LibRadar: Fast and Accurate Detection of Third-party Libraries in Android Apps

Ziang Ma, Haoyu Wang, Yao Guo, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education),
School of Electronics Engineering and Computer Science, Peking University
{maziang, howiepku, yaoguo, cherry}@pku.edu.cn

ABSTRACT

We present LibRadar, a tool that is able to detect third-party libraries used in an Android app accurately and instantly. As third-party libraries are widely used in Android apps, program analysis on Android apps typically needs to detect or remove third-party libraries first in order to function correctly or provide accurate results. However, most previous studies employ a whitelist of package names of known libraries, which is incomplete and unable to deal with obfuscation. In contrast, LibRadar detects libraries based on stable API features that are obfuscation resilient in most cases. After analyzing one million free Android apps from Google Play, we have identified possible libraries and collected their unique features. Based on these features, LibRadar can detect third-party libraries in a given Android app within seconds, as it only requires simple static analysis and fast comparison. LibRadar is available for public use at http://radar.pkuos.org. The demo video is available at: https://youtu.be/GoMYjYxsZnI

1. INTRODUCTION

Android apps have been studied extensively in recent literature. One particular feature of Android apps is that most of them use *third-party libraries*. For example, app developers use advertising service libraries to generate revenue; they can also use social networking libraries and other development libraries to facilitate development and add new features to their apps. Libraries such as social networking libraries and mobile analytic tools are very popular [1]. Many apps use more than 20 third-party libraries [3].

Because third-party libraries account for a large portion of the code in Android apps, program analysis on Android apps typically requires detecting or removing third-party libraries first. Third-party libraries usually introduce significant noise and they could significantly affect the results of many tasks. For example, in app clone detection [9, 16] and app behavior analysis, most approaches need to extract code level features from decompiled apps. The results will differ greatly if third-party libraries are not correctly removed.

At the same time, wide use of third-party libraries brings new security and privacy risks. Recent research [14] reveals that third-party libraries, even the most popular ones, may pose threats to privacy of mobile users.

It is challenging to identify third-party libraries from compiled binary code (or byte code) accurately. Most of the state-of-the-art approaches use a whitelist of package names (directories) to identify third-party libraries by comparing their package names. For example, Centroid [8] uses a list of 73 libraries to filter third-party libraries, while AdRisk[11] uses a list of 100 libraries.

However, relying on package names to identify third-party libraries can be very inaccurate, as package names can be modified in many ways (e.g., package name obfuscation). In contrast to package names, *API features* such as the call frequencies of different APIs are typically unobfuscated.

We present *LibRadar*[1], a tool to detect third-party libraries used in Android apps accurately and instantly. One key idea of LibRadar is that we use stable code feature, instead of a whitelist of package names, to detect the third-party libraries directly within an app.

We first extend a clustering-based technique $Wukong$ [15] to identify possible libraries from analyzing one million apps from Google Play. For the over 10,000 library versions identified through clustering, we use a hashing-based representation to extract unique features from each library. This step is time-consuming, but only needs to be done once to generate the list. We may also need to update the list of libraries regularly to include new libraries or new library versions.

Once the list of features has been collected, LibRadar is able to identify the third-party libraries in a given app instantly, as it only requires simple static analysis and feature comparison. LibRadar is also robust against code obfuscation, because it relies on stable code feature that cannot be obfuscated with obfuscation tools.

2. TOOL DESIGN

The overall architecture of LibRadar is shown in Figure 1. LibRadar includes three steps: *feature extraction* to retrieve stable code feature and convert them in to a light-weight hashing-based format, *multi-level clustering* on a large set of apps to identify potential library candidates, and *instant detection* on a given app to detect the libraries in it based on the clustered results.

For feature generation, we first decompile the repository of Android apps from their DEX byte code into intermediate Smali [4] code using Apktool [2]. We extract stable code

ICSE '16 Companion, May 14-22, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4205-6/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889178

[1] http://radar.pkuos.org

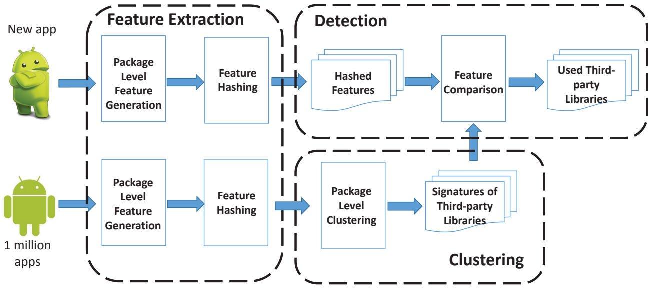


Figure 1: Overall architecture of LibRadar.

feature for all the folders in the Smali code. We use feature hashing to get the signature of each folder.

