Out of the Box Testing

Hubert Kario[0009−0007−8694−4270]

Red Hat Czech s.r.o, Purkyňova 115, 612 00 Brno, Czech Republic
hkario@redhat.com

16 May 2023

Abstract. In this paper we analyse typical timing data that can be collected over loopback interface, in local, and in metropolitan area networks. We evaluate performance of few statistical test for detecting differences in timing of server responses. The evaluated tests include the popular Box test, as well as sign test, Wilcoxon signed-rank test, and paired sample t-test. We found that the Box test offers poor performance, as it's an incorrect test to use for the measurements we collected. Use of appropriate tests also allows for robust differentiation between much smaller differences than the existing literature would suggest. We were able to detect side channels of single-digit CPU cycles over regular gigabit Ethernet. Those alternative tests were also found to be robust against noise in production networks, allowing detection of side channel of just few nanoseconds with 6 network hops between test systems.

Keywords: Non-parametric tests · side-channel attacks · timing attacks · box test · sign test.

1 Introduction

Cryptographic implementations not only need to compute expected outputs for all valid inputs and handle errors gracefully, they also must not expose the values of processed secret data indirectly.

If an implementation exposes certain property or values of the secret values through its behaviour (though timing of responses, memory accesses, electromagnetic fields, sound, power use, etc.) we say that it has side-channel leakage. The timing of responses is particularly interesting as it allows for attacking it remotely through normal interfaces.

RSA is one of the algorithms notably impacted by attacks like this. Particularly seminal paper on this topic, introducing the so-called box test, was written by Crosby et.al[1]. That same test was then reused by many other researchers[5, 3, 2, 6].

Unfortunately, none of those researchers, not even ones that considered use of the box-test but ultimately decided not to use it[4], have verified the assumptions of the test. That test, like many others, requires samples to be identically and independently distributed.

1.1 Contributions

Our work makes the following contributions:

- We examine the timing behaviour of synthetic servers in detail
- We present a new technique that follows established statistical practice to measure timing differences that outperforms the box test.
- We show that by using correct statistical approach, measurements of much smaller timing differences are possible than previously shown.
- We show that differences of single clock cycles are measurable over loopback interface with few hundred thousand measurements.
- We show that robust measurements of side channel over metropolitan area networks as small 10ns are also possible.

1.2 Paper structure

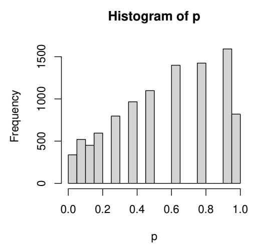
We begin the paper with a refresher on statistical theory: p-values. After that, we attempt use of the statistical tests without verifying their assumptions of independence. We discuss what statistical independence is and how can we achieve it by performing a double-blind study. Having established a proper study setup we investigate the effect of different factors on the test performance: used CPU, the baseline response delay, introduction of real network link, and real production network.

2 P-values

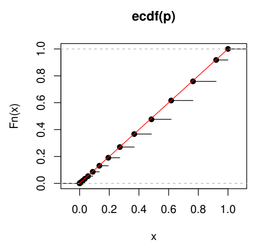
In frequentist statistics, the tests generally return two values, a test statistic and a p-value. While the test statistic and its interpretation varies from test to test, the interpretation of a p-value is the same independent of the specific test used.

When using a p-value, we need to select a value α, representing the acceptable rate of false positive errors (situations when the test reports a signal or effect when there is none). When the calculated p-value is smaller that the value α, the test is considered to be positive, while if it is larger than the α, the result is considered to be negative. This works, because independent of the test used, if the null hypothesis is true (there is no effect present), the distribution of p-values needs to follow a uniform distribution between 0 and 1.

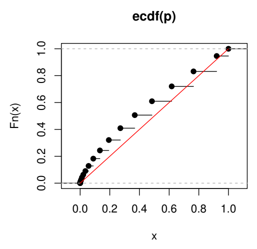
Let's consider a simple binomial test: perform n coin tosses, obtain k positive outcomes and calculate the associated p-value. For large numbers of unbiased coin tosses, p-values from such a test will be distributed uniformly. But because both k and n are discrete values, the corresponding p-values are similarly discrete. Plotted on a histogram, like in Fig. 1a, p-values won't look like a uniform distribution. Naïve interpretation of that would suggest the used test is incorrect. Such interpretation would be wrong though, as if we look at the Empirical Cumulative Distribution Function (ECDF) of the p-values on fig. 1b, we can see that the distribution of the calculated p-values is a non-continuous uniform



