User-Side Updating of Third-Party Libraries for Android Applications

Hiroki Ogawa
Nagoya Institute of Technology
Aichi, Japan
h.ogawa.334@nitech.jp

Eiji Takimoto
Ritsumeikan University
Siga, Japan
takimoto@asl.cs.ritsumei.ac.jp

Koichi Mouri
Ritsumeikan University
Siga, Japan
mouri@cs.ritsumei.ac.jp

Shoichi Saito
Nagoya Institute of Technology
Aichi, Japan
shoichi@nitech.ac.jp

***Abstract*—A Third-Party Library(TPL) is often used in developing Android applications, however older TPLs may have vulnerabilities. Hence developers need to keep them in their applications the latest version. Nevertheless, there is a lot of applications using older TPLs.**

In this paper, we propose a new method which users enable to update TPLs in Android applications. An Android application and TPLs can be converted to smali file which is more of an assembly based language. A smali file can be replaced with another smali file on the same class. Our method takes advantage of its properties and exchanges a vulnerable TPL for an security fixed one. Moreover, we apply it to real applications and evaluate feasibility of it.

***Index Terms*—Library update, Third Party Library, Android, Android security**

I. Introduction

In developing Android applications, Third-Party Libraries(TPL) are essential parts. Developers can improve development efficiency and speed using TPL. Purposes of TPLs are, for instance, displaying advertisement, implementing an authentication function of a social networking service, and enhancing convenience. In this way, TPL has many advantages.

On the other hand, TPLs can be a security threat. Some TPLs are vulnerable depending on versions. For instance, versions of a facebook's TPL older than 3.16 have account hijacking vulnerability [1]. In addition, Dropbox's TPL [2] and Apache CC's TPL [3] also have versions with vulnerabilities. There are a lot of TPL which have vulnerable versions. Using older versions of TPLs are dangerous, and developers should keep TPLs in applications up to date. However, there are applications without the latest TPL because developers forgot to update or completed development [4]. Meanwhile, users are not provided a method to update applications. Therefore, if users want to update them, they cannot do it. In this paper, we propose a method for them to update TPLs used by their applications.

Our method identifies TPLs in applications by cooperating with an external server and updates TPLs if applications are linked with older versions of TPLs. Herewith, users can update TPLs in applications and fix vulnerabilities included in older versions of TPLs.

An Android application includes some TPLs, and both an Android application and TPLs can be converted to smali files. A smali file is more of an assembly based language, and can be replaced with another smali file on the same class. Our method takes advantage of its properties and exchanges a vulnerable TPL for an security fixed one.

The rest of paper is organized as follows: Section 2 gives related works survey. Section 3 explains about the proposed method and details of it. Section 4 talks about implementation of it and Section 5 shows evaluation of it. And we discuss the future works in Section 6. The paper is summarized in Section 7.

II. Related works

This section describes related works. Firstly we describe techniques to identify TPLs used by applications and update applications. Lastly we discuss a problem of application updating methods.

A. Techniques of identifying TPL used by applications

Techniques to identify TPLs used by applications are mainly classified into a whitelist method [5] [6] [7], a machine learning method [8] [9] [10] and a clustering method [11] [12].

In the whitelist method, a TPL can be identified by its package name. This method has advantages of being fast and applying easily to any applications but cannot identify TPLs if their package's names are obfuscated.

In the machine learning method, TPLs for advertisement can be identified with high accuracy. However, this method is aimed at identifying TPLs for advertisement [13]. Thus, it is difficult for this method to recognize other types of TPLs.

In the clustering method, various TPLs are clustered to collect some information about API before identifying TPL used by applications and then can be identified based on them. This method can identify TPLs with high accuracy like the machine learning method. In addition, this method can identify TPLs other than TPLs for advertisement unlike the machine learning method [13]. However, it is difficult for this method to identify TPLs that are used by fewer applications because of little information.

