Wireless Pers Commun (2018) 103:491–503
https://doi.org/10.1007/s11277-018-5456-4

Identify and Inspect Libraries in Android Applications

Hongmu Han[1] · Ruixuan Li[1] · Junwei Tang[1]

Published online: 8 February 2018

Abstract Libraries may become a liability for users security. Existing studies show that libraries can be exploited to propagate malware. Hackers utilize fake or modified libraries to execute malicious behaviours. Vetting library instances in applications are desirable. However, it is impeded by the absence of robust library detection method and library vetting method. This paper proposes a hybrid library detection method that it combines name-based method and feature-based method to identify library instances in applications. It can resist simple identifier renaming. Furthermore, this paper proposes an abnormal library detection method that it utilizes frequent pattern to measure the normal degree of library instances. Comparing with existing methods, the abnormal library detection method can not rely on original library files. A ground truth dataset that it consists of 177 malicious applications with abnormal library instance and 81,317 benign apps is used to demonstrate the effectiveness of proposed approaches. Experimental results show that the approaches can precisely detect library instances and effectively reduce the cost of abnormal library detection.

Keywords Android apps · Library detection · Frequent pattern · Abnormal library · Malware

✉ Ruixuan Li
rxli@hust.edu.cn

Hongmu Han
hanhongmu@hust.edu.cn

Junwei Tang
jwtang@hust.edu.cn

[1] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

1 Introduction

Android OS has dominated globe mobile OS market share. Hundreds of millions third-party applications(apps) promote the market prosperity, i.e., Google Play contains more than 2 million apps in 2016. Apps leverage libraries to provide various advanced functions and speed development process. Existing studies shown that more than 60% sub-packages are libraries in 100,000 apps [1].

Although libraries play an important role in Android ecosystem, they may cause some security problems to mobile users. A vulnerable code library may threaten mass of apps. Ad libraries can abuse permission and leak user privacy. In addition, aggressive ad libraries can execute malicious behaviors. At last, libraries can be exploited to propagate malware. Piggybacked apps insert hooks into library to activate malicious payload. Besides modified existing libraries, malicious developers can also create a fake library. For instance, FakeUM is a type of malware that it pretends a famous library named UMeng. Public code repositories (e.g., GitHub) and online forums can be utilized to propagate fake libraries. Benign app developers may integrate abnormal libraries into their apps by mistake.

Abnormal libraries high impact on the health of the entire Android ecosystem. Although anti-virus software can detect some malicious behaviors in library code, malicious writers are still able to modify their code quickly and against current detection techniques [2]. Library detection technology impedes abnormal library detection. Code obfuscators are widely used in app development. It is difficult to distinguish library instances from apps.

Recently, some methods were proposed to identify and analyze libraries in Android apps. Li et al. [3] leverage frequent item to discover potential library packages. Duet proposed an integrity verification method to find masquerading or modified library in apps [4]. But this method relies on original library files. App developers usually not update libraries while their update their apps [5]. It is hard to collect historical library files for ever library. In addition, some library providers allow app developers to add other libraries in their libraries namespace. Therefore, it is difficult to leverage library integrity verification to discover abnormal library instances from apps.

This paper proposes a hybrid library detection approach that it combines name-based method and feature-based method to identify libraries. Although code obfuscator replaces identifiers with meaningless symbols to protect programs, op-code and Android tags are not be changed by simple identifier renaming. So the hybrid library detection can resist code obfuscator that leverages identifier renaming to protect code. Later, this paper introduces frequent pattern to measure the normal degree of library instance in apps. The abnormal library detection cant rely on historical library files to discover abnormal library instances from apps.

The contributions of this paper are as follows:

1. This paper proposed a library detection method that can resist simple identifier renaming technology to identify libraries. The library detection method combines name-based method and feature-based method to discover library instances from apps.
2. This paper introduces a new method that leverages frequent pattern to measure the normal degree of libraries within Android apps.
3. This paper collected 185 malicious apps and 80,000 benign apps to evaluate proposed methods. The 185 malicious apps use abnormal library to propagate malware.

