POSTER: Deployment-quality and Accessible Solutions for Cryptography Code Development

Sazzadur Rahaman[1], Ya Xiao[1], Sharmin Afrose[1], Ke Tian[1], Miles Frantz[1], Na Meng[1],
Barton P. Miller[2], Fahad Shaon[3], Murat Kantarcioglu[3], Danfeng (Daphne) Yao[1]
[1]Computer Science, Virginia Tech, Blacksburg, VA
[2] Computer Science, University of Wisconsin-Madison, Madison, WI
[3] Computer Science, University of Texas at Dallas, Dallas, TX
{sazzad14,yax99,sharminafrose,ketian,frantzme,nm8247,danfeng}@vt.edu,
bart@cs.wisc.edu,{fahad.shaon,muratk}@utdallas.edu

ABSTRACT

Cryptographic API misuses seriously threaten software security. Automatic screening of cryptographic misuse vulnerabilities has been a popular and important line of research over the years. However, the vision of producing a scalable detection tool that developers can routinely use to screen millions of line of code has not been achieved yet.

Our main technical goal is to attain a high precision and high throughput approach based on specialized program analysis. Specifically, we design inter-procedural program slicing on top of a new on-demand flow-, context- and field- sensitive data flow analysis. Our current prototype named CryptoGuard can detect a wide range of Java cryptographic API misuses with a precision of 98.61%, when evaluated on 46 complex Apache Software Foundation projects (including, Spark, Ranger, and Ofbiz). Our evaluation on 6,181 Android apps also generated many security insights. We created a comprehensive benchmark named CryptoApi-Bench with 40-unit basic cases and 131-unit advanced cases for in-depth comparison with leading solutions (e.g., SpotBugs, CrySL, Coverity). To make CryptoGuard widely accessible, we are in the process of integrating CryptoGuard with the Software Assurance Marketplace (SWAMP). SWAMP is a popular *no-cost* service for continuous software assurance and static code analysis.

CCS CONCEPTS

• **Security and privacy → Software and application security**.

KEYWORDS

Accuracy; Cryptographic API Misuses; Static Program Analysis; False Positive; False Negative; Benchmark; Java;

1 INTRODUCTION

Cryptography offers provable security guarantees in the presence of adversaries. Various software libraries and frameworks provide a variety of cryptographic APIs to support secure coding. Cryptographic API misuses, such as exposed secrets, predictable random numbers, and vulnerable certificate verification, have critical impact on software security [7–9, 12].

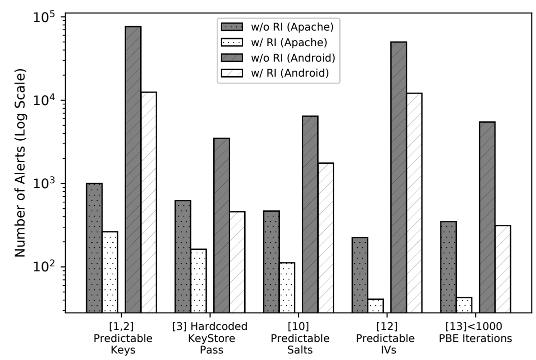


Figure 1: Reduction of false positives with refinement insights in 46 Apache projects and 6,181 Android apps. Top 6 rules with maximum reductions are shown [13].

The research solution that we aim in this project addresses the pervasive problem of cryptographic coding vulnerabilities in real-world software. Specifically, our ongoing goals are to produce high quality code screening tools and make them accessible to the developers in various convenient form, including, standalone, IDE plugin (e.g., Eclipse, IntelliJ IDEA), build tool plugin (e.g., Gradle, Maven), code screening as a service (e.g., Software Assurance Marketplace aka, SWAMP).