There are two other methods except for the above. The first is LibScout [4] which extracts class hierarchy information from a TPL, stores it in the LibScout's database, and identifies TPLs in applications using it. The second is LibRadar [14] which is a method that extends the clustering method. It extracts Android API call frequency from a TPL and classifies TPLs based on the frequencies.

B. Techniques of updating application

There is an update method for Android applications called Hotfix [15]. Hotfix is a method for developers to quickly provide a bug fixed applications to user. Developers prepare scripts to fix bugs in the application, and then users download and execute it.

There is a method to check TPLs updatability for Android applications [16]. It uses two data sets to determine whether and to which extent TPLs in applications can be updated. They are for evaluating robustness and usage of library APIs. The robustness data set can evaluate how many APIs are remaining in the successor version. The usage data set can evaluate what parts of TPLs are used in applications. The paper [16] combines them and checks whether TPLs in applications can be replaced to another versions.

C. Problem about method of updating application

The existing methods have a problem that users cannot update their applications alone. Even if any vulnerabilities in TPLs or APIs are disclosed, users cannot update them. From a developers point of view, their burden is heavy because they are constantly concerned about the vulnerabilities of TPLs and need to update their applications quickly. Therefore, new method for users to update their applications without developers is necessary.

III. Proposal

This section describes overview of a method to update TPLs used by applications on Android devices by cooperating with an external server and details of it.

A. Overview of proposal

We propose a method to update TPLs used by applications on a user-side Android device. This proposed method enables users to update their applications by cooperating with an external server. The behavior of the proposed method is shown in Fig.1.

The updating steps is shown below.

(1) User's Android device sends an original apk file of the application an user wants to update to the external server.
(2) The external server analyzes the original apk file and identifies TPLs and their versions.
(3) The external server updates TPLs and creates an updated apk file.
(4) The external server informs the Android device that updating is completed.
(5) The external server sends the updated apk file to the Android device.
(6) The Android device uninstalls the original apk file.
(7) The Android device installs the updated one.

If updating is not necessary, our method executes just steps 1,2, and 4. This behavior is shown below.

(1) User's Android device sends an original apk file of the application an user wants to update to the external server.
(2) The external server analyzes the original apk file and identifies TPLs and their versions.
(4) The external server informs the Android device that updating is unnecessary.

B. Techniques of identifying TPL

The proposed method uses LibScout, which is described in Section II-A, to identify TPLs and their versions. There are three reasons for using LibScout and they are shown in below.

- Source code is published on github,
- LibScout can add TPL information to a database,
- LibScout can distinguish whether a TPL version could be identified or not.

C. Library update

Android applications are executed on Dalvik VM. Thus, a program of Android application is compiled into dex file which is bytecode of Dalvik VM. It is converted to smali file by disassembling. Smali file is more of assembly based language and more understandable form than dex file. TPLs can be converted to smali file as well. Smali file in an Android application can be replaced with another smali file on the same class. Therefore, the whole smali files of a TPL in an application can be replaced for another TPL which has the same classes. The proposed method updates the versions of TPL based on this character.

An overview of the proposed updating procedure is as follows. In advance the external server converts the latest version of TPLs to smali files and stores them in the database. When a user updates TPLs, the external server decompiles an apk file and replaces old smali files with pre-prepared smali files. Finally, it recompiles the apk file and signs it.

D. Structure of the proposed method

The proposed method needs to analyze an apk file. However, it is difficult to analyze an apk file in an Android device because there are few analysis tools for an Android device. In addition, the proposed method uses a database that has information of TPLs and latest or security-fixed TPLs. Thus, it is difficult for an Android device to have the database because the size of it is large. Therefore, we use an external server. The proposed method consists of an Android application and an external server. In this paper, we call the Android application and the external server as *Updater Application(UpApp)* and *Updater Server(UpSrv)* respectively. UpApp mainly interacts with user and communicates with UpSrv. UpSrv mainly analyzes apk files.