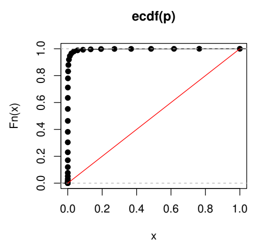
(a) Histogram of p-values with no effect



(b) ECDF of p-values with no effect



(c) ECDF of p-values with a biased generator ($p = 0.54$)



(d) ECDF of p-values with a very biased generator ($p = 0.7$)

Fig. 1: Distribution of p-values of 10 thousand binomial tests with $n = 100$, and expected p of 0.5.

distribution, matching a continuous uniform distribution (the red line on the graphs) very closely.

Since the binomial distribution can also deal with biased events, like weighted coins, it's defined with additional parameter: p – the probability of success of any one instance of the event (like a coin toss in our example). For an unbiased coin it's 0.5, for a coin that lands always on one side p would be either 0 or 1 (depending if it always lands heads or tails), and a coin that prefers one side over the other would have the p parameter other than those three values, with values closer to 0 and 1 indicating heavier bias. It should be noted that p and p-value are separate: p-value is a generic concept while p here is a parameter specific to the binomial distribution and the associated test.

If there is an effect (the expected p doesn't match the p of the underlying distribution), the calculated p-values will not follow uniform distribution of $(0, 1)$. The departure from uniformity may be slight, but for large number of p-values easy to notice when compared to an Cumulative Distibution Function (CDF) of the uniform distribution, as visible on fig. 1c.

```
ret = read(sock, &x, 4);
for (i=x; i != 0; i--) {
   asm ("" : "+g" (i) ::);
}
```

Listing 1: Down for() server processing function in C

For higher differences between the expected p and actual probability p of the sample, smaller p-values are more likely. The same holds true for bigger samples.

In other words, a presence of an effect or a signal doesn't guarantee a small p-value from a statistical test, it just makes it *more likely*. At the same time, a small α makes it *less likely* that the test will generate a p-value that is smaller than it when there is no signal or no effect. In more practical terms, if we execute a test 1000 times, we can expect that we'll see p-values smaller than 0.01 around 10 times, *even if there is no effect*.

Since our goal is to use those tests in automated environments, executing the tests over and over, where we expect to see very small differences in case of a side channel, we'll be using small α values, more typical of physics (10^{-5}) than biology or sociology (0.05) so that the probability of false positives is very low.

3 Synthetic server

To test effectiveness of different approaches we need some server with known behaviour, preferably where we can control the overall response time and amount of side-channel leakage.

3.1 Experimental server

A server we used for the tests reads an integer from the network socket, counts down until it is equal zero and then writes a reply to the client. We used such configuration for a multitude of reasons. When the data to be processed comes from the network, the compiler can't anticipate the values to be processed, and thus optimise around them. While local attacks are possible, the biggest attack surface comes through a network thus testing a network service makes it more realistic. Moreover, if we can test servers over the network, we don't need to modify real servers to test if they are vulnerable or not (we can perform black-box testing of real servers). Given the cloud environments, it's also relatively easy to get a machine in the same data centre as the target, while getting on the same physical machine requires more specific circumstances. The reason to make a server counting down is that such loop on an x86_64 platform can be implemented by two machine instructions, providing very fast execution, and thus high granularity between different inputs.

In listing 1 you can find the core function generating the side-channel for measurement. We read the value of the counter from the network directly to avoid

```
0x0040139c <+12>:  lea    0xc(%rsp),%rsi
0x004013a1 <+17>:  call   0x401080 <read@plt>
0x004013aa <+26>:  mov    0xc(%rsp),%eax
0x004013ae <+30>:  test   %eax,%eax
0x004013b0 <+32>:  je     0x4013bd <doprocessing+45>
0x004013b2 <+34>:  nopw   0x0(%rax,%rax,1)
0x004013b8 <+40>:  sub    $0x1,%eax
0x004013bb <+43>:  jne    0x4013b8 <doprocessing+40>
```

Listing 2: Down for() server in assembly

```
conversation = Connect(host, port)
node = conversation
node = node.add_child(RawSocketWriteGenerator(
↪  baseline.to_bytes(byte_len, "little" )))
node = node.add_child(ExpectAlert(AlertLevel.fatal))
node.add_child(ExpectClose())
conversations["probe A, {0} cycles".format(baseline)] = conversation
```

Listing 3: Down for() client

any bias in methods like `sscanf()` and to make sure that all values require the same number of bytes to transmit between attacker and the tested system. We then compare it to 0, as that is translated to a single instruction, `jne`, subtraction is compiled to the `sub` instruction with an immediate value, as can be seen in listing 2, where the body of the loop is the last two instructions (previous ones handle the special case of initial iterator equal to 0). The `asm` block is used to trick the compiler into assuming that the loop can't be optimised out as it processes all the intermediate values of iterator `i`.

