APIHelper: Helping Junior Android Programmers Learn API Usage

Jingjing Zhao, Tao Song, Yuxia Sun*

***Abstract*—Android SDK provides programmers with a variety of APIs to conveniently build applications, but due to the misunderstanding of the API usage, programmers may misuse the APIs, leading to various problems including security issues, energy consumption issues, and user experience issues. To alleviate those issues, this paper proposes a tool, called APIHelper, to help junior Android programmers learn the API usage. By tracking and managing API invocation, APIHelper enable junior Android programmers to follow how an app implements a functionality with its API call sequence, to understand the necessity of a specific API for a specific functionality implementation, and to optimize an app by finding and removing needless APIs. And because of that, APIHelper can also be used as a teaching tool.**

***Index Terms*—API invocation, energy consumption issues, security issues, user experience issues.**

I. Introduction

API (Application Programming Interface) is a predefined function that provides programmers with the ability to access a set of routines based on a piece of software or hardware[1]. As Android SDK evolves, programmers can conveniently build applications with a variety of APIs it provides. However, when programmers misunderstand the API usage, they will misuse the API, which may cause some security, energy consumption, and user experience issues, so how to help the programmers understand the API usage becomes a subject to be studied. For a certain functionality of an application, if you want to minimize the impact on system performance and get the best user experience, you need to use the most suitable and necessary API, but identify which API is the most suitable for the implementation of the functionality needs programmers' thoroughly understanding for API. For junior programmers or students, they may not understand the API well, so learning the specific functionality implementation of widely used applications is a quick way to learn the API usage.

Android uses the permission mechanism to protect user's sensitive and privacy information. The existing permission management mechanism will affect all the APIs related to the permission, and can not meet the requirements of the one-by-one management of the API, but APIHelper can. When implementing a certain functionality, the misuse of energy-consuming APIs may cause a large amount of power consumption of the Android device, and frequent charging and discharging will also affect the battery life. The way an application embeds an advertisement can bring profits to programmers, but the introduction of advertisements may degrade the performance of the device, affect the user experience, and increase battery consumption. So which of the most appropriate and necessary energy-consuming APIs and advertising APIs are used to fulfill the requirements is a valuable skill that junior programmers need to learn. They can use APIHelper to learn how these APIs are used by existing applications on the market, in order to understand the API usage well.

In this paper, we design and implement a tool named APIHelper that monitors and manages various Android APIs in real time, which is convenient for junior programmers to understand the usage of learning API. We focus on three APIs, the Permission API, the Energy-consuming API, and the Advertising API. APIHelper is based on the Xposed open source framework. It can hook various types of APIs, track and manage the execution of these APIs, and generate a API invocation log of a certain functionality. The programmers can run a certain functionality of an app, set the start and end time of the monitoring, view the sequence of API calls related to this functionality displayed in the log, analyze the APIs appearing in the sequence, which can make them learn how to implement the same or similar functionality. APIHelper also provides the ability to manage APIs, that is, programmers can allow or deny API invocation, then they can test whether the API call affects the application's implementation of a certain functionality. If no effect, they should study this API and avoid using it when implementing the same functionality. APIHelper also implements a black and white list that saves the programmer's choices, which can make the process easier and faster, and supports APIs that require custom configuration. All in all, APIHelper is extensive and flexible, providing a way for programmers to better understand the API.

Our main contribution in this paper are as follows:

1) A tool, named APIHelper, is developed by tracking and managing API invocation of Android apps to help junior Android programmers learn API usage. Design and implementation of the tool is dilated.
2) The following typical application scenarios of APIHelper are demonstrated: (1) APIHelper enables programmers to follow the API call sequence of any functionality implemented by any app. (2) APIHelper helps programmers understand the necessity of a specific API in implementing a specific functionality of an app. (3) APIHelper guides programmers to optimize the implementation of apps by finding and removing needless APIs.

II. Related Work

Due to there is no studies for using the tools which monitor API calls to help junior programmers learn the API usage,

This work was supported by the National Natural Science Foundation (#61402197) of China, Guangdong Province Science and Technology Plan Project (#2017A040405030) in China, Tianhe District Science and Technology Plan Project (#201702YH108) in Guangzhou City of China.