IV. Implementation

This section describes implementation of the proposed method. The proposed method cooperates with the server to update TPLs. We describe implementation of UpApp and implementation of UpSrv.

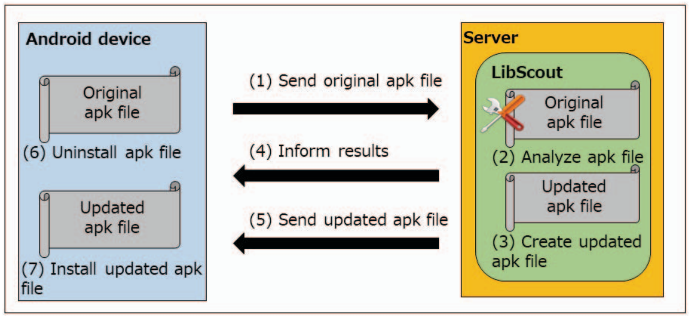


Fig. 1. Overviwe of proposal

A. *Implementation of updating application in Updater Application*

We create UpApp which performs the necessary operations on Android devices. Users need to install and use it to update TPLs in their applications. The functions of it are shown below.

- Sending an original apk file installed on an Android device to UpSrv,
- Receiving an updated apk file,
- Uninstalling the original apk file,
- Installing the updated apk file.

UpApp performs the steps (1), (6), and (7) in Fig.1. It sends an original apk file and receives an updated apk file. In addition, it uninstalls the original apk file and installs the updated apk file if TPLs were updated.

1) Send and receive apk file: A user selects an application and enters it's name into UpApp. UpApp gets path of the apk file of the selected application using the entered name from the Package Manager in Android system. After that, (1)it sends the apk file to UpSrv and receives the updated apk file. UpApp determines whether to receive the updated apk file based on a notification from UpSrv. If UpSrv informs that there is no necessary for updating, it disconnects a connection with UpSrv. And it waits for input from user again. If UpSrv informs that there is need for updating, it receives the updated apk file from UpSrv.

2) Uninstall and install apk file: If UpApp receives an updated apk file, (6)It uninstalls the original apk file on Android device because only one apk file with the same package name can exist on one Android device. The proposed method doesn't change package name. Thus, there is necessary for uninstalling the original apk file to install the updated apk file. After uninstallation, (7)it installs the updated apk file on Android device. Uninstalling and installing apk files are done by issuing intent. After installation, it waits for input from user again.

B. *Implementation of Updater Server*

Functions that need to be implemented in UpSrv are shown below. We use Ubuntu 14.04LTS for UpSrv.

- Sending and receiving apk files,
- Identifying TPLs,
- Updating TPLs,
- Preparing latest TPLs.

UpSrv performs the steps (2)-(5) in Fig.1. It receives the original apk file which user wants to update from UpApp. After that, it analyzes the apk file by calling Smali Replacer LibScout which is a tool adding a function of updating TPLs to LibScout. It updates TPLs in the received apk file from the old version to the security-fixed version by replacing smali file.

Thus, We describe 1)Update TPL, 2)Prepare latest TPLs, and 3)Support Multidex below.

1) Update TPLs: (2)UpSrv analyzes an original apk file and (4)informs whether it has been updated or not to UpApp. If update is needed, (3)UpSrv creates an updated apk file. First of all, it calls Smali Replacer LibScout, which replaces old smali files to new smali files and creates an updated apk file. Then, Smali Replacer LibScout decompiles the apk file when it detects an older TPL and replaces smali files in the decompiled apk file to the security-fixed version. After that, it recompiles the replaced apk file. Here the recompiled apk file is not signed. However, an apk file are signed in order to insall to an Android device. Therefore, Smali Replacer LibScout signs the recompiled apk file with a signature file prepared by the user or the developer of this system in advance. We uses *apktool* to decompile and recompile apk files and *jarsigner* to sign recompiled apk files.

