Journal of Information Processing Vol.27 603–612 (Sep. 2019)

[DOI: 10.2197/ipsjjip.27.603]

Recommended Paper

Chronological Analysis of Source Code Reuse Impact on Android Application Security

Hironori Imai[1,a)] Akira Kanaoka[1,b)]

Received: December 10, 2018, Accepted: June 11, 2019

Abstract: Application developers consider open discussion forum on software development such as question and answer (Q&A) forums to be very important. There are cases where snippets which are partial source code on such forums contains vulnerabilities, and application developers divert snippets without knowing it. Previous works were focused on security-related codes such as a TLS connection, and not on actual vulnerable codes that are used widely. Thus, a time series investigation on the spread of such codes has not been conducted. In this paper, a method that enables the chronological analysis of source code reuse is proposed. By determining source code reuse in applications, we can investigate the context using time information such as the respective publication dates and time, and clarify how many cases are not source code reuse. An evaluation of the proposed method is achieved using large-scale data which includes 527,249 snippets of source code and 249,987 applications. The result shows that the appearance rate of applications having the same code as the snippet has increased after the release of the snippet. Furthermore, experiments on extracting vulnerable snippets from all snippets show that vulnerable snippets often have a greater impact than the overall snippet trend.

Keywords: Android, security, code clone

1. Introduction

The latest smartphones have a lot of sensors such as Camera, LTE, WiFi, NFC, GPS, compass, Bluetooth Low Energy module, barometer, three-axis gyro, accelerometer, proximity sensor, ambient light sensor, microphone, iris sensor, fingerprint sensor, heart rate sensor, SpO2 sensor, color spectrum, etc. And the improvement of living with them is remarkable.

The improvement in the quality of life can be attributed to not only high-spec smartphones but also the applications that can effectively handle these specifications. Application developers make use of various new sensors and OS functions to provide innovative applications.

Security and privacy have become problematic with applications. The combination of smartphones equipped with numerous multifunction sensors and applications that enrich our lives has led to the continuous accumulation of sensitive, personal information of users. Thus appropriate security and privacy in applications have become very important. As the possibilities of applications have expanded, so too have issues associated with security and privacy.

Developers who must make use of new sensors are striving to acquire new knowledge. They also have to update their knowledge of security and privacy further, which is becoming more difficult. Recent studies have revealed that poor development of security and privacy causes serious risk [1], [2], [3]. In such circumstances, occupying an important position among developers is an open forum on software development, i.e., a question and answer (Q&A) forum. StackOverflow is a representative service of such Q&A forum.

There are cases where a snippet, which is partial source code, described in StackOverflow contains vulnerabilities, and software developers divert snippets without knowing it. Acer et al. performed experiments on the occurrence of such incidents and the developer's attitude and highlighted the threat [4]. Fischer et al. further focused on the snippets of a specific security function called security-related code, and proposed a detection method [5]. They showed that snippets have been diverted to applications, and such snippets often contain insecure code in security-related parts.

Findings by Acer et al. have shed light on new threats. It is surprising that Fischer et al. found out how much insecure code exists and how much it is actually reused in the real world. However, the target was focused on security-related codes such as a TLS connection, not on actual and widely-used vulnerable codes. Thus, they have not investigated the spread from chronological viewpoint. Therefore, in this paper, we raise the following research questions.

Research Questions

- Are applications that are identified as code reuse really applications that the developer reused snippets?
- Among code reuse cases, does reusedvulnerable code have any special features?

[1] Toho University, Funabashi, Chiba 274–8510, Japan
[a)] 6518002i@st.toho-u.ac.jp
[b)] akira.kanaoka@is.sci.toho-u.ac.jp

The preliminary version of this paper was published at Multimedia, Distributed, Cooperative, and Mobile Symposium (DICOMO 2018), July 2018. The paper was recommended to be submitted to Journal of Information Processing (JIP) by the chief examiner of SIGCSEC.

© 2019 Information Processing Society of Japan

In order to respond to these research questions, a method that enables the chronological analysis of code reuse from snippets is proposed. In determining code reuse from snippets in applications, we can investigate the context using time information such as the respective publication dates and time, and clarify how many cases are not code reuse. Firstly, an inclusion detection method which detects snippet inclusion in an application is proposed. Secondly, an impact index of code reuse which can conduct chronological analysis is proposed.

The effect was evaluated experimentally using the proposed method. Snippets were collected from StackOverflow and applications were collected from the Androzoo dataset. Inclusion detection and chronological analysis were performed from those data, and their characteristics were evaluated. In the experiment, 527,249 snippets and 249,987 applications were used and analyzed. In addition, vulnerable snippets were extracted from the collected snippets and the impact of vulnerable snippets was analyzed. The result of the chronological analysis showed that the appearance rate of applications with the same code as the snippet increases after the release of the snippet. Experiments extracting vulnerable snippets from all snippets show that vulnerable snippets often affected more than the overall snippet trend.

Our contributions in this paper are as follows

- Proposea method that enables the chronological analysis of code reuse impact
- Understand theimpact of code reusein chronological order by analyzing with large-scale data
- Demonstrate that vulnerable snippets are more frequently reused than regular snippets

The structure of this paper is as follows. Related works are described in Section 2. An overview of the proposed chronological analysis architecture is explained in Section 3. The two main parts of the proposal, Inclusion Detection and Code Reuse Chronological Analysis, are explained in Sections 4 and 5, respectively. Section 6 describes the evaluation and the results. From the results, Section 7 discusses the limitations of the proposal. Finally, Section 8 concludes.

2. Related Works

2.1 StackOvewflow Impact

Acar et al. investigated the impact of information source on code security. They found that developers using Stack Overflow for looking up security-related issues often copy&paste insecure code [4].

