How Reliable is the Crowdsourced Knowledge of Security Implementation?

Mengsu Chen* Felix Fischer† Na Meng* Xiaoyin Wang‡ Jens Grossklags†
Virginia Tech* Technical University of Munich† University of Texas at San Antonio‡
mschen@vt.edu, flx.fischer@tum.de, nm8247@vt.edu, xiaoyin.wang@utsa.edu, jens.grossklags@tum.de

***Abstract*—Stack Overflow (SO) is the most popular online Q&A site for developers to share their expertise in solving programming issues. Given multiple answers to a certain question, developers may take the accepted answer, the answer from a person with high reputation, or the one frequently suggested. However, researchers recently observed that SO contains exploitable security vulnerabilities in the suggested code of popular answers, which found their way into security-sensitive high-profile applications that millions of users install every day. This observation inspires us to explore the following questions: How much can we trust the security implementation suggestions on SO? If suggested answers are vulnerable, can developers rely on the community's dynamics to infer the vulnerability and identify a secure counterpart?**

To answer these highly important questions, we conducted a comprehensive study on security-related SO posts by contrasting secure and insecure advice with the community-given content evaluation. Thereby, we investigated whether SO's gamification approach on incentivizing users is effective in improving security properties of distributed code examples. Moreover, we traced the distribution of duplicated samples over given answers to test whether the community behavior facilitates or prevents propagation of secure and insecure code suggestions within SO.

We compiled 953 different groups of similar security-related code examples and labeled their security, identifying 785 secure answer posts and 644 insecure answer posts. Compared with secure suggestions, insecure ones had higher view counts (36,508 vs. 18,713), received a higher score (14 vs. 5), and had significantly more duplicates (3.8 vs. 3.0) on average. 34% of the posts provided by highly reputable so-called *trusted users* were insecure.

Our findings show that based on the distribution of secure and insecure code on SO, users being laymen in security rely on additional advice and guidance. However, the community-given feedback does not allow differentiating secure from insecure choices. The reputation mechanism fails in indicating trustworthy users with respect to security questions, ultimately leaving other users wandering around alone in a software security minefield.

***Index Terms*—Stack Overflow, crowdsourced knowledge, social dynamics, security implementation**

I. Introduction

Since its launch in 2008, Stack Overflow (SO) has served as the infrastructure for developers to discuss programming-related questions online, and provided the community with crowdsourced knowledge [1], [2]. Prior work shows that SO is one of the most important information resources that developers rely on [3], [4]. Meanwhile, researchers also revealed that some highly upvoted, or even accepted answers on SO contained insecure code [5], [6]. More alarmingly, Fischer *et al.* found that insecure code snippets from SO were copied and pasted into 196,403 Android applications available on Google Play [5]. Several high-profile applications containing particular instances of these insecure snippets were successfully attacked, and user credentials, credit card numbers and other private data were stolen as a result [7].

These observations made us curious about SO's reliability regarding suggestions for security implementations. Taking a pessimistic view, such insecure suggestions can be expected to be prevalent on the Q&A site, and consistent corrective feedback by the programming community may be amiss. Consequently, novice developers may learn about incorrect crowdsourced knowledge from such Q&A sites, propagate the misleading information to their software products or other developers, and eventually make our software systems vulnerable to known security attacks.

Therefore, within this paper, we conducted *a comprehensive in-depth investigation of the popularity of both secure and insecure coding suggestions on SO, and the community activities around them*. To ensure a fair comparison between secure and insecure suggestions, we focused on the discussion threads related to Java security. We used Baker [8] to mine for answer posts that contain any code using security libraries, and extracted 25,855 such code snippets. We reused the security domain expertise summarized by prior work [5] to manually label whether a given code snippet is secure or not. However, different from prior work [5] that studied the application of insecure SO answers to production code, our work focuses on the SO suggestions themselves. More specifically, we studied coding suggestions' popularity, social dynamics, and duplication. We also inquired how developers may be misled by insecure answers on SO.