The rest of the paper structured as follows. This paper presents a description of background, code protection and the threat model of libraries in Sect. 2. It describes approaches in Sect. 3. Then, it presents experimental results in in Sect. 4. The works are discussed in Sect. 5. Finally, it reviews the related work in Sect. 6, and concludes the paper in Sect. 7.

2 Background

2.1 Code Library

The Android system is an open source software stack. It includes Linux kernel, libraries, android runtime, android framework and apps. According to the purpose and task of libraries, libraries can be divided into two categories. The first category of libraries is the Android core libraries (Android Runtime). The second category of libraries is embedded in apps. This paper focuses on the security threat of the latter type of libraries. Libraries provide various advanced functions for app developers, for instance analytic, crash reporting, authentication, advertisement, cryptography, push messaging and more. Although developers like to integrate libraries into their apps, few of them will validate the security of libraries. A library provides a set of well-defined app programming interfaces (APIs). So libraries offer conveniently for app developing, mass of apps are built upon different libraries.

Library providers usually maintain their libraries online. Some library providers public their libraries on their official websites, the others can scatter their libraries by a various sources, for instance, public code repositories, forums and underground. However, not all of library providers will maintain historical versions available. Many library providers only provide latest versions. Since providing a vulnerable history library may cause security threats to apps.

This paper defines the abnormal libraries that are fake or modified libraries.

2.2 Code Protection

Code obfuscator can protect apps from reverse engineering. Simple code obfuscator replaces identifiers with semantically obscure names. Furthermore,advanced code obfuscator can provide some advanced functions, such dex obfuscation.

ADT integrates an open source code obfuscator that its name is ProGuard [6]. This tool can shrink, optimize, and obfuscate program code. Although there are many advanced obfuscators, ProGuard is a popular obfuscator for Android development. However, incorrect code obfuscation may cause program crash. It is not recommend obfuscating code twice.

2.3 Library Security Threat

Libraries are vital part for Android ecosystem. With the help of libraries, developers can scalable develop an app. Since libraries are used pervasively, the security of libraries should attract more attention. Android apps are mainly written in Java. It is not difficult to decompile and rebuild an app. With more and more security analysts pay attention to mobile platform. Various sophisticated technologies are used to propagate malware on the mobile platform. It was reported that libraries were used to propagate malware. Taking Piggybacked app as an example, some piggybacked apps inject hooks into libraries to activate malicious rider code [7]. Using masquerade library is another attack method to avoid detection. Malicious code uses reputation namespaces to pretend benign libraries [8]. For instance, DroidKungFu and FakeUM are two malware families that pretend well-known libraries. Abnormal libraries can be exploited to propagate malware.

3 Approaches

This paper has two experimental goals. The first one is to precisely identify libraries. The second one is to discover abnormal libraries which are malicious, fake or modified libraries in apps.

3.1 Library Identification

Existing studies proposed two types of approach to discover libraries from apps. It is a simple type of method that leverages package name to discover libraries. But this method can be confused by code obfuscator. Another type of method leverages class hierarchy features to identify libraries [5]. A library structure consists of three layers that they are method, class and package. However, this method may cause ambiguous that two different libraries have the same class hierarchy features. This paper proposes a hybrid approach that it combines name-based method and feature-based method to identify library. Furthermore, the feature-based method extracts more fine features to reduce ambiguous detection results.

