How Reliable is the Crowdsourced Knowledge of Security Implementation?

Mengsu Chen

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Computer Science and Application

Na Meng, Chair
Eli Tilevich
Danfeng Yao

December 6, 2018
Blacksburg, Virginia

Keywords: Stack Overflow, crowdsourced knowledge, social dynamics, security implementation, clone detection

Copyright 2019, Mengsu Chen

How Reliable is the Crowdsourced Knowledge of Security Implementation?

Mengsu Chen

(ABSTRACT)

The successful crowdsourcing model and gamification design of Stack Overflow (SO) Q&A platform have attracted many programmers to ask and answer technical questions, regardless of their level of expertise. Researchers have recently found evidence of security vulnerable code snippets being possibly copied from SO to production software. This inspired us to study how reliable is SO in providing secure coding suggestions.

In this project, we automatically extracted answer posts related to Java security APIs from the entire SO site. Then based on the known misuses of these APIs, we manually labeled each extracted code snippets as secure or insecure. In total, we extracted 953 groups of code snippets in terms of their similarity detected by clone detection tools, which corresponds to 785 secure answer posts and 644 insecure answer posts. Compared with secure answers, counter-intuitively, insecure answers has higher view counts (36,508 vs. 18,713), higher score (14 vs. 5), more duplicates (3.8 vs. 3.0) on average. We also found that 34% of answers provided by the so-called trusted users who have administrative privileges are insecure.

Our finding reveals that there are comparable numbers of secure and insecure answers. Users cannot rely on community feedback to differentiate secure answers from insecure answers either. Therefore, solutions need to be developed beyond the current mechanism of SO or on the utilization of SO in security-sensitive software development.

How Reliable is the Crowdsourced Knowledge of Security Implementation?

Mengsu Chen

(GENERAL AUDIENCE ABSTRACT)

Stack Overflow (SO), the most popular question and answer platform for programmers today, has accumulated and continues accumulating tremendous question and answer posts since its launch a decade ago. Contributed by numerous users all over the world, these posts are a type of crowdsourced knowledge. In the past few years, they have been the main reference source for software developers. Studies have shown that code snippets in answer posts are copied into production software. This is a dangerous sign because the code snippets contributed by SO users are not guaranteed to be secure implementations of critical functions, such as transferring sensitive information on the internet. In this project, we conducted a comprehensive study on answer posts related to Java security APIs. By labeling code snippets as secure or insecure, contrasting their distributions over associated attributes such as post score and user reputation, we found that there are a significant number of insecure answers (644 insecure vs 785 secure in our study) on Stack Overflow. Our statistical analysis also revealed the infeasibility of differentiating between secure and insecure posts leveraging the current community feedback system (eg. voting) of Stack Overflow.

Dedication

To my parents

To my wife and daughter

Acknowledgments

I would like to express my sincere gratitude to my advisor, Prof. Na Meng. It is her continuous guidance and timely feedback that make this work possible. I greatly appreciate her help in preparing the corresponding technical paper of this thesis and submitting to the conference. I would like to thank my collaborators: Felix Fischer, Prof. Xiaoyin Wang, Prof. Jens Grossklags for their contributions that make this thesis in its current shape. I would like to extend my thanks to my committee members Prof. Eli Tilevich and Prof. Danfeng Yao for their valuable comments. I also would like to thank Prof. Wu Feng for his recommendation and the opportunity to be involved in HPC researches. My special thanks go to Prof. Vito Scarola for providing the opportunity to study computer science in addition to my research.

Lastly, I would like to thank my parents for their unconditional support. I would like to thank my beautiful wife Naling for her company and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Question and answer (Q&A) websites have been popularly used by software developers to discuss technical problems. Even though there are still websites providing answers from only verified experts, crowdsourcing has been the approach used by many websites to build a community that can provide more answers and faster responses to questions. The gamification design on top of the crowdsourcing model further incents users to give answers regardless of their level of expertise or where they obtain the answers. Although such mechanism can help to filter out "better" answers based on community voting, when it comes to more subtle issues such as software security, it is unclear if such crowdsourced knowledge is still reliable.

As the most popular Q&A platform for software developers since its launch a decade ago, Stack Overflow (SO) is also built on top of the crowdsourcing and gamification model [2, 3]. It contains answers contributed by unknown developers with different background and level of expertise. Consequently, the abundance of answers attracts developers to use it as one of the main reference resource [4, 5]. Search engines also recommend SO to developers who searching for direct answers for the problem they have encountered with [4, 6]. Researchers have seen duplicated codes in production software. Some of them are actually copied SO, according to the interview with some software developers [7]. On the other hand, not all code snippets are secure, because not all the users of SO are programmer expertise, a natural result of the crowdsourcing model. Researchers have found that even some highly upvoted and accepted answers can contain insecure code [8, 9]. Such insecure code examples will

certainly impact the security of production software if developers just reused the code. In a recent work, Fischer *et al.* found security vulnerable code snippets on SO were copied into 196,403 Android applications available on Google Play [8]. Some of these vulnerable code snippets were successfully attacked resulting in user credentials, credit card numbers and other private data stolen. Such correlation between the insecure code examples on SO and insecure code snippets in production software inspires us to study how reliable is SO in providing secure coding suggestions. To what extent can we trust the code snippets on SO? Does the community feedback introduced by the gamification design help in picking code snippets with better security implementation?

To answer these question, we carry out an empirical study on a subset of answer posts, which are related to Java security APIs. These APIs are widely used in many applications including Android apps. Their security vulnerabilities due to API misuse has been pointed out by many previous studies [10, 11, 12, 13, 14, 15, 16]. We extracted this subset of answer posts The subset answer posts are then extracted by running static analysis on the code snippets contained in answer posts. Clone detection is further used to extract answer posts that have duplicated code snippets with other answers. Lastly, base on the rules used in the previous study [8] to identify the secure and insecure usage of these APIs, we manually labeled each of the code snippets we extracted as secure or insecure. Use these label information, clone detection output, and original community feedback data of question and answer posts, we are able to perform both descriptive and inferential statistical analysis on various aspects of the reliability of SO in providing knowledge of security implementation.

Different from the previous work [8], in this thesis, we are more interested in SO itself. We aimed to quantitatively characterize the popularity of secure and insecure answers on SO, and the difference of community feedback around them. The popularity is characterized by the occurrence and duplication of both secure and insecure answers within SO. The community

feedback is characterized by answer score, user reputation, question view count etc. We aim to provide quantitative assessments of how reliable can we trust answers on SO for security implementation as well as the possible source of insecure answers through case studies.

Using JavaBaker [17], we first extracted 25,855 code snippets from 47 million answer posts of SO. These code snippets all contain potential type definitions or function calls to a set of Java security APIs we defined based on previous works. Then using the token-based clone detection tool CCFinder [18], we further identified 3,121 code snippets out of those extracted by JavaBaker. As a result of CCFinder, these code snippets are clustered into 953 clone groups based on their similarity. We then manually inspected each of these code snippets and assigned secure or insecure labels. To have a finer study of different security application scenarios, based on the API usage, we also classified each code snippet into one or multiple of five categories: SSL/TLS, symmetric cryptography, asymmetric cryptography, one-way hash function, secure random number generation. The labels of answer posts are then derived from the principle that a post is insecure as long as it contains one or more insecure code snippets otherwise is secure. Through this labeling process, we identified 785 secure answer posts and 644insecure answer posts and can contrast the community feedback on secure and insecure answers. The labeling also classified the 953 clone groups into 587 secure groups, 326 insecure groups, and 40 mixed groups which contain both secure and insecure code snippets.

We then explored the following research questions (RQs):

- **RQ1:** *How prevalent are insecure coding suggestions on SO?*

  Prior work witnessed the existence of vulnerable code on SO, and indicates that such code can mislead developers and compromise the quality of their software products [5, 8, 9]. To understand the number and growth of secure and insecure options that

developers have to choose from, we (1) compared the occurrence counts of insecure and secure answers, and (2) observed the distributions of both kinds of answers across a 10-year time frame (2008-2017). We did the comparison for all Java security-resulted posts and for posts of specific security categories. As a viable heuristic, if there are significantly more secure answers than insecure answers on SO, developers are less likely to be exposed to vulnerable code snippets, hence SO can be trusted most of the time. Or if secure answers are usually posted later than insecure ones, a developer without much security expertise could rely on answers' timestamps to identify secure answers.

Our study shows that, *as with secure answers, insecure answers are prevalent on SO across the entire studied time frame.* The inspected 3,121 snippets from different clone groups correspond to 785 secure posts and 644 insecure ones. Among the 505 SSL/TLS-related posts, 355 posts (70%) suggest insecure solutions, which makes SSL/TLS-related answers the most unreliable ones on SO. Further, at least 41% of the security-related answers posted every year are insecure, which shows that security knowledge on SO in general is not significantly improving over time. This suggests that developers cannot rely on answers' timestamps to identify secure solutions, as both secure and insecure answers span over the years.

- **RQ2:** *Do the community dynamics or SO's reputation mechanism help developers choose secure answers over insecure ones?*

  Reputation mechanisms and voting were introduced to crowdsourcing platforms to (1) incentivize contributors to provide high-quality solutions, and (2) facilitate question askers to identify responders with high expertise [19, 20, 21]. We conducted statistical testing to compare secure and insecure answers in terms of votes, answerers' reputations, *etc.* Hypothetically, if secure answers received more votes or the providers of

secure suggestions usually have higher reputations, a user could trust such community dynamics to select secure answers over insecure ones.