Jingjing Zhao, Tao Song, Yuxia Sun are with the Department of Computer Science, Jinan University, Guangzhou, 510632, China. (*Corresponding author: Yuxia Sun, e-mail: tyxsun@email.jnu.edu.cn)

this paper reviews the various technologies of monitoring and managing APIs.

Android permission management is a hot topic of research today, because it plays a vital role in protecting user privacy and sensitive information. In view of the current coarse granularity of Android permission management mechanism, it is necessary to fine-tune the management granularity to the API level, which requires obtaining APIs related to permission. These APIs can be collected in Android documents by crawlers implemented by python. However, the permissions provided by Android documents are limited[2]. Stowaway[2], a tool for detecting excessive permission grants in compiled Android applications, lists 1,259 permission APIs for specific versions. Perman[3] monitors KRM (kernel reference monitor) and FRM (framework reference monitor) to intercept application permission requests and track the asynchronous execution of applications at the thread level, it can distinguish the permission request is from third-party library or from original application. Users can choose to grant or deny permission requests from different modules, which is great for application permission management. However, this method affects all APIs related to permissions, which is different from APIHelper that can monitor the permissions APIs one by one. He Y[4] detects the privacy leaks of third-party libraries by hooking APIs related to Android privacy.

As for the relationship between energy consumption and API, Abhinav Pathak et al.[5] mentioned that incorrect programming and improper use of API are the reasons for high energy consumption. Mario Linares-Vásquez et al.[6] measured the energy consumption data of the hardware and analyzed the usage mode of the high-energy API based on the data, which is convenient for developers to use this mode to reduce energy consumption. SAAD[7] detects resource leak by context sensitive analysis and produce energy report , it can detect more than eighty resource leaks according to resource API information. Hu Y[8] proposed a predictive model that maps energy consumption to APIs, which can give developers guidance. Cruz L[9] studied the commit issue and pull request of 1027 Android program and proposed 22 design patterns that can improve energy efficiency. Nijim et al.[10] proposed an energy-aware data mining predictive technique for hybrid storage systems named En-Stor, which uses data mining predictive to save energy.

For handling advertisements in Android apps, miAdBlocker[11] uses aspect-oriented programming to insert a force monitor at the bytecode level, enabling users to disable advertisements for each app. Perman[3] restricts the display of advertisements by denying the permission requests(such as INTERNET and ACCESS_NETWORK_STATE) of the advertising platform module. However, APIHelper can restrict advertisements by prohibiting APIs provided by third-party ad platform SDK, the granularity is smaller. Boyuan He[12] are the first to reveal the preference of developers and users for ad networks and ad types, they study 697 unique APIs from 164 ad networks. AdCapsule[13] is a user-level solution to confine advertisements, it uses permission sandbox and file sandbox to isolate the permission and file used by advertisement, so ad library and ad content cannot read or write any file outside sandbox, which can relieve the security and privacy issues. LS Chen et al.[14] tried to identify which type of advertisement can attract customers more and the important mobile advertisements factors of influencing the customer's loyalty, so as to make a balance between the customer's experience and the profit.

The dynamic monitoring technology of Android applications includes kernel layer monitoring and application layer monitoring. Yoon C et al. [15] propose a energy metering system which monitors and computes the energy consumption of Android program at the kernel level. Aurasium[16] is an application layer monitoring tool that is different from hooking technology. It uses Dalvik layer code tampering and repackaging techniques to monitor specific sensitive APIs. Cheng Sun et al.[17] proposed a monitoring system which uses Xposed to track sensitive API calls and log, but his text is not specifically regulated.

III. Target APIs

The research in this paper is aimed at three APIs(permission API, energy consumption API, and advertising API). It provides ideas for the research of helping junior Android programmers learn the API usage, we will add more types of APIs later.

We first build a collection of permission APIs that APIHelper needs to use. PScout[18] extracts the permission specification for any Android version, provides a complete mapping between API calls and permissions. So we first parse the CSV file that holds this mapping, then collect the APIs and their related information, that is, the API name, the class to which the API belongs, and the permission corresponding to the API. Finally, we use the collected information to construct a collection of more than 30,000 APIs related to more than 90 permissions. But there are some permissions only corresponding to an API, as shown in Table I.

