Poster: Scientific Comparison on Accuracy and Scalability of Cryptographic API Misuse Detection

Sharmin Afrose[1], Ya Xiao[1], Sazzadur Rahaman[2], Barton P. Miller[3], Danfeng (Daphne) Yao[1]
[1]*Computer Science, Virginia Tech, Blacksburg, VA*
[2]*Computer Science, University of Arizona, Tucson, AZ*
[3]*Computer Science, University of Wisconsin-Madison, Madison, WI*
sharminafrose@vt.edu, yax99@vt.edu, sazz@cs.arizona.edu, bart@cs.wisc.edu, danfeng@vt.edu

***Abstract*—Studies showed that misuses of cryptographic APIs are common in real-world code. To detect misuses, several open-sourced and commercial tools are developed for screening Java programs. We develop two comprehensive benchmarks (CryptoAPI-Bench, ApacheCryptoAPI-Bench) that enable the first scientific in-depth comparison on accuracy and scalability of misuse detection. CryptoAPI-Bench contains 181 unit test cases covering basic cases, as well as complex cases (i.e., interprocedural, field sensitive, multiple class test cases, and path sensitive cases). The ApacheCryptoAPI-Bench consists of 121 cryptographic cases from 10 Apache projects. Both benchmarks include correct test cases for testing false-positive. We evaluate four tools, namely, SpotBugs, CryptoGuard, CrySL, and another tool (anonymous) using both benchmarks. We present their performance and comparative analysis. The ApacheCryptoAPI-Bench also examines the scalability of the tools. Our benchmarks are beneficial for advancing state-of-the-art solutions in the space of misuse detection.**

***Index Terms*—Cryptographic APIs, Benchmarks, Java**

I. Introduction

Various studies have shown that a vast majority of Java and Android applications misuse cryptographic libraries and APIs, causing devastating security and privacy implications [1]–[5]. The prominent causes for cryptographic misuses are the deficiency in understanding of security API usage [2], [6], complex API designs [6], [7], the lack of cybersecurity training [2], insecure code generation tools and insecure/misleading suggestions in Stack Overflow [2], [8]. The reality is that most developers, with tight project deadlines and short product turnaround time, spend little effort on improving their knowledge or hardening their code for long-term benefits [9]. Recognizing these practical barriers, automatic cryptographic code generation, and misuse detection tools [3] play a significant role in assisting developers with writing and maintaining secure code. The security community has produced several impressive static (e.g., CryptoLint [1], CrySL [10], FixDroid [11], MalloDroid, CryptoGuard [3], Parfait [12]) and dynamic code screening tools (e.g., Crylogger, SMV-Hunter, and AndroSSL) to detect API misuses in Java. The static analysis does not require a program to execute, rather it is performed on a version of the code (e.g., source code, intermediate representations or binary). Many abstract security rules are reducible to concrete program properties that are enforceable via generic static analysis techniques [3].

To advance and monitor the scientific progress in generating effective tools, a mechanism for comparative studies is required. Unfortunately, for detection of cryptographic API misuses, no suitable mechanism or benchmark exists. Such a benchmark needs to have several requirements: cover a wide range of misuse instances, cover interesting program properties (e.g., flow-, context-, field-, path-sensitivity, etc.), written in easily compilable source codes so that both source code and binary code analysis tools can be easily evaluated.

None of the existing benchmarks follows these criteria (e.g., DroidBench, Ghera). For example, DroidBench only contains binaries. Ghera has sources of provided Android apps. However, both DroidBench and Ghera barely cover cryptographic API misuses.

Our contribution is to develop two benchmarks [13], [14]. First, We develop a benchmark named CryptoAPI-Bench that utilizes various interesting program properties (e.g., field-, context-, and path-sensitivity) to produce a diverse set of test cases. We also provide another benchmark named ApacheCryptoAPI-Bench for checking the scalability property of the cryptographic vulnerability detection tools. We run CryptoAPI-Bench and ApacheCryptoAPI-Bench on four static analysis tools (i.e., SpotBugs [15], CryptoGuard, CrySL, and Tool A (anonymous) and perform a comparative analysis of these tools.