For every app, it uses a bottom-up approach that it uses a hash tree to represent an app hierarchy. Figure 1 shows the bottom-up approach. An app contains several packages. A package may include multiple class files. A class consists of several methods. It extracts method-level features to construct hash tree. Androguard is used to construct a control flow graph (CFG) for every method. Then, the method-level features are extracted from the CFG. Those features include opcodes and Android type tags. A method hash value is computed by hashing all features in the method. It sorts all method hash values in an

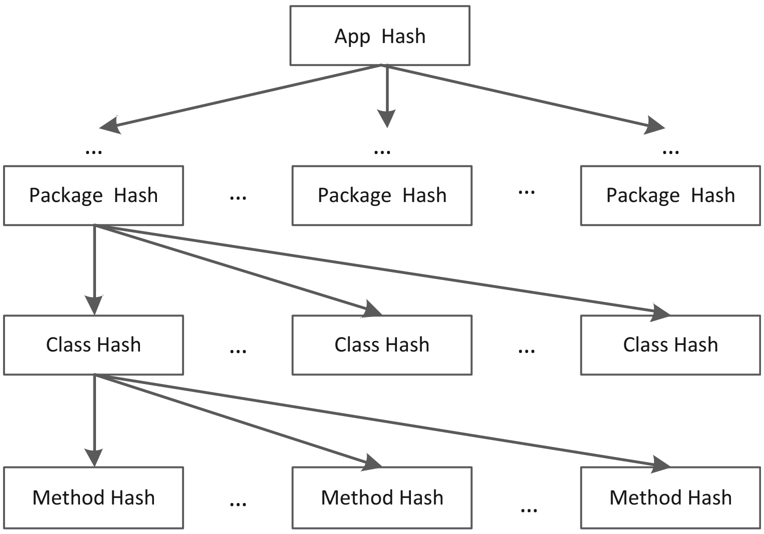


Fig. 1 Hash tree for app hierarchy

ascending order in the same class. Then, the class hash value can obtain by hashing its method hash sequences. The same strategy can be used to compute the package hash value and app hash value. At last, the hash tree will be stored, but it excludes the method-level hash values to save the storage space. A two-tuples $< name, hash >$ is used to represent a node in the hash tree.

There are two challenges to identify libraries. The first one is that code obfuscator can replace identifiers with meaningless strings. The other challenge is that a library includes several versions. While storing all hash trees in a database, the library detection method takes the following two steps to discover libraries.

1. It leverages package names to identify libraries and corresponding package hash values. The package names can be specified or statistics frequently used library names in the database.
2. It uses package hash values to discover all potential libraries whether they are obfuscated or not.

3.2 Frequent Pattern Based Abnormal Library Detection

This section discusses the detail approach that it is designed to discover abnormal libraries from a plenty of Android apps. This paper introduces a measure that it uses a frequent pattern to evaluate the normal degree of a library in Android app. The basic idea is very simple. If a library is used in many apps, it means that the library is a normal library. The concepts of frequent pattern mining are described as following.

Let $I = \{I_1, I_2, \ldots, I_m\}$ is a set of items. Let D is a set of database transactions where each transaction T is a set of items. T is a subset of I. An absolute support of X is the occurrence frequency that how many transactions contain the X. A relative support of X is the percentages of transactions in D containing X.

Let X is a frequent itemset in set I, if there exists no super-itemset Y such that $X \subset Y$ and Y is a frequent itemset in set I, then the itemset X is a maximal frequent itemset (or max-itemset) in set I.

Frequent pattern outlier factor (FPOF) is proposed to measure the normal degree of a transaction [9]. Giving a support threshold ξ, let F be a complete set of frequent itemsets for a transaction database DB, $Sup(X)$ represents the support value of an frequent itemset X, $X \in F(DB, \xi)$, the frequent pattern outlier factor for transaction t is defined as follow:

$$FPOF(t) = \frac{\sum_{X \subseteq t, X \in F(DB,\xi)} Sup(X)}{||F(DB, \xi)||} \tag{1}$$