Then we use a clustering-based approach to identify third-party libraries. Packages of third-party libraries are clustered into large clusters because a large number of apps use it. The clustering result is a list of potential libraries and their hashed features, which will be used by LibRadar to detect libraries in a given app.

For a given app, we extract the stable code feature for all the folders in the package structure and use feature hashing to get the signature of each folder. Then we compare the hashed features with the list of all potential libraries to detect third-party libraries in this app.

2.1 Feature Extraction

We mainly focus on two challenges in feature extraction: code obfuscation and efficient feature representation for faster comparison.

2.1.1 Stable Code Feature

In order to enable fast and accurate comparison, we want features that remain stable even if the code is obfuscated, which we call *stable code feature*. The feature we choose is the frequency of different Android API calls. This features cannot be modified during obfuscation and they capture the low-level behaviors of third-party libraries, thus they could provide accurate indication during third-party library detection. API call features are extracted by analyzing the Smali code.

2.1.2 Feature Hashing

For the extracted features of each package, we represent them in a light-weight format using feature hashing, mainly with two reasons: (1) We are applying our enhanced clustering-based approach to over 1 million apps, which contains more than 71 million packages, while each package has thousands of features, making it difficult to directly use these features in the following clustering phase. (2) LibRadar aims to detect third-party libraries used in a given app instantly. For a given app, it is time-consuming to compare the overall features of all the packages in it with all the features of potential libraries.

As shown in Figure 2, to get the fingerprint of each package, we compute the hashing value of the feature vector as follows:

$$hash_{pac_m} = (\sum_{i=1}^{n}(i * V_i))\%LARGEPRIME$$

Note it is possible that different feature vectors get the same hash value. In order to get a unique fingerprint for each package, we also use two meta features of extracted feature vectors: (1) total number of API calls; (2) total kinds of API calls. With the aid of these meta features, the fingerprint of each package is unique and we have not found cases that two different feature vectors have same fingerprint in our dataset, which means that they could achieve the same accuracy but runs much faster.

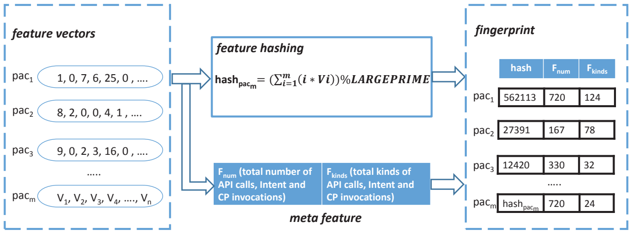

feature vectors
pac1 1, 0, 7, 6, 25, 0 , ….
pac2 8, 2, 0, 0, 4, 1 , ….
pac3 9, 0, 2, 3, 16, 0 , ….
…..
pacm V1, V2, V3, V4, …., Vn
feature hashing
hashpacm = (∑m i=1 (i ∗ Vi))%LARGEPRIME
Fnum (total number of API calls, Intent and CP invocations)
Fkinds (total kinds of API calls, Intent and CP invocations)
meta feature
fingerprint
hash Fnum Fkinds
pac1 562113 720 124
pac2 27391 167 78
pac3 12420 330 32
…..
pacm hashpacm 720 24

Figure 2: The process of feature hashing.

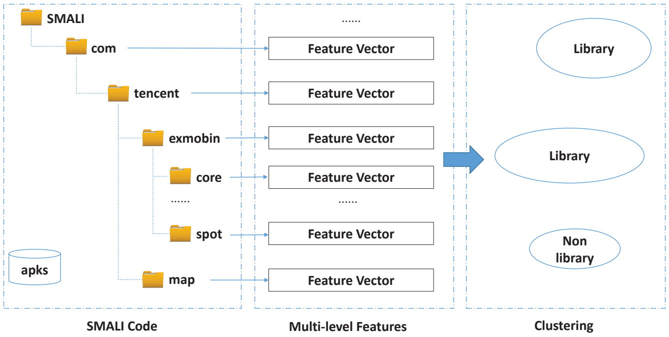


Figure 3: Clustering-based library identification.

2.2 Multi-level Clustering

2.2.1 Overall Process

We use a multi-level clustering method, which is an extension of the previous proposed clustering-based approach[15] to identify all possible libraries from Google Play.

