OverTime:

Remote Timing Attacks against IoT devices

hardwear.io USA 2019

Cristofaro Mune
(c.mune@pulse-sec.com)
@pulsoid

An interesting code

- A *strcmp()* classic implementation
- Taken from open-source project uClibc-ng
- Full source code available

Vulnerable?

```c
int strcmp (const char *p1, const char *p2)
{
  register const unsigned char *s1 = (const unsigned char *) p1;
  register const unsigned char *s2 = (const unsigned char *) p2;
  unsigned reg_char c1, c2;

  do
    {
      c1 = (unsigned char) *s1++;
      c2 = (unsigned char) *s2++;
      if (c1 == '\0')
        return c1 - c2;

    }
  while (c1 == c2);

  return c1 - c2;
}
```

Early termination

Timing attack

https://elixir.bootlin.com/uclibc-ng/v1.0.31/source/libc/string/generic/strcmp.c

Timing: Wrong → Shorter

```c
int strcmp (const char *p1, const char *p2)
{
  register const unsigned char *s1 = (const unsigned char *) p1;
  register const unsigned char *s2 = (const unsigned char *) p2;
  unsigned reg_char c1, c2;

  do
    {
      c1 = (unsigned char) *s1++;
      c2 = (unsigned char) *s2++;
      if (c1 == '\0')
        return c1 - c2;

    }
  while (c1 == c2);

  return c1 - c2;
}
```

Wrong character

https://elixir.bootlin.com/uclibc-ng/v1.0.31/source/libc/string/generic/strcmp.c

Timing: Right → Longer

```c
int strcmp (const char *p1, const char *p2)
{
  register const unsigned char *s1 = (const unsigned char *) p1;
  register const unsigned char *s2 = (const unsigned char *) p2;
  unsigned reg_char c1, c2;

  do
    {
      c1 = (unsigned char) *s1++;
      c2 = (unsigned char) *s2++;
      if (c1 == '\0')
        return c1 - c2;

    }
  while (c1 == c2);

  return c1 - c2;
}
```

Right character

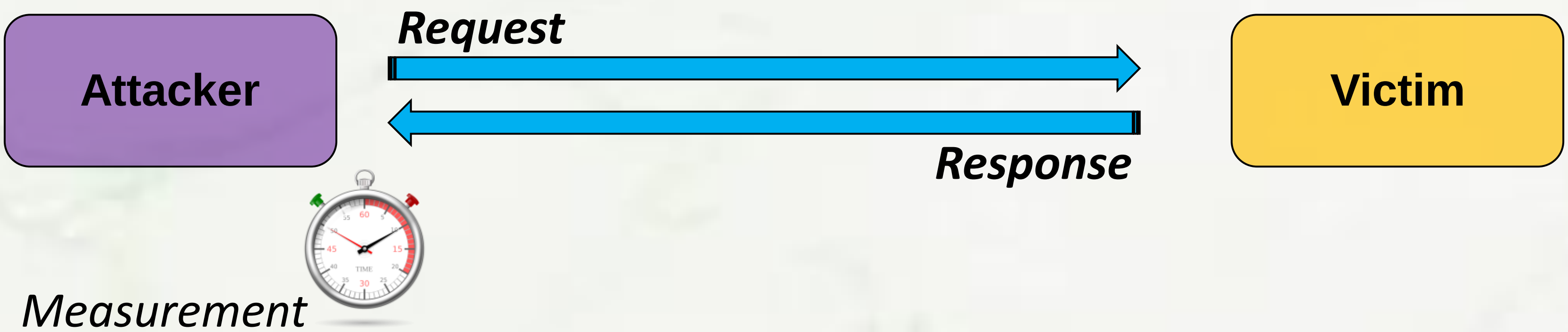
Con

https://elixir.bootlin.com/uclibc-ng/v1.0.31/source/libc/string/generic/strcmp.c

Non-time constant SW

- Timing differences may:
  - Leak information on non-observable processes
  - Be used as oracles for testing assumptions/models:
    - E.g. black-box testing
- If:
  - *present on an observable channel*
  - *They can be measured:*
    - With sufficient precision and accuracy

Attack: Basic idea



- Collect time measurements for the first character:
  - Multiple request for each candidate
- Analyze results by candidate
  - Choose an estimator
  - Set thresholds
- Distinguish **one candidate** distribution *from* ***all*** *the others*
- Go to the next character

Previous research

- Relevant application:
  - Cryptographic key extraction
  - SQL injection
  - Remote password guessing
  - Cache timing attacks *(NetSpectre)*