Our study shows that, *the community dynamics and SO's reputation mechanisms are not reliable indicators for secure and insecure answers.* Compared with secure posts, insecure ones obtained higher *scores*, more *comments*, more *favorites*, and more *views*. Although the providers of secure answers received significantly higher *reputation* points, the effect size is negligible ($\leq$0.147). 239 of the 536 examined *accepted answers* (45%) are insecure. 26 out of the 72 posts (35%) suggested by "trusted users" (with $\geq$20K reputation scores [22]) are insecure. These observations imply that reputation and voting on SO are not reliable to help users distinguish between secure and insecure answers.

- **RQ3:** *Do secure coding suggestions have more duplicates than insecure ones?*

  When certain answers are repetitively suggested, it is likely that developers will encounter such answers more often. Moreover, if these answers are provided by different users, the phenomenon might facilitate users' trust in the answers' correctness. Therefore, we compared the degree of repetitiveness for insecure and secure answers.

  Our study shows that, *the degree of duplication among insecure answers is significantly higher than that of secure ones.* On average, there are more clones in an insecure group than a secure one (3.8 vs. 3.0). It means that users may have to deal with a large supply of insecure examples for certain questions, before obtaining secure solutions.

- **RQ4:** *Why did users suggest duplicated secure or insecure answers on SO?*

  We were curious about why certain code was repetitively suggested, and we explored this facet of community behavior by examining the duplicated answers posted by the same or different users.

Our study shows that, *users seem to post duplicated answers, while ignoring security as a key property.* Duplicated answers were provided due to duplicated questions or users' intent to answer more questions by reusing code examples. This behavior is incentivized by the reputation system on SO. The more answers are posted by a user and up-ranked by the community, the higher the reputation the user gains. So far, we have not identified any user that purposely propagated vulnerable code to mislead people.

The contents of this thesis has been accepted for publication in the 41st ACM/IEEE International Conference on Software Engineering [23, 24].

Chapter 2

Background

To better understanding the methodology we used to study SO community activities around security implementations, in this chapter, we will first introduce SO's crowdsourcing model in Sec. 2.1, then summarize the domain knowledge used to label secure and insecure code snippets in Sec. 2.2.

2.1 Stack Overflow as a Crowdsourcing Platform

Figure 2.1: A typical SO discussion thread contains one question post and one or multiple answer posts [1]

Some observers believe that the success of SO lies in its crowdsourcing model and the reputa-

tion system [3, 25]. The four most common forms of participation are i) question asking, ii) question answering, iii) commenting, and iv) voting/scoring [25]. Figure 2.1 presents an exemplar SO discussion thread, which contains one question and one or many answers. When multiple answers are available, the asker decides which answer to **accept**, and marks it with "✓". This accepted answer will then occupy the top slot of the answer area permanently. The owner of the accepted answer will received 15 points of reputation to the answerers.

After a user posts a question, an answer, or a comment, other users can vote for or against the post. Users gain **reputation** for each up-vote their posts receive. For instance, answers earn their authors 10 points per up-vote, questions earn 5, and comments earn 2 [26]. All users initially have only one reputation point. As users gain more reputation, they are granted more administrative privileges to help maintain SO posts [22]. For instance, a user with 15 points can vote up posts. A user with 125 points can vote down posts. Users with at least 20K points are considered "**trusted users**", and can edit or delete other people's posts.

The **score** of a question or answer post is decided by the up-votes and down-votes the post received. Users can **favorite** a question post if they want to bookmark the question and keep track of any update on the discussion thread. Each question contains one or many **tags**, which are words or phrases to describe topics of the question. Each post has a **timestamp** to show when it was created. Each discussion thread has a **view count** to indicate how many times the thread has been viewed.

2.2 Categorization of Java Security Implementations

Based on state-of-the-art security knowledge, researchers defined five categories of security issues relevant to library misuses [8]. Table 2.1 shows their criteria, which we use in this project to decide whether a code snippet is insecure or not.

Table 2.1: Criteria used to decide code's security property

Category	Parameter	Insecure
SSL/TLS	HostnameVerifier	allow all hosts
	Trust Manager	trust all
	Version	<TLSv1.1
	Cipher Suite	RC4, 3DES, AES-CBC MD5, MD2
	OnReceivedSSLError	proceed
Symmetric	Cipher/Mode	RC2, RC4, DES, 3DES, AES/ECB, Blowfish
	Key	static, bad derivation
	Initialization Vector (IV)	zeroed, static, bad derivation
	Password Based Encryption (PBE)	<1k iterations, <64-bit salt, static salt
Asymmetric	Key	RSA < 2,048 bit, ECC < 224 bit
Hash	PBKDF	<SHA224, MD2, MD5
	Digital Signature	SHA1, MD2, MD5
	Credentials	SHA1, MD2, MD5
Random	Type	Random
	Seeding	setSeed→nextBytes, setSeed with static values

2.2.1 SSL/TLS

There are five key points concerning how to *securely* establish SSL/TLS connections. First, developers should use an implementation of the `HostnameVerifier` interface to verify servers' hostnames instead of allowing all hosts [11]. Second, when implementing a custom `TrustManager`, developers should validate certificates instead of blindly trusting all certificates. Third, when "TLS" is passed as a parameter to `SSLContext.getInstance(...);`, developers should explicitly specify the version number to be at least 1.1, because TLS' lower versions are insecure [27]. Fourth, the usage of insecure cipher suites should be avoided. Fifth, when overriding `onReceivedSslError()`, developers should handle instead of skipping any certificate validation error. Listing 2.1 shows a vulnerable snippet that allows all hosts, trusts all

certificates, and uses TLS v1.0.

Listing 2.1: An example to demonstrate three scenarios of insecurely using SSL/TLS APIs [28]

```
1  // Create a trust manager that does not validate certificate chains (trust all)
2  private TrustManager[] trustAllCerts = new TrustManager[] {
3    new X509TrustManager() {
4      public java.security.cert.X509Certificate[] getAcceptedIssuers() {return null;}
5      public void checkClientTrusted(...) {}
6      public void checkServerTrusted(...) {}     }};
7  public ServiceConnectionSE(String url) throws IOException {
8    try {
9      // Use the default TLSv1.0 protocol
10     SSLContext sc = SSLContext.getInstance("TLS");
11     // Install the trust-all trust manager
12     sc.init(null, trustAllCerts, new java.security.SecureRandom()); ... }  ...
13   connection = (HttpsURLConnection) new URL(url).openConnection();
14   // Use AllowAllHostnameVerifier that allows all hosts
15   ((HttpsURLConnection) connection).setHostnameVerifier(new AllowAllHostnameVerifier()); }
```

2.2.2 Symmetric

There are ciphers and modes of operations known to be insecure. Cryptographic keys and initialization vectors (IV) are insecure if they are statically assigned, zeroed, or directly derived from text. Password Based Encryption (PBE) is insecure if the iteration number is less than 1,000, the salt's size is smaller than 64 bits, or a static salt is in use. Listing 2.2

presents a vulnerable code example that insecurely declares a cipher, a key, and an IV.

Listing 2.2: An example to present several insecure usage scenarios of symmetric cryptography [29]

```
1  // Declare a key parameter with a static value
2  private static byte[] key = "12345678".getBytes();
3  // Declare an IV parameter with a static value
4  private static byte[] iv = "12345678".getBytes();
5  public static String encrypt(String in) {
6    String cypert = in;
7    try {
8      IvParameterSpec ivSpec = new IvParameterSpec(iv);
9      // Create a secret key with the DES cipher
10     SecretKeySpec k = new SecretKeySpec(key, "DES");
11     // Declare a DES cipher
12     Cipher c = Cipher.getInstance("DES/CBC/PKCS7Padding");
13     c.init(Cipher.ENCRYPT_MODE, k, ivSpec);
14      ... } }
```

2.2.3 Asymmetric

Suppose that a code snippet uses either RSA or ECC APIs to generate keys. When the specified key lengths for RSA and ECC are separately shorter than 2,048 bits and 224 bits, we consider the API usage to be insecure. Listing 2.3 shows a vulnerable code example.