TABLE I
Permission API

API	Corresponding Permission
forceRemoteSubmixFull Volume	android.permission.CAPTURE_AUDIO_OUTPUT
clearApplicationUserData	android.permission.CLEAR_APP_USER_DATA
deleteApplicationCacheFiles	android.permission.DELETE_CACHE_FILES
getPackageSizeInfo	android.permission.GET_PACKAGE_SIZE
checkShowToOwnerOnly	android.permission.INTERNAL_SYSTEM_WINDOW
movePackage	android.permission.MOVE_PACKAGE
getNextEntry	android.permission.READ_LOG
setRingtonePlayer	android.permission.REMOTE_AUDIO_PLAYBACK
setTouchCalibrationFor InputDevice	android.permission.SET_INPUT_CALIBRATION
tryPointerSpeed	android.permission.SET_POINTER_SPEED
setWallpaper	android.permission.SET_WALLPAPER
setWallpaperComponent	android.permission.SET_WALLPAPER_COMPONENT
shutdown	android.permission.SHUTDOWN

Then we build a collection of energy-consuming APIs that APIHelper needs to use. Network behavior is the most

energy-consuming behavior in Android applications[19], especially the operation of making http requests, so the API that makes http requests belongs to the energy consumption API. Android natively provides two network access methods based on HttpUrlConnection and HttpClient, the latter is removed in Android 6.0, so we only focus on the API that makes http requests based on HttpUrlConnection. In addition, Android has many network request frameworks, among which the widely used application is OkHttp, so we extract the API related to the network request from OkHttp's Javadoc document. In addition to the APIs associated with network behavior, applications can use the lock mechanism to prevent the system or WiFi from going to sleep, which also increases power consumption, so the APIs associated with WakeLock and WifiLock are also energy-consuming APIs. The above APIs are summarized as shown in Table II.

TABLE II
ENERGY-CONSUMING API

Energy-Consuming Operation	Corresponding Class	Corresponding API
NetWork	java.net.URL	openConnection
NetWork	java.net.HttpURLConnection	setRequestMethod
NetWork	com.squareup.okhttp.Request.Builder	url
NetWork	com.squareup.okhttp.Request.Builder	build
WakeLock	android.os.PowerManager.WakeLock	acquire
WifiLock	android.net.wifi.WifiManager.WifiLock	acquire

Finally, we build a collection of ad APIs that APIHelper needs to use. The third-party ad platform SDK access guide lists which contains various APIs for adding banner ads, interstitial ads, native ads, and incentive advertising, etc., such as the loadAd() method for requesting ads, and the show() method for displaying ads, etc. The access of the third-party advertising platform SDK is not unique, that is, the developer can choose to access advertising platforms such as AdMob, Xunfei, and Youmi. Because the use of Admob advertising platform is more common, we use the API provided by it as APIHelper's advertising API, as shown in Table III.

TABLE III
ADVERTISING API

Ad Formats	Corresponding Class	Corresponding API
Banner	com.google.android.gms.ads.AdView	loadAd
Interstitial	com.google.android.gms.ads.InterstitialAd	loadAd
Interstitial	com.google.android.gms.ads.InterstitialAd	show
Rewarded Video	com.google.android.gms.ads.reward. RewardedVideoAd	loadAd
Rewarded Video	com.google.android.gms.ads.reward. RewardedVideoAd	show
Native	com.google.android.gms.ads.AdLoader	loadAd
Native	com.google.android.gms.ads.AdLoader	loadAds

IV. DESIGN AND IMPLEMENTATION OF APIHELPER

A. Design of APIHelper

Based on the API collected in the previous section, APIHelper implements the following functionality:

1) During the running of the application, when the API is called, if the API is not in the cache or black and white list, APIHelper will block the calling thread of this API, pop up the widow to display the API information, and then wait for the programmers to select the next operation, such as, allowing or denying the API call, adding the API to the blacklist or whitelist, as shown in Figure1. When the programmer chooses to allow or deny the API call, APIHelper will allow or disable the operation of the API correspondingly, and add the programmer's selection result to the cache. In the preset expiration time (e.g., 20 seconds), when the application calls the API again, the choice will not be asked.