II. Crypto Misuse Categories

In this section, we discuss 5 groups of 18 Java cryptographic API misuse categories. We got insights into these misuse categories from previous literature, NIST documents, and other blogs. We describe reasons for vulnerability and possible security solutions for these misuse categories. The considered misuse groups (categories) are: Predictable secrets (cryptographic key, password in PBE, password in KeyStore, credentials in string), vulnerability in SSL/TLS (hostname verifier, certificate validation, SSL socket, HTTP protocol), predictable PRNGs (predictable random number generator, seed in PRNG), vulnerable parameters (salt in PBE, mode of operation, initialization vector, iteration in PBE), vulnerable algorithms (symmetric ciphers, asymmetric ciphers, cryptographic hash, cryptographic MAC)

TABLE I

EVALUATION RESULT ON CRYPTOAPI-BENCH AND APACHECRYPTOAPI-BENCH SHOWS TOOL'S PERFORMANCE ON SIX COMMON VULNERABILITY RULES. PREC. AND REC. STANDS FOR PRECISION AND RECALL. THE VALUES ARE SHOWN IN PERCENTAGE.

Tools	CryptoAPI-Bench				ApacheCryptoAPI-Bench
	Basic Cases		Advanced Cases		
	Prec.	Rec.	Prec.	Rec.	Rec.
SpotBugs	81.25	92.86	0.00	0.00	73.68
CryptoGuard	100.00	100.00	83.33	95.59	78.95
CrySL	58.82	71.43	55.56	58.82	89.47
Tool A*	100.00	92.86	52.00	19.12	78.95

III. DESIGN OF BENCHMARKS

CryptoAPI-Bench: In this section, we present the design of the CryptoAPI-Bench. We manually generate 181 unit test cases guided by 18 types of misuses. We divide all test cases into two types, i.e., basic cases and advanced cases that incorporate possible variations in the perspective of program analysis to detect cryptographic vulnerability.

ApacheCryptoAPI-Bench: We include the early version of real-world large 10 Apache projects to check the scalability property of different tools. The spark project is the largest among 10 considered projects containing 311,856 lines of code. Deltaspike contains the lowest number of lines of code (i.e., 5,116 LoC).

We enlist 121 test cases in ApacheCryptoAPI-Bench. Among them, 79 are basic cases, and 42 are advanced test cases. We detect 88 cryptographic misuses, i.e., true positive alerts. Regarding true negatives, we consider only the cases where a tool shows the case as a false alert. With this consideration, we show 33 true negative cases.

IV. EVALUATION

Performance Evaluation: Our experimental evaluation revealed some interesting insights. For complex cases, specialized tools (e.g., CryptoGuard, CrySL) detect more cryptographic misuses and cover more rules than general-purpose tools (e.g., SpotBugs, Tool A). Currently, none of these tools supports path-sensitive analysis. In the real world codes, the number of basic cases is much higher than advanced cases.

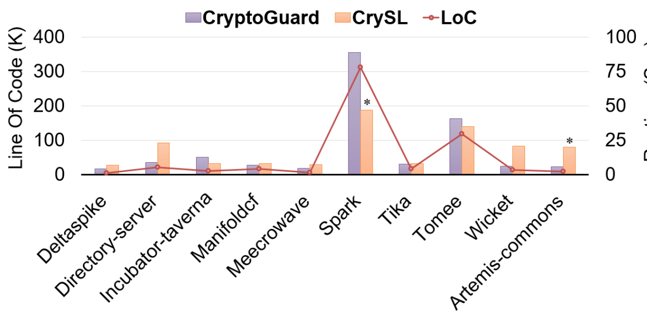

Fig. 1. The runtime of CryptoGuard and CrySL during analyzing Apache projects. Star (*) symbol indicates that the analysis was unsuccessful.