To identify prevalent topics on SO, we used CCFinder [9] to detect code clones (*i.e.*, duplicated code) in the data extracted by Baker. These clones are clustered within clone groups. 953 clone groups were observed to use security library APIs and implement functionalities like SSL/TLS, symmetric and asymmetric encryption, *etc.* Moreover, we found that code examples within clone groups are more likely to be viewed than non-duplicated code snippets on SO. This further motivates our sampling method as we can expect clones to have a higher impact on users and production code. Among the 953 clone groups, there were 587 groups of duplicated secure

This work was supported by ONR Grant N00014-17-1-2498.

code, 326 groups of similar insecure code, and 40 groups with a mixture of secure and insecure code snippets. These clone groups covered 1,802 secure code snippets and 1,319 insecure ones. By mapping cloned code to their container posts, we contrasted insecure suggestions with secure ones in terms of their popularity, users' feedback, degree of duplication, and causes for duplication.

We 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 [4]–[6]. 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).
- **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 [10]–[12]. We conducted statistical testing to compare secure and insecure answers in terms of votes, answerers' reputations, *etc.*
- **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.
- **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.

In our study, we made four major observations:

1) *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. 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.
2) *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* scores, the effect size is negligible (<0.147). 239 of the 536 examined *accepted answers* (45%) are insecure. 26 out of the 72 posts (36%) suggested by "trusted users" (with ≥20K reputation scores [13]) are insecure. These observations imply that reputation and voting on SO are not reliable to help users distinguish between secure and insecure answers.
3) *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.
4) *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 reputation the user gains.

Our source code and data set are available at https://github.com/mileschen360/Higgs.

Fig. 1: A typical SO discussion thread contains one question post and one or multiple answer posts [14]

II. Background

To facilitate the discussion of SO community activities around security implementations, we will first introduce SO's crowdsourcing model, and then summarize the domain knowledge used to label secure and insecure code snippets.

A. Stack Overflow as a Crowdsourcing Platform

Some observers believe that the success of SO lies in its crowdsourcing model and the reputation system [2], [15]. The four most common forms of participation are i) question asking, ii) question answering, iii) commenting, and iv) voting/scoring [15]. Figure 1 presents an exemplar SO discussion thread, which contains one question and one or many answers.

TABLE I: 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

When multiple answers are available, the asker decides which answer to **accept**, and marks it with "✓".

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 [16]. All users initially have only one reputation point. As users gain more reputation, they are granted more administrative privileges to help maintain SO posts [13]. 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.

B. Categorization of Java Security Implementations

Based on state-of-the-art security knowledge, researchers defined five categories of security issues relevant to library misuses [5]. Table I shows their criteria, which we use in this project to decide whether a code snippet is insecure or not.

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 [7]. 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 [17]. 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 1 shows a vulnerable snippet that allows all hosts, trusts all certificates, and uses TLS v1.0.

Listing 1: An example to demonstrate three scenarios of insecurely using SSL/TLS APIs [18]

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

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

Listing 2: An example to present several insecure usage scenarios of symmetric cryptography [19]

```
// Declare a key parameter with a static value
private static byte[] key = "12345678".getBytes();
// Declare an IV parameter with a static value
private static byte[] iv = "12345678".getBytes();
public static String encrypt(String in) {
  String cypert = in;
  try {
    IvParameterSpec ivSpec = new IvParameterSpec(iv);
    // Create a secret key with the DES cipher
    SecretKeySpec k = new SecretKeySpec(key, "DES");
    // Declare a DES cipher
    Cipher c = Cipher.getInstance("DES/CBC/PKCS7Padding");
    c.init(Cipher.ENCRYPT_MODE, k, ivSpec);
    ... }    }
```

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 3 shows a vulnerable code example.

Listing 3: An example that insecurely uses RSA by specifying the key size to be 1024 [20]

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

Hash: In the context of password-based key derivation, digital signatures, and authentication/authorization, developers may explicitly invoke broken hash functions. Listing 4 shows an example using MD5.

Listing 4: Insecurely creating a message digest with MD5 [21]

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

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 5 presents an example using `SecureRandom` insecurely.

Listing 5: Using `SecureRandom` with a static seed [22]

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

III. Methodology

To collect secure and insecure answer posts, we first extracted code snippets from SO that used any security API (Section III-A). Next, we sampled the extracted code corpus by detecting duplicated code (Section III-B). Finally, we manually labeled sampled code as secure, insecure, or irrelevant, and mapped the code to related posts (Section III-C). Additionally, we compared the view counts of the sampled posts vs. unselected posts to check samples' prevalence (Section III-D).

