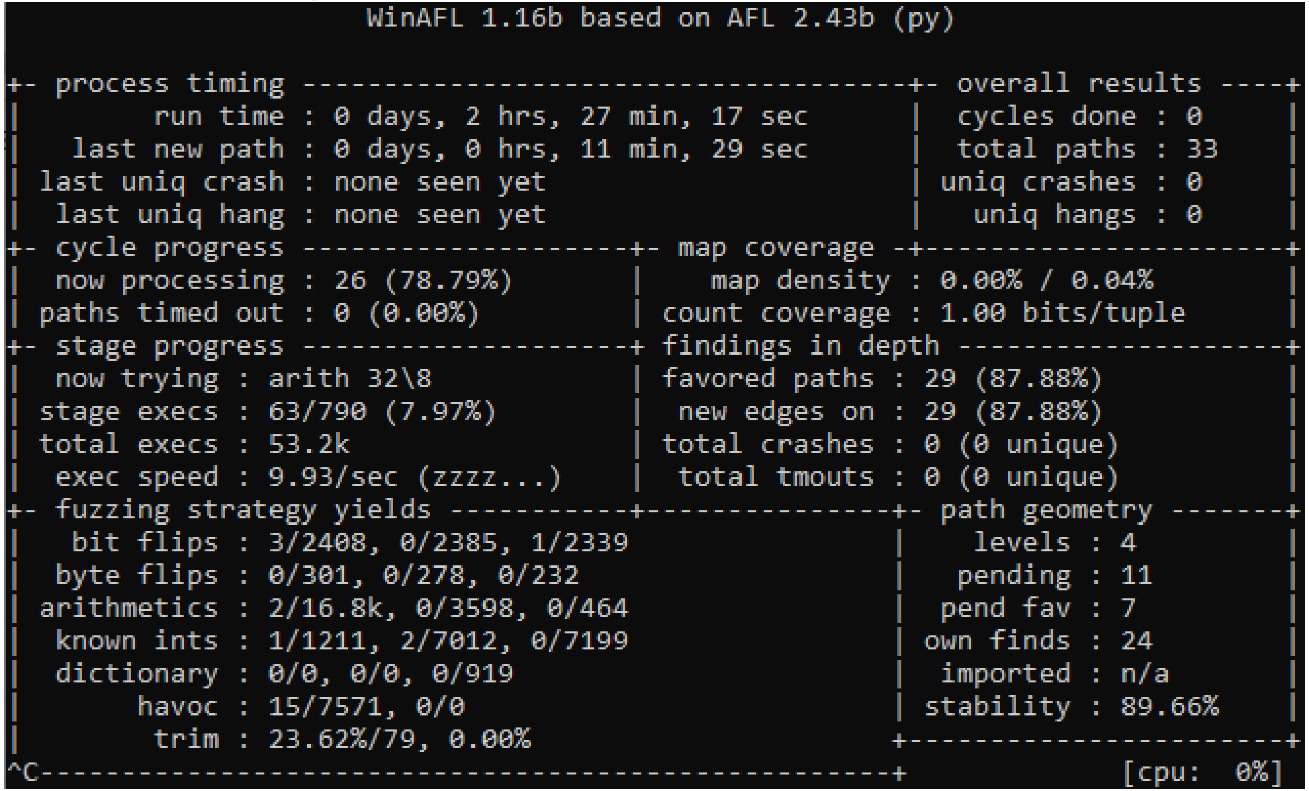
- **Apply the fuzzing method to Uboot of PinePhone**
- **Available docs and source code for verification**
- **Real life, but not the most secure implementation**

No Corpus case

- Started with empty corpus
- responses and timing
- No help to the fuzzer
- Responses are verbose
- 16 CMDs in 2h
- ~6 execs per second