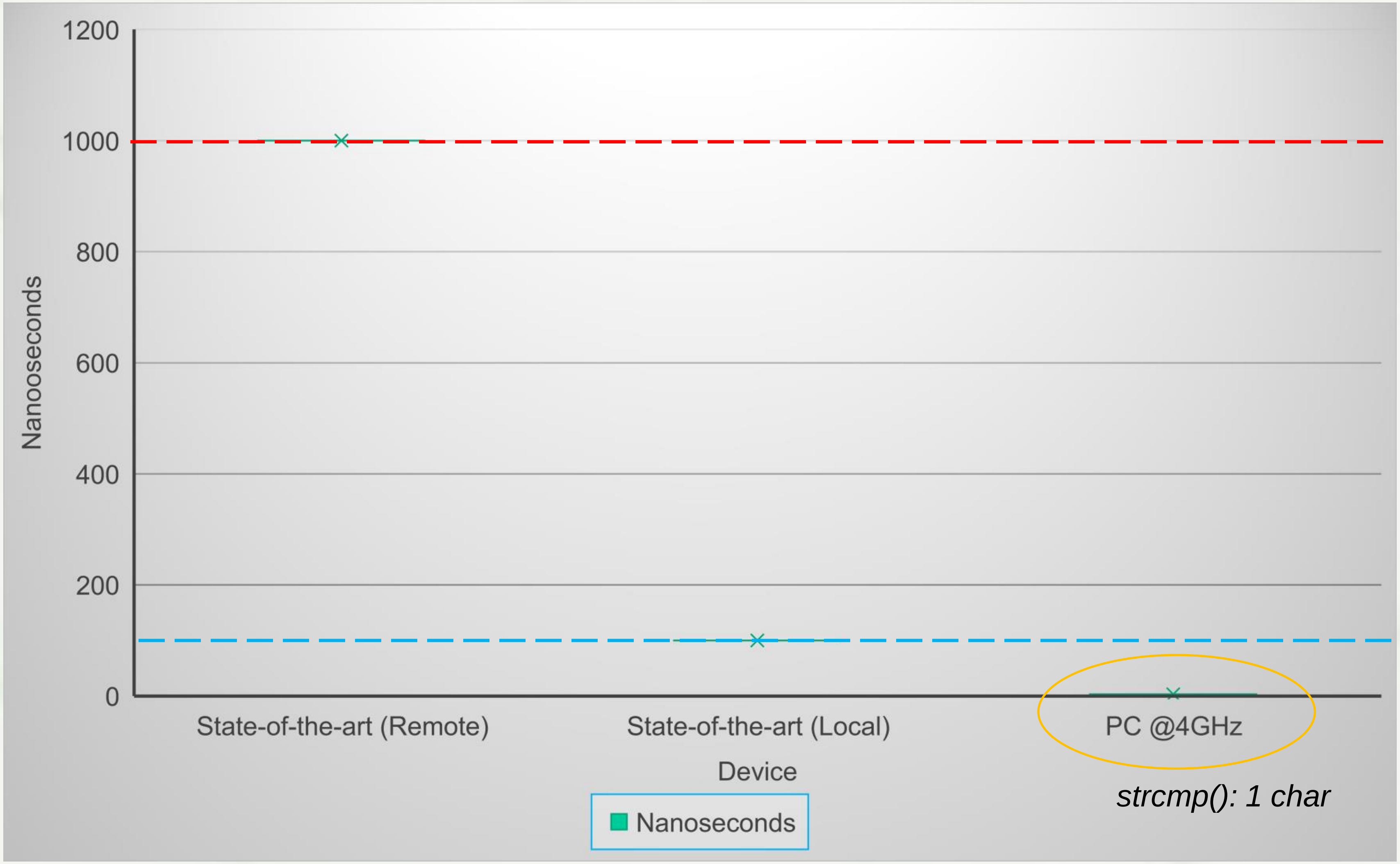
- *State-of-the art precision:*
  - ***100ns (LAN)***
  - ***1us (remotely)***

…focus on fast remote servers…

Remote attack strcmp(): feasible?

- **Fast PC:**
  - 4GHz clock
  - 1 clock tick = 250 picoseconds (10^-12)
- We focus on *strcmp()*:
  - Worst case scenario for a timing attack
  - i.e. If attack can detect one byte difference remotely… it can work on anything
- One single *strcmp()* loop *(1 char)*:
  - 13 instructions (x86)
  - ~3 ns
    - Assuming 1 instruction/clock_cycle

Magnitude orders



Impractical

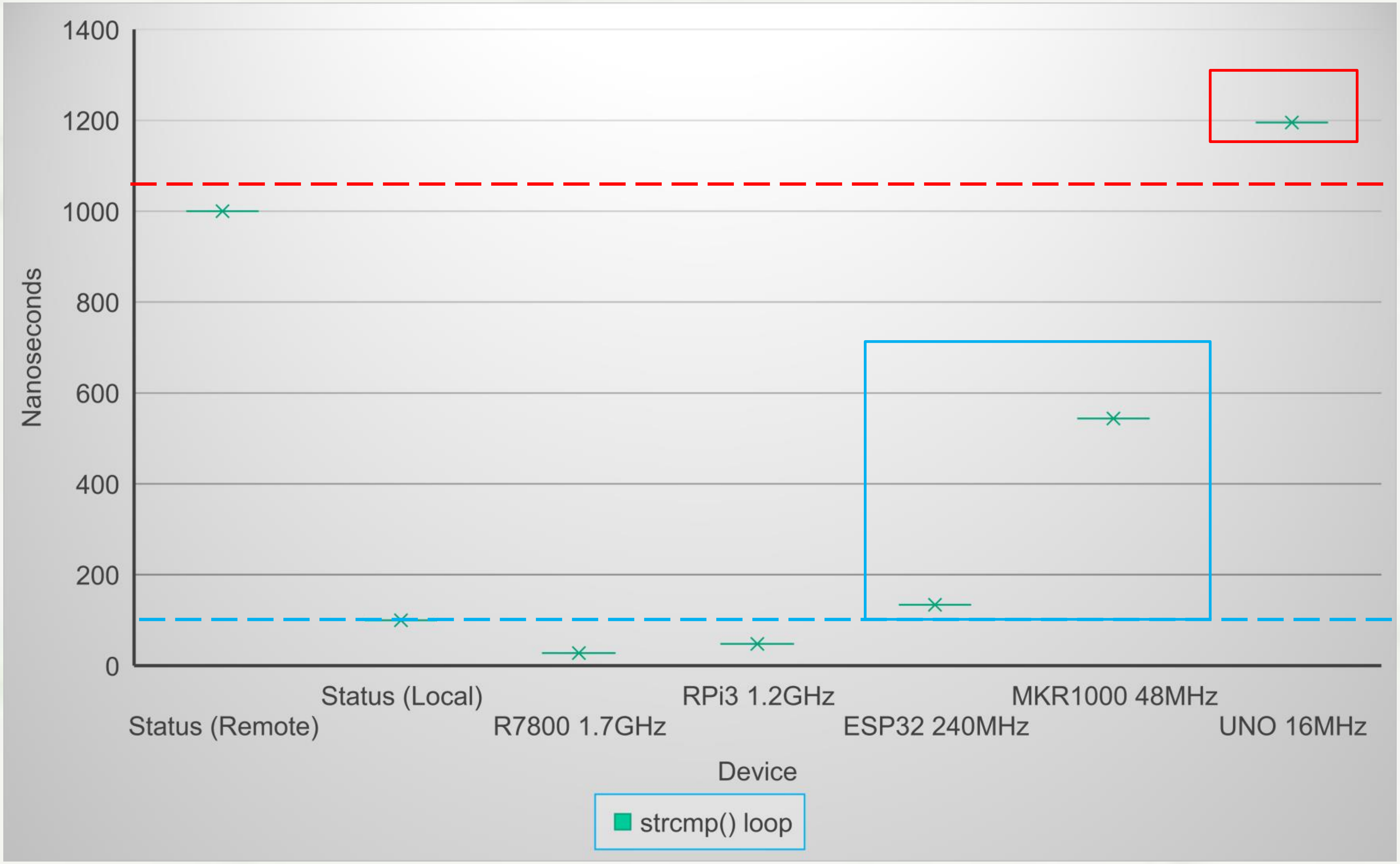
- String comparison:
  - *“**Impractical** in the vast majority of cases” 2015 – Morgan & Morgan*
  - Remote servers with *fast CPUs*

- ***But…IoT systems clocks are much slower!***

that differ in the first character vs. strings that differ only at the 10th character. This indicates that timing attacks on regular string comparison have to be assumed feasible for any embedded system.*

** 2014 – Mayer, Sandin – “Time Trial”*

Embedded devices: a very different outlook



An old attack…in a new context…

- Modern IoT devices:
  - Slower clocks
  - Fast network interfaces:
    - E.g. Ethernet 100 Mbit
- Single *strcmp()* loop within range of remote measurability
- **Older devices may be even slower!**
  - E.g. 2-16 MHz
  - May get network connected

For new impacts!

