Red Hat
Enterprise Linux

Side-channel leakage verification using statistical approach

Alicja Kario
Principal QE in Red Hat Crypto Team
2025-10-08 The OpenSSL Conference

Red Hat

Introduction & Motivation: The QE Perspective

Who I Am: A Quality Engineer, not a security researcher.

My Primary Goal: Show absence of side-channels, not just find them.

This requires a universal, reproducible, and robust methodology.

Universal Approach

Scope: all libraries we ship in RHEL, not just OpenSSL

Languages: C, Go, Java, ASM, etc. – can't rely on language-specific tools

Architectures: x86_64, ARM, ppc64le, s390x, ... – must be architecture-agnostic

Environment: Preferably works against network target, with a complete black-box implementation.

The Starting Point: Chasing Ghosts

Initial Target: Investigate a timing variant of the ROBOT Attack

The Problem: Inconsistent and non-reproducible results. While the data hinted at an issue, it wasn't a solid proof.

Red Hat

Hanno Böck, Juraj Somorovsky, Craig Young, "Return of Bleichenbacher's Oracle Threat (ROBOT)".

Short Detour: Python libraries

To understand the Bleichenbacher attack, I shifted to the Python RSA implementations (M2Crypto, pyca/cryptography, python-rsa)

Finding: They were vulnerable

Reason: Vulnerabilities in all three, but they were “easy”. Malformed ciphertexts caused exceptions, leading to huge, obvious differences. Not the subtle ones I’ve seen from looking at OpenSSL over a network

Red Hat

The Real Problem: “Industry Standard” Methods

Returned to OpenSSL, attempted to use standard statistical analysis (like the Crosby’s Box Test)

Result: Non-reproducible results. Was it 20-year old attack or the measurement was flawed?

Are we using the right tools for the job?