In the study, they first investigated what information sources are used by developers when dealing with security and privacy related issues. The results of this study showed that a large number of developers (74.6%) use StackOverflow and search engines, while only a small number of developers consult Android's official documents (9.0%) and books (3.0%).

Secondly, they conducted a user study in which participants were requested to develop a security related code. In this study, the participants were divided into 4 groups. Restrictions were placed on each group with regard to the information that they could refer to while coding, namely StackOverflow, Android's official documents, books, and no reference. The results of this study showed that the group that used StackOverflow tend to produce functional but insecure code, while the group that referred to Android's official documents tend to produce unfunctional but secure code. Thus, among the 139 snippets (sample codes) used by the StackOverflow group, 25% were functional code and 17% were secure code.

From the above experimental results, the following inferences can be made.

- Official API documentation is not easy to use, but it is safe. Informal documents such as StackOverflow are easy to use in many cases, but they include vulnerabilities.
- Although paid books are practical and safe, less number of developers prefer using books.
- Given the time and economic constraints, it is expected that developers will use informal documents.

Fischer et al. [5] focused on the source code of a specific security function called Security-related code and proposed a detection method that detects used source code parts from Android applications. In addition, they proposed a classification method based on Expert's viewpoint for classifying the detected code into secure/insecure, and further classified it by using machine learning. Furthermore, they applied PPA (Partial Program Analysis) to evaluate the matching level between snippets and codes. Finally, copied&pasted codes are identified using Program Dependency Graphs and Jaccard Similarity. The result also shows that the snippets used for Android applications often contain insecure codes.

These studies have been developed as a new field as usability of cryptographic functions in recent years [6], [7], [8].

2.2 Clone Detection for Java Applications

Clone detection for applications is a field that has been studied for a long time. It depends on the target language and application.

Keivanloo et al. proposed a clone detection model SeByte focused onJava bytecode [9]. In SeByte, the features of a Java code are defined as fingerprints, and clone detection is performed by extracting and comparing the fingerprints from applications. In this work, three types of fingerprint dimensions are defined, as follows.

- Java Type Fingerprints
  - Sequential class names on method calls in a bytecode
- Method Call Fingerprints
  - Sequential method names on method calls in a bytecode
- Instructions Fingerprints
  - Sequential instruction names of Java Virtual Machine Instruction Set

Fingerprint elements are extracted from the three dimensions of fingerprints for each method in a bytecode. Then the elements from each method are combined into one. We call this FD (Fingerprint Data) for an unification of terms. FD matching is performed by comparing the elements of each dimension.

A comparison is performed using Jaccard similarity and each dimension has a threshold value for Jaccard similarity score to judge whether it is matched or not. In the original work, the threshold value of Java Type Fingerprints dimension was 0.16 and that of Method Call Fingerprints dimension was 0.53.

© 2019 Information Processing Society of Japan

By collecting the matching result for each method, SeByte searches and detects the clone.

Their studies show a high accuracy of clone detection. On the other hand, the accuracy of clone detection in obfuscated applications is still unclear, as there is no discussion of obfuscation which is currently applicable to many Java applications.

Clone detection is a field that has been studied for a long time, and survey articles are also substantial [10], [11].

3. Chronological Analysis Architecture

The study by Acar et al. [4] and the study by Fischer et al. [5] found the impact of StackOverflow on application developers from a security perspective. Their studies show the actual state of application developers who reuse snippets from StackOverflow. However, the chronological viewpoint was missing in their study. Since the chronological viewpoint is missing, there is a possibility that the detection of source code reuse was miscounted. If a snippet that is used to detect source code reuse from StackOverflow is a very general code, some applications will be misdetected as "code reused application" even if the applications were released before the snippet was published.

Therefore, a novel analysis method which enables chronological analysis for snippet reusing is proposed in this paper. In this section, an overview of the proposed method is provided.

We expand the clone detection technique of SeByte [9] for detecting of the inclusion of a snippet in an application. In the clone detection using SeByte, firstly Fingerprint Data (FD) is extracted for each method from bytecodes of an application. Then, clone detection is performed using Jaccard similarity based on the number of FD matchings. In the proposed method, improved and customized FD for an Android application is extracted from both snippets and Android applications. Then, the number of matches between FD is counted, and the inclusion detection is performed based on the number of matches.

Thus, the time information of both snippets and applications is obtained and used for analyzing the impact of snippet reuse, along with inclusion detection results.

Figure 1 shows an overview of the proposed analysis. The detailed method of inclusion detection is described in Section 4, and the detailed method of chronological analysis is described in Section 5.

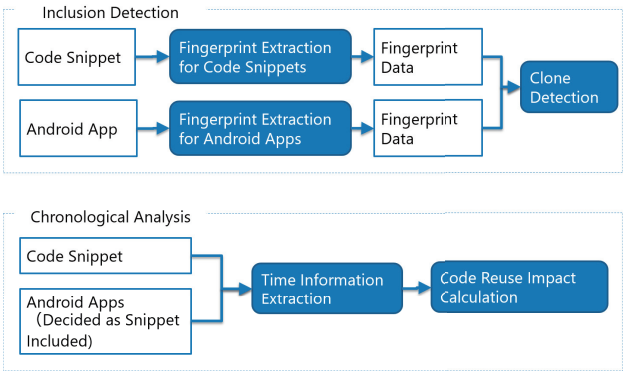

Fig. 1 Overview of Chronological Analysis Architecture. The detailed method of the inclusiondetection is described in Section 4, and the detailed method of chronologicalanalysis is described in Section 5.

4. Inclusion Detection from Code Snippet

Unlike clone detection which detects whether two applications are clones, it is necessary to compare snippets that are provided as source code and Android applications that are provided as execution code, in order to evaluate whether the snippet is reused in the application.

In consideration of such asymmetry, fingerprint extraction from snippets and fingerprint extraction from an Android application are proposed based on SeByte clone detection technique.