2) Prepare security-fixed TPLs: We target TPLs whose versions can be identified because LibScout has the function to distinguish whether TPL version could be identified. UpSrv uses this function to add information about new TPL to database for identifying TPL. We prepare smali files of a security-fixed version by obtaining jar files and converting it. However, it cannot be directly converted to smali files. Thus, we convert it to smali files via dex files.

3) Support Multidex: Our proposed method supports a multidex apk file, which can refer to more methods than the upper limit from a single dex file. When an application uses multidex, the application has multiple dex. Therefore, the proposed method analyzes all dex files and searches smali files of the target TPLs from them.

V. EVALUATION

This section describes the evaluation of the proposed method. We examine influence of the proposed method on applications and check whether the proposed method can fix vulnerability or not. Then, we summarize results and consider them.

A. How to evaluate

We evaluate the following two items.

1) effects of our proposed method on applications,
2) whether it can fix vulnerabilities.

In investigation on influence of the proposed method, we confirm whether applications to which the proposed method is applied works properly by using 19 applications that are published at Fossdroid [17] or Google Play. They are shown in Table I. This table contains application names, its classifications, TPLs, original versions, and updated(latest or security-fixed) versions. In the evaluation of vulnerability mending, we make a test application that has vulnerability and confirm whether the proposed method can fix it. The evaluation environment is shown in Table II.

B. Evaluation of influence of the proposed method on applications

Influence of the proposed method on the 19 applications is shown in Table III. This table shows results of recompiling, launching, and behavior of applications. From that results, we confirmed that some applications succeed in updating, but others failed. Thus, we investigated the cause of failure in updating.

We selected 12 target applications which failed to recompile, launch, or operate in Table III. We analyzed an apk file if recompiling was failed and searched log files, and if launching or operating was failed. As a result, we found four patterns for update failure. Update failure patterns are shown in Table IV.

In the Pattern 1, a missing method and class causes recompiled applications to crash. In this pattern, " NoClassDefFoundError " and " NoSuchMethodError " occur when the applications crashed. Because the method or class of the version before updating is deleted in the updated version. Therefore, they crash when the applications use the deleted method or class. If there are many changes between versions before and after the updating, this pattern often occurs.

In the Pattern 2, source code obfuscating causes recompilation to fail. In this pattern, as a result of analyzing applications by apktool, a smali file name, class names and method names were obfuscated. Thus, obfuscation makes it difficult that the proposed method identifies the target smali file. In order to solve this pattern, the proposed method needs to decode obfuscation.

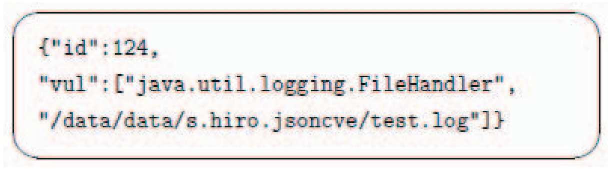
In the Pattern 3, applications did not work properly due to signature verification. In this pattern, messages that a server refused a connection are outputted in a log file. Thus, signature verification to prevent tempering an application causes execution failure of the updated application. The proposed method changes an application's signature because it does not have the correct signature and has to sign an updated application by its own signature. In order to solve this pattern, it is necessary to update the applications without changing the signature. We think solving this pattern is very difficult.

The Pattern 4 is that recompilation fails due to the limitation of Multidex. In this pattern, as a result of analyzing applications by apktool, there were multiple directories that have smali files of a target TPL. Thus, the cause is that the proposed method cannot identify the correct smali file replacement directory because the code of a TPL exists in multiple dex. Because a code of applications is divided into multiple dex if applications have methods above the upper limit of Multidex. Thus, the code of a TPL may be divided. In order to solve this pattern, the proposed method needs to identify the replacement destination of the smali file.