https://www.cs.rice.edu/~dwallach/pub/crosby-timing2009.pdf

Agenda

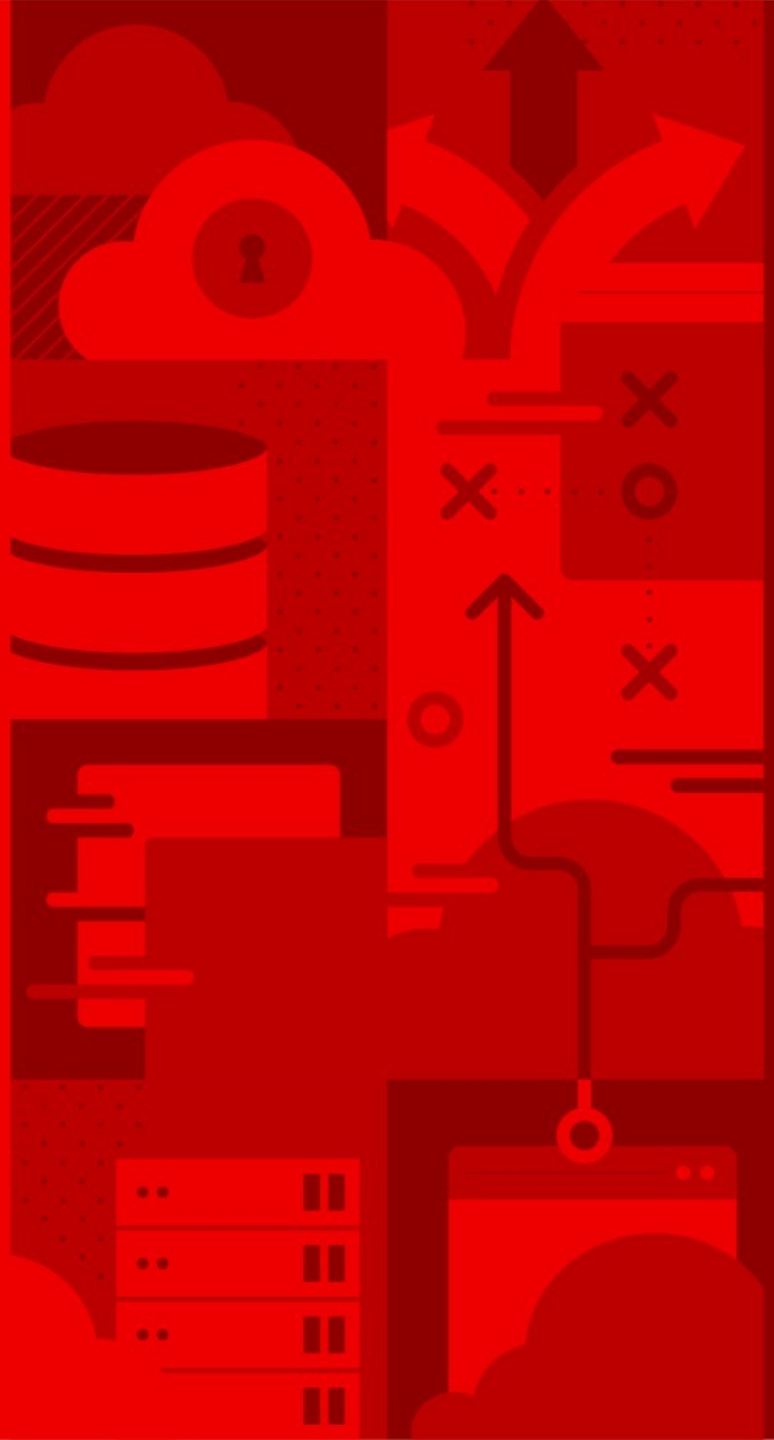
The Problem with Timing Measurements

Building a Reliable Measurement Methodology

Applying the Method to RSA Encryption: The Marvin Attack

Beyond RSA: Finding Leaks in ECDSA

Conclusion and a Path Forward



Building a Reliable Measurement Methodology

Red Hat

Back to Basics: the Simple Network Server

To understand the noise, I created a minimal server.

Operation: Read a binary number from the network, count down that many cycles in a for() loop, reply.

Such controlled and easy to understand environment should reveal the behavior of the measurement environment.

Towards Forging a Reliable Methodology

Two things become apparent:

1. The measurements are not independent
2. The test harness has effect on the server under test

Measurements are NOT Independent!

Statistical tests (t-test, box test) assume Independent and Identically Distributed (IID) data.

My experiment showed that this assumption is false for timing measurements.

Reasons: Caching (Data, µOp, Instruction), Branch Prediction, CPU Frequency Scaling (thermal throttling). The past operation affects the next one!

The Observer Effect

Turns out that the client code (the “observer”) was not blind to the data it was sending.

Its timing variation in sending the probes was correlated with data classes, in effect polluting the server’s response times.

Solution Part 1: A Double-Blind Test Setup

To remove observer effects, I implemented a double-blind study.

1. **Randomise:** Generate all test probes in random order.

2. **Isolate:** Write the randomized probes to a file on disk.

3. **Execute Blindly:** Use a dimple, “dumb” client that just reads from the file and sends it to the server, unaware of what it’s sending.

4. **Reassemble:** Using prior knowledge de-randomise the collected data points, assign them to appropriate classes.

Solution Part 2: The Right Statistical Tools

Since data is not Independent, we must use tests that handle paired, dependent data.