Listing 2.3: An example that insecurely uses RSA by specifying the key size to be 1024 [30]

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg. initialize (1024);
KeyPair kp = kpg.generateKeyPair();
RSAPublicKey pub = (RSAPublicKey) kp.getPublic();
RSAPrivateKey priv = (RSAPrivateKey) kp.getPrivate();
```

2.2.4 Hash

In the context of password-based key derivation, digital signatures, and authentication/authorization, developers may explicitly invoke broken hash functions. Listing 2.4 shows an example using MD5.

Listing 2.4: Insecurely creating a message digest with MD5 [31]

```
final MessageDigest md = MessageDigest.getInstance("md5");
// It is also insecure to hardcode the plaintext password
final byte[] digestOfPassword = md.digest("HG58YZ3CR9".getBytes("utf-8"));
```

2.2.5 Random

To make the generated random numbers unpredictable and secure, developers should use `SecureRandom` instead of `Random`. When using `SecureRandom`, developers can either (1) call `nextBytes()` only, or (2) call `nextBytes()` first and `setSeed()` next. Developers should not call `setSeed()` with static values. Listing 2.5 presents an example using `SecureRandom` insecurely.

Listing 2.5: Using `SecureRandom` with a static seed [32]

```
1 byte[] keyStart = "encryption key".getBytes();
2 SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
3 sr.setSeed(keyStart);
```

Chapter 3

Methodology

To collect secure and insecure answer posts, we first extracted code snippets from SO that used any security API (Section 3.1). Next, we sampled the extracted code corpus by detecting duplicated code (Section 3.2). Finally, we manually labeled sampled code as secure, insecure, or irrelevant, and mapped the code to related posts (Section 3.3). Additionally, we compared the view counts of the sampled posts vs. unselected posts to check samples' prevalence (Section 3.4). An overview of the work flow is shown in Fig.3.1.

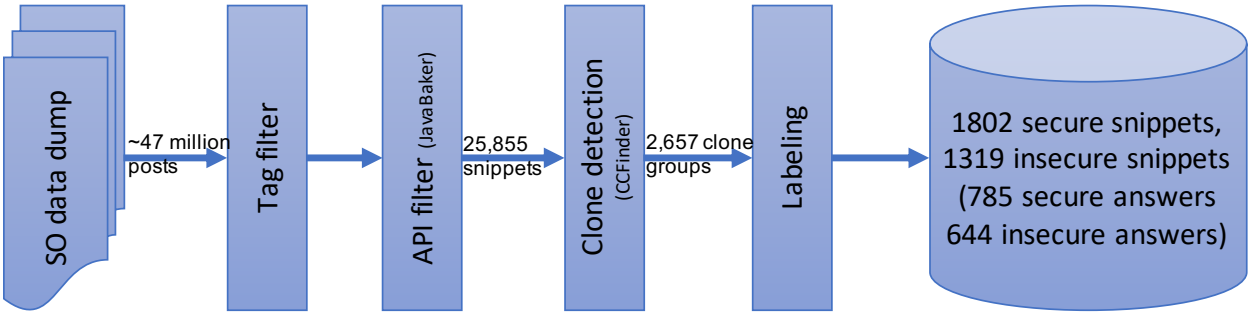

Figure 3.1: The work flow of extracting and labeling security-related code snippets.

3.1 Code Extraction

To identify coding suggestions, this step extracts security-related answer posts by analyzing (1) tags of question posts, and (2) the code snippets' API usage of answer posts. After downloading the Stack Overflow data as XML files [33], we used a tool stackexchange-dump-to-postgres [34] to convert the XML files to Postgres database tables. Each row in

the database table "Posts" corresponds to one post. A post's body text may use the HTML tag pair `<code>` and `</code>` to enclose source code, so we leveraged this tag pair to extract code. Since there were over 40 million posts under processing, and one post could contain multiple code snippets, *it is very challenging to efficiently identify security implementations from a huge amount of noisy code data.* Thus, we built two heuristic filters to quickly skip irrelevant posts and snippets.

Table 3.1: Tags used to locate relevant SO discussion threads

Category	Tags
Java platforms	android, applet, eclipse, java, java1.4, java-7, java-ee, javamail, jax-ws, jdbc, jndi, jni, ...
Third-party tools/libraries	axis2, bouncycastle, gssapi, httpclient, java-metro-framework, openssh, openssl, spring-security, ...
Security	aes, authentication, certificate, cryptography, des, encoding, jce, jks, jsse, key, random, rsa, security, sha, sha512, single-sign-on, ssl, tls, X509certificate, ...

3.1.1 Filtering by question tags

As tags are defined by askers to describe the topics of questions, we relied on tags to skip obviously irrelevant posts. To identify as many security coding suggestions as possible, we inspected the 64 cryptography-related posts mentioned in prior work [9], and identified 93 tags. If a question post contains any of these tags, we extracted code snippets from the corresponding answer posts. As shown in Table 3.1, these tags are either related to Java platforms, third-party security libraries or tools, or security concepts.

3.1.2 Filtering by security API usage

Similar to prior work [8], we used JavaBaker [17] to automatically decide whether a code snippet invokes any security API. This thesis focuses on the following APIs:

- Java-based platform security: android.security.*, com.sun.security.*, java.security.*, javax.crypto.*, javax.net.ssl.*, javax.security.*, javax.xml.crypto.*, org.jetf.jgss.*;

- Third-party security libraries: BouncyCastle [35], GNU Crypto [36], jasypt [37], keyczar [38], scribejava [39], SpongyCastle [40].

To decide whether a code snippet invokes any security API, a naive approach can compare the methods invoked by the snippet with all known security APIs; if any method's name matches, the snippet is security-related. However, such naive approaches do not work well. They can cause many false positives and wrongly identify irrelevant code, because such approaches cannot differentiate between the methods with identical names but declared by different classes. Thus, we chose to use JavaBaker to better locate security-related code.

After taking in a set of libraries and a code snippet, JavaBaker (1) extracts all APIs of types, methods, and fields from the libraries, (2) extracts names of types, methods, and fields, used in the code snippet, and (3) iteratively deduces identifier mappings between the extracted information. Intuitively, when multiple type APIs (*e.g.*, `a.b.C` and `d.e.C`) can match a used type name `C`, JavaBaker compares the invoked methods on `C` against the method APIs declared by each candidate type, and chooses the candidate that contains more matching methods.

3.2 Clone Detection

With the filters described above, we extracted 25,855 code snippets that are likely to implement security. Since it is almost impossible to manually check all these snippets to identify secure and insecure code, we decided to (1) sample representative extracted code via clone detection, and then (2) manually label the samples. In addition to sampling, clone detection facilitates our research in two further ways. First, by identifying duplicated code with CCFinder [18], we could explore the degree of duplication among secure and insecure code. Second, through clustering code based on their similarity, we could efficiently read similar code fragments, and determine their security property in a consistent way. With the default parameter setting in CCFinder, we identified 2,657 clone groups that contained 8,690 code snippets, with each group having at least two snippets.

3.3 Code Labeling

We first studied prior work's rule definition [8] and dataset [41] to learn the criteria of code labeling. Then, we manually examined each snippet in the 2,657 clone groups. If a snippet meets any criteria of insecure code shown in Table 2.1, it is labeled as "insecure". If the snippet uses any security API but does not meet any criteria, it is labeled as "secure"; otherwise, it is "irrelevant". Depending on the APIs involved, we also decided to which security category a relevant post belongs. When unsure about certain posts, we had discussions to achieve consensus. Finally, we randomly explored a subset of the labeled data to double-check the correctness.

Table 3.2 presents our labeling results for the inspected 2,657 clone groups. After checking individual code snippets, we identified 587 secure groups, 326 insecure groups, 40 mixed

Table 3.2: Code labeling results for 2,657 clone groups

	Secure	Insecure	Mixed	Irrelevant	Total
# of clone groups	587	326	40	1,704	2,657
# of snippets	1,802	1,319	0	5,569	8,690
# of answer posts	785	644	0	2,133	3,562

groups, and 1,704 irrelevant groups. In particular, a mixed group has both secure snippets and insecure ones, which are similar to each other. Although two filters were used (see Section 3.1), 64% of the clone groups from refined data were still irrelevant to security, which evidences the difficulty of automatically identifying security implementation.