As shown in Figure 3, there are two libraries *mobwin* and *tencent map* in the app. We calculate features for all the packages including *com/tencent*, *com/tencent/exmobwin*, *com/tencent/map*, *com/tencent/exmobwin/core* etc. The feature of a package is a union set of the features of all the classes and sub-packages under it. Then we cluster these features, and identify the largest possible library. As a result, *com/tencent/exmobwin* and *com/tencent/map* will be identified as the package root.

2.2.2 Clustering

When clustering all the features into groups, strict comparison is enforced, which means that only when the features of two packages are exactly the same can they be clustered into one group.

We assume the app dataset is large enough. A library will be detected as long as the library is popular enough. We will first use a threshold (set at 50) to select clusters that belong to third-party libraries.

The clustering results might include multiple versions of the same library due to library evolution or compiler optimization. In these cases, the libraries are still used in many apps, so they are detected as potential libraries too. The multiple versions will not affect the final detection result because we can detect only one particular version within a given app.

2.2.3 *Package Name Identification*

Because the name of a package could often be obfuscated, it cannot be used as a meaningful representation of the package. However, because not all packages are obfuscated, we are able to infer the package names of a package within a cluster based on the unobfuscated package names.

Specifically, we keep the package name information at the clustering step. For each cluster, we have a list of package names. Meaningful package names with higher frequency will be highlighted and obviously obfuscated package names such as *com/a/a* will be deleted. Finally, most representative package names are not obfuscated and recognizable.

2.3 Instant Library Detection

For a given app, we extract the stable code feature and use feature hashing to get the signature of each folder. We compare the hashed features with all identified libraries at all different levels. After the comparison, we get a list of possible libraries included in the given app. Because the identified libraries contain all different folder levels, some of them might be inclusive of one another. Thus we need to remove the libraries that are contained in another library. This can be done by filtering the list of libraries that matches the fingerprints by traversing through the folder hierarchy to identify only those libraries at the highest level.

3. IMPLEMENTATION

3.1 Feature Extracting and Clustering

We have implemented the prototype of LibRadar and make it available to public. To achieve high coverage and high accuracy, we have analyzed 1,027,584 free Android apps collected by PrivacyGrade [5] in March 2015. We first preprocess all the apps, decompile them and extract stable code feature for all of the packages according to the package structure. Then we cluster these feature vectors and find potential third-party libraries. The total process takes more than 20 days to complete with 10 Amazon EC2 servers(m3.medium).

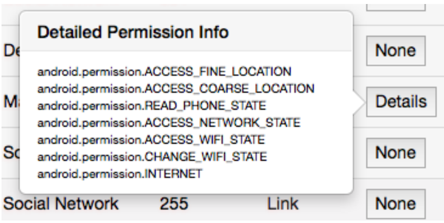
As stated earlier, this serves as the preprocessing step for LibRadar that only needs to be done once. Future updates can also be done in an incremental way to reduce the processing time cost significantly.

3.2 Library Information

As mentioned earlier, LibRadar can find the original package name for an obfuscated library, which is useful for users to know the identity of the library. However, we want to provide more information such that it will help users understand how and why the libraries are used in a given app.

We first sort the detected libraries based on their usage frequency. Then we manually check the package names of the top 200 libraries and add descriptions such as library names, common package names, library categories and official websites. These libraries represent more than 80% of all libraries used (counting all use cases of each library) in one million Google apps. These information will be shown in the library detection results of LibRadar.

Permissions used by each library are important information for the security and privacy of Android apps. As we have already identified all APIs and Intents used in each library, we can generate a list of permissions used by each library, which is based on the API-permission maps provided by PScount [6]. The list of permissions will also be shown in the LibRadar detection results, as in Figure 4.

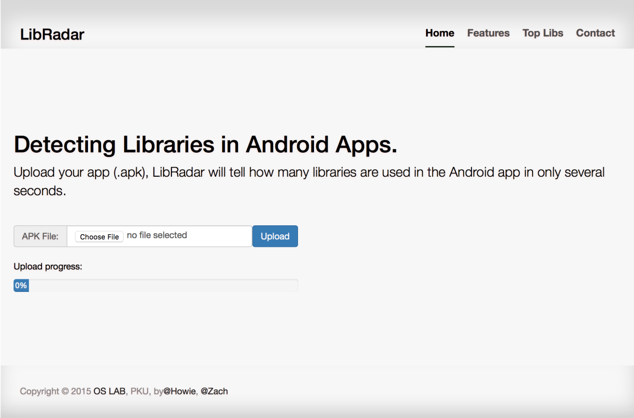


Detailed Permission Info

android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION
android.permission.READ_PHONE_STATE
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.CHANGE_WIFI_STATE
android.permission.INTERNET

Social Network 255 Link None

Figure 4: Permissions used in a detected library.