C. Evaluation of vulnerability fix

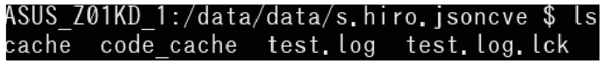
We evaluate whether the proposed method can fix vulnerability. We made two test applications for evaluation. The first application has Jackson-databind's [30] vulnerability(CVE-2017-7525) [31]. The second application has ZeroTurnaround's [32] vulnerability(CVE-2018-1002201) [33]. Jackson-databind and ZeroTurnaround are TPLs. We describe the evaluations of amending two vulnerabilities.

1) Amending Jackson-databind's vulnerability: Jackson-databind's vulnerability allows unauthenticated remote code execution because of deserialization flaws which can be deserialized into a class that enables code execution. This vulnerability exists in versions less than 2.6.7.1, 2.7.9.1 and 2.8.9, hence we updated Jackson-databind from version 2.8.8 to version 2.8.11.2 by the proposed method for evaluation. The test application, which we made for this evaluation, reads a JSON code and deserializes it by Jackson-databind library. We attacked the vulnerability and made it create a log file which is designated in the JSON code by using a vulnerable Jackson-databind library and a malicious one. JSON is shown in Fig.2.



```
{"id":124,
"vul":["java.util.logging.FileHandler",
"/data/data/s.hiro.jsoncve/test.log"]}
```

Fig. 2. Malicious code



```
ASUS_Z01KD_1:/data/data/s.hiro.jsoncve $ ls
cache  code_cache  test.log  test.log.lck
```

Fig. 3. Storage before applying the proposed method

Files named " test.log " and " test.log.lck " were created by the test application without the proposed method. The storage after executing the test application without the proposed

TABLE I
APPLICATIONS

Application name	Classification	TPL used by application	Original version	Updated version
PDF Creator	PDF Creator	itextpdf [18]	5.5.10	5.5.11
Kinolog	Movie acquisition	parceler [19]	1.1.8	1.1.9
Fill	Game	appsflyer [20]	4.7.3	4.7.4
Stationary	Game	retrofit [21]	2.2.0	2.3.0
		okio [22]	1.11.0	1.14.1
30nitidesikkusupakku*	Health care	Joda-Time [23]	2.9.3	2.9.9
Sekainosinbun*	Newspaper	jsoup [24]	1.9.2	1.11.3
Podcast Addict	Radio	Twitter4J [25]	4.0.4	4.0.6
Okusuritetyo*	Health care	Calligraphy [26]	2.1.0	2.3.0
Capitole du Libre	Event	okhttp [27]	3.4.2	3.5.0
Konohanahanandesuka?*	Flower dictionary	retrofit	2.1.0	2.3.0
Spirit Fanfiction	Book	okhttp	3.9.1	3.10.0
LUCRA	Information for women	okhttp	3.8.1	3.8.4
Insight Timer	Timer	Joda-time	2.9.3	2.9.9
Sing!	Karaoke	Twitter4J	4.0.3	4.0.7
Tsuridamasi*	Fishing	Twitter4J	4.0.4	4.0.7
Color by Number	Game	Android-GIF-Drawable [28]	1.2.10	1.2.11
Night Filter	Health care	retrofit	2.1.0	2.3.0
Himatyatto*	Chat	EventBus [29]	3.0.0	3.1.1
Koukou/Daigakujuken*	Study	okio	1.13.0	1.14.1

*:These application names are originally in Japanese.

