Poster: Systematic Evaluation of Automated Tools for Side-Channel Vulnerability Detection in Cryptographic Libraries

Antoine Geimer[1,2], Mathéo Vergnolle[3], Frédéric Recoules[3], Lesly-Ann Daniel[4], Sébastien Bardin[3], and Clémentine Maurice[1]

[1] Univ. Lille, CNRS, Inria
[2] Univ. Rennes, CNRS, IRISA
[3] Université Paris-Saclay, CEA, List
[4] KU Leuven, imec-DistriNet

1 Introduction*

Side-channel attacks exploit the physical effects of a program's execution, reconstructing secret data from observations such as power consumption or in the case of microarchitectural side-channels, execution time and cache accesses. Numerous solutions have been proposed to mitigate these attacks. For microarchitectural attacks, the most commonly deployed in cryptographic libraries is the constant-time programming discipline [4,3], where developers must avoid branches, memory accesses that depends on secret values. In practice however, constant-time programming can result in complex code, harder to both understand and maintain. Various side-channel detection frameworks [6,2,13,16,22], have thus been proposed to help developers abide this principle. Despite this, past research has shown that many side-channels vulnerabilities can still be found in libraries, often manually [14]. Two factors could explain this paradox: either developers do not use these tools, or these tools are unable to find these vulnerabilities. While a recent survey by Jancar et al. [12] suggests that the former is often true, the latter possibility remains unexplored.

We explore this question by providing a thorough state-of-the-art of side-channel detection tools and of recent vulnerabilities, answering the following research questions:

RQ1 How to compare these frameworks, as their respective publications offer differing evaluation?

RQ2 Could an existing framework have detected these vulnerabilities found manually?

RQ3 What features might be missing from existing frameworks to find these vulnerabilities?

*This extended abstract is an abridged version of previously published works [8] and describes additional preliminary results included in the poster.

Contributions Our contributions are as follows:

1. We present a multi-criterion qualitative classification of 34 side-channel vulnerability detection tools;
2. We compare a subset of these frameworks on a unified benchmark, comprised of representative cryptographic operations;
3. We give a classification of recently published side-channel vulnerabilities, offering new insights into where to find potential vulnerabilities;
4. We verify whether 4 of these vulnerabilities could have been detected with the aforementioned frameworks, issuing recommendations to the community to improve the state-of-the-art.

Additional results In addition, we present preliminary results on the impact of compiler optimizations on constant-time source code. While it has been known for a while that compilers can introduce constant-time violations [18,17], our results bring nuance to the question by relating vulnerable instructions to their corresponding source code lines. We also give details on how particular *combinations* of optimizations break constant-time. We find for example instances where complex interactions between function inlining, arithmetic simplifications and loop unswitching introduce secret-dependent branches.

2 Framework classification

Criteria Our classification is structured in two broad categories of tools, static and dynamic, the former being geared towards formal verification and the latter towards bug-finding. For each tool we also detail the methods employed by the authors (e.g., symbolic execution, fuzzing). Our classification is enriched with the type of inputs used by the analysis (e.g., C source, binaries), and the information outputted: estimation on the number of bits leaked, origin of the leakage, and whether a witness triggering the vulnerability is given. Other criteria include whether the analysis has a soundness claim, whether blinding is supported, and finally an approximate estimation of the analysis' scalability is provided.

Insights Dynamic approaches have become more popular since 2017, representing a shift in research communities and a focus on more scalable bug-finding approaches. We note that this advantage mainly holds for single-trace approaches, as approaches based on trace comparison can require a time-consuming acquisition phase. Single-trace approaches lack coverage, but could be supplemented with fuzzing. We also note that static and dynamic symbolic execution has become a popular approach, with advances in SMT solving making it practical for side-channel detection.

3 Vulnerability classification

Criteria We classify recent (2017-2022) side-channel vulnerabilities in two broad categories: known and new vulnerabilities. Known vulnerabilities can

resurface for two reasons: when known-vulnerable functions are used in *new contexts*, or in *new libraries*. In the first case, developers keep vulnerable functions in the code-base for performance reasons, carefully avoiding using them when manipulating secret data. This leaves the door open to new vulnerabilities where these known-vulnerable functions are used in a new context (e.g., using *square-and-multiply* in RSA key generation) [20,21,1,7,5]. In the second case, the lack of developer awareness may prevent side-channel mitigation transfer from one library to the other [11].

Insights The majority of recent publications reproduce vulnerabilities which have been long known. Analysis should not simply focus on detecting CT violations in their code, but rather detect their incorrect use in the wider code-base. Test methodologies like TriggerFlow [10] are promising in this regard, and complementing them with fuzzing approaches could allow for wider exploration of libraries. New vulnerabilities are not found in usual cryptographic primitives directly, but in newer protocols/schemes [19,15], or in lower-level utility functions [9]. Detection tools thus need to be able to fully analyze programs, including utility functions, and scale to full protocol runs.