LibRadar

Home Features Top Libs Contact

Detecting Libraries in Android Apps.

Upload your app (.apk), LibRadar will tell how many libraries are used in the Android app in only several seconds.

APK File: Choose File no file selected Upload

Upload progress:

0%

Copyright © 2015 OS LAB, PKU, by@Howie, @Zach

Figure 5: The LibRadar tool homepage.

4. USAGE EXAMPLE

LibRadar is provided as a web service with both website support and exposed web APIs. Figure 5 shows the web page of the LibRadar tool. Besides the uploading and analysis feature, we also provide information on the most frequently used libraries ("Top Libraries" on the page).

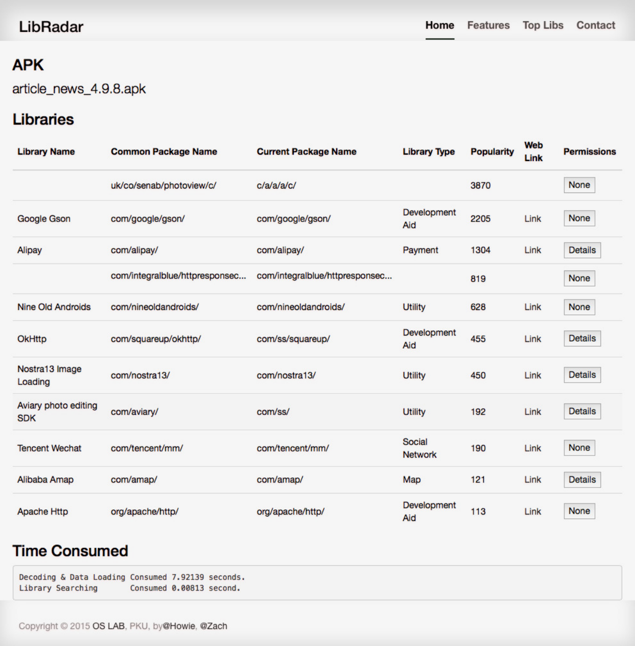
The user can upload an Android APK file to analyze its libraries. Take *ArticleNews*, a famous news app, as an example. Click the *Upload* button and wait for a few seconds, the results will be as shown in Figure 6. The results include a list of libraries used in the given app sorted by their popularity, along with a list of information about each library.

4.1 Performance Information

As shown in Figure 6, LibRadar takes less than 8 seconds for decoding and data loading, which is mainly consumed by the Apktool [2]. Library search only takes less than 10 miliseconds because the database is pre-sorted and the stable code feature are hashed. We tested more than 100 Android apps and most apps take less than 10 seconds to complete, which is good enough for instant detection.

4.2 Obfuscated Libraries

Many Android apps are obfuscated and package names are unable to recognize. In ArticleNews, we could not recognize what the package *c/a/a/a/c* and *com/ss* is, but LibRadar is able to tell that *c/a/a/a/c/* is *uk/co/senab/photoview/c/* (*Senab photoview library*) in fact and *com/ss/* refers to *com/aviary* (*Aviary library*) here. These two libraries are both popular photo rendering libraries for Android development. Senab Photoview is a development aid library which aims to help produce a zooming Android ImageView. Aviary is an image editing component in the *Adobe Creative SDK*. It is difficult to manually recognize obfuscated libraries but can be done easily with LibRadar.



APK

article_news_4.9.8.apk

Libraries

Library Name	Common Package Name	Current Package Name	Library Type	Popularity	Web Link	Permissions
	uk/co/senab/photoview/c/	c/a/a/a/c/		3870		None
Google Gson	com/google/gson/	com/google/gson/	Development Aid	2205	Link	None
Alipay	com/alipay/	com/alipay/	Payment	1304	Link	Details
	com/integralblue/httpresponsec...	com/integralblue/httpresponsec...		819		None
Nine Old Androids	com/nineoldandroids/	com/nineoldandroids/	Utility	628	Link	None
OkHttp	com/squareup/okhttp/	com/ss/squareup/	Development Aid	455	Link	Details
Nostra13 Image Loading	com/nostra13/	com/nostra13/	Utility	450	Link	Details
Aviary photo editing SDK	com/aviary/	com/ss/	Utility	192	Link	Details
Tencent Wechat	com/tencent/mm/	com/tencent/mm/	Social Network	190	Link	None
Alibaba Amap	com/amap/	com/amap/	Map	121	Link	Details
Apache Http	org/apache/http/	org/apache/http/	Development Aid	113	Link	None

Time Consumed

```
Decoding & Data Loading Consumed 7.92139 seconds.
Library Searching       Consumed 0.00813 second.
```

Copyright © 2015 OS LAB, PKU, by@Howie, @Zach

Figure 6: An example of third-party libraries detected in a given app.