Critical Infrastructures

Industrial IoT

Physical Security

Slow clock Connected

Devices

Smart Cities

ICS/SCADA

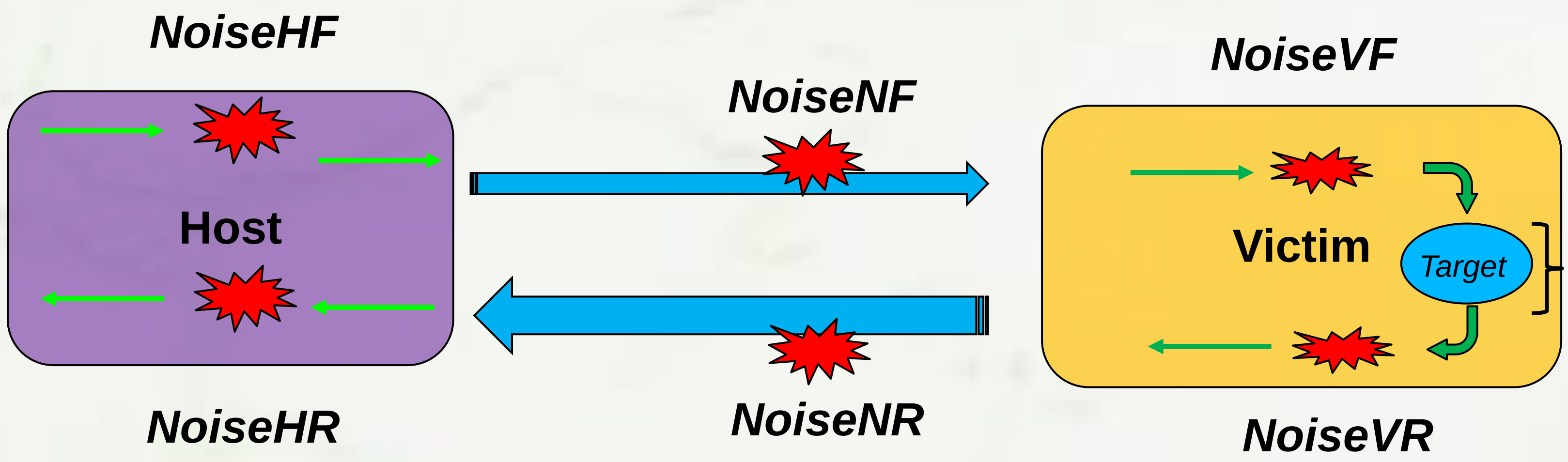
Smart homes

Confidential

Challenges

- Target *connectivity*:
  - May not be sufficiently fast for reasonable attacks
- *Acquisition* noise:
  - *Network*:
    - Route changes, Buffering, QoS, other Delays
  - *Target*:
    - Scheduling, Bandwidth saturation, Frequency scaling, Clock drifts
  - *Host* (Attacker):
    - Same as Target
- *Analysis* noise:
  - Preprocessing induced artifacts, Wrong estimators, Bias,…

Acquisition Noise



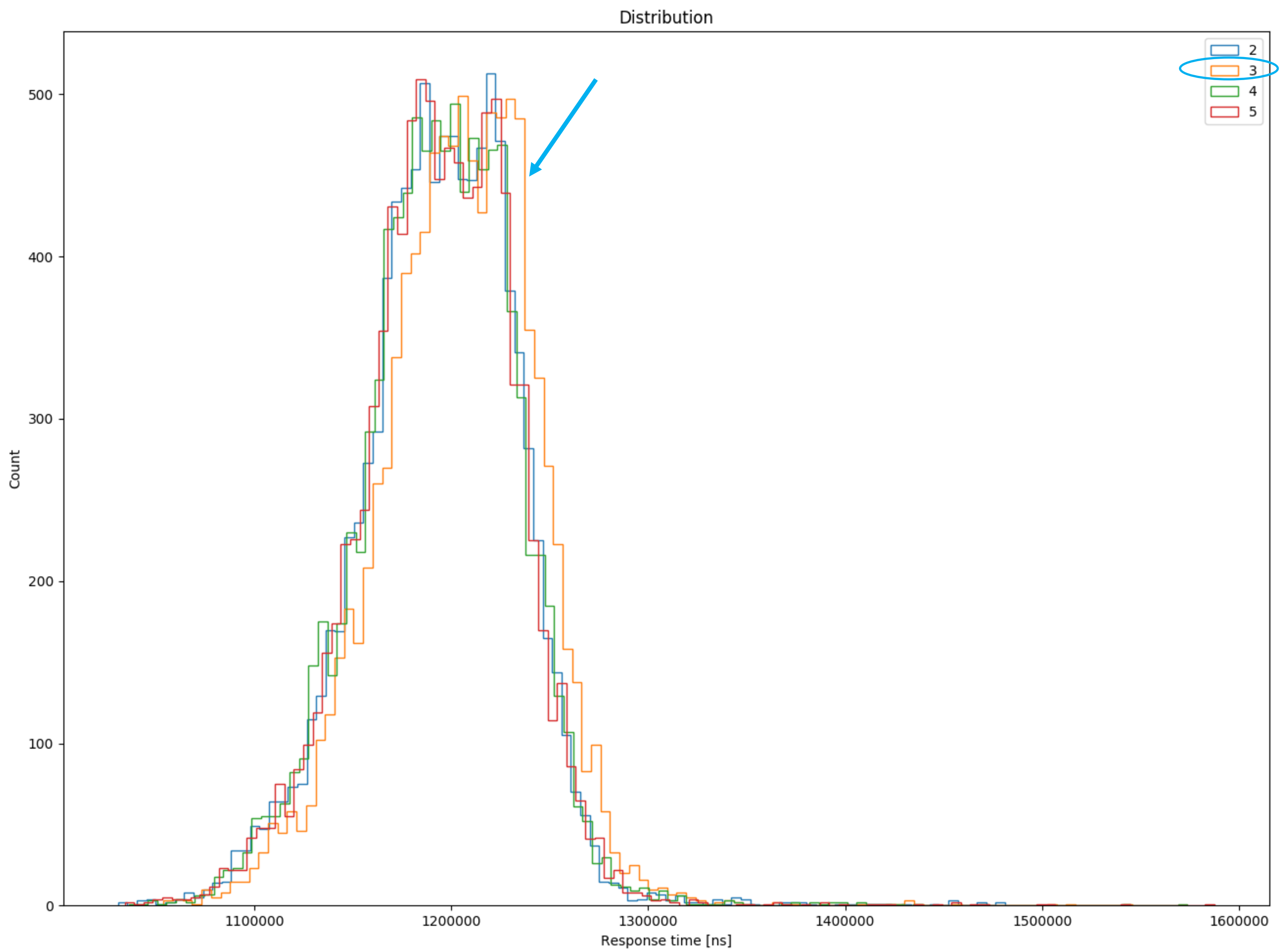
- *Acquisition* noise:
  - Host (*H*), Network (*N*), Victim (*V*)
  - Different between the Forward (*F*) and the Return (*R*) paths
- Some noise may be assumed constant
  - under very short timeframes