For any frequent itemset X and its subsets, the FPOF method will duplicate add the support degrees. In addition, if an infrequent item i is added to a transaction t to create a new transaction t', the value of $FPOF(t)$ equals the value of $FPOF(t')$. Therefore, the outlier measure $FPOF$ cannot be fully taken into account the normal degree of a transaction. According to the concept of frequent itemset, the longer frequent pattern has more subset frequent patterns. To be a normal transaction, a transaction may include a longer superset frequent pattern than others. This paper introduces another measure called long frequent pattern outlier factor $LFPOF$ to evaluate the normal degree of a transaction [10]. It defines as follows:

$$LFPOF(t) = \frac{|X_{max}|}{|t|} \tag{2}$$

Where $F(DB, \xi)$ is a frequent pattern set and $F(t, \xi) = X|X \in F \bigcap X \subseteq t$, X_{max} is the longest frequent pattern in $F(t, \xi)$, $|X_{max}|$ and $|t|$ are the length of X_{max} and transaction t respectively.

Since X is a subset of t, the value of $LFOPF(t)$ is less than or equal to 1. If the value of $LFOPF$ equals 1, which the transaction t may be a normal transaction.

If there are n apps contain library L, the abnormal library detection extracts the library information from the n apps and build a transaction database $DB_L = \{t_1, t_2, \ldots, t_n\}$. A library may contain hundreds of class files. The abnormal library detection treats a class file as an item. A set of items is used to represent a library.

For every transaction t_i in the database, the abnormal library detection computes its long frequent patter outlier factor $LFOPF(t_i)$. If the value of $LFOPF(t_i)$ is less than 1, the transaction t_i is likely to be an anomaly.

4 Evaluation

4.1 Experimental Data

There are two types of data to evaluate the proposed methods. The first type is malicious apps that leverage abnormal libraries to propagate malware. The second type is benign apps. The dataset describes as following:

4.1.1 Malware Dataset

The malware dataset consists of 177 malicious apps with abnormal libraries. There are 132 piggybacked apps that they were injected hooks into libraries to carry out rider code. Those apps are collected from AndroZoo project [7]. The rest of 45 malicious apps use fake UMeng and Android support library to execute malicious behaviors. These malicious apps were gathered from CNCERT website. All of the malicious apps are listed in Table 1.

4.1.2 Benign Dataset

Since malicious apps were found in all of app markets. This paper utilizes VirusTotal that integrates 54 anti-virus scanners to screen benign apps [11]. The VirusTotal provides public APIs for users to check suspicious files. Users can quickly get analyzed results by searching VirusTotals cache. VirusTotal defines limited queries per day rate. A free

Table 1 The malware dataset

Category	Library	Behavior	Number
Fake	UMeng	Malware	5
Fake	Android support v4	Malware	40
Modified	Unity3D	Hook	95
Modified	Corona	Hook	17
Modified	AndEngine	Hook	10
Modified	Prime31	Hook	10

member can send 5760 query requests to VirusTotal one day. With the help of VirusTotal, this benign dataset consists of 81,317 benign apps.

4.2 Library Instance Detection

Code obfuscator is widely used in Android development. It helps developers to protect their code. This paper tries to identify obfuscated library instances from apps. The experiments take the six libraries as study targets. The six libraries are shown in Table 1. The experiments use benign dataset to discover library instances.

Existing library detection methods can be classified into two types. The first one leverages library names to identify libraries [3]. The second one uses class hierarchy information to find libraries [5]. The former cannot identify obfuscated libraries. The latter may cause ambiguous results that two different libraries have the same class hierarchy information. This paper proposes a hybrid approach that combines name-based method and feature-based method to discover library instances from apps. Comparing with existing methods, it extracts more information to label method, class and package. It analyzes and builds a hash tree for every app. Then, the hash tree will be stored.

For the six libraries, library names will be used to find library package hash values at first. Later, using package hash values discover obfuscated library instances from apps. Since method-level features include opcode and Android tags. Those features weren't be changed by simple code obfuscator, e.g., Proguard. If an updated library version modifies any opcode or Android tags, the feature hash value also will change. So the hybrid library detection method can precisely discover library instances. The experimental results show in Table 2.

