Industrial Strength Static Detection for Cryptographic API Misuses

Ya Xiao*, Yang Zhao†, Nicholas Allen†, Nathan Keynes†, Danfeng (Daphne) Yao*, Cristina Cifuentes†
*Computer Science Department, Virginia Tech, Blacksburg, VA, USA
†Oracle Labs, Brisbane, QLD, Australia
{yax99, danfeng}@vt.edu, {nicholas.allen, cristina.cifuentes}@oracle.com

***Abstract*—We describe our experience of building an industrial-strength cryptographic vulnerability detector, which aims to detect cryptographic API misuses in Java™[1]. Based on the detection algorithms of the academic tool CryptoGuard, we integrated the detection into the Oracle internal code scanning platform Parfait. The goal of the Parfait-based cryptographic vulnerability detection is to provide precise and scalable cryptographic code screening for large-scale industrial projects. We discuss the needs and challenges of the static cryptographic vulnerability screening in the industrial environment.**

***Index Terms*—cryptographic vulnerability detection, static analyzer, industrial environment**

I. Introduction

Cryptographic vulnerabilities that are caused by improper or flawed cryptography implementation have become one of the most serious threats [1]–[5]. Because of insufficient security expertise, developers face challenges to understand the implicit security rules behind the cryptographic APIs, such as Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE) libraries [6], [7]. Misuses of these cryptographic APIs could result in various security vulnerabilities, such as exposing secrets (e.g., password, key), bypassing authentication [8], [9].

To aid this situation, static analysis is widely used to screen the code and expose cryptographic vulnerabilities. Many tools, such as CryptoLint [4], FixDroid [10], CogniCrypt [11], CryptoGuard [12], are presented for this purpose. Despite the progress, the acceptance and prevalence of these tools in the industrial community are still low [13], which suggests a gap between the state-of-the-art tools and the industrial demands. In this work, we focus on an industrial-strength cryptographic vulnerability detector. We realize the high-precision detection algorithm presented by an academic tool CryptoGuard [12] on the support of Parfait [14], an Oracle internal static code analysis platform designed for large-scale codebases. Our Parfait-based cryptographic vulnerability detection shows nearly perfect precision and excellent scalability on large-scale industrial projects.

II. Industrial-strength Precision and Scalability

Most of the cryptographic vulnerabilities we focus on can be attributed to assigning improper values to certain security-critical parameters used with cryptographic API calls. For example, a cryptographic API call `new PBEKeySpec(password)` (an API method that generates a cryptographic key from a given password) accepts a hard-coded password. The detection requires a backward dataflow analysis to figure out the values assigned to the security-critical parameters. Our backward dataflow analysis follows the interprocedural, finite, distributive subset (IFDS) analysis algorithm presented by Reps *et al.* [15]. We briefly introduce our technical enablers for the high precision and scalability. Please check the full paper [16] for the details.

High Precision. The technical enabler for our high precision is the refined slicing algorithm presented in CryptoGuard [12]. When detecting the hard-coded security-critical parameters (e.g., secret key, password), the precision challenge is caused by a phenomenon, referred to as pseudo-influences [12]. Pseudo-influences are the constants captured by the backward dataflow analysis, however, have non-security impacts. For example, a file location constant is used to retrieve a cryptographic key. With the refinement insights given in CryptoGuard [12], we are able to remove five language-specific scenarios that involve pseudo-influences without resulting in hard-coded values. These pseudo-influences include state indicators, resource identifiers, and bookkeeping indices to retrieve the value. The contextually incompatible constants, and constants in infeasible paths are also removed by the refinement insights. Table I shows the detection results with and without the refinement insights on a well-known cryptographic vulnerability benchmark, CryptoAPI-Bench [17], [18]. The refinement insights are able to reduce all false positives except for the test cases that require path sensitivity to detect.

Scalability. Industrial projects are usually at a large scale, which results in a higher requirement for scalability. Our Parfait-based cryptographic vulnerability detection achieves excellent scalability by two designs, the *layered scheduler for caller methods*, and the *summarization for callee methods.* An interprocedural analysis might go across multiple methods, which makes the analysis take too much time. Instead of running the analyses one after another, Parfait offers a layered framework to optimize the execution order. The interprocedural analyses are broken down into several layers and scheduled layer by layer. In this way, the analyses requiring less time

[1]Java is a registered trademark of Oracle and/or its affiliates.

2022 IEEE Secure Development Conference (SecDev) | 978-1-6654-7182-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/SECDEV53368.2022.00022

978-1-6654-7182-4/22/$31.00 ©2022 IEEE
DOI 10.1109/SecDev53368.2022.00022

TABLE I
FALSE POSITIVE REDUCTION DERIVED FROM APPLYING THE REFINEMENT INSIGHTS (RIS). WE COMPARE PARFAIT CRYPTOGRAPHIC VULNERABILITY DETECTION WITH ITS INTERMEDIATE VERSION WITHOUT THE REFINEMENT INSIGHTS.