Uboot fuzzing, no corpus case



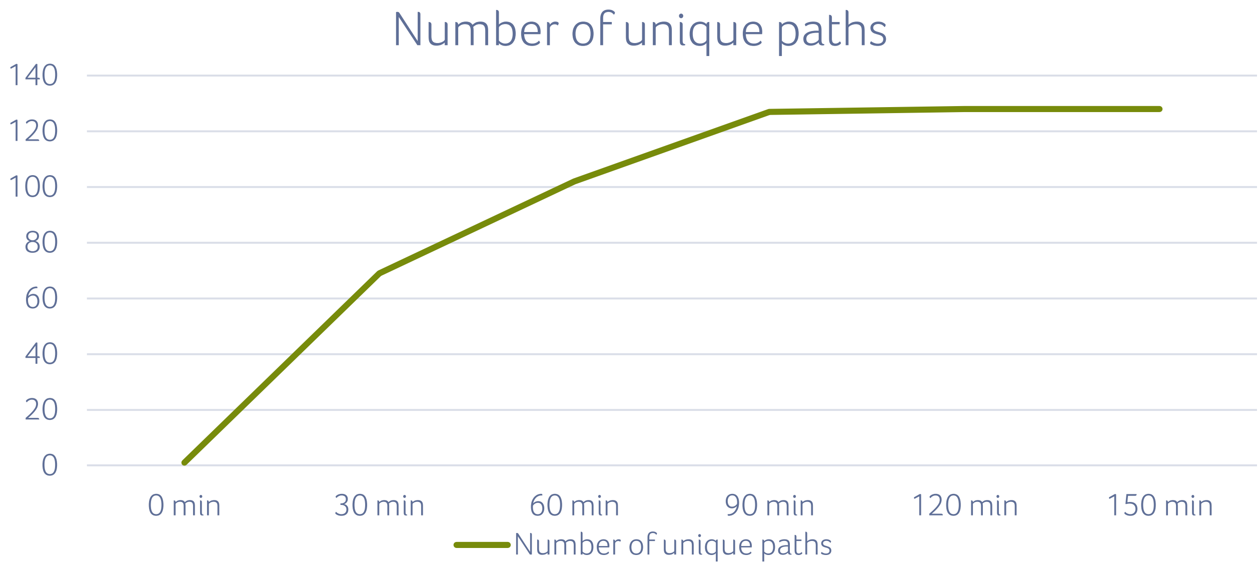
```
                      WinAFL 1.16b based on AFL 2.43b (py)

+- process timing -------------------------------------+- overall results ----+
|        run time : 0 days, 2 hrs, 27 min, 17 sec      |  cycles done : 0     |
|   last new path : 0 days, 0 hrs, 11 min, 29 sec      |  total paths : 33    |
| last uniq crash : none seen yet                      | uniq crashes : 0     |
|  last uniq hang : none seen yet                      |   uniq hangs : 0     |
+- cycle progress --------------------+- map coverage -+----------------------+
|  now processing : 26 (78.79%)       |    map density : 0.00% / 0.04%        |
| paths timed out : 0 (0.00%)         | count coverage : 1.00 bits/tuple      |
+- stage progress --------------------+ findings in depth --------------------+
|  now trying : arith 32\8            | favored paths : 29 (87.88%)           |
| stage execs : 63/790 (7.97%)        |  new edges on : 29 (87.88%)           |
| total execs : 53.2k                 | total crashes : 0 (0 unique)          |
|  exec speed : 9.93/sec (zzzz...)    |  total tmouts : 0 (0 unique)          |
+- fuzzing strategy yields -----------+---------------+- path geometry -------+
|   bit flips : 3/2408, 0/2385, 1/2339                |    levels : 4         |
|  byte flips : 0/301, 0/278, 0/232                   |   pending : 11        |
| arithmetics : 2/16.8k, 0/3598, 0/464                |  pend fav : 7         |
|  known ints : 1/1211, 2/7012, 0/7199                | own finds : 24        |
|  dictionary : 0/0, 0/0, 0/919                       |  imported : n/a       |
|       havoc : 15/7571, 0/0                          | stability : 89.66%    |
|        trim : 23.62%/79, 0.00%                      +-----------------------+
^C----------------------------------------------------+             [cpu:  0%]
```