4.1 Fingerprint Extraction

From Dalvik bytecodes, Java Type Fingerprints, Method Call Fingerprints, and Arg Name Fingerprints are extracted as fingerprints. First two fingerprints are similar to SeByte. Arg Name Fingerprint is a novel fingerprint which can especially be extracted fromDalvik bytecodes. It is a sequential argument name, which is included in Dalvik bytecode but not in Java bytecode.

This extraction is common to Fingerprint extraction from both snippets and Android applications.

An example of a Dalvik byte code is shown as follows.

Example of Dalvik bytecode

```
# virtual methods
.method public add(JLjava/lang/String;)V
  .locals 3
  .param p1, "l"    # J
  .param p3, "s"    # Ljava/lang/String;

  .prologue
  .line 3
  sget-object v0, Ljava/lang/System;->
   out:Ljava/io/PrintStream;

  invoke-static {p3}, Ljava/lang/Long;->
   parseLong(Ljava/lang/String;)J

  move-result-wide v1

  add-long/2addr v1, p1

  invoke-virtual {v0, v1, v2},
   Ljava/io/PrintStream;->print(J)V

  .line 4
  return-void
.end method
```

The corresponding fingerprint is shown as follows.

© 2019 Information Processing Society of Japan

Example of Extracted Fingerprints

```
javaType: java.lang.Long,java.io.PrintStream
methodCall: parseLong,print
instruction: sget-object,invoke-static,
   move-result-wide,add-long/2addr,
   invoke-virtual,return-void
argName: l,s
```

4.2 Pretreatment for Snippets toward Fingerprint Extraction

To obtain fingerprints from snippets, a snippet is compiled into Dalvik bytecode, then fingerprints are extracted from the Dalvik bytecode. Since a snippet cannot be compiled by itself, preprocessing is required for snippets in order to extract fingerprints.

4.2.1 Removing XML Questions and Answers

In StackOverflow, snippets are tagged with '<code>' and '</code>' in their HTML source code. The snippets include not only a partial source code but also a preference or configuration file such as XML. Since such snippets are not required to extract fingerprints, non-Java snippets should be removed beforehand.

Regular expressions are used for XML detection. However, XML described in the Java source code is also detected. Therefore, regular expressions for detecting Java source code are also prepared. A snippet that is detected by XML regular expressions and not detected by Java source code regular expressions is detected as an XML snippet. Regular expressions for XML detection are as follows.

/android:\w+?=".+?"|<\/\S+?>|\/>/

Regular expressions for Java source code detection are as follows.

/\.\w+?\s*?\([^\)]*?\)\s*?;|import\s+?[\w\.]+?\s*?;/

4.2.2 Inserting Java Import Statements

Most of the snippets omit the import sentence; therefore, the error "cannot find symbol" occurs at compile time, if we compile the snippet directly. Therefore, the expected import statements are inserted in snippets to perform compiling successfully.

Since it is difficult to specify the required package from the snippet contents and errors, candidates of import statements are prepared in advance and are then inserted at once. The list of packages described in Android API packages [12] is used for inserting the packages.

From the list, 275 packages are used to insert, with the exception of 11 packages. Since Android SDK in API Level 26 does not include these packages and there are several packages which contain the same named class, the following 11 packages are not used.

- Packages that cannot be referenced from the jar file included in the Android SDK (API Level 26)
  - android.support.text.emoji
  - android.support.text.emoji.bundled
  - android.support.text.emoji.widget
  - android.support.v4.utils
  - android.support.v8.renderscript
  - android.support.wear
  - android.support.wear.widget
  - android.support.wear.widget.drawer
- The packages which contains same named class
  - java.net
  - java.security.cert
  - java.sql

4.2.3 Supplementing Class and Method Names

The minimum unit required for compiling is a single class. If a snippet is not formed as a single class, supplementation of the class name is necessary. Since the class name does not affect the result during fingerprint extraction, the declaration of a class with an appropriate fixed name is prepared and a snippet is put into the class. Therefore, if a snippet is just a short sentence that is not formed as a single method, the supplementation of the method is also necessary. In that case, the declaration of a method with an appropriate fixed name is also prepared. This method name does not affect the fingerprint extraction.

4.3 Inclusion Evaluation

In an inclusion evaluation using fingerprints, a similarity evaluation is performed for each dimension. Each similarity evaluation is performed using the Jaccard similarity coefficient like as in FD matching in SeByte.

$$J(s_1, s_2) = \frac{|s_1 \bigcap s_2|}{|s_1 \bigcup s_2|} \quad (1)$$

where s_1 is the fingerprint element of a method in a snippet and s_2 is the fingerprint element of a method in an application. The value of similarity $J(s_1, s_2)$ is obtained from the number of common fingerprint elements between a method in a snippet and a method in an application ($|s_1 \bigcap s_2|$) and the number of all fingerprint elements in both the method in a snippet and the method in an application excluding duplicates($|s_1 \bigcup s_2|$). The detection threshold is set to 0.16 for Java Type Fingerprints dimension and 0.53 for Method Call Fingerprints dimension, which are the same as in the study by Keivanloo et al. Since Arg Name Fingerprints is a novel fingerprint type dimension and the number of this type is low, the detection threshold of Arg Name Fingerprints is set to 1.0 for avoiding false negatives. If two or more of the three similarity evaluations exceed the threshold value, the snippet is determined to be included in the application.

5. Code Reuse Chronological Series Analysis

Inclusion detection allows us to find whether a snippet is included in an application. However,the actual impact of snippet reuse is not accurately measured just by investigating the inclusion of snippets. If the chronological viewpoint is missing, there is a possibility that the detection of code reuse was miscounted. Previous works had the same problem. In this section, a method that can perform chronological analysis of the impact of snippets on the Android market is proposed.