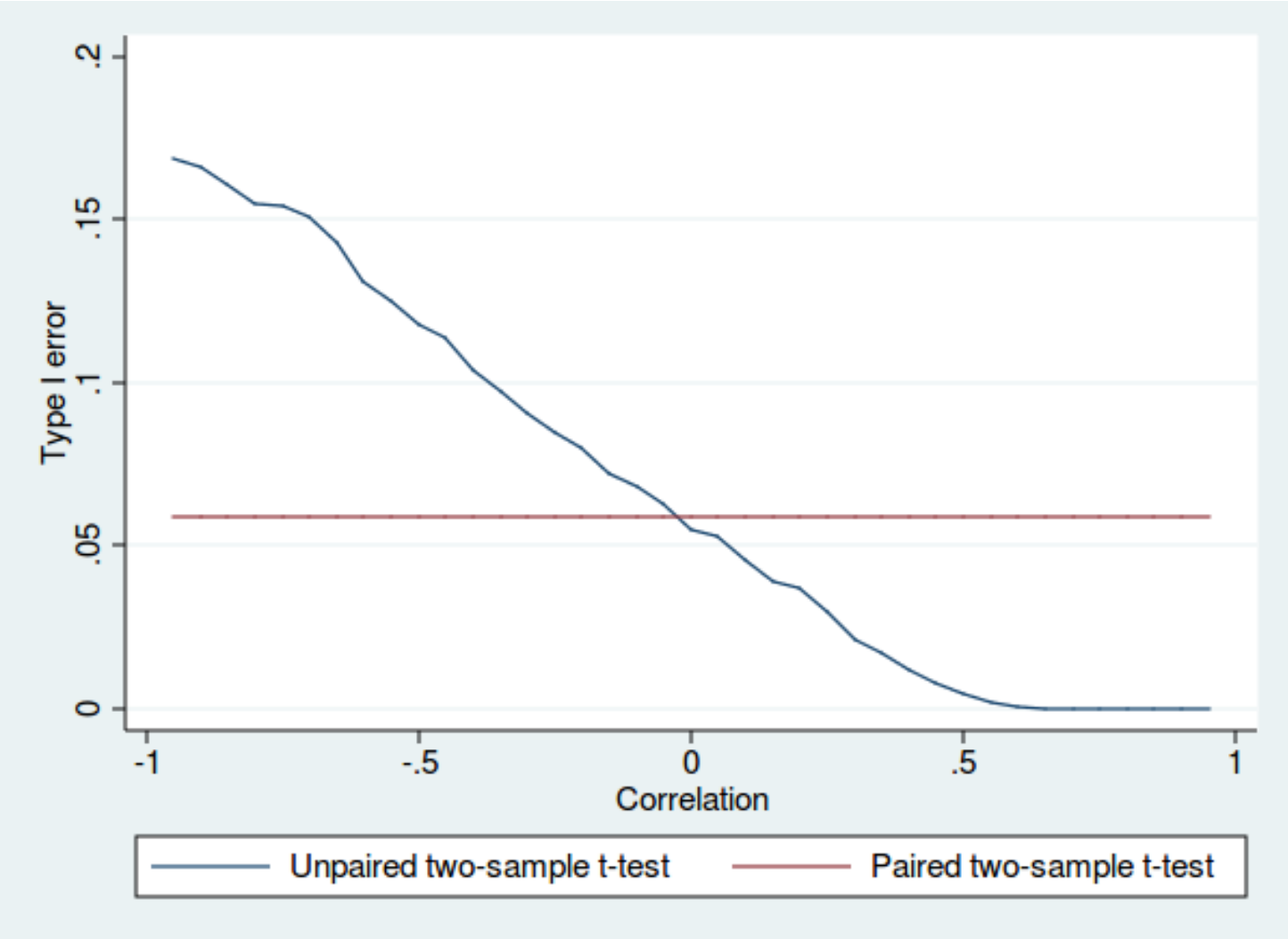
Don't Use: Unpaired t-test, Box Test

Do Use:

Sign Test

Wilcoxon Signed-Rank Test

(For multiple samples: Friedman or Skillings-Mack test)



Source:
https://en.wikipedia.org/wiki/File:Type_1_error.png

Result: Nanosecond Precision over the Network

Combining the double-blind setup with correct statistical tests yielded amazing results.

Precision: Reliable measuring differences of single clock cycle (<1ns)

Environment: Standard, real-world Gigabit Ethernet network in an office setting.

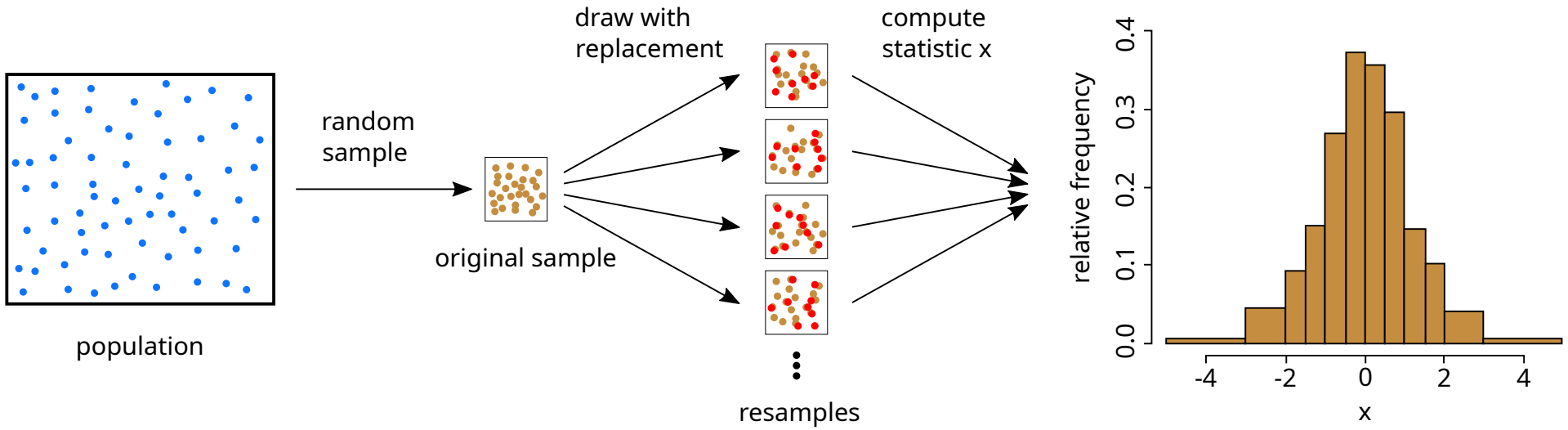
How Much Data is “Enough”?