Runtime: Fig. 1 shows the line of code of Apache projects and runtime for CryptoGuard and CrySL. For Tool A and SpotBugs, we use the web version, therefore, we cannot calculate their original runtime for comparison. Among the 8 successful analyzed projects, we find the average runtime for CrySL is 14.64 seconds and CryptoGuard is 11.46 seconds. For the largest Apache project Spark (LoC: 311,856), CryptoGuard successfully analyzes in 88.68 seconds and CrySL shows the failure of analysis report after 46.84 seconds. Overall, SpotBugs and CryptoGuard successfully analyze all 10 Apache projects.

V. CONCLUSION

Scientific, reproducible, and in-depth comparisons are essential components. In this paper, we present CryptoAPI-Bench and ApacheCryptoAPI-Bench to evaluate the accuracy, scalability, and security guarantees of various cryptographic misuse detection tools.

REFERENCES

[1] M. Egele, D. Brumley, Y. Fratantonio *et al.*, "An Empirical Study of Cryptographic Misuse in Android Applications," in *ACM Conference on Computer and Communications Security, CCS'13*, 2013, pp. 73–84.

[2] N. Meng, S. Nagy, D. Yao *et al.*, "Secure Coding Practices in Java: Challenges and Vulnerabilities," in *International Conference on Software Engineering, ICSE'18*, May 2018.

[3] S. Rahaman, Y. Xiao, S. Afrose *et al.*, "CryptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects," in *ACM Conference on Computer and communications security, CCS'19*, Nov. 2019, pp. 2455–2472.

[4] M. Islam, S. Rahaman, N. Meng *et al.*, "Coding practices and recommendations of spring security for enterprise applications," in *2020 IEEE Secure Development (SecDev)*, 2020, pp. 49–57.

[5] Y. Xiao, M. Frantz, S. Afrose *et al.*, "Tutorial: Principles and practices of secure cryptographic coding in java," in *2020 IEEE Secure Development (SecDev)*, 2020, pp. 5–6.

[6] Y. Acar, M. Backes, S. Fahl *et al.*, "Comparing the Usability of Cryptographic APIs," in *IEEE Symposium on Security and Privacy, SP'17, San Jose, CA, USA, May 22-26*, 2017, pp. 154–171.

[7] S. Nadi, S. Krüger, M. Mezini *et al.*, "Jumping Through Hoops: Why Do Java Developers Struggle with Cryptography APIs?" in *International Conference on Software Engineering, ICSE'16*, 2016, pp. 935–946.

[8] Y. Acar, M. Backes, S. Fahl *et al.*, "You Get Where You're Looking for: The Impact of Information Sources on Code Security," in *IEEE Symposium on Security and Privacy, SP'16, San Jose, CA, USA, May 23-25*, 2016, pp. 289–305.

[9] H. Assal and S. Chiasson, "Security in the Software Development Lifecycle," in *Fourteenth Symposium on Usable Privacy and Security, SOUPS'18*, 2018, pp. 281–296.

[10] S. Krüger, J. Späth, K. Ali *et al.*, "CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs," in *European Conference on Object-Oriented Programming, ECOOP'18*, 2018, pp. 10:1–10:27.

[11] D. C. Nguyen *et al.*, "A Stitch in Time: Supporting Android Developers in Writing Secure Code," in *ACM Conference on Computer and Communications Security, CCS'17*, 2017, pp. 1065–1077.

[12] Y. Xiao, Y. Zhao, N. Allen *et al.*, "Industrial Experience of Finding Cryptographic Vulnerabilities in Large-Scale Codebases," *Digital Threats*, dec 2021, just Accepted. [Online]. Available: https://doi.org/10.1145/3507682

[13] S. Afrose, Y. Xiao, S. Rahaman *et al.*, "Evaluation of Static Vulnerability Detection Tools with Java Cryptographic API Benchmarks," *IEEE Transactions on Software Engineering*, pp. 1–1, 2022.

[14] S. Afrose, S. Rahaman, and D. Yao, "CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses," in *2019 IEEE Cybersecurity Development (SecDev)*, 2019, pp. 49–61.

[15] "SpotBugs: Find Bugs in Java Programs," https://spotbugs-.github.io/, online; Last accessed: Dec 3, 2020.