Confidential

My test setup

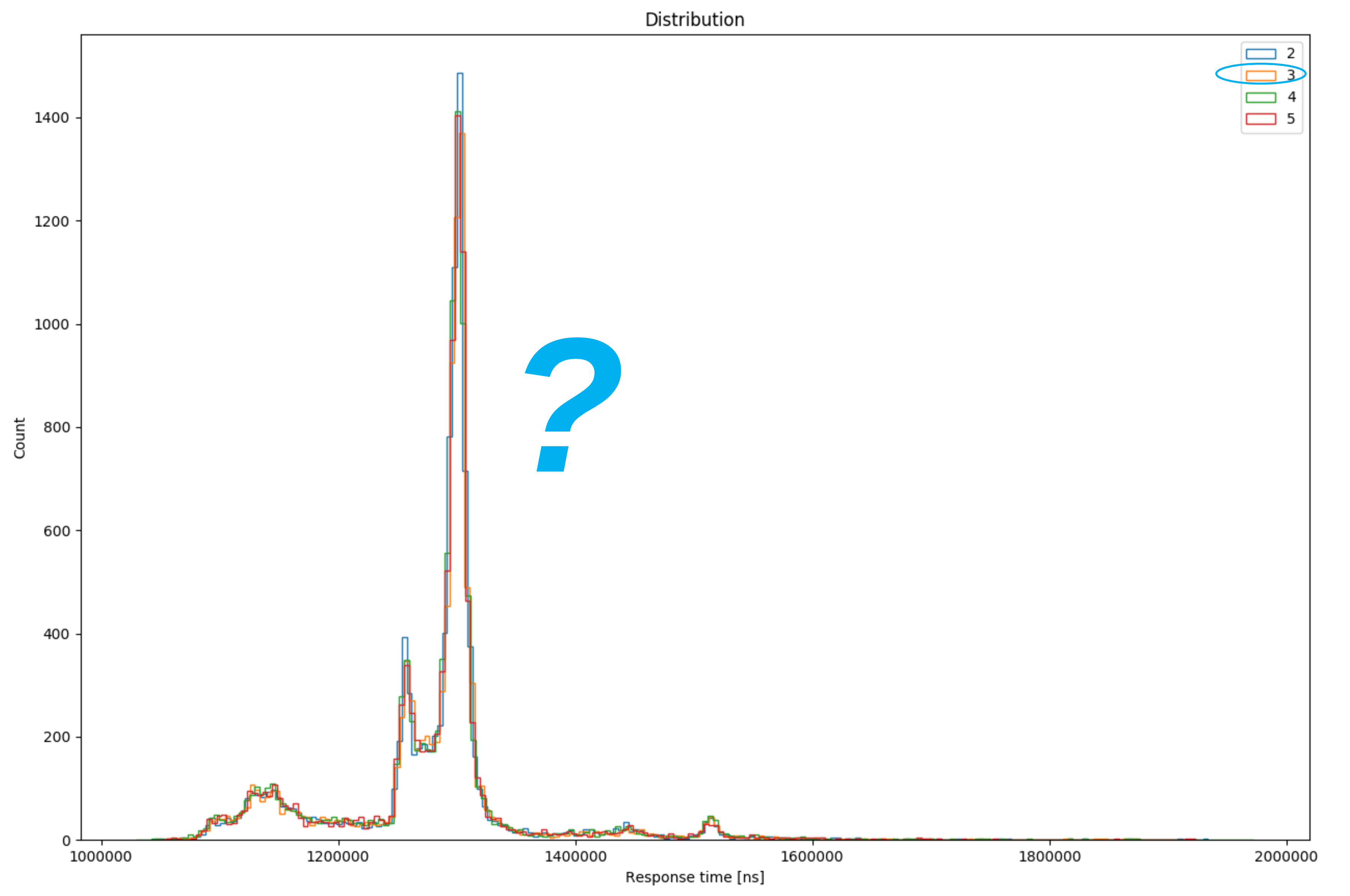
- *Arduino UNO:*
  - *16MHz*
  - 100MBit Eth shield
  - Direct connection to Host
- *uClibc strcmp()* implementation:
  - UDP server with password authentication
- Host measurements:
  - We start simple:
    - *Round-trip time*
    - From *userspace*

Timing Distribution (10us difference)



Confidential

Timing Distribution (1us difference)