We assume that each snippet and each Android application has time information. In the case of snippets, it will refer to the time the snippet was published. When we consider the case of StackOverflow, it corresponds to the date and time the question and/or the answer was made. In the case of an Android application, it corresponds to the date and time when the application was published or added to the data set.

© 2019 Information Processing Society of Japan

Suppose you have performed an inclusion detection on an Android application data set using a snippet a published at time t. At that time, let N_a be the total number of applications judged as the snippet inclusion, and let $N_a(t)$ be the number of applications that have its time information after time t in N_a.

Let N be the total number of applications in the Android application whole dataset, and let $N(t)$ be the total number of applications that have its time information after time t.

At this time, the impact score $S_a(t)$ on the data set caused by publishing the snippet a after time t is defined as

$$S_a(t) = \frac{N_a(t)}{N_a} - \frac{N(t)}{N} \tag{2}$$

when the published time of the snippet a is t_a.

Here, some time information important for analysis using $S_a(t)$ will be listed. First, let t_a be the time when the snippet a was published. Next, let $t_{A_{a,f}}$ be the time when the first application $A_{a,f}$ determined to include the snippet a was published, and let $t_{A_{a,l}}$ be the time when the last application $A_{a,l}$ determined to include the snippet a was published.

Based on these time information, an analysis of various viewpoints can be achieved using $S_a(t)$. In this paper, we focus on t_a and analyze mainly on $S_a(t_a)$. The analysis of $S_a(t)$ based on other time information is discussed in Section 7.4.

Using this impact score S, we can discuss the transversal impact of a Q&A forum on the Android market. A Q&A forum's transversal impact T can be defined as

$$T = \frac{|\{a \in V | S_a(t_a) > 0\}| - |\{a \in V | S_a(t_a) < 0\}|}{|V|} \tag{3}$$

when V is a set of all snippets in the Q&A forum. T is obtained from a number of snippets which have a positive impact score and a number of snippets which have a negative impact score. Therefore it is a reflection of the positiveness or the negativeness of snippets. Using this score, we can see the transversal impact of snippets in Q&A Forum on Android markets.

Next, the impact ratio R_a on the data set caused by publishing the snippet a is defined as follows.

$$R_a = \frac{N_a(t_a)}{N_a} \cdot \frac{N}{N(t_a)} \tag{4}$$

If $R_a = 1$, it means that the snippet a has completely no impact on the Android application market. If $R_a < 1$, it means that the snippet a has a negative impact, that is, since its publication, the expression of that snippet ceased to be used. If $R_a > 1$, it means that the snippet a has a positive impact, that is, since its publication, the expression of that snippet came to be used.

Here, if $V(t)$ is the number of snippets published at time t, the peak impact of snippets on the Android market after t, $P(t)$ is expressed as follows.

$$P(t) = \frac{\sum_{a \in V(t)} R_a}{|V(t)|} \tag{5}$$

If R_a has a high absolute value, it means a snippet a has a large impact on the Android market. $P(t)$ reflects the peak impact of each snippet in a Q&A Forum on Android markets.

6. Evaluation

In order to verify the effectiveness of the proposed methods, we first evaluate the capability of inclusion detection in this section. After that, we collect actual snippets and Android applications, perform a chronological analysis on them, and analyze the impact of the snippets reuse on the market. In addition, we extract vulnerable snippets from information on the vulnerability of Android applications caused by the source code, analyze the impact of vulnerable snippets on the market, and compare it with the overall impact of all the snippets.

6.1 Inclusion Detection Capability

To measure the inclusion detection capability, we compare the extraction of FD from data sources that were used in the original SeByte work. The datasets used for the experimentation include EIRC [13] and FreeCol [14]. FreeCol(Server) indicates the software that has only server-side functions in FreeCol. Also, FreeCol(Full) indicates the entire software of FreeCol. Evaluation of SeByte was performed by actual bytecode comparison; however, the proposed inclusion detection,which is an improvement of the FD matching of SeByte, and cannot be directly compared. Therefore, we tried to generate FD by using the source code and Java application dataset, which SeByte used for bytecode comparison, and investigated whether it is possible to generate a similar degree of FD.

Table 1 shows the result. '# of FD' means the number of FD in each method and '# of Class' indicates the number of classes obtained. The proposed method and the original SeByte show asimilar trend when the proposed method generates a larger number of FDs than the original SeByte. From the result, it is seen that the proposed inclusion detection has a capability equivalent to that of the original FD of SeByte.

6.2 Collecting of Android Applications

As an Android application, 249,987 APK (Android Application Package) files were randomly acquired from the Androzoo data set [15].

When FD is extracted from all the classes included in an application, it includes many FDs extracted from third party packages added by the import statement in addition to the code created by the developer him/herself. These are not suitable for analysis. Therefore, we searched for packages that contain classes containing application startup activities from AndroidManifest.xml, and extracted FDs from classes corresponding to those packages. As a result, the number of Android applications that succeeded in acquiring FD was 245,717 (98.3%).

6.3 Collecting of Snippets

Snippets were obtained from StackOverflow, which is a popular Q&A forum. Questions and answers were gathered using Stack Exchange API, which can obtain StackOverflow information. Then, questions that were tagged as 'android' and were browsed by 1,000 or more users were selected, and the answers from these questions were collected. Finally, snippets were obtained from the gathered questions and answers.

© 2019 Information Processing Society of Japan

Table 1 Comparison between Inclusion Detection and FD Matching.

	# of FD		# of Class	
	Proposed	SeByte	Proposed	SeByte
EIRC	451	198	19	24
FreeCol	4766	708	48	46
FreeCol (full)	116138	1593	288	149

Table 2 Number of sample snippets targeted for experiment.

Category	downloaded entries	obtained snippets
Question	207,694	250,517
Answer	496,806	398,178

The number of questions, answers, and snippets is shown in **Table 2**.