3.2 Experimental client

For the client we used the `tlsfuzzer` framework. The basic connection was specified as in listing 3. This prepares a series of actions performed by the framework when the conversation is actually executed: first a TCP connection is established (the `Connect` node), then a byte string representing the numerical value of `baseline` is written to the opened socket. Finally the framework expects a TLS Alert message (the `ExpectAlert` node) and a TCP connection closure (the `ExpectClose`) node.

Three copies of such conversation are prepared: two that send the same value (by comparing them to each-other we can detect false positives), and one with an increased value (the true positive).

If iterating the loop different amount of times takes different amount of time, we expect then to see similar processing time for the conversations with the same values and different processing time when we compare either of the first two with the third one.

3.3 Experimental setup

The tlsfuzzer framework executes the specified conversations provided number of times. To minimise the effects of branch-prediction and as a general good statistical practice, it performs a randomized trial: it executes the conversations in random order. To get highest precision of measurements, instead of measuring response time on Python level, it performs a packet capture, and after executing all the connections, it maps the collected response timings to the conversations, in the process ensuring that the time measurement isn't influenced by the exchanged data (as neither the packet capture process nor the parser inspect the exchanged data, the parser just looks for the time between last message from client and a server response to it).

4 Statistical independence

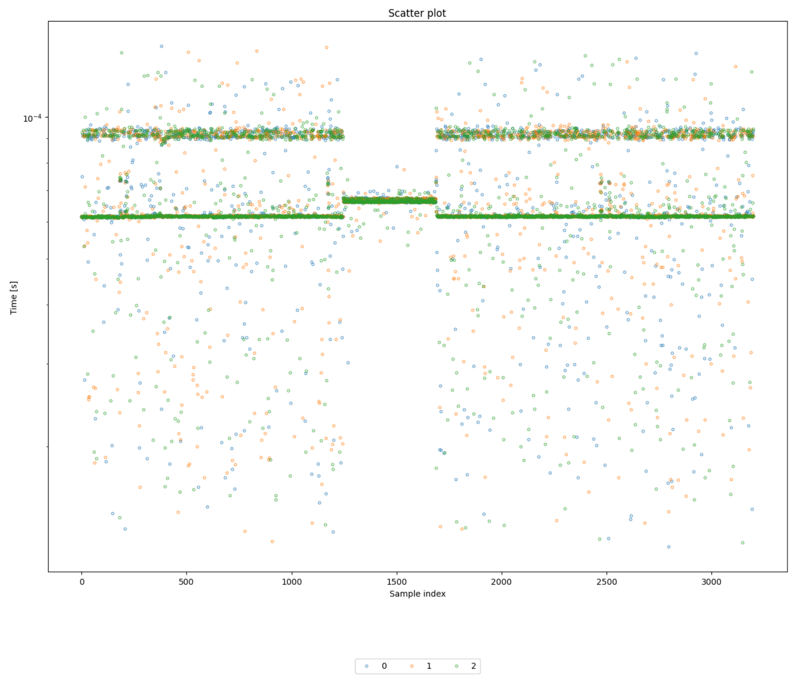


Fig. 2: Scatter plot of individual measurements, note the markedly different distribution for measurements with indexes between around 1250 and 1700. The 0, 1, and 2 in the legend refer to the three different probes executed.

By executing a client that sends three types of probes we essentially get three data samples, each comprised of server response times for a particular input. The widely applied approach for analysing such data is to perform so-called box test[1]. One of the criteria mandated by the box test, is that the samples need to be independent and identically distributed. Unfortunately, independence of measurements wasn't a property that was verified either in the original study or in the studies that used the same test[5, 3, 2, 6], or ones that considered box test, but found it to be under-performing when compared to Mann—Whitney U test[4]. Note that Mann—Whitney U test also requires independent samples.

In statistics, we say that two events are independent if occurrence of one has no impact on occurrence of another. That also means that the measurements have no impact on one another, in particular, that by inspecting one measurement we don't learn anything about measurements in its immediate vicinity.