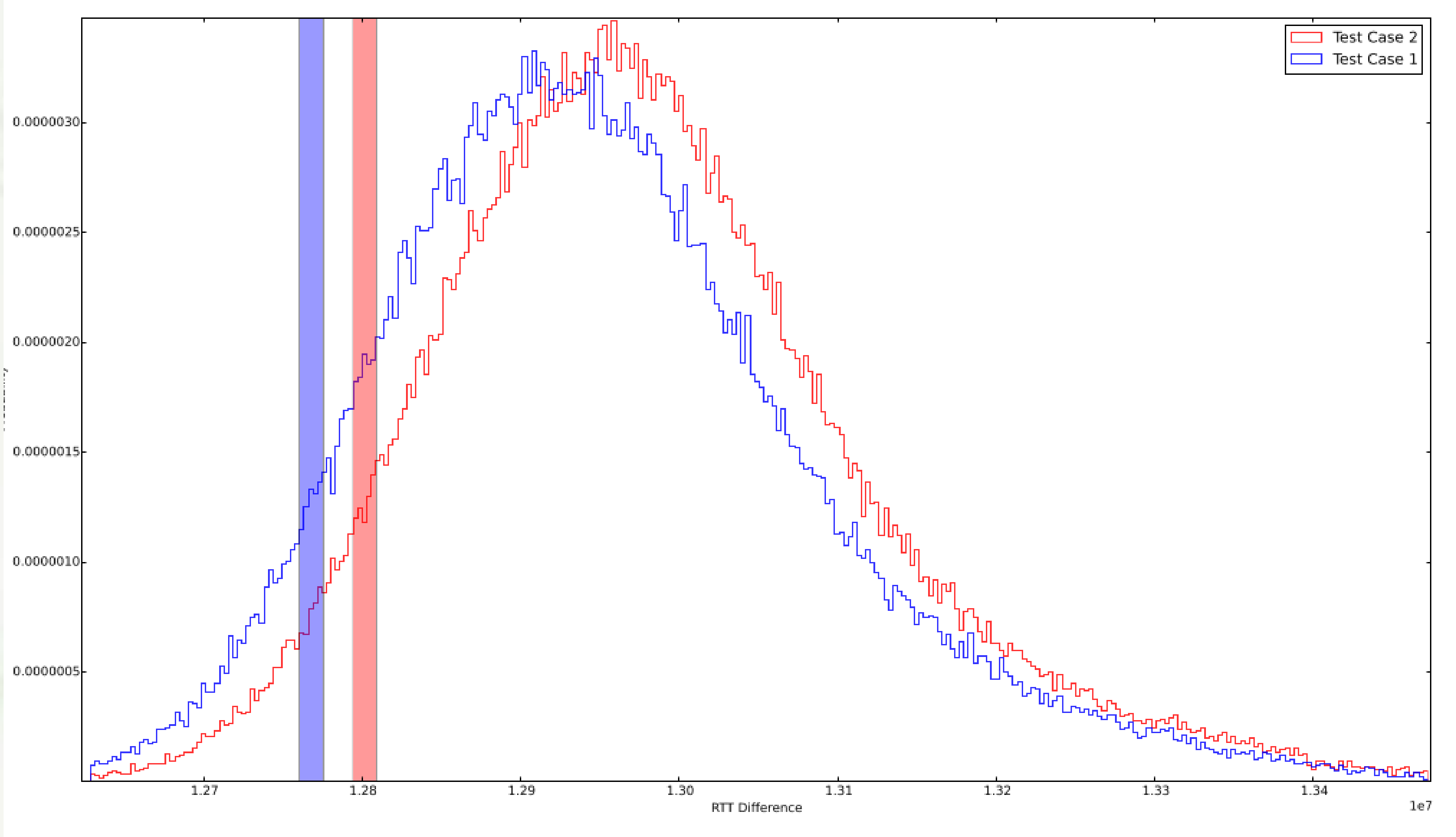
Confidential

Distribution

- Non-Gaussian
- Unknown

- Generic, *distribution-independent estimators required*.
  - Few statistical estimators can be applied
- A good solution:
  - "Box test" *(2009 – Crosby et al.)*