Frequentist tests only say “yes” or “no” to a difference, not “how big is the difference?”

Solution: Bootstrapping.

By resampling the collected differences, we can calculate confidence interval for the average difference

Bootstrapping



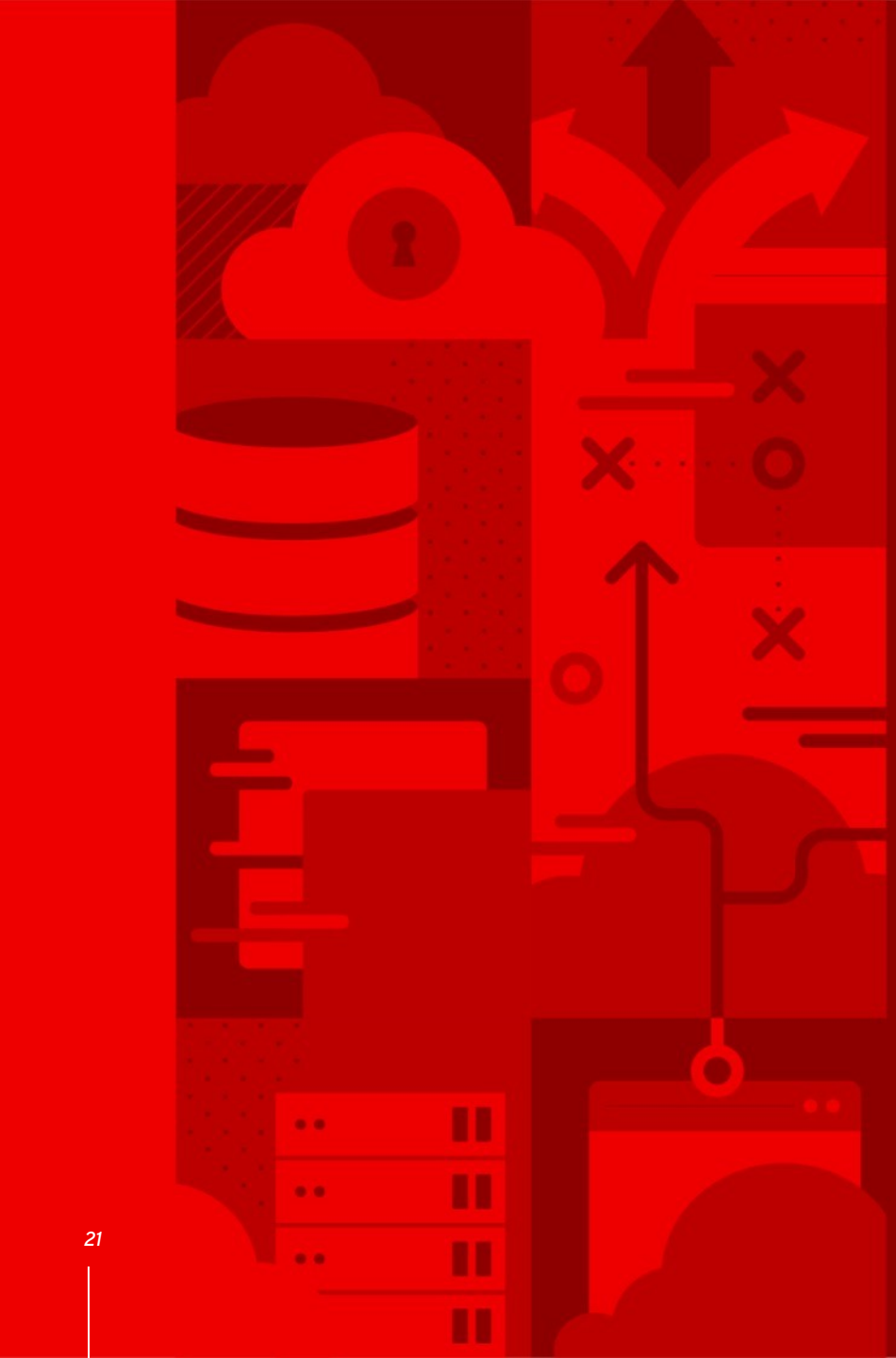
Source:
https://en.wikipedia.org/wiki/Bootstrapping_(statistics)