The clone groups cover 1,802 secure snippets, 1,319 insecure ones, and 5,569 irrelevant ones. When mapping these snippets to the answer posts (which contain them), we identified 785 secure answers, 644 insecure ones, and 2,133 irrelevant ones. One answer can have multiple snippets of different clone groups. Therefore, we label the post as "insecure" if any of the contained snippets is insecure; otherwise, if at least one snippet is secure, the post is labeled as "secure". If a post does not contain any (in)secure snippet, it is labeled as "irrelevant".

To obtain the the community dynamics towards these answer posts, we associate the meta data of the Q&A threads these answers belong to. Important meta data about a question thread used in this study include: the score of the answer, the comment count of the answer, the reputation of the user that posted the answer, the favorite count of the question, the view count of the Q&A thread. We will compare the distribution of secure answer posts and the distribution of insecure answer posts over these variables in Sec. 4.2, to examine the effectiveness of SO's gamification design in differentiating secure and insecure answers.

3.4 Verifying the Prevalence of Sampled Posts

To check whether our clone-based approach actually included representative SO posts, we separately computed the cumulative distribution functions (CDF) [42] of view count for the included 3,562 posts (as mentioned in Table 3.2), the excluded 19,662 posts, and the complete set of 23,224 posts identified by JavaBaker. As shown in Fig. 3.2, the "*included*" curve is beneath the "*all*" and "*excluded*" curves. This shows that the highly viewed answers take up a higher percentage in our sample set than the excluded answers.

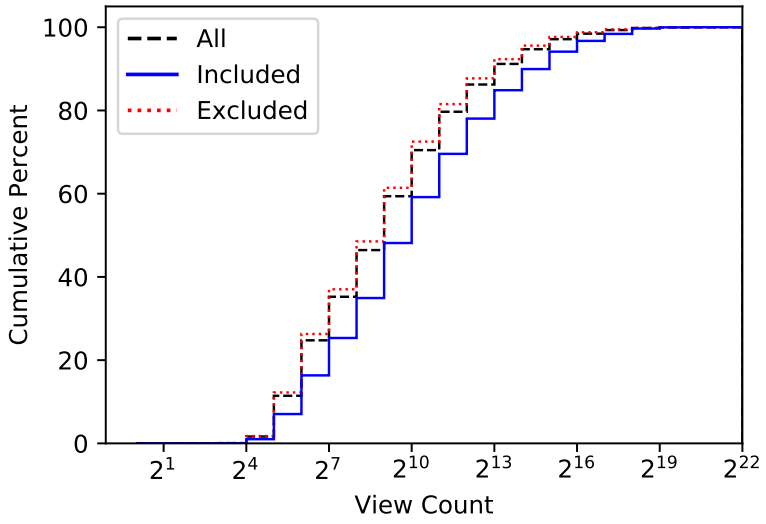

Figure 3.2: CDFs of view count among the included answers, excluded ones, and all answers related to JavaBaker's output

Chapter 4

Results

In this chapter, we present our results as main findings regarding our stated research questions in chapter 1.

4.1 Popularity of Secure and Insecure Code Suggestions

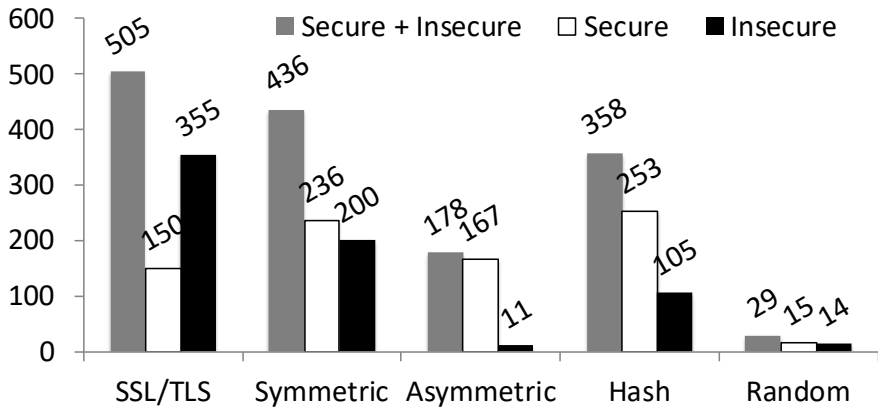

Figure 4.1: The distribution of posts among different categories

Figure 4.1 presents the distribution of 1,429 answer posts among the 5 security categories. Since some posts contain multiple snippets of different categories, the total number of plotted secure and insecure posts in Figure 4.1 is 1,506, slightly larger than 1,429. Among the

categories, *SSL/TLS* contains the most posts (34%), while *Random* has the fewest posts (2%). Two reasons can explain such a distribution. First, developers frequently use or are more concerned about APIs of *SSL/TLS*, *Symmetric*, and *Hash*. Second, the criteria we used to label code contain more diverse rules for the above-mentioned three categories, so we could identify more instances of such code.

There are many more insecure snippets than secure ones in the *SSL/TLS* (355 vs. 150) category, and slightly more insecure snippets for *Random* (14 vs. 15). It highlights that developers should be quite cautious when searching for code within these two categories. Meanwhile, secure answers dominate the other three categories, accounting for 93% of *Asymmetric* posts, 70% of *Hash* posts, and *54%* of Symmetric posts. However, notice that across these 3 categories, only 67% of the posts are secure; that is, considerable room for error remains.

Finding 1: *644 out of the 1,429 inspected answer posts (45%) are insecure, meaning that insecure suggestions popularly exist on SO. Insecure answers dominate, in particular, the SSL/TLS category.*

To explore the distribution of secure and insecure answers over time, we clustered answers based on their timestamps. As shown in Figure 4.2, both types of answers increased year-by-year from 2008 to 2014, and decreased in 2015-2017. This may be because SO reached its saturation for Java security-related discussions in 2014. In 2008, 2009, and 2011, insecure answers were posted more often than secure ones, taking up 54%-100% of the sampled data of each year. For the other years, secure posts constitute the majority within the yearly sampled data set, accounting for 53%-59%.

To further determine whether older posts are more likely to be insecure, we considered post IDs as logical timestamps. We applied a Mann-Whitney U test (which does not require

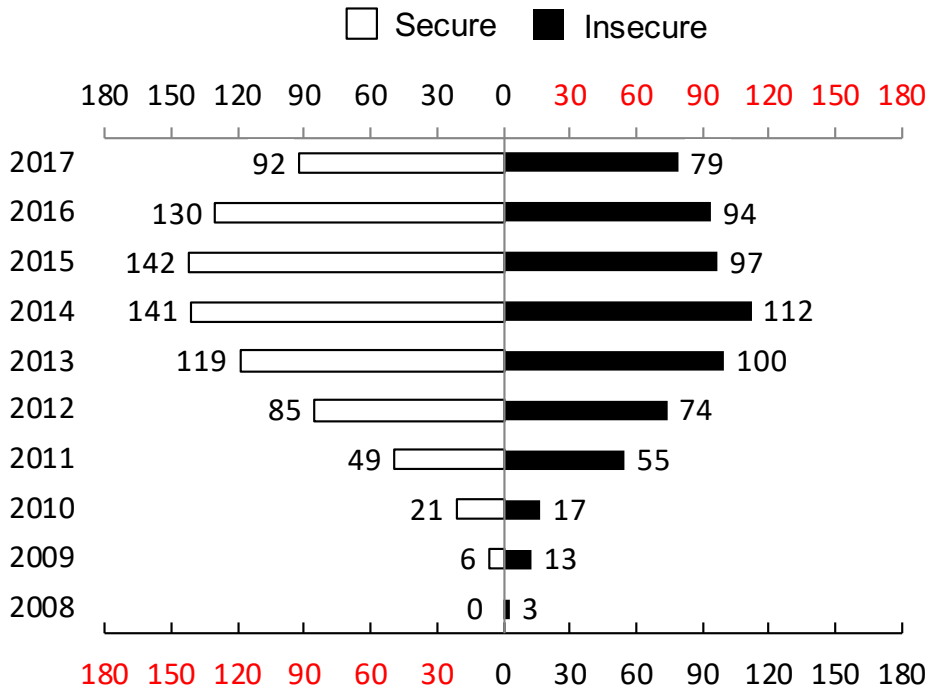

Figure 4.2: The distribution of posts over during 2008-2017

normally distributed data [43]), and calculated the Cliff's delta size (which measures the difference's magnitude [44]). The resulting p-value is 0.02, with Cliff's Δ=0.07. It means that secure answers are significantly more recent than insecure ones, but the effect size is negligible.

Two reasons can explain this finding. First, some vulnerabilities were recently revealed. Among the 17 insecure posts in 2008 and 2009, 6 answers use MD5, 6 answers trust all certificates, and 4 answers use TLS 1.0. However, these security functions were found broken in 2011-2012 [11, 12, 45, 46], which made the answers obsolete and insecure. Second, some secure answers were posted to correct insecure suggestions. For instance, we found a question inquiring about fast and simple string encryption/decryption in Java [47]. The accepted answer in 2011 suggested DES—an insecure symmetric-key algorithm. Later, various comments pinpointed the vulnerability, and a secure answer was provided in 2013.

Note that there can be a significant lag until the community adopts new, secure technologies, and phases out technologies known to be insecure. Although MD5's vulnerability was exploited by Flame malware in 2012 [45], as shown in Fig. 4.3, MD5 was still popularly suggested afterwards, obtaining a peak number of related answers in 2014.

Finding 2: *Insecure posts led the sampled data in 2008-2011, while secure ones were dominant afterwards. Older security-related posts are less reliable, likely because recently revealed vulnerabilities outdated older suggestions. We found only few secure answers suggested to correct outdated, insecure ones.*

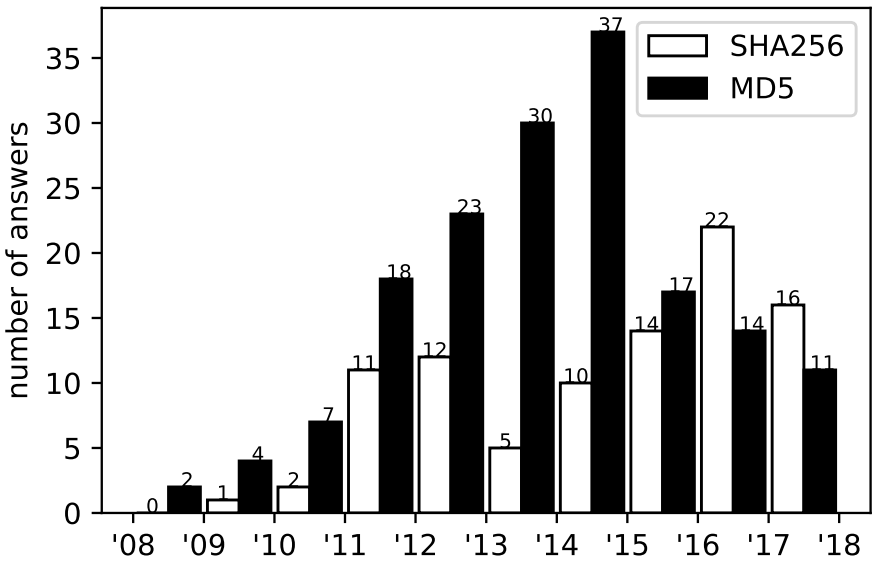

Figure 4.3: Distributions of MD5 and SHA256 related posts

4.2 Community Dynamics Towards Secure and Insecure Code

For each labeled secure or insecure post, we extracted the following related information from the Postgres database: (1) score, (2) comment count, (3) the answerer's reputation score, (4) the question's favorite count, and (5) the discussion thread's view count.

Effect of community feedback. We first want to understand how much do developers utilized the community feedback. In Table 4.1, we computed the Spearman's correlation coefficients ρ between the five attributes of answer posts and additional information of Q&A threads such as answer count. We found that answers' view count, which reflects the number of times developers refer to the answers, has a positive correlation ($\rho = 0.54$) with answers' score. Questions' favorite count, which reflects the number of users keeping the threads for recurring reference, also has a positive correlation ($\rho = 0.81$) with questions' score. This suggests the community feedback information does affect developers in finding code examples on SO for their reference. Therefore it is worth to study if they reflected the difference between secure and insecure answers.

Comparison of mean values.

Table 4.2 compares these information categories for the 785 secure posts and 644 insecure ones and applies Mann-Whitney U tests to determine significant results. On average, secure posts' answerers have higher reputation (18,654 vs. 14,678). However, for the SSL/TLS posts, the insecure answer providers have higher reputation (15,695 vs. 14,447). Moreover, insecure posts have higher scores, and more comments, favorites, and views. Users seemed to be more attracted by insecure posts, *which is counterintuitive*. We would expect secure answers to be seen more favorable; with more votes, comments and views.