Box test

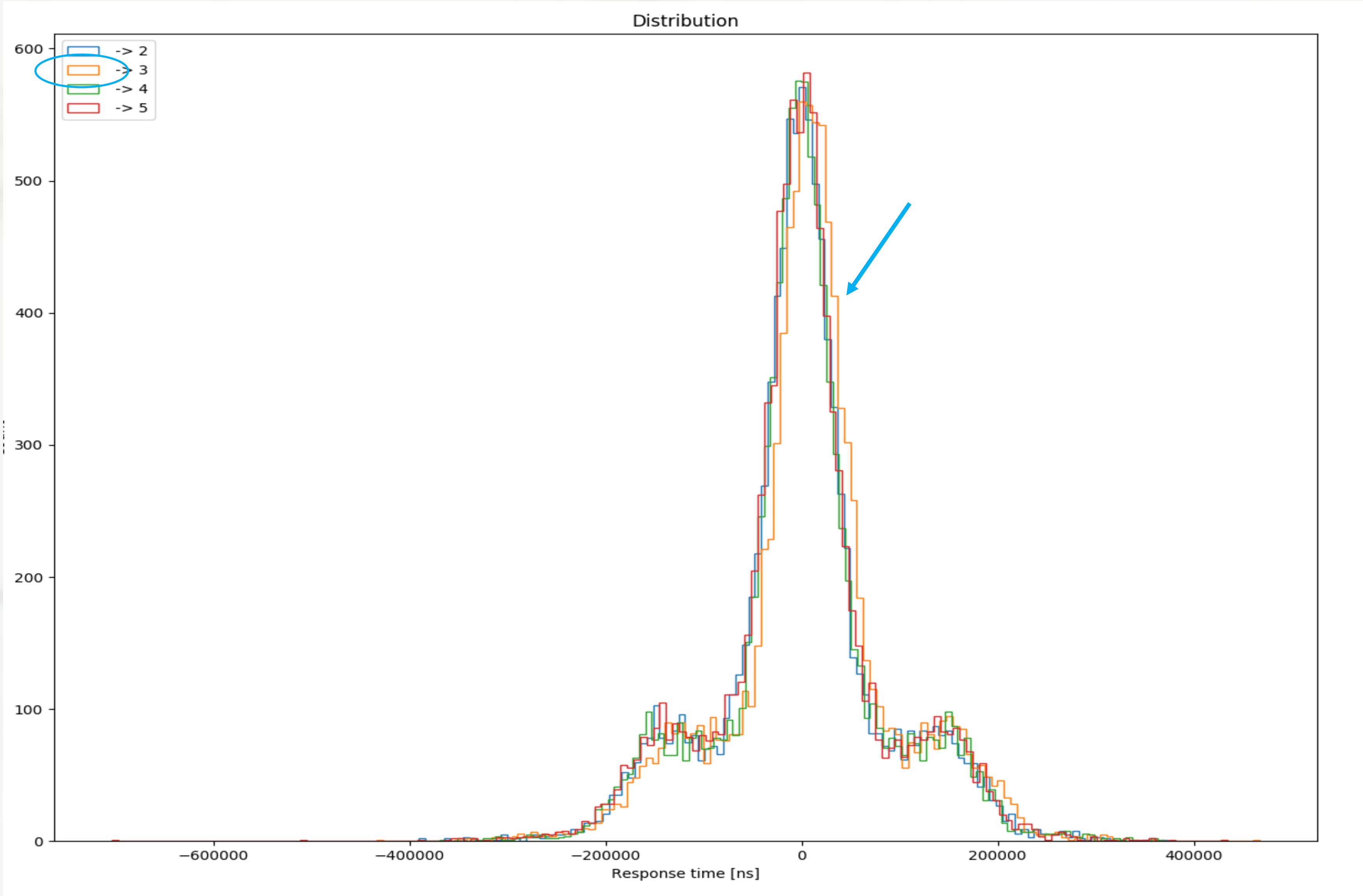


Confidential

Back to Gaussian: differential pairs

- Measurement of *differential pairs* may significantly reduce “stable” noise
- Approach:
  - Take a reference measurement (*ref*)
  - Perform the real measurement (*m1*)
  - Compute *m1-ref=m0*
  - Take m0 as your measurement
- Improvements:
  - *Constant noise is canceled out*
  - *m0* Distribution is *symmetrical* an *zero centered*
  - Under some assumptions it *may be Gaussian*.
  - More *advanced statistical analysis* (and ML) available

Timing Distribution (10us difference)



Confidential

Advantages

- *Quality:* Constant noise is canceled out
- *Processing:* More advanced analysis possible
  - Better estimators available

- ML approach via GMM possible!
  - **Gaussian** Mixture Modeling, assumes Gaussian distributions
  - See great post at:
    - https://parzelsec.de/timing-attacks-with-machine-learning/

Research

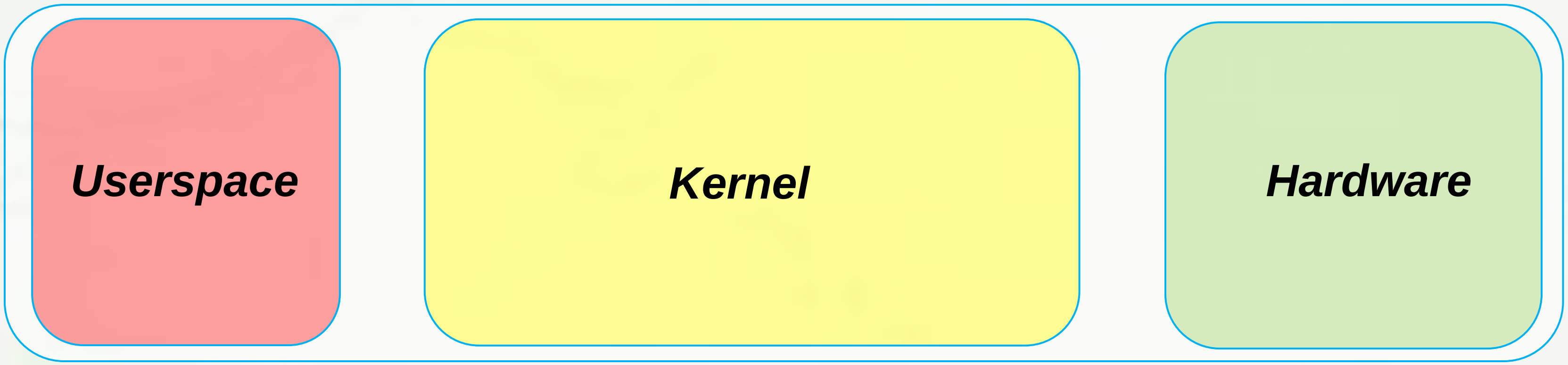
- Mostly focused on:
  - *Reducing network noise*
  - Improving analysis techniques

- Past approaches:
  - Using *TCP timestamps as time source*:
    - May bypass network noise
  - *Differential pairs:*

- **My research goal: *Reducing Host-side Noise***

Confidential

Host-side measurements



- Multiple *different measurement points*:
  - Root privileges/Kernel access may be required
- Multiple *time sources* available:
  - Precision and accuracy may significantly vary
- Golden rule:
  - *"The closer the measurement is to the hardware, the less noise is introduced"*