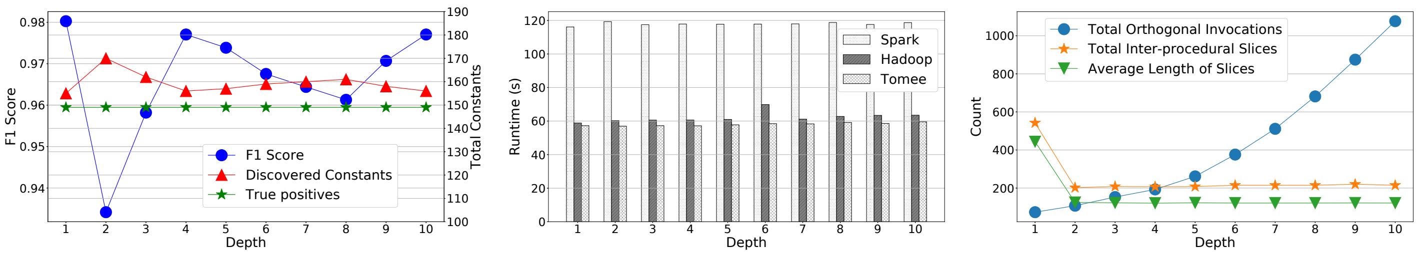
We have made substantial progress toward building a high accuracy and low runtime static analysis solution for detecting 16 types of cryptographic and SSL/TLS API misuse vulnerabilities. The main technical enabler is the use of highly optimized forward and backward program slicing algorithms, which are built on top of on-demand flow-, context- and field-sensitive data-flow analysis [13].

Threat model. Our prototype CryptoGuard [13][1] aims to detect 16 types of Cryptographic and SSL/TLS API misuses. It detects three types of predictable secrets (i.e., symmetric keys and passwords), four types of SSL/TLS MitM attacks, two types of predictability of PRNGs, three types of chosen-ciphertext attacks (i.e., static salts and IVs, ECB mode of symmetric ciphers) and 4 types

[1]Available at https://github.com/CryptoGuardOSS/cryptoguard

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11.
https://doi.org/10.1145/3319535.3363252



(a) (b) (c)

Figure 2: The impact of the orthogonal exploration depth on F1 scores and the number of discovered constants in (a), runtime in (b), and analysis properties in (c) for 8 rules.

of brute-force attacks (i.e., less than 1000 password-based encryption (PBE) iterations, insecure symmetric, asymmetric, and cryptographic hashes). We also categorize their severity into high, medium, and low, based on i) attacker's gain and ii) attack difficulty. Vulnerabilities from predictable secrets, SSL/TLS MitM, and insecure Hash are immediately exploitable, hence are classified as high risks. Vulnerabilities from predictability and CPA provide substantial advantages to attackers by significantly reducing attack efforts [14]. They are at medium-level risks. Brute-forcing ciphers, requiring non-trivial effort, is low risk.

Detection accuracy. Most of the cryptographic vulnerabilities in our threat model require finding constants. To improve detection accuracy, our inter-procedural data-flow analysis adopts a set of refinement insights that systematically discard false alerts. These *refinement insights (RI)* are deduced by observing common programming idioms and language restrictions to remove irrelevant elements, i.e., resource identifiers, arguments about states of operations, constants on infeasible paths, and bookkeeping values. For eight of our rules, these refinement algorithms reduce the total number of alerts by 76% in Apache and 80% in Android (Figure 1). Our manual analysis shows that CryptoGuard has a precision of 98.61% on Apache [13].

Our analysis shows that the adoption of these refinement insights is often more useful to clip orthogonal exploration in order to achieve better performance.

We measure the impact of the orthogonal exploration, we conducted an experiment with 30 Apache root-subprojects and varied the clipping of the exploration from depth 1 to 10 (Figure 2) [13]. The total number of discovered constants across all projects increases slightly with the depth (Figure 2(a) right Y-axis). However, our manual analysis revealed that none of the new constants is a true positive. Thus, the increase of the orthogonal exploration depth does not improve the recall in this specific experiment, causing a decrease in the F1 score (Figure 2(a) left Y-axis). Interestingly, the runtime does not increase with the increasing depth (Figure 2(b)). Figure 2(c) shows that the number of inter-procedural slices and their average sizes are drastically reduced when the depth increases from 1 to 2. The reason is that when the analysis explores inside a method, influences on an argument the method's specific invocation might become irrelevant. Given these observations, we set the orthogonal exploration depth to 1 for the rest of our experiments, as it returns the fewest number of irrelevant constants.