4 Unified benchmark and case-study

Unified benchmark To fairly compare the scalability and vulnerabilities reported by existing approaches (**RQ1**), we create a unified benchmark, comprised of representative cryptographic operations from 3 libraries, totaling 25 benchmarks. We run this benchmark on a subset of side-channel detection tools: Abacus [2], Binsec/Rel [6], Microwalk-CI [22], dudect [16], and ct-grind [13].

Case-study We use these tools to determine if the vulnerabilities in our classification could have been discovered automatically (**RQ2**) to determine features that are missing from the state-of-the-art (**RQ3**). We target vulnerabilities from three publications, two stemming from functions known to be vulnerable used in a new context [1,7] and one representing a new vulnerability [9].

Insights Our benchmark and case-studies reveal that scalability remains an issue for most tools, especially when analyzing asymmetric cryptography. We also note that some tools can miss vulnerabilities because of a lack of support for SIMD instructions which are often used to speed up operations. Lack of support for internal secret generation (e.g., key generation) can cause vulnerabilities to be missed. Finally the presence of implicit information flows in program can lead to missed vulnerabilities.

References

1. Aldaya, A.C., García, C.P., Tapia, L.M.A., Brumley, B.B.: Cache-timing attacks on RSA key generation. TCHES (2019)

2. Bao, Q., Wang, Z., Li, X., Larus, J.R., Wu, D.: Abacus: Precise side-channel analysis. In: ICSE (2021)
3. Barthe, G., Betarte, G., Campo, J.D., Luna, C.D., Pichardie, D.: System-level non-interference for constant-time cryptography. In: CCS (2014)
4. Bernstein, D.J., Lange, T., Schwabe, P.: The security impact of a new cryptographic library. In: LATINCRYPT (2012)
5. Braga, D.D.A., Fouque, P., Sabt, M.: PARASITE: password recovery attack against srp implementations in the wild. In: CCS (2021)
6. Daniel, L., Bardin, S., Rezk, T.: Binsec/rel: Efficient relational symbolic execution for constant-time at binary-level. In: S&P (2020)
7. García, C.P., ul Hassan, S., Tuveri, N., Gridin, I., Aldaya, A.C., Brumley, B.B.: Certified side channels. In: USENIX Security (2020)
8. Geimer, A., Vergnolle, M., Recoules, F., Daniel, L., Bardin, S., Maurice, C.: A systematic evaluation of automated tools for side-channel vulnerabilities detection in cryptographic libraries. In: CCS (2023)
9. Genkin, D., Valenta, L., Yarom, Y.: May the fourth be with you: A microarchitectural side channel attack on several real-world applications of curve25519. In: CCS (2017)
10. Gridin, I., García, C.P., Tuveri, N., Brumley, B.B.: Triggerflow: Regression testing by advanced execution path inspection. In: DIMVA (2019)
11. ul Hassan, S., Gridin, I., Delgado-Lozano, I.M., García, C.P., Chi-Domínguez, J.J., Aldaya, A.C., Brumley, B.B.: Déjà vu: Side-channel analysis of mozilla's NSS. In: CCS (2020)
12. Jancar, J., Fourné, M., Braga, D.D.A., Sabt, M., Schwabe, P., Barthe, G., Fouque, P., Acar, Y.: "they're not that hard to mitigate": What cryptographic library developers think about timing attacks. In: S&P (2022)
13. Langley, A.: Ctgrind. `https://www.imperialviolet.org/2010/04/01/ctgrind.html` (April 2010)
14. Lou, X., Zhang, T., Jiang, J., Zhang, Y.: A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography. ACM Comput. Surv. **54**(6), 122:1–122:37 (2022)
15. Pessl, P., Bruinderink, L.G., Yarom, Y.: To BLISS-B or not to be: Attacking strongswan's implementation of post-quantum signatures. In: CCS (2017)
16. Reparaz, O., Balasch, J., Verbauwhede, I.: Dude, is my code constant time? In: DATE (2017)
17. Schneider, M., Lain, D., Puddu, I., Dutly, N., Capkun, S.: Breaking bad: How compilers break constant-time implementations. arXiv **abs/2410.13489** (2024)
18. Simon, L., Chisnall, D., Anderson, R.J.: What you get is what you C: controlling side effects in mainstream C compilers. In: EuroS&P (2018)
19. Tibouchi, M., Wallet, A.: One bit is all it takes: A devastating timing attack on bliss's non-constant time sign flips. J. Math. Cryptol. (2021)
20. Tuveri, N., ul Hassan, S., García, C.P., Brumley, B.B.: Side-channel analysis of SM2: A late-stage featurization case study. In: ACSAC (2018)
21. Weiser, S., Spreitzer, R., Bodner, L.: Single trace attack against RSA key generation in intel SGX SSL. In: AsiaCCS (2018)
22. Wichelmann, J., Sieck, F., Pätschke, A., Eisenbarth, T.: Microwalk-ci: Practical side-channel analysis for javascript applications. In: CCS (2022)