While there are statistical tests for independence, they're not foolproof. Thankfully, in the case of timing measurements we just need to look at a graph of measurements to be able to tell that they are not independent, as in figure 2.

While often the lack of independence isn't as striking as in fig. 2, it is the counter-example for the hypothesis that the samples are always independent. Or in other words, if we were to execute the different samples one after another, instead of in parallel, we would be comparing values distributed as between index 0 and 1000, and then between index 1000 and 2000, despite them having vastly different distributions. It would be like going to Phoenix, Arizona, measuring the daily temperature, every hour, every day for the whole January, coming up with an average of 13.8°C, then travelling to Seattle, Washington, doing the same there in June, coming up with an average of 16.7°C, and based on that coming to the absurd conclusion that on average Seattle is hotter that Phoenix.

When we ignore requirements of a statistical test we risk two effects: the test will be producing false positives at an increased rate (above the selected alpha), or it will not be sensitive to real differences. We'll be getting random results not directly related to the analysed data.

The solution is to use a test that can handle dependent measurements, like most paired two-sample tests: where instead of estimating the means of samples individually, the samples are considered to be ordered, and the values are compared in pairs. The first measurement from first sample is compared only with the first measurement from the second sample. Same for second and subsequent measurements. Then we inspect the distribution of the results of those comparisons.

One of the first statistical tests ever described is the sign test. To perform the test, we calculate pairwise differences of the measurements and note the number of positive and negative results (in standard sign test ties are ignored). Then the count of positive values is compared with the binomial distribution with the probability parameter p=0.5. So for example, if we counted 450 positive values and 445 negative values, the p-value of the test will equal 0.89, which is a non-significant result. As such, we cannot assume that either sample median is larger (technically, sign test checks for stochastic dominance, which for our purposes means the same, but is a separate concept).

A more advanced test is the Wilcoxon signed-rank test, which accounts for not only the sign, but also the magnitude of the differences.

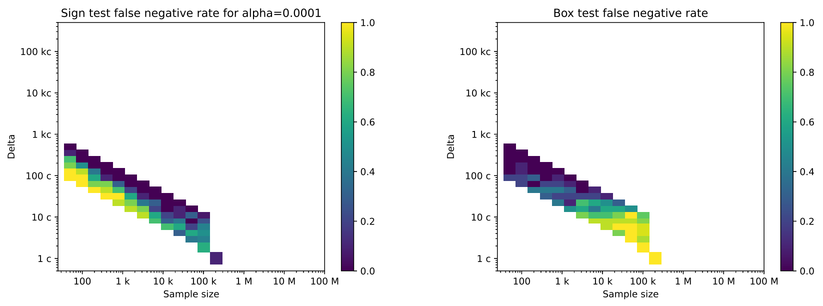


Fig. 3: False negative rates of the test executed against synthetic server measured over loopback interface, with baseline loop count of 10 cycles. White means "no data".

In case of measurements over the loopback interface, the Box test is significantly less sensitive than either the sign test or the Wilcoxon signed-rank test.

Fig. 3 shows the empirically acquired fraction of test runs that returned negative result when there was a difference between the tested samples as a product of the sample size and tested difference (delta) between the two samples. Dark blue colour indicates that almost all tests correctly identified presence of side-channel while yellow indicates that almost none did.

In fig. 3 we can see that, as expected, the sign test scales with an inverse square root of sample size, consistently detecting side-channels an order of magnitude smaller with samples two orders of magnitude larger. At the same time, while the box test seems to outperform the sign test for small sample sizes, it is due to high rate of false positives (fig. 4) rather than it being more sensitive. Also, the box test isn't able to detect differences at all when the difference becomes really small (10 cycles in this example).

5 Double-blind study

The statistical best practice calls for performing a randomized double blind study. In such experiments, the participants are assigned randomly to a control group and a treatment group (the "randomized" part) and where both the investigator applying the treatment and the participant don't know if the participant is part of control group or not (the "double blind" part).

While we don't have human actors in the experiment we're running, the investigator (test client) can still have an effect on the study participant (test server). Effects can range from the time between individual connections, and time to sending data after opening the connection, to cumulative stress (physically heating it up, cache lines evicted, branch predictor state, etc.) imposed on the CPU beforehand. All of these can have effect on how quickly the server responds. If those things are correlated with the sent probes they will impact server behaviour, even if the set of instructions executed by the server is exactly the same. That's because the *environment* they're executed in is different, and correlated with the probes.