Runtime overhead and coverage. Existing flow-, context- and field-sensitive analysis techniques build a super control-flow graph of the entire program, which has a significant impact on runtime. In contrast, our on-demand slicing algorithms run much faster, which start from the slicing criteria and only propagate to the methods that have the potential to impact security. Hence, a large portion of the code base is not touched. For Apache projects, the average runtime was 3.3 minutes with a median of around 1 minute. For Android apps, we terminated unfinished analysis after 10 minutes. The average runtime was 3.2 minutes with a median of 2.85 minutes [13].

Comparison with other tools. We construct CryptoApi-Bench [6][2], a comprehensive benchmark with 171 cases for comparing the quality of cryptographic vulnerability detection tools. CryptoApi-Bench covers 16 types of cryptographic misuses. In CryptoApi-Bench, there are 40 basic test cases and 131 advanced test cases. Experimental evaluation in Table 1 shows that CryptoGuard outperforms the state-of-the-art open source and commercial solutions in this space, including CrySL [11], SpotBugs [2], and the free online version of Coverity [1], in terms of precision and recall [13]. For runtime comparison, we ran CrySL and CryptoGuard on 10 randomly selected Apache projects. Unfortunately, CrySL crashed and exit prematurely for 7 of them. For the 3 completed projects, CrySL is slower, but comparable on 2 projects (5 vs. 3 seconds, 25 vs. 19 seconds). However, it is 3 orders of magnitude slower than CryptoGuard on `kerbaros-codec` [13]. During our experiments, we use CrySL 2.0 (commit id *5f531d1*), SpotBugs 3.1.0 (from SWAMP) and the results from Coverity was obtained before Mar 29, 2019. CrySL is an actively maintained project and got benefited from our benchmark effort to improve its performance [3].

New security findings. Using our research prototype CryptoGuard, we have successfully screened 46 large open source projects on Apache Software Foundation and 6,181 Android apps from Google Play Market. We discovered a wide range of security issues in real-world coding practices. In Apache projects, there is a widespread insecure practice of storing plaintext passwords in code or in configuration files. Insecure uses of SSL/TLS APIs are set as the default configuration in some cases. For some of them, end-users are susceptible to use insecure default configurations due to lack of proper warning or documentations. In Android apps, 95% of the vulnerabilities come from the libraries that are packaged

[2] Available at https://github.com/CryptoGuardOSS/cryptoapi-bench

Table 1: CryptoApi-Bench comparison of CrySL, Coverity, SpotBugs and CryptoGuard on six common threat models with CryptoApi-Bench's common 20 basic and 84 advanced cases. GT, TP, FP, FN, FPR, FNR stand for ground truth, true positive, false positive and false negative, false positive rate, false negative rate, respectively. Detailed comparison can be found in [6].

Tools	Basic Benchmark							Advanced Benchmark						
	GT:14			Result Summary				GT: 68			Result Summary			
	TP	FP	FN	FPR(%)	FNR(%)	Prec.(%)	Rec.(%)	TP	FP	FN	FPR(%)	FNR(%)	Prec.(%)	Rec.(%)
CrySL	10	6	4	50	28.57	62.5	71.43	40	32	28	66.67	41.18	55.56	58.82
Coverity	13	0	1	0	7.14	100.00	92.86	13	12	55	42.86	80.88	52.00	19.12
SpotBugs	13	0	1	0	7.14	100.00	92.86	0	22	68	57.89	100.00	0.00	0.00
CryptoGuard	13	0	1	0	7.14	100.00	92.86	65	13	3	44.83	4.41	83.33	95.59

with the applications. Some libraries are from renowned software firms.

