Tutorial: Principles and Practices of Secure Crypto Coding in Java*

Sazzadur Rahaman Na Meng Danfeng (Daphne) Yao
Department of Computer Science
Virginia Tech
{*sazzad14, nm8247, danfeng*}*@vt.edu*

***Abstract*—Various software libraries and frameworks provide a variety of APIs to support secure coding. However, misusing these APIs can cost developers tremendous time and effort, introduce security vulnerabilities to software, and cause serious consequences like data leakage or Denial of Service (DoS) on servers. No prior tutorial educates people on the best practice of secure coding, the pitfalls that should be avoided, and the remediation of insecure code.**

To increase the security awareness of developers and improve the quality of their software products, we propose a 90-minute tutorial to teach participants the principles and practices of Java secure coding. In this tutorial, we will introduce the principles of using security APIs, analyze typical API misuse cases to explain the causes and effects. We will also introduce a tool that we recently developed to automatically detect API misuse in Java.

There are five parts in our tutorial. To reveal the existing status of secure coding practice, we will first introduce the findings in our recent study on StackOverflow posts relevant to Java security. Second, we will discuss the recommended principles of API usage by security experts. Third, to correlate the principles with existing practice, we will also discuss some API misuse examples for the hash digest, message encryption and decryption, key generation, and SSL/TLS connection. Fourth, we will ask participants to examine extra code examples and discuss the security property of each example. Finally, We will give an overview of the available tools and resources, demonstrate a tool named RIGORITYJ that we recently developed to automatically detect API misuse in Java. We will also help participants install and use RIGORITYJ on their own machines and ask them for trials.

By actively involving participants in code discussion and tool trial, we aim to raise the security awareness among developers, improve their secure coding capabilities, and equip them with the tools they need for secure coding.

1. Introduction

Various software libraries or frameworks (e.g., BouncyCastle [6]) provide a variety of features to enable secure coding. For instance, Java platform defines the Java Cryptography Architecture (JCA), which contains APIs for hashes, keys, certificates, digital signatures, and encryption [10]. Java Secure Socket Extension (JSSE) includes APIs for standard secure communication protocols like SSL/TLS [1]. Misusing these libraries and frameworks not only slows development time, but also leads to security vulnerabilities in the resulting software [4], [15].

Our recent study on StackOverflow posts revealed a worrisome reality in the software development industry [11]. Based on the discussions among developers on StackOverflow, we found that a substantial number of developers did not appear to understand the concepts or implications of security API usage. Such lack of domain knowledge creates frustration in developers, who sometimes ended up using APIs in easy but completely insecure ways. Additionally, we collected substantial empirical evidence showing that (1) the security API usage is over complicated and poorly documented; (2) the error reporting systems of Java platform security APIs cause confusion; and (3) the understanding on multi-language support for secure coding is rather weak. This indicates a strong need of better secure coding education for developers, to raise their security awareness when designing, implementing, and using security APIs.

Till date, the number of studies offering automatic verification for secure use of cryptography is fairly limited (e.g., CryptoLint for Android [8], FixDroid for Android IDE [12], RIGORITYJ [13] and CogniCrypt [17] for Java, TaintCrypt for C/C++ [14], etc.). Moreover, there is no prior tutorial session focusing secure cryptographic coding practice in Java. Existing online tutorials and documents introduce some best practices of using security APIs (e.g., [1], [2], [3]). Nevertheless, these documents do not include many code examples to thoroughly demonstrate the correct API usage, neither do they discuss or explain developers' frequent mistakes when using the APIs. Online forums like StackOverflow [16] contain sufficient code snippets to demonstrate and recommend security API usage, but some of the recommended usage is incorrect or insecure [5].

To create mass awareness, we propose a 90-minute tutorial on the principles and practices of secure coding with proper usage of cryptographic APIs. In this tutorial, we will include hands-on components and audience interactions to ensure the quality of participants' learning experience. Specifically, there are five parts in the tutorial.

Part I: Review of Existing Secure Coding Practices. To emphasize the importance of secure coding practice,

**This work has been supported by ONR Grant N00014-17-1-2498.*

we will introduce the key findings in our recent study on StackOverflow posts [11]. The study examined the common questions and solutions frequently discussed about Java secure coding practice, and made observations from two perspectives: software engineering (SE) and security. Such findings provide actionable advices to library builders for better API design and to application developers on avoiding API misuses for better security.