Table 4.1: Spearman's correlation coefficients between meta data of Q&A threads

	U.R	**A.CC**	**Q.FC**	**Q.VC**	**A.S**	**Q.S**	**Q.AC**	**Q.CC**
U.R	1.00	0.31	0.05	0.07	0.29	0.06	-0.04	0.03
A.CC	0.31	1.00	0.19	0.17	0.35	0.17	0.04	0.09
Q.FC	0.05	0.19	1.00	0.79	0.56	0.81	0.59	0.04
Q.VC	0.07	0.17	0.79	1.00	0.54	0.77	0.64	0.03
A.S	0.29	0.35	0.56	0.54	1.00	0.54	0.27	0.01
Q.S	0.06	0.17	0.81	0.77	0.54	1.00	0.59	0.03
Q.AC	-0.04	0.04	0.59	0.64	0.27	0.59	1.00	0.04
Q.CC	0.03	0.09	0.04	0.03	0.01	0.03	0.04	1.00

Meaning of row and column names. U.R: users' reputation; A.CC: answers' comment count; Q.FC: questions' favorite count; Q.VC: questions' view count; A.S: answers' score; Q.S: questions' score; Q.AC: questions' answer count; Q.CC: questions' comment count.

Two reasons can explain our observation. First, software developers often face time constraints. When stuck with coding issues (*e.g.*, runtime errors or exceptions), developers are tempted to take simple solutions, even though they may be insecure. Meng *et al.* [9] observed such justifications by SO users for their insecure SSL/TLS usage. Take the vulnerable SSL/TLS usage in Listing 2.1 for example. The insecure code has been frequently suggested on SO and many users voted for such suggestions probably because the code is simple and useful to resolve connection exceptions. Nevertheless, the simple solution essentially skips SSL verification and voids the protection mechanism. In comparison, a better solution should use certificates from a Certification Authority (CA) or self-signed certificates to drive the customization of `TrustManager`, and verify certificates with more complicated logic [48]. Second, some insecure algorithms are widely supported by Java-based libraries, which can promote developers' tendency to code insecurely. For instance, up till the current version Java 9, Java platform implementations have been required to support MD5—the well-known broken hash function [49].

Finding 3: *On average, insecure posts received higher scores, more comments, more favorites, and more views. It implies that (1) more user attention is attracted by insecure answers; and (2) users cannot rely on the voting system to identify secure answers.*

Comparison of p-values and Cliff's Δ. Table 4.2 shows that among all posts, insecure ones obtained significantly more comments ($p = 0.02$) and views ($p = 1.5e - 3$), while the effect sizes are negligible. Meanwhile, the writers of secure answer posts have significantly higher reputation ($p = 0.02$), but the magnitude is also negligible. (Note that some information including reputation is missing for a few answer writers for reasons that could not be determined.)

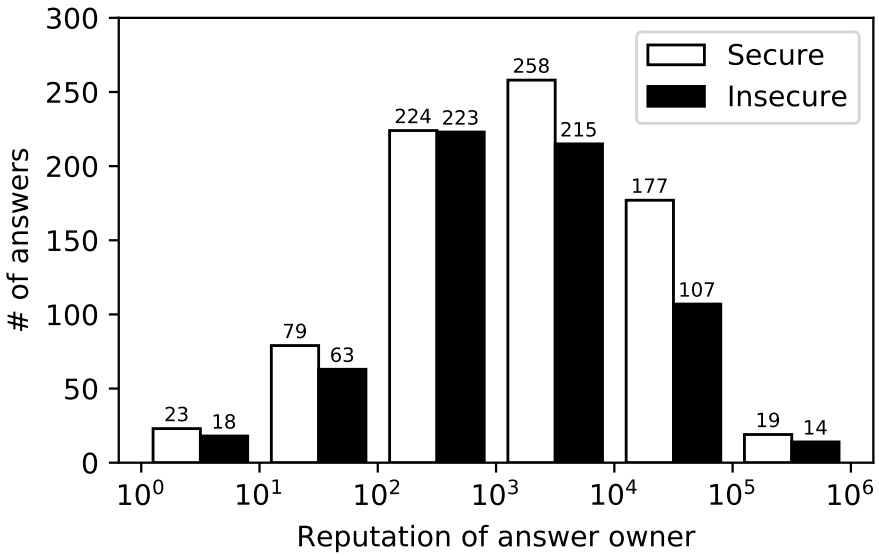

Figure 4.4: The distribution of answers based on their owners' reputation

Figure 4.4 further clusters answers based on their providers' reputation scores. We used logarithmic scales for the horizontal axis, because the scores vary a lot within the range [1, 990,402]. Overall, the secure and insecure answers have similar distributions among

Table 4.2: Comparison between secure and insecure posts

		Score	Comment count	Reputation	Favorite count	View count
All	**Secure mean**	5	2	18,654	8	18,713
	Insecure mean	14	3	14,678	15	36,580
	p-value	0.97	0.02	0.02	0.09	1.5e-3
	Cliff's Δ	-	0.07 (negligible)	0.07 (negligible)	-	0.10 (negligible)
Category 1: SSL/TLS	**Secure mean**	7	2	14,447	9	21,419
	Insecure mean	18	3	15,695	19	37,445
	p-value	0.24	3.3e-4	0.42	0.86	0.31
	Cliff's Δ	-	0.20 (small)	-	-	-
Category 2: Symmetric	**Secure mean**	5	3	19,347	7	16,232
	Insecure mean	7	3	10,057	6	16,842
	p-value	0.29	0.82	0.45	0.36	0.10
	Cliff's Δ	-	-	-	-	-
Category 3: Asymmetric	**Secure mean**	5	2	17,079	4	11,987
	Insecure mean	8	2	14,151	3	9,470
	p-value	0.17	0.45	0.72	0.95	0.77
	Cliff's Δ	-	-	-	-	-
Category 4: Hash	**Secure mean**	5	2	20,382	8	21,254
	Insecure mean	14	2	20,018	22	74,482
	p-value	0.26	0.78	0.18	0.20	0.07
	Cliff's Δ	-	-	-	-	-
Category 5: Random	**Secure mean**	1	3	33,517	0	1,031
	Insecure mean	21	6	17,202	31	56,700
	p-value	0.04	0.02	0.27	0.02	0.01
	Cliff's Δ	0.58 (large)	0.68 (large)	-	0.64 (large)	0.74 (large)

Similar to prior work [50], we interpreted the computed Cliff's delta value v in the following way: (1) if $v < 0.147$, the effect size is "negligible"; (2) if $0.147 \leq v < 0.33$, the effect size is "small'; (3) if $0.33 \leq v < 0.474$, the effect size is "medium"; (4) otherwise, the effect size is "large".

different reputation groups. For instance, most answers were provided by users with scores within $[10^2, 10^4)$, accounting for 62% of secure posts and 71% of insecure posts. Among the 208 posts by trusted users, 71 answers (34%) are insecure and not reliable. One reason to explain why high reputation scores do not guarantee secure answers can be that users earned scores for being an expert in areas other than security. Responders' reputation scores do not necessarily indicate the security property of the provided answers. Therefore, SO users should not blindly trust the suggestions given by highly reputable contributors.

Finding 4: *The users who provided secure answers have significantly higher reputation than the providers of insecure answers, but the difference in magnitude is negligible. Users cannot rely on the reputation mechanism to identify secure answers.*

Comparison of accepted answers. It is natural for SO users to trust accepted answers. Among the 1,429 posts, we found 536 accepted answers (38%). 297 accepted answers are secure, accounting for 38% of the inspected secure posts. 239 accepted answers are insecure, accounting for 37% of the inspected insecure posts. It seems that accepted answers evenly distribute among secure and insecure posts; they are not good indicators of suggestions' security property.

Finding 5: *Accepted answers are also not reliable for users to identify secure coding suggestions.*

4.3 Duplication of Secure and Insecure Code

Among the 953 security-related clone groups, we explored the degree of code duplication for secure clone groups, insecure groups, and mixed groups. Figure 4.5 shows the distribution of clone groups based on their sizes. Similar to Fig. 4.4, we used logarithmic scales for both

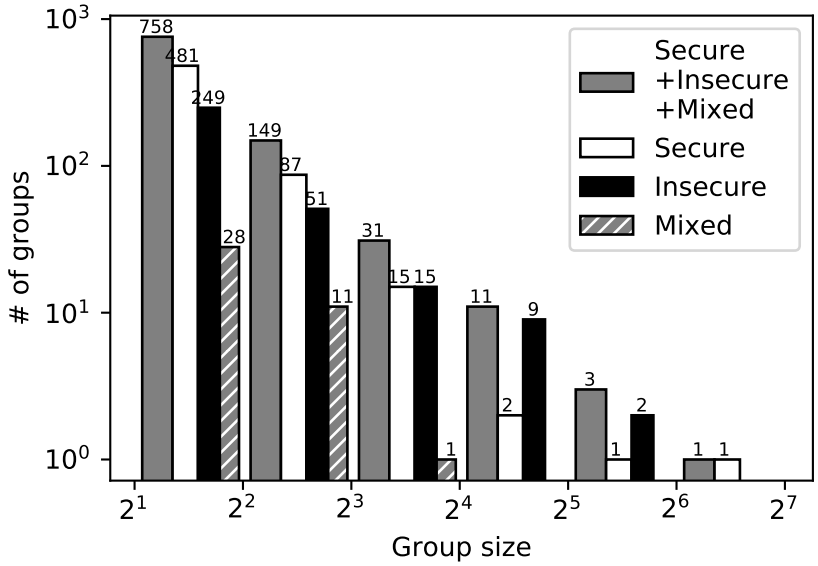

Figure 4.5: The distribution of clone groups based on their sizes

the horizontal and vertical axes. According to Fig. 4.5, most clone groups are small, with two or three similar snippets. The number of groups decreases dramatically as the group size increases. Interestingly, within $[2^3, 2^6)$, there are more insecure groups than secure ones. We used the Mann-Whitney U test to compare the sizes of secure and insecure groups (see Table 4.3). Surprisingly, insecure groups have significantly larger sizes than secure groups, although the difference is negligible. Our observations imply that *the frequently mentioned code snippets on SO are not necessarily more secure than less frequent ones.* Users cannot trust suggestions' repetitiveness to decide the security property.

Table 4.3: Comparison between secure and insecure groups in terms of their group sizes

	Secure groups' mean	**Insecure groups' mean**	**p-value**	**Cliff's Δ**
Size	3.0	3.8	1.3e-4	0.13(negligible)

Finding 6: *Repetitiveness does not guarantee security, so users cannot assume a snippet to be secure simply because it is recommended many times.*

To understand why there are mixed groups that contain similar secure and insecure implementations, we conducted a case study on 10 randomly selected mixed groups. Among *all* these groups, secure snippets differ from insecure ones by using distinct parameters when calling security APIs. This implies a great opportunity to build security bug detection tools that check for the parameter values of specific APIs.

Listing 4.1: A clone group with both secure and insecure code