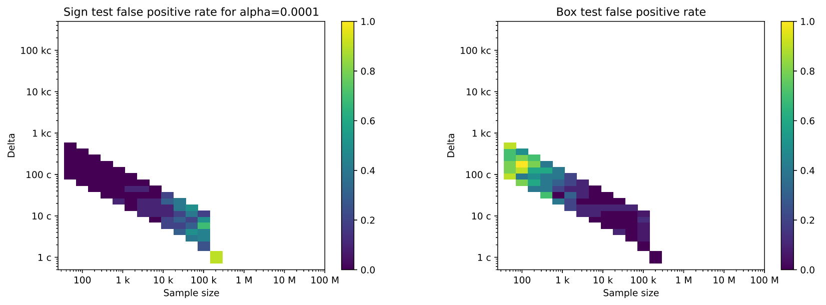


Fig. 4: False positive rates of the test executed against synthetic server measured over loopback interface, with baseline loop count of 10 cycles.

Indeed, when we investigate the false positive rate of the sign test in fig. 4, we can notice that it fairly uniformly rises as the sample size increases. For moderately large samples (200k observations) the sign test isn't any better than a coin toss at detecting if a side channel is present, that's despite use of a very small alpha value (0.0001), which for an unbiased test should show around one false positive in ten thousand tries. Here we see it in eight out of ten tests.

From the experiments we performed it's caused by a mixture of effects, all pointing to non-uniform behaviour on client side causing non-uniform behaviour on server side, even if exactly the same data is sent over the network.

We tried different approaches to eliminate this effect: using other timings of the TCP connections, like the time to establish the connection (time from initial SYN to reply with SYN-ACK, to time to the first ACK packet), time to generate the query, to time to close the connection. None of those values were sufficient to eliminate this confounding while using stratification based on ranges, k-means or HDBSCAN classifiers. Similarly, linear regression, non-parametric regression, simple Bayesian inference, and Bayesian logistic regression were ineffective in elimination of this confounding.

The solution that works is to employ double blinding, where the values to be sent to the server are first generated in a random order, saved to a file on disk, and then processed by one and the same conversation in tlsfuzzer, as seen in listing 4. Effectively executing exactly the same CPU instructions on the client side every time, irrespective of data being sent to the server.

```
data_file = open('data_values.bin', 'rb')
byte_len = 4
conversation = Connect(host, port)
node = conversation
node = node.add_child(RawSocketWriteGenerator(
    data_file=data_file,
    data_length=byte_len))
node = node.add_child( ExpectAlert( AlertLevel.fatal ) )
node.add_child(ExpectClose())
```

Listing 4: Down for() client with pre-generation

This kind of test execution eliminated the excessive false positive rate and had no negative effect on false negative rate. In fig. 5 we can see that there were no false positives detected. The sudden increase in true detections for tests with above 50 thousand observations seems to be related to the mobile Intel i7-10510Y CPU reaching steady temperature or power state or positive resonance with the test client.

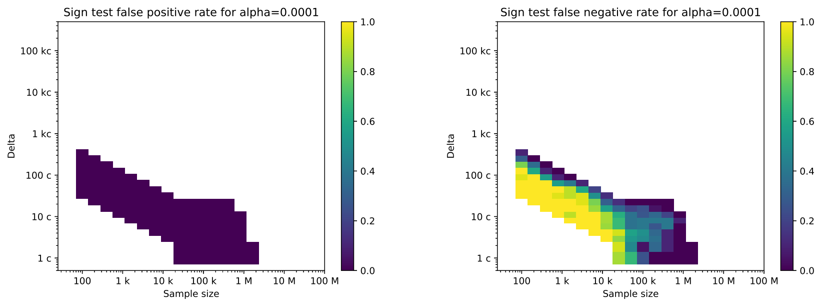
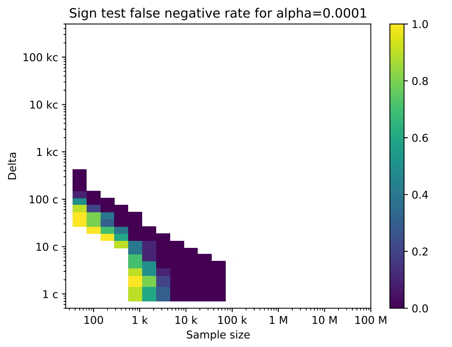
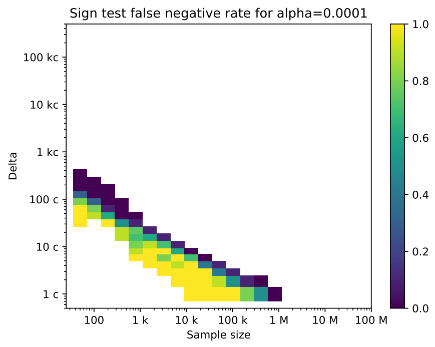


Fig. 5: False positive and false negative rates of sign test of the synthetic server measured over loopback with probe pre-generation. Each cell represents at least 10 individual test executions. i7-10510Y CPU.