Type	**Vulnerabilities**	**FPs** (w/o RIs)/(w RIs)
Basic	24	1/0
Multiple Methods	56	3/0
Multiple Classes	18	1/0
Field Sensitivity	18	2/0
Path Sensitivity	0	19/19
Heuristics	9	12/0
Total	**125**	**38/19**

can be finished first. It guarantees that more vulnerabilities can be reported within less time. Another design improving the scalability is the summarization mechanism for the callee methods. When a callee method is explored by the analysis, we generate the summary edges for the callee method and store them for future usage to avoid re-exploration. We scan 11 real world projects provided by Oracle. Our detector achieves an average runtime 338.8s for the 11 projects with average 395.4k line of code. Besides, our detector reports 42 vulnerabilities with 0 false positives, achieving 100% precision.

```
1 public class DesEncrypter{
2   private byte[] salt = { (byte) 0xC9, (byte) 0xDB
    , (byte) 0xA3, (byte) 0x52, (byte) 0x56, (byte)
    0x35, (byte) 0xE8, (byte) 0xB0};
3   private int iterationCount = 20;
4   public DesEncrypter(final String passPhrase){
5       initDesEncrypter(passPhrase);}
6   private void initDesEncrypter(final String
    passPhrase){
7   ...
8   AlgorithmParameterSpec paramSpec = new
    PBEParameterSpec(salt,iterationCount);}}
```

Listing 1. A real-world vulnerability about using constant salt and insufficient iteration count (We modified the code to make the codebase unidentifiable.)

Listing 1 shows vulnerabilities of using constant salt and insufficient iteration count as PBE parameters. This case represents the most common vulnerable pattern of the sensitive cryptographic materials (e.g., passwords, salts, IVs, etc) to be hard-coded in the initialization.

III. FUTURE WORK

Based on our interaction with developers, there are many future directions required to do to close the gap between industrial demands and state-of-the-arts. One direction is to generate more useful fixing suggestions. Many developers pointed out that the fixing suggestions generated by the current tools are too simple and not useful enough. Developers have difficulties correcting the cryptographic code even though the vulnerabilities are noticed [19]. Another interesting direction is the context-aware cryptographic vulnerability detector. Many developers pointed out that the alerts generated by the state-of-the-art tools ignore the context of the cryptographic APIs, which may overestimate the security threats.

ACKNOWLEDGMENT

Virginia Tech authors have been partly supported by the National Science Foundation under Grant No. CNS-1929701.

REFERENCES

[1] 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 *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 289–305.

[2] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. A. Argoty, "Secure Coding Practices in Java: Challenges and Vulnerabilities," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 372–383.

[3] 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 *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 38–49.

[4] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An Empirical Study of Cryptographic Misuse in Android Applications," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 73–84.

[5] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1296–1310.

[6] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through Hoops: Why do Java Developers Struggle with Cryptography APIs?" in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 935–946.

[7] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the Usability of Cryptographic APIs," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 154–171.

[8] *OWASP Top Ten*, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[9] *CWE Top 25 Most Dangerous Software Errors*, 2020. [Online]. Available: https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html

[10] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl, "A Stitch in Time: Supporting Android Developers in Writing Secure Code," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1065–1077.

[11] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler *et al.*, "Cognicrypt: Supporting developers in using cryptography," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 931–936.

[12] 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 *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2455–2472.

[13] Y. Zhang, M. M. A. Kabir, Y. Xiao, D. D. Yao, and N. Meng, "Automatic Detection of Java Cryptographic API Misuses: Are We There Yet," *IEEE Transactions on Software Engineering*, 2022.

[14] C. Cifuentes and B. Scholz, "Parfait: Designing a Scalable Bug Checker," in *Proceedings of the 2008 workshop on Static analysis*, 2008, pp. 4–11.

[15] T. Reps, S. Horwitz, and M. Sagiv, "Precise Interprocedural Dataflow Analysis via Graph Reachability," in *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1995, pp. 49–61.

[16] Y. Xiao, Y. Zhao, N. Allen, N. Keynes, D. Yao, and C. Cifuentes, "Industrial Experience of Finding Cryptographic Vulnerabilities in Large-scale Codebases," *Digital Threats: Research and Practice*, 2020.

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

[18] S. Afrose, Y. Xiao, S. Rahaman, B. Miller, and D. D. Yao, "Evaluation of Static Vulnerability Detection Tools with Java Cryptographic API Benchmarks," *IEEE Transactions on Software Engineering*, 2022.

[19] Y. Xiao, S. Ahmed, W. Song, X. Ge, B. Viswanath, and D. Yao, "Embedding Code Contexts for Cryptographic API Suggestion: New Methodologies and Comparisons," *arXiv preprint arXiv:2103.08747*, 2021.