A. 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 [23], we used a tool stackexchange-dump-to-postgres [24] 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 II: 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, ...

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 [6], 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 II, these tags are either related to Java platforms, third-party security libraries or tools, or security concepts.

2) Filtering by security API usage: Similar to prior work [5], we used Baker [8] to decide whether a snippet calls any security API. This paper focuses on the following APIs:

- Java platform security: org.jetf.jgss.*, android.security.*, com.sun.security.*, java.security.*, javax.crypto.*, javax.net.ssl.*, javax.security.*, javax.xml.crypto.*;
- Third-party security libraries: BouncyCastle [25], GNU Crypto [26], jasypt [27], keyczar [28], scribejava [29], SpongyCastle [30].

After taking in a set of libraries and a code snippet, Baker (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`, Baker compares the invoked methods on `C` against the method APIs declared by each candidate type, and chooses the candidate that contains more matching methods.

We included a code snippet if Baker finds at least one API (class or method) with an exact match. However, Baker's result set is not fully accurate and requires a number of post-processing steps to reduce false positives. These include a blacklist filter for standard Java types (e.g., String) and very popular methods (e.g., get()). Baker's oracle contains only the given security APIs, which helped reduce false positives when detecting secure code but did not help reduce false negatives.

B. Clone Detection

With the filters described above, we identified 25,855 code snippets (from 23,224 posts) that are probably security-related. 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 [9], 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.

C. Code Labeling

We manually examined each of those 8,690 snippets and labeled code based on the criteria mentioned in Section II. If a snippet meets any criteria of insecure code shown in Table I, 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

TABLE III: 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

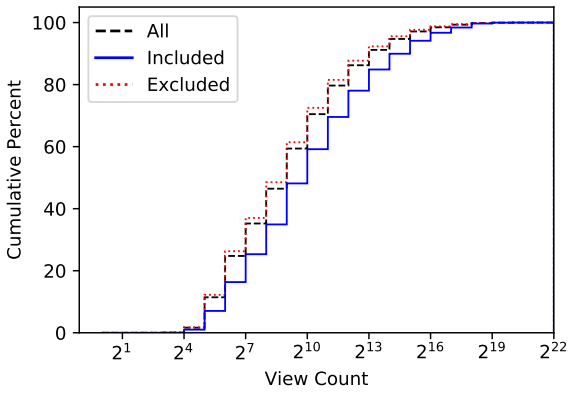

Fig. 2: CDFs of view count among the included answers, excluded ones, and all answers related to Baker's output

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 III 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 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 III-A), 64% of the clone groups from refined data were still irrelevant to security, which evidences the difficulty of precisely identifying security implementation with Baker.

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 consider a post "insecure" if it contains any labeled insecure code. A post was labeled "secure" if it has no insecure snippet but at least one secure snippet. If a post does not contain any (in)secure snippet, it is labeled as "irrelevant".

D. 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) [31] of view count for the included 3,562 posts (as mentioned in Table III), the excluded 19,662 posts, and the complete set of 23,224 posts identified by Baker. As shown in Fig. 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.

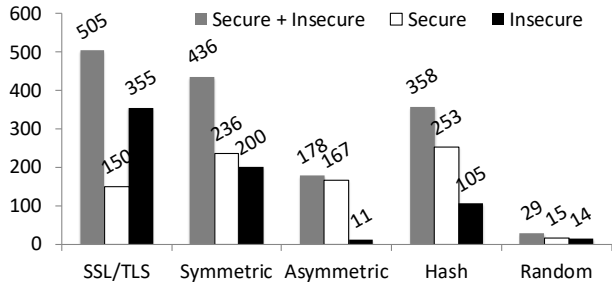

Fig. 3: The distribution of posts among different categories

IV. Major Findings

In this section, we present our results and discuss the main findings regarding our stated research questions.