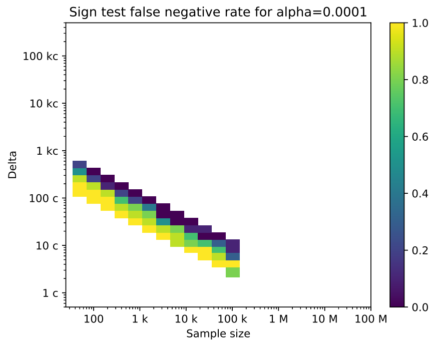
Executing the same kind of test on different machines we see similar effect: no false positives above what's expected given the selected alpha level and number of test executions, even for very large sample sizes or noisy environments. At the same time, the sample size at which a particular size of side-channel is detectable depends on the baseline count in the `for()` loop. In fig. 6 we can see comparison of the effectiveness of the sign test for different baseline counts of the `for()` loop. First one is for 10 cycles, an arbitrary small number, showing similar sudden increase in test effectiveness for larger sample sizes as in fig. 5. For larger baselines we see the more expected linear (on a log-log plot) relationship between sample size and side-channel. The 850 cycles were chosen as being 1µs slower than 10 cycles. The 3174300 cycles were chosen as that is the typical response time for OpenSSL 1.1.1p processing a 2048 bit RSA ClientKeyExchange message in TLS.



(a) 10 `for()` cycles, about 14µs



(b) 850 `for()` cycles, about 15µs

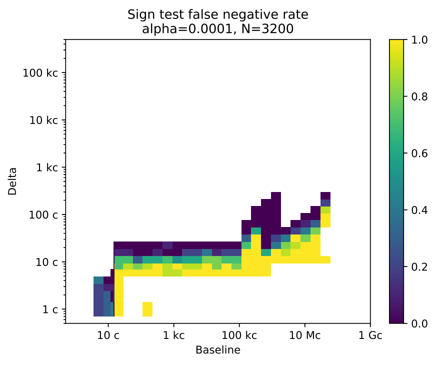


(c) 3174300 `for()` cycles, about 5.2ms

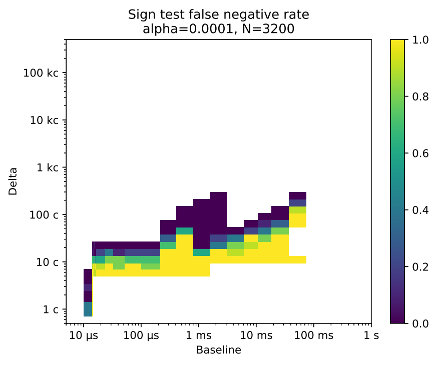
Fig. 6: False negative rate for tests with different baseline delay and size of side-channel based on sample size. Tests with probe pre-generation. Each cell represents at least 10 individual test executions. Intel Xeon E5603 CPU

.

When we ran the same sample size for the tests (N=3200) and instead varied the baseline iteration count, we've noticed, as shown in fig. 7, that the higher than expected sensitivity was consistent for all small baselines. All baselines of 15 cycles and below allowed differentiation of 1 cycle differences with just 3200 probes. At the same time all probes with 16 or more, up to around 100 thousand cycles, have the same sensitivity. Only when the baseline count increases enough to cause the server response to take few hundred µs to 1ms that it has an effect on the test sensitivity, with the beginning of the linear (on the log-log plot) relationship starting with 1ms response time.



(a) Baseline in cycles



(b) Baseline in seconds

Fig. 7: False negative rate as a function of side-channel size (Delta) and baseline iteration count for the probes. Intel Xeon E5603 CPU

In other words, the sign test is much more sensitive and robust than the box test. Additionally, the behaviour of CPUs varies widely by the executed workload, and thus has very significant impact on how easy it is to detect a side channel. Finally, even realistic server response times for computationally intensive workloads require just few million probes to detect side channels as small as single-digit CPU cycles.