U-Boot with all the commands in the corpus

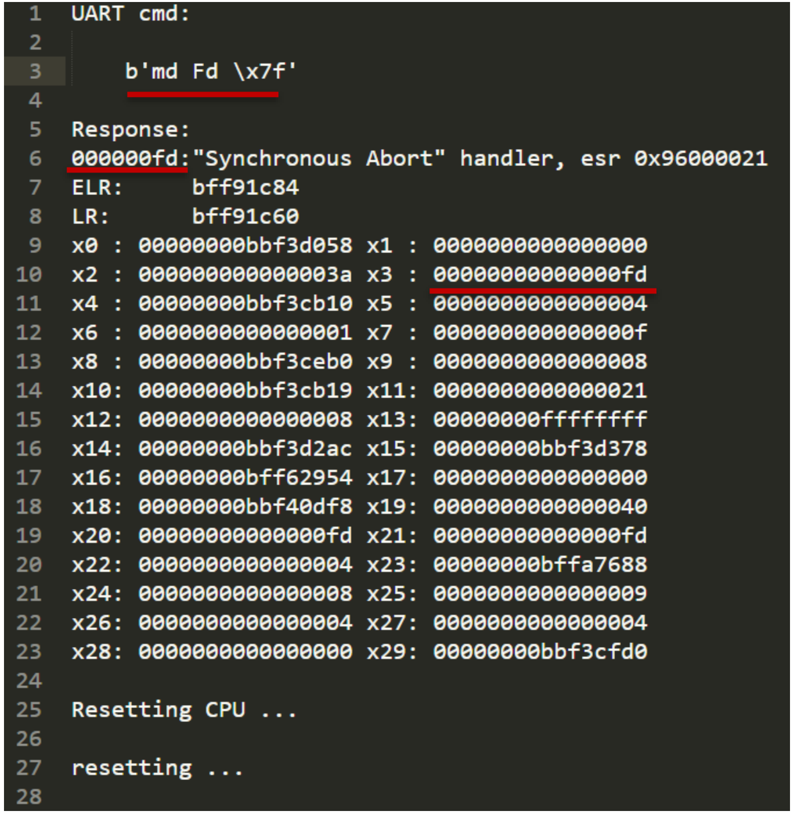
- Started with all of the available commands
- Corpus has correct syntax
- Fuzzer finds commands independently
- Crashes

U-Boot with all the commands in the corpus

- **State explosion**
- **Syntax errors**



U-Boot crashes



```
UART cmd:

    b'md Fd \x7f'

Response:
000000fd:"Synchronous Abort" handler, esr 0x96000021
ELR:     bff91c84
LR:      bff91c60
x0 : 00000000bbf3d058 x1 : 0000000000000000
x2 : 000000000000003a x3 : 00000000000000fd
x4 : 00000000bbf3cb10 x5 : 0000000000000004
x6 : 0000000000000001 x7 : 000000000000000f
x8 : 00000000bbf3ceb0 x9 : 0000000000000008
x10: 00000000bbf3cb19 x11: 0000000000000021
x12: 0000000000000008 x13: 00000000ffffffff
x14: 00000000bbf3d2ac x15: 00000000bbf3d378
x16: 00000000bff62954 x17: 0000000000000000
x18: 00000000bbf40df8 x19: 0000000000000040
x20: 00000000000000fd x21: 00000000000000fd
x22: 0000000000000004 x23: 00000000bffa7688
x24: 0000000000000008 x25: 0000000000000009
x26: 0000000000000004 x27: 0000000000000004
x28: 0000000000000000 x29: 00000000bbf3cfd0

Resetting CPU ...

resetting ...