However, these security issues are the tip of the insecure coding iceberg. Through our disclosure interactions with developers and observations from StackOverflow forum [4, 5, 12], we found that a substantial number of developers did not appear to understand the concepts or implications of security API usage. The unfortunate reality is that most developers, with tight project deadlines and short product turnaround time, are not willing to spend effort on hardening their code for long-term benefits. Thus, it is unrealistic to assume that developers will better themselves on their own without any external help.

Ongoing and future work. We aim to transition secure cryptographic coding research solutions to practice. Our ongoing effort is to make code screening convenient and accessible to mass developers. We are integrating CryptoGuard with Software Assurance Marketplace (SWAMP), one of the most popular free-of-cost services for continuous software assurance and static code analysis. In SWAMP, programmers can access over 30 scanning tools for a wide variety of languages and platforms. Typically, developers upload their codes or binaries to SWAMP for analysis. There is also a locally installable version of the SWAMP, called SWAMP-in-a-Box, for users that cannot upload their code to an external facility. Each week, the SWAMP performs thousands of assessments, and hundreds of copies of SWAMP-in-a-Box have been downloaded. After successful integration, SWAMP will be able to offer a comprehensive cryptographic misuse detection service to thousands of its users. We plan to create CryptoGuard plugins for popular Java IDE environments, namely IntelliJ IDEA and Eclipse. The only cryptography-related IDE is Eclipse's CogniCrypt plugin, which is for a code assistant tool (i.e., auto-complete of crypto APIs) [10], not for vulnerability detection. We also plan to create CryptoGuard plugins for Apache Maven and Gradle. Enabling crypto code screening in the early stages of the software development cycle will be more effective. Orthogonally, we plan to upgrade relevant Java static analysis tools (namely, Soot) to newer versions of Java, which will generate impact beyond the specific crypto problem [3].

2 ACKNOWLEDGMENT

This project was supported in part by NSF grant CNS-1929701 and ONR Grant N00014-17-1-2498.

[3]The current Soot does not support Java 9 or above.

REFERENCES

[1] Coverity Static Application Security Testing (SAST). https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html. Accessed: January 28, 2019.
[2] SpotBugs: Find Bugs in Java Programs. https://spotbugs.github.io/. Accessed: January 15, 2019.
[3] Headless test cases for CryptoGuard Crypto-API Benchmark. "https://github.com/CROSSINGTUD/CryptoAnalysis/issues/134", 2019. [Online; accessed Aug 25, 2019].
[4] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. Comparing the Usability of Cryptographic APIs. In *IEEE S&P'17*, pages 154–171, 2017.
[5] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *IEEE S&P'16*, 2016.
[6] S. Afrose, S. Rahaman, and D. Yao. CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses. In *IEEE Secure Development Conference (SecDev)*, September 2019.
[7] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An Empirical Study of Cryptographic Misuse in Android Applications. In *ACM CCS'13*, 2013.
[8] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben. Why Eve and Mallory Love Android: An Analysis of Android SSL (in) Security. In *ACM CCS'12*, pages 50–61, 2012.
[9] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software. In *ACM CCS'12*, 2012.
[10] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler, and R. Kamath. CogniCrypt: Supporting Developers in Using Cryptography. In *IEEE/ACM ASE'17*, pages 931–936, 2017.
[11] S. Krüger *et al.* CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs. In *ECOOP'18*, 2018.
[12] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Arango-Argoty. Secure Coding Practices in Java: Challenges and Vulnerabilities. In *ACM ICSE'18*, 2018.
[13] S. Rahaman, Y. Xiao, S. Afrose, F. Shaon, K. Tian, M. Frantz, M. Kantarcioglu, and D. Yao. CryptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects. In *ACM CCS'19*, 2019.
[14] S. Rahaman and D. Yao. Program Analysis of Cryptographic Implementations for Security. In *IEEE Secure Development Conference (SecDev)*, pages 61–68, 2017.