TABLE II
EVALUATION ENVIRONMENT

Server OS	Ubuntu 14.04LTS
Android device	Zenfone4
Android OS	8.0.0

TABLE III
INFLUENCE OF PROPOSED METHOD ON APPLICATIONS

App name	Recompile	Launch	Behavior
PDF Creator	success	success	correct
Kinolog	success	success	correct
Fill	success	success	correct
Stationary	success	success	correct
30nitidesikkusupakku*	success	success	correct
Sekainosinbun*	success	success	correct
Podcast Addict	success	success	correct
Okusuritetyo*	success	failure	-
Capitole du Libre	success	success	crash
Konohanahanandesuka?*	success	failure	-
Spirit Fanfiction	success	failure	-
LUCRA	failure	-	-
Insight Timer	failure	-	-
Sing!	success	success	connection refused from server
Tsuridamasi*	failure	-	-
Color by Number	failure	-	-
Night Filter	failure	-	-
Himatyatto*	failure	-	-
Koukou/Daigakujuken*	failure	-	-

*:These application names are originally in Japanese.

TABLE IV
UPDATE FAILURE PATTERN

App name	Pattern
Okusuritetyo*	1
Capitole du Libre	1
Konohanahanandesuka?*	1
Spirit Fanfiction	1
LUCRA	2
Insight Timer	2
Sing!	3
Tsuridamasi*	4
Color by Number	4
Night Filter	4
Himatyatto*	4
Koukou/Daigakujuken*	4

*:These application names are originally in Japanese.

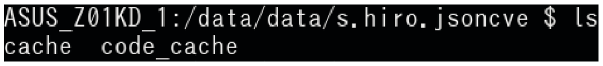
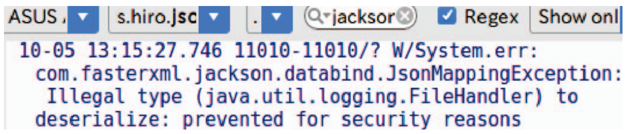

```
ASUS_Z01KD_1:/data/data/s.hiro.jsoncve $ ls
cache  code_cache
```

Fig. 4. Storage after applying the proposed method

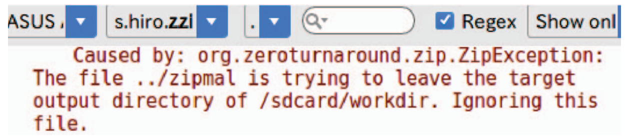
method is shown in Fig.3. Next, we applied the proposed method to it and did the same attack. As a result, the files were not created. The storage after executing it with the proposed method is shown in Fig.4. In addition, messages that deserialization was not done because of security reasons was outputted in a log. The log is shown in Fig.5. Therefore, we confirmed that the proposed method can fix Jackson-databind's vulnerability.

2) Amending ZeroTurnaround's vulnerability: ZeroTurnaround's vulnerability is a kind of zip slip that is a critical archive extraction vulnerability and allows attackers to write arbitrary files on the system. The vulnerability exists in versions less than 1.13, hence we updated ZeroTurnaround from version 1.12 to version 1.13 by the proposed method for evaluation. The test application, which we made for this evaluation, extracts a zip file in an SD Card to the "workdir" directory for ZeroTurnaround. We attacked the vulnerability



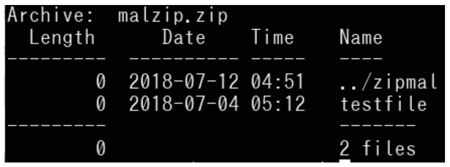
```
10-05 13:15:27.746 11010-11010/? W/System.err:
  com.fasterxml.jackson.databind.JsonMappingException:
   Illegal type (java.util.logging.FileHandler) to
  deserialize: prevented for security reasons
```

Fig. 5. Messages that deserialization was not done because of security reasons



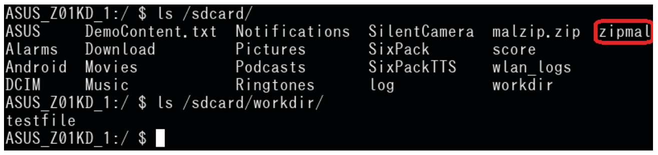
```
    Caused by: org.zeroturnaround.zip.ZipException:
  The file ../zipmal is trying to leave the target
  output directory of /sdcard/workdir. Ignoring this
  file.
```