```

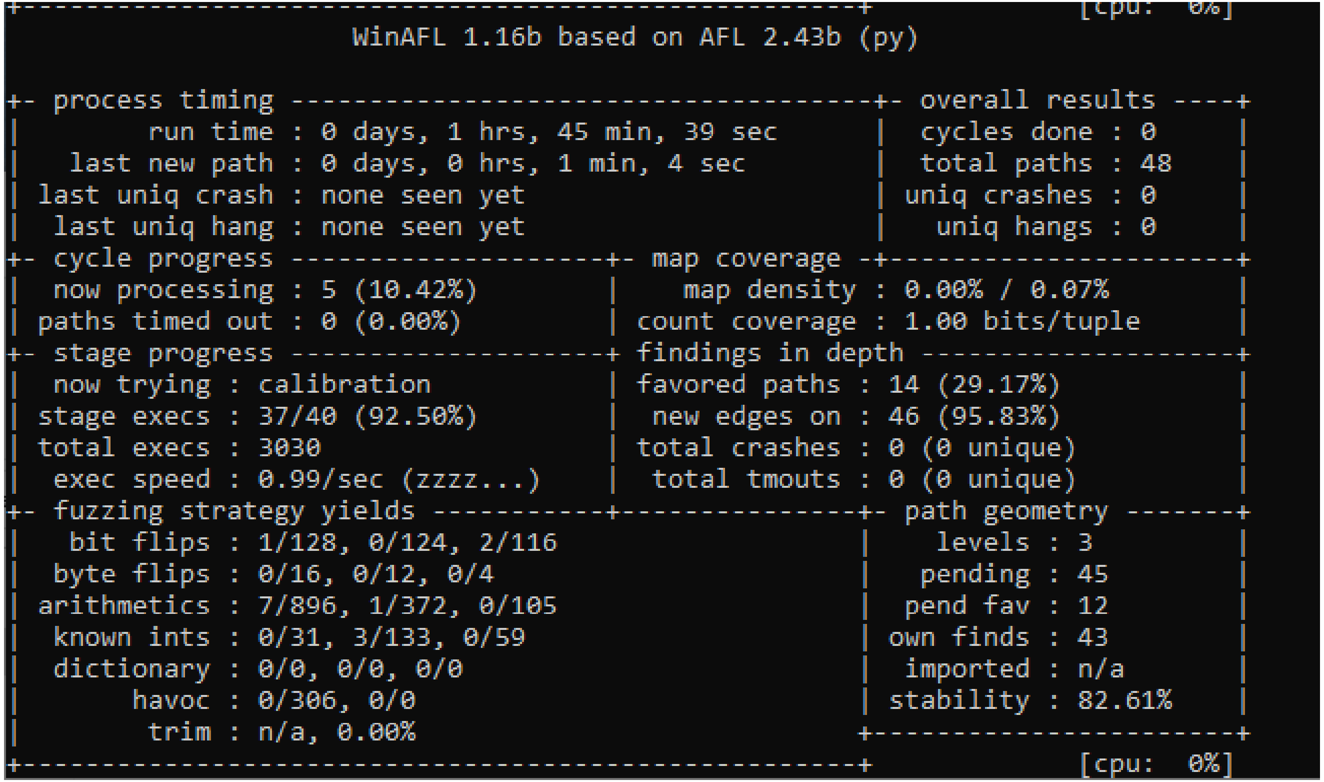
CRC32 command use case

- `crc32 0x40000000 0x4000`
- `CRC 40000000 … 40003fff => 0xbf13d15a`

- Initial run produced 16 different labels
- The returned data was different

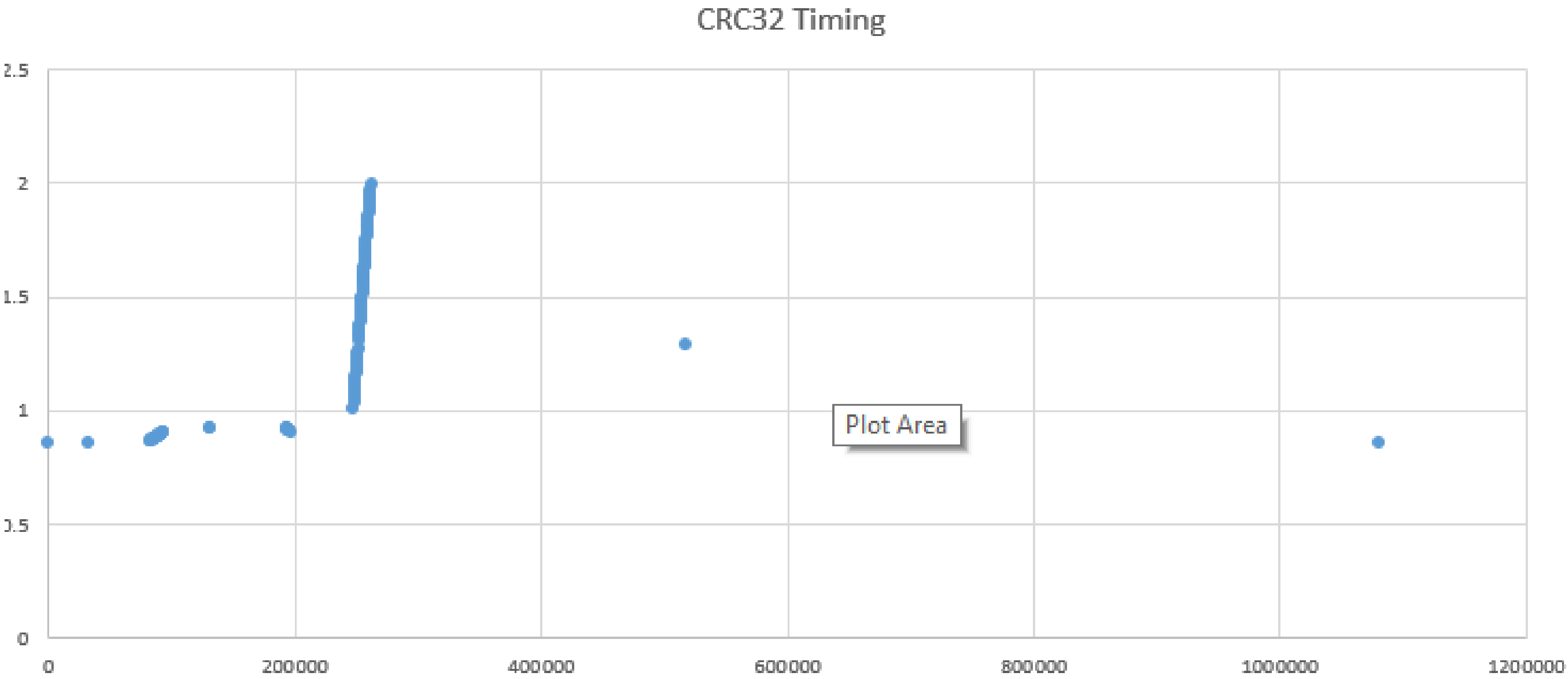
- … but the timing also differs

CRC32 command use case

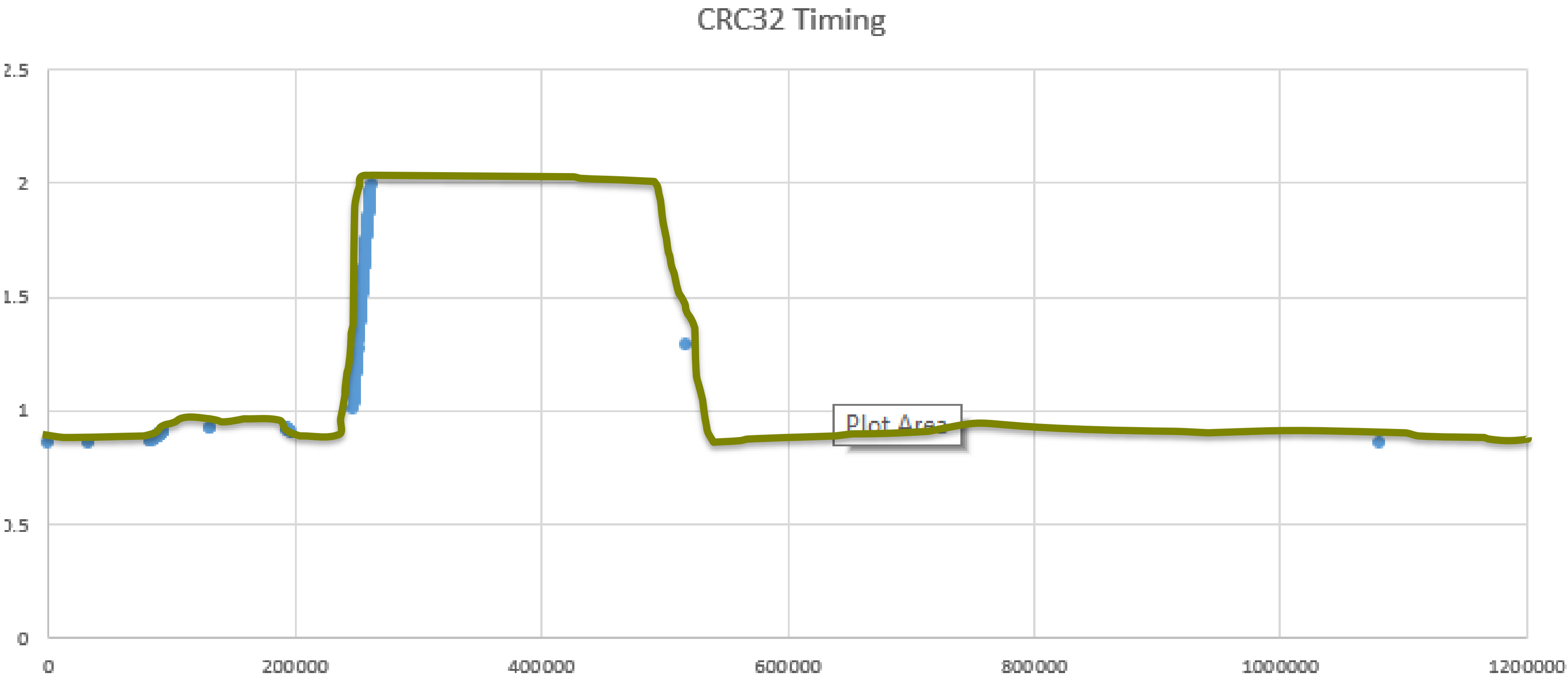


```
+-------------------------------------------------------+             [cpu:  0%]
                       WinAFL 1.16b based on AFL 2.43b (py)

+- process timing -------------------------------------+- overall results ----+
|        run time : 0 days, 1 hrs, 45 min, 39 sec      |  cycles done : 0     |
|   last new path : 0 days, 0 hrs, 1 min, 4 sec        |  total paths : 48    |