A. Popularity of Secure and Insecure Code Suggestions

Figure 3 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 3 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, indicating that developers should be quite cautious when searching for such code. Meanwhile, secure answers dominate the other categories, accounting for 94% of *Asymmetric* posts, 71% of *Hash* posts, 54% of *Symmetric* posts, and 52% of *Random* posts. However, notice that across these 4 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, 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 2014-2015. In 2008, 2009, and 2011, insecure answers were posted more often than secure ones, taking up 53%-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 normally distributed data [32]), and calculated the Cliff's delta

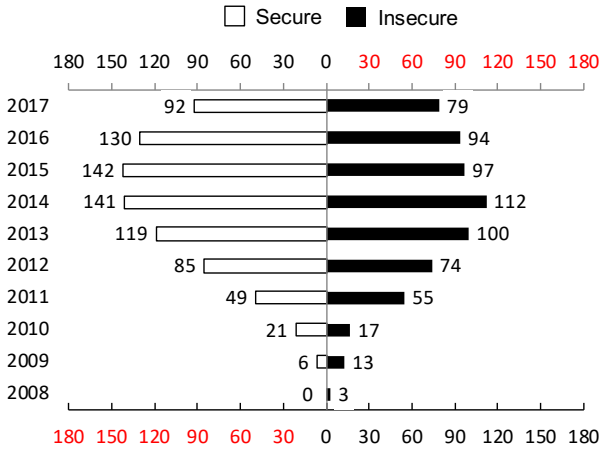

Fig. 4: The distribution of posts over during 2008-2017

size (which measures the difference's magnitude [33]). 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 [7], [34]–[36], 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 [37]. 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 [34], as shown in Fig. 5, 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.*

B. Community Dynamics Towards Secure and Insecure Code

For each labeled secure or insecure post, we extracted the following information: (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.

Comparison of Mean Values. Table IV 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

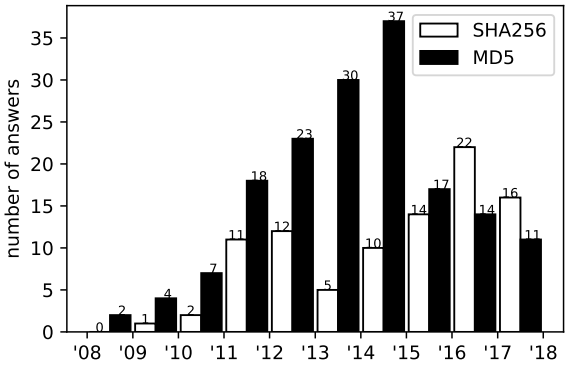

Fig. 5: Distributions of MD5 and SHA256 related posts

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.

Three reasons can explain our observation. First, software developers often face time constraints. When stuck with coding issues (*e.g.*, runtime errors), developers are tempted to take insecure but simpler solutions [6]. Take the vulnerable SSL/TLS usage in Listing 1 for example. The insecure code was frequently suggested on SO, and many users voted for it mainly 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 [39].

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 [40]. Third, insecure posts are less recent and may have accumulated more positive scores than recent secure posts.

> **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 IV shows that among all posts, insecure ones obtained significantly more comments ($p = 0.02$) and views ($p = 1.5\text{e}{-3}$), while the effect sizes are negligible. Specifically for the *Random* category, insecure posts have significantly higher view counts ($p = 0.01$) and the effect size is large. Meanwhile, the owners of secure answer posts have significantly higher reputation ($p = 0.02$) but the magnitude is also negligible.

TABLE IV: 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 [38], 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".

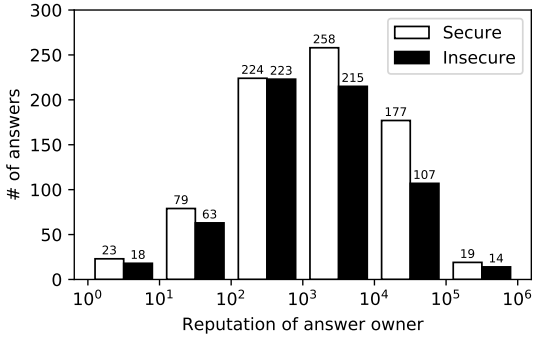

Fig. 6: The answer distribution based on owners' reputation

Figure 6 further clusters answers based on their owners' 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 different reputation groups. For instance, most answers were provided by users with scores within $[10^2, 10^4)$, accounting for 61% of secure posts and 68% 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.*