Since snippets may contain vulnerabilities, snippets can be divided into four groups as follows:

- All the questions
- Questions that may contain vulnerabilities
- All the answers
- Answers that may contain vulnerabilities

In order to distinguish vulnerable snippets, vulnerabilities derived from the source code were listed up. The snippets were examined to find whether they contain these vulnerabilities and were classified into the above four categories. The vulnerabilities to be examined were obtained from AndroBugs [16]. The total number of snippets classified as vulnerable were 5,675 (in Questions) and 5,278 (in Answers). Detailed vulnerability information and number of obtained snippets are shown in **Table 3**.

Vulnerable snippets are extracted by performing string matching with vulnerability strings obtained from AndroBugs.

Apart from AndroBugs, there are other sites that disclose vulnerability data; however, none of them include source code data indicating vulnerabilities, so they are excluded from this survey.

- Ostorlab
- Quixxi
- NVISO
- SandDroid
- QARK
- Mobile App Scanner

Table 4 shows the number of snippets excluding snippets determined as XML, among the snippets acquired from StackOverflow. **Table 5** shows the number of snippets that succeeded in compiling and obtaining FD.

The main factors responsible for the failure of snippets that failed to compile are shown below.

- Non Java source code (fragment of the log)
- Undeclared variables and methods
- Incorrectly abbreviated form of Java syntax that can be understood by humans
- Comment statements that have not been commented out

6.4 Inclusion Detection Results

This subsection shows the results of inclusion detection in four categories using the successfully compiled snippets shown in Table 5 and 245,717 applications from Androzoo dataset.

In the all question category, 3,453 snippets out of 5,006 snippets (69.0%) are included in applications. In the all answer category, 11,065 snippets out of 15,214 snippets (72.7%) are included in applications. In the vulnerable question category, 168 snippets out of 183 snippets (91.8%), are included in applications. In the vulnerable answer category 563 snippets out of 632 snippets (89.1%) are included in applications. Compared with the overall trend, we found that vulnerable snippets have a higher inclusion rate. **Table 6** shows the result.

Table 3 Candidate vulnerabilities included in snippets.

Type of Vuln.	Question	Answer
SSL_CN2	1	3
SSL_X509	60	111
WEBVIEW_RCE	217	291
HACKER_PREVENT_SCREENSHOT_CHECK	627	574
COMMAND	1,928	747
SSL_CN3	0	2
SSL_DEFAULT_SCHEME_NAME	94	65
WEBVIEW_JS_ENABLED	430	289
DB_DEPRECATED_USE1	1	0
MODE_WORLD_READABLE _OR_MODE_WORLD_WRITEABLE	0	0
EXTERNAL_STORAGE	2,222	3,030
SENSITIVE_DEVICE_ID	95	166

Table 4 Classified snippets.

	All Snippets	Vulnerable Snippets
Question	202,076	5,675
Answer	325,173	5,278

Table 5 Successfully compiled snippets.

	All Snippets	Vulnerable Snippets
question	5,006 (2.5%)	183 (3.2%)
answer	15,214 (4.7%)	632 (12.0%)

6.5 Chronological Analysis Results

In this subsection, we show the results of the chronological analysis on the included snippets obtained in the previous subsection.

6.5.1 Transversal Impact

Figure 2 shows a plot of $S_a(t_a)$ obtained from the time information held by those snippets against the included detection result for all questions category, and indicates the impact score. If $S_a(t_a) > 0$, it means that the snippet has a positive impact on the Android market. **Figures 3**, **4**, **5** show the results for the categories of all answer, vulnerable question,and vulnerable answer, respectively. From the results, we can see that there are many positive impacts in every category.

Next, the T value of each category is analyzed. The impact score $S_a(t_a)$ was obtained for all snippets $a \in V$, then T was calculated therefrom for each category. **Table 7** shows T and related values in each category. We can see that the snippets have a positive impact on all categories. Thus, we can see vulnerable snippets have a T value higher than all snippets. It means vulnerable snippets have a transversal impact on the Android market in contrast to other snippets.

In Table 7, questions that may contain vulnerabilities (Vuln Qs and answers that may contain vulnerabilities (Vuln As) have less "# of Negatives" compared to all the questions (All Qs) and all the answers (All As). In All Qs, 759 snippets (21.95%) have negative $S_a(t_a)$ values among 3,453 snippets in total. On the other hand, in Vuln Qs, only 10 of the 168 (5.95%) snippets have negative $S_a(t_a)$ values. This shows that vulnerable snippets tend to have

© 2019 Information Processing Society of Japan

Table 6 Comparison result between StackOverflow snippets and Androzoo datasets.

	Vuln. Qs	Vuln. As	All Qs	All As
Examined Snippets	183	632	5,006	15,214
Included Snippets	168	563	3,453	11,065
Inclusion Rate	91.8%	89.1%	69.0%	72.4%

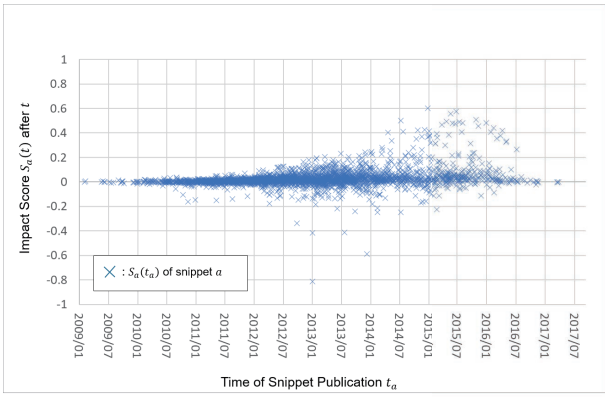

Fig. 2 Code reuse impact of snippets (Category: all question).