5. RELATED WORK

Most recent studies use whitelist-based approaches to detect libraries in Android apps. For example, AdRisk [11] proposes to identify potential risks posed by advertisement libraries and they use a whitelist of 100 advertisement libraries. Book *et al.* [7] use a whitelist of 68 advertisement libraries to investigate the changes over time in the behavior of advertisement libraries. For Android app clone detection, Chen *et al.* [8] use a list of 73 libraries in their whitelist to filter third-party libraries when detecting app clones.

Because whitelist-based approaches only compare the package names, it is fast and could be easily applied to thousands of apps. However, this approach has low coverage of third-party libraries and cannot handle name obfuscation, which is widely used during Android app development nowadays.

AdDetect [13] and PEDAL [12] use machine-learning to detect advertisement libraries in Android apps and achieve very high accuracy. However, these approaches only focus on detecting advertisement library, not all the third-party libraries. Many other kinds of third-party libraries (e.g., social networking, game engine, development tool, map service) may be difficult to detect with machine learning.

Because the same libraries are used by many different apps, it can be detected by clustering over many apps. AdDarwin [10] and WuKong [15] use clustering-based approaches to detect third-party libraries. Both of them focus on repackaging detection and they need to filter third-party libraries first. One major limitation of clustering-based approaches is that they rely on the existence of a large number of apps to build a repository of possible third-party libraries to detect those libraries appearing in many apps. It is impossible to perform instant detection based on clustering alone.

In LibRadar, as we have already collected features of all possible libraries in a preprocessing step by analyzing one million apps, the features can be used directly for instant comparison. Thus it works as fast as the whitelist-based approaches using package names, while it is as accurate as the clustering-based approaches.

6. CONCLUSION

We have presented LibRadar, a tool to detect third-party libraries in Android apps accurately and instantly. We perform a preprocessing step by analyzing more than a million apps from Google Play to gather a complete set of features for all potential Android libraries. The results include the information of 29,279 potential libraries, by far the largest list of third-party libraries for Android apps. Based on the list of library signatures, LibRadar is able to detect the third-party libraries used in a given app instantly based on simple static analysis and feature comparison.

7. ACKNOWLEDGEMENT

This work is supported in part by the High-Tech Research and Development Program of China under Grant No. 2015AA01A202, and the National Natural Science Foundation of China under Grant No. 61421091, 61103026.

8. REFERENCES

[1] Android library statistics. http://www.appbrain.com/stats/libraries/.

[2] Apktool: A tool for reverse engineering Android apk files. http://ibotpeaches.github.io/Apktool/.

[3] Apps with most 3rd party libraries. http://privacygrade.org/stats.

[4] An assembler/disassembler for Android's dex format. https://github.com/JesusFreke/smali.

[5] Privacygrade: Grading the privacy of smartphone apps. http://privacygrade.org/.

[6] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. PScout: Analyzing the Android permission specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12)*, 2012.

[7] T. Book, A. Pridgen, and D. S. Wallach. Longitudinal analysis of Android ad library permissions. In *Proceedings of IEEE Mobile Security Technologies (MoST'13)*, 2013.

[8] K. Chen, P. Liu, and Y. Zhang. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, 2014.

[9] J. Crussell, C. Gibler, and H. Chen. Attack of the clones: Detecting cloned applications on Android markets. In *Proceedings of the 17th European Symposium on Research in Computer Security (ESORICS '12)*, 2012.

[10] J. Crussell, C. Gibler, and H. Chen. Scalable semantics-based detection of similar Android applications. In *Proceedings of ESORICS'13*, 2013.

[11] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC'12)*, 2012.

[12] B. Liu, B. Liu, H. Jin, and R. View. Efficient privilege de-escalation for ad libraries in mobile apps. In *Proceedings of the The 13th International Conference on Mobile Systems, Applications, and Services (MobiSys'15)*, 2015.

[13] A. Narayanan, L. Chen, and C. K. Chan. AdDetect: Automated detection of Android ad libraries using semantic analysis. In *Proceedings of IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP'14)*, 2014.

[14] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in Android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, 2012.

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

[16] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party Android marketplaces. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy (CODASPY '12)*, 2012.