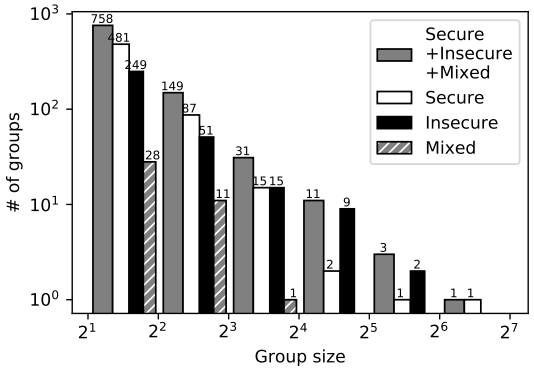

Fig. 7: The distribution of clone groups based on their sizes

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.*

C. 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 7 shows the distribution of clone groups based on their sizes. Similar to Fig. 6, we used logarithmic scales for both the horizontal and vertical axes. In Fig. 7, most clone groups are small, with 2-3 similar snippets.

TABLE V: 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)

The number of groups decreases dramatically as the group size increases. Interestingly, within $[2^4, 2^6)$, there are more insecure groups than secure ones. Table V compares the sizes of secure and insecure groups. Surprisingly, insecure groups have significantly larger sizes than secure ones, 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 code's repetitiveness to decide the security property.

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 6: A clone group with both secure and insecure code

```
// An insecure snippet using AES/ECB to create a cipher [41]
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding",
    "SunJCE");
Key skeySpec=KeyGenerator.getInstance("AES").generateKey();
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
System.out.println(Arrays.toString(cipher.doFinal(new byte
    [] { 0, 1, 2, 3 })));
//A secure snippet using AES/CFB to create a cipher [42]
final Cipher cipher=Cipher.getInstance("AES/CFB/NoPadding",
    "SunJCE");
final SecretKey skeySpec=KeyGenerator.getInstance("AES").
    generateKey();
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
System.out.println(Arrays.toString(cipher.doFinal(new byte
    [] { 0, 1, 2, 3 })));
```