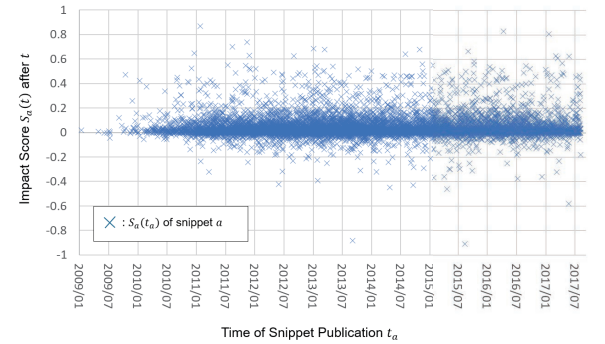

Fig. 3 Code reuse impact of snippets (Category: all answer).

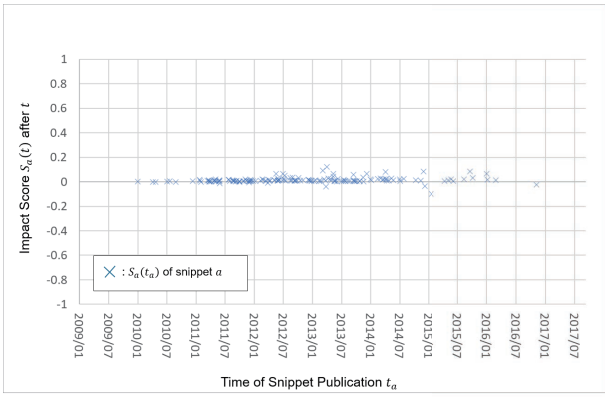

Fig. 4 Code reuse impact of snippets (Category: vulnerable question).

fewer "# of Negatives". The fact that $S_a(t_a)$ is a negative value indicates that app developers have widely used the snippet before it was firstly posted to StackOverflow. It would be a snippet about the more general or basic questions and answers.

On the other hand, vulnerable snippets are a group of vulnerabilities caused by the source code provided by AndroBugs as shown in Table 3. Looking back at these vulnerabilities, we can see that they are technically limited snippets. For example, in EXTERNAL_STORAGE, which is the most frequent, the use of the *getExternalStorageDirectory*method of the *android.os.Environment* class is a trigger to detect as a vulnerability.

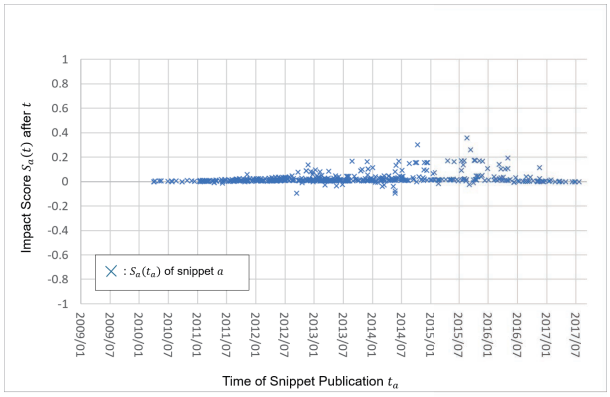

Fig. 5 Code reuse impact of snippets (Category: vulnerable answer).

Table 7 T values in each category.

	All Qs	All As	Vuln Qs	Vuln As
# of Positives	2694	8754	158	527
# of Negatives	759	2311	10	35
T value	0.560	0.582	0.881	0.875

Table 8 $P(t)$ values.

	Avg.	Median
All Qs	1.050	1.016
All As	1.135	1.033
Vuln Qs	1.022	1.012
Vuln As	1.082	1.017

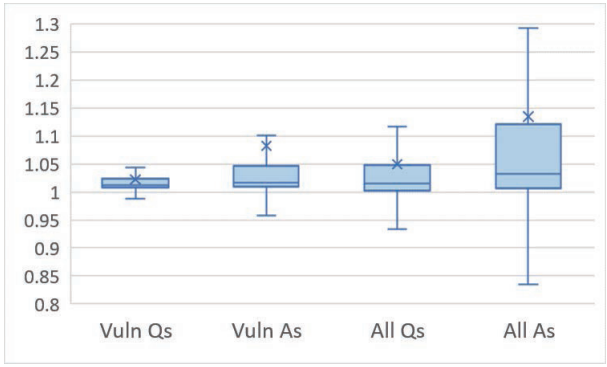

Fig. 6 Box plot of $P(t)$ value in each category.

In the next most frequent COMMAND,the use of the *exec* method of the *java.lang.Runtime* class is a trigger to detect as a vulnerability. These technically limited snippets appear when there is a need for a specific application. It is thought that the "# of Negatives" decreased as the snippet was not widely used when it first appeared on StackOverflow.

6.5.2 Peak Impact

Peak impact $P(t)$ of each category is also analyzed. R_a was obtained for all snippets and $P(t)$ was calculated for each category. **Table 8** shows the average and median of $P(t)$ in each category. We can see that the snippets also have a positive impact on all categories. The box plot of the result is shown in **Fig. 6**. From the result, we can see that answers tend to be reused, and vulnerable snippets tend to have a low peak $P(t)$. Even though the $P(t)$ value of vulnerable snippets is relatively low, its value is still over 1.

The rate of snippets that have $R_a > 1$ is 78.09% in all Qs category, 81.34% in all As category, 94.64% in vuln. Qs, and 94.28% in vuln. As, respectively. We can see most of the vulnerable snippets have a positive impact in contract to all the snippets.

© 2019 Information Processing Society of Japan

7. Limitation and Discussion

7.1 Increase Success Rate of Snippet Compilation

In the experiments conducted in this study, the rate of successfully compiled snippets was found to be low. As described in Section 6-C, the main factors for the failure of snippets that failed to compile are as follows.