```
1  // An insecure snippet using AES/ECB to create a cipher [51]
2  Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding", "SunJCE");
3  Key skeySpec=KeyGenerator.getInstance("AES").generateKey();
4  cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
5  System.out.println(Arrays.toString(cipher.doFinal(new byte[] { 0, 1, 2, 3 })));
6  //A secure snippet using AES/CFB to create a cipher [52]
7  final Cipher cipher=Cipher.getInstance("AES/CFB/NoPadding", "SunJCE");
8  final SecretKey skeySpec=KeyGenerator.getInstance("AES").generateKey();
9  cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
10 System.out.println(Arrays.toString(cipher.doFinal(new byte[] { 0, 1, 2, 3 })));
```

Listing 4.1 shows a mixed clone group, where the insecure code uses "`AES/ECB`" to create a cipher, and the secure code uses "`AES/CFB`". Actually, both snippets were provided by the same user, which explains why they are so similar. These answers are different because the askers inquired for different modes (ECB vs. CFB). Although the answerer is an expert in using both APIs and has a high reputation score 27.7K, he/she did not mention anything

about the vulnerability of ECB. This may imply a lack of security expertise or vulnerability awareness of highly reputable SO users, and a well-motivated need for automatic tools to detect and fix insecure code.

Finding 7: *Secure and insecure code in the same mixed group often differs by passing distinct parameters to the same security APIs highlighting opportunities for automatic tools to handle security weaknesses.*

4.4 Creation of Duplicated Secure and Insecure Code

We conducted two case studies to explore why duplicated code was suggested.

Case Study I: Duplicated answers by different users. We examined the largest secure group and largest insecure group. The secure group has 65 clone instances, which are similar to the code in Listing 4.2. These snippets were offered to answer questions on how to enable an Android app to log into Facebook. The questions are similar but different in terms of the askers' software environments (*e.g.*, libraries and tools used) and potential solutions they tried. Among the 65 answers, only 18 (28%) were marked as accepted answers. The majority of duplicated suggestions are relevant to the questions, but cannot solve the issues. SO users seemed to repetitively provide "generally best practices", probably because they wanted to earn points by answering more questions.

Listing 4.2: An exemplar snippet to generate a key hash for Facebook login [53]