Measuring from Userspace

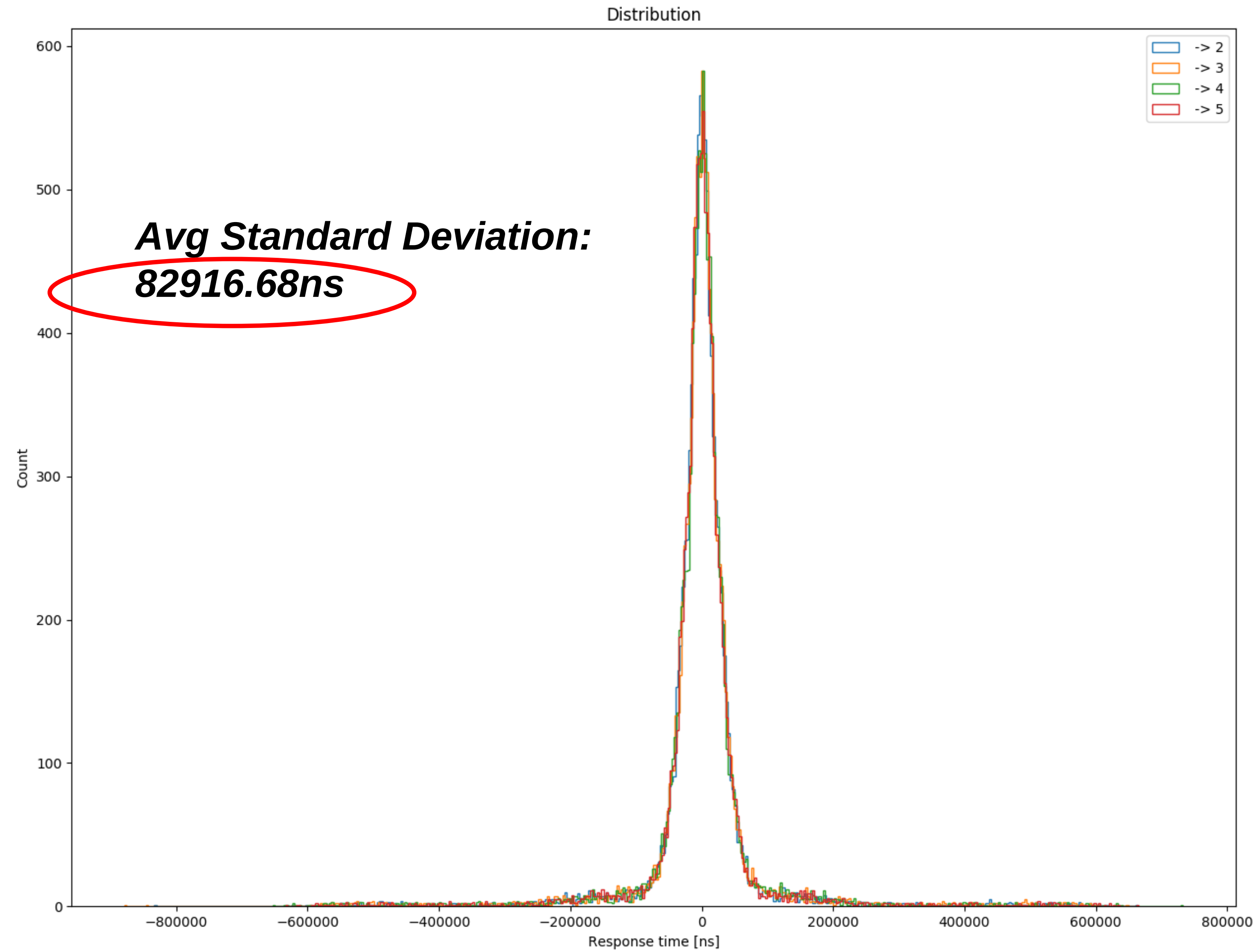
- Used in most researches
- Measurement:
  - Started as soon the packet is sent *(from userspace)*
  - And stopped, as soon as the packet is received *(by userspace)*
- Available time sources:
  - Clocks (monotonic and non)
  - *CPU counters*
- Notes:
  - Some clocks may not have *nanosecond* precision
    - Still, CPU counters may reach sub-nano seconds precisions
  - *Accuracy may be low*:
    - Scheduling, Interrupts,…

Example:

- Python 3.7 supports *time.performance_counter_ns()*
  - Convenient access to *high resolution performance counters*
  - *Nanoseconds precision*

- Notes:
  - Additional delays may be introduced by the Python interpreter.
  - Not an issue if delays are *roughly* constant (in a short timeframe)
    - Cancel out by using measurement of *differential pairs*

Userspace measurement: perf_counter_ns ()



Avg Standard Deviation: 82916.68ns

Confidential

Kernel measurements

- Can be performed by accessing time sources in *kernel space*
- An easy way: ***libpcap***
  - Packets are *timestamped* by the *kernel*
  - That’s how Wireshark receives timestamps! ☺
- Callbacks can be isntalled on packet reception and sending
- Latest libpcap versions support *nanosecond precision!*

```
// Set timestamp type on device
ret = pcap_set_tstamp_type(handle, PCAP_TSTAMP_HOST)

// Set timestamp precision on device
ret = pcap_set_tstamp_precision(handle, PCAP_TSTAMP_PRECISION_NANO)
```

(Do we have) Hardware-level timestamps???

```
sudo ethtool -T eno1
Time stamping parameters for eno1:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        all                   (HWTSTAMP_FILTER_ALL)
        ptpv1-l4-sync         (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
        ptpv1-l4-delay-req    (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
        ptpv2-l4-sync         (HWTSTAMP_FILTER_PTP_V2_L4_SYNC)
        ptpv2-l4-delay-req    (HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ)
        ptpv2-l2-sync         (HWTSTAMP_FILTER_PTP_V2_L2_SYNC)
        ptpv2-l2-delay-req    (HWTSTAMP_FILTER_PTP_V2_L2_DELAY_REQ)
        ptpv2-event           (HWTSTAMP_FILTER_PTP_V2_EVENT)
        ptpv2-sync            (HWTSTAMP_FILTER_PTP_V2_SYNC)
        ptpv2-delay-req       (HWTSTAMP_FILTER_PTP_V2_DELAY_REQ)
```

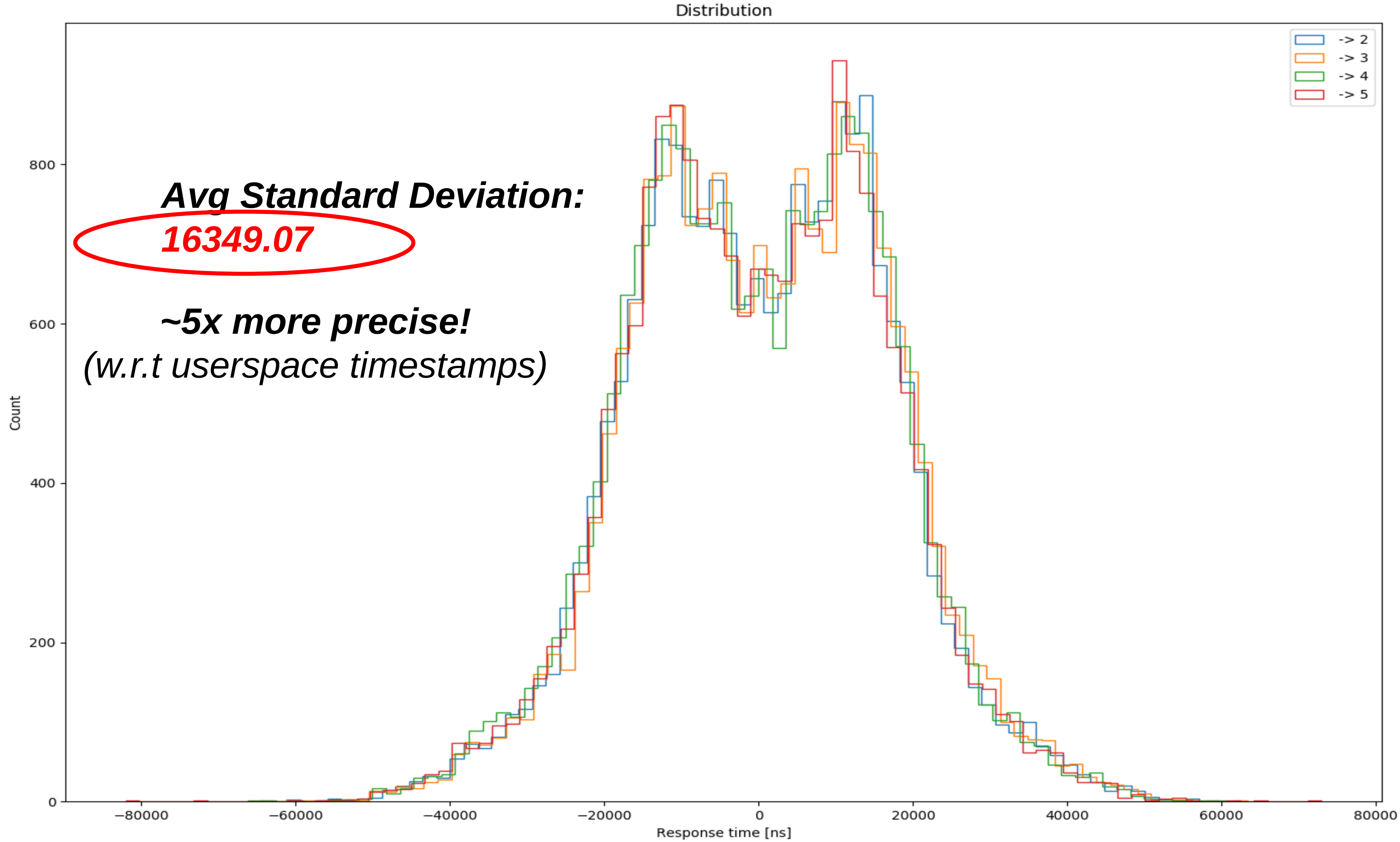
Hardware timestamping!!!