4.3 Abnormal Library Analysis

This paper introduces frequent pattern to measure the normal degree of library instances. This paper should verify an assumption that not all of app developers modified libraries in their apps at first. Then, the paper should verify the frequent pattern based outlier detection that it can be used to find abnormal library instance in apps.

4.3.1 Frequent Used Libraries

Reverse-engineering tool is used to gather libraries information. A single library has hundreds of class files. Merging the content of all class files of the library, the experiment calculates a cryptographic hash value of the content. Although hash value can be changed by code shrink, code optimization and code obfuscation. The experimental result shows

Table 2 Identified library instances

Library	Number
UMeng	7189
Android support v4	35,978
Unity3D	13,146
Corona	2272
AndEngine	2576
Prime31	3144

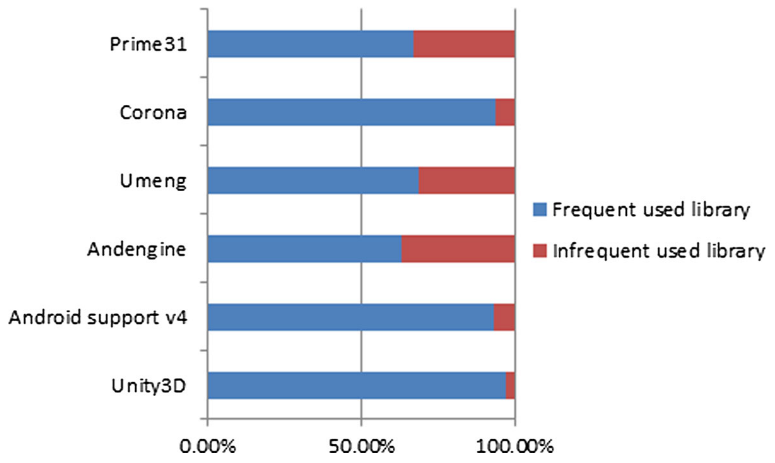


Fig. 2 An investigation of library usage

Fig. 2. The result indicates that many apps have the same library hash value. It means that many developers arent modified the libraries in their apps.

Although experimental result shows that many apps share the same libraries in their apps, it needs to verify app library packages equal to library original packages [4]. This paper adopts a compile and retargeting method that it integrates library original packages into new apps and decompile the new apps. It is difficult to collect a complete library history. Because some library providers arent wanting to offer their history libraries available. The experiments only collect three of the six libraries' historical version files. It developed 44 apps to integrate those historical version library files.

As previous section described, this paper compares app library packages with original library file at class-level. A library has hundreds of class files. It treats a class as an item and a library package as a transaction in a database. Then, frequent pattern mining tool is used to mine frequent patterns from apps. To reduce the number of frequent itemsets, experiments mine maximal frequent itemsets. Frequent pattern mining defines support threshold as 10 to mine maximal frequent itemsets. While maximal frequent itemsets are found, it compares them with original library information at class-level. The experimental results are shown in Table 3.

The experimental results show that many apps use a subset of orrginial libraries. There are several reasons for this problem. Apps may use a subset of a library. In addition, app developers may modify libraries in their apps. Although maximal frequent itemsets are not

Table 3 The results of maximal frequent itemsets verification

Name	Instances	# of maximal frequent itemsets	# of original library
Umeng	1433	14	12
Andengine	1288	10	1
Android support v4	15,522	42	30

equal original libraries, experimental results point out that most of app developers aren't modify libraries.

4.3.2 Library Outlier Detection

In the frequent pattern based abnormal library detection process, the experiments combine malicious dataset and benign market dataset as experimental dataset. For every app, it leverages reverse-engineering tool to decompile it and extract information from decompiled files.

For every library, the experiments traverse all apps and build a set of items in the library. It treats a class as an item and a library instance as a transaction, a transaction database is created for a library. In order to avoid abnormal library instances are treated as normal. It leverages the transactions of benign apps to mine frequent itemsets. Then, it uses the set of frequent itemsets to compute LFPOF for every library instance whether come from benign dataset or malicious dataset.