```
1 PackageInfo info = getPackageManager().getPackageInfo("com.facebook.samples.hellofacebook",
      PackageManager.GET_SIGNATURES);
2 for (Signature signature : info.signatures) {
```

```
3  MessageDigest md = MessageDigest.getInstance("SHA");
4  md.update(signature.toByteArray());
5  Log.d("KeyHash:", Base64.encodeToString(md.digest(), Base64.DEFAULT)); }
```

The largest insecure group contains 32 clone instances, which are similar to the code in Listing 2.1. The questions are all about how to implement SSL/TLS or resolve SSL connection exceptions. 13 of these answers (41%) were accepted. We noticed that only one answer warns "*Do not implement this in production code …*" [54]. Six answers have at least one comment talking about the vulnerability. The remaining 25 answers include nothing to indicate the security issue.

Case Study II: Duplicated answers by the same users. In total, 109 users reused code snippets to answer multiple questions. Among the 207 clone groups these users produced, there are 111 secure groups, 90 insecure groups, and 6 mixed groups. 66 users repetitively posted secure answers, and 49 users posted duplicated insecure answers. Six among these users posted both secure and insecure answers. Most users (*i.e.*, 92) only copied code once and produced two duplicates. One user posted nine insecure snippets, with seven snippets using an insecure version of TLS, and two snippets trusting all certificates. This user has 17.7K reputation (top 2% overall) and is an expert in Android. By examining the user's profile, we did not find any evidence to show that the user intentionally misled people. It seems that the user was not aware of the vulnerability when posting these snippets.

To understand whether duplicated code helps answer questions, we randomly sampled 103 (of the 208) clone groups resulting in 56 secure clone pairs, 45 insecure pairs, and 2 mixed pairs. Unexpectedly, we found that 46 pairs (45%) did not directly answer the questions. A user posted an answer to advertise the library he wrote [55] or posted code without reading the question [56]. In the other 57 cases, duplicated code was provided to answer similar or

identical questions.

Finding 8: *Duplicated answers were created because (1) users asked similar or related questions; and (2) some users blindly copied and pasted code to answer more questions and earn points. However, we did not identify any user that intentionally misled people by posting insecure answers.*

Chapter 5

Discussion

5.1 Recommendations

By analyzing SO answer posts relevant to Java-based security library usage, we observed the wide-spread existence of insecure code. It is worrisome to learn that SO users cannot rely on either the reputation mechanism or voting system to infer an answer's security property, A recent Meta Exchange discussion thread also shows the frustration of SO developers to keep outdated security answers up to date [57]. Below are our recommendations based on this analysis.

For Tool Builders Explore approaches that accurately and flexibly detect and fix security bugs. Although a few existing tools automatically identify security API misuses through static program analysis or machine learning [8, 13, 58, 59], they are still unsatisfactory due to the (1) hard-to-extend API misuse patterns hardcoded in tools, and (2) hard-to-explain machine learning results. We suggest developing new approaches that (i) compare identified insecure implementations with secure counterparts, (ii) generalize bug patterns and fixing strategies from the comparison, and (iii) search for buggy code matching any pattern to suggest security fixes.

For SO Developers Integrate static checkers to scan existing corpus and SO posts under submission. Automatically add warning messages or special tags to any post that has vulnerable code. Encourage moderators or trusted users to exploit clone detection technologies in order to efficiently detect and remove both duplicated questions and answers. Such deduplication practices will not only save users' time and effort of reading/answering useless duplicates, but also mitigate the misleading consensus among multiple similar insecure suggestions. Instead of presenting a responder's overall reputation score to indicate an answer's credibility, show the responder's score in related areas or for the specific tags.

For Designers of Crowdsourcing Platforms Provide incentives to users for detailing vulnerabilities and suggesting secure alternatives. Introduce certain mechanisms to encourage owners of outdated or insecure answers to proactively archive or close such posts. Currently on SO, if a user downvotes a post, the user loses two reputation points [22]. In the future, platforms can instead encourage security experts to downvote insecure answers by rewarding them with reputation points.

5.2 Threats to Validity

5.2.1 Threats to external validity

This study labels insecure code snippets based on the Java security rules summarized by prior work [8], so our studied insecure snippets are limited to Java code and these rules. Since we leveraged the state-of-the-art insecurity criteria, our manual analysis revealed as diverse insecure code as possible. In the future, we plan to reduce this threat by considering different programming languages, more security rules, and more scalable labeling techniques.

5.2.2 Threats to construct validity

Although we tried our best to accurately label code, our analysis may be still subject to human bias and cannot scale to handle all crawled data or more security categories. We conservatively assumes that a snippet that does not match any given rule is secure. However, it is possible that some labeled secure snippets actually match the insecurity criteria not covered by this study, or will turn out to be insecure when future attack technologies are created. We concluded that insecure answers are popular on SO and gain high scores, votes, and views. Even if the labels of some existing secure answers will be corrected as insecure in the future, our conclusion generally holds.

5.2.3 Threats to internal validity

We leveraged clone detection to sample the extracted code snippets and reduce our manual analysis workload. Based on code's occurrence repetition, clone detection can ensure the representativeness of sampled data. However, the measurement on a sample data set may be still different from that of the whole data set. Once we build automatic approaches to precisely identify security API misuses, we can resolve this threat.

Chapter 6

Related Works

As our work focus on the security issue of Stack Overflow posts. In this chapter, we would like to review previous works on awareness of this issue and the potential solutions. Before reviewing the related works, we want to emphasize the unique aspects of the problem we study in this work. Information security has always been a challenging problem in the human history, but it only becomes more relevant to more people and more aspects of our life until the emergence of information technology in the last few decades, especially the invention of smartphones. Together with this trend is the growth in the number of software developers, more and more developers are forced to deal with sensitive information such as financial information, system access information. Unfortunately, not all developers have received training in information security, or have professional support teams such as those in large companies. The success of the Stack Overflow as the technical question and answering platform has attracted a lot of developers to rely on it to solve various issues they encounter when developing software. The search engine friendly design also attracts a lot of technical question search to the Stack Overflow. Therefore, in the current security issues found in many production software, Stack Overflow must play a role. Several previous works have observed the sign of such correlation. But none of them has systematically studied the distribution of these security-related code snippets on Stack Overflow, and the attitude of the Stack Overflow community toward them. We believe the understanding of this problem depends on two aspects: understanding of security itself and understanding of Stack Overflow. The

understanding of the security can help us to identify the current challenges developers have faced and the reason for these challenges. The understanding of Stack Overflow can help us find out if Stack Overflow has helped in solving these problems or worsen in spreading these problems, and the potential solution we can take to developers to write more secure production software. Our review hence is organized in the following major aspects.

6.1 Security API Misuses

Prior studies showed that API misuses caused security vulnerabilities [8, 10, 11, 12, 13, 14, 15, 16]. For instance, Lazar et al. analyzed 369 published cryptographic vulnerabilities in the CVE database, and found that 83% of them were caused by API misuses [14]. Egele *et al.* built a static checker for six well-defined Android cryptographic API usage rules (e.g., "Do not use ECB mode for encryption"). They analyzed 11,748 Android applications for any rule violation [13], and found 88% of the applications violating at least one checked rule. Instead of checking for insecure code in CVE or software products, we focused on SO. Because the insecure coding suggestions on SO can be read and reused by many developers, they have a profound impact on software quality.

The research by Fischer *et al.* [8] is closely related to our work. In their work, Android-related code snippets are extracted from SO. The authors manually labeled a subset of the data as "secure" or "insecure", and used the data to train a classifier that automatically labels the remaining extracted data. With the labeled data, the authors searched for insecure code in Android apps via clone detection. Our research is different in two aspects. First, our research scope is different. We aimed to analyze the popularity, surrounding social dynamics, and duplication of insecure code on SO. Second, our (in)security labeling is more reliable. As it is always challenging to determine whether a given snippet is secure or not, we chose to

conduct manual analysis instead of using machine learning to accurately identify insecure code.

6.2 Developer Studies

Researchers conducted interviews or surveys to understand developers' security coding practices [5, 60, 61, 62]. For example, Nadi *et al.* surveyed 48 developers and revealed that developers found it difficult to use cryptographic algorithms correctly [62]. Xie *et al.* interviewed 15 developers, and found that (1) most developers had reasonable knowledge about software security, but (2) they did not consider security assurance as their own responsibility [60]. Acar *et al.* surveyed 295 developers and conducted a lab user study with 54 students and professional Android developers [5]. They observed that most developers used search engines and SO to address security issues. These studies inspired us to explore how much we can trust the crowdsourced knowledge of security coding on SO.

6.3 Empirical Studies on Stack Overflow

Researchers conducted various studies on SO [4, 9, 16, 63, 64, 65, 66]. Specifically, Zhang *et al.* studied the JDK API usage recommended by SO, and observed that 31% of the studied posts misused APIs [9]. Meng *et al.* manually inspected 503 SO discussion threads related to Java security [9]. They revealed various secure coding challenges (*e.g.*, hard-to-configure third-party frameworks) and vulnerable coding suggestions (*e.g.*, SSL/TLS misuses). Mamykina *et al.* revealed several reasons (*e.g.*, high response rate) to explain why SO is one of the most visible venues for expert knowledge sharing [4]. Vasilescu *et al.* studied the associations between SO and GitHub, and found that GitHub committers usually ask

fewer questions and provide more answers [67]. Bosu *et al.* analyzed the dynamics of reputation building on SO, and found that answering as many questions as possible can help users quickly earn reputation [63].

In comparison, this thesis quantitatively and qualitatively analyzed secure and insecure SO suggestions in terms of (1) their popularity, (2) answerers' reputations, (3) the community's feedback to answers (*e.g.*, votes and comments), and (4) the degree and causes of duplicated answers. We are not aware of any prior work that analyzes SO posts in these aspects.

6.4 Duplication Detection Relevant to Stack Overflow

Researchers used clone detection to identify duplication within SO or between SO and software products [68, 69, 70, 71, 72]. Specifically, Ahasanuzzaman *et al.* detected duplicated SO questions with machine learning [69]. An *et al.* compared code between SO and Android apps, and observed unethical code reuse phenomena on SO [70]. Yang *et al.* found 1.9M Python snippets in GitHub projects to be clones of code in SO [72]. Chen and Kim built a tool to (1) search for SO questions that have code clones of a given snippet under debugging, and (2) suggest code based on the related answer posts [68]. Different from prior work, we did not invent new clone detection techniques, or compare code between SO and software projects. We used clone detection to (1) sample crawled security-related code, and (2) explore why SO users posted similar code to answer questions.

Chapter 7

Conclusion

By extracting and studying 1,429 security-related answers posts and their corresponding community feedback, we assessed the reliability of the crowdsourced knowledge of security implementation on SO. The quantitative results of contrasting secure answers with insecure answers represent a negative view toward learning or reusing code examples on SO for security implementations.