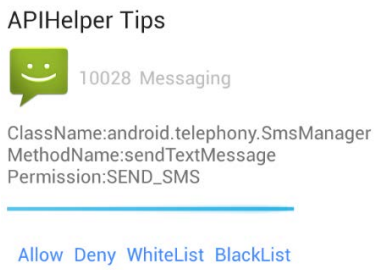
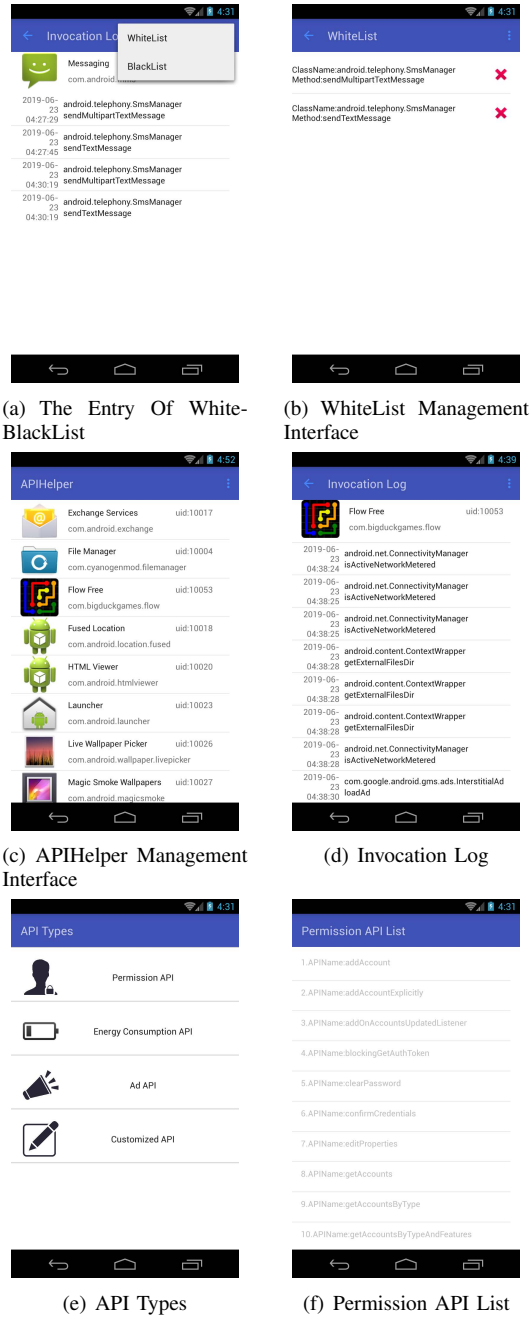


Fig. 1. user selection Interface

2) API whose thread is blocked can be added to the blacklist or whitelist by the programmer. If the API in the black and white list is called again by the same application, APIHelper no longer prompts the user to select the option but disables or allows it by default. APIHelper provides programmers with a interface for managing APIs, they can view or remove APIs in the black and white list, as shown in Figure 2(a) and 2(b).
3) APIHelper can monitor the use of APIs and generate the API invocation log during the running of the application. The management interface is shown in Figure2(c), displaying many types of applications, the programmer can click one application(e.g., Flow Free) and view the invocation log, as shown in Figure2(d).
4) APIHelper supports the programmers to have special research on some APIs by custom configuration. Programmers can select a certain type of API in the interface to study, as shown in Figure 2(e). They also can select the permission API first, then monitor some specific APIs in the permission API list and ignore others, as shown in Figure2(f).

B. Implementation of APIHelper

APIHelper is based on Xposed and is mainly composed of AhService(APIHelper Service), ToolInit, HookManager and database module. The AhService is the core of interaction between modules. The class ToolInit is a module registered to Xposed and is the entry point for APIHelper startup. The class HookManager uses a lot of Java reflection technology



(a) The Entry Of White-BlackList

(b) WhiteList Management Interface

(c) APIHelper Management Interface

(d) Invocation Log

(e) API Types

(f) Permission API List

Fig. 2. Functional diagram

to get the instance of all APIs to be monitored. At the same time, in order to implement APIHelper database management, the DBUtils module supporting ORM in the xUtils framework is used. The overall architecture of APIHelper is shown in Figure 3.