6 Local network attacks

While assuming a threat model in which the attacker can measure server response times over loopback is natural, given the prevalence of virtualization and containerization, the most realistic is the expectation that the attacker is able to deploy a system in the same data centre.

As such, we've executed tests between two hosts on the same local network, with a copper gigabit Ethernet connection between them, connected with a single managed switch (D-Link DGS-1100-24). Average ICMP ping RTT between them was about 0.086 ms.

The best test performance we've achieved was when the test server was running on the fastest machine available in that network segment (an Intel Xeon E3-1220), while the test client (attacker) was running on the second fastest machine (an Intel Xeon E5-2407 v2).

As seen in in fig. 8 the measured test performance varied widely, especially for very small sample sizes (<1000) only very large side channels (>15000 cycles)

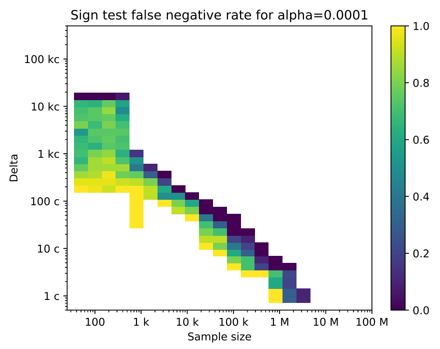


Fig. 8: False negative rate for measurements over gigabit Ethernet (0.086ms RTT) against a `for()` loop server running on Xeon E3-1220 and a client running on Xeon E5-2407 v2, with the baseline of 10 cycles

were reliably discoverable. But with increasing sample size, the test fell to the expected linear (on log-log plot) relationship between sample size and measurable side channel. With 1.6M measurements being necessary to detect side channels as small as 1 `for()` loop cycle. For comparison, over loopback on the same system, though for a baseline of 850 cycles, only 410 thousand measurements are necessary to detect side channels as small as 1 cycle. This is over two orders of magnitude smaller difference than previously deemed detectable[1].

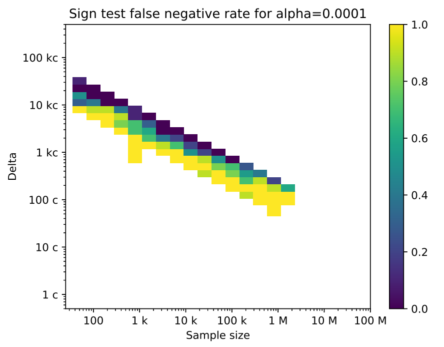
In other words, introduction of a real network between the server and the attacker requires increasing the sample size by a factor of 4 to achieve similar test sensitivity as can be achieved over the loopback interface (same-system network connections). So, if an attack is practical over loopback, it's most likely practical over real local network connections too.

7 Metropolitan Area Networks

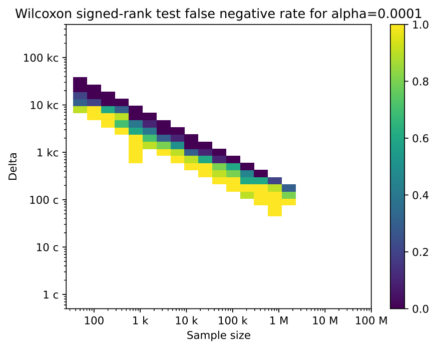
We've also tested the performance of the test when executed against a server located further away. When the server was about 0.534ms, 6 router hops, and about 5km away, we still saw quite good test performance. The server was running on an Xeon E3-1220 at 3.191GHz. With the client running on Core i9-12900KS at 5.225GHz.

This test environment is also the first time when the paired t-test is competitive when compared to either sign test or Wilcoxon signed-rank test (in all the other examples with the sign test, the results from paired t-test were all false negatives). This is most likely caused by the fact there are multiple noise sources contributing to the overall measurement error, causing them to average out to a more normal-like distribution because of central limit theorem.