Part II: Introduction of Secure Coding Principles. A number of security literatures introduced various rules to use APIs securely [1], [8], [9], [10]. We will selectively introduce four to six well-defined, popularly-used, and easy-to-understand rules (e.g., SSL/TLS misconfiguration, Hard-coded keys, IVs etc). By introducing these basic important principles, we aim to provide participants with the necessary domain knowledge of secure coding.

Part III: Explanation of Insecure API usage. To improve participants' understanding of the introduced principles, we will also present some counterexamples to explain why they violate the general principles, and how to fix the API misuse. For instance, a counterexample can be a case where a password is synthesized with several predictable components. Although the password is not hardcoded, still the predictability can cause leaking the password. These counterexamples compliment our instructions on the best practices, concretize the concepts in code, and clarify the boundaries between secure and insecure API usage.

Part IV: Discussion of Code Examples. Different from existing online tutorials and API documentation that impart knowledge in one way, our tutorial is unique because we aim to establish a two-way communication between the instructors and the audience. By showing some code examples that are secure or insecure, we will ask participants to discuss (1) whether each example is secure or not, and (2) how to fix problems if a case is insecure. In this way, we will guide participants to correlate the abstract security concepts in mind with the coding practice in life. Additionally, we can also leverage this opportunity to check how participants digest our instructions, and may clarify some essential concepts if participants are confused.

Part V: Demonstration of Automatic Detectors. We will introduce several misuse detection tools in Java (e.g., (e.g., FixDroid [12], CogniCrypt [17], RIGORITYJ [13], DHS SWAMP [7]). To provide hands on experience, we will demonstrate RIGORITYJ [13]. Given a Java program, RIGORITYJ conducts program analysis to check whether the code violates any of the known principles for cryptographic operations. We will walk the participants through the procedure of installing, configuring and using RIGORITYJ to identify insecure code.

In summary, there are three learning objectives we aim to achieve with this tutorial:

1) To increase the security awareness in developers,
2) to improve developers' capability of diagnosing and fixing insecure code, and
3) to enrich developers' hands-on experience of using security bug detectors in the coding practice.

2. Prerequisites and Target Audience

To ensure the quality of hands-on experience, we expect people to have the basic knowledge of Java programming. Participants are highly recommended to bring their laptops and try out our research tool.

Security researchers, software developers, students, and other software professionals at all levels are welcome to attend the tutorial. For the participants that have little domain knowledge of security, our tutorial will cover a set of interesting information about the principles of secure coding, the existing status of secure coding practice, the gap between the principles and practices, and the potential solutions to narrow the gap (e.g., developer education and tool support). For security experts and security library builders, this tutorial will be also valuable, because it provides a platform for people to share their expertise and learn about the needs of better software support for secure coding practice.

References

[1] Java Secure Socket Extension (JSSE) Reference Guide. https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html#Introduction.

[2] Security Tips. https://developer.android.com/training/articles/security-tips.html#Crypto.

[3] Security with HTTPS and SSL. https://developer.android.com/training/articles/security-ssl.html.

[4] State of software security. https://www.veracode.com/sites/default/files/Resources/Reports/state-of-software-security-volume-7-veracode-report.pdf, 2016.

[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*, pages 289–305, 2016.

[6] Bouncy castle. https://www.bouncycastle.org.

[7] Welcome to the SWAMP. https://continuousassurance.org, 2018.

[8] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in Android applications. In *ACM CCS*, pages 73–84, 2013.

[9] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl. Stack Overflow considered harmful? The impact of copy&paste on Android application security. In *IEEE S&P*, 2017.

[10] Java cryptography architecture. http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html.

[11] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. A. Argoty. Secure Coding Practices in Java: Challenges and Vulnerabilities. In *ACM ICSE*, 2018.

[12] 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 *ACM CCS*, pages 1065–1077. ACM, 2017.

[13] S. Rahaman, Y. Xiao, K. Tian, F. Shaon, M. Kantarcioglu, and D. Yao. RIGORITYJ: Deployment-quality detection of java cryptographic vulnerabilities. *arXiv preprint arXiv:1806.06881*, 2018.

[14] S. Rahaman and D. Yao. Program analysis of cryptographic implementations for security. In *IEEE SecDev*, pages 61–68, 2017.

[15] F. Y. Rashid. Library misuse exposes leading Java platforms to attack. http://www.infoworld.com/article/3003197/security/library-misuse-exposes-leading-java-platforms-to-attack.html, 2017.

[16] StackOverflow. https://stackoverflow.com.

[17] S. K. *et al.* Cognicrypt: supporting developers in using cryptography. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017*, pages 931–936, 2017.