The class ToolInit is an initialization module based on the Xposed framework. It is automatically loaded by Xposed and is the first module launched by APIHelper. ToolInit implements the interfaces IXposedHookZygoteInit and IXposedHookLoadPackage provided by XposedBridge.jar package. The method initZygote(StartupParam startupParam) in the IXposedHookZygoteInit interface will be called when Zygote is initialized, it should complete two operations: register and start AhService; call the HookManager module to monitor the system's API. When the App package is loaded, the method handleLoadPackage(XC_LoadPackage.LoadPackage

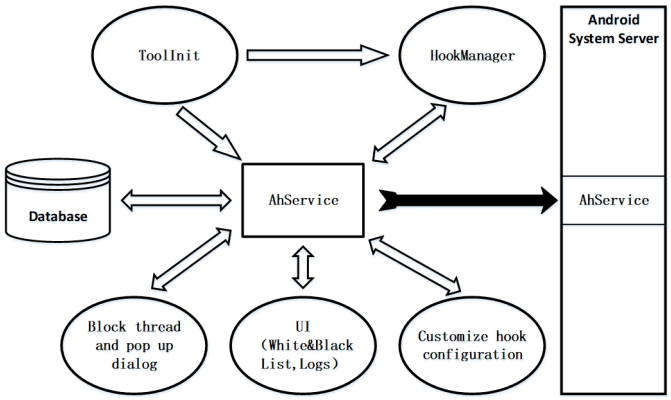


Fig. 3. The Overall Architecture Of APIHelper

Param loadPackageParam) in the IXposedHookLoadPackage interface is called, and the parameter loadPackageParam is used to obtain the class Class of the third-party library. In the handleLoadPackage, the HookManager module is called to monitor the API of the third-party library.

HookManager is the control class for monitoring APIs. It obtains an instance of the API by using Java's reflection technology according to the collected API set, and then calls the method hookMethod(Member hookMethod, XC_MethodHook callback) provided by the XposedBridge.jar package to monitor the API. The instance of an API that needs to be monitored may be a constructor of a class or a general method, for which the overload and override need to be considered. The idea of getting an API instance is: (1) Initialize listMember to store instance of API; (2) Add the constructor directly to listMember; (3) For general methods, initialize listParameters to save the list of parameters of APIs added to the listMember, extract all methods of the class of the API, filter the abstract method, after that you may get multiple overloaded methods of the same name, then use listParameters to filter the overridden method, and add the last remaining methods and their parameter lists to listMember and listParameters, take their parent class to continue the operation until the parent class is empty; finally, call the method hookMethod(Member hookMethod, XC_Method Hook callback) provided by XposedBridge.jar to hook the API in listMember.

The AhService is the core of APIHelper. It provides a control interface for blocking API calls, popping up dialog box, recording logs, and obtaining black and white lists. In order to start the AhService when the Android system starts, APIHelper registers the AhService as the service process of the system by calling the method addService of the class ServiceManager in the reflection way. Because AhService runs in the system service process, and the APIs that need to be hooked run in their respective third-party application processes, APIHelper interacts with third-party APIs through the Binder interprocess communication library. APIHelper also calls the method getService to obtain the AhService proxy object and performs the following initialization operations: (1) setting the AhService semaphore to limit the popup window and handle the ANR problem; (2) obtaining the process context for manipulating the UI; (3) initializing worker thread to process the message queue; (4)initializing the database and cache.

V. Application scenarios of APIHelper

A. Learn API call sequence of app functionality

Given a specific functionality of an app, by using the APIHelper, the junior Android programmers can learn the implementation of the functionality at the API level, that is, follow the API call sequence of specific functionality. The API call sequence corresponding to a specific functionality is the API sequences shown in the invocation log in the period from the start to the end of using the functionality, which will show differences due to the difference of functionality. The call sequence will include permission APIs, energy consumption APIs, and advertising APIs, so programmers can learn the usage of a certain API from an application, that is, the type of API used, the number of API used, and the time of API calls, providing reference and preliminary preparation for writing their own program.