- Non-Java source code (fragment of the log)
- Undeclared variables and methods
- Incorrectly abbreviated form of Java syntax that can be understood by humans
- Comment statements that have not been commented out

Here, we discuss the solutions to each of these four problems.

7.1.1 Non-Java Source Code

As already mentioned, snippets may contain data other than Java code. In the proposed method, XML is detected and removed by pretreatment, but data other than Java code and XML data exist. Logs form a significant portion of such data. There are various questions and answers about Android such as the meanings of error logs and access logs. There are many types of notation for logs; therefore, it is difficult to accurately detect and delete logs. Though snippets are extracted from questions with 'android' tag, there may be snippets in other languages. Recently, it has become possible to develop applications using languages other than Java. It is considered to be present in the background.

From these facts, it is considered that there is a limit to the method of considering various cases and eliminating them at the time of extraction. Conversely, it is possible to consider a method that can extract only Java code. However, if there are a few rows of snippets, it will be difficult.

7.1.2 Undeclared Variables and Methods

In the snippets, there are cases where only important parts of the problem are indicated, and the declaration of variables or instances used therein is not described. Compile errors are obtained when extracting the fingerprint elements, since the proposed method does not correspond to that. As with the import statement, it is possible to insert variables or instances that can be expected. Although these insertions affect the results of the fingerprint elements, post-treatment after obtaining the fingerprint elements is required in this case.

7.1.3 Incorrectly Abbreviated Form of Java Syntax That Can Be Understood by Humans

It is conceivable to comment out incorrect parts as Java syntax, but to do this, it is necessary to have a technique for detecting the abbreviated parts. However, proper detection is considered difficult.

7.1.4 Comment Statements That Have Not Been Commented Out

In order to comment out a comment part that has not been commented out, it is necessary to have a method for detecting a comment part, as with Section 7.1.3.

7.2 Adjustment of Type and Weight of Fingerprint Used for Comparison

In this experiment, the fingerprint used for comparison was determined based on the study by Keivanloo et al. Although Java bytecode and Dalvik bytecode have high similarity, there is also information peculiar to Dalvik bytecode such as Arg Name Fingerprint. Therefore, it is expected that finding another available feature as the fingerprint will increase the accuracy. In addition, the weighting dimension can also be expected to increase accuracy.

7.3 A Snippet That Exists in a Q&A Forum Other than StackOverflow

There are many Q&A forums for IT engineers other than StackOverflow. By analyzing sample snippets existing in these forums, more universal analysis results can be obtained. In addition, depending on the forum, there are cases where there are many users from different regions and languages, and by considering these, it is possible to perform analysis according to the country and region.

7.4 Further Investigation of Snippet Reuse Using $S_a(t)$

In this paper, the discussion about the transversal impact of snippets is almost based on time of a snippet publication t_a. However, we can discuss in depth about the infiltration of each snippet using this $S_a(t)$.

It will take a certain amount of time for a snippet to be widely used after being published. If we investigate $S_a(t)$ value for time $t > t_a$, we can discuss such delay or timing of a snippet expansion.

Naturally, we know from its definition that $S_a(t_a) = S_a(t_{A_{a,f}})$. By looking at the difference between t_a and $t_{A_{a,f}}$, we can discuss how long it is allowed to post a fix as an answer on StackOverflow before a vulnerable snippet is spread. We can also see the pattern of the epidemic by looking at $S_a(t)$ during $t_{A_{a,f}} < t < t_{A_{a,l}}$.

7.5 Code Obfuscation

At present, many applications are often obfuscated as a countermeasure against reverse engineering. The work of Wermke et al. is a large-scale survey of the current state of code obfuscation in Android [17]. Their survey shows that 24.9% of the main package is obfuscated. The study reveals the current situation where many apps are obfuscated.

If an app is obfuscated, our proposed inclusion detection may not work correctly. We believe that the trend in the results obtained in this paper does not change, as 75% apps on the Google Play market are not currently obfuscated. However, a more precise investigation is needed. The inclusion detection technique we proposed is a technology based on SeByte proposed by Keivanloo et al. [9]. In SeByte, the influence of obfuscation on the clone detection accuracy has not been discussed. We believe that both our inclusion detection method and the SeByte clone detection method should discuss the impact of the obfuscation in the future.

7.6 Other FD Extraction and Inclusion Detection Methods

In the proposed method, in order to extract an FD from a snippet, once the snippet is compiled and converted to bytecode, then the FD is acquired. However, the acquisition of FD or the inclusion detection can be considered other approaches. Here we introduce two other approaches and discuss their Pros and Cons.

© 2019 Information Processing Society of Japan

7.6.1 Direct FD Extration from Snippet

It is also conceivable to obtain the FD from the snippet directly instead of compiling the snippet once and converting it to bytecode, and then obtaining the FD. If a compiler is deterministic about generating bytecode for the part involved in FD extraction, it is possible to learn in advance the mapping of information between source code fragment and FD, and create a conversion tool from source code to FD instead of a compilation. However, at present, the discussion about whether there is such determinacy in the compiler of the Android development environment is not enough as far as the authors know. Therefore, this method was not adopted in this paper.

7.6.2 Comparing Using Source Codes via Decompiling

A method of executing inclusion detection using source codes is also conceivable. In this method, inclusion detection is performed after decompiling the application and obtaining the source code and compared the source code and snippets to detect inclusion.

This method makes a difference in performance. The difference is due to the performance of the "FD extraction from bytecode" and "decompile to get source code" tasks. While extracting FD from bytecode is a simple task of extracting three parts, the task of decompiling to get the source code is more complicated. Therefore, it takes much time, and there are few performance benefits. That point becomes particularly important when it comes to large-scale analysis. Therefore, this method was not adopted in this paper.