| last uniq crash : none seen yet                      | uniq crashes : 0     |
|  last uniq hang : none seen yet                      |   uniq hangs : 0     |
+- cycle progress ---------------------+- map coverage -+----------------------+
|  now processing : 5 (10.42%)         |    map density : 0.00% / 0.07%        |
| paths timed out : 0 (0.00%)          | count coverage : 1.00 bits/tuple      |
+- stage progress ---------------------+ findings in depth --------------------+
|  now trying : calibration            | favored paths : 14 (29.17%)           |
| stage execs : 37/40 (92.50%)         |  new edges on : 46 (95.83%)           |
| total execs : 3030                   | total crashes : 0 (0 unique)          |
|  exec speed : 0.99/sec (zzzz...)     |  total tmouts : 0 (0 unique)          |
+- fuzzing strategy yields -----------+---------------+- path geometry -------+
|   bit flips : 1/128, 0/124, 2/116                    |    levels : 3         |
|  byte flips : 0/16, 0/12, 0/4                        |   pending : 45        |
| arithmetics : 7/896, 1/372, 0/105                    |  pend fav : 12        |
|  known ints : 0/31, 3/133, 0/59                      | own finds : 43        |
|  dictionary : 0/0, 0/0, 0/0                          |  imported : n/a       |
|       havoc : 0/306, 0/0                             | stability : 82.61%    |
|        trim : n/a, 0.00%                             +-----------------------+
+-----------------------------------------------------+             [cpu:  0%]
```