- Can be accessed in the same way from libpcap
- Timestamp provided directly by the network card!
- With *nanosecond precision!*
- ***1st time ever applied to timing attacks (AFAIK)***

```
// Set timestamp type on device
ret = pcap_set_tstamp_type(handle, PCAP_TSTAMP_ADAPTER_UNSYNCED)

// Set timestamp precision on device
ret = pcap_set_tstamp_precision(handle, PCAP_TSTAMP_PRECISION_NANO)
```

HW timestamps (via libpcap)



Avg Standard Deviation:
16349.07

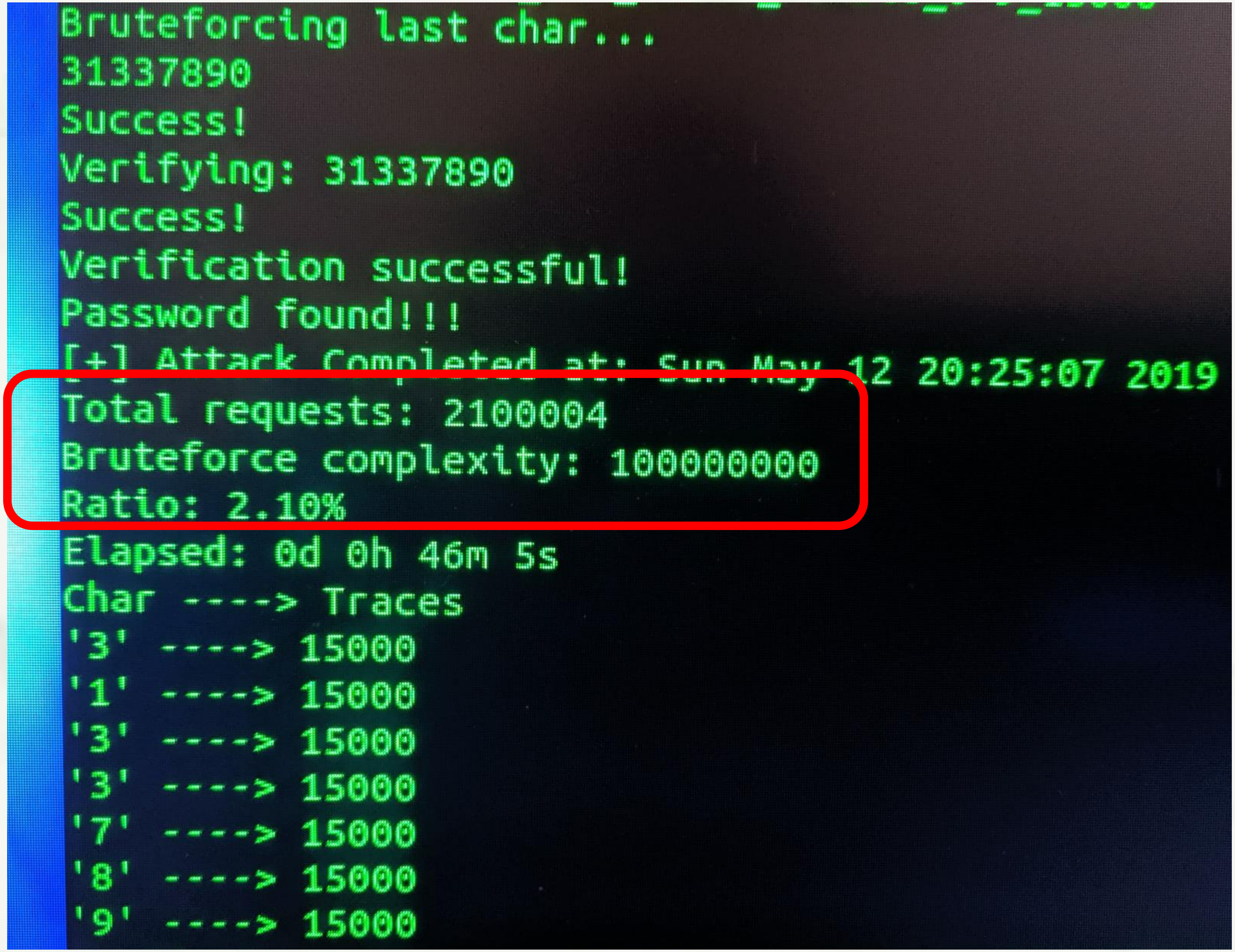
~5x more precise!
(w.r.t userspace timestamps)

Demo

Demo

- Target: Arduino UNO
  - Clock speed: 16 Mhz
  - Media: Ethernet 100Mbit
- Numeric PIN: 8 digits
- Measurements:
  - Differential pairs
  - *Hardware timestamping* enabled

Demo



Confidential

Contacts

Cristofaro Mune

Product Security Consultant

c.mune@pulse-sec.com

Confidential