My Criterion for “Absence of Vulnerability”

If a statistical test (like Sign Test) is negative ($p > 10^{-5}$)

AND the 95% confidence interval from bootstrapping is a smaller than the duration of a single CPU clock cycle…

then we can be reasonably sure there is no remaining side-channel. If the interval is larger, we need more data.



Using the Method

Red Hat

The Marvin Attack

Applying the New Methodology to OpenSSL: The original target.

The Finding: A timing variant of Daniel Bleichenbacher's 1998 attack was still exploitable.

A nearly 20-year-old vulnerability was missed by other researchers.

https://people.redhat.com/~hkario/marvin/

Why Was This Missed for So Long

The limitations of existing analysis.

Static Analysis: Focused on small, isolated cryptographic primitives

The Blind Spot: While the modular exponentiation primitive was constant-time, the surrounding deblinding code was not. The vulnerability was in the integration, not the low level implementation.

Outcome: Co-authored and published "The Marvin Attack" (CVE-2022-4304). Over 35 other implementations vulnerable.

https://people.redhat.com/~hkario/marvin/

If RSA is Leaky, What Else is?

RSA result called everything into question. Next target: ECDSA.

Previous academic research (Masaryk University) in the Minerva Attack had concluded OpenSSL's ECDSA was secure.

My methodology showed otherwise

CVE-2024-13176 CVE-2025-27587 https://github.com/openssl/openssl/issues/23860
https://minerva.crocs.fi.muni.cz/

ECDSA Nonce Leak

The Leak: The bit length of the nonce used in the signing operation was leaking.

The Cause: Again, not in the core scalar multiplication algorithm, but in the nonce generation code.

Worked with my colleague George Pantelakis and upstream developers to find the issue, identify the vulnerable code, and verify the fix.

CVE-2024-13176 CVE-2025-27587 https://github.com/openssl/openssl/issues/23860

The Complexity of a “Complete” Fix

Fixing it was not as simple.

Identified three separate vulnerable code paths: random nonce, deterministic nonce, and legacy API with externally provided nonce.

Discovered architecture-specific bugs on ARM, ppc64le, and s390x, which were missed when only x86_64 was tested.

CVE-2024-13176 CVE-2025-27587 https://github.com/openssl/openssl/issues/23860

The Common Denominator: BIGNUM

What did the RSA and ECDSA vulnerabilities have in common?

The BIGNUM is not inherently side-channel secure, it's very easy to use it in a way that will leak the bit size of the operands or result.

Thankfully, it's happening: OpenSSL PR #28522

The Road Ahead: Post-Quantum Cryptography

Applying the Proven Methodology: Proactively testing PQC algorithms for side-channel resistance.

ML-KEM: Initial analysis of the OpenSSL implementation has not revealed any vulnerabilities.

Next Steps: Applying the same rigorous testing to ML-DSA and other upcoming PQC standards.

A Note on Confidence

We have run hundreds of tests with this methodology.

Sample size as large as 2 billion measurements per class.

Using strict alpha (significance level) of 1 on 100000.

Result: Zero false positives, as expected from statistical theory.