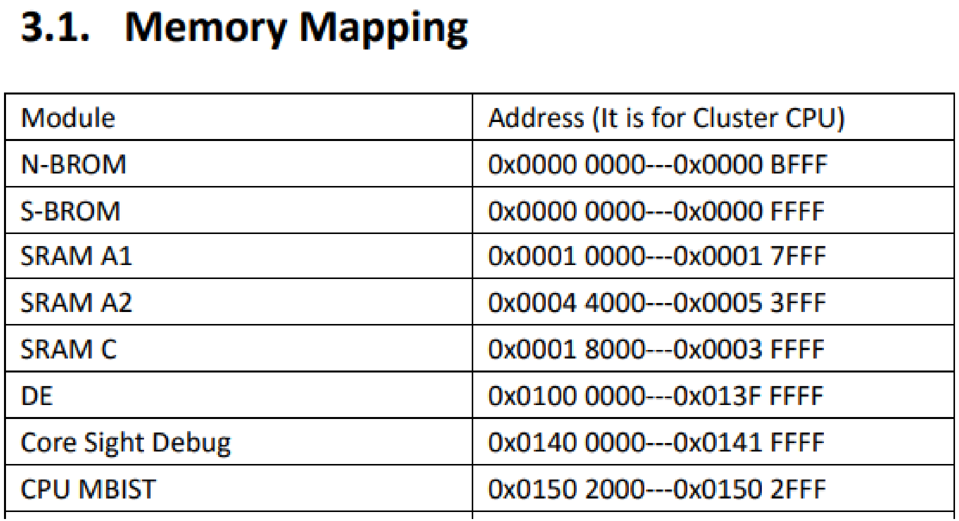
CRC32 command use case



CRC32 command use case

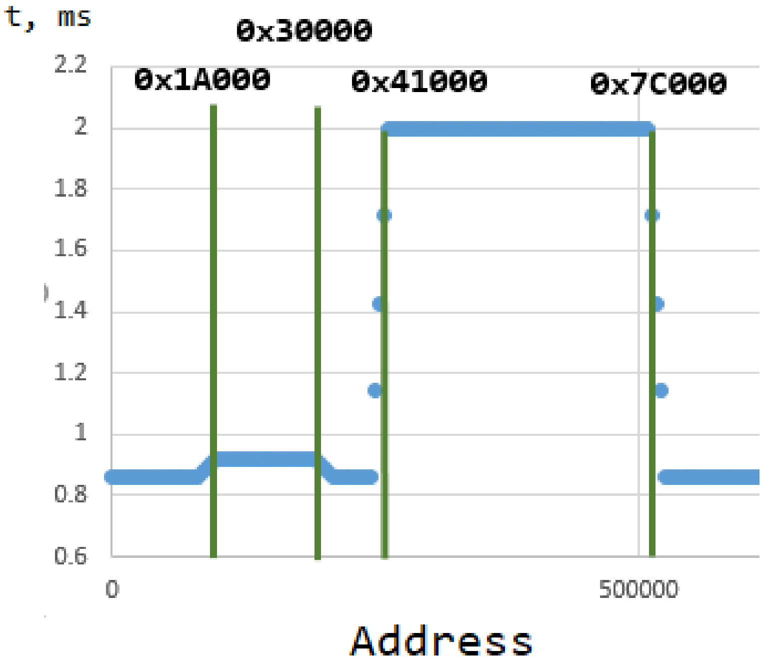


CRC32 command use case

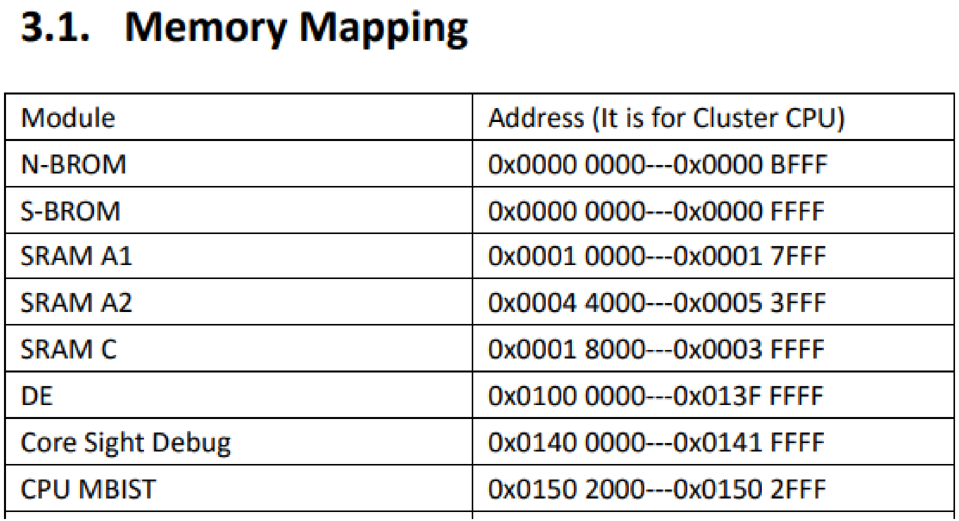


3.1. Memory Mapping

Module	Address (It is for Cluster CPU)
N-BROM	0x0000 0000---0x0000 BFFF
S-BROM	0x0000 0000---0x0000 FFFF
SRAM A1	0x0001 0000---0x0001 7FFF
SRAM A2	0x0004 4000---0x0005 3FFF
SRAM C	0x0001 8000---0x0003 FFFF
DE	0x0100 0000---0x013F FFFF
Core Sight Debug	0x0140 0000---0x0141 FFFF
CPU MBIST	0x0150 2000---0x0150 2FFF