B. Learn necessary APIs of functionality implementation

By using the APIHelper, the junior Android programmers can be impressed with the necessity of a specific API for a specific functionality implementation in an app. They can perform the following operations : First, clear APIHelper's cache and black and white list, and then use a certain functionality of an application. When the popup window is displayed, programmers can choose to allow or deny the API call separately, observe the impact of two cases, if the functionality is abnormal, then the API can be considered to have an important role in this functionality, and conversely, the existence of the API is considered to be insignificant. So the programmers can understand the necessity of the API for the specific functionality better. For example, when the programmer sends a text message to "10010", APIHelper monitors the API sendTextMessage() in the class SmsManager of the short message application, and pops up a window waiting for the programmer's selection. The programmer clicks the allow button, the short message is normally sent; but when the programmer clicks deny button, the short message is failed to sent, so the API sendTextMessage() is necessary for the functionality of sending message.

C. Optimize App by finding and removing needless APIs

For a specific functionality of an app, when the junior Android programmer disables a specific API and the functionality is still normal, this API will be considered to be needless API and should be removed, so as to alleviate some issues to optimize program.

There may exists security issues in the app, such as, excessive grants of permissions. By using the APIHelper, they can find this problem by determining if the unnecessary API is the only API corresponding to a permission, if so, this permission is an excessively applied permission and should be removed. They can also find this issue by comparing the APIs in the invocation log with the APIs present in the application and associated with the permission this application applied for. For example, an application applies for the permission SET_WALLPAPER, and the only corresponding API for this permission is setWallpaper(), but the invocation log shows that this API has never been called, so the permission SET_WALLPAPER is an excessively applied permission and should be removed.

There may exists energy consumption issues in the app due to the unnecessary use of energy consuming APIs. By using the APIHelper, the programmers can find this problem by finding the needless energy consumption APIs and solve it by removing the needless APIs.

There may exists user experience issues in the app due to the frequent use of ad API, because if the frequency of the ad API call is too high, the ad will be repeatedly popped up on the user interface, making user very annoyed. The programmers can find this problem by finding the unnecessary ad APIs and solve it by reducing the APIs appropriately, which can optimize the user experience on the condition that guaranteeing the programmers' fundamental interests.

D. Help students learn the API usage

In fact, besides the junior Android programmers, the teacher also can use the APIHelper to help students who will become Android programmers in the future learn the API usage as a teaching tool, which is a good example of interesting and practical teaching. As mentioned before, APIHelper enables students to learn the API call sequence and necessary APIs of specific functionality, which can impress them with the API usage and make them use the API more appropriately when needed. APIHelper also can help students learn to optimize the implementation of apps by finding and removing needless APIs, which can make them understand the API usage more thoroughly.

In addition, APIHelper enables students to learn the usage of a specified type of APIs. For example, if a student wants to learn which advertisement APIs are invocated in any App developed by others, he only needs to choose the "Ad API" option in the "API Types" menu and run the APP. Then, APIHelper will display in sequence all the Ad APIs that are invocated during the APP's execution, and ignore all the other types of APIs. APIHelper can facilitate students to learn such commonly-used API types as permission API, energy-consumption API and Ad API.

VI. User Survey of APIHelper

As presented in the previous section, APIHelper can be used as a teaching tool. In this section, we will report the result of a user survey conducted among 38 students who are junior Android programmers. The students are asked to use APIHelper in the above three application scenarios, and then to score (on 10 points scale) whether APIHelper can help them in learning App development.

As the survey results show, most students think APIHelper helpful for learning how to use APIs. Table IV lists the average scores of three learning scenarios. As the table shows, in the view of the students, APIHelper can effectively help them optimize Apps by finding and removing needless APIs, which is its most attractive application scenario with the highest score of 9.6. The students think APIHelper useful when they want to learn API call sequence belonging to a specified app functionality, which is scored 9.1. For the third application scenario shown in the table, some students consider it duplicate with the second scenario in the table to some extent, but other students don't feel so, they think these two scenarios are different because the third scenario focuses

on the necessary APIs specially, which has one more process of finding and removing needless APIs in the particular functionality implementation. And thus the score is lower than 9.0. As for the implementation of monitoring specified type of APIs, the students believe that it is convenient in the above three scenarios when they want to study the usage of a specified type of API.