Fig. 6. Log after applying the proposed method



```
Archive:  malzip.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2018-07-12 04:51   ../zipmal
        0  2018-07-04 05:12   testfile
---------                     -------
        0                     2 files
```

Fig. 7. Malicious zip file



```
ASUS_Z01KD_1:/ $ ls /sdcard/
ASUS     DemoContent.txt  Notifications  SilentCamera  malzip.zip  zipmal
Alarms   Download         Pictures       SixPack       score
Android  Movies           Podcasts       SixPackTTS    wlan_logs
DCIM     Music            Ringtones      log           workdir
ASUS_Z01KD_1:/ $ ls /sdcard/workdir/
testfile
ASUS_Z01KD_1:/ $
```

Fig. 8. Extracting result before applying the proposed method

and made it create files in the parent directory of " workdir " by using a vulnerable ZeroTurnaround and a malicious zip file. The malicious zip file is shown in Fig.7. The file was created in the parent directory of " workdir " by the test application without the proposed method, and the result is shown in Fig.8. Next, we apply the proposed method to it and did the same attack. As a result, The file was not created. In addition, a log in Fig.6 was outputted. From this log, we saw that creation of a file in the parent directory was prevented. Therefore, we confirmed that the proposed method can fix ZeroTurnaround's vulnerability.

VI. Future work

Improvement of library update accuracy will be a future work. There are four patterns of failure as explained in section V-B. It is difficult to solve the Pattern 2, 3, and 4 because of system and security reasons. Therefore, the future work is to solve Pattern 1. The cause of Pattern 1 is that there are many differences between versions of TPLs before and after updating. However, the purpose of the proposed method is to fix vulnerabilities. Thus, we think that we can reduce the failure of the proposed method by updating to a version immediate after fixing vulnerability or a version fixing only the vulnerability.

The another future work is to add a function to take over the data of an application before updating. The proposed method needs to uninstall it to update temporarily. Hence, the data of it before updating lose. The proposed method needs to save the data of it before updating.

VII. Conclusion

In this paper, we proposed a method to update TPLs by users. They can fix vulnerability of TPLs in an application by updating them by cooperating with the external server. From the evaluation result of applying the proposed method to applications, we confirmed that updating by smali file replacement succeeded and vulnerability of TPLs was fixed by the proposed method. In the future, we will enable the proposed method to update using the vulnerability fixed version closest to the current version to reduce update failures. In addition, we will implement a function to save the data of application before updating.

References

[1] News,T, H,:Facebook SDK Vulnerability Puts Millions of Smartphone Users'Accounts at Risk, http://thehackernews.com/2014/07/facebook-sdk-vulnerability-puts.html, Accessed 19 July 2018.

[2] Blog,D,D,:Security bug resolved in the Dropbox SDKs for Android, https://blogs.dropbox.com/developers/2015/03/security-bug-resolved-in-the-dropbox-sdks-for-android/, Accessed 19 July 2018.

[3] Database, V, N,: Apache Commons Collections Java library insecurely deserializes data, http://www.kb.cert.org/vuls/id/576313, Accessed 19 July 2018.

[4] Backes, M, Bugiel, S, and Derr, E,: Reliable third-party library detection in Android and its security applications, Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp.356–367 (2016).

[5] Grace, M, C, Zhou, W, Jiang, X, and Sadeghi, A.R.: Unsafe exposure analysis of mobile in-app advertisements, Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks,pp. 101–112 (2012).

[6] Book, T, Pridgen, A, and Wallach, D, S,: Longitudinal analysis of android ad library permissions, arXivpreprint arXiv:1303.0857 (2013).

[7] Chen, K, Liu, P, and Zhang, Y,: Achieving accuracy and scalability simultaneously in detecting application clones on android markets, Proceedings of the 36th International Conference on Software Engineering, pp. 175–186 (2014).