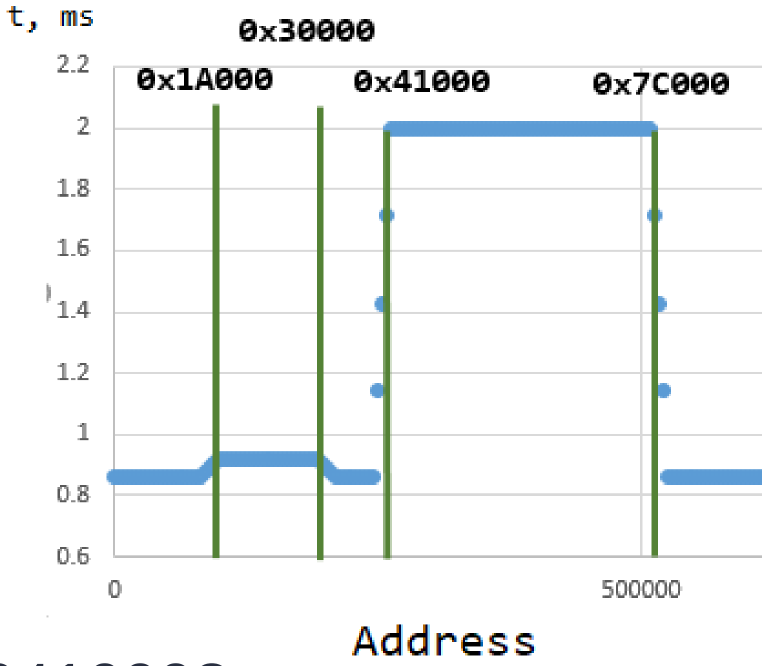


CRC32 command use case



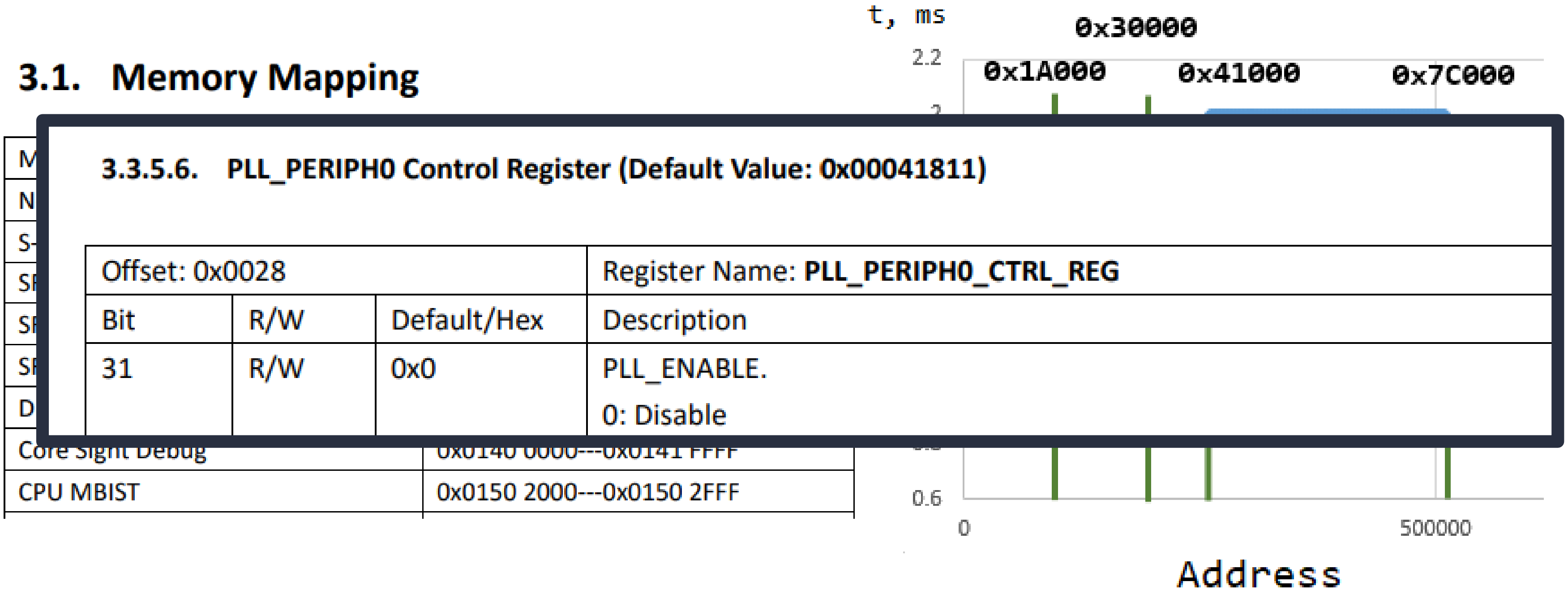
3.1. Memory Mapping

Module	Address (It is for Cluster CPU)
N-BROM	0x0000 0000---0x0000 BFFF
S-BROM	0x0000 0000---0x0000 FFFF
SRAM A1	0x0001 0000---0x0001 7FFF
SRAM A2	0x0004 4000---0x0005 3FFF
SRAM C	0x0001 8000---0x0003 FFFF
DE	0x0100 0000---0x013F FFFF
Core Sight Debug	0x0140 0000---0x0141 FFFF
CPU MBIST	0x0150 2000---0x0150 2FFF



What is at 0x00041000?

CRC32 command use case



3.1. Memory Mapping

M	
N	
S-	
SP	
SP	
SP	
D	
Core Sight Debug	0x0140 0000---0x0141 FFFF
CPU MBIST	0x0150 2000---0x0150 2FFF

3.3.5.6. PLL_PERIPH0 Control Register (Default Value: 0x00041811)

Offset: 0x0028			Register Name: **PLL_PERIPH0_CTRL_REG**
Bit	R/W	Default/Hex	Description
31	R/W	0x0	PLL_ENABLE. 0: Disable

CRC32 command use case

- CRC computation of 0x1000 bytes from different locations:

Location	CRC command timing (len = 0x1000)
BROM	38 us
SRAM A1	38 us
SRAM A2	322 us
SRAM C	56 us
DDR	<29 us

Takeaways

- Coverage tracking for black box targets is possible

- Limited performance requires good corpus and syntax

- The approach can detect not only different SW execution paths, but also different HW

Sergei Volokitin
Senior Security Analyst
✉ sergei@riscure.com

riscure

Challenge your security