Additionally, some interesting suggestions to improve APIHelper are made by the students. For examples, more Ad APIs can be added, more API types can be supported, API callers from the third-party library or Android system can also be included.

TABLE IV
THE SCORE OF THREE APPLICATION SCENARIOS

Application Scenario	Score
Optimize App by finding and removing needless APIs	9.6
Learn API call sequence of app functionality	9.1
Learn necessary APIs of functionality implementation	8.3

VII. CONCLUSION AND FUTURE WORK

In this paper, we implement a tool named APIHelper based on the Xposed framework, which helps the junior Android programmers learn the usage of the permission APIs, energy consumption APIs, and advertising APIs. And we have demonstrated four typical application scenarios of APIHelper, that is, enabling junior programmers follow the API call sequence of any specific functionality of any app, helping them understand the necessity of a specific API for a specific functionality implementation, guiding them to optimize app by finding and removing needless APIs, etc. It has extensiveness and flexibility, and provides a friendly interface for the programmers.

APIHelper can be improved in the following directions: showing the caller of API is from a third-party library or the Android system, recommending API sequences to implement a specified functionality, supporting more API types, and so on.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their valuable and constructive comments.

REFERENCES

[1] "Api," [Online]. Available: http://baike.baidu.com/link?url=KUnTlX6Jg47PdvE0NHhOMyHjj1Fsz15RTcpAk82QrUDEcc8WFPaEDNt3TJf6U990EohGTvzRFyYWMKp-tpVb4q. [Accessed: June-2018].

[2] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 627–638.

[3] J. Fu, Y. Zhou, H. Liu, Y. Kang, and X. Wang, "Perman: Fine-grained permission management for android applications," in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2017, pp. 250–259.

[4] Y. He, X. Yang, B. Hu, and W. Wang, "Dynamic privacy leakage analysis of android third-party libraries," *Journal of Information Security and Applications*, vol. 46, pp. 259–270, 2019.

[5] A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, p. 5.

[6] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: an empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 2–11.

[7] H. Jiang, H. Yang, S. Qin, Z. Su, J. Zhang, and J. Yan, "Detecting energy bugs in android apps using static analysis," in *International Conference on Formal Engineering Methods*. Springer, 2017, pp. 192–208.

[8] Y. Hu, J. Yan, D. Yan, Q. Lu, and J. Yan, "Lightweight energy consumption analysis and prediction for android applications," *Science of Computer Programming*, vol. 162, pp. 132–147, 2018.

[9] L. Cruz and R. Abreu, "Catalog of energy patterns for mobile applications," *Empirical Software Engineering*, pp. 1–27, 2019.

[10] M. Nijim and H. Albataineh, "En-stor: Energy-aware hybrid mobile storage system using predictive prefetching and data mining engine." *Engineering Letters*, vol. 26, no. 2, pp. 252–256, 2018.

[11] K. El-Harake, Y. Falcone, W. Jerad, M. Langet, and M. Mamlouk, "Blocking advertisements on android devices using monitoring techniques," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2014, pp. 239–253.

[12] B. He, H. Xu, L. Jin, G. Guo, Y. Chen, and G. Weng, "An investigation into android in-app ad practice: Implications for app developers," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2465–2473.

[13] X. Zhu, J. Li, Y. Zhou, and J. Ma, "Adcapsule: Practical confinement of advertisements in android applications," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[14] L.-S. Chen and C.-C. Liu, "Using feature selection approaches to identify crucial factors of mobile advertisements," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2015.

[15] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring," in *Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12)*, 2012, pp. 387–400.

[16] R. Xu, H. Saïdi, and R. Anderson, "Aurasium: Practical policy enforcement for android applications," in *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 539–552.

[17] C. Sun and S. Qin, "A monitoring method of sensitive calls based on the android platform software behavior," in *2015 5th International Conference on Computer Sciences and Automation Engineering (ICCSAE 2015)*. Atlantis Press, 2016.

[18] 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*. ACM, 2012, pp. 217–228.

[19] D. Li, S. Hao, J. Gui, and W. G. Halfond, "An empirical study of the energy consumption of android applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 121–130.