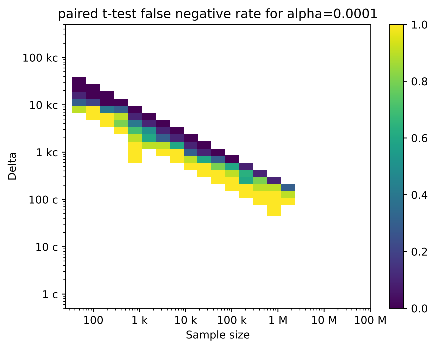
From graphs in fig. 9, we can expect that a sample of few hundred million measurements per probe type should be sufficient to detect differences as small



(a) Sign test results



(b) Wilcoxon signed-rank test results



(c) Paired sample *t*-test

Fig. 9: False negative rate for tests against a networked server separated by 6 router hops, with average return trip time of about 0.534ms. Tests with probe pre-generation. Each cell represents at least 10 individual test executions. `for()` loop server with 10 cycle baseline. Intel Xeon E3-1220 CPU on server side and Intel Core i9-12900KS on client side

.

as 10 `for()` loop cycles. As expected, after scheduling a very large test, with 284M measurements with a 10 cycle delta we found data to show statistically significant results. The bootstrapped estimate of difference in response between 10 cycle and 20 cycle probes was equal $3.0^{+1.0}_{-2.0}$ ns for median and $10.69^{+3.06}_{-2.83}$ ns for mean at 95% confidence interval. The calculated sign test p-value was equal to 1.98e-6, the Wilcoxon signed-rank test p-value was equal to 5.06e-9, and the dependent t-test for paired samples was equal 2.65e-9.

From those results we can conclude that the smallest size of difference measurable is not limited by the noise or network distance, but the size of the sample we're willing, or able, to collect.

It should be noted that all three statistical tests used here permit combining data from multiple sources to detect smaller differences, as all three do not require consistency in magnitude of differences between different data pairs. Which means that parallelization of data collection and its later analysis is quite trivial and does not require extensive considerations when running the tests.

8 Future work

In this work we have used one of the simplest approaches to detecting sample differences. Use of more advanced statistical models, like with Bayesian inference, might outperform those old tests.

There exist tests for testing multiple samples at once, like the Friedman test. Quantifying their performance in comparison to the sign test and bootstrapped confidence intervals would allow to estimate the size of the maximum remaining side channel. That would be particularly useful in case different side-channels can be tested in parallel.

Good explanation of the sudden jumps in performance for very small baselines or larger samples sizes might point to a way of constructing a test harness that regularly outperforms the presented test results.

While we found that combining test data from multiple systems together improved the confidence of the test results, we haven't quantified when and how beneficial that is in general. In particular, we've seen in one example that combining similarly sized data from two systems with very different signal to noise ratios made the overall result worse than analysis of data from just the better of them.

9 Summary and recommendations

We've shown that by using correct statistical methods and best practices we can detect much smaller timing side-channels than previously deemed possible.

When executing statistical tests, users should check if the data meets the expectations of the statistical test used. We found that including "canary" probes (ones that are treated as separate by the test but should generate the same behaviour from system under test—i.e. known true negatives) to be a very good approach for verifying sanity of the test setup. When the failure rate from comparing them exceeds the selected α level, the test setup should be re-evaluated.

When measuring very small differences, on the order of nanoseconds, it's necessary to take into account the side-channels generated by the test harness. Even things like the memory location from which the data is copied to the network can have measurable effect on server response times. We found pre-generating values of all the probes to be sent and writing them to disk before executing even a single network connection to be a flexible and robust way to ensure constant-time (or at least probe independent) behaviour from the test harness.

By using such practices we were able to detect side-channels as small as few clock cycles (below 1ns) over regular gigabit Ethernet. Thus we don't think that general purpose implementations can ignore presence of timing side channels when they are determined to be small: it just means they are harder to detect, not that they are impossible to detect over the network.

10 References

[1] Scott A. Crosby, Dan S. Wallach, and Rudolf H. Riedi. "Opportunities and Limits of Remote Timing Attacks". In: *ACM Trans. Inf. Syst. Secur.* 12.3 (Jan. 2009). ISSN: 1094-9224. DOI: `10.1145/1455526.1455530`. URL: `https://doi.org/10.1145/1455526.1455530`.

[2] Tom Van Goethem et al. "Timeless Timing Attacks: Exploiting Concurrency to Leak Secrets over Remote Connections". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1985–2002. ISBN: 978-1-939133-17-5. URL: `https://www.usenix.org/conference/usenixsecurity20/presentation/van-goethem`.

[3] JS Daniel Mayer and J Sandin. "Time trial: Racing towards practical remote timing attacks". In: *Black Hat US Briefings* (2014).

[4] Robert Merget et al. "Raccoon Attack: Finding and Exploiting Most-Significant-Bit-Oracles in TLS-DH(E)". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1151.

[5] Christopher Meyer et al. "Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks". In: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 733–748. ISBN: 978-1-931971-15-7. URL: `https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/meyer`.

[6] Mathy Vanhoef and Eyal Ronen. "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 517–533.