[8] Narayanan, A, Chen, L, and Chan, C, K,: Addetect:Automated detection of android ad libraries using semantic analysis, Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on, pp. 1–6 (2014).

[9] Liu, B, Liu, B, Jin, H, and Govindan, R,: Efficient privilege de-escalation for ad libraries in mobile apps, Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, pp. 89–103 (2015).

[10] Gibler, C, Stevens, R, Crussell, J, Chen, H, Zang,H, and Choi, H,: Adrob: Examining the landscape and impact of android application plagiarism, Proceeding of the 11th annual international conference on Mobile systems, applications, and services, pp. 431–444 (2013).

[11] Crussell, J, Gibler, C, and Chen, H,: Andarwin: Scalable detection of semantically similar android applications, European Symposium on Research in Computer Security, pp. 182–199 (2013).

[12] Wang, H, Guo, Y, Ma, Z, and Chen, X,: Wukong: A scalable and accurate two-phase approach to android app clone detection, Proceedings of the 2015 International Symposium on Software Testing and Analysis, pp. 71–82 (2015).

[13] Li, M, Wang, W, Wang, P, Wang, S, Wu, D, Liu, J,Xue, R, and Huo, W,: LibD: scalable and precise third-party library detection in android markets, 2017 IEEE/ACM 39th International Conference on Software Engineering(ICSE), IEEE, pp. 335–346 (2017).

[14] Ma, Z, Wang, H, Guo, Y, and Chen, X,: Libradar:Fast and accurate detection of third-party libraries in android apps, Proceedings of the 38th International Conference on Software Engineering Companion, pp. 653–656 (2016).
[15] Mindorks: Android-HotFix, https://github.com/MindorksOpenSource/Android-HotFix, Accessed 19 July 2018.
[16] Derr, Erik and Bugiel, Sven and Fahl, Sascha and Acar, Yasemin and Backes, Michael: Keep me updated: An empirical study of third-party library updatability on Android, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2187-2200 (2017).
[17] Simonin, D,: Fossdroid, https://fossdroid.com/, Accessed 19 July 2018.
[18] iText Software:ITEXT Developers, http://developers.itextpdf.com/, Accessed 19 July 2018.
[19] Ericksen, J,:Parceler, https://github.com/johncarl81/parceler, Accessed 19 July 2018.
[20] Appsflyer: Appsflyer, https://www.appsflyer.com/, Accessed 19 July 2018.
[21] Square:Retrofit, http://square.github.io/retrofit/, Accessed 19 July 2018.
[22] Square: okio, https://github.com/square/okio, Accessed 19 July 2018.
[23] jodastephen: Joda-Time, http://www.joda.org/joda-time/, Accessed 19 July 2018.
[24] jhy: jsoup, https://jsoup.org/, Accessed 19 July 2018.
[25] Twitter4J: Twitter4J, http://twitter4j.org/en/index.html, Accessed 19 July 2018.
[26] chrisjenx:Calligraphy, https://github.com/chrisjenx/Calligraphy, Accessed 19 July 2018.
[27] Square: OkHttp, http://square.github.io/okhttp/, Accessed 19 July 2018.
[28] Karol: android-gif-drawable, https://github.com/koral–/android-gif-drawable, Accessed 29 August 2018.
[29] greenrobot: EventBus, http://greenrobot.org/eventbus, Accessed 29 August 2018.
[30] cowtowncoder: Jackson-databind, https://github.com/FasterXML/jackson-databind, Accessed 19 July 2018.
[31] Common Vulnerabilities and Exposures:CVE-2017-7525, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7525, Accessed 19 July 2018.
[32] ZeroTurnaround:ZeroTurnaround,https://zeroturnaround.com/, Accessed 19 July 2018.
[33] Snyk: Zip Slip Vulnerability,https://snyk.io/research/zip-slip-vulnerability, Accessed 19 July 2018.