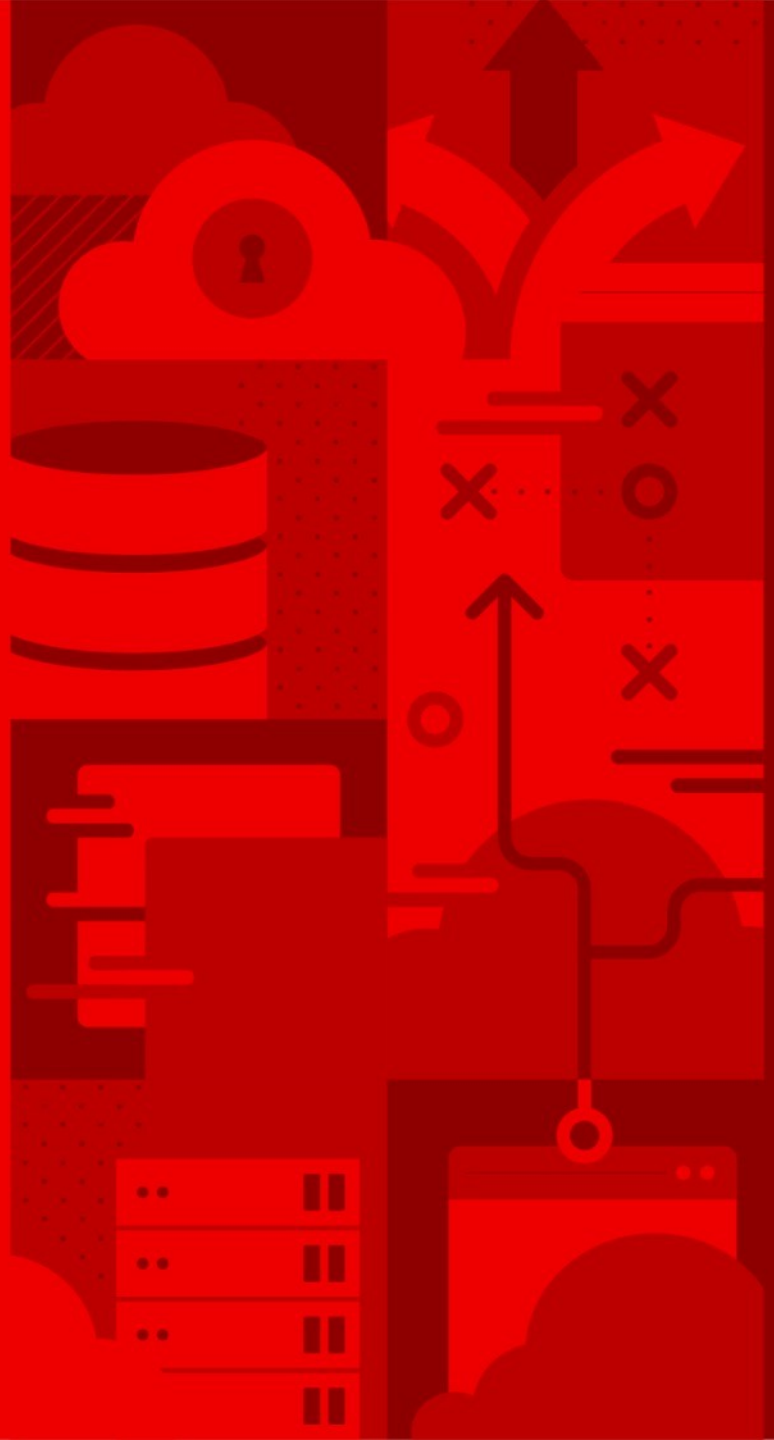
Summary of Findings

Discovered and helped fix the Marvin Attack in OpenSSL's RSA

Discovered and helped fix the Minerva Attack in OpenSSL's ECDSA

Identified the non-constant-time BIGNUM library as significant issue.

Developed a highly reliable, universal methodology for detecting timing side-channels



Conclusion

Red Hat

Verify Your Assumptions

The Takeaway: Statistical tools are powerful and reliable, but only when used correctly.

You must verify that you data meets the assumptions of the test you're using.

For timing analysis, the Independence assumption is almost always false.

Recommendations for Practicioners

Assume a lack of independence for all timing measurements.

Structure tests as double-blind experiment to eliminate observer effects.

Use the Right Math: Employ tests designed for paired data.

Sign Test, Wilcoxon Signed-Rank Test,

Friedman Test, Skillings-Mack Test (for >2 samples)

Quantify uncertainty with bootstrapping to determine if you have enough data.

Materials

Base library/toolkit: https://github.com/tlsfuzzer/tlsfuzzer

Toy example: https://securitypitfalls.wordpress.com/2023/10/16/experiment-with-side-channel-attacks-yourself/

Toolkit for RSA: https://github.com/tomato42/marvin-toolkit/

Toolkit for ECDSA: https://github.com/GeorgePantelakis/minerva-toolkit/

Toolkit for arbitrary precision arithmetic: https://github.com/tomato42/ctmpi

Toolkit for ML-KEM: https://github.com/tlsfuzzer/mlkem-sct-toolkit (WIP)

My contact: hkario@redhat.com

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat