A Qualitative Investigation of Insecure Code Propagation from Online Forums

Wei Bai
University of Maryland
wbai@umd.edu

Omer Akgul
Univerity of Maryland
akgul@umd.edu

Michelle L. Mazurek
University of Maryland
mmazurek@umd.edu

Abstract—**Research demonstrates that code snippets listed on programming-oriented online forums (e.g., Stack Overflow) – including snippets containing security mistakes – make their way into production code. Prior work also shows that software developers who reference Stack Overflow in their development cycle produce less secure code. While there are many plausible explanations for why developers propagate insecure code in this manner, there is little or no empirical evidence. To address this question, we identify Stack Overflow code snippets that contain security errors and find clones of these snippets in open source GitHub repositories. We then survey (n=133) and interview (n=15) the authors of these GitHub repositories to explore how and why these errors were introduced. We find that some developers (perhaps mistakenly) trust their security skills to validate the code they import, but the majority admit they would need to learn more about security before they could properly perform such validation. Further, although some prioritize functionality over security, others believe that ensuring security is not, or should not be, their responsibility. Our results have implications for attempts to ameliorate the propagation of this insecure code.**

I. Introduction

Many or even most security vulnerabilities do not arise from new or unknown problems; they often occur when software developers make well-known mistakes. For example, insufficient validation of user input is a well-known problem that can lead to vulnerabilities ranging from buffer overflows to SQL injection. Despite the fact that proper checking of user input is all but axiomatic in the security community, vulnerabilities related to this issue remain common and serious [1], [2].

There are many possible explanations for the persistence of security errors that are (in this sense) "solved" problems, including but not limited to human error, lack of security education, overly complex APIs, addressing security at the end of the development lifecycle rather than building it in from the beginning, and the prioritization of other requirements over security [3]–[6].

Prior research suggests that one contributing factor is the presence of insecure code in online forums — such as the programming Q&A site Stack Overflow — that developers reference when searching for help [3], [5]. These works have demonstrated that developers who reference Stack Overflow tend to produce less secure code, and have found evidence of insecurities propagating into production code. Prior work, however, has not explored why or how these effects occur. For example, do developers fail to recognize that code drawn from online forums may not be secure, or do they understand this but reuse it anyway? If they do recognize possible security concerns, why and how do they overcome them before using code? Conventional wisdom suggests that perhaps developers simply prioritize convenience over security [3], [7], but there are other possible explanations. Without understanding developer motivations and practices when propagating this code, it will be difficult to design effective mitigations for this source of vulnerability.

To address these questions, we designed a qualitative, descriptive survey and interview study to investigate why developers propagate insecure code from Stack Overflow. We identified a set of common cryptography-related errors drawn from the security literature, then searched for Stack Overflow posts containing code snippets that exhibit these errors. Using the MOSS code-matching tool, we located GitHub repositories that appear to reuse these snippets. After ranking these repositories according to both their similarity score and various popularity-relevant GitHub statistics, we invited developers of these repositories to take a survey about their projects (without mentioning the potential vulnerabilities), and then invited survey participants to a follow-up interview. In total, 133 developers took the survey, and 15 participated in interviews.

Many participants were defensive about admitting that reusing code from Stack Overflow had led to potential vulnerabilities. Nonetheless, we find that most participants seem cognizant that code drawn from online forums raises potential security concerns. Participants use such code anyway, for several often overlapping reasons. Some participants said they did not know enough about security to properly validate code they encounter, while others trust their own security skills and believe (evidently not always correctly) that they can properly vet such code. Some participants explicitly report prioritizing functionality or convenience over security, and some feel that security is not (or should not be) their job, but rather should be outsourced to a security team or a service provider. Despite the fact that all the projects we identified include some cryptographic code — which suggests some level of interest in security — some participants reported that security was not important to the referenced project.

These results have important implications for how to think about preventing or mitigating the propagation of insecure code from Stack Overflow and other online forums into production code.

The rest of the paper is organized as follows: We describe related work (II), present the study protocol (III), describe the participants and their projects(IV), present our findings(V,VI), and discuss implications from our findings for alleviating insecure code propagation (VII).

II. Related Work

We briefly discuss related work in two key areas: the effects of Q&A platforms such as Stack Overflow on software development in general, and specific investigations of how these platforms intersect with security.

A. *Stack Overflow and software development, broadly*

Vasilescu et al. measured how question asking and answering on Stack Overflow relates to committing changes to open-source repositories‘ [8]. They found that among developers who actively commit to Github repositories, asking and answering questions boosts GitHub commits. Using the same dataset, Wu et al. examined how developers use source code from Q&A platforms [9]. By manually inspecting 289 source code files that reference Stack Overflow posts, they conclude that 31.5% of developers who cite Stack Overflow code modify the code in some way before using it. The authors also find via a survey (n=380) that the majority of developers report preferring to reimplement code rather than reusing code from Stack Overflow directly. The top reason given for reimplementation is that developers need to fit the code into their existing code bases. Yang et al. perform quantitative analysis to find how snippets from Stack Overflow get ported into GitHub projects [10]. While they find empirical evidence of copy & pasting behavior, they claim that only a small portion is exact duplication of code. In this study, we primarily identify exact or very close matches as candidates for our survey and interview.

Treude et al. studied the types of questions that are most commonly asked and answered on Stack Overflow [11], finding that conceptual questions or questions asking for code reviews have a higher chance of being answered. Relatedly, Nasehi et al. investigated attributes that make good answers on Stack Overflow [12]. Via qualitative analysis, they identify how customized the answer is to the original question as the most important feature of a high-quality answer. In a similar study Yang et al. find that question edits are related to a question's quality [13].

Yang, et al. use qualitative and quantitative analysis to investigate how ready-to-use Stack Overflow code snippets are, finding that snippets in dynamically typed languages are substantially more likely than in statically typed languages to be ready to use [14]. In an empirical study, Zhang et al. find that 58.4% of obsolete answers on Stack Overflow were probably obsolete when first posted, and only a small portion of these answers ever get updated [15].

Wang et al. investigate askers and answerers, finding that most askers ask only one question, most answerers answer only one question, and a majority of contributors only ask and never answer [16].

B. *Stack Overflow, code misuse, and security*

An et al. identified code-license violations (such as Creative Commons violations) that seem to originate with code posted to or taken from Stack Overflow [17]. Baltes et al. also note that many copied code snippets are missing attribution [18].

Nadi, et al. investigated Stack Overflow posts to identify problems developers encounter when trying to implement cryptography [6]. They find that a majority of the top cryptography-related posts are associated with confusing API documentation, and many relate to confusion when trying to use symmetric-key encryption. Our participants may have experienced similar confusion, which led them to the Stack Overflow posts we flagged.

In a lab study, Acar et al. found that while Stack Overflow is easier to use than official documentation, it is more likely to result in insecure code [3]. This paper builds on this finding by examining how and why developers use security-relevant, possibly insecure code they may find on Stack Overflow.

Perhaps most closely related to this paper is work by Fischer et al. measuring how code from Stack Overflow propagates to Android applications [5]. The authors use a classifier on program dependency graphs to match security-relevant code snippets from Stack Overflow with code found in Android applications, finding that 15.4% of their Android-application dataset contains security-related code from Stack Overflow, and 97.9% of these applications contain at least one insecure code snippet. Our work complements these findings by examining qualitatively why and how this insecure code propagates via surveys and interviews with implicated developers.

III. Study Protocol

In this study, we first identified security vulnerabilities by reviewing related literature. We then found a set of Stack Overflow posts which included these vulnerabilities. Next, we built a crawler to search for projects on GitHub which reused code snippets from these Stack Overflow posts entirely or in part. Finally, we surveyed and interviewed the developers of these projects to further understand how and why the insecure code was propagated.

It is important to note that, because we have no control group, our results obtained from this methodology are descriptive and qualitative. For context, we provide numbers of participants who provided a given response. Further, for open-ended responses in particular, a participant failing to mention a particular idea may not indicate disagreement; it may simply have been left out. As such, our results are not necessarily statistically generalizable beyond our sample; however, we believe they provide useful insights into the problem of insecure code propagation. See Section III-I for more details.

Our study protocol was approved by the University of Maryland's Institutional Review Board (IRB).

A. *Identifying security vulnerabilities*

We reviewed literature about popular security vulnerabilities related to cryptography, and extracted a few common problems, such as generating insufficiently random numbers, and implementing encryption/decryption incorrectly.

As an example, one common Java vulnerability is use of insecure pseudo-random number generation methods (PRNG), such as `java.util.Random()`. These pseudo-random number generators use a seed and a deterministic algorithm to approximate some properties of truly random number sequences, but the generated sequences are not cryptographically secure and may be predictable to attackers in some situations [19]. As a remedy, Java provides an alternative class, java.security.SecureRandom(), which complies with statistical random number generator tests specified by NIST in [20].

Table I list all the vulnerabilities we investigated in this study, including the sources we drew them from.

B. Identifying insecure Stack Overflow posts

Stack Overflow [1] is a well known Q&A website in the software development community. On Stack Overflow, developers can ask questions about the problems they encounter during development, up- and down-vote answers, and make comments. These answers, comments and ratings can serve as a resource for both the asker and for other developers who may be looking for related answers or topics. However, prior work has found some correlations between Stack Overflow and vulnerable code [3], [5]. While Stack Overflow is not the only such online forum, based on its popularity we chose to use it as a source of potentially insecure code snippets to investigate.

We searched for answers with insecure code snippets related to the vulnerabilities listed in Table I. For instance, to search for code containing ECB mode for encryption, we searched both Google and Stack Overflow for "How to implement AES encryption?" and "AES encryption example in Java" (or other languages). Because prior work we reference focused on Java and Android, we focused on Java as well, but we also identified some snippets with similar vulnerabilities in languages such as PHP, Javascript, and C#. We checked each post we identified manually and consulted with a security expert to ensure that each snippet we selected contained a targeted vulnerability. We then prioritized the selected snippets to focus on the most viewed and most voted for answers.

The number of Stack Overflow posts we identified for each vulnerability is listed in Table II.

C. Searching GitHub repositories

For each selected Stack Overflow post, we identified keywords from the code it contained, which were used to search projects in GitHub. Table I lists one such keyword example (i.e., the line of code in one Stack Overflow post) for each vulnerability. We built a crawler on Amazon Web Services (AWS) to use the GitHub search API to search for projects containing these keywords. Table II lists the number of GitHub repositories we found across all Stack Overflow posts related to that vulnerability.

For each search result (repository), we extracted the matched file and used the MOSS tool (Measure Of Software Similarity, [23]) to calculate the similarity of this code to the code snippet from the relevant Stack Overflow post. We then prioritized our findings based on the MOSS similarity score, as well as repository attributes like the number of *stars*, the number of *watchers*, the number of *forks*, the *last modification date*, the number of *commits*, and the number of *contributors*. In general, we favored repositories with more popular, and more recent, activity. Finally, we recorded the developers responsible for editing these files as potential participants.

D. Recruitment

We sent the identified developers email invitations to participate in our initial survey. Some email addresses were listed on these developers' GitHub profiles; some were identified in the repository commit logs. We applied a marketing tool, *Mail Merge*[2], to send emails in bulk.

The invitation email, which was explicitly designed not to mention security or vulnerabilities, was as follows:

> Hi [username],
>
> We are [institution] researchers conducting a study of how software developers reuse code across GitHub and StackOverflow. We are interested in learning about you and your project [project_name] on GitHub. If you'd like to tell us about it, please consider taking our survey: [url].

Table II shows how many invitations were sent for each vulnerability, as well as how many invitees participated.

E. Survey

The survey contained three main sections.

In the first part, we asked about the participant's software development practices and experiences, such as how many years of software development experience they had, whether they had contributed to open-source software, how often they engage in code review activities, how they evaluate the quality of code obtained from online forums, and how often they asked and answered questions in Stack Overflow or similar websites.

In the second part, we asked participants about their security background, including how often they handle tasks related to security, how they evaluate the security aspects of code obtained from online forums, and whether they have used any software verification or static analysis tools (e.g., FindBugs, Pylint).

In the final part, we asked general demographic questions.

The survey took a median of 10 minutes to complete and was not compensated.

F. Interview

At the end of the survey, we asked respondents if they were willing to participate in a follow-up interview and if they were comfortable conducting an interview in English. We invited willing, English-speaking participants to schedule a video interview via Skype or Google Hangouts. Interviews

[1] https://stackoverflow.com/

[2] https://digitalinspiration.com/product/gmail-mail-merge

ID	Security Vulnerability	Description	Code Snippet Keyword Example
1	Using Crypto-insecure Pseudo-Random Number Generators (RNG) [21]	A Crypto-insecure PRNG increases the attacker's ability to predict the random number. E.g., Using the java.util.Random function instead of java.security.SecureRandom	k += r * random.nextInt(3);
2	Using ECB mode for encryption [21]	ECB block cipher mode for AES is not *semantically secure*, i.e., observing the ciphertext can reveal information about the plaintext.	var decipher = crypto.createDecipheriv('aes-128-ecb', new Buffer(key, 'hex'), '');
3	Using Non-random Initialization Vector (IV) for CBC encryption [21]	Using a non-random IV increases the attacker's ability to guess the plaintext.	private String iv = "fedcba9876543210";
4	Using constant (hardcoded) encryption keys [21], [22]	Increases the attacker's ability to recover the encrypted text.	String myKey = "ThisIsAStrongPasswordForEncryptionAndDecryption";
5	Using constant salts for password-based encryption [21]	Makes the ciphertext vulnerable towards dictionary-based offline attacks	byte[] salt = "DYKSalt".getBytes()
6	Using fewer than 1000 iterations for password based encryption [21]	Decreases the average time needed for an attacker to crack the ciphertext.	PBEParameterSpec pbeParamSpec = new PBEParameterSpec(salt, 20);
7	Using static seeds to seed RNGs [21]	Non-random seeding increases the attacker's ability to predict/guess the next number in the sequence.	Random random = new Random(37461831);
8	Improperly seeding RNGs [22]	Seeding a RNG with a predictable value (e.g., time, process ID) can increase the attacker's ability to guess a future generated number.	init("123456");

TABLE I
List of Security Vulnerabilities Considered for Our Analysis.

ID	Security Vulnerabilities	#SO Posts	#Repos	#Surveys Sent	#Surveys Taken	#Interview
1	Using Crypto-insecure Pseudo-Random Number Generators (RNG)	8	887	221	7	1
2	Using ECB mode for encryption [21]	6	953	271	11	0
3	Using Non-random Initialization Vector (IV) for CBC encryption [21]	15	1766	426	29	4
4	Using constant (hardcoded) encryption keys [21], [22]	18	953	199	7	4
5	Using constant salts for password-based encryption [21]	14	2576	50	6	1
6	Using fewer than 1000 iterations for password based encryption [21]	41	10521	1062	73	9
7	Using static seeds to seed RNGs [21]	1	394	3	0	0
8	Improperly seeding RNGs [22]	2	45	25	0	0

TABLE II
Security Vulnerabilities in Different Phases of Our Study. Vulnerabilities without participants are grayed out.

lasted 30 minutes on average (min 18.5 minutes, max 60 minutes), and participants were compensated with a $15 or equivalent (if in a different currency) Amazon gift card.

In the interview, we first asked about the identified project and its development broadly, including the goal of the project, who it was designed for, whether the participant worked alone or collaboratively, and whether there was a deadline.

Next, we asked specifically about the potentially vulnerable code we had identified. After asking about its functionality, we explained the potential vulnerability. If we had identified other vulnerabilities not taken directly from the matched Stack Overflow post, we also mentioned these vulnerabilities to the participant. We then asked whether someone else had commented on this vulnerability before, whether the vulnerability was important or not, how the problem occurred, and whether and how the participant intended to fix it. More broadly, we asked what supports and obstacles were available when the participant tried to write secure code.

G. Pilot Study

Before conducting the surveys and interviews described above, we conducted a pilot interview study with nine participants. For this pilot, we manually searched Stack Overflow for posts with insecure code snippets and then manually searched GitHub for code that appeared to include these snippets. We invited the developers of these projects to an interview study. These interviews suggested several potential reasons for reusing insecure code, including lack of expertise and tight deadlines; responses to these interviews shaped the questions we included in our final survey and interview protocol.

H. Data Analysis

For closed-item survey questions, we report aggregate descriptive statistics.

To analyze the data from the interviews as well as the two free response questions from the survey, we used open coding with two coders. For the two free-response questions in the survey, the coders initially worked together to code a subset of answers and develop a codebook, stopping when no new codes emerged (19 and 20 responses, respectively). The two coders then independently coded the remaining responses to each question. After discarding any responses deemed invalid (unclear or off-topic) by either coder, the two coders achieved reliability (measured using Cohen's Kappa) of $K = .90$

and 0.82 for the two questions respectively. Kappa values greater than 0.8 are commonly considered “almost perfect agreement” [24].

For the interviews, we first transcribed the audio recordings. Two researchers worked together to open code eight interviews, stopping when no new codes were being added. The two then independently coded 3 interviews (20%) and calculated $K = 0.81$ (again, “almost perfect”). With sufficient reliability, one researcher coded the rest of the interviews.

In all cases, having reached sufficient reliability, the authors resolved all differences for 100% agreement.

I. Limitations

As with most empirical studies, our results should be considered in the context of our limitations.

Most importantly, we were unable to obtain a control group of developers who do not produce vulnerable code when using code from online forums. (It is unclear how such a group could be identified.) Thus, we can use only descriptive statistics, and we do not know if the habits and behaviors observed in this study are specific to developers who produce vulnerable code or not. Further, we cannot estimate whether the projects we identified represent our participants’ typical outcomes. Nonetheless, our findings shed light on the mindsets and behaviors of developers who do at least sometimes produce vulnerable code drawn from online forums, and therefore provide insight into this unfortunate propagation.

Our results exhibit limitations common to self-reported data, including satisficing, social desirability, and difficulty remembering. (We discuss some of these in the context of our results below.) Despite these limitations, our surveys and interviews produced rich data about developers’ motivations and practices, as well as the contexts in which they work.

We examined a limited set of vulnerabilities, exclusively focusing on issues related to misuse of cryptography (Table I). Further, these issues were not evenly represented in the Stack Overflow posts or GitHub repositories that we found, and our participant recruitment is correspondingly unbalanced (Table II): most participants were recruited from three individual vulnerabilities. This limits the generalizability of our findings, but we believe our results still have value for understanding how at least some vulnerabilities propagate from online forums into open-source repositories.

IV. Participants and their projects

In this section we describe the participants in our survey and interview studies; we also describe the projects our participants contributed to in which we identified security problems.

A. Participants

We received 133 complete survey responses. Of these, 58 expressed interest in follow-up interviews and were comfortable using English. We invited 48 of these to interviews; 16 of these scheduled interviews, and one no-showed, resulting in 15 completed interviews. 10 were not scheduled because we had reached saturation in the interviews, with no new themes emerging.

Gender	Male	94.7%
	Female	2.3%
	Others	0.8%
Age	18-24	10.5%
	25-29	33.1%
	30-39	33.1%
	40-49	13.5%
	50+	2.3%
Education	Completed H.S. or below	12.0%
	Some college, no degree	7.5%
	Associate’s degree	1.5%
	Bachelor’s degree	44.4%
	Master’s degree or higher	33.8%
Occupation	Software developer	54.1%
	Faculty member	1.5%
	Graduate students	2.3%
	Undergraduate student	1.5%
Years of development experience	0-2	9.8%
	3-4	19.5%
	5-9	29.3%
	10-14	21.1%
	15-24	15.0%
	25+	5.3%

TABLE III
PARTICIPANT DEMOGRAPHICS FOR SURVEY. PERCENTAGES MAY NOT ADD TO 100% DUE TO “OTHER” CATEGORIES AND ITEM NON-RESPONSE.

Demographic information about the survey and interview participants is summarized in Tables III and IV respectively.

In many respects, our interview participants appear to be a fairly representative sample of our survey respondents. Among survey participants, 116 (87%) reported having attended at least some college, including 59 with a bachelor’s degree and 45 with a postgraduate degree. Survey participants also reported an average of 9.2 years (range: 0.5–35 years) of software development experience and an average of 4.9 years

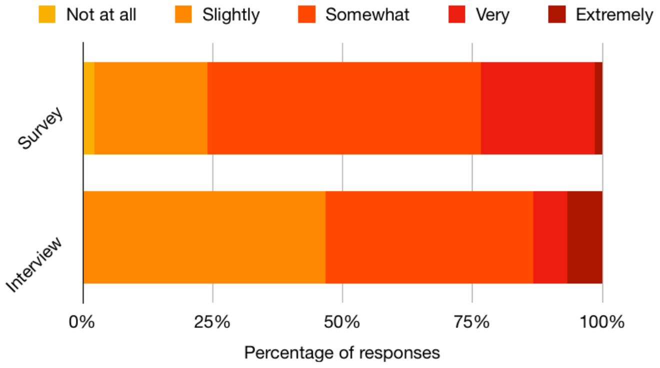

Fig. 1. **Responses to the survey question: “How would you rate your background in computer security?” Answer choices ranged from “Not at all knowledgeable” to “Extremely knowledgeable” on a five-point scale.**

ID	Age	Education	Job	Vuln	Action	Team size	Monetary compensation	Deadline Pressure	Encption usage
P1	38	MS/PhD	G.S.	1	Fixed	2-6	Yes-emp.	N.D.	File encryption with public key cryptography.
P2	18	HS	U.S.	6	N/A	2-6	No	N.D.	User credential storage on client device.
P3	21	BS	Dev.	4	N/A	1	No	Somewhat	Encryption for client-server communications.
P4	41	CND	Dev.	6	Fixed	2-6	Yes-emp.	N.D.	User credential storage on client device.
P5	-	MS/PhD	Dev.	6	N/A	1	Yes-emp.	N.D.	User credential storage on server.
P6	25	BS	Dev.	3,4	N/A	1	No	Somewhat	Encrypting arbitrary strings.
P7	22	MS/PhD	Dev.	3,4	Repo Del.	2-6	No	Very	Enc. for client-server comm. and User data storage.
P8	27	HS	Dev.	3	Acc. Del.	2-6	No	N.D.	Encryption for client-server communications.
P9	28	BS	Dev.	3,4	N/A	1	Yes-emp.	N.D.	Encryption of intellectual property on client device.
P10	20	CND	U.S.	5,6	N/A	1	No	N.D.	User credential storage on client device.
P11	28	CND	Dev.	6	N/A	1	Yes-ent	Somewhat	Unique id generation for client verification.
P12	44	BS	Dev.	5,6	N/A	1	Yes-ent	N.D.	Encryption of software configuration files.
P13	32	BS	G.S.	5,6	N/A	2-6	Yes-emp.	Extremely	FTP functionality integration.
P14	23	BS	-	5,6	N/A	1	No	N.D.	Encryption of data stored in a database.
P15	24	MS/PhD	Dev.	5,6	N/A	2-6	No	Not at all	User data storage in a database.

TABLE IV

Demographics and vulnerable code use description of interview participants. **Education**: **HS**–High School; **CND**–Attended college, no degree; **MS/PhD**–Post graduate degree; **BS**–Bachelor of Science. **Job**: **Dev.**–Software Developer; **G.S.**–Graduate Student; **U.S.**–Undergraduate Student. **Action** indicates the actions taken by participants after our interview: **Fixed**–vulnerability was fixed; **Repo Del.**–entire repository was deleted; **Acc. Del.**–Github account was deleted; **N/A**–no action. **Monetary compensation** means whether and how developers got paid: **Yes-ent.**–paid by developing and selling software; **Yes-emp.**–monetized an entrepreneurial effort. **N.D.**: No Deadline.

(range: 0–18 years) of education in computer science or a related field.

Similarly, 12 (80%) of the interview participants reported having attended at least some college: four reported bachelor's degrees and six reported postgraduate degrees. Our interview participants reported an average of 9.4 years of software development experience (range: 1–35) and 4.2 years (range: 0–11 years) of education.

The majority of survey participants (72, 62%) indicated their primary occupation was software development. In addition, seven respondents indicated they were in academia (two faculty members, three PhD students, and two undergraduates). Overall, 111 survey participants (83%) reported having been employed as a software developer in the past year. Eighty-two survey participants (62%) reported contributing to open-source software projects in the past year.

The statistics for interview participants were fairly similar: 10 (67%) reported a primary occupation of software development, two were undergraduate students, two were graduate students, and one did not specify. The majority (12, 80%) of interview participants were employed as a software developer in the past year, and 10 (67%) reported contributing to open-source projects in the past year.

The vast majority of survey participants (126, 95%) reported being male, along with three female, one other, and three who opted not to answer. Similarly, but unfortunately, all of our interview participants were male. The average age was 33 years for survey participants (range: 18–55) and 28 years for interview participants (range: 18–44).

On average survey participants considered themselves "somewhat knowledgeable" (the middle of five Likert options) about computer security, as illustrated in Figure 1. Interview participants were roughly comparable, but reported slightly less security knowledge overall.

Majorities of both survey and interview participants reported experience with software verification or static analysis tools. Among survey respondents, 82 (62%) reported having such experience, 35 (26%) said they did not, and 10 (8%) did not recognize the terms. For interview participants, these numbers were 11 (73%), three (20%), and one (7%) respectively.

B. Participants' GitHub projects

Here we outline some characteristics of the GitHub projects in which we identified vulnerable code potentially originating from Stack Overflow.

Among the 15 projects associated with our interviewees, four were written for personal use, four for colleagues, and three aimed at software developers generally. Other projects (n=3) were written for specific users populations: stock traders, people who play a specific video game, and students. One was written for the general population.

Two of our interviewees' projects were written as coursework, and one project was created for a job interview. Seven interviewees reported being compensated for their work on the indicated project. For nine projects (60%), the interviewee was the sole author; five (33%) were authored together with friends or colleagues, and one was written cooperatively by the open-source community. Similarly, among survey respondents, 80 (60%) worked on the project alone, 42 (32%) worked in groups of up to five people, two (2%) worked in teams of 6-10 people, two (2%) worked in teams of 11-15 people, and six (5%) worked in groups of more than 15 people.

More than half of the survey respondents (74, 56%) reported having a deadline when working on the identified (potentially vulnerable) project. Similarly, seven interview participants (47%) reported working under a deadline. The reported pressure associated with these deadlines is illustrated in Figure 2.

In the survey, we asked about the languages and development platforms used in the projects we identified. Of 127 respondents with valid answers to this free-response question, the most common language reported was Java (n=56). The most commonly mentioned editors were Eclipse (n=17), Visual Studio (n=11), and Android Studio (n=9). Ten respondents mentioned Linux, nine mentioned Windows, and three mentioned MacOS. Interview participants' responses were fairly typical of these trends: they mentioned Java (n=10), Eclipse (n=3), Android Studio (n=1), and Linux (n=2).

As expected, all interview participants said the potentially vulnerable code was related to encryption. Short descriptions of what the encryption was used for can be found in table IV. None of the interview participants said they had previously been informed about the vulnerability before we pointed it out. After the interview, two participants fixed the vulnerabilities. One deleted that Github repository, and one closed his Github account. The rest of 11 participants didn't make changes for the vulnerabilities we pointed out.

Seven interview participants said the vulnerable code was not important to the function of the overall program. We discuss this further in Section VI-G.

V. Behaviors when drawing from online sources

Here we discuss how our participants report using online sources, such as Stack Overflow, during software development.

A. *Developers do refer to online sources*

The vast majority of both our survey and interview participants report referring to online sources when developing software. We also see that participants report caring about community feedback that is available from these resources.

Among 111 respondents who had been employed as a software developer in the past year, two participants (2%) said they never refer to Stack Overflow during professional development, and seven (6%) said they rarely refer to it. Among 82 respondents who reported contributing to open-source software in the past year, one (1%) and 14 (17%) said they never or rarely referred to the site during such development.

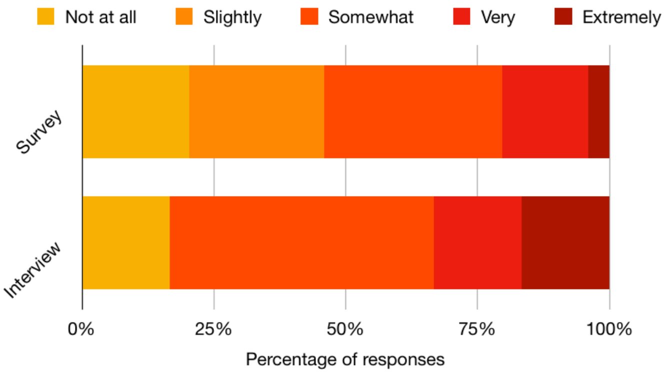

Fig. 2. **Responses to the survey question: "How pressured did you feel to complete the project by the deadline?"**

We also asked specifically about referring to Stack Overflow for questions relating to security. Among past-year open-source contributors, 45 said they never or rarely do so, 28 said they sometimes do so, and 8 said they often or always do so.

When asked about the vulnerable project specifically, 84 out of all 133 participants (63%) reported they had used online forums, 57 (43%) had requested help from other collaborators, and 39 (29%) did both.

These trends are reflected in our interviewees: all reported they used online forums in context of the vulnerable project. (Two interviewees reported in the survey that they had not used online forums for this project, but both did report using such forums for the project during the interview.) In more detailed interview questions, 12 interview participants mentioned Stack Overflow explicitly, including four who mentioned that they use Google search and then click on Stack Overflow threads within the results. For example, P5 said "I used Google to query how to do things, and that normally provides me with Stack Overflow."

In line with prior work [16], [25], 96 of 133 survey respondents reported having asked at least one question on online programming forums, 98 out of 133 answered at least one question on them. In addition, we observe that 34 out of 96 participants asked security related questions; 23 out of 98 participants answered security related questions on the aforementioned platforms.

B. *Many claim to take precautions when importing code*

In the survey, we asked a free-response question about how respondents evaluate the quality of code drawn from online forums. (For the rest of Section V, we report only on the 104 respondents who provided valid answers to this question.) Respondents generally report skepticism of such code, describing several strategies for evaluating it before using it.

Thirty participants (29%) reported checking for good coding practices, including well-documented or well-written code, as well as code with correct syntax and good resource management. Additionally, 25 respondents (24%) mentioned trying to understand the snippet via code review, or code-review-like behaviors, such as tracing the logic of the provided code. For instance, one survey respondent noted, "I evaluate code based on software engineering principles. . . It is important to me that I understand code snippets before I am comfortable using them in my own software." Another common strategy (n=14, 13%) was to write test cases for code the respondent intended to import. Overall, 54 respondents (52%) mentioned at least one of these three strategies, all of which reflect some degree of confidence in their own ability to correctly assess code they plan to reuse. As we will discuss in Section VI-D, many participants also reflect this confidence in a security context.

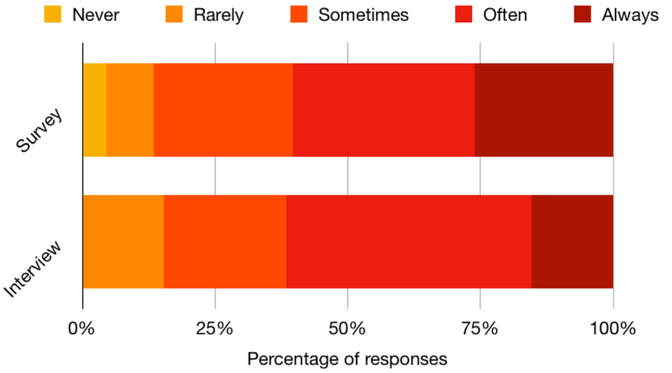

Fig. 3. **Responses to the survey question: "How often do you engage in the following activities in professional software development? - writing tests'**

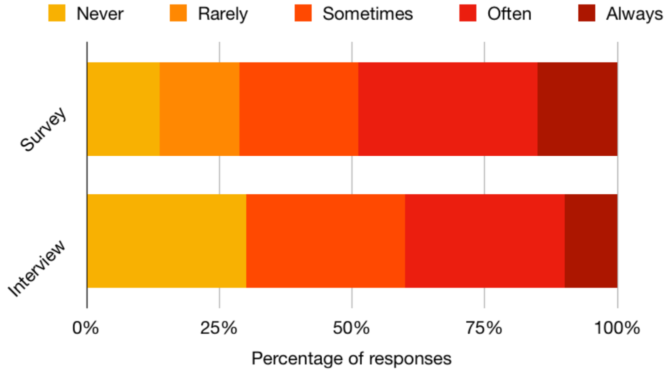

Fig. 4. **Responses to the survey question: "How often do you engage in the following activities in open source software development? - Writing test cases for your code?'**

On the other hand, several respondents described strategies that rely less on their own skills and more on external validation. Sixteen participants (15%) reported using community feedback features to evaluate the trustworthiness of code snippets. Examples include upvotes (n=8), community comments about the snippet (n=3), and author reputation (n=3). Further, seven respondents (7%) mentioned searching for multiple solutions to the same problem and then comparing them to obtain a consistent answer. As we will discuss below, a number of participants report relying on external validation for security code as well, either because they feel they have insufficient knowledge (Section VI-C) or because security is or should be someone else's job (Section VI-F).

C. Others claim they do not copy code

In response to the same free-response question, several developers (17/104, 16%) claimed to rarely or never use code from online forums directly. These participants claimed to gather ideas from online forum answers, but then to modify or even completely rewrite it before using it. One survey participant wrote, "I everything re-parse, re-write in my own code style that any piece of taken code corresponded same naming conventions and other code style guidelines. Every class name, method, property, variable, everything. Even comments could rewrite that more clearly explained what this code does and often re-implement some parts if I have better ideas about them."

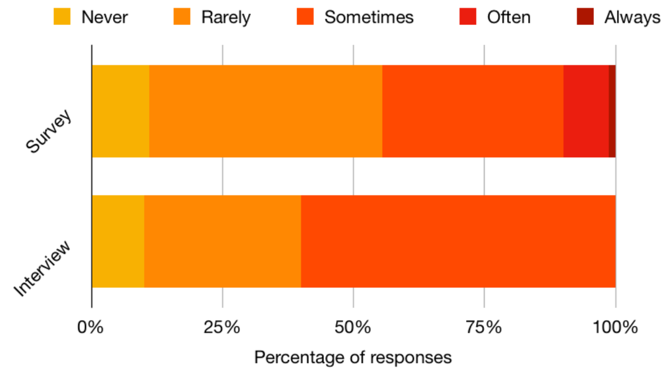

Fig. 5. **Responses to the survey question: "How often do you refer to code snippets relating to computer security on online forums such as Stack Overflow or others?"**

One interviewee (P12), relatedly, claimed copying directly from forums was an error made primarily by junior programmers: "I call it monkey behavior ... it's a problem of juniors. ... I advise them not to do this stuff, search Google or whatever, developer forums, and directly insert code into and run. I think this is not behavior for professional company."

As all of our participants were recruited by identifying code matches with Stack Overflow posts (Section III-C), the participants who claimed never to engage in such copying raise interesting questions. Perhaps the matching was incorrect, perhaps the "modifications" made by the developer were too small to prevent MOSS matching, or perhaps a collaborator added that code. It's also possible, however, that these participants are exhibiting a social desirability bias, because they believe that admitting to copying code from online forums would be embarrassing. We note that we manually checked the code for all interviewees and found the MOSS matching to be correct in all cases.

D. Sometimes functionality is all that matters

A large minority of participants (20/104), by contrast, readily admitted in response to the same free response question that they prioritize code functionality over other concerns. These participants indicated they would run code found in online forums and then accept it, as-is, if the desired functionality was observed. As one survey respondent put it, "I try them out, if they work, I use them." This type of behavior can also be seen in a security context, as we explore further in Section VI-E.

VI. Security-relevant behavior

In this section we describe security relevant practices reported in the survey and interview responses.

A. Introducing security problems

We identified participants because of the overlap between their code and vulnerable code that we found on Stack Overflow. When we asked participants how the vulnerability occurred, however, only five of the 15 interview participants explicitly blamed an external source. (Three mentioned Stack Overflow, one mentioned a book, and one did not specify a source.) The three participants who mentioned Stack Overflow directly admitted the bug might have been caused by copy & pasting. P14 said, "I think it's just copy and paste from another website . . . I need fast-forward encryption techniques, not how it works." Interestingly, P1 blamed the external source while avoiding mentioning copying code: "In that book, there were a lot of receipts you have to follow. Maybe the part to create the random generator was not written in the book. It just says how to do the encryption."

The other causes for these security bugs that participants mentioned included that they weren't prioritizing security (n=8), they didn't know enough to properly do security evaluation (n=4), they prioritized code efficiency (n=2). Two participants said they could not remember. We note that these reasons are not necessarily incompatible with copy & pasting, but rather may explain why they copy & pasted vulnerable code. We explore these issues in more detail in the following subsections.

B. Participants claim to be skeptical about security code

Many participants claimed to be particularly skeptical about drawing code from online forums in a security context.

We asked survey respondents who are open-source software developers and reported referring to security code on stack overflow whether and how they validate such code. Seven out of 43 participants (16%) said they do not make security evaluations at all. (We discuss the strategies used by the other 36 in further subsections). However, only four out of 12 interviewees who said they had used online forums for the vulnerable projects claimed to have considered security aspects when using code from these forums.

There are several possible explanations for this disparity: perhaps our interview sample was not representative of the survey sample in this respect, or perhaps our interview participants typically validate security code found online carefully but didn't for the projects we referenced. On the other hand, the survey responses might reflect a social desirability bias: developers recognize that they "should" perform additional validation for security-relevant code, but in practice they either choose not to or are unable to.

We hypothesize that the propagation of vulnerable code from online forums to GitHub projects suggests either that developers do not realize that security-relevant code may need extra validation, or that they know but either do not or cannot perform this validation; these results suggest the latter rather than the former. In the following subsections, we explore reasons why developers do not or cannot perform this validation.

C. Insufficient security knowledge

Many participants in both the survey and interview suggested that their security knowledge is insufficient to perform security-relevant tasks like validating code from online forums and fixing identified bugs. Several participants suggested that further reading and education might help them secure their code.

All but one of the interviewees referenced the importance of security knowledge during the interview. Four of 15 explicitly said the security bug we identified originated because they didn't know enough about security to properly validate code they wrote, and eight said that lack of security knowledge would hinder them from properly integrating security-relevant code (including that from online forums). Nine participants said that in order to integrate security-relevant code correctly and efficiently, they would need educational resources, such as blogs and articles (n=4) or online forums (n=3). However, these resources themselves must be vetted for correctness: P5 wanted to find "Well-written articles to explain the problems, explain the pitfalls, explain mistakes people commonly make. And I would love to see an article written like that. . . if you do enough googling, you can find the answer to almost anything. You just have to get good at choosing which results to believe and which ones not to believe."

Further, six interviewees said they would need to learn more about security before they could avoid future problems similar to the vulnerability we identified. When asked whether and how they would fix the code in question, three participants said they did not know how to fix it but claimed they would learn about the problem and implement a fix. To explain how he would fix the bug, P1 said, "The first thing to check is what is the secure way to do this kind of password encryption in Java right now." Interestingly, P1 did eventually fix their pseudorandom-number-generator problem (Table IV).

As mentioned in Section VI-B above, 36 survey respondents described specific strategies for validating security-relevant code found in online forums. Of these, 13 mentioned learning from information resources, including online forums, blogs, and articles (n=9) as well as more official sources such as security-industry organizations or official documentation for security libraries (n=4). As examples of the latter, one respondent specifically mentioned the OWASP Top Ten list[3] and another mentioned documentation a Java EE security tutorial[4].

D. Developers trust their security skills

In contrast to participants who claim they do not know enough to implement security correctly, other participants do trust their own secure-development skills to protect their code.

When asked how they evaluate security aspects of code they find in online sources, many respondents indicated that they can evaluate security code sufficiently themselves. Of the 36 respondents who gave specific validation strategies (see

[3] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[4] https://docs.oracle.com/javaee/6/tutorial/doc/bnbwj.html

Section VI-B), 19 said they would try to understand how the code works, seven said they would write tests or otherwise try to break the code, and four reasoned that simpler code would inherently be more secure than complex code. Two respondents said they simply will not reuse code from online sources if it deals with user data.

We see a similar theme with our interviewees: seven said they would try to prevent future vulnerabilities (like the ones we identified) by trying to understand code drawn from online forums. Eleven participants reported that they knew enough to fix the vulnerabilities we identified and provided specific mechanisms for doing so. (Notably, this was after we had pointed out and explained the vulnerabilities, many of which are simple to fix once identified.) For instance, P11 identified multiple paths to fixing the issue: "I would probably use JWT, Java Web Tokens. So I might hide the secret with a public key, maybe even PGP, because this is like a message. Not just fix one line or two, but to rewrite all the code."

Two interview participants said that in order to easily integrate security-relevant code into their projects, they would rather write their own code instead of using existing libraries/frameworks. P6 noted, "It's not hard to rewrite the code if the encryption and security aspects is more important than efficiency, performance. Then writing the function or class by yourself, means more safety." This contradicts conventional wisdom that reusing validated libraries is more secure than "rolling your own" security software [26].

These responses suggest that in many cases, developers are confident enough in their own skills that they use (or reuse) code without referring to external resources or validating it more formally. The propagation of insecure code from online forums into GitHub and other production code (as identified by our work and in prior work such as [5]) suggests that sometimes this confidence may be misplaced.

Trusting their own skills, or wanting to learn more? We note that claiming insufficient security knowledge (Section VI-D) and trusting one's own security knowledge to be sufficient (this subsection) seem at least somewhat contradictory. In the survey data, these two populations seem mostly disjoint; five participants report validation strategies that fall into both categories.

In the interview data, however, a majority participants sometimes indicate that they trust their security skills but also suggest that they need to learn more. We suspect that the in-depth interview (in which security flaws are pointed out) allows participants to reflect more on their processes and recognize areas where they may want to learn more.

E. Security isn't the top priority

As previous work has shown, some developers prioritize other features of software — such as adding functionality or ensuring computational efficiency — over security [27]. We found evidence of this trend as well.

Twelve interview participants said they had consulted online forums when writing the project we identified as vulnerable. When asked if they considered security aspects when deciding to use code from these forums, four of the 12 said yes; the other eight reported that they did not consider security at all.

Two participants who said they did not consider security explicitly said that they just needed the code to work. When asked how or why a security problem occurred in their project, eight indicated that functionality was the highest priority, or that security was "an afterthought" (P4). According to P12, "This was an acceptable solution. I did not search again and again to find the best solution or to find the weakness in my code. I grabbed it from some forum and oracle developers or something like that. Just take, use, and go on."

Two participants explicitly cited efficiency concerns. For example, P15, who used a hard-coded salt when generating a key from a password, said that "the hard-coded thing probably is because it took less time for me to encrypt and decrypt."

Other participants (n=5) suggested that prioritizing functionality would inhibit the correct integration of security code in general, not just in the project we identified as potentially vulnerable. As P6 said, "I usually use the cipher method that looks simple: encryption stuff comes with it. Use the easiest way to finish the crypto task. None of my projects are focusing on the security aspects. I didn't see any point to get into more details about encryption stuff."

F. Security should be someone else's job

We also found evidence that some developers would rather not deal with security-relevant programming, by themselves or at all, and that some consider security not to be a core aspect of software development but rather someone else's responsibility. This result accords with prior work in software engineering [28],as well as research into end users' security decision-making [29].

Five interview participants said they would need security-oriented code reviews to avoid similar security bugs (to the ones we pointed out) in the future. P1 said, "When you use the code into your project, have some[one] to check or give it to some security department in the company to verify a little bit." Two of these five specifically stated they would refer to mentors for security-oriented code reviews. A sixth participant cited lack of security-oriented code review as a factor that might prevent developers from integrating security-relevant code correctly in general.

Two participants said that to prevent future security problems, they would want to abstract security-related code using methods "trusted by the community" (P4, P10). Two others said they would prefer to completely outsource security-relevant code to other developers. P11 said he wanted "Someone else in a service to do it for me, like some other company, to offload problems to someone else. I would probably use some service like Firebase from Google, they have all the authentication service."

Two participants wished for automated code analysis tools to help with integrating security-relevant code: "Being able to pop this code into whatever editor it was I was using, if the editor had the ability to highlight potential vulnerabilities I should focus in on, . . . it would be pretty handy" (P13).

In all of these responses we see that participants are trying to offload the responsibility of writing secure code to some degree. This is either done by trying to use "community trusted methods," looking for security oriented code reviews, outsourcing security code completely, or using code analysis tools. From these responses, we hypothesize that believing that security is or should be someone else's responsibility may align well with a willingness to copy & paste security-relevant code from an online forum with limited validation.

G. Security is not important in my context

A large minority of our interviewees (n=7) reported that the potentially vulnerable code we identified was not of concern in their project. Four said the vulnerability was important, and three were unsure. P10 argued that the relevance of any security issue depends on "the scope of the platform, and the scope of the project," but refused to answer whether the particular issue we identified was important in his project.

Four said the potential vulnerability was unimportant because the code in question is not used by many people. For example, P3 said, "It's not used by a lot of audience, so it's not very important. But it's important in a broader sense." Two interviewees (P1,P4) argued that because the code was used only in internal, offline tasks, security was irrelevant.

P4 and P11 claimed that the code in question was not written for a security-sensitive purpose. P11, for example, was developing a game cheat program, in which gamers pay periodic fees to enable the functionality. What we identified as potentially vulnerable code; is code that used encryption to create unique identifiers for each subscriber, in order to detect whether a single subscription was being used by multiple people or devices. P11 argued that this unique identifier use does not have important security implications: "It's not for password encryption or related. It's just used as an identifier. Even if you crack it, you really don't do anything. It's just unique information. Just to warn me ... if someone is sharing their subscriptions."

We note that all participants used cryptographic code in their potentially vulnerable projects, suggesting that on some level they believed there was some security-relevant purpose. (For example, P11 could have used a non-encryption-related unique identifier if he was not concerned about cheating.) Denying the relevance of security to their project may arise from participants' general defensiveness when asked about the problems we identified, a kind of social desirability bias.

VII. Discussion

In this section we discuss some of our descriptive findings and propose different methods of mitigating insecure code propagation via online forums such as StackOverflow.

We find that by and large, developers are aware that there are risks associated with importing code (in general, and security-specific) from online forums. Our results demonstrate that many participants recognize that they "should" perform validation via code reviews and tests, checking with external resources (including community feedback and "official" sources), and other strategies.

However, the prevalence of vulnerable code propagation [5] — in addition to the problems we identified in participants' own code — indicate that there is a critical gap between participants recognizing the importance of validation and implementing it. It is straightforward, and not unusual, to blame developers for prioritizing convenience or functionality over ensuring security, and indeed we find substantial evidence of such prioritization. We also, however, find other potentially important explanations, including participants who feel ill-equipped to properly conduct such validation and participants who believe, rightly or wrongly, that security should or will be handled by others. These differing motivations for propagating insecure code inform our recommended mitigations.

A. Security-oriented feedback system

In response to participants who do not have confidence in their ability to vet security-relevant code, as well as participants who rely on third parties and community feedback to help them, we suggest extending online forums to explicitly encourage community feedback on the security aspects of provided code snippets. (This idea was also suggested in [3].)

For Stack Overflow, this could mean extending the already-in-place flagging system [30] or duplicate-marking system [31] to allow flagging or marking for insecure code. Further, Stack Overflow could provide a commenting option specific to security-relevant concerns, perhaps highlighted with coloring or placement on the page. These feedback features might help developers who want to be skeptical of security-relevant code to make better choices.

Of course, any such community-feedback system can only be as useful as the expertise of its contributors. To this end, Stack Overflow could extend its reputation system to include security reputation, limiting certain community-feedback functions to those users who have previously demonstrated security expertise in other threads.

Such a system also has potential to influence developers who do not prioritize security or do not believe security is their responsibility. The availability of quasi-expert feedback addresses some developers' desire to outsource security decision-making. Further, forum administrators could use security feedback system as a factor in search results or recommendation systems, which would reduce the likelihood of developers who prioritize convenience encountering insecure results.

B. Linking to educational material

Another potential response to developers who expressed a desire to learn more about security in order to properly evaluate code they encounter is to provide them with educational resources. In particular, we suggest providing an explicit mechanism to link from threads on Stack Overflow and similar forums to related educational materials. The sources for these materials could be official documentation as well as white-listed experts and publications.

Users of online forums could be incentivized to link this material by allowing such linking (and subsequent approval by the community) to contribute to the user's reputation. Alternatively, researchers have explored automated mechanisms for linking forum posts and other documentation sources [32], [33]. Further, advancements in post deduplication would allow both community security feedback and links to educational resources to propagate among posts covering similar topics [34]–[36].

C. Removing problematic posts

For those developers who do not prioritize security, who do not believe security matters in their context, or who believe that security is or should be someone else's job, these mitigations may not be sufficient. Such developers may be likely to ignore warnings and links to educational materials in favor of effortlessly importing code that superficially solves a problem. To mitigate this, the problematic code snippets will have to be either made less visible to search algorithms or even removed altogether.

This goal could potentially be achieved via ranking posts using the community security-feedback features described above. If this approach proves insufficient, it might also be possible for carefully selected parties — for example, the U.S. Computer Emergency Readiness Team (US-CERT) — to explicitly request takedowns of problematic code snippets. Safeguards to prevent abuse would be required, and perhaps removed posts could be archived in a fashion that allows them to be accessed, but with sufficient inconvenience to deter developers searching for a quick fix. This might require a large resource investment by the trusted organizations, but even an effort to remove a few of the worst offenders could provide dividends.

VIII. Conclusion

Previous work has shown insecure code propagates from online programming forums to production code. In this study, we explore the reasons behind insecure code propagation from Stack Overflow to open-source GitHub repositories. In particular, we identified (from prior work) specific cryptography-related security vulnerabilities and searched for Stack Overflow posts that instantiate these vulnerabilities as code snippets. Using MOSS, we matched these snippets to code in GitHub repositories. We surveyed 133 authors of these repositories and conducted follow-up interviews with 15 of them. We find that while developers can articulate a variety of strategies for properly vetting security-relevant code they encounter in online forums, this vetting fails in practice for a variety of reasons, including choosing not to prioritize security, believing security is or should not be their job, and having insufficient knowledge or skill to evaluate specific code.

IX. Acknowledgments

The authors wish to thank Arunesh Mathur, Angel Plane, Heba Aly, and Ahmed Taha for their help with early versions of this work. We also thank all our participants, our anonymous reviewers, and our shepherd, Shriram Krishnamurthi. This work was supported in part by the U.S. Department of Commerce, National Institute for Standards and Technology, under Cooperative Agreement 70NANB15H330.

References

[1] OWASP, *OWASP Top Ten Project*, 2017. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[2] M. Corporation, *Common Vulnerabilities and Exposures*, 2019. [Online]. Available: https://cve.mitre.org/

[3] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 289–305.

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

[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 *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 121–136.

[6] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: Why do java developers struggle with cryptography apis?" in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 935–946.

[7] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, "Why do developers get password storage wrong?: A qualitative usability study," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 311–328. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134082

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

[9] Y. Wu, S. Wang, C.-P. Bezemer, and K. Inoue, "How do developers utilize source code from stack overflow?" *Empirical Software Engineering*, pp. 1–37, 2018.

[10] 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)*. IEEE, 2017, pp. 280–290.

[11] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?: Nier track," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 804–807.

[12] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 25–34.

[13] J. Yang, C. Hauff, A. Bozzon, and G.-J. Houben, "Asking the right question in collaborative q&a systems," in *Proceedings of the 25th ACM conference on Hypertext and social media*. ACM, 2014, pp. 179–189.

[14] D. Yang, A. Hussain, and C. V. Lopes, "From query to usable code: an analysis of stack overflow code snippets," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 391–402.

[15] H. Zhang, S. Wang, T.-H. P. Chen, Y. Zou, and A. E. Hassan, "An empirical study of obsolete answers on stack overflow," *IEEE Transactions on Software Engineering*, 2019.

[16] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in stackoverflow," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1019–1024.

[17] 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)*. IEEE, 2017, pp. 283–293.

[18] S. Baltes, R. Kiefer, and S. Diehl, "Attribution required: Stack overflow code snippets in github projects," in *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017, pp. 161–163.

[19] O. Docs, *Random (Java Platform SE 8)*, 2018. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

[20] N. I. of Standards and T. (NIST), *FIPS 140-2, Security Requirements for Cryptographic Modules*, 2002. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

[21] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516693

[22] 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

[23] S. University, *MOSS: A System for Detecting Software Similarity*, 2017. [Online]. Available: https://theory.stanford.edu/~aiken/moss/

[24] J. Landis and G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[25] StackExchange. (2019) All sites - stack exchange. [Online]. Available: https://stackexchange.com/sites?view=list#users

[26] B. Schneier, *Amateurs Produce Amateur Cryptography*, 2015. [Online]. Available: https://www.schneier.com/blog/archives/2015/05/amateurs_produc.html

[27] 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, 2014.

[28] H. Mouratidis, P. Giorgini, and G. Manson, "When security meets software engineering: a case of modelling secure information systems," *Information Systems*, vol. 30, no. 8, pp. 609–629, 2005.

[29] E. M. Redmiles, S. Kross, and M. L. Mazurek, "How i learned to be secure: a census-representative survey of security advice sources and behavior," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 666–677.

[30] StackExchange. (2019) Privileges flag posts. [Online]. Available: https://stackoverflow.com/help/privileges/flag-posts

[31] StackExchange. (2019) Why are some questions marked as duplicate. [Online]. Available: https://stackoverflow.com/help/duplicates

[32] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 392–403.

[33] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 643–652.

[34] 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*. ACM, 2016, pp. 402–412.

[35] 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*. International World Wide Web Conferences Steering Committee, 2017, pp. 1221–1229.

[36] Y. Zhang, D. Lo, X. Xia, and J.-L. Sun, "Multi-factor duplicate question detection in stack overflow," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 981–997, 2015.

APPENDIX

THE SURVEY

1) I am age 18 or older, and I have read this consent form and agree to participate in the survey.

○ Yes
○ No

2) How many years of software development experience do you have? *numeric free response*

3) How many years did you study computer science (or a related discipline) in school? *numeric free response*

4) In the past year, have you been employed as a professional software developer?

○ Yes
○ No

[*If **Yes** for question 4*]
How often do you engage in the following activities in professional software development?

a) Writing test cases for your code?
Never - Rarely - Sometimes - Often - Always - N/A
b) Collaborate with others for writing code?
Never - Rarely - Sometimes - Often - Always - N/A
c) Engage in code reviews with others?
Never - Rarely - Sometimes - Often - Always - N/A
d) Refer to code snippets from online forums or StackOverflow?
Never - Rarely - Sometimes - Often - Always - N/A

5) In the past year, have you contributed code to open source software?

○ Yes
○ No

[*If **Yes** for question 5*]
How often do you engage in the following activities in open source software development?

a) Writing test cases for your code?
Never - Rarely - Sometimes - Often - Always - N/A
b) Collaborate with others for writing code?
Never - Rarely - Sometimes - Often - Always - N/A
c) Engage in code reviews with others?
Never - Rarely - Sometimes - Often - Always - N/A
d) Refer to code snippets from online forums or StackOverflow?
Never - Rarely - Sometimes - Often - Always - N/A

6) In your own words, please explain how you evaluate the quality of code from online code snippets. How do you decide whether to accept or reject the code?
free response

7) Have you ever asked questions on online forums such as StackOverflow or others?

○ Yes
○ No
○ I don't remember

8) Have you ever answered questions on online forums such as StackOverflow or others?

○ Yes
○ No
○ I don't remember

9) Check all the statements that describe your background in computer security?

○ I have never take never taken any courses in computer

security
○ I have taken a course that did not focus on computer security, but included computer security as a module
○ I have taken a basic course in computer security
○ I have taken an advanced course in computer security

10) How would you rate your background in computer security?
○ Not at all knowledgeable
○ Slightly knowledgeable
○ Somewhat knowledgeable
○ Very knowledgeable
○ Extremely knowledgeable

[*Asked If* ***Yes*** *for question 5*]
11) In your open source software development routine, how often do you implement and handle tasks that relate to computer security?
Never - Rarely - Sometimes - Often - Always - N/A

[*Asked if participant does refer to online code snippets*]
12) How often do you refer to code snippets relating to computer security on online forums such as StackOverflow or others?
Never - Rarely - Sometimes - Often - Always - N/A

[*Asked if participant does refer to online code snippets relating to computer security*]
13) When you refer to code from online sources in your open source development, how do you evaluate the security aspects of the code?
free response

[*Asked if participant has asked questions on stack overflow before*]
14) Have you ever asked questions relating to computer security on online forums such as StackOverflow or others?
○ Yes
○ No
○ I don't remember

[*Asked if participant has answered questions on stack overflow before*]
15) Have you ever answered questions relating to computer security on online forums such as StackOverflow or others?
○ Yes
○ No
○ I don't remember

16) Have you previously used any software verification or static analysis tools (e.g. FindBugs, Pylint)?
○ Yes
○ No
○ I don't remember
○ I don't know what these are

These questions are specific to your project listed on Github [link to the project]

17) What platform was the project designed on? Please also list all the tools and software you used.
free response
18) How many other people excluding you worked on the project?
○ I worked on the project myself
○ 1 - 5 people
○ 6 - 10 people
○ 11 - 15 people
○ > 15 people

19) Did you seek help from online forums when working on this project?
○ Yes
○ No
○ I don't remember

20) Did you seek help from other collaborators when working on this project?
○ Yes
○ No
○ I don't remember

21) Did you have a deadline for completing this project?
○ I had no deadline to complete this project
○ I had a self-imposed deadline to complete this project
○ Someone else imposed a deadline to complete this project

[*Asked if participant had a deadline*]
22) How difficult was it for you to meet the aforementioned deadline for this project?
○ Not at all difficult
○ Slightly difficult
○ Somewhat difficult
○ Very difficult
○ Extremely difficult

[*Asked if participant had a deadline*]
23) How pressured did you feel to complete the project by by the deadline?
○ Not at all pressured
○ Slightly pressured
○ Somewhat pressured
○ Very pressured
○ Extremely pressured

24) In which year were you born?
numeric free response
25) What is the highest level of education that you have completed?
    - ○ 12th grade or less
    - ○ Graduated high school or equivalent
    - ○ Some college, no degree
    - ○ Associate degree
    - ○ Bachelor's degree
    - ○ Post-graduate degree

26) What is your primary occupation?
free response

27) What is your gender?
    - ○ Make
    - ○ Female
    - ○ Other

28) Would you like to participate in a 20-30 mins follow-up interview about your project? If you are selected, you will be compensated with a $15 Amazon Gift Card for your time.
    - ○ Yes
    - ○ No

[*if yes to question 28*]

29) Are you comfortable holding an online interview in the English language?
    - ○ Yes
    - ○ No

[*if yes to question 28*]

30) If not, which language would you feel most comfortable with? Please note we cannot guarantee that we will find a translator for the language you specify.
free response

[*if yes to question 28*]

31) Please enter your preferred email address, chat handle, or other ways to contact you in the box below. We will only use your contact details to set up the interview and will not contact you for any other reason.
free response

THE INTERVIEW PROTOCOL:

1) What is the goal of the project? What is its purpose? Who was it designed for? Was it supposed to work with end users or other entities?
2) Did you work by yourself on this code? Were you part of a team working on a broader project or was this a solo effort?
3) Did you use any help from online forums or other collaborators to write this code?
4) Did you have a tight deadline while implementing this project?
5) Did you get any monetary compensation while writing this project?
6) What is the functionality of this code snippet in your code? How important is the code to the overall functioning of the project?

The interviewer will explain to the interviewee the vulnerability of this code snippet and then ask the following:

7) Has anyone commented about security aspects of this code snippet before?
8) Do you think the vulnerability we pointed out is important? Why/why not?
9) How/why do you think this security problem occurred?
    - If you used online forums, did you consider security aspects when deciding to use the code?
10) How would you go about fixing this piece of code?
11) Given your existing knowledge, what (if any) measures would you consider next time you are implementing a security or cryptography task to avoid this kind of problem?
12) What would help you easily integrate security-related code into your tasks correctly and efficiently?
13) What would get in the way of integrating security-related code correctly?
14) Is there anything else we didn't discuss that's important for considering the risks and consequences of online security code usage?
15) Any other comments or information you would like to tell us?