The LFPOF usually is considered to be in an ascending sort order. The top n LFPOFs represent outliers. The n is a threshold. However, ascending LFPOF isn't suitable for abnormal libraries detection. Taking piggybacked app as an example, malicious hackers only modify a library class to add a hook. Therefore, the LFPOF of modified library is very close to 1. Therefore, it is hard to discover this type of modified libraries by using top n as threshold.

The experiments define the normal value of LFPOF as 1. Any value of LFPOF that is less than 1 is defined as an outlier. The experimental results are shown in the Table 4. Experimental results show that a large proportion of library instances can pass outlier detection in benign dataset. None of abnormal library instances can pass outlier detection in malicious dataset. There are many reasons for these results. Some library providers may consecutive public several versions to fix bugs. But not all of app developers will update libraries in their apps. This reason may cause that a few of library versions are used by app developers. Although frequent pattern based outlier detection labels many benign library instance as outliers, this approach is the first method to discover abnormal library instances in apps without having to collect history library files. It significantly reduces the cost to discover abnormal library instances.

Table 4 Frequent pattern based abnormal library detection

Name	Type	Number	Outlier (%)
Android support v4	Benign	15,522	15.85
	Malware	40	0
UMeng	Benign	1433	56.94
	Malware	4	0
Unity3D	Benign	7111	16.60
	Malware	95	0
Corona	Benign	2273	33.11
	Malware	18	0
AndEngine	Benign	1288	68.38
	Malware	10	0
Prime31	Benign	1819	65.41
	Malware	10	0

5 Discussion

Libraries can be exploited to propagate malware. This paper pays attention to identify libraries and discover potential abnormal libraries. It proposed a hybrid library detection method that it combines name-based and feature-based library detection methods to discover library instances from mass of apps. The library detection method extracts more fine features that can resist simple identifier renaming. In addition, this paper introduces frequent pattern based outlier detection that leverages the character of library usage to discover abnormal library instances. Comparing with existing library integrity verification method, the frequent pattern based outlier detection needn't to collect historical library versions [4]. The approach leverages frequently used benign library instances to vet library instances in apps. Although this approach relies on benign dataset to mine frequent patterns, it is easy to get a benign dataset for app stores and security analysts. Furthermore, frequent pattern based abnormal library detection can reduce the cost of abnormal detection, it only relies on a large number of benign apps to identify abnormal library instances.

6 Related work

Android is the most popular operation system for smartphone. As Android become ever more prevalent, its security problems become hot topics in the research community.

Android permission and privacy: Android leverages permission mechanism to build a finer-grained access control on sensitive resource. Developers must explicitly define a permission request in the Manifest file. Over-privilege is a serious problem in Android ecosystem. Malicious attackers could exploit permissions to launch attacks. Several policies are proposed to prevent privilege from escalation [12]. Many studies utilize apps' permissions as feature vectors and use machine learning methods to identify malicious apps [13]. However, as Android operation system evolution, apps become more and more sophisticated. In addition, most of free apps contain one or more ad libraries. It becomes difficult to leverage permission to distinguish malicious apps from benign apps. Smartphone is a private intensive device. Advertising networks like to gather users' location information and push a precise ad. Several methods are also proposed understanding users' privacy comprehensively [14].

Library studies: Libraries are used pervasively in Android app developing. According to their functions, libraries can be divided into several categories of libraries, such as analytic, crash reporting, authentication, advertisements, cryptography and push messaging. Ad library plays an important role in Android ecosystem. But many ad libraries have been discovered to gather users' private information, especially for aggressive ad libraries [15]. Pluto [16] is a framework which can discover user data exposes to ad libraries.