Listing 6 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.*

D. 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 7. 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 7: An exemplar snippet to generate a key hash for Facebook login [43]

```
PackageInfo info = getPackageManager().getPackageInfo("com.
    facebook.samples.hellofacebook", PackageManager.
    GET_SIGNATURES);
for (Signature signature : info.signatures) {
  MessageDigest md = MessageDigest.getInstance("SHA");
  md.update(signature.toByteArray());
  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 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 ...*" [44]. 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. For instance, a user posted code without reading the question and received down-vote (i.e., -1) [45]. 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.*

V. Related Work

A. Security API Misuses

Prior studies showed that API misuses caused security vulnerabilities [5], [7], [35], [46]–[50]. 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 [48]. 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 [47], 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.* [5] is closely related to our work. In their work, secure and insecure snippets from SO were used to search for code clones in Android apps. Our research is different in three aspects. First, it investigates the evolution and distribution of secure and insecure coding suggestions within the SO ecosystem itself. Second, while [5] compares average score and view counts for secure and insecure snippets, they merely do this for snippets whose exact copies have migrated into apps but not for our much broader set of snippets on SO. Therefore, the dataset of [5] is not representative to evaluate the impact of severity, community's awareness, and popularity of unreliable SO suggestions on secure coding. Third, we conducted not only statistical testing on a comprehensive dataset to quantitatively contrast score, view count, comment count, reputation, and favorite count, but also case studies to qualitatively analyze the differences. We further explored the missing link between gamification and security advice quality on crowdsourcing platforms.

B. Developer Studies

Researchers conducted interviews or surveys to understand developers' security coding practices [4], [51]–[53]. For example, Nadi *et al.* surveyed 48 developers and revealed that developers found it difficult to use cryptographic algorithms correctly [53]. 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 [51]. Acar *et al.* surveyed 295 developers and conducted a lab user study with 54 students and professional Android developers [4]. 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.

C. Empirical Studies on Stack Overflow

Researchers conducted various studies on SO [3], [6], [50], [54]–[57]. Specifically, Zhang *et al.* studied the JDK API usage recommended by SO, and observed that 31% of the studied posts misused APIs [6]. Meng *et al.* manually inspected 503 SO discussion threads related to Java security [6]. 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 [3]. Vasilescu *et al.* studied the associations between SO and GitHub, and found that GitHub committers usually ask fewer questions and provide more answers [58]. 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 [54].

In comparison, our paper 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.

D. Duplication Detection Related to SO or Vulnerabilities

Researchers used clone detection to identify duplication within SO or between SO and software products [59]–[63]. Specifically, Ahasanuzzaman *et al.* detected duplicated SO questions with machine learning [60]. An *et al.* compared code between SO and Android apps and observed unethical code reuse phenomena on SO [61]. Other researchers used static analysis to detect vulnerabilities caused by code cloning [64]–[67]. For instance, Kim et al. generate a fingerprint for each Java method to efficiently search for clones of a given vulnerable snippet [67]. 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.

VI. Our 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 [68]. Below are our recommendations based on this analysis.

a) ***For Tool Builders****:* Explore approaches that accurately and flexibly detect and fix security bugs. Although a few tools identify security API misuses through static program analysis or machine learning [5], [47], [69], [70], they are unsatisfactory due to the (1) hard-to-extend API misuse patterns hardcoded in tools, and (2) hard-to-explain machine learning results. People report vulnerabilities and patches on

CVE and in security papers. Tools like LASE [71] were built to (i) generalize program transformation from concrete code changes, and (ii) leverage the transformation to locate code for similar edits. If tool developers can extend such tools to compare secure-insecure counterparts, they can automatically fix vulnerabilities in a flexible way.

*b) **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. As user profiles include top tags to reflect the frequently asked/answered questions by users. instead of accumulating a single reputation score for each user, SO developers can compute one score for each top tag to better characterize users' expertise.

*c) **For Designers of Crowdsourcing Platforms**:* Provide incentives to users for downvoting or detailing vulnerabilities and suggesting secure alternatives. Introduce certain mechanisms to encourage owners of outdated or insecure answers to proactively archive or close such posts. We expect researchers from the usable security and HCI domain to evaluate and test new design patterns that integrate security evaluation in the gamification approach.

VII. Threats to Validity

a) Threat to External Validity: This study labels insecure code based on the Java security rules summarized by prior work [5], so our studied insecure snippets are limited to Java code and these rules. Since we used the state-of-the-art insecurity criteria, our analysis revealed as diverse insecure code as possible. In the future, we plan to identify more kinds of insecure code by considering different programming languages and exploiting multiple vulnerability detection tools [72], [73].

b) Threat 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 assume a snippet to be secure if it does not match any given rule. 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.

c) Threat 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.

VIII. Conclusion

We aimed to assess the reliability of the crowdsourced knowledge on security implementation. Our analysis of 1,429 answer posts on SO revealed 3 insights.

1) In general secure and insecure advices 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 certain cryptographic API usage scenarios, the situation is even worse: insecure coding suggestions about SSL/TLS 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 [5].
2) 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.
3) 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. Our future work will focus on building the needed tool support.

Acknowledgment

We thank reviewers for their insightful comments. We also thank Dr. Kurt Luther for his valuable feedback.

REFERENCES

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

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

[3] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design Lessons from the Fastest Q&A Site in the West," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2857–2866.

[4] 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)*, May 2016, pp. 289–305.

[5] 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 *38th IEEE Symposium on Security and Privacy*, 2017.