In general secure and insecure advice more or less balance each other (55% secure and 45% insecure). Users without security knowledge may heavily rely on the community to provide helpful feedback in order to identify secure advice. Unfortunately, we found the community's feedback to be almost useless. For several cryptographic API usage scenarios, the situation is even worse: insecure coding suggestions about SSL/TLS and `SecureRandom` APIs dominate the available options. This is particularly alarming as SSL/TLS is one of the most common use cases in production systems according to prior work [8].

The reputation mechanism and voting system popularly used in crowdsourcing platforms turn out to be powerless to remove or discourage insecure suggestions. Insecure answers were suggested by people with high reputation and widely accepted as easy fixes for programming errors. On average, insecure answers received more votes, comments, favorites, and views than secure answers. As a countermeasure, security evaluation can be included in the voting and reputation system to establish missing incentives for providing secure and correcting insecure content.

When users are motivated to earn reputation by answering more questions, the platform encourages contributors to provide duplicated, less useful, or insecure coding suggestions. Therefore, with respect to security, SO's gamification approach counteracts its original purpose as it promotes distribution of secure and insecure code. Although we did not identify any malicious user that misuses SO to propagate insecure code, we do not see any mechanism designed to prevent such malicious behaviors, either.

When developers refer to crowdsourced knowledge as one of the most important information resources, it is crucially important to enhance the quality control of crowdsourcing platforms. This calls for a strong collaboration between developers, security experts, tool builders, educators, and platform providers. By educating developers to contribute high-quality security-related information, and integrating vulnerability and duplication detection tools into platforms, we can improve software quality via crowdsourcing.

The future work of this project includes three directions. First, the reasons for insecure posts on SO worth more in-depth studies. Understanding these reasons help us to design better mechanisms to reduce the number of insecure posts. For example, if high score answers become insecure because of newly found vulnerabilities, shall we incent users to update these answers? Second, our labeling data can be used to build assistant tools, for example as browser plugins, to help users of SO avoid insecure code examples. One can also use the data for building security analysis tools, for example using the machine learning techniques of document classification by treating code snippets as documents. Lastly, we have set up an infrastructure for analyzing the large amount of SO data, it can be applied to understand more aspects about SO, such as the duplication of questions and the response time to questions.

Bibliography

[1] KSOAP 2 Android with HTTPS. https://stackoverflow.com/questions/3440062, 2010.

[2] Stack Overflow goes beyond Q&As and launches crowd-sourced documentation. https://techcrunch.com/2016/07/21/stack-overflow-goes-beyond-qas-and-launches-crowdsourced-documentation/, 2016.

[3] Stack Overflow's Crowdsourcing Model Guarantees Success. https://www.theatlantic.com/technology/archive/2010/11/stack-overflows-crowdsourcing-model-guarantees-success/66713/, 2010.

[4] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. Design Lessons from the Fastest Q&A Site in the West. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2857–2866, New York, NY, USA, 2011. ACM.

[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 *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, May 2016.

[6] C. Treude, O. Barzilay, and M. Storey. How do programmers ask and answer questions on the web?: NIER track. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 804–807, May 2011.

[7] L. An, O. Mlouki, F. Khomh, and G. Antoniol. Stack Overflow: A code laundering

platform? In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 283–293, February 2017.

[8] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack Overflow considered harmful? The impact of copy&paste on Android application security. In *38th IEEE Symposium on Security and Privacy*, 2017.

[9] Na Meng, Stefan Nagy, Daphne Yao, Wenjie Zhuang, and Gustavo Arango Argoty. Secure coding practices in Java: Challenges and vulnerabilities. In *ICSE*, 2018.

[10] Fred Long. Software vulnerabilities in Java. Technical Report CMU/SEI-2005-TN-044, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[11] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why Eve and Mallory love Android: An analysis of Android SSL (in)security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS, pages 50–61, New York, NY, USA, 2012. ACM.

[12] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 38–49, New York, NY, USA, 2012. ACM.

[13] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in Android applications. In *Proceedings of the ACM Conference on Computer and Communications Security*, CCS, pages 73–84, New York, NY, USA, 2013. ACM.

[14] David Lazar, Haogang Chen, Xi Wang, and Nickolai Zeldovich. Why does cryptographic

software fail? A case study and open problems. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, APSys '14, pages 7:1–7:7, New York, NY, USA, 2014. ACM.

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

[16] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? A large-scale study of Stack Overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, Sep 2016.

[17] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. Live API documentation. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 643–652, New York, NY, USA. ACM.

[18] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *TSE*, pages 654–670, 2002.

[19] Pawel Jurczyk and Eugene Agichtein. Discovering authorities in question answer communities by using link analysis. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, pages 919–922, New York, NY, USA, 2007. ACM.

[20] H. Xie, J. C. S. Lui, and D. Towsley. Incentive and reputation mechanisms for online crowdsourcing systems. In *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pages 207–212, June 2015.

[21] Aikaterini Katmada, Anna Satsiou, and Ioannis Kompatsiaris. A reputation-based incentive mechanism for a crowdsourcing platform for financial awareness. In *Interna-*

tional Workshop on the Internet for Financial Collective Awareness and Intelligence, pages 57–80, 2016.

[22] Privileges. https://stackoverflow.com/help/privileges.

[23] Mengsu Chen, Felix Fischer, Na Meng, Xiaoyin Wang, and Jens Grossklags. How reliable is the crowdsourced knowledge of security implementation? In *The 41st ACM/IEEE International Conference on Software Engineering (ICSE)*, 2019.

[24] M. Chen, F. Fischer, N. Meng, X. Wang, and J. Grossklags. How Reliable is the Crowdsourced Knowledge of Security Implementation? *arXiv e-prints*, January 2019.

[25] The Success of Stack Exchange: Crowdsourcing + Reputation Systems. https://permut.wordpress.com/2012/05/03/the-success-of-stack-exchange-crowdsourcing-reputation-systems/, 2012.

[26] What is reputation? How do I earn (and lose) it? https://stackoverflow.com/help/whats-reputation.

[27] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7525, May 2015.

[28] KSOAP 2 Android with HTTPS. https://stackoverflow.com/questions/4957359, 2010.

[29] Android - Crittografy Cipher decrypt doesn't work. https://stackoverflow.com/questions/14490575, 2013.

[30] RSA Encryption-Decryption : BadPaddingException : Data must start with zero. https://stackoverflow.com/questions/14086194, 2012.

[31] How do I use 3DES encryption/decryption in Java? https://stackoverflow.com/questions/20670, 2008.

[32] where to store confidential data like decryption key in android so that it can never be found by hacker. https://stackoverflow.com/questions/35082426, 2016.

[33] Stack Exchange Data Dump. https://archive.org/details/stackexchange, 2018.

[34] Networks-Learning/stackexchange-dump-to-postgres. https://github.com/Networks-Learning/stackexchange-dump-to-postgres, Visited on 7/31/2018.

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

[36] The GNU Crypto project. https://www.gnu.org/software/gnu-crypto/, Visited on 7/31/18.

[37] jasypt. http://www.jasypt.org, 2014.

[38] Arkajit Dey and Stephen Weis. *Keyczar: A Cryptographic Toolkit*.

[39] scribejava. https://github.com/scribejava/scribejava, Visited on 7/31/2018.

[40] spongycastle. https://github.com/rtyley/spongycastle/releases, Visited on 7/31/2018.

[41] Stack Overflow Considered Harmful? The Impact of Copy & Paste on Android Application Security. https://www.aisec.fraunhofer.de/en/stackoverflow.html, Last visited on 7/31/2018.

[42] James E. Gentle. *Computational Statistics*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[43] Michael P Fay and Michael A Proschan. Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys*, 4:1–39, 2010.

[44] Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, 10:545 – 555, 05 2011.

[45] Laboratory of Cryptography and System Security (CrySyS Lab). skywiper (a.k.a. flame a.k.a. flamer): A complex malware for targeted attacks. Technical report, Budapest University of Technology and Economics, 2012.

[46] T. Duong and J. Rizzo. Here come the xor ninjas. Unpublished manuscript 2011.

[47] Fast and simple String encrypt/decrypt in JAVA. https://stackoverflow.com/questions/5220761, 2011.

[48] SSL Certificate Verification: javax.net.ssl.SSLHandshakeException. https://stackoverflow.com/questions/25079751/ssl-certificate-verification-javax-net-ssl-sslh

[49] MessageDigest. https://docs.oracle.com/javase/9/docs/api/java/security/MessageDigest.html, Recently visited on 08/13/2018.

[50] Shaowei Wang, Tse-Hsun Chen, and Ahmed E. Hassan. Understanding the factors for fast answers in technical Q&A websites. *Empirical Software Engineering*, 23(3):1552–1593, Jun 2018.

[51] is there a Java ECB provider? https://stackoverflow.com/questions/5665680, 2011.

[52] Java: How implement AES with 128 bits with CFB and No Padding. https://stackoverflow.com/questions/6252501, 2011.

[53] Android Facebook API won't login. https://stackoverflow.com/questions/22150331, 2014.

[54] Trusting all certificates using HttpClient over HTTPS. https://stackoverflow.com/questions/4837230, 2011.

[55] Trust Anchor not found for Android SSL Connection. http://stackoverflow.com/questions/39883606, 2011.

[56] encrypt message with symmetric key byte[] in Java. http://stackoverflow.com/questions/27621392, 2014.

[57] Keeping answers related to security up to date. https://meta.stackexchange.com/questions/301592/keeping-answers-related-to-security-up-to-date, 2017.

[58] B. He, V. Rastogi, Y. Cao, Y. Chen, V. N. Venkatakrishnan, R. Yang, and Z. Zhang. Vetting SSL usage in applications with SSLINT. In *2015 IEEE Symposium on Security and Privacy*, pages 519–534, May 2015.

[59] Sazzadur Rahaman and Danfeng Yao. Program analysis of cryptographic implementations for security. In *IEEE Security Development Conference (SecDev)*, pages 61–68, 2017.

[60] J. Xie, H. R. Lipford, and B. Chu. Why do programmers make security errors? In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 161–164, September 2011.

[61] R. Balebako and L. Cranor. Improving App Privacy: Nudging App Developers to Protect User Privacy. *IEEE Security & Privacy*, 12(4):55–58, July 2014.

[62] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do Java developers struggle with cryptography APIs? In *Proceedings of the 38th*

International Conference on Software Engineering, ICSE, pages 935–946, New York, NY, USA, 2016. ACM.

[63] A. Bosu, C. S. Corley, D. Heaton, D. Chatterji, J. C. Carver, and N. A. Kraft. Building reputation in stackoverflow: An empirical investigation. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 89–92, May 2013.

[64] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*, 19(3):619–654, Jun 2014.

[65] Muhammad Sajidur Rahman. An empirical case study on Stack Overflow to explore developers' security challenges. Master's thesis, Kansas State University, 2016.

[66] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. Are Code Examples on an Online Q&A Forum Reliable?: A Study of API Misuse on Stack Overflow. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 886–896, New York, NY, USA, 2018. ACM.

[67] B. Vasilescu, V. Filkov, and A. Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*, pages 188–195, Sept 2013.

[68] Fuxiang Chen and Sunghun Kim. Crowd debugging. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 320–332, New York, NY, USA, 2015. ACM.

[69] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. Mining Duplicate Questions in Stack Overflow. In *Proceedings*

of the 13th International Conference on Mining Software Repositories, MSR '16, pages 402–412, New York, NY, USA, 2016. ACM.

[70] L. An, O. Mlouki, F. Khomh, and G. Antoniol. Stack overflow: A code laundering platform? In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 283–293, Feb 2017.

[71] Wei Emma Zhang, Quan Z. Sheng, Jey Han Lau, and Ermyas Abebe. Detecting Duplicate Posts in Programming QA Communities via Latent Semantics and Association Rules. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1221–1229, 2017.

[72] D. Yang, P. Martins, V. Saini, and C. Lopes. Stack Overflow in Github: Any Snippets There? In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 280–290, May 2017.