Libraries provide various advanced functions to speed up apps development. Li et al. [3] investigated Android common libraries and published a set of libraries. LibScout [5] leverages library profiles to precise identify third libraries. Existing studies show that many apps slowly adapt new library versions. This result indicates many apps contain old libraries that may have known security vulnerabilities [17]. LibD extracts opcode as feature to discover libraries from apps [18]. LibRadar [19] also used API features to detect third-party libraries in Android apps. LibCage [20] is a tool which can leverage sandbox to prevent third-party libraries abuse permission. Duet [4] utilized integrity verification to protect libraries from modifying, masquerade and aggressive library threat. Existing methods rely on library original files to extract feature information. It is impossible to collect all historical versions for every library.

7 Conclusion

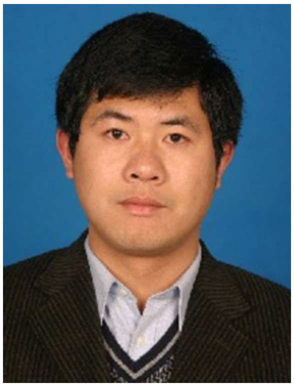
Libraries are widely used in Android apps developing. However, libraries can be exploited to propagate malware. Suspicious developers hide malicious code or inject hooks in libraries to evade anti-virus detection. It is necessary to identify and vet libraries to protect apps. But code obfuscator makes it is difficult to discover library instances from apps. In addition, there is a lack of an effective method to identify abnormal library instances in apps. This paper proposed a hybrid library detection approach that it can resist identifier renaming and identify library instances in apps. Furthermore, this paper introduces frequent pattern based outlier detection to discover abnormal library instances in apps. This paper uses a ground truth dataset that includes ground truth malware apps with abnormal libraries and benign apps to verify proposed approaches. The experimental results demonstrate that the proposed methods are effective. The abnormal method can’t rely on original library files and reduce the cost of abnormal library detection. According to the study of this paper, it is not recommended programmers to integrate unknown library code into their apps. It is better to get a library from a trusted website. Furthermore, the risk of Android libraries should attract more attention in the future.

Acknowledgements This work is supported by the National Key Research and Development Program of China (No. 2016YFB0800402), National Natural Science Foundation of China under Grants 61572221, U1401258, 61433006, 61300222, 61370230 and 61173170, Innovation Fund of Huazhong University of Science and Technology under grants 2015TS069 and 2015TS071, Science and Technology Support Program of Hubei Province under Grant 2015AAA013 and 2014BCH270, and Science and Technology Program of Guangdong Province under Grant 2014B010111007.

References

1. Wang, H., Guo, Y., Ma, Z., & Chen, X. (2015). WuKong: A scalable and accurate two-phase approach to Android app clone detection. In *Proceedings of the 2015 international symposium on software testing and analysis* (pp. 71–82). ACM.
2. Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., et al. (2015). Android security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys and Tutorials*, *17*(2), 998–1022.
3. Li, L., Bissyand, T. F., Klein, J., & Traon, Y. L. (2015). An Investigation into the Use of Common Libraries in Android Apps. Preprint arXiv:1511.06554.
4. Hu, W., Octeau, D., McDaniel, P. D., & Liu, P. (2014). Duet: Library integrity verification for android applications. In *Proceedings of the 2014 ACM conference on security and privacy in wireless and mobile networks* (pp. 141–152). ACM.
5. Backes, M., Bugiel, S., & Derr, E. (2016). Reliable Third-Party Library Detection in Android and its Security Applications. In *Proceedings of the 23rd ACM conference on computer and communication security (CCS16)* (pp. 356–367). ACM.
6. ProGuard. (2017). http://developer.android.com/tools/help/proguard.html.
7. Li, L., Li, D., Bissyand, T. F., Klein, J., Traon, Y. L., Lo, D., et al. (2017). Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security*, *12*(6), 1269–1284. https://doi.org/10.1109/TIFS.2017.2656460.
8. Zhou, Y., & Jiang, X. (2012). Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy (SP)* (pp. 95–109). IEEE.