[6] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. A. Argoty, "Secure coding practices in Java: Challenges and vulnerabilities," in *ICSE*, 2018.

[7] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. 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*, ser. CCS. New York, NY, USA: ACM, 2012, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382205

[8] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, pp. 643–652.

[9] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *TSE*, pp. 654–670, 2002.

[10] P. Jurczyk and E. 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*. New York, NY, USA: ACM, 2007, pp. 919–922.

[11] 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)*, June 2015, pp. 207–212.

[12] A. Katmada, A. Satsiou, and I. Kompatsiaris, "A reputation-based incentive mechanism for a crowdsourcing platform for financial awareness," in *International Workshop on the Internet for Financial Collective Awareness and Intelligence*, 2016, pp. 57–80.

[13] "Privileges," https://stackoverflow.com/help/privileges.

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

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

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

[17] Y. Sheffer, R. Holz, and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," RFC 7525, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7525.txt

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

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

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

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

[22] "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.

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

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

[25] "Bouncy castle," https://www.bouncycastle.org.

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

[27] "jasypt," http://www.jasypt.org, 2014.

[28] A. Dey and S. Weis, *Keyczar: A Cryptographic Toolkit*.

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

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

[31] J. E. Gentle, *Computational Statistics*, 1st ed. Springer Publishing Company, Incorporated, 2009.

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

[33] "Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, pp. 545 – 555, 05 2011. [Online]. Available: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1657-92672011000200018&nrm=iso

[34] Laboratory of Cryptography and System Security (CrySyS Lab), "sky-wiper (a.k.a. flame a.k.a. flamer): A complex malware for targeted attacks," Budapest University of Technology and Economics, Tech. Rep., 2012.

[35] 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 ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, pp. 38–49.

[36] T. Duong and J. Rizzo, "Here come the xor ninjas," unpublished manuscript 2011.

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

[38] S. Wang, T.-H. Chen, and A. E. Hassan, "Understanding the factors for fast answers in technical Q&A websites," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1552–1593, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9558-5

[39] "SSL Certificate Verification: javax.net.ssl.SSLHandshakeException," https://stackoverflow.com/questions/25079751/ssl-certificate-verification-javax-net-ssl-sslhandshakeexception.

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

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

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

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

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

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

[46] F. Long, "Software vulnerabilities in Java," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2005-TN-044, 2005. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7573

[47] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in Android applications," in *Proceedings of the ACM Conference on Computer and Communications Security*, ser. CCS. New York, NY, USA: ACM, 2013, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516693

[48] D. Lazar, H. Chen, X. Wang, and N. Zeldovich, "Why does cryptographic software fail? A case study and open problems," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, ser. APSys '14. New York, NY, USA: ACM, 2014, pp. 7:1–7:7. [Online]. Available: http://doi.acm.org/10.1145/2637166.2637237

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

[50] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? A large-scale study of Stack Overflow posts," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 910–924, Sep 2016. [Online]. Available: https://doi.org/10.1007/s11390-016-1672-0

[51] 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)*, Sep. 2011, pp. 161–164.

[52] R. Balebako and L. Cranor, "Improving App Privacy: Nudging App Developers to Protect User Privacy," *IEEE Security & Privacy*, vol. 12, no. 4, pp. 55–58, Jul. 2014.

[53] 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*, ser. ICSE. New York, NY, USA: ACM, 2016, pp. 935–946. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884790

[54] 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)*, May 2013, pp. 89–92.

[55] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in Stack Overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, Jun 2014. [Online]. Available: https://doi.org/10.1007/s10664-012-9231-y

[56] M. S. Rahman, "An empirical case study on Stack Overflow to explore developers' security challenges," Master's thesis, Kansas State University, 2016.

[57] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. 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*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 886–896.

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

[59] F. Chen and S. Kim, "Crowd debugging," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 320–332. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786819

[60] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Mining Duplicate Questions in Stack Overflow," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 402–412.

[61] 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)*, Feb 2017, pp. 283–293.

[62] W. E. Zhang, Q. Z. Sheng, J. H. Lau, and E. 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*, 2017, pp. 1221–1229.

[63] 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)*, May 2017, pp. 280–290.

[64] N. H. Pham, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Detection of recurring software vulnerabilities," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 447–456. [Online]. Available: http://doi.acm.org/10.1145/1858996.1859089

[65] J. Jang, A. Agrawal, and D. Brumley, "Redebug: Finding unpatched code clones in entire os distributions," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 48–62. [Online]. Available: https://doi.org/10.1109/SP.2012.13

[66] Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, and J. Hu, "Vulpecker: An automated vulnerability detection system based on code similarity analysis," in *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: ACM, 2016, pp. 201–213. [Online]. Available: http://doi.acm.org/10.1145/2991079.2991102

[67] S. Kim, S. Woo, H. Lee, and H. Oh, "Vuddy: A scalable approach for vulnerable code clone discovery," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 595–614.

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

[69] 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*, May 2015, pp. 519–534.

[70] S. Rahaman and D. Yao, "Program analysis of cryptographic implementations for security," in *IEEE Security Development Conference (SecDev)*, 2017, pp. 61–68.

[71] N. Meng, M. Kim, and K. McKinley, "Lase: Locating and applying systematic edits," in *ICSE*, 2013, p. 10.

[72] "Flawfinder," https://dwheeler.com/flawfinder/.

[73] "Checkmarx," https://www.checkmarx.com.