8. Conclusion

In this research, we proposed a method to investigate how the code snippets available in Q&A forums such as StackOverflow affect actual application developers, and compare actual snippets and Android applications. We collect 527,249 snippets and 249,987 applications for experiments. In addition, the impacts of actual snippets on Android applications are analyzed.

The proposed method is capable of chronological analysis, which could not be achieved by the previous methods. As a result of the chronological analysis, it is shown that the appearance rate of applications having the same code as the snippet has increased after the release of the snippet. Experiments on extracting vulnerable snippets from all snippets show that vulnerable snippets often have a transversally greater reuse impact than the overall snippet reuse trend.

References

[1] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D. and Shmatikov, V.: The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software, *Proc. 2012 ACM Conference on Computer and Communications Security*, *CCS '12*, pp.38–49, ACM (online), DOI: 10.1145/2382196.2382204 (2012).

[2] Egele, M., Brumley, D., Fratantonio, Y. and Kruegel, C.: An Empirical Study of Cryptographic Misuse in Android Applications, *Proc. 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS '13*, pp.73–84, ACM (online), DOI: 10.1145/2508859.2516693 (2013).

[3] Reaves, B., Scaife, N., Bates, A., Traynor, P. and Butler, K.R.: Mo(bile) Money, Mo(bile) Problems: Analysis of Branchless Banking Applications in the Developing World, *24th USENIX Security Symposium* (*USENIX Security 15*), pp.17–32, USENIX Association (2015) (online), available from ⟨https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/reaves⟩.

[4] Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M.L. and Stransky, C.: You Get Where You're Looking for: The Impact of Information Sources on Code Security, *2016 IEEE Symposium on Security and Privacy* (*SP*), pp.289–305 (online), DOI: 10.1109/SP.2016.25 (2016).

[5] Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M. and Fahl, S.: Stack Overflow Considered Harmful? The Impact of Copy Paste on Android Application Security, *2017 IEEE Symposium on Security and Privacy* (*SP*), pp.121–136 (online), DOI: 10.1109/SP.2017.31 (2017).

[6] Acar, Y., Stransky, C., Wermke, D., Mazurek, M.L. and Fahl, S.: Security Developer Studies with GitHub Users: Exploring a Convenience Sample, *Thirteenth Symposium on Usable Privacy and Security* (*SOUPS 2017*), pp.81–95, USENIX Association (2017) (online), available from ⟨https://www.usenix.org/conference/soups2017/technical-sessions/presentation/acar⟩.

[7] Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M.L. and Stransky, C.: Comparing the Usability of Cryptographic APIs, *2017 IEEE Symposium on Security and Privacy* (*SP*), pp.154–171 (online), DOI: 10.1109/SP.2017.52 (2017).

[8] Nguyen, D.C., Wermke, D., Acar, Y., Backes, M., Weir, C. and Fahl, S.: A Stitch in Time: Supporting Android Developers in WritingSecure Code, *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security*, *CCS '17*, pp.1065–1077, ACM (online), DOI: 10.1145/3133956.3133977 (2017).

[9] Keivanloo, I., Roy, C.K. and Rilling, J.: SeByte: A semantic clone detection tool for intermediate languages, *2012 20th IEEE International Conference on Program Comprehension* (*ICPC*), pp.247–249 (online), DOI: 10.1109/ICPC.2012.6240495 (2012).

[10] Roy, C.K. and Cordy, J.R.: A survey on software clone detection research, *Queen's School of Computing TR*, Vol.541, No.115, pp.64–68 (2007).

[11] Rattan, D., Bhatia, R. and Singh, M.: Software clone detection: A systematic review, *Information and Software Technology*, Vol.55, No.7, pp.1165–1199 (2013).

[12] Developers, A.: Package Index, available from ⟨https://developer.android.com/reference/packages.html⟩ (accessed 2017-08-27).

[13] Kohen, J.: Eteria IRC Client, available from ⟨http://eirc.sourceforge.net/⟩.

[14] FreeCol - the Colonization of America, available from ⟨http://www.freecol.org/index.html⟩.

[15] Allix, K., Bissyandé, T.F., Klein, J. and Traon, Y.L.: AndroZoo: Collecting Millions of Android Apps for the Research Community, *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories* (*MSR*), pp.468–471 (online), DOI: 10.1109/MSR.2016.056 (2016).

[16] Lin, Y.-C.: AndroBugs (Yu-Cheng Lin), available from ⟨https://github.com/AndroBugs⟩ (accessed 2018-03-15).

[17] Wermke, D., Huaman, N., Acar, Y., Reaves, B., Traynor, P. and Fahl, S.: A Large Scale Investigation of Obfuscation Use in Google Play, *Proc. 34th Annual Computer Security Applications Conference, ACSAC '18*, pp.222–235, ACM (online), DOI: 10.1145/3274694.3274726 (2018).

Editor's Recommendation

The paper successfully unravels the relationship between vulnerable snippets on online developer forums and Android applications. It sheds light on the chronological characteristic of frequently reused but vulnerable codes, and its evaluation technique to compare source code snippets and Android application binaries is practical and useful. The paper should give a new insight to readers in this research field, and thus was selected as a recommended paper.

(Chief examiner of SIGCSEC Masayuki Terada)

© 2019 Information Processing Society of Japan

Hironori Imai received his B.E degree from Toho University in 2018. He is currently a master course student at Graduate School of Science, Toho University. He received the Paper Award and the Presentation Award at the Multimedia, Distributed, Cooperative, and Mobile Symposium (DICOMO2018).

Akira Kanaoka received his Ph.D. degree in engineering from University of Tsukuba, Japan in 2004. He worked at SECOM Co., Ltd. from 2004 to 2007, and at University of Tsukuba from 2007 to 2013. He is currently an associate professor of Department of Information Science, Faculty of Science, Toho University. His research interests include usable security and privacy.

© 2019 Information Processing Society of Japan