9. He, Z., Xu, X., Huang, J. Z., & Deng, S. (2005). FP-outlier: Frequent pattern based outlier detection. *Computer Science and Information Systems*, *2*(1), 103–118.
10. Zhang, W., Wu, J., & Yu, J. (2010). An improved method of outlier detection based on frequent pattern. In *2010 WASE international conference on information engineering (ICIE)* (Vol. 2, pp. 3–6). IEEE.
11. Virustotal—free online virus, malware and url scanner. (2017). www.virustotal.com.
12. Liu, B., Liu, B., Jin, H., & Govindan, R. (2015). Efficient privilege de-escalation for ad libraries in mobile apps. In *Proceedings of the 13th annual international conference on mobile systems, applications, and services* (pp. 89–103). ACM
13. Sarma, B. P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., & Molloy, I. (2012). Android permissions: A perspective combining risks and benefits. In *Proceedings of the 17th ACM symposium on access control models and technologies* (pp. 13–22). ACM.
14. Nan, Y., Yang, M., Yang, Z., Zhou, S., Gu, G., & Wang, X. (2015). Uipicker: User-input privacy identification in mobile applications. In USENIX Security (pp. 993–1008).
15. Short, A., & Li, F. (2014). Android smartphone third party advertising library data leak analysis. In *2014 IEEE 11th international conference on mobile ad hoc and sensor systems (MASS)* (pp. 749–754). IEEE.
16. Demetriou, S., Merrill, W., Yang, W., Zhang, A., & Gunter, C. (2016). A. Free for all! assessing user data exposure to advertising libraries on android. In *Proceedings of the 23th annual network and distributed system security symposium (NDSS), San Diego, California, USA*, February 21–24, 2016.
17. Derr, E., Bugiel, S., Fahl, S., Acar, Y., & Backes, M. (2017). Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 24rd ACM conference on computer and communication security (CCS17)* (pp. 2187–2200). ACM.
18. Li, M., Wang, W., Wang, P., Wang, S., Wu, D., Liu, J., et al. (2017). LibD: Scalable and precise third-party library detection in android markets. Paper presented at the Proceedings of the 39th International Conference on Software Engineering, Buenos Aires, Argentina.
19. Ma, Z., Wang, H., Guo, Y., & Chen, X. (2016). LibRadar: Fast and accurate detection of third-party libraries in Android apps. In *Proceedings of the 38th international conference on software engineering companion* (pp. 653–656). ACM.
20. Wang, F., Zhang, Y., Wang, K., Liu, P., & Wang, W. (2016). Stay in Your Cage! A *Sound Sandbox for Third-Party Libraries on Android*. In I. Askoxylakis, S. Ioannidis, S. Katsikas, & C. Meadows (Eds.), *Computer Security ESORICS 2016: 21st European symposium on research in computer security, Heraklion, Greece, September 26–30, 2016, Proceedings* (pp. 458–476). Cham: Springer International Publishing.



Hongmu Han is currently a Ph.D. student in the School of Computer Science and Technology at Huazhong University of Science and Technology. He received his B.S. and M.S. degrees in Computer Science from Wuhan Institute of Technology, in 2004 and 2007, respectively. His research interests include mobile security, machine learning and cloud security.

Ruixuan Li is a full Professor of School of Computer Science and Technology at Huazhong University of Science and Technology (HUST), Wuhan, P. R. China. He holds a B.S. and M.Sc. in Computer Science from Huazhong University of Science and Technology in 1997 and 2000 respectively, and a Ph.D. in Computer Science from the same university in 2004. His interests include cloud computing, big data management and analysis, distributed system security, information retrieval, data mining, social network, peer-to-peer computing, data integration, semantic web and ontology.

Junwei Tang is currently a Ph.D student in the School of Computer Science and Technology at Huazhong University of Science and Technology. He received his B.S. degree in Computer Science from Huazhong University of Science and Technology in 2013. His research interests include mobile security and natural language processing.